DOS EXTRA Nr. 11 – 1990

DOS EXTRA

18,– DM 145 öS 18,– sfr 55,– Fmk

Sonderausgabe DOS EXTRA

Nr. 11 1990

ISSN 0934 – 2842

DMV-Verlag

Eine Publikation der
DOS INTERNATIONAL

Turbo Pascal total

- Strukturen
- Overlays
- Units

MS-DOS-Speicher-verwaltung

Turbo Pascal intern

- Speicherverwaltung
- Hardware-Zugriffe
- Interrupt-Handler

Durchblick mit Turbo Pascal

Spitzenprogramme zum Abtippen

- HPGL-Erweiterung
- SAA-Blockbefehle für Turbo Pascal

Hilfreiches Utility: Diskettendoktor

Neuer Shareware-Hit
OFFIX Ihr BÜRO

Nur bei uns und in Deutsch! OFFIX macht den PC zu Ihrem besten Freund, im Büro und Zuhause! (Serien)-Briefe, Adressen, Mailings, Listen, Kunden, Berichte, Formulare, Etiketten, usw.. - Stichworte, die jeder Anwender kennt, und die kaum ein Programm optimal löst. OFFIX dagegen bildet Ihr Büro nach, wie Sie es gewöhnt sind, mit Aktenschränken, Ordnern, Mappen und Formularen. Mit seinem integrierten Lernprogramm , Onlinehilfe und der intuitiven Bedienung, kann OFFIX auch von absoluten Neulingen sofort eingesetzt werden. Selten war es einfacher mit dem PC zu arbeiten! Die Sensation auf dem Sharewaremarkt und dennoch eine durch und durch professionelle Software aus professionellem Haus!

☐ **OFFIX - Das (Ihr) persönliche (s) Büro nur DM 29!**

GESCHÄFTE...

☐ **NEU! HAUS-MASTER - HAUSVERWALTUNG 4.0** DM 12
#1007 **Deutsch!** Professionelle Hausverwaltung für 12 Objekten je 45 Mieter. Menügesteuert erfassen Sie alle relevanten Daten (Mieter, Hausmeister, Heizkosten, Nebenkosten, etc.) in übersichtlichen Masken. Die Bedienung ist auch für Laien geeignet. Noch nie war es einfacher und bequemer lästige Abrechnungsarbeiten durchzuführen. **Die exclusive deutsche Neuerscheinung!**

☐ **NEU: VIDEKA - VIDEOTHEKENVERWALTUNG 2 DISKS** DM 24
#1012 **Deutsch!** Professionelles Geschäftsprogramm für kleine und große Videotheken. Hier sollten Videothekare zugreifen!

☐ **DER BUCHHALTER 3 DISKS** DM 36
#927 **Deutsch!** Reinrassiges Buchhaltungsprogramm mit zu 20.000 Personenkonten, 10.000 Wertpapier- und Anlagevermögenskonten. Kontierungshilfe. Tilgungsplanerstellung. Kontenplan gem. Datev oder frei definierbar. Ust-Voranmeldung und GUV gem. Bilanzrichtliniengesetz. Taschenrechner und On-Linehilfe.

☐ **NEU! ORGANIZE!** DM 24
Deutsch! Integriertes Management-Tool für Erfolgreiche, mit Termin- Adressverwaltung, Projektplaner, Textverarbeitung, etc. Mit ORGANIZE! bleiben Sie am Ball.

☐ ***NEU!* WARNGULB 3.32b** DM 12
#883 **Deutsch!** Professionelle Terminverwaltung, mit vielen automatischen Funktionen zur perfekten Terminüberwachung.

☐ **GS-GESCHÄFTSPAKET - NEUE VERSIONEN- 3 Disks** DM 36
Deutsch! Update (1.50/1.50 oder neuer) aller GS-Programme. Die GS-Serie ist ein absolut professionelles Programmpaket zum EDV-gestützen Erledigen fast aller geschäftlichen Vorfälle, wie Fakturierung, Adressverwaltung, etc. Die GS-Serie ist anerkanntermaßen eines der besten Softwarepakete für Selbständige. Spitze: GS-Adress verfügt bereits über ein PLZ/Ortsverzeichnis der Post. **NEU:** Datenaustausch zwischen Adresse und Faktura!
NEU DABEI: VOLLVERSION von GS-MENÜ 3.15 das Spitzenmenüsystem von GS. Mit GS-MENÜ integrieren Sie alle Programme problemlos zu einem Gesamtsystem. GS-MENÜ zeigt, wie gut deutsche Tools sein können.
AUCH NEU: Vollversionen von ☐ GS-AUFTRAG und ☐ -ADRESS jeweils nur DM 79.-. Die GS-MENÜ Vollversion gibt's gratis dazu.

☐ **AKTIEN UND BÖRSENMANAGER** DM 89
Deutsch! Sind Sie auch börsensüchtig? Dann helfen Ihnen STOCKY, der Börsenmanager, EIDICHART, STOCKMASTER3, Shareholder, Exchange, Prognos den Überblick zu behalten und Ihr Geld zu mehren. Echte Profis! Jetzt neu 10 Disks.

☐ **C.T.O. GESCHÄFTSPAKET** DM 36
Deutsch! Neueste Versionen von EHO Einzelhandel 4.0, EA-Buchhaltung 3.0 und Adressverwaltung, Fakturierung (Auftragsbestätigung, Lieferscheine, Rechnungen, Mahnwesen) und Buchhaltung mit perfekter Datenintegration bietet dieses Superpaket. Sie müssen keine Daten doppelt eingeben. Die Verbuchung geschieht vollautomatisch. Das spart eine Menge Geld. Ausgereifte Software.

☐ **BAUSTELLE-PC UND RAPPORTE NEUE VERSIONEN** DM 60
Deutsch! Neu, jetzt die komplette neue Version von BAUSTELLE PC plus RAPPORTE in einem Kompaktset, zum Superpreis. Die Programme mit dem unglaublichen Leistungsumfang für alle kostenbewußten Bauhandwerker.

☐ **RW-PAKET (BUCH/KASSE/FILE/MENÜ)** DM 48
Deutsch! RW-BUCH/KASSE. Die Erfolgsprogramme des deutschen Sharewareprofis Wachtmann Software. Elektronisches Kassenbuch nach den Vorschriften des Finanzamtes. RW-BUCH 3.0 - Die perfekte Buchhaltung für Ihren PC. Modernes System mit Pull-Down Menüs. 1000 Konten, mandantenfähig, G+V. 4 Disks. **NEU DABEI:** RW-MENÜ das prof. Menüsystem mit SAA-Standard. Jetzt mit dBaseIII-Dateien.

☐ **HEIM-MANAGER UND STEUER - 2 Disks** DM 24
Deutsch! Überlassen Sie doch lästige Verwaltungs und Berechnungsarbeiten Ihrem PC, mit dem Ferienplaner, Kontoverwaltung, Kapitalrechnungen, Reisekostenabrechnung und den Entscheidungshilfen sind Sie dem **Alltag gewachsen.** Die Steuererklärung erstellen Sie dann noch mit dem Steuer 89-Programm.

DIENSTPROGRAMME

☐ **NEU! Back & Forth 1.12 Programmschalter/SWAPPING** DM 24
Exclusive Neuheit! Blitzschnelles Umschalten zwischen bis zu 9(!) Anwendungen (auch mit Grafik), ohne die Programme abschließen zu müssen. Große Zeitersparnis. Datentransfer (Cut&Paste). Voll 4DOS kompatibel. Minimalster RAM-Bedarf! Aufruf großer Programme als DOS-Shell möglich, etc. Residenter Speicherbedarf z.B. von PC-TOOLS nur noch 9 KB statt 70! Ein neues Shareware-Spitzenprogramm (siehe Testberichte). 2 Disks.

☐ **4DOS-KOMMANDOINTERPRETER *BRANDNEUE VERSION 3.01*** DM 12
#795. **ENDLICH LIEFERBAR!** Stark erweiterte Version 3.0 (.BTM, Logbuch, Files, Maus, Screen-Control, uvm.), des voll kompatiblen Ersatzes für COMMAND.COM mit vielen neuen Befehlen. 4DOS hat was die Sie schon immer vermißt haben, z.B.: Maussteuerung, DOS-Kommandopuffer, Environmenteditor, ALIAS-Namen für Befehle, Programmzuordnung für Dateigruppen, Datei-Full-Screen-LIST und -Auswahl-Funktion, On-line-Hilfe, erneut erweiterter BATCH-Prozessor mit eigener Programmiersprache, UNIX-Wildcards, EGA/VGA 40/50-Zeilen Mode, Dateikommentierung, Mehrfachargumente und Befehle, usw. 4DOS das Supertool für alle DOS-Anwender, vom Einsteiger bis zum Poweruser. Wer 4DOS nicht hat ist selber schuld! **Super-Tests in PC+, c't und PD (Zitat aus c't: "4DOS bietet die Funktionen von DOS 10.0..."). Noch Wünsche?**
☐ 4DOS Offizielle registrierte Version mit HB DM 129.-.

☐ **NEU: EXTENDED ZU EMS40-TREIBER** DM 12
Verwandelt brachliegendes EXTENDED MEMORY in wertvollen EMS 4.0 LIM Speicher, der von vielen Programmen genutzt wird. DOS-RAM wird bis auf 736KB (statt nur 640KB) erweitert. Mit Quellcode!

☐ **SENSATION: VIRENSCANNER - CLEANUP - Jetzt 3 Disks** DM 36
VOLLAUTOMATISCHES AUFSPÜREN UND VERNICHTEN von über 70 gefährlichen Viren (Jerusalem/Island, etc.). Mit den professionellen Programmen von McAfee, **jetzt auch für Netzwerke!** Über 20 Virenkiller der Spitzenklasse auf 3 Disketten Flushot+ (V.1.7), FI-CHECK, etc.) lassen Sie wieder ruhiger schlafen. Sicherer geht's kaum noch! **ACHTUNG:** *INFORMIEREN SIE SICH NOCH HEUTE ÜBER UNSEREN EXCLUSIVEN, ANTI-VIREN-UPDATE SERVICE!*

☐ **PROFI-DATENSCHUTZPAKET - 4 Disks** DM 48
Auswahl der besten Sharewareprogramme zum Verschlüsseln von Daten (nach Pentagon DES-Verfahren) und Einrichten von Zugriffssperren. Kennwortschutz für Programme (sehr wichtig). Hier sollten Sie keine Kompromisse eingehen.

☐ **PROFI-DATEIKOMPRESSION NEUESTE VERSIONEN - 10 Disks** DM 89
BESTSELLER: Das exclusive Paket vom Spezialisten. Viele Programme wurden erneut upgedatet. Unsere Komprimierprogramme ermöglichen es, bisher für unmöglich gehaltene Datenmengen bequem auf Festplatten oder Disketten zu speichern. Dateiverkleinerung: bis 90%, von einer Datei mit 1MB bleiben gerade 100KB übrig. Dateien (Texte, Programme, Kundenkarteien, Grafiken, etc.) beliebigen Inhalts können zu Archiven zusammengefaßt werden. Sehr gut zur professionellen Datensicherung geeignet. Ersetzt oftmals einen teuren Streamer oder eine neue Festplatte. Die enthaltenen Hilfsprogramme mit komfortablen Benützeroberflächen machen die Bedienung zum Kinderspiel! **NewSpace** komprimiert beliebige Dateien im Hintergrund, mit max. 100% Kapazitätssteigerung der Festplatte. Komprimierte Programme bleiben voll lauffähig. Komprimieren ganzer Verzeichnisse oder Platten. Volle Datensicherheit gewährleistet. Das Paket enthält alle professionell verwendbaren Programme mit allen wichtigen Tools von ARCMASTER über Shez bis PKZIP. BEI UNS NUR NEUESTE US-VERSIONEN!

☐ **DEUTSCHE ANLEITUNG DATEIKOMPRESSION** DM 29
EXCLUSIV VON UNS! Sehr ausführliche deutsche Disk-Anleitung (400 Seiten) zu fast allen Komprimierprogrammen und Tools, mit Übersichtstabellen und konkreten Anwendungs- und Installationshinweisen. Endlich kann jeder Anwender sofort und problemlos Dateien komprimieren.

☐ **HERCULES UTILITIES** DM 12
#3. Die besten Dienstprogramme rund um **Ihre Herculeskarte.** 5 CGA Emulatoren, davon einer für Spiele mit eigenem Betriebssystem (bootresistent)! Endlich laufen viele Grafikprogramme auch auf Ihrer Herculeskarte: GWBASIC, Turbo Pascal, Spiele. Mit Hardcopyprogramm für Grafikbildschirme! Wenn Sie eine Herculeskarte haben, brauchen Sie diese unentbehrliche Diskette!

☐ **Diskettenverwaltung KAT 5.0** DM 29
Deutsch: Professionelle Disketten- und Dateiverwaltung, mit unbegrenzter Dateien- und Diskettenzahl, Sortieren, Selektieren und Drucken (Etiketten) nach fünf Kriterien, automatsches Einlesen von Disketten, Kommentarvergabe, uvm. 40 Seiten On-Line-Hilfe. Pull-Down/Mausbedienung. **KEINE SHAREWARE!**

☐ **NEUE VERSION: HARDDISK-BOOSTERS - 3 Disks** DM 36
Die Supertools zum Optimieren Ihrer Festplatte. Mit QUICK-CACHE (NEU V.4.2), dem Cache-Programm der Superlative, integrierter High-Speed Backup-Funktion auf Disketten, Diskoptimizer, Interleavefaktor-Ändern. Den PC auf Touren bringen!

☐ **SUPER DOS-SHELLS 5 DISKS** DM 60
Die leistungsfähigen Menüsysteme für Ihren PC. AUTOMENUE 4.5 mit Passwortschutz, Maussteuerung und vielen Extras, HDM IV die Profi-Menüshell mit SAA-Oberfläche, TREEVIEW, POINT & SHOT und STILL RIVER SHELL, der Altmeister der Profiklasse. Da bleiben wirklich keine Wünsche offen!

☐ **PROF. BATCH PROGRAMMING** DM 48
EBL, PC-EXEC, Screenpaint, Batch Tools Sampler erlauben Ihnen endlich echtes Programmieren und Großrechnerkomfort mit PC-Batch Dateien. 4 Disks.

☐ **cALMER UTILITIES** DM 48
NEU! Sehr reichhaltige Utilitysammlung mit über 30 Utilities für DOS, System-Setup, Screen, Anti-Viren, Cursor, Drucker, Memory, ZIP, Tastatur, Datei- und Directory, residente Programme, Backup, Löschen, uvm. Umfangreiche Anleitung. Das Schweizer Taschenmesser für Ihren Rechner! 4 komprimierte Disks.

☐ **NEUE Ver: 1.32. CP/M-DISK-FREMDFORMATE** DM 12
Lesen und Schreiben von CP/M Fremddisketten unter DOS. 180 Formate vordefiniert (neue Formate leicht anpaßbar, kein Apple II). Mit CP/M-2.2 Emulation.

☐ **PC-SPRACHAUSGABE-PROGRAMME** DM 20
Sprachausgabe für PC's. Ohne Hardwarezusätze kann Ihr PC beliebig lange Texte, in Deutsch, durch den PC Lautsprecher sprechen! Schnittstellen zu BASIC, Turbo Pascal und Assembler.

☐ **FLASHBAK - FESTPLATTENBACKUP** DM 20
professionelles, bequememes Festplattenbackupprogramm, mit Pull-Down Menüs, automatischem Formatieren der Sicherungsdisketten. Unentbehrlich für alle Festplattenbesitzer. **Mit ausf. dt. Anleitung.**

☐ **NEWKEY - TASTATURMAKROS - NEU V.5.2** DM 32
Der Cadillac unter den Macroprogrammen. Belegen Sie mit NEWKEY beliebige Tasten mit beliebig langen Texten oder Befehlen, wie Floskeln, Absender, Befehle, usw. Test in PD. **Mit ausf. dt. Anleitung.**

☐ **ROLAND SCHUFF-UTILITIES - VOLLVERSIONEN - 4 Disks** DM 48
Mit die besten **deutschen** Dienstprogramme. FNSK erlaubt es Ihnen beispielsweise, Ihren PC (z.B. die Tastatur durch Erstellen Ihres persönlichen KEYBGR) ganz nach Ihren Wünschen zu konfigurieren. I.A.M. optimiert den Interleavefaktor Ihrer Festplatte vollautomatisch und ermöglicht so drastische Leistungssteigerungen der Platte. Mit dem BIOS Editor entlocken Sie Ihrem PC die letzten Geheimnisse. PC-HACKER bietet viele Überraschungen. Deutsche Spitzenqualität.

☐ **NEU! LZEXE/SCRUNCH - EXE/COM KOMPRIMIERER** DM 12
Komprimiert fast alle .EXE oder .COM-Dateien auf 50% der Ausgangsgröße, wobei die Programme voll lauffähig bleiben. Eine feine Sache, besonders für Laptops.

☐ **DIGITIZED VOICE - Profi Sprachdigitalisierer** DM 24
BRANDNEU! Eines der wenigen brauchbaren Low-Cost-Sprachsysteme für den PC. Einlesen beliebiger Sprach- und Tonsignale über den seriellen Port. Der komplette Schalt- und Platinenätzplan für die nur wenige Mark kostende Elektronik ist auf der Diskette enthalten. Hintergrundton möglich. Der PC als perfekte elektronische Schneidemaschine. Ideal für Lern-, Demo oder Installations- oder beliebige andere Programme! Mit Schnittstellen für ASM, TP, C . Eine echte Ingenieurleistung. (für AT)

☐ **POWER MENUE 4.5** DM 12
Neue Version des sehr beliebten DOS-Menüsystems . Kaum ein Menüprogramm kann mit so vielen Funktionen aufwarten.

☐ **MAXI Disk 3.0** DM 49
Deutsch! Haben Sie auch genug von der sehr bescheidenen Speicherkapazität der MS/PC-DOS Disketten? Mit MAXI-Disk kennen Sie ab sofort keine Platzprobleme mehr. DER DURCHBRUCH BEI DOS-DISKETTENKAPAZITÄT! Exclusiv von Computer Solutions München gibt es jetzt ein Programm, mit dem die Kapazität von MS/PC-DOS Disketten drastisch gesteigert werden kann. Folgende Steigerungen sind möglich: 420 statt 360, 800 statt 720KB, 1,4 statt 1,2, 1,6 statt 1,44 MB! MAXI-Disk erlaubt es, auf ganz normalen PC's mit Diskettenkapazitäten zu arbeiten, die bisher für unmöglich gehalten wurden. Keinerlei Hardwareänderung oder Spezialdisketten nötig. Auf vielen Rechnern lassen sich sogar Systemdisketten der erhöhten Kapazität erstellen. Mehr läßt sich aus DOS-Disketten wirklich nicht herausholen! By Herne Data Systems.

☐ **OS/2-PROFESSIONAL - PAKET** DM 89
10 Disks für den "durstigen" OS/2 Anwender: DFÜ, Terminplaner, Archivierprogramm, Editoren, Datenbank, Paramid Utilities, etc.

☐ **NEUE UPDATES! MS-WINDOWS PROFIPAKET** DM 109
WINDOWS-Tools à la carte: u.a. Filelister, Command-Post, DFÜ-Programme, Terminverwaltung, Menüprogramm, Terminplanung und Tools, Tools... Damit macht WINDOWS so richtig Spaß. Jetzt neu 15 komprimierte Disks.

☐ **TURBO DISKCOPY TOOLS** DM 36
Sammlung mit der BESTEN, SCHNELLSTEN und BEQUEMSTEN Diskettenkopierprogramme. Kleinste Kopierzeit bis 5 Sekunden möglich! Feldmüller DISK-Tools 3.1 (rg.Gb. je 25.-) 3 Disks.

☐ **VMIX-386 MULTITASKING *NEUE VERSION 2.3*** DM 12
#134. Multitasking/Multiuser-AddOn für DOS und 8088 bis 80486 PC's, mit UNIX-Features. Mehrere Anwender können an einem PC arbeiten oder mehrere Programme werden simultan abgearbeitet! Spitzenprodukt!

☐ **PC-HARDWARE TESTPROGRAMME - BENCHMARK** DM 60
NEUE VERSIONEN DER PC-MAGAZINE BENCHMARKS EINGETROFFEN! Testen Sie Ihre PC-Hardware (Speicher, Festplatte, Floppy, Monitor, CPU, usw.) auf Herz und Nieren, spüren Sie Schwachpunkte auf und entdecken Sie Fehler bevor es zu spät ist. Benchmark-Tests der Profis vom PC-MAGAZINE (USA). Unentbehrlich für den Computerneukauf. So trennen Sie die Spreu vom Weizen! 5 Disks.

☐ **NEUE VER. 2.01: ANA-/TELEDISK/PROF.MASTERKEY 3.0 D** DM 24
Zwei raffinierte Disk-Tools: ANADISK der Bit-Beißer für Disketten knackt Formate, auch von NICHT-DOS Disketten. Das Disk-Tool für Anspruchsvolle. TELEDISK verwandelt Disketten in komprimierte Dateien, dies ermöglicht es z.B. kopiergeschützte Software oder ganze Systemdisketten in Dateien abzuspeichern und wieder zurückzuverwandeln. PMK 3.0D, das DOS-Disk Tool der Superlative, ist in der **brandneuen deutschen Version** enthalten.

DATENBANKEN

☐ **SENSATION! POP-DBF** DM 12
Speicherresidente (belegt nur 34KB!) dBase III/IV/Clipper datenkompatible Datenbank. Dateien beliebiger Größe können verarbeitet werden. Menügesteuert, Hilfefunktion. Umlaute werden verarbeitet. Superprogramm!

☐ **RW-FILE NEUE VERSION 3.2** DM 12
Deutsch. Die benützerfreundliche Bestsellerdatenbank von RW-Software. Mit RW-FILE lassen sich 99% der meistbenötigten Datenverwaltungsfunktionen, wie: Adressenverwaltung, Serienbrieferstellung, Einzelbrieferstellung, etc. viel schneller als mit manchen teuren kommerziellen Programmen erledigen. Viele wertvolle Detailfunktionen. Superprogramm.

☐ **WAMPUM - FOXBASE+ VERSION** DM 149
Was alle Wampum Anwender bisher vermißt haben, wird jetzt endlich wahr: dBase III interaktiver Modus und teilw. Programmierung. Alle Foxbase+ Erweiterungen stehen Ihnen zur Verfügung. WAMPUM Foxbase+ ist DIE Power-Datenbank nicht nur für den kleinen Geldbeutel! Diese Version ist glatt bis zu 6x schneller als die Datenbank, die Sie kennen! Keine Shareware, exclusiv von uns!

☐ **NEU! SHAREWARE WAMPUM - *VERSION 4.0!*** DM 12
Neue Version des Shareware-Bestsellers. Mit vielen neuen interessanten Funktionen, wie z.B. PowerBrowse, relationales Look-Up, etc. Das Update lohnt!

☐ **NEU! V.2.22 - SSQL-DATENBANK** DM 12
SSQL ermöglicht jedermann den preiswerten Einstieg in die aufregende Welt der SQL-Programmierung. Wenn Sie auch morgen noch mitreden wollen, hier zugreifen!

KALKULATION

☐ ***BRANDNEU!* ASEASYAS - VERSION 4.00n** DM 12
Völlig neue Version von AsEasyAs, der 1-2-3 kompatiblen Tabellenkalkulation. Jetzt mit vielen Erweiterungen, wie 8192 Zeilen, Worksheet-Link, beliebig erweiterbar durch Add-Ins und User-defined Functions (Welches andere Spreadsheet kann das?), direkter dBaseIII/IV Import/Export, Datenbankmasken, viele Stringfunktionen, Finanzfunktionen, viele neue Grafiktypen (auch mehrfach), umfangreiche Makrosprache, lineare Optimierung, Matrizen, 3-D, usw. Für CGA, EGA, Hercules, M24. Ausgabe nach 24-Nadel, HP-Laser und .PIC Files. Erweiterte On-Line-Hilfe.
☐ Reg. Version exclusiv von uns DM 168.-.

☐ **LOTUS 1-2-3® WORKSHEETS/TOOLS** DM 149
Power Worksheets! Statistik, Grafik, Datenverwaltung und Finanzen. Tips und Tricks. Werden auch Sie mit unseren vorgefertigten Arbeitsblättern 1-2-3 zum Lotus 1-2-3 Profi! 20 Disks. **Wir empfehlen hierzu unseren Lotus 1-2-3 Clone AsEasyAs.**

DFÜ-PROGRAMME

☐ **BRANDNEU: TELEMATE DFÜ V.2.02** DM 24
Senkrechtstarter der professionellen DFÜ-Programme, mit: SAA-Standard, Maus, Multitasking, Kontext-Hilfe, Editor, Backscroll, bis 115200Bd,15 Protokolle, View, umfangr. Scriptsprache, Makros, Terminalemulation, und und.. Wahnsinnsprogramm! 2 Disks.

☐ ***NEU:* xBTX - Grafikfähiger BTX-Softwaredecoder** DM 12
Vollwertiger BTX-Decoder mit voller Grafikdarstellung der BTX-Zeichen, nicht nur primitive Textdarstellung. Für CGA, EGA, VGA, Hercules. Für alle Hayes-komp. Modems.

☐ **TELIX TELECOM *NEUE VERSION: 3.12*** DM 24
Profi DFÜ Programm mit allen Schikanen, Script Compiler bis 64KB, viele Protokolle. Bis 115200 Baud! DFÜ-à la Carte. 2 Disks.

☐ **EQUAL-PARTNERS 5.0/ZIP 1.41/COMX/SFCOPY 2.0** DM 12
#115. Fünf superschnelle Programme zur seriellen Verbindung zweier PC's (bis 115200 Baud) auf der normalen DOS-Ebene miteinander. Die Laufwerke des einen Rechners können vom anderen PC wie dessen eigene Disketten oder Festplatte angesprochen werden.

☐ **RBBS-PC - MAILBOX - WIEDER NEUE VERSION 17.3** DM 60
Das ultimative Mailboxprogramm aus den USA, mit allen Schikanen (invl. QB-Quellcode). Die vielen Features und die ausführliche Doku machen RBBS-PC zum wohl meistverbreiteten Mailboxsystem. Profis aus allen Bereichen verwenden RBBS-PC! 6 Disks.

☐ **DFÜ-PROFIPAKET** DM 189
DFÜ-Programme vom Feinsten: QMODEM, PROCOMM, BOYAN-D4, RBBS-PC Mailbox, Magpie, GT Powercomm, KERMIT, ZCOMM, und DFÜ Tools! Ein unentbehrliches Paket für alle DFÜ-Anwender! 30 Disks.

ENTSPANNUNG/MUSIK

☐ **SCHACHBOX** DM 12
Die besten Shareware-Schachprogramme jetzt auf einer Diskette! 3-D CHESS, ED-CHESS, CYRUS und PC-CHESS. Für CGA/EGA und Hercules.

☐ **EGA-GAMES-PAKET-NORMALPAKET I.** DM 89
Die besten EGA-Games mit Super-Grafik: EGA- TREK, -RISK, -Blackjack, -DRAG, -Mahjongg, Card-Games, -Solitaire, Arcade, uvm... Alles in Super EGA-Grafik! Eine Augenweide nicht nur für Spielefreaks! Test in PD 6/89. 10 Disks.

☐ **EGA-GAMES-PAKET-SUPERPAKET II.** DM 149
Mit dem Inhalt von Paket I und zusätzlich die neuesten Superspiele aus den USA: BASSTOUR, Captain Comic, KUNG FU LOUIE KARATE, Mosaix, Cunning Football, NC Risk, EGA-Gammon, Poker Solitaire, Puzzle. 20 Disks mit Superspielen.

☐ **DIGITALE MUSIK + KOMPONIEREN** DM 89
NEU! Jetzt endlich mit Software zur **mehrstimmigen** Musikerzeugung. Verwandeln Sie Ihren PC in einen Kompositionscomputer! Die besten Musikprogramme von Bach bis Mozart. 10 Disketten.

GRAFIK UND CAD

☐ **AUTOCAD-PROFIPAKET** DM 149
Exclusiv von uns: Mehrere hundert Tools und Grafiken für AutoCAD, aus: 3-D Auto-LISP, Movies, Blocks, Demos, Zeichnungen, Füllmuster, Info, Menüs, Newsletters, Dias, Schriften, Utilities, Bemaßung, Editieren, Elektro, Elektronik, Datei-Management, Layers, und und. 20 komprimierte Disks für den den **echten** AutoCAD Anwender!

☐ **DTP - MEGA- BILDER-PACK- DER RENNER!** DM 149
DIE GRAFIKSENSATION ! Über 2000 Clipart Grafiken aus den Themen Geschichte, Technik, Tägliches Leben, Freizeit, Business, Symbole, etc. auf 30 komprimierten Disks (90% im gängigen PCX-Format, fertig zum Einlesen in alle DTP und grafikfähige Textprogramme. Machen Sie mehr aus Ihren Publikationen. Preiswerter geht es kaum noch! Dazu gratis Screenshot Tools und Konvertierprogramme.

☐ **SCREENSHOT/HARDCOPY/VIEW/CONVERTER 10 Disks** DM 89
Sammlung professioneller Screenshot- /Hardcopy/View- und Konvertprogramme für CGA/EGA/VGA/Hercules. Grafiken, Bilder und Masken übernehmen und usw.! Der Werzeugkasten für den DTP-Profi. Eine runde Sache.

☐ **EXCLUSIVVERTRIEB! 3D - 3DESIGNER** DM 24
Ungewöhnlich leistungsstarkes Programm zur wissenschaftlichen Untersuchung und grafischen Darstellung (Animation) von Daten und Funktionen aller Art. Beliebige Funktionen, wie z.B. f(x)= cos(x^3)/tan(x^2) lassen sich als 3-D-Rastergrafik ausgeben. Meßwertdaten können als 3-D Gebilde eingelesen und verarbeitet werden. Fraktalgenerator für fazinierende Apfalmnnchen. 3-D Objekte können mit dem Editor (hidden-line removal) manipuliert werden. Diashows. HGC/EGA/VGA, NEC und EPSON 24-Pin.

☐ **EPG - EASY PRESENTATION GRAPHICS** DM 12
Professionelles Grafik-Präsentationssystem. Vergleichen Sie EPG mit den besten kommerziellen Systemen und Sie werden von EPG begeistert sein. 12 Grafikgrundtypen, freier Text, 3-D Effekte, Pictogramme, etc. Über 226 Drucker werden unterstützt. Superdruckqualität auch auf 9-Nadeldruckern (240 dpi, Laser 300dpi.). Für CGA/EGA/VGA/VEGA7/Hercules, Laser-(HP)/9/24-Pindrucker, Plotter.

☐ **DANCAD3D - 3-DIMENSIONALES CAD** DM 48
Echtes dreidimensionales CAD System mit Animation in Farbe. Das System ermöglicht das Erstellen von 3-D Grafiken mit Animation. Der Kopf glaubt nicht, was die Augen sehen! **NEU für CGA/EGA/Hercules. Laser-(HP- und Postscript)/24-Pindrucker . Plotter. BESTSELLER! Neu 4 Disks.**

☐ **Elektronik Power Pack/Autorouting** DM 89
Die Programme für Elektroniker: SMARTROUTE/ELEKTRONIK (**Deutsch, Rest Englisch**), PC ROUTE 2.21/DROEGE/SCHEMATIC FILER/CCICAP/. Jetzt endlich Layout **mit Autorouting!** Filterberechnungen, Bode-Diagramme, Planen, Erstellen und Analysieren elektronischer Schaltungen. 10 Disks.

☐ **PC-DRAFT*-CAD-PROGRAMM UPDATE V.2.05** DM 12
Für anspruchsvolle CAD-Anwender. Alle bekannten Zeichenfunktionen und mehr: Makros, autom. Bemaßung, unlimitierte Layer, gruppierte logische Objekte, Zoom und Pan, gr. Symbolbibl., viele Schriften, direkte Schnittstelle zu GEM und DTP Ventura. Matrix, Plotter und HP-Laserjet Support bis 300 dpi. *(* kein PC-DRAFT von RHV)*

☐ **DRAFT CHOICE- CAD UPDATE V.1.4** DM 12
Objektorientiertes CAD Programm mit allen Schikanen, wie: autom. Bemaßung/Flächenberechnung, Beziers, Macros, Symbolbibl., etc. Für alle anspruchsvollen CAD Aufgaben bestens geeignet. Das beste Shareware CAD-Programm?! Für CGA/EGA/Hercules. Für EPSON, Laserjet und HP Plotter. Ein neues Meisterwerk der Autoren von AsEasyAs.

☐ **CASCADE 2D - CAD SYSTEM** DM 12
Deutsch! Version 3.14 oder neuer des CAD Bestsellers für CGA, EGA, VGA, HGC. CASCADE befriedigt auch anspruchsvolle CAD-Anwender: Multilayer, etc.. Ausgabe auf Drucker oder HP-Plotter. Eines der besten deutschen Sharewareprodukte! Ideal für Architekten, Planer und auch Hobby und Studium.

☐ **CAD PROFIPAKET 10 DISKS** DM 89
Paket mit fast allen Shareware CAD-Programmen. 10 CAD Programme plus PC-KEY-DRAW 3.71 (neu) im Paket. CAD Spezialisten bitte zugreifen.

LERNEN UND WISSEN

☐ **NEU! SOFT-TUTOR-AUTORENSYSTEM** DM 20
Exclusiv bei uns in Deutsch! Professionelles Autorensystem zur Erstellung von Lernprogrammen, Demos und Präsentationen. Der leistungsfähige Befehlssatz erlaubt die Gestaltung komplexer interaktiver Abläufe (für HGC). Superprogramm.

☐ **PFGEN 2.6** DM 12
Deutsch! Prüfungsfragengenerator für Lehrer. Erfaßt, verwaltet und mischt Ihre Prüfungsaufgaben. Nie war es bequemer diese lästige Arbeit zu erledigen!. Datenaustausch mit Lehrerkollegen möglich. PFGEN - ein Grund mehr für einen PC!

☐ **MATHE MADE EASY I - MATHEMATIKPROGRAMME** DM 89
Meist in Deutsch! Wohl die 10 besten Matheprogramme für Schüler, Studenten (Ideal für Matheprüfungen mit dem Laptop!), Lehrer, Ingenieure: MatheAss, TurboPlot , Kom-Math, ST17, Diskusp 2.1, Funktionsanimator, Kwikstat, Mathplot, XYPLOT, Mathpak! Mit: UPN-Rechner, Geometrie, (z.B. Kugelschnitte), Analysis, (z.B. Funktionenplott, Kurvendiskussion), umfangreiche Statistikfunktionen, Lineare Algebra und Komplexe Zahlen. HI-RES-Grafikausgabe CGA, EGA, Hercules, 9/24-Nadel-Druckfunktion. So lösen Sie Matheprobleme auf zeitgemäße Art! 10 Disks - Mathe auf bequeme Art.
☐ **MATHE MADE EASY II Profipaket mit 20 Disks DM 149.**
Dieses Paket enthält zusätzlich zum Paket I noch Programme für polynomniale Ausgleichsrechnung, Statistik, Algebra und FFT.

☐ **PC-MS-DOS - LERNPROGRAMM** DM 12
Deutsch! Speziell für PC Einsteiger. Das System führt Sie schrittweise in den Umgang mit Ihrem PC und DOS ein.

☐ **FB-TRANSLATOR 3.0 ENGLISCH-DEUTSCH** DM 12
Automatisches Übersetzungsprogramm Englisch-Deutsch. Mit Pull-Down Menüs, Editor, On-Line Hilfe, mit Wörterbuch für Computerbegriffe. Incl. Vokabeltrainer.

☐ **10FINGER - TASTATURLERNPROGRAMM** DM 30
Deutsch! 10FINGER ist die ideale Ergänzung zu unserem PC-Lernprogramm. In 180 didaktisch aufgebauten Lektionen lernen Sie auf amüsante Weise das Blindschreiben auf Ihrem PC. Sie können Ihr persönliches Tastaturlayout anpassen. Nur im Exclusivvertrieb von uns.

☐ **POP-HELP HINTERGRUNDHILFE-DEUTSCH** DM 20
Deutsch! Hintergrundhilfe für alle DOS-Befehle. Ihr perfektes DOS Handbuch auf Tastendruck. Nie wieder müssen Sie mühsam im Handbuch nachschlagen - POP-HELP hilft Ihnen aus jeder Verlegenheit! Siehe auch die Beschreibung bei den englischen Programmen. Eigene Texte können problemlos eingefügt werden! Nur im Exclusivvertrieb von uns erhältlich.

☐ **VOKABELTRAINER** DM 12
Deutsch! Bringen Sie Ihre Fremdsprachenkenntnisse mit mit unserem Vokabeltrainer für Französisch, Englisch, Italienisch wieder auf Vordermann.

Die (C)omputer Solutions (S)oftware (L)ibrary (CSL) bietet:

- *garantiert nur neueste Versionen (Wer hat sie schneller?)*
- *keine als Shareware getarnten Demoprogramme oder "Cripple-Ware"*
- *Lieferung auf europäischenSpitzendisketten*
- *Virengeprüfte Qualitätsprogramme direkt aus Autorenhand ...und noch viel mehr!*
- *professionelle Beratung und Service*

PROGRAMMIEREN

☐ **FranzBasic - BASIC-Übersetzer** DM 12
#1014. **Deutsch!** Komplettes, sehr schnelles BASIC-Entwicklungssystem mit vielen außergewöhnlichen Sonderfunktionen, wie: direkte Verarbeitung von dBase III Dateien (.dbf/.ndx), Maussteuerung, Menüprogrammierung, Auswahlboxen, moderner Full-Scren Editor mit Pull-Down, Autosave und vieles mehr! **Deutsches Spitzenprodukt!**

☐ **NEU! Deutsche TP Tools-Paket** DM 89
Programmier-Tools für Turbo Pascal von Feinsten: Multi-Tasking Subsystem, MENU-STAR, MASK-ED, MASKSTAR, HARM'S TP 5.0 TOOLS, VIO, HELPER, WINWORLD, Grafik Toolbox TP 5. Hier zeigt sich was deutsche TP-Programmierer können. 9 Disks.

☐ **TOP-CODE dBase Programmgenerator** DM 12
Deutsch! Sehr leistungsstarker Programmgenerator für dBase III und Foxbase. Mit Top-Code verkürzen Sie drastisch Ihre Programmentwicklungszeiten.

☐ **MODULA2-COMPILER V.2.0 Mit ausf. dt. Anleitung. - 2 Disks** DM 39
#148. Vollwertiger Modula 2 Compiler mit integrierter Entwicklungsumgebung, Editor, Make, Inline Assembler, MS-Link Format. TestChip Tool 6/89. 2 Disks.

☐ **A86/D86 + ASMWIZ - Neu V. 3.22.** DM 32
#114. Prof. Makroassembler m. Screen Debugger, erzeugt direkt .COM und MS-.OBJ-Dateien. Weitgehend MASM kompatibel. **Nur bei uns mit deutscher Doku für A86 und D86. Gratis dabei ASMWIZ die ASM-Superlibrary mit 101 Funktionen. Profis greifen zu!.**

☐ **NEU! Mdebug - High Performance Debugging Tool** DM 12
#1020 **Exclusivvertrieb. Deutsch!** Leistungsfähiges, residentes Debuggingsystem/Monitorprogramm. Mdebug erlaubt die bequeme Fehlersuche und Analyse in Programmen aller Art. Mdebug bietet dabei Möglichkeiten, die sonst nur bei weit teureren Debuggern zu finden sind. Ein deutsches Programmiertool der Spitzenklasse. Mdebug wurde mit A86 entwickelt!

☐ **Assembler Helper & Tutor - Der Assembler Privatlehrer** DM 12
Ideal zum Lernen der 80x86 Assemblersprache. Ihr Einstieg in die faszinierende Welt der Assemblerprogrammierung. Viele Beispielprogramme! Ersetzt teure Assemblerbücher.

☐ **HELP-PC (INSIDERDISK)** DM 12
Diese Diskette enthält Retrievalsystem (ca. 500KB Information), mit dem gesamten PC-Wissen eines der fähigsten US-Programmieres. Hier finden Sie, was in keinem Buch steht, undokumentierte DOS-Funktionen, Tips und Tricks, Bugs, etc. Sorgfältig aufbereitete Hard-Core-Informationen für echte Programmierer.

☐ **MD86 - DISASSEMBLER** DM 12
Einer der besseren Disassembler zum Rückübersetzen von DOS Programmen. Für Leute, die wirklich hinter die Kulissen von Profiprogrammen schauen wollen. Erzeugt direkt assemblierbaren MASM-Output.

☐ ***UPDATE:* C-Task 2.0/OS88 Multitasking für TC und MSC** DM 24
#74/763. Programmieren von echten Multitaskingprogrammen unter DOS. Mit Real-Time-Task Monitor. MSC/Turbo C. C-Task mit Source. 2 Disks.

☐ **NEU VER. 1.2c - PERSONAL C - COMPILER** DM 32
#477. Der ideale C Compiler nicht nur für Einsteiger, Vorläufer des berühmten Desmet C Compilers. Superschnelle, kompakte Programme, Inline-Assembler. Ihr Einstieg in die professionelle C Programmierung. Keine "Tiny"-Version. Test in PD 6/89. **Mit dt. Handbuch (Paperback).**

☐ **PERSONAL C SCREEN** DM 12
#977. **Deutsch!** Screen Manager (Library) speziell für den Personal C Compiler. Bequemes Erstellen von Masken und Fenstern. Superschnell.

NEWS NEWS NEWS NEWS

☐ **CXL- Professionelle C Programmierbibliothek** DM 39
CXL ist anerkanntermaßen eine der besten C Bibliotheken für MSC und Turbo C. Über 220 Funktionen aus allen Bereichen (System, Window, Maus, I/O Routinen, uvm.) machen CXL zum unentbehrlichen Freund des C-Einsteigers, wie des C-Profis. Im Nu schreiben Sie Programme mit professionellem Outfit und Inhalt. Wir kennen keine bessere C-Bibliothek! 2 Disks. **Mit 170 Seiten dt. Anleitung.**

☐ **TesSeRact (Res.Programme)** DM 12
Einheitliche Programmierschnittstelle für residente Programme. Erlaubt das fehlerfreie Programmieren residenter Programme in ASM, TC, MSC und TP 5.0.

☐ **Turbo Pascal 5 Profi-Paket 30 Disks** DM189
☐ **Turbo Pascal 5 De-Luxe-Paket 50 Disks** DM 299
Nachschub für echte Turbo Pascal 5 Programmierer! Mit: Litecomm Communications Tools, PXL Crossreference Programm, Turbo Technojocks Toolkit, Turbo Overdrive Package TP zu TC Translator mit Source, PIBTERM, EZ-DOSS Manager, Datei(de)kompression (ARC), TP Tutor, Boosters, TP4MENU, TP 5.5 Multitasking Tools uvm. Sourcecode und Units. Tausende begeisterte Anwender arbeiten bereits damit.

☐ **dBase III/CLIPPER Prof. Tools** DM 299
ACHTUNG: Brandneue Version unseres Bestsellers (Update möglich). 50 Disks professioneller dBase III Programme und Tools. Fast alle Disketten sind nunmehr komprimiert, d.h. für Sie noch mehr Programme für's Geld. Inhalt: z.B. Interrupts, Assembler, C und dBase III, Programmgeneratoren, Adressenverwaltung, dBIII Dokumentator, Tools, Tools, Tools, usw... Kurz das Non-Plus-Ultra für alle dBase III/Clipper/Foxbase+ Profis und solche, die es werden wollen.
☐ UPDATEPREIS bei Einsendung des alten Paketes nur DM 149

COMPUTER SOLUTIONS SHAREWARE

SHAREWARE
IMMER ZWEI SCHRITTE VORAUS.

☐ Turbo C PROFIPAKET 30 DISKS	DM 189.-
☐ Turbo C PROFIPAKET 50 DISKS	DM 299.-
☐ MICROSOFT C PROFIPAKET 30 DISKS	DM 189.-
☐ MICROSOFT C PROFIPAKET 50 DISKS	DM 299.-

C Programmierpakete speziell für TC u. MSC
C Routinen (Source und Libraries) für fast alle Einsatzgebiete: Windowtechnik, Serieller Port, C-Tutor, TP zu C-Übersetzer, Grafik, Sound, Joystick, Maus, Multitasking, Dateimanagement, DOS, uvm! Diese Pakete enthalten sämtliche verfügbaren C Shareware-Tools mit mehreren hundert Routinen für alle Einsatzgebiete.
WICHTIG: Unbedingt gewünschte Version ankreuzen:!

☐ **MICROSOFT-QUICKBASIC Tools 10 Disks** DM 89
Eine Sammlung nützlicher Routinen und Anwendungen für Ihren Quickbasic Compiler, u.a. einen Fullscreen-Editor mit Source! Sehr umfangreiche Funktionsbibliotheken für Windowing, String, Mouse, etc. Ein Muß für QB-Programmierer!

ACHTUNG SHAREWAREAUTOREN!

Als eine der weltweit größten Shareware-Bibliotheken, suchen wir ständig gute und neue PC-Software. Interessant sind alle nicht-trivialen Programme, beliebiger Themenbereiche. Falls Sie an einer Zusammenarbeit mit Deutschlands Sharewareprofi interessiert sind, sollten Sie uns Ihre fertig konfektionierten Shareware oder PD-Programme (bitte keinesfalls Demos oder Pseudo-Shareware-programme mit eingeschränkten Datenmengen oder Ähnlichem. Für Rückfragen rufen Sie uns bitte an.

TEXTVERARBEITUNG-DRUCKEN

☐ **DTP - Rubicon Publisher 2.0** DM 36
Befehlsorientieres Satzsystem à la TeX, mit genauer Kontrolle der Textformatierung, so wird auch in professionellen Druckereien gearbeitet. Für HP Laserjet II (Fonts dabei) und Postscript. 3 Disks.

☐ **TEXTER-PC 2.1 - Multifonttextverarbeitung** DM 24
Deutsch! Die neue exclusive Textverarbeitung mit dem echten Plus an Features: Zeichenfunktion mit Kreuzungsautomatik, Blockoperationen mit Spaltenmodus, menügesteuert, Dateimanager, Grafikmodus mit 11 Zeichensätzen (!) (ähnlich Bradford , aber mit Umlauten), Rechnen im Text, Programmierermodus, Textlänge nur durch RAM begrenzt, sehr schnell hochauflösender Treiber für 9/24-Nadeldrucker (NEC/EPSON). Da können viele Profi-Programme nicht mithalten.

☐ ***SHAREWARE SENSATION:* QFONT 1.15B - HP FONTEDIT** DM 24
Exclusiv: 2 voll professionelle Editoren für HP Softfonts. Schriften lassen sich beliebig verändern (über 72Pt.) . Funktionen: Zoom, Blickpunkt Scrolling, Fett-, Outline- und Proportionalschriften, Invertieren, Vergrößern und Verkleinern. Incl. Beispielfonts. Ein Muß für jeden Laserdrucker. Wir kennen keine besseren Schrifteneditoren!Neueste Versionen. 2 Disks.

☐ **Neue Ver.: 1.05 Pro-Cr - OCR-Software - Schrifterkennung** DM 12
Exclusiv! Jetzt gibt es endlich eine professionelle OCR-Software auch als Shareware. PRO-CR liest Ihre Texte direkt vom Scanner ein (HP-Scanjet) oder aus TIFF-Files, die von jedem Scanner (auch HANDY-SCANNER, SCAN MAN, etc.) erzeugt werden. Pro-Cr erzeugt in Windeseile ASCII-Dateien. Ein Schrift-Lernmodus ist eingebaut. OCR-Programme kosten oft über 3000 DM. PRO-CR zeigt, daß es auch preiswerter geht.

☐ **NEU: POSTSCRIPT-FONTS** DM 12
DIEGO und HEIDELBERG sind zwei wunderschöne, vollwertige Postscript-Schriften, fertig für Pagemaker, Ventura, etc. Normalerweise kosten solche Fonts jeweils über DM 600!

☐ **DRUCKER/GRAFIKKARTEN - FONTEDIT** DM 48
Klasse Schrifteneditoren für Grafik-Karten, NEC- und EPSON-24-Nadeldrucker. Erstellt mühelos beliebige Zeichensätze. Inclusive komplettem kyrillischem Zeichensatz. Multifont-Textverarbeitung zum Spartarif! 4 Disks

☐ **PC-TEXT 2.0 Textverarbeitung/Adressen** DM 12
Deutsch! Starke Textverarbeitung mit integrierter Adressverarbeitung, Serien- und Einzeladressenfunktion und vielem mehr. Mit den besten Testergebnissen. R. Otter.

☐ **GALAXY WORD - TEXTVERARBEITUNG V. 2.42** DM 32
Extrem schnelles Textsystem der vierten Generation mit Pull-Down Menüs, Macros, Spellchecker, WordStar kompatibel! Textlänge bis über 200KB im RAM. Volle Druckersteuerung, auch Laser! Ein voll professionell einsetzbares Produkt! **Mit ausf. dt. Anleitung.**

☐ **HP-LASERJET-DTP-TOOLKIT** DM 189
30 Disketten-Set für den Laserjetanwender. Dieses Toolkit beinhaltet: 5 nützliche Utilitydisketten (z.B. res. Printer Control, Barcodedruck, Screen Capture, WP Macros, usw..). Dazu Fonts für MS-WORD. Für Ventura Publisher 6 Disketten Fonts von 8 bis 60 Pt Größe in verschiedenen Schriftarten! GEM Clipart für Ventura und GEM. Postscript- und BIT-mapped Zeichenprogramme mit GEM IMG Export. Laserjet Besitzer bitte zugreifen! 30 Disks.

☐ **WORD-PERFECT 5.0 TOOLS** DM 89
Voller raffinierter Tools für WordPerfect 5.0: Macro-Sammlungen, Editor, GIF Import, Grafikbilder zum Einlesen in Dokumente, PTR-Tools, WordPerfect Tutor, uvm. Kein WP-Anwender sollte darauf verzichten müssen! 10 Disks.

☐ **LQ-PRINT-QUALITÄT FÜR 9-NADEL DRUCKER** DM 12
Nun können Sie mit Ihrem "alten" 9-Nadeldrucker endlich auch in Briefqualität drucken. Und das auch noch in vielen verschiedenen Schriftarten. Auch für viele 24-Nadeldrucker verwendbar. Mit grafischem Briefkopf Generator und Plakatdruck. **SUPER! BESTSELLER!**

☐ **PRINTER-CONTROL** DM 12
Vier Druckertools der Spitzenklasse: Unbegrenzter Druckspooler mit Umleitung auf eine Datei, Seitwärtsdruck (auch resident), Sidewriter und Diskspool.

☐ **PC-OUTLINE - DEUTSCH** DM 12
Endlich! Die offizielle deutsche Version des wohl besten Outliningprogrammes (Struktur- und Entwurfsplaner) auf dem Markt. Arbeiten auch Sie mit dem Ideenprozessor des Jahres!

Computer Solutions Software GmbH

POSTFACH 1180 - D-8018 GRAFING/MÜNCHEN - KEIN LADENVERKAUF

Tel: 08092/5018   FAX: 08092/31727   BTX: 080925018 (Teilnr.)

Achten Sie zu Ihrer Sicherheit unbedingt auf Original-Disketten mit dem Computer Solutions Aufkleber!

Ladenverkauf (kein Versand) bei folgenden Firmen:
6000 Frankfurter Fachbuchhandlung M. Kohl,Roßmarkt10, 069/29890492
6800 Mannheim, Fach-Buchhandlung Löffler, B1,5, Tel. 0621/107830
8000 München, Columbus Datentechnik, Clemensstr. 81, Tel. 089/3086832
8012 Ottobrunn, Computershop Ottobrunn, Laurinweg 14, Tel.089/6098639

Bestellschein

Bitte ankreuzen und einsenden an: Computer Solutions, Postfach 8018 Grafing/M.:
Liefern Sie uns per/gegen Bar ☐ Scheck ☐ Nachnahme ☐ Rechnung ☐
Lieferung gegen Rechnung nur bei Großfirmen und Institutionen. Ausland nur EUROSCHECK bar ausgestellt (nicht zur Verrechnung!) oder BAR. Firmenbestellungen bitte mit Firmenpapier ausführen und Bezug auf diese Anzeige (Heftname) nehmen. Es gelten unsere Geschäftsbedingungen. Angebot freibleibend. Änderungen vorbehalten.

PREISE:
1-9 Disks nur DM 12.-/ 10-19 St. DM 10.- / 20-29 St. DM 8.-/ ab 30 St. 7.-
Bei Lieferung auf 3,5 Zoll 3 DM Aufpreis pro Diskette ☐

Programmpakete sind von der Rabattstaffel ausgenommen!
Summe: ________ DM+ 6 DM Versandkosten = ____________ DM

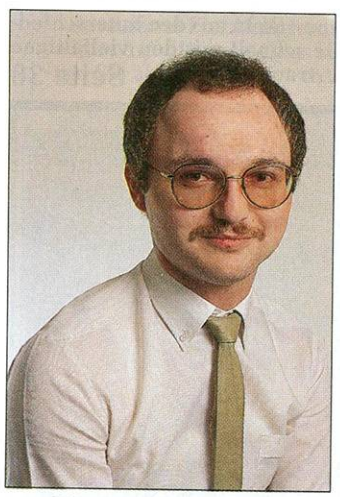
Ihre Kundennummer bei uns: ____________
(Unbedingt angeben, falls bekannt oder ?, falls nicht parat.)

Name: ____________
Firma: ____________
Straße: ____________
PLZ/Ort : ____________
Telefon: Tags (______) ________ Datum: ____ . ____ . 1990

DOEX 11/90

Der (un-)heimliche Standard

Fast schon historisch muten »Erfahrungsberichte« alter Computerhasen an, wenn sie mit Einsteigern in die hohe Kunst des Programmierens über ihre eigenen Anfänge plaudern. Ein beliebtes Thema ist in jedem Falle die Schilderung des »endlosen« Programmentwicklungskreislaufs mit den Tätigkeiten Editieren, Compilieren, Linken und Testen. Heutige Programmiereinsteiger können den mit dieser Situation verbundenen Ärger und dessen Tücken kaum mehr nachvollziehen: Integrierte Programmentwicklungssysteme machen ihnen das Leben wesentlich leichter. Eines der ersten dieser Systeme auf dem PC basierte auf der Programmiersprache Pascal und erschien 1983 auf dem Markt: Turbo Pascal. Eine große Verbreitung war Pascal bis zu diesem Zeitpunkt nicht beschieden: Der typische Computereinsteiger beschäftigte sich mit einem der zahlreichen Basic-Dialekte, und wenn die Anforderung hinsichtlich der Ausführungsgeschwindigkeiten der Programme es notwendig machte, wandte man sich eben einem Assembler zu. Pascal war als Lehrsprache für strukturiertes und modulares Programmieren entwickelt worden und deshalb bis dato nur an Hochschulen und Universitäten verbreitet. Das sollte sich jedoch schnell ändern. Schon bald erschien Turbo Pascal 3.0 und gewann innerhalb kürzester Zeit eine zahlenmäßig nicht zu unterschätzende Anwender-, das heißt Programmiererschar. Verständlich, lag doch zum ersten Mal eine sensationell zu bezeichnende Kombination aus Editor, Compiler und Linker vor, mit dem sich in einer bis dahin kaum vorstellbaren Geschwindigkeit kompakte und schnelle Programme entwickeln ließen. Und dabei handelte es sich bei Turbo Pascal selbst keineswegs um einen »Programmgiganten«; lediglich etwas mehr als 30 KByte Arbeitsspeicher beanspruchte die Version 3.0. Inzwischen sind die Anforderungen an den Arbeitsspeicher Ihres Computers allerdings deutlich gestiegen: Turbo Pascal 5.5 gibt sich mit 30 KByte nicht mehr zufrieden. Doch auch die Leistungsfähigkeit dieser Version ist in mindestens demselben Maße angewachsen. Eine menügesteuerte, leicht zu bedienende (weil übersichtliche) Benutzeroberfläche, ein situationsabhängiges Hilfesystem und ein sehr hilfreicher integrierter Debugger sind einige Schlagworte dieser Compiler-Version. Die konsequent auf die Forderungen und Wünsche der Programmierer abgestimmte Entwicklung von Turbo Pascal ging und geht unaufhaltsam weiter. Daher nimmt es auch kaum Wunder, daß Turbo Pascal längst zum Standard dieser Programmiersprache avanciert ist.



Ihr

Johann Zieger

Eingabehilfe

SCheck 2.02 6
– *Checksummer für sichere Listingeingabe*

Befehlsreferenz

Die Standardprozeduren von Turbo Pascal im Überblick 10

Arithmetische Funktionen 18

Der Sound gehört zum guten Ton 20

Der Weg nach außen 22
– *Ein- und Ausgabebefehle*

Die Grafikbefehle von Turbo Pascal 24

Die Variablen in Turbo Pascal 32

Turbo Pascal und die Dateiverwaltung 33

Durch und durch strukturiert 37
– *Kontrollstrukturen in Turbo Pascal*

Grundlagen

Wie sieht ein Pascal-Programm aus? 38

Dateiverwaltung unter Turbo Pascal 42

Die Grafikprogrammierung in Turbo Pascal 44

Bringen Sie dem PC die Flötentöne bei 54

Datenstrukturen nutzen in Turbo Pascal 58

Rekursive Algorithmen unter Turbo Pascal 64

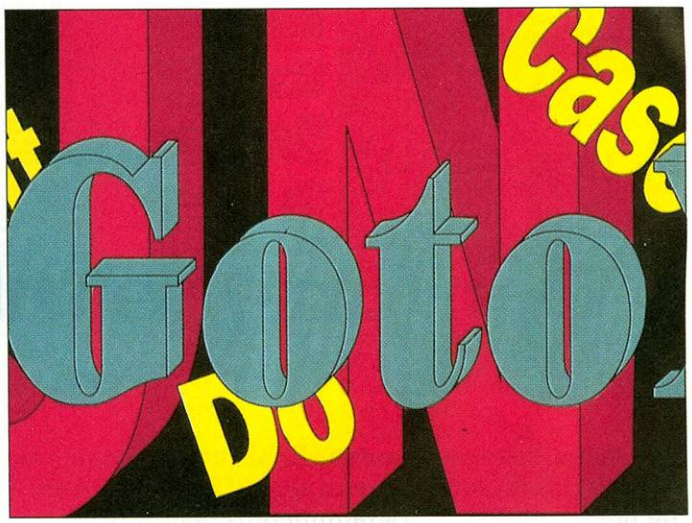
Kontrollstrukturen unter Turbo Pascal 70

Modularisierung mit Units 72

Overlays mit Turbo Pascal ab Version 5.0 76

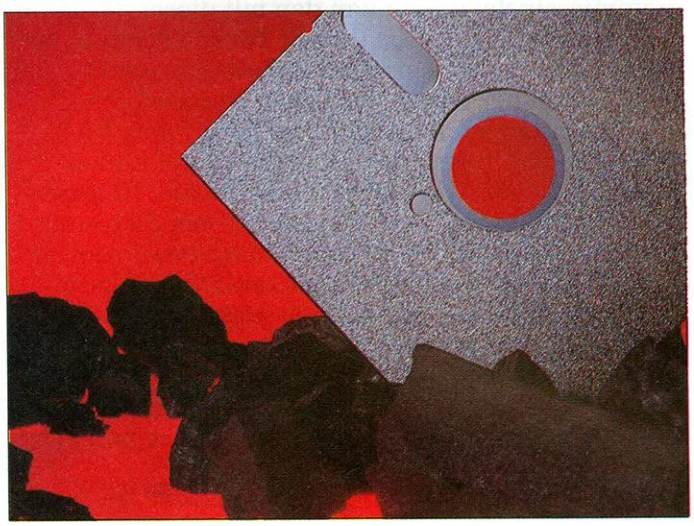
Der Diskettendoktor hilft in allen Lebenslagen 80

Pascal von A bis Z 88



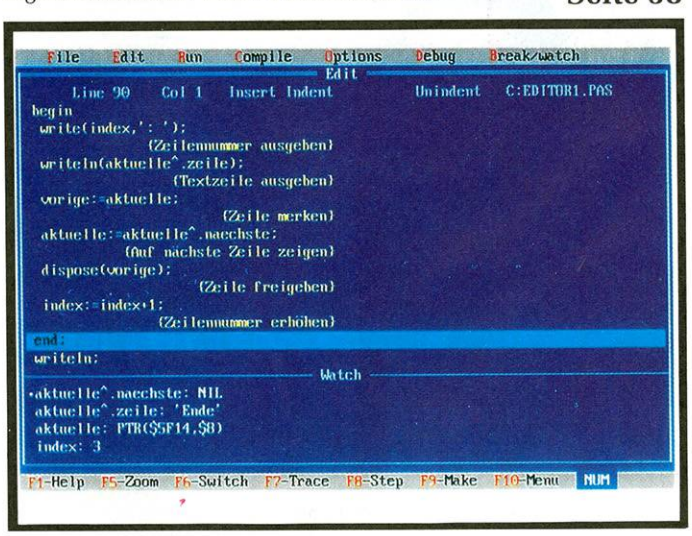
Der Referenzteil dieser Ausgabe bildet ein ideales Nachschlagewerk für alle Turbo-Pascal-Programmierer oder die, die es werden wollen. Kurz und bündig erfahren Sie in tabellarischer Form alles Wissenswerte über die Besonderheiten von Turbo Pascal.

Seite 10



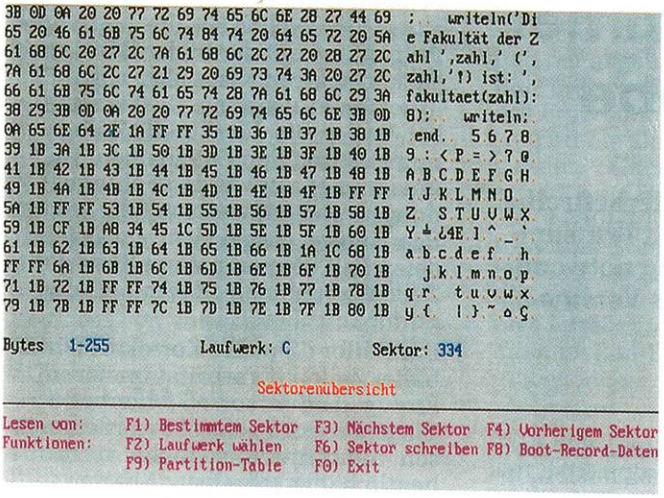
Mit den umfangreichen Grundlagenartikeln aus den unterschiedlichsten Fachbereichen werden Sie schnell mit den vielfältigen Eigenschaften von Turbo Pascal vertraut.

Seite 38



Mit dem integrierten Debugger steht Ihnen ein außerordentlich leistungsfähiges Werkzeug zur Fehlersuche in Programmen zur Verfügung. Daher finden Sie in dieser Ausgabe eine ausführliche Anleitung zu dieser Programmierhilfe.

Seite 124

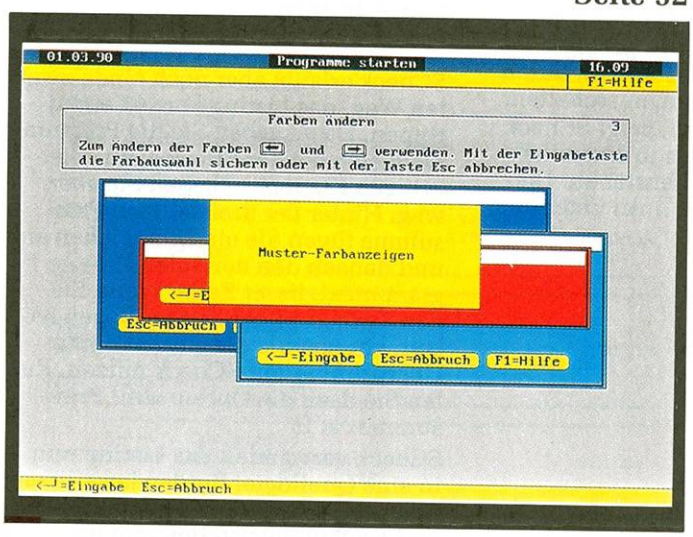


Spätestens beim Auftreten eines Lesefehlers sind Kenntnisse der Datenstrukturen einer Diskette wichtig. Um zu vermeiden, daß Sie einer solchen Situation völlig hilflos gegenüberstehen, zeigt Ihnen der »Diskdoktor« viele interessante Details Ihrer wichtigen Massenspeicher.

Seite 80

Interessante Einblicke in die Funktionsweise Ihres Computers, bezogen auf die speziellen Eigenschaften von Turbo Pascal, erwarten Sie ab

Seite 92



Bereits vor mehr als 10 Jahren definierte IBM den SAA-Standard. Bei der Entwicklung professioneller Software sollten Sie über die wesentlichen Grundlagen und Eigenschaften Bescheid wissen.

Seite 127

Pascal Intern

Management statt Bürokratie 92
– *Speicherverwaltung unter Turbo Pascal*

Assemblerunterstützung für Turbo-Pascal-Programme 97

Die Schnittstelle zwischen Turbo Pascal und Maschinensprache 103

Kurze Unterbrechung bitte! 109
– *Aufruf von Interrupts*

Der Zugriff auf das »Innenleben« Ihres Computers 113

Interrupt-Handler in Pascal 118

Software-Entwicklung

Bringen Sie Ihren Computer zur Raserei 122
– *Beschleunigung von Programmen*

Dem Fehler auf der Spur 124
– *Arbeiten mit dem integrierten Debugger*

Benutzeroberflächen im SAA- Standard 127

Listings

HPGL-Extender 136
– *Die preiswerte HPGL-Erweiterung*

Wie wär's mal mit Komfort? 145
– *SAA-Blockbefehle für Turbo Pascal*

Die Speicherverwaltung unter MS-DOS 152

Rubriken

Editorial 3
– *Der (un-)heimliche Standard*

Bücher zu Turbo Pascal 165

Impressum 166

SCheck 2.02: Checksummer für sichere Listing-Eingabe

Die Programme in der DOS International und der Sonderheftreihe DOS Extra werden nach einem speziellen Verfahren mit Prüfsummen versehen und abgedruckt. Das für die Überprüfung notwendige Prüfsummenprogramm »SCheck« liegt jetzt auch als Version 2.02 für GW-Basic und SBasic vor.

Alle in der DOS International und DOS Extra abgedruckten Programme sind mit Prüfsummen und Zeilennummern versehen. Jede Programmzeile beginnt mit einer dreistelligen Prüfsumme. Sie ist in eckige Klammern gesetzt und wird beim Abtippen nicht mit eingegeben. Nach der Klammer folgt ein Leerzeichen, und darauf der eigentliche Text der Programmzeile (nur bei Basic-Programmen) oder eine Zeilennummer, die Sie ebenfalls nicht mit eingeben, und darauf erst der Programmtext.

Nur diesen letzten Teil der abgedruckten Zeile geben Sie beim Abtippen ein.

Bei allen Programmzeilen wird im Abdruck nach jeweils 55 Zeichen ein Zeilenumbruch vorgenommen. Die umgebrochene Teilzeile erhält dabei keine Prüfsumme. Bei der Eingabe geben Sie umgebrochene Zeilen in einem Stück ein, drücken also nicht die Enter-Taste, wenn ein Zeilenumbruch im Ausdruck steht, sondern nur, wenn eine neue Zeilennummer auftaucht.

Beim Abtippen von Programmen sind Tippfehler kaum zu vermeiden. Mit dem Checksummenprogramm »SCheck« überprüfen Sie die abgetippten Listings auf eventuelle Fehler. Das Programm erlaubt nicht nur die Diagnose von fehlerhaft eingegebenen Zeilen, sondern mittels eines integrierten Editors auch gleich die Korrektur der fehlerhaften Zeilen. SCheck wird unter GW-Basic mit

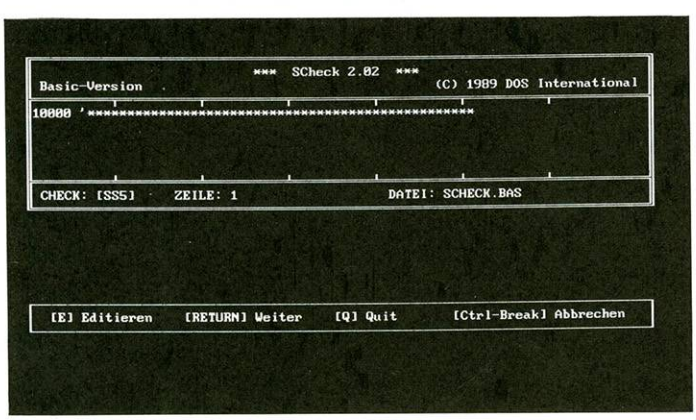
```
run "scheck"
```

gestartet. Als erstes wählen Sie die gewünschte Prüfsummenversion. In der vorliegenden DOS-Ausgabe brauchen Sie immer »SCheck Version 2.0«. Nur wer Programme aus Ausgaben der DOS International vor Oktober 1988 eingeben will, muß den ebenfalls integrierten Prüfsummenalgorithmus »SCheck Version 1.x« aufrufen. Danach geben Sie den gewünschten Prüfmodus – normalerweise »Standard« – an und abschließend den Programmnamen des Listings, so wie Sie es auf dem Datenträger abgelegt haben.

Achtung, Basic-Listings müssen dabei im ASCII-Format vorliegen und dürfen nicht im Token-Format gespeichert sein. Geben Sie dazu im GW-Basic-Interpreter beim Speichern des Programms »name« den Befehl

```
SAVE"name",A
```

ein, um dieses Format zu erzwingen. Zuvor sollten Sie sich aber sicherheitshalber eine Kopie Ihres Listings anfertigen. SCheck arbeitet zwar nicht mit dem Original, sondern mit einer temporären Zwischendatei und verändert die Originaldatei erst ganz zum Schluß, nach dem erfolgreichen Abschluß der Arbeit. Dennoch sollten Sie auch ein noch so geringes Risiko eines Datenverlusts, etwa durch Fehler beim Speichern, ausschalten. Wie Sie im Bild sehen, zeigt SCheck die aktuelle Zeile stets in einem besonderen Bildschirmfenster an. Die Prüfsumme erscheint links unter dem Fenster. Beim Überprüfen Ihres Listings vergleichen Sie die angezeigte Prüfsumme mit der in der DOS-Ausgabe in eckigen Klammern abgedruckten Prüfsumme. Rechts daneben finden Sie die aktuelle Zeilennummer und den Dateinamen. Im unteren Fenster werden Fehlermeldungen und Hinweise auf die weitere Bearbeitung ausgegeben. Im abgebildeten Fall können Sie mit [Enter] die Zeile übernehmen oder mit [E] den Editor aufrufen.



Der Arbeitsbildschirm von SCheck 2.02 in Aktion

Der Editor dient der Korrektur fehlerhafter Zeilen. Er arbeitet immer im Einfügemodus, so daß fehlerhafte Eingaben extra gelöscht werden müssen. Sie können also nichts versehentlich überschreiben. Mit den Cursor-Tasten steuern Sie den Cursor im Editierfenster. [Home] bringt den Cursor an den Anfang der angezeigten Zeile, [End] entsprechend ans Ende.

Löschen können Sie das Zeichen links vom Cursor mit der großen Löschtaste oberhalb der Enter-Taste, [Del] entfernt das Zeichen unter dem Cursor. Mit [Enter] übernehmen Sie die korrigierte Zeile. Steuerzeichen wie zum Beispiel Tabulatoren dürfen in einer Zeile nicht verwendet werden. Haben Sie so das gesamte Programm oder auch nur »verdächtige« Teile davon durchgearbeitet, so speichert ein letzter Druck auf die Enter-Taste das korrigierte Listing. SCheck wird beendet, und das fehlerfreie Programmlisting steht zu Ihrer Verfügung.

Wenn Sie die Prüfsummen mit abtippen, kann SCheck Ihr Listing automatisch überprüfen

SCheck erlaubt aber noch einen zweiten Weg, das Listing korrekt einzutippen. Dabei geben Sie die Prüfsumme gleich bei der Eingabe ein. Die eckigen Klammern lassen Sie aber weg. Hinter der dreistelligen Prüfsumme fügen Sie ein Leerzeichen ein und danach den normalen Programmtext dieser Zeile. Wenn Sie den Text so aufbereitet vorliegen haben, können Sie die automatische Überprüfung mit SCheck nutzen. Rufen Sie dazu die Option »Mit Prüfsumme« auf.

SCheck verwandelt das Listing nun in eine Quelldatei für den jeweiligen Compiler oder Interpreter, indem die drei Prüfsummenstellen und das Leerzeichen vom Rest der Zeile abgetrennt werden. Aus der so entstandenen Programmzeile berechnet SCheck die Prüfsumme und ver-

gleicht sie mit der zuvor abgetrennten Summe, die Sie eingegeben haben. Falls ein Unterschied auftritt, haben Sie Gelegenheit, die fehlerhafte Zeile manuell zu korrigieren.
Nach einem solchen Durchlauf von SCheck steht wieder das fehlerfreie Programm, diesmal allerdings ohne Prüfsumme, auf der Diskette. Auch hier gilt, daß Sie nicht mit dem Original arbeiten sollten, sondern nur mit einer Kopie, um im Zweifelsfall nochmal von vorne beginnen zu können.
SCheck verfügt aber noch über eine weitere Option. Die von Fall zu Fall abgedruckten hexadezimalen Listings können Sie nämlich mit SCheck im Binärmodus direkt in lauffähige Programme umwandeln. Dazu müssen Sie das Hexdump-Listing mit einem ganz normalen Editor (zum Beispiel Turbo Pascal, Wordstar oder Edlin) als Textdatei eingeben. Anschließend überprüfen Sie mit SCheck zunächst die Korrektheit der abgetippten Daten wie bei jedem anderen Listing. Befindet sich das Programm korrekt auf Diskette, so wird die Option »Hexdump« aufgerufen. SCheck wird nun dazu veranlaßt, eine Hexdump-Textdatei in ein ablauffähiges Programm umzuwandeln. SCheck erzeugt in jedem Fall eine Datei mit der Namenserweiterung »bin«. Um daraus eine lauffähige Datei zu machen, müssen Sie sie mit dem Ren-Befehl von MS-DOS umbenennen:

```
ren <name>.bin <name>.com
```

Falls eine Exe-Datei erzeugt werden soll (bitte entnehmen Sie dies der jeweiligen Programmbeschreibung) lautet der entsprechende Befehl:

```
ren <name>.bin <name>.exe
```

Sie sehen, mit SCheck 2.02 ist das korrekte Abtippen von Programmlistings kein bißchen komplizierter geworden – aber durch einen verbesserten Prüfsummen-Algorithmus um einiges sicherer. (zi)

SCheck mit SBasic compilieren

Um die Eingabehilfe SCheck mit dem SBasic-Compiler compilieren zu können, müssen Sie lediglich aus Programmzeile 10070 die Zeichenfolge »key off: « löschen und die Zeile 10720 sowie 11240 vollständig entfernen. Schließlich müssen Sie noch die Programmzeile 11170 in

```
11170 GOTO 10750
```

umändern und die Zeile

```
10735 IF EOF(1) THEN 11160
```

neu in das Programm einfügen.

Name	SCheck.bas
Sprache	GW-Basic und SBasic
Grafikkarte	jede
Besonderheiten	Checksummer für Listings in DOS International und DOS Extra

```
[SS5] 1ØØØØ '*****************************************
      *******
[F29] 1Ø01Ø '* SCheck - Prüfsummengenerator für DOS-Lis
      tings *
[ML4] 1Ø02Ø '*
            *
[8Ø7] 1Ø03Ø '*           Konzeption: Volker Everts
            *
[AC7] 1Ø04Ø '*         Basic-Version: Markus Zietlow
            *
[XS5] 1Ø05Ø '*****************************************
      *******
[4UØ] 1Ø06Ø '
[FL7] 1Ø07Ø KEY OFF: DEFINT A-O,Q-Z: DEFSNG P: COLOR 7,
      Ø,1
[5G8] 1Ø08Ø T = 1: TEMP = 2: F = 3'Ein-/Ausgabekanäle f
      estlegen
[332] 1Ø09Ø FALSE = Ø
[9Ø2] 1Ø1ØØ TRUE = -1
[EY7] 1Ø11Ø ZEICHEN$ = "Ø123456789ABCDEFGHKLMNPRSTUVWXY
      Z"
[9H2] 1Ø12Ø LAUF = FALSE
[6B2] 1Ø13Ø BIN = FALSE
[851] 1Ø14Ø CLS
[9S1] 1Ø15Ø PRINT
[KØ7] 1Ø16Ø PRINT CHR$(2Ø1); STRING$(78, 2Ø5); CHR$(187
      );
[Z5C] 1Ø17Ø PRINT CHR$(186); STRING$(28, 32); "***  SCh
      eck 2.Ø2  ***"; STRING$(29, 32); CHR$(186);
[ØKE] 1Ø18Ø PRINT CHR$(186); " Basic-Version
                                (C) 1989 DOS Internatio
      nal "; CHR$(186);
[C5C] 1Ø19Ø PRINT CHR$(2Ø4); "═══════════════════════
      ═══════════════════════
      ═══"; CHR$(185);
[DR3] 1Ø2ØØ FOR ZAEHLER = 1 TO 5
[ZY6] 1Ø21Ø   PRINT CHR$(186); STRING$(78, 32); CHR$(18
      6);
[CR2] 1Ø22Ø NEXT ZAEHLER
[84C] 1Ø23Ø PRINT CHR$(2Ø4); "═══════════════════════
      ═══════════════════════
      ═══"; CHR$(185);
[ØØD] 1Ø24Ø PRINT CHR$(186); " CHECK:          ZEILE:
                        DATEI:
          "; CHR$(186);
[PØ7] 1Ø25Ø PRINT CHR$(2ØØ); STRING$(78, 2Ø5); CHR$(188
      );
[2D2] 1Ø26Ø LOCATE 2Ø, 1
[B17] 1Ø27Ø PRINT CHR$(218); STRING$(78, 196); CHR$(191
      );
```

```
[ØØ7] 1Ø28Ø PRINT CHR$(179); STRING$(78, 32); CHR$(179)
      ;
[C27] 1Ø29Ø PRINT CHR$(192); STRING$(78, 196); CHR$(217
      );
[4V1] 1Ø3ØØ COLOR 7
[9F1] 1Ø31Ø Y = 7
[9V6] 1Ø32Ø LOCATE 7, 31: PRINT "SCheck  Version 1.x"
[PR6] 1Ø33Ø LOCATE 8, 31: PRINT "SCheck  Version 2.Ø"
[M7H] 1Ø34Ø LOCATE 21, 4: PRINT "Bitte wählen Sie mit d
      en Pfeiltasten aus und bestätigen Sie durch [Ente
      r] ";
[2PD] 1Ø35Ø COLOR 14, 1: FOR Q = 3Ø TO 5Ø: LOCATE Y, Q:
       PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 7, Ø
[UP6] 1Ø36Ø TASTE$ = INKEY$: IF TASTE$ = "" THEN 1Ø36Ø
[P96] 1Ø37Ø IF TASTE$ <> CHR$(Ø) + CHR$(72) THEN 1Ø4ØØ
[4ND] 1Ø38Ø IF Y <> 7 THEN FOR Q = 3Ø TO 5Ø: LOCATE Y,
      Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 1
[532] 1Ø39Ø GOTO 1Ø35Ø
[1A6] 1Ø4ØØ IF TASTE$ <> CHR$(Ø) + CHR$(8Ø) THEN 1Ø43Ø
[1ND] 1Ø41Ø IF Y <> 8 THEN FOR Q = 3Ø TO 5Ø: LOCATE Y,
      Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 1
[232] 1Ø42Ø GOTO 1Ø35Ø
[F35] 1Ø43Ø IF TASTE$ <> CHR$(13) THEN 1Ø36Ø
[8N2] 1Ø44Ø FOR Q = 7 TO 8
[U16] 1Ø45Ø   LOCATE Q, 3Ø: PRINT STRING$(21, 32)
[KN1] 1Ø46Ø NEXT Q
[8Y2] 1Ø47Ø VERSION = Y - 6
[GV1] 1Ø48Ø COLOR 7
[NF1] 1Ø49Ø Y = 7
[MA5] 1Ø5ØØ LOCATE 7, 36: PRINT "Standard"
[4A6] 1Ø51Ø LOCATE 8, 34: PRINT "Mit Prüfsumme"
[C65] 1Ø52Ø LOCATE 9, 37: PRINT "Hexdump"
[V7H] 1Ø53Ø LOCATE 21, 4: PRINT "Bitte wählen Sie mit d
      en Pfeiltasten aus und bestätigen Sie durch [Ente
      r] ";
[9PD] 1Ø54Ø COLOR 14, 1: FOR Q = 3Ø TO 5Ø: LOCATE Y, Q:
       PRINT CHR$(SCREEN(Y, Q)): NEXT Q: COLOR 7, Ø
[3R6] 1Ø55Ø TASTE$ = INKEY$: IF TASTE$ = "" THEN 1Ø55Ø
[4B6] 1Ø56Ø IF TASTE$ <> CHR$(Ø) + CHR$(72) THEN 1Ø59Ø
[BND] 1Ø57Ø IF Y <> 7 THEN FOR Q = 3Ø TO 5Ø: LOCATE Y,
      Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y - 1
[E32] 1Ø58Ø GOTO 1Ø54Ø
[GA6] 1Ø59Ø IF TASTE$ <> CHR$(Ø) + CHR$(8Ø) THEN 1Ø62Ø
[CND] 1Ø6ØØ IF Y <> 9 THEN FOR Q = 3Ø TO 5Ø: LOCATE Y,
      Q: PRINT CHR$(SCREEN(Y, Q)): NEXT Q: Y = Y + 1
[B32] 1Ø61Ø GOTO 1Ø54Ø
[P35] 1Ø62Ø IF TASTE$ <> CHR$(13) THEN 1Ø55Ø
[GN2] 1Ø63Ø FOR Q = 7 TO 9
[126] 1Ø64Ø   LOCATE Q, 3Ø: PRINT STRING$(21, 32)
[TN1] 1Ø65Ø NEXT Q
[WG4] 1Ø66Ø IF Y = 8 THEN LAUF = TRUE
[NA4] 1Ø67Ø IF Y = 9 THEN BIN = TRUE
[RU5] 1Ø68Ø LOCATE 12, 54: PRINT STRING$(25, 32)
[8DF] 1Ø69Ø GOSUB 1315Ø: LOCATE 21, 12: PRINT "Bitte ge
      ben Sie den Dateinamen der gewünschten Datei ein
      ";
[9P8] 1Ø7ØØ Y = 12: X = 54: LAENGE = 24: MENGE$ = "alle
      ": GOSUB 1Ø89Ø
[N6D] 1Ø71Ø IF (TASTE$ = CHR$(27)) OR (Q$ = "") THEN LO
      CATE 12, 54: PRINT STRING$(25, 32): GOTO 1Ø68Ø
[LM3] 1Ø72Ø ON ERROR GOTO 1116Ø
```

Listing. »SCheck.bas« für GW-Basic

```
[C74] 10730 OPEN Q$ FOR INPUT AS #T
[732] 10740 GOTO 11230
[3Ø2] 10750 COLOR 12
[2A6] 10760 LOCATE 8, 3Ø: PRINT "D A T E I F E H L E R"
[VV1] 10770 COLOR 7
[5GC] 10780 GOSUB 13150: LOCATE 21, 21: PRINT "Drücken
      Sie bitte eine beliebige Taste ";
[B83] 10790 WHILE INKEY$ = ""
[GX2] 10800   PRINT CHR$(7);
[CE1] 10810 WEND
[6U5] 10820 LOCATE 8, 3Ø: PRINT STRING$(21, 32)
[842] 10830 GOTO 10680
[2VØ] 10840 '
[184] 10850 '***********************************
[UD6] 10860 '* Unterprogramm zur Dateneingabe *
[384] 10870 '***********************************
[6VØ] 10880 '
[GG1] 10890 Q$ = ""
[EF2] 10900 LOCATE Y, X
[FS3] 10910 PRINT Q$; CHR$(219);
[GR6] 10920 TASTE$ = INKEY$: IF TASTE$ = "" THEN 10920
[ZK5] 10930 IF (TASTE$ <> CHR$(8)) THEN 10990
[224] 10940   IF LEN(Q$) = Ø THEN 10920
[X44] 10950   Q$ = LEFT$(Q$, LEN(Q$) - 1)
[6N2] 10960   LOCATE Y, X
[V44] 10970   PRINT Q$; CHR$(219); " ";
[8A2] 10980   GOTO 10920
[T98] 10990 IF TASTE$ = CHR$(13) THEN LOCATE Y, X: PRIN
      T Q$; " ";
[639] 11000 IF (TASTE$ = CHR$(13)) OR (TASTE$ = CHR$(27
      )) THEN RETURN
[C55] 11010 IF (ASC(TASTE$) < 32) THEN 10920
[NX4] 11020 IF MENGE$ <> "zahl" THEN 11050
[ZC8] 11030   IF (ASC(TASTE$) < 48) OR (ASC(TASTE$) > 5
      7) THEN 10920
[892] 11040   GOTO 11070
[PU4] 11050 IF MENGE$ = "alle" THEN 11070
[E86] 11060 IF INSTR(MENGE$, TASTE$) = Ø THEN 10920
[YF6] 11070 IF LEN(Q$) < LAENGE THEN Q$ = Q$ + TASTE$
[T22] 11080 GOTO 10900
[6H8] 11090 Y = 10: X = 65: LAENGE = 5: MENGE$ = "zahl"
      : GOSUB 10890
[BV5] 11100 IF (TASTE$ = CHR$(27)) THEN RETURN
[6UØ] 11110 '
[NK2] 11120 '******************
[TR3] 11130 '* Fehler-Routine *
[RK2] 11140 '******************
[AUØ] 11150 '
[KN1] 11160 CLOSE
[ZG2] 11170 RESUME 10750
[DUØ] 11180 '
[8G2] 11190 '*****************
[TH3] 11200 '* Hauptprogramm *
[4G2] 11210 '*****************
[BUØ] 11220 '
[KP3] 11230 COLOR 7: GOSUB 13150
[N43] 11240 ON ERROR GOTO Ø
[K62] 11250 DATEI$ = Q$
[DF7] 11260 GROSS$ = DATEI$: GOSUB 13040: DATEI$ = GROS
      S$
[3W3] 11270 I = INSTR(DATEI$, ".")
[6K7] 11280 IF I = Ø THEN DATEI$ = DATEI$ + ".": I = LE
      N(DATEI$)
[U64] 11290 EXT$ = MID$(DATEI$, I, 4)
[8T4] 11300 DATEI$ = LEFT$(DATEI$, I - 1)
[HN5] 11310 LOCATE 12, 54: PRINT DATEI$; EXT$
[HB4] 11320 TEMPNAME$ = DATEI$ + ".$$$"
[5E6] 11330 OPEN TEMPNAME$ FOR OUTPUT AS #TEMP
[CZ7] 11340 IF BIN THEN OPEN DATEI$ + ".BIN" FOR OUTPUT
       AS #F
[CP7] 11350 IF VERSION = 2 THEN NUMMER = 1 ELSE NUMMER
      = Ø
[KYC] 11360 IF BIN THEN GOSUB 13150: LOCATE 21, 25: PRI
      NT ">>> Binärdatei wird erzeugt <<<"
[TE3] 11370 WHILE NOT EOF(T)
[X24] 11380   LINE INPUT #T, ZEILE$
[LDB] 11385   WHILE RIGHT$(ZEILE$,1)=" " : ZEILE$=LEFT$
      (ZEILE$,LEN(ZEILE$)-1) : WEND
[333] 11390   WEITER = FALSE
[WT3] 11400   IF WEITER THEN 11500
[WN6] 11410   GOSUB 11820 'Gelesene Zeile anzeigen
[TS4] 11420   IF BIN THEN WEITER = TRUE
[ZT3] 11430   IF WEITER THEN 11500
[AM6] 11440   CH$ = INKEY$: IF CH$ = "" THEN GOTO 11440
[3G3] 11450   ERLAUBT = FALSE
[ZH8] 11460   IF CH$ = CHR$(13) THEN ERLAUBT = TRUE: WE
      ITER = TRUE
[N1C] 11470   IF (CH$ = "e") OR (CH$ = "E") THEN ERLAUB
      T = TRUE: GOSUB 12060'Zeile bearbeiten
[8ME] 11480   IF (CH$ = "q") OR (CH$ = "Q") THEN IF NOT
       LAUF THEN GOSUB 12420: IF SCHLUSS THEN GOTO 1163
      Ø'Quit
[F57] 11490   IF ERLAUBT THEN GOTO 11400 ELSE GOTO 1144
      Ø
[8Y3] 11500   PRINT #TEMP, ZEILE$
[DU3] 11510   IF NOT BIN THEN 11610
[D99] 11520   WHILE LEFT$(ZEILE$, 1) = " ": ZEILE$ = MI
      D$(ZEILE$, 2): WEND
[364] 11530   IF ZEILE$ = "" THEN 11610
[K75] 11540   B = VAL("&H" + LEFT$(ZEILE$, 2))
[TSH] 11550   IF (B < Ø) OR (B > 255) THEN GOSUB 13150:
       LOCATE 21, 22: PRINT "+++++ ABBRUCH WEGEN DATENF
      EHLER +++++": LOCATE 25, 1: END
[HN1] 11560   BY = B
[TK3] 11570   PRINT #F, CHR$(BY);
[YA4] 11580   ZEILE$ = MID$(ZEILE$, 3)
[M99] 11590   WHILE LEFT$(ZEILE$, 1) = " ": ZEILE$ = MI
      D$(ZEILE$, 2): WEND
[T92] 11600   GOTO 11530
[DY3] 11610   NUMMER = NUMMER + 1
[8E1] 11620 WEND
[S7B] 11630 WHILE NOT EOF(T): LINE INPUT #T, ZEILE$: PR
      INT #TEMP, ZEILE$: WEND
[4P1] 11640 CLOSE
[8U5] 11650 OPEN DATEI$ + EXT$ FOR INPUT AS #1
[XØ6] 11660 OPEN DATEI$ + ".BAK" FOR OUTPUT AS #2
[XGB] 11670 WHILE NOT EOF(1): LINE INPUT #1, ZEILE$: PR
      INT #2, ZEILE$: WEND: CLOSE
[6D5] 11680 OPEN TEMPNAME$ FOR INPUT AS #1
[576] 11690 OPEN DATEI$ + EXT$ FOR OUTPUT AS #2
[UGB] 11700 WHILE NOT EOF(1): LINE INPUT #1, ZEILE$: PR
      INT #2, ZEILE$: WEND: CLOSE
[713] 11710 KILL TEMPNAME$
[GTN] 11720 GOSUB 12560: LOCATE 12, 9: PRINT STRING$(11
      , 32): LOCATE 12, 25: PRINT STRING$(21, 32): LOCA
      TE 12, 53: PRINT STRING$(27, 32)'Statuszeile lösc
      hen
[494] 11730 IF BIN THEN EXT$ = ".BIN"
[N4H] 11740 GOSUB 13150: LOCATE 21, (80 - (31 + LEN(DAT
      EI$ + EXT$))) / 2: PRINT "Die Datei '"; DATEI$ +
      EXT$; "' wurde gespeichert."
[CE2] 11750 LOCATE 25, 1
[H51] 11760 END
[4VØ] 11770 '
[TH3] 11780 '****************************
[B65] 11790 '* Gelesene Zeile anzeigen *
[NH3] 11800 '****************************
[2VØ] 11810 '
[BBB] 11820 IF LAUF THEN CHECK$ = "[" + LEFT$(ZEILE$, 3
      ) + "]": ZEILE$ = MID$(ZEILE$, 5)
[MDN] 11830 FOR Q = 1 TO LEN(CHECK$): IF ASC(MID$(CHECK
      $, Q, 1)) > 96 AND ASC(MID$(CHECK$, Q, 1)) < 123
      THEN MID$(CHECK$, Q, 1) = CHR$(ASC(MID$(CHECK$, Q
      , 1)) - 32)
[3P1] 11840 NEXT Q
[B57] 11850 LINKS = 2: OBEN = 6: RECHTS = 79: UNTEN = 1
      Ø
[CX4] 11860 LOCATE 12, 26: PRINT NUMMER
[Z45] 11870 GOSUB 12560'Fenster löschen
[WX8] 11880 GOSUB 12660'Zeile innerhalb des Fensters an
      zeigen
[366] 11890 GOSUB 12800'Check-Summe errechnen
[ØM2] 11900 LOCATE 12, 10
[HB2] 11910 PRINT PS$
[HM7] 11920 IF BIN THEN GOTO 12000'Binärdatei erzeugen
[N8C] 11930 IF LAUF THEN GOTO 11970'Übereinstimmung der
       Prüfsummen feststellen
[X74] 11940 GOSUB 13150: LOCATE 21, 4
[LKD] 11950 PRINT "[E] Editieren    [RETURN] Weiter
      [Q] Quit       [Ctrl-Break] Abbrechen"
[152] 11960 GOTO 11990
[ATP] 11970 GOSUB 13150: IF PS$ = CHECK$ THEN LOCATE 21
      , (80 - (22 + LEN(STR$(NUMMER)))) / 2: PRINT "Prü
      fsumme OK für Zeile"; NUMMER: WEITER = TRUE: GOTO
       12000
[1NH] 11980 LOCATE 21, 4: PRINT CHR$(7); "Prüfsummenfeh
      ler!  [E] Editieren  [RETURN] Weiter  [Ctrl-Break
      ] Abbrechen"
[XM2] 11990 LOCATE 12, 15
[KV1] 12000 RETURN
[5UØ] 12010 '
[MK2] 12020 '******************
[4H3] 12030 '* Zeile eingeben *
[PK2] 12040 '******************
[9UØ] 12050 '
[HRD] 12060 GOSUB 13150: LOCATE 21, 17: PRINT "Ändern S
      ie die Programmzeile        [Enter] Ende"
[245] 12070 GOSUB 12560'Fenster löschen
[7B5] 12080 GOSUB 12660'Zeile darstellen
[FL3] 12090 LOCATE OBEN, LINKS
[7G3] 12100 X = 1: PRINT "";
[8E2] 12110 CH$ = INKEY$
[FE6] 12120 IF LEFT$(CH$, 1) <> CHR$(Ø) THEN GOTO 12220
[4N3] 12130   CH$ = RIGHT$(CH$, 1)
[4NE] 12140   IF CH$ = "M" THEN IF X > LEN(ZEILE$) THEN
       PRINT CHR$(7); : GOTO 12110 ELSE GOSUB 12340: X
      = X + 1
[KZC] 12150   IF CH$ = "K" THEN IF X = 1 THEN PRINT CHR
      $(7); : GOTO 12110 ELSE GOSUB 12340: X = X - 1
[YZC] 12160   IF CH$ = "H" THEN IF X < 79 THEN PRINT CH
      R$(7); : GOTO 12110 ELSE GOSUB 12340: X = X - 78
[YCF] 12170   IF CH$ = "P" THEN IF X + 77 > LEN(ZEILE$)
       THEN PRINT CHR$(7); : GOTO 12110 ELSE GOSUB 1234
      0: X = X + 78
[TV7] 12180   IF CH$ = "G" THEN IF X <> 1 THEN GOSUB 12
      340: X = 1
```

Listing. »SCheck.bas« für GW-Basic

```
[41C] 1219Ø   IF CH$ = "O" THEN IF X <> LEN(ZEILE$) + 1
      THEN GOSUB 1234Ø: X = LEN(ZEILE$) + 1
[8PM] 122ØØ   IF CH$ = "S" THEN IF X <= LEN(ZEILE$) THE
      N ZEILE$ = LEFT$(ZEILE$, X - 1) + MID$(ZEILE$, X
      + 1): GOSUB 1266Ø ELSE PRINT CHR$(7); : GOTO 1211
      Ø
[G92] 1221Ø   GOTO 1224Ø
[XXN] 1222Ø IF CH$ = CHR$(8) THEN IF X > 1 THEN ZEILE$
      = LEFT$(ZEILE$, X - 2) + MID$(ZEILE$, X): GOSUB 1
      266Ø: GOSUB 1234Ø: X = X - 1 ELSE PRINT CHR$(7);
      : GOTO 1211Ø
[LZN] 1223Ø IF CH$ >= " " THEN IF LEN(ZEILE$) = 255 THE
      N PRINT CHR$(7); : GOTO 1211Ø ELSE ZEILE$ = LEFT$
      (ZEILE$, X - 1) + CH$ + MID$(ZEILE$, X): GOSUB 12
      66Ø: X = X + 1
[7YB] 1224Ø LOCATE OBEN + INT((X - 1) / 78), LINKS - 1
      + X - INT((X - 1) / 78) * 78: PRINT "■":
[LK5] 1225Ø IF CH$ <> CHR$(13) THEN GOTO 1211Ø
[648] 1226Ø IF LAUF THEN ZEILE$ = MID$(CHECK$, 2, 3) +
      " " + ZEILE$
[XL6] 1227Ø GOSUB 1234Ø: GOSUB 1266Ø'Zeile anzeigen
[2W1] 1228Ø RETURN
[NUØ] 1229Ø '
[CP3] 123ØØ '****************************
[2H5] 1231Ø '* Zeichen wiederherstellen *
[EP3] 1232Ø '****************************
[LUØ] 1233Ø '
[UM9] 1234Ø LOCATE OBEN + INT((X - 1) / 78), LINKS - 1
      + X - INT((X - 1) / 78) * 78
[YXA] 1235Ø IF MID$(ZEILE$, X, 1) = "" THEN PRINT " " E
      LSE PRINT MID$(ZEILE$, X, 1)
[4W1] 1236Ø RETURN
[RUØ] 1237Ø '
[ØA2] 1238Ø '***************
[3X2] 1239Ø '* Quit (Ende) *
[W92] 124ØØ '***************
[NUØ] 1241Ø '
[U63] 1242Ø SCHLUSS = FALSE
[KYB] 1243Ø SICHER$ = "": FOR Q = 2 TO 79: SICHER$ = SI
      CHER$ + CHR$(SCREEN(21, Q)): NEXT Q
[YDD] 1244Ø GOSUB 1315Ø: LOCATE 21, 16: PRINT "Bitte dr
      ücken Sie [J], um die Arbeit zu beenden "
[GA6] 1245Ø CH$ = INKEY$: IF CH$ = "" THEN GOTO 1245Ø
[R3B] 1246Ø IF (CH$ = "j") OR (CH$ = "J") THEN SCHLUSS
      = TRUE: PRINT #TEMP, ZEILE$
[BA2] 1247Ø GOSUB 1315Ø
[P78] 1248Ø IF NOT SCHLUSS THEN LOCATE 21, 2: PRINT SIC
      HER$
[BW1] 1249Ø RETURN
[651] 125ØØ END
[TUØ] 1251Ø '
[6R2] 1252Ø '*******************
[U24] 1253Ø '* Fenster löschen *
[8R2] 1254Ø '*******************
[XUØ] 1255Ø '
[Ø44] 1256Ø FOR Q = OBEN TO UNTEN
[Z53] 1257Ø LOCATE Q, LINKS
[8B6] 1258Ø   PRINT STRING$(RECHTS - LINKS + 1, 32)
[ZN1] 1259Ø NEXT Q
[AW1] 126ØØ RETURN
[XUØ] 1261Ø '
[265] 1262Ø '*****************************************
      *
[D98] 1263Ø '* Inhalt von »zeile$« im Fenster anzeigen
      *
[465] 1264Ø '*****************************************
      *
[1VØ] 1265Ø '
```

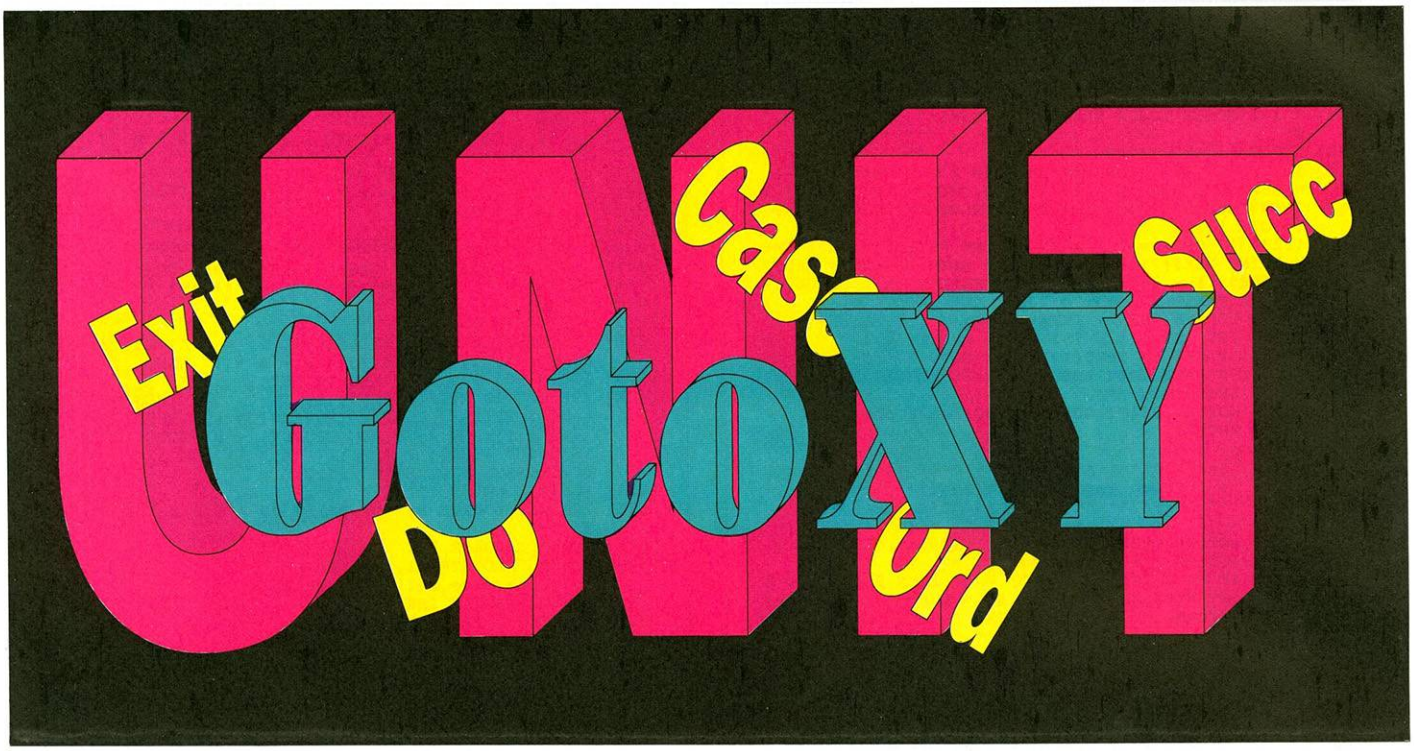
```
[SR2] 1266Ø F.X = Ø: F.Y = Ø
[CC5] 1267Ø FOR ZAEHLER = 1 TO LEN(ZEILE$)
[F85] 1268Ø   LOCATE OBEN + F.Y, LINKS + F.X
[UH5] 1269Ø   PRINT MID$(ZEILE$, ZAEHLER, 1);
[SK2] 127ØØ   F.X = F.X + 1
[597] 1271Ø   IF F.X >= RECHTS - 1 THEN F.X = Ø: F.Y =
      F.Y + 1
[6S2] 1272Ø NEXT ZAEHLER
[RX6] 1273Ø LOCATE OBEN + F.Y, LINKS + F.X: PRINT " ";
[KW1] 1274Ø RETURN
[5VØ] 1275Ø '
[AH4] 1276Ø '************************************
[ET6] 1277Ø '* Zeile auf Check-Summe überprüfen *
[CH4] 1278Ø '************************************
[9VØ] 1279Ø '
[F42] 128ØØ P = Ø: A = 1
[8SB] 1281Ø WHILE (MID$(ZEILE$, A, 1) = " ") OR (MID$(Z
      EILE$, A, 1) = CHR$(9)): A = A + 1: WEND
[374] 1282Ø FOR I = A TO LEN(ZEILE$)
[4CX] 1283Ø   IF MID$(ZEILE$, I, 1) >= " " THEN GROSS$
      = MID$(ZEILE$, I, 1): IF VERSION = 1 THEN P = ((I
      - A + 1) AND 7) * ASC(GROSS$) + P: GOTO 1292Ø EL
      SE GOSUB 13Ø4Ø: P = (((I - A + 1) AND 3) + 1) * A
      SC(GROSS$) + P: GOTO 1292Ø
[S22] 1284Ø   COLOR 12
[5N6] 1285Ø   LOCATE 8, 3Ø: PRINT "D A T E I F E H L E
      R"
[Ø22] 1286Ø   COLOR 7
[X7C] 1287Ø   LOCATE 21, 11: PRINT "          Drücken S
      ie bitte eine beliebige Taste         ";
[CC3] 1288Ø   WHILE INKEY$ = ""
[D93] 1289Ø     PRINT CHR$(7);
[5L1] 129ØØ   WEND
[R16] 1291Ø   LOCATE 8, 3Ø: PRINT STRING$(21, 32)
[ØP1] 1292Ø NEXT I
[KF2] 1293Ø PS$ = "[   ]"
[AM2] 1294Ø FOR I = 2 TO 4
[K25] 12945   IF P > 32767 THEN P = P - 32768
[5T7] 1295Ø   MID$(PS$, I, 1) = MID$(ZEICHEN$, (P AND 3
      1) + 1, 1)
[HZ2] 1296Ø   P = FIX(P / 32)
[5P1] 1297Ø NEXT I
[YW1] 1298Ø RETURN
[HVØ] 1299Ø '
[ZX3] 13ØØØ '********************************
[KØ6] 13Ø1Ø '* In Großbuchstaben umwandeln *
[934] 13Ø2Ø '********************************
[AUØ] 13Ø3Ø '
[9ZN] 13Ø4Ø FOR Q = 1 TO LEN(GROSS$): IF ASC(MID$(GROSS
      $, Q, 1)) > 96 AND ASC(MID$(GROSS$, Q, 1)) < 123
      THEN MID$(GROSS$, Q, 1) = CHR$(ASC(MID$(GROSS$, Q
      , 1)) - 32)
[8T8] 13Ø5Ø IF MID$(GROSS$, Q, 1) = "ä" THEN MID$(GROSS
      $, Q, 1) = "Ä"
[NU8] 13Ø6Ø IF MID$(GROSS$, Q, 1) = "ö" THEN MID$(GROSS
      $, Q, 1) = "Ö"
[UU8] 13Ø7Ø IF MID$(GROSS$, Q, 1) = "ü" THEN MID$(GROSS
      $, Q, 1) = "Ü"
[DN1] 13Ø8Ø NEXT Q
[YV1] 13Ø9Ø RETURN
[BUØ] 131ØØ '
[YZ2] 1311Ø '**********************
[9N4] 1312Ø '* Meldezeile löschen *
[ØØ3] 1313Ø '**********************
[FUØ] 1314Ø '
[ZT5] 1315Ø LOCATE 21, 2: PRINT STRING$(78, 32)
[ZV1] 1316Ø RETURN
```

Listing. »SCheck.bas« für GW-Basic

Utilities gesucht

Programme, die den Bildschirminhalt auf den Kopf stellen, suchen unsere Leser sicherlich nicht. Aber einen wissenschaftlichen Taschenrechner, ein Festplatten-Backup-Programm, einen Extended-Memory-Treiber, Druckerspooler oder Cache-Speicher. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation.
Falls Sie sich vorweg informieren möchten: Zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (089) 4 61 00 10 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:
Redaktionsbüro Everts & Hagedorn
»DOS Software«
Hans-Stießberger-Str. 3
8013 Haar/München



Die Standardprozeduren von Turbo Pascal im Überblick

Es ist sehr umständlich, zu einer Turbo-Pascal-Funktion die entsprechenden Informationen im Handbuch zu finden, wenn es schnell gehen soll. Die folgende Referenztabelle informiert Sie über den Aufbau der einzelnen Kommandos und liefert Ihnen Informationen über ihre Funktion.

Die Turbo-Pascal-Referenz listet alle Standardfunktionen und -unterprogramme der aktuellen Turbo-Pascal-Version 5.5 auf. Die Erklärungen zu den einzelnen Kommandos sind aus Platzgründen knapp gehalten. Um Ihnen die Form der an ein Unterprogramm übergebenen Werte zu verdeutlichen, finden Sie im Parameterteil Informationen zu den Datentypen der übermittelten Größen. Sie dürfen bei einem Funktionsaufruf natürlich nicht mit angegeben werden.

Tabelle 1 enthält eine alphabetische Auflistung der Standardprozeduren und -funktionen von Turbo Pascal zusammen mit den Routinen der Standard-Units »Dos«, »Crt« und »Graph«.

In Tabelle 2 finden Sie eine Aufstellung der wichtigsten Systemvariablen mit ihren Initialisierungswerten. Wenn Sie darüber hinaus auf ausführlichere Informationen angewiesen sind, beschäftigen Sie sich bitte mit den spezifischen Erklärungen im Handbuch.

Zum Abschluß noch ein paar Regeln zum Umgang mit der Referenztabelle:

- Parameter (Variablen und Werte) sind grundsätzlich in spitze Klammern eingeschlossen, um Verwechslungen zu vermeiden.
- Ausdrücke in eckigen Klammern (»[« und »]«) sind optional und dürfen weggelassen werden.

Die Referenzartikel stellen in übersichtlicher Form die Funktionen oder Prozeduren von Turbo Pascal zu einem bestimmten Themengebiet dar. Sie erlauben Ihnen das Nachschlagen von Details über die Parameter und die Verwendung spezieller Befehle. Einen umfassenden Überblick über die Programmiermöglichkeiten erhalten Sie in den Grundlagenartikeln, die themenbezogen in die Programmierung einführen.

Für das Verständnis der Referenzartikel ist die Kenntnis der verwendeten Symbole Voraussetzung. Bei der Beschreibung einzelner Kommandos geht es um den formal korrekten Aufruf einer Funktion oder Prozedur. Parameter, die übergeben werden müssen, werden in spitzen Klammern (»<« und »>«) eingefaßt. Dabei finden für die Übergabewerte symbolische Namen Verwendung, auf die in der Erläuterung Bezug genommen wird. Bei den übergebenen Parametern wird zwischen Variablen- und Wertparametern unterschieden. Wertparameter übergeben bestimmte Zahlen oder Texte an eine Prozedur oder Funktion, ohne daß die Möglichkeit besteht, die Inhalte der Variablen im Programmcode der Prozedur oder Funktion zu ändern. Bei Variablenparametern hingegen kann das aufgerufene Unterprogramm die Inhalte der Parameter ändern. Turbo Pascal arbeitet an vielen Stellen mit Variablenparametern, um an ein aufrufendes Programm Werte zurückzuliefern, obwohl eine Prozedur aufgerufen wird, die direkt keine Ergebnisse zurückliefern kann.

In eckigen Klammern (»[« und »]«) stehende Parameter oder Reihen von Parametern sind optional. Das heißt, diese Parameter müssen beim Aufruf der Funktion nicht angegeben, können aber verwendet werden.

Durch einen senkrechten Strich (»|«) werden Alternativen bezeichnet. Als Parameter wird einer der Ausdrücke neben den senkrechten Strichen verwendet.

Viele der Befehle von Turbo Pascal sind als Prozeduren oder Funktionen angelegt. Um den Unterschied zwischen den beiden Gruppen von Kommandos in den Referenzartikeln zu verdeutlichen, wird hinter Prozeduren ein Semikolon gesetzt. Darüber hinaus ist in der Beschreibung eines einzelnen Befehls ebenfalls aufgeführt, ob es sich um eine Prozedur oder eine Funktion handelt.

(Markus Zietlow/dm)

Funktion mit Parameterliste	Aufgabe
Abs(<x>)	gibt den absoluten Wert des Objekts zurück
Addr(<x>)	liefert die Adresse des angegebenen Arguments
Append(var <f>:text)	öffnet eine bestehende Datei, um weitere Daten anzuhängen
Arc(<x>,<y>:integer;<stangle>,<endangle>,<radius>:word)	zeichnet einen Kreisausschnitt beziehungsweise Kreisbogen mit dem durch »x«, »y« angegebenen Mittelpunkt ab dem durch »stangle« definierten Anfangspunkt bis zum Endpunkt »endangle«. »radius« gibt den Radius an.
Arctan(<x>:real)	gibt den Arcustangens des angegebenen Arguments an
Assign(var <f>;<name>:string)	ordnet einer Dateivariablen einen externen Dateinamen zu
AssignCRT(var <f>:text)	weist dem Bildschirm (CRT) eine Textdatei zu
Bar(<x1>,<y1>,<x2>,<y2>:integer)	zeichnet einen Kasten mit den Eckpunkten »x1«;»y1« und »x2«;»y2«
Bar3d(<x1>,<y1>,<x2>,<y2>:integer;<depth>:word;<top>:boolean)	zeichnet einen gefüllten Quader mit den Eckpunkten »x1«;»y1« und »x2«;»y2«. »depth« gibt die Tiefe in Bildschirmpunkten an und »top«, ob eine Balkendecke gezeichnet werden soll.

Funktion mit Parameterliste	Aufgabe
BlockRead(var <f>:file;var <buf>;<count>:word[;<result>:word])	liest einen oder mehrere Datenblöcke der Datei »f« in die Puffervariable »buf«. »count« gibt die Anzahl der zu lesenden Records an. »result« enthält bei Erreichen des Dateiendes die Anzahl der zuletzt gelesenen Blöcke.
BlockWrite(var <f>:file;var <buf>;<count>:word[;<result>:word])	schreibt einen oder mehrere Datenblöcke der Datei »f« aus der Puffervariablen »buf«. »count« gibt die Anzahl der zu schreibenden Records an, und »result« enthält bei Auftreten eines Fehlers die Anzahl der geschriebenen Blöcke.
ChDir(<s>:string)	wechselt das Standardinhaltsverzeichnis
Chr(<x>:byte)	gibt das angegebene ASCII-Zeichen zurück
Circle(<x>,<y>:integer;<radius>:word)	zeichnet einen Kreis um den Mittelpunkt »x«;»y« mit dem Radius »radius«
ClearDevice	löscht den aktuellen Grafikbildschirm
ClearViewport	löscht das aktuelle Zeichenfenster
Close(var <f>)	schließt den Zugriff auf die spezifizierte Datei
CloseGraph	beendet das Grafikpaket
ClrEOL	löscht alle Zeichen von der aktuellen Cursor-Position bis zum Ende der Zeile

Software und Service aus einer Hand!
Werner Stähle
Softwarelösungen
Rahm 27 • 8500 Nürnberg 20
Telefon 0911/55 46 51

Jetzt ist er da! Unser neuer Hauptkatalog 2/90 liegt ab sofort kostenlos für Sie bereit!

Viele Neuheiten und preisgünstige Angebote warten auf Sie! Hier einige Beispiele:

Video-Datenbank PC	DM 49.90
OMEGA-ALPHA-HOROSKOP	DM 99.00
Finanz-Manager PC 4.0	DM 79.50
CD Verwaltung PC	DM 49.90
ATLANTIK PC - Oberfläche	DM 99.00
ATLANTIK PC - Datenbank	DM 198.00
TextMaker 2.2	DM 278.00
DAC-EASY-ACCOUNTING	DM 299.00
TIMEWORKS DTP	DM 448.00

Xplus DM 8.00(Versand) b.Bestellung unter DM 150.00!

RPG II - RPG III für PC

LATTICE
RPG II Lattice Development 4 450,00 DM

NATIVE
RPG II Native (Micro 36 Development) 4 770,00 DM
RPG II " Runtime 665,00 DM
RPG II " Micro 36 Netw.) 5 112,00 DM
RPG II " Runtime 1 150,00 DM
RPG III " Micro 38 Development 7 165,00 DM
RPG III " Runtime 2 300,00 DM
APC 400 Development (AS400) 5 145,00 DM

SOFTWARE-WEST
RPG II SW Development 3 420,00 DM
RPG II SW Runtime 684,00 DM

AT & T
RPG II Software 386 7 734,00 DM
RPG II Runtime Softw. 4 885,00 DM

zuzüglich 25,00 DM Versandkostenpauschale.

Wir führen sämtliche Software zu supergünstigen Preisen, die wir Ihnen auf Anfrage gerne mitteilen.

Software-Vertrieb
D. Kaczmarek
Egonstr. 25-27
7800 Freiburg
Telef. 07 61-27 44 69
Fax. 07 61-27 64 76

ab 5.6.90:
Weisserlenstr. 8
7800 Freiburg
Telef. 07 61-13 23 87
Fax. 07 61-13 23 05

ABACOMP GmbH

Schriftliche Bestellungen:
Abt.D, Kransberger Weg 24,
6000 Frankfurt/M. 50
Ladenverkauf: Mo-Fr 10-12 + 14-18h
Heerstr.149, 6000 Frankfurt 90
Technische Auskunft:
Mo-Sa 8-9.30h
(069) 76 30 39

PC´s ab 741,- !

ABACO-Personalcomputer made in Germany
Fordern Sie unseren aktuellen Gratiskatalog an !

ABACOMP - der richtige Partner für NETZWERKE

Arcnet-Karte 199,50
Kabel hierzu, 93 Ω, pro Meter 1,71
Stecker mit Montage ..... je 5,70
Aktiv-Verteilerkarte f. Arcnet 228,-
TOP-Ware, 4-Benutzer-Software ... 997,50
Komplettes Netzwerk-Starterkit für 4 Benutzer, mit Software und allen notwendigen Karten, Kabellänge insgesamt 20m (bitte Längen angeben), Paketpreis 1824,-

PC-Zubehör
Drucker Präsident 6320, 100 Z/sec. 342,-
Herkules-komp. Grafikkarte 57,-
Disklaufwerk 3 1/2" 1,44MB 159,60
30MB-Festplattenkit inklusive Controller und Kabelsatz 513,-

NEU! DIN A4 Flachbettscanner "CHINON", 300dpi, komplett mit Scansoftware für gängige Personalcomputer **1596,-**

Einfach und preisgünstig:
Machen Sie aus Ihrer 3 1/2"-Harddisk eine Wechselplatte ! Wechseleinheit (benötigt 5 1/4"-Einschub) **142,50**

ABACOMP in 7 Städten der DDR !

Chemnitz 9031 ABACOMP-MEGware Dr. G. Maudrich, Ehrlichstr. 44
Dresden 8036 ABACOMP-Computersysteme Schmidt, Herzberger Str. 8
Eisenach 5900 ABACOMP-Computersysteme Schmidt, Jahnstr. 19
Erfurt 5023 ABACOMP-Computersysteme Peter Seeber, Meineckestr. 116
Güsten 3256 ABACOMP-Computersysteme Dr. K. Gerner, Aschersleberner Landstr. 4
Kirschau 8604 ABACOMP-Computersysteme D. Manitz, Lessingstr. 3
Suhl 6016 ABACOMP-Computersysteme André Kasche, Straße der DSF 28

Weitere Filialen in Vorbereitung !

Ihr Ansprechpartner in Prag:
Petr Bazant, Provaznicka 8, 11000 Prag

Hinweis: Einzelne Artikel aus unserem Sortiment sind aufgrund herstellerbedingter Fertigungsengpässe nicht sofort lieferbar. Viele weitere Produkte auf Anfrage, Händleranfragen erwünscht.

Funktion mit Parameterliste	Aufgabe
ClrScr	löscht den Bildschirm und setzt den Cursor in die linke obere Ecke
ConCat(<s1>[,<s2>,...,<sn>]:string)	verbindet zwei oder mehrere Teile einer Zeichenkette miteinander
Copy(<s>:string;<index>,<count>:integer)	kopiert den Teil der Zeichenkette »s« ab der Position »index« mit einer Länge von »count«
Cos(<x>:real)	gibt den Cosinus-Wert des angegebenen Arguments zurück
CSeg	übergibt die Adresse des aktuellen Codesegments
Dec(<x>[,<n>]:longint)	dekrementiert das angegebene Argument um den Wert »n«. Fehlt »n«, dann wird das Argument um den Wert 1 dekrementiert.
Delay(<ms>:word)	hält für die angegebene Zahl von Millisekunden den Programmablauf an. Die Zeitangabe ist nicht hundertprozentig genau, da sie von dem verwendeten Computer abhängig ist.
Delete(var <s>:string;<index>,<count>:integer)	löscht den Teil einer Zeichenkette »s« ab Position »index« mit der Länge »count«
DelLine	löscht eine Zeile auf dem Bildschirm
DetectGraph(var <graphdriver>,<graphmode>:integer)	überprüft die verwendete Bildschirmkarte und schlägt einen Grafikmodus und -treiber vor
DiskFree(<drive>:byte)	gibt den freien Speicherplatz des in Laufwerk »drive« befindlichen Datenträgers an
DiskSize(<drive>:byte)	gibt den gesamten Speicherplatz des in Laufwerk »drive« befindlichen Datenträgers an
Dispose(var <p>:pointer)	gibt den durch eine dynamische Variable beanspruchten Arbeitsspeicher wieder frei
DOSExitCode	übergibt den sogenannten »Exitcode« eines durch eine Unterfunktion aufgerufenen MS-DOS-Programms
DOSVersion	ermittelt die Versionsnummer des eingesetzten Betriebssystems
DrawPoly(<numpoints>:word;var <polypoints>)	zeichnet ein Vieleck (»Polygon«) mit den in »polypoints« enthaltenen Koordinaten. »numpoints« gibt die Anzahl der Koordinaten an.
DSeg	übergibt die Adresse des aktuellen Datensegments
Ellipse(<x>,<y>:integer;<stangle>,<endangle>;<xradius>,<yradius>:word;)	zeichnet eine Ellipse beziehungsweise einen Ellipsenausschnitt um den Mittelpunkt »x«;»y« mit dem horizontalen Radius »xradius« und dem vertikalen Radius »yradius«. »stangle« definiert den Anfangs- und »endangle« den Endpunkt.
EnvCount	übergibt die Anzahl der Argumente in der Environment-Tabelle
EnvStr(<index>:integer)	übergibt den durch »index« angegebenen Eintrag in der Environment-Tabelle
EOF(var <f>)	überprüft, ob das Ende einer Datei erreicht ist
EOLn(var <f>)	überprüft, ob sich der Zeiger innerhalb einer Datei auf einem Zeilenende befindet
Erase(var <f>)	löscht die durch »f« spezifizierte Datei
exec(<path>,<cmdline>:string)	führt das in »path« enthaltene Programm aus und übergibt die in »cmdline« befindlichen Parameter
Exit	verläßt den aktuellen Kontrollblock
exp(<x>:real)	errechnet den Exponentialwert des angegebenen Arguments
Fexpand(<path>:pathstr)	erweitert einen bruchstückhaft angegebenen Dateinamen um den zugeordneten Verzeichnispfad
FilePos(var <f>)	gibt die aktuelle Position des Zeigers innerhalb einer Datei an
FileSize(var <f>)	übergibt die Dateigröße der mit »f« spezifizierten Datei
FillChar(var <x>;<count>:word;<value>)	füllt den angegebenen Speicherbereich ab Adresse »x« »count«-mal mit dem Wert »value«

Funktion mit Parameterliste	Aufgabe
FillEllipse(<x>,<y>:integer;<xradius>,<yradius>:word)	zeichnet eine ausgefüllte Ellipse um den Mittelpunkt »x«;»y« mit dem horizontalen Radius »xradius« und dem vertikalen Radius »yradius«
FillPoly(<numpoints>:word;var <polypoints>)	zeichnet ein ausgefülltes Vieleck (»Polygon«) mit den in »polypoints« enthaltenen Koordinaten. »numpoints« gibt die Anzahl der Koordinaten an.
FindFirst(<path>:string;<attr>:byte;var <s>:searchstr)	durchsucht ein Verzeichnis »path« nach dem ersten Eintrag mit dem angegebenen Attributbyte »attr« und Suchparameter »s«
FindNext(var <s>:searchstr)	durchsucht ein Verzeichnis nach dem nächsten Eintrag mit dem Suchparameter »s«
FloodFill(<x>,<y>:integer;<border>:word)	füllt den durch »border« begrenzten Bereich mit dem momentanen Füllmuster
Flush(var <f>:text)	leert den Dateipuffer einer Textdatei, indem er ihn auf Datenträger schreibt
Frac(<x>:real)	übergibt den nichtganzzahligen Teil des angegebenen Arguments
FreeMem(var <p>:pointer;<size>:word)	gibt einen »size« Byte großen Speicherbereich auf dem durch »p« spezifizierten Heap frei
FSearch(<path>:pathstr;<dirlist>:string)	sucht den Dateiantrag »path« in der Verzeichnisliste »dirlist«
FSplit(<path>:pathstr;var <dir>:dirstr;var <name>:namestr;var <ext>:extstr)	teilt einen Dateinamen in seine Komponenten, Suchweg »path«, Dateiname »name« und Suffix »ext« auf
GetArcCoords(var <arccoords>:arccoordstype)	übermittelt Angaben über den zuletzt aktivierten Arc-Aufruf
GetAspectRatio(var <xasp>,<yasp>:word)	errechnet das momentane Höhen- und Seitenverhältnis des Bildschirms
GetBkColor	ermittelt die aktuelle Farbe des Hintergrundes
GetCBreak(var <break>:boolean)	ermittelt den Status des Control-Break-Kennzeichens von MS-DOS, das heißt, bei welchen Operationen MS-DOS auf Control-Break reagiert
GetColor	ermittelt die aktuelle Zeichenfarbe
GetDate(var <year>,<month>,<day>,<dayofweek>:word)	teilt das aktuelle Systemdatum in die Komponenten Jahr »year«, Monat »month«, Tag »day« und Wochentag »dayofweek« auf
GetDefaultPalette(var <pal>:palettetype)	übergibt die beim Aufruf des Grafiktreibers gesetzte Farbpalette »pal«
getdir(<d>:byte;var <s>:string)	gibt das aktuelle Verzeichnis »s« des momentanen Laufwerks »d« an
GetDriverName	übergibt den Namen des aktuellen Grafiktreibers als Zeichenkette
GetEnv(<envvar>:string)	ermittelt den Inhalt des durch »envvar« spezifizierten Eintrags in der Environment-Tabelle
GetFAttr(var <f>;var <attr>:word)	übergibt das Attribut »attr« der durch »f« definierten Datei
GetFillPattern(var <fillpattern>:fillpatterntyp)	gibt das zuletzt gesetzte Füllmuster »fillpattern« an
GetFillSettings(var <fillinfo>:fillsettingstype)	gibt das zuletzt gesetzte Füllmuster und die Füllfarbe in »fillinfo« an
GetFTime(var <f>;var <time>:longint)	übergibt das Datum und die Uhrzeit der letzten Veränderung beziehungsweise Modifikation einer Datei
GetGraphMode	liefert den aktuellen Grafikmodus als Parameter zurück
GetImage(<x1>,<y1>,<x2>,<y2>:integer;var <bitmap>)	kopiert den durch die Koordinaten »x1«, »y1«, »x2« und »y2« angegebenen Bildschirmabschnitt in eine Puffervariable
GetIntVec(<intno>:byte;var <vector>:pointer)	übergibt den Inhalt »vector« des Interrupt-Vektors des Interrupts »intno«
GetLineSettings(var <lineinfo>:linesettingstype)	gibt die durch den letzten Aufruf des Befehls »setlinestyle« definierten Parameter in »lineinfo« zurück
GetMaxColor	ermittelt die im aktuellen Bildschirmmodus einsetzbare höchste Farbnummer
GetMaxMode	übergibt die Nummer des maximal möglichen Bildschirmmodus

Funktion mit Parameterliste	Aufgabe
GetMaxX	liefert den Wert der höchstmöglichen Koordinate auf der horizontalen Koordinatenachse
GetMaxY	liefert den Wert der höchstmöglichen Koordinate auf der vertikalen Koordinatenachse
GetMem(var <p>:pointer;<size>:word)	reserviert einen »size« Byte großen Bereich auf dem Heap
GetModeName(<modenumber>:integer)	stellt den Namen des durch »modenumber« spezifizierten Grafikmodus zur Verfügung
GetModeRange(<graphdriver>:integer;var <lomode>, <himode>:integer)	stellt den Wertebereich der Bildschirmmodi für den angegebenen Grafiktreiber fest
GetPalette(var <palette>:palettetype)	übergibt die Größe der beim Aufruf des Grafiktreibers gesetzten Farbpalette »palette« und ihrer einzelnen Einträge
GetPaletteSize	übergibt die Größe der aktuellen Farbpalette »palette«, das heißt die Anzahl ihrer Einträge
GetPixel(<x>,<y>:integer)	ermittelt die Farbe des an der Position »x«,»y« befindlichen Bildpunktes
GetTextSettings(var <textinfo>:textsettingstype)	gibt die durch den letzten Aufruf der Befehle »outtext« und »outtextxy« definierten Parameter in »textinfo« zurück
GetTime(var <hour>, <min>,<second>, <sc100>:word)	ermittelt die aktuelle Systemzeit in ihren Bestandteilen Stunde »hour«, Minute »min«, Sekunde »second« und Hundertstelsekunden »sc100«
GetVerify(var <verify>:boolean)	übergibt den Status des durch MS-DOS kontrollierten Verify-Kennzeichens, das festlegt, ob Schreiboperationen auf einem Datenträger überprüft werden sollen
GetViewSettings(var <viewport>:viewporttype)	stellt Informationen über das aktuelle, durch »viewport« festgelegte Zeichenfenster zur Verfügung
GetX	übergibt die aktuelle Position des Grafik-Cursors auf der X-Koordinate
GetY	übergibt die aktuelle Position des Grafik-Cursors auf der Y-Koordinate
GotoXY(<x>,<y>:byte)	setzt den Text-Cursor auf die durch die Koordinaten »x« und »y« festgelegte Position
GraphDefaults	setzt alle Parameter und Definitionen des aktuellen Grafiktreibers auf die bei der Initialisierung festgelegten Werte zurück
GraphErrorMsg(<errorcode>:integer)	übergibt die durch die Fehlernummer »errorcode« spezifizierte Fehlermeldung in Textform
GraphResult	übergibt den Fehlerstatus der zuletzt ausgeführten Grafikoperation
Halt[(<exitcode>:word)]	unterbricht die Ausführung eines Programms und kehrt in die aufrufende Kommandoebene (in den meisten Fällen MS-DOS) zurück. Wenn der Ende-Code »exitcode« angegeben ist, erhält diese MS-DOS-Variable diesen Wert.
Hi(<x>)	errechnet das höherwertige Byte des spezifizierten Arguments »x«
HiVideo	erhöht den Kontrast und die Helligkeit beziehungsweise die Intensität für folgende Ausgaben von Zeichen
ImageSize(<x1>,<y1>, <x2>,<y2>:integer)	liefert die Größe einer Puffervariablen für den mittels der Koordinaten »x1«, »y1«, »x2« und »y2« festgelegten Bildschirmausschnitt
Inc(<x>[,<n>:longint])	erhöht das angegebene Argument um den Wert »n«; fehlt »n«, dann wird das Argument um den Wert 1 erhöht
InitGraph(var <graphdriver>:integer,var <graphmode>:integer,<pathtodriver>:string)	initialisiert den Grafiktreiber »graphdriver« und setzt den angegebenen Grafikmodus

Funktion mit Parameterliste	Aufgabe
Insert(<source>:string; var <s>:string; <index>:integer)	fügt die in »source« spezifizierte Zeichenkette ab der Position »index« in die Zeichenkette »s« ein
InsLine	fügt eine Leerzeile an der aktuellen Cursor-Position auf dem Bildschirm ein
InstallUserDriver(<name>:string;<autodetectptr>:pointer)	installiert den benutzerdefinierten Grafiktreiber »name« mittels des Zeigers »autodetectptr«
InstallUserFont(<fontfilename>:string)	installiert den in »fontfilename« angegebenen benutzerdefinierten Zeichensatz für die Verwendung in Turbo Pascal
Int(<x>:real)	übergibt den ganzzahligen Teil des angegebenen Arguments
Intr(<intno>:byte;var <regs>:registers)	ruft den Interrupt mit der Interrupt-Nummer »intno« aus und übergibt dabei die in »regs« abgelegten Registerwerte
IOResult	übergibt den Fehlerstatus der zuletzt ausgeführten Ein-/Ausgabeoperation
Keep(<exitcode>:word)	unterbricht die Ausführung des aktuellen Programms und legt es speicherresident im Arbeitsspeicher ab
KeyPressed	stellt fest, ob eine Taste vom Anwender gedrückt wurde
Length(<s>:string)	ermittelt die dynamische Länge der übergebenen Zeichenkette »s« ohne die zugeordnete Speicherverwaltung
Line(<x1>,<y1>,<x2>, <y2>:integer)	zeichnet eine Linie mit dem festgelegten Linientyp von den Koordinaten »x1«, »y1« zu der Position »x2«, »y2«
LineR(<dx>,<dy>:integer)	zeichnet relativ zu der aktuellen Grafik-Cursor-Position eine Linie mit dem horizontalen Abstand »dx« und der vertikalen Differenz »dy«
LineTo(<x>,<y>:integer)	zeichnet von der aktuellen Grafik-Cursor-Position eine Linie zu dem durch »x« und »y« spezifizierten Bildpunkt
Ln(<x>:real)	berechnet den natürlichen Logarithmus des übergebenen Arguments »x«
Lo(<x>)	errechnet das niederwertige Byte des spezifizierten Arguments »x«
LowVideo	setzt den Kontrast und die Helligkeit beziehungsweise die Intensität für folgende Ausgaben von Zeichen auf den Standardwert
Mark(var <p>:pointer)	legt die aktuelle Position der Spitze des Heaps in der Zeigervariablen »p« ab
MaxAvail	übergibt die maximale Größe eines freien Speicherblocks im Heap
MemAvail	übergibt die Gesamtgröße des freien Speicherbereichs im Heap
MkDir(<s>:string)	richtet ein Unterverzeichnis mit dem Verzeichnisnamen »s« im aktuellen Verzeichnis ein
Move(var <source>, <dst>;<count>:word)	kopiert einen »count« Byte großen Speicherbereich von der Adresse »source« an die Adresse »dst«
MoveRel(<dx>,<dy>:integer)	bewegt den Grafik-Cursor relativ zu der aktuellen Grafik-Cursor-Position mit dem horizontalen Abstand »dx« und der vertikalen Differenz »dy«, ohne eine Linie zu zeichnen
MoveTo(<x>,<y>:integer)	bewegt den Grafik-Cursor von der aktuellen Grafik-Cursor-Position zu dem durch »x« und »y« spezifizierten absoluten Bildpunkt, ohne eine Linie zu zeichnen
MSDOS(var <regs>:registers)	führt eine MS-DOS-Funktion aus und übergibt dabei die in »regs« festgelegten Register
New(var <p>:pointer)	generiert eine dynamische Variable und setzt den Zeiger »p« auf die aktuelle Position der Variablen im Arbeitsspeicher
NormVideo	legt das Textattribut für nachfolgende Textausgaben auf den Wert, der bei der Initialisierung festgelegt wurde

Funktion mit Parameterliste	Aufgabe
NoSound	schaltet den eingebauten Lautsprecher ab, der nach Angabe eines Tones durch »sound« sonst immer aktiviert wäre
Odd(<x>:longint)	überprüft, ob das übergebene Argument »x« eine gerade Zahl ist
Ofs(<x>)	übergibt den Offset-Bereich der in »x« spezifizierten Speicheradresse
Ord(<x>)	ermittelt den ASCII-Wert des als Parameter übergebenen Arguments
OutText(<textstring>: string)	gibt die in »textstring« enthaltene Zeichenkette an der aktuellen Position des Grafik-Cursors aus
OutText(<x>,<y>: integer;<textstring>: string)	gibt die in »textstring« enthaltene Zeichenkette an der durch »x« und »y« spezifizierten Position des Grafik-Cursors aus
OvrClearBuf	löscht den Overlay-Puffer
OvrGetBuf	ermittelt die momentane Größe des Overlay-Puffers und stellt sie dem Programm zur Verfügung
OvrInit(<filename>: string)	initialisiert die Verwaltung von Overlay-Dateien und öffnet die in »filename« übergebene Overlay-Datei
OvrInitEMS	kopiert die Overlay-Datei in eine EMS-Speicherkarte und gibt auf diese Weise den vorher belegten Hauptspeicherbereich wieder frei
OvrSetBuf(<size>: longint)	bestimmt die durch »size« angegebene Größe des Overlay-Puffers
PackTime(var <dt>:datetime;var <time>:longint)	konvertiert einen Datensatz des Typs »datetime« in einen gepackten Wert für die Verwendung in »setftime«
ParamCount	liefert die Anzahl der in der Kommandozeile übergebenen Parameter an das Programm
ParamStr(<index>:word)	übergibt den durch »index« spezifizierten Kommandozeilenparameter an das Programm
Pi	stellt den Wert der mathematischen Größe Pi zur Verfügung (entspricht ungefähr 3,141592653589793285)
PieSlice(<x>,<y>: integer;<stangle>, <endangle>,<radius>: word)	zeichnet einen ausgefüllten Kreisausschnitt beziehungsweise Kreisbogen mit dem durch »x«, »y« angegebenen Mittelpunkt ab dem durch »stangle« definierten Anfangspunkt bis zum Endpunkt »endangle«, »radius« gibt den Radius an. Da der Mittelpunkt mit den Endpunkten verbunden wird, entsteht ein sogenanntes »Kuchenstück«.
Pos(<subst>:string; <s>:string)	sucht eine Zeichenkette »s« nach der Zeichenkette »subst« ab und übergibt die Position. Wenn die erste Zeichenkette nicht in der zweiten enthalten ist, erhält »pos« den Wert 0.
Pred(<x>)	übergibt den Vorgänger des Arguments »x«
Ptr(<seg>,<ofs>:word)	rechnet die zwei Angaben Segment »seg« und Offset »ofs« in einen Wert mit dem Datentyp »Zeiger« um
PutImage(<x>,<y>: integer;var <bitmap>; <bitblt>:word)	kopiert den Inhalt der Puffervariable »bitmap« an den durch die Koordinaten »x« und »y« angegebenen Bildschirmabschnitt, nicht ohne die in »bitblt« spezifizierte logische Verknüpfung zu beachten
PutPixel(<x>,<y>: integer;<color>:word)	setzt den an den Koordinaten »x«;»y« befindlichen Bildpunkt auf die in »color« abgelegte Farbe
Random[(<range>:word)]	stellt eine durch den Zufallszahlengenerator erzeugte Real-Zahl zwischen 0 und 1 zur Verfügung. Wenn mit »range« ein Bereich angegeben wird, erhält man einen Integer-Wert zwischen 0 und »range«.
Randomize	initialisiert den Zufallszahlengenerator

Funktion mit Parameterliste	Aufgabe
Read([var <f>:text;] <v1>[,<v2>,...,<vn>])	liest die mit »v1« bis »vn« belegten Werte aus der Datei »f« und bewegt den Dateizeiger, wenn eine Datei »f« angegeben ist
ReadKey	liest ein Zeichen von der Tastatur, ohne es auf der Ausgabeeinheit auszugeben. Dieser Befehl eignet sich besonders, um programmgesteuerte Benutzereingaben zu kontrollieren.
ReadLn([var <f>:text;] <v1>[,<v2>,...,<vn>])	liest die mit »v1« bis »vn« belegten Werte aus der Datei »f«, springt zur nächsten Zeile und bewegt den Dateizeiger, wenn eine Datei »f« angegeben ist
Rectangle[<x1>,<y1>, <x2>,<y2>:integer)	zeichnet ein Rechteck mit den Eckpunkten »x1«;»y1« und »x2«;»y2«
RegisterBGIDriver (<driver>:pointer)	erfaßt einen als Objektdatei in Turbo Pascal eingebundenen Grafiktreiber
RegisterBGIFont(<font>: pointer)	erfaßt einen als Objektdatei in Turbo Pascal eingebundenen Vektorzeichensatz
Release(var <p>:pointer)	setzt die Position der Spitze des Heaps auf den mit »mark« gesetzten Wert
Rename(var <f>; <newname>:string)	benennt die auf Datenträger befindliche Datei »f« in den mit »newname« bezeichneten Dateinamen um
Reset(var <f>[:<file>; <recsize>:word])	öffnet eine bereits auf Datenträger gespeicherte Datei für Lese- und Schreibzugriffe
RestoreCRTMode	initialisiert den Bildschirmmodus, der vor dem Aufruf des Grafiktreibers aktiv war
ReWrite(var <f>[:<file>; <recsize>:word])	erzeugt und öffnet eine bisher noch nicht auf Datenträger befindliche Datei für Lese- und Schreibzugriffe oder überschreibt eine bereits existierende
RmDir(<s>:string)	löscht ein Unterverzeichnis mit dem Verzeichnisnamen »s« im aktuellen Verzeichnis
Round(<x>:real)	rundet das im Fließkommaformat vorliegende Argument »x« auf einen ganzzahligen Wert
RunError[(<errorcode>: word)]	generiert einen Laufzeitfehler, der das Programm kontrolliert abbricht
Sector(<x>,<y>: integer;<stangle>, <endangle>,<xradius>, <yradius>:word)	zeichnet einen Ellipsenausschnitt um den Mittelpunkt »x«;»y« mit dem horizontalen Radius »xradius« und dem vertikalen Radius »yradius«. »stangle« definiert den Anfangs- und »endangle« den Endpunkt. Da der Mittelpunkt mit den Endpunkten verbunden wird, entsteht ein sogenanntes »elliptisches Kuchenstück«.
Seek(var <f>;<n>: longint)	setzt den Positionszeiger in der durch »f« festgelegten Datei auf die durch »n« bestimmte Stelle
SeekEOF[(var <f>:text)]	stellt fest, ob zwischen der aktuellen Position des Dateizeigers und dem Ende der durch »f« angegebenen Datei lesbare Daten gespeichert sind
SeekEOLn[(var <f>:text)]	stellt fest, ob zwischen der aktuellen Position des Dateizeigers und dem nächsten Zeilenende der durch »f« angegebenen Datei lesbare Daten gespeichert sind
Seg(<x>)	übergibt den Segmentbereich des in »x« spezifizierten Objekts
SetActivePage(<page>: word)	bestimmt die Grafikseite »page«, auf der die nachfolgenden Zeichenbefehle arbeiten
SetAllPalette(var <palette>)	setzt sämtliche Komponenten einer Farbpalette neu
SetAspectRatio(<xasp>, <yasp>:word)	legt das Höhen- und Seitenverhältnis des Bildschirms neu fest
SetBkColor(<colornum>: word)	legt die aktuelle Farbe »colornum« des Hintergrundes fest
SetCBreak(<break>: boolean)	setzt den Status »break« des Control-Break-Kennzeichens von MS-DOS neu fest, das heißt, es setzt fest, bei welchen Operationen MS-DOS auf Control-Break reagiert

Funktion mit Parameterliste	Aufgabe
SetColor(<color>:word)	setzt die aktuelle Zeichenfarbe »color«
SetDate(<year>, <month>,<day>, <dayofweek>:word)	legt das aktuelle Systemdatum mit den Komponenten Jahr »year«, Monat »month«, Tag »day« und Wochentag »dayofweek« neu fest
SetFAttr(var <f>;<attr>: byte)	setzt das Attribut »attr« der durch »f« definierten Datei
SetFillPattern(<fillpattern>:fillpatterntyp; <color>:word)	definiert das Füllmuster »fillpattern« und die zugeordnete Farbe »color« mit neuen Werten
SetFillStyle(<fillpattern>: word;<color>:word)	setzt ein vordefiniertes Füllmuster
SetFTime(var <f>; <time>:longint)	setzt Datum und Uhrzeit der letzten Speicherung beziehungsweise Änderung einer Datei direkt und benutzt dabei das in »packtime« verwendete Pack-Verfahren
SetGraphBufSize(<bufsize>:word)	bestimmt die Größe des Grafikpuffers für Fülloperationen und Polygone
SetGraphMode (<mode>:integer)	schaltet in den Grafikmodus »mode« und löscht den Bildschirm
SetIntVec(<intno>:byte; <vector>:pointer)	setzt den Inhalt des Interrupt-Vektors mit der Interrupt-Bezeichnung »intno« auf die in »vector« übergebene Adresse
SetLineStyle(<linestyle>,<pattern>, <thickness>:word)	legt die Art, das Aussehen und die Stärke einer Linie fest
SetPalette(<colornum>:word; <color>:shortint)	modifiziert einen Eintrag in der aktuellen Farbpalette
SetRGBPalette(<colornum>,<redvalue>, <greenvalue>, <bluevalue>:integer)	modifiziert einen Eintrag in der aktuellen Farbpalette für die IBM-Grafikkarte 8514 und VGA-Bildschirmkarten im 256-Farben-Modus
SetTextBuf(var <f>:text; var <buf>[;<size>: word])	ordnet einer Textdateivariablen »f« einen Puffer »buf« zu, dessen Größe mit »size« verändert werden kann
SetTextJustify(<horiz>, <vert>:word)	bestimmt die Ausrichtung von ausgegebenem Text für »outtext« und »outtextxy«
SetTextStyle(<font>, <direction>:word; <charsize>:word)	legt den Zeichensatz »font«, die Ausgaberichtung »direction« und die Zeichengröße »charsize« für Textausgaben auf Grafikbildschirmen fest
SetTime(<hour>,<min>, <second>,<sc100>: word)	setzt die aktuelle Systemzeit mit Hilfe ihrer Bestandteile Stunde »hour«, Minute »min«, Sekunde »second« und Hundertstelsekunden »sc100«
SetUserCharSize (<multx>,<divx>, <multy>,<divy>:word)	bestimmt die Skalierfaktoren »multx«, »divx«, »multy«, »divy« für Grafikzeichensätze
SetVerify(<verify>: boolean)	legt den Status des durch MS-DOS kontrollierten Verify-Kennzeichens fest, das bestimmt, ob Schreiboperationen überprüft werden sollen
SetViewport(<x1>,<y1>, <x2>,<y2>:integer; <clip>:boolean)	definiert ein neues Zeichenfenster für Grafikoperationen
SetVisualPage (<page>:word)	wählt eine der Grafikseiten für folgende Zeichenoperationen aus
SetWriteMode(<writemode>:integer)	bestimmt, ob Grafikoperationen den vorherigen Bildschirminhalt überschreiben sollen oder über Boolesche Vergleiche mit ihm verknüpft werden
Sin(<x>:real)	errechnet den Sinus-Wert des in »x« übergebenen Arguments
SizeOf(<x>)	berechnet die Anzahl von Bytes, die das in »x« übergebene Argument im Speicher belegt
Sound(<hz>:word)	schaltet den eingebauten Lautsprecher ein und belegt ihn mit einer Frequenz von »hz« Hertz

Funktion mit Parameterliste	Aufgabe
Sptr	enthält den aktuellen Wert des Stapelzeigers, das heißt, des SP-Registers
Sqr(<x>)	berechnet die Quadratzahl des in »x« übergebenen Arguments
Sqrt(<x>:real)	ermittelt die Quadratwurzel der in »x« abgelegten Real-Zahl
SSeg	enthält den aktuellen Wert des Segments des Stapelzeigers
Str(<x>[:<width> [:<decimals>]],var <s>:string]	wandelt einen numerischen Wert »x« in eine Zeichenkette »s« um und berücksichtigt dabei die Breite »width« und die Mindeststellen »decimals«
Succ(<x>)	übergibt den Nachfolger des Arguments »x«
Swap(<x>)	tauscht das höherwertige und das niederwertige Byte des Arguments »x« aus
SwapVectors	vertauscht die vom System belegten Interrupt-Vektoren mit den entsprechenden Variablen der Unit »system«
TextBackground (<color>:byte)	bestimmt die Hintergrundfarbe für Ausgabeoperationen im Textmodus
TextColor(<color>:byte)	bestimmt die Vordergrundfarbe für Ausgabeoperationen im Textmodus
TextHeight (<textstring>:string)	berechnet die Höhe einer Zeichenkette, ohne sie auf dem Bildschirm auszugeben
TextMode(<mode>:word)	legt einen neuen Textmodus fest
TextWidth (<textstring>:string)	berechnet die Breite einer Zeichenkette, ohne sie auf dem Bildschirm auszugeben
Trunc(<x>:real)	konvertiert die Real-Zahl »x« durch Abschneiden der Nachkommastellen in einen Integer-Wert
Truncate(var <f>)	schneidet eine Datei an der momentanen Position des Dateizeigers ab
UnpackTime(<time>: longint;var <datetime>)	wandelt Datum und Uhrzeit aus dem gepackten Format in den Datentyp »datetime« um
UpCase(<ch>:char)	konvertiert das in »ch« übergebene Zeichen in einen Großbuchstaben; die übrigen Zeichen werden ignoriert und bleiben, wie sie vor dem Funktionsaufruf waren
Val(<s>:string;<v>; <varcode>:integer)	wandelt eine Zeichenkette »s«, soweit es möglich ist, in einen numerischen Wert um und speichert ihn in der Variablen »v«
WhereX	ermittelt die Spaltenposition des Cursors relativ in bezug auf das aktuelle Fenster
WhereY	ermittelt die Zeilenposition des Cursors relativ in bezug auf das aktuelle Fenster
Window(<x1>,<y1>, <x2>,<y2>:byte)	bestimmt einen Bereich des Bildschirms als momentanes Textfenster
Write(var <f>,<v1> [,<v2>,...,<vn>])	schreibt die mit »v1« bis »vn« belegten Werte in die Datei »f« und bewegt den Dateizeiger, wenn eine Datei »f« angegeben ist. Als Ausgabeeinheit ist standardmäßig der Bildschirm angegeben.
WriteLn([var <f>:text;] <v1>[,<v2>,...,<vn>])	schreibt die mit »v1« bis »vn« belegten Werte in die Datei »f«, springt zur nächsten Zeile und bewegt den Dateizeiger, wenn eine Datei »f« angegeben ist. Als Ausgabeeinheit ist standardmäßig der Bildschirm angegeben.

Tabelle 1. Das Verzeichnis der Standardroutinen und -funktionen erleichtert den Einsatz des Turbo-Pascal-Befehlssatzes

Grundlegende Systemvariablen

Unit	Name	Anfangswert	Bedeutung	Typ
Crt	**Ein-/Ausgabe**			
	Input	–	Crt-Treiber	Variable
	Output	–	Crt-Treiber	Variable
	Verschiedene			
	CheckBreak:Boolean	true	Prüfen von Ctrl-Break	Variable
	CheckEof:Boolean	false	Erzeugen einer Dateiendemarkierung	Variable
	CheckSnow:Boolean	true	CGA-Zugriff nur während des Strahlrücklaufs	Variable
	DirectVideo:Boolean	true	direkt in Bildschirmspeicher schreiben	Variable
	LastMode:Word	Init-Wert	aktueller Textmodus	Variable
	TextAttr:Byte	Init-Wert	aktuelles Textattribut	Variable
	WindMax:Word	Init-Wert	absolute Koordinate des aktiven Fensters	Variable
	WindMin:Word	0	absolute Koordinate des aktiven Fensters	Variable
	Vorder- und Hintergrundfarben			
	Black	0	Schwarz	Konstante
	Blink	128	Blinkend	Konstante
	Blue	1	Blau	Konstante
	Brown	6	Braun	Konstante
	Cyan	3	Türkis	Konstante
	Green	2	Grün	Konstante
	LightGray	7	Hellgrau	Konstante
	Magenta	5	Fuchsinrot	Konstante
	Red	4	Rot	Konstante
	Vordergrundfarben			
	DarkGray	8	Dunkelgrau	Konstante
	LightBlue	9	Hellblau	Konstante
	LightCyan	11	helles Türkis	Konstante
	LightGreen	10	Hellgrün	Konstante
	LightMagenta	13	helles Fuchsinrot	Konstante
	LightRed	12	Hellrot	Konstante
	White	15	Weiß	Konstante
	Yellow	14	Gelb	Konstante
	Videomodi			
	BW40	0	40x25 s/w, CGA-Karte	Konstante
	BW80	2	80x25 s/w, CGA-Karte	Konstante
	C40	1	entspricht CO40 für 3.0	Konstante
	C80	3	entspricht CO80 für 3.0	Konstante
	CO40	1	40x25 farbig, CGA-Karte	Konstante
	CO80	3	80x25 farbig, CGA-Karte	Konstante
	Font8x8	256	80 Zeichen mit 43, 50 Zeilen für EGA, VGA	Konstante
	Mono	7	80x25 s/w, Monochromkarte	Konstante
Dos	**Dateiattribute**			
	AnyFile	$3F	Datei »Sonstige Datei«	Konstante
	Archive	$20	Datei »Archiv«	Konstante
	Directory	$10	Datei »Verzeichnis«	Konstante
	Hidden	$02	Datei »versteckt«	Konstante
	ReadOnly	$01	Datei »nur lesbar«	Konstante
	SysFile	$04	Datei »Systemdatei«	Konstante
	VolumeID	$08	Datei »Datenträgerbezeichnung«	Konstante
	Dateibearbeitung			
	fmClosed	$D7B0	Modus »Datei schließen«	Konstante
	fmInOut	$D7B3	Modus »Datei lesen und schreiben«	Konstante
	fmInput	$D7B1	Modus »Datei lesen«	Konstante
	fmOutput	$D7B2	Modus »Datei schreiben«	Konstante
	Fehlercodes			
	DosError:Integer	–	Übergabe von Fehlercodes im MS-DOS-Format	Variable
	Flag-Verarbeitung			
	FAuxiliary	$0010	Hilfskennzeichen	Konstante
	FCarry	$0001	Übertragskennzeichen	Konstante
	FOverflow	$0800	Overflow-Kennzeichen	Konstante
	FParity	$0004	Paritätskennzeichen	Konstante
	FSign	$0080	Sign-Kennzeichen	Konstante
	FZero	$0040	Nullkennzeichen	Konstante
Graph	**Farbkonstanten**			
	Black	0	Schwarz	Konstante
	Blue	1	Blau	Konstante
	Brown	6	Braun	Konstante
	Cyan	3	Türkis	Konstante
	DarkGray	8	Dunkelgrau	Konstante
	Green	2	Grün	Konstante
	LightBlue	9	Hellblau	Konstante
	LightCyan	11	helles Türkis	Konstante

Grundlegende Systemvariablen

Unit	Name	Anfangswert	Bedeutung	Typ
	LightGray	7	Hellgrau	Konstante
	LightGreen	10	Hellgrün	Konstante
	LightMagenta	13	helles Fuchsinrot	Konstante
	LightRed	12	Hellrot	Konstante
	Magenta	5	Fuchsinrot	Konstante
	Red	4	Rot	Konstante
	White	15	Weiß	Konstante
	Yellow	14	Gelb	Konstante
	Fehlercodes			
	grError	-11	allgemeiner Fehler	Konstante
	grFileNotFound	-3	Grafiktreiberdatei nicht gefunden	Konstante
	grFontNotFound	-8	Zeichensatzdatei nicht gefunden	Konstante
	grInvalidDriver	-4	Grafiktreiberprogramm fehlerhaft	Konstante
	grInvalidFont	-13	ungültige Zeichensatzdatei	Konstante
	grInvalidFontNum	-14	ungültige Zeichensatznummer	Konstante
	grInvalidMode	-10	Grafikmodus von Grafiktreiber nicht unterstützt	Konstante
	grIOError	-12	Ein-/Ausgabefehler beim Lesen von BGI- oder CHR-Dateien	Konstante
	grNoFloodMem	-7	nicht genug Speicherplatz für »FloodFill«	Konstante
	grNoFontMem	-9	nicht genug Speicherplatz für Zeichensatz	Konstante
	grNoInitGraph	-1	Grafiktreiber nicht installiert	Konstante
	grNoLoadMem	-5	nicht genug Speicherplatz für Grafiktreiber	Konstante
	grNoScanMem	-6	nicht genug Speicherplatz für Grafikoperation	Konstante
	grNotDetected	-2	keine grafikfähige Bildschirmkarte	Konstante
	grOk	0	fehlerfreie Ausführung	Konstante
Printer	Lst	–	der Geräteeinheit »LPT1« zugeordnet	Variable
System	ErrorAddr:Pointer	nil	Adresse eines eventuellen Laufzeitfehlers	Konstante
	ExitCode:Integer	0	Exit-Code des Programms	Konstante
	ExitProc:Pointer	nil	zuletzt eingesetzte Exit-Prozedur	Konstante
	FileMode:Byte	2	Standardmodus für das Öffnen von Dateien	Konstante
	FreeMin:Word	0	minimale Größe der Fragmentliste	Konstante
	FreePtr:Pointer	nil	Start der Fragmentliste	Konstante
	HeapError:Pointer	nil	für eigene Fehlerbehandlungsroutinen	Konstante
	HeapOrg:Pointer	nil	Start des regulären Heaps	Konstante
	HeapPtr:Pointer	nil	aktuelle Spitze des Heaps	Konstante
	InOutRes:Integer	0	Status für IOResult	Konstante
	Input:Text	–	Standardeingabe	Variable
	Output:Text	–	Standardausgabe	Variable
	OvrCodeList:Word	0	CSeg-Liste der Overlay-Verwaltung	Konstante
	OvrDebugPtr:Pointer	nil	»Einhängepunkt« für den Debugger	Konstante
	OvrDosHandle:Word	0	Handle der Overlay-Datei	Konstante
	OvrEMSHandle:Word	0	Handle für OvrInitEMS	Konstante
	OvrHeapEnd:Word	0	Obergrenze des Overlay-Puffers	Konstante
	OvrHeapOrg:Word	0	Startadresse des Overlay-Puffers	Konstante
	OvrHeapPtr:Word	0	momentane Spitze des Overlay-Puffers	Konstante
	OvrHeapSize:Word	0	Größe des Overlay-Puffers	Konstante
	OvrLoadList:Word	0	Liste der geladenen Segmente	Konstante
	PrefixSeg:Word	0	PSP-Segmentadresse	Konstante
	RandSeed:LongInt	0	Startwert des Zufallszahlengenerators	Konstante
	SaveInt00:Pointer	?	Vektor $00 – Division durch 0	Variable
	SaveInt02:Pointer	?	Vektor $02 – NMI	Variable
	SaveInt1B:Pointer	?	Vektor $1B – Ctrl-Break	Variable
	SaveInt23:Pointer	?	Vektor $23 – Ctrl-C	Variable
	SaveInt24:Pointer	?	Vektor $24 – Critical Error	Variable
	SaveInt36:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt37:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt38:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt39:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt3A:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt3B:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt3C:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt3D:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt3E:Pointer	?	Emulator-Interrupts (nur im Modus {$N+})	Variable
	SaveInt75:Pointer	?	Vektor $75 – Fließkommafehler	Variable
	StackLimit:Word	0	Untergrenze des Stapels	Konstante
	Test8087:Byte	0	Ergebnis der Prüfung auf einen Coprozessor	Konstante

Tabelle 2. Die Systemvariablen und -konstanten der Standard-Units sind für nuancierte Programmroutinen unerläßlich

Arithmetische Funktionen

Zu einer professionellen Programmierumgebung gehören leistungsfähige Funktionen und Befehle, mit denen sich arithmetische Routinen programmieren lassen. Turbo Pascal bietet eine Reihe dieser Kommandos an.

Im umfangreichen Compiler-Handbuch zu Turbo Pascal sind die einzelnen Arithmetikfunktionen leider nicht in einem Block zusammengefaßt, sondern in die Beschreibung aller Befehle alphabetisch eingegliedert. Deshalb ist es wünschenswert, gerade diese Funktionsgattung kompakt im Überblick vorliegen zu haben. Nachstehend finden Sie diese Zusammenfassung der Befehle mit einem kurzen Kommentar zu den Aufgaben der jeweiligen Funktion. So haben Sie alle Arithmetikfunktionen schnell und übersichtlich zur Hand.

Abs(<Argument>)
Diese Funktion liefert den absoluten Wert des mit <Argument> übergebenen Parameters zurück. Der absolute Wert einer Zahl ist bei positiven Zahlen mit der Zahl identisch, bei negativen Zahlen wird das Vorzeichen für die Bildung des absoluten Wertes ignoriert. <Argument> darf ein beliebiger Ganzzahl- oder Real-Datentyp sein, das Funktionsergebnis wird im gleichen Typ zurückgegeben.

Arctan(<Argument>)
Die Funktion »Arctan« liefert den Arcustangens des mit <Argument> übergebenen Real-Parameters. <Argument> wird als Winkel im Bogenmaß interpretiert. Der Arcustangens ist der Wert, dessen Tangens wieder <Argument> ergibt, also die Umkehrfunktion des Tangens. Das Funktionsergebnis liefert den Datentyp »real«.

Cos(<Argument>)
Diese Funktion ermittelt den Cosinus des mit dem Real-Parameter <Argument> übergebenen Winkels. <Argument> wird als Winkel im Bogenmaß interpretiert.

Dec(<Variable>[,<Wert>]);
»Dec« vermindert den Wert einer Variablen, die Sie mit dem Parameter <Variable> übergeben. Wird ein Integer-Parameter <Wert> angegeben, vermindert die Dec-Prozedur die angegebene Variable um die Zahl <Wert>, führt also die folgende Zuweisung aus:

```
<Variable>:=<Variable>-<Wert>;
```

Ist <Wert> nicht angegeben, wird für <Wert> intern die Zahl 1 benutzt, das heißt, <Variable> wird um den Wert 1 vermindert. <Variable> darf eine Ganzzahlvariable sein, Real-Variablen kann der Dec-Befehl nicht verarbeiten.

Exp(<Argument>)
Die Exp-Funktion bildet Potenzen von der Basis des natürlichen Logarithmus e (2,718281). Übergeben wird mit dem Parameter <Argument> ein Real-Wert. Daraus bildet die Funktion den Term e + <Argument> und liefert das Ergebnis als Real-Zahl zurück.

Frac(<Kommazahl>)
»Frac« ermittelt die Nachkommastellen der mit dem Real-Parameter <Kommazahl> übergebenen Zahl. Das Funktionsergebnis wird als Real-Typ geliefert.

Hi(<Argument>)
Die Hi-Funktion bestimmt das höherwertige Byte des mit dem Parameter <Argument> übergebenen Wertes. <Argument> darf ein Word- oder Integer-Ausdruck sein. Als Ergebnis liefert die Hi-Funktion eine Bytezahl.

Inc(<Variable>[,<Wert>]);
»Inc« erhöht den Wert einer Variablen, die Sie mit dem Parameter <Variable> übergeben. Geben Sie den Integer-Parameter <Wert> an, erhöht die Inc-Prozedur die angegebene Variable um die Zahl <Wert>, führt also die folgende Zuweisung aus:

```
<Variable>:=<Variable>+<Wert>;
```

Ist <Wert> nicht angegeben, wird für <Wert> intern die Zahl 1 benutzt, das heißt, <Variable> wird um den Wert 1 erhöht. <Variable> darf eine Ganzzahlvariable sein, Real-Variablen kann der Inc-Befehl nicht verarbeiten.

Int(<Argument>)
Die Int-Funktion bestimmt den ganzzahligen Teil einer mit dem Parameter <Argument> übergebenen Real-Zahl. Das Funktionsergebnis ist ebenfalls vom Datentyp »real«. Im Gegensatz zur Trunc-Funktion liefert »Int« bei negativen Kommazahlen die nächstniedrigere Ganzzahl. Beispiel:

```
int(-1.5):=-2
```

Ln(<Argument>)
Diese Funktion bestimmt den natürlichen Logarithmus des mit <Argument> übergebenen Real-Wertes. Der natürliche Logarithmus einer Zahl ist der Wert, mit dem die Konstante e (2,718281) potenziert werden muß, um wieder <Argument> zu erhalten. Um den wesentlich häufiger benutzten Logarithmus zur Basis 10 zu erhalten, teilen Sie das Ergebnis der LN-Funktion durch »Ln(10)«, also:

```
LN10:=LN(x)/LN(10)
```

Lo(<Argument>)
Die Lo-Funktion bestimmt das niederwertige Byte des mit dem Parameter <Argument> übergebenen Wertes. <Argument> darf ein Word- oder Integer-Ausdruck sein. Als Ergebnis liefert die Lo-Funktion eine Bytezahl.

Umwandlung von Gradmaß in Bogenmaß
Die trigonometrischen Funktionen »Sin«, »Cos« und »Arctan« arbeiten mit Winkeln im Bogenmaß. Diese Messung der Winkelgröße hat ihren Ursprung in der Berechnung des Umfangs eines vollen Kreises. Ein Winkel von 360 Grad entspricht einem Winkel von 2*Pi in Gradmaß, also ungefähr einem Wert von 6,28. Um beliebige Winkel von Gradmaß in Bogenmaß umzuwandeln und umgekehrt, können Sie die folgenden Formeln verwenden:

```
Winkel in Gradmaß = Winkel in Bogenmaß
* 360/(2*Pi)
Winkel in Bogenmaß = Winkel in Gradmaß
* (2*Pi)/360
```

Zur Automatisierung der Umrechnung bieten sich die beiden folgenden Funktionen an:

```
function grad2bogen(grad:real):real;
begin
 grad2bogen:=grad*360/(2*pi);
end;
function bogen2grad(bogen:real):real;
begin
 bogen2grad:=bogen*(2*pi)/360;
end;
```

Odd(<Argument>)
»Odd« untersucht, ob die mit dem Parameter <Argument> übergebene Ganzzahl ungerade ist. Handelt es sich um eine ungerade Zahl (also eine Zahl, die sich nicht ohne Rest durch 2 teilen läßt), liefert die Odd-Funktion den Wert »true«, ansonsten den Wert »false«. <Argument> darf eine beliebige Ganzzahl sein, das Funktionsergebnis besitzt den Datentyp »boolean«.

Pi
Die Pi-Funktion liefert den Wert der Kreiskonstanten Pi zurück. Diese Konstante wird beispielsweise für die Berechnung des Umfangs und der Fläche eines Kreises verwendet. Das Ergebnis besitzt den Datentyp »real«.

Round(<Argument>)
Diese Funktion rundet den mit dem Parameter <Argument> übergebenen Real-Wert auf den nächsten ganzzahligen Wert auf oder ab und liefert ihn als Ergebnis vom Datentyp »longint«.

Sin(<Argument>)
»Sin« ermittelt den Sinus des mit <Argument> übergebenen Winkels. <Argument> ist ein Real-Parameter und wird als Winkel im Bogenmaß interpretiert. Die Funktion gibt den Sinus als Real-Funktionsergebnis zurück.

Sqr(<Argument>)
Diese Funktion bestimmt das Quadrat der mit dem Parameter <Argument> übergebenen Zahl. <Argument> darf ein Ganzzahl- oder Real-Typ sein. Das Funktionsergebnis wird mit dem gleichen Datentyp geliefert, den der Parameter <Argument> besitzt.

Sqrt(<Argument>)
Die Sqrt-Funktion bestimmt die Quadratwurzel der mit dem Parameter <Argument> übergebenen Zahl. Sowohl <Argument> als auch das Funktionsergebnis besitzen den Datentyp »real«.

Trunc(<Argument>)
»Trunc« schneidet von der mit dem Parameter <Argument> übergebenen Real-Zahl den Nachkommateil ab und liefert das Ergebnis als Wert mit dem Datentyp »longint«. Im Gegensatz zur Int-Funktion macht »trunc« bei negativen Zahlen keine Unterscheidung. Beispiel:

```
trunc(-1.5):=-1
```

(Martin Althaus/dm)

Berechnung des Tangens
Turbo Pascal bietet zwar Funktionen zur Berechnung des Sinus, des Cosinus und des Arcustangens an, jedoch keine Routine, die den Tangens eines Winkels ermittelt. Wenn Sie die Tangensberechnung in eigenen Programmen benötigen, können Sie sich mit folgendem Trick behelfen: Der Arcustangens eines Winkels ist mathematisch definiert als Quotient von 1 und dem Tangens des Winkels. Daher lautet die Formel für die Berechnung des Tangens:

```
Tangens = 1 / Arcustangens
```

Einfacher als die wiederholte Berechnung »per Hand«, nimmt diese Aufgabe die folgende Funktion wahr:

```
function tan(x:real):real;
begin
 tan:=1/arctan(x);
end;
```

DOS INTERNATIONAL

Zur Ergänzung unseres jungen, engagierten Redaktionsteams suchen wir einen versierten

Fachredakteur/in

Sie haben bereits grundlegende Erfahrungen im journalistischen Bereich, fundierte PC- und Softwarekenntnisse, Sie drücken sich gewandt aus und bringen die Bereitschaft mit, selbständig zu arbeiten.

Sind Sie interessiert? Dann senden Sie uns Ihre Bewerbungsunterlagen. Wir freuen uns auf ein persönliches Gespräch.

Redaktionsbüro Everts & Hagedorn
Redaktion DOS International
Hans-Stießberger-Str. 3
8013 Haar/München

Der Sound gehört zum guten Ton

Die im PC eingebaute Sound-Hardware überzeugt nicht gerade durch Vielfalt und abwechslungsreiche Klangfarben. Dennoch kennt Turbo Pascal einige Kommandos zur Tonerzeugung. Die abgedruckten Listings erweitern das Funktionsspektrum des Sound-Befehlssatzes durch einige nützliche Funktionen.

Sound (Frequenz)
Die Funktion »Sound« gibt über den eingebauten Lautsprecher einen Ton aus. Die als Parameter übergebene Frequenz ist in Hertz (Schwingungen pro Sekunde) angegeben. Der Lautsprecher bleibt so lange aktiv, bis ihn der No-Sound-Befehl wieder ausschaltet.

NoSound
Dieses Kommando schaltet den über »Sound« aktivierten Lautsprecher wieder aus.

Die genannten Turbo-Pascal-Routinen »Sound« und »NoSound« bieten einem versierten Anwender wenig Gestaltungsmöglichkeiten. Um diesen Einschränkungen abzuhelfen, stellen die folgenden Erläuterungen zusätzlich Unterprogramme vor, deren Aufbau Sie aus den Listings ersehen können.

Beep
Bei der Verwendung des Befehls »Beep« sind keine zusätzlichen Parameter vorgesehen. Die in Listing 1 vorgestellte Prozedur gibt einen 800-Hertz-Ton für die Dauer einer Viertelsekunde aus.

ESound (Frequenz, Dauer)
Die Frequenz des mit »ESound« aus Listing 2 auszugebenden Tons kann theoretisch zwischen 0 und 65535 Hertz liegen. Beachten Sie jedoch, daß der PC-Lautsprecher nur einen schmalen Bereich dieses Frequenzspektrums wiedergeben kann. »Dauer« darf ebenfalls einen Wert zwischen 0 und 65535 annehmen und gibt die Länge des auszugebenden Tons in Millisekunden an. Die Zeitangabe ist vom verwendeten PC abhängig und kann gelegentlich etwas schwanken. Turbo Pascal spielt jeden Ton im Vordergrund ab, das heißt, daß der Compiler während der Tonausgabe nicht mit der Programmausführung fortfährt.

Für komplexe Klangfolgen benutzen Sie den Befehl »Play«.

Play (Sound-Zeichenkette)
Die in Listing 3 enthaltene Funktion »Play« eignet sich für komplexere Melodien. Die Sound-Zeichenkette stellt entweder eine Zeichenkettenkonstante oder eine Zeichenkettenvariable dar. Die in ihr enthaltenen Kommandos führt der Turbo-Pascal-Compiler nacheinander aus.

`C, D, E, F, G, A, H [+,#] [-,b]`

spielen einen Ton in der jeweiligen Tonhöhe. Ein Plus- oder Nummernzeichen erhöht ihn, ein Minuszeichen oder »b« setzt ihn um einen Halbton tiefer. Für die Interpretation spielt es keine Rolle, ob die übergebenen Parameter in Groß- oder Kleinbuchstaben vorliegen.

`L <Länge>`

definiert die Länge aller folgenden Noten bis zum nächsten L-Kommando als Teile einer ganzen Note. »L16« legt beispielsweise Sechzehntelnoten fest. Der Wertebereich für den zu übergebenden Parameter erstreckt sich von 1 bis 99. Die Initialisierungsroutine legt für die Länge den Wert 1 fest (eine ganze Note).

`O <Oktave>`

setzt die zu spielende Oktave für die nachfolgenden Sound-Befehle. Werte zwischen 1 und 8 sind zulässig, die Standardeinstellung lautet 4. Eine Aufstellung der gültigen Werte und ihre zugeordneten Oktaven finden Sie in der Tabelle.

`P <Pause>`

Turbo Pascal setzt die Tonausgabe für die Dauer einer mit »L« definierten Note aus, das heißt, der Lautsprecher wird bis zum nächsten Ton ausgeschaltet.

`T <Tempo>`

legt das Abspieltempo eines Musikstücks beziehungsweise einer Klangfolge fest. Der Parameter »Tempo« spezifiziert die Anzahl der ganzen Noten pro Minute und muß im Bereich zwischen 1 und 99 liegen. Die Standardeinstellung beträgt 60 ganze Noten pro Minute.

`>`

erhöht die aktuell eingestellte Oktave um den Wert 1, wenn der maximale Endwert noch nicht erreicht ist.

`<`

vermindert die aktuell eingestellte Oktave um 1, wenn der Minimalwert noch nicht erreicht ist.

Um die Funktionsweise des Play-Kommandos besser verstehen zu können, finden Sie in Listing 3 ein Anwendungsbeispiel, das den Anfang des »Bolero« von Ravel spielt.

(Markus Zietlow/ks)

Oktave	Name
1	Subkontra-Oktave
2	Kontra-Oktave
3	große Oktave
4	kleine Oktave (Standard)
5	eingestrichen
6	zweigestrichen
7	dreigestrichen
8	viergestrichen

Tabelle. Die neu vorgestellte Play-Routine erlaubt für die Tonerzeugung acht Oktavenwerte

```
[ZR5]     1: (* Einbinden des Kommandos 'beep'      *)
[KB6]     2: (* in den Befehlssatz von Turbo-Pascal *)
[ØØØ]     3:
[563]     4: program beep_demo;
[ØØØ]     5:
[EN2]     6: procedure beep;
[MUØ]     7: begin
[Z72]     8:  write (chr(7));
[ZNØ]     9: end;
[ØØØ]    1Ø:
[MUØ]    11: begin
[DTØ]    12:   beep;
[KNØ]    13: end.
```

Listing 1. Das Unterprogramm »Beep« erzeugt einen simplen Piepton auf dem PC. Ein kleines Demonstrationsprogramm hierzu finden Sie in »sounref1.pas«.

```
[K16]     1: (* Einbinden des Kommandos 'esound'    *)
[KB6]     2: (* in den Befehlssatz von Turbo-Pascal *)
[Z26]     3: (* Syntax: esound(frequenz, dauer)     *)
[ZB5]     4: (* 'dauer' in Millisekunden            *)
[ØØØ]     5:
[SR3]     6: program esound_demo;
[ØØØ]     7:
[ASØ]     8: uses
[7SØ]     9:   crt;
[ØØØ]    1Ø:
[727]    11: procedure esound (frequenz, dauer : word);
[MUØ]    12: begin
[YT2]    13:   sound(frequenz);
[352]    14:   delay(dauer);
[3G1]    15:   nosound;
[ZNØ]    16: end;
[ØØØ]    17:
[MUØ]    18: begin
[DU4]    19:   (* Kammerton 'A' eine Sekunde *)
```

```
[FP2]   2Ø:   (* lang spielen *)
[SB2]   21:   esound(44Ø,1ØØØ);
[KNØ]   22: end.
```

Listing 2. Die Prozedur »ESound« im Programm »sounref2 .pas« gibt einen Ton mit der angegebenen Frequenz für eine bestimmte Dauer auf dem eingebauten Lautsprecher aus

```
[9T5]    1: (* Einbinden des Kommandos 'play'      *)
[KB6]    2: (* in den Befehlssatz von Turbo-Pascal *)
[6F6]    3: (* Syntax: play(sound_zeichenkette)     *)
[ØØØ]    4:
[F73]    5: program play_demo;
[ØØØ]    6:
[5H1]    7: uses crt;
[ØØØ]    8:
[6XØ]    9: const
[PW3]   1Ø:   subkontra_c=16.351592869;
[ØØØ]   11:
[RNØ]   12: var
[5W6]   13:   notenlaenge, oktave, tonlaenge : integer;
[ØØØ]   14:
[3S3]   15: procedure sound_init;
[MUØ]   16: begin
[ZM2]   17:   notenlaenge:=1;
[9P1]   18:   oktave:=4;
[WA2]   19:   tonlaenge:=6Ø;
[ZNØ]   2Ø: end;
[ØØØ]   21:
[HU5]   22: procedure play (kommando : string);
[RNØ]   23: var
[YA4]   24:   befehl, halbtoene, hv, note,
[KP4]   25:   schleife, zaehler : integer;
[PT2]   26:   schwingung:real;
[ØØØ]   27:
[9Z5]   28:  function gross(var zkette : string)
[8F1]   29:           : string;
[RNØ]   3Ø:  var
[DX2]   31:    zaehler:integer;
[9C4]   32:    zwischenspeicher:string;
[MUØ]   33:  begin
[BG3]   34:    zwischenspeicher:='';
[KZ5]   35:    for zaehler:=1 to length(zkette) do
[T56]   36:      zwischenspeicher:=zwischenspeicher
[TG4]   37:      +upcase(zkette[zaehler]);
[FB4]   38:    gross:=zwischenspeicher;
[ZNØ]   39:  end;
[ØØØ]   4Ø:
[WX6]   41:  procedure fehler(fehlermeldung : string);
[MUØ]   42:  begin
[3G1]   43:    nosound;
[G16]   44:    writeln(#7,'Fehler: ',fehlermeldung);
[XF1]   45:    writeln;
[536]   46:    write  ('Der Fehler befindet sich an ');
[CV5]   47:    writeln('Stelle ',zaehler,' von:');
[W53]   48:    writeln(kommando);
[DUØ]   49:    halt;
[ZNØ]   5Ø:  end;
[ØØØ]   51:
[727]   52:  procedure esound (frequenz, dauer : word);
[MUØ]   53:  begin
[YT2]   54:    sound(frequenz);
[352]   55:    delay(dauer);
[3G1]   56:    nosound;
[ZNØ]   57:  end;
[ØØØ]   58:
[465]   59:  procedure pause (dauer : word);
[MUØ]   6Ø:  begin
[3G1]   61:    nosound;
[352]   62:    delay(dauer);
[ZNØ]   63:  end;
[ØØØ]   64:
[MUØ]   65: begin
[P94]   66:   if length(kommando)<>Ø then
[MUØ]   67:   begin
[NX1]   68:     zaehler:=1;
[5K4]   69:     kommando:=gross(kommando);
[ØR5]   7Ø:     while zaehler<=length(kommando) do
[MUØ]   71:      begin
[CF5]   72:        befehl:=ord(kommando[zaehler]);
[WA2]   73:        case befehl of
[P24]   74:          ord('>') : if oktave<8 then
[UØ2]   75:                       inc(oktave);
[224]   76:          ord('<') : if oktave>2 then
[NZ1]   77:                       dec(oktave);
[HD1]   78:          ord('L') :
[MUØ]   79:           begin
[FT2]   8Ø:             if zaehler=length
[RG2]   81:                          (kommando) then
[LG3]   82:               fehler('Es fehlt die '+
[H84]   83:                'Angabe der Notenlänge')
[ZRØ]   84:             else
[MUØ]   85:              begin
[H62]   86:                inc(zaehler);
[4F3]   87:                befehl:=ord(kommando
[8Ø2]   88:                [zaehler]);
[WA2]   89:                case befehl of
[VH2]   9Ø:                  ord('Ø')..ord('9') :
[VX3]   91:                  val(kommando[zaehler],
[VC3]   92:                  notenlaenge,befehl);
[ZRØ]   93:                else
[NT3]   94:                  fehler('Falsche Angabe '
[T14]   95:                  +'einer Tonlänge; Wert '
[G14]   96:                  +'muß zwischen Ø und 99 '
[EE1]   97:                  +'liegen')
[ZNØ]   98:                end;
[FØ3]   99:                if zaehler<>length
[RG2]  1ØØ:                (kommando) then
[MUØ]  1Ø1:                 begin
[H62]  1Ø2:                   inc(zaehler);
[4F3]  1Ø3:                   befehl:=ord(kommando
[8Ø2]  1Ø4:                   [zaehler]);
[WA2]  1Ø5:                   case befehl of
[9L2]  1Ø6:                     ord('Ø')..ord('9'):
[MUØ]  1Ø7:                      begin
[153]  1Ø8:                        hv:=1Ø*Notenlaenge;
[P42]  1Ø9:                        val(kommando
[5T1]  11Ø:                        [zaehler],
[S42]  111:                        notenlaenge,
[381]  112:                        befehl);
[GU2]  113:                        notenlaenge:=hv
[Ø82]  114:                        +notenlaenge;
[ZNØ]  115:                      end;
[ZRØ]  116:                   else
[C52]  117:                     dec(zaehler);
[ZNØ]  118:                   end;
[ZNØ]  119:                 end;
[ZNØ]  12Ø:              end;
[ZNØ]  121:          end;
[ØA1]  122:          ord('O'):
[MUØ]  123:           begin
[FT2]  124:             if zaehler=length
[RG2]  125:             (kommando) then
[RD4]  126:               fehler('Es fehlt die Angabe'
[D32]  127:               +' der Oktave')
[ZRØ]  128:             else
[MUØ]  129:               begin
[H62]  13Ø:                 inc(zaehler);
[4F3]  131:                 befehl:=ord(kommando
[8Ø2]  132:                 [zaehler]);
[WA2]  133:                 case befehl of
[XH2]  134:                   ord('1')..ord('8') :
[VX3]  135:                   val(kommando[zaehler],
[MH2]  136:                   oktave,befehl);
[ZRØ]  137:                 else
[NT3]  138:                   fehler('Falsche Angabe '
[ØL3]  139:                   +'einer Oktave; Wert '
[3W3]  14Ø:                   +'muß zwischen 1 und 8 '
[EE1]  141:                   +'liegen')
[ZNØ]  142:                 end;
[ZNØ]  143:               end;
[ZNØ]  144:           end;
[K74]  145:          ord('P') : pause(round(6ØØØØ/
[834]  146:          notenlaenge/tonlaenge));
[FA1]  147:          ord('T'):
[MUØ]  148:           begin
[FT2]  149:             if zaehler=length
[RG2]  15Ø:             (kommando) then
[RD4]  151:               fehler('Es fehlt die Angabe'
[YD3]  152:               +' des Abspieltempos')
[ZRØ]  153:             else
[MUØ]  154:               begin
[H62]  155:                 inc(zaehler);
[4F3]  156:                 befehl:=ord(kommando
[8Ø2]  157:                 [zaehler]);
[WA2]  158:                 case befehl of
[VH2]  159:                   ord('Ø')..ord('9') :
[VX3]  16Ø:                   val(kommando[zaehler],
[EØ3]  161:                   tonlaenge,befehl);
[ZRØ]  162:                 else
[NT3]  163:                   fehler('Falsche Angabe '
[9H3]  164:                   +'des Abspieltempos; '
[B14]  165:                   +'Wert muß zwischen Ø '
[DG2]  166:                   +'und 99 liegen')
[ZNØ]  167:                 end;
[FØ3]  168:                 if zaehler<>length
[RG2]  169:                 (kommando) then
[MUØ]  17Ø:                   begin
[H62]  171:                     inc(zaehler);
[4F3]  172:                     befehl:=ord(kommando
[8Ø2]  173:                     [zaehler]);
[WA2]  174:                     case befehl of
[9L2]  175:                       ord('Ø')..ord('9'):
[MUØ]  176:                        begin
[9R2]  177:                          hv:=1Ø*tonlaenge;
[P42]  178:                          val(kommando
[5T1]  179:                          [zaehler],
[RR1]  18Ø:                          tonlaenge,
```

Listing 3. Die Play-Routine erleichtert die Tonausgabe erheblich. Einen kleinen Vorgeschmack bekommen Sie mit dem Demonstrationsprogramm in »sounref3.pas«.

```
[381]  181:                              befehl);
[982]  182:                              tonlaenge:=hv
[2Z1]  183:                              +tonlaenge;
[ZNØ]  184:                            end;
[ZRØ]  185:                          else
[C52]  186:                            dec(zaehler);
[ZNØ]  187:                          end;
[ZNØ]  188:                        end;
[ZNØ]  189:                    end;
[ZNØ]  19Ø:                  end;
[A71]  191:                 65,67..72:
[MUØ]  192:                  begin
[L62]  193:                    note:=befehl;
[UV3]  194:                    if note=72 then note:=73;
[ØW3]  195:                    if note=65 then note:=72;
[YØ2]  196:                    dec(note,67);
[NN2]  197:                    note:=note shl 1;
[6U3]  198:                    if note>5 then dec(note);
[1P4]  199:                    if zaehler<>length(kommando)
[2RØ]  2ØØ:                    then
[MUØ]  2Ø1:                      begin
[H62]  2Ø2:                        inc(zaehler);
[4F3]  2Ø3:                        befehl:=ord(kommando
[8Ø2]  2Ø4:                        [zaehler]);
[WA2]  2Ø5:                        case befehl of
[3K2]  2Ø6:                          ord('+'),ord('#') :
[HP1]  2Ø7:                          inc(note);
[5P2]  2Ø8:                          ord('-'),ord('B') :
[CN1]  2Ø9:                          dec(note);
[ZRØ]  21Ø:                        else
[C52]  211:                          dec(zaehler);
[ZNØ]  212:                        end;
[ZNØ]  213:                      end;
[HB4]  214:                    schwingung:=subkontra_c;
[HU4]  215:                    halbtoene:=(oktave-1)*12+note;
[Z43]  216:                    if halbtoene=-1 then
[8X3]  217:                      fehler('Der niedrigste '
[994]  218:                      +'auszugebende Ton ist das '
```

```
[4G2]  219:                        +'Subkontra-C');
[C63]  22Ø:                      if halbtoene<>Ø then
[PM4]  221:                        for schleife:=1 to halbtoene
[5C4]  222:                         do schwingung:=schwingung
[FE1]  223:                         *1.Ø594631;
[XA4]  224:                      esound(round(schwingung),
[NX3]  225:                      round(6ØØØØ/notenlaenge/
[DY1]  226:                      tonlaenge));
[ZNØ]  227:                    end;
[ZRØ]  228:                  else
[2V4]  229:                    fehler('Das angegebene Sound-'
[9D5]  23Ø:                    +'Kommando ist nicht verfügbar');
[ZNØ]  231:                  end;
[H62]  232:                  inc(zaehler);
[ZNØ]  233:                end;
[ZNØ]  234:              end;
[ZNØ]  235: end;
[ØØØ]  236:
[TØ7]  237: (* Die folgende Routine übermittelt als *)
[ZW6]  238: (* Eindruck von den Sound-Eigenschaften *)
[KS5]  239: (* des PCs den 'Bolero' von Ravel       *)
[ØØØ]  24Ø:
[MUØ]  241: begin
[662]  242:   sound_init;
[326]  243:   play('o2t2Ø14>cl8cl16<h>c<d>c<hal8>c');
[126]  244:   play('l16c<al4>cl8cl16<h>c<agefl2gl16');
[FH6]  245:   play('gfedefgal4ggl16gahagfeded18cl16c');
[XA6]  246:   play('dl8efl4dl2gl8pl4>dl8dcl16<hah>cd');
[YB6]  247:   play('cl8<hl16h>c<ha>c<haf18fl16ffl8fa');
[H86]  248:   play('l16>c<ahgl8fl16ffl8fal16hgafl8d');
[576]  249:   play('l16dcl4dl8dl16ddl8dfl16afgel8d');
[466]  25Ø:   play('l16dcl4dl8dl16dcl8dl16efl2gl16g');
[L92]  251:   play('fedl3c');
[KNØ]  252: end.
```

Listing 3. Die Play-Routine erleichtert die Tonausgabe erheblich. Einen kleinen Vorgeschmack bekommen Sie mit dem Demonstrationsprogramm in »sounref3.pas«.

Der Weg nach außen – die Ein- und Ausgabe

Daten müssen auch nach außen (etwa auf Drucker oder Bildschirm) gelangen beziehungsweise von externen Geräten wie einer seriellen Schnittstelle oder der Tastatur eingelesen werden können. Hier erfahren Sie, wie Sie auf Bildschirm, Tastatur und andere Ein- oder Ausgabegeräte zugreifen können. Lediglich Disketten- und Festplattenzugriffe sind ausgespart, sie werden in einem eigenen Artikel dargestellt.

Für den Zugriff auf Ein- und Ausgabegeräte bietet Turbo Pascal neben Standardbefehlen wie dem »Writeln«- und »Readln«-Kommando einige Routinen in den beiden Units »crt« und »printer« an. Diese öffnen »den Weg nach außen«, indem sie die Ports sowie Bildschirm und Tastatur ansprechen können.

ClrEOL;
löscht alle Zeichen in der aktuellen Cursor-Zeile von der momentanen Cursor-Position bis zum Zeilenende. Die Prozedur ist in der Unit »crt« definiert.

ClrScr;
löscht den gesamten Textbildschirm. Der Aufruf der Prozedur erfolgt über die Unit »crt«.

Delay(<Zeit>);
verzögert die Ausführung des Programms um die mit dem Word-Parameter <Zeit> angegebene Anzahl von Millisekunden. Die Prozedur ist in der Unit »crt« deklariert.

DelLine;
Diese Prozedur der Unit »crt« löscht die aktuelle Cursor-Zeile und rollt alle unterhalb des Bildschirms liegenden Zeilen um eine Zeile nach oben. Die unterste Bildschirmzeile wird anschließend gelöscht.

GotoXY(<X>,<Y>);
bewegt den Cursor in die mit dem Byteparameter <X> übergebene Bildschirmzeile und in die mit dem Byteparameter <Y> übergebene Bildschirmspalte. Die Prozedur ist in der Unit »crt« deklariert und läßt sich nur im Textmodus verwenden.

HighVideo;
Die Prozedur der Unit »crt« schaltet die intensive (leuchtende) Darstellung von Zeichen bei der Ausgabe auf dem Bildschirm ein.

InsLine;
fügt an der aktuellen Zeilenposition des Cursors eine Leerzeile ein. Alle unterhalb der Zeile liegenden Zeilen werden um eine Zeile nach unten verschoben. Die unterste Bildschirmzeile ist nach diesem Vorgang nicht mehr sichtbar. Die Prozedur ist in der Unit »crt« deklariert.

Keypressed;
Diese Funktion aus der Unit »crt« liefert einen Boolean-Wahrheitswert zurück. Wenn Sie eine Taste auf der Tastatur betätigen und der Tastencode im Tastaturpuffer bereitliegt, wird »true« als Funktionsergebnis zurückgegeben, ansonsten »false«.

LowVideo;
Die »LowVideo«-Prozedur schaltet die Intensität der Darstellung von Zeichen auf dem Bildschirm auf geringe (»dunkle«) Intensität. Der Aufruf der Prozedur geschieht über »crt«.

NormVideo;
Diese Prozedur der Crt-Unit schaltet die Zeichenintensität für die Ausgabe von Texten auf dem Bildschirm auf normale Intensität.

Port[<Port-Adresse>]
Die Port-Funktion erlaubt lesende Zugriffe auf Hardware-Bausteine. Eigentlich handelt es sich beim Port-Befehl nicht um eine Funktion, sondern um ein Pseudo-Array. An der mit dem Word-Parameter <Port-Adresse> übergebenen Adresse löst der Prozessor einen Lesebefehl aus und gibt den Wert des Ports als Bytefunktionsergebnis zurück.

Modus	Konstante	Wert
40x25 Zeichen, schwarzweiß (CGA)	BW40	0
40x25 Zeichen, farbig (CGA)	C40	1
80x25 Zeichen, schwarzweiß (CGA)	BW80	2
80x25 Zeichen, farbig (CGA)	C80	3
80x25 Zeichen, schwarzweiß (MDA, HGC)	Mono	7
Zuletzt gesetzter Modus	Last	-1

Tabelle. Erlaubte Werte für die Festlegung des Textmodus

Port[<Port-Adresse>]: = <Ausdruck>;
Dieser Befehl gibt den mit dem Byteparameter <Ausdruck> übergebenen Wert an den Hardware-Baustein am Port mit der Adresse <Port-Adresse> aus. Wie alle Port-Befehle ist er intern als Pseudo-Array konzipiert.

PortW[<Port-Adresse>]: = <Ausdruck>;
Dieser Befehl verrichtet die gleiche Aufgabe wie der Port-Befehl, liest jedoch anstelle eines Bytes ein Wort vom Hardware-Baustein ein. Das funktioniert technisch allerdings nur bei Anschlüssen mit 16-Bit-Daten-Bus wie 16-Bit-VGA-Grafikkarten in ATs.

PortW[<Port-Adresse>];
Die Portw-Funktion arbeitet wie der Port-Befehl, kann jedoch anstelle von Bytes auch ganze Worte an Portbausteine ausgeben. Das funktioniert technisch allerdings nur bei Anschlüssen mit 16-Bit-Daten-Bus, also zum Beispiel 16-Bit-VGA-Grafikkarten in ATs.

Read(<Var 1> [,<Var 2>, ..]);
liest einen Variableninhalt von der Tastatur ein. Dazu übergeben Sie als Parameter den Namen einer deklarierten Variablen. Turbo Pascal ermittelt den Typ der Variable und erwartet ein passendes Ergebnis. Vor der Abfrage der Tastatur wird der Cursor an der aktuellen Cursor-Position eingeschaltet. Mit dem Read-Befehl können Ganzzahl-, Real- und Zeichenvariablen neue Werte erhalten; ein Einlesen von Pointer-Variablen über die Tastatur ist nicht möglich. Der Read-Befehl kann mehr als eine Variable verarbeiten. Nach dem Lesen der Variablen bleibt der Cursor in der aktuellen Zeile stehen, es erfolgt kein Zeilenvorschub.

ReadLn(<Var 1> [,<Var 2>, ...]);
Dieses Kommando arbeitet wie der Read-Befehl, führt aber nach dem Einlesen der Variablen einen Zeilenvorschub durch und setzt den Cursor an den Anfang der nächsten Zeile.

ReadKey;
Die »ReadKey«-Funktion liest genau ein Zeichen von der Tastatur. Das Zeichen wird nicht auf dem Bildschirm ausgegeben. Das Funktionsergebnis besitzt den Datentyp »char«. Die Funktion ist in der Unit »crt« deklariert.

RestoreCrt;
Diese Prozedur der Unit »crt« setzt den Videomodus, der beim Start des Programms aktiv war. Sie ist nützlich, um einen veränderten Videomodus wiederherzustellen.

TextBackground(<Farbe>);
legt die Hintergrundfarbe für die Ausgabe von Zeichen auf dem Bildschirm fest. Die Prozedur ist in der Unit »crt« definiert und erwartet mit dem Byteparameter <Farbe> eine gültige Farbe für den Hintergrund.

TextColor(<Farbe>);
Diese Prozedur der Crt-Unit legt die Zeichenfarbe für die Ausgabe von Texten auf dem Bildschirm fest. Dazu übergeben Sie mit dem Byteparameter <Farbe> eine gültige Farbe.

TextMode(<Modus>);
setzt einen Textvideomodus. Die Prozedur ist in der Unit »crt« definiert und erwartet einen Videomodus, den Sie mit dem Integer-Parameter <Modus> übergeben. Für <Modus> sind die in der Tabelle aufgeführten Werte zulässig. Mit der Text-Mode-Prozedur lassen sich nur Textmodi aktivieren, Grafikmodi setzen die Initialsierung der Unit »graph« voraus.

WhereX;
Diese Funktion der Unit »crt« ermittelt die aktuelle Spalte, in der sich der Cursor befindet und liefert die Spaltennummer als Bytefunktionsergebnis zurück.

WhereY;
ermittelt die aktuelle Zeile, in der sich der Cursor befindet und liefert ihre Nummer als Bytefunktionsergebnis zurück. Die Funktion ist in der Crt-Unit deklariert.

Window (<X1>,<Y1>,<X2>, <Y2>);
Diese Prozedur der Unit »crt« definiert einen Teil des Textbildschirms als Fenster. Alle übergebenen Parameter besitzen den Datentyp »byte«. Mit <X1> und <Y1> legen Sie die Spalte und Zeile der linken oberen Ecke des Textfensters fest, die Parameter <X2> und <Y2> definieren dessen rechte untere Ecke. Nach dem Aufruf der Funktion beziehen sich alle koordinatenorientierten Befehle (etwa das GotoXY-Kommando) auf das neue Textfenster. Um den Standardbildschirm zurückzusetzen, muß der Window-Befehl mit der maximal möglichen Zeilenzahl (25) und der aktuell maximal möglichen Spaltenzahl (40 oder 80) aufgerufen werden.

Write(<Ausdr. 1> [,<Ausdr. 2> ,..]);
Das Write-Kommando wertet den mit dem Parameter <Ausdruck 1> übergebenen Ausdruck aus und gibt sein Ergebnis ab der aktuellen Cursor-Position auf dem Bildschirm aus. Der Write-Befehl verarbeitet auch mehr als einen Ausdruck. Die Trennung der Ausdrücke erfolgt durch Kommata. Bei <Ausdruck> kann es sich um eine Variable (beispielsweise »wert«), einen zusammengesetzten Term (beispielsweise »100 + 5*40«) oder eine Kombination aus Variablen und Termen handeln. Das Ausgabeformat der übergebenen Ausdrücke läßt sich mit dem Write-Kommando näher spezifizieren. Dazu können Sie hinter jedem Ausdruck in der Form »:<Min>[:<Nach>]« angeben, in welchem Format die Ausgabe des Ausdrucks erfolgen soll. Wird weder <Min> noch <Nach> angegeben, benutzt Turbo Pascal das Standardformat. Der Formatparameter <Min> läßt sich bei allen Datentypen angeben. Bei Ganzzahltypen (zum Beispiel »gzahl:5«) bewirkt die Angabe von <Min>, daß – sofern die Ganzzahl nicht bereits die mit <Min> angegebene Länge besitzt – Turbo Pascal so viele führende Leerzeichen voranstellt, daß insgesamt <Min>-Zeichen ausgegeben werden. Bei Char- und String-Typen (zum Beispiel »text:10«) werden Ausgaben auch, falls sie für sich noch nicht lang genug sind, mit führenden Leerzeichen auf die Länge <Min> gestreckt. Bei Real-Typen gilt das gleiche wie bei Ganzzahltypen, allerdings lassen sich hier mit dem zweiten Parameter <Nach> die Anzahl der Nachkommastellen zusätzlich angeben (zum Beispiel »kommazahl :4:2«). Bei Boolean-Datentypen sorgt die Angabe von <Min> dafür, daß einer der beiden String »true« oder »false« (je nach Wert des Ausdrucks) durch führende Leerzeichen auf die mit <Min> angegebene Minimallänge gebracht wird. Sind alle Ausdrücke der Write-Anweisung verarbeitet, bleibt der Cursor an der Cursor-Position stehen.

WriteLn(<Ausdruck 1> [,<Ausdruck 2> ,..]);
Dieser Befehl arbeitet wie das Write-Kommando. Nach der Verarbeitung der Ausdrücke wird allerdings ein Zeilenvorschub ausgeführt. Der Cursor wird in der ersten Spalte der folgenden Zeile plaziert; falls alle Zeilen des Bildschirms voll sind, wird der Bildschirm um eine Zeile nach oben gerollt. Verwenden Sie die Writeln-Anweisung ohne Parameter (also als »writeln;«), erfolgt nur die Ausgabe einer Leerzeile.

(Martin Althaus/dm)

Die Grafikbefehle von Turbo Pascal

Die Grafikprogrammierung ist eines der wesentlichen Einsatzgebiete von Turbo Pascal. Kein Wunder, da die Unit »graph« über eine Vielzahl von Funktionen und Prozeduren zur Erstellung und Verwaltung von grafischen Elementen verfügt.

Für PCs gibt es eine Vielzahl unterschiedlicher Grafikkarten. Die am weitesten verbreiteten sind die Hercules-, CGA-, EGA-, MCGA- und VGA-Karte. Jede dieser Grafikkarten erwartet für die Grafikdarstellung eine individuelle Programmierung. Turbo Pascal wickelt Zugriffe auf Grafikkarten daher über spezielle Treiber ab, die »Borland Graphics Interface« (auf deutsch: Borland-Grafikschnittstelle) genannt werden. Diese Grafiktreiber tragen die Namen »att.bgi« (für die Grafikkarte von Computern der Firma AT&T), »cga.bgi« (für CGA-Karten), »herc.bgi« (für Hercules-Karten), »egavga.bgi« (für EGA-, MCGA- und VGA-Karten) und »3270.bgi« (für 3270-PCs). Je nachdem, welche Grafikkarte angesprochen werden soll, muß das Turbo-Pascal-Programm auf den dazugehörigen Treiber zugreifen können, der also im aktuellen Verzeichnis oder über den Suchpfad zu finden sein sollte. Zur Initialisierung des Grafikmodus und des Treibers wird das Kommando »InitGraph« verwendet.

Um die standardmäßig zur Verfügung stehenden Grafikfunktionen und -prozeduren von Turbo Pascal zu nutzen, wird in den jeweiligen Turbo-Pascal-Quelltext über das Kommando »uses« die Unit »graph« eingebunden. Sie enthält alle Befehle von Turbo Pascal, die in Zusammenhang mit der Erstellung und Veränderung von Grafiken stehen.

Einige der Funktionen und Prozeduren adressieren die Parameter über Record-Datentypen, die ebenfalls in der Unit »graph« vordefiniert sind. Für die Übergabe dieser Parameter müssen Sie vor Aufruf der Funktion entsprechende Variablen definieren. Die vordefinierten Datentypen sehen Sie in Tabelle 1.

Viele der Funktionen und Prozeduren arbeiten mit Linien und Flächen. Aussehen und Erscheinungsform dieser Linien und Flächen werden bei Turbo Pascal global gespeichert. Zum Lesen oder zur Veränderung der Parameter benutzen Sie die Funktionen »GetFillSettings«, »GetLineSettings« und »SetFillPattern«, »SetFillStyle« und »SetLineStyle«. Für den Umgang mit den Grafikbefehlen definiert Turbo Pascal eine Vielzahl von Datentypen, die bei unterschiedlichen Funktionen und Prozeduren Verwendung finden. Die einzelnen Datentypen sind in den Tabellen aufgeschlüsselt. Es folgt nun eine Beschreibung der einzelnen Grafikfunktionen und eine Erläuterung über die Aufrufsyntax.

Arc(<X>,<Y>,<Anfangswinkel>,<Endwinkel>,<Radius>); zeichnet einen Kreisbogen um den Mittelpunkt <X> und <Y>, die beide als Integer-Werte interpretiert werden. Der Bogen wird zwischen <Anfangswinkel> und <Endwinkel> mit dem Radius <Radius> gezeichnet. Die Worte für <Anfangswinkel>, <Endwinkel> und <Radius> sind Parameter vom Datentyp »Word«. Zum Zeichnen der Linien werden die aktuellen Linienparameter verwendet. Die Werte der Winkel interpretiert der Compiler als Parameter in Gradmaß. Bei Verwendung von Bogenmaß sind diese zunächst umzurechnen, und zwar folgendermaßen:

```
arc(100,100,0,90,100);
{ein Viertelkreis}
arc(100,100,180,360,100);
{und das gegenüberliegende Viertel}
```

Bar3D(<X1>,<Y1>,<X2>,<Y2>,<Tiefe>,<Kopf>); erstellt einen dreidimensionalen Balken. Zwischen den Koordinaten <X1>, <Y1> und <X2>, <Y2>, die als Integer-Werte zu übergeben sind, wird zunächst ein Rechteck gezeichnet. Zum Zeichnen der Linien werden die aktuellen Linienparameter verwendet. Der Word-Parameter <Tiefe> legt die räumliche Tiefe des Balkens fest. Der Boolean-Parameter <Kopf> gibt an, ob die obere Fläche des Balkens gefüllt werden soll oder nicht.

```
bar3d(10,100,30,10,20,true);
{einen Balken mit Kopf}
bar3d(70,100,100,10,20,false);
{und einen ohne Kopf}
```

Circle(<X>,<Y>,<Radius>); zeichnet einen kompletten Kreis um den Mittelpunkt, der durch die beiden Integer-Parameter <X> und <Y> festgelegt wird. Der Word-Parameter <Radius> legt den Radius des Kreises fest. Zum Zeichnen des Kreises werden die aktuellen Linienparameter verwendet.

```
circle(150,100,50);
{ein großer Kreis}
circle(150,100,20);
{und ein kleiner hinein}
```

ClearDevice; löscht den Bildschirm und setzt alle Parameter auf ihre Standardwerte. Das gilt auch für Vorgaben wie das Aussehen von Linien oder die Farbe und Muster der Flächenfüllung.

```
cleardevice;
{lösche alles}
```

Name des Record-Datentyps	Funktion	Bestandteile
ArcCoordsType	Koordinaten eines Kreisbogens	X: Integer Y: Integer Xs: Word Ys: Word Xend: Word Yend: Word
FillPatternType	Füllmuster	array[1..8] of Byte;
FillSettingsType	Füllmuster und -farbe	Pattern: Word Color: Word
LineSettingsType	Liniendarstellung	LineStyle: Word Pattern: Word Thickness: Word
PaletteType	Parameter einer Farbpalette	Size: Byte Colors: array[0..MaxColors] of Shortint
PointType	Definition von Punkten	X: Word Y: Word
TextSettingsType	Parameter der Textdarstellung	Font: CharSize Direction: CharSize Horiz: Word Vert: Word
ViewPortType	Parameter eines Grafikfensters	X1: Word Y1: Word X2: Word Y2: Word Clip: Boolean

Tabelle 1. Die voreingestellten Datentypen der Unit »graph«

Grafikkarte	Code	Name
Automatische Feststellung	0	Detect
CGA	1	CGA
MCGA	2	MCGA
EGA	3	EGA
EGA (64 KByte Bildschirmspeicher)	4	EGA64
EGA (einfarbig)	5	EGA-Mono
Hercules	7	Herc-Mono
ATT400	8	ATT400
VGA	9	VGA
PC3270	10	PC3270

Tabelle 2. Die Codierung der unterschiedlichen Grafikkarten

ClearViewPort; löscht das aktuelle logische Grafikfenster und füllt es mit der aktiven Farbe. Das Grafikfenster wird über die Prozedur »SetViewPort« gesetzt, als Füllfarbe wird der erste Eintrag der aktiven Palette verwendet. Die aktive Palette läßt sich über die Kommandos »SetRGBPalette«, »SetPalette« und »SetAllPalette« modifizieren.

```
clearviewport;
{lösche Fenster mit}
{Palette(0)}
```

CloseGraph; schließt die Arbeit im Grafikmodus ab. Der Grafiktreiber, der bei der Initialisierung mit dem Kommando »InitGraph« festgelegt wurde, wird aus dem Speicher entfernt. Der vor Aufruf des »InitGraph«-Befehls verwendete Textmodus wird wiederhergestellt.

```
closegraph;
{zurück in den Textmodus}
```

DetectGraph(<Treiber>, <Modus>); stellt die angeschlossene Grafikkarte fest und setzt die Werte in den Variablenparametern <Treiber> und <Modus>. Beide Parameter sind Integer-Variablen. <Treiber> enthält nach dem Aufruf einen der in Tabelle 2 dargestellten Codes für die verwendete Grafikkarte. Die Variable <Modus> codiert nach dem Aufruf den Grafikmodus. Sie nimmt einen der in Tabelle 3 dargestellten Werte an.

```
var grtab,modus:word;
{Definiere Variable}
detectgraph(grtreib,modus);
{stelle Grafikkarte fest}
```

DrawPoly(<Anzahl>, <Punktefeld>); zeichnet ein Vieleck (Polygon) mit einer beliebigen Anzahl von Eckpunkten. Der Word-Parameter <Anzahl> legt fest, wie viele Punkte zu zeichnen sind. Der Parameter <Punktefeld> enthält ein Array, in dem die X- und Y-Koordinaten der Punkte gespeichert sind. Dazu wird der vordefinierte Datentyp »PointType« verwendet, der in der Tabelle aufgeführt ist. Zum Zeichnen der Linien werden die aktuellen Linienparameter verwendet.

```
var punkte: array[1..10] of pointtype;
{Definition des Punktefeldes}
punkte[1].x:=100;
punkte[1].y:=150;
{Festlegung des ersten Punktes}
drawpoly(10,punkte);
{Zeichnen des Vielecks}
```

Ellipse(<X>,<Y>,<Anfangswinkel>,<Endwinkel>, <X-Radius>,<Y-Radius>); zeichnet eine Ellipse oder einen Ellipsenausschnitt mit dem Mittelpunkt an den Koordinaten <X> und <Y>, die beide als Integer-Werte übergeben werden. Die Word-Parameter <X-Radius> und <Y-Radius> legen die beiden Radien der Ellipse fest. Mit den Word-Werten <Anfangswinkel> und <Endwinkel> kann das zu zeichnende Segment festgelegt werden. Beide Parameter werden als Winkelmaße in Grad interpretiert. Um eine volle Ellipse zu zeichnen, werden beispielsweise die Werte 0 und 360 für <Anfangswinkel> beziehungsweise <Endwinkel> verwendet.

```
ellipse(100,100,0,90,150,100);
{zeichne ein Viertel}
ellipse(100,100,180,360,150,100);
{und ein weiteres Viertel}
```

FillEllipse(<X>,<Y>, <X-Radius>,<Y-Radius>); zeichnet eine Ellipse mit dem Mittelpunkt an der Stelle mit den Koordinaten <X> und <Y>, die beide als Integer-Werte zu übergeben sind. Als Radien werden die Word-Parameter <X-Radius> und <Y-Radius> verwendet. Die Ellipse wird mit den aktuellen Linienparametern gezeichnet. Nach dem Zeichnen der Ellipse wird sie mit der aktiven Füllfarbe und dem aktuellen Füllmuster ausgefüllt.

```
fillellipse(100,100,150,100);
{Zeichne eine Ellipse}
setfillstyle(LineFill,1);
{lege neues Muster fest}
fillellipse(100,100,100,80);
{und zeichne kleinere Ellipse}
```

FillPoly(<Anzahl>,<Punkte>); zeichnet ein Polygon wie der DrawPoly-Befehl, jedoch wird das Vieleck nach den aktuellen Vorgaben für das Füllmuster und die Füllfarbe ausgefüllt.

```
var punkte: array[1..10] of pointtype;
{Definition des Punktefeldes}
punkte[1].x:=60;
punkte[1].y:=160;
{Festlegung des ersten Punktes}
fillpoly(10,punkte);
{Zeichnen des Vielecks}
```

FloodFill(<X>,<Y>, <Randfarbe>); füllt eine beliebige Figur, die den mit den Word-Parametern <X> und <Y> angegebenen Punkt umgibt. Die Figur muß geschlossen sein, damit die »FloodFill«-Prozedur korrekt arbeitet. Als Füllmuster und -farbe werden die aktiven Vorgaben benutzt. Als dritter Parameter legt der als Word-Zahl übergebene Wert von <Randfarbe> fest, in welcher Farbe die Umrandung der gefüllten Fläche gezeichnet werden soll;

```
circle(100,100,50);
{zeichne einen Kreis}
floodfill(100,100,3);
{und fülle ihn aus}
```

GetArcCoords(<Koordinaten>); ermittelt die beim letzten Aufruf der Arc-Prozedur benutzten Koordinaten. Der Parameter <Koordinaten> wird mit dem vordefinierten Datentyp »ArcCoordsType« (Tabelle 1) dekla-

Grafikkarte	Modus	Name	Code
CGA	320x200 (Palette 1)	CGAC1	0
CGA	320x200 (Palette 2)	CGAC2	1
CGA	640x200	CGAHi	1
MCGA	320x200 (Palette 1)	MCGAC1	0
MCGA	320x200 (Palette 2)	MCGAC2	1
MCGA	640x200	MCGAMed	2
MCGA	640x480	MCGAHi	3
EGA	640x200 (4 Seiten)	EGALo	0
EGA	640x350 (16 Farben, 2 Seiten)	EGAHi	1
EGA	640x200 (16 Farben)	EGA64Lo	0
EGA	640x350 (4 Farben)	EGA64Hi	1
EGA	640x350 (2 Farben)	EGAMonHi	3
Hercules	720x348	HercMonHi	0
ATT-400	320x200 (Palette 1)	ATT400C1	0
ATT-400	320x200 (Palette 2)	ATT400C2	1
ATT-400	640x200	ATT400Med	2
ATT-400	640x400	ATT400Hi	3
VGA	640x200 (16 Farben)	VGALo	0
VGA	640x350 (16 Farben)	VGAMed	1
VGA	640x480 (16 Farben)	VGAHi	2
VGA	640x480 (2 Farben)	VGAHi2	3
PC-3270	720x350	PC3270Hi	0

Tabelle 3. Die Codierung der Grafikmodi

riert. Die Prozedur ermittelt die Koordinaten des Mittelpunktes (in den Feldern »X« und »Y« des Datentyps), des Anfangspunktes (in den Feldern »Xs« und »Ys«) und des Endpunktes (in den Feldern »Xend« und »Yend«) des zuletzt gezeichneten Kreisbogens. Durch Verwendung dieser Prozedur ist es möglich, Kreisbögen mit Punkten, Linien und so weiter zu verbinden.

```
var kordi: ArcCoordsType;
{für die beschriebene Funktion}
arc(100,100,0,135,100);
{zeichne einen 3/8 Kreis}
getarccoords(kordi);
{lese Punkte aus};
line(kordi.xs,kordi.ys,
kordi.xend,kordi.yend);
{und verbinde Endpunkte}
```

GetAspectRatio(<Breite>, <Höhe>); dient zur Ermittlung des Verhältnisses der Breite zur Höhe des Grafikbildschirms. Die Breite und die Höhe wird von Turbo Pascal festgestellt und als Word-Wert an die Variablenparameter <Breite> und <Höhe> übergeben. Die Funktion vermeidet Verzerrungen beim Zeichnen von Objekten, die daraus resultieren, daß die Grafikpunkte in der Waagrechten einen anderen Abstand voneinander besitzen als die Punkte in der Senkrechten.

```
var br,ho: word;
{Variablen-Parameter}
getaspectratio(br,ho);
{Funktion aufrufen}
```

GetBkColor; stellt die aktuelle Hintergrundfarbe fest und gibt sie als Word-Funktionsergebnis zurück.

```
write('Die Hintergrundfarbe: ');
writeln(getbkcolor);
{Ermittle die Hintergrundfarbe}
```

GetColor; ermittelt die aktuelle Zeichenfarbe und gibt sie als Word-Funktionsergebnis zurück. Die Farbe kann mit der Prozedur »SetColor« verändert werden.

```
write('Die Zeichenfarbe: ');
writeln(getcolor);
{Ermittle die Zeichenfarbe}
```

GetDefaultPalette(<Palette>); stellt fest, welche Palette bei der Initialisierung des Grafiktreibers aktiviert ist. Der Variablenparameter <Palette> wird mit dem vordefinierten Datentyp »PaletteType« deklariert (Tabelle 1). Nach dem Aufruf der Prozedur wird hier die aktuelle Palette gespeichert. Die Palette ist ein Array von Farbwerten, die benutzt werden, um Farben zu vergeben. Die festgestellte Palette muß nicht die aktuelle Palette sein, die über die Prozedur »GetPalette« ermittelt wird. Die Prozedur »GetDefaultPalette« findet Verwendung, um eine veränderte Palette wieder auf ihre ursprünglichen Werte zu setzen,

Name	Code	Füllmuster
EmptyFill	0	Hintergrundfarbe
SolidFill	1	gewählte Farbe
LineFill	2	waagrechte Linien
LtSlashFill	3	dünne, schräge Linien nach rechts
SlashFill	4	schräge Linien nach rechts
BkSlashFill	5	schräge Linien nach links
LtBk-SlashFill	6	dünne, schräge Linien nach links
HatchFill	7	schraffiert
XHatchFill	8	gekreuzt schraffiert
InterLeave-Fill	9	Verändern der Linien
WideDotFill	10	weit auseinanderliegende Punkte
CloseDot-Fill	11	eng beieinanderliegende Punkte
UserFill	12	frei für Benutzerdefinition

Tabelle 4. Codierung der Füllmuster mit dem Datentyp »FillSettingsType.Pattern«

ohne den Treiber zu initialisieren. Dazu wird die Palette zunächst mit »GetDefaultPalette« ausgelesen und die übergebene Palette danach mit der »SetPalette«-Prozedur als aktive Palette gesetzt.

```
var palette: PaletteType
{definiere die Variable}
var index: word;
{zum Indizieren}
getdefaultpalette(palette);
{lese die palette aus}
for index:=0 to MaxColors do
{zeige Werte in einer Schleife}
begin
write('Farbe ',index,': ');
writeln(palette[index]);
end;
```

GetDriverName; bestimmt den Namen des Grafiktreibers und gibt ihn als String-Funktionsergebnis zurück. Als Namen der Grafiktreiber kommen die in Tabelle 2 unter »Name« aufgeschlüsselten Ergebnisse in Betracht.

```
write('Name des Treibers: ');
writeln(getdrivername);
{stelle Treiber fest}
```

GetFillSettings(<Füllung>); ermittelt das aktuelle Füllmuster und die Füllfarbe und übergibt die Information an den Variablenparameter <Füllung>. <Füllung> wird mit dem vordefinierten Datentyp »FillSettingsType« deklariert (Tabelle 1). Im

Name	Code	Linienart
SolidLn	0	durchgehend
DottedLn	1	gepunktet
CenterLn	2	Strich, Punkt, Strich
DashedLn	3	gestrichelt
UserBitLn	4	frei für Benutzerdefinition

Tabelle 5. Die Codierung der Linienarten mit dem Datentyp »LineSettingsType.Style«

Feld »Pattern« des Datentyp-Records wird das Bitmuster festgehalten, das zum Füllen benutzt wird, das Feld »Color« enthält die aktuelle Füllfarbe.

```
var muster:fillsettingstype;
{definiere den Datentyp}
getfillsettings(muster);
{lese Einstellung aus}
write('Füllfarbe: ');
writeln(muster.Color);
{gebe die Farbe aus}
```

GetGraphMode; stellt den aktuell gesetzten Grafikmodus fest und übergibt ihn als ein Word-Funktionsergebnis. Als Funktionswerte sind die in Tabelle 2 unter dem Namen »Code« aufgeführten Werte möglich. Der tatsächliche Modus kann nur in Kombination mit der Funktion »GetDriver« festgestellt werden, da die Werte für »Code« sich auf die Verwendung von bestimmten Grafikkarten beziehen.

```
write('Der Grafikmodus: ');
writeln('getgraphmode);
{ermittle Modus}
```

GetImage(<X1>,<Y1>,<X2>, <Y2>,<Speicherblock>); kopiert einen Ausschnitt des Grafikbildschirms in einen Speicherblock. Der Bildschirmausschnitt wird durch die Übergabe von zwei Eckpunkten mit den Integer-Parametern <X1>, <Y1> und <X2>, <Y2> festgelegt. Als Bereich zum Speichern des Ausschnitts erwartet Turbo Pascal einen Speicherblock, der mit dem untypisierten Parameter <Speicherblock> spezifiziert wird. <Speicherblock> kann ein Array oder ein Zeiger auf einen freien Block im Heap sein. Es muß lediglich gewährleistet sein, daß der Block groß genug ist, um alle Daten des Bildschirmausschnitts aufzunehmen. In den ersten beiden Worten des Speicherbereichs legt Turbo Pascal die Höhe und Breite des gespeicherten Ausschnitts ab. Wie Turbo Pascal Variablen intern speichert, erfahren Sie im Artikel »Management statt Bürokratie – Die Speicherverwaltung von Turbo Pascal« in dieser Ausgabe. Das Schreiben des Bildausschnitts zurück in den Bildschirmspeicher erledigt die Prozedur »PutImage«.

```
var block: array[0..100,0..100]
of boolean;
{definiere Speicherblock}
getimage(50,50,150,150,block);
{und speichere ihn}
```

GetLineSettings<Linie>); ermittelt die aktuellen Einstellungen zum Zeichnen von Linien und übergibt sie an den Variablenparameter <Linie>. <Linie> wird über den vordefinierten Datentyp »LineSettingsType« (Tabelle 1) deklariert. Es lassen sich die Dicke, das Aussehen und die Linienmuster feststellen. Das Feld »Style«

des Datentyp-Records hält das Aussehen der Linie fest. Die möglichen Werte sind in Tabelle 5 zusammengefaßt. Die Dicke der Linien wird im Feld »Thickness« vermerkt, mögliche Inhalte sind in Tabelle 6 aufgeführt.

```
var line: linesettingstype;
{definieren Variable}
getlinesettings(linie);
{lese Parameter aus}
```

GetMaxMode; stellt den maximal möglichen Wert für den Parameter <Modus> bei den Grafikfunktionen und -prozeduren fest, die einen Grafikmodus setzen oder auslesen. Das Funktionsergebnis wird als Word-Wert zurückgeliefert.

```
write('Maximal möglich ist ');
writeln(getmaxmode);
{gebe max. Anzahl aus}
```

GetMaxX; stellt die maximal verwendbare X-Koordinate des aktuellen Bildschirms fest und übergibt sie als Word-Funktionsergebnis.

```
write('Anzahl der horiz. Punk');
writeln('te: ',getmaxx);
{gebe Anzahl der X-Punkte aus}
```

GetMaxY; ermittelt die maximal verwendbare Y-Koordinate des aktuellen Bildschirms und liefert sie als Word-Funktionsergebnis an das aufrufende Programm zurück.

```
write('Anzahl der vert. Punk');
writeln('te: ',getmaxy);
{gebe Anzahl der Y-Punkte aus}
```

GetModeName; stellt den Namen des aktuellen Grafikmodus fest und übergibt ihn als String-Funktionsergebnis. Als Namen kommen für den Modus die in Tabelle 1 unter »Name« aufgeführten Strings in Betracht.

```
var modus: string;
{definiere Variable}
modus:=getmodename;
{und speichere Modus}
```

GetPalette(<Palette>); bestimmt, welche Farbpalette zur Zeit aktiv ist und legt sie in dem Variablenparameter <Palette> ab. <Palette> wird mit dem vordefinierten Datentyp »PaletteType« (Tabelle 1) deklariert.

```
var palette:palettetype;
{definiere Variable}
var index: word;
{für Schleife}
getpalette(palette);
{lese Parameter aus}
for index:=0 to maxcolors do
begin
write('Farbe ',index,': ');
writeln(palette[index]);
end;
```

GetPaletteSize; stellt die Anzahl der Einträge in der aktuellen Palette fest und übergibt sie als ein Word-Funktionsergebnis.

```
write('Anzahl der Einträge: ');
writeln(getpalettesize);
```

Name	Code	Liniendicke
NormalWidth	1	normale Dicke
ThickWidth	3	doppelte Dicke

Tabelle 6. Die Codierung der Liniendicke mit dem Datentyp »LineSettingsType. Thickness«

GetPixel(<X>,<Y>); liest die Farbe des Punktes mit den Integer-Koordinaten <X> und <Y> aus und liefert sie als Word-Funktionsergebnis an das aufrufende Programm zurück.

```
var wert: word;
{Zum Speichern eines Punktes}
wert:=getpixel(100,100);
{speichere den Punkt 100,100}
```

GetTextSettings(<Text>); stellt die aktuellen Einstellungen für die Darstelllung von Texten im Grafikmodus fest und übergibt sie an den Variablenparameter <Text>. <Text> wird über den vordefinierten Datentyp »TextSettingsType« deklariert (Tabelle 1). Die Größe eines Zeichens des Zeichensatzes wird mit dem Feld »CharSize« des Datentyps festgelegt, sie liegt zwischen 1 und 10. Die Felder »Horiz« und »Vert« halten die Breite und Höhe eines Zeichens in Punkten fest. Die Richtung, in die geschrieben wird, steuern die beiden Felder »HorizDir« und »VertDir«.

```
var text:textsettingstype;
{Variable definieren}
gettextsettings(text);
{Parameter auslesen}
```

GetViewSettings<Viewport>; übergibt die Parameter des aktuellen Grafikfensters an den Variablenparameter <Viewport>. <Viewport> wird über den vordefinierten Datentyp »ViewPortType« deklariert. Die Felder »X1«, »Y1«, »X2« und »Y2« des Datentyp-Records enthalten zwei Ecken des Grafikfensters, das Boolean-Feld »Clip« legt fest, ob Linien, Kreise und andere Objekte geclippt (also abgeschnitten) werden sollen, wenn sie über das Grafikfenster hinausreichen.

```
var bild:vierporttype;
{definiere Variable}
```

Name	Code	Zeichensatz
DefaultFont	0	normaler Zeichensatz
TriplexFont	1	Triplex-Zeichensatz
SmallFont	2	Zeichensatz mit kleinen Buchstaben
SansSerifFont	3	Zeichensatz mit Sans-Serif-Schriftart
GothicFont	4	Zeichensatz mit Gothic-Schriftart

Tabelle 7. Die Codierung der Zeichensätze mit dem Datentyp »TextSettingsType. Font«

```
Getviewport(bild);
{lese Fenster aus}
```

GetX; ermittelt die aktuelle X-Koordinate des Grafik-Cursors und übergibt sie als Integer-Funktionsergebnis.

```
var x_alt:integer;
{zum Speichern von X}
x_alt:=getx;
{Speichere X}
```

GetY; stellt die aktuelle Y-Koordinate des Grafik-Cursors fest und liefert sie als Integer-Funktionsergebnis an das aufrufende Programm zurück.

```
var y_alt:integer;
{zum Speichern von Y}
Y_alt:=getx;
{Speichere Y}
```

GraphErrorMsg(<Fehler>); ermittelt eine Fehlermeldung, die zu dem mit <Fehler> übergebenen Fehlercode paßt. Als Funktionsergebnis wird ein String zurückgegeben. Der mit dem Parameter <Fehler> übergebene Fehlercode wird über die Funktion »GraphResult« ermittelt.

```
write(grapherrormsg(fehler));
{Gebe Fehlermeldung aus}
```

GraphResult; stellt fest, ob die zuletzt durchgeführte Operation einen Fehler bewirkt hat. Als Funktionsergebnis wird ein Integer-Wert zurückgeliefert, der einen der in Tabelle 12 aufgeführten Werte annehmen kann.

```
write('Fehler: ');
writeln(graphresult);
{Gebe den Fehler-Code aus}
```

ImageSize(<X1>,<Y1>,<X2>,<Y2>); ermittelt die gesamte Größe, die ein Bildschirmausschnitt benötigt, um über die Prozedur »GetImage« gespeichert zu werden. Die beiden Koordinatenpaare <X1>, <Y1> und <X2>, <Y2> geben die linke obere und rechte untere Ecke des Bildschirmausschnitts an. Das Ergebnis wird als Word-Funktionswert zurückgeliefert.

```
var an:word;
{Lege Variable an}
an:=imagesize(1,100,100,1);
{Lese Größe aus}
```

InitGraph(<Treiber>,<Modus>,<Pfad>); initialisiert den Grafikbildschirm, indem der zu der Grafikkarte passende Grafiktreiber vom Datenträger geladen wird. Die beiden Variablenparameter <Treiber> und <Modus> (beide Integer-Typen) spezifizieren den Grafiktreiber (Tabelle 2) und den Grafikmodus (Tabelle 3). Der String-Parameter <Pfad> enthält den kompletten Suchpfad zum Grafiktreiber. Grafiktreiber werden als Dateien mit der Endung »bgi« gespeichert. Ein gültiger String für <Pfad> ist beispielsweise »c:\turbo5\bgidrv\«. Die Angabe der Parameter <Treiber> und <Modus> ist

optional. Wird für <Treiber> der Wert 0 übergeben, so erkennt Turbo Pascal automatisch, welche Grafikkarte installiert ist (durch Aufruf der Prozedur »DetectGraph«), und setzt die entsprechenden Werte für <Treiber> und <Modus>.

```
var treiber,modus: integer;
{definiere Variablen}
const pfad='c:\turbo5\';
{lege Pfad fest}
treiber:=0;
{automatische Erkennung}
initgraph(treiber,modus,pfad);
{initialisiere Grafikmodus}
```

Line(<X1>,<Y1>,<X2>,<Y2>); zeichnet eine Linie vom Anfangspunkt mit den Koordinaten <X1> und <Y1> bis zum Endpunkt mit den Koordinaten <X2> und <Y2>. Alle Koordinaten sind Integer-Werte. Zum Zeichnen wird die aktuelle Linienart und -farbe verwendet.

```
line(10,10,10,100);
line(10,100,100,100);
line(100,100,100,10);
line(100,10,10,10);
{zeichne Quadrat}
```

LineRel(<X-Differenz>, <Y-Referenz>); erzeugt eine Linie wie die Line-Prozedur. Allerdings werden die Anfangs- und Endkoordinaten nicht direkt, sondern relativ zum letzten gezeichneten Punkt angegeben. <X-Differenz> gibt die Anzahl der waagrechten Punkte, <Y-Differenz> die Anzahl der senkrechten Punkte an. Beide Parameter sind Integer-Werte. Bei positivem Vorzeichen wird nach rechts beziehungsweise oben, bei negativem Vorzeichen nach links beziehungsweise unten gezeichnet. Zum Zeichnen wird die aktuelle Linienart und -farbe verwendet.

```
line(10,10,100,100);
{eine Linie}
linerel(90,90);
{wird verdoppelt}
```

LineTo(<X>,<Y>); zeichnet eine Linie von der zuletzt angesprochenen Position des Grafik-Cursors bis zu dem Punkt, der durch die Integer-Koordinaten <X> und <Y> angegeben wird. Die Linie wird mit den aktiven Einstellungen für Linienstil und -farbe gezeichnet.

```
line(10,10,100,100);
{erste Linie}
lineto(100,10);
{und zweite hintendran}
```

MoveRel(<X-Differenz>,<Y-Referenz>); bewegt den Grafik-Cursor von der zuletzt angesprochenen Koordinate weiter, ohne dabei eine Linie zu zeichnen. Der Cursor wird zu der Koordinate bewegt, die sich aus der Summe der aktuellen X- beziehungsweise Y-Koordinate und den Integer-Parametern <X-Differenz> und <Y-Differenz> ergibt. Für positive Werte der Parameter wird der Cursor nach rechts beziehungsweise oben bewegt, für negative nach links beziehungsweise unten.

```
line(10,10,200,10);
{zeichne Linie}
moverel(0,50);
{bewege Cursor}
lineto(10,10);
{und zum Ausgangspunkt}
```

MoveTo(<X>,<Y>); bewegt den Grafik-Cursor zu dem Punkt mit den Koordinaten <X> und <Y>, ohne dabei eine Linie zu zeichnen. Beide Parameter werden als Integer-Werte übergeben.

```
line(10,10,50,50);
{zeichen Linie}
moveto(60,60);
{bewege Cursor}
linerel(40,40);
{und zeichne weiter}
```

OutText(<Text>); schreibt die mit dem String-Parameter <Text> übergebene Zeichenkette ab der aktuellen Position des Grafik-Cursors auf den Bildschirm. Für das Aussehen des Textes werden die aktuellen Einstellungen benutzt, die sich über die Prozeduren und Funktionen »GetTextSettings« beziehungsweise »SetTextStyle« und »SetTextJustify« auslesen oder verändern lassen.

Name	Code	Farbe
Black	0	Schwarz
Blue	1	Blau
Green	2	Grün
Cyan	3	Türkis
Red	4	Rot
Magenta	5	Weinrot
Brown	6	Braun
LightGray	7	Hellgrau
DarkGray	8	Dunkelgrau
LightBlue	9	Hellblau
LightGreen	10	Hellgrün
LightCyan	11	Helltürkis
LightRed	12	Hellrot
LightMagenta	13	Hellweinrot
Yellow	14	Gelb
White	15	Weiß

Tabelle 9. Die Codierung der Standardfarben

Name	Code	Schreibmodus
NormalPut	0	Punkte werden ersetzt
XorPut	1	Punkte werden logisch Exklusiv-Oder-verknüpft
OrPut	2	Punkte werden,logisch Oder-verknüpft
AndPut	3	Punkte werden logisch Und-verknüpft
NotPut	4	Punkte werden invers gesetzt

Tabelle 8. Die Codierung des Schreibmodus bei Arbeiten mit »PutImage«

```
outtext('Das ist ein Text');
{gebe Text aus}
```

OutTextXY(<X>,<Y>,<Text>); schreibt einen Text im Grafikmodus wie »OutText«, jedoch unter zusätzlicher Angabe der Koordinaten, ab welchen der Text ausgegeben werden soll. Dazu übergeben Sie die beiden Integer-Parameter <X> und <Y>, mit denen die Koordinaten festgelegt werden.

```
const tr:string='Probetext';
{definiere Text}
outtextxy(tr,100,100);
{und gebe ihn an 100,100 aus}
```

PieSlice(<X>,<Y>,<Startwinkel>,<Endwinkel>,<Radius>); zeichnet einen Kreisausschnitt. Der Mittelpunkt des Kreises wird durch die Integer-Werte <X> und <Y> festgelegt. Der Kreisausschnitt wird zwischen den beiden durch die Word-Werte <Startwinkel> und <Endwinkel> angegebenen Winkeln gezeichnet. <Startwinkel> und <Endwinkel> werden als Gradzahlen im Winkelmaß interpretiert. Den Radius des Kreises legen Sie über den Word-Parameter <Radius> fest. Der Kreisausschnitt wird mit den aktuellen Einstellungen für Linienstil und -farbe erstellt.

```
pieslice(100,100,0,90,100);
{Zeichne Viertelkreis mit R=100}
pieslice(100,100,270,360,50);
{und gegenüber mit R=50}
```

PutImage(<X>,<Y>,<Puffer>,<Modus>); erlaubt das Kopieren eines Speicherblocks in den Bildschirmspeicher. Turbo Pascal geht davon aus, daß sich ab der durch den untypisierten Variablenparameter <Puffer> angegebenen Adresse ein kontinuierlicher Speicherblock befindet, dessen erste beiden Worte die Breite und Höhe des Ausschnitts definieren. Zusammenhängende Speicherblöcke können am leichtesten auf dem Heap oder durch Arrays erzeugt werden. Der Artikel »Management statt Bürokratie – Die Speicherverwaltung von Turbo Pascal« stellt dar, wie Turbo Pascal Variablen intern speichert. Die Integer-Parameter <X> und <Y> legen die linke obere Ecke fest, an der die Darstellung des Bildausschnitts beginnen soll. Mit dem Word-Parameter <Modus> läßt sich bestimmen, auf welche Weise der Bildausschnitt mit eventuell vorhandenen Punkten des Bildschirms verknüpft werden soll. Tabelle 8 zeigt, welche Werte Turbo Pascal hier erlaubt. Der Befehl »PutImage« wird in der Regel verwendet, um zuvor mit der Prozedur »GetImage« gesicherte Bildausschnitte wieder auf dem Bildschirm darzustellen.

Name	Code	horizontale Justierung
LeftText	0	Text beginnt am Grafik-Cursor
Center-Text	1	Grafik-Cursor steht in Textmitte
RightText	3	Text endet am Grafik-Cursor

Tabelle 10. Die Codierung der horizontalen Textposition

```
var block:array[0..10,0..10] of byte;;
{Variable für Mono-Screen}
block[0]:=10;
block[1]:=0;
block[2]:=10;
block[3]:=0;
{Höhe und Breite auf 10}
putimage(100,100,block,normalput);
{Block darstellen}
putimage(150,100,block,notput);
{und daneben invers}
```

PutPixel(<X>,<Y>,<Farbe>); setzt einen Bildpunkt auf dem Bildschirm an den Koordinaten, die durch die Integer-Parameter <X> und <Y> angegeben werden. Die durch den Word-Parameter <Farbe> angegebene Farbe wird zur Einfärbung des Punktes benutzt.

```
putpixel(100,100,5);
{setze Punkt}
```

Rectangle(<X1>,<Y1>,<X2>,<Y2>); zeichnet ein Rechteck, von dem mit den beiden Koordinatenpaaren <X1/Y1> und <X2/Y2> die obere linke und die untere rechte Ecke angegeben werden. Die Übergabe aller Koordinaten erfolgt mit Integer-Werten. Zum Zeichnen werden die aktuellen Einstellungen für Linienstil und -farbe verwendet.

```
rectangel(10,100,100,10);
{erstelle ein Quadrat}
```

RestoreCrtMode; setzt den Bildschirm zurück in den Textmodus, der vor der Initialisierung des Grafikmodus mit dem Befehl »InitGraph« aktiv war.

```
restorecrtmode;
{zurück in den Textmodus}
```

Sector(<X>,<Y>,<Startwinkel>,<Endwinkel>,<X-Radius>,<Y-Radius>); erstellt ein ausgefülltes Kreis- oder Ellipsensegment. Als Mittelpunkt der Ellipse (ein Kreis ist eine Ellipse mit zwei gleichen Radien) wird der Punkt mit den Integer-Koordinaten <X> und <Y> verwendet. Die Ellipse wird zwischen den Winkelbögen <Startwinkel> und <Endwinkel> gezeichnet, die als Word-Parameter Winkel in Gradmaß übergeben. Die Festlegung der beiden Radien der Ellipse erfolgt durch die Word-Parameter <X-Radius> und <Y-Radius>. Zum Zeichnen der Ellipse werden die aktuellen Einstellungen für die Linienart und -farbe verwendet. Nach dem Zeichnen füllen die Routinen die Ellipse mit den aktuellen Parametern für Füllmuster und -farbe. Mit der Sector-Prozedur lassen sich auf einfache Weise »Kuchenstücke« und »Tortenstücke« erstellen.

```
sector(100,10,0,90,50,50);
{zeichne Kreissegment}
sector(100,10,90,180,50,100);
{und Ellipsensegment}
```

Name	Code	vertikale Justierung
Bottom-Text	0	Text steht oberhalb des Grafik-Cursors
Center-Text	1	Text, Grafik-Cursor steht in Textmitte
TopText	2	Text steht unterhalb des Grafik-Cursors

Tabelle 11. Die Codierung der vertikalen Textposition

SetAllPalette(<Palette>); erlaubt die Neufestsetzung der Farbpalette.

Ordnung und Übersicht schaffen die beliebten DMV Sammelmappen

Bitte Bestellkarte benutzen
DMV Verlag · Postfach 250 · 3440 Eschwege

Eine Bitte an unsere Abonnenten

Vermerken Sie bei Schriftverkehr und Zahlungen neben der vollständigen Anschrift stets Ihre Abo-Nummer.

Sie vermeiden damit unnötige Verzögerungen bei der Bearbeitung Ihres Abonnements.

Vielen Dank

Ihre DMV-Versandabteilung

24-Nadeldrucker - ein Problem ?

24-Nadeldrucker liefern erheblich bessere Schriftbilder als 9-Nadeldrucker. Dies ist nicht immer der Fall. Der überwiegende Teil der SoftWare hat keinen 24-Nadel-DruckerTreiber. D.h. Schriften oder Grafik sind im Ausdruck meist blaß und zu lang. Dies liegt im fehlenden oder falschen Treiber. Da für jedes Programm ein richtiger Treiber notwendig ist, beginnt die Suche. Schnell werden Sie feststellen, daß es garnicht für jede SoftWare einen passenden Treiber gibt. Und wenn einer aufgetrieben wird, so ist er manchmal kompliziert zu installieren oder bringt immer noch nicht die Qualität, zu der Ihr Drucker in der Lage wäre. Wir haben dieses Problem auf elegante und komfortable Weise mit unserem HardWareEmulator **BITMASTER-plus** gelöst. Einmal installiert und für alle Programme geeignet. Interessiert? Senden Sie uns bitte ein mit DM 4,- frankiertes A4 Kuvert und wir senden Ihnen eine komplette Infomappe mit Testberichten zu. Übrigens, sollten Sie gerade einen guten 24-Nadeldrucker oder Monitor suchen, hier ein paar Top Angebote:

NEC P2plus	DM	789,-
NEC P6plus	DM	1198,-
NEC P7plus	DM	1598,-
NEC LC-866	DM	4995,-
NEC LC-890	DM	6999,-
NEC 2A	DM	1099,-
NEC 3D	DM	1499,-
NEC 4D	DM	2998,-
NEC 5D	DM	5498,-

RKT GmbH
Postfach 710844
8000 München 71
Tel: 089-7592026
Fax: 089-7595150

DOEX 11/90

Dazu wird mit dem Parameter <Palette> eine komplette Farbpalette übergeben. <Palette> wird über den vordefinierten Datentyp »PaletteType« (Tabelle 1) deklariert. Für die Festlegung der einzelnen Farben der Palette lassen sich die in Tabelle 9 aufgeführten vordefinierten Konstanten verwenden. Eine Änderung der Palette ist vor allem dann sinnvoll, wenn Sie Farbwechsel auf einem bestehenden Grafikbildschirm vornehmen wollen, ohne alle Punkte neu setzen zu müssen. Soll innerhalb der Palette ein Wert aus der vorher aktiven Palette unverändert übernommen werden, wird <Palette> am entsprechenden Index auf -1 gesetzt.

```
var palette:palettetype
{definiere Variable}
palette[2]:=red;
{Grün wird zu rot}
```

SetAspectRatio(<Breite>, <Höhe>); gibt mit den Word-Parametern <Breite> und <Höhe> das Verhältnis der horizontalen zur vertikalen Bildschirmauflösung an. Die Funktion können Sie benutzen, um bei Bildschirmen, deren Auflösungsverhältnis nicht dem Standard entspricht, neue Parameter vorzugeben und eine Verzerrung beispielsweise von Kreisen auszuschließen.

```
setaspectratio(300,200);
{lege neues Verhältnis fest}
```

SetBkColor(<Farbe>); legt die Hintergrundfarbe des Bildschirms fest. Der Word-Parameter <Farbe> wird als Zeiger in die aktuelle Palette verwendet.

```
setbkcolor(0);
{setze Hintergrund auf erste}
{Farbe der Palette}
```

SetColor(<Farbe>); stellt die Zeichenfarbe ein. Der Word-Parameter <Farbe> wird als Index in die aktuelle Farbpalette verwendet.

```
setcolor(15);
{setze die letzte Farbe der}
{Palette als aktuelle Farbe}
```

SetFillPattern(<Muster>, <Farbe>); legt ein benutzerdefiniertes Füllmuster fest, mit dem Flächen ausgefüllt werden. Der Parameter <Muster> wird über den vordefinierten Datentyp »FillPatternType« (Tabelle 1) deklariert. Damit läßt sich ein Muster von 8x8 Bit erzeugen, das als Füllmuster verwendet wird. Der Word-Parameter <Farbe> wird als Index in die aktuelle Farbpalette interpretiert. Das Füllmuster verwendet Turbo Pascal, sobald über die Prozedur »SetFillStyle« als Stil »UserFill« festgelegt wird.

```
const muster:
fillpatterntype=(0,1,2,3,4,5,6,7);
{lege Muster fest}
```

Name	Code	Fehler
grOK	0	kein Fehler
grNoInitGraph	–1	Grafiktreiber nicht installiert
grNotDetected	–2	Grafikkarte konnte nicht festgestellt werden
grFileNotFound	–3	Grafiktreiber nicht auf Datenträger gefunden
grInvalidDriver	–4	Grafiktreiber paßt nicht zur Grafikkarte
grNoLOadMem	–5	kein Speicherplatz zum Laden des Grafiktreibers
grNoScanMem	–6	kein Speicherplatz für ScanFill
grNoFloodFill	–7	kein Speicherplatz für FloodFill
grFontNotFound	–8	Zeichensatz nicht auf Datenträger gefunden
grNoFontMem	–9	kein Speicherplatz für Zeichensatz
grInvalidMods	–10	Grafikmodus paßt nicht zu Grafikkarte

Tabelle 12. Die Codierung der Fehlermeldungen

```
setfillpattern(muster,5);
{und setze es}
```

SetFillStyle(<Stil>,<Farbe>); setzt die Art und Weise, in der Flächen gefüllt werden. Der Word-Parameter <Stil> kann einen der in Tabelle 4 dargestellten Werte annehmen. Er gibt an, auf welche Weise Flächen zu füllen sind. Falls keine der voreingestellten Flächenvorgaben verwendet werden soll, läßt sich über den Befehl »SetFillPattern« ein neues Muster definieren. Es läßt sich mit »SetFillStyle« aktivieren, indem für <Stil> der vordefinierte Wert »UserFill« verwendet wird. Der Word-Parameter <Farbe> wird als Index in die aktuelle Farbpalette interpretiert und legt die Farbe fest, mit der Flächen zu füllen sind.

```
setfillstyle(HatchFill,15);
{schraffierte Flächen mit}
{Farbe 15}
```

SetGraphMode(<Modus>); setzt einen neuen Grafikmodus. Die Funktion läßt sich nur aktivieren, wenn ein erfolgreicher Aufruf von »InitGraph« bereits erfolgt ist. Mit dem Word-Parameter <Modus> wählen Sie einen Grafikmodus aus, der aktiviert wird, ohne die aktuellen Einstellungen für den Grafiktreiber zu ändern. Für den Grafikmodus können die in Tabelle 3 aufgeführten vordefinierten Konstanten verwendet werden.

```
setgraphmode(cgahi);
{Setze 640x200-CGA-Modus}
```

SetLineStyle(<Stil>,<Muster>, <Dicke>); legt fest, welches Aussehen Linien haben, mit denen Geraden, Kreise, Ellipsen und so weiter gezeichnet werden. Der Word-Parameter <Stil> kann einen der in Tabelle 5 dargestellten Werte annehmen und gibt an, wie Linien zu zeichnen sind. Den Word-Parameter <Muster> müssen Sie nur übergeben, wenn für <Stil> der Wert »UserBitLn« übergeben wird. In diesem Fall definiert <Muster> ein 16-Bit-Muster, das zum Zeichnen der Linien Verwendung findet. Mit dem Parameter <Dicke> kann die Dicke der Linien gesetzt werden. Dazu sind die in Tabelle 6 aufgeführten Konstanten vordefiniert.

```
setlinestyle(dashedln,0,thickwidth);
{gestrichelte, dicke Linien}
```

SetPalette(<Farbindex>, <Farbe>); gibt für einen Eintrag der Farbpalette eine neue Farbe vor. Der Word-Parameter <Farbindex> gibt die Nummer der Farbe an, die in der Palette zu ändern ist. Mit dem Word-Wert <Farbe> wird die neue Farbe gesetzt. Dazu können Sie die in Tabelle 9 aufgeführten, vordefinierten Konstanten verwenden.

```
setwritemode(xorput);
{lege XOR-Verknüpfung fest}
```

TextHeight(<Text>); stellt fest, wie viele Punkte die mit dem String-Parameter <Text> übergebene Zeichenkette in der Höhe belegt, wenn sie mit den aktuellen Einstellungen zur Ausgabe von Text auf dem Bildschirm ausgegeben wird. Das Funktionsergebnis wird als Word-Wert zurückgeliefert.

```
const tr:string='Ein Text';
{definiere den Text}
var anzahl: word;
{Zur Speicherung der Anzahl}
anzahl:=textheight(tr);
{Ermittle die Punktanzahl}
```

TextWidth(<Text>); bestimmt, wie viele Punkte die mit dem String-Parameter <Text> übergebene Zeichenkette in der Breite belegt, wenn sie mit den aktuellen Einstellungen zur Ausgabe von Text auf dem Bildschirm ausgegeben wird. Das Funktionsergebnis wird als Word-Wert zurückgeliefert.

```
const tr:string='Ein Text';
{definiere den Text}
var anzahl: word;
{Zur Speicherung der Anzahl}
anzahl:=textwidth(tr);
{Ermittle die Punktanzahl}
```

Mit dieser Beschreibung der Grafikroutinen von Turbo Pascal eröffnen sich vielfältige Möglichkeiten der Grafikprogrammierung. Vielleicht haben auch Sie das eine oder andere Programm, das Sie gerne grafisch »aufpeppen« möchten – dem steht nun nichts mehr im Wege.

(Martin Althaus/dm)

DATARAM
computertechnik

ZUBEHÖR

Genius Maus GM-F 302	99,-
Genius Maus GM 6000	89,-
LowCost Maus 3Tasten	59,-
Geniscan GS-4500 inkl. Text.	399,-
Monitor NEC 2A, deutsch	1148,-
Monitor NEC 3D, deutsch	1448,-
Monitor 14" Monochrom	229,-
Optima VGA, 512K, 16 Bit 1024x768, Non-Interlace	389,-
VGA-Karte, 16Bit, 256K	249,-
Grafikkarte + Printerport	49,-
LW 3.5" 1.44 MB	179,-
LW 5.25" 1.2 MB	169,-

Top Angebote für
SEAGATE-Festplatten
und sonstiges Zubehör

!! Rufen Sie uns an !!

AT 286-12VGA

AT-12MHz, 0 Wait, EMS 4.0
1 MB Ram, bis 4 MB on Board
Sockel Co-Prozessor 80287
Uhr/Kalender akkugepuffert
Turbo/Reset/Schlüsselschalter
1Ser./1Par./1Gameport
Baby AT-Gehäuse + 200 W Netzteil
Speeddisplay LED, 8 Slots
Einschub 3x5.25" u. 1x3.5"
VGA 16 Bit, 256K, 800x600
LW 5.25" 1.2MB oder 3.5" 1.44MB
16 Bit MFM Controller 1:1
21MB/38ms Festplatte
14" VGA Monitor paperwhite
Maus Genius GM-F302
MS-DOS 3.3 + GW Basic
1 JAHR VOLLGARANTIE

2498,-

DRUCKER

Star LC 24-10	648,-
Star LC 24-15	1248,-
Star XB 24-10	1448,-
Star XB 24-15	1798,-
Star Laser 8II	4798,-

NEC, OKI, EPSON, usw. auf Anfrage
Als autorisierter
STAR-Händler
gewähren wir auf alle Produkte
der Firma STAR
1 Jahr Garantie !

AT 386-20VGA

AT-20MHz, 0 Wait, EMS 4.0
2 MB Ram, bis 8 MB on Board
Sockel Co-Prozessor 80387
Uhr/Kalender akkugepuffert
Turbo/Reset/Schlüsselschalter
1Ser./1Par./1Gameport
Towergehäuse 6fach
Speeddisplay LED
Netzteil 200 W
VGA 16 Bit, 256K, 800x600
LW 5.25" 1.2MB oder 3.5" 1.44MB
16 Bit MFM Controller 1:1
42MB/28ms Festplatte
14" VGA Monitor paperwhite
Maus Genius GM-F302
MS-DOS 3.3 + GW Basic
1 JAHR VOLLGARANTIE

4499,-

DATARAM Computertechnik
Lipper Weg 131, 4370 Marl, TEL.: 02365/691431 FAX: 63207

Öffnungszeiten: Mo-Fr 9-13 u.15-18 Uhr Sa 10-13 Uhr

Alle Angebote freibleibend, Preisänderungen und Irrtümer vorbehalten
Versand erfolgt per UPS-Nachnahme zu Selbstkosten

HÄNDLERANFRAGEN nur mir Gewerbenachweis !

Die Variablen in Turbo Pascal

Turbo Pascal bietet eine Menge unterschiedlicher Variablen an. Die korrekte Auswahl der Variablentypen ist ein wichtiger Aspekt bei der Programmierung. Dieser Artikel zeigt die Unterschiede der einzelnen Typen und gibt Hinweise für ihre Verwendung.

Variablen werden in Turbo Pascal an zwei Stellen definiert: in der Var-Anweisung im Deklarationsteil eines Programms, einer Prozedur oder einer Funktion und im Kopf einer Funktion oder Prozedur. Falls Sie mit den Grundbegriffen wie »Deklarationsteil«, »Prozedur«, »Funktion« oder »Kopf« noch nicht vertraut sind, empfiehlt sich zunächst die Lektüre des Artikels »Wie sieht ein Pascal-Programm aus?« auf Seite 38.

Die Auswahl der richtigen Variablentypen für die Programmierung bestimmter Probleme entscheidet wesentlich über die Ausführungsgeschwindigkeit eines Programms. Je komplexer ein Datentyp ist, desto mehr Zeit wird für seine Verarbeitung benötigt.

Für die Definition von Variablen bietet Turbo Pascal einige Standardtypen an. Diese Standardtypen legen fest, welchen Inhalt eine Variable bekommen darf. Die Standardtypen lassen sich in drei Gruppen aufteilen: Typen für ganze Zahlen, Typen für Fließkommazahlen und Typen für sonstige Variablen.

Typen für ganze Zahlen

Mit diesen Datentypen werden Variablen definiert, die ganze Zahlen enthalten sollen, also Zahlen ohne Nachkommastellen. Turbo Pascal bietet zwei Grundtypen von ganzen Zahlen an: vorzeichenlose und vorzeichenbehaftete Zahlen. Die vorzeichenlosen Zahlen haben einen positiven Wertebereich, können also keine Werte kleiner Null aufnehmen, während die vorzeichenbehafteten Zahlen auch Werte unter Null beinhalten können.

Tabelle 1 faßt die Pascal-Datentypen für ganze Zahlen zusammen. Im Gegensatz zu den Typen für Fließkommazahlen werden die Inhalte von Ganzzahlvariablen bei der Ausgabe auf dem Bildschirm in gewohnter Schreibweise und nicht in Exponentialschreibweise dargestellt.

Name	Wertebereich
Byte	0 bis 255
Word	0 bis 65535
Shortint	-128 bis 127
Integer	-32768 bis 32767
Longint	-2147483648 bis 2147483647

Tabelle 1. Die Variablentypen für ganze Zahlen

Byte

Dieser vorzeichenlose Datentyp wird für kleine Zahlen verwendet, die einen Wertebereich von 0 bis 255 besitzen. Bytevariablen werden häufig für kleine Zählschleifen verwendet. Ein Byte läßt sich im Speicher unmittelbar codieren. Beim direkten Zugriff auf einzelne Speicherstellen werden daher in der Regel Bytevariablen verwendet.

Word

Der vorzeichenlose Word-Datentyp erlaubt einen Wertebereich von 0 bis 65535. Word-Variablen werden häufig in Schleifen und zur Adressierung von Speicherstellen verwendet.

Shortint

Der vorzeichenbehaftete Shortint-Typ nimmt Daten im Bereich von -128 bis 127 auf. Intern wird dieser Typ wie eine Bytevariable ebenfalls in einem einzigen Byte gespeichert.

Integer

Integer-Variablen sind vorzeichenbehaftet und stellen einen Wertebereich von -32768 bis 32767 zur Verfügung. Sie belegen im Speicher des Computers ein Wort, also zwei Byte.

Longint

Die vorzeichenbehafteten Longint-Variablen können Werte im Bereich von -2147483648 bis 2147483647 aufnehmen. Sie sind für Anwendungen konzipiert, in denen der Wertebereich des Integer-Typs nicht mehr ausreicht.

Name	Wertebereich	Genauigkeit
Real	$2{,}9\times10^{-39}$ bis $1{,}7\times10^{38}$	11 bis 12 Stellen
Single	$1{,}5\times10^{-45}$ bis $3{,}4\times10^{38}$	7 bis 8 Stellen
Double	5×10^{-324} bis $1{,}7\times10^{308}$	15 bis 16 Stellen
Extended	$1{,}9\times10^{-4951}$ bis $1{,}1\times10^{4932}$	19 bis 20 Stellen
Comp	$-9{,}2\times10^{18}$ bis $9{,}2\times10^{18}$	18 bis 19 Stellen

Tabelle 2. Die Variablentypen für Kommazahlen

Typen für Fließkommazahlen

Für den Umgang mit Real-Zahlen, also Zahlen, die einen Vor- und Nachkommateil besitzen, stellt Turbo Pascal fünf unterschiedliche Datentypen zur Verfügung, die auch als Real-Typen bezeichnet werden. Im Gegensatz zu den ganzen Zahlen gibt es keine Real-Datentypen, die nur positive Zahlen aufnehmen können, also vorzeichenlos arbeiten. Tabelle 2 faßt die Pascal-Datentypen für Kommazahlen zusammen. Bei der Ausgabe auf dem Bildschirm werden die Real-Datentypen im Gegensatz zu den Ganzzahltypen immer in Exponentialschreibweise dargestellt.

Real

Mit dem Real-Datentyp werden Variablen mit einem Wertebereich von $2{,}9\times10^{-39}$ bis $1{,}7\times10^{38}$ definiert. Die Genauigkeit liegt bei 12 Stellen.

Single

Im Wertebereich von $1{,}5\times10^{-45}$ bis $3{,}4\times10^{38}$ mit einer Genauigkeit von 8 Stellen liegt der Single-Datentyp.

Double

Double-Variablen können Werte aufnehmen, die im Bereich von 5×10^{-324} bis $1{,}7\times10^{308}$ bei einer Genauigkeit von 16 Stellen liegen.

Extended

Der Extended-Datentyp ist der komplexeste der Real-Typen. Er kann Daten im Bereich von $1{,}9\times10^{-4951}$ bis $1{,}1\times10^{4932}$ mit einer Genauigkeit von bis zu 20 Stellen aufnehmen.

Name	Funktion	Beispielinhalt
Char	einzelne Zeichen	»A«
String	Zeichenketten	»Ein Text«
Boolean	Wahrheitswerte	Falsch
Pointer	Zeiger auf eine Struktur	(A000hex,0000hex)

Tabelle 3. Weitere Standardtypen

Name	Funktion
Array	Datenfeld, das indiziert werden kann
Record	Verbund unterschiedlicher Typen zu einer Einheit

Tabelle 4. Die strukturierten Variablentypen

Comp
Der Comp-Datentyp erreicht mit 19 Stellen eine ähnliche Genauigkeit wie Extended-Variablen, ist aber auf einen Wertebereich von $-9{,}2 \times 10^{18}$ bis $9{,}2 \times 10^{18}$ beschränkt.

Typen für weitere Variablen
Neben den Datentypen für ganze Zahlen und Kommazahlen bietet Turbo Pascal vier weitere Grundtypen an. Sie sind für unterschiedliche Anwendungsgebiete bestimmt. Tabelle 3 faßt die vier Typen zusammen.

Char
Der Char-Datentyp ist für die Aufnahme einzelner Zeichen bestimmt. Er wird häufig bei der Abfrage einzelner Tasten der Tastatur verwendet.

String
Zeichenkettenvariablen werden über den String-Datentyp definiert. Eine Zeichenkette ist eine maximal 255 Zeichen lange Kette aus einzelnen Zeichen.

Boolean
Boolean-Variablen werden für logische Entscheidungen verwendet. Sie können nur einen der beiden Inhalte »true« (für »wahr«) oder »false« (für »falsch«) erhalten.

Pointer
Eine Variable, die über den Pointer-Datentyp definiert wird, ist für die Aufnahme eines Zeigers konzipiert. Ein Zeiger enthält die Adresse einer Variablen, einer Prozedur, einer Funktion oder einer Datenstruktur im Speicher.

Typen für strukturierte Daten
Über die Standardtypen hinaus bietet Turbo Pascal zwei Datentypen an, mit denen sich strukturierte Datentypen neu aufbauen lassen. Was es mit strukturierten Datentypen auf sich hat und wie sie definiert werden, erfahren Sie auf Seite 58. Tabelle 4 zeigt alle Datentypen zum Aufbau strukturierter Typen.

ARRAY[<Start 1>..<Ende 1>[,<Start N>..<Ende N> [,..]]] OF <Typ>;
Über den Array-Datentyp können strukturierte Felder von Variablen des einheitlichen Datentyps definiert werden, der mit dem Parameter <Typ> festgelegt wird. Der Bereich für die verwendbaren Indizes eines Feldes wird mit den Parametern <Start 1> und <Ende 1> angegeben. Wenn das Feld mehr als eine Dimension enthält, können weitere Definitionen für den Index in den Parametern <Start N> und <Ende N> folgen.

RECORD
<Deklaration 1>
..
<Deklaration N>
END;
Mit einer Record-Definition werden unterschiedliche Datentypen zu einer einzigen Einheit zusammengefaßt. Dazu werden zwischen die Schlüsselwörter »record« und »end« mit den Parametern <Deklaration 1> bis <Deklaration N> einzelne Deklarationen von Variablen vorgenommen.

(Martin Althaus/ks)

Turbo Pascal und die Dateiverwaltung

Die Verwaltung von auf externen Datenträgern abgelegten Daten gehört zu den wichtigsten Aufgaben des Betriebssystems und damit auch von Turbo Pascal. Einige Prozeduren der Programmiersprache beschäftigen sich ausschließlich mit diesem Themenkomplex und erleichtern daher die Arbeit mit dem PC wesentlich.

Im großen Funktionsumfang des Turbo-Pascal-Compilers sind viele Befehle enthalten, die sich speziell mit der Verwaltung von Daten und Dateien befassen. Die folgende Kurzreferenz gibt Ihnen einen Überblick über diese wichtigen Funktionen.

DiskFree (<Laufwerk>)
»DiskFree« liefert die Größe des freien Speicherplatzes auf einem Laufwerk in Byte. Für den Laufwerksparameter sind die Werte

```
0, 1, 2, 3, ...
```

erlaubt. Dabei steht »0« für das aktuelle Laufwerk, »1« für Laufwerk »A«, »2« für »B« und so weiter. Falls sich die Laufwerksbezeichnung außerhalb des gültigen Wertebereichs befindet oder ungültig ist, gibt die Funktion »DiskFree« das Ergebnis »-1« zurück.

DiskSize (<Laufwerk>)
Im Gegensatz zu »DiskFree« liefert »DiskSize« die Größe des gesamten Speicherplatzes auf einem Laufwerk in Byte zurück. Der Laufwerksparameter entspricht dem der Funktion »DiskFree«.
Neben den Funktionen zur Ermittlung des Speicherplatzes auf externen Datenträgern gibt es auch Unterprogramme zur Bearbeitung der Inhaltsverzeichnisse.

FindFirst (<Pfad>,<Attribut>,<S>)
»FindFirst« sucht ein Inhaltsverzeichnis nach dem ersten Vorkommen eines Dateinamens ab. Dabei gibt »Pfad« den zu suchenden Dateinamen an, der auch aus Wildcards bestehen kann (beispielsweise »*.pas«). »Pfad« kann zusätzlich den Suchpfad und/oder die Laufwerksbezeichnung enthalten. Wenn diese Angaben fehlen, benutzt das Kommando das aktuelle Unterverzeichnis. »Attribut« bestimmt die Attribute der zu suchenden Datei, deren gültige Werte in Tabelle 1 zusammengefaßt sind. Der Parameter »S« ist ein Datensatz des Typs »SearchRec«, dessen Struktur in Listing 1 abgebildet ist. »S« muß vor dem Aufruf des Kommandos nicht initialisiert werden und beinhaltet nach der Befehlsausführung alle wichtigen Informationen (Länge, Datum und so weiter) zu der gefundenen Datei. Trifft »FindFirst« auf einen Fehler beim

Dateiattribute	
Konstante	Wert
AnyFile	3Fhex
Archive	20hex
Directory	10hex
Hidden	02hex
ReadOnly	01hex
SysFile	04hex
VolumeID	08hex

Tabelle 1. Die in der Unit »Dos« vordefinierten Dateiattribute bestimmen den Typ einer Datei

Die Turbo-Pascal-Variable »DosError«	
Inhalt	**Bedeutung**
02hex	Datei nicht gefunden
03hex	Suchpfad nicht gefunden
05hex	Zugriff nicht erlaubt
06hex	Datei-Handle nicht definiert oder ungültig
08hex	nicht ausreichend Arbeitsspeicher vorhanden
0Ahex	Environment-Variable ungültig
0Bhex	ungültiges Befehlsformat
12hex	keine weiteren Verzeichniseinträge

Tabelle 2. Die Variable »DosError« ist für die Befehle »FindFirst« und »FindNext« von entscheidender Bedeutung

Beispiele für »FExpand«	
Funktion	**Ergebnis**
FExpand('..*.exe')	'C:\WORD*.EXE'
FExpand('c:\turbo\doktor.pas')	'C:\TURBO\DOKTOR.PAS'
FExpand('test.txt')	'C:\WORD\DOKUMENTE\TEST.TXT'
FExpand('wOrD.?X?')	'C:\WORD\DOKUMENTE\WORD.?X?'

Tabelle 3. »FExpand« vervollständigt einen fragmentarischen Dateinamen mit den aktuellen Verzeichnisangaben

Ablauf der Routine, belegt es die global definierte Variable »DosError« mit dem entsprechenden Fehlercode. Die gültigen Fehlercodes sind in Tabelle 2 zusammengefaßt.

FindNext (<S>)
»FindNext« sucht im Gegensatz zu »FindFirst« ein Inhaltsverzeichnis nach dem nächsten Vorkommen eines Dateinamens ab. Dabei benutzt »FindNext« den mit »FindFirst« definierten Dateinamen. Der Parameter »S« ist auch in diesem Fall ein Datensatz des Typs »SearchRec«. »S« muß vor dem Aufruf des Kommandos nicht initialisiert werden und beinhaltet nach der Befehlsausführung alle wichtigen Informationen zu der gefundenen Datei. Trifft »FindNext« auf einen Fehler beim Ablauf des Unterprogramms, belegt es ebenfalls die global definierte Variable »DosError«. Von Interesse ist dann allerdings nur die Fehlermeldung »keine weiteren Verzeichniseinträge«, die darauf hinweist, daß das Ende der Dateieintragsliste erreicht ist.

FSearch (<Pfad>; <VzListe>)
Die im Zusammenhang mit den beiden vorgenannten Befehlen interessante Funktion »FSearch« sucht in den mit »VzListe« angegebenen Unterverzeichnissen nach der durch durch »Pfad« spezifizierten Datei. Falls der Dateiname nicht gefunden wird, gibt die Funktion eine leere Zeichenkette als Parameter zurück, ansonsten enthält die übergebene Zeichenkette das Verzeichnis, in dem die Suche erfolgreich verlaufen ist. »FSearch« sucht zuerst in dem aktuellen Inhaltsverzeichnis und danach in den in »VzListe« aufgezählten Verzeichnissen. Die einzelnen Einträge müssen durch Semikolons getrennt werden. Ein Anwendungsbeispiel finden Sie in Listing 2.

GetFAttr (<DateiV>,<Attribut>)
»GetFAttr« übergibt die in Tabelle 1 zusammengefaßten Attribute einer Datei. Die Datei, die nicht bereits für Lese- oder Schreibzugriffe geöffnet sein darf, definieren Sie über die Dateivariable »DateiV«. Turbo Pascal setzt die Variable »DosError«, wenn ein Fehler beim Ermitteln der Dateiattribute aufgetreten ist. Als Demonstration dient das Programm, das in Listing 3 abgebildet ist.

SetFAttr (<DateiV>,<Attribut>)
»SetFAttr« setzt im Gegensatz zu »GetFAttr« die in Tabelle 1 zusammengefaßten Attribute einer Datei. Den Dateinamen legen Sie über die Dateivariable »DateiV« fest. Auch in dieser Situation setzt Turbo Pascal die Variable »DosError«, wenn ein Fehler beim Ermitteln der Dateiattribute aufgetreten ist.

FExpand (<Pfad>)
Die Funktion »FExpand« komplettiert den in »Pfad« übergebenen unvollständigen Dateinamen. Das heißt, sie wandelt alle Klein- in Großbuchstaben um und fügt sowohl den gewünschten Verzeichnisnamen als auch die Laufwerksbezeichnung hinzu. Als Ergebnis liefert die Funktion eine Zeichenkette mit einer maximalen Länge von 79 Zeichen. Einige Anwendungsmuster, die den aktuellen Pfad »c:\word\dokumente« voraussetzen, finden Sie in Tabelle 3.

FSplit (<Pfad>;<Verzeichnis>; <Dateiname>;<Suffix>)
Die Funktion »FSplit« zerlegt den in »Pfad« übergebenen Dateinamen in seine einzelnen Bestandteile Pfad, Name und Suffix. Als Ergebnis übergibt die Funktion die einzelnen Komponenten. Die Definition der Parameter finden Sie in Listing 4. Sollte der vollständige Dateiname eine oder mehrere Komponenten nicht enthalten, legt Turbo Pascal in den Variablen leere Zeichenketten ab. »FSplit« teilt beispielsweise die Zeichenkette

```
c:\word\dokumente\test.txt
```

in das Verzeichnis

```
c:\word\dokumente\
```

den Dateinamen

```
test.txt
```

und das Suffix

```
.txt
```

auf. Die folgenden Kommandos des Turbo-Pascal-Befehlssatzes benutzt man für die standardisierte Ausgabe von Dateien und das Einlesen von Daten. Auf ihre Bedeutung gehen die Erläuterungen nur kurz ein.

Append (<DateiV>)
öffnet die durch »DateiV« angegebene Datei und hängt die durch nachfolgende Schreiboperationen übertragenen Daten an ihr Ende an.

Assign (<DateiV>,<Dateiname>)
ordnet der Dateivariablen »DateiV« den spezifizierten Dateinamen zu.

BlockRead (<DateiV>,<Puffer>, <Anzahl>)
Der Befehl »BlockRead« liest aus der durch »DateiV« spezifizierten Datei »Anzahl« Datensätze in den Dateipuffer »Puffer«.

BlockWrite (<DateiV>,<Puffer>, <Anzahl>)
Das Kommando «BlockWrite« schreibt im Gegensatz zu »BlockRead« »Anzahl« Datensätze aus dem Dateipuffer »Puffer« in die durch »DateiV« spezifizierte Datei.

ChDir (<Verzeichnis>)
wechselt vom aktuellen Verzeichnis in das durch »Verzeichnis« angegebene.

Close (<DateiV>)
schließt die durch »DateiV« festgelegte Datei.

EOF (<DateiV>)
prüft, ob das Ende der durch »DateiV« spezifizierten Datei erreicht ist. Das Ergebnis ist vom Typ Boolean, das heißt, es liefert den Wert »true« (wahr) oder »false« (falsch).

EOLn (<DateiV>)
prüft nach, ob sich der Zeiger in der durch »DateiV« spezifizierten Datei auf einem Zeilenende befindet. Das Ergebnis ist ebenfalls vom Typ Boolean, das heißt, es liefert den Wert »true« (wahr) oder »false« (falsch).

Erase (<DateiV>)
löscht die angegebene Datei vom externen Datenträger.

FilePos (<DateiV>)
liefert die Position des Zeigers in der an die Routine übergebenen Datei »DateiV«.

Show-Manager

selbstablaufende Demonstrationen mit Ihrem PC

- **Präsentation**
- **Vorführung**
- **Schulung**
- **Werbung**

Bisher nur für Profis erschwinglich, können jetzt auch Sie mit dem DMV-Show-Manager die Überzeugungskraft selbstablaufender Computer-Demonstrationen nutzen:

- Grafikbildschirme von Anwendungen »fotografieren« Sie mit dem beigefügten Pickup-Programm
- Farbige Textbildschirme erstellen Sie komfortabel mit dem integrierten Maskeneditor
- Externe Programme rufen Sie von der Präsentation aus auf und kehren danach automatisch zurück
- Alle Elemente Ihrer Präsentation stellen Sie in einer leichtverständichen Tabellenprogrammierung zusammen
- Verzweigungen, Rücksprünge und Benutzereingaben steuern Sie durch eine BASIC-ähnliche Makrosprache
- Die in eine einzige Datei komprimierte Präsentation nutzen Sie mit dem beigefügten Vorführprogramm
- Mit dem Trainer, der Online-Hilfe und dem umfangreichen Handbuch lernen Sie Show-Manger in kürzester Zeit
- Durch den günstigen Sparpreis amortisieren Sie Show-Manager schon durch einen einzigen Einsatz

Die Anwendungsmöglichkeiten von DMV-Show-Manager sind praktisch unbegrenzt und lassen Ihrer Kreativität freien Raum:

- Untermauern Sie Ihre Vorträge und Präsentationen mit Business-Grafiken, Tabellen etc.
- Führen Sie Softwareprodukte anschaulich vor, und geben Sie selbstablaufende Demonstrationen heraus
- Optimieren Sie Schulungen, Einarbeitungen oder Bedienungsanleitungen durch Bildelemente
- Ihre Werbung wird effektiver durch selbstablaufende, farbige Bildfolgen und Werbeaussagen
- Machen Sie Ihren Freunden eine unvergessene Überraschung mit einer sensationellen Computer-Show

DMV-Show-Manager
selbstablaufende Demonstrationen mit dem PC — **DM 199.-***

dazu passend empfehlen wir:

ConText PRO
Schnelle Textverarbeitung mit Blockgrafikeditor — **DM 199.-***

* Unverbindliche Preisempfehlung. Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,- DM bzw. für das Ausland 6,- DM Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

DMV-Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

Flush (<DateiV>)
schreibt den Inhalt des Dateipuffers auf den externen Datenträger.

GetDir (<Laufwerk>,<Pfad>)
ermittelt den dem Laufwerk »Laufwerk« zugewiesenen aktuellen Verzeichnisnamen »Pfad«.

MkDir (<Pfad>)
erzeugt das in »Pfad« übergebene Unterverzeichnis.

Read (<DateiV>, <Variablenliste>)
liest die in der Variablenliste spezifizierten Daten aus der in »DateiV« festgelegten Datei.

ReadLn (<DateiV>, <Variablenliste>)
liest die in der Variablenliste spezifizierten Daten aus der in »DateiV« festgelegten Datei bis zum nächsten Zeilenende.

Rename (<DateiV>, <NeuerName>)
benennt die in der DateiVariablen »DateiV« festgehaltene Datei in die neue Datei »NeuerName« um.

Reset (<DateiV>)
öffnet eine bereits existierende Datei, die den in »DateiV« enthaltenen Dateinamen trägt, für Schreib- und Leseoperationen.

ReWrite (<DateiV>)
erzeugt eine Datei mit dem Dateinamen »DateiV« und öffnet sie für Schreib- und Leseoperationen.

RmDir (<Pfad>)
löscht das in »Pfad« gespeicherte Unterverzeichnis vom externen Datenträger.

Seek (<DateiV>,<Position>)
setzt den Positionszeiger in der Datei »DateiV« auf die in »Position« gespeicherte Stelle.

Truncate (<DateiV>)
schneidet die Datei »DateiV« an der aktuellen Zeigerposition ab und löscht alle Daten zwischen Zeigerposition und Dateiende.

Write (<DateiV>, <Variablenliste>)
schreibt die in der Variablenliste spezifizierten Daten in die mit »DateiV« bezeichnete Datei.

WriteLn (<DateiV>, <Variablenliste>)
schreibt die in der Variablenliste spezifizierten Daten in die mit »DateiV« bezeichnete Datei und fügt ein Zeilenende ein.

(Markus Zietlow/dm)

```
[STØ]    1: type
[EZ2]    2:   SearchRec = record
[184]    3:                 Fill: array[1..21] of Byte;
[SX1]    4:                 Attr: Byte;
[BC2]    5:                 Time: Longint;
[XD2]    6:                 Size: Longint;
[ET2]    7:                 Name: string[12];
[ZNØ]    8:               end;
```

Listing 1. Für die Verzeichnisroutinen erwartet Turbo Pascal einen Parameter des Typs »SearchRec«

```
[AR3]    1: program FSearch_Demo;
[ØØØ]    2:
[ASØ]    3: uses
[9G1]    4:   crt, dos;
[ØØØ]    5:
[RNØ]    6: var
[3A5]    7:   Dateiliste, Dateiname : string;
[E12]    8:   S : PathStr;
[ØØØ]    9:
[MUØ]   1Ø: begin
[DB1]   11:   ClrScr;
[EZ5]   12:   write('Geben Sie bitte den Namen der '+
[5G4]   13:         'zu suchenden Datei ein: ');
[RY2]   14:   readln(Dateiname);
[746]   15:   write('Geben Sie bitte die Posten der '+
[YB4]   16:         'Dateiliste ein (durch ";" '+
[4B2]   17:         'getrennt): ');
[Ø83]   18:   readln(Dateiliste);
[XF1]   19:   writeln;
[XF5]   2Ø:   S:=FSearch(Dateiname,Dateiliste);
[Z72]   21:   S:=FExpand(S);
[4V5]   22:   writeln('Die angegebene Datei ist '+
[DF4]   23:           'unter dem folgenden Namen '+
[HK2]   24:           'aufzufinden:');
[XF1]   25:   writeln;
[RZ1]   26:   writeln(S);
[XF1]   27:   writeln;
[XF1]   28:   writeln;
[4L6]   29:   writeln('Drücken Sie bitte irgendeine '+
[EL1]   3Ø:           'Taste !');
[Y44]   31:   repeat until KeyPressed;
[KNØ]   32: end.
```

Listing 2. Die Funktion »FSearch« dient dazu, eine Verzeichnisliste nach einem bestimmten Dateinamen zu durchsuchen

```
[114]    1: program GetFAttr_Demo;
[ØØØ]    2:
[ASØ]    3: uses
[9G1]    4:   crt, dos;
[ØØØ]    5:
[RNØ]    6: var
[VD2]    7:   Datei     : text;
[F93]    8:   Dateiname : string;
[5V2]    9:   Attribut  : word;
[ØØØ]   1Ø:
[MUØ]   11: begin
[DB1]   12:   ClrScr;
[EZ5]   13:   write('Geben Sie bitte den Namen der '+
[BD5]   14:         'zu untersuchenden Datei ein: ');
[RY2]   15:   readln(Dateiname);
[XF1]   16:   writeln;
[9Z3]   17:   assign(Datei,Dateiname);
[4C4]   18:   GetFAttr(Datei,Attribut);
[RF6]   19:   writeln('Die spezifizierte Datei hat '+
[GV3]   2Ø:           'folgende Attribute:');
[XF1]   21:   writeln;
[H53]   22:   if DosError<>Ø then
[MUØ]   23:     begin
[P42]   24:       write(chr(7));
[UA4]   25:       writeln('Fehler bei der '+
[TN2]   26:               'Dateianalyse:');
[XF1]   27:       writeln;
[KS2]   28:       case DosError of
[T84]   29:         $Ø2 : writeln('Datei nicht '+
[PU1]   3Ø:               'gefunden');
[LT4]   31:         $Ø3 : writeln('Suchpfad nicht '+
[PU1]   32:               'gefunden');
[KN4]   33:         $Ø5 : writeln('Zugriff nicht '+
[3U1]   34:               'erlaubt');
[2G4]   35:         $Ø6 : writeln('Datei-Handle '+
[VØ3]   36:               'nicht definiert '+
[SX2]   37:               'oder ungültig');
[NE5]   38:         $Ø8 : writeln('Nicht ausreichend '+
[V53]   39:               'Arbeitsspeicher '+
[W22]   4Ø:               'vorhanden');
[ZH4]   41:         $ØA : writeln('Environment-'+
[XK3]   42:               'Variable ungültig');
[BK4]   43:         $ØB : writeln('Ungültiges '+
[US2]   44:               'Befehlsformat');
[TV4]   45:         $12 : writeln('Keine weiteren '+
[ØB2]   46:               'Verzeichnis'+
[A42]   47:               'einträge');
[ZRØ]   48:       else
[L55]   49:         writeln('Unbekannter Fehler');
[ZNØ]   5Ø:       end;
[4MØ]   51:     end
[ZRØ]   52:   else
[MUØ]   53:     begin
[MC5]   54:       if Attribut and Hidden   <> Ø then
[6L3]   55:         writeln('Versteckt');
[EN5]   56:       if Attribut and SysFile  <> Ø then
[734]   57:         writeln('Systemdatei');
[GN5]   58:       if Attribut and VolumeID <> Ø then
[DH3]   59:         writeln('Volume-ID');
[LT5]   6Ø:       if Attribut and Directory<> Ø then
[G14]   61:         writeln('Verzeichnis');
[NK5]   62:       if Attribut and Archive  <> Ø then
[L33]   63:         writeln('Archiv');
[ZNØ]   64:     end;
[XF1]   65:   writeln;
[XF1]   66:   writeln;
[4L6]   67:   writeln('Drücken Sie bitte irgendeine '+
[EL1]   68:           'Taste !');
[Y44]   69:   repeat until KeyPressed;
[KNØ]   7Ø: end.
```

Listing 3. »GetFAttr« eignet sich für die Analyse der Organisation einzelner Verzeichniseinträge

```
[STØ]    1: type
[C71]    2:   (* Pfad *)
[FM3]    3:   PathStr = string[79];
[1H2]    4:   (* Verzeichnis *)
[VD3]    5:   DirStr  = string[67];
[752]    6:   (* Dateiname *)
[7L3]    7:   NameStr = string[8];
[ZK1]    8:   (* Suffix *)
[CF3]    9:   ExtStr  = string[4];
```

Listing 4. Die Unit »Dos« definiert für »FSplit« vier verschiedene Datentypen

Durch und durch strukturiert

Kontrollstrukturen zählen zu den unverzichtbaren Komponenten der Programmiersprache Pascal. Erst sie schaffen die Voraussetzungen für unkomplizierte und themenorientierte Routinen. Einige Kommandos des von Wirth definierten Befehlssatzes befassen sich ausnahmslos mit diesem Bereich und vereinfachen auf diese Weise die Ausarbeitung von sinnvollen Problemlösungen.

CASE <Selektor> OF <Vergleichsliste> ELSE <Anweisung>; END

Der Case-Befehl besteht aus der Vergleichsvariablen »Selektor« und einer Liste von Anweisungen, die ausgeführt werden, wenn die einleitenden Argumente mit dem Selektor übereinstimmen.

Der Inhalt des Selektors muß eine ganze Zahl sein, die sich im Wertebereich von -32768 bis 32767 befindet. Sollte keiner der in der Vergleichsliste angegebenen Parameter mit dem Inhalt des Selektors übereinstimmen, führt Turbo Pascal den Else-Zweig des Blocks aus. Wenn kein Else-Zweig definiert ist, fährt der Computer mit der Ausführung des Befehls fort, der der Case-Anweisung folgt. Um die Funktionsweise besser zu verstehen, finden Sie in Listing 1 ein Anwendungsbeispiel, das den Programmablauf verdeutlicht.

IF <Bedingung> THEN <Anweisung1> ELSE <Anweisung2>;
Das If-Kommando gehört zu den grundlegenden Sprachelementen von Pascal. Wenn das Kriterium »Bedingung« erfüllt ist, führt das Programm die Direktive »Anweisung1« aus, andernfalls bearbeitet es die Vorschrift »Anweisung2«. Die Bedingung muß immer den Ergebniswert »boolean« aufweisen, das heißt, sie muß wahr oder falsch sein. Es ist erlaubt, mehrere If-Anweisungen ineinander zu verschachteln. Ein Praxisbeispiel ist in Listing 2 abgebildet.

FOR <Variable>:= <Anfangswert> TO <Endwert> DO <Anweisung>;
Der For-Befehl führt eine einfache oder eine Verbundanweisung »Anweisung« in einer Schleife aus, während er der Laufvariablen »Variable« aufeinanderfolgende Werte zuweist. Wenn Sie statt des Schlüsselwortes »To« die Variation »DownTo« verwenden, zählt das Programm abwärts vom »Anfangswert« zum »Endwert« anstatt – wie normalerweise – aufwärts. Listing 3 zeigt ein Anwendungsbeispiel.

REPEAT <Anweisung> UNTIL <Bedingung>;
Dieses Kommando führt den Befehlsblock »Anweisung« so lange aus, bis die angegebene »Bedingung« erfüllt ist. Der Befehlsblock wird mindestens einmal ausgeführt, das heißt, die Abbruchbedingung wird erst nach der Anweisung zum ersten Mal überprüft. Listing 4 demonstriert die korrekte Ausführung des Befehls.

WHILE <Bedingung> DO <Anweisung>;
Dieses Kommando führt den Befehlsblock »Anweisung« so lange aus, bis die angegebene »Bedingung« erfüllt ist. Das Programm führt den Befehlsblock im Gegensatz zu »Repeat« nicht unbedingt aus, das heißt, die Abbruchbedingung wird schon vor dem ersten Schleifendurchlauf überprüft. Listing 5 zeigt die exakte Struktur der Instruktion.

(Markus Zietlow/ks)

```
[Z12]    1: case Taste of
[VW2]    2:   Esc          : Abbruch;
[SC4]    3:   Return       : Zeilenumbruch;
[WV3]    4:   PfeilHoch    : Aufwaerts;
[W24]    5:   PfeilRunter  : Abwaerts;
[X93]    6:   PfeilLinks   : Links;
[4M3]    7:   PfeilRechts  : Rechts;
[ZRØ]    8: else
[9H2]    9:   Tastenanalyse;
[ZNØ]   1Ø: end;
```

Listing 1. Der Case-Befehl akzeptiert nur Ordinal-Typen als Selektoren

```
[ØZ2]    1: program if_demo;
[ØØØ]    2:
[RNØ]    3: var
[T72]    4:   zahl:integer;
[ØØØ]    5:
[MUØ]    6: begin
[Ø32]    7:   readln(zahl);
[1B2]    8:   if zahl<1Ø then
[UB5]    9:     writeln('Zahl ist kleiner als 1Ø')
[ZRØ]   1Ø:   else
[SD2]   11:     if zahl<1ØØ then
[AH5]   12:       writeln('Zahl ist kleiner als 1ØØ')
[ZRØ]   13:     else
[3H2]   14:       if zahl<1ØØØ then
[445]   15:         writeln('Zahl ist kleiner als '+
[AUØ]   16:                 '1ØØØ')
[ZRØ]   17:       else
[ZR5]   18:         writeln('Zahl ist größer oder '+
[A62]   19:                 'gleich 1ØØØ');
[KNØ]   2Ø: end.
```

Listing 2. Das If-Kommando erleichtert die Einteilung eines Programms in bedingungsabhängige Abschnitte

```
[423]    1: program for_demo;
[ØØØ]    2:
[RNØ]    3: var
[DX2]    4:   zaehler:integer;
[8K2]    5:   zahl   :longint;
[ØØØ]    6:
[MUØ]    7: begin
[LK6]    8:   write('Geben Sie bitte eine natürliche '+
[HZ1]    9:         'Zahl ein: ');
[Ø32]   1Ø:   readln(zahl);
[XF1]   11:   writeln;
[WR6]   12:   writeln('Die eingegebene Zahl ist durch'+
[L95]   13:           ' die folgenden Zahlen zwischen'+
[DA3]   14:           ' 1 und 1ØØØ teilbar:');
```

```
[ØN1]   15:   write('1');
[5P3]   16:   for zaehler:=2 to 1ØØØ do
[UE6]   17:     if zahl/zaehler=int(zahl/zaehler) then
[1Y2]   18:       write(', ',zaehler);
[XF1]   19:   writeln;
[KNØ]   2Ø: end.
```

Listing 3. Mit einer For-Schleife lassen sich die Teiler einer natürlichen Zahl leicht ermitteln

```
[P31]    1: repeat
[7A1]    2:   inc(x,1);
[491]    3:   dec(y,1);
[B54]    4:   ergebnis1:=ergebnis1+x+y;
[P54]    5:   ergebnis2:=ergebnis2+x-y;
[N54]    6:   ergebnis3:=ergebnis3-x+y;
[254]    7:   ergebnis4:=ergebnis4-x-y,
[VW4]    8:   gesamt1:=ergebnis1/ergebnis2;
[7X4]    9:   gesamt2:=ergebnis3/ergebnis4;
[6T1]   1Ø: until x>y;
```

Listing 4. »Repeat Until« wiederholt einen Pascal-Block, bis das Abbruchkriterium erfüllt ist

```
[CY3]    1: program DateiEinlesen;
[ØØØ]    2:
[RNØ]    3: var
[7B1]    4:   f:text;
[U52]    5:   name:string;
[TS2]    6:   anzahl:integer;
[ZA2]    7:   liste:string;
[ØØØ]    8:
[MUØ]    9: begin
[W56]   1Ø:   write('Geben Sie bitte den Dateinamen '+
[Y41]   11:         'ein: ');
[722]   12:   readln(name);
[XF1]   13:   writeln;
[KF2]   14:   assign(f,name);
[VE1]   15:   reset(f);
[ZN1]   16:   anzahl:=Ø;
[Z23]   17:   while not eof(f) do
[MUØ]   18:     begin
[D94]   19:       readln(f,liste[anzahl+1]);
[X82]   2Ø:       inc(anzahl,1);
[ZNØ]   21:     end;
[ZY5]   22:   writeln('Die Datei enthält ',anzahl,
[U12]   23:           ' Elemente.');
[KD1]   24:   close(f);
[KNØ]   25: end.
```

Listing 5. Dieses Beispiel zählt mit Hilfe einer While-Schleife die Anzahl der Zeilen innerhalb einer Datei

Die Grundlagenartikel führen systematisch in die Programmierung von Turbo Pascal mit jeweils einer bestimmten Themenstellung ein. Dabei steht die Darstellung der angewandten Programmierung einschließlich der Hinweis auf mögliche Fehlerquellen und »Fußangeln« im Vordergrund. Sie erhalten einen Überblick über den Befehlsumfang von Turbo Pascal in bezug auf ein bestimmtes Aufgabengebiet. Die genaue Handhabung der beschriebenen Befehle inklusive der notwendigen Parameterübergabe wird in den Referenzartikeln erläutert.

Die erste wichtige Feststellung, die in bezug auf Pascal-Programme getroffen werden kann, ist die, daß Programm und Daten streng getrennt sind. »Daten« sind in einem Pascal-Programm alle Strukturen, in denen Werte irgendeiner Form gespeichert werden können. »Programm« ist die Summe der Routinen, mit denen die Daten verarbeitet werden.
Alle Daten werden in Pascal in Variablen gespeichert. Eine Variable kann man sich gut als einen leeren Karton vorstellen, in den der Computer bei Bedarf einen bestimmten Inhalt hineinpackt. Um den Inhalt der Variablen im Computerspeicher zu finden, erhält jeder Karton einen Namen, unter dem er angesprochen wird. Jede verwendete Variable muß vor ihrer Benutzung in einem Programm deklariert werden. Dazu stellt Turbo Pascal Datentypen zur Verfügung. Je nachdem, welchen Inhalt eine Variable aufnehmen soll, müssen Sie sie mit einem unterschiedlichen Datentyp definieren. Für die Definition von Variablen, die Zeichenketten enthalten, wird beispielsweise der Datentyp »string« verwendet. Für die Definition von ganzzahligen Variablen benötigen Sie den Datentyp »integer«. Die Deklaration der Variablen geschieht in einem eigenen Teil des Pascal-Programms, dem sogenannten Deklarationsteil. Der Deklarationsteil wird durch das Schlüsselwort »var« eingeleitet. Diesem Ausdruck folgen die Namen der Variablen und die dazugehörigen Datentypen. Möchten Sie beispielsweise eine Zeichenketten- oder String-Variable mit dem Namen »zkette« und eine Ganzzahlvariable mit dem Namen »gzahl« deklarieren, erledigt das die folgendende Befehlssequenz:

```
var
zkette: string;
zahl: integer;
```

Anhand der drei Programmzeilen können Sie gleich eine wichtige Eigenschaft von Pascal erkennen. Jede einzelne Anweisung in einem Pascal-Programm wird mit einem Semikolon abgeschlossen. Ausnahmen bilden lediglich einige reservierte Wörter (wie »var«), die für sich alleine keine Anweisungen darstellen.
Anstelle der Namen »gzahl« und »zkette« können die Variablen auch andere Namen erhalten. Variablennamen können Sie fast beliebig vergeben. Lediglich bereits reservierte Schlüsselwörter (wie zum Beispiel »var«) dürfen nicht verwendet werden.
Vor dem Deklarationsteil, ganz zu Beginn des Programms, steht in einem Pascal-Programm der Programmkopf. Das ist eine Zeile, in der hinter dem Schlüsselwort »program« der Name des Programms vergeben wird. Der Name dient dem Programmierer dazu, verschiedene Programme zu unterscheiden. Möchten Sie ein Programm beispielsweise »hallo« nennen, wird der Name mit der folgenden Zeile festgelegt:

```
program hallo;
```

Im Deklarationsteil werden Variablentypen definiert

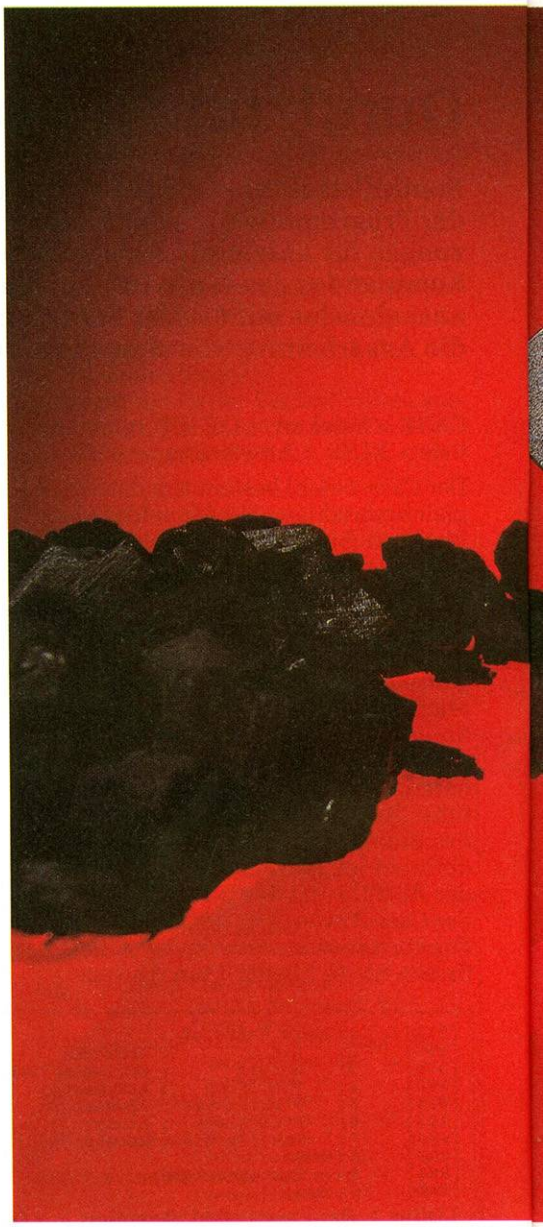
Nach dem Programmkopf wird der Deklarationsteil definiert, der bereits weiter oben behandelt wurde. Nach dem Deklarationsteil folgt der eigentliche Rumpf des Programms, also das Programm, das auf die definierten Datenstrukturen zugreift. Dieser Teil eines Pascal-Programms wird auch Hauptprogramm genannt. Eingeleitet wird der Programmrumpf durch das Schlüsselwort »begin« und abgeschlossen mit »end«. Zwischen diesen werden die Anweisungen eines Pascal-Programms notiert.
Es gibt eine Vielzahl von Befehlen, die für die Verwendung in einem Pascal-Programm vordefiniert sind. Eines der wichtigsten Kommandos ist der Writeln-Befehl. Er erlaubt die Ausgabe von Texten und anderen Daten auf dem Bildschirm. Soll beispielsweise der Text »Hallo, Welt!« auf dem Bildschirm ausgegeben werden, erledigt das die Anweisung:

```
writeln('Hallo, Welt!');
```

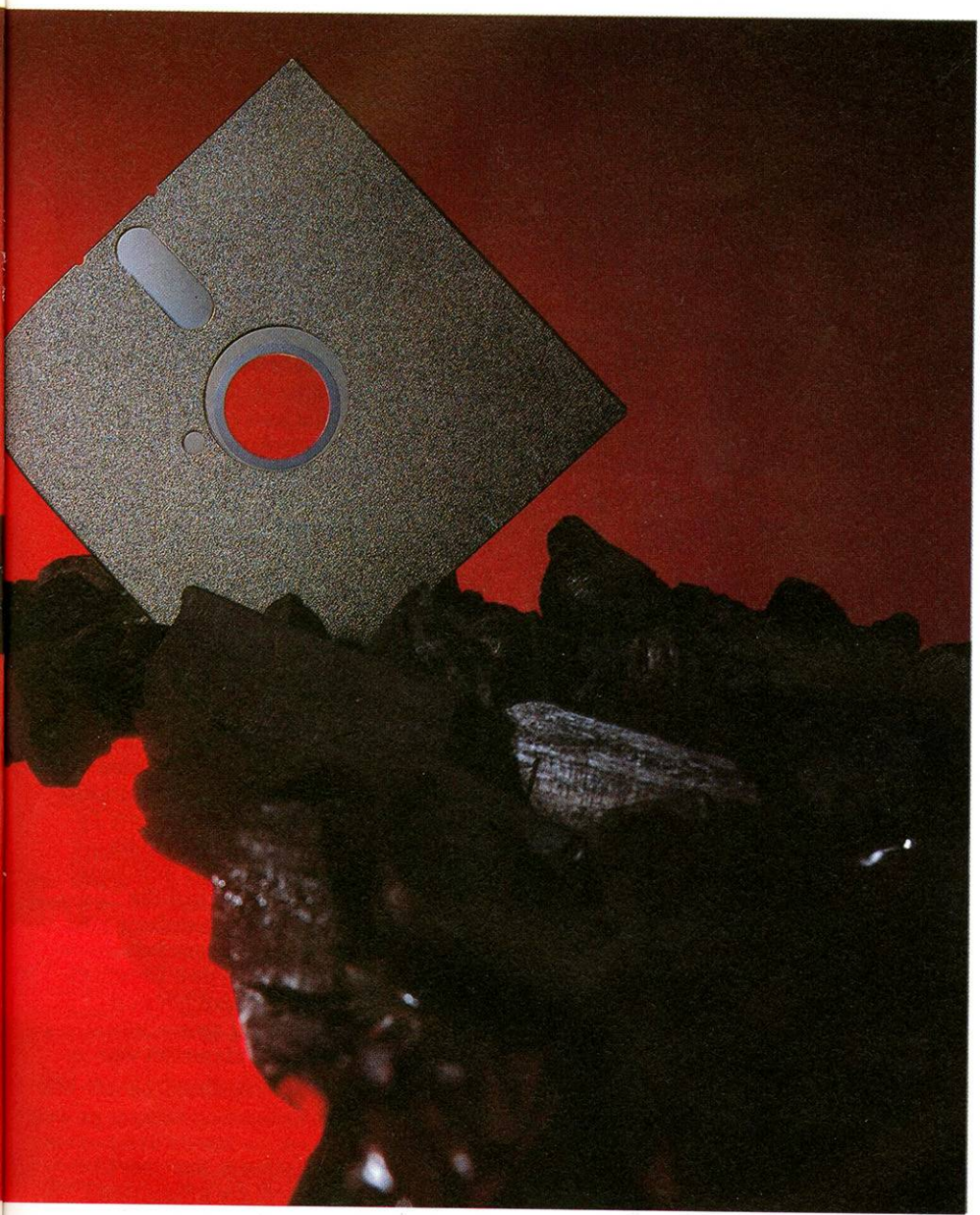
Wenn der Writeln-Befehl das einzige Kommando im Pascal-Programm ist, sieht der resultierende Programmrumpf wie folgt aus:

```
begin
writeln('Hallo, Welt!');
end.
```

Neben dem Writeln-Befehl stellt Ihnen Pascal eine Vielzahl von anderen Befehlen zur Verarbeitung von Daten zur Verfügung. Oft kann es sein, daß



Sie bestimmte Teile eines Programms an mehreren unterschiedlichen Stellen aufrufen möchten. Um zu verhindern, daß Sie diese Programmteile mehrmals schreiben müssen, bietet Pascal die Möglichkeit, Programmteile als Unterprogramme zu deklarieren. Ein Unterprogramm ist ein Programmteil, der an einer Stelle im Programm definiert ist und von mehreren Positionen im Programm aus aufgerufen werden kann. In Pascal heißen die Unterprogramme Prozeduren und Funktionen. Jede Prozedur oder Funktion muß in einem Programm gesondert vor dem Hauptprogramm definiert werden. Der Unterschied zwischen einer Prozedur und einer Funktion liegt darin, daß Funktionen Werte direkt zurückliefern, während Prozeduren lediglich Blöcke des Programmcodes ausführen.
Jede Prozedur wird durch das Wort »procedure« eingeleitet, eine Funktion durch die Anweisung »function«. Der Programmtext einer Proze-



Wie sieht ein Pascal-Programm aus?

Die Computersprache Pascal wurde von dem Schweizer Informatikprofessor Niklas Wirth in Zürich entwickelt. Ihren Namen gab Wirth seiner neuen Sprache in Anlehnung an den berühmten französischen Mathematiker Blaise Pascal. Welche Grundstruktur ein Pascal-Programm besitzt und was die wesentlichen Sprachelemente von Pascal sind, verrät dieser Artikel.

dur oder Funktion steht zwischen den Schlüsselwörtern »begin« und »end«. Wie die Variablen können Funktionen und Prozeduren symbolische Namen erhalten, unter denen Sie sie aufrufen können. Um die Writeln-Zeile aus dem oben angesprochenen Hauptprogramm in einer eigenen Prozedur mit dem Namen »hallo« unterzubringen, reicht es, wenn Sie die folgenden Zeilen eingeben:

```
procedure hallo;
begin
writeln('Hallo, Welt!');
end;
```

Diese Programmzeilen werden zwischen dem Deklarationsteil und dem Hauptprogramm untergebracht. Vom Hauptprogramm (oder einer anderen Prozedur oder Funktion) aus läßt sich die neue Prozedur »hallo« wie einer der Standardbefehle von Pascal verwenden. Wenn das Hauptprogramm also für die Ausgabe des Textes »Hallo, Welt!« die neue Prozedur »hallo« verwenden soll, müssen Sie dies wie folgt umschreiben:

```
begin
hallo;
end;
```

Im Hauptprogramm wird also direkt auf den symbolisch definierten Namen der Prozedur Bezug genommen. Dadurch »weiß« der Pascal-Compiler, welcher Programmcode an der entsprechenden Stelle im Programm aufzurufen ist.

Funktionen liefern im Gegensatz zu Prozeduren Werte zurück

Im Gegensatz zu einer Prozedur gibt eine Funktion einen bestimmten Wert an das aufrufende Programm zurück. In Ihr Programm soll zum Beispiel eine Funktion eingebunden werden, die zu einer bestimmten Zahl den Wert 5 addiert und das Ergebnis wieder an das aufrufende Programm zurückgibt. Betrachten Sie dazu einmal die folgende Deklaration:

```
function dazu(wert:integer):integer;
begin
dazu := wert + 5;
end;
```

Diese Funktionsdeklaration ist schon etwas komplizierter als die einfache Hallo-Prozedur. Sie wird durch das Schlüsselwort »function« eingeleitet, das dem Pascal-Compiler anzeigt, daß eine Funktionsdeklaration folgt. Unmittelbar nach »function« kommt der Name, den die Funktion erhalten soll. Im obigen Fall ist das »dazu«. Dahinter steht in Klammern der Text »wert:integer«. Das bedeutet, daß der Funktion eine Variable oder ein Wert vom Typ »integer« (also Ganzzahl) übergeben wird, die im Rumpf der Funktion unter dem Namen »wert« verfügbar sein soll.
Einen solchen Namen, der die Übergabe von Werten signalisiert, nennt man Parameter. Hinter den Klammern steht nach einem Doppelpunkt erneut eine Typenbezeichnung: in

diesem Fall »integer«. Sie legt fest, in welchem Datenformat das Ergebnis der Funktion geliefert werden soll. Die obige Kopfzeile bietet Ihnen also die folgenden Informationen: Es wird eine Funktion definiert, die unter dem Namen »dazu« verfügbar sein soll und der ein Integer-Wert mit dem Namen »wert« übergeben wird. Die Funktion liefert ein Ergebnis vom Datentyp »integer«.
Ein Aufruf der Funktion »dazu« in einem Programm könnte wie folgt aussehen:

```
zahl := dazu(5);
```

Eine Funktion wird beim Namen genannt

Im Gegensatz zu einem Prozeduraufruf kann der Aufruf einer Funktion nicht durch schlichtes Nennen des Funktionsnamens erfolgen, weil Pascal dann keine Möglichkeit hat, das Ergebnis der Funktion zu speichern. Im obigen Fall wird der Variablen »zahl« der Wert der Funktion »dazu« übergeben, die mit einem Parameter von 5 aufgerufen wird. Welchen Wert besitzt »zahl« nach diesem Aufruf? Dafür werfen Sie am einfachsten nochmals einen Blick auf die Deklaration der Dazu-Funktion. Ihr wird in Klammern ein Wert übergeben, der innerhalb der Funktion den Namen »zahl« trägt. Alle Stellen, in denen im Rumpf der Funktion »dazu« der Parameter »zahl« vorkommt, ersetzt Turbo Pascal bei der Ausführung mit dem tatsächlichen Wert 5. Die einzige Zeile der Funktion im Anweisungsteil lautet daher:

```
dazu := 5 + 5;
```

Das Ergebnis dieser Operation steht fest: 5 plus 5 ist 10. Der Wert von 10 wird also der Variablen »dazu« zugewiesen. Da es sich bei »dazu« um den Namen der Funktion handelt, gibt die Funktion den Wert 10 als Funktionsergebnis an das aufrufende Programm zurück. Die Variable »zahl« im Hauptprogramm erhält also den Wert 10.
Nach dem gleichen Muster geht Pascal bei der Ausführung jeder Funktion oder Prozedur vor. Die Parameter einer Funktion oder Prozedur (auch Prozeduren dürfen Parameter definieren) werden bei der Ausführung des Programms durch die tatsächlich übergebenen Werte ersetzt.
In Prozeduren und Funktionen können Sie einen eigenen Deklarationsteil mit eigenen Variablen definieren, die nur innerhalb der Prozedur oder Funktion verwendet werden dürfen. Dieser lokale Deklarationsteil hat den gleichen Aufbau wie der globale Deklarationsteil zu Beginn eines Programms.
Die Verwendung von Prozeduren und Funktionen in Pascal hat einen weiteren wichtigen Vorteil: Sie erhöht die Übersichtlichkeit und Lesbarkeit der Programme.
Neben der Definition von Prozeduren und Funktionen, also Programmstrukturen, bietet Turbo Pascal auch einige Möglichkeiten, um Datenstrukturen zu definieren. Dazu kann im Deklarationsteil eines Programms ein Konstantendeklarationsteil enthalten sein. Diesen können Sie durch das Schlüsselwort »const« einleiten und durch symbolische Konstanten definieren, mit denen im Programm gearbeitet werden kann.
Wird beispielsweise ein Programm geschrieben, das an vielen Stellen im Programmcode den Wert 5 verwendet, kann dieser als Konstante (beispielsweise unter dem Namen »fuenf«) deklariert werden. Das erledigen die folgenden Anweisungen:

```
const
fuenf = 5;
```

Der Vorteil dieses Verfahrens liegt in der leichten Anpassungsmöglichkeit von Programmen. Soll anstelle der 5 mit einem Wert von 10 gearbeitet werden, reicht es aus, die Konstante mit den Anweisungen:

```
const
fuenf = 10;
```

auf den neuen Wert von 10 umzudefinieren, und alle Programmteile, die mit der symbolischen Konstante »fuenf« arbeiten, bekommen fortan eine 10 »untergeschoben«.

Die Verwendung von Prozeduren und Funktionen erhöht die Lesbarkeit von Programmen

Für die Definition von neuen oder modifizierten Datentypen kann in der Deklaration eines Programms ein Typendeklarationsteil eingebunden werden. Er wird durch das Schlüsselwort »type« eingeleitet, und Sie können neue Variablentypen definieren oder vorhandene Typen umdefinieren. Um beispielsweise den Datentyp »integer« auch unter dem Namen »gzahl« zur Verfügung zu stellen, wird die folgende Deklaration verwendet:

```
type
gzahl = integer;
```

Das Programm »geruest.pas« im Listing zeigt, wie sich die einzelnen Bestandteile eines Pascal-Programms zu einer Einheit verbinden lassen. Sie können das Listing als Basis für Ihre eigenen Experimente in Pascal verwenden.

(Martin Althaus/ks)

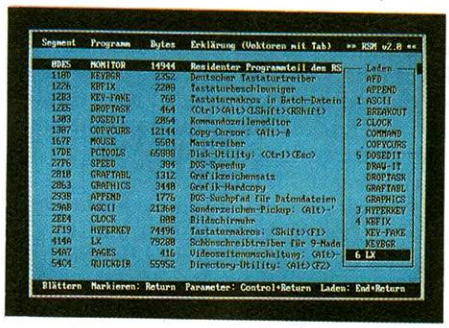
```
[BA2]    1: program welt;
[124]    2: {-----------------------------------
[4D1]    3:                 welt.pas
[4WØ]    4:                 --------
[TY5]    5:  Zeigt die grundsätzliche Struktur
[6L4]    6:  eines Pascal-Programms auf.
[WT3]    7:  von Martin Althaus 4/9Ø
[4Ø4]    8:  (C) 199Ø Redaktionsbüro
[L83]    9:      Everts und Hagedorn
[HR3]   1Ø:  -----------------------------------
[U7Ø]   11: }
[5H1]   12: uses crt;
[TK5]   13:    {Hier werden Units eingebunden.}
[RR5]   14:   {Eine Unit ist eine Sammlung von}
[RB5]   15:         {Prozeduren und Funktionen}
[6XØ]   16: const
[Z91]   17:  fuenf=5;
[D16]   18:   {Der Konstanten-Deklarationsteil}
[7H5]   19:    {sorgt für die Definition aller}
[1C2]   2Ø:                          {Konstanten}
[STØ]   21: type
[PE2]   22:  gzahl=integer;
[N76]   23:  {Im Typen-Deklarationsteil werden}
[1L5]   24:    {neue Datentypen definiert oder}
[ZP5]   25:    {vorhandene Datentypen umdefin-}
[XK1]   26:                                {iert.}
[RNØ]   27: var
[N12]   28:  zahl:gzahl;
```

```
[FX5]   29:     {Im Variablen-Deklarationsteil}
[BØ5]   3Ø:       {werden alle Variablen defi-}
[YG1]   31:                                {niert.}
[FU2]   32: procedure hallo;
[MUØ]   33: begin
[CU3]   34:  writeln('Hallo, Welt!');
[ZNØ]   35: end;
[ØW6]   36: {Prozeduren sind Anweisungsblöcke,}
[XN5]   37:   {die keine Ergebnisse direkt zu-}
[3U3]   38:                  {rückgeben können}
[ØØØ]   39:
[F96]   4Ø: function dazu(zahl:integer):integer;
[MUØ]   41: begin
[752]   42:  dazu:=zahl+5;
[ZNØ]   43: end;
[2W6]   44: {Funktionen sind Anweisungsblöcke,}
[8N6]   45: {die Funktionsergebnisse unmittel-}
[VS4]   46:            {bar zurückgeben können}
[MUØ]   47: begin
[YØ1]   48:  hallo;
[A72]   49:  zahl:=dazu(5);
[KNØ]   5Ø: end.
[8G6]   51: {Das Hauptprogramm muß sich in je-}
[WM5]   52:     {dem Pascal-Programm befinden.}
```

Listing. Das Programm »geruest.pas« demonstriert die grundsätzliche Struktur eines Pascal-Programms

Resident Software Manager

Menügesteuertes Löschen und Nachladen von residenten TSR-Programmen löst Ihre Speicherplatzprobleme:



RSM: zeigt den noch freien Arbeitsspeicher * registriert geladene residente Programme * erinnert an die Aktivierungstastenkombination * informiert über benutzte Interrupt-Vektoren * löscht beliebige, speicherresidente Programme * lädt mehrere TSR-Programme in einem Arbeitsgang * bietet umfassende, komfortable Menüsteuerung * besitzt einen universellen Maustreiber * belegt lediglich 15 KByte Speicherplatz * informiert mit einem umfangreichen Handbuch

Für alle PCs, XTs, ATs und Kompatible

nur DM **69,–** *

GW-Basic-Toolboxen

Professionelle Leistung und komfortable Bedienung: kompatibel zum GW-Basic-Interpreter, gleichzeitiger Gebrauch mehrerer GW-Basic-Toolboxen, leistungsfähige Beispielprogramme inklusive.

Jede GW-Basic-Toolbox nur DM **69,–** *

GW-Basic-Toolbox "Oberfläche"

Für komfortable Benutzeroberflächen mit Pulldown-Menüs und kompletter Mausunterstützung:

* Vollständige Maussteuerung im Text- und Grafikmodus
* Spezielle Menüleisten für häufig benötigte Eingaben
* Popup-Menüs mit Meldungen bis zu fünf Zeilen Länge
* Position, Farben und Rahmen aller Menüs frei wählbar
* Unterstützung aller Bildschirmmodi von GW-Basic
* Inklusive einem fertigen Malprogramm als Beispiel

GW-Basic-Toolbox "DOS-Funktionen"

Mehr als 30 DOS-Funktionen für das MS-DOS-System:

* Kompatibel zu MS-DOS von Version 2.1 bis 4.01
* Laufwerke wechseln, aktuelles Laufwerk ermitteln etc.
* Suchen nach Dateien mit Wildcards
* Beschleunigte Bildschirmausg. über Systemfunktionen
* Zugang zu wichtigen, internen Daten von MS-DOS
* Benutzung des Druckprogramms Print von GW-Basic
* Inklusive Beispielprogramm: Eine komplette DOS-Shell

GW-Basic-Toolbox "Hercules-Grafik"

Programmieren der Hercules-Karte im Grafikmodus:

* Routinen zum Ein- und Ausschalten des Grafikmodus
* Zwei verfügbare Grafikseiten mit Umschaltmöglichkeit
* Funktionen für Linie, Bogen, Kreis, Ellipse und Vieleck
* Ausfüllen von Flächen mit beliebigen Punktmustern
* Speichern und Restaurieren des Grafikbildschirms
* Setzen, Löschen und Abfragen einzelner Bildpunkte
* Emulation der GW-Basic-Befehle "View" und "Window"
* Inklusive Beispielprogramm: Präsentationsgrafik

Unser Superhit weiter im Programm!

3D-DRAW 1.0

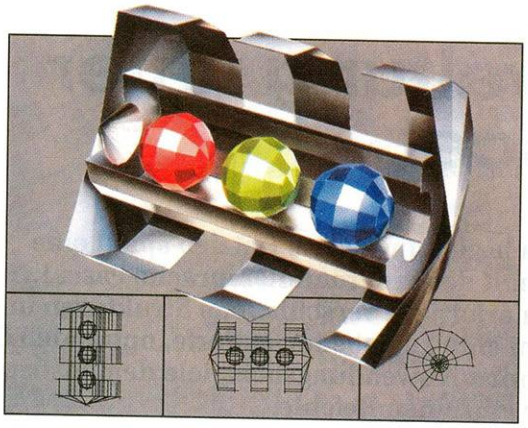
Dreidimensionales CAD-Programm

* Projektionswinkel frei wählbar * 2D-Editor zur schnellen Konstruktion von Objekten * Integrierter Taschenrechner * Größe und perspektivische Verzerrung einstellbar * Kombination von getrennt konstruierten Teilen * Punkte, Kanten und Flächen frei editierbar * Drei Raumachsen * Verdeckte Flächen * Lichtquellensimulation * Animationsroutine * Verformen der Objekte durch Zufallszahl * Rotations- und Translationskörper * Programmierer-Schnittstelle * Viele Ausgabeformate * Pulldown-Menüs * viele Beispielobjekte

Systemvoraussetzungen:
Computer: PC, XT, AT oder Kompatibler mit mindestens 640 KByte Arbeitsspeicher. Hercules-, CGA-, EGA- oder VGA-Karte. Microsoft-kompatible Maus unbedingt erforderlich. MS-DOS ab Version 3.0
Ausgabemedien: Epson-kompatibler Drucker, Postscript-fähiger Laserdrucker, Plotter mit HPGL-Standard

nur DM **99,–** *

3D-DRAW PROFESSIONAL



Komfortables, dreidimensionales Konstruktionsprogramm

Neu in der Version »Professional«:

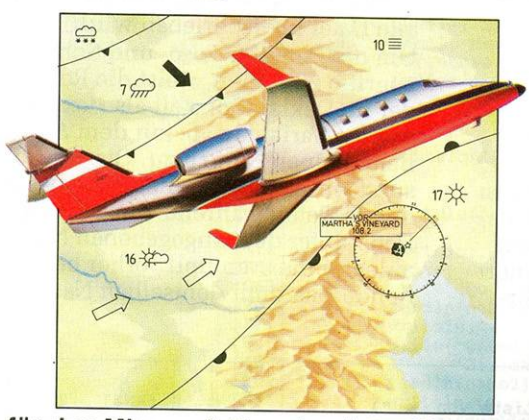
3D-Modus: Kantenmodelle und Flächenmodelle mit und ohne Lichtquellen animierbar * Anwählbare Lichtquellen * Rotationen und Translationen um beliebige Raumachsen * Manipulierbare Bereiche * Konfigurierbarer Druckertreiber * 2000 Punkte und Flächen sowie 4000 Kanten darstellbar * Einzelbehandlung von Körpern * Leuchtbalken-Menüs zur besseren Anwählbarkeit
2D-Modus: Die Mauszeiger-Bedienung auch über Tastatur * Erhöhte Arbeitsgeschwindigkeit aller Programme * Zurücknehmen des letzten Arbeitsschrittes * Quick-Modus für einfache Grundformen wie Kreise oder Vielecke * Werkzeuge zum Pressen, Stauchen und Dehnen von Objekten, Körpern und Bereichen

Systemvoraussetzungen
Computer: PC, XT, AT oder Kompatibler mit mindestens 640 KByte Arbeitsspeicher (Extendend Memory wird unterstützt), ein oder zwei Diskettenlaufwerke beziehungsweise ein Diskettenlaufwerk und eine Festplatte, Portables mit LCD-Displays
Grafikkarten: Hercules-, LCD-, CGA-, EGA-, VGA- oder AT & T-Karte
Ausgabemedien: Epson-kompatible 8-, 9- oder 24-Nadel-Drucker, HP-kompatible Plotter, Postscript-Drucker (einfarbig oder mehrfarbig)
Sonstiges: Microsoft-kompatible Maus unbedingt erforderlich, MS-DOS ab Version 3.0

nur DM **199,–** *

Jetzt noch naturgetreuer und noch realistischer!

Navigator 4.0



für den Microsoft Flugsimulator III und IV

Entdecken Sie den Microsoft Flugsimulator neu! Wählen Sie Ihre Startposition aus mehr als 550 Flughäfen aus. Die Übergabe der Daten an den Flugsimulator erfolgt automatisch ohne Umweg über die Tastatur.
Die Leistungsmerkmale auf einen Blick:
* Auf Wunsch automatische Eingaben in die Menüs des Microsoft Flugsimulators * 252 Flugrouten mit allen zu überfliegenden VOR-Stationen * Formatierte Ausdrucke aller Daten * Absolut realistischer Wettergenerator * Zehn verschiedene Flugzeugtypen * Anzeige der benötigten Flugzeit für alle Maschinen * Selbständige Berechnung des Kurses zu jedem gewünschten Zielflughafen * Bibliothek für 250 Flugrouten * Ausführliches Fluglog * Umschaltung in Schwarzweißmodus * Für Boden- und Höhenwinde ausschaltbare Turbulenzen

Systemvoraussetzungen:
Microsoft Flugsimulator 3.0 oder 4.0, 512 KByte Arbeitsspeicher, bel. Grafikkarte

nur DM **69,–** *

DOS SOFTWARE SOFTWARE SOFTWARE SOFTWARE SOFTWARE

NEU

3D-DRAW PROFESSIONAL

KOMFORTABLES DREIDIMENSIONALES KONSTRUKTIONS-PROGRAMM

NAVIGATOR 4.0

FÜR DEN MICROSOFT FLUGSIMULATOR III UND IV

* Sämtliche Programme sind wahlweise auf 5 1/4-Zoll- oder 3 1/2-Zoll-Diskette erhältlich. (Bitte bei Bestellung angeben.)
Unabhängig von der Anzahl der bestellten Artikel berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung.
Unverbindliche Preisempfehlung. Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

Dateiverwaltung unter Turbo Pascal

Die Organisation von Daten beziehungsweise Dateien spielt eine wichtige Rolle im Umgang mit der Programmiersprache Pascal. Unterschiedliche Verfahren erfordern verschiedene Strukturen und Verarbeitungsmethoden, um eine aufgabenorientierte, optimale Lösung zu erzielen. Die folgenden Anwendungsbeispiele demonstrieren die Funktionsweise der einzelnen Befehle.

Die Dateiorganisation von Turbo Pascal besteht grundsätzlich aus zwei Komponenten: der Datenverarbeitung auf Basisniveau und der auf höherer Ebene. Der Grundlagenteil zur Dateiverwaltung ist aus diesem Grund in zwei Abschnitte aufgeteilt. In diesem Beitrag finden Sie Ausführungen zu den Sprachelementen, die Turbo Pascal für die Verwaltung und Modifikation von Dateien zur Verfügung stellt. Der Artikel »Der Diskettendoktor hilft in allen Lebenslagen« geht auf die Dateiverwaltung im prozessornahen Bereich ein, das heißt, er beschäftigt sich mit der direkten Ansteuerung von externen Datenträgern.
Turbo Pascal enthält im Grunde genommen Sprachelemente für die Bearbeitung von Textdateien, typisierten und untypisierten Dateien.

Textdateien werden sinnvollerweise sequentiell gelesen und geschrieben

Eine Textdatei setzt sich aus Zeilen zusammen, das heißt, die Datenblöcke sind durch ein Enter-Zeichen voneinander getrennt. Jede Zeile besteht aus beliebigen Zeichen, auch Sonderzeichen sind erlaubt. Da normalerweise jede Zeile eine im Vergleich zu den anderen Zeilen unterschiedliche Länge aufweist, können Sie nicht direkt auf einen bestimmten Datensatz zugreifen. Wenn Sie beispielsweise die 59. Zeile einer Datei bearbeiten wollen, müssen Sie erst die vorausgehenden 58 Zeilen lesen. Auf diese Art und Weise läßt sich zwar eine Datei von der Größe her optimal ausnutzen, dies geht jedoch auf Kosten der Verarbeitungsgeschwindigkeit.

Typisierte Dateien ermöglichen den wahlfreien Zugriff

Das Beispiel in Listing 1 liest den Inhalt einer Textdatei zeilenweise, sortiert ihn und schreibt ihn wieder geordnet in dieselbe Datei zurück. Beachten Sie bei der Erprobung, daß die alte Datei überschrieben wird.
Um die Datei für Lese- und Schreibzugriffe zu öffnen, weist die Routine der im Deklarationsteil als »text« definierten Variablen »f« mit dem Assign-Befehl einen Namen zu. Die Anweisung »reset« öffnet eine schon bestehende Datei, während »rewrite« eine Datei unter dem angegebenen Namen anlegt; eine eventuell vorher bestehende Datei mit demselben Namen wird in diesem Fall überschrieben.
Textdateien stellen die einzige Dateiart dar, an der man sowohl mit »read« und »write« als auch mit den entsprechenden Pendants »readln« und »writeln« arbeiten darf.
Typisierte Dateien, auch als »relative Dateien« bezeichnet, haben im Gegensatz zu Textdateien den Vorteil, daß Sie wahlfrei auf die einzelnen Bestandteile zugreifen können. Ein Anwendungsmodell finden Sie in Listing 2, das die grundlegenden Bestandteile einer Adreßverwaltung zeigt. Eine typisierte Datei besteht aus einer Menge von Datensätzen, die den gleichen Komponententyp besitzen. In unserem Muster ist das die Datenstruktur »adressentyp«, die im Deklarationsteil als Grundlage für die Variable »adresse« dient.
Die Initialisierungsroutinen gleichen dem Verfahren zum Öffnen einer Textdatei. Der Unterschied besteht darin, daß Sie den Positionszeiger mit dem Seek-Kommando an eine bestimmte Stelle setzen können. Mit den Read- und Write-Befehlen legen Sie die einzelnen Komponenten dann auf dem externen Datenträger ab.

Untypisierte Dateien gehören zu den variablen Dateitypen

Die untypisierten Dateien sind die variantenreichste Dateiart. Da man sie über die Kommandos »blockread« und »blockwrite« anspricht, das heißt, blockweise liest und schreibt, interessiert die zugrundeliegende Datenstruktur nicht.
Das Löschen oder Umbenennen ist wie bei den anderen Dateitypen zulässig. Als Beispiel dient Listing 3, das eine Datei über den Umweg des Zwischenspeichers »puffer« kopiert, ohne eventuell auftretende Kopierfehler zu beachten.

(Markus Zietlow/dm)

```
[S35]    1: (* Sortieren einer Liste von    *)
[TD5]    2: (* Zeichenketten mithilfe eines *)
[245]    3: (* gemischten (rekursiven und   *)
[RY4]    4: (* iterativen) Algorithmus      *)
[WS4]    5: (* unter Zuhilfenahme der       *)
[AD5]    6: (* datenträgerorientierten      *)
[7T4]    7: (* Ein- und Ausgabebefehle      *)
[XU4]    8: (* Autor: Markus Zietlow        *)
[ØØØ]    9:
[8B5]   1Ø: program gemischtes_sortieren;
[ØØØ]   11:
[RNØ]   12: var
[7B2]   13:   f              : text;
[Ø33]   14:   name           : string;
[R74]   15:   anzahl, zaehler: integer;
[2T5]   16:   liste          : array[1..2ØØ] of string;
[ØØØ]   17:
[PZ5]   18: procedure sortieren(linke_grenze,
[S83]   19:                     rechte_grenze, lft,
[KG2]   2Ø:                     rgt : integer);
```

```
[RNØ]   21: var
[573]   22:   mitte, x, y:string;
[MUØ]   23: begin
[Y77]   24:   mitte:=liste[(linke_grenze+rechte_grenze)
[2A1]   25:                div 2];
[P31]   26:   repeat
[FH2]   27:    x:=liste[lft];
[YP2]   28:    while x<mitte do
[MUØ]   29:      begin
[LW1]   3Ø:        lft:=lft+1;
[YB2]   31:        x:=liste[lft]
[ZNØ]   32:      end;
[7K2]   33:    y:=liste[rgt];
[2P2]   34:    while mitte<y do
[MUØ]   35:      begin
[WX1]   36:        rgt:=rgt-1;
[PC2]   37:        y:=liste[rgt]
[ZNØ]   38:      end;
[GN2]   39:    if lft<=rgt then
[MUØ]   4Ø:      begin
```

```
[2M2]   41:         liste[rgt]:=x;
[BL2]   42:         liste[lft]:=y;
[LW1]   43:         lft:=lft+1;
[GP1]   44:         rgt:=rgt-1
[ZNØ]   45:       end;
[ZE2]   46:     until rgt<lft;
[284]   47:     if linke_grenze<rgt then
[8Y4]   48:       sortieren(linke_grenze, rgt,
[2C3]   49:                 linke_grenze, rgt);
[M94]   5Ø:     if lft<rechte_grenze then
[UZ4]   51:       sortieren(lft, rechte_grenze,
[UD3]   52:                 lft, rechte_grenze);
[ZNØ]   53: end;
[ØØØ]   54:
[MUØ]   55: begin
[T26]   56:   writeln('Programm zum rekursiven '+
[6K4]   57:           'Sortieren einer Liste von '+
[P12]   58:           'Elementen');
[Y15]   59:   writeln('------------------------------'+
[YK3]   6Ø:           '------------------------------');
[XF1]   61:   writeln;
[W56]   62:   write('Geben Sie bitte den Dateinamen '+
[Y41]   63:         'ein: ');
[722]   64:   readln(name);
[XF1]   65:   writeln;
[KF2]   66:   assign(f,name);
[VE1]   67:   reset(f);
[ZN1]   68:   anzahl:=Ø;
[Z23]   69:   while not eof(f) do
[MUØ]   7Ø:     begin
[D94]   71:       readln(f,liste[anzahl+1]);
[X82]   72:       inc(anzahl,1);
[ZNØ]   73:     end;
[KD1]   74:   close(f);
[1V4]   75:   sortieren(1,anzahl,1,anzahl);
[YY1]   76:   rewrite(f);
[XF1]   77:   writeln;
[XF1]   78:   writeln;
[XF1]   79:   writeln;
[3R6]   8Ø:   writeln('Die sortierte Liste sieht wie '+
[852]   81:           'folgt aus:');
[XF1]   82:   writeln;
[TF4]   83:   for zaehler:=1 to anzahl do
[MUØ]   84:     begin
[CF5]   85:       writeln('Das ',zaehler,'. Element '+
[D14]   86:               'ist: ',liste[zaehler]);
[6L4]   87:       writeln(f,liste[zaehler]);
[ZNØ]   88:     end;
[XF1]   89:   writeln;
[KD1]   9Ø:   close(f);
[KNØ]   91: end.
```

Listing 1. Textdateien bearbeitet man im Gegensatz zu den anderen Dateitypen auch mit »readln« und »writeln«

```
[644]    1: (* Demonstration der        *)
[664]    2: (* Eigenschaften einer      *)
[TN4]    3: (* typisierten Datei anhand *)
[134]    4: (* der Fragmente einer      *)
[744]    5: (* Adreßverwaltung          *)
[XG4]    6: (* Autor: Markus Zietlow    *)
[ØØØ]    7:
[NC4]    8: program TypisierteDatei;
[ØØØ]    9:
[5H1]   1Ø: uses crt;
[ØØØ]   11:
[STØ]   12: type
[AA3]   13:   Adressentyp=record
[XG3]   14:     Vorname : string[2Ø];
[X33]   15:     Name    : string[25];
[KK3]   16:     Strasse : string[3Ø];
[Z53]   17:     Plz     : string[4];
[YK3]   18:     Wohnort : string[2Ø];
[ZG3]   19:     Telefon : string[15];
[ZNØ]   2Ø:   end;
[ØØØ]   21:
[RNØ]   22: var
[5M3]   23:   f:file of adressentyp;
[X32]   24:   fDaten:text;
[KP3]   25:   adresse:adressentyp;
[U52]   26:   name:string;
[3A4]   27:   maxadre, zaehler:integer;
[ØØØ]   28:
[RW4]   29: procedure AdresseSpeichern;
[MUØ]   3Ø:   begin
[ER2]   31:     rewrite(fDaten);
[F14]   32:     writeln(fDaten,maxadre);
[M14]   33:     writeln(fDaten,zaehler);
[WG2]   34:     seek(f,zaehler);
[FW2]   35:     write(f,adresse);
[U92]   36:     close(fDaten);
[ZNØ]   37:   end;
[ØØØ]   38:
[A44]   39: procedure AdresseLesen;
[MUØ]   4Ø:   begin
[3B2]   41:     reset(fDaten);
[YU3]   42:     readln(fDaten,maxadre);
[EX3]   43:     readln(fDaten,zaehler);
```

```
[WG2]   44:     seek(f,zaehler);
[7G2]   45:     read(f,adresse);
[U92]   46:     close(fDaten);
[ZNØ]   47:   end;
[ØØØ]   48:
[PR3]   49: procedure Initialize;
[MUØ]   5Ø: begin
[NX1]   51:   zaehler:=1;
[3Y1]   52:   maxadre:=1;
[UX3]   53:   assign(f,'adressen.ran');
[GY4]   54:   assign(fDaten,'adressen.dat');
[DXØ]   55:   {$I-}
[VE1]   56:   reset(f);
[BXØ]   57:   {$I+}
[T53]   58:   if IOResult<>Ø then
[MUØ]   59:   begin
[YY1]   6Ø:     rewrite(f);
[213]   61:     adressespeichern;
[ZNØ]   62:   end;
[N92]   63:   adresselesen;
[ZNØ]   64: end;
[ØØØ]   65:
[MUØ]   66: begin
[H12]   67:   Initialize;
[P31]   68:   repeat
[DB1]   69:     ClrScr;
[WG2]   7Ø:     seek(f,zaehler);
[7G2]   71:     read(f,adresse);
[ET4]   72:     writeln('Adresse: ',zaehler);
[XF1]   73:     writeln;
[XF1]   74:     writeln;
[2R2]   75:     with adresse do
[MUØ]   76:       begin
[7W4]   77:         writeln('Vorname: ',vorname);
[EW3]   78:         writeln('Name:    ',name);
[B95]   79:         writeln('Straße:  ',strasse);
[TS3]   8Ø:         writeln('Plz:     ',plz);
[EØ5]   81:         writeln('Wohnort: ',wohnort);
[KU4]   82:         writeln('Telefon: ',telefon);
[ZNØ]   83:       end;
[W15]   84:     write('Welche Adresse soll ich '+
[9M4]   85:           'anzeigen ? ("Ø" für "Ende") ');
[LK2]   86:     readln(zaehler);
[VR2]   87:   until zaehler=Ø;
[KD1]   88:   close(f);
[KNØ]   89: end.
```

Listing 2. Der wahlfreie Zugriff in einer typisierten Datei läßt sich anhand der Fragmente einer Adreßverwaltung leicht nachvollziehen

```
[YU4]    1: (* Einfaches Kopierprogramm *)
[L25]    2: (* ohne Fehlerüberprüfung   *)
[XG4]    3: (* Autor: Markus Zietlow    *)
[ØØØ]    4:
[D63]    5: program BlockDemo;
[ØØØ]    6:
[RNØ]    7: var
[ØT3]    8:   quelle, ziel         : file;
[BY3]    9:   name1, name2         : string;
[EH4]   1Ø:   lergebnis, sergebnis : word;
[F55]   11:   puffer : array[1..2Ø48] of byte;
[ØØØ]   12:
[ØØØ]   13:
[MUØ]   14: begin
[CØ4]   15:   writeln('Programm zum '+
[LM4]   16:           'Kopieren einer Datei ohne '+
[6Ø3]   17:           'Fehlerprüfung');
[WL4]   18:   writeln('--------------------------'+
[Z93]   19:           '--------------------------');
[XF1]   2Ø:   writeln;
[EZ5]   21:   write('Geben Sie bitte den Namen der '+
[D63]   22:         'Quelldatei ein: ');
[W62]   23:   readln(name1);
[XF1]   24:   writeln;
[EZ5]   25:   write('Geben Sie bitte den Namen der '+
[4Z2]   26:         'Zieldatei ein: ');
[X62]   27:   readln(name2);
[XF1]   28:   writeln;
[1E3]   29:   assign(quelle,name1);
[273]   3Ø:   assign(ziel,name2);
[AL2]   31:   reset(quelle,1);
[4N2]   32:   rewrite(ziel,1);
[P31]   33:   repeat
[BT6]   34:     blockread(quelle,puffer,sizeof(puffer),
[MY1]   35:               lergebnis);
[SP5]   36:     blockwrite(ziel,puffer,lergebnis,
[2Z1]   37:                sergebnis);
[TH3]   38:   until (lergebnis=Ø) or
[8Z3]   39:         (lergebnis<>sergebnis);
[UP5]   4Ø:   writeln('Die kopierte Datei ist ',
[PL5]   41:           filesize(quelle),' Bytes groß.');
[UB2]   42:   close(quelle);
[TZ1]   43:   close(ziel);
[KNØ]   44: end.
```

Listing 3. Untypisierte Dateien setzen keine Angaben über die Art und Organisation der gespeicherten Daten voraus

Die Grafikprogrammierung in Turbo Pascal

Gerade Compiler-Sprachen wie Turbo Pascal sind sehr beliebt, um grafische Anwendungen zu programmieren, da Compiler in der Regel schnelle Programme gewährleisten. Turbo Pascal erzeugt einen sehr effektiven Code. Diesen Vorteil können Sie gewinnbringend nutzen, wie der folgende Artikel veranschaulicht.

Computerbildschirme sind flüchtige Rasterbildschirme. Ein Punkt auf dem Bildschirm verschwindet nach Bruchteilen einer Sekunde wieder, wird er nicht erneut aktiviert. Ein Videobild wird 50 bis 70mal in der Sekunde neu aufgebaut, um den Eindruck eines stehenden Bildes zu erzeugen. Dazu »wandert« ein Elektronenstrahl über den Bildschirm und aktiviert gezielt einzelne Punkte. Die Anzahl solcher Punkte, die ein Monitor beziehungsweise eine Grafikkarte darstellen kann, bezeichnet man als Auflösung. Eine hohe Auflösung bedeutet, daß sich horizontal oder vertikal viele Punkte darstellen lassen. Eine niedrige Auflösung zeigt das Gegenteil an. Grafikkarten stellen in der Regel eine ganze Reihe von unterschiedlichen Auflösungen zur Verfügung. Von Auflösung zu Auflösung variiert die Anzahl der darstellbaren Farben. Die Kombination von Auflösung und der dazugehörigen Farbanzahl bezeichnet man als Bildschirmmodus oder Videomodus. Tabelle 1 zeigt die verschiedenen Videomodi für alle gängigen PC-Grafikkarten.

Die unterschiedlichen Bildschirmmodi werden gleichermaßen unterstützt

Turbo Pascal faßt seinen kompletten Befehlsumfang zur Grafikprogrammierung in der Unit »graph« zusammen. Daher ist das Kommando »uses graph« das erste, das in ein Programm gehört, in dem auf Grafikbefehle zugegriffen wird. Mit dem Kommando wird der Programmcode der Graph-Unit in das aktuelle Programm eingebunden.

Bevor eine Grafik dargestellt werden kann, müssen Sie einen Bildschirmmodus initialisieren. Dazu dient das Initgraph-Kommando. Mit Aufruf des Initgraph-Befehls weisen Sie Turbo Pascal an, einen zur angeschlossenen Grafikkarte gehörenden Grafiktreiber in den Speicher zu laden und über diesen Grafiktreiber einen Bildschirmmodus zu initialisieren. Das Konzept der unterschiedlichen Grafiktreiber ist eine Konsequenz aus den Inkompatibilitäten der einzelnen Grafikkarten zueinander, die aus der unterschiedlichen Hardware resultieren. Turbo Pascal stellt als Schnittstelle zu den Grafikkarten die BGI-Treiber (»BGI« steht als Abkürzung für »Borland Graphics Interface«, also »Borland-Grafikschnittstelle«) bereit. Das sind auf eine bestimmte Grafikkarte abgestimmte Programmbibliotheken, mit denen die Ausführung der Grafikkommandos von Turbo Pascal gesteuert wird. Tabelle 2 zeigt die unterschiedlichen BGI-Treiber von Turbo Pascal.

Turbo Pascal verwendet für den Zugriff auf Grafikkarten die BGI-Treiber

Den Aufruf des Initgraph-Befehls können Sie am einfachsten so durchführen, daß Turbo Pascal selbständig die angeschlossene Grafikkarte erkennt und den dazugehörigen Treiber vom Datenträger lädt. Dazu wird die Funktion wie folgt aufgerufen:

```
treiber:=detect;
{Suche passenden Treiber}
modus:=0;
{Setze Dummy für Modus}
pfad:='c:\turbo5\';
{Setze Pfad}
initgraph(treiber,modus,pfad);
{Initialisiere Grafikmodus}
```

Turbo Pascal erkennt in diesem Fall die Grafikkarte und setzt sie in ihren »Standardmodus«. In der Unit »graph« ist die Konstante »detect« bereits vordefiniert. Wichtig für den InitGraph-Aufruf ist, daß Sie für »treiber« und »modus« tatsächlich Variablen übergeben, eine direkte Übergabe der Werte ist nicht zulässig. Der Grund liegt darin, daß Sie nach dem Aufruf der Prozedur in den beiden Variablen die tatsächlich für den Grafiktreiber und den Bildschirmmodus gesetzten Werte auslesen können.

Für viele Anwendungsgebiete ist es wichtig, einen bestimmten Grafikmodus setzen zu können, zum Beispiel einen CGA-Modus auf einer VGA-Karte. In solchen Fällen rufen Sie die InitGraph-Prozedur direkt mit den Werten für »treiber« und »modus« auf. Hilfreich ist in diesen Fällen der vorherige Aufruf der DetectGraph-Prozedur. Ihr übergeben Sie ebenfalls zwei Variablen für »treiber« und »modus«. Nach dem Aufruf wird in diese Variablen eingetragen, welche Werte Turbo Pascal verwendet, wenn »InitGraph« mit der Konstanten »detect« aufgerufen wird.

Die Initialisierung des Grafikmodus mit dem InitGraph-Kommando ist nur notwendig, wenn Sie den Grafikmodus zum ersten Mal setzen möchten. Ist der Grafiktreiber einmal geladen, können Sie durch Verwendung der SetGraphMode-Prozedur einen Grafikmodus setzen, ohne den Treiber erneut laden zu müssen.

Ein universelles Grafikprogramm, das auf allen unterstützten Grafikkarten in der höchsten Auflösung arbeiten soll, tut gut daran, nach der automatischen Initialisierung des Grafikmodus über die Konstante »detect« die aktuellen Parameter des Grafikmodus auszulesen. Die beiden wichtigsten Werte ermitteln Sie mit den Funktionen »getmaxx« und »getma-

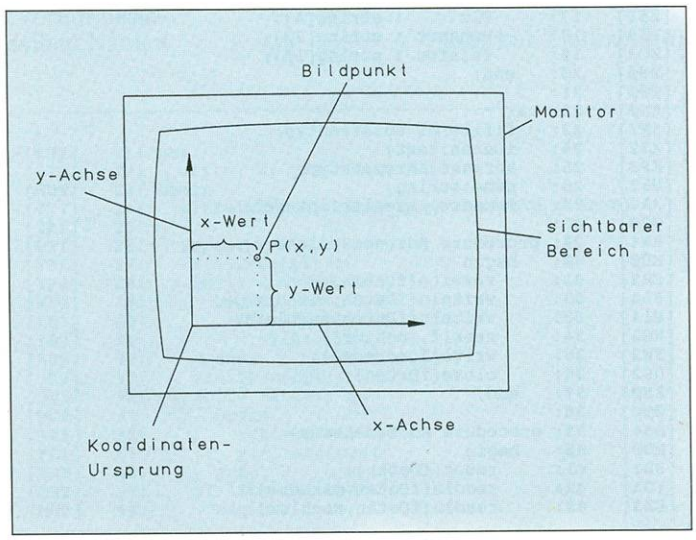


Bild 1. Der Bildschirm mit einem Koordinatensystem

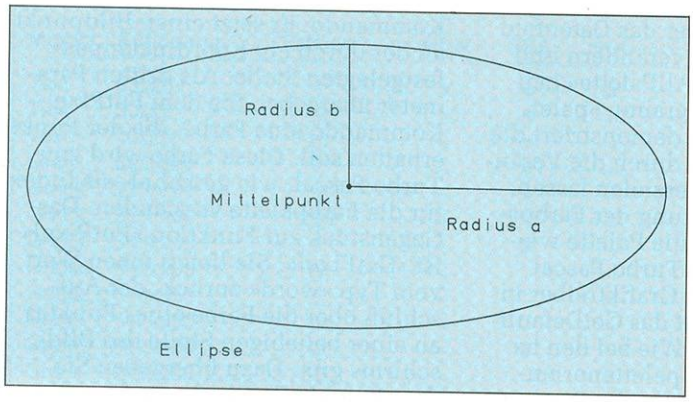


Bild 2. Die Parameter einer Ellipse

xy«. Als Resultat geben Sie die maximal darstellbaren, horizontalen und vertikalen Punkte an. Sind diese Werte bekannt, können schon erste Gehversuche mit dem Grafikbefehlssatz gemacht werden.

Für die Adressierung der Bildpunkte wird das Koordinatensystem verwendet

Die einfachsten Befehle dienen dem Zugriff auf einzelne Punkte des Bildschirms. Mit der PutPixel-Prozedur setzen Sie einen Bildpunkt an einer bestimmten Stelle auf dem Bildschirm. Zur Festlegung, wo der Punkt erscheinen soll, verwenden Sie das Koordinatensystem. Es haben sich unterschiedliche Konventionen eingebürgert, um Punkte auf dem Bildschirm zu benennen und zu lokalisieren. Dabei werden verschiedene Koordinatensysteme verwendet. Das wichtigste und am weitesten verbreitete ist das kartesische (rechtwinklige) Koordinatensystem.
Möglicherweise haben Sie diesen Begriff schon gehört. Auch in der Schule wird bei mathematischen Zeichnungen meist mit dem kartesischen Koordinatensystem gearbeitet. Erinnern Sie sich: Als Sie in der Schule Graphen von Funktionen zeichnen mußten, haben Sie zunächst zwei Achsen auf einem Blatt Papier festgelegt. Die eine wurde als x-Achse, die andere als y-Achse und der Schnittpunkt zwischen beiden Achsen als Koordinatenursprung bezeichnet. Um einzelne Punkte des Funktionsgraphen zu ermitteln, wird zu bestimmten x-Werten, also zu festen Punkten auf der x-Achse, je ein y-Wert errechnet. Dieser Wert wird auf der y-Achse abgetragen und »über« oder »unter« dem x-Wert plaziert. Jeder Punkt hat zwei Koordinaten: einen x-Wert und einen y-Wert. Mit diesem System läßt sich jeder Punkt in der Ebene adressieren und eindeutig lokalisieren. Es ist einfach, dieses System auf einen Bildschirm zu übertragen. So können Sie sich den Koordinatenursprung links unten in der Bildschirmecke vorstellen. Die x-Achse verläuft parallel zur Bildschirmunterseite, die y-Achse vertikal an der linken Bildschirmseite. Bild 1 zeigt diese Aufteilung. Bei der Programmierung unter Turbo Pascal wird der Koordinatenursprung in der linken oberen Ecke des Bildschirms festgelegt.
In diesem Koordinatensystem kann jeder Punkt auf dem Bildschirm eindeutig durch einen x-Wert und einen y-Wert identifiziert werden. Beide bezeichnet man als Koordinaten des Punktes. Je nach Auflösung des Bildschirms existieren unterschiedlich viele Punkte in x- und in y-Richtung.
Nahezu alle Grafikbefehle von Turbo Pascal, die bestimmte Objekte oder grafische Elemente auf den Bildschirm bringen, erwarten ein oder mehrere Paare von Koordinaten. Die Koordinaten werden immer als Integer-Parameter übergeben.
Jedem einzelnen Punkt auf dem Grafikbildschirm können Sie auch eine Farbe zuordnen. Während einfarbige Grafikkarten nur zwei Farben kennen (Weiß und Schwarz beziehungsweise »Punkt an« und »Punkt aus«), erlauben farbfähige Grafikkarten je nach Bildschirmmodus eine unterschiedliche Farbanzahl für einen Punkt.

Die Farbpaletten übernehmen die Festlegung der Bildschirmfarben

Die Farben von Punkten auf dem Bildschirm steuern bei farbfähigen Grafikkarten die sogenannten Paletten. Einfach ausgedrückt ist eine Palette eine Kette von Werten. Jeder dieser Werte codiert eine Farbe. Die Farbwerte, die beim Setzen von Punkten im Bildschirmspeicher gespeichert werden, interpretiert die Grafikkarte als Index für die Palette. In den 16-Farben-Modi der EGA-, MCGA- und VGA-Karten enthält die Farbpalette beispielsweise insgesamt 16 Farben. Beim Aufbau des Videobildes durch den Controller werden die einzelnen Farbwerte nicht direkt an den Monitor gesandt. Vielmehr wird zu jedem Farbwert der korrespondierende Wert aus der Palette gelesen und dieser Wert an den Monitor übergeben. Dieses Verfahren mit Paletten bietet einige große Vorteile. Auf der einen Seite können Sie Änderungen der Farben bereits gesetzter Punkte auf dem Bildschirm sehr einfach vornehmen. Auf der anderen Seite ist die physikalische Codierung der Farbe unabhängig von der logischen Codierung im Bildschirmspeicher. Für den ersten Aspekt ein Beispiel: Sie möchten ein Rechteck auf dem Bildschirm mit der Farbe 0Fhex füllen. Das ist Weiß bei Verwendung der Standardpalette, wie Turbo Pascal sie einstellt. Um das Rechteck in ein gelbes Rechteck zu verwandeln, müssen Sie aufgrund des Palettenprinzips nicht jeden einzelnen Punkt mit einer neuen Farbe versehen. Vielmehr reicht es, den Eintrag für 0Fhex in der Palette auf den Wert für Gelb zu setzen, und schon erhalten alle Punkte auf dem Bildschirm mit der »Farbe« 0Fhex ein neues Aussehen.
Der zweite Vorteil der Paletten, die Unabhängigkeit zwischen physikalischer und logischer Codierung, bedeutet in der Praxis, daß Sie Punkte mit wenigen Bits codieren können, obwohl die tatsächlich angezeigten Farbwerte sehr viel »größer« sind. So ist es denkbar, Punkte im Bildschirmspeicher mit 4 Bit zu codieren, während sie in der Palette mit beispielsweise 32 oder 64 Bit und mehr repräsentiert werden. Dieses Verfahren erspart eine große Menge Platz im Bildschirmspeicher, da nicht jeder einzelne Punkt mit 32 oder 64 Bit und mehr gespeichert wird.
Für die Auswahl der Palette stellt Ihnen Turbo Pascal den vordefinierten Datentyp »PaletteType« zur Verfügung. Er ist als Verbund definiert und enthält neben der Anzahl der Farben in der Palette ein Variablenfeld, in dem alle einzelnen Farbwerte gespeichert sind. Wird eine Variable für den Zugriff auf die Palette beispielsweise mit der Anweisung

```
var palette:palettetype;
```

definiert, so können Sie die aktuelle Palette mit dem Kommando:

```
getpalette(palette);
```

auslesen. Das Datenfeld »palette.size« gibt Ihnen dann die Anzahl der Farben in der Palette zurück. Das Byte-Array »palette.colors[]« enthält die Farbwerte.
Für die Feststellung, ob Farbdarstellung auf einer Grafikkarte möglich ist, können Sie die Funktion »GetPaletteSize« verwenden. Sie liefert als Funktionsergebnis vom Typ »word«

die Anzahl der Farben in der aktuellen Palette zurück. Das Programm »zeigfarb.pas« in Listing 1 demonstriert den Zugriff auf den Datentyp »PaletteType«, indem es alle Farben der aktuellen Palette und einige andere Informationen über den Bildschirmmodus und den Grafikadapter auf dem Bildschirm ausgibt.

Über die Veränderung der Paletten lassen sich interessante Farbeffekte erzielen

Eine bestimmte Farbe aus der Palette wird von Turbo Pascal als aktuelle Zeichenfarbe eingestellt. Die Nummer dieser Farbe in der Palette können Sie mit dem GetColor-Kommando ermitteln und mit dem SetColor-Befehl neu setzen.
Fast alle Befehle von Turbo Pascal, mit denen grafische Elemente gezeichnet werden, greifen auf die aktuelle Palette zu. Bedingt durch die Tatsache, daß Turbo Pascal den Wert, mit dem ein Bildpunkt auf dem Bildschirm verschlüsselt ist, als Index in der Farbpalette versteht, lassen sich durch die Manipulation der Paletten eindrucksvolle Effekte erzielen. Zeichnen Sie beispielsweise einen Kreis, nachdem Sie zuvor mit der SetColor-Anweisung die aktuelle Zeichenfarbe auf den Wert 4 eingestellt haben, so werden alle einzelnen Punkte des Kreises mit dem Wert 4 im Bildschirmspeicher codiert. Die tatsächliche Farbe, mit der die Punkte des Kreises dargestellt werden, hängt jedoch davon ab, welcher Wert an der Stelle »palette.colors[4]« in der Palette gespeichert ist. Eine Änderung dieses Wertes reicht aus, um allen Punkten des Kreises eine neue Farbe zu verleihen, ohne den Kreis neu zeichnen zu müssen. Dazu wird mit der SetPalette-Anweisung die Nummer eines Palettenelements und ein dazugehöriger Farbwert zugewiesen. Turbo Pascal setzt den übergebenen Wert dann in die Palette ein. Für das oben dargestellte Beispiel des Kreises mit der Farbe 4 legt die folgende Anweisung eine neue Kreisfarbe fest:

```
setpalette(4,50);
{Setze Farbe 4 auf Farbwert 50}
```

Beabsichtigen Sie, nicht nur einen einzigen Wert der Palette, sondern eine Reihe von Werten zu verändern, bietet sich der SetAllPalette-Befehl an. Er legt eine komplett neue Palette für die Farbdarstellung fest. Dazu übergeben Sie eine Variable, die mit dem Datentyp »PaletteType« definiert wird. Für die Manipulation einer Palette empfiehlt es sich, die Farbwerte zunächst mit dem GetPalette-Befehl auszulesen, dann über das Datenfeld »palette.colors[]« zu verändern und schließlich mit »SetAllPalette« neu festzulegen. Das Programm »paletten.pas« in Listing 2 demonstriert die Farbeffekte, die sich durch die Veränderung der Paletten erzielen lassen. Nach einer Veränderung der Farbpalette ist es nützlich, die Palette wiederherzustellen, die Turbo Pascal nach dem Laden des Grafiktreiber initialisiert. Dazu dient das GetDefaultPalette-Kommando. Wie bei den bereits beschriebenen, palettenorientierten Befehlen, erlaubt die Direktive »GetDefaultPalette« das Auslesen der Standardpalette in eine Palettenvariable. Diese wird über den Datentyp »PaletteType« deklariert.

Turbo Pascal definiert eine Vielzahl von Datenstrukturen in der Graph-Unit

Eine Farbe der Palette ist stets die aktuelle Zeichenfarbe. Sie wird verwendet, wenn Punkte, Linien, Kreise, Kreisausschnitte, Ellipsen, Boxen und so weiter gezeichnet werden. Die aktuelle Zeichenfarbe können Sie über die GetColor-Funktion auslesen. Diese Funktion liefert Ihnen ein Funktionsergebnis vom Typ »word« zurück, das den Index der aktuellen Zeichenfarbe in der Farbpalette spezifiziert. Die Festlegung einer neuen Zeichenfarbe geschieht über den SetColor-Befehl. Er erlaubt die Übergabe eines Word-Wertes, der als neuer Farbwert verwendet wird. Neben der Zeichenfarbe speichert Turbo Pascal auch die Hintergrundfarbe. Ausgelesen wird sie mit der GetBkColor-Funktion, die als Funktionsergebnis den Index der aktuellen Hintergrundfarbe in der Farbpalette ermittelt. Sie können diese über das SetBkColor-Kommando neu festlegen. Der einfachste Grafikbefehl ist das PutPixel-Kommando. Er setzt einen Bildpunkt an der durch ein Koordinatenpaar festgelegten Stelle. Als dritten Parameter übergeben Sie dem PutPixel-Kommando eine Farbe, die der Punkt erhalten soll. Diese Farbe wird von Turbo Pascal, wie gewohnt, als Index für die Farbpalette verstanden. Das Gegenstück zur Funktion »PutPixel« ist »GetPixel«. Sie liefert einen Wert vom Typ »word« zurück, der Aufschluß über die Farbe eines Punktes an einer beliebigen Stelle des Bildschirms gibt. Dazu übergeben Sie dem Befehl »GetPixel« die Koordinaten des auszulesenden Punktes.

Während das Zeichnen von einzelnen Punkten relativ selten vorkommt, finden sich Kommandos zum Zeichnen von Linien in fast jedem Grafikprogramm. Hier kennt Turbo Pascal zwei unterschiedliche Verfahren: das absolute Zeichnen und das relative Zeichnen. Beim absoluten Zeichnen bestimmen Sie die Position der Linie direkt durch zwei Koordinatenpaare. Turbo Pascal verbindet die beiden Punkte durch eine Linie und zeichnet sie mit der aktuellen Farbe. Der Befehl dazu lautet »line«. Bei jedem Zeichenkommando »merkt« sich Turbo Pascal die Koordinaten des Bildpunktes, der als letzter angesprochen wurde.

Punkte können in Turbo Pascal sowohl relativ als auch absolut gezeichnet werden

Hier setzt das LineTo-Kommando an. Bei diesem Befehl werden nur die Koordinaten des eines Endpunktes angegeben, die Linie wird automatisch vom zuletzt angesprochen Bildpunkt bis zum übergebenen Endpunkt gezeichnet.

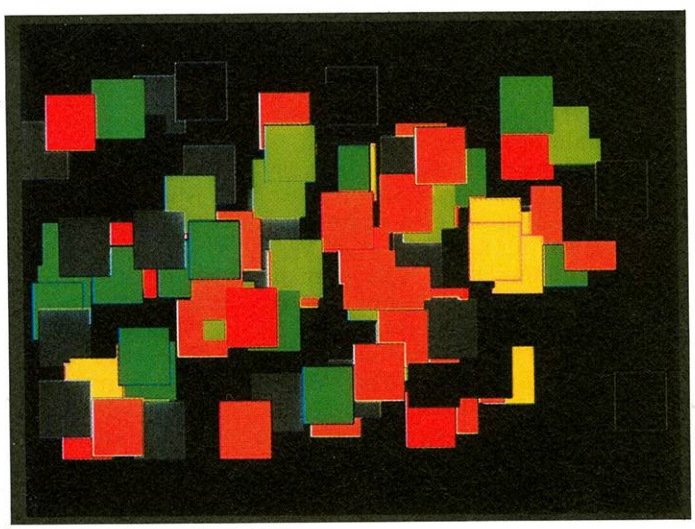


Bild 3. Der Bildschirm nach dem Start von »sprite.pas«

Modus	Auflösung	Farben	MDA	CGA	HGC	EGA	MCGA	VGA
Text	80x25	2	●	●	●	●	●	●
Text	80x25	16	–	●	–	●	●	●
Text	40x25	16	–	●	–	●	●	●
Grafik	320x200	4	–	●	–	●	●	●
Grafik	320x200	16	–	●	–	●	●	●
Grafik	320x200	256	–	–	–	–	●	●
Grafik	640x200	2	–	●	–	●	●	●
Grafik	640x200	16	–	–	–	●	●	●
Grafik	640x350	2	–	–	–	●	●	●
Grafik	640x350	16	–	–	–	●	–	●
Grafik	640x480	2	–	–	–	–	●	●
Grafik	640x480	16	–	–	–	–	–	●
Grafik	720x348	2	–	–	●	–	–	–

Tabelle 1. Die Bildschirmmodi der gängigen PC-Grafikkarten

Eine dritte Variante des Line-Befehls findet sich in der LineRel-Prozedur. Sie zeichnet eine Linie ab der aktuellen Grafikposition. Dazu übergeben Sie der Funktion zwei Parameter, die als Differenz der x- und y-Koordinate verstanden werden. Turbo Pascal bildet intern die Summe aus den aktuellen Koordinaten und den übergebenen Werten und benutzt das Ergebnis als Endpunkt der Linie. Die drei Kommandos lassen sich natürlich problemlos kombinieren. Die folgende Befehlssequenz zeichnet ein Dreieck mit den Endpunkten (50,50), (150,50) und (100,100), indem für jeden der Schenkel eine andere Line-Variante benutzt wird:

```
line(50,50,150,50);
{zeichne Unterseite
lineto(100,100);
{zeichne rechten Schenkel}
linrel(-50,-50);
{und linken Schenkel}
```

Eine leistungsfähige Eigenschaft von Turbo Pascal besteht darin, daß das Aussehen der Linien beim Zeichnen festlegt werden kann. Dazu ist in der Unit »graph« der Datentyp »LineSettingsType« definiert, der Informationen über das Erscheinungsbild von Linien aufnimmt oder festlegt. Um die aktuellen Parameter bei der Darstellung von Linien auszulesen, wird zunächst eine Variable über diesen Datentyp definiert:

```
var linie:linesettingstype;
```

Über die GetLineSettings-Prozedur können Sie die aktuelle Einstellung für das Zeichnen der Linien auslesen. Der Datensatz »LineSettingsType« stellt die beiden Felder »LineStyle« und »Thickness« zur Verfügung, mit denen Sie die Art der Linie (gepunktet, gestrichelt, durchgezogen, Strich-Punkt-Strich) und die Dicke der Linie (normal oder fett) auslesen oder festlegen können. Tabelle 3 zeigt die möglichen Werte für die Datenfelder. Verändert werden alle Einstellungen über die Prozedur »SetLineStyle«. Die folgende Sequenz liest die aktuellen Einstellungen aus, ändert »LineStyle« auf »gestrichelt« und setzt die neuen Parameter:

```
getlinesettings(linie);
{Lese Parameter aus}
linie.linestyle:=dashedln;
{Schalte auf "gestrichelt"}
setlinestyle(linie.linestyle,
linie.thickness));
{Lege Einstellung fest}
```

Die Einstellungen für die Linienart und -dicke werden von Turbo Pascal nicht nur beim Zeichnen von Linien verwendet. Auch das Zeichnen von Kreisen, Ellipsen und anderen Objekten ist von den eingestellten Parametern abhängig.

Der Grafik-Cursor wird über die Move-Kommandos bewegt

Wollen Sie die Position des internen Grafik-Cursors ändern, ohne dabei zu zeichnen, verwenden Sie die Move-Kommandos. Der MoveRel-Befehl arbeitet analog zum LineRel-Kommando, er zeichnet jedoch keine Linie, sondern bewegt den Grafik-Cursor unsichtbar zu den neuen Koordinaten. Mit dem MoveTo-Befehl können die Koordinaten des Grafik-Cursors direkt angegeben werden. Soll die aktuelle Position des Cursors in einem Programm weiterverarbeitet werden, so können Sie diese mit den beiden Funktion »GetX« und »GetY« auslesen. Beide Funktionen liefern einen Integer-Wert zurück, der die x-beziehungsweise y-Koordinate des Grafik-Cursors definiert.
Kreise werden in Turbo Pascal mit dem Circle-Befehl gezeichnet. Ihm übergeben Sie die Koordinaten des Mittelpunkts und des Kreisradius. Im Gegensatz zum Circle-Kommando ist der Arc-Befehl in der Lage, bestimmte Abschnitte eines Kreises zu zeichnen. Dazu werden außer den Koordinaten des Mittelpunkts und des Radius zwei Winkel angegeben. Die Prozedur zeichnet einen Kreisbogen zwischen den beiden Winkeln. Die Angabe der Winkel erfolgt in Gradmaß. Den Arc-Befehl können Sie sehr gut verwenden, um kurze Bögen zu zeichnen, die mit anschließenden Linien oder ähnlichem zu komplexeren grafischen Objekten zusammenwachsen. Dadurch lassen sich Kreisbögen beispielsweise mit Linien zu größeren grafischen Elementen verbinden. Das Problem beim Anschluß von Kreisbögen an andere Elemente liegt darin, die genauen Koordinaten der Endpunkte des Kreisbogens zu bestimmen. Turbo Pascal bietet mit der Prozedur »GetArcCoords« eine sehr elegante Lösung dieses Problems. Das GetArcCoords-Kommando erwartet eine Variable, die über den vordefinierten Datentyp »ArcCoordsType« deklariert ist. Nach dem Aufruf der Prozedur erhalten Sie in den Feldern »x« und »y« des Verbundes die Koordinaten des Mittelpunktes, während die vier Felder »xs«, »ys«, »xend« und »yend« über die Anfangs- und Endpunktkoordinaten des Kreisbogens Aufschluß geben. Die ermittelten Werte können Sie verwenden, um einen Kreisbogen an Linien und so weiter anzuschließen.
Bild 2 stellt dar, welche Parameter bei der Darstellung einer Ellipse wichtig sind. Im Gegensatz zu einem Kreis wird eine Ellipse über zwei Radien konstruiert, einen für die x- und einen für die y-Achse. Daher ist ein Kreis eine besondere Form der Ellipse, nämlich eine Ellipse mit zwei gleichen Radien. Ellipsen zeichnen Sie mit der Pascal-Prozedur »Ellipse«. Neben den Koordinaten des Mittelpunktes und den beiden Radien werden zwei Werte übergeben, die – analog zum Arc-Kommando – das Intervall festlegen, in dem der Ellipsenbogen gezeichnet werden soll.

Ein Kreis ist eine Spezialform der Ellipse

Für eine komplette Ellipse wählen Sie die Werte 0 und 360. Mit dem Fill/Ellipse-Befehl können Sie komplette Ellipsen zeichnen und ausfül-

Grafikkarte	Treibername
CGA	cga.bgi
MCGA	egavga.bgi
EGA	egavga.bgi
Hercules-Karte	herc.bgi
ATT-400	att.bgi
VGA	egavga.bgi
PC-32701	pc3270.bgi
8514-A	ibm8514.bgi

Tabelle 2. Die unterschiedlichen BGI-Treiber

len. Im Gegensatz zum Ellipse-Kommando entfällt die Angabe der Werte für das Winkelintervall, da nur geschlossene Flächen gefüllt werden können. Zum Füllen der Ellipse werden die aktuellen Einstellungen verwendet. Turbo Pascal stellt zwölf unterschiedliche Füllmuster zur Verfügung. Sie sind in Tabelle 4 zusammengefaßt. Um die aktuellen Parameter zur Füllung auszulesen, verwenden Sie den Datentyp »FillSettings-Type«, mit dem sich über die Prozedur »GetFillSettings« die aktuellen Einstellungen erhalten lassen. Das Datenfeld »FillSettingsType« enthält zwei Felder vom Typ »word«: Das Feld »Pattern« hält fest, welches Füllmuster ausgewählt ist (siehe Tabelle 4), das Feld »Color« gibt Aufschluß über die Farbe des Füllmusters. Sollen die Einstellungen geändert werden, wird das SetFillStyle-Kommando verwendet. Sie können eine Farbe und ein Muster übergeben, das bei nachfolgenden Flächen zum Füllen verwendet wird.
Ebenfalls im Zusammenhang mit Flächen steht der Bar3d-Befehl. Er erlaubt das Zeichnen von dreidimensionalen Balken, wie man sie aus professionellen Balkendiagrammen kennt. Dazu geben Sie die linke obere und die rechte untere Ecke eines Rechtecks an. Ferner spezifizieren Sie die räumliche Tiefe des Balkens. Mit dem letzten Parameter kann eingestellt werden, ob die Oberseite des Balkens gezeichnet werden soll oder nicht. Das Programm »balken.pas« in Listing 3 demonstriert, wie die Bar3d-Prozedur benutzt wird, um numerische Daten grafisch aufzubereiten.

Das Zeichnen von dreidimensionalen Balken ist kein Problem

Die gefüllte Fläche, die der Bar3d-Befehl erzeugt, wird, wie gewohnt, gemäß den aktuellen Einstellungen für das Muster und die Farbe erzeugt.
Der Alleskönner unter den Kommandos zum Füllen von Flächen ist der FloodFill-Befehl. Ihm wird die Koordinate eines Punktes auf dem Bildschirm übergeben. Von diesem Punkt ausgehend, füllt das Kommando alle Punkte aus, bis es auf einen Rand trifft, der die Farbe besitzt, die Sie dem Befehl als dritten Parameter übergeben. Dieser Rand wird noch mit dem angegebenen Farbwert gezeichnet. Das FillPoly-Kommando arbeitet nur korrekt, wenn der Punkt, an dem das Ausfüllen der Fläche beginnt, tatsächlich innerhalb eines geschlossenen Objektes liegt.
Das Rectangle-Kommando zeichnet ein Rechteck, von dem die linke obere und rechte untere Ecke mit zwei übergebenen Koordinatenpaaren angegeben werden. Zum Füllen von mit »Rectangle« gezeichneten Rechtecken können Sie den FloodFill-Befehl verwenden.
Neben dem Zeichnen von Standardelementen wie Kreisen, Ellipsen und Rechtecken bietet Turbo Pascal auch die Möglichkeit, beliebige Vielecke zu zeichnen, ohne die einzelnen Linien mit der Line-Anweisung konstruieren zu müssen. In der Unit »crt« ist für diesen Zweck der Datentyp »PointType« definiert, der als Komponenten eine x- und eine y-Koordinate aufnehmen kann. Zum Zeichnen eines Vielecks mit beispielsweise zwölf Eckpunkten definieren Sie ein Datenfeld über den Datentyp »PointType« wie folgt:

```
var vieleck:array[1..12] of pointtype;
```

Die Koordinaten des ersten Punktes sind nach dieser Deklaration in den Variablen »vieleck[1].x« und »vieleck[1].y« ansprechbar, die weiteren Koordinaten mit entsprechend höheren Indizes. Nachdem Sie alle Koordinaten des Vielecks festgelegt haben, wird die Prozedur »DrawPoly« aufgerufen. Sie übernimmt als Parameter einen Wert, der die Anzahl der Punkte angibt und ein, wie beschrieben, definiertes Punktefeld. Die Punkte dieses Feldes werden gemäß den aktuellen Einstellungen für Linienart und -farbe zu einem Polygon verbunden.
Eine Variante des DrawPoly-Kommandos ist der FillPoly-Befehl. Er zeichnet analog zu »DrawPoly« ein Polygon. Nach dem Zeichnen des Vielecks wird dieses gemäß den aktuellen Einstellungen für das Füllen von Flächen ausgefüllt.

Name	Code	Füllmuster
EmptyFill	0	Hintergrundfarbe
SolidFill	1	gewählte Farbe
LineFill	2	waagerechte Linien
LtSlashFill	3	dünne, schräge Linien nach rechts
SlashFill	4	schräge Linien nach rechts
BkSlashFill	5	schräge Linien nach links
LtBkSlashFill	6	dünne, schräge Linien nach links
HatchFill	7	schraffiert
XHatchFill	8	gekreuzt schraffiert
InterLeaveFill	9	Verändern der Linien
WideDotFill	10	weit auseinanderliegende Punkte
CloseDotFill	11	eng beieinanderliegende Punkte
UserFill	12	frei für Benutzerdefinition

Tabelle 4. Die unterschiedlichen Füllmuster

Für das Zeichnen von Vielecken werden die Eckpunkte in einem Variablenfeld festgehalten

Besonders interessante Effekte erlaubt Turbo Pascal bei der Darstellung von Text auf dem Grafikbildschirm. Das OutText-Kommando schreibt eine Zeichenkette an die aktuelle Position des Grafik-Cursors. Dazu übergeben Sie der Prozedur den String, der dargestellt werden soll. Ähnlich arbeitet das Kommando »OutTextXY«. Hier können Sie über die Festlegung des Textes hinaus angeben, an welcher Koordinate die Anzeige beginnen soll.
Für die Darstellung von Texten bedient sich Turbo Pascal global gespeicherter Parameter. Dabei lassen sich unterschiedliche Zeichensätze, verschiedene Schriftgrößen und

Name	Code	Linienart
SolidLn	0	durchgehend
DottedLn	1	gepunktet
CenterLn	2	Strich, Punkt, Strich
DashedLn	3	gestrichelt
UserBitLn	4	frei für Benutzerdefinition
Name	**Code**	**Liniendicke**
NormalWidth	1	normale Dicke
ThickWidth	3	doppelte Dicke

Tabelle 3. Die Codierung der Linienart und -dicke

Name	Code	Zeichensatz
Default Font	0	normaler Zeichensatz
Triplex Font	1	Triplex-Zeichensatz
Small Font	2	Zeichensatz mit kleinen Buchstaben
Sans Serif Font	3	Zeichensatz mit Sans-Serif-Schriftart
Gothic Font	4	Zeichensatz mit Gothic-Schriftart

Tabelle 5. Die Zeichensätze von Turbo Pascal

Name	Code	horizontale Justierung
LeftText	0	Text beginnt am Grafik-Cursor
CenterText	1	Grafik-Cursor steht in Textmitte
RightText	3	Text endet am Grafik-Cursor
Name	**Code**	**vertikale Justierung**
BottomText	0	Text steht oberhalb des Grafik-Cursors
CenterText	1	Text, Grafik-Cursor steht in Textmitte
TopText	2	Text steht unterhalb des Grafik-Cursors

Tabelle 6. Die Festlegung der Positionierung von Texten

Richtungsangaben verwenden. Bei den Zeichensätzen bietet Turbo Pascal die Auswahl aus insgesamt fünf vorhandenen. Sie sind in Tabelle 5 zusammengefaßt. Einzelne Zeichen des Zeichensatzes können Sie in unterschiedlichen Größen ausgeben. Es sind Vergrößerungsfaktoren zwischen 1 und 10 erlaubt. Als weitere Besonderheit lassen sich Texte unter Turbo Pascal im Grafikmodus nicht nur waagerecht, sondern auch senkrecht anzeigen. Die beschriebenen Einstellungen ändern Sie über die SetTextStyle-Prozedur.

Für die Textdarstellung stehen unterschiedliche Zeichensätze zur Verfügung

Mit dem ersten Parameter definieren Sie den Zeichensatz. Die Richtung, in der der Text angezeigt wird, bestimmen Sie mit dem zweiten Wert. Die Angabe von 0 bewirkt eine waagerechte, ein Wert von 1 eine senkrechte Darstellung. Als letzter Parameter wird die Größe übergeben, mit der einzelne Zeichen des Zeichensatzes ausgegeben werden sollen. Die Anweisung:

```
settextstyle(gothicfont,0,10);
```

definiert den Gothic-Zeichensatz als aktuellen Zeichensatz. Texte werden senkrecht in der Größe 10 dargestellt.
Für die Ausgabe von Text legt das Kommando »SetTextJustify« fest, wie der Text relativ zum aktuellen Grafik-Cursor (bei Verwendung von »OutText«) beziehungsweise zur übergebenen Koordinate (bei Verwendung von »OutTextXY«) erscheinen soll. Sie können den Text entweder rechts oder links vom Grafik-Cursor positionieren oder ihn so plazieren, daß der Grafik-Cursor sich in der Textmitte befindet. Dem SetTextJustify-Befehl wird durch die Übergabe von zwei Parametern für die waagerechte und senkrechte Darstellung gesondert vorgegeben, wo der Text relativ zum Grafik-Cursor erscheinen soll. Tabelle 6 faßt die möglichen Werte für die Parameter zusammen. Das Programm »texte.pas« in Listing 4 demonstriert die unterschiedlichen Zeichensätze im Turbo-Pascal-Grafikmodus.
Zum Auslesen der Parameter der Textdarstellung können Sie die Prozedur »GetTextSettings« verwenden. Für das Holen der Textparameter ist der Datentyp »TextSettingsType« in der Unit »crt« vordefiniert. Bis auf Informationen zum aktuellen Zeichensatz sind hier alle Parameter für die Ausgabe verzeichnet.

Mit dem Image-Befehl werden Bildschirmausschnitte in einen anderen Speicherbereich kopiert

Bei der Anzeige von Texten im Grafikmodus ist es häufig schwierig, abzuschätzen, welche Ausmaße ein Text annehmen wird. Um zu verhindern, daß Teile davon nicht mehr auf dem Bildschirm zu erkennen sind, bietet Turbo Pascal zwei Funktionen an, mit denen Sie vor der Textausgabe feststellen können, wie viele Punkte dafür benötigt werden. Zu diesem Zweck wird beiden Funktionen ein String übergeben. Die TextHeight-Funktion liefert anhand dieses Strings zurück, wie viele Punkte in der Höhe benötigt werden, wenn der Text gemäß der aktuellen Einstellungen für den Zeichensatz, die Textgröße und so weiter angezeigt wird. Die TextWidth-Funktion liefert Ihnen analog dazu das Resultat, wie viele Punkte für die Breite des Textes auf dem Bildschirm benötigt werden.
Für die Erstellung von bewegter Grafik hat Turbo Pascal drei Befehle parat. Ein bestimmter, rechteckiger Ausschnitt des Bildschirms (ein sogenanntes »Image«) kann damit in einen anderen Speicherbereich kopiert und später wieder auf dem Bildschirm dargestellt werden. Das Übertragen eines Bildausschnittes in den Arbeitsspeicher erledigt die GetImage-Prozedur. Als Parameter übergeben Sie die linke obere und rechte untere Ecke des Ausschnitts mit zwei Koordinatenpaaren. Weiterhin erwartet der Befehl die Adresse des Speicherbereichs, in welchem der Bildausschnitt abgelegt werden soll. Dabei muß es sich um ein kontinuierliches, nicht fragmentiertes Datenfeld handeln, das ausreichend groß dimensioniert ist, um alle Bytes des Ausschnitts aufnehmen zu können. Da die Anzahl der Bits, die benötigt werden, um einen Bildpunkt zu speichern, von Grafikkarte zu Grafikkarte und von Auflösung zu Auflösung unterschiedlich ist, bietet Turbo Pascal eine spezielle Funktion an, die den Platzbedarf eines Bildausschnittes berechnet: die ImageSize-Funktion. Sie erwartet als Parameter – analog zu »GetImage« – die Koordinaten der beiden Eckpunkte des Bildausschnitts und liefert die Anzahl der Bytes zurück, die benötigt werden, um den Ausschnitt zu speichern. Dabei berücksichtigt Turbo Pascal automatisch die Anzahl der Bits, die intern benötigt werden, um einen Bildpunkt zu speichern. Zu der Anzahl der Bytes wird 4 addiert, da zwei Worte zu Beginn des Speicherbereichsausschnittes die Breite und Höhe des Images in Punkten enthalten müssen.
Es bietet sich an, Bildausschnitte auf dem Heap zu speichern, also mit dynamischen Variablen zu arbeiten. Falls Sie mit dem Konzept der dynamischen Variablen noch nicht vertraut sind, empfiehlt sich die Lektüre des Grundlagenartikels »Datenstrukturen nutzen unter Turbo Pascal« auf Seite 58.
Für eine dynamische Speicherung der Images auf dem Heap ermitteln Sie zunächst mit der ImageSize-Funktion den Speicherbedarf des gewünschten Bildausschnitts. Die festgestellte Byteanzahl wird über die GetMem-Prozedur auf dem Heap reserviert und die Adresse an eine Zeigervariable übergeben. Die Zeigervariable wird als Parameter in der GetImage-Prozedur verwendet, um die

Name	Code	Fehler
grOK	0	kein Fehler
grNoInitGraph	–1	Grafiktreiber nicht installiert
grNotDetected	–2	Grafikkarte konnte nicht festgestellt werden
grFileNotFound	–3	Grafiktreiber nicht auf Datenträger gefunden
grInvalidDriver	–4	Grafiktreiber paßt nicht zur Grafikkarte
grNoLOadMem	–5	kein Speicherplatz zum Laden des Grafiktreibers
grNoScanMem	–6	kein Speicherplatz für ScanFill
grNoFloodFill	–7	kein Speicherplatz für FloodFill
grFontNotFound	–8	Zeichensatz nicht auf Datenträger gefunden
grNoFontMem	–9	kein Speicherplatz für Zeichensatz
grInvalidMode	–10	Grafikmodus paßt nicht zur Grafikkarte

Tabelle 7. Die möglichen Fehlercodes

DMV-Versandservice

Die preiswerte Alternative für aktive PC-Anwender!

Anwendungen

ConText
Textverarbeitung mit Konzept

Kompromißlose WYSIWYG-Darstellung
Einfache und blitzschnelle Bedienung
Vielseitigkeit, Tempo und Flexibilität

ConText PRO (M)(D) DM 199.-
Professionell mit Spaltenblockbefehlen, Blockgrafikeditor und Rechenfunktionen

ConText EASY (M) DM 99.-
Einfache und praxisgerechte Bedienung, speziell für Einsteiger und Hobbyanwender

Upgrade ConText PC und ConText EASY auf ConText PRO DM 129.-

Zusatzprogramme zu ConText:

ConText-Adreßverwaltung DM 49.-
(auch universell einsetzbar)

ConText Zeicheneditor DM 99.-
Für Context-Bildschirm- und Druckerzeichen

ConText Rechtschreibhilfe DM 69.-
(auch universell einsetzbar)

ERGO 3.0 (D) DM 199.-
Die DOS-Benutzeroberfläche im SAA-Standard, die einfach alles hat!
Inklusive Kalender, Taschenrechner, Optimizer, Editor, TSR-Verwaltung, ASCII-Tabelle, Paßwortstruktur und Menüsystem

DMV-Power-Pack DM 349.-
Die Grundausstattung für Einsteiger und Profis: ERGO 3.0, ConText PRO und ConText Adreßverwaltung. **Sie sparen** DM 98.- gegenüber den Einzelpreisen!

VirusDoktor DM 99.-
Umfassender Schutz vor PC-Computerviren durch fünf Einzelprogramme, u.a. Sperren von Dateien, Virus-Checker mit Prüfsummen, C-MOS-Backup zum Einsteigerpreis

Hyperkey Plus 4.0 DM 99.-
Komfortabler Editor zur Tastaturbelegung für PC/XT/AT mit Makroprozessor inklusive speicherplatzsparender Abspiel-Version

MaskEdit Plus 5.0 (M)(D)
Maskengenerator mit Pulldownmenü-Technik, Blockgrafikeditor und gepackter Abspeicherung. Neben ASCII, ANSI und dBase III+ -Format im Grundpaket werden Treiber für folgende Programmiersprachen angeboten:

Turbo Pascal 3/4/5, Turbo C und Microsoft C, Topspeed-Modula-2, Turbo Basic, Quick Basic

MaskEdit Plus, ein Treiber DM 99.-
MaskEdit Plus, zwei Treiber DM 149.-
MaskEdit Plus, drei Treiber DM 199.-

PCpur-Software

Leicht anzuwenden, einfach zu bedienen und zu anwendergerechten Preisen - die Softwareserie der Zeitschrift PCpur

MultiFont Plus DM 69.-
Generator für eigene Fonts mit residentem Treiber für Matrix-Drucker

zusätzliche Fontware zu Multifont Plus DM 39.-

ASCII-Tiger DM 69.-
Schneller und wendiger Editor mit einstellbarer Bedieneroberfläche

Festplatten-Optimierer DM 69.-
Mehrere Optimierungsstrategien und komfortable Optionen schaffen minimale Zugriffszeiten auf Ihrem Massenspeicher

DOS-Software

RSM (M) DM 69.-
Resident-Software-Manager: menügesteuertes Laden und Löschen residenter Programme

3D-DRAW (D) DM 99.-
Das komfortable CAD-Programm mit Lichtquellensimulation und Animationsmöglichkeit

3D-DRAW Professional DM 199.-
Zusätzlich farbige Gestaltung von Objekten, Integration von Schrift und Verzerrungsfunktionen. Systemvor.: 640 KByte RAM, DOS ab 3.0.

Upgrade von 3D-DRAW auf Professional DM109.-

GW-Basic-Toolboxen je DM 69.-
Komfortable und leicht zu installierende Funktionserweiterungen für GW-Basic in drei Paketen:

GW-Basic-Toolbox: **Benutzeroberfläche**

GW-Basic-Toolbox: **DOS-Funktionen**

GW-Basic-Toolbox: **Hercules-Grafik**

Navigator 4.0 (D) DM 69.-
Zusatzprogramm zur realistischen Simulation von Wind, Wetter und Flugplänen für den Microsoft Flugsimulator III und IV

TOOLBOX-Spezial

Programme und Quellcodes aus der Zeitschrift toolbox zum Superpreis. Dokumentation zum Ausdrucken als ASCII-File auf der Diskette.

TOOLBOX-Spezial (D) je DM 35.-

Spezial II:	**PasComp** - Stand. Pascal Compiler
Spezial IV:	**CALC & Kurvendiskussion**
Spezial V:	**STRUKTO Plus "Pascal"**
Spezial VII:	**Deutsche Silbentrennung**
Spezial IX:	**HPGL-Plotter - Treiber-Toolbox**
Spezial XI:	**MAP, MARK & RELEASE**
Spezial XIII:	**toolbox-Trickkiste Vol. 1**
Spezial XIV:	**toolbox-Trickkiste Vol. 2**
Spezial XV:	**Fitting, numerische Integration und lineare Gleichungssysteme**
Spezial XVI:	**Natürlichsprachl. Schnittstelle**
Spezial XVII:	**EMS-Speicher-Utilities**
Spezial XVIII:	**Nomenklatur von Alkanen**

Software-Entwicklung

BCI-DisAsm DM 199.-
Intelligenter MS-DOS-Disassembler zur Umwandlung von Programmcode in ein dokumentiertes Assemblerlisting

H.A.I - Tech DM 199.-
Hochsprachen-Assembler-Interface mit vordefinierten Makros zur schnellen Anbindung von Assemblerroutinen an die gängigen Hochsprachencompiler.

TopSpeed Modula-2
Programmentwicklung mit Zukunft durch TopSpeed-Modula. Bei uns eingedeutschte Compiler-Software und deutsche Dokumentationen.

TopSpeed Compiler DM 348.-
inklusive Dateiverwaltung und Maskengenerator

TopSpeed TechKit DM 248.-
TopSpeed Debugger DM 248.-
Paketpreis Comp. + Debugger DM 555.-
TopSpeed OS/2-Version DM 698.-

C-Funktionssammlung DM 99.-
für Turbo C und Microsoft C

GEMInterface PC je DM 198.-
Toolkit zur Erstellung eigener GEM-Applikationen. Vers. für T. Pascal 4/5, TopSpeed-Modula-2

DMV-Faktura DM 398.-
Komfortable Auftragsbearbeitung mit Kunden- und Artikelverwaltung. Integriert ist eine Adreßverwaltung mit Etikettendruck sowie ein Editor zur Brieferstellung mit Serienbrieffunktion. Eine Offene-Posten-Liste, Lagerbestand und eine kunden- und artikelbezogene Statistik kann erstellt werden.

DMV-Show-Manager (M)(D) DM 199.-
Mit dem integrierten Maskengenerator oder dem Pickup-Programm erstellte farbige Bildschirmansichten können menügesteuert zu selbstablaufenden Shows zusammengestellt werden. Ein Profiprodukt zum Low-Cost-Preis!

Unterhaltung

Fraktal Generator 3D (D) DM 69.-
Herrliche dreidimensionale Fraktalgrafiken in Sekundenschnelle. Systemanforderungen: EGA/VGA-Karte oder Schneider PC 1512.

PC-Spiele-Sampler DM 49.-
Bimbo II, Bouncer, Jakey, Starvision

POPULOUS + 5 neue Welten DM 69.95
Sie spielen Gott und Herscher

Know PC DM 49.-
Lustiges Frage- und Antwortspiel

Ergänzungsfragen Know PC DM 29.-

Zeitschriften

Soweit noch vorrätig, können Sie alle nicht mehr im Handel befindlichen Ausgaben unserer Zeitschriften DOS International, DOS TEST, toolbox, PCpur, AMIGA DOS und PC Amstrad über den DMV-Verlag beziehen. Beachten Sie bitte die günstigen Paketangebote für ältere Jahrgänge.

PC Amstrad Ultrapack DM 50.-
1/87 - 12/87 plus 3 Ausg. '86 und 2 Sammelord.

PASCAL-Superpack

Jeweils 6 Ausgaben freier Wahl DM 25.-

6 Ausgaben 7'87 bis 12'87 DM 25.-

7 Ausgaben 1'88 bis 8/9'88 DM 28.-

Für Paket-Angebote an DATABOXEN beachten Sie bitte die Werbung in den entspr. Ausgaben.

Sonderhefte

Alle nicht mehr im Handel erhältlichen Sonderhefte können, soweit noch vorrätig, über den DMV-Verlag bezogen werden.

DOS Extra je DM 18.-
Lieferbar sind die Hefte III bis XI

DATABOX DOS Extra III bis XI je DM 35.-

PASCAL Sonderdruck DM 25.-
Das Beste aus PASCAL 12'87 bis 6'87 für T P 3

DATABOX PASCAL Sonderdruck DM 35.-

DMV-Computer-Wissen I DM 18.-
Alles über Basic mit vielen nützl. Programmen

DATABOX DMV-Computer-Wissen I DM 35.-

Bücher

DOS-Taschenbücher je DM 29.80
Die beliebtesten Serien aus DOS International und DOS Extra jetzt als Taschenbücher! Darunter auch das vergriffene DOS Extra 1

Jeder Band ca. 270 Seiten, Paperback, gebunden, mit zahlreichen Abbildungen, Tabellen und Listings. Ein Index erleichtert das Auffinden wichtiger Funktionen.

Band 1, **Von Assembler bis MS-DOS**

Band 2, **Word 3.0 und 4.0**

Band 3, **Der PC glasklar**

Band 4, **DOS Praxis**

In Vorbereitung:

Band 5, **Die dBase-Werkstatt**

Alle Listings, Beispiele und Programme zu den DOS-Taschenbüchern Band 1, Band 3 und Band 4 erhalten Sie auch auf Datenträger zum günstigen DATABOX-Preis.

DATABOX,
Band 1, 3, und 4 je DM 24.-

Die Basic-2-Toolbox DM 49.-
Exklusiv v. uns für Amstrad/Schneider-Anwender

Diskette mit Listings zum Buch DM 29.-

Sammelmappen

Ordnung und Übersicht behalten Sie mit den praktischen Sammelschubern für unsere Zeitschriften. Für DOS International, DOS TEST, toolbox, PCpur, AMIGA DOS und PC Amstrad

2 Stück Sammelordner DM 15.80

Neu in unserem Angebot!!

DOS-Spielekiste

POPULOUS
Computer Simulationsspiel

Versandbedingungen:

Unabhängig von der Anzahl der bestellten Produkte, Programme und Bücher berechnen wir bei jeder Sendung für das Inland DM 4.-, für das Ausland DM 6.- Porto und Verpackung.

Alle PC-Software-Produkte sind wahlweise auf 5 1/4-Zoll- und 3 1/2-Zoll-Disketten erhältlich. Für Atari ST und Amiga liefern wir auf 3 1/2-Zoll-Disketten. Bitte beachten Sie die jeweiligen Bestellnummern. Benutzen Sie bitte die der Ausgabe beigefügte Bestellkarte.

Systemvoraussetzungen:

Soweit nicht anders vermerkt, sind alle Softwareprodukte auf MS-DOS-Computern ab DOS 2.1, minimal 256 kByte RAM und den Grafikkarten CGA, MGA, EGA, VGA und Hercules lauffähig.

Legende:

(M) = inklusive universell verwendbarem Maus-Cursortasten-Emulator.

(D) = Demonstrationsdiskette für DM 5.- erhältlich.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverl

Adresse zu bestimmen, an die der Bildausschnitt kopiert werden soll. Nach diesem Vorgang ist der Bildausschnitt auf dem Heap gespeichert. Sie können Bildausschnitte statt auf dem Heap auch im Speicherbereich von Variablenfeldern ablegen. Voraussetzung ist immer, daß der Speicherbereich ausreichend groß ist, um alle Daten aufzunehmen.
Zum Zurückschreiben eines Bildausschnittes auf den Bildschirm dient die PutImage-Prozedur. Ihr übergeben Sie die Koordinaten der linken oberen Ecke eines Bildausschnittes und einen Zeiger auf den Bildausschnitt. Als Resultat wird der gespeicherte Bildausschnitt an der festgelegten Stelle dargestellt.

Für die dynamische Speicherung von Images läßt sich der Heap verwenden

Das Programm »sprite.pas« in Listing 5 demonstriert den Umgang mit den Image-Befehlen von Turbo Pascal. Es werden farbige Rechtecke in jeweils eigenen Images dynamisch auf dem Heap gespeichert und nach dem Zufallsprinzip auf dem Bildschirm dargestellt, bis eine Taste gedrückt wird. Die Anzahl der Rechtecke, die erzeugt werden, hängt von der Farbfähigkeit der Grafikkarte ab. Für EGA-, MCGA- und VGA-Karten werden 16 verschiedenfarbige Rechtecke angelegt, für CGA-Karten vier und beim Betrieb von Hercules-Karten zwei Rechtecke. Die Größe der Rechtecke wird über die Konstante »seite« im Quelltext gesteuert, die Farbanzahl über die Konstante »farben«. Bild 3 zeigt den Bildschirm nach dem Start von »sprite.pas«.
Es ist wichtig für die reibungslose Arbeit mit den gespeicherten Bildausschnitten, auf dem Heap belegten Speicherplatz nach Abschluß der Grafikdarstellung wieder freizugeben.
Beim Umgang mit dem Grafikbefehlssatz von Turbo Pascal können unterschiedliche Fehler auftreten. Ursachen solcher Fehler können falsch übergebene Koordinaten, fehlerhafte Farbwerte und so weiter sein. Turbo Pascal stellt Ihnen daher die Funktion »GraphResult« zur Verfügung, deren Rückgabewert nach jedem Aufruf einer Grafikfunktion oder -prozedur auf den neuesten Stand gebracht wird. Liefert die Funktion den Wert 0, hat der zuletzt verwendete Befehl fehlerfrei gearbeitet. Bei einem anderen Wert ist ein Fehler aufgetreten. Tabelle 7 zeigt die möglichen Fehlercodes. Mit der Funktion »GraphErrorMsg« können Sie zu einem Fehlercode die passende Fehlermeldung als String ermitteln. Das ist bei der Programmierung von Grafikprogrammen nützlich, bei denen nicht alle Fehler abgefangen werden können.

Ist die Arbeit im Grafikmodus beendet, kann der Textmodus wieder eingestellt werden. Dazu gibt es zwei unterschiedliche Verfahren. Die Prozedur »RestoreCrtMode« setzt den vor der Initialisierung des Grafiktreibers aktuellen Textmodus. Dabei können Sie durch die Verwendung des SetGraphMode-Befehls erneut auf den Grafikmodus umschalten, ohne den zugehörigen Treiber neu initialisieren zu müssen. Im Gegensatz dazu setzt die Prozedur »CloseGraph« den Textmodus zurück, entfernt aber gleichzeitig den Grafiktreiber aus dem Speicher. Eine erneute Initialisierung des Grafikmodus kann nur durchgeführt werden, indem Sie den Treiber über das InitGraph-Kommando erneut installieren.

(Martin Althaus/ks)

```
[TX2]    1: program scrinfo;
[9Ø3]    2: {-------------------------
[742]    3:       scrinfo.pas
[T81]    4:       -----------
[4G4]    5:  Ananlysiert die aktuellen
[754]    6:  Parameter des Grafikbild-
[R44]    7:  schirms und gibt sie auf
[X73]    8:  dem Bildschirm aus.
[VT3]    9:  von Martin Althaus 3/9Ø
[4Ø4]   1Ø:  (C) 199Ø Redaktionsbüro
[L83]   11:      Everts und Hagedorn
[6R2]   12:  -------------------------
[U7Ø]   13: }
[9P2]   14:  uses crt,graph;
[RNØ]   15: var
[1X3]   16:  treiber,modus:integer;
[ZM3]   17:  palette:palettetype;
[3U4]   18:  fuellung:fillsettingstype;
[2A4]   19:  linie:linesettingstype;
[GA4]   2Ø:  hix,hiy,maxmode,aktmode,
[244]   21:  aktfarbe,akthfarbe:word;
[LZ1]   22:  index:byte;
[YT2]   23:  nmodus: string;
[MUØ]   24: begin
[WU2]   25:  treiber:=detect;
[5C4]   26:                  {automatische Erkennung}
[UE1]   27:  modus:=Ø;
[V53]   28:                        {Dummy-Parameter}
[K64]   29:  initgraph(treiber,modus,
[ØB2]   3Ø:            'c:\turbo4\');
[HG5]   31:       {Treiber laden und Modus setzen}
[7A2]   32:  hix:=getmaxx;
[CA2]   33:  hiy:=getmaxy;
[PR5]   34:        {Maximale Koordinaten auslesen}
[PA3]   35:  getpalette(palette);
[6G3]   36:                        {Palette auslesen}
[WH3]   37:  maxmode:=getmaxmode;
[MØ5]   38:              {Maximaler Wert für Modus}
[TV3]   39:  aktmode:=getgraphmode;
[F65]   4Ø:          {Nummer des aktuellen Modus}
[L25]   41:  nmodus:=getmodename(aktmode);
[MM4]   42:              {Name des aktuellen Modus}
[MR4]   43:  getfillsettings(fuellung);
[Y24]   44:                  {Parameter zum Füllen}
[564]   45:  getlinesettings(linie);
[AN5]   46:       {Parameter zum Linien-Zeichnen}
[3E3]   47:  aktfarbe:=getcolor;
[GC4]   48:                  {aktuelle Zeichenfarbe}
```

```
[7V3]   49:  akthfarbe:=getbkcolor;
[W45]   5Ø:            {aktuelle Hintergrundfarbe}
[4V2]   51:  restorecrtmode;
[ZG5]   52:           {Zum Textmodus zurückkehren}
[XF1]   53:  writeln;
[XT4]   54:  write('Die Analyse des Grafik');
[ØØ5]   55:  write('modus hat folgende Er');
[ZR4]   56:  write('gebnisse an Licht ge');
[493]   57:  writeln('bracht:');
[XF1]   58:  writeln;
[MA5]   59:  write('Beim Initialisieren des');
[D65]   6Ø:  write(' Grafikmodus wird der');
[EV4]   61:  write(' Modus ',nmodus,' ver');
[993]   62:  writeln('wendet.');
[9Z4]   63:  write('Es wird Modus ',aktmode);
[U65]   64:  write(' betrieben, maximal mög');
[145]   65:  writeln('lich ist ',maxmode,'.');
[XF1]   66:  writeln;
[W45]   67:  write('Es lassen sich horizon');
[8X4]   68:  write('tal ',hix+1,' und verti');
[CP4]   69:  write('kal ',hiy+1,' Punkte ');
[YY3]   7Ø:  writeln('darstellen.');
[XF1]   71:  writeln;
[NB5]   72:  write('Die aktuelle Zeichenfar');
[W75]   73:  write('be ist ',aktfarbe,', der ');
[WM5]   74:  write('Hintergund wird in Farbe ');
[916]   75:  writeln(akthfarbe,' dargestellt.');
[XN4]   76:  write('Die Palette enthält ');
[WT5]   77:  write(palette.size,' Farben, und');
[C43]   78:  writeln(' zwar: ');
[ØG5]   79:  for index:=Ø to palette.size do
[MR5]   8Ø:   write(palette.colors[index],', ');
[YS2]   81:  writeln(#8,#8,'.');
[XF1]   82:  writeln;
[LR5]   83:  write('Zur Darstellung von Linien');
[1S2]   84:  write(' werden ');
[514]   85:  case linie.linestyle of
[E95]   86:   solidln: write('durchgehende ');
[3A5]   87:   dottedln: write('gepunktete ');
[NE5]   88:   centerln: write('Strich-Punkt-',
[6U1]   89:                 'Strich-');
[PC5]   9Ø:   dashedln: write('gestrichelte ');
[ZNØ]   91:  end;
[PD5]   92:  writeln('Linien verwendet, die');
[UT1]   93:  write('in ');
[EZ3]   94:  case linie.thickness of
[PS4]   95:   normwidth: write('normaler');
```

Listing 1. »scrinfo.pas« schlüsselt die Parameter der Grafikdarstellung auf

```
[K95]   96:   thickwidth: write('doppelter');
[ZNØ]   97:  end;
[2Y4]   98:  writeln(' Dicke erscheinen.');
[XF1]   99:  writeln;
[G35]  1ØØ:  write('Als Muster zum Füllen ');
[CD4]  1Ø1:  write('von Flächen wird ');
[255]  1Ø2:  write(fuellung.pattern,' mit ');
[XW2]  1Ø3:  write('der Farbe ');
[E55]  1Ø4:  write(fuellung.color,' verwen');
[WG2]  1Ø5:  writeln('det.');
[VL3]  1Ø6:                       {Ergbnisse ausgeben}
[KNØ]  1Ø7: end.
```

Listing 1. »scrinfo.pas« schlüsselt die Parameter der Grafikdarstellung auf

```
[813]    1: program paletten;
[A73]    2: {--------------------------
[142]    3:         paletten.pas
[6A1]    4:         -----------
[3X4]    5:  demonstriert den Umgang mit
[R34]    6:  Farb-Paletten anhand von
[U64]    7:  Ellipsen, die auf dem Bild-
[H64]    8:  schirm gezeichnet werden.
[VT3]    9:  von Martin Althaus 3/9Ø
[4Ø4]   1Ø:  (C) 199Ø Redaktionsbüro
[L83]   11:      Everts und Hagedorn
[113]   12:  --------------------------
[U7Ø]   13: }
[9P2]   14:  uses crt,graph;
[6XØ]   15: const
[3P1]   16:  anzahl=5Ø;
[GW3]   17:                   {Anzahl der Ellipsen}
[CX1]   18:  panzahl=16;
[CW3]   19:                   {Anzahl der Paletten}
[RNØ]   2Ø: var
[1X3]   21:  treiber,modus:integer;
[DW4]   22:  palette:array[Ø..(panzahl-1)]
[9M2]   23:           of palettetype;
[ØH4]   24:  hix,hiy,mx,my,rx,ry,ox,oy,
[ZC3]   25:  farbzahl,oben:word;
[LC4]   26:  index,pindex,toggle:byte;
[4W1]   27:  lese: char;
[MUØ]   28: begin
[WU2]   29:  treiber:=detect;
[5C4]   3Ø:                {automatische Erkennung}
[UE1]   31:  modus:=Ø;
[V53]   32:                        {Dummy-Parameter}
[K64]   33:  initgraph(treiber,modus,
[ØB2]   34:            'c:\turbo4\');
[HG5]   35:       {Treiber laden und Modus setzen}
[7A2]   36:   hix:=getmaxx;
[CA2]   37:   hiy:=getmaxy;
[PR5]   38:        {Maximale Koordinaten auslesen}
[2D2]   39:  mx:=hix div 2;
[9D2]   4Ø:  my:=hiy div 2;
[VN4]   41:              {Mittelpunkt des Systems}
[ZA1]   42:  rx:=mx;
[5B1]   43:  ry:=my;
[D45]   44:           {Ellipsen-Radien festlegen}
[PE3]   45:  ox:=mx div (anzahl+1);
[WE3]   46:  oy:=my div (anzahl+1);
[BE4]   47:              {Schrittweite festlegen}
[FF4]   48:  farbzahl:=getpalettesize;
[S54]   49:  for index:=1 to anzahl do
[MUØ]   5Ø:  begin
[HU4]   51:   setcolor(index mod farbzahl);
[T54]   52:                {Lege Zeichenfarbe fest}
[S94]   53:   ellipse(mx,my,Ø,36Ø,rx,ry);
[DF3]   54:                      {Ellipse zeichnen}
[1R1]   55:   rx:=rx-ox;
[9R1]   56:   ry:=ry-oy;
[EY3]   57:                   {Ellipse verkleinern}
[ZNØ]   58:  end;
[M14]   59:  getpalette(palette[Ø]);
[6G3]   6Ø:                      {Palette auslesen}
[V64]   61:  for index:=1 to panzahl-1
[B85]   62:   do palette[index]:=palette[Ø];
[MX3]   63:                   {Palette übernehmen}
[4N4]   64:  for pindex:=1 to panzahl-1 do
[MUØ]   65:  begin
[2U4]   66:   for index:=1 to maxcolors do
[AW5]   67:   palette[pindex].colors[index]:=
[ZZ5]   68:   palette[pindex-1].colors[index+1];
[EK5]   69:          {Erstelle veränderte Palette}
[ZN6]   7Ø:   palette[pindex].colors[maxcolors]:=
[DØ5]   71:   palette[pindex-1].colors[Ø];
[4N5]   72:        {letzte Farbe wird erste Farbe}
[ZNØ]   73:  end;
[RM1]   74:  toggle:=1;
[P31]   75:  repeat
[FP5]   76:   setallpalette(palette[toggle]);
[XU4]   77:         {Palette schnell wechseln}
[HV4]   78:   toggle:=(toggle+1) mod panzahl
```

```
[K13]   79:  until keypressed;
[N26]   8Ø:  repeat lese:=readkey; until lese<>#Ø;
[MA3]   81:                        {Auf Taste warten}
[RK4]   82:  setallpalette(palette[Ø]);
[WZ4]   83:              {Palette wiederherstellen}
[4V2]   84:  restorecrtmode;
[ZG5]   85:           {Zum Textmodus zurückkehren}
[KNØ]   86: end.
```

Listing 2. Interessante Farbeffekte durch die Veränderung der Farbpalette demonstriert »paletten.pas«

```
[PR2]    1: program balken;
[4A3]    2: {--------------------------
[TN1]    3:         balken.pas
[531]    4:         ----------
[3X4]    5:  demonstriert den Umgang mit
[ØM4]    6:  dem BAR3D-Befehl anhand von
[KH4]    7:  Werten, die in einem Balken-
[EX4]    8:  diagramm ausgewertet werden.
[VT3]    9:  von Martin Althaus 3/9Ø
[4Ø4]   1Ø:  (C) 199Ø Redaktionsbüro
[L83]   11:      Everts und Hagedorn
[E23]   12:  --------------------------
[U7Ø]   13: }
[9P2]   14:  uses crt,graph;
[6XØ]   15: const
[X52]   16:  maxanzahl=5Ø;
[GW3]   17:                   {Anzahl der Ellipsen}
[RNØ]   18: var
[9M5]   19:  wert: array[1..maxanzahl] of real;
[K65]   2Ø:           {Zur Speicherung der Werte}
[1X3]   21:  treiber,modus:integer;
[VR5]   22:  hix,hiy,anzahl,punktex,punktexb,
[W94]   23:  punktext,aktx,akty:word;
[LZ1]   24:  index:byte;
[9C4]   25:  yfakt,werte,max,min:real;
[4W1]   26:  lese: char;
[LR4]   27:              {Zur Abfrage der Tastatur}
[MUØ]   28: begin
[DB1]   29:  clrscr;
[P31]   3Ø: repeat
[8U5]   31:  write('Wieviele Werte sollen dar');
[LK4]   32:  write('gestellt werden? ');
[LG2]   33:  readln(anzahl);
[335]   34:            {Anzahl der Werte einlesen}
[VL5]   35:  if (anzahl>maxanzahl) or (anzahl<3)
[BH6]   36:  {Mindestens 3 und höchstens Maxzahl}
[FW5]   37:  then writeln('Zu viele/zu wenige');
[U44]   38:  until (anzahl<=maxanzahl)
[7E2]   39:        or (anzahl>=3);
[XF1]   4Ø:  writeln;
[S54]   41:  for index:=1 to anzahl do
[WU3]   42:                 {Alle Werte einlesen}
[MUØ]   43:  begin
[P31]   44:   repeat
[2F4]   45:   write(index,'.ter Wert: ');
[PE3]   46:   readln(wert[index]);
[YK3]   47:   until wert[index]>Ø;
[U65]   48:         {Nur positive Werte erlaubt}
[ZNØ]   49:  end;
[C92]   5Ø:  max:=wert[1];
[W82]   51:  min:=wert[1];
[7C5]   52:       {Max. und Min. auf ersten Wert}
[S54]   53:  for index:=1 to anzahl do
[GR5]   54:    {In Schleife Max. und Min. suchen}
[MUØ]   55:  begin
[D63]   56:   if wert[index]>max
[5Y3]   57:      then max:=wert[index];
[KD4]   58:             {Neues Maximum gefunden?}
[R53]   59:   if wert[index]<min
[VY3]   6Ø:      then min:=wert[index];
[D74]   61:              {Neues Minmum gefunden?}
[ZNØ]   62:  end;
[WU2]   63:  treiber:=detect;
[5C4]   64:               {automatische Erkennung}
[UE1]   65:  modus:=Ø;
[V53]   66:                       {Dummy-Parameter}
[K64]   67:  initgraph(treiber,modus,
[ØB2]   68:            'c:\turbo4\');
[HG5]   69:       {Treiber laden und Modus setzen}
[7A2]   7Ø:  hix:=getmaxx;
[CA2]   71:  hiy:=getmaxy;
[PR5]   72:        {Maximale Koordinaten auslesen}
[174]   73:  punktex:=hix div anzahl;
[UA5]   74:         {Anzahl der Punkte pro Balken}
[AR5]   75:  punktexb:=punktex-(punktex div 5);
[WE4]   76:              {Abstand zwischen Balken}
[Y74]   77:  punktext:=punktex div 1Ø;
[ZG3]   78:                    {Tiefe des Balkens}
[L24]   79:  werte:=abs(min)+abs(max);
[8R3]   8Ø:                  {Differenz der Werte}
[ØX2]   81:  yfakt:=hiy/werte;
[RM3]   82:                  {Y-Punkte pro 1-Wert}
```

```
[N91]    83:  aktx:=1;
[393]    84:                          {Startwert für X}
[UY1]    85:  setcolor(1);
[F64]    86:                  {Blau als aktuelle Farbe}
[FV4]    87:  setfillstyle(solidfill,red);
[K55]    88:           {Schraffierte, rote Flächen}
[S54]    89:  for index:=1 to anzahl do
[FW5]    9Ø:     {Schleife zum Zeichnen der Balken}
[MUØ]    91:  begin
[943]    92:                          {Balken zeichnen}
[AL4]    93:   bar3d(aktx,hiy-trunc(yfakt*
[EZ4]    94:         wert[index]),aktx+punktexb,
[AH3]    95:         hiy,punktext,true);
[3E3]    96:   aktx:=aktx+punktex;
[2Y4]    97:           {X um Balkenabstand erhöhen}
[ZNØ]    98:  end;
[Y44]    99:  repeat until keypressed;
[N26]   1ØØ:  repeat lese:=readkey; until lese<>#Ø;
[MA3]   1Ø1:                          {Auf Taste warten}
[4V2]   1Ø2:  restorecrtmode;
[ZG5]   1Ø3:           {Zum Textmodus zurückkehren}
[KNØ]   1Ø4: end.
```

Listing 3. »balken.pas« stellt Balkendiagramme mit Hilfe der Pascal-Prozedur »Bar3D« dar

```
[UH2]     1: program texte;
[A73]     2: {--------------------------
[6L1]     3:         texte.pas
[YYØ]     4:         ---------
[244]     5:  demonstriert die unter-
[UP4]     6:  schiedlichen Zeichensätze,
[AL4]     7:  die Turbo Pascal im Grafik-
[L45]     8:  modus zur Verfügung stellt.
[VT3]     9:  von Martin Althaus 3/9Ø
[4Ø4]    1Ø:  (C) 199Ø Redaktionsbüro
[L83]    11:      Everts und Hagedorn
[113]    12:  --------------------------
[U7Ø]    13: }
[9P2]    14:  uses crt,graph;
[RNØ]    15: var
[1X3]    16:  treiber,modus:integer;
[T34]    17:                       {Für den Grafikmodus}
[VC3]    18:  index,zaehler:byte;
[S64]    19:                    {Zur Schleifenbildung}
[CL3]    2Ø:  hix,hiy,ywert:word;
[BR1]    21:  lese:char;
[LR4]    22:                {Zur Abfrage der Tastatur}
[RW2]    23:  ausg,ww:string;
[7W3]    24:                       {Temporäre Strings}
[MUØ]    25: begin
[WU2]    26:  treiber:=detect;
[5C4]    27:                  {automatische Erkennung}
[UE1]    28:  modus:=Ø;
[V53]    29:                          {Dummy-Parameter}
[K64]    3Ø:  initgraph(treiber,modus,
[ØB2]    31:                'c:\turbo4\');
[HG5]    32:       {Treiber laden und Modus setzen}
[7A2]    33:  hix:=getmaxx;
[CA2]    34:  hiy:=getmaxy;
[PR5]    35:        {Maximale Koordinaten auslesen}
[CN3]    36:  ausg:='Zeichensatz: ';
[FV4]    37:             {Textkonstante definieren}
[L73]    38:  for index:=Ø to 4 do
[AP5]    39:         {Stelle alle Zeichensätze dar}
[MUØ]    4Ø:  begin
[GG3]    41:   setgraphmode(modus);
[YS3]    42:                        {Grafikmodus setzen}
[534]    43:   settextstyle(index,Ø,5);
[HT5]    44:          {Zeichensatz für Überschrift}
[PD3]    45:   settextjustify(1,2);
[U84]    46:                  {Texte waag. zentrieren}
[DE2]    47:   str(index,ww);
[CG5]    48:         {INDEX in String WW umwandeln}
[P65]    49:   outtextxy(hix div 2,1Ø,ausg+ww);
[EM3]    5Ø:                      {Kopfzeile ausgeben}
[5F3]    51:   for zaehler:=1 to 5 do
[9H6]    52:         {Fünf unterschiedliche Größen}
[MUØ]    53:   begin
[AN1]    54:    ywert:=2Ø;
[A83]    55:                          {Bei Y=2Ø starten}
[1F5]    56:    settextstyle(index,Ø,zaehler*2);
[D74]    57:                    {Textgröße Festlegen}
[ØK4]    58:    ywert:=ywert+2Ø+zaehler*4Ø;
[YH4]    59:                   {Koordinate ausrechnen}
[W23]    6Ø:    str(zaehler*2,ww);
[1D5]    61:             {Größe in String umwandeln}
[ØE4]    62:    outtextxy(hix div 2,ywert,
[VF2]    63:              'Groesse: '+ww);
[A83]    64:                           {Größe ausgeben}
[ZNØ]    65:   end;
[Y44]    66:   repeat until keypressed;
[N26]    67:   repeat lese:=readkey; until lese<>#Ø;
[MA3]    68:                          {Auf Taste warten}
[ZNØ]    69:  end;
[4V2]    7Ø:  restorecrtmode;
[ZG5]    71:           {Zum Textmodus zurückkehren}
[KNØ]    72: end.
```

Listing 4. Das Programm »texte.pas« demonstriert die vielfältigen Möglichkeiten zur Textdarstellung im Grafikmodus

```
[YU2]     1: program sprite;
[4A3]     2: {--------------------------
[1T1]     3:         sprite.pas
[531]     4:         ----------
[LW4]     5:  Stellt die Möglichkeiten von
[BL4]     6:  Turbo Pascal dar, Teile des
[1B4]     7:  Grafikbildschirms als Bit-
[6F4]     8:  muster auf dem Heap zu spei-
[AZØ]     9:  chern.
[VT3]    1Ø:  von Martin Althaus 3/9Ø
[4Ø4]    11:  (C) 199Ø Redaktionsbüro
[L83]    12:      Everts und Hagedorn
[113]    13:  --------------------------
[U7Ø]    14: }
[9P2]    15:  uses crt,graph;
[6XØ]    16: const
[8D1]    17:  seite=5Ø;
[9V3]    18:                       {Länge des Rechtecks}
[FM1]    19:  farben=16;
[T66]    2Ø:    {für EGA/VGA:16, CGA:4, Hercules:2}
[RNØ]    21: var
[1X3]    22:  treiber,modus:integer;
[T34]    23:                       {Für den Grafikmodus}
[FD2]    24:  index,mi:byte;
[S64]    25:                    {Zur Schleifenbildung}
[ØW4]    26:  hix,hiy,mx,my,groesse:word;
[AY4]    27:              {Parameter des Bildschirms}
[8A6]    28:  adresse:array[1..farben] of pointer;
[BR1]    29:  lese:char;
[LR4]    3Ø:                {Zur Abfrage der Tastatur}
[MUØ]    31: begin
[WU2]    32:  treiber:=detect;
[5C4]    33:                  {automatische Erkennung}
[UE1]    34:  modus:=Ø;
[V53]    35:                          {Dummy-Parameter}
[K64]    36:  initgraph(treiber,modus,
[ØB2]    37:                'c:\turbo4\');
[HG5]    38:       {Treiber laden und Modus setzen}
[7A2]    39:  hix:=getmaxx;
[CA2]    4Ø:  hiy:=getmaxy;
[ZH5]    41:         {Maximal-Koordinaten auslesen}
[2D2]    42:  mx:=hix div 2;
[9D2]    43:  my:=hiy div 2;
[686]    44:      {Mitelpunkt-Koordinaten auslesen}
[W34]    45:  for index:=1 to farben do
[5P6]    46:    {Für jede Farbe ein eigenes Sprite}
[MUØ]    47:  begin
[GT2]    48:   setcolor(index);
[AB4]    49:                    {Zeichenfarbe festlegen}
[B26]    5Ø:   rectangle(mx,my,mx+seite,my+seite);
[UR3]    51:                       {Rechteck zeichnen}
[DE5]    52:   setfillstyle(solidfill,index+1);
[F14]    53:                     {Füllfarbe festlegen}
[6Z3]    54:   floodfill(mx+seite div 2,
[AE3]    55:                my+seite div 2,index);
[653]    56:                          {Rechteck füllen}
[R94]    57:   groesse:=imagesize(mx,my,
[7A3]    58:              mx+seite,my+seite);
[444]    59:                       {Größe feststellen}
[SE5]    6Ø:   getmem(adresse[index],groesse);
[FH5]    61:       {Platz auf dem Heap reservieren}
[954]    62:   getimage(mx,my,mx+seite,
[3N4]    63:              my+seite,adresse[index]^);
[FF5]    64:         {Sprite auf dem Heap spechern}
[ZNØ]    65:  end;
[1T1]    66:  randomize;
[ZN5]    67:         {Zufallsgenerator einschalten}
[P31]    68:  repeat
[NN3]    69:   mx:=random(hix-seite);
[VN3]    7Ø:   my:=random(hiy-seite);
[LY5]    71:       {Koordinaten für Rechteck laden}
[R83]    72:   mi:=random(farben)+1;
[D15]    73:            {Nummer des Rechtecks laden}
[F55]    74:   putimage(mx,my,adresse[mi]^,
[812]    75:                normalput);
[383]    76:                          {Sprite zeichnen}
[K13]    77:  until keypressed;
[WW3]    78:                   {bis zum Tastendruck}
[N26]    79:  repeat lese:=readkey; until lese<>#Ø;
[MA3]    8Ø:                          {Auf Taste warten}
[4V2]    81:  restorecrtmode;
[ZG5]    82:             {Zum Textmodus zurückkehren}
[W34]    83:  for index:=1 to farben do
[4L5]    84:   freemem(adresse[index],groesse);
[GL3]    85:                       {Speicher freigeben}
[KNØ]    86: end.
```

Listing 5. Das Programm »sprite.pas« erzeugt Grafik-Images unter Turbo Pascal

Bringen Sie dem PC die Flötentöne bei

Töne in ihren verschiedenen Variationen finden auf dem PC als Warnsignale oder Untermalungen für Spiele Verwendung. Es ist gar nicht schwer, Klangbilder beziehungsweise bekannte oder sogar eigene Kompositionen ertönen zu lassen. Wie Sie für die Tonerzeugung die Register des Timer-Bausteins 8253 verwenden können, zeigt Ihnen der folgende Artikel.

Die Tonerzeugung gehört nicht zu den Stärken des PC, da das Betriebssystem den Mehrkanalton und die Lautstärkeregelung nicht unterstützt. Außerdem ist die Hardware des Rechners nicht geeignet, um komplexere Klänge zu erzeugen. Der eingebaute Lautsprecher eignet sich aber immerhin dazu, einfache Melodien und Geräusche erklingen zu lassen.
Viele Programmiersprachen enthalten in ihrem Befehlssatz Kommandos, mit denen sich über den Lautsprecher Töne ausgeben lassen. In Turbo Pascal ist die Tonerzeugung aber auf das einfache Kommando »Sound« beschränkt. C und Assembler ignorieren den Lautsprecher sogar, so daß Sie sich selbst Routinen zur Tonausgabe schreiben müssen. Wenn Sie verstanden haben, wie ein Ton prinzipiell erzeugt wird und welche Funktion die PC-Bausteine bei der Tonerzeugung übernehmen, ist diese Aufgabe leicht zu lösen.
Töne und Geräusche entstehen, indem ein vibrierendes Medium (beispielsweise eine Lautsprechermembran) Luftteilchen in Bewegung versetzt. Bewegt sich die Membran in eine Richtung, dann wird die davorliegende Luft komprimiert. Bewegt sie sich in die Ausgangsposition zurück, dann reißt der Druckabfall die Luftteilchen auseinander. Die dabei entstehende Luftbewegung nennt man Schwingung. Eine Schwingung hat die Kollision vieler Luftpartikel zur Folge. Der Zusammenstoß der Luftpartikel setzt sich als Kettenreaktion in alle Richtungen fort. Diese Bewegung bezeichnet man als Welle.
Die Lautsprechermembran des PC wird durch elektrische Impulse in Bewegung versetzt. Der Computer gibt entweder eine Spannung an den Lautsprecher ab oder nicht. Das bedeutet, daß sich die Membran aktiv vorwärtsbewegt oder ruht (beziehungsweise in die Ausgangsposition zurückkehrt). Die Anzahl der Schwingungen, die die Membran pro Zeiteinheit ausführt, nennt man Frequenz. Das physikalische Maß für die Frequenz ist die Einheit »Hertz« (Hz), die die Anzahl der Schwingungen pro Sekunde bezeichnet. Je mehr Schwingungen in einer bestimmten Zeiteinheit erzeugt werden, desto höher ist die dabei entstehende Frequenz.
Die von einer Lautsprechermembran ausgesandten, hörbaren Schwingungen bezeichnet man als Töne. Prinzipiell können Töne die verschiedensten Frequenzen besitzen. Für wohlklingende Musik ist das jedoch nicht ausreichend. Die Frequenzen für die Töne einer musikalischen Oktave sind daher genau festgelegt. Tabelle 1 zeigt die Frequenzen für alle Töne der unteren acht Oktaven.
Die Töne unterhalb der Oktave, die mit der Frequenz 16,35 Hz beginnt, sind nicht von Interesse, weil der Mensch erst Frequenzen ab 16 Hz als Töne wahrnimmt. Das menschliche Ohr kann alle Frequenzen unterhalb dieser Grenze in Einzelimpulse auflösen und hört deshalb keinen zusammenhängenden Ton mehr.
Einigen Frequenzen sind, den vorangegangenen Ausführungen gemäß, entsprechende Tonbezeichnungen zugeordnet.
Will man beispielsweise ein »eingestrichenes G« ertönen lassen, dann muß die Lautsprechermembran, den Angaben in Tabelle 1 zufolge, 392mal in einer Sekunde hin- und herschwingen. Das »eingestrichene G« entspricht einem Ton von 392 Hz.
Der Bezugspunkt für alle Töne in Tabelle 1 ist der Ton A in der fünften Oktave mit einer Frequenz von 440 Hz. Diesen Ton bezeichnet man auch als »Kammerton A«.
Da die Frequenz des Tons A in der benachbarten Oktave stets doppelt oder halb so groß ist wie die ursprüngliche Frequenz und da eine Oktave aus 12 Halbtönen besteht, ist der Unterschied zwischen zwei benachbarten Halbtönen stets ein Faktor mit dem Wert der zwölften Wurzel aus 2. Um von der Frequenz eines Halbtons ausgehend die Frequenz des unmittelbar folgenden Halbtons (ungefähr) zu berechnen, multiplizieren Sie einfach die Frequenz des ersten Halbtons mit 1,0594631.
Im PC sind mehrere Bausteine für die Tonerzeugung über den eingebauten Lautsprecher zuständig. Sehr wichtig ist der Timer-Baustein PIT 8253 (PIT für »Peripheral Interval Timer«). Der 8253 erfüllt im PC viele Aufgaben, zu denen die Ansteuerung des Tonausgangs, die Überwachung der Echtzeituhr, der RAM-Refresh und der Zugriff auf Kassetten gehören. Im AT arbeitet eine erweiterte Version des 8253, der 8254, der sich aber grundsätzlich nur durch die verbesserte Architektur von seinem Vorgängermodell unterscheidet.

Der Timer-Baustein 8253 übernimmt die Tonerzeugung

Der Timer-Baustein verwaltet drei Kanäle, die unterschiedliche Aufgabengebiete abdecken. Der Kanal 0 (Portadresse 40hex) steuert die Systemuhr an, während Kanal 1 (Portadresse 41hex) benutzt wird, um dem DMAC den Takt vorzugeben, mit dem die dynamischen RAMs aufgefrischt werden müssen. Kanal 2 (Portadresse 42hex) ist mit dem Lautsprecherausgang verbunden und aus diesem Grund für die Sound-Programmierung von besonderem Interesse.
Die Hauptaufgabe des Timers ist es zu zählen. Die einzelnen Divisoren der Kanäle werden im Systemtakt des Bausteins fortlaufend vermindert. Ist der Wert 0 erreicht, dann wird das Ausgangssignal ausgegeben, und das Zählerregister erhält wieder den Initialisierungswert. Sie fragen sich bestimmt, warum man für diese Aufgabe extra einen Baustein benutzt und das nicht über Software erledigt. Dieses Verfahren hat einen wichtigen Grund: Da der PC mit Interrupts arbeitet, kann es sein, daß eine programmgesteuerte Zählroutine unterbrochen wird. Die Synchronisation mit dem Mikroprozessor und der Hardware ist in diesem Fall ungenau und somit fast undurchführbar. Ein zusätzliches IC, das die Timer-Aufgaben übernimmt, ist eine elegante Lösung des Problems. Komplikationen mit anderen Bausteinen sind so gut wie ausgeschlossen.
Neben den Registern für die drei Timer-Kanäle bestimmt ein Steuerregister mit der Portadresse 43hex, dessen Belegung in Tabelle 2 aufgeführt ist, die Funktionsweise des 8253. Für die Ansteuerung des Lautsprechers wird die Betriebsart angewählt, die ein Rechtecksignal erzeugt. Um den Lautsprecherkanal anzusprechen, legen Bit 6 und 7 des Steuerregisters fest,

daß der Zähler 2 verwendet werden soll. Gleichzeitig legen Sie mit Bit 4 und 5 des Steuerregisters die Anweisung zum Lesen des Zählerwertes fest. Der neue Zählerstand wird dann über die Portadresse 42hex in das Zählerregister des Kanals 2 geschrieben. Denken Sie bitte daran, zuerst das niederwertige und dann das höherwertige Byte zu übergeben.

Die Frequenz wird über die Portadresse 42hex festgelegt

Wenn Sie beispielsweise den Kammerton A über den PC-Lautsprecher ertönen lassen wollen, müssen Sie den Timer-Baustein 8253 so programmieren, daß er 440mal in der Sekunde ein elektrisches Signal erzeugt, das die Lautsprechermembran zum Schwingen anregt. Um dieses Ergebnis zu erreichen, übergeben Sie dem 8253 mit Hilfe der Portadresse 42hex den Wert, der die gewünschte Frequenz festlegt.
Der 8253 ist jedoch nicht der einzige Chip, der im PC für die Tonerzeugung zuständig ist. Fast alle Ein- und Ausgabeoperationen des PC steuert die programmierbare Peripherieschnittstelle PPI 8255 (PPI ist die Abkürzung für »Programmable Peripheral Interface«). Beim 8255 handelt es sich um einen Chip, der im PC hauptsächlich für die Verwaltung der Tastatur und die Kontrolle der Konfigurationsdaten verwendet wird.
Der 8255 verfügt über drei 8-Bit-Register, von denen zwei für die Eingabe- und eines für Ausgabefunktionen zuständig sind. Das Ausgaberegister spricht man über die Portadresse 61hex an. Den Registern für die beiden Eingabeschnittstellen sind die Adressen 60hex und 62hex zugewiesen.

Die unteren beiden Bits des Registers für die Ausgabeschnittstelle steuern den Lautsprecher an. Bit 0 stellt eine Verbindung zum 8253-Zeitgeber her. Ist Bit 0 gesetzt, dann wird der 8253 veranlaßt, ein Rechtecksignal auf Kanal 2 zu erzeugen. Wenn Bit 1 zusätzlich gesetzt ist, aktiviert der PC den Lautsprecher. Das bedeutet, daß das Rechtecksignal an den Lautsprecher weitergeleitet wird. Dieser gibt über die schwingende Membran den gewünschten Ton aus. Der Lautsprecher bleibt so lange aktiv, bis Bit 0 wieder gelöscht wird. Ist Bit 0 gelöscht, dann generiert der 8253 kein Rechtecksignal. Neben den drei Ein-/Ausgaberegistern wird der 8255 mit Hilfe eines Steuerregisters über die Portadresse 63hex verwaltet. Das Steuerregister legt fest, welche Kanäle für die Ein- und Ausgabe zuständig sind. Da das Betriebssystem die Werte automatisch initialisiert, ist dieses Register für Sie ohne Bedeutung.
Der Ihnen wahrscheinlich vertrauteste Ton ist das Piepen, das nach dem Einschalten und dem folgenden Selbsttest des Computers erklingt. Entdecken die Diagnoseroutinen des PC beim Start einen Hardware-Fehler, ruft das Betriebssystem ein Unterprogramm auf. Diese Prozedur konfiguriert den 8253- und den 8255-Baustein für das Erzeugen von Frequenzen und wählt die gewünschte Frequenz und Tondauer aus.

Eine Methode zur Tonerzeugung verzichtet auf den 8253

Es gibt grundsätzlich zwei Wege, um einen Ton zu erzeugen. Die erste Methode, die mit Listing 1 an einem Beispiel verdeutlicht wird, verwendet ausschließlich den 8255-Chip. Die CPU kontrolliert den Lautsprecher direkt und muß aus diesem Grund die Aufgaben übernehmen, die normalerweise vom Timer-Baustein erledigt werden. Dieses Verfahren ist zwar etwas unübersichtlich, dafür erlaubt es die direkte Beeinflussung der Tonprogrammierung und erleichtert die Erzeugung spezieller Geräuscheffekte.
Die Tonerzeugung mit der programmierbaren Peripherieschnittstelle 8255 erfordert nichts, als das mit dem Lautsprecher verbundene Frequenzbit des Port B ein- und auszuschalten (Bit 1). Port B wird über die Adresse 61hex angesprochen. Wenn ein Programm das Bit zu schnell setzt und wieder löscht, ist die resultierende Frequenz zu hoch, um nützlich zu sein. Aus diesem Grund sollten Sie Warteschleifen zwischen dem Ein- und Ausschalten des Lautsprechers einfügen.
Ein großer Nachteil dieser Vorgehensweise liegt in der Abhängigkeit des Programms vom Hardware-Aufbau. Schnellere und auch langsamere PCs benötigen andere Werte als in unserem Beispiel. Der größte Mangel dieses Funktionsprinzips ist seine Anfälligkeit gegenüber Interrupts. Tritt während der Tonerzeugung ein Interrupt auf, so kann es durchaus sein, daß der zu erzeugende Ton verzerrt wird oder gar nicht ertönt. Das ablauffertige Programm in Listing 2 verdeutlicht den Sachverhalt, wenn Sie es einmal auf unterschiedlichen Computern laufen lassen.
Die zweite Methode zur Tonerzeugung ist von diesen Problemen nicht betroffen. Damit sowohl Tonlänge und -höhe als auch die Verarbeitungsgeschwindigkeit nicht von der Taktfrequenz des verwendeten Computers abhängen und störende Interrupts die Tonausgabe nicht verzerren, bedienen wir uns des programmierbaren Zeitgebers 8253. Der Timer-Baustein erzeugt das Rechtecksignal für den Lautsprecher, ohne von der Arbeit der CPU abhängig zu sein.
Um dem 8253 einen bestimmten Ton zu entlocken, müssen Sie in das Register mit der Portadresse 42hex einen 16-Bit-Wert schreiben, durch den die interne Taktfrequenz des 8253 geteilt wird. Das Ergebnis dieser Division gibt die Frequenz für das zu erzeugende Rechtecksignal an. Vor der Übergabe des Frequenzteilers laden Sie das Steuerregister des 8253 über die Portadresse 43hex mit dem Wert B6hex, um den Baustein für den Lautsprecherbetrieb zu initialisieren.
Die interne Taktfrequenz des 8253 beträgt 1193180 Hz. Das ist ein Zwölftel des Systemtaktes von 14,31818 MHz, aus dem beispielsweise auch der 4,77-MHz-Takt für den Prozessor gewonnen wird. Auch in PCs, die mit höheren Taktfrequenzen arbeiten, ver-

Die wichtigsten Tonfrequenzen auf einen Blick

Oktave 1		Oktave 2		Oktave 3		Oktave 4		Oktave 5		Oktave 6		Oktave 7		Oktave 8	
Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz	Ton	Frequenz
C	16,35	C	32,70	C	65,41	C	130,81	C	261,63	C	523,25	C	1046,50	C	2093,00
Cis	17,32	Cis	34,65	Cis	69,30	Cis	138,59	Cis	277,18	Cis	554,37	Cis	1108,74	Cis	2217,46
D	18,35	D	36,71	D	73,42	D	146,83	D	293,66	D	587,33	D	1174,66	D	2349,32
Dis	19,45	Dis	38,89	Dis	77,78	Dis	155,56	Dis	311,13	Dis	622,25	Dis	1244,51	Dis	2489,02
E	20,60	E	41,20	E	82,41	E	164,81	E	329,63	E	659,26	E	1328,51	E	2637,02
F	21,83	F	43,65	F	87,31	F	174,61	F	349,23	F	698,46	F	1396,91	F	2793,83
Fis	23,12	Fis	46,25	Fis	92,50	Fis	185,00	Fis	369,99	Fis	739,99	Fis	1479,98	Fis	2959,96
G	24,50	G	49,00	G	98,00	G	196,00	G	392,00	G	783,99	G	1567,98	G	3135,96
Gis	25,96	Gis	51,91	Gis	103,83	Gis	207,65	Gis	415,30	Gis	830,61	Gis	1661,22	Gis	3322,44
A	27,50	A	55,00	A	110,00	A	220,00	A	440,00	A	880,00	A	1760,00	A	3520,00
Ais	29,14	Ais	58,27	Ais	116,54	Ais	233,08	Ais	466,16	Ais	923,33	Ais	1864,66	Ais	3729,31
H	30,87	H	61,74	H	123,47	H	246,94	H	493,88	H	987,77	H	1975,53	H	3951,07

Tabelle 1. Alle Töne der unteren acht Oktaven mit den zugehörigen Frequenzen

Bitkombination	Bedeutung
xxxxxxx0	Binärzähler
xxxxxxx1	BCD-Zähler
xxxx000x	Interrupt-Erzeugung
xxxx001x	programmierbarer Monoflop
xxxx010x	Taktgenerator
xxxx011x	Rechteckgenerator
xxxx100x	Triggerung des Ausgangs durch Software
xxxx101x	Triggerung des Ausgangs durch Hardware
xxxx110x	reserviert
xxxx111x	reserviert
xx00xxxx	Zähler zum Lesen übernehmen
xx01xxxx	niederwertiges Byte laden
xx10xxxx	höherwertiges Byte laden
xx11xxxx	erst niederwertiges und dann höherwertiges Byte laden
00xxxxxx	Zähler 0 verwenden
01xxxxxx	Zähler 1 verwenden
10xxxxxx	Zähler 2 verwenden
11xxxxxx	reserviert

Tabelle 2. Das Steuerregister bestimmt die unterschiedlichen Funktionen des Timer-Bausteins

verwendet der 8253 intern einen Takt von 1193180 Hz, so daß dieser Wert allgemeingültig ist.

Die interne Taktfrequenz des 8253 beträgt 1193180 Hz

Für die Zeitangabe ist ein Oszillator (Schwingkreis) zuständig, der in einer Sekunde 1193180 Impulse erzeugt, die er an den 8253 weiterleitet. Mit der zugehörigen Portadresse teilen Sie dem Zeitgeber mit, nach welcher Anzahl von Schwingungen er eine Aktion veranlassen soll.
Möchten Sie beispielsweise mit dem Timer-Baustein den Kammerton A erzeugen, so müssen Sie den Wert 2712 in das Register mit der Portadresse 42hex schreiben, da 1193180 geteilt durch 2712 den Frequenzwert 440 ergibt. Man berechnet den Frequenzteiler für den 8253, indem die interne Takfrequenz durch die Frequenz des gewünschten Tons geteilt wird. Wenn Sie der Klangprozedur im DI-Register die Frequenz mitteilen, übergeben die folgenden Zeilen den korrekten Wert für den zweiten Zeitgeber in AX:

```
mov dx,12h
mov ax,34dch
div di
```

In diesem Beispiel wird die interne Taktfrequenz des 8253 als Hexadezimalwert 1234DChex in das Registerpaar DX:AX geschrieben und durch den Frequenzwert in DI geteilt. Das Ergebnis dieser Division entnehmen Sie dem Register AX.
Um für die verschiedenen Töne eine konstante Länge zu garantieren, benötigt man in vielen Fällen eine Zeitschleife. Eine Zeitverzögerung von zehn Millisekunden erreichen Sie auf einem PC mit einer Taktfrequenz von 4,772727 MHz durch die Zeilen

```
milli: mov cx,2808
schleif: loop schleif
```

Die Zahl 2801 ergibt sich aus der Taktfrequenz des PC und den Ausführungszeiten der Assemblerbefehle. Der Prozessor führt das Mov-Kommando in vier Taktzyklen aus und die Loop-Funktion entweder in siebzehn Takten (wenn eine Verzweigung stattfindet) oder in fünf Takten. Da das Programm verzweigt, bis CX den Wert 1 enthält, ergibt sich die Gleichung

```
17*(2808-1)+5+4=47728
```

Die Gesamtzeit errechnet sich aus den Taktzyklen. Die Taktfrequenz von 4,772727 führt zu einem Taktzyklus von

```
1/4772727s=209,5ns
```

Dauer. Damit ergibt sich eine Gesamtdauer der Schleife von

```
47728*209,5ns=10ms
```

Besitzen Sie einen PC, der mit einer anderen Rate getaktet ist, dann müssen Sie den Multiplikationswert den Umständen anpassen.
Listing 3 zeigt Ihnen konkret an einem Beispiel, wie Sie Töne beziehungsweise vollständige Melodien über den Lautsprecher Ihres PC ausgeben. Das Programm, das Sie auch als ablauffertige Version in Listing 4 finden, läßt das Lied »Fuchs, du hast die Gans gestohlen« aus dem eingebauten Lautsprecher ertönen. Jede Note und die zugehörige Tondauer wird aus den Tabellen »note« und »laenge« ausgelesen, die sich im Datensegment befinden. Der Wert 0 kennzeichnet das Ende der Tabelle.
Für die Tonerzeugung verwendet das Programm die Vorgehensweise, die oben ausführlich erläutert wurde. Wichtig ist, daß Sie alle Register vor der Tonausgabe auf dem Stapel ablegen, um so sicherzustellen, daß keine Daten verlorengehen.
Die abgedruckten Quelltexte sind ohne Änderung nach der Übersetzung mit dem Masm und der Bearbeitung mit dem zugehörigen Linker lauffähig. Für Turbo-Pascal-Programmierer ist es natürlich wichtig, wie sich die Sound-Routinen direkt in Pascal-Programme aufnehmen lassen. Die Einbindung von Assemblerroutinen in Turbo-Pascal-Programme erläutert der Artikel »Assemblerunterstützung für Turbo-Pascal-Programme« ab Seite 97.

Tabellen mit Tonlänge und Tonhöhe erleichtern die Programmierung

Mit den Grundkenntnissen der Sound-Programmierung haben Sie einen wichtigen Einblick in die interne Rechnerarchitektur erhalten. Es sollte Ihnen mit diesen Kenntnissen keine großen Probleme mehr bereiten, eigene Ideen und kreative Vorstellungen auf dem Gebiet der Sound-Programmierung zu verwirklichen.
Eine Schwierigkeit können Sie aber trotz vieler Bemühungen nicht beseitigen: Die erzeugten Töne klingen immer »trocken« und »steif«. Das resultiert einerseits daraus, daß die Qualität des eingebauten Lautsprechers nicht die beste ist; andererseits steht auf dem PC nur ein Ausgabekanal zur Verfügung.

(Markus Zietlow/ks)

```
  Name.............. SOUND-1.ASM

  Sprache......... Masm ab 5.Ø
                   Tasm

  Grafikkarte..... jede

                  ═══════════════════════════

[2N6]     1: ;Assembler-Programm, das einen Pieps auf
[V94]     2: ;dem Lautsprecher erzeugt
[P3Ø]     3: ;
[KN2]     4: tonlaenge equ 8ØØ
[UK2]     5: frequenz  equ 15Ø
[8B2]     6: port      equ  61h
[P3Ø]     7: ;
[XV3]     8: code  segment para 'code'
```

```
[W82]     9: pieps proc far
[EA2]    1Ø:       assume cs:code
[1W2]    11:       push ax ;Sichern
[132]    12:       push bx ;der
[TM3]    13:       push cx ;wichtigsten
[TZ3]    14:       push dx ;Speicherwerte
[RM4]    15:       cli     ;Interrupts sperren
[NV4]    16:       mov  dx,tonlaenge ;Tonlänge
[V45]    17:       in   al,port ;Status des Port B
[3M4]    18:       and  al,1111111Øb ;holen, keine
[F35]    19:      ;Rechteckschwingung erzeugen
[DW3]    2Ø: ton:  or   al,ØØØØØØ1Øb ;und
[R74]    21:      ;Lautsprecher aktivieren
[TU4]    22:       out  port,al ;Status setzen
[LG5]    23:       mov  cx,frequenz  ;Warteschleife
[KN5]    24: hochton: loop hochton ;aktivieren
[SU3]    25:       and  al,11111Ø1b ;Erzeuge
[ME4]    26:      ;Rechteckschwingung nicht
[4G5]    27:       out  port,al ;an den Lautsprecher
```

```
[TB2]    28:       ;weiterleiten
[6L5]    29:        mov  cx,frequenz ;Warteschleife
[6R5]    3Ø: tiefton: loop tiefton ;aktivieren
[V56]    31:        dec  dx ;Zähler für die Tonlänge
[742]    32:       ;vermindern
[HU5]    33:        jnz  ton ;Wenn <> Ø, Sprung zu 'ton'
[DG4]    34:        sti ;Interrupts freigeben
[E93]    35:        pop  dx ;Die auf dem
[UW4]    36:        pop  cx ;Stapel gesicherten
[BE2]    37:        pop  bx ;Werte
[DX3]    38:        pop  ax ;wiederstellen
[MW4]    39:        mov  ah,4ch ;Rückkehr in die
[WE3]    4Ø:        int  21h ;MS-DOS-Ebene
[8S4]    41: pieps endp ;Ende der Prozedur
[5F5]    42: code  ends ;Ende des Code-Segments
[WZ4]    43:       end  pieps ;Ende des Programms
```

Listing 1. Die einfache Methode der Tonerzeugung ist von der verwendeten Hardware abhängig. Das Programm »sound-1 .asm« erzeugt einen Piepton.

```
[Y85]     1: 4D 5A 2A ØØ Ø2 ØØ ØØ ØØ 2Ø ØØ ØØ ØØ FF FF ØØ ØØ
[175]     2: ØØ ØØ 9B AE ØØ ØØ ØØ ØØ 1E ØØ ØØ ØØ Ø1 ØØ ØØ ØØ
[ØZ4]     3: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     4: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     5: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     6: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     7: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     8: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     9: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    1Ø: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    11: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    12: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    13: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    14: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    15: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    16: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    17: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    18: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    19: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    2Ø: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    21: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    22: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    23: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    24: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    25: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    26: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    27: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    28: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    29: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    3Ø: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    31: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    32: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[NK5]    33: 5Ø 53 51 52 FA BA 2Ø Ø3 E4 61 24 FE ØC Ø2 E6 61
[AP5]    34: B9 96 ØØ E2 FE 24 FD E6 61 B9 96 ØØ E2 FE 4A 75
[HK3]    35: EB FB 5A 59 5B 58 B4 4C CD 21
```

Listing 2. Mit »sound-1.exe« erhalten Sie Ihren sofort lauffähigen »Pieps«

```
Name..... SOUND-2.ASM

Sprache..... Masm ab 5.Ø
             Tasm

Grafikkarte..... jede
```

```
[F77]     1: ;Assembler-Programm zur Demonstration der
[4F3]     2: ;Sound-Eigenschaften
[P3Ø]     3: ;
[PU3]     4: data segment para 'data'
[KW5]     5: freq dw 98,11Ø,123,131,147,147,147,147,165,
             131
[945]     6: dw 196,165,147,165,131,196,165,147,147,131
[7Z4]     7: dw 131,131,131,123,123,123,123,11Ø,123,11Ø
[8X4]     8: dw 98,123,147,147,147,131,131,131,131,123
[5L3]     9: dw 123,123,123,11Ø,123,11Ø, 98
[XHØ]    1Ø: dw Ø
[136]    11: laenge dw 12 dup (1ØØ), 1 dup (4ØØ), 4 dup
             (1ØØ)
[3Z4]    12: dw 1 dup (4ØØ),28 dup (1ØØ), 1 dup (4ØØ)
[DG1]    13: data ends
[P3Ø]    14: ;
[XT3]    15: code segment para 'code'
[VA2]    16: musik proc far
[DU3]    17:      assume cs:code, ds:data
[1W2]    18:      push ax ;Sichern
[132]    19:      push bx ;der
[TM3]    2Ø:      push cx ;wichtigsten
[TZ3]    21:      push dx ;Speicherwerte
[F65]    22:      mov  ax,seg data ;Datensegment
[5F4]    23:      mov  ds,ax ;initialisieren
[UT4]    24:      lea  si,freq ;Tabellenzeiger
```

```
[TT4]    25:      lea  bp,ds:laenge ;einrichten
[DX6]    26: spielen: mov di,[si] ;Folgenden Ton aus
[FD2]    27:     ;Tabelle holen
[HM6]    28:      cmp  di,Ø ;Wenn die Melodie beendet ist
             ,
[XZ4]    29:      je   schluss ;Sprung zum Ende,
[B96]    3Ø:      mov  bx,ds:[bp] ;ansonsten Tondauer
[ZV1]    31:     ;ermitteln
[UL5]    32:      mov  al,182 ;Ausgaberegister für
[C55]    33:      out  43h,al ;Timer-Baustein 8253
[GS1]    34:     ;schreiben
[YØ5]    35:      mov  dx,12h ;Für den zweiten
[SD6]    36:      mov  ax,34dch ;Zeitgeber die benötigte
[SS4]    37:      div  di ;Frequenz bestimmen
[S36]    38:      out  42h,al ;Das niederwertige und das
[9P5]    39:      mov  al,ah ;höherwertige Byte des
[TN5]    4Ø:      out  42h,al ;Wertes an den 8253 geben
[C35]    41:      in   al,61h ;Aktuellen Inhalt des
[4Ø3]    42:      ;Ausgaberegisters
[LM4]    43:      mov  ah,al ;des PPI sichern
[YX4]    44:      or   al,3 ;und den Lautsprecher
[3Z3]    45:      out  61h,al ;einschalten
[EC6]    46: milli: mov cx,28Ø8 ;Einen Zeitintervall
[TV4]    47: schleif: loop schleif ;warten
[WA6]    48:      dec  bx ;Zähler dekrementieren, bis
[BC4]    49:      jnz  milli ;er gleich Ø ist
[FA6]    5Ø:      mov  al,ah ;Ursprüngliche PPI-Werte
[GY4]    51:      out  61h,al ;wiederherstellen
[LH4]    52:      add  si,2 ;Zeiger der Ton- und
[SM3]    53:     ;Lautstärken-Tabelle
[4H4]    54:      add  bp,2 ;um zwei erhöhen,
[D35]    55:      jnz  spielen ;bis Tabellenende
[ØP4]    56: schluss: pop dx ;-Die auf dem
[WR4]    57:      pop  cx ; Stapel gesicherten
[YM2]    58:      pop  bx ; Werte
[8G4]    59:      pop  ax ; wiederherstellen
[MW4]    6Ø:      mov  ah,4ch ;Rückkehr in die
[WE3]    61:      int  21h ;MS-DOS-Ebene
[7U4]    62: musik endp ;Ende der Prozedur
[5F5]    63: code  ends ;Ende des Code-Segments
[ZZ4]    64:       end musik ;Ende des Programms
```

Listing 3. Es ist nur wenig Aufwand nötig, um auf dem PC Melodien erklingen zu lassen. »sound-2.asm« unterhält Sie mit einem kleinen Lied.

```
[G85]     1: 4D 5A 12 Ø1 Ø2 ØØ Ø1 ØØ 2Ø ØØ ØØ ØØ FF FF ØØ ØØ
[265]     2: ØØ ØØ 83 9B ØØ ØØ ØC ØØ 1E ØØ ØØ ØØ Ø1 ØØ Ø5 ØØ
[TØ5]     3: ØC ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     4: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     5: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     6: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     7: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     8: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]     9: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    1Ø: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    11: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    12: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    13: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    14: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    15: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    16: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    17: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    18: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    19: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    2Ø: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    21: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    22: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    23: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    24: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    25: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    26: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    27: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    28: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    29: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    3Ø: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    31: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[ØZ4]    32: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
[A85]    33: 62 ØØ 6E ØØ 7B ØØ 83 ØØ 93 ØØ 93 ØØ 93 ØØ 93 ØØ
[WB5]    34: A5 ØØ 83 ØØ C4 ØØ A5 ØØ 93 ØØ A5 ØØ 83 ØØ C4 ØØ
[T75]    35: A5 ØØ 93 ØØ 93 ØØ 83 ØØ 83 ØØ 83 ØØ 83 ØØ 7B ØØ
[MB5]    36: 7B ØØ 7B ØØ 7B ØØ 6E ØØ 7B ØØ 6E ØØ 62 ØØ 7B ØØ
[375]    37: 93 ØØ 93 ØØ 93 ØØ 83 ØØ 83 ØØ 83 ØØ 83 ØØ 7B ØØ
[6A5]    38: 7B ØØ 7B ØØ 7B ØØ 6E ØØ 7B ØØ 6E ØØ 62 ØØ ØØ ØØ
[G55]    39: 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ
[C55]    4Ø: 64 ØØ 64 ØØ 64 ØØ 64 ØØ 9Ø Ø1 64 ØØ 64 ØØ 64 ØØ
[W55]    41: 64 ØØ 9Ø Ø1 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ
[G55]    42: 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ
[G55]    43: 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ
[G45]    44: 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 64 ØØ 9Ø Ø1 ØØ ØØ
[AD5]    45: 5Ø 53 51 52 B8 ØØ ØØ 8E D8 8D 36 ØØ ØØ 8D 2E 6Ø
[WH5]    46: ØØ 8B 3C 83 FF ØØ 74 32 3E 8B 5E ØØ BØ B6 E6 43
[BT5]    47: BA 12 ØØ B8 DC 34 F7 F7 E6 42 8A C4 E6 42 E4 61
[SS5]    48: 8A EØ ØC Ø3 E6 61 B9 F8 ØA E2 FE 4B 75 F8 8A C4
[MM5]    49: E6 61 83 C6 Ø2 83 C5 Ø2 75 C7 5A 59 5B 58 B4 4C
[6LØ]    5Ø: CD 21
```

Listing 4. Das lauffähige Programm »sound-2.exe«, dessen Quelltext Sie in Listing 3 abgedruckt sehen

Datenstrukturen nutzen in Turbo Pascal

Eine Stärke von Pascal liegt in den mächtigen Datenstrukturen, mit deren Hilfe auch sehr aufwendige Datenbestände ohne großen Aufwand zu bearbeiten sind. Neben den Standarddatentypen, die ebenfalls im Referenzteil dieses Sonderhefts erläutert werden, kann der Anwender eigene kombinierte Datenstrukturen anlegen.

Der bekannteste kombinierte Datentyp ist das »Array« oder »Feld« (oft auch »Matrix« genannt). Ein Array ist eine festgelegte Anzahl von Variablen, die sich unter einem einzigen Namen ansprechen lassen. Arrays sind ein enorm wichtiger Bestandteil der Programmierung. Angenommen, ein Programm fragt 100 unterschiedliche Werte ab und führt an diesen Werten eine statistische Erhebung durch. Ohne Arrays kann ein solches Programm nur realisiert werden, indem für jeden möglichen Wert eine eigene Variable angelegt wird, also beispielsweise »wert1«, »wert2«, »wert3« und so fort. Zum Einlesen der Werte benötigt das Programm dann jedoch immer eine eigene Readln-Anweisung. Dadurch wird der Programmcode sehr lang und das Eingeben und Schreiben von Programmen zu einer Geduldsprobe.

Was soll außerdem geschehen, wenn es sich nicht um 100, sondern um 1000 oder 10000 Werte handelt? Mit Arrays läßt sich eine Variable definieren, die beispielsweise 100 oder 1000 einzelne Elemente mit nur einem Variablennamen kombiniert. Soll beispielsweise eine Variable »wert« definiert werden, die 100 Ganzzahlen enthält, erledigt das die folgende Anweisung:

```
var wert: array[1..100] of integer;
```

Die einzelnen Elemente der Variablen lassen sich über Indizes ansprechen, die in eckigen Klammern hinter den Variablennamen gesetzt werden. Die Variable »wert[1]« adressiert so das erste Element des Arrays, die Variable »wert[50]« das 50. Array-Element.

Das wäre gegenüber einzelnen Variablen noch kein direkter Vorteil. Der Trick bei der Verwendung von Arrays liegt darin, als Index eine andere Variable zu verwenden. Hat beispielsweise die Variable »nummer« den Wert 34, dann wird durch die Konstruktion »wert[nummer]« das 34. Element des Arrays »wert« angesprochen. Durch diese Möglichkeit der variablen Indizierung lassen sich Arrays sehr vielseitig einsetzen. Das Einlesen von 100 Werten in Variablen muß nicht mit einem Readln-Befehl pro Variable erledigt werden, sondern kann in einer Schleife geschehen:

```
for index:=1 to 100 do
readln(wert[index]);
```

Zwei Programmzeilen reichen also aus, um 100 Werte in ein Array einzulesen (die Variablen »index« und »wert« müssen natürlich bereits definiert sein). Die Indizes, über die sich eine Variable ansprechen läßt, werden bei der Deklaration der Variablen festgelegt. Die Anweisung

```
var wert: array[51..150]
```

erzeugt ebenfalls ein Array »wert« mit 100 Elementen, allerdings sind das erste und das letzte Element hier unter den Namen »wert[51]« beziehungsweise »wert[150]« verfügbar. Der Versuch, die Variable »wert[1]« anzusprechen, führt zu einer Fehlermeldung. Generell erzeugt Turbo Pascal bei versuchter Indizierung oberhalb oder unterhalb des erlaubten Bereichs während der Ausführung einen Laufzeitfehler, der das Programm zum Abbruch bringt.

Arrays können in Pascal mehr als einen Index besitzen. Man spricht in diesem Zusammenhang auch von ein- und mehrdimensionalen Arrays. Die Definition von mehrdimensionalen Arrays wird vorgenommen, indem anstelle eines Bereichs für den Index zwei solcher Bereiche in die Deklaration aufgenommen werden. Die Deklaration

```
var matrix: array[1..5][1..4]: char;
```

legt beispielsweise insgesamt 20 Elemente vom Datentyp »char« an. Die ersten vier Elemente werden durch die Variablen »matrix[1][1]« bis »matrix[1][4]« angesprochen, die zweiten vier Elemente durch »matrix[2][1]« bis »matrix[2][4]« und so fort. Sie können sich zweidimensionale Variablen gut als Elemente einer Tabelle vorstellen, die durch Zeilen und Spalten adressiert werden. Die erste Dimension der Variablen (also der erste Index) stellt die Zeile der Tabelle dar, die zweite Dimension die Spalte der Tabelle. Tabelle 1 veranschaulicht diesen Zusammenhang für die Variable »matrix« nach obiger Deklaration. Zweidimensionale Variablen werden immer da angewandt, wo Elemente einer Datenstruktur wiederum mehrere einzelne Elemente enthalten. Um beispielsweise den Inhalt des Bildschirms zu speichern, kann die folgende Variable »bildschirm« angelegt werden:

```
var scr: array[1..25][1..80] of word;
```

In diesem Feld mit insgesamt 2000 Komponenten (80x25) lassen sich die Zeichencodes und Attribute eines ganzen Bildschirms mit 25 Zeilen und 80 Spalten unterbringen.

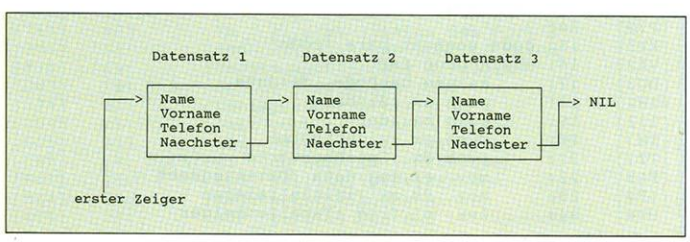
Es lassen sich in Pascal auch mehr als zwei Dimensionen für ein Array verwenden. Ein Beispiel: Eine Grundschule besitzt vier Jahrgangsstufen mit je drei Klassen. Jede dieser Klassen umfaßt 20 Schüler und Schülerinnen. Auf dem Jahreszeugnis werden Noten für zehn Fächer vergeben. Um die Noten aller Schüler in einem einzigen Feld zu speichern, darf die folgende Konstruktion verwendet werden:

```
var noten: array[1..4][1..3][1..20]
[1..10] of byte;
```

In der ersten Dimension dieses Feldes wird die Jahrgangsstufe eines Schülers gespeichert, die zweite Dimension verwaltet die Klasse, in der der Schüler Unterricht erhält. Die dritte Dimension bezeichnet die Nummer des Schülers in der Klasse, die sich aus der alphabetischen Reihenfolge der Schülernamen ergeben kann. Die vierte Dimension schließlich gibt die Nummer der Note an, auf die zugegriffen werden soll. Angenommen, die Mathematiknote von Schülerin Zuhlig aus der Klasse 3b soll festgestellt werden. »Zuhlig« steht in der Namensliste ihrer Klasse an letzter Stelle, und die Mathematiknote wird als zweiter Wert des Notenfeldes abgespeichert. Um Fräulein Zuhlig die Note »sehr gut« in Mathematik zuzuweisen, dient die folgende Anweisung:

```
noten[3][2][20][2]:=1;
      {3  b   Z   M}
```

Der Kommentar unter der Anweisung gibt Aufschluß darüber, wie die Indi-



Eine einfach verkettete Liste

	erster Index	zweiter Index		
	1	2	3	4
1	matrix[1][1]	matrix[1][2]	matrix[1][3]	matrix[1][4]
2	matrix[2][1]	matrix[2][2]	matrix[2][3]	matrix[2][4]
3	matrix[3][1]	matrix[3][2]	matrix[3][3]	matrix[3][4]
4	matrix[4][1]	matrix[4][2]	matrix[4][3]	matrix[4][4]
5	matrix[5][1]	matrix[5][2]	matrix[5][3]	matrix[5][4]

Tabelle 1. Darstellung eines zweidimensionalen Arrays als Tabelle

zes ermittelt werden, über die auf die Note der Schülerin zugegriffen wird.

Neben Arrays finden auch die sogenannten »Records« häufig Einsatz in Programmen. »Record« bedeutet übersetzt »Datensatz«. Damit läßt sich die Zielsetzung eines Records sehr gut charakterisieren. Mit Records werden verschiedene Datentypen zu einem einzigen Datentyp kombiniert. Daher spricht man im Zusammenhang mit Records auch oft von »Verbünden«.

Records erlauben die Verbindung unterschiedlicher Datentypen

Ein Record erhält – ähnlich wie ein Array – einen Namen, unter dem sich mehr als eine Komponente ansprechen läßt. Im Gegensatz zum Array können diese Komponenten beim Record aber aus unterschiedlichen Datentypen bestehen. Für das vorige Beispiel der Schulklassen einer Grundschule könnten Daten von allen Schülern und Schülerinnen in einer Datei gespeichert sein. Zu jedem Schüler wird dabei der Name, das Alter, die Jahrgangsstufe und die Klasse abgelegt. Der Name und die Klasse wird als String gespeichert, das Alter und die Jahrgangsstufe jedoch als Byte. Am einfachsten ist es, zunächst einen Datentyp »schuelertyp« zu definieren, der die vier Komponenten eines Schülers enthält:

```
type schuelertyp = record
name,klasse: string;
alter,jahrgang: byte;
end;
```

Mit dieser Anweisung wird der Datentyp »schuelertyp« definiert, der als Record codiert ist. Eine Variable »schueler«, in der die Daten eines Schülers gespeichert werden sollen, läßt sich dann über den neuen Datentyp deklarieren:

```
var schueler: schuelertyp;
```

Ähnlich wie bei der Anwendung eines Arrays, besteht auch bei Records die Möglichkeit, auf die einzelnen Komponenten des Verbunds zuzugreifen. Im Gegensatz zu Arrays werden dabei aber keine Indizes, also Zahlen, sondern die symbolischen Namen der Deklaration verwendet. Diese Namen werden vom Variablennamen durch den sogenannten »Separator«, den Punkt, getrennt. Um auf die Komponente »alter« des Records »schueler« zugreifen zu können, findet beispielsweise der Variablenname »schueler.alter« Verwendung. Über diese Technik lassen sich alle Komponenten eines Records problemlos ansprechen.

Wichtig ist dabei, daß ein Record seinerseits andere Records oder Arrays enthalten kann oder Bestandteil eines Arrays sein darf. Sollen etwa alle Mädchen unserer Beispielschule in einem Array gespeichert werden, kann die folgende Deklaration hilfreich sein:

```
var maedchen: array[1..500] of
schuelertyp;
```

Mit dieser Definition lassen sich in der Variablen »maedchen« insgesamt 500 Elemente ansprechen, die alle vom Datentyp »schuelertyp« sind, also die Komponenten »name«, »klasse«, »alter« und »jahrgang« enthalten. Die zu adressierenden Komponenten werden dabei wie gewohnt durch den Separator getrennt an den Namen der (jetzt indizierten) Variablen gehängt. Der Name des ersten Mädchens wird also in der Variablen »maedchen[1].name« gespeichert.

Für in sich verschachtelte Records bietet sich der With-Befehl an

Es kann sehr lästig sein, bei verschachtelten Records (beispielsweise »klasse[1].schueler.geschlecht«) immer die kompletten Record-Namen mitzuführen. Pascal bietet daher eine Möglichkeit, in Abschnitten des Programms automatisch festzulegen, welche Record-Variable gemeint ist: den Befehl »with«. Mit dem With-Kommando wird der Name einer Record-Variablen festgelegt, die im Geltungsbereich des »with« immer als aktuelle Record-Variable gilt. Das hat den Vorteil, daß innerhalb der With-Struktur nur die Komponentennamen des Records aufgeführt werden müssen. Der Geltungsbereich des With-Befehls wird durch ein »begin« eingeleitet und durch ein »end« abgeschlossen. Ein Beispiel dazu:

```
with maedchen do
begin
klasse:='3b';
jahrgang:=3;
end;
```

ist gleichbedeutend mit der Befehlsfolge:

```
meadchen.klasse:='3b';
maedchen.jahrgang:=3;
```

Neben den Arrays und Records gibt es unter Pascal noch weitere »strukturierte Typen«. Während Arrays und Records im Prinzip beliebige Informationen enthalten können, werden mit aufzählbaren Datentypen sogenannte »diskrete Typen« definiert. Das bedeutet, daß diese Typen nur bestimmte, festgelegte Inhalte annehmen dürfen. Eine Boolean-Variable ist beispielsweise ein diskreter Datentyp, während eine Fließkommazahl im Prinzip keinen diskreten Datentyp darstellt. Daß sich an der Unterscheidung zwischen diskreten und nichtdiskreten Datentypen die akademischen Geister scheiden, soll an dieser Stelle außer acht gelassen werden.

In Turbo Pascal werden Datentypen, die durch eine Aufzählung ihrer Inhalte definiert werden, »Aufzählungstypen« genannt. Ein Beispiel für einen Aufzählungstyp ist der Datentyp »daviscup«, der die männlichen Teilnehmer des bundesdeutschen Daviscup-Teams von 1989 enthält:

```
type daviscup = ('Becker',
'Steeb','Jelen','Kühnen');
```

Eine Variable, die über den Datentyp »daviscup« definiert wird, darf nur einen dieser vier Inhalte annehmen, jeden anderen Versuch weist Turbo Pascal durch eine Fehlermeldung zurück. Der Vorteil von Variablen, die über Aufzählungstypen definiert sind, liegt darin, daß über spezielle Funktionen die »Vorgänger« und »Nachfolger« eines der möglichen Werte ermittelt werden können. Die Anweisung »succ('Becker')« ergibt den String »Steeb«, die Funktion »pred('Jelen')« den String »Steeb«. Eine ähnliche Grundstruktur wie Aufzählungstypen weisen »Mengen« auf.

Mit Mengen lassen sich besondere Operationen durchführen

Eine Menge ist eine Sammlung von individuellen Werten zu einer Gesamtheit. Im Unterschied zu Aufzählungstypen können mit Mengen besondere Operationen, zum Beispiel das Bilden der Schnitt- oder Vereinigungsmenge, vollzogen werden.

Operation	Bedeutung	Beispiel
+	Addition	['a','b','c'] + ['c','d','e'] = ['a','b','c','d','e']
–	Subtraktion	['a','b','c'] – ['c','d','e'] = ['a','b','d','e']
*	Schnittmenge	['a','b','c']*['c','d','e'] = ['c']
=	Gleichheit	['a','b','c'] = ['c','d','e'] = FALSE, ['c','d','e'] = ['c','d,'e'] = TRUE
< =	Impliziert in	['a','b'] < = ['c','d','e'] = TRUE, ['a','c'] < = ['c','d'] = FALSE
> =	Impliziert	['a','b','e'] > = ['b','e'] = TRUE, ['a','c'] > = ['c','d'] > = FALSE

Tabelle 2. Die Mengenoperation mit Beispielen

Mengen werden in Turbo Pascal durch die Anweisung »Set-of« definiert. Die Deklaration

```
var vokale: set of char;
```

weist den Compiler an, die Variable »vokale« als eine Menge zu betrachten, die einzelne Zeichen enthält. Wichtig ist, daß Mengen in Pascal nur mit den Datentypen »byte«, »char« und »boolean« gebildet werden dürfen. Die Zuweisung einer Menge geschieht über eine gewohnte Wertzuweisung, bei der auf der rechten Seite des Gleichheitszeichens in eckigen Klammern alle Elemente der Menge durch Kommata getrennt aufgezählt werden. Um der Variablen »vokale« alle Vokale zuzuweisen, wird beispielsweise die folgende Zuweisung verwendet:

```
vokale:=['a','e','i','o','u'];
```

Für ganze Blöcke eines Datentyps wird der Punktoperator »..« akzeptiert, um Start- und Endwert eines Blocks festzulegen. Eine gültige Zuweisung ist also auch:

```
grossbuchstaben:=
['A'..'Z','Ä','Ö','Ü'];
```

Die Leistungsfähigkeit von Mengen liegt darin, daß sich Mengenoperationen mit ihnen durchführen lassen. Jeder, der die Mengenlehre kennt, kann dieses Wissen mit Pascal-Programmen direkt umsetzen. Tabelle 2 stellt die Mengenoperationen von Pascal mit Beispielen dar.

Bislang wurden in diesem Artikel Strukturen vorgestellt, die dem Compiler in ihrer Lage und Ausdehnung bekannt sind. Wird ein Record mit den Komponenten »integer« und »char« angelegt, so reserviert der Compiler für diesen Record einen bestimmten Bereich im Datensegment (auf welche Weise das geschieht, zeigt der Artikel »Management statt Bürokratie« in dieser Ausgabe). Einmal definiert, verändert sich die Adresse und vor allem die Länge einer Variablen in der Regel nicht. Man spricht daher in diesem Zusammenhang auch von »statischen Variablen«. Ein typisches Dilemma statischer Variablen ist zum Beispiel die Tatsache, daß die Größe eines Arrays bei der Deklaration feststehen muß. Die variable Dimensionierung, wie sie Basic kennt, ist in Pascal nicht zugelassen. Das hat den Effekt, daß bei der Verwendung von Arrays immer eine Obergrenze festgelegt werden muß, an der sich aus dem laufenden Programm heraus nicht rütteln läßt. Ganz anders stellt sich die Situation bei den »dynamischen Variablen« dar. Dies sind Variablen, deren Komponenten im laufenden Programm neu angelegt werden können. Werden beispielsweise 30 neue Speicherbereiche für Strings benötigt, lassen sich diese aus dem laufenden Programm heraus anlegen. Turbo Pascal besitzt zu diesem Zweck einen speziellen Speicherbereich, den sogenannten »Heap«. Der Heap beginnt bei einem laufenden Programm oberhalb des Daten- und Stack-Bereichs und kann den gesamten noch freien Arbeitsspeicher belegen.

Dynamische Datenstrukturen werden erst während der Laufzeit angelegt

Werden neue dynamische Datenstrukturen angelegt, besetzt Turbo Pascal den Heap langsam von unten nach oben. Dieser Neubelegung werden erst dann Grenzen gesetzt, wenn das Ende des Arbeitsspeichers erreicht ist. Über den Heap läßt sich also der volle freie Arbeitsspeicher innerhalb des MS-DOS-Bereichs von 640 KByte nutzen.

Dynamische Datenstrukturen werden über »Zeiger« verwaltet. Zeiger sind Elemente, die eine Speicheradresse angeben. Das Verständnis des Zeigerkonzepts von Turbo Pascal ist sehr wesentlich für den Umgang mit dynamischen Datenstrukturen. Zeiger werden durch das Zeichen »^« markiert. Die Anweisung

```
var zeige: ^byte;
```

legt einen Zeiger an, der auf ein Element des Datentyps »byte« weist. Wird dann eine Variable des Datentyps »byte« beispielsweise mit

```
var wert: byte;
```

definiert, so kann der Zeiger die Adresse des Bytes im Speicher festhalten. Die Anweisung

```
zeige:=@wert;
```

weist dem Zeiger »zeige« die Adresse der Variablen »wert« zu. Das Besondere am Zeiger ist nun, daß er den tatsächlichen Speicherplatz der Struktur, auf die er zeigt (im obigen Fall ein Byte), direkt ansprechen kann, indem hinter den Variablennamen das Zeichen »^« gesetzt wird. Der Befehl

```
zeige^:=100;
```

verändert daher ein Byte im Speicher. Das erfolgt genau an der Adresse, auf die »zeige« weist. Im obigen Beispiel wird also der Wert der Variablen »wert« auf 100 gesetzt, ohne daß der Name der Variablen explizit ins Spiel gebracht wird. Listing 1 (»zeigdemo.pas«) erläutert diese Manipulation im Rahmen eines vollständigen Programms.

Das Beispiel demonstriert, daß sich Zeiger auf zwei Arten nutzen lassen: ohne und mit dem Zeichen »^«. Immer, wenn eine Zeigervariable ohne dieses Zeichen verwendet wird, spricht sie eine Adresse an. Daher muß auf der Zeigerzuweisung ohne »^« in unserem Beispiel auf der rechten Seite der Zuweisung der Adreß-Operator »@« von Turbo Pascal stehen. Ein Zeiger mit »^« spricht nicht mehr eine Adresse, sondern einen Wert an einer Adresse an. Dabei kann es sich um ein Byte, ein Wort oder einen anderen Datentyp handeln.

Der Umgang mit Zeigern ist gewöhnungsbedürftig. Ihr Verständnis verlangt gewisse Abstraktionsfähigkeiten. Sicherheit beim Arbeiten mit Zeigern läßt sich nur durch Praxis erreichen.

Neben diesen Zeichen, die auf bestimmte Datentypen festgelegt sind, bietet Turbo Pascal auch einen Standarddatentyp »pointer«, mit dem sich Zeiger auf beliebige Speicherstellen richten lassen. Dynamische Datenstrukturen werden unter Turbo Pascal im Prinzip über diesen Zeigertyp gesteuert.

Dynamische Datenstrukturen werden durch die Kombinationen von Records und Zeigern gebildet

Der Trick liegt darin, einen Zeigertyp mit einer statischen Struktur (üblicherweise einem Record) zu verbinden, der wiederum einen (oder mehrere) Zeiger enthält, die auf die gleiche statische Struktur zeigen. Auch wenn diese Methode auf den ersten Blick wie ein Sich-an-den-eigenen-Haaren-aus-dem-Sumpf-Ziehen anmutet, läßt sie sich in Turbo Pascal formal deklarieren. Die Idee, die dahintersteht, geht davon aus, daß ein Record als eine Komponente einen

Zeiger auf den nach ihm folgenden Record enthält. Ein Programm kann sich anhand dieser Zeiger von einem Record zum anderen »durchhangeln«, um alle Felder ansprechen zu können. Angenommen, eine Liste von Namen und Telefonnummern soll durch eine dynamische Datenstruktur gespeichert werden. Der Record für einen Datensatz dieser Liste soll die Komponenten »name«, »vorname« und »telefon« enthalten, die alle vom Typ »string« sein sollen. Um eine dynamische Datenstruktur zu erreichen, erhält der Record eine weitere Komponente, in der ein Zeiger auf den nächsten Record gespeichert wird.

Eine solche Datenstruktur nennt man »einfach verkettete Liste«, weil die einzelnen Records nur über ihre Zeiger in eine Richtung angesprochen werden können. Das Bild stellt das Prinzip der einfach verketteten Liste an einem Beispiel dar.

Die tatsächliche Position der Elemente der Liste im Speicher ist dabei völlig irrelevant. Die Reihenfolge der Elemente wird nur durch die Zeiger auf das folgende Element festgelegt. Theoretisch kann also das erste Element ganz hinten im Speicher stehen, das letzte Element irgendwo in der Speichermitte und das vierte Element am Beginn des Heap. Das wird zwar in der Praxis bei sukzessiver Neubelegung des Heaps nicht auftreten, ist aber nach Manipulationen am Datenfeld durchaus ein denkbarer Zustand.

Ein Blick auf das Bild wirft die Frage auf, wie sich solche Strukturen formal in Pascal deklarieren lassen. Um die im Bild dargestellte Datenstruktur anzulegen, können die folgenden Zeilen verwendet werden:

```
type
satzzeig=^datensatz;
datensatz=record
name,vorname,telefon:string;
naechster:satzzeig;
end;
var
erster,aktueller:satzzeig;
```

In der Datensatzdeklaration werden die drei Datenfelder »name«, »vorname« und »telefon« angelegt. Als letztes Feld folgt ein Zeiger. Dieser Zeiger wird über den Datentyp »satzzeig« definiert, der auf die Datensatzdeklaration selbst weist.

Als einzige Variablen werden die Zeiger »erster« und »aktueller« definiert, die als Zeiger auf die dynamische Datenstruktur Verwendung finden. Der Record »datensatz« belegt bis zum Start des Programms noch keinen Speicherplatz, da er keiner Variablen direkt zugewiesen wird. Lediglich die beiden Zeiger sind mit je vier Byte im Speicher vertreten.

Ein dynamischer Datensatz enthält immer mindestens einen Zeiger

Der Speicherbereich für die Datensätze wird erst dann auf dem Heap belegt, wenn er im Programm tatsächlich benötigt wird. Die Pascal-Anweisung »new« ist für diese Belegung zuständig. Das Kommando

```
new(erster);
```

legt nach der obigen Deklaration einen Speicherblock für den Record »datensatz« im Heap an. Die Variable »erster« zeigt auf diesen Speicherblock. Wie werden die einzelnen Komponenten des Records nun adressiert? Dazu dient die grobe Unterscheidung zwischen der Verwendung von Zeigern: Zeiger ohne das

DOS-Software gesucht

Wir suchen für das Betriebssystem MS-DOS ab Version 3.0

- vermarktungsfähige Programme aller Kategorien (von Anwendungs- und Hilfsprogrammen bis hin zu Spielen oder Programmbibliotheken)
- Programme für den geschäftlichen Alltag
- Software für spezielle Anwendungsgebiete
- Branchenlösungen

Ob noch in der Entwicklung, gerade fertiggestellt oder bereits in Kleinserien vertrieben: Wir sind der richtige Vertriebspartner für Ihre Software. Möglicherweise wird gerade Ihr Programm zum "Bestseller". Scheuen Sie sich also nicht, den ersten Schritt zu unternehmen. Senden Sie Ihr Programm völlig unverbindlich mit Anleitung beziehungsweise Handbuch auf Diskette an

Redaktionsbüro Everts & Hagedorn
»DOS-Software«
Hans-Stießberger-Straße 3
8013 Haar/München
Tel: (0 89) 46 10 01-67

Wir behandeln Ihre Einsendung diskret und freuen uns auf Ihre Reaktion.

Zeichen »^« enthalten eine Adresse, Zeiger mit »^«-Zeichen Daten. Um auf den Record zuzugreifen, wird daher die Variable »erster^« verwendet. Um dem Record einen Namen und eine Telefonnummer zuzuweisen, wird die folgende Befehlsfolge programmiert:

```
erster^.name:='Kessler';
erster^.vorname:='Karlo';
erster^.telefon:='0211/303010';
```

Die Komponentenkennung des Datensatzes wird, wie von »normalen« Variablen bekannt, an den Zeiger gehängt. Das einzige Feld, das im Record »datensatz« noch nicht belegt ist, ist das Feld »naechster«, das einen Zeiger auf den nächsten Datensatz enthalten soll. Zur Verknüpfung wird zunächst ein weiterer Datensatz auf dem Heap belegt:

```
new(aktueller);
```

Nach diesem Befehl sind zwei Datensätze auf dem Heap gespeichert. Auf den ersten zeigt die Variable »erster«, auf den zweiten die Variable »aktueller«. Der Trick bei Strukturen, die mehr als nur diese zwei Datensätze enthalten, besteht darin, mit dem Feld »naechster« des ersten Datensatzes auf den zweiten zu zeigen, der wiederum mit seinem »naechster«-Feld auf den folgenden zeigt und so weiter.
Dazu wird in das jeweilige »naechster«-Feld ein Zeiger eingetragen. Für die Verknüpfung der Inhalte von »erster« mit »naechster« dient die Anweisung

```
erster^.naechster:=aktueller;
```

Damit wird der Zeiger »aktueller« in den ersten Datensatz übernommen. Dieser Zeiger wiederum weist auf den zweiten Datensatz. Bei langen Ketten von Datensätzen reicht es aus, einen Zeiger auf den ersten Datensatz zu kennen. Über dessen »naechster«-Feld wird der zweite Datensatz lokalisiert. Dieser enthält einen Zeiger auf den nächsten Satz und so weiter. Bei der Verwendung von dynamischen Datenstrukturen reichen demnach wenige statische Zeigervariablen aus, um große Datenmengen auf dem Heap zu belegen.
Für das Ende einer Liste steht ein vordefinierter Bezeichner zur Verfügung. Der Bezeichner »nil« ist der »Nullzeiger«, der anzeigt, daß in einer Liste kein Element mehr folgt. Der letzte Datensatz einer Kette wird in der Regel mit der folgenden Anweisung gekennzeichnet:

```
letzter^.naechster:=nil;
```

Zur Erläuterung der Unterschiede zwischen statischen und dynamischen Records an einem Beispiel dienen die Listings 2 (»namfeld.pas«) und 3 (»namzeig.pas«). Beide erfüllen die gleiche Aufgabe: das Lesen einer Anzahl von Namen und Telefonnummern in einen Record, das Sortieren dieser Datensätze nach dem Alphabet und das anschließende Ausgeben auf den Bildschirm.
Während Listing 1 die Datensätze über ein Array, also eine statische Datenstruktur, belegt, verwendet Listing 2 eine auf Zeigern aufbauende dynamische Datenstruktur.
In beiden Fällen wird der Algorithmus »Bubble Sort« verwendet, der in der Regel (wenn das Feld nicht bereits sortiert ist) Datensätze austauschen muß. Während bei der statischen Lösung die Datensätze tatsächlich ausgetauscht werden, reicht es bei der dynamischen Realisierung aus, zwei Zeiger auszutauschen.
Für viele Anwendungen genügen einfach verkettete Listen, wie sie bisher dargestellt wurden, nicht. Der Nachteil dieser einfach verketteten Listen liegt auf der Hand: Die Liste kann nur in einer Richtung abgearbeitet werden. Es besteht keinerlei Möglichkeit, anhand eines Datensatzes seinen Vorgänger festzuhalten.
Einen Ausweg bieten »doppelt verkettete« Listen. Darunter versteht man Datenstrukturen, die als Komponenten des Records nicht nur einen, sondern zwei Zeiger enthalten. Der erste Zeiger weist, wie oben dargestellt, auf das dem aktuellen Datensatz folgende Element. Der zweite Zeiger hingegen zeigt auf den Datensatz, der vor der aktuellen Variablen liegt. Für die Definition des Datentyps »datensatz« ergibt sich daraus:

```
type
satzzeig=^datensatz;
datensatz=record
name,vorname,telefon:string;
naechster,voriger:satzzeig;
end;
```

Von jedem Datensatz aus lassen sich durch diese Konstruktion die »links« und »rechts« von ihm liegenden Datensätze einfach adressieren.
Das Gegenteil des New-Befehls stellt das Kommando »dispose« dar. Mit »dispose« werden Datenstrukturen vom Heap entfernt:

```
dispose(loeschen);
```

Turbo Pascal merkt sich das entstehende »Loch« im Heap in einer internen Liste und belegt es umgehend, sobald eine Datenstruktur den gleichen Platz auf dem Heap anfordert. Beim Löschen von Elementen einer Liste muß der Programmierer tunlichst darauf achten, daß die vor und hinter der Lücke liegenden Zeiger angepaßt werden, um das Loch zu überbrücken. Ansonsten kommt es schnell zu »Datenmüll«.
Mit den Kenntnissen über die Turbo-Pascal-Datenstrukturen lassen sich viele interessante Anwendungen programmieren, seien es nun Dateiverwaltungen oder Spiele, die mit vielen Variablen und Feldern arbeiten. Vielleicht haben auch Sie beim Lesen dieses Artikels Lust bekommen, etwas mit den Datenstrukturen herumzuexperimentieren.

(Martin Althaus/dm)

```
[KØ3]    1: program zeigdemo;
[ØØØ]    2:
[HR5]    3: {Demonstriert die Manipulations-
[SD5]    4:  möglichkeiten, die sich aus dem
[TØ5]    5:  Einsatz von Zeigern ergeben
[ØØØ]    6:
[M34]    7:  Autor: Martin Althaus 2/9Ø
[ØØØ]    8:
[4Ø4]    9:  (C) 199Ø Redaktionsbüro
[WS2]   1Ø:            Everts & Hagedorn
[U7Ø]   11: }
[RNØ]   12: var
[BF2]   13:  zeiger: ^byte;
[CH4]   14:  {Der Zeiger auf ein Byte}
[HX1]   15:  wert: byte;
[5S3]   16:  {und das Byte selbst}
[ØØØ]   17:
[MUØ]   18: begin
[ED1]   19:  wert:=1Ø;
[YZ4]   2Ø:  {das Byte erhält einen Wert}
[815]   21:  writeln('WERT hat jetzt den ',
[552]   22:          'Wert ',wert);
[KL4]   23:  {der Wert wird ausgegeben}
[RE2]   24:  zeiger:=@wert;
[464]   25:  {der Zeiger zeigt auf den
[984]   26:   Datenbereich des Bytes}
[B12]   27:  zeiger^:=2Ø;
[C85]   28:  {und verändert diesen direkt}
[PD4]   29:  writeln('WERT hat nun den ',
[552]   3Ø:          'Wert ',wert);
[D65]   31:  {dadurch ergibt sich ein neuer
[LØ1]   32:   Wert}
[KNØ]   33:  end.
```

Listing 1. Die Manipulation von Speicherstellen über Zeiger (»zeigdemo.pas«)

```
[6V2]    1: program namfeld;
[ØØØ]    2:
[ØR5]    3: {Programm zur Demonstration der
[495]    4:  Verwaltung einer Adressenliste
[BG2]    5:  über ein Array
[ØØØ]    6:
[M34]    7:  Autor: Martin Althaus 2/9Ø
[ØØØ]    8:
[4Ø4]    9:  (C) 199Ø Redaktionsbüro
[WS2]   1Ø:            Everts & Hagedorn
[U7Ø]   11: }
[ØØØ]   12:
[5H1]   13: uses crt;
[ØØØ]   14:
[STØ]   15: type
```

```
[4Y2]   16:  datensatz=record
[MØ5]   17:   name,vorname,telefon:string;
[ZNØ]   18:  end;
[ØØØ]   19:
[RNØ]   2Ø: var
[1G3]   21:  namen: array[1..2Ø] of
[KT1]   22:  datensatz;
[BD5]   23:  {die konstante Felddefinition}
[CG4]   24:  anzahl,index,zaehler:byte;
[US2]   25:  help:datensatz;
[ØØØ]   26:
[MUØ]   27: begin
[UE5]   28:  clrscr; {Lösche Bildschirm}
[545]   29:  write('Wie viele Namen möchten',
[PW2]   3Ø:  ' Sie eingeben? ');
[LG2]   31:  readln(anzahl);
[VY2]   32:  {Lese Anzahl ein}
[RU3]   33:  writeln; {Leerzeile}
[S54]   34:  for index:=1 to anzahl do
[MUØ]   35:  begin
[ØD4]   36:   writeln(index,'. Eingabe:');
[R23]   37:   write('Vorname? ');
[245]   38:   readln(namen[index].vorname);
[HØ3]   39:   write('Nachname? ');
[KG4]   4Ø:   readln(namen[index].name);
[F13]   41:   write('Telefon? ');
[L35]   42:   readln(namen[index].telefon);
[XF1]   43:   writeln;
[ZNØ]   44:  end;
[ØØØ]   45:
[2G4]   46:  {alle Adressen und Telefon-
[9Z4]   47:   nummern sind nun eingelesen,
[5H5]   48:   beginne nun mit dem Sortieren}
[ØØØ]   49:
[ØG4]   5Ø: for zaehler:=1 to anzahl-1 do
[3M4]   51:   for index:=1 to anzahl-1 do
[ZY3]   52:   {benutze Bubble-Sort}
[MUØ]   53:   begin
[KR3]   54:    if (namen[index].name>
[V94]   55:    namen[index+1].name) then
[MUØ]   56:    begin
[3G3]   57:     help:=namen[index];
[BN3]   58:     namen[index]:=namen[
[TC1]   59:     index+1];
[CP3]   6Ø:     namen[index+1]:=help;
[ZNØ]   61:    end;
[ZNØ]   62:   end;
[ØØØ]   63:
[5B5]   64:  {Jetzt ist alles sortiert, also
[5R4]   65:   gebe die fertige Liste aus}
[ØØØ]   66:
[GB5]   67:  clrscr; {Bildschirm löschen}
[S54]   68:  for index:=1 to anzahl do
[MUØ]   69:  begin
[6Z4]   7Ø:   writeln(index,'. Datensatz:');
[XV4]   71:   writeln(namen[index].name);
[WD5]   72:   writeln(namen[index].vorname);
[PC5]   73:   writeln(namen[index].telefon);
[XF1]   74:   writeln;
[ZNØ]   75:  end;
[KNØ]   76: end.
```

Listing 2. Eingabe und Sortieren einer Liste über ein Array (»namfeld.pas«)

```
[XV2]    1: program namzeig;
[ØØØ]    2:
[ØR5]    3: {Programm zur Demonstration der
[495]    4:  Verwaltung einer Adressenliste
[G52]    5:  über Zeiger
[ØØØ]    6:
[M34]    7:  Autor: Martin Althaus 2/9Ø
[ØØØ]    8:
[4Ø4]    9:  (C) 199Ø Redaktionsbüro
[WS2]   1Ø:           Everts & Hagedorn
[U7Ø]   11: }
[ØØØ]   12:
[5H1]   13: uses crt;
[ØØØ]   14:
[STØ]   15: type
[ØS3]   16:  satzzeig= ^datensatz;
[4Y2]   17:  datensatz=record
[MØ5]   18:   name,vorname,telefon:string;
[9L3]   19:   naechster:satzzeig;
[ZNØ]   2Ø:  end;
[LS4]   21:  {hier ist der variable Record
[MX1]   22:   definiert}
[ØØØ]   23:
[RNØ]   24: var
[6B3]   25:  erster,alter,neuer,
[4B4]   26:  weiterer,merke: satzzeig;
[V25]   27:  {die variable Felddefinition}
[CG4]   28:  anzahl,index,zaehler:byte;
[ØØØ]   29:
[MUØ]   3Ø: begin
```

```
[UE5]   31:  clrscr; {Lösche Bildschirm}
[YZ1]   32:  new(neuer);
[CB4]   33:  {lege einen Datensatz an}
[YG2]   34:  erster:=neuer;
[P65]   35:  {halte ersten Datensatz fest}
[2W2]   36:  erster^.name:='';
[KF3]   37:  erster^.vorname:='';
[3F3]   38:  erster^.telefon:='';
[G84]   39:  erster^.naechster:=nil;
[7B5]   4Ø:  {lösche alle Komponenten und
[TW2]   41:   zeige auf nichts}
[545]   42:  write('Wie viele Namen möchten',
[PW2]   43:  ' Sie eingeben? ');
[LG2]   44:  readln(anzahl);
[VY2]   45:  {Lese Anzahl ein}
[RU3]   46:  writeln; {Leerzeile}
[PG2]   47:  neuer:=erster;
[594]   48:  {zeige auf ersten Satz}
[DG2]   49:  alter:=erster;
[594]   5Ø:  {zeige auf ersten Satz}
[S54]   51:  for index:=1 to anzahl do
[MUØ]   52:  begin
[YZ1]   53:   new(neuer);
[E85]   54:   {neuen Datensatz reservieren}
[ØD4]   55:   writeln(index,'. Eingabe:');
[R23]   56:   write('Vorname? ');
[RY3]   57:   readln(neuer^.vorname);
[HØ3]   58:   write('Nachname? ');
[193]   59:   readln(neuer^.name);
[F13]   6Ø:   write('Telefon? ');
[6Y3]   61:   readln(neuer^.telefon);
[G84]   62:   alter^.naechster:=neuer;
[D25]   63:   {mit letztem Satz verketten}
[XF1]   64:   writeln;
[M72]   65:   alter:=neuer;
[ZE4]   66:   {aktuellen Satz merken}
[ZNØ]   67:  end;
[ØØØ]   68:
[2G4]   69:  {alle Adressen und Telefon-
[9Z4]   7Ø:   nummern sind nun eingelesen,
[NØ5]   71:   die Kette kann nun geschlossen
[HL1]   72:   werden}
[ØØØ]   73:
[AV3]   74:  neuer^.naechster:=nil;
[SM5]   75:  {es folgen keine Elemente mehr}
[ØØØ]   76:
[WV4]   77:  {Liste ist erstellt, beginne
[L54]   78:   nun mit dem Sortieren}
[ØØØ]   79:
[TF4]   8Ø: for zaehler:=1 to anzahl do
[MUØ]   81:  begin
[DG2]   82:   alter:=erster;
[484]   83:   neuer:=alter^.naechster;
[4X4]   84:   {zeige auf ersten Datensatz}
[3M4]   85:   for index:=1 to anzahl-1 do
[ZY3]   86:   {benutze Bubble-Sort}
[MUØ]   87:   begin
[VW4]   88:    weiterer:=neuer^.naechster;
[375]   89:    {zeige auf nachfolgenden Satz}
[FM2]   9Ø:    if (neuer^.name>
[9D3]   91:        weiterer^.name) then
[MUØ]   92:    begin
[6W4]   93:     {nun sind drei Sätze in der
[V12]   94:      Reihenfolge
[CL3]   95:      alter-neuer-weiterer
[MU3]   96:      im Speicher, die in die
[V12]   97:      Reihenfolge
[CH3]   98:      alter-weiterer-neuer
[ZH4]   99:      gebracht werden müssen}
[BW4]  1ØØ:     merke:=weiterer^.naechster;
[V43]  1Ø1:     {merke den Zeiger}
[SW4]  1Ø2:     alter^.naechster:=weiterer;
[KW4]  1Ø3:     weiterer^.naechster:=neuer;
[D74]  1Ø4:     neuer^.naechster:=merke;
[RB2]  1Ø5:    end; {von if}
[V74]  1Ø6:    alter:=alter^.naechster;
[S44]  1Ø7:    {merke den letzten Satz}
[484]  1Ø8:    neuer:=alter^.naechster;
[XF4]  1Ø9:    {zeige auf nächsten Satz}
[4H3]  11Ø:   end; {von for index}
[TØ4]  111:  end; {von for zaehler}
[ØØØ]  112:
[5B5]  113:  {Jetzt ist alles sortiert, also
[5R4]  114:   gebe die fertige Liste aus}
[ØØØ]  115:
[GB5]  116:  clrscr; {Bildschirm löschen}
[8D4]  117:  neuer:=erster^.naechster;
[S54]  118:  for index:=1 to anzahl do
[MUØ]  119:  begin
[6Z4]  12Ø:   writeln(index,'. Datensatz:');
[EP3]  121:   writeln(neuer^.name);
[F84]  122:   writeln(neuer^.vorname);
[774]  123:   writeln(neuer^.telefon);
[684]  124:   neuer:=neuer^.naechster;
[XF1]  125:   writeln;
[ZNØ]  126:  end;
[KNØ]  127: end.
```

Listing 3. Eingabe und Sortieren einer Liste über einen dynamischen Record (»namzeig.pas«)

Rekursive Algorithmen unter Turbo Pascal

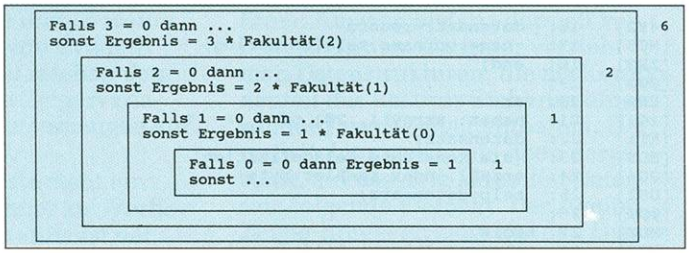


Bild 1. Die Funktionsweise der rekursiven Fakultätsberechnung wird anhand des Anwendungsbeispiels besonders deutlich

Funktionen und Prozeduren gehören zu den wichtigsten Sprachelementen von Turbo Pascal. Sie erleichtern es, eine Aufgabenstellung in übersichtliche Bereiche zu unterteilen, und führen auf diese Weise zu strukturierten und kompakten Programmen. Mit der Rekursion stellt sich eine besondere Variante von Unterprogrammen vor, die sich zur Lösung von komplexen Problemen anbietet und sich aufgrund ihres verständlichen Aufbaus besonders dafür eignet.

Um mehrmalige Anweisungen nicht wiederholt in einen Quelltext einfügen zu müssen, benutzen Programmierer in der Regel Unterprogramme beziehungsweise Prozeduren, die die Programmlänge und Entwicklungszeit erheblich verkürzen.
Wie Sie bereits wissen, kann eine Funktion oder Prozedur Aufrufe von bereits bestehenden Funktionen oder Prozeduren enthalten. Neben diesen in fast allen Programmiersprachen enthaltenen Standardelementen gibt es eine spezielle Variante, die auch in Turbo Pascal zu finden ist. Jede Prozedur beziehungsweise jedes Unterprogramm ist in der Lage, sich selbst aufzurufen. Dieser Vorgang wird »Rekursion« genannt (lateinisch »recurrere«, entspricht »zurücklaufen«).
Vor der wissenschaftlichen Erläuterung der Rekursion und ihrer Umsetzung in Turbo Pascal erhalten Sie Einblick in die zugrundeliegenden Denkprozesse und die vielfältigen Einsatzmöglichkeiten.
Die Rekursion ist ein sehr aufregendes Gebiet der Mathematik und hat aufgrund einiger theoretisch unlogischer Merkmale sowohl Wissenschaftler als auch Philosophen zu langen Diskussionen und tiefen Selbstzweifeln getrieben.
Ihnen ist sicherlich die folgende Geschichte, die eine interessante Frage aufwirft, in dieser oder einer ähnlichen Version schon begegnet:
»Ein Handwerksbursche kam nach langem Marsch in ein kleines Dorf. Da es sehr heiß war und er den ganzen Tag über gewandert war, trieb es ihn in den einzigen, im Ort liegenden Gasthof, wo er seinen mächtigen Hunger stillte und seinen schier unbezwingbaren Durst löschte. Nach Speis und Trank gelüstete es ihn nach einem reinigenden Bade und einer kräftigen Rasur. Er bat den Wirt um Rat, und der gab ihm bereitwillig Auskunft: "In unserem Dorf lebt ein Friseur, der im ganzen Umkreis hoch angesehen ist, da er seine Zunft würdig und pflichterfüllt vertritt. Er hat allerdings die Eigenart, ausschließlich die Männer zu rasieren, die sich nicht selbst rasieren. Ihr werdet ihn leicht finden, da er als gepflegtester Mann in unserem Dorfe gilt und aus diesem Grund schon von weitem zu erkennen ist."

Die Rekursion ähnelt dem menschlichen Denken in vielerlei Hinsicht

Der Handwerksbursche machte sich auf den Weg und dachte die ganze Zeit darüber nach, ob der eigenartige Friseur sich wohl selbst rasiere. "Wenn er sich nicht selbst rasiert, bedeutet das, daß er zu den Männern gehört, die sich nicht rasieren, also rasiert er sich. Rasiert er sich hingegen, dann gehört er zu der Gruppe von Männern, die sich selbst rasieren, also rasiert er sich nicht." Der Geist des bedauernswerten Gesellen war von diesen Gedanken überfordert und verwirrte sich so sehr, daß der Bursche bis zu seinem Lebensende verrückt durch die Welt lief.«
Das traurige Ende der Geschichte betrübt sehr; das soll uns jedoch nicht davon ablenken, zu der Funktionsweise der Rekursion zurückzukehren. Der vorgestellte Widerspruch läßt sich anhand eines mathematischen Beispiels verdeutlichen, das unter dem Namen »Russellsche Antinomie« bekannt ist.
Das Beispiel basiert auf dem Grundgedanken, eine mathematisch definierte Menge zu finden, die alle Mengen enthält. Bevor Sie an die nobelpreisträchtige Lösung dieses Rätsels gehen, bedenken Sie bitte, daß die Aufgabe unlösbar ist. Jede Menge, die sämtliche anderen Mengen enthält, bildet eine neue Menge. Da die Menge aber alle Mengen enthalten soll, muß sie auch sich selbst enthalten und bildet damit wieder eine neue erweiterte Menge. Dieser Selbstbezug läßt sich unendlich fortsetzen und führt damit zu keinem verstandesmäßig erfaßbaren Ergebnis.
Um Ihnen die Funktionsweise der Rekursion zu verdeutlichen, erfahren Sie ihren Aufbau im folgenden anhand von konkreten Programmbeispielen. Viele Probleme sind rekursiver Natur; damit Sie verstehen, wie die Lösungen aufgebaut sind, werden sie mit den äquivalenten iterativen Lösungen verglichen. Die Iteration (lateinisch »iterare«, entspricht »wiederholen«) ist die bekannteste und am meisten eingesetzte Programmiertechnik, die anhand von Wiederholungsanweisungen beziehungsweise Schleifen Aufgaben löst.
Zuerst wird die Berechnung der Fakultät einer natürlichen Zahl inklusive der 0 betrachtet. Die Fakultät einer Zahl »n«, die man auch mit »n!« bezeichnet, ist gleich 1, wenn »n« gleich 0 ist, ansonsten entspricht sie dem Produkt aus »n« und der Fakultät von »n – 1«. Formal ausgedrückt bedeutet das:

```
0! = 1
n! = n * (n - 1)! für n > 0
```

Natürlich können Sie jetzt ausschließlich Schleifen benutzen, um das erwünschte Ergebnis zu erhalten (Listing 1).

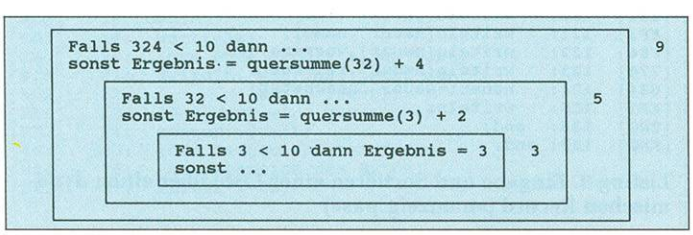


Bild 2. Um die Quersumme von 324 zu berechnen, braucht der rekursive Algorithmus nur drei Schritte

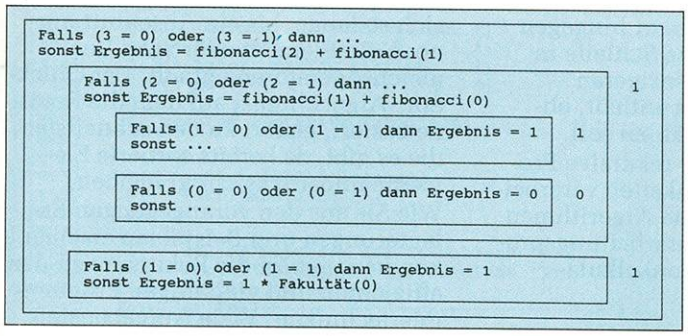


Bild 3. Um eine Fibonacci-Zahl zu errechnen, muß das Programm aus Listing 7 zwei Abbruchkriterien beachten

Zuerst definiert die Anweisung »Var« die verwendeten Schleifenvariablen und Zwischenspeicher. Nach der Eingabe einer nicht negativen Zahl nimmt das Programm zuerst für die Fakultät den Wert 1 an. Wenn die eingegebene Zahl gleich 0 ist, bleibt es bei diesem Ergebnis, ansonsten multipliziert die Routine die Fakultät mit jedem Schleifenwert.

Rekursionen speichern die Zwischenergebnisse auf dem internen Stapel

Sie erhalten zwar das richtige Ergebnis, aber eigentlich ist der eingeschlagene Weg ziemlich unbefriedigend. Das liegt einerseits daran, daß beispielsweise die Fakultät für 0 nicht errechnet, sondern vor dem eigentlichen Rechenvorgang festgelegt wird, weil sie die kleinste Zahl darstellt. Darüber hinaus entsprechen die benutzten Programmschritte nicht exakt den vorher festgelegten Konventionen.
Aufgrund der Definition der Fakultät bietet es sich an, den folgenden Algorithmus zu benutzen:

```
Falls n = 1
dann ist das Ergebnis = 1
sonst Ergebnis = n * (n-1)!
```

Dieses Verfahren findet sich in Listing 2 wieder, das schon durch seinen Aufbau wesentlich übersichtlicher und einfacher als Listing 1 wirkt. Die Funktion »fakultaet« ruft sich selber so oft auf, bis die Abbruchbedingung erfüllt ist. Die Funktionsweise läßt sich anhand einiger Beispiele leicht nachvollziehen. Angenommen, das Programm soll die Fakultät von 0 berechnen, also:

```
n = 0
```

Die Funktion trifft beim Überprüfen der Zahl »n« sofort auf die Abbruchbedingung und weist sich selbst richtigerweise den Wert 1 zu.
Nehmen Sie als nächstes Beispiel die Zahl 1. Die Funktion überprüft die Abbruchbedingung. Da sie nicht erfüllt ist, multipliziert die Funktion die Zahl 1 mit der Fakultät von 0 (»fakultaet: = n*fakultaet(n-1)«). Die Abbruchbedingung ist in diesem Fall erfüllt, also erhält »fakultaet« den Wert 1. Die Routine multipliziert daraufhin die gespeicherten Werte und erhält als Ergebnis den korrekten Wert

```
1 * 1 = 1
```

Das Ablaufdiagramm aus Bild 1 zeigt Ihnen, wie das Programm mit größeren Zahlen arbeitet.

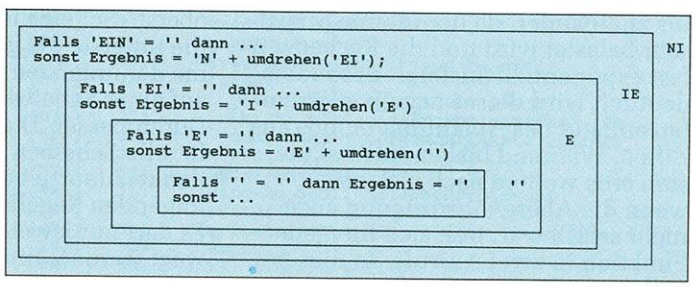


Bild 4. Der rekursive Algorithmus zum Umdrehen eines Wortes berücksichtigt auch den Sonderfall, daß die umzukehrende Zeichenkette leer ist

Bevor Sie sich mit weiteren Beispielen beschäftigen, erhalten Sie Einblick in den internen Ablauf von Rekursionen. Die Routine aus Listing 2 speichert die Zwischenwerte auf dem programminternen Stapel (englisch »stack«) und verknüpft sie nach Erreichen der Abbruchbedingung miteinander. Durch den Aufruf einer rekursiven Funktion oder Prozedur entsteht eine Verschachtelung von Aufrufen, die bei erfüllter Abbruchbedingung aufgelöst wird.

Jeder Rekursionsaufruf bildet eine unabhängige Variablenebene

Jeder Aufruf für sich ist unabhängig von den anderen und wird auch dann korrekt ausgeführt, wenn die lokalen Variablen denselben Namen haben. Das Programm erzeugt für jeden Aufruf eine eigene Variablenebene.

Ähnlich wie das Fakultätsprogramm funktioniert der Algorithmus »q(n)« zur Berechnung der Quersumme einer Zahl. Man beschreibt ihn auch durch folgende generell anwendbare Formel:

```
q(n) = n für n < 10, sonst
q(n) = q(n div 10) + n mod 10
```

Diese Definition läßt sich, wie Listing 3 zeigt, direkt in eine rekursive Routine verwandeln. Beachten Sie bitte, daß das Programm die Quersumme von Zahlen so lange ermittelt, bis Sie eine 0 eingeben. Bild 2 zeigt die Funktionsweise des Unterprogramms in der praktischen Anwendung.
Im Vergleich zur iterativen Lösung aus Listing 4 wirkt die Rekursion wesentlich übersichtlicher und leichter verständlich. Darüber hinaus arbeitet sie mit weniger Variablen und ohne Schleifen.
Bisher wurden nur ganze Zahlen behandelt. Das Potenzieren einer Zahl zeigt, daß Rekursionen auch mit reellen Zahlen korrekt arbeiten. Die Potenz für einen ganzzahligen, nicht negativen Exponenten »n« ist durch folgende Definition festgelegt:

```
0^n = 0 für n > 0
a^n = 1 für n = 0
a^n,n = (a^n-1 * a) für n > 0
```

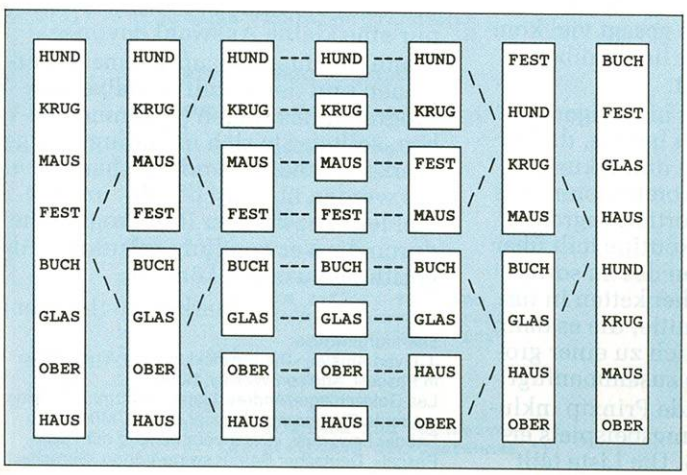


Bild 5. Diese Abbildung zeigt besonders deutlich, wie das Programm aus Listing 13 die zu sortierenden Zeichenketten in kleine Abschnitte unterteilt und damit eine hohe Geschwindigkeit erreicht

Dieses Schema findet sich in Listing 5 wieder, das im Vergleich zu der entsprechenden iterativen Routine in Listing 6 mit weniger Programmzeilen auskommt.
Das nächste Beispiel veranschaulicht, daß es dabei auch nicht sinnvoll sein kann, Rekursionen zu benutzen. Zur Demonstration wird die Fibonacci-Folge (»f_n«) benutzt, die wie folgt definiert ist:

$f_0 = 0$
$f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$ für $n \geq 2$

Das bedeutet, daß für größere Zahlen zwei Rekursionsaufrufe nötig sind, wie Listing 7 zeigt.
Wie Sie sehen, treten bei der Berechnung der Fibonacci-Folge die Nachteile der Rekursion deutlich hervor. Um beispielsweise die Fibonacci-Zahl von 11 zu errechnen, braucht das Programm aus Listing 8 zehn Schleifendurchläufe, während die rekursive Lösung immerhin 287 Funktionsaufrufe ausführt. Übersichtlichkeit und Effizienz stehen in diesem Beispiel in einem schlechten Verhältnis zueinander, da der interne Stapel sehr belastet wird und die Rechenzeit fast exponentiell ansteigt. Besonders deutlich wird dieses negative Verhalten anhand des Ablaufdiagramms aus Bild 3. Während bisher jede Rekursion eine weitere nach sich zog, wenn die Abbruchbedingung noch nicht erfüllt war, teilt sich hier jede Funktion in zwei Aufrufe. In diesem Fall ist die iterative Routine aus Listing 8 sinnvoller. Sie ist zwar unübersichtlicher als die Rekursion, überzeugt jedoch durch höhere Geschwindigkeit.

Eine unangemessen hohe Zahl von Rekursionsaufrufen führt zu uneffektiven Ergebnissen

Daß die Stapelbelastung nicht ohne Begrenzung fortgesetzt werden kann, zeigt Listing 9, das den Stapel ohne ein effektives Ergebnis mit Werten belegt. Probieren Sie es aus, um herauszufinden, wann ein Programm die sinnvollen Grenzen überschreitet.
Listing 10 und 11 verdeutlichen, daß sich nicht nur numerische Aufgaben mit Hilfe der Rekursion und Iteration lösen lassen. Die Routinen drehen eine Zeichenkette um. Das Ablaufdiagramm in Bild 4 zeigt Ihnen die Verfahrensweise anhand eines Beispiels. Die Rekursion aus Listing 10 arbeitet auch dann zuverlässig, wenn sie eine leere Zeichenkette als Parameter enthält, da sie diesen Sonderfall in ihrer Abbruchbedingung berücksichtigt.
Die Iteration funktioniert hingegen nicht korrekt, weil die Schleife in diesem Fall einen niedrigeren Anfangs- als Endwert enthält, obwohl sie rückwärts zählen soll.
Das Einsatzgebiet für rekursive Programme, die Zeichenketten verarbeiten, ist groß. Ähnliche Algorithmen finden sich in Textverarbeitungsprogrammen und Tabellenkalkulationen.

Das Problem der Türme von Hanoi ist das klassische Beispiel für den Einsatz von Rekursionen

Als klassisches Beispiel für die Arbeit mit Rekursionen gilt das Problem der »Türme von Hanoi«. Ausgegangen wird hierbei von drei nebeneinanderstehenden Stäben. Auf dem linken Stab befindet sich eine mit »n« bezeichnete Anzahl von Scheiben mit verschiedenen Durchmessern. Die oberste Scheibe ist die kleinste und die unterste die größte. Mit Ausnahme der untersten liegt jede Scheibe auf einer Scheibe mit größerem Durchmesser. Die Aufgabe liegt darin, alle Scheiben vom linken Stab auf den rechten zu befördern, wobei die folgenden Regeln zu beachten sind: Es darf nur jeweils eine Scheibe bewegt werden, und jede Scheibe muß auf einer Scheibe mit größerem Durchmesser liegen, außer wenn es sich um die jeweils unterste handelt.
Der Algorithmus zu diesem Problem läßt sich auf drei Grundschritte begrenzen: »Bewege erstens »n – 1« Scheiben vom linken Stab auf den mittleren, setze zweitens die Scheibe vom linken auf den rechten Stab, und bewege drittens die »n – 1« Scheiben vom mittleren Stab auf den rechten.«
Die rekursive Routine ist in Listing 12 aufgeführt. Sie bildet eine elegante und sehr kurze Lösung der schwierigen Aufgabenstellung. Iterative Funktionen sind wie gesagt viel komplizierter und wesentlich unübersichtlicher aufgebaut.
Als Ergebnis unserer bisherigen Bemühungen stellt sich heraus, daß es sinnvoll ist, iterative und rekursive Lösungsansätze zu kombinieren. Als Beispiel dient das Sortierprogramm aus Listing 13. Die Routine teilt über rekursive Funktionen das zu sortierende Feld aus Zeichenketten in immer kleinere Abschnitte, die es dann bearbeitet und letztlich zu einer großen, sortierten Liste zusammenfügt.
Das zugrundeliegende Prinzip inklusive eines Anwendungsbeispiels ersehen Sie aus Bild 5. Die Liste teilt sich so lange, bis ein Abschnitt nur noch ein Element oder zwei Vergleichsoperatoren enthält. Der Quick-Sort-Algorithmus, auf dem die Routine basiert, gehört zu den schnellsten, die es gibt, da bereits sortierte Elemente unberücksichtigt bleiben.
Wie Sie aus den vorangegangen Erläuterungen und Beispielen ersehen konnten, gehört die Rekursion zu den effizienten und vielfältigen Programmiertechniken. Viele Aufgabenstellungen lassen sich mit diesem Verfahren elegant und unkompliziert lösen. Es eignet sich auch dazu, komplexe Themen auf einfachem Wege zu bewältigen.
Die Hauptnachteile der Rekursion liegen im Speicherverbrauch beziehungsweise in der Stapelbelastung und in der bei komplexen Problemen negativ auffallenden Ausführungsgeschwindigkeit.

Licht und Schatten: Rekursionen bewältigen komplexe Aufgaben, können aber den Rechner sehr beanspruchen

Wie das letzte Beispiel zeigte, ist es in vielen Fällen sinnvoll, iterative und rekursive Lösungsansätze zu kombinieren und damit ein in der Effizienz optimales Ergebnis zu erzielen.
Um die Funktionsweise der Rekursion zu verstehen, sollten Sie versuchen, auf der Basis der vorgestellten Grundlagen eigene Programme zu entwickeln. Der Einsatzbereich reicht von einfachen arithmetischen Aufgaben, wie beispielsweise der Suche nach dem größten gemeinsamen Teiler mehrerer Zahlen, bis zu schwierigen Problemen wie der binären Suche.
Gerade zu diesem Themengebiet sind auf dem Markt viele interessante und gute Bücher erhältlich (die Literaturhinweise am Ende des Artikels sind nur eine kleine Auswahl davon).
Wenn Sie Interesse am Thema Rekursionen und den damit erzielbaren Programmleistungen bekommen haben, sollten Sie sich unbedingt einige Werke anschaffen und durcharbeiten. Sie werden überrascht sein, um wieviel leistungsfähiger Ihre Programme durch die Verwendung rekursiver Algorithmen werden können.

(Markus Zietlow/dm)

Literaturhinweise:
J.J. van Amstel, »Die Entwicklung von Algorithmen in Pascal«, Addison-Wesley, Bonn
Les Goldschlager/Andrew Lister, »Informatik – eine moderne Einführung«, Hanser-Verlag, München
Frieder Lewerenz, »Datenverarbeitung mit Turbo Pascal«, Deutscher Betriebswirte-Verlag, Gernsbach

```
[2U4]    1: (* Iterative Berechnung der *)
[BG4]    2: (* Fakultät einer Zahl      *)
[XG4]    3: (* Autor: Markus Zietlow    *)
[F53]    4: (* Listing 1                *)
[ØØØ]    5:
[N55]    6: program iterative_fakultaet;
[ØØØ]    7:
[RNØ]    8: var
[CS3]    9:   zahl, zaehler:integer;
[Y53]   1Ø:   fakultaet:longint;
[ØØØ]   11:
[MUØ]   12: begin
[BR5]   13:   write('Programm zur iterativen ');
[AL5]   14:   write('Berechnung der Fakultät ');
[7M3]   15:   writeln('einer Zahl');
[HU4]   16:   write('------------------------------');
[1Z4]   17:   writeln('--------------------------');
[XF1]   18:   writeln;
[Y26]   19:   write('Geben Sie bitte die Zahl ein: ');
[Ø32]   2Ø:   readln(zahl);
[Y42]   21:   fakultaet:=1;
[VB2]   22:   if zahl<>Ø then
[VZ3]   23:     for zaehler:=1 to zahl do
[EØ5]   24:       fakultaet:=fakultaet*zaehler;
[M36]   25:   write('Die Fakultät der Zahl ',zahl);
[XV3]   26:   write(' (',zahl,'!) ist: ');
[CL3]   27:   writeln(fakultaet:8);
[XF1]   28:   writeln;
[KNØ]   29: end.
```

Listing 1. Während die iterative Berechnung der Fakultät einer Zahl (»it_fakul.pas«) unstrukturiert aussieht, ...

```
[UU4]    1: (* Rekursive Berechnung der *)
[BG4]    2: (* Fakultät einer Zahl      *)
[XG4]    3: (* Autor: Markus Zietlow    *)
[G53]    4: (* Listing 2                *)
[ØØØ]    5:
[M65]    6: program rekursive_fakultaet;
[ØØØ]    7:
[RNØ]    8: var
[G92]    9:   zahl:longint;
[ØØØ]   1Ø:
[TW6]   11: function fakultaet(n : longint) : longint;
[MUØ]   12: begin
[7N1]   13:   if n=Ø then
[812]   14:     fakultaet:=1
[ZRØ]   15:   else
[YE4]   16:     fakultaet:=n*fakultaet(n-1)
[ZNØ]   17: end;
[ØØØ]   18:
[MUØ]   19: begin
[AS5]   2Ø:   write('Programm zur rekursiven ');
[AL5]   21:   write('Berechnung der Fakultät ');
[7M3]   22:   writeln('einer Zahl');
[HU4]   23:   write('------------------------------');
[1Z4]   24:   writeln('--------------------------');
[XF1]   25:   writeln;
[Y26]   26:   write('Geben Sie bitte die Zahl ein: ');
[Ø32]   27:   readln(zahl);
[M36]   28:   write('Die Fakultät der Zahl ',zahl);
[XV3]   29:   write(' (',zahl,'!) ist: ');
[9M4]   3Ø:   writeln(fakultaet(zahl):8);
[XF1]   31:   writeln;
[KNØ]   32: end.
```

Listing 2. ... wirkt die rekursive Lösung (»re_fakul.pas«) übersichtlich und leicht verständlich

```
[8X4]    1: (* Rekursives Ermitteln der *)
[CC3]    2: (* Quersumme                *)
[XG4]    3: (* Autor: Markus Zietlow    *)
[H53]    4: (* Listing 3                *)
[ØØØ]    5:
[G85]    6: program rekursive_quersumme;
[ØØØ]    7:
[RNØ]    8: var
[FP3]    9:   eingabe_zahl:longint;
[ØØØ]   1Ø:
[AC7]   11: function quersumme(zahl:longint):longint;
[RNØ]   12: var
[GØ4]   13:   letzte_ziffer:integer;
[ØB2]   14:   rest:longint;
[MUØ]   15: begin
[1B2]   16:   if zahl<1Ø then
[XT2]   17:     quersumme:=zahl
[ZRØ]   18:   else
[MUØ]   19:     begin
[NV2]   2Ø:       rest:=zahl div 1Ø;
[YR4]   21:       letzte_ziffer:=zahl mod 1Ø;
[4U4]   22:       quersumme:=quersumme(rest)+
[3E2]   23:                  letzte_ziffer
[4MØ]   24:     end
[ZNØ]   25: end;
[ØØØ]   26:
[MUØ]   27: begin
[PR5]   28:   write('Programm zum rekursiven ');
[6M5]   29:   write('Ermitteln der Quersumme ');
[7M3]   3Ø:   writeln('einer Zahl');
[HU4]   31:   write('------------------------------');
[1Z4]   32:   writeln('--------------------------');
[XF1]   33:   writeln;
[P31]   34:   repeat
[D55]   35:     write('Geben Sie die Zahl ein: ');
[DK3]   36:     readln(eingabe_zahl);
[D45]   37:     write('Die Quersumme der Zahl ');
[RM4]   38:     write(eingabe_zahl,' ist: ');
[MT5]   39:     writeln(quersumme(eingabe_zahl));
[2E1]   4Ø:     writeln
[RM3]   41:   until eingabe_zahl=Ø;
[KNØ]   42: end.
```

Listing 3. Wenn Sie die Quersumme einer Zahl mittels einer Rekursion errechnen (»re_quer.pas«), kommen Sie mit einer globalen und zwei lokalen Variablen aus

```
[EW4]    1: (* Iterativer Ermitteln der *)
[CC3]    2: (* Quersumme                *)
[XG4]    3: (* Autor: Markus Zietlow    *)
[K53]    4: (* Listing 4                *)
[ØØØ]    5:
[H75]    6: program iterative_quersumme;
[ØØØ]    7:
[RNØ]    8: var
[GØ4]    9:   letzte_ziffer:integer;
[Y35]   1Ø:   eingabe_zahl, quersumme, rest,
[MF4]   11:   zwischenspeicher:longint;
[ØØØ]   12:
[MUØ]   13: begin
[RP5]   14:   write('Programm zum iterativen ');
[6M5]   15:   write('Ermitteln der Quersumme ');
[7M3]   16:   writeln('einer Zahl');
[HU4]   17:   write('------------------------------');
[1Z4]   18:   writeln('--------------------------');
[XF1]   19:   writeln;
[P31]   2Ø:   repeat
[D55]   21:     write('Geben Sie die Zahl ein: ');
[DK3]   22:     readln(eingabe_zahl);
[Z72]   23:     quersumme:=Ø;
[PM5]   24:     zwischenspeicher:=eingabe_zahl;
[P31]   25:     repeat
[G94]   26:       rest:=eingabe_zahl div 1Ø;
[B76]   27:       letzte_ziffer:=eingabe_zahl mod 1Ø;
[6F6]   28:       quersumme:=quersumme+letzte_ziffer;
[W63]   29:       eingabe_zahl:=rest
[E72]   3Ø:     until rest=Ø;
[ØM5]   31:     eingabe_zahl:=zwischenspeicher;
[D45]   32:     write('Die Quersumme der Zahl ');
[2K6]   33:     write(eingabe_zahl,' ist: ',quersumme);
[2E1]   34:     writeln
[RM3]   35:   until eingabe_zahl=Ø;
[KNØ]   36: end.
```

Listing 4. Um die Quersumme einer Zahl über den iterativen Weg herauszufinden (»it_quer.pas«), benötigen Sie insgesamt fünf globale Variablen

```
[3N5]    1: (* Demonstration der rekursiven *)
[P15]    2: (* Programmierung anhand des    *)
[Y14]    3: (* Potenzierens                 *)
[XU4]    4: (* Autor: Markus Zietlow        *)
[LF3]    5: (* Listing 5                    *)
[ØØØ]    6:
[MM4]    7: program rekursive_potenz;
[ØØØ]    8:
[RNØ]    9: var
[DT2]   1Ø:   grundzahl:real;
[AY2]   11:   hochzahl:integer;
[ØØØ]   12:
[ØØØ]   13:
[UZ4]   14: function potenz(basis : real;
[TD4]   15:                 exponent : integer) : real;
[MUØ]   16: begin
[Ø13]   17:   if exponent=Ø then
[LK1]   18:     potenz:=1
[ZRØ]   19:   else
[DC2]   2Ø:     if basis=Ø then
[HK1]   21:       potenz:=Ø
[ZRØ]   22:     else
[9R3]   23:       potenz:=potenz(basis,
[NU2]   24:                      exponent-1)*basis
[ZNØ]   25: end;
[ØØØ]   26:
[MUØ]   27: begin
[AS5]   28:   write('Programm zur rekursiven ');
[816]   29:   writeln('Berechnung einer Potenz');
[HU4]   3Ø:   write('------------------------------');
[MM3]   31:   writeln('----------------');
[XF1]   32:   writeln;
[L56]   33:   write('Geben Sie bitte die Basis ein: ');
[913]   34:   readln(grundzahl);
[6D6]   35:   write('Geben Sie den Exponenten ein: ');
```

Listing 5. Die rekursive Berechnung einer Potenzzahl (»rek_pot.pas«) enthält im Gegensatz zu den bisher vorgestellten Algorithmen zwei Abbruchkriterien

```
[FT2]    36:   readln(hochzahl);
[RF5]    37:   write('Die Potenz hat den Wert: ');
[MU6]    38:   writeln(potenz(grundzahl,hochzahl):8:2);
[XF1]    39:   writeln;
[KNØ]    4Ø: end.
```

Listing 5. Die rekursive Berechnung einer Potenzzahl (»rek_pot.pas«) enthält im Gegensatz zu den bisher vorgestellten Algorithmen zwei Abbruchkriterien

```
[9M5]     1: (* Demonstration der iterativen *)
[P15]     2: (* Programmierung anhand des    *)
[Y14]     3: (* Potenzierens                 *)
[XU4]     4: (* Autor: Markus Zietlow        *)
[MF3]     5: (* Listing 6                    *)
[ØØØ]     6:
[NL4]     7: program iterative_potenz;
[ØØØ]     8:
[RNØ]     9: var
[725]    1Ø:   grundzahl, zwischenwert:real;
[XE4]    11:   hochzahl, zaehler:integer;
[ØØØ]    12:
[ØØØ]    13:
[UZ4]    14: function potenz(basis : real;
[TD4]    15:          exponent : integer) : real;
[MUØ]    16: begin
[Ø13]    17:   if exponent=Ø then
[LK1]    18:     potenz:=1
[ZRØ]    19:   else
[DC2]    2Ø:     if basis=Ø then
[HK1]    21:       potenz:=Ø
[ZRØ]    22:     else
[LT2]    23:       zwischenwert:=1;
[CR4]    24:       for zaehler:=1 to exponent do
[TY5]    25:         zwischenwert:=zwischenwert*basis;
[UR3]    26:       potenz:=zwischenwert
[ZNØ]    27: end;
[ØØØ]    28:
[MUØ]    29: begin
[BR5]    3Ø:   write('Programm zur iterativen ');
[816]    31:   writeln('Berechnung einer Potenz');
[HU4]    32:   write('-------------------------------');
[MM3]    33:   writeln('---------------');
[XF1]    34:   writeln;
[L56]    35:   write('Geben Sie bitte die Basis ein: ');
[913]    36:   readln(grundzahl);
[767]    37:   write('Geben Sie bitte den Exponenten ein
          : ');
[FT2]    38:   readln(hochzahl);
[RF5]    39:   write('Die Potenz hat den Wert: ');
[MU6]    4Ø:   writeln(potenz(grundzahl,hochzahl):8:2);
[XF1]    41:   writeln;
[KNØ]    42: end.
```

Listing 6. Auch die iterative Berechnung einer Potenz (»it_pot.pas«) läßt sich in Form einer Funktion definieren, die allerdings die Hilfsvariable »zwischenwert« benutzt

```
[WZ4]     1: (* Rekursive Berechnung der   *)
[HT4]     2: (* Fibonacci-Folge einer Zahl *)
[ZN4]     3: (* Autor: Markus Zietlow      *)
[RA3]     4: (* Listing 7                  *)
[ØØØ]     5:
[Z35]     6: program rekursive_fibonacci;
[ØØØ]     7:
[RNØ]     8: var
[G92]     9:   zahl:longint;
[ØØØ]    1Ø:
[VT6]    11: function fibonacci(n : longint) : longint;
[MUØ]    12: begin
[U73]    13:   if (n=Ø) or (n=1) then
[EZ1]    14:     fibonacci:=n
[ZRØ]    15:   else
[U24]    16:     fibonacci:=fibonacci(n-1)+
[U52]    17:                fibonacci(n-2)
[ZNØ]    18: end;
[ØØØ]    19:
[MUØ]    2Ø: begin
[AS5]    21:   write('Programm zur rekursiven ');
[PH6]    22:   write('Berechnung der Fibonacci-Folge ');
[7M3]    23:   writeln('einer Zahl');
[HU4]    24:   write('-------------------------------');
[HU4]    25:   write('-------------------------------');
[NA2]    26:   writeln('---');
[XF1]    27:   writeln;
[Y26]    28:   write('Geben Sie bitte die Zahl ein: ');
[Ø32]    29:   readln(zahl);
[N16]    3Ø:   write('Die Fibonacci-Folge der Zahl ');
[1B6]    31:   writeln(zahl,' ist: ',fibonacci(zahl));
[XF1]    32:   writeln;
[KNØ]    33: end.
```

Listing 7. Die Berechnung der Fibonacci-Folge (»rek_fib.pas«) verdeutlicht, welche Vorteile die Rekursion ...

```
[4Z4]     1: (* Iterative Berechnung der   *)
[HT4]     2: (* Fibonacci-Folge einer Zahl *)
[ZN4]     3: (* Autor: Markus Zietlow      *)
[SA3]     4: (* Listing 8                  *)
[ØØØ]     5:
[Ø35]     6: program iterative_fibonacci;
[ØØØ]     7:
[RNØ]     8: var
[4X3]     9:   fibonacci, zaehler, zahl,
[MF4]    1Ø:   zwischenspeicher:longint;
[ØØØ]    11:
[MUØ]    12: begin
[BR5]    13:   write('Programm zur iterativen ');
[PP5]    14:   write('Berechnung der Fibonacci-');
[WG4]    15:   writeln('Folge einer Zahl');
[HU4]    16:   write('-------------------------------');
[HU4]    17:   write('-------------------------------');
[NA2]    18:   writeln('---');
[XF1]    19:   writeln;
[Y26]    2Ø:   write('Geben Sie bitte die Zahl ein: ');
[Ø32]    21:   readln(zahl);
[HE3]    22:   zwischenspeicher:=1;
[622]    23:   fibonacci:=Ø;
[KX1]    24:   zaehler:=Ø;
[9T3]    25:   while zaehler<>zahl do
[MUØ]    26:     begin
[W93]    27:       zaehler:=zaehler+1;
[YD3]    28:       fibonacci:=fibonacci+
[XØ3]    29:                  zwischenspeicher;
[XX4]    3Ø:       zwischenspeicher:=fibonacci-
[XØ3]    31:                        zwischenspeicher;
[ZNØ]    32:     end;
[N16]    33:   write('Die Fibonacci-Folge der Zahl ');
[W95]    34:   writeln(zahl,' ist: ',fibonacci);
[XF1]    35:   writeln;
[KNØ]    36: end.
```

Listing 8. ... gegenüber der unübersichtlichen iterativen Lösung (»it_fib.pas«) des Problems besitzt

```
[GM4]     1: (* Belastung des internen    *)
[GZ4]     2: (* Stapels durch Rekursionen *)
[GN4]     3: (* Autor: Markus Zietlow     *)
[AA3]     4: (* Listing 9                 *)
[ØØØ]     5:
[PS3]     6: program stapel_test;
[ØØØ]     7:
[RNØ]     8: var
[ET2]     9:   anzahl:longint;
[ØØØ]    1Ø:
[TF6]    11: function rekursion(zaehler : longint):
[BG1]    12:                    longint;
[MUØ]    13: begin
[R86]    14:   write('Rekursiver Funktionsaufruf ');
[BØ3]    15:   writeln(zaehler);
[YD5]    16:   rekursion:=rekursion(zaehler+1)
[ZNØ]    17: end;
[ØØØ]    18:
[MUØ]    19: begin
[1L6]    2Ø:   write('Test der maximalen Belastung ');
[194]    21:   write('des Stapels durch ');
[645]    22:   writeln('Rekursionsaufrufe');
[HU4]    23:   write('-------------------------------');
[HU4]    24:   write('-------------------------------');
[X42]    25:   writeln('--');
[XF1]    26:   writeln;
[1P1]    27:   anzahl:=1;
[GR4]    28:   writeln(rekursion(anzahl));
[KNØ]    29: end.
```

Listing 9. Mit dieser Testroutine (»staptest.pas«) finden Sie heraus, welche Belastung der Stapel (»Stack«) verkraftet

```
[S75]     1: (* Umdrehen einer Zeichenkette *)
[M44]     2: (* durch Rekursion             *)
[AR4]     3: (* Autor: Markus Zietlow       *)
[WC3]     4: (* Listing 1Ø                  *)
[ØØØ]     5:
[S75]     6: program rekursives_umdrehen;
[ØØØ]     7:
[RNØ]     8: var
[SN3]     9:   eingabe_wort:string;
[ØØØ]    1Ø:
[737]    11: function umdrehen(wort : string) : string;
[RNØ]    12: var
[BL5]    13:   letzter_buchstabe, rest:string;
[MUØ]    14: begin
[MG3]    15:   if length(wort)=Ø then
[DH2]    16:     umdrehen:=wort
[ZRØ]    17:   else
[MUØ]    18:     begin
[XM5]    19:       rest:=copy(wort,1,length(wort)-1);
[UG3]    2Ø:       letzter_buchstabe:=
[3C4]    21:         copy(wort,length(wort),1);
[525]    22:       umdrehen:=letzter_buchstabe+
[7C2]    23:                 umdrehen(rest)
[4MØ]    24:     end
[ZNØ]    25: end;
[ØØØ]    26:
```

```
[MUØ]   27: begin
[K96]   28:   write('Programm zum Umdrehen eines ');
[VW5]   29:   writeln('Wortes über Rekursion');
[HU4]   3Ø:   write('-----------------------------');
[5X3]   31:   writeln('------------------');
[XF1]   32:   writeln;
[P31]   33:   repeat
[8X4]   34:     write('Geben Sie bitte das zu ');
[Z45]   35:     writeln('drehende Wort ein: ');
[4N3]   36:     readln(eingabe_wort);
[UE4]   37:     write('Die Umkehrung von "');
[Z75]   38:     write(eingabe_wort,'" lautet "');
[416]   39:     writeln(umdrehen(eingabe_wort),'".');
[2E1]   4Ø:     writeln
[5U3]   41:   until eingabe_wort='';
[KNØ]   42: end.
```

Listing 10. Um eine Zeichenkette über Rekursion umzudrehen, speichert das Programm (»rek_um.pas«) den abgeschnittenen Teil der Zeichenkette auf dem Stapel

```
[S75]    1: (* Umdrehen einer Zeichenkette *)
[U34]    2: (* durch Iteration             *)
[AR4]    3: (* Autor: Markus Zietlow       *)
[YC3]    4: (* Listing 11                  *)
[ØØØ]    5:
[T65]    6: program iteratives_umdrehen;
[ØØØ]    7:
[RNØ]    8: var
[2A6]    9:   eingabe_wort, umdrehen, rest:string;
[DX2]   1Ø:   zaehler:integer;
[ØØØ]   11:
[MUØ]   12: begin
[K96]   13:   write('Programm zum Umdrehen eines ');
[1W5]   14:   writeln('Wortes über Iteration');
[KH4]   15:   write('----------------------------');
[464]   16:   writeln('--------------------');
[XF1]   17:   writeln;
[P31]   18:   repeat
[8X4]   19:     write('Geben Sie bitte das zu ');
[TH4]   2Ø:     write('drehende Wort ein: ');
[4N3]   21:     readln(eingabe_wort);
[VØ2]   22:     umdrehen:='';
[GH3]   23:     rest:=eingabe_wort;
[Z56]   24:     for zaehler:=length(rest) downto 1 do
[MUØ]   25:       begin
[525]   26:         umdrehen:=umdrehen+copy(rest,
[YG2]   27:                   length(rest),1);
[HK5]   28:         rest:=copy(rest,1,length(rest)-1);
[ZNØ]   29:       end;
[MF4]   3Ø:     write('Das Umkehrung von "');
[Z75]   31:     write(eingabe_wort,'" lautet "');
[TP3]   32:     writeln(umdrehen,'".');
[2E1]   33:     writeln
[5U3]   34:   until eingabe_wort='';
[KNØ]   35: end.
```

Listing 11. Der iterative Algorithmus in »it_um.pas« verwendet neben einer dekrementierenden Schleife zusätzlich eine Schleifenvariable

```
[NF5]    1: (* Lösung des "Türme von Hanoi"- *)
[SC5]    2: (* Problems durch Rekursion      *)
[GZ4]    3: (* Autor: Markus Zietlow        *)
[6N3]    4: (* Listing 12                   *)
[ØØØ]    5:
[FP4]    6: program tuerme_von_hanoi;
[ØØØ]    7:
[RNØ]    8: var
[TS2]    9:   anzahl:integer;
[ØØØ]   1Ø:
[9F6]   11: procedure turmbewegung(n, stab1, stab2,
[BS2]   12:                        stab3 : integer);
[MUØ]   13: begin
[YG2]   14:   if n=1 then begin
[GW5]   15:     write('Versetzen Sie die oberste ');
[WZ4]   16:     write('Scheibe von Stab ',stab1);
[4C4]   17:     writeln(' nach Stab ',stab3)
[4MØ]   18:   end
[ZRØ]   19:   else
[MUØ]   2Ø:     begin
[NA6]   21:       turmbewegung(n-1, stab1, stab3, stab2
          );
[GW5]   22:       write('Versetzen Sie die oberste ');
[WZ4]   23:       write('Scheibe von Stab ',stab1);
[UF4]   24:       writeln(' nach Stab ',stab3);
[936]   25:       turmbewegung(n-1, stab2, stab1, stab3
          )
[ZNØ]   26:     end;
[ZNØ]   27: end;
[ØØØ]   28:
[MUØ]   29: begin
[F37]   3Ø:   write('Programm zur Lösung des "Türme ');
[L85]   31:   writeln('von Hanoi"-Problems');
[HU4]   32:   write('-----------------------------');
[X24]   33:   writeln('-------------------');
[XF1]   34:   writeln;
[P31]   35:   repeat
[TK5]   36:     write('Geben Sie bitte die Anzahl ');
[BA5]   37:     write('der zu übertragenden ');
[YP3]   38:     write('Scheiben ein: ');
[LG2]   39:     readln(anzahl);
[XF1]   4Ø:     writeln;
[SP4]   41:     turmbewegung(anzahl, 1, 2, 3);
[XF1]   42:     writeln;
[LK2]   43:   until anzahl=1;
[KNØ]   44: end.
```

Listing 12. Das Problem der »Türme von Hanoi« gehört zu den Aufgaben, die sich durch eine Rekursion elegant und unkompliziert lösen lassen (»hanoi.pas«)

```
[S35]    1: (* Sortieren einer Liste von     *)
[SA5]    2: (* Zeichenketten mittels eines   *)
[245]    3: (* gemischten (rekursiven und    *)
[RY4]    4: (* iterativen) Algorithmus       *)
[XU4]    5: (* Autor: Markus Zietlow         *)
[NG3]    6: (* Listing 13                    *)
[ØØØ]    7:
[8B5]    8: program gemischtes_sortieren;
[ØØØ]    9:
[RNØ]   1Ø: var
[684]   11: anzahl, zaehler:integer;
[KX4]   12: liste:array[1..1ØØ] of string;
[ØØØ]   13:
[PZ5]   14: procedure sortieren(linke_grenze,
[GM5]   15:           rechte_grenze, lft, rgt:integer);
[RNØ]   16: var
[573]   17:   mitte, x, y:string;
[MUØ]   18: begin
[XU4]   19:   mitte:=liste[(linke_grenze+
[ØM3]   2Ø:                 rechte_grenze) div 2];
[P31]   21:   repeat
[FH2]   22:    x:=liste[lft];
[YP2]   23:    while x<mitte do
[MUØ]   24:      begin
[LW1]   25:        lft:=lft+1;
[YB2]   26:        x:=liste[lft]
[ZNØ]   27:      end;
[7K2]   28:    y:=liste[rgt];
[2P2]   29:    while mitte<y do
[MUØ]   3Ø:      begin
[WX1]   31:        rgt:=rgt-1;
[PC2]   32:        y:=liste[rgt]
[ZNØ]   33:      end;
[GN2]   34:    if lft<=rgt then
[MUØ]   35:      begin
[2M2]   36:        liste[rgt]:=x;
[BL2]   37:        liste[lft]:=y;
[LW1]   38:        lft:=lft+1;
[GP1]   39:        rgt:=rgt-1
[ZNØ]   4Ø:      end;
[ZE2]   41:   until rgt<lft;
[666]   42:   if linke_grenze<rgt then sortieren(
[Z43]   43:                          linke_grenze, rgt,
[2C3]   44:                          linke_grenze, rgt);
[B86]   45:   if lft<rechte_grenze then sortieren(
[BC3]   46:                           lft, rechte_grenze,
[UD3]   47:                           lft, rechte_grenze);
[ZNØ]   48: end;
[ØØØ]   49:
[MUØ]   5Ø: begin
[PR5]   51:   write('Programm zum rekursiven ');
[9V5]   52:   write('Sortieren einer Liste von ');
[ØH3]   53:   writeln('Elementen');
[HU4]   54:   write('-------------------------------');
[UY4]   55:   writeln('-----------------------------');
[XF1]   56:   writeln;
[K56]   57:   write('Geben Sie bitte die Anzahl der ');
[1G6]   58:   write('zu sortierenden Elemente ein: ');
[LG2]   59:   readln(anzahl);
[XF1]   6Ø:   writeln;
[TF4]   61:   for zaehler:=1 to anzahl do
[MUØ]   62:     begin
[LH4]   63:       write('Geben Sie bitte das ');
[T45]   64:       write(zaehler,'. Element ein: ');
[5V3]   65:       readln(liste[zaehler])
[ZNØ]   66:     end;
[1V4]   67:   sortieren(1,anzahl,1,anzahl);
[XF1]   68:   writeln;
[XF1]   69:   writeln;
[XF1]   7Ø:   writeln;
[GB6]   71:   write('Die sortierte Liste sieht wie ');
[KM3]   72:   writeln('folgt aus:');
[XF1]   73:   writeln;
[TF4]   74:   for zaehler:=1 to anzahl do
[CF5]   75:     writeln('Das ',zaehler,'. Element '+
[D14]   76:             'ist: ',liste[zaehler]);
[XF1]   77:   writeln;
[KNØ]   78: end.
```

Listing 13. Die Sortierroutine (»gem_sort.pas«) zeigt, daß eine sinnvolle Kombination von iterativen und rekursiven Programmschritten zu überraschend schnellen und einfachen Lösungen führen kann

Kontrollstrukturen unter Turbo Pascal

Wie schon in der Befehlsreferenz erwähnt, gehören Kontrollstrukturen zu den wichtigsten Bestandteilen von Turbo Pascal. Sie eignen sich besonders, um bedingungsabhängig bestimmte Prozeduren oder Programmteile auszuführen oder die zu verarbeitenden Daten anwendungsorientiert zu verändern beziehungsweise einzuschränken.

Der Case-Befehl wählt aus einer Reihe von Vergleichskriterien ein Kriterium aus und führt die zugeordnete Anweisung beziehungsweise Verbundanweisung aus. Um die Funktionsweise besser zu verstehen, finden Sie in Listing 1 und Bild 1 eine Anwendung, die den Programmablauf verdeutlicht. Im Beispiel wird eine Eingabe von der Tastatur gelesen und der Anwender darüber informiert, welche Taste er gedrückt hat.
Die Repeat-Anweisung wiederholt einen Anweisungsblock so lange, bis das Abbruchkriterium erfüllt ist. Das bedeutet, daß Turbo Pascal den Block mindestens einmal ausführt. Listing 2 und Bild 2 demonstrieren die korrekte Ausführung des Befehls.
Als nächstes Kontrollkommando stellt sich »while« vor, das in einem Beispiel in Listing 3 und Bild 3 abgedruckt ist. Die exakte Struktur des Befehls ermittelt die Quersumme einer Zahl, bis Sie eine »0« als Parameter übergeben.
Der For-Befehl, dessen Anwendung in Listing 4 abgebildet ist, führt eine einfache oder eine Verbundanweisung in einer Schleife aus, währenddessen er der Laufvariablen aufeinanderfolgende Werte zuweist. Normalerweise, das heißt, wenn der Programmierer das Schlüsselwort »To« benutzt, zählt die Schleife aufwärts. Wenn Sie statt dessen die Variation »DownTo« einsetzen, zählt das Programm wie im abgedruckten Beispiel abwärts vom Anfangswert zum Endwert. Beachten Sie bitte, daß Pascal die Schleife erst dann abbricht, wenn der Inhalt der Schleifenvariable kleiner als der Endwert ist.
Kontrollstrukturen sind von großem Interesse, wenn es darum geht, ein Programm anwenderfreundlich und bedienungsorientiert zu gestalten. So benutzt man im Zusammenhang mit Interrupt-gesteuerten Unterprogrammen Befehle, die beispielsweise überprüfen, ob der Anwender einen Menüpunkt in einer fensterorientierten Benutzeroberfläche mit der Maus oder über die Tastatur angewählt hat.
Denken Sie bitte daran, daß Turbo Pascal für die vorgestellten Kontrollstrukturen Bedingungen vom Datentyp »boolean« erwartet beziehungsweise in der Case-Anweisung nur Integer-Zahlen von -32768 bis 32767 akzeptiert.
Achten Sie darauf, daß Sie die formale Einteilung der Kontrollstrukturen berücksichtigen. Auf diese Weise gewinnen Ihre Programme an Übersichtlichkeit und Lesbarkeit.

(Markus Zietlow/ks)

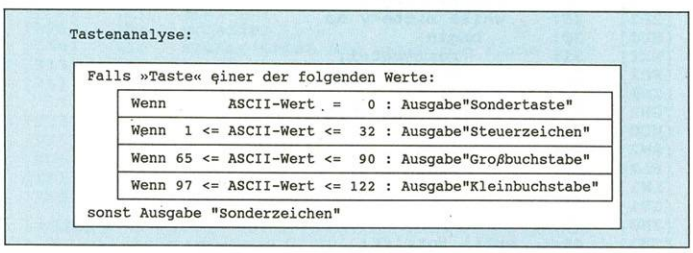


Bild 1. Das Ablaufdiagramm verdeutlicht die Funktionsweise des Case-Befehls

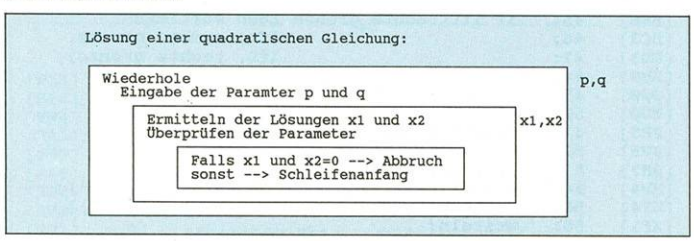


Bild 2. Das Repeat-Kommando führt den eingeschlossenen Block mindestens einmal aus

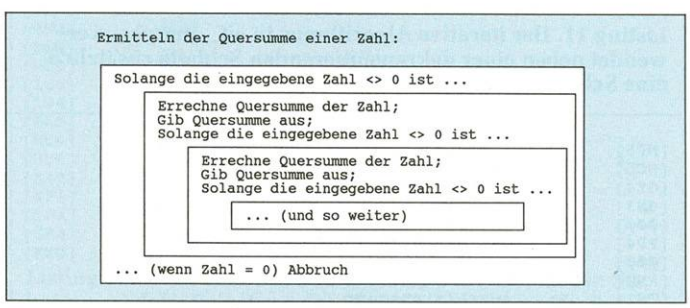


Bild 3. Die While-Anweisung bearbeitet den eingeschlossenen Block bedingungsabhängig

```
[873]    1: program case_demo;
[ØØØ]    2:
[ASØ]    3: uses
[7SØ]    4:   crt;
[ØØØ]    5:
[RNØ]    6: var
[572]    7:   Zeichen:char;
[ØØØ]    8:
[MUØ]    9: begin
[DB1]   1Ø:   ClrScr;
[P31]   11:   repeat
[7C5]   12:     write('Drücken Sie eine Taste !');
[CY2]   13:     Zeichen:=ReadKey;
[165]   14:     write(' Die Taste hat den Typ: ');
[2A3]   15:     case ord(Zeichen) of
[2V1]   16:        Ø       : begin
[L34]   17:                    writeln('Sondertaste');
[CY2]   18:                    Zeichen:=ReadKey;
[ZNØ]   19:                  end;
[GB5]   2Ø:        1 .. 32 : writeln('Steuerzeichen');
[9F5]   21:       65 .. 9Ø : writeln('Großbuchstabe');
[8K5]   22:       97 ..122 : writeln('Kleinbuchstabe');
[ZRØ]   23:     else
[W84]   24:       writeln('Sonderzeichen');
[ZNØ]   25:     end;
[RK3]   26:   until Zeichen=chr(27);
[KNØ]   27: end.
```

Listing 1. Mit dem Case-Kommando ist es nicht schwierig zu überprüfen, welche Taste der Anwender gedrückt hat

```
[ZC5]    1: program QuadratischeGleichung;
[ØØØ]    2:
[RNØ]    3: var
[GM2]    4:   p, q, x1, x2:real;
[ØØØ]    5:
[MUØ]    6: begin
[P31]    7:   repeat
[836]    8:     write('Geben Sie bitte den Parameter '+
[6N1]    9:           '"p" ein: ');
[GH1]   1Ø:     readln(p);
[836]   11:     write('Geben Sie bitte den Parameter '+
[AN1]   12:           '"q" ein: ');
[HH1]   13:     readln(q);
```

```
[124]   14:     x1:=-p/2+sqrt((p/2)*(p/2)-q);
[824]   15:     x2:=-p/2-sqrt((p/2)*(p/2)-q);
[4D5]   16:     writeln('Lösung 1: ',x1:1Ø:5,' / '+
[6F3]   17:             'Lösung 2: ',x2:1Ø:5);
[973]   18:   until (p=Ø) and (q=Ø);
[KNØ]   19: end.
```

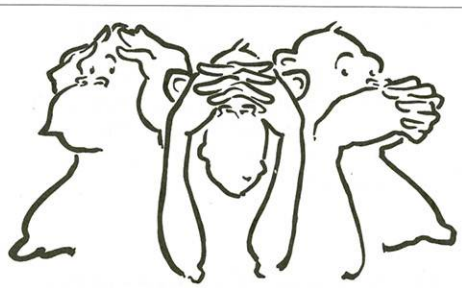
Listing 2. Die Repeat-Anweisung eignet sich dazu, eine Routine so lange auszuführen, bis ein Abbruchkriterium erfüllt ist

```
[X25]    1: (* Iteratives Ermitteln der    *)
[AE5]    2: (* Quersumme zur Demonstration *)
[754]    3: (* der While-Schleife          *)
[AR4]    4: (* Autor: Markus Zietlow       *)
[ØØØ]    5:
[G63]    6: program WhileDemo;
[ØØØ]    7:
[RNØ]    8: var
[GØ4]    9:   letzte_ziffer:integer;
[Y35]   1Ø:   eingabe_zahl, quersumme, rest,
[MF4]   11:   zwischenspeicher:longint;
[ØØØ]   12:
[MUØ]   13: begin
[CN2]   14:   eingabe_zahl:=1;
[CZ5]   15:   writeln('Programm zum iterativen '+
[8C5]   16:           'Ermitteln der Quersumme einer '+
[YA4]   17:           'Zahl (über »While«)');
[Y15]   18:   writeln('------------------------------'+
[MK3]   19:           '------------------------------'+
[PU1]   2Ø:           '------------');
[XF1]   21:   writeln;
[5Ø4]   22:   while eingabe_zahl<>Ø do
[MUØ]   23:     begin
[265]   24:       write('Geben Sie bitte die Zahl '+
[Y41]   25:             'ein: ');
[DK3]   26:       readln(eingabe_zahl);
[Z72]   27:       quersumme:=Ø;
[PM5]   28:       zwischenspeicher:=eingabe_zahl;
[P31]   29:       repeat
[G94]   3Ø:         rest:=eingabe_zahl div 1Ø;
[B76]   31:         letzte_ziffer:=eingabe_zahl mod 1Ø;
[6F6]   32:         quersumme:=quersumme+letzte_ziffer;
[W63]   33:         eingabe_zahl:=rest
[E72]   34:       until rest=Ø;
[ØM5]   35:       eingabe_zahl:=zwischenspeicher;
[UL5]   36:       writeln('Die Quersumme der Zahl ',
[MC3]   37:               eingabe_zahl,' ist: ',
[YØ2]   38:               quersumme);
[XF1]   39:     writeln;
[ZNØ]   4Ø:   end;
[KNØ]   41: end.
```

Listing 3. Den While-Befehl können Sie einsetzen, wenn ein Abbruchkriterium für eine Befehlsfolge schon vor dem ersten Schleifendurchlauf geprüft werden soll

```
[S75]    1: (* Umdrehen einer Zeichenkette *)
[MD4]    2: (* durch Iteration zur         *)
[LA4]    3: (* Demonstration der           *)
[PR3]    4: (* For-Schleife               *)
[AR4]    5: (* Autor: Markus Zietlow      *)
[ØØØ]    6:
[T65]    7: program iteratives_umdrehen;
[ØØØ]    8:
[RNØ]    9: var
[2A6]   1Ø:   eingabe_wort, umdrehen, rest:string;
[DX2]   11:   zaehler:integer;
[ØØØ]   12:
[MUØ]   13: begin
[XH6]   14:   writeln('Programm zum Umdrehen eines '+
[RC4]   15:           'Wortes über Iteration');
[Y15]   16:   writeln('------------------------------'+
[LL2]   17:           '------------------');
[XF1]   18:   writeln;
[776]   19:   write('Geben Sie bitte das zu drehende '+
[U12]   2Ø:         'Wort ein: ');
[4N3]   21:   readln(eingabe_wort);
[VØ2]   22:   umdrehen:='';
[GH3]   23:   rest:=eingabe_wort;
[Z56]   24:   for zaehler:=length(rest) downto 1 do
[MUØ]   25:     begin
[525]   26:       umdrehen:=umdrehen+copy(rest,
[YG2]   27:                  length(rest),1);
[HK5]   28:       rest:=copy(rest,1,length(rest)-1);
[ZNØ]   29:     end;
[VU4]   3Ø:   writeln('Das Umkehrung von "',
[824]   31:           eingabe_wort,'" lautet "',
[F72]   32:           umdrehen,'".');
[XF1]   33:   writeln;
[KNØ]   34: end.
```

Listing 4. Die For-Schleife akzeptiert mit dem Schlüsselwort »DownTo« auch Abwärtszählungen



WALDSTERBEN, WIESO? IST DOCH ALLES GRÜN.

Irrtum. Denn der Wald ist zu über 50 Prozent geschädigt.

Auf den ersten Blick sehen unsere Wälder gesund aus. Aber jeder zweite Baum ist krank. Dies konnte auch die bisherige Umweltpolitik nicht verhindern. Halbherzige Kompromisse helfen höchstens das schlechte Gewissen zu beruhigen.

Wenn Sie wissen wollen, was zur Rettung unserer letzten Waldgebiete getan werden muß, dann füllen Sie den Coupon aus. Wir beantworten Ihre Frage, sagen Ihnen, wo wir uns noch engagieren, warum Sie uns dabei helfen können.

Ja, ich will von ROBIN WOOD wissen, was zur Rettung unserer letzten Waldgebiete getan werden muß.

Name, Vorname

Straße

Wohnort

dmv

Weil ich helfen will, lege ich 3 Mark in Briefmarken bei.
Coupon bitte einsenden an
ROBIN WOOD, Postfach 10 21 22, 2800 Bremen 1

ROBIN WOOD

Gewaltfreie Aktionsgemeinschaft für Natur und Umwelt e.V.

Modularisierung mit Units

Vor allem beim Schreiben großer, komplexer Programme ist es hinderlich und unübersichtlich, das gesamte Programm in einem einzigen großen Quelltext zu speichern. Turbo Pascal bietet daher die Möglichkeit, bestimmte Teile eines Programms in eine Unit »auszulagern«.

Eine Unit ist eine Ansammlung von Prozeduren, Funktionen und Konstanten, die sich in anderen Programmen verwenden lassen. Turbo Pascal stellt bereits einige Standard-Units zur Verfügung. Um beispielsweise Befehle zum Zugriff auf den Bildschirm anwenden zu können, wird die Unit »crt« in einem eigenen Programm aufgerufen, für Grafikkommandos steht die Unit »graph« zur Verfügung. Programme, die Befehle einer Unit nutzen möchten, binden die Unit mit dem Kommando »uses« in den Quelltext ein. Für den Aufruf der Unit »graph« lautet das Kommando beispielsweise

```
uses graph;
```

Nach dem Einbinden stehen die Befehle der Unit zur Verfügung, als gehörten sie zum Standardumfang von Turbo Pascal. Die Umwandlung von eigenen Funktionsbibliotheken in selbstdefinierte Units erlaubt die Erstellung leistungsfähiger Programme. Zu bestimmten Themenstellungen können Sie Prozeduren und Funktionen in einer Unit zusammenfassen. In anderen Programmen werden diese Kommandos dann direkt genutzt, indem Sie die Unit in den Quelltext einbinden. Ein ähnliches Konzept benutzen auch andere Programmiersprachen wie »C« oder die neuen Basic-Compiler, bei denen sogenannte »Libraries« (Bibliotheken) in die Programme eingebunden werden.
Als Beispiel für die Definition und Nutzung einer Unit soll hier der 320x200-Punkte-Modus in 256 Farben dienen, den die VGA- und MCGA-Grafikkarten bieten. Dieser Modus wird von Turbo Pascal in der Unit »graph« nicht unterstützt. Das Programm »vga256.pas« in Listing 1 stellt Funktionen und Prozeduren zur Verfügung, mit denen sich der 320x200-Punkte-Modus in eigenen Programmen einsetzen läßt. Die technischen Hintergründe der Routinen sollen an dieser Stelle nicht weiter interessieren. Das Programm »vga256.pas« enthält insgesamt fünf Prozeduren und Funktionen, mit denen der 320x200-Punkte-Modus verwendet werden kann. Die hier abgedruckte Tabelle faßt die Routinen zusammen.
Der Videomodus mit 320x200 Punkten in 256 Farben wird über die Prozedur »mode320x200« gesetzt. Für diesen Modus stehen die Routinen »set320x200«, »read320x200« und »line320x200« zur Verfügung. Mit »set320x200« setzen Sie einen Bildpunkt an einer bestimmten Bildschirmkoordinate und versehen ihn mit einer Farbe, »read320x200« liest einen Punkt und liefert die Farbe zurück. Die Prozedur »line320x200« zeichnet eine Gerade im neuen Videomodus, und zwar nach dem Geradengleichungssystem.
Die gezeichnete Linie kann in einer der 256 möglichen Farben erscheinen. Zur Rückkehr in den Textmodus verwenden Sie die Prozedur »modetext«. Sie setzt den Bildschirmmodus, der vor Aufruf der Prozedur »mode320x200« aktiv war.

Als Beispiel für eine selbstgeschriebene Unit dient der 320x200-Punkte-Modus in 256 Farben

Im Hauptprogramm des Listings »vga256.pas« werden mit Hilfe der neuen Funktionen Linien auf dem Bildschirm in 256 Farben gezeichnet. Sollen die neuen VGA-Routinen auch in anderen Programm verwendet werden, ist es sehr lästig, den Quelltext jedesmal in das Programm einzufügen. Außerdem besteht die Gefahr, daß bestimmte Schlüsselwörter und ähnliches mehrfach deklariert werden. Daher läßt sich an den Routinen zum Zugriff auf den 320x200-Punkte-Modus der VGA-Karte sehr gut die Leistungsfähigkeit des Unit-Konzepts darstellen. Die Routinen zum Zugriff auf die VGA-Karte werden dabei in einer eigenen Unit zusammengefaßt. Diese Unit soll den Namen »co256« tragen. Für die Nutzung des 320x200-Punkte-Modus in 256 Farben in anderen Programmen reicht es aus, die neue Unit mit der Anweisung

```
uses co256;
```

in das Programm einzubinden, und schon stehen alle Funktionen und Prozeduren der Unit zur Verfügung.
Wie werden die Routinen aber in einer eigenen Unit zusammengefaßt? Eine Unit besteht in Turbo Pascal aus drei Bestandteilen: dem Interface-Teil, dem Implementationsteil und dem Initialisierungsteil. Im Interface-Teil definieren Sie alle Funktionen, Prozeduren, Typen, Variablen und Konstanten, die vom aufrufenden Programm aus »sichtbar« sein sollen. Den Interface-Teil einer Unit leiten Sie durch das Schlüsselwort »interface« ein. Alles, was Sie im Interface-Teil definieren, können Programme, in die die Unit eingebunden wird, aufrufen und verwenden.

Eine Unit besteht aus drei Teilen: Interface, Implementation und Initialisierung

Im Implementationsteil nehmen Sie die »unsichtbare« Realisierung der Prozeduren und Funktionen des Implementationsteils vor. Das Schlüsselwort »implementation« leitet diesen Unit-Abschnitt ein. Hier definieren Sie lokale Variablen, Konstanten, Typen sowie Prozeduren und Funktionen, die sich aus anderen Programmen nicht aufrufen lassen.
Der Initialisierungsteil schließlich wird von Turbo Pascal vor dem Start jedes Programms, das die Units verwendet, aufgerufen. Hier findet die Initialisierung von Variablen oder der Aufruf von Initialsierungsroutinen statt. Der Initialsierungsteil wird durch die beiden reservierten Wörter »begin« und »end« eingeleitet beziehungsweise abgeschlossen.
Im Programmkopf unterscheidet sich eine Unit-Deklaration von einem gewöhnlichen Programm dadurch, daß das Schlüsselwort »program« durch »unit« ersetzt wird. Formal gesehen besitzt eine Unit die folgende Struktur:

```
unit <Name>;
interface
```

Aufgabe	Prozedur/Funktion
Modus einschalten	mode320x200
Modus ausschalten	modetext
Grafikpunkt setzen	set320x200
Grafikpunkt lesen	read320x200
Linie zeichnen	line320x200

Tabelle. Die Routinen der Unit »co256«

```
{Hier folgen die Deklarationen}
const ...;
{Verwendete Konstanten}
type ...;
{Verwendete Typen-Festlegungen}
var ...;
{Verwendete Variablen}
procedure ...;
{Köpfe der verwendeten Prozeduren}
function ...;
{Köpfe der verwendeten Funktionen}
implementation
{Hier folgt die Beschreibung der Funktionen/Prozeduren}
const ...;
{Lokale Konstanten der Unit}
type ...;
{Lokale Typen der Unit}
var ...;
{Lokale Variablen der Unit}
procedure ...;
{Anweisungen der verwendeten Prozeduren}
function ...;
{Anweisungen der verwendeten Funktionen}
procedure ...;
{Lokale Prozeduren}
function ...;
{Lokale Funktionen}
begin
{Start des Initialisierungsteils}
end;
```

Für das konkrete Beispiel der Umwandlung der Prozeduren und Funktionen aus Listing 1 in die Unit »co256« sieht der Interface-Teil wie folgt aus:

```
interface
procedure mode320x200;
procedure modetext;
procedure set320x200(x,y:integer;
farbe:byte);
procedure line320x200(x1,y1,x2,y2:
integer; farbe:byte);
function read320x200(x,y:integer):byte;
```

Alle Prozeduren und Funktionen, die in anderen Programmen zugänglich sein sollen, werden also mit ihrem Deklarationskopf aufgeführt. Die eigentliche Programmierung der Funktionen und Prozeduren erfolgt im Implementationsteil. Im Implementationsteil führen Sie die Funktions- und Prozedurköpfe erneut auf, es folgt auch die Definition von lokalen Variablen und Konstanten der Funktionen beziehungsweise Prozeduren sowie der eigentliche Anweisungsteil. Der Initialisierungsteil der neuen Unit besteht nur aus drei Zeilen:

```
begin
benutzt:=false;
end.
```

Hier wird die Boolean-Variable »benutzt« auf den Wert »false« gesetzt. Mit dieser Variablen hält die Unit intern fest, ob der Videomodus bereits gesetzt ist. Die komplette Unit »co256« ist in Listing 2 aufgeführt.
Eine Unit wird wie ein Pascal-Programm compiliert. Units müssen immer extern compiliert werden. Turbo Pascal erkennt automatisch, daß es sich um eine Unit handelt, und speichert den Programmcode auf dem Datenträger. Um Units von anderen Programmen zu unterscheiden, vergibt Turbo Pascal als Namenserweiterung die Buchstaben »tpu« (für »Turbo Pascal Unit«).

Selbstdefinierte Units binden Sie über das Uses-Kommando in Ihre eigenen Programme ein

Ist die Unit einmal compiliert, können andere Programme die definierten Funktionen und Prozeduren verwenden. Das Programm »demo256.pas« in Listing 3 demonstriert den Aufruf der Co256-Routinen an einem Beispiel. Die Co256-Unit wird im Listing mit dem Kommando

```
uses co256;
```

in das Programm eingebunden. Beim Compilieren des Programms sucht Turbo Pascal im aktuellen Verzeichnis nach der Unit »co256.tpu« und bindet sie in das aktuelle Programm ein. Alle im Interface-Teil deklarierten Funktionen und Prozeduren stehen dem Programm dann zur Verfügung.
Was hier am Beispiel des 320x200-Punkte-Modus erklärt wurde, läßt sich auf andere Problemfälle ebenfalls übertragen. Wenn Sie eigene Units schreiben, sollten Sie sich an einige Grundregeln halten. Es ist sinnvoll, Bestandteile einer Unit in einem kurzen, unkomplizierten Programm zu testen und die Routinen erst danach in eine eigenständige Unit zu überführen. Dadurch wird die Fehlersuche erheblich vereinfacht. Außerdem sollten Sie zuerst mehrere kleine Routinen einzeln auf ordnungsgemäße Funktion testen, bevor Sie diese zu einem größeren Programm beziehungsweise einer TPU zusammenbinden.

Über das Programm »tpumover« binden Sie Units in die Standardbibliotheken von Turbo Pascal ein

Seit Erscheinen der Turbo-Pascal-Version 4.0 sind Turbo-Pascal-Programme nicht mehr auf einen Codeumfang von 64 KByte begrenzt. Diese Öffnung der Speichergrenzen hängt mit dem Unit-Konzept zusammen. Im Prinzip ist ein Turbo-Pascal-Programm nämlich weiterhin auf 64 KByte Code begrenzt. Da für jede Unit jedoch ein eigenes Segment im Speicher angelegt wird, lassen sich mit jeder Unit weitere 64 KByte Speicher adressieren. Wer also sehr große Programme schreibt, kommt nicht umhin, sie in unterschiedliche Units aufzusplitten. Bei umfangreichen Programmen empfiehlt es sich, für alle Bestandteile des Programms eigene Units aufzubauen. Diese Units werden dann im Hauptprogramm zusammengeführt, das nicht mehr als 64 KByte Code enthalten darf. Es erscheint beispielsweise bei großen Programmen ratsam, alle Konstanten, alle Variablen und alle Typdeklarationen in jeweils einer eigenen Unit zusammenzufassen, falls der Speicherbedarf des Hauptprogramms steigt.
Die Standard-Units, die Turbo Pascal zur Verfügung stellt, sind »system«, »crt«, »printer«, »graph«, »overlay« (ab Version 5.0) und »dos«. Die System-Unit wird in jedes Turbo-Pascal-Programm eingebunden, ohne daß Sie sie mit dem Befehl »Uses« deklarieren müssen. Sie enthält die Standardlaufzeitbibliothek. Die anderen Units müssen Sie explizit aufrufen, damit ihre Kommandos in eigenen Programmen verfügbar sind.
Alle Standard-Units bis auf »graph« sind in der Datei »turbo.tpl« zusammengefaßt. Diese Units besitzen also keinen Programmcode in eigenen TPU-Dateien. Mit einem speziellen Turbo-Pascal-Hilfsprogramm lassen sich auch selbstdefinierte Units in die Datei »turbo.tpl« aufnehmen. Dazu verwenden Sie das Programm »tpumover«, das Bestandteil des Turbo-Pascal-Systems ist. Hinter diesem Dateinamen verbirgt sich der »Turbo Pascal Unit Librarian Manager«. Um die in diesem Artikel definierte Unit »co256« in die Standard-Unit »turbo.tpl« einzubinden, rufen Sie das Programm »tpumover« wie folgt auf:

```
tpumover turbo co256
```

Der Unit-Manager bindet daraufhin den Programmcode der Unit »co256« in den Code der Datei »turbo.tpl« ein. Der Vorteil dieses Verfahrens liegt darin, daß der Code der Unit »co256« nun fest eingebunden ist und somit jederzeit zur Verfügung steht.
Das Unit-Konzept eignet sich vor allem für zwei Zielsetzungen: zur Modularisierung von unterschiedlichen Gruppen neuer Kommandos in Bibliotheken und zur Realisierung großer Programme mit mehr als 64 KByte Code. Für Anwendungen, bei denen so große Programme entstehen, daß sie nicht mehr in den Arbeitsspeicher des verwendeten Rechners passen, stellt Turbo Pascal ein besonderes Feature zur Verfügung: die Overlay-Units. Die Programmierung dieser Module wird in einem eigenen Artikel in diesem Sonderheft erläutert.

(Martin Althaus/dm)

```
  Name..... VGA256.PAS

  Sprache..... Turbo Pascal

  Grafikkarte..... VGA und MCGA
```

```
[3M2]    1: program vga256;
[UØ4]    2: { ---------------------------------
[6K1]    3:              vga256.pas
[531]    4:              ----------
[CS5]    5:   Programm für den Zugriff auf den
[CR4]    6:   32Øx2ØØ-Modus in 256 Farben (Mo-
[HU3]    7:   dus 13hex) der VGA-Karte.
[VT3]    8:   von Martin Althaus 3/9Ø
[4Ø4]    9:   (C) 199Ø Redaktionsbüro
[L83]   1Ø:       Everts und Hagedorn
[AL3]   11:   ---------------------------------
[U7Ø]   12: }
[S62]   13: uses crt,dos;
[KH4]   14:              {für die Register-Def.}
[RNØ]   15: var
[YT2]   16:  cpu: registers;
[E43]   17:  index,modus: byte;
[ØZ2]   18:  benutzt: boolean;
[8H5]   19:  {Zeigt an, ob MODE32ØX2ØØ benutzt}
[ØØØ]   2Ø:
[4K3]   21: procedure mode32Øx2ØØ;
[ZL3]   22: {------------------------------
[RØ5]   23:  Setze die VFA/MCGA-Karte Modus
[894]   24:  13hex mit 32Øx2ØØ Punkten in
[ØS1]   25:  256 Farben.
[F93]   26:  ------------------------------
[U7Ø]   27: }
[MUØ]   28: begin
[2X1]   29:  cpu.ah:=$Øf;
[1V3]   3Ø:              {Lese Videomodus aus}
[KL5]   31:  intr($1Ø,cpu);    {über Int 1Øhex}
[LW5]   32:  modus:=cpu.al;   {Merke den Modus}
[EL2]   33:  benutzt:=true;
[EL4]   34:          {zeige gemerkten Modus an}
[8M1]   35:  cpu.ah:=Ø;
[334]   36:           {Setze Modus 13hex über}
[5V1]   37:  cpu.al:=$13;
[6M5]   38:     {Funktion ØØhex von Int. 1Øhex}
[742]   39:  intr($1Ø,cpu);
[ZNØ]   4Ø: end;
[ØØØ]   41:
[2G3]   42: procedure modetext;
[ZL3]   43: {------------------------------
[AD5]   44:  Kehre zurück in den Textmodus,
[BY4]   45:  der vor dem Setzen mit MODE32Ø
[MA2]   46:  X2ØØ aktiv war.
[F93]   47:  ------------------------------
[U7Ø]   48: }
[MUØ]   49: begin
[XK2]   5Ø:  if benutzt then
[574]   51:           {Ist ein Modus gemerkt?}
[YG4]   52:  begin             {Ja, setze ihn}
[8M1]   53:   cpu.ah:=Ø;
[KA4]   54:           {Benutze Funktion ØØhex}
[DD5]   55:   cpu.al:=modus;   {von Int. 1Øhex}
[742]   56:   intr($1Ø,cpu);
[ZNØ]   57:  end;
[ZNØ]   58: end;
[ØØØ]   59:
[KB5]   6Ø: procedure set32Øx2ØØ(x,y:integer;
[8Y1]   61:                      farbe:byte);
[6S3]   62: {------------------------------
[5H4]   63:  Setze einen Punkt im 32Øx2ØØ-
[UA5]   64:  Modus an der Stelle X,Y und gebe
[F73]   65:  ihm die Farbe FARBE
[GG3]   66:  ------------------------------
[U7Ø]   67: }
[RNØ]   68: var
[682]   69:  offset: word;
[DZ5]   7Ø: begin       {X-Koordinate gültig?}
[HA2]   71:  if x<Ø then x:=Ø
[UX3]   72:         else if x>319 then x:=319;
[W64]   73:              {Y-Koordinaten gültig}
[MA2]   74:  if y<Ø then y:=Ø
[XX3]   75:         else if y>199 then y:=199;
[UH2]   76:  offset:=32Ø*y+x;
[7B4]   77:              {Berechne Punkt-Adresse}
[D24]   78:  mem[$aØØØ:offset]:=farbe;
[L33]   79:                    {Und setze Punkt}
[ZNØ]   8Ø: end;
[ØØØ]   81:
[EF5]   82: function read32Øx2ØØ(x,y:integer):
[UWØ]   83:                       byte;
[6S3]   84: {------------------------------
[DV4]   85:  Lese einen Punkt aus dem Bild-
[PY4]   86:  schirmspeicher aus und liefere
[XD5]   87:  seine Farbe als Funktionsergeb-
[R42]   88:  nis zurück.
[GG3]   89:  ------------------------------
[U7Ø]   9Ø: }
[RNØ]   91: var
[682]   92:  offset: word;
[7Y5]   93: begin       {justiere X-Koordinate}
[HA2]   94:  if x<Ø then x:=Ø
[UX3]   95:         else if x>319 then x:=319;
[RH4]   96:              {justiere Y-Koordinate}
[MA2]   97:  if y<Ø then y:=Ø
[XX3]   98:         else if y>199 then y:=199;
[UH2]   99:  offset:=32Ø*y+x;
[7B4]  1ØØ:              {Berechne Punkt-Adresse}
[PW4]  1Ø1:  read32Øx2ØØ:=mem[$aØØØ:offset];
[RZ2]  1Ø2:                    {Lese Punkt aus}
[ZNØ]  1Ø3: end;
[ØØØ]  1Ø4:
[6E5]  1Ø5: procedure tausche(x,y:integer);
[6S3]  1Ø6: {------------------------------
[F75]  1Ø7:  Interne Prozedur zum Tauschen
[TØ3]  1Ø8:  von zwei Integers
[GG3]  1Ø9:  ------------------------------
[U7Ø]  11Ø: }
[RNØ]  111: var
[TG5]  112:  help:integer;      {Hilfsvariable}
[MUØ]  113: begin
[ML4]  114:  help:=x;                {X merken}
[S64]  115:  x:=y;                   {Y wird X}
[VH4]  116:  y:=help;                {X wird Y}
[ZNØ]  117: end;
[ØØØ]  118:
[UA5]  119: procedure line32Øx2ØØ(x1,y1,x2,y2:
[RH3]  12Ø:              integer; farbe:byte);
[6S3]  121: {------------------------------
[HX4]  122:  Zeichnet eine Linie im 32Øx2ØØ-
[ZD5]  123:  Modus vom Anfangspunkt X1,Y1 zum
[145]  124:  Endpunkt X2,Y2 und gibt ihr die
[UW1]  125:  Farbe FARBE
[GG3]  126:  ------------------------------
[U7Ø]  127: }
[RNØ]  128: var
[8W1]  129:  y: integer;
[BE5]  13Ø:          {für das Funktionsergebnis}
[MH1]  131:  m,b: real;
[ZG5]  132:      {Steigung und Achsenabschnitt}
[ØØØ]  133:
[MUØ]  134: begin
[212]  135:  if x2<x1 then
[Z85]  136:        {Müssen die Koordianten ge-}
[NV4]  137:   begin           {tauscht werden?}
[2E2]  138:    tausche(x1,x2);
[GW3]  139:              {ja, also tausche sie}
[ZZ4]  14Ø:    tausche(y1,y2);      {beide aus}
[ZNØ]  141:   end;
[K32]  142:  if x2<>x1 then
[EY4]  143:         {kann man Geradengleichung}
[CD4]  144:   begin                {anwenden?}
[NM2]  145:    m:=(y2-y1)/(x2-x1);
[CP2]  146:                    {ja, berechne M}
[M24]  147:    b:=y1-(m*x1);              {und B}
[S13]  148:    for x1:=x1 to x2 do
[MH3]  149:                 {Berechne in einer}
[A15]  15Ø:     begin       {Schleife zu jedem}
[7M2]  151:       y:=trunc(m*x1+b);
[D62]  152:                          {X ein Y und}
[6N3]  153:       set32Øx2ØØ(x1,y,farbe);
[2S3]  154:                    {zeichne den Punkt}
[M45]  155:     end;       {Schleife schließen}
[4MØ]  156:   end
[AF5]  157:  else      {Es ist eine Senkrechte}
[MUØ]  158:   begin
[GL4]  159:    if y2<y1 then tausche(y1,y2);
[Ø23]  16Ø:    for y1:=y1 to y2 do
[6B4]  161:            {setze in einer Schleife}
[LM3]  162:       set32Øx2ØØ(x1,y1,farbe);
[6Y2]  163:                    {die Senkrechte}
[ZNØ]  164:   end;
[ZNØ]  165: end;
[ØØØ]  166:
[MUØ]  167: begin
[3P1]  168: {-------------
[NC2]  169:  Hauptprogramm
[RS1]  17Ø:  -------------}
[XS2]  171: benutzt:=false;
[2Z4]  172:         {MODE32ØX2ØØ ist unbenutzt}
[ZP1]  173: mode32Øx2ØØ;
[HZ3]  174:           {Setze den Modus 32Øx2ØØ}
[SH3]  175: for index:=Ø to 255 do
[NN5]  176:     {Zeichne 255 senkrechte Linien}
[UY3]  177:  line32Øx2ØØ(index,1,index,
[5S1]  178:               199,index);
[C75]  179:       {über die neue Line-Funktion}
[KK3]  18Ø: for index:=Ø to 199 do
[ZH5]  181:    {Zeichne 199 waagerechte Linien}
[MT4]  182:  line32Øx2ØØ(256,index,319,index,
[871]  183:               index);
[C75]  184:       {über die neue Line-Funktion}
[Y44]  185: repeat until keypressed;
```

```
[215]  186:       {Warte auf einen Tastendruck}
[VP6]  187: modetext; {Setze Textmodus zurück}
[ØØØ]  188:
[KNØ]  189: end.
```

Listing 1. Das Programm »vga256.pas« verwendet den 320x200-Punkte-Modus der VGA-Karte

```
  Name.............. CO256.PAS

  Sprache......... Turbo Pascal

  Grafikkarte..... VGA und MCGA

  ═══════════════════════════════
```

```
[VV1]    1: unit co256;
[X83]    2: { ---------------------------
[DC1]    3:          co256.pas
[YYØ]    4:          ---------
[P25]    5:   Pascal-Unit für den Zugriff
[1P3]    6:   auf den 32Øx2ØØ-Modus in
[1P3]    7:   256 Farben (Modus 13hex)
[VT3]    8:   von Martin Althaus 3/9Ø
[4Ø4]    9:   (C) 199Ø Redaktionsbüro
[L83]   1Ø:       Everts und Hagedorn
[113]   11:   ---------------------------
[U7Ø]   12: }
[SK1]   13: interface
[4K3]   14:  procedure mode32Øx2ØØ;
[M34]   15:            {Setzt den 32Øx2ØØ-Modus}
[2G3]   16:  procedure modetext;
[NN4]   17:            {Zurück in den Textmodus}
[KB5]   18:  procedure set32Øx2ØØ(x,y:integer;
[8Y1]   19:                       farbe:byte);
[2S3]   2Ø:                  {Setzt einen Punkt}
[UA5]   21:  procedure line32Øx2ØØ(x1,y1,x2,y2:
[RH3]   22:            integer; farbe:byte);
[RN3]   23:               {Zeichnet eine Linie}
[EF5]   24:  function read32Øx2ØØ(x,y:integer):
[UWØ]   25:                       byte;
[ØR3]   26:                  {Liest einen Punkt}
[4L2]   27: implementation
[VG1]   28:  uses dos;
[KH4]   29:               {für die Register-Def.}
[RNØ]   3Ø: var
[YT2]   31:  cpu: registers;
[ZZ1]   32:  modus: byte;
[ØZ2]   33:  benutzt: boolean;
[8H5]   34:  {Zeigt an, ob MODE32ØX2ØØ benutzt}
[ØØØ]   35:
[4K3]   36: procedure mode32Øx2ØØ;
[ZL3]   37: {-----------------------------
[RØ5]   38:  Setze die VFA/MCGA-Karte Modus
[894]   39:  13hex mit 32Øx2ØØ Punkten in
[ØS1]   4Ø:  256 Farben.
[F93]   41:  -----------------------------
[U7Ø]   42: }
[MUØ]   43: begin
[2X1]   44:  cpu.ah:=$Øf;
[1V3]   45:                  {Lese Videomodus aus}
[742]   46:  intr($1Ø,cpu);
[ZX2]   47:                       {über Int 1Øhex}
[SA2]   48:  modus:=cpu.al;
[SX2]   49:                     {Merke den Modus}
[EL2]   5Ø:  benutzt:=true;
[EL4]   51:            {zeige gemerkten Modus an}
[8M1]   52:  cpu.ah:=Ø;
[334]   53:             {Setze Modus 13hex über}
[5V1]   54:  cpu.al:=$13;
[6M5]   55:     {Funktion ØØhex von Int. 1Øhex}
[742]   56:  intr($1Ø,cpu);
[ZNØ]   57: end;
[ØØØ]   58:
[2G3]   59: procedure modetext;
[ZL3]   6Ø: {-----------------------------
[AD5]   61:  Kehre zurück in den Textmodus,
[BY4]   62:  der vor dem Setzen mit MODE32Ø
[MA2]   63:  X2ØØ aktiv war.
[F93]   64:  -----------------------------
[U7Ø]   65: }
[MUØ]   66: begin
[XK2]   67:  if benutzt then
[574]   68:             {Ist ein Modus gemerkt?}
[MUØ]   69:  begin
[ZU2]   7Ø:                       {Ja, setze ihn}
[8M1]   71:   cpu.ah:=Ø;
[KA4]   72:             {Benutze Funktion ØØhex}
```

```
[6E2]   73:   cpu.al:=modus;
[NS2]   74:                       {von Int. 1Øhex}
[742]   75:   intr($1Ø,cpu);
[ZNØ]   76:  end;
[ZNØ]   77: end;
[ØØØ]   78:
[KB5]   79: procedure set32Øx2ØØ(x,y:integer;
[8Y1]   8Ø:                       farbe:byte);
[6S3]   81: {-----------------------------
[5H4]   82:  Setze einen Punkt im 32Øx2ØØ-
[UA5]   83:  Modus an der Stelle X,Y und gebe
[F73]   84:  ihm die Farbe FARBE
[GG3]   85:  -----------------------------
[U7Ø]   86: }
[RNØ]   87: var
[682]   88:  offset: word;
[MUØ]   89: begin
[W34]   9Ø:                {X-Koordinate gültig?}
[HA2]   91:  if x<Ø then x:=Ø
[UX3]   92:         else if x>319 then x:=319;
[W64]   93:                {Y-Koordinaten gültig}
[MA2]   94:  if y<Ø then y:=Ø
[XX3]   95:         else if y>199 then y:=199;
[UH2]   96:  offset:=32Ø*y+x;
[7B4]   97:              {Berechne Punkt-Adresse}
[D24]   98:  mem[$aØØØ:offset]:=farbe;
[L33]   99:                     {Und setze Punkt}
[ZNØ]  1ØØ: end;
[ØØØ]  1Ø1:
[EF5]  1Ø2: function read32Øx2ØØ(x,y:integer):
[UWØ]  1Ø3:                        byte;
[6S3]  1Ø4: {-----------------------------
[DV4]  1Ø5:  Lese einen Punkt aus dem Bild-
[PY4]  1Ø6:  schirmspeicher aus und liefere
[XD5]  1Ø7:  seine Farbe als Funktionsergeb-
[R42]  1Ø8:  nis zurück.
[GG3]  1Ø9:  -----------------------------
[U7Ø]  11Ø: }
[RNØ]  111: var
[682]  112:  offset: word;
[MUØ]  113: begin
[LH4]  114:               {justiere X-Koordinate}
[HA2]  115:  if x<Ø then x:=Ø
[UX3]  116:         else if x>319 then x:=319;
[RH4]  117:               {justiere Y-Koordinate}
[MA2]  118:  if y<Ø then y:=Ø
[XX3]  119:         else if y>199 then y:=199;
[UH2]  12Ø:  offset:=32Ø*y+x;
[7B4]  121:              {Berechne Punkt-Adresse}
[PW4]  122:  read32Øx2ØØ:=mem[$aØØØ:offset];
[RZ2]  123:                     {Lese Punkt aus}
[ZNØ]  124: end;
[ØØØ]  125:
[6E5]  126: procedure tausche(x,y:integer);
[6S3]  127: {-----------------------------
[F75]  128:  Interne Prozedur zum Tauschen
[TØ3]  129:  von zwei Integers
[GG3]  13Ø:  -----------------------------
[U7Ø]  131: }
[RNØ]  132: var
[N72]  133:  help:integer;
[W33]  134:                        {Hilfsvariable}
[MUØ]  135: begin
[5D1]  136:  help:=x;
[FY1]  137:                              {X merken}
[1TØ]  138:  x:=y;
[RX1]  139:                              {Y wird X}
[5C1]  14Ø:  y:=help;
[PX1]  141:                              {X wird Y}
[ZNØ]  142: end;
[ØØØ]  143:
[UA5]  144: procedure line32Øx2ØØ(x1,y1,x2,y2:
[RH3]  145:            integer; farbe:byte);
[6S3]  146: {-----------------------------
[HX4]  147:  Zeichnet eine Linie im 32Øx2ØØ-
[ZD5]  148:  Modus vom Anfangspunkt X1,Y1 zum
[145]  149:  Endpunkt X2,Y2 und gibt ihr die
[UW1]  15Ø:  Farbe FARBE
[GG3]  151:  -----------------------------
[U7Ø]  152: }
[RNØ]  153: var
[8W1]  154:  y: integer;
[BE5]  155:          {für das Funktionsergebnis}
[MH1]  156:  m,b: real;
[ZG5]  157:      {Steigung und Achsenabschnitt}
[ØØØ]  158:
[MUØ]  159: begin
[212]  16Ø:  if x2<x1 then
[Z85]  161:         {Müssen die Koordianten ge-}
[NV4]  162:   begin            {tauscht werden?}
[2E2]  163:    tausche(x1,x2);
[GW3]  164:                  {ja, also tausche sie}
[ZZ4]  165:    tausche(y1,y2);       {beide aus}
[ZNØ]  166:   end;
[K32]  167:  if x2<>x1 then
[EY4]  168:          {kann man Geradengleichung}
[CD4]  169:   begin                  {anwenden?}
```

Listing 2. Die Unit »co256« faßt die neuen Routinen in einer Unit zusammen

```
[NM2]  17Ø:     m:=(y2-y1)/(x2-x1);
[CP2]  171:                       {ja, berechne M}
[M24]  172:     b:=y1-(m*x1);              {und B}
[S13]  173:     for x1:=x1 to x2 do
[MH3]  174:                     {Berechne in einer}
[A15]  175:      begin          {Schleife zu jedem}
[7M2]  176:        y:=trunc(m*x1+b);
[D62]  177:                           {X ein Y und}
[6N3]  178:        set32Øx2ØØ(x1,y,farbe);
[2S3]  179:                     {zeichne den Punkt}
[M45]  18Ø:      end;         {Schleife schließen}
[4MØ]  181:    end
[AF5]  182:   else       {Es ist eine Senkrechte}
[MUØ]  183:    begin
[GL4]  184:     if y2<y1 then tausche(y1,y2);
[Ø23]  185:     for y1:=y1 to y2 do      '
[6B4]  186:           {setze in einer Schleife}
[LM3]  187:        set32Øx2ØØ(x1,y1,farbe);
[6Y2]  188:                           {die Senkrechte}
[ZNØ]  189:    end;
[ZNØ]  19Ø: end;
[ØØØ]  191:
[MUØ]  192: begin
[6S3]  193: {--------------------------------
[TU2]  194:  Initialisierung
[GG3]  195:  --------------------------------
[U7Ø]  196: }
[XS2]  197:  benutzt:=false;
[2Z4]  198:          {MODE32ØX2ØØ ist unbenutzt}
[KNØ]  199: end.
```

Listing 2. Die Unit »co256« faßt die neuen Routinen in einer Unit zusammen

```
Name..... DEMO256.PAS

Sprache..... Turbo Pascal

Grafikkarte..... VGA und MCGA
```

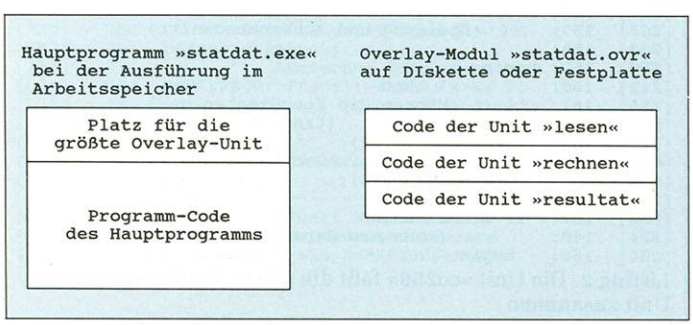
```
[4P2]    1: program demo256;
[124]    2: {---------------------------------
[KW1]    3:              demo256.pas
[T81]    4:              -----------
[2L5]    5:  Demonstriert den Umgang mit der
[RC5]    6:  Unit CO256 am Beispiel von einigen
[1K5]    7:  Linien, die auf dem Bildschirm ge-
[MT2]    8:  zeichnet werden.
[PV3]    9:  von Martin Althaus Ø3/9Ø
[4Ø4]   1Ø:  (C) 199Ø Redaktionsbüro
[L83]   11:      Everts und Hagedorn
[HR3]   12:  ---------------------------------
[U7Ø]   13: }
[4L2]   14:  uses co256, crt;
[K84]   15:           {binde die Unit CO256 ein}
[RNØ]   16: var
[LZ1]   17:  index:byte;
[1U3]   18:                     {Schleifenvariable}
[MUØ]   19: begin
[ZP1]   2Ø:  mode32Øx2ØØ;
[HZ3]   21:           {Setze den Modus 32Øx2ØØ}
[SH3]   22:  for index:=Ø to 255 do
[NN5]   23:     {Zeichne 255 senkrechte Linien}
[UY3]   24:   line32Øx2ØØ(index,1,index,
[5S1]   25:                  199,index);
[C75]   26:        {über die neue Line-Funktion}
[KK3]   27:  for index:=Ø to 199 do
[ZH5]   28:    {Zeichne 199 waagerechte Linien}
[MT4]   29:   line32Øx2ØØ(256,index,319,index,
[871]   3Ø:                  index);
[C75]   31:        {über die neue Line-Funktion}
[Y44]   32:  repeat until keypressed;
[215]   33:        {Warte auf einen Tastendruck}
[VP6]   34:  modetext; {Setze Textmodus zurück}
[KNØ]   35: end.
```

Listing 3. Mit dem Programm »demo256.pas« wird auf Routinen der Unit »co256« zugegriffen

Overlays mit Turbo Pascal ab Version 5.0

Wenn Sie Turbo Pascal noch in der Version 3.0 kennen (in der Programme maximal 64 KByte Code enthalten durften), werden Sie sich an die Möglichkeit erinnern, durch den Gebrauch von Overlays den Speicherbedarf eines Programms zu senken. Mit der Überwindung der Speicherbarriere von 64 KByte ab Version 4.0 wurden die Overlays aus dem Konzept von Turbo Pascal verbannt. Seit Version 5.0 stellt die neue Standard-Unit »overlay« wieder Funktionen zur Verwaltung von Overlays zur Verfügung.

Unter einem »Overlay« versteht man einen Block von Programmcode, der nur bei Bedarf in den Speicher geladen wird. Ein gutes Beispiel für ein Overlay-Modul sind die Grafiktreiber, die Turbo Pascal in den Speicher lädt, wenn ein Programm mit dem Befehl »InitGraph« den Grafikmodus initialisiert. Für alle gängigen Grafikkarten speichert Turbo Pascal auf Diskette oder Festplatte spezielle Treiberprogramme, die Sie an der Endung »bgi« identifizieren können. Für EGA- und VGA-Karten heißt dieses Programm beispielsweise »egavga.bgi«, für Hercules-Karten »herc.bgi« und so weiter. Bei der Initialisierung eines Grafikmodus wird nur das zur verwendeten Grafikkarte passende Treiberprogramm in den Speicher geladen. Das hat den Vorteil, daß nicht der gesamte Programmcode für den Zugriff auf alle gängigen Grafikkarten komplett im Speicher stehen muß. Statt dessen lädt das Programm nur die tatsächlich für die angeschlossene Grafikkarte benötigten Teile.



Die Logik von Overlay-Units

Es gibt mehrere Anwendungsfälle, in denen der Gebrauch von Overlays nützlich sein kann: zur Realisierung von Programmen, die die Grenzen des Hauptspeichers sprengen, zur Codierung von alternativ zu verwendenden Modulen und zum Schreiben von besonders kleinen Programmen mit wenig Speicherbedarf. Overlays werden nur in den Arbeitsspeicher geladen, wenn sie das Hauptprogramm tatsächlich benötigt. Dazu reserviert das Programm im Arbeitsspeicher genügend Platz für das größte Overlay-Modul.

Die kleinste Einheit, die sich als Overlay definieren läßt, ist eine Unit. Es ist nicht möglich, einzelne Funktionen oder Prozeduren direkt als Overlays zu definieren, die entsprechenden Programmteile müssen in einer Unit zusammengefaßt werden. Beim Compilieren übersetzt Turbo Pascal normalerweise einen Pascal-

Quelltext in eine Exe-Datei. Werden in einem Pascal-Quelltext Overlays definiert, wird zusätzlich eine Datei mit der Endung »ovr« erzeugt. In dieser Datei sind alle Overlay-Module gespeichert. Nach dem Start des Programms (also der Exe-Datei) werden automatisch die momentan benötigten Overlays in den Arbeitsspeicher geladen. Die gesamte Verwaltung der Overlays übernimmt ein so erzeugtes Programm automatisch, wenn Sie sich an einige Grundregeln zum Schreiben von Overlays halten. Diese Grundregeln und die Antwort auf die Frage »Wie sieht das Gerüst für ein Overlay-Modul aus?« werden Sie in den folgenden Abschnitten an einem konkreten Beispiel kennenlernen.

Als Anwendungsbeispiel für den Umgang mit Overlays dient ein kleines Statistikprogramm. Es liest eine Anzahl von Daten, berechnet daraus einige statistische Kenngrößen und gibt die Ergebnisse auf dem Bildschirm aus. Für unsere Zwecke wird das Programm in drei Teile gegliedert: einen Programmteil zum Einlesen der Daten, ein Modul zur Ermittlung der statistischen Größen und einen weiteren Programmabschnitt, der die Ausgabe der Ergebnisse erledigt. Jeder dieser drei Programmteile wird als eigenes Overlay definiert. Wie bereits beschrieben, erwartet Turbo Pascal für jedes Overlay eine eigene Unit. Für unsere Zwecke werden drei Overlay-Units mit den Namen »lesen«, »rechnen« und »resultat« definiert. Die Units enthalten den Programmcode für das jeweilige Aufgabengebiet und werden wie »normale« Units über das Kommando »uses« im Hauptprogramm definiert. Die daraus resultierende Konstellation ist im Bild dargestellt.

Das Hauptprogramm, das für unser Beispiel den Namen »statdat.pas« trägt (Listing 1), deklariert die globalen Variablen und sorgt dafür, daß immer das benötigte Overlay im Speicher verfügbar ist. Für die Overlay-Verwaltung in Programmen stellt Turbo Pascal eine eigene Unit zur Verfügung, mit der Sie die Overlays verwalten. Diese Unit trägt den Namen »overlay« und muß als erste Unit definiert werden. Für unser Beispiel lautet die Uses-Anweisung daher:

```
uses overlay,global,lesen,rechnen,
resultat;
```

Der Befehl bewirkt, daß Turbo Pascal neben der Overlay-Unit, die für die interne Verwaltung der externen Module benötigt wird, die Funktionen und Prozeduren der drei Units »lesen«, »rechnen« und »resultat« dem Hauptprogramm zur Verfügung stellt. Zuvor wird unter dem Namen »global« eine weitere Unit eingebunden. Diese Unit soll alle globalen Variablen- und Typendefinitionen enthalten; dazu später mehr. Über die Compiler-Anweisung »$O« wird Turbo Pascal im nächsten Schritt mitgeteilt, daß die drei Units »lesen«, »rechnen« und »resultat« als Overlays arbeiten. Dazu dienen die drei folgenden Zeilen:

```
{$O lesen}
{$O rechnen}
{$O resultat}
```

Beim Compilieren bewirken diese Zeilen, daß Turbo Pascal nicht den gesamten Code der drei Units in das Programm einbindet, sondern statt dessen eine Ovr-Datei anlegt und darin alle Overlays speichert.

Eine Overlay-Unit unterscheidet sich nur durch wenige Anweisungen von einer »normalen« Unit

Für das konkrete Beispiel des Programms »statdat.pas« erhält die Overlay-Datei automatisch den Namen »statdat.ovr«. Aus der Ovr-Datei lädt Turbo Pascal nach dem Programmstart bei Bedarf den Programmcode des benötigten Overlays in den Arbeitsspeicher nach. Der Programmierer merkt davon nichts; Turbo Pascal erledigt die notwendigen Ladeoperationen ohne Zutun des Benutzers. Voraussetzung dafür ist, daß der mit dem Uses-Befehl eingebundene Overlay-Manager zunächst initialisiert wird. Dazu dient das Kommando »OvrInit«. Dem Befehl wird der Name der zu verwendenden Ovr-Datei übergeben. Für unser Beispiel lautet die erforderliche Anweisung

```
ovrinit('statdat.ovr');
```

Nach dieser Anweisung weiß Turbo Pascal bei der Programmausführung, daß sich der Programmcode der drei mit der $O-Option definierten Overlays in der Datei »statdat.ovr« befindet. Anhand der Informationen in dieser Datei bestimmt Turbo Pascal dann die Größe des Speicherblocks im Hauptspeicher, der für die größte der Overlay-Units benötigt wird.

Ist der Overlay-Manager mit der Anweisung »OvrInit« einmal installiert, sorgt Turbo Pascal intern dafür, daß sich alle Routinen der Units so nutzen lassen, als seien sie komplett in den Programmcode eingebunden. Ein so compiliertes Programm prüft vor jedem Aufruf einer der Overlay-Units, ob sie sich bereits im Speicher befindet. Wenn das nicht der Fall ist, wird die entsprechende Unit in den Overlay-Puffer geladen. Für unser Anwendungsbeispiel heißt das, daß Sie Aufrufe von Routinen aus den drei Units »lesen«, »rechnen« und »resultat« in beliebiger Reihenfolge kombinieren können. Der Overlay-Manager gewährleistet, daß der Programmcode bei Bedarf verfügbar ist.

Im Beispiel enthält jede der drei Units nur eine Prozedur. Die Routine »einlesen« in der Unit »lesen« liest eine Anzahl von Werten in ein komplettes Datenfeld in die Variable »daten«. Die Prozedur »ausrechnen« in der Unit »rechnen« ermittelt in diesem Datenfeld Minimum, Maximum und den Mittelwert und hält die Ergebnisse in einem Record fest. Die Prozedur »ausgeben« aus der Unit »resultat« schließlich gibt die statistischen Ergebnisse auf dem Bildschirm aus. Damit nicht in allen Units die Deklarationen der verwendeten Typen und Variablen erneut erfolgen müssen, ist es zweckmäßig, globale Definitionen für Variablen und Typen in einer eigenen Unit zusammenzufassen, die nicht als Overlay verwaltet wird. In unserem Fall erledigt das die Unit »global«.

Das Hauptprogramm »statdat.pas«, das die drei Overlays verwalten soll, fragt den Anwender zunächst nach der Anzahl der einzulesenden Werte und speichert sie in der Variable »anzahl«. Danach werden die drei Prozeduren »einlesen«, »ausrechnen« und »ausgeben« der drei Overlay-Units »lesen«, »rechnen« und »resultat« aufgerufen. Das komplette Programm finden Sie in Listing 1. Bei einem Blick auf das Listing werden Sie feststellen, daß sich das Programm nur durch die $O-Anweisung und den Befehl »OvrInit« von einem Programm unterscheidet, das die drei Units als statische Units und nicht als Overlays in den Programmcode einbindet.

Wie sieht es jedoch mit den aufgerufenen Units selbst aus? Die Unit »global« in Listing 2 unterscheidet sich nicht von gewöhnlichen Units, da es sich bei »global« nicht um ein Overlay handelt. Anders stellt sich die Situation bei den drei Units »lesen«, »rechnen« und »resultat« dar. In jeder der drei Units in den Listings 3 bis 5 finden Sie die Compiler-Anweisungen »$F + « und »$O + «. Diese Schalter müssen in jeder Overlay-Unit gesetzt werden. Die erste Anweisung sorgt dafür, daß alle Prozeduren und Funktionen der Unit über »Far-Sprünge« angesteuert werden. Die zweite Anweisung bindet in jede Prozedur und Funktion zusätzlichen Code ein, der überprüft, ob die Funktion sich bereits im Speicher befindet, wenn das Hauptprogramm sie anfordert. Abgesehen von diesen beiden Anweisungen unterscheiden sich die Programme für die drei Units »lesen«, »rechnen« und »resultat« nicht von den Listings »gewöhnlicher« Units. In jeder der Units wird

im Implementationsteil mit der Anweisung

```
uses global;
```

die Unit »global« eingebunden, in der die globalen Typen und Variablen definiert sind.

Nach dem Start des Hauptprogramms »statdat.pas« geschieht folgendes: Zunächst sucht Turbo Pascal bei der Ausführung von »OvrInit« nach der Datei »statdat.ovr« auf dem Datenträger. Anhand der Informationen in dieser Datei reserviert es ausreichend großen Overlay-Puffer, der genauso groß ist, daß er die größte der drei Overlay-Units »lesen«, »rechnen« und »resultat« aufnehmen kann. Bei der Bearbeitung der Prozedur »einlesen« erkennt Turbo Pascal, daß der Programmcode in der Overlay-Unit »lesen« definiert ist. Daraufhin wird die Unit »lesen« aus der Datei »statdat.ovr« in den Overlay-Puffer geladen und die Prozedur »lesen« ausgeführt. Danach trifft Turbo Pascal auf die Prozedur »einlesen«, die in der Unit »rechnen« deklariert ist. Da sich diese noch nicht im Overlay-Puffer befindet, wird »lesen« aus dem Puffer entfernt und statt dessen die Unit »rechnen« aus der Overlay-Datei »statdat.ovr« geladen und die Prozedur »einlesen« im Overlay-Puffer ausgeführt.

Obwohl die drei Units mit Hilfe der Compiler-Anweisungen als Overlay-Units definiert sind, können sie auch als »normale« Units verwendet werden. Wenn Sie aus dem Hauptprogramm »statdat.pas« in Listing 1 die $O-Anweisungen und den Befehl »OvrInit« herausnehmen und das Programm neu compilieren, erzeugt Turbo Pascal ein Programm mit insgesamt vier Units, die direkt in den Programmcode eingebunden sind. Für unser Beispiel wird man zwischen beiden Versionen, also Definition als Overlay-Units und als »normale« Units, kaum einen Unterschied in der Ausführungsgeschwindigkeit feststellen.

Das liegt daran, daß die Prozeduren der einzelnen Overlays sehr wenig Programmcode enthalten. Entweder passen sie bereits alle in den Overlay-Puffer, oder der Zugriff auf die Datei »statdat.ovr« geschieht so schnell, daß der Anwender ihn nicht bemerkt. Bei großen, umfassenden Overlay-Units hingegen kann es schnell zu einem Geduldsspiel werden, wenn unterschiedliche Prozeduren aus verschiedenen Overlay-Units in abwechselnder Reihenfolge aufgerufen werden. Wenn die Routinen ständig von der Festplatte (oder schlimmer: Diskette) geladen werden, erhält der Anwender häufig ungewollte Arbeitspausen.

Um diesen Mißstand zu beheben, bietet der Overlay-Manager von Turbo Pascal die Möglichkeit, Overlay-Units in Expanded Memory auszulagern. Expanded Memory ist der Speicherplatz auf einer Speichererweiterung, der über einen speziellen Treiber angesprochen wird. Expanded Memory wird häufig mit dem Begriff »EMS« abgekürzt. Für die Auslagerung in das Expanded Memory wird die folgende Anweisung im Programm untergebracht:

```
OvrInitEMS;
```

Dieser Befehl muß unmittelbar dem Ovrinit-Kommando folgen. Bei der Ausführung des Programms untersucht Turbo Pascal, ob eine EMS-Karte (beziehungsweise ein EMS-Treiber) installiert ist. Ist das der Fall, wird festgestellt, ob genug freier EMS-Speicher verfügbar ist, um die gesamte Ovr-Datei in den EMS-Speicher zu lesen. Wenn Turbo Pascal ausreichend Speicher findet, lädt es die Ovr-Datei in den EMS-Speicher. Die Overlay-Units werden bei der Programmausführung nicht direkt im EMS-Speicher abgearbeitet. Vielmehr wird die benötigte Unit lediglich anstatt von der Diskette oder Festplatte von der EMS-Karte in den Overlay-Puffer geladen und bei Bedarf bearbeitet.

Ein Aufruf des Befehls »OvrInitEMS« hat keinerlei negative Konsequenzen, wenn kein ausreichender EMS-Speicher gefunden wird. Dann wird wie gewohnt mit der Diskette oder Festplatte gearbeitet. Daher sollten Sie es sich zur Gewohnheit machen, das Kommando »OvrInitEMS« in jedes Programm einzubauen, das Overlays verwaltet.

(Martin Althaus/dm)

```
Name..... STATDAT.PAS

Sprache..... Turbo Pascal 5.5

Grafikkarte..... jede
```

```
[4X2]    1: program statdat;
[ØH4]    2:  uses overlay,global,lesen,
[CØ3]    3:       rechnen,resultat;
[MG3]    4:                {Binde die Units ein}
[2R5]    5:  {$o lesen}       {Diese Units sind}
[X15]    6:  {$o rechnen}           {alle drei}
[EV5]    7:  {$o resultat}      {Overlay-Units}
[RNØ]    8: var
[YX5]    9:  anzahl:byte;    {Anzahl der Werte}
[MUØ]   1Ø: begin
[C34]   11:  ovrinit('statdat.ovr');
[SW5]   12:   {Installiere den Overlay-Manager}
[XF1]   13:  writeln;
[FD5]   14:  write('Wieviele Werte möchten ');
[EL3]   15:  write('Sie eingeben? ');
[CC5]   16:     {Frage die Anzahl der Werte ab}
[P31]   17:  repeat
[LG2]   18:   readln(anzahl);
[KK2]   19:  until anzahl>Ø;
[NF6]   2Ø:   {Es muß mindestens ein Wert sein}
[CØ4]   21:  einlesen(anzahl,daten);
[XF3]   22:                 {Lese die Werte ein}
[D46]   23:  ausrechnen(anzahl,daten,ergebnisse);
[WB4]   24:           {Errechne die Ergebnisse}
[RM3]   25:  ausgeben(ergebnisse);
[HB4]   26:           {und gebe den Report aus}
[KNØ]   27: end.
```

Listing 1. Das Programm »statdat.pas« verwendet drei Overlays

```
Name..... GLOBAL.PAS

Sprache..... Turbo Pascal 5.5

Grafikkarte..... jede
```

```
[K22]    1: unit global;
[124]    2: {-----------------------------------
[TP1]    3:              global.pas
[531]    4:              ----------
[BF5]    5:  In dieser Unit wird die Defintion
[NC5]    6:  der globalen Typen und Variablen
[M72]    7:  vorgenommen.
[WT3]    8:  von Martin Althaus 4/9Ø
[4Ø4]    9:  (C) 199Ø Redaktionsbüro
[L83]   1Ø:      Everts und Hagedorn
[HR3]   11:  -----------------------------------
[U7Ø]   12: }
[SK1]   13: interface
[STØ]   14: type
[SB4]   15:  feld= array[1..1ØØ] of real;
[L64]   16:              {Zur Aufnahme der Daten}
[LWØ]   17:  wert=
[Y11]   18:   record
[Y45]   19:    maximum,minimum,mittelwert:
[TTØ]   2Ø:    real;
[ZNØ]   21:   end;
[545]   22:        {Zur Aufnahme der Ergebnisse}
[RNØ]   23: var
[9X1]   24:  daten: feld;
[KY2]   25:  ergebnisse: wert;
[Y46]   26: implementation   {Es ist keine Im-}
[KW5]   27: end.       {plementation notwendig}
```

Listing 2. Alle globalen Definitionen nimmt die Unit »global.pas« vor

```
Name..... LESEN.PAS

Sprache..... Turbo Pascal 5.5

Grafikkarte..... jede
```

```
[RZ1]    1: unit lesen;
[124]    2: {---------------------------------
[5K1]    3:              lesen.pas
[YY0]    4:              ---------
[GN5]    5:  Overlay-Unit zum Einlesen von Da-
[5K3]    6:  ten in ein real-Array.
[WT3]    7:  von Martin Althaus 4/90
[4Ø4]    8:  (C) 199Ø Redaktionsbüro
[L83]    9:      Everts und Hagedorn
[HR3]   1Ø:  ---------------------------------
[U7Ø]   11: }
[TY5]   12:  {$f+}  {Beide Schalter müssen für}
[3Y5]   13:  {$o+}    {Overlays gesetzt werden}
[SK1]   14: interface
[G22]   15:  uses global;
[KB5]   16:    {Bindet Variablen- und Typende-}
[SØ3]   17:                    {finitionen ein}
[MC5]   18: procedure einlesen(anzahl:byte;
[1N2]   19:                    var daten:feld);
[AH4]   2Ø:                {Definition des Kopfes}
[4L2]   21: implementation
[RB5]   22: uses crt;               {für CLRSCR}
[MC5]   23: procedure einlesen(anzahl:byte;
[1N2]   24:                  var daten:feld);
[124]   25: {---------------------------------
[HC5]   26:  Liest die mit ANZAHL übergebene
[LG5]   27:  Zahl von Werten in das Real-Array
[NE1]   28:  DATEN ein
[HR3]   29:  ---------------------------------
[U7Ø]   3Ø: }
[RNØ]   31: var
[UA6]   32:  index:byte;     {für Zählschleife}
[MUØ]   33: begin
[GX4]   34:  clrscr;            {leerer Screen}
[RL5]   35:  write('Erfassung statistischer');
[2Y2]   36:  writeln(' Daten');
[DY3]   37:  write('----------------------');
[ZK2]   38:  writeln('------');
[XF1]   39:  writeln;
[WX3]   4Ø:              {Ausgabe der Headline}
[S54]   41:  for index:=1 to anzahl do
[MK4]   42:           {Verarbeite ANZAHL Werte}
[MUØ]   43:  begin
[8A4]   44:   write(index,'.-ter Wert: ');
[LN3]   45:   readln(daten[index]);
[EV2]   46:                    {Daten einlesen}
[N64]   47:  end;                    {von FOR}
[ZX4]   48: end;                  {von EINLESEN}
[4Ø5]   49: end.                {Das war's schon}
```

Listing 3. Die Overlay-Unit »lesen.pas« liest Daten in ein Array

```
Name..... RECHNEN.PAS

Sprache..... Turbo Pascal 5.5

Grafikkarte..... jede
```

```
[B72]    1: unit rechnen;
[124]    2: {---------------------------------
[722]    3:              rechnen.pas
[T81]    4:              -----------
[995]    5:  Overlay-Unit zum Errechnen von
[DT5]    6:  statitischen Größen eines Real-
[C51]    7:  Arrays
[WT3]    8:  von Martin Althaus 4/90
[4Ø4]    9:  (C) 199Ø Redaktionsbüro
[L83]   1Ø:      Everts und Hagedorn
[HR3]   11:  ---------------------------------
[U7Ø]   12: }
[TY5]   13:  {$f+}  {Beide Schalter müssen für}
[3Y5]   14:  {$o+}    {Overlays gesetzt werden}
[SK1]   15: interface
[G22]   16:  uses global;
[TE5]   17:      {Bindet globale Variablen und}
[ØX3]   18:               {Typ-Defintionen ein}
[6T5]   19: procedure ausrechnen(anzahl:byte;
[EN2]   2Ø:                    var daten:feld;
[ØP3]   21:                var ergebnisse:wert);
[LY3]   22:               {Defintion des Kopfes}
[4L2]   23: implementation
[6T5]   24: procedure ausrechnen(anzahl:byte;
[EN2]   25:                    var daten:feld;
[ØP3]   26:                var ergebnisse:wert);
[124]   27: {---------------------------------
[9G5]   28:  Berechnet in dem Real-Array DATEN
[8Ø6]   29:  der Größe ANZAHL statistische Da-
[XL5]   3Ø:  ten und legt die Ergebnisse im Re-
[M22]   31:  cord WERT ab.
[HR3]   32:  ---------------------------------
[U7Ø]   33: }
[RNØ]   34: var
[UA6]   35:  index:byte;     {für Zählschleife}
[MUØ]   36: begin
[S63]   37:  with ergebnisse do
[R56]   38:  {Adressiere den ERGEBNISSE-Record}
[MUØ]   39:  begin
[HS3]   4Ø:   mittelwert:=daten[1];
[Ø43]   41:                    {Initialisiere}
[KX5]   42:   maximum:=daten[1];  {die statis-}
[UG5]   43:   minimum:=daten[1];      {tischen}
[ZA2]   44:                  {Variablen}
[T54]   45:   for index:=2 to anzahl do
[TF5]   46:   begin    {Durchsuche das gesamte}
[791]   47:                              {Feld}
[U84]   48:    if daten[index]<minimum
[4D4]   49:          {Neues Minimum gefunden?}
[ZW4]   5Ø:     then minimum:=daten[index];
[AG2]   51:                    {ja, setze es}
[C94]   52:    if daten[index]>maximum
[KD4]   53:          {Neues Maximum gefunden?}
[9W4]   54:     then maximum:=daten[index];
[AG2]   55:                    {Ja, setze es}
[1B4]   56:    mittelwert:=mittelwert+
[9C2]   57:                daten[index];
[FN3]   58:                 {Summe aufaddieren}
[VØ4]   59:   end;                   {von FOR}
[YN5]   6Ø:   mittelwert:=mittelwert / anzahl;
[Y94]   61:              {Mittelwert errechnen}
[5A4]   62:  end;                    {von WITH}
[L25]   63: end;                {von AUSRECHNEN}
[4Ø5]   64: end.               {Das war's schon}
```

Listing 4. Die Berechnung statistischer Daten führt die Overlay-Unit »rechnen.pas« durch

```
Name..... RESULTAT.PAS

Sprache..... Turbo Pascal 5.5

Grafikkarte..... jede
```

```
[1H2]    1: unit resultat;
[124]    2: {---------------------------------
[W42]    3:              resulat.pas
[T81]    4:              -----------
[AT5]    5:  Overlay-Unit zur Ausgabe von sta-
[2H5]    6:  titischen Daten, die über andere
[L94]    7:  Overlays berechnet wurden
[WT3]    8:  von Martin Althaus 4/90
[4Ø4]    9:  (C) 199Ø Redaktionsbüro
[L83]   1Ø:      Everts und Hagedorn
[HR3]   11:  ---------------------------------
[U7Ø]   12: }
[TY5]   13:  {$f+}  {Beide Schalter müssen für}
[3Y5]   14:  {$o+}    {Overlays gesetzt werden}
[SK1]   15: interface
[G22]   16:  uses global;
[WM5]   17:    {Bindet die Variablen und Typen}
[H11]   18:                              {ein}
[9R5]   19: procedure ausgeben(var ergebnisse:
[WØ1]   2Ø:                            wert);
[AH4]   21:              {Definition des Kopfes}
[4L2]   22: implementation
[9R5]   23: procedure ausgeben(var ergebnisse:
[WØ1]   24:                            wert);
[124]   25: {---------------------------------
[7A5]   26:  Gibt die zuvor vom Unit RECHNEN
[136]   27:  errechneten statitischen Größen,
[HB5]   28:  die im Record ERGEBNISSE stehen,
[YR3]   29:  auf dem Bildschirm aus
[HR3]   3Ø:  ---------------------------------
[U7Ø]   31: }
[MUØ]   32: begin
[S63]   33:  with ergebnisse do
[8G5]   34:      {Adressiere ERGEBNISSE-Record}
[MUØ]   35:  begin
[XF1]   36:   writeln;
[YL4]   37:   writeln('Die Ergebnisse:');
[VM3]   38:   writeln('---------------');
[455]   39:   writeln;           {Überschrift}
[7N4]   4Ø:   write('Das Minimum beträgt');
[CF3]   41:   writeln(minimum,'.');
[W35]   42:   write('Als Maximalwert wurde');
[RD5]   43:   writeln(maximum,' ermittelt.');
[P35]   44:   write('Der Mittelwert liegt ');
[YY4]   45:   writeln('bei ',mittelwert,'.');
[5A4]   46:  end;                    {von WITH}
[5X4]   47: end;                {von AUSGEBEN}
[4Ø5]   48: end.               {Das war's schon}
```

Listing 5. Die Overlay-Unit »resultat.pas« gibt die Ergebnisse aus

Der Diskettendoktor hilft in allen Lebenslagen

Die Organisation von Speichermedien wie Festplatten oder Disketten gehört zu den interessantesten Gebieten der PC-Technik. In vielen Fällen ist es hilfreich, den Inhalt eines Datenträgers direkt unter die Lupe zu nehmen. Der Diskettendoktor nimmt sich dieser Aufgabe gerne an.

Viele Anwender stehen vor der Situation, daß ihre Datenträger nicht das beinhalten, was sie eigentlich sollten. Konsultieren Sie in leichten oder sogar schwerwiegenden Fällen doch einfach den Diskettendoktor. Er weiß fast immer Rat und bringt Ihre elektronischen Datenträger auf Vordermann.

Das nachstehende Programm enthält alle Funktionen, die ein guter Diskettеneditor benötigt. Compilieren läßt es sich mit Turbo Pascal ab Version 4.0. Der Diskettendoktor ist sowohl mit Farb- als auch monochromen Bildschirmen lauffähig. Die farbige Gestaltung erhöht die Übersichtlichkeit und erleichtert den Umgang mit dem Befehlssatz des Utilities.

Nachdem Sie das Listing abgetippt und compiliert haben, starten Sie den Diskettendoktor »diskdoc.exe«, der sich daraufhin mit seinem Titelbild präsentiert. Nach dem Betätigen einer Taste gelangen Sie zum Datenbildschirm. Hier werden im unteren Bereich des Monitors die erlaubten Kommandos dargestellt, während der obere Bereich zur Ausgabe der Diskettendaten dient (Bild 1). Eine Übersicht der Programmfunktionen finden Sie in Tabelle 1.

Im oberen Teil des Bildschirms zeigt das Programm den Inhalt eines Sektors des eingelegten Datenträgers im ASCII- und im Hexadezimalformat an. Mit den Cursor-Tasten führen Sie die Schreibmarke über das gesamte Datenfeld. Sind Sie an der Position angekommen, an der Sie Daten ändern wollen, dann geben Sie einfach die neuen Werte ein (im Hexadezimalfeld Hexadezimalzahlen und im ASCII-Bereich einzelne Zeichen). Nach jeder Eingabe springt der Cursor zum nächsten Eingabefeld, bis das Seitenende erreicht ist. Wenn Sie einen Farbmonitor besitzen, erleichtert Ihnen der Diskettendoktor auch die Unterscheidung zwischen Steuercodes und üblichen ASCII-Zeichen. Da Zeichen mit einem ASCII-Wert kleiner 32 nicht sinnvoll dargestellt werden können, gibt der Diskettendoktor an ihrer Stelle Punkte im ASCII-Bereich aus. Um die Steuercodes vom »tatsächlichen« Punkt (ASCII-Wert 46) abzugrenzen, erscheinen diese Punkte in brauner Farbe auf dem Bildschirm.

Ein Sektor enthält bekanntlich 512 Byte. Da der Diskettendoktor nur 256 Byte auf einmal anzeigen kann, ist ein Sektor in zwei Bildschirmseiten aufgeteilt. Mit den Tasten [PgUp] und [PgDn] wechseln Sie zwischen den beiden Seiten.

Mit [F2] wählen Sie das gewünschte Laufwerk. Geben Sie zu diesem Zweck einfach den gewünschten Laufwerksbuchstaben ein (beispielsweise »b«). Ein abschließendes Quittieren mit der Enter-Taste ist nicht notwendig. Ähnlich verfahren Sie mit der Funktionstaste [F1]: Diese dient dazu, einen bestimmten Sektor anzuwählen. Der Unterschied besteht darin, daß Sie in diesem Fall eine Zahl eingeben, die den Sektor bestimmt, und anschließend die Enter-Taste drücken müssen. Mit [F3] und [F4] lesen Sie den folgenden beziehungsweise den vorhergehenden Sektor von der Diskette. Die durch die Funktionstaste [F6] aufgerufene Routine schreibt hingegen den aktuellen Sektor auf den Datenträger.

Doch nicht nur Disketten, sondern auch Festplatten lassen sich mit dem Diskettendoktor »manipulieren«. Das Betriebssystem steuert eine Festplatte normalerweise genauso an wie ein Diskettenlaufwerk. Doch obwohl der Anwender auch bei der Festplatte den Kopf, die Spur und den Sektor spezifizieren muß, gibt es einige Unterschiede.

So ist der Benutzer in der Lage, die Festplatte für verschiedene Betriebssysteme einzurichten. Sie können beispielsweise einen Teil der Platte für MS-DOS, einen weiteren für Xenix und einen anderen für CP/M verwenden. Diese Aufteilung in mehrere Bereiche nennt man »partitionieren«, die einzelnen Bereiche heißen »Partitions« (englisch für »Aufteilung«).

Taste	Funktion
[F1]	Lesen eines bestimmten Sektors
[F2]	Auswählen des aktiven Laufwerks
[F3]	Lesen des folgenden Sektors
[F4]	Lesen des vorhergehenden Sektors
[F6]	Schreiben des aktuellen Sektors auf den Datenträger
[F8]	Anzeige der im Boot-Record gespeicherten Daten
[F9]	Anzeige der Partition-Tabelle der Festplatte
[F10]	Beenden des Diskettendoktors

Tabelle 1. Der Diskettendoktor läßt sich auf einfache Weise über Funktionstasten bedienen

Die Partition-Table gibt Aufschluß über die Aufteilung der Festplatte

Da die Partitionierung eine wichtige Rolle in der Datenverwaltung des Anwenders spielt, informiert Sie der Diskettendoktor über die wichtigsten zugehörigen Daten. Die Informationen, welches Betriebssystem welche Bereiche der Festplatte benutzt, sind in der sogenannten »Partition-Table« (Aufteilungstabelle) abgelegt. In dieser Aufstellung ist außerdem gespeichert, an welcher Stelle welches Betriebssystem abgelegt ist, wie groß der Bereich ist und welcher Bereich

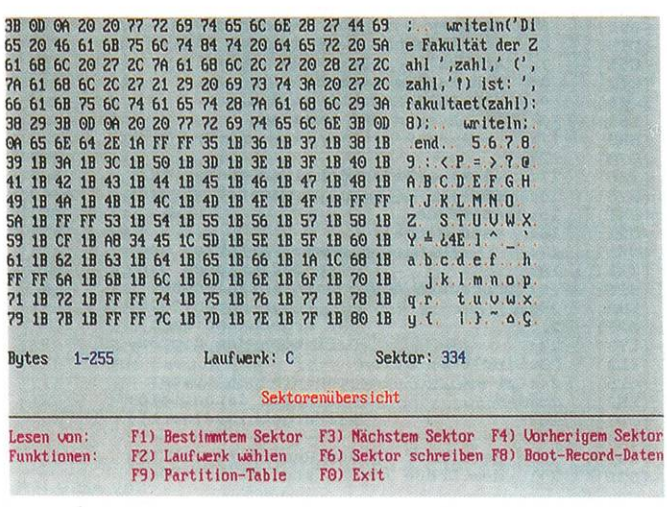


Bild 1. Neben der Hexadezimalform stellt das Utility die Daten auch als ASCII-Zeichen auf dem Bildschirm dar

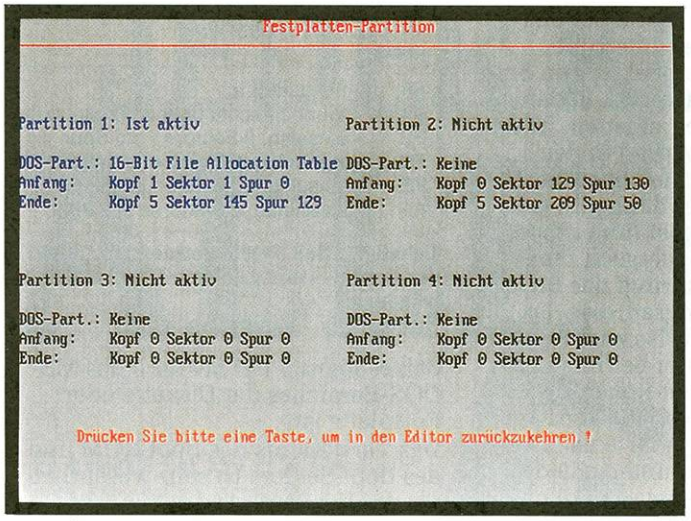


Bild 2. Die Partition-Daten liefern wichtige Informationen über die Struktur der Festplatte

aktiviert wurde. Der Diskettendoktor informiert Sie auch über dieses komplexe Gebiet der Datenverwaltung. Nachdem Sie die Funktionstaste [F9] betätigt haben, gibt das Utility die Festplatten-Partition-Tabelle auf dem Bildschirm aus (Bild 2). Das Programm liest die Daten über den Betriebssystem-Interrupt 13hex ein. Die Tabelle ist 64 Byte lang und befindet sich im ersten Sektor auf Spur 1 des ersten Kopfes. Das Programm liest die Tabelle in den Datenpuffer der Feldvariable »Datenfeld«. Da jeder Informationsblock aus 16 Byte besteht, lassen sich höchstens vier Betriebssysteme gleichzeitig einsetzen. Der Diskettendoktor zeigt Ihnen aus diesem Grund vier Felder auf dem Bildschirm, die Informationen über die jeweiligen Bereiche liefern.

In der obersten Zeile jedes Blockes wird angegeben, ob die zugehörige Partition aktiv ist. Aktive Partitions sind auf Farbbildschirmen blau hervorgehoben. Die folgende Zeile gibt darüber Auskunft, ob der Bereich zu MS-DOS gehört oder mit einem unbekannten Betriebssystem belegt ist. Die letzten beiden Zeilen informieren Sie über den physikalischen Anfang und das Ende der jeweiligen Partition. Nach der Anzeige der Festplatten-Partition kehren Sie durch Drücken einer beliebigen Taste in das Hauptmenü zurück.

Neben den Standardkommandos eines Diskettenmonitors deckt der Befehlssatz des Diskettendoktors auch Sonderbereiche ab. So läßt sich mit Hilfe einer Spezialfunktion der Inhalt des sogenannten »Boot-Sektors« anzeigen, der Informationen über wichtige Konfigurationsdaten liefert.

Kaum ein PC-Benutzer weiß, was passiert, wenn sein Computer das Betriebssystem von Diskette oder Festplatte lädt. In diesem Moment schlägt die Stunde beziehungsweise Sekunde des Boot-Sektors. Seine Aufgabe ist durch die geschichtliche Entwicklung von MS-DOS festgelegt. Ausgehend von der Aufgabenstellung, ein Betriebssystem zu entwickeln, das möglichst rechnerunabhängig eingesetzt werden kann, stellten IBM und Microsoft gemeinsam das Betriebssystem MS-DOS beziehungsweise PC-DOS (so heißt MS-DOS auf Original-IBM-Computern) vor.

Der Boot-Record liefert dem Betriebssystem Informationen über den Aufbau eines Datenträgers

Die erste Version dieses Betriebssystems nannte man MS-DOS 1.0. Sie unterscheidet sich zum Teil erheblich von den heute in vielen Fällen verwendeten Versionen der 3er- oder 4er-Reihe. Von vornherein konzeptionierten die Entwickler MS-DOS (»Microsoft Disk Operating System«) so, daß es problemlos auf unterschiedlichen Computern lief. Auf der anderen Seite mußte garantiert sein, daß man innovative Neu- oder Weiterentwicklungen einsetzen konnte, ohne daß die unter früheren MS-DOS-Versionen entwickelten Programme ihre Funktionsfähigkeit verloren. Die beiden Schlagworte zum Betriebssystem MS-DOS hießen – und heißen auch heute noch: Portabilität (»Übertragbarkeit«) und Kompatibilität (»Vergleichbarkeit«).

Aus programmtechnischer Sicht war ein solches Konzept jedoch nicht leicht in Software umzusetzen. Die Probleme fingen schon beim Starten des Computers an: Jeder Benutzer eines PC hat sich schon einmal gefragt, was in seinem Rechner geschieht, wenn er ihn einschaltet. Denn bevor der Computer einsatzfähig ist, benötigt er einen relativ langen Zeitraum, ehe er sich mit der Eingabeaufforderung (englisch »Prompt«) meldet. Den Vorgang, den der PC ausführt, um die Hardware des Computers zu testen und das Betriebssystem zu laden, nennt man »Booten«.

Schritt	Aufgabe
1	Verhindern der NMI (»Nicht Maskierbarer Interrupt«)
2	POST-Selbsttest
3	NVR-Test (batteriegepufferter Systemspeicher)
4	Initialisierung verschiedener Hardware-Bausteine
5	Initialisierung des POST-Variablenbereichs
6	Initialisierung der POST-Interrupts
7	Initialisierung der VDU
8	Initialisierung des 8259-Interrupt-Controllers
9	Ausgabe der Einschaltmeldung
10	NMI erlauben
11	Initialisierung aller externen ROMs
12	Booten des Betriebssystems

Tabelle 2. Der Boot-Vorgang eines PC läuft in genau festgelegten Schritten ab

Die Startphase läßt sich in zwei Perioden unterteilen: Bevor der Computer das angeschlossene Disketten- oder Festplattenlaufwerk anspricht, initialisiert er die Hardware und das Betriebssystem. Die Routine für die Hardware-Initialisierung und der Selbsttest werden an der Adresse FFFF0hex gestartet, der Startadresse der CPU nach dem Zurücksetzen des PC (englisch »CPU Reset Entry Point«). Der an dieser Adresse gespeicherte Wert gibt den Anfang der Routine an, die alle notwendigen Aufgaben erfüllt. Sie initialisiert die Hardware, leitet Selbsttests ein und formatiert den Variablen- und Parameterbereich des »POST« (englisch »Power On Self Test«). Der POST benutzt den Arbeitsspeicher des PC erst, wenn dieser erfolgreich überprüft wurde. Die Initialisierung läuft in dem in Tabelle 2 wiedergegebenen Zeitplan ab.

Ein Problem tritt bei Punkt 12 der Beschreibung aus Tabelle 2 auf: Wie soll man auf unterschiedlichen Computern dieselbe Boot-Routine verwenden, wenn doch die Hardware-Voraussetzungen völlig anders sind? Die Entwickler von Microsoft und IBM fanden eine ideale Lösung – den Boot-Record. Das Verfahren basiert auf der Überlegung, die Informationen, wie der Computer geladen werden muß, auf Diskette statt im ROM abzulegen. Bei einem weiterentwickelten PC genügt es dann, den Boot-Record zu ändern, um die Boot-Routine an die gestiegene Leistungsfähigkeit anzupassen.

Der Boot-Record enthält neben den wichtigsten Daten des angeschlossenen Festplatten- oder Diskettenlaufwerks eine Miniroutine zum Auslö-

Byte (hex)	Bedeutung
00 bis 02	Sprungbefehl zum Boot-Programm
03 bis 0A	Name der Diskette (8 Byte ASCII)
0B bis 0C	Anzahl der Bytes pro Sektor
0D	Anzahl der Sektoren pro Cluster
0E bis 0F	Anzahl der Sektoren des Boot-Records
10	Anzahl der Kopien der FAT
11 bis 12	Anzahl der Einträge im Stammverzeichnis
13 bis 14	Anzahl der Sektoren der Diskette/Festplatte
15	Identifizierungscode der Diskette/Festplatte (Tabelle 4)
16 bis 17	Anzahl der Sektoren der FAT
18 bis 19	Anzahl der Sektoren pro Spur
1A bis 1B	Anzahl der Diskettenseiten/Plattenköpfe
1C bis 1D	Anzahl der reservierten Sektoren

Tabelle 3. Die Struktur des Boot-Sektors vereinfacht den Startvorgang des Betriebssystems

sen des Boot-Vorgangs. Mit den Informationen des Boot-Records läßt sich einerseits ein Betriebssystem (oder ein anderes Programm) laden, und andererseits wird dem System damit das aktuelle Laufwerk gemeldet.

Der Boot-Record befindet sich immer im logischen Sektor 0 des Disketten- oder Festplattenlaufwerks und enthält die in Tabelle 3 aufgeführten Informationen. An die Parametertabelle wird eine Routine zum Starten des Rechners angehängt. Die ersten drei Byte des Boot-Records weisen auf diese Routine. Das Boot-ROM reserviert zunächst einen 512 Byte großen Speicherpuffer, der bei der Adresse 0000:7C00hex im Arbeitsspeicher beginnt und bei 0000:7DFFhex endet. Dannach wird bis zu viermal versucht, das angeschlossene Diskettenlaufwerk zurückzusetzen. War dieser Vorgang erfolgreich, dann benutzt die ROM-Routine den Interrupt 13hex, um den ersten Sektor von Seite 0, Spur 0 auf Laufwerk A in den eingerichteten Puffer zu laden. Ist der erste 16-Bit-Wert (»Wort«) in dem Puffer gleich 0 oder gleicht er den folgenden zehn Byte, dann gibt die Routine eine Fehlermeldung aus und hält an. War der Leseversuch hingegen erfolgreich, so springt das BIOS danach zu der Anfangsadresse des Puffers beziehungsweise zum Beginn des Boot-Records. Hier befinden sich drei Byte, die die Boot-Routine des Boot-Records aufrufen. Die genaue Form dieses Sprunges ist je nach verwendetem Computer unterschiedlich. Es kann sich dabei sowohl um einen 2 Byte langen »Short Jump« als auch um einen 3 Byte langen »Far Jump« handeln.

Es kann aber auch der Fall eintreten, daß das Diskettenlaufwerk nicht bereit ist, weil beispielsweise keine Diskette eingelegt, die Verschlußklappe noch geöffnet oder gar kein Diskettenlaufwerk angeschlossen ist. In diesem Fall versucht das Boot-ROM bis zu dreimal, den logisch ersten Sektor von der Festplatte in den Puffer zu lesen. War das erfolgreich, wird das letzte gelesene Wort mit dem Wert AA55hex verglichen. Sind beide gleich, dann folgt wiederum der Sprung zu der ersten Anweisung im Puffer. In allen anderen Fällen wiederholt die Boot-Routine den Vorgang. War es nicht möglich, von der Festplatte zu booten, verzweigt das BIOS zur Interrupt-Routine 18hex. Bei älteren Computern befindet sich hier ein Zeiger auf das ROM-Basic, die neueren PCs liefern dagegen eine Meldung der Form »Bitte eine Systemdiskette einlegen«.

Bei Festplatten kann der Boot-Record an einer anderen Stelle als am physikalischen Anfang des Datenträgers liegen (zum Beispiel bei unterschiedlichen Partitionen). Wichtig ist nur, daß es logisch der Sektor 0 des MS-DOS-Bereiches der Diskette oder Festplatte ist.

Inhalt (hex)	Datenträgertyp
F8	Festplatte
F9	Vierfachdichte oder 3,5 Zoll
FC	einseitig, 9 Sektoren pro Spur
FD	zweiseitig, 9 Sektoren pro Spur
FE	einseitig, 8 Sektoren pro Spur
FF	zweiseitig, 8 Sektoren pro Spur

Tabelle 4. Das Byte Nummer 21 (15hex) des Boot-Sektors gibt den Typ des Datenträgers an

Das Verständnis des Boot-ROM und des Boot-Sektors ist sehr wesentlich für das Verständnis des MS-DOS. Wenn Sie die interne Struktur dieses wichtigen Systemvorgangs erkannt haben, ist es ziemlich einfach, ihn zu verändern. Sie rufen diese Spezialfunktion des Diskettendoktors über die Funktionstaste [F8] auf. Nach der Anzeige der Daten des Boot-Records kehren Sie durch das Drücken einer beliebigen Taste in das Hauptmenü zurück.

Gegenüber vielen anderen Diskettenditoren zeichnet sich Diskdoc durch einen wesentlichen Vorteil aus: Fast alle eingesetzten Betriebssysteme benutzen die gleichen BIOS-Interrupts für ihr internes Datenmanagement. Mit dem BIOS können Sie also auf die gesamte Festplatte zugreifen, während die Aktivitäten von MS-DOS auf die Bereiche des Datenträgers begrenzt sind, die auch tatsächlich für MS-DOS zugänglich sind. Da der Diskettendoktor die BIOS-Interrupts benutzt, bearbeiten Sie mit ihm ohne Schwierigkeiten unterschiedliche Betriebssysteme auf einer Festplatte.

Mit Hilfe der Funktionstaste [F10] beenden Sie schließlich das Programm und kehren in die Benutzeroberfläche zurück.

(Markus Zietlow/zi)

```
[GH5]    1: (* Programmautor:    Markus Zietlow
[PM4]    2:    Programmname:     Diskdoktor
[6E3]    3:    Programmversion:  2.Ø
[6H5]    4:    (C)opyright:      DOS International
[685]    5:    Aufgabe:          Diskетteneditor   *)
[ØØØ]    6:
[Z24]    7: program Diskdoktor_II;
[ØØØ]    8:
[S62]    9: uses crt,dos;
[ØØØ]   1Ø:
[STØ]   11: type
[9H4]   12:   Feld = array[Ø..5ØØØ] of byte;
[ØØØ]   13:
[6XØ]   14: const
[AM1]   15:   Error=255;
[ØØØ]   16:
[RNØ]   17: var
[BL3]   18:   Hauptlaufwerk : byte;
[D82]   19:   Fehler : word;
[FA2]   2Ø:   Sektor : word;
[272]   21:   Puffer : Feld;
[YF3]   22:   Register : Registers;
[ØØØ]   23:
```

```
[DD6]   24: function FillString(Anzahl : integer;
[D64]   25:                     Zeichen:string):string;
[RNØ]   26: var
[NØ3]   27:   Zaehler : integer;
[WR2]   28:   Zkette  : string;
[MUØ]   29: begin
[8U1]   3Ø:   Zkette:='';
[TF4]   31:   for Zaehler:=1 to Anzahl do
[T44]   32:     Zkette:=Zkette + Zeichen;
[CG3]   33:   FillString:=Zkette;
[ZNØ]   34: end;
[ØØØ]   35:
[XN3]   36: procedure HideCursor;
[RNØ]   37: var
[ØT2]   38:   Reg : Registers;
[MUØ]   39: begin
[RS1]   4Ø:   Reg.ah:=$Ø2;
[ZL1]   41:   Reg.bx:=Ø;
[792]   42:   Reg.dx:=$ffff;
[U32]   43:   intr($1Ø,Reg);
[ZNØ]   44: end;
```

Listing. Mit »diskdoc.pas« fühlen Sie Ihren Datenträgern auf den Zahn

ALLE BRAUCHEN DOS
JEDER KENNT DOS
ALLE LESEN

DOS UND DOS EXTRA IM ABONNEMENT MIT PREISVORTEIL

DOS

Im Inland bzw. West-Berlin
6 Ausgaben = 42,– DM
12 Ausgaben = 82,50 DM
Im europäischen Ausland
6 Ausgaben = 55,– DM
12 Ausgaben = 110,– DM
Im außereuropäischen Ausland
6 Ausgaben = 75,– DM
12 Ausgaben = 150,– DM

DOS EXTRA

Im Inland bzw. West-Berlin
4 Ausgaben = 60,– DM
Im europäischen Ausland
4 Ausgaben = 70,– DM
Im außereuropäischen Ausland
4 Ausgaben = 80,– DM

Bitte benutzen Sie die Bestellkarte.
DMV-VERLAG · POSTFACH 250
3440 ESCHWEGE

DMV Daten- und Medienverlag

Widerrufsrecht:
Jeder Abonnent hat das Recht, seine Bestellung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich zu widerrufen. Die rechtzeitige Absendung des Widerrufsschreibens genügt zur Fristwahrung.

```
[ØØØ]   45:
[783]   46: procedure Warnung;
[MUØ]   47: begin
[Z72]   48:   write (chr(7));
[ZNØ]   49: end;
[ØØØ]   5Ø:
[1X5]   51: procedure TastaturpufferLoeschen;
[RNØ]   52: var
[DH2]   53:   Zeichen : char;
[MUØ]   54: begin
[BD6]   55:   while KeyPressed do Zeichen:=ReadKey;
[ZNØ]   56: end;
[ØØØ]   57:
[U74]   58: procedure Fehlermeldung;
[MUØ]   59: begin
[KF1]   6Ø:   Warnung;
[3T2]   61:   window(14,6,66,12);
[6M3]   62:   if LastMode = CO8Ø then
[8E4]   63:     TextAttr := White + Red * 16;
[DB1]   64:   ClrScr;
[6T2]   65:   window(14,6,67,12);
[KC6]   66:   writeln(chr(2Ø1),FillString(51,chr(2Ø5)),
[ØF1]   67:         chr(187));
[8A6]   68:   writeln(chr(186),FillString(51,chr(32)),
[WE1]   69:         chr(186));
[VB3]   7Ø:   write(chr(186),chr(32));
[7C2]   71:   case Fehler of
[T76]   72:     $Ø2 : write('Unbekannter Fehler (beim'+
[HA2]   73:                 ' Zugriff auf '+
[5M2]   74:                 'Datenträger)');
[9E5]   75:     $Ø3 : write('Schreibzugriff auf '+
[PØ4]   76:                 'schreibgeschützter '+
[MM2]   77:                 'Datenträger');
[YA6]   78:     $Ø4 : write('Gewünschter Sektor auf '+
[TF3]   79:                 'Datenträger nicht '+
[PU1]   8Ø:                 'gefunden');
[ME5]   81:     $Ø8 : write('Fehlerhafter CRC auf '+
[383]   82:                 'Datenträger beim '+
[YK2]   83:                 'Lesezugriff');
[GD5]   84:     $4Ø : write('SEEK-Operation war '+
[GP3]   85:                 'fehlerhaft-Zugriff '+
[WD2]   86:                 'abgebrochen');
[FE5]   87:     $8Ø : write('Geräteeinheit beim '+
[N74]   88:                 'Zugriffsversuch nicht '+
[4V1]   89:                 'reagiert');
[ZNØ]   9Ø:   end;
[Ø34]   91:   writeln(chr(32),chr(186));
[8A6]   92:   writeln(chr(186),FillString(51,chr(32)),
[WE1]   93:         chr(186));
[B86]   94:   writeln(chr(186),' Drücken Sie bitte '+
[885]   95:         'eine Taste um weiterzumachen '+
[NU1]   96:         '!  ',chr(186));
[8A6]   97:   writeln(chr(186),FillString(51,chr(32)),
[WE1]   98:         chr(186));
[PU5]   99:   write(chr(2ØØ),FillString(51,chr(2Ø5)),
[4F1]  1ØØ:         chr(188));
[Ø32]  1Ø1:   HideCursor;
[Y44]  1Ø2:   repeat until KeyPressed;
[2B4]  1Ø3:   TastaturpufferLoeschen;
[GH2]  1Ø4:   window(1,1,8Ø,25);
[6M3]  1Ø5:   if LastMode = CO8Ø then
[5L5]  1Ø6:     TextAttr := Black + LightGray * 16;
[ZNØ]  1Ø7: end;
[ØØØ]  1Ø8:
[K76]  1Ø9: procedure SektorLesen(Error : word;
[G93]  11Ø:                       Lw,Skt,Az : integer;
[L23]  111:                       var Buffer : feld);
[MUØ]  112: begin
[A95]  113:   inline($1e);          {  push ds          }
[1A6]  114:   inline($c5/$5e/$Ø4); {  lds  bx,[bp+Ø4] }
[UC6]  115:   inline($8b/$4e/$Ø8); {  mov  cx,[bp+Ø8] }
[8D6]  116:   inline($8b/$56/$Øa); {  mov  dx,[bp+Øa] }
[PB6]  117:   inline($8a/$46/$Øc); {  mov  al,[bp+Øc] }
[G75]  118:   inline($55);          {  push bp          }
[T75]  119:   inline($cd/$25);      {  int  25          }
[6H5]  12Ø:   inline($72/$Ø2);      {  jc   weiter      }
[1M5]  121:   inline($33/$cØ);      {  xor  ax,ax       }
[H45]  122:   inline($9d);          {f:popf             }
[R65]  123:   inline($5d);          {  pop  bp          }
[W76]  124:   inline($89/$46/$Øe); {  mov  [bp+Øe],ax }
[775]  125:   inline($1f);          {  pop  ds          }
[6G3]  126:   Fehler:=Error div 256;
[X85]  127:   if Fehler<>Ø then Fehlermeldung;
[ZNØ]  128: end;
[ØØØ]  129:
[DW6]  13Ø: procedure SektorSchreiben(Error : word;
[413]  131:                           Lw,Skt,Az:integer;
[3W2]  132:                           var Buffer:feld);
[MUØ]  133: begin
[A95]  134:   inline($1e);          {  push ds          }
[1A6]  135:   inline($c5/$5e/$Ø4); {  lds  bx,[bp+Ø4] }
[UC6]  136:   inline($8b/$4e/$Ø8); {  mov  cx,[bp+Ø8] }
[8D6]  137:   inline($8b/$56/$Øa); {  mov  dx,[bp+Øa] }
[PB6]  138:   inline($8a/$46/$Øc); {  mov  al,[bp+Øc] }
[G75]  139:   inline($55);          {  push bp          }
[Ø85]  14Ø:   inline($cd/$26);      {  int  26          }
[RW4]  141:   inline($72/$Ø2);      {  jc   f           }
[1M5]  142:   inline($33/$cØ);      {  xor  ax,ax       }
[H45]  143:   inline($9d);          {f:popf             }
[R65]  144:   inline($5d);          {  pop  bp          }
[W76]  145:   inline($89/$46/$Øe); {  mov  [bp+Øe],ax }
[775]  146:   inline($1f);          {  pop  ds          }
[6G3]  147:   Fehler:=Error div 256;
[X85]  148:   if Fehler<>Ø then Fehlermeldung;
[ZNØ]  149: end;
[ØØØ]  15Ø:
[9P4]  151: procedure Initialisierung;
[RNØ]  152: var
[NØ3]  153:   Zaehler : integer;
[MUØ]  154: begin
[YR3]  155:   if (LastMode <> CO8Ø) and
[GA3]  156:      (LastMode <> BW8Ø) and
[2B6]  157:     (LastMode <> Mono) then TextMode(CO8Ø);
[3R3]  158:   if LastMode <> CO8Ø then
[WE5]  159:     TextAttr := White + Black * 16 else
[5L5]  16Ø:       TextAttr := Black + LightGray * 16;
[GH2]  161:   window(1,1,8Ø,25);
[DB1]  162:   ClrScr;
[FP3]  163:   for Zaehler:=Ø to 5ØØØ do
[4H3]  164:     Puffer[Zaehler]:=Ø;
[ZNØ]  165: end;
[ØØØ]  166:
[YH3]  167: procedure Titelbild;
[MUØ]  168: begin
[6M3]  169:   if LastMode = CO8Ø then
[395]  17Ø:     TextAttr := Blue + LightGray * 16;
[SZ1]  171:   gotoxy(25,1);
[YD6]  172:   writeln(chr(218),fillstring(3Ø,chr(196)),
[BE1]  173:          chr(191));
[WZ1]  174:   gotoxy(25,2);
[LG5]  175:   writeln(chr(179),' D I S K - D O K T O '+
[7V2]  176:          'R    I I ',chr(179));
[ØØ2]  177:   gotoxy(25,3);
[WD6]  178:   writeln(chr(192),fillstring(3Ø,chr(196)),
[DE1]  179:          chr(217));
[6M3]  18Ø:   if LastMode = CO8Ø then
[B95]  181:     TextAttr := Red + LightGray * 16;
[WØ2]  182:   gotoxy(27,9);
[DC6]  183:   writeln('(C)opyright 199Ø:  DOS Extra');
[D42]  184:   gotoxy(25,11);
[S56]  185:   writeln('Redaktionsbüro Everts & '+
[5U1]  186:          'Hagedorn');
[M42]  187:   gotoxy(28,12);
[6A7]  188:   writeln('Hans-Stießberger-Straße 3');
[R42]  189:   gotoxy(29,13);
[7S5]  19Ø:   writeln('8Ø13 Haar (bei München)');
[T42]  191:   gotoxy(29,15);
[7M5]  192:   writeln('Autor: Markus Zietlow');
[CØ2]  193:   gotoxy(6,25);
[6M3]  194:   if LastMode = CO8Ø then
[MU4]  195:     TextAttr := LightRed + Blink +
[7D2]  196:     LightGray * 16;
[LC6]  197:   write('Drücken Sie bitte eine Taste, '+
[D45]  198:        'um den Disketten-Editor zu '+
[SX1]  199:        'starten !');
[Ø32]  2ØØ:   HideCursor;
[Y44]  2Ø1:   repeat until KeyPressed;
[2B4]  2Ø2:   TastaturpufferLoeschen;
[6M3]  2Ø3:   if LastMode = CO8Ø then
[5L5]  2Ø4:     TextAttr := Black + LightGray * 16;
[DB1]  2Ø5:   ClrScr;
[ZNØ]  2Ø6: end;
[ØØØ]  2Ø7:
[EP6]  2Ø8: function Hexadezimal(Zahl : byte) : char;
[MUØ]  2Ø9: begin
[FK4]  21Ø:   if (Zahl<=Ø) or (Zahl>15) then
[5N2]  211:     Hexadezimal:='Ø'
[ZRØ]  212:   else
[WH4]  213:     if (Zahl>Ø) and (Zahl<=9) then
[495]  214:       Hexadezimal:=chr(ord('1')+Zahl-1)
[ZRØ]  215:     else
[TP5]  216:       Hexadezimal:=chr(ord('A')+Zahl-1Ø);
[ZNØ]  217: end;
[ØØØ]  218:
[DE6]  219: function Dezimal(Letter : char) : byte;
[MUØ]  22Ø: begin
[K74]  221:   if Letter in ['Ø'..'9'] then
[RS4]  222:     Dezimal:=ord(Letter)-ord('Ø')
[ZRØ]  223:   else
[X84]  224:     if Letter in ['A'..'F'] then
[G65]  225:       Dezimal:=ord(Letter)-ord('A')+1Ø
[ZRØ]  226:     else
[X84]  227:       if Letter in ['a'..'f'] then
[G65]  228:         Dezimal:=ord(Letter)-ord('a')+1Ø
[ZRØ]  229:       else
[KT2]  23Ø:         Dezimal:=Error;
[ZNØ]  231: end;
[ØØØ]  232:
[W84]  233: procedure LaufwerkHolen
[WK3]  234:           (var Datenfeld : Feld);
[RNØ]  235: var
[GB3]  236:   LaufwerkNeu : char;
[EU3]  237:   LaufwerkAlt : integer;
[MUØ]  238: begin
[GH2]  239:   window(1,1,8Ø,25);
[N42]  24Ø:   gotoxy(35,18);
```

```
[ZM1]  241:   write ('?');
[N42]  242:   gotoxy(35,18);
[P31]  243:   repeat
[FS3]  244:     LaufwerkNeu:=ReadKey;
[Y56]  245:   until LaufwerkNeu in ['A'..'Z','a'..'z'];
[3B4]  246:   if ord(LaufwerkNeu)>96 then
[3E6]  247:     LaufwerkNeu:=chr(ord(LaufwerkNeu)-32);
[V15]  248:   LaufwerkAlt:=Hauptlaufwerk;
[HZ5]  249:   Hauptlaufwerk:=ord(LaufwerkNeu)-65;
[577]  25Ø:   SektorLesen(fehler,Hauptlaufwerk,Sektor,
[922]  251:               1,Datenfeld);
[YN2]  252:   if fehler<>Ø then
[MUØ]  253:     begin
[MX4]  254:       Hauptlaufwerk:=LaufwerkAlt;
[4W5]  255:       SektorLesen(fehler,Hauptlaufwerk,
[U73]  256:                   Sektor,1,Datenfeld);
[ZNØ]  257:     end;
[ZNØ]  258: end;
[ØØØ]  259:
[Z87]  26Ø: procedure NeuerSektor(var Datenfeld:Feld);
[RNØ]  261: var
[W54]  262:   SektorNeu : string[1Ø];
[UV4]  263:   SektorAlt,Korrekt : integer;
[MUØ]  264: begin
[GH2]  265:   window(1,1,8Ø,25);
[N42]  266:   gotoxy(54,18);
[E92]  267:   write('?       ');
[N42]  268:   gotoxy(54,18);
[M13]  269:   readln(SektorNeu);
[N63]  27Ø:   SektorAlt:=Sektor;
[9B5]  271:   val(SektorNeu,Sektor,Korrekt);
[WY2]  272:   if Korrekt<>Ø then
[MUØ]  273:     begin
[KF1]  274:       Warnung;
[Ø73]  275:       Sektor:=SektorAlt;
[ZNØ]  276:     end;
[577]  277:   SektorLesen(fehler,Hauptlaufwerk,Sektor,
[922]  278:               1,Datenfeld);
[YN2]  279:   if fehler<>Ø then
[Ø73]  28Ø:     Sektor:=SektorAlt;
[ZNØ]  281: end;
[ØØØ]  282:
[4S6]  283: function HexFeld(X,Y : integer) : boolean;
[MUØ]  284: begin
[B15]  285:   if (X>=1) and (X div 3<16) and (Y<=17)
[FG4]  286:   and (Y>=1) and (X mod 3<>Ø) then
[UA2]  287:     HexFeld:=true
[ZRØ]  288:   else
[EE2]  289:     HexFeld:=false
[ZNØ]  29Ø: end;
[ØØØ]  291:
[FY6]  292: function ASCIIFeld(X,Y : integer):boolean;
[MUØ]  293: begin
[3G4]  294:   if (X>=5Ø) and (X<=65) and (Y<=17)
[MB2]  295:   and (Y>=1) then
[9P2]  296:     ASCIIFeld:=true
[ZRØ]  297:   else
[HV2]  298:     ASCIIFeld:=false;
[ZNØ]  299: end;
[ØØØ]  3ØØ:
[HT6]  3Ø1: procedure Statuszeile(Seite : integer);
[MUØ]  3Ø2: begin
[AT2]  3Ø3:   window(1,18,8Ø,19);
[DB1]  3Ø4:   ClrScr;
[5D2]  3Ø5:   write('Bytes ');
[6M3]  3Ø6:   if LastMode = CO8Ø then
[395]  3Ø7:     TextAttr := Blue + LightGray * 16;
[6D2]  3Ø8:   if Seite=Ø then
[132]  3Ø9:     write('  1-255')
[ZRØ]  31Ø:   else
[M82]  311:     write('256-512');
[6M3]  312:   if LastMode = CO8Ø then
[5L5]  313:     TextAttr := Black + LightGray * 16;
[A64]  314:   write('          Laufwerk: ');
[6M3]  315:   if LastMode = CO8Ø then
[395]  316:     TextAttr := Blue + LightGray * 16;
[MS4]  317:   write(chr(65+Hauptlaufwerk));
[6M3]  318:   if LastMode = CO8Ø then
[5L5]  319:     TextAttr := Black + LightGray * 16;
[XM3]  32Ø:   write('          Sektor: ');
[6M3]  321:   if LastMode = CO8Ø then
[395]  322:     TextAttr := Blue + LightGray * 16;
[KE2]  323:   write(Sektor);
[6M3]  324:   if LastMode = CO8Ø then
[5L5]  325:     TextAttr := Black + LightGray * 16;
[ZNØ]  326: end;
[ØØØ]  327:
[7U6]  328: procedure Ausgabe(Reihe,Spalte : integer;
[D82]  329:                   Value : byte);
[MUØ]  33Ø: begin
[7L3]  331:   gotoxy(Spalte,Reihe);
[BB5]  332:   write(Hexadezimal(value shr 4));
[WD5]  333:   write(Hexadezimal(value and $f));
[5Y4]  334:   gotoxy((Spalte div 3)+5Ø,Reihe);
[MG2]  335:   if value<32 then
[MUØ]  336:     begin
[3R3]  337:       if LastMode <> CO8Ø then
[XB4]  338:         TextAttr := White + Blink +
[XF5]  339:         Black * 16 else TextAttr := Brown +
[7D2]  34Ø:         LightGray * 16;
[XM1]  341:       write('.');
[3R3]  342:       if LastMode <> CO8Ø then
[WE5]  343:         TextAttr := White + Black * 16 else
[5L5]  344:         TextAttr := Black + LightGray * 16;
[4MØ]  345:     end
[ZRØ]  346:   else
[YZ2]  347:     write(chr(Value));
[ZNØ]  348: end;
[ØØØ]  349:
[YB6]  35Ø: procedure ZeigeSeite(Datenfeld : Feld;
[MT2]  351:                      Seite : integer);
[RNØ]  352: var
[UW3]  353:   Reihe,Spalte : integer;
[492]  354:   Inhalt : byte;
[MUØ]  355: begin
[EH2]  356:   window(1,1,8Ø,17);
[DB1]  357:   ClrScr;
[R83]  358:   for Reihe:=Ø to 15 do
[YF3]  359:     for Spalte:=Ø to 15 do
[MUØ]  36Ø:       begin
[KL5]  361:         Inhalt:=Datenfeld[Seite+Reihe*16+
[CG1]  362:                  Spalte];
[2N5]  363:         Ausgabe(Reihe+1,3*Spalte+1,Inhalt);
[ZNØ]  364:       end;
[ZNØ]  365: end;
[ØØØ]  366:
[CY4]  367: procedure Hexadezimalausgabe
[4E5]  368:           (var X,Y:integer;Parameter:word;
[UD3]  369:            var Datenfeld : Feld;
[MT2]  37Ø:            Seite : integer);
[RNØ]  371: var
[L74]  372:   Position,Wert : integer;
[MUØ]  373: begin
[Ø43]  374:   if HexFeld(X,Y) then
[MUØ]  375:     begin
[GR4]  376:       Position:=(X-1) div 3 + (Y-1)*16;
[XT4]  377:       Wert:=Dezimal(chr(Parameter
[UF1]  378:             and $ff));
[B93]  379:       if Wert<>Error then
[MUØ]  38Ø:         begin
[YG2]  381:           if X mod 3=1 then
[DY4]  382:             Datenfeld[Position+Seite]:=
[4R4]  383:             (Datenfeld[Position+Seite]
[XA2]  384:             and $f)+Wert*16
[ZRØ]  385:           else
[DY4]  386:             Datenfeld[Position+Seite]:=
[4R4]  387:             (Datenfeld[Position+Seite]
[T52]  388:             and $fØ)+Wert;
[554]  389:           Ausgabe(1+Position div 16,
[Y63]  39Ø:           (Position mod 16)*3+1,
[KW4]  391:           Datenfeld[Position+Seite]);
[GT1]  392:           if X=47 then
[MUØ]  393:             begin
[2T1]  394:               if Y<16 then
[MUØ]  395:                 begin
[TRØ]  396:                   X:=1;
[7F1]  397:                   inc (Y,1);
[ZNØ]  398:                 end;
[4MØ]  399:             end
[8U2]  4ØØ:           else if (X mod 3=2)
[EB2]  4Ø1:                   and (X<48) then
[N61]  4Ø2:                 inc(X,2)
[ZRØ]  4Ø3:               else
[7A1]  4Ø4:                 inc(X,1);
[ZNØ]  4Ø5:         end;
[4MØ]  4Ø6:     end
[ZRØ]  4Ø7:   else
[FF3]  4Ø8:     if ASCIIFeld(X,Y) then
[MUØ]  4Ø9:       begin
[NP3]  41Ø:         Position:=X-5Ø+(Y-1)*16;
[DY4]  411:         Datenfeld[Position+Seite]:=
[VT2]  412:         Parameter and $ff;
[554]  413:         Ausgabe(1+Position div 16,
[Y63]  414:         (Position mod 16)*3+1,
[KW4]  415:         Datenfeld[Position+Seite]);
[ET1]  416:         if X=65 then
[MUØ]  417:           begin
[2T1]  418:             if Y<16 then
[MUØ]  419:               begin
[SWØ]  42Ø:                 X:=5Ø;
[9A1]  421:                 inc(Y,1);
[ZNØ]  422:               end;
[4MØ]  423:           end
[ZRØ]  424:         else
[7A1]  425:           inc(X,1);
[ZNØ]  426:       end;
[ZNØ]  427: end;
[ØØØ]  428:
[NN6]  429: procedure ZeigeInhalt(Datenfeld : Feld;
[ØR3]  43Ø:                       Seite,Modus:integer);
[MUØ]  431: begin
[XD2]  432:   if Modus=2 then
[265]  433:     ZeigeSeite(Datenfeld,Seite*256);
```

Listing. Mit »diskdoc.pas« fühlen Sie Ihren Datenträgern auf den Zahn

```
[ZNØ]  434: end;
[ØØØ]  435:
[5K3]  436: procedure Kommandos;
[MUØ]  437: begin
[4S2]  438:   window(1,2Ø,8Ø,24);
[DB1]  439:   ClrScr;
[6M3]  44Ø:   if LastMode = CO8Ø then
[B95]  441:     TextAttr := Red + LightGray * 16;
[NØ5]  442:   writeln(FillString(31,chr(32)),
[AU3]  443:          'Sektorenübersicht');
[6M3]  444:   if LastMode = CO8Ø then
[5L5]  445:     TextAttr := Black + LightGray * 16;
[KØ5]  446:   write(FillString(8Ø,chr(196)));
[6M3]  447:   if LastMode = CO8Ø then
[2Ø6]  448:     TextAttr := Magenta + LightGray * 16;
[YV5]  449:   write('Lesen von:     F1) Bestimmtem '+
[ZB5]  45Ø:        'Sektor  F3) Nächstem Sektor  '+
[E24]  451:        'F4) Vorherigem Sektor');
[6N5]  452:   write('Funktionen:    F2) Laufwerk '+
[CD5]  453:        'wählen    F6) Sektor schreiben '+
[2X3]  454:        'F8) Boot-Record-Daten');
[4Ø3]  455:   write('                F9) '+
[WP4]  456:        'Partition-Table    FØ) Exit');
[6M3]  457:   if LastMode = CO8Ø then
[5L5]  458:     TextAttr := Black + LightGray * 16;
[ZNØ]  459: end;
[ØØØ]  46Ø:
[NA7]  461: procedure ZeigePartition(Datenfeld : Feld);
[6XØ]  462: const
[XZ1]  463:   Zeiger=$1be;
[RNØ]  464: var
[NØ3]  465:   Zaehler : integer;
[YF3]  466:   Register : Registers;
[MUØ]  467: begin
[AW2]  468:   Register.ax:=$2Ø1;
[ES2]  469:   Register.dx:=$8Ø;
[SM2]  47Ø:   Register.cx:=1;
[7R4]  471:   Register.es:=seg(Datenfeld);
[PR4]  472:   Register.bx:=ofs(Datenfeld);
[553]  473:   intr($13,Register);
[GH2]  474:   window(1,1,8Ø,25);
[DB1]  475:   ClrScr;
[6M3]  476:   if LastMode = CO8Ø then
[B95]  477:     TextAttr := Red + LightGray * 16;
[LØ5]  478:   writeln(FillString(3Ø,chr(32)),
[N84]  479:          'Festplatten-Partition');
[465]  48Ø:   writeln(FillString(8Ø,chr(196)));
[6M3]  481:   if LastMode = CO8Ø then
[5L5]  482:     TextAttr := Black + LightGray * 16;
[UE3]  483:   for Zaehler:=Ø to 3 do
[MUØ]  484:     begin
[2C4]  485:       gotoxy((Zaehler mod 2)*4Ø+1,
[LØ3]  486:              (Zaehler div 2)*8+6);
[CF5]  487:       if Datenfeld[Zeiger+Zaehler*16]
[6E1]  488:          <>$8Ø then
[MUØ]  489:         begin
[6M3]  49Ø:           if LastMode = CO8Ø then
[Y43]  491:             TextAttr := Black +
[7D2]  492:             LightGray * 16;
[4MØ]  493:         end
[ZRØ]  494:       else
[6M3]  495:         if LastMode = CO8Ø then
[FØ3]  496:           TextAttr := Blue +
[7D2]  497:           LightGray * 16;
[BC5]  498:       write('Partition ',Zaehler+1,': ');
[CF5]  499:       if Datenfeld[Zeiger+Zaehler*16]
[6E1]  5ØØ:          <>$8Ø then
[Ø52]  5Ø1:         write('Nicht')
[ZRØ]  5Ø2:       else
[ØZ1]  5Ø3:         write('Ist');
[CZ2]  5Ø4:       writeln(' aktiv');
[2C4]  5Ø5:       gotoxy((Zaehler mod 2)*4Ø+1,
[VØ3]  5Ø6:              (Zaehler div 2)*8+8);
[Y43]  5Ø7:       write('DOS-Part.: ');
[KG5]  5Ø8:       if Datenfeld[Zeiger+Zaehler*16+4]
[S31]  5Ø9:          =1 then
[7V4]  51Ø:         write('12-Bit File Allocation '+
[W51]  511:               'Table')
[ZRØ]  512:       else
[KG5]  513:         if Datenfeld[Zeiger+Zaehler*16+4]
[141]  514:            =4 then
[FV4]  515:           write('16-Bit File Allocation '+
[W51]  516:                 'Table')
[ZRØ]  517:         else
[WC2]  518:           write('Keine');
[2C4]  519:       gotoxy((Zaehler mod 2)*4Ø+1,
[ZØ3]  52Ø:              (Zaehler div 2)*8+9);
[974]  521:       writeln('Anfang:    Kopf ',
[VA5]  522:            Datenfeld[Zeiger+Zaehler*16+1],
[8N4]  523:            ' Sektor ',Datenfeld[Zeiger+
[9H3]  524:            Zaehler*16+2],' Spur ',
[EC5]  525:            Datenfeld[Zeiger+Zaehler*16+3]);
[2C4]  526:       gotoxy((Zaehler mod 2)*4Ø+1,
[L43]  527:              (Zaehler div 2)*8+1Ø);
[9Ø4]  528:       writeln('Ende:      Kopf ',
[3B5]  529:            Datenfeld[Zeiger+Zaehler*16+5],
[8N4]  53Ø:            ' Sektor ',Datenfeld[Zeiger+
[DH3]  531:            Zaehler*16+6],' Spur ',
[PC5]  532:            Datenfeld[Zeiger+Zaehler*16+7]);
[ZNØ]  533:       end;
[6M3]  534:     if LastMode = CO8Ø then
[MU4]  535:       TextAttr := LightRed + Blink +
[7D2]  536:                   LightGray * 16;
[2Ø2]  537:     gotoxy(8,22);
[HP6]  538:     writeln('Drücken Sie bitte eine Taste, '+
[7Z2]  539:             'um in den Editor '+
[DE3]  54Ø:             'zurückzukehren !');
[Ø32]  541:     HideCursor;
[Y44]  542:     repeat until KeyPressed;
[2B4]  543:     TastaturpufferLoeschen;
[6M3]  544:     if LastMode = CO8Ø then
[5L5]  545:       TextAttr := Black + LightGray * 16;
[ZNØ]  546: end;
[ØØØ]  547:
[WH6]  548: procedure BootRecord(Datenfeld : Feld);
[RNØ]  549: var
[NØ3]  55Ø:   Zaehler : integer;
[MUØ]  551: begin
[GH2]  552:   window(1,1,8Ø,25);
[DB1]  553:   ClrScr;
[6M3]  554:   if LastMode = CO8Ø then
[B95]  555:     TextAttr := Red + LightGray * 16;
[X66]  556:   writeln(FillString(3Ø,chr(32)),'Daten '+
[763]  557:          'des Boot-Records');
[465]  558:   writeln(FillString(8Ø,chr(196)));
[DA6]  559:   SektorLesen(Fehler,Hauptlaufwerk,Ø,1,
[XW1]  56Ø:              Datenfeld);
[6M3]  561:   if LastMode = CO8Ø then
[5L5]  562:     TextAttr := Black + LightGray * 16;
[XF1]  563:   writeln;
[9M6]  564:   write('Datenträger formatiert mit:    ');
[6M3]  565:   if LastMode = CO8Ø then
[395]  566:     TextAttr := Blue + LightGray * 16;
[YN3]  567:   for Zaehler:=3 to 1Ø do
[5A5]  568:     write(chr(Datenfeld[Zaehler]));
[6M3]  569:   if LastMode = CO8Ø then
[5L5]  57Ø:     TextAttr := Black + LightGray * 16;
[XF1]  571:   writeln;
[XF1]  572:   writeln;
[XF1]  573:   writeln;
[VW6]  574:   write('Datenträger identifiziert als: ');
[6M3]  575:   if LastMode = CO8Ø then
[395]  576:     TextAttr := Blue + LightGray * 16;
[5F3]  577:   case Datenfeld[21] of
[ØC4]  578:     $F8 : writeln('Festplatte');
[8R5]  579:     $F9 : writeln('Vierfachdichte oder '+
[EM1]  58Ø:                   '3,5 Zoll');
[T56]  581:     $FC : writeln('einseitig, 9 Sektoren '+
[VW1]  582:                   'pro Spur');
[S76]  583:     $FD : writeln('zweiseitig, 9 Sektoren'+
[NX1]  584:                   ' pro Spur');
[X56]  585:     $FE : writeln('einseitig, 8 Sektoren '+
[VW1]  586:                   'pro Spur');
[Z76]  587:     $FF : writeln('zweiseitig, 8 Sektoren'+
[NX1]  588:                   ' pro Spur');
[ZRØ]  589:     else
[V65]  59Ø:       writeln('unbekanntes Medium');
[ZNØ]  591:   end;
[6M3]  592:   if LastMode = CO8Ø then
[5L5]  593:     TextAttr := Black + LightGray * 16;
[2H5]  594:   write('Bytes pro Sektor:              ');
[6M3]  595:   if LastMode = CO8Ø then
[395]  596:     TextAttr := Blue + LightGray * 16;
[KV6]  597:   writeln(Datenfeld[11]+256*Datenfeld[12]);
[6M3]  598:   if LastMode = CO8Ø then
[5L5]  599:     TextAttr := Black + LightGray * 16;
[T16]  6ØØ:   write('Sektoren pro Cluster:          ');
[6M3]  6Ø1:   if LastMode = CO8Ø then
[395]  6Ø2:     TextAttr := Blue + LightGray * 16;
[924]  6Ø3:   writeln(Datenfeld[13]);
[6M3]  6Ø4:   if LastMode = CO8Ø then
[5L5]  6Ø5:     TextAttr := Black + LightGray * 16;
[646]  6Ø6:   write('Sektoren im Boot-Record:       ');
[6M3]  6Ø7:   if LastMode = CO8Ø then
[395]  6Ø8:     TextAttr := Blue + LightGray * 16;
[5V4]  6Ø9:   writeln(Datenfeld[14]+256*int
[M83]  61Ø:          (Datenfeld[15]):1:Ø);
[6M3]  611:   if LastMode = CO8Ø then
[5L5]  612:     TextAttr := Black + LightGray * 16;
[V15]  613:   write('Kopien der FAT:                ');
[6M3]  614:   if LastMode = CO8Ø then
[395]  615:     TextAttr := Blue + LightGray * 16;
[C24]  616:   writeln(Datenfeld[16]);
[6M3]  617:   if LastMode = CO8Ø then
[5L5]  618:     TextAttr := Black + LightGray * 16;
[UZ6]  619:   write('Max. Hauptverzeichniseinträge: ');
[6M3]  62Ø:   if LastMode = CO8Ø then
[395]  621:     TextAttr := Blue + LightGray * 16;
[8V4]  622:   writeln(Datenfeld[17]+256*int
[U83]  623:          (Datenfeld[18]):1:Ø);
[6M3]  624:   if LastMode = CO8Ø then
[5L5]  625:     TextAttr := Black + LightGray * 16;
[UB5]  626:   write('Sektorenanzahl:                ');
[6M3]  627:   if LastMode = CO8Ø then
```

```
[395]  628:     TextAttr := Blue + LightGray * 16;
[AV4]  629:   writeln(Datenfeld[19]+256*int
[B83]  630:           (Datenfeld[20]):1:0);
[6M3]  631:   if LastMode = CO80 then
[5L5]  632:     TextAttr := Black + LightGray * 16;
[9K5]  633:   write('Sektoren in der FAT:          ');
[6M3]  634:   if LastMode = CO80 then
[395]  635:     TextAttr := Blue + LightGray * 16;
[7V4]  636:   writeln(Datenfeld[22]+256*int
[H83]  637:           (Datenfeld[23]):1:0);
[6M3]  638:   if LastMode = CO80 then
[5L5]  639:     TextAttr := Black + LightGray * 16;
[ZP5]  640:   write('Sektoren pro Spur:            ');
[6M3]  641:   if LastMode = CO80 then
[395]  642:     TextAttr := Blue + LightGray * 16;
[9V4]  643:   writeln(Datenfeld[24]+256*int
[N83]  644:           (Datenfeld[25]):1:0);
[6M3]  645:   if LastMode = CO80 then
[5L5]  646:     TextAttr := Black + LightGray * 16;
[R45]  647:   write('Seitenanzahl:                 ');
[6M3]  648:   if LastMode = CO80 then
[395]  649:     TextAttr := Blue + LightGray * 16;
[BV4]  650:   writeln(Datenfeld[26]+256*int
[T83]  651:           (Datenfeld[27]):1:0);
[6M3]  652:   if LastMode = CO80 then
[5L5]  653:     TextAttr := Black + LightGray * 16;
[PV5]  654:   write('Reservierte Sektoren:         ');
[6M3]  655:   if LastMode = CO80 then
[395]  656:     TextAttr := Blue + LightGray * 16;
[DV4]  657:   writeln(Datenfeld[28]+256*int
[X83]  658:           (Datenfeld[29]):1:0);
[6M3]  659:   if LastMode = CO80 then
[5L5]  660:     TextAttr := Black + LightGray * 16;
[577]  661:   SektorLesen(Fehler,Hauptlaufwerk,Sektor,
[922]  662:              1,Datenfeld);
[6M3]  663:   if LastMode = CO80 then
[MU4]  664:     TextAttr := LightRed + Blink +
[7D2]  665:                 LightGray * 16;
[202]  666:   gotoxy(8,22);
[HP6]  667:   writeln('Drücken Sie bitte eine Taste, '+
[7Z2]  668:          'um in den Editor '+
[DE3]  669:          'zurückzukehren !');
[032]  670:   HideCursor;
[Y44]  671:   repeat until KeyPressed;
[2B4]  672:   TastaturpufferLoeschen;
[6M3]  673:   if LastMode = CO80 then
[5L5]  674:     TextAttr := Black + LightGray * 16;
[ZN0]  675: end;
[000]  676:
[B94]  677: procedure ZeigeSektoren
[WK3]  678:           (var Datenfeld : Feld);
[RN0]  679: var
[XX2]  680:   Schluss:boolean;
[Y94]  681:   Seite,X,Y,Modus:integer;
[EK3]  682:   Register:Registers;
[MU0]  683: begin
[1S2]  684:   Schluss:=false;
[SW0]  685:   X:=50;
[VR0]  686:   Y:=1;
[ZD1]  687:   Seite:=0;
[WE1]  688:   Modus:=2;
[577]  689:   SektorLesen(Fehler,Hauptlaufwerk,Sektor,
[922]  690:              1,Datenfeld);
[6T1]  691:   Kommandos;
[UE3]  692:   Statuszeile(Seite);
[WP4]  693:   ZeigeSeite(Datenfeld,Seite);
[YB3]  694:   while not Schluss do
[MU0]  695:     begin
[F02]  696:       gotoxy(X,Y);
[FM2]  697:       Register.ax:=0;
[853]  698:       intr($16,Register);
[FG4]  699:       if Register.ax and $ff=0 then
[793]  700:         case (Register.ax) of
[UN1]  701:           $3b00: begin
[254]  702:                    NeuerSektor(Datenfeld);
[6T1]  703:                    Kommandos;
[UE3]  704:                    Statuszeile(Seite);
[BV3]  705:                    ZeigeInhalt(Datenfeld,
[VZ0]  706:                                Seite,
[T71]  707:                                Modus);
[ZN0]  708:                  end;
[YN1]  709:           $3c00: begin
[LC2]  710:                    LaufwerkHolen
[8W1]  711:                    (Datenfeld);
[6T1]  712:                    Kommandos;
[UE3]  713:                    Statuszeile(Seite);
[BV3]  714:                    ZeigeInhalt(Datenfeld,
[VZ0]  715:                                Seite,
[T71]  716:                                Modus);
[ZN0]  717:                  end;
[2P1]  718:           $3d00: begin
[X82]  719:                    inc(Sektor,1);
[AE3]  720:                    SektorLesen(Fehler,
[PR3]  721:                    Hauptlaufwerk,Sektor,
[922]  722:                    1,Datenfeld);
[YN2]  723:                    if Fehler<>0 then
[S72]  724:                    dec(Sektor,1);
[UE3]  725:                    Statuszeile(Seite);
[BV3]  726:                    ZeigeInhalt(Datenfeld,
[VZ0]  727:                                Seite,
```

```
[T71]  728:                                Modus);
[ZN0]  729:                  end;
[GG3]  730:           $3e00: if Sektor>0 then
[MU0]  731:                    begin
[S72]  732:                      dec(Sektor,1);
[AE3]  733:                      SektorLesen(Fehler,
[PR3]  734:                      Hauptlaufwerk,Sektor,
[922]  735:                      1,Datenfeld);
[UE3]  736:                      Statuszeile(Seite);
[BV3]  737:                      ZeigeInhalt(Datenfeld,
[VZ0]  738:                                  Seite,
[T71]  739:                                  Modus);
[4M0]  740:                    end
[ZR0]  741:                  else
[KF1]  742:                    Warnung;
[NL1]  743:           $4000: begin
[554]  744:                    SektorSchreiben(Fehler,
[PR3]  745:                    Hauptlaufwerk,Sektor,
[922]  746:                    1,Datenfeld);
[YN2]  747:                    if Fehler<>0 then
[MU0]  748:                      begin
[UE3]  749:                        Statuszeile(Seite);
[942]  750:                        ZeigeInhalt
[YR2]  751:                        (Datenfeld,Seite,
[T71]  752:                        Modus);
[ZN0]  753:                      end;
[ZN0]  754:                  end;
[XL1]  755:           $4200: begin
[NP3]  756:                    BootRecord(Datenfeld);
[DB1]  757:                    ClrScr;
[6T1]  758:                    Kommandos;
[UE3]  759:                    Statuszeile(Seite);
[BV3]  760:                    ZeigeInhalt(Datenfeld,
[VZ0]  761:                                Seite,
[T71]  762:                                Modus);
[ZN0]  763:                  end;
[1M1]  764:           $4300: begin
[2L2]  765:                    ZeigePartition
[8W1]  766:                    (Datenfeld);
[DB1]  767:                    ClrScr;
[6T1]  768:                    Kommandos;
[UE3]  769:                    Statuszeile(Seite);
[BV3]  770:                    ZeigeInhalt(Datenfeld,
[VZ0]  771:                                Seite,
[T71]  772:                                Modus);
[ZN0]  773:                  end;
[T83]  774:           $4400: Schluss:=true;
[7H2]  775:           $4800: if Y>1 then
[491]  776:                    dec(y,1);
[BK2]  777:           $4b00: if X>1 then
[RS1]  778:                    if X=50 then
[MR0]  779:                      X:=47
[ZR0]  780:                    else
[DV1]  781:                      if (X mod 3=1)
[EB2]  782:                      and (X<48) then
[G51]  783:                        dec(X,2)
[ZR0]  784:                      else
[291]  785:                        dec(X,1);
[TR2]  786:           $4d00: if X<65 then
[GT1]  787:                    if X=47 then
[7R0]  788:                      X:=50
[ZR0]  789:                    else
[FV1]  790:                      if (X mod 3=2)
[EB2]  791:                      and (X<48) then
[N61]  792:                        inc(X,2)
[ZR0]  793:                      else
[7A1]  794:                        inc(X,1);
[9N2]  795:           $5000: if Y<16 then
[9A1]  796:                    inc(Y,1);
[ZN0]  797:           $5100,
[TM1]  798:           $4900: begin
[FM3]  799:                    Seite:=(Seite+1) mod 2;
[UE3]  800:                    Statuszeile(Seite);
[BV3]  801:                    ZeigeInhalt(Datenfeld,
[VZ0]  802:                                Seite,
[T71]  803:                                Modus);
[ZN0]  804:                  end;
[4M0]  805:         end
[ZR0]  806:       else
[XD2]  807:         if Modus=2 then
[993]  808:           Hexadezimalausgabe
[NG4]  809:           (X,Y,Register.ax,Datenfeld,
[5P1]  810:           Seite*256);
[ZN0]  811:     end;
[ZN0]  812: end;
[000]  813:
[4W4]  814: (* Anfang des Hauptprogramms *)
[000]  815:
[MU0]  816: begin
[HX2]  817:   Hauptlaufwerk:=0;
[ZN1]  818:   Sektor:=0;
[MW2]  819:   Initialisierung;
[ZS1]  820:   Titelbild;
[2V3]  821:   ZeigeSektoren(Puffer);
[GH2]  822:   window(1,1,80,25);
[DB1]  823:   ClrScr;
[KN0]  824: end.
```

Listing. Mit »diskdoc.pas« fühlen Sie Ihren Datenträgern auf den Zahn

Pascal von A bis Z

Pascal ist ein interessanter Sprachstandard, der sich im Laufe der Zeit zu einer modernen und vielfältigen Programmiersprache entwickelt hat. Ihr Erfolg basiert auf dem klar strukturierten Aufbau und den mannigfaltigen Einsatzmöglichkeiten. Das Pascal-Lexikon verschafft einen Überblick über die zugehörigen Fachtermini und die Bedeutung der grundlegenden Begriffe.

Da Turbo Pascal an dem Erfolg der Programmiersprache entscheidend mitgewirkt hat, findet dieser PC-Standard in den folgenden Erläuterungen gebührend Beachtung. Dabei gilt: Ist eine Begriffsbezeichnung kursiv gedruckt und mit einem vorangestellten Pfeil versehen (»→*Begriff*«), finden Sie eine genauere Erläuterung zu diesem Begriff an anderer Stelle in den alphabetisch sortierten Beschreibungen (Querverweis). Eine fettgedruckte Begriffsbezeichnung (**»Begriff«**) leitet eine genauere Erläuterung zu dem entsprechenden Begriff ein.

Abbruch: Der →*Compiler* →*Turbo Pascal* erlaubt Ihnen den Abbruch der Programmausführung durch das Drücken der Tastenkombination [Ctrl-Break]. In der Version 5.5 von →*Turbo Pascal* läßt sich die Prüfung auf die Tastenkombination bei Ein- und Ausgaben über Tastatur und Bildschirm mittels der →*Variablen* »CBreak« abschalten. Um die Ausführungsgeschwindigkeit zu erhöhen und den Programmablauf zu beeinflussen, verwendet man →*Compiler-Befehle*.

Alphanumerische Variablen: Eine alphanumerische →*Variable* ist ein Platzhalter, dessen Inhalt aus beliebigen Zeichen bestehen kann. Im Gegensatz zu →*numerischen Variablen* sind also auch Buchstaben und Sonderzeichen erlaubt. Den Inhalt einer alphanumerischen Variablen bezeichnet man auch als →*Zeichenkette* oder →*String*.

Anschluß: Ein- oder Ausgabegabekanal zu und von einem Computer für die Ansteuerung von peripheren Geräten. (→*Port*)

ANSI: Das ANSI (»American National Standard Institute«) ist ein amerikanisches Institut, das ähnlich dem DIN-Institut Industrienormen festlegt. Das ANSI hat beispielsweise den →*ASCII*-Zeichensatz und den Standardbefehlssatz von →*Pascal* festgelegt.

Anweisung: Folge von →*Kommandos* in Pascal, die den Computer veranlassen, bestimmte Operationen durchzuführen

Array: →*Feld*

ASCII: ASCII (»American Standard Code for Information Interchange«) ist ein 8-Bit-Zeichencode, der von den meisten Computern und Turbo-Pascal für den Austausch von Informationen benutzt wird. ASCII wurde vom →*ANSI* als Standardzeichensatz festgelegt.

Bedingte Ausführung: Die »If...Then ...Else..«-Anweisung veranlaßt Pascal, einen Programmteil nur dann auszuführen, wenn eine bestimmte Bedingung erfüllt ist. Die Anweisung

```
if a = 0 then b := 1 else b := 2;
```

setzt die Variable »b« nur dann auf den Wert 1, wenn die Variable »a« eine 0 enthält. Andernfalls (»else«) erhält »b« den Wert 2. Für die Überprüfung einer Bedingung können sowohl die logischen →*Vergleichsoperationen* als auch die Operatoren »=«, »<«, »>«, »<=« und »>=« genutzt werden.

Befehl: →*Kommando*

Begrenzungszeichen: Begrenzungszeichen werden benutzt, um den Anfang beziehungsweise das Ende einer Dateneinheit auf einem Speichermedium zu markieren. Üblicherweise verwendet man Kommata, Leerzeichen, Punkte und Semikolons als Begrenzungszeichen zur Trennung und Gliederung von Daten. In →*Pascal* hat sich der Einsatz des Return-Zeichens (→*ASCII*-Code 13) als Begrenzungszeichen bewährt.

BGI-Treiber: BGI steht als Abkürzung für »Borland Graphics Interface«, dies sind die verschiedenen Treiber des Grafikpakets. Mit BGI-Dateien bindet man beispielsweise Vektorzeichensätze in Turbo-Pascal-Programme ein oder initialisiert das Grafikpaket für eine spezielle Grafikkarte.

Bildschirmkarten: Im Laufe der Zeit sind viele Bildschirmkarten auf dem Markt erschienen (»MDA«, »Hercules«, »CGA«, »EGA« und »VGA«). In bezug auf die Grafikanpassung gibt es bei den verschiedenen Turbo-Pascal-Fassungen keine Probleme. Borland hat die aktuelle Version 5.5 an die bekannten Grafikkarten angepaßt. Auffallend ist, daß bei den → *BGI-Treibern* der Super-VGA-Standard keine Beachtung findet.

Boolesche Logik: Unter diesem Begriff faßt man alle logischen →*Vergleichsoperationen* der Programmiersprache →*Pascal* zusammen. Variablen mit dem →*Datentyp* Boolean belegen 1 Byte im Speicher und können nur zwei Werte annehmen: wahr (englisch »true«), das heißt, eine Be-

Compiler-Schalter und -Befehle

Kommando	Aufgabe
$A+/-	Ausrichten von Variablen im Speicher
$B+/-	Auswerten Boolescher Ausdrücke
$D+/-	Zusatzinformationen zur Fehlersuche erzeugen
$DEFINE *Symbolname*	Symbol für bedingte Compilierung definieren
$E+/-	Einbinden der gesamten 8087-Bibliothek
$ELSE	Sonst-Zweig einer bedingten Compilierung einleiten
$ENDIF	Bereichsende für bedingte Compilierung festlegen
$F+/-	Erzwingen von Far-Aufrufen
$I *Dateiname*	Einfügen der Datei in den Quelltext
$I+/-	automatisches Prüfen von Ein-/Ausgaben
$IF	Bedingung für bedingte Compilierung überprüfen
$IFDEF *Symbolname*	Abkürzung für »IF symbol DEFined«
$IFNDEF *Symbolname*	Abkürzung für »IF symbol Not DEFined«
$IFOPT *Compiler-Schalter*	Abkürzung für »IF OPTion«
$L *Dateiname*	angegebene Objektdatei einbinden
$L+/-	lokale Symbole aufnehmen
$M *Speicherbelegung*	Platz für Stapel und Heap-Bereich reservieren
$N+/-	numerische Operationen über Coprozessor leiten
$O *Unitname*	Unit als Overlay deklarieren
$O+/-	Overlay-Prüfung durchführen
$R+/-	Bereichswerte überprüfen
$S+/-	Stapelprüfung kontrollieren
$UNDEF *Symbolname*	Symbol in bedingter Compilierung löschen
$V+/-	Überprüfen der Übergabe von Zeichenketten

Tabelle 1. Die Compiler-Schalter und -Befehle gehören zu den bedeutenden Sprachwerkzeugen von Turbo Pascal

Typ	Wertebereich	Genauigkeit	Coprozessor
Double	$5{,}0 * 10^{-324}$ bis $1{,}7 * 10^{308}$	15 bis 16 Stellen	erforderlich
Extended	$1{,}9 * 10^{-4951}$ bis $1{,}1 * 10^{4932}$	19 bis 20 Stellen	erforderlich
Real	$2{,}9 * 10^{-39}$ bis $1{,}7 * 10^{38}$	11 bis 12 Stellen	nicht erforderlich
Single	$1{,}5 * 10^{-45}$ bis $3{,}4 * 10^{38}$	7 bis 8 Stellen	erforderlich

Tabelle 2. Real-Typen sind eine Teilmenge der reellen Zahlen, die Turbo Pascal im Fließkommaformat verarbeitet

Typ	Wertebereich	Aufbau
Byte	0 bis 255	8 Bit vorzeichenlos
Integer	-32768 bis 32767	16 Bit mit Vorzeichen
LongInt	-2147483648 bis 2147483647	32 Bit mit Vorzeichen
ShortInt	-128 bis +127	8 Bit mit Vorzeichen
Word	0 bis 65535	16 Bit vorzeichenlos

Tabelle 3. Turbo Pascal sieht den Einsatz von fünf Integer-Typen vor, die eine Untermenge der ganzen Zahlen aufnehmen

dingung trifft zu (entspricht Zahlenwert 1), und falsch (englisch »false«), das heißt, eine Bedingung trifft nicht zu (Zahlenwert 0).

Compiler: Ein Compiler wie →*Quick Pascal* oder →*Turbo Pascal* übersetzt einen Quelltext nach erfolgreicher Fehlerbeseitigung in einen maschinenspracheähnlichen Code (»Objektcode«) oder sogar direkt in Maschinensprache. Ein auf diese Weise übersetztes Programm erreicht wesentlich höhere Geschwindigkeiten als ein →*Interpreter*.

Compiler-Befehle: Ein Compiler-Befehl weist den Turbo-Pascal-Compiler an, während der Übersetzung des Programms in maschinennahem Code bestimmte Operationen auszuführen. Eine Aufstellung der Compiler-Befehle finden Sie in Tabelle 1.

Datentyp: Fast alle in einem Pascal-Programm enthaltenen Informationen sind in die grundlegenden Datentypen »Integer-Zahlen«, »Fließkommawerte«, »Text«, »Wahrheitswerte« und »Zeiger« eingeteilt. Die genauen Beschreibungen der Datentypen lassen sich unter den Stichworten →*Boolsche Logik*, →*Fließkommavariablen*, →*Integer-Variablen*, →*alphanumerische Variablen* und →*Zeiger* finden.

Feld: Ein Feld oder →*Array* ist eine strukturierte Sammlung von Daten, in der das Argument vor der Funktion steht. Ein Feld setzt sich aus mehreren →*Variablen* zusammen. Die Vorteile dieses →*Variablentyps* sind auffällig: Einerseits kann jedes Element des Feldes unter demselben Variablennamen angesprochen werden; der einzige Unterschied besteht im Indexwert. Andererseits sind Sie in der Lage, jedes Feldelement auf die Art und Weise zu bearbeiten wie eine Einzelvariable des gleichen Variablentyps. Listing 1 zeigt Ihnen ein praktisches Beispiel. Im Deklarationsteil teilen Sie dem Computer mit dem Array-Befehl mit, daß Sie unter dem Variablennamen »zahl« eine Gruppe von maximal 10 Werten zusammenfassen wollen. Der →*Compiler* reserviert Ihnen unter dem Namen »zahl« Speicherplatz für maximal zehn Werte. Die einzelnen Werte sind durchnumeriert, wobei die Nummern in eckigen Klammern nach dem Variablennamen angegeben werden müssen. Die Angabe der Nummer wird auch als →*Index* bezeichnet. Die Variable »zahl« betrachtet man als Feld.

Fließkommavariablen: Die Fließkommavariablen sind eine Teilmenge der →*numerischen Variablen*. Fließkommavariablen können mit rationalen Werten arbeiten. Unter →*Turbo Pascal* sind ihnen Zahlen mit den →*Variablentypen* »real«, »single«, »double« oder »extended« zugewiesen. Eine Aufstellung der Real-Typen mit den zugeordneten Wertebereichen finden Sie in Tabelle 2 (→*Variable*).

Index: Der Index bestimmt innerhalb eines →*Feldes* ein Feldelement. Dieses Element wird wie eine →*Variable* angesprochen.

Integer-Variablen: Die Integer-Variablen sind eine Teilmenge der →*numerischen Variablen*. Integer-Variablen können nur mit ganzen Zahlen arbeiten. Eine Aufstellung der Integer-Typen mit den zugeordneten Wertebereichen finden Sie in Tabelle 3 (→*Variable*).

Interpreter: Ein Interpreter übersetzt ein in einer Hochsprache geschriebenes Programm während der Ausführung Befehl für Befehl in Maschinensprache. Weil jedoch beim Ausführen eines →*Befehls* neben der Übersetzung und den gesamten Fehlerabfangroutinen auch noch die vom jeweiligen →*Kommando* benötigten Systemroutinen aufgerufen werden müssen, ist die erreichbare Verarbeitungsgeschwindigkeit in den meisten Fällen vergleichsweise gering. Ein Interpreter arbeitet wesentlich langsamer als ein →*Compiler*. Da in Pascal Definitions- und Programmteil getrennt sind und der Deklarationsteil alle Variablen enthalten muß, wäre eine Interpreter-Version wenig zweckmäßig.

Interrupt-Programmierung: Unter Turbo Pascal sind Sie in der Lage, die Abfrage gewisser Zustände wie beispielsweise der Tastatur in einen Interrupt einzubinden. Der Interrupt wird mehrere Male pro Sekunde aufgerufen. Dabei nimmt Turbo Pascal jedesmal die vorher definierte Abfrage vor. Abhängig vom Ergebnis führt das Programm ein →*Unterprogramm* aus und fährt dann mit der Programmausführung im Hauptprogramm fort. Gerade im PC-Bereich ist die Interrupt-Programmierung ein wichtiger Bestandteil bei der Programmerstellung, da spezielle Betriebssystemroutinen sehr einfach durch einen Interrupt-Aufruf aktiviert werden können. So lassen sich Programme relativ kompakt erstellen.

Kommando: Ein Turbo-Pascal-Kommando oder →*-Befehl* ist ein vordefiniertes Wort aus einer Folge von Zeichen, das den Computer veranlaßt, bestimmte Operationen wie beispielsweise die Ausgabe einer Zeichenkette durch »Write« durchzuführen. Jedes Kommando wird vom →*Compiler* bearbeitet und in ausführbaren Code übersetzt. Das →*ANSI* hat den Standardbefehlssatz von Pascal festgelegt, der in →*Turbo Pascal* und →*Quick Pascal* enthalten ist. Eine Aufstellung aller Turbo-Pascal-Kommandos finden Sie in Tabelle 4 (→*Anweisung*).

Listing: Im allgemeinen werden unter diesem Begriff gedruckte Programmzeilen in Zeitschriften und Büchern verstanden. Es ist demnach auch korrekt, ein → *Programm* als Listing zu bezeichnen.

Numerische Variablen: Eine numerische →*Variable* ist ein Platzhalter, dessen Inhalt ausschließlich aus Zahlen, einem Vorzeichen und einem Dezimalpunkt bestehen kann. Im Gegensatz zu →*alphanumerischen Variablen* sind weder Buchstaben noch Sonderzeichen erlaubt.

Overlay: Wenn Listings unübersichtlich erscheinen oder Programme so groß sind, daß sie nicht zusammen in den Speicher passen, dann verwendet man die Overlay-Technik. Overlay-Technik bedeutet, daß das aktuelle →*Programm* ein anderes Teilprogramm lädt und es automatisch startet. Auf diese Art und Weise wird die Programmausführung (durch das Nachladen der Module) geringfügig langsamer, jedoch sind so auch große

Programme unter Rechnern mit weniger Arbeitsspeicher lauffähig.

Pascal: Pascal ist eine der bekanntesten und weitverbreitetsten Programmiersprachen. Sie hat ihren Namen zu Ehren des französischen Philosophen und Mathematikers Blaise Pascal (1623 – 1662) erhalten, der als Neunzehnjähriger eine Rechenmaschine erfand. Im Jahre 1971 hat Niklas Wirth an der Eidgenössischen Technischen Hochschule in Zürich diese Programmiersprache entworfen, die als Weiterentwicklung von »Algol 60« angesehen wird. Pascal gilt als zeitsparendes Mittel zur Programmentwicklung und ist eine der am leichtesten zu erlernenden Programmiersprachen. Ihr guter Ruf beruht hauptsächlich auf der →*strukturierten Programmierung* und der Vielseitigkeit. Auf dem PC-Sektor hielt Pascal über den Borland-Standard →*Turbo Pascal* mit überwältigendem Erfolg Einzug in den Computerbereich. Im halbprofessionellen und Heimbereich hat die Programmiersprache durch die Realisation von modernen Konzepten und die konsequente Weiterentwicklung einen bemerkenswerten Erfolg erlebt.

Port: Ein- und Ausgabekanal eines Computers zum Anschluß einer Kommunikationsleitung (→*Anschluß*)

Programm: die Zusammenfassung von Anweisungen, die dazu dienen, komplexe Probleme zu lösen. Je nach Geschick bei der Programmierung und Aufgabenstellung können die Programme unterschiedlich groß und schnell werden.

Prozedur: Eine Prozedur ist das gleiche wie ein →*Unterprogramm*.

Quick Pascal: Der moderne Pascal-Dialekt »Quick Pascal« von Microsoft ist der härteste Konkurrent von →*Turbo Pascal*. Der →*Compiler* arbeitet mit einer optisch aufwendigen, menügeführten Benutzeroberfläche und kann mit Turbo Pascal konzipierte Programme bearbeiten. Der integrierte Editor ist wie unter Turbo Pascal Wordstar-kompatibel und überzeugt durch Schnelligkeit und seine vielfältigen Funktionen zur Bearbeitung des eingegebenen Quelltextes. Quick Pascal präsentiert zu jedem benötigten Themenkomplex ein Pulldown-Menü und überzeugt durch ausgewogene Fenstertechnik, Mausbedienung und eine ausgefeilte kontextsensitive Hilfe.

Rekursion: Eine Rekursion ist ein →*Unterprogramm*, das sich immer wieder selbst aufruft. Eine Rekursion bricht ab, wenn entweder eine Abbruchbedingung erfüllt ist oder der interne Rechenstapel für weitere Wiederholungen nicht mehr ausreicht. Das Beispiel in Listing 2 zeigt Ihnen den Aufbau einer Rekursion in Turbo Pascal. Das Programm berechnet die Fakultät einer Zahl (die Fakultät beispielsweise von 5 ist gleich 1*2*3*4*5 = 120). Aufgrund der zugrundeliegenden Programmstruktur eignet sich →*Pascal* besonders für die rekursive Programmierung. In den modernen Pascal-Varianten wie →*Quick Pascal* oder →*Turbo Pascal* gehört diese Unterprogrammvariante natürlich zum Standard. Weitere Informationen zu diesem interessanten Thema finden Sie in dem Artikel »Rekursive Algorithmen unter Turbo Pascal« in dieser Ausgabe.

Relative Datei: →*typisierte Datei*

Schleife: Eine Schleife ist gleichbedeutend mit einer Folge von Computerinstruktionen, die wiederholt werden, bis eine vorgegebene Bedingung erfüllt ist. Ein Beispiel für eine Schleife finden Sie in Listing 1. Das →*Kommando* »for« leitet eine Schleife ein. Die Anweisungen, die nach dem →*Befehl* stehen, werden ausgeführt, bis der Endwert der Schleife erreicht ist. Ist die Endbedingung erfüllt, dann fährt das Programm mit der Programmausführung hinter der Schleife fort.

Sequentielle Datei: Eine sequentielle Datei wird Zeichen für Zeichen ausgelesen.

Sprungbefehle: Sprungbefehle werden benutzt, um von der aktuellen Listing-Position in einen anderen Programmteil zu springen. Das →*Kommando* »goto« springt dabei ohne Rückkehr, während die Angabe eines Prozedurnamens nach Ausführung der Unterroutine ins Hauptprogramm zurückführt (→*Verzweigung*).

String: Das Wort »String« ist die englische Bezeichnung für eine →*Zeichenkette*.

Strukturierte Programmierung: Die strukturierte Programmierung ist eine Programmiertechnik, die den Erfolg von →*Pascal* entscheidend mitgetragen hat. Pascal erlaubt aufgrund der zur Verfügung gestellten Datenstrukturen, Algorithmen direkt in die Programmiersprache umzusetzen. Da man komplexe Programme aus kleineren modularen Einheiten zusammensetzen kann, erhält man leicht lesbare Quelltexte.

Textdatei: In einer Textdatei sind alle Datensätze mit einer variablen Länge und einer unterschiedlichen Struktur hintereinander auf dem Speichermedium abgelegt (→*Begrenzungszeichen*).

Turbo Pascal: Turbo Pascal ist auf PCs der bekannteste →*Compiler* für die Programmiersprache →*Pascal*. Turbo Pascal wartete als erstes Pascal mit einer fenster- und menüorientierten Benutzeroberfläche auf. Turbo Pascal unterstützt verschiedene →*Bildschirmkarten* und den Großteil der →*ANSI*-Pascal-Funktionen wie beispielsweise →*Rekursionen*. Ein interessantes Leistungsmerkmal von Turbo Pascal sind die zahlreichen Compiler-Befehle. Der Inline-Befehl ermöglicht das direkte Einfügen von Maschinensprache in den Programmcode. Bemerkenswert ist die Fähigkeit, den Quelltext eines Turbo-Pascal-Programms bedingt zu übersetzen. Turbo Pascal ist in der Version 5.5 abwärtskompatibel zu seinen Vorgängerversionen.

Der Standardbefehlssatz von Turbo Pascal

Abs	Addr	Append	ArcTan
Assign	BlockRead	BlockWrite	ChDir
Chr	Close	Concat	Copy
Cos	CSeg	Dec	Delete
Dispose	DSeg	EOF	EOLn
Erase	Exit	Exp	FilePos
FileSize	FillChar	Flush	Frac
FreeMem	GetDir	GetMem	Halt
Hi	Inc	Insert	Int
IOResult	Length	Ln	Lo
Mark	MaxAvail	MemAvail	MkDir
Move	New	Odd	Ofs
Ord	ParamCount	ParamStr	Pi
Pos	Pred	Ptr	Random
Randomize	Read	Readln	Release
Rename	Reset	Rewrite	RmDir
Round	RunError	Seek	SeekEof
SeekEOLn	Seg	SetTextBuf	Sin
SizeOf	SPtr	Sqr	Sqrt
SSeg	Str	Succ	Swap
Trunc	Truncate	UpCase	Val
Write	Writeln		

Tabelle 4. Alle Turbo-Pascal-Kommandos auf einen Blick. Die Befehle der Standard-Units sind nicht berücksichtigt.

Operator	Operation	Operandentyp	Ergebnistyp
and	logische Verknüpfung	Boolean	Boolean
not	logische Negation	Boolean	Boolean
or	logisches Oder	Boolean	Boolean
xor	logische Ambivalenz	Boolean	Boolean

Tabelle 5. Boolesche Operationen sind für Vergleiche und Bedingungen von elementarer Bedeutung

Typisierte Datei: In einer typisierten Datei sind alle Datensätze mit einer festen Länge und einer gleichen Struktur auf dem Speichermedium abgelegt. Der Zugriff erfolgt über eine Satznummer. Da alle Datensätze den gleichen Aufbau vorweisen, ist der Zugriff wahlfrei und somit schnell.

Unterprogramm: Es gibt viele Situationen, in denen ein Abschnitt eines →*Programms* mehrfach benutzt werden muß. In diesem Fall helfen →*Schleifen* nicht weiter. Für diese Zwecke gibt es Unterprogramme. Ein Unterprogramm wird durch die Angabe seines Namens aktiviert und kehrt nach Ausführung der in ihm enthaltenen →*Befehle* in den aufrufenden Programmteil zurück. Unterprogramme verbessern außerdem die Struktur eines Listings und machen ein Programm übersichtlicher. Ein Pascal-Programm besteht im Regelfall aus einem kleinen Hauptprogramm und vielen Unterprogrammen (Funktionen und Prozeduren), die vom Hauptprogramm je nach Bedarf aufgerufen werden. Dies hat den Vorteil, daß von der Eignung her unterschiedliche Programmteile auch im eigentlichen Listing getrennt erscheinen. Listing 2 zeigt Ihnen den Einsatz eines Unterprogramms in einer →*Rekursion*.

Variable: Variablen sind einheitliche Zeichengruppen, die als Platzhalter für bestimmte Werte in Programmen fungieren. Die Variablenarten legt man in einem Deklarationsteil fest, der den →*Variablentyp* angibt. Unter →*Turbo Pascal* ist die Namensgebung für Variablen Regeln unterworfen. Erstens müssen die Variablennamen mit einem Buchstaben oder dem Unterstrich (»_«) beginnen, bestehen nach Vereinbarung nur aus Buchstaben, Ziffern und Unterstrichen und dürfen keine nationalen Sonderzeichen wie beispielsweise Umlaute beinhalten. Zweitens unterscheidet Turbo Pascal nicht zwischen Groß- und Kleinschreibung und verbietet, reservierte Wörter als Variablennamen zu verwenden. Die ersten 63 Zeichen eines Variablennamens sind signifikant. Das heißt, Turbo Pascal betrachtet zwei Variablennamen als identisch, deren erste 63 Zeichen gleich sind. Das ist unabhängig davon, ob weitere unterschiedliche Zeichen folgen. Listing 2 demonstriert, wie einer Variablen ein Wert zugewiesen wird.

Variablentyp: Der Typ einer →*Variablen* bestimmt, welche Werte in ihr abgelegt werden können. Die Größe (Genauigkeit) von →*numerischen Variablen* und →*alphanumerischen Variablen* ist vom →*Compiler* begrenzt und von ihrer Deklaration abhängig (→*Integer*, →*Fließkommavariablen*).

Vergleichsoperation: Jede Vergleichsfunktion wird benutzt, um zwei Datenfelder zu vergleichen und bei Bedarf das Ergebnis dieses Vergleichs in einem Datenfeld abzulegen. Unter →*Turbo Pascal* gibt es mehrere Vergleichsoperationen, die Tabelle 5 zusammenfaßt. Die Funktion der Vergleichsoperationen wurde 1864 von dem englischen Mathematiker George Boole festgelegt (→*Boolesche Logik*).

Verzögerungsschleife: Eine Verzögerungsschleife ist eine →*Schleife*, die die Programmausführung verlangsamt. Die Anweisung

```
for zaehler:=1 to 10000 do;
```

beispielsweise wartet, bis die Variable »zaehler« den Wert 1000 erreicht und fährt erst dann mit der Programmausführung fort. Achten Sie bitte darauf, daß die Wartezeit computerabhängig ist. Auf einem AT ist die Ausführung schneller beendet als auf einem PC. Allerdings empfiehlt es sich, gerade bei Schleifen, die nur warten sollen, einen anderen Weg einzuschlagen. Dazu existiert im Pascal-Dialekt ein anderer Befehl, der rechnerunabhängig ist und nach der realen Zeit geht.

Verzweigung: Eine Verzweigung setzt die Programmausführung an der angegebenen Stelle fort. Zu diesem Zweck benutzt man →*Sprungbefehle*, die die Art des Programmaufrufs bestimmen (→*bedingte Ausführung*).

Zeichenkette: Eine Zeichenkette ist unter →*Turbo Pascal* eine Folge von beliebigen →*ASCII*-Codes (→*alphanumerische Variablen*). Dies können ganze Sätze, Zahlenkolonnen oder Buchstaben sein.

Zeiger: Ein Zeiger enthält im Gegensatz zu den restlichen →*Datentypen* von →*Pascal* keinen direkt zugeordneten Wert, sondern die Adresse eines Wertes, das heißt seine Position im Arbeitsspeicher. Diese Technik erlaubt komplexe und umfangreiche Datenstrukturen.

(Markus Zietlow/dm)

```
[VL3]    1: program Feldzugriff;
[ØØØ]    2:
[RNØ]    3: var
[L53]    4:   schleife : integer;
[EB5]    5:   zahl     : array [1..1Ø] of integer;
[ØØØ]    6:
[MUØ]    7: begin
[3T3]    8:   for schleife:=1 to 9 do
[MUØ]    9:   begin
[UW5]   1Ø:     write('Geben Sie bitte die ',schleife,
[762]   11:           '. Zahl ein: ');
[UY3]   12:     readln(zahl[schleife]);
[ZNØ]   13:   end;
[XF1]   14:   writeln;
[3T3]   15:   for schleife:=1 to 9 do
[BD4]   16:     writeln(schleife,'. Zahl: ',
[RT2]   17:             zahl[schleife]);
[KNØ]   18: end.
```

Listing 1. Das Assemblerprogramm »saaws.asm« erlaubt SAA-Blockbefehle bei Wordstar-kompatiblen Editoren

```
[M65]    1: program rekursive_fakultaet;
[ØØØ]    2:
[RNØ]    3: var
[G92]    4:   zahl:longint;
[ØØØ]    5:
[TW6]    6: function fakultaet(n : longint) : longint;
[MUØ]    7: begin
[7N1]    8:   if n=Ø then
[812]    9:     fakultaet:=1
[ZRØ]   1Ø:   else
[YE4]   11:     fakultaet:=n*fakultaet(n-1)
[ZNØ]   12: end;
[ØØØ]   13:
[MUØ]   14: begin
[336]   15:   writeln('Programm zur rekursiven '+
[VS4]   16:           'Berechnung der Fakultät '+
[X42]   17:           'einer Zahl');
[Y15]   18:   writeln('----------------------------'+
[GF3]   19:           '----------------------------');
[XF1]   2Ø:   writeln;
[Y26]   21:   write('Geben Sie bitte die Zahl ein: ');
[Ø32]   22:   readln(zahl);
[D96]   23:   writeln('Die Fakultät der Zahl ',zahl,
[YL2]   24:           ' (',zahl,'!) ist: ',
[Z43]   25:           fakultaet(zahl):8);
[XF1]   26:   writeln;
[KNØ]   27: end.
```

Listing 2. Das ausführbare Programm »saaws.com« zum Eingeben mit SCheck

Management statt Bürokratie

Die Speicherverwaltung von Turbo Pascal

Mit Turbo Pascal entwickeln Sie Programme, die in Form einer Exe-Datei vom Betriebssystem MS-DOS in den Arbeitsspeicher geladen und ausgeführt werden. Diese Dateien weisen Strukturen auf, die der Compiler selbst bestimmt. Die Kenntnis der Strukturen erlaubt Ihnen, leistungsfähige Programme zu schreiben und führt zu einem grundlegenden Verständnis des Turbo-Pascal-Compilers.

Um ein Programm in den Arbeitsspeicher zu laden, durchsucht das Betriebssystem MS-DOS zunächst die Reihe der Speicherblöcke, bis ein ausreichend großer Bereich gefunden ist. Dieser Speicherblock wird dann in zwei Teile aufgeteilt. Der erste Abschnitt umfaßt 256 Byte und wird PSP (»program segment prefix«, also »Programmsegmentvorspann«) genannt. Im PSP sind Informationen über die Parameter der Kommandozeile und einige nützliche interne Zeiger von MS-DOS abgelegt.

Direkt hinter dem PSP wird ein weiterer Speicherblock angelegt, der den Umgebungsinformationen Platz bietet. Umgebungs-Strings, die Sie mit dem Set-Kommando von MS-DOS erzeugen, finden hier ihren Platz. Hinter dem Umgebungsblock beginnt der Speicherblock für das eigentliche Programm. An diese Stelle lädt das Betriebssystem Exe- und Com-Dateien, das heißt ausführbare Programme.

Wurde das geladene Programm mit dem Turbo-Pascal-Compiler erzeugt, enthält es in der Regel eine Vielzahl von Segmenten. Die grundsätzliche Unterteilung dieser Programme ist in Bild 1 dargestellt. Der mittlere Teil der Abbildung führt Ihnen vor Augen, in welcher Reihenfolge Turbo Pascal einzelne Segmente anlegt. Rechts neben der Abbildung ist verzeichnet, welche Namen an die Segmente vergeben werden. Auf der linken Seite sind Turbo-Pascal-Funktionen aufgeführt, mit denen sich die Adressen der Speicherstrukturen eines Programms feststellen lassen.

Der PSP, den das Betriebssystem MS-DOS vor jedem Programm im Arbeitsspeicher anlegt, läßt sich unter Turbo Pascal direkt ansprechen. In der Systemvariablen »PrefixSeg« ist die Segmentadresse des »program segment prefix« gespeichert. Die Struktur eines PSP ist in der Tabelle wiedergegeben. Listing 1 zeigt Ihnen, wie Sie – ohne Zugriff auf die Standardfunktion »ParamCount« – über die Speicherstelle 80hex des PSP ermitteln können, ob einem Programm Parameter übergeben worden sind.

Interessant ist der Wort-Eintrag im Feld 2Chex des »program segment prefix«. Hier speichert MS-DOS die Segmentadresse der Umgebung, die automatisch hinter dem PSP und vor dem eigentlichen Programm angelegt wird. Dabei handelt es sich um eine Kopie der globalen

MS-DOS-Umgebung. Der Umgebungsblock liefert eine Reihe von einzelnen Strings der Form

```
Variable=Wert
```

Jeder String wird durch ein Nullbyte abgeschlossen. Das Ende des gesamten Blocks macht eine weitere Null kenntlich. Listing 2 zeigt Ihnen, wie Sie über das PSP die Adresse des Umgebungsblocks ermitteln. Alle aktuellen Umgebungsvariablen werden dabei auf dem Bildschirm dargestellt. Das Programm erfüllt somit die gleiche Funktion, die auch der Set-Befehl unter MS-DOS ausführt, sofern er ohne Parameter aufgerufen wird.

Turbo Pascal legt für jede genutzte Unit ein eigenes Codesegment an

Ein möglicher Zugriff auf die Umgebungsvariablen erweist sich gerade auch in eigenen Programmen als nützlich, beispielsweise um spezielle Optionen zu aktivieren. Für ein Programm, das den mathematischen Coprozessor 8087 optional unterstützt, läßt sich zum Beispiel festlegen, daß die Unterstützung nur dann stattfindet, wenn im Umgebungsblock eine Variable »8087« auf den Wert »true« gesetzt ist.
Der Programmcode eines Turbo-Pascal-Programms folgt direkt nach dem Umgebungsblock. Die zugehörige Segmentadresse läßt sich über die Systemvariable »CSeg« abfragen. Der Programmcode umfaßt die Laufzeitbibliothek, den Maschinencode der eingebundenen Units und den Code des eigentlichen Hauptprogramms. Für jede Unit legt Turbo Pascal ein eigenes Segment an. Dieses darf bis zu 64 KByte Code enthalten. Das gilt auch für Standard-Units wie »Crt« oder »Dos«. Die Laufzeitbibliothek ist in der Unit »System« untergebracht. Sie wird automatisch in jedes Programm eingebunden und belegt ebenfalls ein eigenes Segment. Alle verwendeten Units eines Programms werden zwischen dem Codesegment der System-Unit und dem Codesegment des Hauptprogramms untergebracht. Dabei fügt Turbo Pascal die Units in der Reihenfolge ihrer Deklaration mit der Uses-Anweisung in den Programmcode ein. Verwendet ein Programm beispielsweise die Standard-Units »Crt« und »Printer« sowie eine selbstdefinierte Unit »Maus«, so arrangiert Turbo Pascal diese Units in der Reihenfolge ihrer Deklaration. Die Namen der Units vergibt der Compiler als Segmentnamen. Die Uses-Anweisung lautet beispielsweise:

```
Uses Crt,Printer,Maus;
```

Daraus ergibt sich die in Bild 2 dargestellte Belegung der einzelnen Codesegmente. Diese Codesegmente, also der Speicherbereich unterhalb der Datensegmente, sind in der von MS-DOS aufgerufenen Exe-Datei gespeichert. Alle Segmente oberhalb dieses Speicherblocks werden dagegen erst nach dem Laden des Turbo-Pascal-Programms angelegt.
Das eigentliche Datensegment ist im Arbeitsspeicher oberhalb der Codesegmente angesiedelt. Seine Segmentadresse läßt sich über die Systemvariable »DSeg« bestimmen. Das Datensegment enthält zwei Bereiche: Im ersten werden die typisierten Konstanten gespeichert, während der zweite für alle globalen Variablen zuständig ist. Wo aber finden die lokalen Variablen der Units und der Prozeduren sowie die Funktionen des Hauptprogramms ihren Platz? Auf den ersten Blick könnte man vermuten, daß Turbo Pascal für lokale Variablen zusätzliche Datenbereiche anlegt. Das ist aber nicht der Fall. Bevor eine Funktion oder Prozedur aufgerufen wird, legt der Compiler die lokalen Variablen auf dem Stack (Stapelspeicher) ab. Im Unterschied zu globalen Variablen belegen lokale Variablen daher nicht beständig Platz im Arbeitsspeicher.
Typisierte Konstanten und Variablen sind im gleichen Speichersegment untergebracht, weil es sich bei typisierten Konstanten im Prinzip nur um Variablen handelt, die einen Initialisierungswert mitbringen. Neben diesen beiden Arten von Daten belegen unter Umständen auch »echte« Konstanten Platz im Datensegment. Das gilt beispielsweise für Strings. Da Turbo Pascal Strings über Zeiger adressiert (dazu später mehr), muß die eigentliche String-Information irgendwo im Speicher stehen. Auch hier benutzt Turbo Pascal das Datensegment.
Oberhalb des Datensegments liegt der Overlay-Puffer, der erst bei Turbo Pascal ab Version 5.0 vorhanden ist. Er dient zur Verwaltung von sich überlappenden Speichersegmenten. Ihm folgt der Stack-Bereich. Auf dem Stack speichert Turbo Pascal alle lokalen Variablen und die Rücksprungadressen der Prozeduren oder Funktionen, die ihrerseits ein Unterprogramm aufrufen. Außerdem erfolgt die Übergabe aller Parameter von Funktionen und Prozeduren über den Stapelspeicher. Der Beitrag »Die Schnittstelle zwischen Turbo Pascal und Maschinensprache« in diesem Sonderheft widmet sich diesem Thema eingehend.

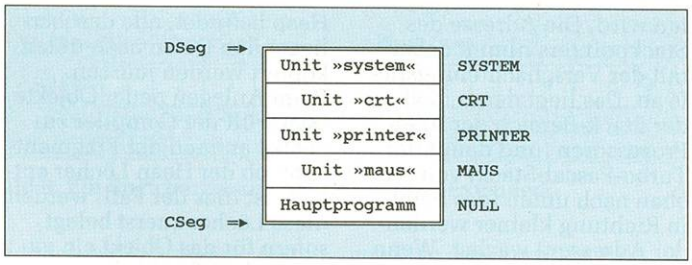


Bild 2. Die Belegung der Codesegmente

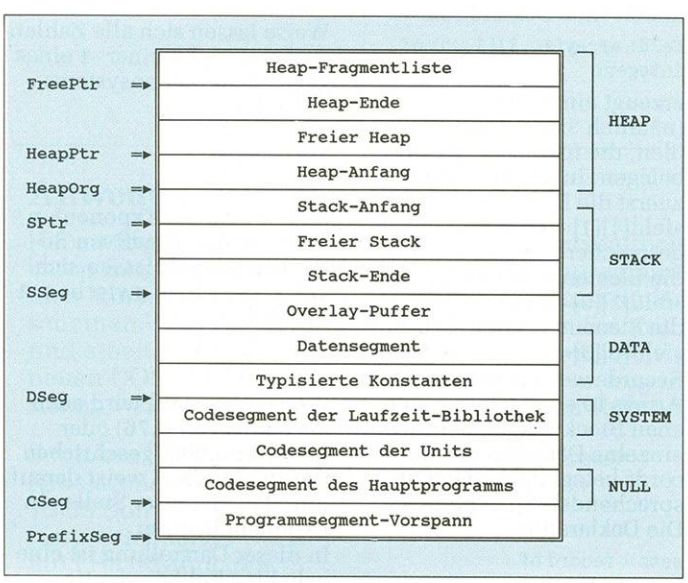


Bild 1. Die grundsätzliche Speicheraufteilung von Turbo-Pascal-Programmen

Der Stapelspeicher läßt sich mit Hilfe zweier Systemvariablen kontrollieren

Der Zustand des Stacks läßt sich über zwei Systemvariablen abfragen. Der Zeiger »SSeg« enthält die Segmentadresse des Stack-Bereichs, der Zeiger »SPtr« hält fest, auf welches Element der Stapelzeiger zur Zeit weist. Das Listing 3 zeigt die Funktionsweise dieser Anweisungen. Es definiert drei Prozeduren, die ineinander verschachtelt sind. Das Hauptprogramm ruft Prozedur »eins« auf, diese aktiviert Prozedur »zwei«, die wiederum in Prozedur »drei« als Unterprogramm verzweigt. Über die globale Prozedur »zeigstapel« wird in jeder Prozedur die aktuelle Position des Stackpointers (Stapelanzeiger) ausgegeben. Wenn Sie das Programm starten, werden Sie zunächst feststellen, daß sich die Adresse des Stackpointers nach jedem Prozeduraufruf geändert hat. Der Grund dafür ist, daß auf dem Stack Parameter übergeben und die Rücksprungadresse zur aufrufenden Prozedur festgehal-

ten wird. Die Adresse des Stackpointers nimmt dabei mit der Verschachtelungstiefe ab. Das liegt daran, daß der Stack-Bereich der Intel-Prozessoren (und damit der Turbo-Pascal-Stack) von oben nach unten (das heißt in Richtung kleiner werdender Adressen) wächst. Wenn Sie sich die Mühe machen, die Differenzen der Werte für den Stackpointer in den unterschiedlichen Ebenen zu zählen, werden Sie feststellen, daß sich immer ein Wert von 8 einstellt. Eine Ausnahme macht lediglich der Aufruf der ersten Prozedur aus dem Hauptprogramm. Da dem ersten Unterprogramm keine Parameter übergeben werden, beansprucht die Aktivierung von Prozedur »eins« weniger Platz auf dem Stack als die anderen Unterprogrammaufrufe.
Oberhalb des Stapelspeichers ist der Heap-Bereich angeordnet. Auf dem Heap legt Turbo Pascal alle dynamischen Datenstrukturen ab. Diese Datenstrukturen lassen sich mit den Kommandos »New« anlegen und mit »Dispose« wieder freigeben. Die Startadresse des Heaps wird in der Systemvariablen »HeapOrg« gespeichert. Der Heap wird ähnlich genutzt wie der Stapelspeicher, jedoch mit einem wichtigen Unterschied: Der Heap-Bereich wächst von unten nach oben, das heißt, bei fortlaufender Neubelegung werden zunehmend größere Speicheradressen verwendet. Die Systemvariable »HeapPtr« legt die momentane Obergrenze des Heaps fest. Dieser kann den gesamten Speicherbereich oberhalb des Stack-Segments bis zum Ende des freien Arbeitsspeichers nutzen. Allerdings »wächst« dem Heap vom Ende des verfügbaren Speichers eine weitere Struktur entgegen: die Fragmentliste.
In dieser Tabelle werden »Löcher« im belegten Heap-Bereich aufgelistet. Dadurch läßt sich vermeiden, daß bei der Freigabe eines Objektes, das sich mitten im belegten Heap befindet, alle darüberliegenden Daten nach unten kopiert werden müssen.
Beim Anlegen neuer Objekte überprüft der Compiler zunächst anhand der Fragmentliste, ob der Heap Löcher enthält. Ist dies der Fall, werden diese Löcher zuerst belegt, sofern für das Objekt ein genügend großes Loch vorhanden ist. Der Beginn der Fragmentliste läßt sich mit der Systemvariablen »FreePtr« lokalisieren. Die Differenz

```
FreePtr-HeapPtr
```

ergibt also den noch verfügbaren Platz auf dem Heap.

Mit zwei Funktionen ermitteln Sie die vollständige Adresse von Variablen

Interessante Anwendungen erlaubt die direkte Manipulation von Daten im Datensegment oder auf dem Heap. Für diesen Zweck stellt Turbo Pascal zwei Funktionen zur Verfügung, mit denen Sie die Adresse einer Variablen im Speicher ermitteln. Die Seg-Anweisung stellt die Segmentadresse, die Ofs-Funktion die Offset-Adresse einer Variablen fest. Über das Mem-Kommando sind Sie dann in der Lage, den Inhalt der Variablen zu lesen oder zu verändern. Listing 4 demonstriert diese Möglichkeit an einem Beispiel. Die Konstante »festwert« manipulieren Sie direkt im Speicher, indem Sie zuerst über die Anweisungen »Ofs« und »Seg« ihre Adresse ermitteln und dann mit Hilfe des Mem-Befehls den Wert der Konstanten verändern.
Um Variablen in eigenen Programmen direkt manipulieren zu können, ist es wichtig, das interne Speicherformat von Turbo Pascal bei der Verwendung von Variablen zu kennen. Bytevariablen legt der Compiler in Form eines Bytes im Arbeitsspeicher ab. Das gleiche gilt für den Datentyp »Char«. Hier schreibt Turbo Pascal den ASCII-Wert des Zeichens in den Speicher. Boolean-Variablen belegen ebenfalls ein Byte. Ein Wert von 0 steht für »false«, ein Wert von 1 für »true«. Integer-Variablen werden als ein Wort (2 Byte) gespeichert; das höchste Bit dieses Wortes stellt das Vorzeichen dar. Ein gesetztes Bit steht für einen negativen, ein gelöschtes für einen positiven Wert. Word-Variablen belegen ein Wort, das heißt ebenfalls 2 Byte im Arbeitsspeicher. Bei der String-Verwaltung werden die einzelnen Buchstaben an aufeinanderfolgenden Speicheradressen abgelegt. Als erstes Byte wird die Länge des Strings gespeichert. Wenn Sie daher auf den Datenbereich eines Strings zeigen wollen, müssen Sie zu der vom Ofs-Befehl ermittelten Offset-Adresse den Wert 1 addieren, um das Längenbyte zu überbrücken.
Arrays legt der Turbo-Pascal-Compiler als Speicherblöcke ohne Löcher an. Jedes Element eines Arrays belegt die durch seinen Datentyp festgelegte Anzahl an Bytes. Das Datenfeld beginnt mit dem ersten Index. Bei mehrdimensionalen Arrays folgen die Elemente der letzten Dimension unmittelbar aufeinander. Eine Deklaration

```
feld: array[1..3][1..5] of
integer;
```

erzeugt ein Feld von 15 (nämlich 3x5) Integer-Variablen, die insgesamt 30 Byte belegen. In diesem Feld sind zuerst die Elemente »feld[1][1]« bis »feld[1][5]« gespeichert, dahinter folgen die Elemente »feld[2][1]« bis »feld[2][5]« und schließlich die Elemente »feld[3][1]« bis »feld[3][5]«.
Records werden ebenso wie Arrays in einem kontinuierlichen Block gespeichert. Jeder einzelne Datentyp des Records belegt dabei den entsprechenden Speicherplatz. Die Deklaration

```
satz = record of
anzahl: integer;
zeichen: char;
end;
```

belegt daher 3 Byte Speicherplatz für jede Variable, der der Datentyp »satz« zugewiesen wurde. Die Integer-Variable belegt die ersten beiden Bytes. Der Datentyp »Char« beansprucht das dritte Byte des Records.
Zeiger belegen unter Turbo Pascal grundsätzlich zwei Worte, wobei das erste Wort die Offset- und das zweite Wort die Segmentadresse des Zeigers repräsentiert.

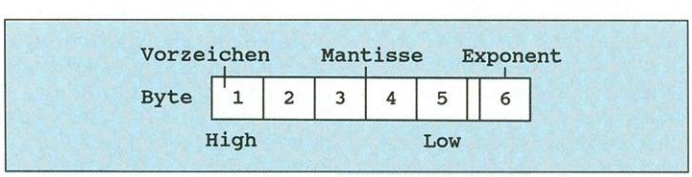


Bild 3. Ein Speicherauszug des Datentyps »Real«

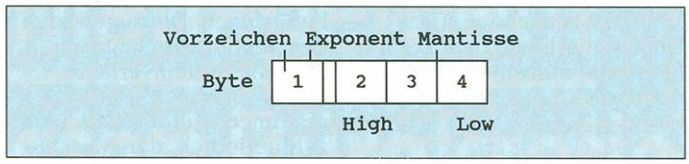


Bild 4. Ein Speicherauszug des Datentyps »Single«

Real-Zahlen speichert Turbo Pascal im »Fließkommaformat«

Etwas komplizierter ist das interne Format von Real-Zahlen. Der Trick zur Speicherung dieser Zahlen liegt im sogenannten »Fließkommaformat«. Das ist eine Methode, die beliebige Dezimalzahlen auf pfiffige Weise in wenigen Informationseinheiten (Bits) unterbringt.
Vielleicht ist Ihnen die wissenschaftliche Notation beim Umgang mit Zahlen bekannt. Dabei wird eine gegebene Zahl zerlegt, indem man sie durch die höchste enthaltene Potenz der Basis des Zahlensystems dividiert. Auf diese Weise lassen sich alle Zahlen größer 1 und kleiner -1 eines gegebenen Zahlensystems darstellen:

$400 = 4 * 10^2$
$33 = 3.3 * 10^1$
$-7340 = -7,34 * 10^3$

Indem man die Exponenten auch auf den negativen Bereich ausdehnt, lassen sich auch die Zahlen zwischen -1 und 1 darstellen:

$0,76 = 7,6 * 10^{-1}$
$0,003 = 3 * 10^{-3}$

Eine solche Zahl wird auch als 7,6E-1 (für 0,76) oder 3E-3 (für 0,003) geschrieben. Der Buchstabe E weist darauf hin, daß an dieser Stelle ein Exponent beginnt.
In dieser Darstellung ist eine beliebige Zahl eindeutig durch die sogenannte »Mantisse« oder den »Signifikan-

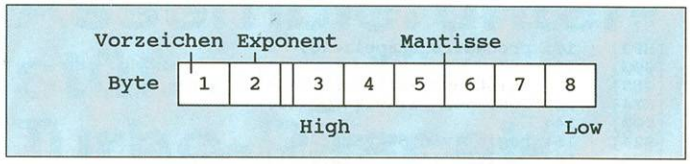


Bild 5. Ein Speicherauszug des Datentyps »Double«

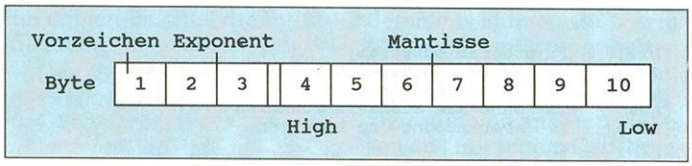


Bild 6. Ein Speicherauszug des Datentyps »Extended«

ten« (das ist die Zahl vor dem E) und den Exponenten der 10 gekennzeichnet (sofern die 10 als Basis des Zahlensystems bekannt ist). Bei einer gegebenen Mantisse von 3 und einem Exponenten von -3 ist also klar, daß es sich um die Zahl 0,003 handelt.

In dieser sogenannten »normalisierten Darstellung« sind Rechenoperationen sehr einfach durchzuführen.
Die Multiplikation von -0,07 mit 0,0053 wird zur Multiplikation von -7E-2 mit 5,3E-3. Sie multiplizieren hier einfach die Mantissen und addieren die Exponenten. Als Resultat ergibt sich -37,1E-5 oder – in normalisierter Darstellung – -3,71E-6, also -0,00000371. Nun werden in einem Computer aber Zahlen nicht im Dezimalformat, sondern vielmehr in binärer Darstellung gespeichert.
Anstelle von Dezimalzahlen liegen Mantisse und Exponent im Binärformat vor. Dabei bezieht sich der Exponent nicht mehr auf das Dezimal-, sondern auf das Binärsystem. So läßt sich die binäre Fließkommazahl 1,011E1 nach folgender Methode in das Dezimalsystem umwandeln:

$$\begin{aligned} & 1 * 2^{1} = 2 \\ + & 0 * 2^{0} = 0 \\ + & 1 * 2^{-1} = 0{,}5 \\ + & 1 * 2^{-2} = 0{,}25 \\ \hline & = 2{,}75 \end{aligned}$$

Wie sieht es aber in diesem System mit der Speicherung von negativen Werten oder von Zahlen, die kleiner als 1 sind, aus? Für den ersten Fall wird die Mantisse negativ, für den zweiten Fall erhält der Exponent einen negativen Wert.
Die einfachste Methode zur Darstellung negativer Binärzahlen liegt in einem separaten Bit, dessen Zustand darüber Aufschluß gibt, ob eine negative oder positive Zahl vorliegt. Sowohl für den Exponenten wie auch für die Mantisse wird dann ein zusätzliches Bit reserviert. Im Gegensatz zu anderen Verfahren (zum Beispiel dem Zweierkomplement) kommt diese »Vorzeichen-Betrags-Darstellung« ohne großes Umrechnen zurecht, allerdings bleibt dafür ein Bit bei der Hälfte der Zahlen unbenutzt.
Die Genauigkeit einer Fließkommazahl hängt davon ab, wie viele Bits dem Exponenten und der Mantisse zur Verfügung stehen.
In Pascal stellt der Datentyp »Real« 40 Bit für die Mantisse und 8 Bit für den Exponenten zur Verfügung. Single-Variablen belegen 24 Bit für die Mantisse und 8 Bit für den Exponenten, Variablen vom Typ »Double« 52 Bit für die Mantisse und 12 Bit für den Exponenten. Extended-Variablen schlagen mit 64 Bit für die Mantisse und 16 Bit für den Exponenten zu Buche. In diese Angaben ist jeweils das Vorzeichenbit mit einbezogen. Die Bilder 3 bis 6 geben die Reihenfolge der Bits der Fließkommadatentypen wieder.

(Martin Althaus/zi)

Offset (hex)	Länge	Bedeutung
00	Wort	Code für »int 20h«
02	Wort	Segmentadresse des Speicherendes
04	Byte	reserviert
05	Byte	Code für »call far«
06	DWort	Adresse des Interrupt 21hex
0A	DWort	Adresse der Terminierungsroutine
0E	DWort	Adresse des Interrupt 23hex
12	DWort	Adresse des Interrupt 24hex
16 bis 2B	Byte	reserviert
2C	Wort	Segmentadresse des Umgebungsblocks
2E bis 4F	Byte	reserviert
50	Wort	Code für »int 21h«
52	Byte	Code für »retf«
53	Wort	reserviert
55 bis 7F	Byte	zwei FCBs
80	Byte	Länge der Kommandozeilenparameter
81 bis FF	Byte	Kommandozeilenparameter

Tabelle. Die Struktur des »program segment prefix« (PSP)

Anwendungen gesucht

Haben Sie eine schnelle und komfortable Textverarbeitung geschrieben? Oder eine Datenbank mit frei wählbaren Eingabemasken? Eventuell ein Zeichenprogramm? Wenn Ihr Programm mit professionellen Programmen konkurrieren kann, ist es mit Sicherheit zu umfangreich, um als Listing in der DOS International zu erscheinen. Zögern Sie deshalb nicht, schicken Sie uns unverbindlich die aktuelle Version Ihres Programms und einen Ausdruck der Dokumentation. Eventuell wird Ihr Programm der Verkaufsschlager im Rahmen der neuen DOS-Software-Schiene. Falls Sie sich vorweg informieren möchten: Zwischen 15.00 und 17.00 Uhr stehen wir Ihnen an allen Werktagen unter der Rufnummer (089) 4 61 00 10 zur Verfügung.

Schicken Sie Ihre Unterlagen an folgende Adresse:
Redaktionsbüro Everts & Hagedorn
»DOS Software«
Hans-Stießberger-Str. 3
8013 Haar/München

```
[DZ2]     1: program chkparam;
[296]     2: {Prüft, ob Parameter an ein Programm
[UT3]     3:  übergeben worden sind,
[AG4]     4:  ohne die Standard-Funktion
[MN4]     5:  »ParamCount« zu benutzen
[ØØØ]     6:
[M34]     7:  Autor: Martin Althaus 2/9Ø
[ØØØ]     8:
[4Ø4]     9:  (C) 199Ø Redaktionsbüro
[WS2]    1Ø:           Everts & Hagedorn
[U7Ø]    11: }
[6XØ]    12: const
[MW2]    13:  zaehler: byte=$8Ø;
[365]    14:  {Offset Zeilenlänge im PSP}
[ØØØ]    15:
[RNØ]    16: var
[W52]    17:  laenge: byte;
[ØØØ]    18:
[MUØ]    19: begin
[Y13]    2Ø:  writeln; writeln;
[NN5]    21:  laenge:=mem[PrefixSeg:zaehler];
[EV3]    22:  {lese Länge aus PSP }
[3G2]    23:  if laenge=Ø then
[315]    24:   writeln('Keine Parameter wur',
[RS2]    25:   'den übergeben!')
[ZRØ]    26:  else
[R35]    27:   writeln('Es wurden Parameter',
[UM2]    28:   ' übergeben!');
[XF1]    29:  writeln;
[KNØ]    3Ø: end.
```

Listing 1. Mit »chkparam.pas« greifen Sie direkter auf ein Feld des PSP zu

```
[Y13]     1: program umgebung;
[5H1]     2: uses crt;
[E86]     3: {Stellt alle Variablen der aktuellen
[HD5]     4:  Umgebung mit ihren Inhalten dar.
[ØØØ]     5:
[M34]     6:  Autor: Martin Althaus 2/9Ø
[ØØØ]     7:
[4Ø4]     8:  (C) 199Ø Redaktionsbüro
[WS2]     9:           Everts & Hagedorn
[U7Ø]    1Ø: }
[ØØØ]    11:
[6XØ]    12: const
[RB3]    13:  umgeb_low:byte=$2c;
[MD3]    14:  umgeb_high:byte=$2d;
[LB4]    15:  {Zeiger auf Feld im PSP}
[ØØØ]    16:
[RNØ]    17: var
[PT3]    18:  umgeb_seg,index:word;
[ØØØ]    19:
[MUØ]    2Ø: begin
[UG4]    21:  umgeb_seg:=mem[PrefixSeg:
[NA5]    22:  umgeb_high]*256+mem[PrefixSeg:
[362]    23:  umgeb_low];
[BP5]    24:  {ermittle Adresse der Umgebung}
[WT4]    25:  index:=Ø; {Starte am Beginn}
[P31]    26:  repeat
[YK5]    27:   {wiederhole für alle Strings}
[7X4]    28:   while (mem[umgeb_seg:index]
[331]    29:         <>Ø) do
[DN4]    3Ø:    {bis zum Ende des Strings}
[MUØ]    31:    begin
[MA4]    32:     write(char(mem[umgeb_seg:
[VE1]    33:           index]));
[A73]    34:     {Gebe Zeichen aus}
[NG2]    35:     index:=index+1;
[PX3]    36:     {Index heraufzählen}
[ZNØ]    37:    end;
[KU3]    38:    writeln; {neue Zeile}
[NG2]    39:    index:=index+1;
[TC4]    4Ø:    {Null-Byte überspringen}
[FØ5]    41:   until (mem[umgeb_seg:index+1]
[9FØ]    42:         =Ø);
[5L5]    43:  {bis Ende der Strings erreicht}
[XF1]    44:  writeln;
[KNØ]    45: end.
```

Listing 2. Das Programm »umgebung.pas« zeigt Ihnen die Umgebungsvariablen an

```
[F83]     1: program stacktest;
[ØØØ]     2:
[LZ4]     3: { Demonstriert das Wachsen des
[BR4]     4:   Stack-Zeigers bei verschach-
[GU5]     5:   telten Unterfunktionsaufrufen }
[ØØØ]     6:
[RNØ]     7: var
[ND2]     8:  zaehler: byte;
[ØØØ]     9:
[HP3]    1Ø: procedure stapelzeig;
[ØØØ]    11:
[GE5]    12: {gibt den Wert des Stapelzeigers
[K74]    13:  auf dem Bildschirm aus}
[ØØØ]    14:
[924]    15: begin {von STAPELZEIG}
[TF5]    16:  writeln('Der Stapelzeiger SPtr',
[NE4]    17:  ' steht in Unterprogramm ',
[4D3]    18:   zaehler,' auf ',SPtr);
[DV3]    19: end; {von STAPELZEIG}
[ØØØ]    2Ø:
[TP2]    21: procedure eins;
[ØØØ]    22:
[RZ3]    23: procedure zwei(b:word);
[ØØØ]    24:
[VW3]    25: procedure drei(c:word);
[ØØØ]    26:
[PU2]    27: begin {von DREI}
[W93]    28:  zaehler:=zaehler+1;
[GL3]    29:  {Eine Ebene tiefer}
[432]    3Ø:  stapelzeig;
[7Ø3]    31:  {zeige Pointer}
[ØA3]    32:  zaehler:=zaehler-1;
[UØ3]    33:  {Eine Ebene rauf}
[EY2]    34: end; {von DREI}
[ØØØ]    35:
[PV2]    36: begin {von ZWEI}
[W93]    37:  zaehler:=zaehler+1;
[GL3]    38:  {eine Ebene tiefer}
[432]    39:  stapelzeig;
[US3]    4Ø:  {zeige Stack-Pointer}
[HD1]    41:  drei(1ØØ);
[7H4]    42:  {rufe Unterprogramm aus}
[ØA3]    43:  zaehler:=zaehler-1;
[UØ3]    44:  {eine Ebene rauf}
[B13]    45: end; {von ZWEI}
[ØØØ]    46:
[8W2]    47: begin {von EINS}
[W93]    48:  zaehler:=zaehler+1;
[GL3]    49:  {eine Ebene tiefer}
[432]    5Ø:  stapelzeig;
[US3]    51:  {zeige Stack-Pointer}
[FF1]    52:  zwei(2ØØ);
[7H4]    53:  {rufe Unterprogramm aus}
[ØA3]    54:  zaehler:=zaehler-1;
[UØ3]    55:  {eine Ebene rauf}
[TZ2]    56: end; {von EINS}
[ØØØ]    57:
[SZ3]    58: begin {Hauptprogramm}
[KX1]    59:  zaehler:=Ø;
[6B2]    6Ø:  {erste Ebene}
[432]    61:  stapelzeig;
[US3]    62:  {zeige Stack-Pointer}
[6VØ]    63:  eins;
[LF4]    64:  {rufe Unterprogramm auf}
[RS3]    65: end. {Hauptprogramm}
```

Listing 3. Mit »stacktest.pas« kontrollieren Sie den Stackpointer

```
[F23]     1: program konstant;
[ØØØ]     2:
[KC5]     3: {Demonstriert die Veränderung
[XF5]     4:  von typisierten »Konstanten«
[9Y1]     5:  im Speicher
[ØØØ]     6:
[M34]     7:  Autor: Martin Althaus 2/9Ø
[ØØØ]     8:
[4Ø4]     9:  (C) 199Ø Redaktionsbüro
[WS2]    1Ø:           Everts & Hagedorn
[U7Ø]    11: }
[ØØØ]    12:
[6XØ]    13: const
[C23]    14:  festwert:byte=1ØØ;
[LA4]    15:  {das ist die Konstante}
[ØØØ]    16:
[RNØ]    17: var
[3A3]    18:  adrseg,adrofs: word;
[ØØØ]    19:
[MUØ]    2Ø: begin
[D25]    21:  write('Ich, Festwert, bin eine');
[3L5]    22:  writeln(' Konstante: ',festwert);
[XT3]    23:  adrseg:=seg(festwert);
[AW3]    24:  adrofs:=ofs(festwert);
[2T5]    25:  {Ermitteln der Speicheradresse}
[U14]    26:  mem[adrseg:adrofs]:=2ØØ;
[5W6]    27:  {und direkte Manipulation im Speicher}
[TE4]    28:  write('Oder etwa nicht? ');
[183]    29:  writeln(festwert);
[KNØ]    3Ø: end.
```

Listing 4. Das Programm »konstant.pas« erlaubt Ihnen die direkte Manipulation von Variableninhalten

Assemblerunterstützung für Turbo-Pascal-Programme

Viele Programmierprobleme bestehen in der Schwierigkeit, bestimmte Routinen statt in Pascal direkt in Maschinensprache zu formulieren. Wie sich kleine Programme und Module direkt in einen Turbo-Pascal-Quelltext einbinden lassen, zeigt dieser Artikel.

Für kurze Maschinenprogramme bietet Turbo Pascal eine benutzerfreundliche und einfache Art, sie in ein Pascal-Programm einzubinden. Turbo Pascal erlaubt es Ihnen, einen Objektcode direkt in einen Pascal-Quelltext zu integrieren. Diese Art der Programmierung hat den Vorteil, daß Sie kleine Maschinenprogramme erzeugen können, selbst wenn Sie keinen Assembler besitzen. Für viele Module in Maschinensprache reicht es aus, diese mit dem MS-DOS-Dienstprogramm »debug« in den Objektcode zu übersetzen und dann direkt in ein Pascal-Programm einzubinden.

Dazu wird die Inline-Anweisung benutzt. Es werden allerdings keine symbolischen Assemblerbefehle (»Mnemonics«), sondern nur »echte« Maschinenbefehle, also Zahlen beziehungsweise Bitmuster, verarbeitet. Zur Erinnerung: Auch wenn die in einem Assemblerquelltext vorkommenden Befehle häufig Maschinensprache genannt werden, sind sie dennoch nur symbolische Namen für die eigentlichen Maschinenspracheanweisungen.

Die einzelnen Zahlen, die ein Maschinenprogramm bezeichnen, werden durch einen Schrägstrich voneinander getrennt. So bindet die folgende Zeile:

```
inline($8b/$ec);
```

zum Beispiel die Maschinencodes für den Befehl

```
mov bp,sp
```

in ein Pascal-Programm ein. Der Compiler übernimmt während der Programmbearbeitung alle Maschinencodes des Inline-Befehls direkt in den Objektcode des Pascal-Programms. Das Schöne am Inline-Kommando ist die Tatsache, daß auch Variablen in den Inline-Code mit aufgenommen werden können. Verwenden Sie beispielsweise die folgende Pascal-Sequenz

```
inline(status);
```

so wird an der entsprechenden Stelle im Programmcode die Adresse der Variable oder Konstanten namens »status« eingefügt. Alle globalen Variablen oder Konstanten stehen dann an der entsprechenden Offset-Adresse im Datensegment. Der Zugriff auf die Variable oder Konstante »status« könnte nach der obigen Vorgabe also beispielsweise durch die Anweisung:

```
mov ax,ds:[status]
```

erfolgen. Bei allen globalen Variablen und Parametern fügt Turbo Pascal also immer den Offset der Variablen zu Beginn des Datensegments in den Inline-Code ein.

Lokale Variablen werden im Stapelsegment gespeichert. Die Funktionsargumente einer Inline-Funktion oder -Prozedur werden dabei als Relativwert zum BP-Register auf dem Stapel abgelegt. Nach der Prozedurdeklaration

```
procedure demoproz (var anzahl: word; adresse: ^word);
begin
...
end;
```

können Sie auf die Variablen »anzahl« und »adresse« innerhalb des Inline-Rumpfes leicht zugreifen. Werden die Namen der Variablen in den Inline-Code eingesetzt, so erzeugt Turbo Pascal für jede Variable den Offset relativ zum BP-Register. Wird also im obigen Beispiel die Inline-Sequenz

```
inline ($8b/$86/anzahl/ (* MOV AX,anzahl[BP] *)
$c4/$be/adresse); (* LES DI,adresse[BP] *)
```

programmiert, berechnet Turbo Pascal an den Stellen, an denen die Variablen »anzahl« und »adresse« im Text vorkommen, automatisch einen Offset zum BP-Register. Dadurch kann eine Inline-Prozedur tatsächlich auf die Variablen und ihre Werte zugreifen.

Es existieren zwei Operatoren, die sich im Zusammenhang mit dem Inline-Code sinnvoll einsetzen lassen. Es handelt sich um die Operatoren »<« und »>«. Mit ihnen kann man den höher- oder niederwertigen Teil eines Wortes auswählen. Das kann sehr hilfreich sein. Nehmen Sie einmal an, Sie wollten in einer Inline-Prozedur die Zwei-Byte-Zahl 1000 auf das AH- und das CH-Register verteilen. Das AH-Register soll den höherwertigen Teil und das CH-Register den niederwertigen Teil der Zahl erhalten. Sollen für diese Ladeoperationen die Mov-Maschinenbefehle mit AH beziehungsweise CH als Argumente verwendet werden, muß anschließend als Parameter das Byte stehen, das in AH beziehungsweise CH geladen werden soll. Sie müssen also die Zahl 1000 in ein niederwertiges und ein höherwertiges Byte unterteilen. Das könnte im Kopf oder aber mit einem Taschenrechner erfolgen. Mit den Operatoren »<« und »>« hingegen wird sofort »<1000« beziehungsweise »>1000« in den Inline-Befehl eingefügt.

Der »<«-Operator liefert als Argument den niederwertigen Teil, während der Operator »>« für den höherwertigen Teil des Bytes zuständig ist. Bei der Übergabe von Argumenten an eine Prozedur oder Funktion, die vollständig als Inline-Routine aufgebaut ist, verfährt Turbo Pascal prinzipiell wie beim Aufruf einer anderen Funktion oder Prozedur.

Trifft der Turbo-Pascal-Compiler auf den Inline-Code, so hat er prinzipiell zwei Möglichkeiten, die entsprechenden Maschinenbefehle in den Programmcode einzubinden. Einerseits können sie – wie bei einem Assemblermakro – direkt an der Stelle in den Code eingefügt werden. Andererseits können sie – wie bei einem Unterprogrammaufruf – an einer Stelle im Programmcode einmal plaziert und von anderen Adressen aus mit einem Call-Kommando aufgerufen werden. Welches Verfahren der Compiler anwendet, hängt von der Position des Inline-Befehls im Programm ab. Steht das Inline-Kommando irgendwo mitten in einer Prozedur, einer Funktion oder inmitten des Hauptprogramms, wird der Code in jedem Fall an der entsprechenden Stelle direkt eingefügt. Das kann natürlich dennoch in einem Unterprogramm sein, nur wird dann für das »Inline« kein eigenes Unterprogramm mehr erzeugt.

Haben Sie eine Funktion oder Prozedur geschrieben, die ausschließlich aus einer Inline-Anweisung besteht, kann der Compiler auch ein Unterprogramm erzeugen. Er tut das immer dann, wenn das Inline-Kommando durch eine Begin- und eine End-Direktive eingeschlossen wird. Soll zum Beispiel durch Einbinden des Cld-Maschinenbefehls innerhalb eines Pascal-Programms das DF-Kennzeichen gesetzt werden, so geschieht das durch die folgende Inline-Anweisung:

```
inline($fc);
```

Plazieren Sie diese Zeile in eine eigene Prozedur ohne »begin« und »end«, ergibt sich folgendes:

```
procedure setdf; inline($fc);
```

In diesem Fall wird der Inline-Code, nämlich das Byte FChex, an der Position, an der der Aufruf der Setdf-Prozedur im Programm auftritt, direkt in das Compilat eingebunden. Verwenden Sie hingegen die Deklaration

```
procedure setdf;
begin
inline($fc);
end;
```

für die gleiche Prozedur, wird an der Stelle, an der »setdf« aufgerufen wird, ein Unterprogramm ausgelöst.

Bevor Sie beginnen, mit dem Inline-Befehl und seiner Benutzung zu experimentieren, sei eine kleine Warnung angebracht. Ein Tippfehler im Maschinenprogramm genügt schon, den Rechner zum »Aussteigen« zu bringen. Speichern Sie Ihr Programm also möglichst mehrmals auf einer Diskette oder der Festplatte.

(Martin Althaus/ks)

Lösungen in

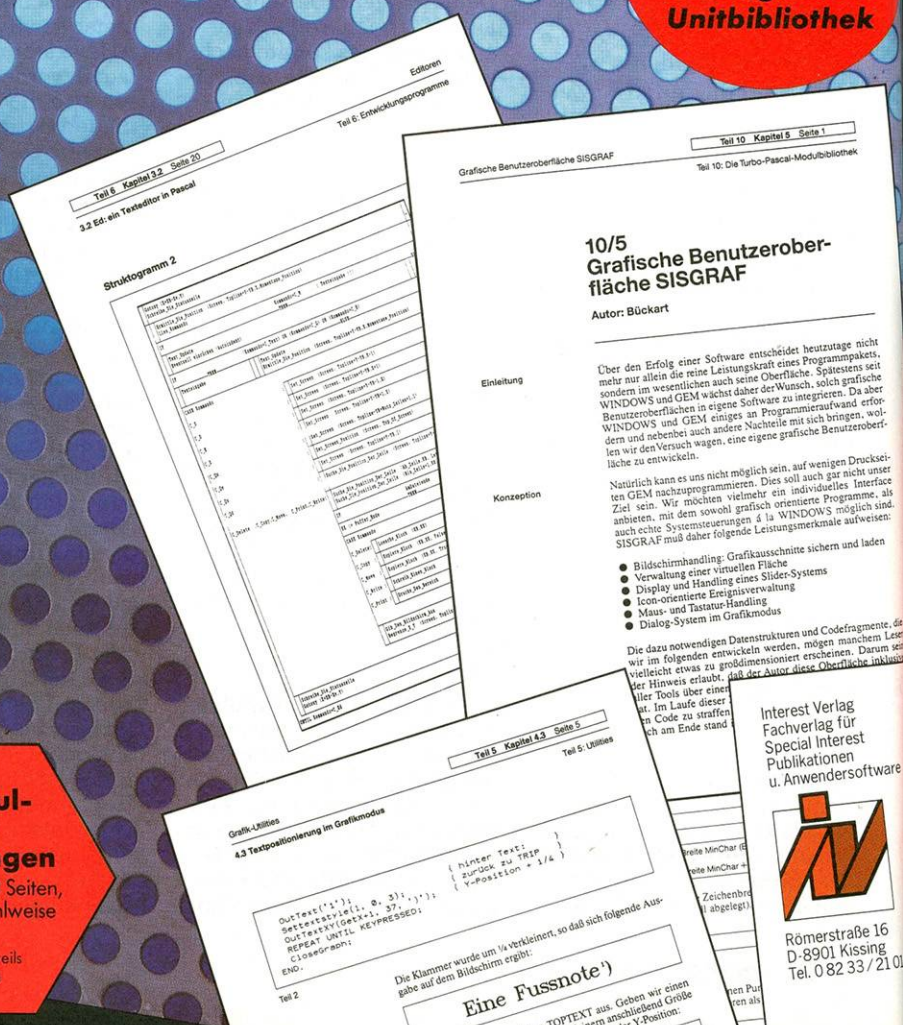
Turbo-Pascal

4.0, 5.0 u. 5.5

Dieses neue Nachschlagewerk bietet Ihnen:

- **detailliertes Know-how zum Turbo-Pascal-System;**
- **den Lehrgang Software-Engineering;**
- **Betriebssystemroutinen,** u.a. ROM-BIOS und IBM-Hardware (u.a. Druckertreiber für Turbo-Pascal V4.0 bis 5.5, asynchrone Datenübertragung, Routinen zur Druckeransteuerung), Turbo-Pascal und MS-DOS-Schnittstelle;
- **hardwarenahe Programmierung:** u.a. Rechnerkopplung, Floppytuning, Drucker-/Bildschirmwechsel mit Turbo V4.0 bis 5.5;
- **raffinierte Utilities: Schnittstellenutilities** (u.a. Turbo-Inliner, Lesen von dBase III-Dateien), **Grafikutilities** (digitale Bildverarbeitung, Grafiken ausdrucken), **Dateiverwaltungs-Utilities** (Ver- und Entschlüsseln von Dateien);
- **grafische Benutzeroberfläche SISGRAF:** GEM-ähnliche Benutzeroberfläche zur Einbindung in eigene Grafikprogramme oder zur Programmierung von Systemsteuerung wie Windows;
- **Grafik-Editor** zum Zeichnen und Ausdrucken von Linien oder Rechtecken; Invertieren, Spiegeln, Verschieben, Füllen von Flächen; **Zeichensatz- und Text-Editor;**
- **Anwenderprogramme für Technik/Mathematik,** u.a. lineare Algebra, statistische Qualitätskontrolle;
- **betriebswirtschaftliche Anwendungen:** z.B. Finanzbuchhaltung „SOBUCH", Restschuldberechnung für Darlehen im Vergleich, sonstige Anwendungen: z.B. Vokabeltraining, Spiele;
- **Turbo-Pascal-Modulbibliotheken,** u.a. String- und DOS-Routinen (z.B. Dialogverarbeitung im Grafik-Modus, Desktop-Steuerung), Mausfunktionen und Interrupts, Terminate Stay Resident-Programmierung (TSR-P);
- **Erweiterungsausgaben zum Grundwerk** mit neuen Utilities, Tools und Musterprogrammen für Turbo Pascal 5.X.

inklusive 3 Disketten mit allen Quelltexten + umfangreicher Unitbibliothek



Fordern Sie am besten gleich mit nebenstehender Karte an:

Musterprogramme, Utilities und Modulbibliotheken für raffinierte Lösungen in Turbo-Pascal und Weiterentwicklungen

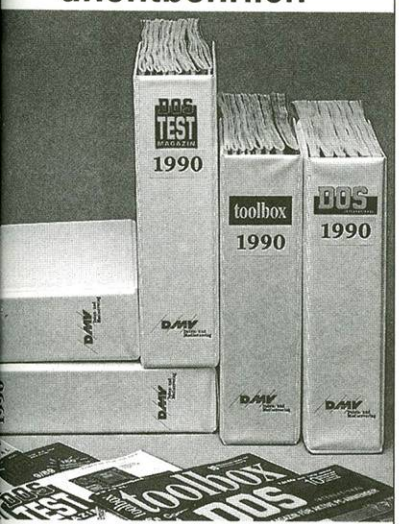
stabiler Ringbuchordner, Format DIN A4, Grundwerk ca. 780 Seiten, inkl. drei 5¼"-Disketten, Best.-Nr.: 2500, Preis: DM 98,–, wahlweise 3½"-Disketten (gegen DM 2,– Aufpreis).

Alle 2 – 3 Monate erhalten Sie Erweiterungsausgaben zum Grundwerk mit jeweils ca. 130 Seiten und einer Diskette zum Gesamtpreis von DM 59,– (Abbestellung jederzeit möglich).

Interest Verlag
Fachverlag für Special Interest Publikationen u. Anwendersoftware

Römerstraße 16
D-8901 Kissing
Tel. 0 82 33 / 21 01

Zum Sammeln unentbehrlich



Die DMV-Sammelordner

Sonderhefte • Sammelordner • Demonstrationen

Ausgabe DOS EXTRA DM 18,–	DATABOX 2 Stk. 5 1/4"-Disk DM 35,–	DATABOX 1 Stk. 3 1/2"-Disk DM 35,–
☐ 356 Nr. 3	☐ 357	☐ 358
☐ 381 Nr. 4	☐ 3811	☐ 3812
☐ 382 Nr. 5	☐ 3813	☐ 3814
☐ 383 Nr. 6	☐ 3815	☐ 3816
☐ 384 Nr. 7	☐ 3817	☐ 3818
☐ 385 Nr. 8	☐ 3819	☐ 3820
☐ 386 Nr. 9	☐ 3821	☐ 3822
☐ 387 Nr. 10	☐ 3823	☐ 3824
☐ 388 Nr. 11	☐ 3825	☐ 3826
Pascal Sonderdruck	☐ 346	☐ 347
DMV-Computer-wissen Band 1	☐ 3512	☐ 3513

DOS-Taschenbücher	5 1/4"-Diskette DM 24,–	3 1/2"-Diskette DM 24,–
BAND 1	☐ 60501	☐ 60502
BAND 3	☐ 60521	☐ 60522
BAND 4	☐ 60531	☐ 60532

Best.-Nr.		DM
☐ 60500	**DOS-Taschenbuch Band 1**	**29,80**
☐ 60510	**DOS-Taschenbuch Band 2**	**29,80**
☐ 60520	**DOS-Taschenbuch Band 3**	**29,80**
☐ 60530	**DOS-Taschenbuch Band 4**	**29,80**
☐ 402	**Die Basic2-Toolbox**	**49,–**
☐ 403	**Die Basic2-Toolbox**, 5 1/4"-Diskette zum Buch	**29,–**
☐ 345	**PASCAL Sonderdruck**	**25,–**
☐ 3511	**DMV-Computer-Wissen**, Band 1, Basic-Toolbox	**18,–**

Sammelordner — **2 Stk. 15,80**

☐ 530 **DOS**
☐ 534 **toolbox**
☐ 536 **DOS TEST**
☐ 531 **PC International**
☐ 535 **PCpur**
☐ 537 **AMIGA DOS**

Zwischensumme + Porto und Verpackung (Inland 4,– DM, Ausland 6,– DM) _____

Demodisketten jeweils DM 5,–

☐ 001 Fraktal 3D + MaskEdit
☐ 003 ERGO 3.0
☐ 005 ConText PRO
☐ 007 3D-Draw 1.0
☐ 008 DMV-Show-Manager
☐ 009 Navigator 4.0

toolbox-PD-Service jeweils DM 10,–

5 1/4"	3 1/2"	
☐ 296	–	PD-Modula-Compiler, MS-DOS
–	☐ 297	PD-Modula-Compiler, Atari ST
–	☐ 298	PD-Modula-Compiler, Amiga
☐ 2652	☐ 2653	PD-Prolog-Compiler, MS-DOS
☐ 2654	☐ 2655	SSQL-Datenbankabfrage
☐ 2656	–	Quick-Basic-Übungscompiler

Gesamtbetrag: _____ DM

PC-Software

vom

DMV Versandservice

- leistungsstark
- praxisbewährt
- preisbewußt

PC-Software

5 1/4"	3 1/2"		DM
2282 ☐	2281 ☐	**ConText EASY,** (MS-DOS)	99,–
2407 ☐	2408 ☐	**ConText PRO,** (MS-DOS)	199,–
2409 ☐	2410 ☐	**Upgrade - ConText PRO**	129,–
2415 ☐	2416 ☐	**ConText-ADREVA,** Adreßverwaltung	49,–
2419 ☐	2420 ☐	**ConText-Zeicheneditor**	99,–
2421 ☐	2422 ☐	**ConText-Rechtschreibhilfe**	69,–
2650 ☐	2651 ☐	**DMV-Power-Pack**	349,–
2615 ☐	2616 ☐	**H.A.I.-Tech,** (MS-DOS)	199,–
2522 ☐	2521 ☐	**ERGO 3.0,** DOS-Benutzeroberfläche	199,–
229 ☐	235 ☐	**Hyperkey,** Version 4.0, (MS-DOS)	99,–
256 ☐	257 ☐	**VirusDoktor,** (MS-DOS)	99,–
2702 ☐	2703 ☐	**DMV-Show-Manager**	199,–
238 ☐	239 ☐	**MaskEdit Plus** mit einem Treiber	99,–
2613 ☐	2614 ☐	**MaskEdit Plus,** Paket mit zwei Treibern	149,–
2621 ☐	2622 ☐	**MaskEdit Plus,** Paket mit drei Treibern	199,–

Turbo Pascal 3/4/5	Turbo C Microsoft C	Top-Speed Modula-2	Turbo Basic	Quick Basic
☐ 2632	☐ 2633	☐ 2634	☐ 2635	☐ 2636

227 ☐	233 ☐	**BCi DisAsm/i86,** (MS-DOS)	199,–
2402 ☐	2401 ☐	**TopSpeed Modula-2,** Compiler	348,–
2404 ☐	2403 ☐	**TopSpeed Modula-2,** TechKit	248,–
2406 ☐	2405 ☐	**TopSpeed Modula-2,** Debugger	248,–
2412 ☐	2411 ☐	**Paket Compiler + Debugger**	555,–
2413 ☐	2412 ☐	**TopSpeed Modula-2,** OS/2-Version	698,–
6001 ☐	6002 ☐	**DOS-Tools 1,** RSM-Manager	69,–
6007 ☐	6008 ☐	**DOS-Tools 4,** 3D-Draw, Version 1.0	99,–
6017 ☐	6018 ☐	**3D-Draw Professional**	199,–
6021 ☐	6022 ☐	**Upgrade von 3D-Draw auf Professional**	109,–
6019 ☐	6020 ☐	**Navigator 4.0**	69,–
6009 ☐	6010 ☐	**DOS-Tools 5,** GW-Basic-Toolbox-Oberfläche	69,–

5 1/4"	3 1/2"		DM
6011 ☐	6012 ☐	**DOS-Tools 6,** GW-Basic-Toolbox-DOS-Funktionen	69,–
6013 ☐	6014 ☐	**DOS-Tools 7,** GW-Basic-Toolbox-Hercules-Grafik	69,–
20500 ☐	20501 ☐	**PCpur Software I,** MultiFont Plus	69,–
20502 ☐	20503 ☐	**PCpur Software Ia,** Zusatzfonts MultiFont Plus	39,–
20504 ☐	20505 ☐	**PCpur Software II,** ASCII-Tiger, Programmeditor	69,–
20506 ☐	20507 ☐	**PCpur Software III,** Festplatten-Optimierer	69,–
2804 ☐	2805 ☐	**toolbox Spezial II,** PASCOMP	35,–
2808 ☐	2809 ☐	**toolbox Spezial IV,** CALC	35,–
2811 ☐	2812 ☐	**toolbox Spezial V,** STRUKTO	35,–
2815 ☐	2816 ☐	**toolbox Spezial VII,** dt. Silbentrennung	35,–
2819 ☐	2820 ☐	**toolbox Spezial IX,** HPGL-Plotter	35,–
2823 ☐	2824 ☐	**toolbox Spezial XI,** MAP, MARK & RELEASE	35,–
2827 ☐	2828 ☐	**toolbox Spezial XIII,** Trickkiste Vol. I	35,–
2829 ☐	2830 ☐	**toolbox Spezial XIV,** Trickkiste Vol. 2	35,–
2831 ☐	2832 ☐	**toolbox Spezial XV,** Fitting & Lin.-Glei.	35,–
2833 ☐	2834 ☐	**toolbox Spezial XVI,** nat.-sprachl. Schnittst.	35,–
2835 ☐	2836 ☐	**toolbox Spezial XVII,** EMS-Speicher-Software	35,–
2837 ☐	2838 ☐	**toolbox Spezial XVIII,** Alkane-Klassifizierung	35,–
2361 ☐	2362 ☐	**MAUSALL Plus**	35,–
246 ☐	247 ☐	**C-Funktionssammlung**	99,–
2952 ☐	____	**GEMInterface PC,** Turbo Pascal 4 + 5.0	198,–
2954 ☐	____	**GEMInterface PC,** TopSpeed Modula-2	198,–
1371 ☐	1372 ☐	**PC-Spiele-Sampler**	49,–
161 ☐	162 ☐	**KNOW-PC, Unterhaltungsspiel**	49,–
1293 ☐	1294 ☐	**Ergänzungsfragen zu KNOW-PC**	29,–
248 ☐	249 ☐	**Fraktal-Generator 3D,** MS-DOS	69,–
1374 ☐	____	**DOS-Spielekiste:** Populous PC	69,95

+ Porto/Verpackung (Inland 4,– DM, Ausland 6,– DM) _____ DM = Gesamtbetrag: _____ DM

Ihre Anforderungskarte

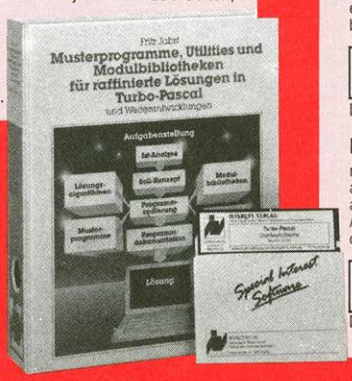
Ja, senden Sie mir sofort

___ Expl. **„Musterprogramme, Utilities und Modulbibliotheken für raffinierte Lösungen in Turbo-Pascal und Weiterentwicklungen"**

stabiler Ringbuchordner, Format DIN A4, Grundwerk ca. 780 Seiten, inkl. drei 5 1/4"-Disketten, Bestell-Nr. 2500, Preis: DM 98,–, wahlweise

☐ zwei 3 1/2"-Disketten (gegen DM 2,– Aufpreis)

Alle 2–3 Monate erhalte ich Erweiterungsausgaben zum Grundwerk mit jeweils ca. 130 Seiten, inkl. Diskette zum Gesamtpreis von DM 59,– (Abbestellung jederzeit möglich).



Bitte abtrennen oder ausschneiden!

Meine Anschrift:

Name _____

Vorname _____

Straße, Haus-Nr. _____

PLZ, Ort _____

Unterschreiben Sie hier bitte Ihre Bestellung!

Bei Minderjährigen ist die Unterschrift eines gesetzlichen Vertreters erforderlich. Ohne Ihre Unterschrift kann die Ansichtsbestellung nicht bearbeitet werden.

✗ _____

Datum Unterschrift

Bitte unterschreiben Sie auch Ihre Sicherheitsgarantie,

mit der Sie folgendes zur Kenntnis nehmen: Sie haben das Recht, Ihr angefordertes Werk innerhalb von 10 Tagen ab Lieferung an die Bestelladresse zurückzusenden, wobei für die Fristwahrung das Datum der Absendung genügt. Sie kommen dadurch von allen Verpflichtungen aus der Bestellung frei.

✗ _____

Datum Unterschrift

PC-Sonderhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/ORT

☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.

☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

1. Informieren

DOS-Taschenbücher und DO
EXTRA-Sonderhefte sind eine
verlässige Informationsquelle
jeden PC-Besitzer. Nutzen au
Sie dieses Wissen, und vertie
Sie es mit den dazu angeboten
Programmen aus dem Diskette
Service. Den PC beherrschen

- **DOS-EXTRA-Sonderhefte**
- **DOS-Taschenbüchern**
- **DOS-Disketten-Service**

Zur Erinnerung:
Meine DOS-Sonderhefte, DO
Taschenbücher, DOS-Disketten w
den bestellt am

PC-Software

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/ORT

☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.

☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

2. Aufrüsten

PC-Software vom DMV-Verlag
unter Kennern bereits ein Begr
und auch ein gutes Angebot
Sie, Ihren PC mit starken und a
wendergerechten Applikation
auszustatten. Von der Textve
arbeitung bis zum Programmie
system, beim DMV-Versan
service kaufen Sie günstig e
Den PC konsequent nutzen
PC-Software aus dem DMV-Ve
sandservice

- **PC-Anwendungen**
- **PC-Applikationen**
- **PC-Entwicklungssysteme**

Zur Erinnerung:
Meine PC-Software wurde bestellt

Nutzen Sie unser zusätzliches Angebot:

___ Expl. **„Intel 16/32 Bit Assemblerhandbuch“**

mit

- Kursen für perfekte Programmiertechnik
- detaillierten Dokumentationen und Befehlsbeschreibungen aller 8086/8088- bis 80386-Prozessorbefehle
- erweiterter 8086-Assembler
- Makrobibliotheken
- bewährten Musterprogrammen und Utilities

stabiler Ringbuchordner, Format DIN A4, Grundwerk ca. 1000 Seiten, inkl. zwei 5¼"-Disketten, Bestell-Nr. 2200, Preis: DM 98,–, wahlweise

☐ 3½"-Diskette (gegen DM 2,– Aufpreis).

Alle 2 – 3 Monate erhalten Sie Ergänzungsausgaben zum Grundwerk mit je ca. 130 Seiten, inkl. Diskette zum Gesamtpreis von DM 59,– (Abbestellung jederzeit möglich).

IN 110 362

60 Pfennig, die sich lohnen!

Postkarte

INTEREST-VERLAG
Fachverlag für Special Interest
Publikationen und Anwendersoftware
z. Hd. Herrn Michel

Römerstraße 16

D-8901 Kissing

Verlagsgarantie

- Sie erhalten von mir PC-Fachinformationen, von absoluten Spezialisten für Sie geschrieben.
- Alle Musterprogramme und Utilities wurde auf den verschiedenen PCs gründlich geprüft.
- Sie haben ein 10tägiges Rückgaberecht, damit Sie das Werk in Ruhe zu Hause prüfen können.
- Ihr Nachschlagewerk ist immer aktuell. Dafür sorgt unser Erweiterungsservice.
- Den Erweiterungsservice können Sie jede zeit kündigen. Sie gehen also kein Risiko ein. Darauf gebe ich Ihnen mein Wort.

Ihr INTEREST-VERLAG
Fachverlag für Special Interest
Publikationen und Anwendersoftware

Michael Boos

Michael Boos
Verlagsleiter

DOS INTERNATIONAL

Kompetenz

für

PC-Personalcomputer

im

Abonnement

deutlich

günstiger

Abo-Order DOS International

Hiermit bestelle ich **»DOS International«** für mindestens

☐ **12 Ausgaben** ☐ **6 Ausgaben**

Preise: **BRD und West-Berlin**	12 Ausgaben	82,50 DM
	6 Ausgaben	42,– DM
Auslandspreise: **Europa**	12 Ausgaben	110,– DM
	6 Ausgaben	55,– DM
Außereuropäisches Ausland	12 Ausgaben	150,– DM
	6 Ausgaben	75,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Abo-Order DOS DATABOX PLUS

Hiermit bestelle ich **»DOS DATABOX PLUS«** für mindestens

☐ **12 Ausgaben** ☐ **6 Ausgaben**

Diskettenformat: ☐ IBM 5 1/4" ☐ IBM 3 1/2"

Inlandspreise: 12 Ausgaben 300,– DM, 6 Ausgaben 150,– DM
Europa: 12 Ausgaben 320,– DM, 6 Ausgaben 160,– DM
Außereurop. Ausland: 12 Ausg. 360,– DM, 6 Ausg. 180,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort *(Bitte genaue Anschrift angeben!)*

Dieses Abonnement verlängert sich automatisch um 6 bzw. 12 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum 1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

☐ **Bequem und bargeldlos durch Bankabbuchung**

Bankleitzahl

Konto-Nr./-Inhaber

Geldinstitut

☐ **Gegen Rechnung** – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten — Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite eine zweite Unterschrift leisten!

DOS INTERNATIONAL

Information

über

PC-Personalcomputer

sammeln Sie mit

- **DOS-Einzelheften**
- **DATABOX PLUS**
  mit mehr als 700 KByte Software und Bonus-programm

DOS-Einzelhefte und DATABOX

Einzelheft Ausgabe 6.50 DM/Stck	DATABOX 5 1/4"-Diskette 24,– DM/Stck	DATABOX 3 1/2"-Diskette 24,– DM/Stck
– 1/88	☐	☐
2/88	☐	☐
☐ 3/88	☐	☐
☐ 4/88	☐	☐
☐ 5/88	☐	☐
☐ 6/88	☐	☐
☐ 7/88	☐	☐
☐ 8/88	☐	☐
☐ 9/88	☐	☐
Einzelheft Ausgabe 7,50 DM/Stck		
☐ 10/88	☐	☐
☐ 11/88	☐	☐
☐ 12/88	☐	☐
☐ 1/89	☐	☐
☐ 2/89	☐	☐
☐ 3/89	☐	☐
☐ 4/89	☐	☐
☐ 5/89	☐	☐
☐ 6/89	☐	☐
☐ 7/89	☐	☐

Einzelheft Ausgabe 7,50 DM/Stck	DATABOX PLUS DATABOX + Bonus 5 1/4" 2 Disketten 24,– DM/Stck	DATABOX PLUS DATABOX + Bonus 3 1/2"-Diskette 24,– DM/Stck
☐ 8/89	☐	☐
☐ 9/89	☐	☐
☐ 10/89	☐	☐
☐ 11/89	☐	☐
☐ 12/89	☐	☐
– 1/90	☐	☐
☐ 2/90	☐	☐
☐ 3/90	☐	☐
☐ 4/90	☐	☐
– 5/90	☐	☐
☐ 6/90	☐	☐
☐ 7/90	☐	☐

+ Porto/Verpackung
(Inland 4,– DM, Ausland 6,– DM) _____ DM Gesamtbetrag _____ DM

DOS INTERNATIONAL

Vorsprung

für

PC-Personalcomputer

mit

DOS EXTRA

im

Abonnement

Abo-Order DOS EXTRA

Hiermit bestelle ich **»DOS EXTRA«** für mindestens

☐ **4 Ausgaben** (vierteljährliche Erscheinungsweise)

Preise: **BRD und West-Berlin**	4 Ausgaben 60,– DM
Auslandspreise: **Europa**	4 Ausgaben 70,– DM
Außereuropäisches Ausland	4 Ausgaben 80,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Dauerbezug DOS EXTRA DATABOX

☐ Hiermit bestelle ich **»DOS EXTRA DATABOX«** im Dauerauftrag mit Kündigungsmöglichkeit zum vierteljährlichen Erscheinungstermin.
Achtung: Nur im Bankabbuchungsverfahren möglich!

Preise: **BRD und West-Berlin:**	je 39,– DM
Auslandspreise: **Europa:**	je 41,– DM
Außereurop. Ausland:	je 42,– DM

Lieferung erfolgt ab der nächsterreichbaren Ausgabe.

Lieferanschrift

☐ Geschäft ☐ Privat

Vorname, Name

Firma (nur wenn Lieferadresse)

Straße, Nr./Postfach

PLZ/Ort *(Bitte genaue Anschrift angeben!)*

Dieses Abonnement verlängert sich automatisch um 6 bzw. 12 Ausgaben, wenn es nicht sechs Wochen vor Ablauf schriftlich gekündigt wird.

Datum 1. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

☐ **Bequem und bargeldlos durch Bankabbuchung**

Bankleitzahl

Konto-Nr./-Inhaber

Geldinstitut

☐ **Gegen Rechnung** – zahlbar innerhalb zwei Wochen nach Erhalt. (Bitte keine Vorauszahlung leisten — Rechnung abwarten.)

Bitte unbedingt auf der Vorderseite eine zweite Unterschrift leisten!

DOS INTERNATIONAL

Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufschreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

1. Abonnieren

Jeden Monat neu, vermitteln Ih
nen die DOS International und
DOS EXTRA wichtige Informatio
nen zu Ihrem Personal-Computer.
Nutzen Sie dieses Wissen mit ei
nem Abonnement zu Ihrem ganz
persönlichen

- **Preisvorteil**
- **Informationsvorsprung**
- **Zustellkomfort**

Zur Erinnerung:
Mein persönliches Abonnement wur
de bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb ei
ner Woche beim DMV-Verlag, Postfach 250, 3440
Eschwege, schriftlich widerrufen kann, wobei bereits
die rechtzeitige Absendung des Widerrufschreibens
zur Fristwahrung ausreicht.

DOS INTERNATIONAL

Einzelhefte

Ja, ich mache von Ihrem Angebot Gebrauch und bestelle die umseitig ausgewählten Produkte. Bitte liefern Sie an folgende Anschrift:

Name

Straße/Postfach

PLZ/ORT

☐ Den Betrag bezahle ich mit dem beigefügten Verrechnungsscheck.

☐ Ich bitte um Lieferung per Nachnahme zuzüglich der Nachnahmegebühr (nur innerhalb der BRD).

Datum Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

2. Sammeln

DOS-International-Leser sind im
mer top-informiert. Daher sollte Ih
nen kein Heft oder Sonderheft in
Ihrer Sammlung fehlen. Das Ein
tippen von Listings und Program
men ersparen Sie sich durch
unseren günstigen Disketten-
Service. Immer gut beraten sind
Sie mit DOS International

- **Einzelheften**
- **Sonderheften**
- **Disketten-Service**

Zur Erinnerung:
Meine Hefte und Disketten wurde
bestellt am

DOS INTERNATIONAL

Abonnement

Rechnungsanschrift

Vorname, Name

Straße, Nr./Postfach

PLZ/Ort

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb einer Woche beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen kann, wobei bereits die rechtzeitige Absendung meines Widerrufschreibens zur Fristwahrung ausreicht. Ich bestätige das durch meine zweite Unterschrift.

Datum 2. Unterschrift (bei Minderjährigen des gesetzlichen Vertreters)

Bitte ausreichend frankieren

Antwortkarte

DMV-Verlag
DOS International
Postfach 250

3440 Eschwege

3. Schenken

Bei jedem Anlaß ist ein Abonne
ment ein Geschenk, das ankomm
und Ihrem Wunsch nach »guter
Erfolg« Ausdruck verleiht. Si
brauchen nur im Feld »Lieferar
schrift« die Adresse des Be
schenkten einzutragen.

- **DOS International**
- **DOS EXTRA**

Zur Erinnerung:
Mein Geschenk-Abonnement wurd
bestellt am

Widerrufsrecht

Ich weiß, daß ich diese Vereinbarung innerhalb ei
ner Woche beim DMV-Verlag, Postfach 250, 344
Eschwege, schriftlich widerrufen kann, wobei bereit
die rechtzeitige Absendung des Widerrufschreiben
zur Fristwahrung ausreicht.

Die Schnittstelle zwischen Turbo Pascal und Maschinensprache

Es gibt programmtechnische Probleme, die mit höheren Programmiersprachen nur unzureichend gelöst werden können, da lediglich die Maschinensprache einen wirklich uneingeschränkten Zugriff auf alle Komponenten des Computers erlaubt. Viele Aufgabenbereiche erfordern zudem höhere Geschwindigkeiten, als diese bei Turbo Pascal erreichbar sind. Wie lassen sich nun ganze Maschinenprogramme oder kleine Routinen in ein Pascal-Programm einbinden?

Für das Einbinden von Maschinensprache in einen Pascal-Quelltext bietet Turbo Pascal viele Möglichkeiten. Sie können ganze Funktionen oder Prozeduren oder zumindest Teile davon komplett in Maschinensprache verfassen. Dabei kann der Maschinencode entweder direkt in den Quelltext eingefügt oder extern definiert werden. Dieser Artikel zeigt Ihnen, wie Sie unter Turbo Pascal komplette Prozeduren in anderen Sprachen (zum Beispiel Assembler) schreiben können. Das direkte Einfügen von Maschinensprache in einen Pascal-Quelltext (sogenannter »Inline-Code«) wird an anderer Stelle in dieser DOS Extra behandelt.

Im folgenden wird eine Tabulatorfunktion in Turbo Pascal eingebunden, die – wie der Tab-Befehl in Basic – bei der Bildschirmausgabe an eine bestimmte Spaltennummer springt. Die Funktion soll als Argument einen Wert entgegennehmen, der eine der Bildschirmspalten zwischen 1 und 80 auswählt. Die Funktion soll einen String, bestehend aus Leerzeichen, zurückgeben, der – wenn er ausgegeben wird – den Cursor genau in der gewünschten Spalte der aktuellen Zeile positioniert. Wenn Sie die Funktion auf diese Weise programmieren, kann sie problemlos innerhalb eines Write-Kommandos verwendet werden.

Ein typischer Aufruf der zu entwickelnden Tab-Funktion sieht wie folgt aus:

```
writeln('Die Summe beträgt',tab(50),mark,' DM.');
```

Die Funktion übernimmt einen Byteparameter und erstellt daraus einen String, der an das aufrufende Programm zurückgegeben wird. Sie können die Funktion wie folgt in einem Pascal-Programm definieren:

```
function tab(spalte: byte):
string; external;
```

Dabei unterscheidet sich die Funktionsdeklaration nur im Hinblick auf das angefügte Schlüsselwort »external« von der üblichen Schreibweise. Dieses Schlüsselwort zeigt dem Pascal-Compiler an, daß die Funktion »außerhalb«, also auf Diskette oder Festplatte, als Maschinenprogramm gespeichert ist. Das Maschinenprogramm muß als Datei mit der Erweiterung »obj« vorliegen. Es ist also schon assembliert, aber noch nicht gebunden.

Natürlich muß sich das Maschinenprogramm an einige Konventionen halten. Dem Codesegment müssen Sie den Namen »code« oder »cseg« und dem Datensegment den Namen »data« oder »dseg« zuweisen; darüber hinaus dürfen keine Segmente definiert werden.

Alle Variablen im Datensegment dürfen nicht initialisiert werden, das heißt, Sie dürfen bei der Reservierung von Werten nur das Fragezeichen verwenden.

Variablen im Datensegment dürfen nicht initialisiert sein

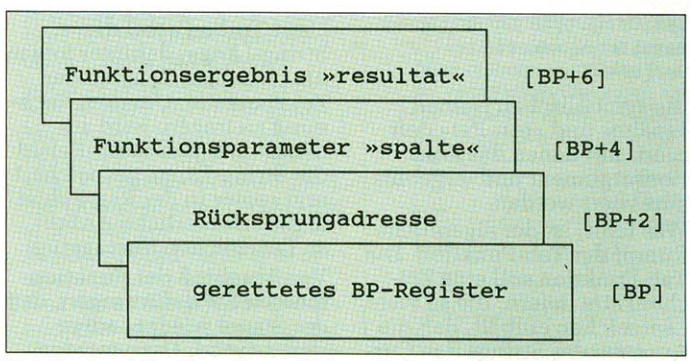
Alle Variablen, Prozeduren oder Funktionen, die von Pascal aus verfügbar sein sollen, müssen als »public« deklariert sein. Ferner muß die Prozedur oder Funktion in Assembler als Near-Prozedur definiert werden, da sie von Turbo Pascal aus mit »near« aufgerufen wird.

Das sind die wichtigsten Regeln bei der Erstellung externer Prozeduren oder Funktionen. Daraus können Sie nun schon das Gerüst für die Tab-Funktion in Assembler bauen:

```
code segment byte public ;Name des Codesegments
assume cs:code ;CS zeigt auf CODE
public tab ;Tabfunktion wird deklariert
tab proc near ;Tabfunktion muß Near-Prozedur sein
;...
;hier folgt der Programmteil
;...
tab endp ;Ende der Prozedur
code ends ;Ende des Segments
end ;Ende des Listings
```

Das Wichtigste fehlt bislang noch in unserem Listing: das eigentliche Programm. In das obige Gerüst muß noch der Assemblerquelltext für die Tabulatorfunktion eingebunden werden. Dabei taucht das erste Problem schnell auf: Wie erhält man das Funktionsargument der Tab-Funktion?

Dazu müssen Sie wissen, wie Turbo Pascal arbeitet, wenn Funktionen oder Prozeduren aufgerufen werden.



Der Zustand des Stapels nach dem Aufrufen der Tab-Funktion

Die Funktionsargumente werden der Reihe nach (also von links nach rechts in der Deklaration) auf den Stapel geschoben. Generell unterscheidet Pascal zwischen Wertparametern und Variablenparametern. Bei Wertparametern (sogenannten Call-By-Value-Parametern) verändert eine Modifizierung des Wertes innerhalb der Funktion oder Prozedur deren Wert im Hauptprogramm nicht. Bei Variablenparametern (sogenannten Call-By-Reference-Parametern) wirkt sich eine Veränderung innerhalb der Funktion oder Prozedur auf das gesamte Programm aus. Das liegt daran, daß Wertparameter direkt übergeben werden, während bei Variablenparametern ein Zeiger auf den Speicherplatz der Variablen überreicht wird. Im Pascal-Quelltext kennzeichnen Sie diese Unterscheidung durch das Einfügen des Schlüsselwortes »var« vor einem Parameter in der Deklaration der Funktion oder Prozedur. Variablenparameter belegen immer zwei Worte Speicherplatz auf dem Stapel: die Segment- und die Offset-Adresse der übergebenen Variablen.

Wertparameter hingegen belegen je nach Datentyp unterschiedlich viel Platz auf dem Stapel. Bytes, Worte, Integer-, Wahrheitswert- und Zeichenvariablen werden als Worte übergeben. Bei den 8-Bit-Datentypen »byte«, »boolean« und »char« liegt der Grund dafür in der Tatsache, daß die Prozessoren der 808x-Reihe von Intel nur Worte, nicht aber Bytes auf den Stapel schieben können.

Die Byteinformation findet sich immer im niederwertigen (»unteren«) Byte des auf den Stapel geschobenen Wortes. Die Wahrheitswerte »true« und »false« werden als 1 und 0 codiert. Strings, Mengen, Arrays, Records und Zeiger übergeben Sie als Doppelwortzeiger auf dem Stapel, in der Reihenfolge Offset, Segment. Tabelle 1 faßt die Möglichkeiten der Übergabe von Funktionsargumenten zusammen.

Funktionsergebnisse werden in den Prozessorregistern zurückgeliefert, wenn es sich um »integer«, »char«, »boolean« oder »real« handelt. Alle Mengen, Arrays, Records oder Zeiger erhalten Sie als Zeiger im Registerpaar DX:AX zurück. Tabelle 2

faßt die verwendeten Register für die einzelnen Datentypen des Funktionsergebnisses zusammen. Eine besondere Rolle spielen Strings. Soll eine Funktion als Ergebnis einen String liefern, müssen Sie vor dem Funktionsaufruf einen Datenbereich für die Speicherung des Strings anlegen. Ein Zeiger auf diesen Datenbereich wird dann als erstes Element (vor allen Funktionsargumenten) auf den Stapel geschoben.

In unserem Fall soll ein Strink Funktionsergebnis werden. Daher wird beim Aufruf der Tab-Funktion zunächst ein Zeiger auf den Datenbereich des Strings auf den Stapel geschoben, dahinter die einzelnen Funktionsargumente. Der Tab-Funktion übergeben Sie ein Byte als Call-By-Value-Variable.

Datentyp	wird auf dem Stack übergeben als
byte	Wort (unteres Byte enthält Byteargument)
word	Wort (vorzeichenlos)
integer	Wort (mit Vorzeichen)
char	Wort (unteres Byte enthält Zeichencode)
boolean	Wort (unteres Byte enthält 0 oder 1)
longinteger	Doppelwort (Reihenfolge: niederwertiges Wort, höherwertiges Wort)
real	Dreifachwort
pointer	Doppelwort (Reihenfolge: Offset, Segment)
array	Doppelwortzeiger auf den String-Datenbereich
record	Doppelwortzeiger auf den Record-Datenbereich
set	Doppelwortzeiger auf den Set-Datenbereich

Tabelle 1. Die Formate bei der Übergabe von Daten an eine Turbo-Pascal-Funktion

Funktionsergebnisse werden über die Prozessorregister zurückgegeben

Wie Sie wissen, besteht diese aus einem Wort. Das untere Byte dieses Wortes enthält den Wert des Funktionsarguments. Nach der Übergabe des Funktionsarguments wird die Funktion aufgerufen. Um das Funktionsargument zu ermitteln, müssen Sie den entsprechenden Wert vom Stapel lesen. Eine Funktion darf unter Turbo Pascal alle Register verändern, nur die Werte der Register BP, SS, DS und SP müssen erhalten bleiben. Daher wird innerhalb einer Funktion zunächst BP auf den Stapel gesichert. Mit BP als Zeiger können dann die Funktionsargumente vom Stapel gelesen werden. Zunächst wird SP nach BP übertragen. Dann wird zu BP 4 addiert, da Turbo Pascal ein Wort (nämlich die Rücksprungadresse zum Pascal-Programm) zusätzlich auf den Stapel schiebt und das eigene Programm ein weiteres Wort (also zwei Byte) für das »Retten« von BP belegt. Dann zeigt BP auf das erste Funktionsargument auf dem Stapel. Zu diesem Zweck können Sie die folgende Sequenz benutzen:

```
push bp ;BP retten
mov bp,sp ;BP zeigt auf den
Stapel
```

Bevor die einzelnen Funktionsargumente in die Register geladen werden, sollten Sie beachten, daß der Stapel eine LIFO-Struktur darstellt (»last in first out«). Das heißt, auch wenn die Funktionsargumente im Hinblick auf die Prozedurdeklaration von links nach rechts auf den Stapel geschoben wurden, müssen sie in der umgekehrten Reihenfolge vom Stapel gelesen werden.

Parameter werden über den Stapel ausgetauscht

Bei der Tab-Funktion spielt das keine Rolle, sie liefert nur ein Funktionsargument, nämlich »spalte«. Das ist nach der obigen Assemblersequenz an der Stelle »SS:[BP + 4]« verfügbar. An der Stelle »SS:[BP + 6]« befindet sich dann der Zeiger auf den Datenbereich der Zeichenkette, der als Resultat der Funktion gespeichert werden soll. Das Bild zeigt den Zustand des Stapels vor dem Aufruf der Tab-Funktion. Mit der Kenntnis über die Belegung des Stapels ist es für Sie ein leichtes, das Funktionsargument zu lesen und das Resultat der Funktion zu speichern. Um das Ganze ein wenig zu vereinfachen, werden mit

```
spalte equ byte ptr ss:[bp+4]
resultat equ dword ptr
ss:[bp+6]
```

die symbolischen Namen »spalte« und »resultat« definiert, mit denen das Funktionsargument und -ergebnis adressiert werden.

Was fehlt, ist der eigentliche Rumpf der Tab-Funktion. Die Tab-Funktion soll eine Zeichenkette liefern, die so viele Leerzeichen enthält, daß die Ausgabe des Strings den Cursor an die Bildschirmspalte »spalte« bewegt. Daher wird die aktuelle Bildschirmspalte ausgelesen und dieser Wert von »spalte« abgezogen. Dadurch erhalten Sie die Anzahl der Leerzeichen, die in den String eingefügt werden müssen. Das geschieht mit der folgenden Assemblersequenz:

```
mov ah,15 ;BIOS-Funktion 15
int 10h ;Videostatus auslesen
mov ah,3 ;BIOS-Funktion 3
int 10h ;Cursorposition aus-
lesen
mov al,spalte ;lade SPALTE
nach AL
sub dl,al ;bilde Differenz
dec al ;ziehe eins ab
mov cl,dl ;übertrage sie nach
CL
xor ch,ch ;und lösche CH
les di,resultat ;lade Adresse
des Strings
;nach ES:DI
mov al,cl ;lade Stringlänge
nach AL
cld ;String vorwärts speichern
stosb ;speichere Stringlänge
ab
mov al,' ' ;Leerzeichen sollen
in den String
rep stosb ;speichere die er-
forderliche
;Anzahl ab
```

Durch diesen Programmteil generiert die Tab-Funktion einen entsprechenden String. Dabei wird der vom aufrufenden Programm eingerichtete Zeiger auf den String-Datenbereich verwendet. Als erstes Zeichen eines Strings speichert Turbo Pascal die String-Länge, dahinter folgen die einzelnen Zeichen der Zeichenkette. Um dem Rechnung zu tragen, wird zunächst mit dem Stosb-Befehl die String-Länge gespeichert und später in der Rep-Schleife die erforderliche Anzahl an Leerzeichen hinzugefügt.

Vor Abschluß der Funktion müssen Sie dafür sorgen, daß der Stapel wieder »aufgeräumt« wird. Dazu werden alle Funktionsargumente vom Stapel entfernt. Zuvor wird das zu Beginn der Funktion gesicherte BP-Register vom Stapel geholt. Danach wird die Funktion mit dem Ret-Kommando verlassen, das zusätzlich zwei Byte vom Stapel holt, nämlich die Bytes, die das Funktionsargument »spalte« belegt hat. Die Abschlußsequenz der Tab-Funktion sieht wie folgt aus:

```
pop bp ;hole BP zurück
ret 2 ;und räume Stapel auf
```

Führen Sie die einzelnen Sequenzen zusammen, und setzen Sie sie in das oben vorgestellte Gerüst für die Tab-Funktion ein, so ist die gesamte Funktion fertiggestellt. Listing 1 zeigt den Assemblerquelltext »tab.asm«.

Turbo Pascal kann bei der Compilierung Obj-Dateien einfügen

Um aus der Assemblerprozedur eine Funktion zu machen, die Turbo Pascal bei der Compilierung einbindet, wird der Quelltext mit einem Assembler übersetzt. Ist die Funktion nach der Assemblierung als Datei »tab.obj« auf Diskette oder Festplatte gespeichert, genügt die folgende Anweisung, um die Funktion in einen Pascal-Quelltext einzubeziehen:

```
(*$1 tab *)
function tab(spalte: byte):
string; external;
```

Damit wird die Datei als Funktion »tab« in den Objektcode des Pascal-Programms eingebunden. Die Compiler-Direktive »$1« sorgt dafür, daß die nachstehende Obj-Datei in den Arbeitsspeicher gelesen wird. Dann kann die Tab-Funktion in einem Programm benutzt werden. Listing 2 zeigt einen entsprechenden Quelltext. Mit diesen Ausführungen sollten Sie nun in der Lage sein, selbst Maschinenprogramme in Ihre Pascal-Programme einzubinden. Dabei können die Anzahl und die Datentypen der Parameter und die Funktionsergebnisse variieren. Für die Übergabe der Parameter ist der wichtigste Punkt, daß der zuletzt übergebene Parameter (in der Deklaration) für unser Modell immer an der Stelle »SS:[BP + 4]« auf dem Stapel zu finden ist. Da der Stapel von oben nach unten wächst,

liegen vorher übergebene Parameter oberhalb des letzten Wertes. Um sie zu adressieren, wird zum ersten Zeiger »SS:[BP + 4]« die Länge des letzten Parameters addiert. Dadurch ergibt sich der vorletzte Parameter. Dieser Vorgang läßt sich für alle übergebenen Parameter durchführen. Da das Verständnis dieser Parameterübergabe von fundamentaler Bedeutung für die Entwicklung externer Prozeduren ist, soll die Vorgehensweise nochmals an einem Beispiel verdeutlicht werden. Angenommen, eine Assemblerfunktion »rechne« übernimmt vier Parameter und liefert ein Integer-Funktionsergebnis. Die Funktion ist im Pascal-Quelltext durch die folgende Anweisung deklariert:

```
function rechne(adseg,adofs:
word; anzahl: byte; wert:lon-
ginteger): integer; external;
```

Der in der Deklaration ganz rechts stehende Parameter, »wert«, ist an der Stelle »SS:[BP + 4]« auf dem Stapel verfügbar. Der Parameter »anzahl« befindet sich daher bei »SS:[BP + 8]«, weil zu dem Offset von 4 die 4-Byte-Länge des Datentypes »long integer« des Parameters »wert« addiert werden (aus Tabelle 1). »adofs« befindet sich auf der Stapeladresse »SS:[BP + 10]«, die sich aus der Summe von 8 mit der Länge des Datentyps »byte« ergibt. Der letzte Parameter, »adseg«, befindet sich an der Adresse »SS:[BP + 12]«. Hier erhalten Sie den Zeiger aus dem Offset von 10 plus der Länge des Parameters »adofs«. Für den Zugriff auf die Parameter können Sie im Assemblerprogramm die folgenden Anweisungen verwenden:

```
adseg equ word ptr ss:[bp+12]
adofs equ word ptr ss:[bp+10]
anzahl equ byte ptr ss:[bp+8]
wert equ dword ptr ss:[bp+4]
```

Für die Verwendung in Units müssen Assemblerprogramme leicht modifiziert werden

Maschinenprogramme können in Pascal auch in eine Unit eingebunden werden. Der wichtigste Unterschied liegt darin, daß Sie die Prozedur dann nicht als Near-, sondern als Far-Programm definieren müssen, da Funktionen und Prozeduren aus Units von Turbo Pascal mit »far« aufgerufen werden. Das kommt daher, daß Turbo Pascal für jede Unit ein eigenes Segment definiert und den Programmcode nicht wie beispielsweise bei der Tab-Funktion mit in das Codesegment einbindet. Aus diesem Grund müssen Sie Unit-Funktionen und -Prozeduren mit Aufrufen vom Typ »far« aktivieren, damit die Segmentadresse des aufrufenden Programmsegments nicht verlorengehen kann.
Die kurze Sequenz

```
unit tabulator;
interface
function tab(spalte:
byte):string;
implementation
(*$l tab *)
function tab; external;
end.
```

genügt schon, um die Tab-Funktion als einzige Funktion einer Unit namens »tabulator« zu deklarieren. Voraussetzung dafür ist natürlich, daß Sie im Assemblerquelltext von »tab.asm« das Near-Kommando durch ein Far-Kommando ersetzen, da alle Routinen innerhalb einer Unit Far-Prozeduren sein müssen. Das hat seinerseits die Konsequenz, daß sich die Speicheradressen der Parameter beim Aufruf einer Funktion innerhalb einer Unit ändern. Die Adressen aller Parameter verschieben sich um zwei Byte nach oben, da die Segmentadresse, die zwei Byte benötigt, zusätzlich auf dem Stapel gespeichert wird. Die Parameter der Tab-Funktion stehen in einer Unit an den folgenden Adressen:

```
resultat equ dword ptr
ss:[bp+8]
spalte equ byte ptr ss:[bp+6]
```

Bisher ist die Richtung der Unterprogrammaufrufe in der Betrachtung vorgegeben. Gehen Sie davon aus, daß ein Assemblerprogramm eine Funktion oder Prozedur aufruft, die in Assembler formuliert ist. Für bestimmte Anwendungen ist es jedoch wichtig, von einem Assemblermodul Zugriffsmöglichkeiten auf Strukturen eines Pascal-Programms zu besitzen. Variablen, die im Pascal-Quelltext bereits definiert sind, müssen dann im Assemblermodul nicht erneut Speicherplatz verbrauchen. Auf der anderen Seite bietet auch die Möglichkeit, Pascal-Funktionen und -Prozeduren aus einem Assemblerpro-

BATH BEACH
SOFTWARE-SERVICE

PUBLIC DOMAIN 130

Translator übersetzt englische Programm-Dokumentationen in Deutsch **Lightning-Press** DTP-Desktop-Publishing, Fantastic **As Easy As** sehr gutes Tabellenkalkulationsprogramm, Bestseller **Prophet-Lotto** Statistikprogramm für Lotterieziehungen **3D-Chess** dreidimensionales Schachprogramm **PC-Write** sehr gute Textverarbeitung mit vielen Möglichkeiten **Free-File** sehr gute Datenverwaltung mit vielen Möglichkeiten **Composer** Musik komponieren, Noten schreiben und ausspielen **PC-Akt** erzeugt digitalisierte Aktbilder auf ihrem PC **Dancad-3D** dreidimensionales CAD-Zeichenprogramm mit Animation **PC-KWIK** beschleunigt ihr Diskettenlaufwerk **Mahjongg** sehr gutes chinesisches Spiel mit guter Grafik **Megagopoly** Spiel, wie erreiche ich eine Million **Nuke New York** zeigt einen nuklearen Angriff und die Auswirkungen auf New York **Designer** ein Zeichenprogramm, das bewegliche Bilder erstellt **Black Jack** spielen wie im Casino **Horoskop** erstellt Ihr persönliches Horoskop **Kniffel** Kartenspiel **Mastermind** populäres Strategiespiel **Star-Traders** Händlerspiel, Aktien kaufen im Weltraum **Etikett** erstellt Etiketten in verschiedenen Formaten und druckt sie aus **Flugsimulator** simuliert Jet-Flüge **PC-Calc** populäre Tabellenkalkulation **Copy-All** nun können Sie fast alle geschützten Programme kopieren **PC-Window** PC-Fenstertechnik **Kalender** erstellt einen Kalender für mit Bildschirm/Druck **Printum** schaltet den Drucker in andere Schriftarten um **Bastools** Basic-Tools **Your Menu** erstellt Ihr eigenes Menü zum Selbst-Ausführen **Form-Master** umfangreicher Formular-Generator **DOS-Help** zeigt und erklärt alle DOS-Befehle **Stock Charting** Aktienverwaltung, Kauf-/Verkaufssignal(graf. Darstellung) **Biorythm** Biorythmen **Finance** Finanzen und Berechnungen **Taskplan** Zeitplanungsprogramm **Cover** druckt Diskettenhüllen mit Inhaltsverzeichnis

130 Programme für nur 129.-DM!

VCR-D-Base verwaltet Video-Casetten und druckt Aufkleber **Bookinventory** Inventurprogramm für bis zu 2000 Bücher **PC-Lock** schützt ihre Festplatte mit Passwort **Best Utilities** jede Menge Hilfsprogramme **Cataloging** erstellt einen Diskettenkatalog **FB-Copy** ein sehr schnelles und komfortabeles Kopierprogramm **Virus Check** Computerviren-Untersuchungsprogramm **Backgammon** Backgammonspiel **Space-Invaders** sehr schönes, bekanntes Arcade-Spiel **PC-Foto** Fotoverwaltungsprogramm incl. Druck von Dia-Aufklebern **Packdisk** macht mehr Platz auf Ihrer Festplatte **Compress** Datenkompakter, der Platz auf der Diskette spart **Chasm** Super Assembler-Programmiersprache **plus weitere 80 Public Domain- und Shareware-Programme.**
Hilfs-, Anwendungs- und Spielprogramme (BAS-, COM- und EXE-Startdateien) für IBM PC/XT/AT mit VGA, EGA-, CGA- und Herculeskarte.
Schnellversand am Tag des Bestelleingangs. INFO-Katalog mit über 1.500 Disketten-Programm-Beschreibungen und zusätzlichen Angeboten liegt jeder Bestellung bei.

LIESSFELD & Partner

Bath Beach Software-Service, Speckstraße 23, D-6749 Schweighofen Telefon: (06342)7932

BESTELL-COUPON

Ja, bitte senden Sie mir umgehend das PC-Paket "Public Domain 130" zum Discount-Preis von 129.-DM. Plus Versandkosten DM 6.-DM.
Bei Verrechnungs-Check zzgl. Nachnahme 10.-DM.
Den Gesamtbetrag (Endpreis) bezahle ich wie angekreuzt. (Ausland: Check od. Bar). Zuzüglich erhalte ich **kostenlos** den ausführlichen INFO-Katalog.

Endpreis: ☐ 51/4"Disketten ☐ Scheck/Bar DM 135.- ☐ Per Nachnahme DM 139.-
Endpreis: ☐ 3,5"Disketten ☐ Scheck/Bar DM 160.- ☐ Per Nachnahme DM 164.-
(3,5"Zoll-Disketten Material-Mehrpreis : 25.-)

Name:
Straße:
PLZ/ORT:
Telefon
Datum /Unterschrift Geburtsdatum
(Bei Minderjährigen der Erziehungsberechtigte)

DOEX 11/90

gramm aufzurufen, interessante Anwendungen.
Zunächst soll Ihr Augenmerk dem ersten Aspekt der Problematik gelten: der Verwendung von Variablen des Pascal-Programms in einem Assemblermodul. Hier lassen sich sehr effektive Anwendungen finden. So können Datenbestände in einer Pascal-Prozedur eingelesen und dann in einem schnellen Assemblermodul verarbeitet werden.
Damit es nicht zu theoretisch wird, demonstrieren wir die Methode an einem Beispiel. Es soll eine Assemblerprozedur »suche« formuliert werden, die in einem Byte-Array einen bestimmten Wert sucht und dessen Häufigkeit zählt.

Auch ein Assemblerprogramm kann Pascal-Variablen und -Funktionen verwenden

Das Byte-Array soll als Variable im Pascal-Quelltext definiert sein.
Für die Formulierung des Assemblermoduls gelten die bereits beschriebenen Vorgaben. Die Funktion »suche« sucht einen Bytewert in einem konstanten Array. Sie läßt sich in Turbo Pascal mit der folgenden Anweisung definieren:

```
function suche(wert: byte):
word; external;
```

Die Anweisung »suche« liefert als Funktionsergebnis die Anzahl zurück, in der »wert« im Array auftritt. Das Array wird im Pascal-Quelltext mit der Maximallänge 100 definiert. Die Anzahl der tatsächlich belegten Felder ist in der Variablen »laenge« gespeichert. Für die Ausführung der Funktion muß das Assemblermodul sowohl auf die Variable »laenge« als auch auf die Variable »feld« (das ist das Byte-Array) Zugriff haben. Da beide im Datensegment von Turbo Pascal liegen, müssen Sie im Assemblermodul ein Datensegment definieren. Innerhalb dieses Segments werden die Variablen als »extrn« definiert:

```
data segment word public
extrn feld:byte
extrn laenge:word
data ends
```

Durch diese Anweisungen kann im Assemblerprogramm mit den symbolischen Namen »feld« und »laenge« auf die Variablen im Datensegment von Turbo Pascal zugegriffen werden.
In einer Schleife läßt sich dann jedes Element des Arrays »feld« mit dem übergebenen Parameter »wert« vergleichen. Für die Indizierung des Arrays können Sie ein Indexregister verwenden. Das erste Element des Arrays befindet sich an der Stelle »DS:FELD[0]«.
Listing 3 zeigt das fertige Assemblermodul »suche.asm«. Die schon assemblierte Objektdatei finden Sie in Listing 4. Bei einem Blick auf das Listing wird Ihnen vielleicht eine weitere Besonderheit auffallen: Neben dem Zugriff auf Variablen, die außerhalb des Assemblerquelltextes definiert sind, ist auch der Aufruf ganzer Prozeduren oder Funktionen, die in Pascal formuliert sind, aus dem Maschinenprogramm möglich.

Datentyp	wird zurückgegeben in
byte	Byte im AL-Register
word	Wort im AX-Register
integer	Wort im AX-Register
char	Zeichencode im AL-Register
boolean	0 oder 1 im AL-Register
longinteger	Doppelwort in DX:AX
real	Dreifachwort in DX:BX:AX
pointer	Doppelwort in DX:AX
array	Doppelwortzeiger in DX:AX
record	Doppelwortzeiger in DX:AX
set	Doppelwortzeiger in DX:AX

Tabelle 2. Die Übergabe der Funktionsresultate an ein Turbo-Pascal-Programm

Register auf den Stapel retten

Als Beispiel soll die Assemblerfunktion »suche« immer dann einen Piepton ausgeben, wenn ein Wert überhaupt nicht in der Tabelle enthalten ist. Die Funktion, die den Piepton ausgibt, soll aber nicht in Assembler, sondern in Pascal definiert werden und wie folgt aussehen:

```
procedure piepen;
begin
write(#7);
end;
```

Die Prozedur gibt den ASCII-Code 7 aus, der einen kurzen Piepton erzeugt.
Diese Pascal-Prozedur wird im Assemblermodul »suche« aufgerufen. Um sie anzusprechen, ist eine »extrn«-Anweisung nötig. Im Gegensatz zu den bereits vorgestellten Extrn-Direktiven für die Definition der Pascal-Variablen im Assemblerquelltext muß die Deklaration von »piepen« im Codesegment erfolgen. Hierfür genügt die folgende Anweisung:

```
extrn piepen:near
```

Sie teilt dem Assembler mit, daß die Near-Prozedur »piepen« erst beim Binden des Programms eingefügt wird. Der eigentliche Aufruf der Prozedur »piepen« gestaltet sich unkompliziert. Nach dem »Retten« der Register rufen Sie die Prozedur mit einem Call-Befehl auf.

```
push ax
push bx
push cx
push dx
call piepen
pop dx
pop cx
pop bx
pop ax
```

Als Demonstration der Möglichkeiten zum wechselseitigen Verwenden von Variablen und Modulen in Pascal- und Assemblerprogrammen dient Listing 5. Es verwendet die Assemblerfunktion »suche«, um aus einem zuvor im Pascal-Programm eingelesenen Array die Anzahl eines bestimmten Wertes festzuhalten. Falls der Wert nicht enthalten ist, wird die Prozedur »piepen« aufgerufen, die einen kurzen akustischen Hinweis gibt. Beachten Sie, daß »piepen« zwar im Pascal-Quelltext definiert ist, dort jedoch nicht aufgerufen wird. Ein Aufruf erfolgt direkt aus dem Assemblermodul.
Falls aus einem Assemblerprogramm Pascal-Funktionen oder Prozeduren aufgerufen werden, die eine Parameterübergabe erwarten, muß im Assemblerprogramm dafür gesorgt werden, daß diese Parameter auf den Stapel gelangen.

Um anstelle von Variablen auch Turbo-Pascal-Konstanten in einem Assemblerprogramm verarbeiten zu können, muß in das Assemblermodul keine zusätzliche Segmentdefinition integriert werden, da Konstanten ebenfalls im Datensegment gespeichert sind.

Typisierte Konstanten legt Turbo Pascal im Datensegment ab

Innerhalb dieses Segmentes müssen »extrn«-Definitionen für Konstanten aus dem Turbo-Pascal-Programm erfolgen. Natürlich lassen sich nur typisierte Konstanten ansprechen, also solche, denen ein Typ und ein Wert zugewiesen wird. Ein Beispiel für eine typisierte Konstante ist

```
wert: array[1..5] of byte =
(2,4,8,16,32);
```

»Echte« Konstanten lassen sich nicht ansprechen, da sie der Pascal-Compiler schon während der Übersetzung direkt durch ihre Werte ersetzt.
Neben der Zusammenarbeit mit Assembler besteht auch die Möglichkeit, Module aus anderen Programmiersprachen in Turbo Pascal einzubinden. Gut verträgt sich Turbo Pascal beispielsweise mit seinem »Bruder« Turbo C. Im Prinzip werden Parameter in Turbo C genauso übergeben wie in Turbo Pascal. C-Funktionen oder C-Variablen, die in einem Pascal-Programm Verwendung finden sollen, müssen analog zur Zusammenarbeit mit Assembler als »extrn« definiert werden. Im C-Quelltext haben Sie keine gesonderte Deklaration vorzunehmen, da hier alle Symbole automatisch innerhalb des gleichen Segments ansprechbar sind. Theoretisch können Sie sogar Pascal-, C- und Assemblerfunktionen sowie -Variablen in einem einzigen Programm mischen. Allerdings ist eine solche Konstellation nicht mehr sonderlich sinnvoll, da die Übersichtlichkeit eines entwickelten Programms unter den vielen Sprachen sehr schnell leidet. Nichtsdestotrotz kann eine Kombination von C-, Assembler-, Pascal-, Basic-, Fortran-, Lisp-, Prolog- und Forth-Modulen zu einem einzigen Programm durchaus ihren individuellen Reiz besitzen.

(Martin Althaus/ks)

```
Name.............. TAB.ASM

Sprache......... Masm ab 5.Ø
                 Tasm

Grafikkarte..... jede
```

```
[S85]     1: ;Demo-Programm zum Aufruf einer
[B35]     2: ;Assembler-Funktion aus einem
[3W3]     3: ;Turbo-Pascal-Programm
[P3Ø]     4: ;
[BU3]     5: ;Autor: Martin Althaus
[YØ4]     6: ;(C) 199Ø Redaktionsbüro
[HX3]     7: ;     Everts und Hagedorn
[P3Ø]     8: ;
[PE3]     9: ;Name des Codesgments
[XC4]    1Ø: code      segment byte public
[SW2]    11: ;CS zeigt auf CODE
[EA2]    12:           assume cs:code
[P3Ø]    13: ;
[9Y4]    14: ;Tab-Funktion wird deklariert
[AL1]    15: public tab
[P3Ø]    16: ;
[1F4]    17: spalte    equ byte ptr [bp+4]
[8T4]    18: resultat equ dword ptr [bp+6]
[P3Ø]    19: ;
[ZH6]    2Ø: ;Tab-Funktion muß NEAR-Prozedur sein
[3R2]    21: tab       proc  near
[D74]    22:           ;BP auf dem Stapel retten
[B91]    23:           push  bp
[VH4]    24:           ;Stapel mit SP adressieren
[8W1]    25:           mov   bp,sp
[2S2]    26:           ;BIOS-Funktion 15
[AN1]    27:           mov   ah,15
[534]    28:           ;Videostatus ermitteln
[G81]    29:           int   1Øh
[UM2]    3Ø:           ;BIOS-Funktion 3
[WE1]    31:           mov   ah,3
[TM4]    32:           ;Cursor-Positon ermitteln
[G81]    33:           int   1Øh
[Ø73]    34:           ;lade SPALTE nach AX
[9L2]    35:           mov   al,spalte
[1S2]    36:           ;bilde Differenz:
[GC4]    37:           ;SPALTE-AKTUELLE_SPALTE
[HS1]    38:           sub   al,dl
[L31]    39:           dec   al
[ES3]    4Ø:           ;übertrage sie nach CL
[2U1]    41:           mov   cl,al
[WN2]    42:           ;und lösche CH
[UT1]    43:           xor   ch,ch
[F34]    44:           ;lade Adresse des Strings
[2V1]    45:           ;nach ES:DI
[8U2]    46:           les   di,resultat
[DS3]    47:           ;vorwärts speichern
[ULØ]    48:           cld
[WX3]    49:           ;speichere sie im String
[NXØ]    5Ø:           stosb
[7H5]    51:           ;Leerzeichen sollen in den String
[3L1]    52:           mov   al,' '
[ZY5]    53:           ;speichere die erforderliche Anzahl
[KX1]    54:           rep   stosb
[EB2]    55:           ;restaurieren
[571]    56:           pop   bp
[TD4]    57:           ;und zwei Bytes entfernen
[B21]    58:           ret   2
[U13]    59:           ;Ende der Prozedur
[YR1]    6Ø: tab       endp
[FØ3]    61:           ;Ende des Segments
[TT1]    62: code      ends
[4MØ]    63:           end
```

Listing 1. Die fertige Assemblerfunktion »tab.asm«

```
[AB5]     1: 8Ø Ø9 ØØ Ø7 74 61 62 2E 41 53 4D 2A 96 Ø7 ØØ ØØ
[M65]     2: Ø4 43 4F 44 45 44 98 Ø7 ØØ 28 23 ØØ Ø2 Ø1 Ø1 12
[N55]     3: 9Ø ØA ØØ ØØ Ø1 Ø3 54 41 42 ØØ ØØ ØØ 8B 88 Ø4 ØØ
[1G5]     4: ØØ A2 Ø1 D1 AØ 27 ØØ Ø1 ØØ ØØ 55 8B EC B4 ØF CD
[HR5]     5: 1Ø B4 Ø3 CD 1Ø 8A 46 Ø4 2A C2 FE C8 8A C8 32 ED
[2P5]     6: C4 7E Ø6 FC AA BØ 2Ø F3 AA 5D C2 Ø2 ØØ C5 8A Ø2
[GVØ]     7: ØØ ØØ 74
```

Listing 2. Der Hexdump der assemblierten Tab-Funktion: »tab.obj«

```
[LV2]     1: program tabdemo;
[ØØØ]     2:
[ØK5]     3:  {Demonstriert das Einbinden von
[2Z4]     4:   externen Assembler-Modulen in
[9B2]     5:   Turbo Pascal}
[ØØØ]     6:
[V35]     7:  {Binde Maschinenprogramm ein}
```


Testen mit System

Checkliste – Labortest – Praxistest
Information – Fazit – Gesamturteil

3 überzeugende Pluspunkte der DOS-TEST-Konzeption:

Vergleichbare Testberichte mit einheitlichen Kriterien helfen Ihnen bei der Kaufentscheidung.

Knallharte Information über interessante Themen, verständlich und locker mit viel Lesespaß.

Durch Tests im praktischen Einsatz wissen Sie schon heute, welche Hard- und Software morgen aktuell ist.

Jetzt bei Ihrem Zeitschriftenhändler

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

```
[ØØØ]     8:
[912]     9:  (*$1 tab.obj *)
[ØØØ]    1Ø:
[5H1]    11:  uses crt;
[ØØØ]    12:
[FZ4]    13:  {Hier wird die Assembler-Funk-
[DT2]    14:   tion definiert.}
[ØØØ]    15:
[CR4]    16:  function tab(spalte: byte):
[B13]    17:           string; external;
[ØØØ]    18:
[RNØ]    19:  var
[EZ1]    2Ø:   index: byte;
[ØØØ]    21:
[MUØ]    22:  begin
[DB1]    23:   clrscr;
[KY4]    24:   write('Dieser Text ist ohne ');
[M65]    25:   write('Tabulator geschrieben');
[Y13]    26:   writeln; writeln;
[793]    27:   for index:=1 to 12 do
[KU3]    28:     writeln(tab(5+index*2),
[VV3]    29:       'und dieser Text mit!');
[KNØ]    3Ø:  end.
```

Listing 3. Demoprogramm »tabdemo.pas« zum Aufruf der Tab-Funktion

```
Name............ SUCHE.ASM

Sprache......... Masm ab 5.Ø
                 Tasm

Grafikkarte..... jede
```

```
[PU5]     1: ;Demoprogramm zur wechselseitigen
[V35]     2: ;Verwendung von Variablen und
[TF5]     3: ;Prozeduren in Turbo-Pascal- und
[YM3]     4: ;Assembler-Programmen
[P3Ø]     5: ;
[BU3]     6: ;Autor: Martin Althaus
[R94]     7: ;(C) 199Ø DOS International
[P3Ø]     8: ;
[4E4]     9: data     segment word public
[P3Ø]    1Ø: ;
[UW1]    11: ; Byte-Array
[XS2]    12: extrn feld: byte
[E23]    13: ; Länge des Feldes
[M23]    14: extrn anzahl: word
[P3Ø]    15: ;
[DU1]    16: data     ends
[P3Ø]    17: ;
[XC4]    18: code     segment byte public
[P3Ø]    19: ;
[WD3]    2Ø: ;Auf Segmente zeigen
[KK3]    21: assume cs:code,ds:data
[P3Ø]    22: ;
[D94]    23: ;Funktion wird deklariert
[TZ1]    24: public suche
[LF3]    25: ;Die Pascal-Funktion
[Y23]    26: extrn piepen: near
[ZB2]    27: ;Der Parameter
[X94]    28: wert     equ byte ptr [bp+4]
[P3Ø]    29: ;
[5Z5]    3Ø: ; Funktion muß NEAR-Prozedur sein
[LW2]    31: suche    proc  near
[D74]    32:          ;BP auf dem Stapel retten
[B91]    33:          push  bp
[VH4]    34:          ;Stapel mit SP adressieren
[8W1]    35:          mov   bp,sp
[YF3]    36:          ;lade Vergleichswert
[U22]    37:          mov   al,wert
[M34]    38:          ;Index auf erstes Element
[3W1]    39:          xor   bx,bx
[TM1]    4Ø:          ;Anzahl:=Ø
[HW1]    41:          xor   dx,dx
[SY2]    42:          ;Schleifenzähler
[P13]    43:          mov   cx,ds:anzahl
[1P2]    44:          ;Wert gefunden?
[TD4]    45: schlf:   cmp   ds:feld[bx],al
[SN2]    46:          ;nein, verzweige
[6Y1]    47:          jne   s_ende
[NV2]    48:          ;Anzahl:=Anzahl+1
[G51]    49:          inc   dx
[U63]    5Ø:          ;nächstes Element
[CH2]    51: s_ende:  inc   bx
[LN3]    52:          ;Schleife schließen
[CY1]    53:          loop  schlf
[FN1]    54:          ;Anzahl=Ø?
[XD1]    55:          cmp   dx,Ø
[872]    56:          ;nein, beenden
[BG1]    57:          jne   ende
[CE3]    58:          ;ja, Register retten
```

```
[F91]    59:          push  ax
[L91]    6Ø:          push  bx
[R91]    61:          push  cx
[V91]    62:          push  dx
[RD4]    63:         ;Pascal-Funktion aufrufen
[AY1]    64:          call  piepen
[ZZ3]    65:         ;Register zurückholen
[N71]    66:          pop   dx
[H71]    67:          pop   cx
[D71]    68:          pop   bx
[971]    69:          pop   ax
[LG2]    7Ø:         ;Suche:=Anzahl
[LR2]    71: ende:    mov   ax,dx
[3X2]    72:         ;BP restaurieren
[571]    73:          pop   bp
[M24]    74:         ;und zwei Byte entfernen
[B21]    75:          ret   2
[U13]    76:         ;Ende der Prozedur
[EX1]    77: suche    endp
[FØ3]    78:         ;Ende des Segments
[TT1]    79: code     ends
[4MØ]    8Ø:          end
```

Listing 4. Die Funktion »suche.asm« greift auf Pascal-Variablen und -Prozeduren zurück

```
[8F5]     1: 8Ø ØB ØØ Ø9 73 75 63 68 65 2E 41 53 4D 45 96 ØC
[N65]     2: ØØ ØØ Ø4 44 41 54 41 Ø4 43 4F 44 45 21 98 Ø7 ØØ
[V65]     3: 48 ØØ ØØ Ø2 Ø1 Ø1 15 98 Ø7 ØØ 28 2E ØØ Ø3 Ø1 Ø1
[9C5]     4: Ø6 8C 17 ØØ Ø4 46 45 4C 44 ØØ Ø6 41 4E 5A 41 48
[9C5]     5: 4C ØØ Ø6 5Ø 49 45 5Ø 45 4E ØØ B3 9Ø ØC ØØ ØØ Ø2
[U75]     6: Ø5 53 55 43 48 45 ØØ ØØ ØØ E5 88 Ø4 ØØ ØØ A2 Ø1
[ND5]     7: D1 AØ 32 ØØ Ø2 ØØ ØØ 55 8B EC 8A 46 Ø4 33 DB 33
[VA5]     8: D2 8B ØE ØØ ØØ 38 87 ØØ ØØ 75 Ø1 42 43 E2 F6 83
[8K5]     9: FA ØØ 75 ØB 5Ø 53 51 52 E8 ØØ ØØ 5A 59 5B 58 8B
[TC5]    1Ø: C2 5D C2 Ø2 ØØ 4F 9C 1Ø ØØ C4 ØC Ø6 Ø1 Ø2 C4 1Ø
[5G4]    11: Ø6 Ø1 Ø1 84 22 Ø6 Ø2 Ø3 EE 8A Ø2 ØØ ØØ 74
```

Listing 5. Die Objektdatei »suche.obj« als Hexdump

```
[L63]     1: program suchedemo;
[ØØØ]     2:
[XP5]     3: {Demonstriert die wechselseitige
[UR4]     4:  Verwendung von Variablen und
[9S4]     5:  Funktionen in Assembler und
[ØF4]     6:  Turbo Pascal-Programmen }
[ØØØ]     7:
[5H1]     8:  uses crt;
[ØØØ]     9:
[RNØ]    1Ø:  var
[3A3]    11:   anzahl,laenge: word;
[7D4]    12:   feld: array[1..1ØØ] of byte;
[8Z2]    13:   index,wert: byte;
[ØØØ]    14:
[Y92]    15:   (*$1 suche.obj *)
[ØØØ]    16:
[4Y3]    17:  function suche(x: byte):
[8P2]    18:            word; external;
[ØØØ]    19:
[PY2]    2Ø:  procedure piepen;
[MUØ]    21:   begin
[4F1]    22:     write(#7);
[ZNØ]    23:   end;
[ØØØ]    24:
[MUØ]    25:  begin
[DB1]    26:   clrscr;
[B95]    27:   {Werte einlesen und Assembler-
[BE3]    28:    Funktion aufrufen}
[LZ3]    29:   write('Wieviele Werte ');
[BP4]    3Ø:   write('sollen verarbeitet ');
[9R2]    31:   write('werden? ');
[LG2]    32:   readln(anzahl);
[XF1]    33:   writeln;
[S54]    34:   for index:=1 to anzahl do
[MUØ]    35:    begin
[2F4]    36:     write(index,'.ter Wert: ');
[LB3]    37:     readln(feld[index]);
[ZNØ]    38:    end;
[XF1]    39:   writeln;
[XD4]    4Ø:   write('Welcher Wert soll ');
[WA4]    41:   write('gezählt werden? ');
[352]    42:   readln(wert);
[XF1]    43:   writeln;
[M15]    44:   write('Ich suche nun den Wert ');
[X45]    45:   writeln(wert,' in der Tabelle.');
[XF1]    46:   writeln;
[T24]    47:   write('Ich habe ihn ins');
[4H4]    48:   write('gesamt ',suche(wert));
[364]    49:   writeln(' mal gefunden.');
[KNØ]    5Ø:  end.
```

Listing 6. Demoprogramm »suchdemo.pas« für den Aufruf der Suchefunktion

Kurze Unterbrechung bitte

Aufruf von Interrupts unter Turbo Pascal

Eine interessante Form der maschinennahen Programmierung unter Turbo Pascal liegt in der Möglichkeit, Interrupts und Funktionen des Betriebssystems aufzurufen. Dies versetzt den Programmierer in die Lage, nahezu alle Fähigkeiten des Betriebssystems auch unter Pascal zu nutzen.

Jeder Computer verfügt bekanntlich über ein Betriebssystem, ohne das der Rechner nicht funktionieren würde. Die wichtigste Aufgabe des Betriebssystems liegt darin, Funktionen und Prozeduren zur Verfügung zu stellen, mit denen sich die Hardware-Einheiten ansteuern lassen. Im PC besteht das Betriebssystem zum einen aus dem geräteabhängigen ROM-BIOS und zum anderen aus MS-DOS. Sowohl MS-DOS als auch das BIOS bieten eine große Anzahl unterschiedlicher Funktionen an, mit denen sich die einzelnen Komponenten des PC ansprechen lassen. Das Spektrum reicht über Routinen zur Ansteuerung von Tastatur, Bildschirm und Speichermedien bis hin zu Datums- und Zeitfunktionen. Die einzelnen Funktionen von BIOS und MS-DOS werden über »Interrupts« aufgerufen.

Ein solcher Interrupt ist ein sogenannter »Software-Interrupt«, weil er nicht von einem Peripheriegerät oder einem angeschlossenen Baustein, sondern von einem Programm ausgelöst wird. Ein spezieller Maschinenbefehl der Prozessoren der Intel-Reihe (8086, 8088, 80286, 80386 und 80486) löst einen Software-Interrupt aus. Dieser Befehl übergibt eine Interrupt-Nummer an den Prozessor. Der Prozessor sieht daraufhin in einer Tabelle nach und findet eine zu der Interrupt-Nummer gehörige Adresse im Hauptspeicher. An dieser Speicheradresse wird anschließend ein Unterprogramm ausgeführt. Im Prinzip handelt es sich bei einem Software-Interrupt also um einen Unterprogrammaufruf. Nach Abarbeitung des Unterprogramms wird das unterbrochene Programm an der auf den Software-Interrupt folgenden Stelle weitergeführt.

Das Besondere an den Software-Interrupts ist, daß sie bei PCs weitgehend standardisiert sind. Egal, von welchem Hersteller der PC gefertigt ist, die Software-Interrupts funktionieren in der Regel auf allen Kompatiblen. Tabelle 1 faßt die wichtigsten Software-Interrupts auf PC und AT zusammen. Die unterschiedlichen Interrupts werden häufig auch BIOS- und MS-DOS-Interrupts genannt. Alle Interrupts ab 21hex sind MS-DOS-Interrupts, die Unterbrechungen mit kleineren Nummern sind BIOS-Interrupts.

Mit dem Interrupt 21hex bietet MS-DOS eine mächtige Funktionsbibliothek

In der Unit »dos« bietet Turbo Pascal die Befehle »msdos« und »intr« an. Mit »msdos« läßt sich speziell der Interrupt 21hex aufrufen, mit »intr« jeder beliebige Interrupt. Für den korrekten Einsatz beider Kommandos ist das Verständnis des Unterbrechungskonzeptes des PC unerläßlich.

Die mächtigste Sammlung von Befehlen stellt der MS-DOS-Funktionsverteiler, der Interrupt 21hex, zur Verfügung. Er bietet eine wahre Fülle von Unterprogrammen. Diese Interrupts erwarten – genau wie die BIOS-Interrupts – in der Regel eine Reihe von Parametern, die über die Prozessorregister übergeben werden. In anderen Fällen werden Funktionsergebnisse in den Registern an das aufrufende Programm zurückgegeben.

Es ist unter Turbo Pascal nicht so ohne weiteres möglich, Zuweisungen an die Prozessorregister vorzunehmen oder deren Werte auszulesen. Speziell für die Zusammenarbeit mit Interrupts bietet Turbo Pascal jedoch eine Datenstruktur an, mit der sich genau diese Aufgabe erfüllen läßt. In Turbo Pascal ab Version 4.0 ist sie in der Unit »dos« als Datentyp »registers« vordefiniert, für ältere Pascal-Versionen muß sie mit der Sequenz

```
type registers = record
case boolean of
true: (ax,bx,cx,dx,bp,si,di,
ds,es,flags:integer);
false: (al,ah,bl,bh,cl,ch,dl,
dh: byte);
end;
```

definiert werden. Wie man an der Deklaration erkennen kann, handelt es sich beim Registerdatentyp um einen Record. Alle Elemente des Datentyps, also die Register AX bis ES, die Register AL bis DH und das Kennzeichenregister, werden als Elemente einer Registervariablen angesprochen. Um eine Funktion des Betriebssystems aufzurufen, wird eine Variable für den Zugriff auf die CPU-Register über den Datentyp »registers« deklariert. Das geschieht beispielsweise mit der Anweisung

```
var
cpu: registers;
```

Nach dieser Deklaration ist jedes Register des Prozessors einzeln adressierbar. Soll beispielsweise das Register

Interrupt	Aufgabe
10hex	Steuerung des Bildschirms im Grafik- und Textmodus
11hex	Systemkonfiguration lesen
12hex	Hauptspeichergröße lesen
13hex	Disketten- und Festplattenzugriff
14hex	serielle Schnittstelle
15hex	Kassettenzugriff und AT-spezifische Funktionen
16hex	Tastaturabfrage
17hex	parallele Schnittstelle
18hex	ROM-Basic starten
19hex	System booten
1Ahex	Systemuhrzugriff
21hex	MS-DOS-Funktionen
25hex	direktes Schreiben auf Diskette und Festplatte
26hex	direktes Lesen von Diskette und Festplatte

Tabelle 1. Die wichtigsten Interrupts

Byte	Inhalt
00hex	Identifizierungscode. Hat den Inhalt ASCII 77 ("M"), wenn weitere MCBs folgen, ansonsten den Inhalt ASCII 90 ("Z")
01hex	Low-Byte der Adresse des PSP im zugehörigen Programm
02hex	High-Byte der Adresse des PSP im zugehörigen Programm
03hex	Low-Byte der Größe des Speicherblocks in 16-Byte-Abschnitten
04hex	High-Byte der Größe des Speicherblocks in 16-Byte-Abschnitten
05hex – 10hex	reserviert

Tabelle 2. Der Aufbau eines MCB

Register	Bedeutung
Eingabe	
AH	135
CX	Anzahl der zu verschiebenden Worte
ES	Segmentadresse einer GDT
SI	Offset-Adresse einer GDT
Ausgabe	
Carry	zeigt Fehler an
AX	Fehlercode

Tabelle 3. Die Parameter der Funktion 87hex von Interrupt 15hex

Register	Bedeutung
Eingabe	
AH	136
Ausgabe	
AX	Größe des Extended Memory in KByte

Tabelle 4. Die Parameter der Funktion 88hex von Interrupt 15hex

AX benutzt werden, wird die Variable »cpu.ax« verwendet, beim CL-Register läßt sich das über die Variable »cpu.cl« bewerkstelligen. Über diese Prozessorvariable können Zuweisungen in beide Richtungen vorgenommen werden. Den Funktionen des Betriebssystems können vor dem Aufruf Parameter in den Registern übergeben werden, oder es lassen sich nach dem Aufruf Ergebnisse in den Registern auslesen.

Zum Aufruf eines Interrupts dienen unter Turbo Pascal die Kommandos »msdos» und »intr«. Die Syntax beider Befehle ist:

```
msdos(<Register-Variable>)
intr(<Interrupt-Nummer>,
<Registervariable>);
```

Beide Funktionen lösen einen Software-Interrupt aus. Die Anweisung »msdos« verwendet automatisch den MS-DOS-Funktionsverteiler 21hex, während der Intr-Befehl alle Interrupts aufrufen kann.

Zum Einrichten von Datenblöcken können Strings und Arrays verwendet werden

Häufig müssen den Funktionen des Betriebssystems Datenpuffer oder ähnliches übergeben werden. Meist werden dazu die Segment- und die Offset-Adresse des Speicherbereichs verlangt. Diese Datenpuffer lassen sich als String-Variablen definieren.

Die Adresse wird dann mit den Turbo-Pascal-Kommandos »ofs« und »seg« ermittelt. Dabei ist zu beachten, daß Turbo Pascal als erstes Byte eines Strings dessen Länge speichert. Dies erübrigt eine Ende-Markierung des Strings.

Ein Beispiel soll das bisher Dargestellte anschaulich erläutern. Funktion 09hex von Interrupt 21hex gibt einen String auf dem Bildschirm aus. Die Funktion wird ausgewählt, indem ein Wert von 9 in das AH-Register geschrieben und der Interrupt 21hex aufgerufen wird. Die Adresse des Strings erwartet die Funktion im Registerpaar DS:DX. Das DS-Register enthält die Segment- und das ES-Register die Offset-Adresse des Strings. Außerdem muß das Ende des Strings durch ein Dollarzeichen markiert sein. Soll beispielsweise der Text »Ich kenne kein Write« über Interrupt 21hex ausgegeben werden, kann das wie folgt geschehen:

```
program textaus;
uses dos;
var
atext: string;
cpu: registers;
begin
atext:='Ich kenne kein
Write$';
cpu.ah:=9;
cpu.ds:=seg(atext[1]);
cpu.dx:=ofs(atext[1]);
msdos(cpu);
end.
```

Zu beachten ist hier, daß als Argument der Befehle »ofs« und »seg« nicht »atext«, sondern »atext[1]« verwendet wird. Diese Konstruktion ist notwendig, damit das Längenbyte, das Turbo Pascal als erstes Zeichen eines Strings speichert, übergangen wird. Anstelle von »ofs(atext[1])« ließe sich auch »ofs(atext) + 1« schreiben.

Die Intr-Funktion arbeitet ähnlich wie »msdos«, ist jedoch nicht an den Interrupt 21hex gebunden. Hier kann mit dem Parameter »Interrupt-Nummer« die Nummer eines Interrupts angegeben werden, der aufgerufen werden soll. Mit der Intr-Anweisung lassen sich demnach auch BIOS-Interrupts aufrufen.

Wie sinnvoll sich Interrupts in eigenen Turbo-Pascal-Programmen einsetzen lassen, zeigt das Programm »zeigmcbs.pas« in Listing 1. Das Programm nutzt eine undokumentierte MS-DOS-Funktion, um die gesamte Belegung des Hauptspeichers darzustellen.

Um die Übersicht über belegte und noch freie Speicherblöcke zu behalten, haben sich die Entwickler von MS-DOS den sogenannten »Memory Control Block« (MCB) einfallen lassen. Ein MCB ist eine Datenstruktur, die präzise Auskünfte über jeden verwalteten Speicherblock enthält.

Jeder MCB steht direkt vor dem von ihm kontrollierten Speicherbereich. Er enthält Informationen darüber, wieviel Platz der nachfolgende Speicherblock beansprucht und wo sich der nächste MCB befindet. Insgesamt belegt ein MCB 16 Byte. Die Belegung ist in Tabelle 2 dargestellt.

Durch die »Memory Control Blocks« erkennt MS-DOS, welche Speicherbereiche besetzt sind

Wichtig ist die Identifikation eines MCB im Speicher. Wenn dem MCB mindestens ein weiterer MCB folgt, enthält das Identifikationsbyte des MCB den Buchstaben »M«, ansonsten den Buchstaben »Z«. Das nächste Feld, die Adresse des PSP (»Program Segment Prefix«), ist nur von Interesse, wenn mit dem MCB ein Environment verwaltet wird. Wichtiger ist das folgende Feld, das die Größe des hinter dem MCB-belegten Speichers in 16-Byte-Abschnitten wiedergibt. Hieraus kann ein Zeiger auf den nächsten MCB ermittelt werden, sofern der aktuelle MCB nicht ein »Z« als Identifikation trägt, denn dann ist er der letzte MCB in der Kette.

Anhand der MCB kann MS-DOS also sehr präzise ermitteln, welche Speicherblöcke belegt sind und wo sich noch freier Platz befindet. Daher ist es interessant, sich die Belegung des Speichers durch die MCB anzeigen zu lassen.

Das Problem dieser Aufgabe besteht darin, die Adresse des ersten MCB zu ermitteln. Hier ist Funktion 52hex von Interrupt 21hex hilfreich. Diese – offiziell nicht dokumentierte – Funktion liefert als Resultat die Adresse des sogenannten »DOS-Information-Block« (DIB) im Registerpaar ES:BX zurück. Neben einigen anderen hochinteressanten Informationen, in die der DIB Einblick gewährt, findet man dort an der Offset-Adresse -4 auch einen Zeiger auf den ersten MCB. Wird dieses Feld ausgelesen, ist es einfach, die gesamte Kette der MCBs und der dazugehörigen Speicherblöcke zu untersuchen. Das Programm »zeigmcbs« (Listing 2) erledigt diese Aufgabe. Es stellt alle MCBs, ihre Adressen und Längen in einer Tabelle dar.

Das ist ein Anwendungsbeispiel für ein Turbo-Pascal-Programm, das sich nur über die Möglichkeit zum Aufruf von Interrupts realisieren läßt.

Ein anderes Beispiel ist der Zugriff auf das Extended Memory des AT. Dieser Speicher liegt außerhalb des normalerweise ansprechbaren konventionellen Arbeitsspeichers und kann nur auf Computern mit den Prozessoren 80286, 80386 oder 80486 angesprochen werden, indem der Prozessor in den »Protected Mode« umgeschaltet wird. Turbo Pascal offeriert

keine Möglichkeit, diesen Speicher in eigenen Programmen anzusteuern. Das AT-BIOS hingegen bietet zwei Funktionen, über die ein Zugriff auf Extended Memory realisierbar ist: die Funktionen 87hex und 88hex von Interrupt 15hex. Tabelle 3 und 4 stellen die Parameter dieser Funktionen zusammen. Mit Funktion 88hex kann die Größe des Extended Memory festgestellt werden, Funktion 87hex erlaubt die Verschiebung eines Speicherblocks aus dem konventionellen Speicher in das Extended Memory oder umgekehrt.

Der Zugriff auf Extended Memory erfolgt im Protected Mode

Während Funktion 88hex noch relativ einfach zu handhaben ist, setzt die Anwendung von Funktion 87hex ausführliche Kenntnisse der Arbeitsweise des Protected Mode voraus, da diese Funktion eine »Global Descriptor Table« (GDT) erwartet. Das ist eine Speicherstruktur, die im Protected Mode Schutzrechte beim Zugriff auf den Systemspeicher erlaubt oder verwehrt. Das folgende kurze Listing zeigt, welches Format eine GDT besitzen muß, damit sie von Funktion 87hex akzeptiert wird:

```
var gdt:string;
gdt:= #0+#0+#0+#0+#0+#0
      +#0+#0+#0+
      #0+#0+#0+#0+#0+#0
      +#0+#255+
      #255+
      #0+#0+#03+ {Quell-
      adresse}
      #97+#0+#0+#255+#
      255+
      #0+#0+#10+ {Ziel-
      adresse}
      #97+#0+#0+#0+#0+#
      0+#0+#0+
      #0+#0+#0+#0+#0+#0
      +#0+#0+
      #0+#0+#0;
```

Das mag auf den ersten Blick sehr kompliziert aussehen, ist jedoch im Prinzip einfach anzuwenden. An den Stellen, an denen der Kommentar »Quelladresse« beziehungsweise »Zieladresse« zu finden ist, wird die Adresse eingetragen, von der aus ein Datenblock übertragen werden soll. Das Feld »Zieladresse« gibt an, wohin die Daten zu speichern sind. Bei beiden Adressen handelt es sich um die physikalischen Speicheradressen mit 24-Bit-Adreßraum. Sie liegen in der Reihenfolge niederwertiges, mittelwertiges und höherwertiges Wort vor.
In der obigen Deklaration wird als Quelladresse die Adresse 030000hex verwendet. Das ist in gewohnter Intel-Schreibweise die Adresse 3000:0000hex, also eine Speicheradresse im konventionellen Speicher. Als Zieladresse dient 10000hex (10000:0000hex in Intel-Schreibweise). Anders ausgedrückt ist dies das erste Byte oberhalb der 1-MByte-Grenze F000:FFFFhex. Durch Anpassung dieser Werte an die eigenen Gegebenheiten können Speicherblöcke in das Extended Memory und zurück kopiert werden. Dazu muß die Adresse des GDT an Funktion 87hex von Interrupt 15hex übergeben werden. Das geschieht durch die Sequenz:

```
cpu.es:=seg(gdt[1]);
cpu.si:=ofs(gdt[1]);
```

Im AX-Register ist festzulegen, wie viele Worte zu verschieben sind. Danach kann Funktion 87hex aktiviert werden. Wenn Sie auf das Extended Memory in eigenen Programmen nicht verzichten möchten, können Sie mit Listing 2 die Größe des Speichers feststellen. Wie dieser Artikel zeigt, lassen sich über Interrupt-Aufrufe unter Turbo Pascal viele Funktionen verwirklichen, für die ansonsten Maschinensprache benötigt wird. Sie sollten sich mit genügend Dokumentation eindecken, um die mehr als reichhaltigen Interrupt-Funktionen Ihres Rechners zu Ihrer Arbeitserleichterung einsetzen zu können. Was wäre Turbo Pascal ohne »intr« und »msdos«?

(Martin Althaus/dm)

Sesam öffne dich!

BCi DisAsm erschließt jedes Programm!

Wandelt den Programmcode in ein dokumentiertes Assemblerlisting um. Mit normalen Editoren können Sie nun verändern und modifizieren, und mit gängigen Assemblern und Debuggern kann danach wieder assembliert werden.

Es gibt kaum einen, der BCi DisAsm das Wasser reichen kann!

BCi DisAsm I86
Intelligenter Disassembler Version 1.0

Auch Experten kennen nicht immer die Lösung, aber sie wissen, wo sie suchen müssen.
BCi DisAsm hilft JEDEM, die verwirrenden und frustrierenden Software-Puzzles, welche sich beim Umgang mit dem Computer ergeben, zu lösen. Und zwar schneller als mit jedem anderen Produkt, das Sie kaufen können, egal, um welche Preisklasse es sich handelt.
Mit dem BCi DisAsm erwerben Sie sofort ein Know-how, daß Sie normalerweise Jahre an Erfahrung kosten würde. Versuchen Sie nicht, das Rad neu zu erfinden!

BCi DisAsm – Der intelligente Disassembler

BCi DisAsm ist ein intelligenter, selbsttätig dokumentierender MS-DOS-Disassembler. Die ausgefeilten Routinen im BCi DisAsm analysieren in kürzester Zeit selbst komplexen und umfangreichen Programmcode und zeigen dabei potentielle Problempunkte auf.

BCi DisAsm – Das Lernmittel

Sie brauchen kein Spezialist in Assembler-Programmierung zu sein. BCi DisAsm produziert aus jedem beliebigen Programm ihres IBM-kompatiblen Computers ein gut dokumentiertes, leicht verständliches Assemblerlisting.

BCi DisAsm – Einfach in der Handhabung

BCi DisAsm kann sowohl über Parameter von der DOS-Kommandozeile aus als auch interaktiv über Menüs gesteuert werden. Die Menüstruktur ist Lotus-kompatibel und selbstverständlich in Deutsch. Es besteht keine Notwendigkeit, neue Befehle zu lernen oder ständig im Handbuch nachzuschlagen. BCi DisAsm ist unmittelbar anwendbar!

Minimale Systemanforderungen
256 kByte RAM, 8088/8086/80186/80286 oder 80386 CPU, MS-DOS oder PC-DOS Version 2.0 oder größer

BCi DisAsm wird Ihnen bei den folgenden Schritten helfen:

1. Lernen Sie Assembler-Programmierung, wenn Sie möchten.
2. Entdecken Sie, warum Programm XYZ auf Ihrem Rechner nicht läuft, und tun Sie etwas dagegen.
3. Ändern oder entfernen Sie Programmteile.
4. Sichern Sie sich gegen destruktive Programme ab. Finden Sie einen "Virus", und entfernen Sie ihn.
5. Integrieren Sie Code aus compilierten Programmen in eigene Anwendungen.
6. Erhöhen Sie die Kompatibilität Ihrer Software.
7. Passen Sie Software an verschiedene MS-DOS-Versionen an.

© 1987 BCi, West Germany und USA
Lotus ist eingetragenes Warenzeichen der Lotus Corp.
MS-DOS ist eingetragenes Warenzeichen der Microsoft Corp.

BCi DisAsm kostet inkl. eines sehr ausführlichen deutschen Handbuchs (mehr als 100 Seiten) nur

199,– DM (unverbindliche Preisempfehlung)
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		Ausland:	
Einzelpreis	199,– DM	Einzelpreis	199,– DM
zzgl. Versandkosten	4,– DM	zzgl. Versandkosten	6,– DM
Endpreis	**203,– DM**	**Endpreis**	**205,– DM**

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

```
[PZ2]     1: program zeigmcbs;
[ØØØ]     2:
[445]     3: {Stellt die Kette der MCBs auf
[XØ3]     4:  dem Bildschirm dar
[ØØØ]     5:
[UT3]     6:  von Martin Althaus 2/9Ø
[ØØØ]     7:
[4Ø4]     8:  (C) 199Ø Redaktionsbüro
[L83]     9:      Everts und Hagedorn
[U7Ø]    1Ø: }
[ØØØ]    11:
[S62]    12: uses dos,crt;
[STØ]    13: type
[WK2]    14:  zeiger = record
[BV2]    15:              sgmnt,ofst:word;
[ZNØ]    16:             end;
[GL3]    17:  vierstr = string[4];
[444]    18:  einsstring = string[1];
[RNØ]    19: var
[YT2]    2Ø:  cpu: registers;
[NG3]    21:  taste,mcb_code: char;
[WD3]    22:  dib_adr,mcb: zeiger;
[FB4]    23:  mcb_nummer,groesse: word;
[ØØØ]    24:
[HE5]    25: {Funktion zur Umwandlung einer
[P35]    26:  Zahl in eine vierstelige Hexa-
[P72]    27:  dezimalzahl}
[ØØØ]    28:
[D95]    29: function make_hex(wert: word):
[KG1]    3Ø:   vierstr;
[6XØ]    31: const
[8F5]    32:  hex_ziffern:array[Ø..15] of char
[SA3]    33:  = ('Ø','1','2','3','4','5','6',
[G53]    34:     '7','8','9','A','B','C','D',
[Ø21]    35:     'E','F');
[RNØ]    36: var
[9E2]    37:  help: vierstr;
[EZ1]    38:  index: byte;
[C82]    39:  teiler: word;
[MUØ]    4Ø: begin
[3T1]    41:  help:='ØØØØ';
[9Z3]    42:  for index:=3 downto Ø do
[MUØ]    43:   begin
[A75]    44:    teiler:=wert shr (index shl 2);
[FS3]    45:    wert:=wert-(teiler shl
[B72]    46:               (index shl 2));
[YK2]    47:    help[4-index]:=
[MT3]    48:    hex_ziffern[teiler];
[ZNØ]    49:   end;
[MP2]    5Ø:  make_hex:=help;
[ZNØ]    51: end;
[ØØØ]    52:
[AB5]    53: {Funktion zum Einrichten eines
[8V3]    54:  variablen Tabulators}
[ØØØ]    55:
[5V4]    56: function tab(position:byte):
[Ø42]    57:   einsstring;
[MUØ]    58: begin
[R84]    59:  gotoxy(position,wherey);
[L31]    6Ø:  tab:='';
[ZNØ]    61: end;
[ØØØ]    62:
[5V5]    63: {Hauptprogramm: verfolgt die MCBs
[ØE5]    64:  und stellt ihren Inhalt auf dem
[7N2]    65:  Bildschirm dar.}
[ØØØ]    66:
[MUØ]    67: begin
[DB1]    68:  clrscr;
[XF1]    69:  writeln;
[VK2]    7Ø:  writeln(tab(26),
[GF3]    71:  '----------------------------');
[VK2]    72:  writeln(tab(26),
[G14]    73:  '-----   ZEIGMCBS V1.Ø  -----');
[VK2]    74:  writeln(tab(26),
[794]    75:  '---- von M. Althaus -9Ø ----');
[VK2]    76:  writeln(tab(26),
[CN4]    77:  '--- (C) DOS International --');
[VK2]    78:  writeln(tab(26),
[GF3]    79:  '----------------------------');
[XF1]    8Ø:  writeln;
[1V1]    81:  cpu.ah:=$52;
[XV1]    82:  msdos(cpu);
[ØZ3]    83:  dib_adr.sgmnt:=cpu.es-1;
[5X3]    84:  dib_adr.ofst:=cpu.bx+12;
[MP1]    85:  mcb.ofst:=
[CS5]    86:   mem[dib_adr.sgmnt:dib_adr.ofst]+
[E25]    87:   256*mem[dib_adr.sgmnt:dib_adr.
[8D1]    88:   ofst+1];
[ØZ1]    89:  mcb.sgmnt:=
[TG5]    9Ø:   mem[dib_adr.sgmnt:dib_adr.ofst+
[E44]    91:   2]+256*mem[dib_adr.sgmnt:
[WT2]    92:   dib_adr.ofst+3];
[YC2]    93:  mcb_nummer:=1;
[STB]    94:  write('╔═══════╦═══════╗');
[ØUB]    95:  write('╠═══════╬═══════╣');
[4UB]    96:  write('╚═══════╩═══════╝');
```

```
[V59]    97:  writeln('═══════════╗');
[K45]    98:  write('║  Nummer  ║  Code  ');
[FD4]    99:  write('  Adresse des      ');
[HH4]   1ØØ:  write('  Größe des        ');
[WF5]   1Ø1:  writeln('  Größe des      ');
[TN4]   1Ø2:  write('  des      ║  des   ');
[XX4]   1Ø3:  write('  Speicherblocks   ');
[XX4]   1Ø4:  write('  Speicherblocks   ');
[Y85]   1Ø5:  writeln('  Speicherblocks ');
[UX4]   1Ø6:  write('  MCBs     ║  MCBs  ');
[3C3]   1Ø7:  write('                   ');
[ZG4]   1Ø8:  write('  (Paragraphen)    ');
[RG4]   1Ø9:  writeln('  (Bytes)        ');
[HFB]   11Ø:  write('╠═══════╬═══════');
[LFB]   111:  write('╠═══════╬═══════');
[HFB]   112:  write('═══════╬═══════');
[6DA]   113:  writeln('═══════╝');
[P31]   114:  repeat
[1T3]   115:   write('  ',mcb_nummer);
[XN1]   116:   mcb_code:=
[A25]   117:   char(mem[mcb.sgmnt:mcb.ofst]);
[AE4]   118:   write(tab(12),'  ',mcb_code);
[ST2]   119:   write(tab(21),'  ',
[6Ø4]   12Ø:    make_hex(mcb.sgmnt+1),':',
[C74]   121:    make_hex(mcb.ofst),'hex');
[CL1]   122:   groesse:=
[G35]   123:    (mem[mcb.sgmnt:mcb.ofst+3]+256*
[EM4]   124:    mem[mcb.sgmnt:mcb.ofst+4]);
[PC4]   125:   write(tab(4Ø),'  ',groesse);
[CU2]   126:   write(tab(59),'  ',
[T14]   127:    longint(groesse) shl 4);
[ZG3]   128:   writeln(tab(77),'');
[XF1]   129:   writeln;
[ØM2]   13Ø:   write('<Taste>');
[KN2]   131:   taste:=readkey;
[YY2]   132:   gotoxy(1,wherey);
[DH2]   133:   writeln('       ');
[C24]   134:   gotoxy(wherex,wherey-2);
[G95]   135:   mcb.sgmnt:=mcb.sgmnt+groesse+1;
[4A4]   136:   mcb_nummer:=mcb_nummer+1;
[993]   137:  until mcb_code<>'M';
[KTB]   138:  write('═══════╩═══════╝');
[XTB]   139:  write('═══════════════');
[3UB]   14Ø:  write('═══════╦═══════');
[Z59]   141:  writeln('═══════╝');
[KNØ]   142: end.
[ØØØ]   143:
```

Listing 1. Die Aufschlüsselung der Speicherblöcke (»zeigmcbs.pas«)

```
[413]     1: program checkxms;
[ØØØ]     2:
[F36]     3:  {Ermittelt die Größe des erwei-
[5S4]     4:   terten Speichers beim AT oder
[5H3]     5:   einem PS/2 ab Modell 3Ø.
[ØØØ]     6:
[UT3]     7:   von Martin Althaus 2/9Ø
[ØØØ]     8:
[4Ø4]     9:   (C) 199Ø Redaktionsbüro
[L83]    1Ø:       Everts und Hagedorn
[U7Ø]    11:  }
[ØØØ]    12:
[VG1]    13: uses dos;
[ØØØ]    14:
[RNØ]    15: var
[YT2]    16:  cpu: registers;
[ØØØ]    17:
[MUØ]    18: begin
[Y13]    19:  writeln; writeln;
[5W1]    2Ø:  cpu.ah:=136;
[C42]    21:  intr($15,cpu);
[KF2]    22:  if (cpu.ax=Ø) or
[MS3]    23:   (cpu.flags and 1 = 1) then
[MUØ]    24:  begin
[HY4]    25:   write('Entweder handelt es ');
[485]    26:   write('sich bei Ihrem Gerät ');
[1X3]    27:   writeln('um einen PC');
[UM4]    28:   write('oder XT oder es ist ');
[YA5]    29:   write('kein erweiterter Spei');
[MS4]    3Ø:   writeln('cher installiert.');
[XF1]    31:   writeln;
[4MØ]    32:  end
[ZRØ]    33:   else
[MUØ]    34:  begin
[HK4]    35:   write('Ihr System besitzt ');
[LU4]    36:   write(cpu.ax,' KByte erweit');
[AK4]    37:   writeln('erten Speicher.');
[ZNØ]    38:  end;
[KNØ]    39:  end.
```

Listing 2. Feststellung der Größe des Extended Memory (»checkxms.pas«)

Der Zugriff auf das »Innenleben« Ihres Computers

Für die meisten Anwendungen bietet Turbo Pascal leistungsfähige Befehle an, dennoch kann es bei bestimmten Problemstellungen zweckmäßig oder notwendig sein, Hardware-Bausteine direkt anzusteuern.

Alle Hardware-Einheiten, die an den PC angeschlossen sind, also beispielsweise die Tastatur, der Bildschirm oder der Druckerausgang, sind mit dem Prozessor verschaltet. Wenn dem nicht so wäre, bestünde für den Prozessor keine Möglichkeit, Peripheriegeräte anzusteuern. Noch nicht einmal der Bildschirm könnte Daten anzeigen.
Bei den Computern der PC- und PS/2-Reihe werden Hardware-Bausteine über spezielle Ein-/Ausgabekanäle (meistens als »Ports« bezeichnet) mit der Zentraleinheit verschaltet. Diese Kanäle sind mit den Steuerregistern einer Hardware-Einheit verbunden (siehe auch Tabelle 1). Indem Werte in einen Port geschrieben werden, gelangen sie in das Hardware-Register des angeschlossenen Gerätes. Werden Lesezugriffe auf einen Port durchgeführt, können Daten aus einem Register des Peripheriegerätes gelesen werden. Die unterschiedlichen Ports der verschiedenen Hardware-Einheiten werden wie Speicherzellen über Adressen angesprochen. Für Ports können allerdings nur Adressen vergeben werden, die 16 Bit breit sind, also im Bereich von 0000hex bis FFFFhex liegen. Damit ist der Adreßbereich von 0000hex bis FFFFhex zweifach belegt: durch die Speicherstellen 0000hex bis FFFFhex und durch die Ports 0000hex bis FFFFhex. Daher gibt es beispielsweise einen Port 03D4hex und eine Speicherstelle 03D4hex. Wie unterscheidet der Prozessor zwischen einem Port und einer Speicherstelle an gleicher Adresse? Im Bild können Sie den Unterschied erkennen. Der Adreßraum für Speicherstellen und der Adreßraum für Ports sind bei PCs völlig voneinander getrennt. Die Adresse eines Ports oder eines Speicherbausteins wird zwar in beiden Fällen auf den Adreßbus ausgegeben, hinzu kommt jedoch ein spezielles Steuersignal. Beim Zugriff auf Speicherstellen heißt dieses Signal »Speicher lesen«, beim Zugriff auf Ports »Port lesen«. Daher werden für die Programmierung von Ports und Speicherstellen unterschiedliche Befehle verwendet. Der wichtigste Maschinenbefehl für das Lesen einer Speicherstelle heißt »mov«, der wichtigste für das Lesen eines Ports hingegen »in«. Je nachdem welches Maschinenkommando in einem Programm steht, erzeugt der Prozessor bei dessen Ausführung das richtige Steuersignal.
In Turbo Pascal werden Zugriffe auf Speicher oder Ports ebenfalls durch zwei Kommandos voneinander unterschieden. Die Funktion (besser: das Pseudo-Array) zum Zugriff auf beliebige Speicherstellen heißt »mem«. Das Mem-Gegenstück für den Portzugriff ist ebenfalls als Pseudo-Array deklariert und trägt den Namen »port«. Hinter dem Schlüsselwort »port« wird in eckigen Klammern die Adresse eines Hardware-Ports angegeben. Je nachdem, auf welcher Seite einer Zuweisung man das Port-Kommando plaziert, wird ein Wert von einem Port in eine Variable eingelesen oder ein Wert von einer Variablen oder Konstanten in einen Port geschrieben. Möchten Sie also beispielsweise den Wert 3 in den Port 3BFhex schreiben, dann können Sie den folgenden Befehl verwenden:

```
port[$3bf]:=3;
```

Soll ein Wert aus Port 3BFhex in die Variable »inhalt« gelesen werden, verwenden Sie das Kommando:

```
inhalt:=port[$3bf];
```

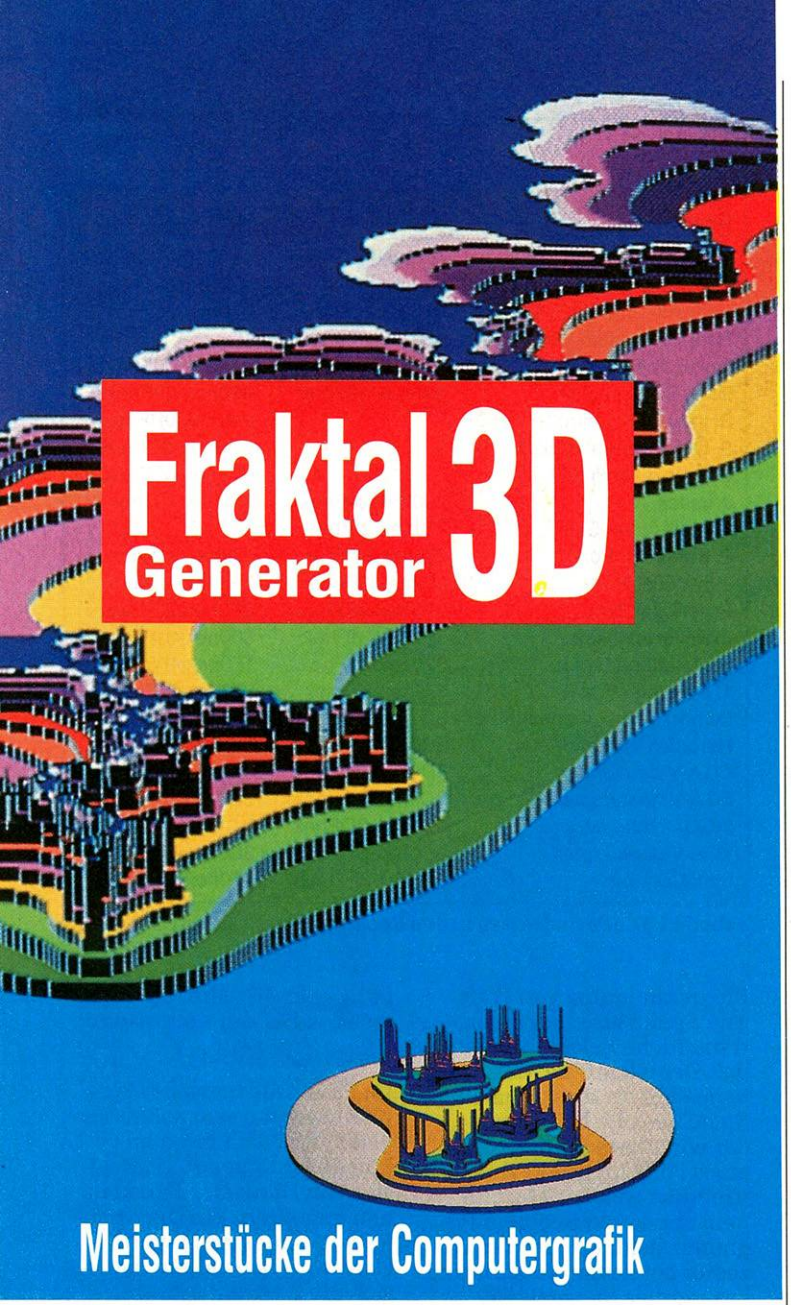
Da Pascal als typische Compiler-Sprache sehr schnell arbeitet, läßt sich mit dem Port-Befehl gut auf die Register von Hardware-Bausteinen zugreifen. Für die direkte Programmierung von Hardware-Bausteinen hat sich ein Konzept eingebürgert. Leistungsfähige Bausteine wie beispielsweise VGA-Karten (die über insgesamt vier VLSI-Chips verfügen) bieten eine große Zahl von internen Registern an, die programmiert werden können. Um die Anzahl der durch die Karte belegten Ports möglichst gering zu halten, wird nicht jedes programmierbare Register über einen eigenen Port adressiert. Vielmehr ist es so, daß die internen Register eines bestimmten Chips über nur zwei Ports angesprochen werden. Dazu werden die Register des Chips indiziert. Jedes Register eines Bausteins erhält eine eindeutige Nummer, den sogenannten Index. Über einen Indexport wählen Sie vor dem Zugriff auf ein Register aus, welches Register verwendet werden soll. Diese Auswahl geschieht durch die Ausgabe des Registerindex an den Indexport. Über einen zweiten Port, den Datenport, werden die eigentlichen Zugriffe auf das

Baustein	im PC und XT	im AT und PS/2
1. DMA-Controller 8237	000hex-00Fhex	000hex-01Fhex
1. Interrupt-Controller 8259	020hex-02Fhex	020hex-03Fhex
Intervall-Timer (8253/8254)	040hex-043hex	040hex-05Fhex
Parallelbaustein 8255	060hex-063hex	–
Tastaturchip (8048)	–	060hex-06Fhex
DMA-Seitenregister	080hex-082hex	080hex-09Fhex
2. Interrupt-Controller 8259	–	0A0hex-0BFhex
2. DMA-Controller 8237	–	0C0hex-0DFhex
Coprozessor 80287	–	0F8hex-0FFhex
Joystick	200hex-20Fhex	200hex-207hex
Erweiterungs-Slot	210hex-217hex	–
2. paralleler Port	278hex-27Fhex	278hex-27Fhex
1. serieller Port	3F8hex-3FFhex	3F8hex-3FFhex
2. serieller Port	2F8hex-2FFhex	2F8hex-2FFhex
Erweiterungskarte	300hex-31Fhex	300hex-31Fhex
Festplattenadapter	320hex-32Fhex	1F0hex-1F8hex
1. paralleler Port	378hex-37Fhex	378hex-37Fhex
monochromer Videoadapter	3B0hex-3BFhex	3B0hex-3BFhex
Hercules-Videoadapter	3B0hex-3BFhex	3B0hex-3BFhex
CGA-Videoadapter	3D0hex-3DFhex	3D0hex-3DFhex
EGA-Videoadapter	3C0hex-3CFhex	3C0hex-3CFhex
MCGA-Videoadapter	3D0hex-3DFhex	–
VGA-Videoadapter	3C0hex-3CFhex	3C0hex-3CFhex
Disk-Controller	3F0hex-3F7hex	3F0hex-3F7hex

Tabelle 1. Die Portadressen in den Rechnern der PC- und PS/2-Serie

Nummer	Funktion
00hex	Gesamtanzahl der Zeichen inklusive Overscan
01hex	Gesamtanzahl der darstellbaren Zeichen
02hex	horizontale Synchronisationsposition
03hex	Zeichendauer eines horizontalen Synchronisationsimpulses
04hex	Buchstabenzeilen pro Refresh
05hex	Anzahl der Rasterzeilen
06hex	Anzahl der Bildschirmzeilen
07hex	vertikale Synchronisationsposition
08hex	Interlace und Skew
09hex	Rasterzeilen pro Zeichen
0Ahex	Start des Cursors
0Bhex	Ende des Cursors
0Chex	höherwertiges Byte, Darstellungsbeginn
0Dhex	niederwertiges Byte, Darstellungsbeginn
0Ehex	höherwertiges Byte, Cursor-Position
0Fhex	niederwertiges Byte, Cursor-Position
10hex	höherwertiges Byte, Lichtgriffelimpuls
11hex	niederwertiges Byte, Lichtgriffelimpuls

Tabelle 2. Die Funktion der 18 internen Hercules-Register



High-Speed - Höchsteffiziente Programmierung in Assembler. Auf dem Amiga jetzt nur noch 7 Sekunden für das "Apfelmännchen"!
Mandelbrot- und Juliamenge - Mit automatischer Glättungsfunktion.
Super-Parallel-Projektion - Frei wählbarer horizontaler Blickwinkel mit 360 Grad: Betrachten Sie das "Fraktalobjekt" von allen Seiten.
Stufenloser vertikaler Blickwinkel: - Wahlweise Sicht von oben, unten, schräg und in der Totalen einzeln und stufenlos einstellbar.
Voller Bedienungskomfort - Auswahl komplett mit Pulldown-Menüs. Wahlweise Steuerung mit der Maus oder über die Tastatur.
Mehrere separate Bildspeicher - Abspeicherung auf dem Amiga im IFF-Format, Verwendung der Bilder in anderen Programmen.
Phantastische Farbmöglichkeiten - Separate Farbzuordnung für die einzelnen Bilder. Animationsmöglichkeit durch Color-Cycling. Die Farben lassen sich auch nachträglich beliebig verändern.

PC Fraktal Generator 3D — MS-DOS ab 2.0; PC-XT/AT mit EGA-Karte oder: Amstrad/Schneider PC 1512. 5 1/4"- oder 3 1/2"-Disk — 69,– DM*

Amiga Fraktal Generator 3D — Commodore Amiga mit 512 KB, 3 1/2"-Disk. — 69,– DM*

Atari Fraktal Generator 3D — Atari ST, 3 1/2" — 69,– DM*

Demodiskette: Fraktal Generator 3D, MS-DOS 5,– DM

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,– DM bzw. 6,– DM Porto/Verpackung. – Unverbindliche Preisempfehlung –

Bitte Bestellkarte benutzen

DMV-Verlag · Postfach 250 · 3440 Eschwege DMV Daten- und Medienverlag

gewünschte Register abgewickelt. Die Steuerlogik des Hardware-Bausteins sorgt dafür, daß immer das zuvor durch den Indexport ausgewählte Register am Datenport verfügbar ist. Dazu ein Beispiel: Der Indexport der Hercules-Karte befindet sich an Adresse 3B4hex, der Datenport an Adresse 3B5hex. Über den Indexport können Sie eines von 18 internen Registern der Hercules-Karte ansteuern, mit denen der Kathodenstrahl zum Bildaufbau programmiert wird. Um beispielsweise das interne Register 0Ahex der Hercules-Karte (es bestimmt die Startzeile der Cursor-Darstellung innerhalb eines Zeichens) mit einem neuen Wert zu versehen, wird zunächst der Wert 0Ahex an den Indexport 3B4hex ausgegeben. Dadurch macht die Hercules-Karte das interne Register mit dem Index 9 am Datenport verfügbar. Wenn nun beispielsweise ein Wert von 2 in dieses Register geschrieben werden soll, wird die Zahl am Datenport 3B5hex ausgegeben. In Pascal erledigen die folgenden zwei Zeilen die beschriebene Aufgabe:

```
port[$3b4]:=9; {Reg. wählen}
port[$3b5]:=2;
{Wert schreiben}
```

Mit dem Prinzip der Indizierung arbeiten fast alle Hardware-Bausteine, die sich an PCs anschließen lassen. Da die Indizierung den Programmieraufwand erhöht (für jeden Registerzugriff werden zwei Port-Befehle benötigt), werden häufig benötigte Register (zum Beispiel Statusregister) meist an zusätzlichen, eigenen Portadressen zur Verfügung gestellt.
Meist befinden sich das Index- und das Datenregister eines Bausteins an aufeinanderfolgenden Portadressen. ATs können im Gegensatz zum PC und XT nicht nur Bytes, sondern darüber hinaus Doppelbytes an Portadressen ausgeben. Bei modernen Hardware-Einheiten (wie beispielsweise den meisten VGA-Karten) kann daher der Zugriff auf den Index- und den Datenport mit einem einzigen Befehl erledigt werden, indem das niederwertige Byte des Wortes den Index des Registers und das höherwertige Byte den Wert für das Register enthält.
Zur Verdeutlichung der Programmierung von Hardware-Einheiten unter Turbo Pascal

Register	Wert für Grafikmodus
00hex	53
01hex	45
02hex	46
03hex	7
04hex	91
05hex	2
06hex	87
07hex	87
08hex	2
09hex	3

Tabelle 3. Registerwerte für den Hercules-Grafikmodus

dient die Unit »punkte« in Listing 1. Sie bindet Routinen zum Setzen von Videomodi und zum Lesen und Setzen von Bildpunkten für alle gängigen Grafikkarten ein. Obwohl sich diese Zugriffe über die Grafikbefehle der Unit »graph« von Turbo Pascal ebenfalls einfach realisieren lassen, hat die Verwendung der Unit »graph« zwei entscheidende Nachteile: Erstens vergrößert sie den Umfang eines Programms um zirka 20 KByte, und zweitens muß der zum Bildschirm passende Grafiktreiber (zum Beispiel »egavga.bgi« für EGA- und VGA-Karten) ebenfalls auf dem Datenträger gespeichert sein. Für viele kleine Grafikanwendungen oder Programme, die mit sehr wenig Speicherplatz auskommen müssen (zum Beispiel TSR-Programme), ist dieser Aufwand unnötig. Die Unit »punkte« benutzt zum Setzen von Videomodi und Punkten die Funktionen des BIOS und kommt daher mit sehr wenig Programmcode aus.
Wenn Sie mit den Prinzipien der Interrupts noch nicht vertraut sind, finden Sie Einzelheiten in diesem Sonderheft auf der Seite 107. Die einzige Ausnahme der Prozedur »SetzeModus« bildet die Hercules-Karte. Da diese Grafikkarte vom Betriebssystem nicht unterstützt wird, kann sie nicht über eine BIOS-Funktion initialisiert werden. Vielmehr müssen Sie sie über die Programmierung der Hardware-Register direkt in den Grafikmodus setzen.
Die neue Unit »punkte« besteht im wesentlichen aus vier Routinen. Mit der Prozedur »NeuerModus« wird ein beliebiger Videomodus gesetzt. Der Prozedur wird als Parameter die Nummer des Videomodus übergeben. Wird ein Wert von FFhex angegeben, wird der Grafikmo-

dus mit 720x348 Punkten der Hercules-Karte gewählt, indem die Register der Hercules-Karte direkt programmiert werden. Bei anderen Videomodi verwendet sie zum Setzen der Punkte die Funktion 00hex vom Interrupt 10hex. Mit der Prozedur »SetzePunkt« können Sie farbige Bildpunkte auf dem Bildschirm anzeigen. Dazu wird Funktion 0Chex des Interrupt 10hex verwendet.

Wurde die Grafikkarte zuvor mit dem Parameter FFhex der Prozedur »NeuerModus« in die Hercules-Betriebsart gesetzt, werden die Bildpunkte nicht über das BIOS, sondern durch direktes Schreiben in den Bildschirmspeicher der Hercules-Karte verändert.

Die Funktion »LesePunkt« ist das Gegenstück zum Setzepunkt-Befehl. Mit »LesePunkt« wird die Farbe eines Punktes an einer bestimmten Stelle auf dem Bildschirm ermittelt. Normalerweise spricht auch die Lesepunkt-Funktion das BIOS an, lediglich die Hercules-Karte verlangt direktes Lesen aus dem Bildschirmspeicher.

Mit der Prozedur »AlterModus« können Sie vor dem Aufruf der Neuermodus-Routine den aktiven Videomodus wiederherstellen.

Um die Unit in eigenen Programmen verwenden zu können, binden Sie sie mit dem Kommando

```
uses punkte;
```

in einen Pascal-Quelltext ein. Einzelheiten zum Umgang mit Units finden Sie im Artikel »Modularisierung mit Units« in diesem Sonderheft ab Seite 72. Das Programm »pdemo.pas« in Listing 2 demonstriert den Umgang mit den neuen Routinen.

Richten Sie Ihr Augenmerk bei der Betrachtung der Unit »punkte« auf die Programmteile, in denen die Hercules-Karte auf der Hardware-Ebene programmiert wird. Wie bereits beschrieben, wird der Kathodenstrahl der Hercules-Karte über insgesamt 18 interne Register gesteuert. Tabelle 2 schlüsselt die Funktionen der einzelnen Register auf. Um die Hercules-Karte in den Grafikmodus zu setzen, werden die ersten neun der 18 Register mit neuen Werten versorgt. In Tabelle 3 sind die notwendigen Parameter für den Hercules-Grafikmodus aufgeschlüsselt.

Im Listing der Unit »punkte« werden diese Werte in dem Array »grafik_modus« gespeichert:

```
const
grafik_modus: array[1..16] of
byte
= ($35,$2d,$2e,$07,$5b,
$02,$57,$57,$02,$03,$00,
$00,$00,$00,$00,$01);
```

Um die neun Hercules-Register mit den dazugehörigen Werten zu versorgen, wird in der Prozedur »NeuerModus« die folgende Schleife programmiert:

```
for zaehler:=1 to 16 do
begin
port[index_reg]:=zaehler-1;
port[daten_reg]:=
grafik_modus[zaehler];
end;
```

Die beiden Konstanten »daten_reg« und »index_reg« initialisieren Sie im Deklarationsteil mit den Werten 3B5hex und 3B4hex. Mit der Schleife wird nach der beschriebenen Methode für jedes Register einzeln zunächst der gewünschte Index an den Indexport ausgegeben und unmittelbar danach der eigentliche Wert in den Datenport geschrieben.

(Martin Althaus/ks)

```
[Z52]    1: unit punkte;
[124]    2: {-----------------------------------
[WS1]    3:                punkte.pas
[531]    4:                ----------
[HA5]    5:  Stellt einige elementare Funkt-
[NC5]    6:  ionen für den Zugriff auf die Gra-
[ZS4]    7:  fikkarten ohne die Unit GRAPH
[UP5]    8:  unter Turbo Pascal zur Verfügung.
[HR5]    9:  Die neue Unit muß in eigenen Pro-
[UB4]   1Ø:  grammen über das Kommando
[4K5]   11:  "uses punkte;" eingebunden werden.
[WT3]   12:  von Martin Althaus 4/9Ø
[4Ø4]   13:  (C) 199Ø Redaktionsbüro
[L83]   14:      Everts und Hagedorn
[HR3]   15:  -----------------------------------
[U7Ø]   16: }
[SK1]   17: interface
[WV5]   18:  procedure neuermodus(modus:byte);
[YP3]   19:  procedure altermodus;
[YZ5]   2Ø:  procedure setzepunkt(zeile,spalte:
[N13]   21:                      word; farbe:byte);
[MP5]   22:  function lesepunkt(zeile,spalte:
[BZ1]   23:                      word):byte;
[4L2]   24: implementation
[VG1]   25: uses dos;
[6XØ]   26: const
[GH2]   27:  max_hgc_x = 719;
[8T5]   28:    {maximale Spaltennummer der HGC}
[BH2]   29:  max_hgc_y = 347;
[4L5]   3Ø:    {maximale Zeilennummer der HGC}
[5U2]   31:  modus_reg = $Ø3b8;
[4A5]   32:       {Port-Adresse Modus-Register}
[L13]   33:  konfig_reg = $Ø3bf;
[8T5]   34:    {Port-Adresse Konfig.-Register}
[CT2]   35:  index_reg = $Ø3b4;
[R85]   36:       {Port-Adresse Index-Register}
[9T2]   37:  daten_reg = $Ø3b5;
[E75]   38:       {Port-Adresse Daten-Register}
[K55]   39:  bank_hgc : array [Ø..3] of word =
[Z43]   4Ø:          ($bØØØ,$b2ØØ,$b4ØØ,$b6ØØ);
[S55]   41:       {Startadressen der HGC-Bänke}
[EV4]   42:  pixel : array [Ø..7] of byte =
[VU2]   43:            (128,64,32,16,8,4,2,1);
[1S5]   44:    {Werte für die Punktberechnung}
[RNØ]   45: var
[YT2]   46:  cpu: registers;
[CW5]   47:    {Variable für Interrupt-Aufruf}
[1K2]   48:  altmodus:byte;
[5F4]   49:  benutzt,hercules:boolean;
[ØØØ]   5Ø:
[WV5]   51: procedure neuermodus(modus:byte);
[124]   52: {-----------------------------------
[RH5]   53:  Setzt einen Videomodus über das
[6N5]   54:  BIOS. Falls als Modus FFhex über-
[1D5]   55:  geben wird, wird der Grafikmodus
[ØF5]   56:  der Hercules-Karte durch direkte
[CA5]   57:  Programmierung der Register ge-
[H21]   58:  setzt.
[HR3]   59:  -----------------------------------
[U7Ø]   6Ø: }
```

```
[6XØ]   61: const
[PK5]   62:  grafik_modus: array[1..16] of byte
[583]   63:   = ($35,$2d,$2e,$Ø7,$5b,$Ø2,
[2S2]   64:      $57,$57,$Ø2,$Ø3,$ØØ,$ØØ,
[7W1]   65:      $ØØ,$ØØ,$ØØ,$Ø1);
[AW4]   66:       {Werte für die 6845-Register}
[RNØ]   67: var
[TC3]   68:  zaehler,maske: byte;
[GØ3]   69:  schleife: integer;
[MUØ]   7Ø: begin
[2X1]   71:  cpu.ah:=$Øf;
[742]   72:  intr($1Ø,cpu);
[NW2]   73:  altmodus:=cpu.al;
[EL2]   74:  benutzt:=true;
[2U2]   75:  if modus<>$ff then
[MUØ]   76:  begin
[8M1]   77:   cpu.ah:=Ø;
[6E2]   78:   cpu.al:=modus;
[742]   79:   intr($1Ø,cpu);
[4MØ]   8Ø:  end
[ZRØ]   81:   else
[MUØ]   82:  begin
[HS2]   83:   hercules:=true;
[AK3]   84:   port[konfig_reg]:=3;
[GY5]   85:   {HGC auf vollen Zugriff schalten}
[ØP3]   86:   port[modus_reg]:=2;
[8U5]   87:   {Grafikmodus ein- und Bildschirm}
[YN3]   88:   for zaehler:=1 to 16 do
[ØH2]   89:                        {ausschalten}
[MUØ]   9Ø:    begin
[PN5]   91:  {16 Register des CRTC-6845 setzen}
[715]   92:     port[index_reg]:=zaehler-1;
[Ø83]   93:                     {Register-Nummer}
[C63]   94:     port[daten_reg]:=
[D34]   95:          grafik_modus[zaehler];
[CG2]   96:                        {Wert setzen}
[ZNØ]   97:    end;
[N74]   98:   for schleife:=Ø to $2ØØØ-1 do
[P74]   99:             {In einer Schleife den}
[H95]  1ØØ:    begin     {gesamten Bildschirm-}
[EM4]  1Ø1:          {speicher der HGC löschen}
[3V3]  1Ø2:     mem[$bØØØ:schleife]:=Ø;
[AZ2]  1Ø3:                        {Bank Ø löschen}
[BV3]  1Ø4:     mem[$b2ØØ:schleife]:=Ø;
[EZ2]  1Ø5:                        {Bank 1 löschen}
[LV3]  1Ø6:     mem[$b4ØØ:schleife]:=Ø;
[KZ2]  1Ø7:                        {Bank 2 löschen}
[VV3]  1Ø8:     mem[$b6ØØ:schleife]:=Ø;
[PZ2]  1Ø9:                        {Bank 3 löschen}
[ZNØ]  11Ø:    end;
[ZNØ]  111:  end;
[AM1]  112:  cpu.bh:=Ø;
[D66]  113:  {Cursor über BIOS-Funktion 2h von}
[CM1]  114:  cpu.ah:=2;
[LB5]  115:    {Interrupt 1Øh nach links oben,}
[YN1]  116:  cpu.dx:=Ø;
[RF5]  117:      {dazu Register initialisieren}
[742]  118:  intr($1Ø,cpu);
```

Listing 1. Die Unit »punkte« kommt mit wenig Programmcode aus

```
[ØS3]  119:                {Interrupt aufrufen}
[DF4]  12Ø:  port[modus_reg]:=(2 or 8);
[235]  121:         {und Bildschirm einschalten}
[ZNØ]  122: end;
[ØØØ]  123:
[YP3]  124: procedure altermodus;
[124]  125: {---------------------------------
[CK5]  126:  Setzt den alten Videomodus wieder
[ZM5]  127:  zurück, der vor Aufruf der Proze-
[5B4]  128:  dur NEUERMODUS aktiv war.
[HR3]  129:  ---------------------------------
[U7Ø]  13Ø: }
[MUØ]  131: begin
[XS2]  132:  benutzt:=false;
[3U2]  133:  hercules:=false;
[8M1]  134:  cpu.ah:=Ø;
[DØ3]  135:  cpu.al:=altmodus;
[742]  136:  intr($1Ø,cpu);
[ZNØ]  137: end;
[ØØØ]  138:
[YZ5]  139: procedure setzepunkt(zeile,spalte:
[N13]  14Ø:                    word; farbe:byte);
[124]  141: {---------------------------------
[E65]  142:  Setzt einen Punkt im aktuellen
[2K5]  143:  Videomodus. Dazu wird die X-Koor-
[995]  144:  dinate in SPALTE und die Y-Koor-
[4Y4]  145:  dinate in ZEILE übergeben. Die
[F95]  146:  Farbe des Punktes steht im Para-
[TY1]  147:  meter FARBE
[HR3]  148:  ---------------------------------
[U7Ø]  149: }
[RNØ]  15Ø: var
[AN5]  151:  bank,punkt_nr,punkt_wert: byte;
[1T5]  152:  bank_adr,reihe,punkt_adr: integer;
[MUØ]  153: begin
[5P3]  154:  if hercules=false then
[MUØ]  155:  begin
[PW1]  156:   cpu.ah:=$Øc;
[VE2]  157:   cpu.dx:=zeile;
[MR2]  158:   cpu.cx:=spalte;
[RC2]  159:   cpu.al:=farbe;
[742]  16Ø:   intr($1Ø,cpu);
[4MØ]  161:  end
[ZRØ]  162:   else
[MUØ]  163:  begin
[RD4]  164:  if (zeile<Ø) or (spalte<Ø) or
[L13]  165:     (zeile>max_hgc_y)
[NG4]  166:  or (spalte>max_hgc_x) then
[MUØ]  167:   begin
[TN4]  168:    write('Falsche Koordinaten');
[Z84]  169:    writeln(zeile,',',spalte);
[ØWØ]  17Ø:    exit;
[ZNØ]  171:   end;
[B13]  172:  bank:=(zeile mod 4);
[NT4]  173:          {berechne Bank des Punktes}
[KE4]  174:  bank_adr:=bank_hgc[bank];
[334]  175:              {lade Adresse der Bank}
[W54]  176:  reihe:=(zeile-bank) shr 2;
[FM4]  177:         {berechne Reihe in der Bank}
[984]  178:  punkt_adr:=(spalte shr 3);
[RB5]  179:     {berechne Byte-Nr. des Punktes}
[Z75]  18Ø:  punkt_adr:=reihe*9Ø+punkt_adr;
[N75]  181:         {als Offset zum Bank-Start}
[A24]  182:  punkt_nr:=(spalte mod 8);
[UE4]  183:          {berechne Bit der Adresse}
[MV6]  184:  punkt_wert:=mem[bank_adr:punkt_adr];
[VU2]  185:                    {lese Byte aus}
[E23]  186:  farbe:=farbe and 1;
[S65]  187:   if (farbe=1)       {Punkt setzen?}
[VX4]  188:    then           {ja, odere Bitwert}
[8N3]  189:                  {in das Byte hinein}
[SØ5]  19Ø:     punkt_wert:=(punkt_wert or
[T53]  191:                  pixel[punkt_nr])
[DD5]  192:    else     {nein, undiere Bitwert}
[YP3]  193:               {aus dem Byte hinaus}
[Ø85]  194:     punkt_wert:=(punkt_wert and
[TZ3]  195:                 (255-pixel[punkt_nr]));
[ØN4]  196:   mem[bank_adr:punkt_adr]:=
[M82]  197:                  punkt_wert;
[DM3]  198:                   {und schreibe Byte}
[GL5]  199:  end;   {in den Bildschirmspeicher}
[AC4]  2ØØ: end;                      {zurück}
[ØØØ]  2Ø1:
[MP5]  2Ø2: function lesepunkt(zeile,spalte:
[BZ1]  2Ø3:                      word):byte;
[124]  2Ø4: {---------------------------------
[TA5]  2Ø5:  Liest einen Punkt an den Koordi-
[WL5]  2Ø6:  naten SPALTE,ZEILE aus und liefert
[GT5]  2Ø7:  seine Farbe als Funktionsergebnis
[W71]  2Ø8:  zurück
[HR3]  2Ø9:  ---------------------------------
[U7Ø]  21Ø: }
[RNØ]  211: var
[AN5]  212:  bank,punkt_nr,punkt_wert: byte;
[1T5]  213:  bank_adr,reihe,punkt_adr: integer;
[MUØ]  214: begin
[5P3]  215:  if hercules=false then
[MUØ]  216:  begin
[UW1]  217:   cpu.ah:=$Ød;
[VE2]  218:   cpu.dx:=zeile;
```

```
[MR2]  219:   cpu.cx:=spalte;
[742]  22Ø:   intr($1Ø,cpu);
[843]  221:   lesepunkt:=cpu.al;
[4MØ]  222:  end
[ZRØ]  223:   else
[MUØ]  224:  begin
[RD4]  225:  if (zeile<Ø) or (spalte<Ø) or
[L13]  226:     (zeile>max_hgc_y)
[NG4]  227:  or (spalte>max_hgc_x) then
[MUØ]  228:   begin
[TN4]  229:    write('Falsche Koordinaten');
[Z84]  23Ø:    writeln(zeile,',',spalte);
[ØWØ]  231:    exit;
[ZNØ]  232:   end;
[B13]  233:  bank:=(zeile mod 4);
[NT4]  234:          {berechne Bank des Punktes}
[KE4]  235:  bank_adr:=bank_hgc[bank];
[334]  236:              {lade Adresse der Bank}
[W54]  237:  reihe:=(zeile-bank) shr 2;
[FM4]  238:         {berechne Reihe in der Bank}
[984]  239:  punkt_adr:=(spalte shr 3);
[RB5]  24Ø:     {berechne Byte-Nr. des Punktes}
[Z75]  241:  punkt_adr:=reihe*9Ø+punkt_adr;
[N75]  242:         {als Offset zum Bank-Start}
[A24]  243:  punkt_nr:=(spalte mod 8);
[UE4]  244:          {berechne Bit der Adresse}
[MV6]  245:  punkt_wert:=mem[bank_adr:punkt_adr];
[VU2]  246:                    {lese Byte aus}
[Ø85]  247:  punkt_wert:=(punkt_wert and
[ØG3]  248:               (pixel[punkt_nr]));
[144]  249:               {undiere Punkt heraus}
[NK5]  25Ø:  if punkt_wert>Ø then lesepunkt:=1
[E33]  251:                  else lesepunkt:=Ø;
[YL4]  252:           {Setze Funktionsergebnis}
[ZNØ]  253:  end;
[ZNØ]  254: end;
[ØØØ]  255:
[124]  256: {---------------------------------
[LM3]  257:  Initialisierungsteil
[HR3]  258:  ---------------------------------
[U7Ø]  259: }
[MUØ]  26Ø: begin
[W46]  261:  benutzt:=false; {Modus noch nicht}
[XF2]  262:                        {gespeichert}
[E66]  263:  hercules:=false;  {Hercules-Karte}
[YP3]  264:                {noch nicht erkannt}
[KNØ]  265: end.
```

Listing 1. Die Unit »punkte« kommt mit wenig Programmcode aus

```
[GF2]    1: program pdemo;
[124]    2: {---------------------------------
[WG1]    3:             pdemo.pas
[YYØ]    4:             ---------
[MK5]    5:  Demonstriert den Zugriff auf die
[932]    6:  Punkte-Unit
[WT3]    7:  von Martin Althaus 4/9Ø
[4Ø4]    8:  (C) 199Ø Redaktionsbüro
[L83]    9:      Everts und Hagedorn
[HR3]   1Ø:  ---------------------------------
[U7Ø]   11: }
[X52]   12: uses punkte;
[ØC3]   13:                 {Binde die Unit ein}
[RNØ]   14: var
[VS3]   15:  zeile, spalte: integer;
[D54]   16:  index, index2, farbe: byte;
[MUØ]   17: begin
[2X5]   18:  neuermodus($ff);  {Setze den Gra-}
[X35]   19:       {fikmodus der Hercules-Karte}
[XK6]   2Ø:  {Für andere Grafikkarten muß hier}
[NN5]   21:   {ein anderer BIOS-Code verwendet}
[BN4]   22:   {werden, z.B. 4 (CGA), 14 (EGA &}
[N83]   23:   {MCGA) oder 18 (VGA)}
[MH4]   24:  farbe:=1;                {Punkte an}
[4D3]   25:  for index2:=Ø to 1Ø do
[MUØ]   26:   begin
[V73]   27:    for index:=Ø to 8 do
[MUØ]   28:     begin
[PR4]   29:      for spalte:=1 to 5Ø+index*5 do
[MUØ]   3Ø:       begin
[S38]   31:        setzepunkt(13Ø-index*1Ø,1ØØ-index*1Ø
        +spalte,farbe);
[C38]   32:        setzepunkt(11Ø+index*1Ø,1ØØ-index*1Ø
        +spalte,farbe);
[LY7]   33:        setzepunkt(13Ø-index*1Ø+spalte,1ØØ-i
        ndex*1Ø,farbe);
[7Y7]   34:        setzepunkt(11Ø-index*1Ø+spalte,1ØØ+i
        ndex*1Ø,farbe);
[ZNØ]   35:       end;
[ZNØ]   36:     end;
[7F3]   37:   farbe:=(farbe +1 ) mod 2;
[H95]   38:   {Punkte an, dann aus, dann an ..}
[ZNØ]   39:  end;
[T16]   4Ø:  altermodus;      {vorheriger Modus}
[KNØ]   41: end.
```

Listing 2. Das Programm »pdemo.pas« demonstriert die Arbeit mit der neuen Unit

Noch erhältlich:

DOS EXTRA Nr. 3

- Datenfernübertragungsprogramm in 100% Assemblercode
- Hercules-Grafik-Toolbox in Turbo Pascal
- Grafikfunktionen für Turbo C
- Zeichensatzeditor für Epson-kompatible Drucker
- Treiber für AT-Tastatur
- Alle Listings zum Kurs »PC-glasklar«
- Demo für Sprachdigitalisierung

DOS EXTRA Nr. 4

- Desktop-Publishing-Programm »SDTP«
- Kommandoeditor für MS-DOS
- Editor für Disketten und Festplatten
- Speicherresidente Programme auflisten und entfernen
- Paßwort-Schutz für MS-DOS
- Tastatureingabe automatisiert
- Clock-Utility für alle Grafikkarten
- Cursor und Zeichensatz für EGA-Karte
- Fenstertechnik für Turbo C
- Generator für Hilfsbildschirme

DOS EXTRA Nr. 5

- 16 Farben für die CGA-Farbgrafikkarte
- Snapshot: Bildschirmfotos per Software
- Druckerzeichensatz selbstgemacht
- Mausgesteuertes Zeichenprogramm
- Editor in GW-Basic
- Setup-Programm für ATs
- Maschinensprache für GW-Basic
- Erklärtes Listing: das Basic-Spiel »Stardos«

DOS EXTRA Nr. 6

- SBasic-Compiler im Quelltext
- Datenbankprogramm »Parat«
- Fensterorientierte Funktionen für GW-Basic und SBasic
- Turbo-Basic-Bibliothek für 16 Farben
- Analyse von Grafikkarten
- Maskengenerator für Binärdateien
- dBase-Datenbanken in Basic
- Superhirn für Quick Basic 4.0

DOS EXTRA Nr. 7

- MS-DOS-Benutzeroberfläche Easydos
- Benutzeroberfläche für SBasic
- Datengenerator für Basic, C und Pascal
- Sprite-Editor für Farbgrafikkarten
- DOS-Packer
- Virenschutzprogramm
- Weltkarte im Computer
- Einfacher Prozessor-Emulator

DOS EXTRA Nr. 8

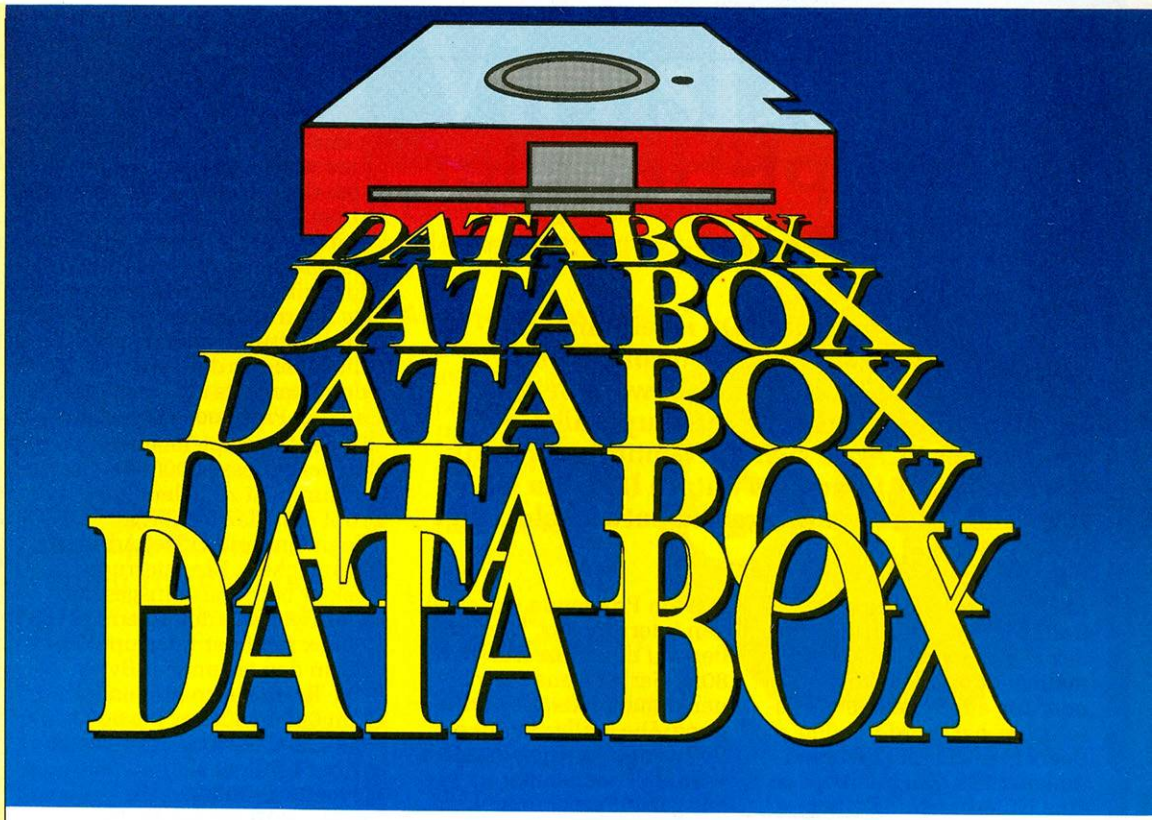
- Makroassembler SAM III
- Exe2bin-Ersatz durch "exe2com"
- Programm zur Verbesserung der Speicherverwaltung unter MS-DOS
- Routine zum Speichern von Hardcopies
- Alle Listings zum Einsteigerschwerpunkt und zu den Werkstätten

DOS EXTRA Nr. 9

- Definitive Eingabe für Batchdateien
- Minidatenbank per Batchdatei
- Komfortable MS-DOS-Befehlseingabe
- Utility für sicheres Dateilöschen
- Pulldown-Menüs unter GW-Basic
- Analoguhr für Selfmade-Designer
- Konvertierung vom EGA- in den Hercules-Modus: Schraffur statt Farbe

DOS EXTRA Nr. 10

- Units-Bibliotheken für Turbo Pascal: Datum, Sound, Bildschirmausgabe
- Prüflisting für die Leistungsdaten der Hardware
- Snapshot-Utility, Version 2.7
- TSR-Datei-Manager
- DOS-Assembler SAM III, Version 1.1
- Schnelles Kopierprogramm mit 115200 Bit pro Sekunde
- Komfortable Dateiumleitung
- Hercules-Grafik in Basic programmiert
- Patch für die BGI-Treiber von Turbo Pascal
- Verbesserungen zu SBASIC



Die Software zu den DOS-EXTRA-Heften

Sagenhaft günstig, extra viel gute Software fürs Geld!

DATABOX:

- **mehr als der übliche Softwareservice**
- **bringt alle Programme der DOS EXTRA auf Datenträger**

Für alle PCs unter MS-DOS

Den Aufbau der Databox-Diskette finden Sie in der Datei »read.me« erklärt. Geben Sie deshalb vor dem Start »type read.me« ein. Sämtliche Pascal- und C-Programme liegen auch in einer compilierten Form für Hercules- und Farbgrafikkarten vor. Es ist allerdings nicht gewährleistet, daß diese Versionen auf Hardware, die nicht vollständig kompatibel zum IBM PC ist, problemlos laufen. In diesem Fall müssen Sie den Quellcode nach der Anpassung mit dem entsprechenden Compiler neu übersetzen.
Die Basic-Programme arbeiten mit dem GW-Basic-Interpreter von Microsoft einwandfrei. Bei Interpretern und Compilern anderer Hersteller kann es zu Problemen bei der Programmausführung kommen, sofern diese nicht voll kompatibel sind.

jeweils 2 Disketten 5 1/4"
1 Diskette 3 1/2"

35,– DM (unverbindliche Preisempfehlung)
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		**Ausland:**	
Einzelpreis	35,– DM	Einzelpreis	35,– DM
zzgl. Versandkosten	4,– DM	zzgl. Versandkosten	6,– DM
Endpreis	**39,– DM**	**Endpreis**	**41,– DM**

Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung.

– Bitte benutzen Sie die Bestellkarte. –

DOS EXTRA Nr. 11

- Preiswerte HPGL-Erweiterung, der HPGL-Extender
- Leistungsstarker Disketteneditor
- Unit für die komfortable Speicherverwaltung unter MS-DOS
- Blockbefehle für den Turbo-Pascal-Editor nach dem SAA-Standard
- Sound-Erzeugung unter Turbo Pascal
- Speicherresidente Uhr

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

Interrupt-Handler in Pascal

Das Betriebssystem MS-DOS ist ein Interrupt-orientiertes Betriebssystem. Sowohl auf der Software- als auch auf der Hardware-Seite bestimmen im wesentlichen Interrupts die Geschicke des PC. Wie sich die Kenntnis der Funktionsweise der Interrupts nutzen läßt, um leistungsfähige Anwendungen zu entwickeln, zeigt dieser Artikel.

Interrupts spielen eine zentrale Rolle im Systemkonzept des PC. Wer sich schon einmal näher mit der Programmierung auf Systemebene beschäftigt hat, wird um den Begriff »Interrupt« nicht herumkommen. Ein Interrupt ist eine Unterbrechung eines laufenden Programms. Wie bei einem Unterprogrammaufruf wird ein laufendes Programm unterbrochen, in ein anderes Programm verzweigt, und nach der Bearbeitung dieser Routine wird wieder zum unterbrochenen Programm zurückgekehrt. Generell sind zwei Arten von Interrupts zu unterscheiden: Hardware-Interrupts und Software-Interrupts. Bei Hardware-Interrupts wird die Unterbrechung eines laufenden Programms von außen gesteuert. Externe Einheiten können beispielsweise durch das Auslösen eines Interrupts eine Unterbrechung des laufenden Programms veranlassen. Das bedeutet, daß sich die Stelle im Programm, an der es durch einen Hardware-Interrupt unterbrochen wird, nicht vorbestimmen läßt. Der Hardware-Interrupt kann an irgendeiner Stelle im Programmcode des laufenden Programms auftreten. Zum Anzeigen von Hardware-Interrupts steht eine spezielle Steuerleitung zur Verfügung. Beim Auftreten eines Interrupts legt der unterbrechende Baustein (beispielsweise der Festplatten-Controller) ein bestimmtes Signal auf diese Leitung. Von einem speziellen Chip auf der Systemplatine, dem PIC-8259 (»Programmable Interrupt Controller«, in Deutsch »programmierbarer Interrupt-Controller«), wird dieses Signal erkannt und die zum Interrupt passende Routine aufgerufen. Ganz anders sieht das bei Software-Interrupts aus. Hier löst ein Programm gezielt einen Interrupt aus. Dazu stehen auf den Prozessoren der 80xx-Serie eigene Maschinenkommandos zur Verfügung. Die Stelle, an der ein Software-Interrupt ausgelöst wird, läßt sich daher im voraus gezielt programmieren. Gleichgültig, ob es sich um einen Hardware- oder Software-Interrupt handelt, die Reaktion des Systems ist im wesentlichen die gleiche. Anhand der Nummer des Interrupts wird aus der sogenannten »Interrupt-Tabelle« die Adresse der Routine bestimmt, die beim Auftreten des Interrupts ausgeführt werden soll. Im nächsten Schritt speichert das System die aktuelle Position im laufenden Programm und einige Statusinformationen auf dem Stapel (Stack). Danach erfolgt die Verzweigung in die Behandlungsroutine für den Interrupt. Der wesentliche Unterschied des Verfahrens bei einem Hardware- und einem Software-Interrupt besteht darin, welche Einheit den Interrupt auswertet und den Start der Behandlungsroutine auslöst.

Der Prozessor unterscheidet zwischen Software- und Hardware-Interrupts

Während bei einem Hardware-Interrupt der PIC-8259-Baustein die Interrupt-Nummer bestimmt und die Verzweigung auslöst, erledigt die gleiche Aufgabe beim Auftreten eines Software-Interrupts der Prozessor, der den Interrupt-Befehl als gültiges Maschinenkommando erkennt und den Sprung zur Behandlungsroutine vornimmt. Die Adresse der Routine wird anhand der Nummer des ausgelösten Interrupts bestimmt. Bei einem Hardware-Interrupt ermittelt der PIC-8259 die auslösende Einheit und die entsprechende Interrupt-Nummer, bei einem Software-Interrupt wird die Nummer des aufzurufenden Interrupts als Parameter an den Prozessor übergeben.

In der Interrupt-Tabelle, die an der Adresse 0000:0000hex beginnt, ist zu jedem Interrupt eine dazugehörige Segment- und Offset-Adresse gespeichert. Ein Interrupt-Vektor belegt also insgesamt vier Byte. Für den Interrupt 00hex steht der Interrupt-Vektor in den ersten vier Bytes der Tabelle, also im Bereich von 0000:0000hex bis 0000:0003hex. Der Interrupt-Vektor 01hex schließt sich in den Byte 0000:0004hex bis 0000:0007hex unmittelbar daran an. Die Adresse eines Vektors in der Interrupt-Tabelle wird daher durch Multiplikation der Interrupt-Nummer mit 4 bestimmt. Das Ergebnis wird als Offset im Segment 0000hex verwendet. Für welche Aufgabengebiete die wichtigsten Interrupts zuständig sind, faßt die Tabelle zusammen.

Durch Veränderung der Vektoren der Interrupt-Tabelle lassen sich eigene Routinen als neue Interrupt-Routinen verwenden. Sinnvoll ist eine Änderung von Interrupt-Vektoren aber nur in einem konkreten Zusammenhang. Das Umleiten von Interrupts ist eng verknüpft mit der Möglichkeit, Programme nach ihrer Ausführung resident im Speicher zu belassen.

Für das Verständnis der residenten Verankerung von Programmen ist die genaue Kenntnis der Abläufe notwendig, die zum Starten eines Programms durchgeführt werden. Vor dem Start eines Programms belegt das Betriebssystem MS-DOS vorsichtshalber den gesamten freien Speicherplatz für das Programm.

Da MS-DOS dafür vorgesehen ist, daß immer nur ein Programm zu einer Zeit ausgeführt werden kann, ist diese Vorgehensweise nur konsequent. Nach der Reservierung des Speichers wird das Programm in den Speicher geladen und ausgeführt. Nach der Ausführung stellt MS-DOS den gesamten vom Programm belegten Speicherplatz wieder dem System zur Verfügung. Vor dem Start des Programms ist also die gleiche Menge an Speicherplatz verfügbar wie nach dem Programmende.

Das Alternativkonzept zu der beschriebenen Methode besteht darin, nach dem Beenden eines Programms nicht den gesamten Speicherplatz wieder freizugeben, sondern den vom Programm tatsächlich benötigten Platz weiter reserviert zu halten. Bei normalen Anwendungen ergibt das keinen Sinn. Denn der Programmcode, der noch im Speicher steht, wird nicht mehr benötigt, das Programm ist ja bereits beendet.

Interrupt-Vektoren lassen sich auf Pascal-Prozeduren umleiten

Im Zusammenhang mit der Möglichkeit, Interrupts auf eigene Routinen und Programme umzuleiten, ergeben sich aus dem »Im-Speicher-Halten« eines Programms jedoch sehr leistungsfähige Möglichkeiten. Der Trick liegt darin, Interrupt-Vektoren auf Programmteile umzuleiten, die resident im Speicher bleiben.

Auch MS-DOS selbst verwendet das Verfahren des residenten Verankerns von Programmen, um die Leistungsfähigkeit des PC zu erhöhen. Ein gutes Beispiel ist der deutsche Tastaturtreiber. Er wird in der Regel in der Datei »autoexec.bat« aufgerufen und trägt den Namen »keyb.com« (bei MS-DOS vor Version 3.3 heißt er »keybgr .com«). Der Unterschied zwischen einer deutschen Tastatur und dem amerikanischen Original liegt unter anderem darin, daß die Buchstaben »y« und »z« auf der deutschen Tastatur vertauscht sind. Für die Bedienung der Tastatur selbst ist es unerheblich, wie die Tasten beschriftet sind, beim Drücken einer bestimmen Taste wird der dazugehörige Tastencode an den PC gesandt. Der Tastaturcode (der sogenannte »Scancode«) wird aus der Position der Taste auf der Tastatur bestimmt. Beim Drücken der Taste »z« auf der deutschen Tastatur wird daher der Code gesendet, den das Betriebssy-

stem als »y« interpretiert. Der Tastaturtreiber »keyb.com« umgeht dieses Problem, in dem er bei jeder Abfrage der Tastatur untersucht, ob als Ergebnis der »z«-Code geliefert wird. Ist das der Fall, wird der Code in den »y«-Code umgewandelt und umgekehrt. Dazu leitet »keyb.com« den Interrupt zur Abfrage der Tastatur (Interrupt 09hex) auf eine neue Routine um und bleibt danach resident im Speicher.
Nach dem gleichen Prinzip lassen sich (fast) beliebige Interrupts auf eigene Routinen umleiten und dadurch die Möglichkeiten des Systems stark erweitern. Viele gängige Software-Tools wie beispielsweise »Sidekick« oder »PC-Tools« arbeiten ebenfalls nach dem Prinzip des Umleitens von Interrupts und der nachfolgenden residenten Verankerung.

Wie aber werden Interrupts von Turbo Pascal aus umgeleitet und Programme resident im Speicher gehalten? Das Umleiten von Interrupts von Turbo Pascal aus besteht im wesentlichen aus zwei Schritten: der Formulierung einer Prozedur, die Interruptfähig ist, und der Umleitung eines bestimmten Interrupt-Vektors auf die neue Prozedur.

Interrupt-Routinen werden resident im Speicher verankert

Für das Schreiben von Interrupt-fähigen Prozeduren hält Turbo Pascal ein reserviertes Schlüsselwort bereit. Die Deklaration einer Prozedur, die als Interrupt-Routine verwen-

Nummer	Typ	Funktion
00hex	H	CPU: Division durch Null
01hex	H	CPU: Einzelschritt
02hex	H	CPU: Parity-NMI
03hex	H	CPU: Breakpoint erreicht
04hex	H	CPU: Überlauf
05hex	S	Hardcopy
06hex	H	CPU: unbekannter Befehl (80286)
07hex	H	reserviert
08hex	H	Timer
09hex	H	Tastatur
0Ahex	H	zweiter PIC-8259
0Bhex	H	serielle Schnittstelle 2
0Chex	H	serielle Schnittstelle 1
0Dhex	H	Festplatten-Controller
0Ehex	H	Disketten-Controller
0Fhex	H	Drucker
10hex	S	BIOS: Videozugriff
11hex	S	BIOS: Konfigurationszugriff
12hex	S	BIOS: RAM-Größe feststellen
13hex	S	BIOS: Disketten- und Plattenzugriff
14hex	S	BIOS: serielle Schnittstelle
15hex	S	BIOS: Kassetten- und AT-Funktionen
16hex	S	BIOS: Tastaturabfrage
17hex	S	BIOS: Druckerzugriff
18hex	S	BIOS: Aufruf des ROM-BASIC
19hex	S	BIOS: Booten des Systems
1Ahex	S	BIOS: Zeit- und Datumszugriff
1Bhex	S	[Break]-Taste
1Chex	S	Timer
1Dhex	S	Videoparameter
1Ehex	S	Diskettenparameter
1Fhex	S	Zeichensatz
20hex	S	MS-DOS: Programm beenden
21hex	S	MS-DOS: Systemfunktion aufrufen
22hex	S	MS-DOS: Beendigungsroutine normal
23hex	S	MS-DOS: [Ctrl]-[Break]-Routine
24hex	S	MS-DOS: Beendigungsroutine resident
25hex	S	MS-DOS: Diskette/Platte lesen
26hex	S	MS-DOS: Diskette/Platte schreiben
27hex	S	MS-DOS: Beendigungsroutine resident

Tabelle. Die wichtigsten Interrupts (dabei gilt «H« für einen Hardware- und »S« für eine Software-Interrupt)

VirusDoktor

Eine Programmsammlung, die Sie gegen alle bekannten PC-Computer-Viren wappnet

Auch schon betroffen?
- Schäden durch infizierte, unbrauchbare Programmdateien?
- Datenverlust durch formatierte und zerstörte Speichermedien?
- Zeitverlust und Ärger durch Verstellen der Systemkonfiguration?

TEST SIEGER
PC Plus 9/89

Name: Virusdoktor
Preis: 98 Mark
Anbieter: DMV-Verlag
Kurzinfo: Das Programm checkt zuverlässig und äußerst schnell. Die durchgängige Logik, die CMOS-RAM-Backup-Möglichkeit und der niedrige Preis sprechen ebenfalls für das Programm.
Schutzeffekt: hoch
Killmechanismus: nein
Handhabung: sehr gut

Prädikat

1 PLUS-Punkt = ungenügend,
10 PLUS-Punkte = hervorragend

VirCheck -
Ein Kontrollprogramm der Superklasse!

- Kontrolle Ihrer Programme auf Längenänderung, Attributierung etc.
- Spezieller Check für virusbedingte Abänderung des Programmcodes
- Überwachung von Umbenennungen und Verschiebungen von Dateien
- Beliebige Auswahl der kontrollierten Programmgruppen
- Hohe Effektivität und Geschwindigkeit durch variables SETUP
- Komfortables, menügesteuertes, grafisches Installationsprogramm

VirSperr -
dreimal Schutz vor Virusinfektion

- Drei Programme zum Absichern Ihrer Dateien vor Veränderungen
- Ausführlicher Report über versuchte Zugriffe auf Ihre Dateien
- Virussicheres Fixieren von gesetzten Read-Only-Attributen
- Ein- und Ausschalten des Schreibschutzes per Tastendruck

Und als Zugabe

CMOS-BKP - Konfigurationssicherung für AT- und AT- kompatible Rechner

VirusDoktor

Fünf wertvolle Programme zum Preis von einem plus ausführliche Programmanleitung inklusive Grundwissen über Computerviren

Für alle MS-DOS-Computer

99,– DM (unverbindliche Preisempfehlung)
Wenn Sie über den DMV-Bestellservice bestellen, gilt folgendes:

Inland:		**Ausland:**	
Einzelpreis	99,– DM	Einzelpreis	99,– DM
zzgl. Versandkosten	4,– DM	zzgl. Versandkosten	6,– DM
Endpreis	**103,– DM**	**Endpreis**	**105,– DM**

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

det werden soll, unterscheidet sich formal dadurch, daß an die Deklaration das Wort »interrupt« angehängt wird. Der Prozedurkopf für eine Interrupt-Prozedur sieht beispielsweise wie folgt aus:

```
procedure neuer_int09hex;
interrupt;
```

Die Verwendung des Schlüsselworts ist notwendig, weil eine Interrupt-Routine auf Maschinenebene anders beendet wird als ein normales Programm.

Die direkte Übergabe von Parametern an eine Interrupt-Prozedur ist nicht möglich; lediglich die Prozessorregister können zur Übergabe von Werten verwendet werden. Sollen alle Register des Prozessors in der Interrupt-Prozedur verfügbar sein, dient dazu die folgende Parameterliste:

```
procedure
int(flags,cs,ip,ax,bx,cx,dx,
si,di,ds,es,bp:word);
interrupt;
```

Werden nicht alle Register in der Prozedur benötigt, können am Beginn der Liste Register weggelassen werden. Ein Auslassen eines bestimmten Registers innerhalb der Liste oder die Übergabe eines anderen Registers als das Prozessorregister BP als letztes Element der Liste führt dazu, daß die Register nicht unter den richtigen Namen in der Prozedur verfügbar sind – der PC wird in der Regel abstürzen.

Innerhalb der Interrupt-Prozedur können die meisten Kommandos der Laufzeitbibliothek von Turbo Pascal verwendet werden. Je nachdem welcher Interrupt auf die Prozedur umgeleitet wird, lassen sich bestimmte Funktionen und Routinen nicht verwenden. Ihr Aufruf führt unweigerlich zu einem Systemabsturz.

Dieser Vorgang hat seinen Ursprung im Problem des »reentranten« Betriebssystems MS-DOS. MS-DOS verwendet mehrere interne Stacks, auf denen Zwischenergebnisse, Rücksprungadressen zu bestimmen Routinen und Statusinformationen gespeichert werden. Je nach aufgerufener Routine finden unterschiedliche Stacks Verwendung. Dabei sind die Entwickler davon ausgegangen, daß immer nur eine MS-DOS-Funktion zu einer Zeit aktiv sein kann. Mit dem Konzept der speicherresidenten Programme, die Interrupts umleiten, kann sich diese Gewichtung jedoch verschieben. Angenommen, es wird eine bestimmte DOS-Funktion aufgerufen. Das Betriebssystem speichert bei der Ausführung der Funktion wichtige Systemdaten auf einem der internen Stacks. Unterbricht nun ein Interrupt die Funktion in der Ausführung, werden als nächstes die Programmschritte der Interrupt-Behandlungsroutine (des sogenannten »Handlers«) ausgeführt. Verwendet dieser Handler seinerseits einen Interrupt von MS-DOS, ergibt sich folgende Situation: Der aufgerufene MS-DOS-Interrupt speichert wiederum wichtige Informationen auf dem internen Stack.

Wird eine aktive MS-DOS-Funktion unterbrochen, kann es zu Systemabstürzen kommen

Dabei werden die Daten des ersten Interrupts überschrieben. Nach der Ausführung der Funktion erfolgt der Rücksprung in den Interrupt-Handler. Auch dieses Programm beendet seine Aufgaben. Nun wird die Ausführung an der Stelle in der Funktion von MS-DOS fortgesetzt, vor der das Programm unterbrochen wurde. Wenn diese MS-DOS-Routine auf die zuvor gespeicherten Daten auf dem internen Stack zuzugreifen versucht, befinden sich dort keine sinnvollen Werte, da diese Daten von der den Interrupt-Handler verwendenden MS-DOS-Funktion überschrieben wurden: Der Rechner stürzt ab.

Daher sollten Sie innerhalb von selbstgeschriebenen Interrupt-Handlern niemals Funktionen verwenden, die ihrerseits auf Routinen von MS-DOS zugreifen.

Nach der Formulierung der Interrupt-Routine in einem Pascal-Programm muß der dazugehörige Interrupt-Vektor auf den neuen Handler umgeleitet werden. Turbo Pascal stellt für die Arbeit mit Interrupt-Vektoren zwei Prozeduren zur Verfügung: »GetIntVec« und »SetIntVec«. Beide Prozeduren erwarten einen Pointer und eine Interrupt-Nummer. Mit »GetIntVec« wird die Zieladresse eines Interrupts ausgelesen, »SetIntVec« leitet einen Interrupt auf einen neuen Handler. Bevor Sie einen Interrupt umleiten, ist es in jedem Fall anzuraten, die Adresse des alten Interrupts zu speichern.

Das ist vor allem für Handler notwendig, in denen der alte Interrupt zusätzlich aufgerufen wird. Das Retten des Interrupt 1Chex in die Variable »altint« erledigt beispielsweise die folgende Sequenz:

```
var altint:pointer;
GetIntVec($1c,altint);
```

Als Gegenstück zum Befehl »GetIntVec« arbeitet das Kommando »SetIntVec«. Es erwartet neben einer Interrupt-Nummer die Adresse einer Prozedur, die als Interrupt-Routine arbeiten kann. Um die Adresse der Prozedur zu bestimmen, verwenden Sie am einfachsten den Adreßoperator »@«. Die Umleitung des Interrupt 1Chex auf die Routine »neuertimer« erledigt die folgende Anweisung:

```
SetIntVec($1c,@neuertimer);
```

Denken Sie daran, die Unit »dos« in Ihr Programm einzubinden, da hier die Interruptorientierten Funktionen untergebracht sind.

Ist der Interrupt erst einmal auf die neue Routine umgeleitet, kann das Programm beendet und im Speicher behalten werden. Dazu dient der Befehl »Keep«.

Mit ihm wird ein Pascal-Programm resident im Speicher verankert. Zusätzlich läßt sich ein Fehlercode an das aufrufende Programm übergeben. In der Regel verwenden Sie »Keep« in der Form

```
Keep(0);
```

Diese Anweisung weist Turbo Pascal an, das gesamte Programm im Speicher zu belassen und ohne Fehlercode zu beenden.

Wie zu Beginn bereits erläutert, weist MS-DOS einem neuen Programm in der Regel den gesamten Speicherplatz zu. Für ein speicherresidentes Programm unter Turbo Pascal hat das die Konsequenz, daß nach dem »residenten Beenden« des Programms überhaupt kein Speicher mehr verfügbar wäre. Noch nicht einmal der Kommando-Interpreter »command.com« könnte mehr geladen werden.

Um das zu verhindern, muß Turbo Pascal über die Compiler-Direktive »$M« mitgeteilt werden, daß nicht der gesamte Speicherplatz durch das Programm belegt werden soll. Der Turbo-Pascal-Linker bindet dann entsprechende Parameter in die Exe-Datei ein.

Mit der $M-Anweisung lassen sich die Größe des Stacks sowie die Minimal- und Maximalgröße des Heaps festlegen. Mit der Anweisung

```
{$M 1024,0,0}
```

wird kein Heap zur Verfügung gestellt und der Stack auf das Minimum von 1024 Byte reduziert (kleinere Werte erlaubt Turbo Pascal nicht). Um keine Laufzeitfehler zu erhalten, wenn der Stack einmal nicht ausreichen sollte, wird die Stack-Prüfung mit der folgenden Compiler-Anweisung ebenfalls abgeschaltet:

```
{$S-}
```

Wie sich anhand der bisher dargestellten Informationen bereits eigene Interrupt-Handler formulieren lassen, zeigt das Programm »zeit.pas« im Listing. Das Programm sorgt dafür, daß ständig die aktuelle Uhrzeit in der rechten oberen Ecke des Bildschirms angezeigt wird. Dazu leitet das Programm den Interrupt 1Chex auf die Prozedur »zeigezeit« um. Diesen Interrupt löst das System über den Hardware-Interrupt 08hex zirka 18,2mal in der Sekunde aus. In der Prozedur »zeigezeit« wird daher die Zeit alle 18 Aufrufe des Interrupts um eine Sekunde heraufgezählt und auf dem Bildschirm im Format »SS:MM:SS« dargestellt. Beim Start des Programms »zeit« wird zunächst die aktuelle Systemzeit ausgelesen und in einen String umgewandelt. Dieser String wird von »zeigezeit« zum Hochzählen der Sekunden, Minuten und Stunden verwendet. In der abgedruckten Version ist die Uhr etwas ungenau. Das liegt daran, daß alle 18 Interrupt-Aufrufe eine Sekunde addiert wird, obwohl erst alle 18,2 Aufrufe tatsächlich eine Sekunde vergangen ist.

Alle fünf Sekunden liegt also die Interrupt-Routine um zirka einen Zähler zu weit voraus. Versuchen Sie ruhig einmal, dieses Problem zu beheben. Ein kleiner Tip: Durch Verwendung eines zweiten Zählers läßt sich mitzählen, wann die Uhr genau eine Se-

kunde falsch geht. In diesem Fall reicht es aus, einmal die Darstellungsroutine der Prozedur »zeigezeit« nicht zu durchlaufen, und schon ist die Zeit wieder richtig justiert.

Auf die beschriebene Art und Weise lassen sich auch andere Interrupts des Systems in eigenen Pascal-Programmen abfangen.
Für die Programmierung von professionellen residenten Programmen, die sich auf eine bestimmte Tastenkombination hin in ein Anwendungsprogramm einschalten, reichen die beschriebenen Verfahren allerdings nicht aus.

Hier bedarf es der Einbindung von Maschinenspracheroutinen, die wichtige, in Pascal nicht realisierbare Programmteile ausführen.

(Martin Althaus/dm)

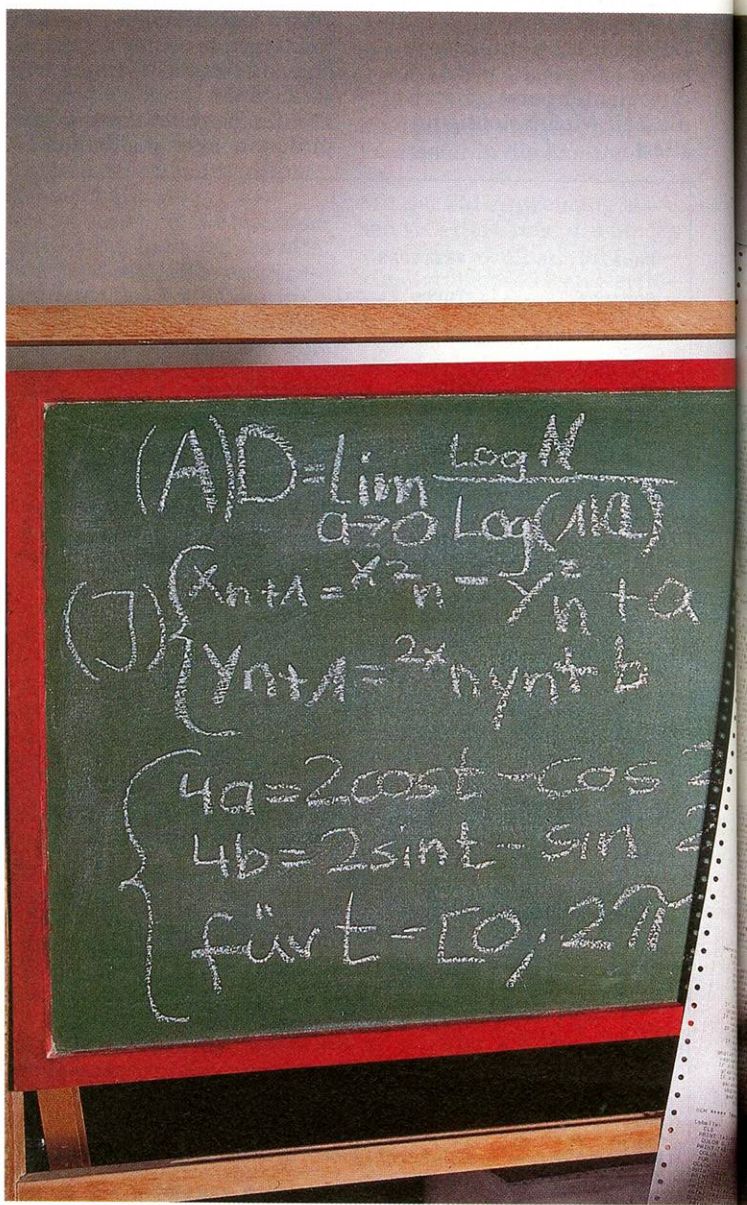
```
  Name..... ZEIT.PAS

  Sprache..... Turbo Pascal

  Grafikkarte..... jede
```

```
[5A2]    1: program zeit;
[S62]    2: uses crt,dos;
[124]    3: {---------------------------------
[YC1]    4:                zeit.pas
[4WØ]    5:                --------
[ØH5]    6:  Speicherresidentes Programm, das
[ZR5]    7:  nach dem Aufruf die Systemzeit be-
[ZY4]    8:  ständig in der rechten oberen
[W84]    9:  Bildschirmecke darstellt.
[ST3]   1Ø:  vom Martin Althaus 4/9Ø
[4Ø4]   11:  (C) 199Ø Redaktionsbüro
[L83]   12:      Everts und Hagedorn
[HR3]   13:  ---------------------------------
[U7Ø]   14: }
[8T5]   15: {$s-}          {Damit noch Speicher}
[535]   16: {$m 1Ø48,Ø,Ø}           {freibleibt}
[6XØ]   17: const
[RE5]   18:  timer=$1c;       {der umzubiegende}
[AG2]   19:                             {Interrupt}
[RNØ]   2Ø: var
[WS2]   21:  oldint:pointer;
[XØ5]   22:         {Zeiger auf den alten Int.}
[1X2]   23:  st,mi,se,hu:word;
[HT4]   24:              {Zähler für HH, MM, SS}
[RX2]   25:  uhr,help:string;
[F12]   26:  count:byte;
[GH4]   27:              {Zählt die Timer-Ticks}
[ØØØ]   28:
[TN5]   29: procedure zeigezeit; interrupt;
[124]   3Ø: {---------------------------------
[PG5]   31:  Stellt die aktuelle Uhrzeit oben
[8C5]   32:  links in der Bildschirmecke dar.
[2K5]   33:  Die Zeit ist in dem String UHR ge-
[9R1]   34:  speichert.
[HR3]   35:  ---------------------------------
[U7Ø]   36: }
[RNØ]   37: var
[CL1]   38:  x,y:byte;
[VK4]   39:              {Bildschirmkoordinaten}
[MUØ]   4Ø: begin
[VB5]   41:  inc(count);           {Tick zählen}
[6D5]   42:  if count=18 then    {eine Sek. um?}
[US4]   43:  begin              {ja, zeige Zeit}
[2L5]   44:   count:=Ø;        {Zähler auf null}
[ØV2]   45:   if uhr[8]<'9' then
[2Z2]   46:                    inc(byte(uhr[8]))
[DD5]   47:          {Einer-Sekunden raufzählen}
[9P4]   48:    else           {Ein Sprung von 9}
[634]   49:    begin                   {auf null}
[S95]   5Ø:     uhr[8]:='Ø';    {Einer-Sekunden}
[C12]   51:                             {auf null}
[VU2]   52:     if uhr[7]<'5' then
[ZY2]   53:                     inc(byte(uhr[7]))
[AR4]   54:             {Zehner-Sekunden zählen}
[ZL4]   55:      else           {Ein Sprung von}
[C45]   56:      begin   {Sekunden auf Minuten}
[Y35]   57:       uhr[7]:='Ø';{Zehner-Sekunden}
[S53]   58:                   {auf null setzen}
[XU2]   59:       if uhr[5]<'9' then
[TY2]   6Ø:                     inc(byte(uhr[5]))
[185]   61:       {Einer-Minuten umgesprungen?}
[9V3]   62:        else                 {ja, setze}
[SE4]   63:         begin       {Einer-Minuten}
[C74]   64:          uhr[5]:='Ø';    {auf null}
[SU2]   65:          if uhr[4]<'5' then
[PY2]   66:                       inc(byte(uhr[4]))
[YR5]   67:      {Zehner-Minuten umgesprungen?}
[XX3]   68:           else      {ja, setze sie}
[MV3]   69:           begin       {auf null und}
[AV3]   7Ø:            uhr[4]:='Ø';     {zähle}
[HB3]   71:                  {die Stunden hoch}
[AZ2]   72:            if (uhr[2]<'9') and
[ØN2]   73:               (uhr[1]<'2') then
[GY2]   74:                    inc(byte(uhr[2]))
[7Z3]   75:            {Kein Sprung von Ø9->1Ø}
[XY2]   76:            {oder von 19->2Ø?}
[L43]   77:            else              {doch}
[MUØ]   78:             begin
[M23]   79:              if (uhr[2]='9') then
[KZ1]   8Ø:                    inc(uhr[1])
[B54]   81:              {Einer-Stunden zählen}
[ZRØ]   82:              else
[MU2]   83:               if uhr[2]<'3' then
[YM4]   84:             {Zehner-Stunden zählen,}
[933]   85:                   {solange < 24 Uhr}
[GY2]   86:                    inc(byte(uhr[2]))
[ZRØ]   87:                else
[MUØ]   88:                begin
[KU2]   89:                 if uhr[1]<'2' then
[DY2]   9Ø:                   inc(byte(uhr[1]))
[ZRØ]   91:                                else
[852]   92:                    uhr:='ØØ:ØØ:ØØ';
[964]   93:           {Wechsel von 23:59->ØØ:ØØ}
[ZNØ]   94:               end;
[ZNØ]   95:            end;
[F83]   96:           end;           {Alle IF-}
[PH4]   97:         end;     {Abfragen beenden}
[ZNØ]   98:       end;
[ZNØ]   99:     end;
[KP5]  1ØØ:    x:=wherex;    {Bildschirmkoordi-}
[ZX4]  1Ø1:    y:=wherey;          {naten merken}
[UG5]  1Ø2:    gotoxy(7Ø,1);   {Ausgabeposition}
[PE2]  1Ø3:                          {ansteuern}
[ØD5]  1Ø4:    write(uhr);       {Zeit ausgeben}
[LE5]  1Ø5:    gotoxy(x,y);    {Alte Bildschirm-}
[895]  1Ø6:          {Position wiederherstellen}
[ZNØ]  1Ø7:  end;
[ZNØ]  1Ø8: end;
[ØØØ]  1Ø9:
[E34]  11Ø: {---------------------------------
[K86]  111:  Initialisierungsteil des residenten
[626]  112:  Programms. Setzt den Vektor und in-
[K64]  113:  initialisiert den Zähler
[5X3]  114:  ---------------------------------
[U7Ø]  115: }
[MUØ]  116: begin
[XF1]  117:  writeln;
[X73]  118:  writeln('TIMER 1.Ø');
[YV2]  119:  writeln('---------');
[XF1]  12Ø:  writeln;
[G25]  121:  write('Ein einfaches speicher');
[EF5]  122:  writeln('residentes Programm');
[H85]  123:  writeln('von Martin Althaus');
[XF1]  124:  writeln;
[ZG3]  125:  gettime(st,mi,se,hu);
[4T4]  126:        {Lese die aktuelle Zeit aus}
[CL5]  127:    {und wandle sie in einen String}
[T72]  128:  str(st,help);
[3K5]  129:      {Stunden in String verwandeln}
[283]  13Ø:  if st<1Ø then uhr:='Ø'
[T15]  131:           {Führende Null einsetzen?}
[MØ2]  132:            else uhr:='';
[LV2]  133:  uhr:=uhr+help+':';
[D15]  134:        {Stunden in String einbauen}
[C62]  135:  str(mi,help);
[385]  136:       {Minuten in String umwandeln}
[T24]  137:  if mi<1Ø then uhr:=uhr+'Ø';
[EP4]  138:           {Führende Null einbauen?}
[LV2]  139:  uhr:=uhr+help+':';
[CC5]  14Ø:       {Minuten an Stunden anhängen}
[C62]  141:  str(se,help);
[825]  142:        {Sekunden in String wandeln}
[R24]  143:  if se<1Ø then uhr:=uhr+'Ø';
[B15]  144:           {Führende Null notwendig?}
[EE2]  145:  uhr:=uhr+help;
[3H5]  146:      {Sekunden an Minuten anhängen}
[H74]  147:  getintvec(timer,oldint);
[Z26]  148:   {alten Interrupt-Vektor auslesen}
[5G1]  149:  count:=Ø;
[4V4]  15Ø:                  {Zähler initialisieren}
[TØ5]  151:  setintvec(timer,@zeigezeit);
[MC4]  152:              {Neuen Interrupt setzen}
[841]  153:  keep(Ø);
[B55]  154:        {Programm im Speicher lassen}
[KNØ]  155: end.
```

Listing. Das Programm »zeit.pas« zeigt die aktuelle Systemzeit ständig auf dem Bildschirm an



Bringen Sie

Es gibt eine Menge Möglichkeiten, Pascal-Programme schneller zu machen. Die meisten ergeben sich direkt aus der Arbeitsweise des Compilers. Vor allem im Bereich der Variablenverwaltung und der Programmstruktur können beträchtliche Geschwindigkeitsgewinne erzielt werden. Bitte beachten Sie, daß die folgenden Tips nur auf einen Geschwindigkeitsgewinn ausgerichtet sind und mit einem sauberen Programmierstil nichts zu tun haben.

Viele der in diesem Artikel aufgeführten Optimierungsverfahren sollten Sie erst dann in ein Programm einbinden, wenn die endgültige, lauffähige Version des Programms feststeht. Dokumentieren Sie die Änderungen in jedem Fall ausführlich!

Verwenden Sie die richtigen Schleifenstrukturen

Pascal kennt zwei Arten von Schleifen: die For-Schleife und die Repeat-until- beziehungsweise While-do-Schleife. Setzen Sie beide Schleifentypen richtig ein! Da For schneller arbeitet als Repeat-until, sollten Sie bei allen Anwendungen, bei denen die Abbruchbedingung von vornherein feststeht und nicht ständig im Programm abgefragt wird, die For-Schleife verwenden.

Wenn Sie beispielsweise in einer Programmschleife eine Datei zeichenweise auslesen, dann verwenden Sie anstelle von:

```
{»datei« ist als »file of byte« definiert}
assign(datei,name);
reset(datei);
while not (eof(datei)) do
begin
...
end;
```

besser eine For-Schleife:

```
assign(datei,name);
reset(datei)
for index:=1 to filesize(datei)
do begin
...
end;
```

Vermeiden Sie mehrdimensionale Felder

Mehrdimensionale Felder kosten zum Speichern oder Laden von Daten viel Rechenzeit, weil zur Lokalisierung der Daten je nach Dimension mehrere interne Berechnungen nötig sind. Daher sollten Sie mehrdimensionale Felder durch mehrere eindimensionale Felder mit unterschiedlichen Namen ersetzen.

Statt:

```
a: array[1..3][1..100] of integer;
```

schreiben Sie besser:

```
a1,a2,a3: array[1..10] of integer;
```

Vereinfachen Sie Division, Multiplikation und Potenzierung

Die Multiplikation und vor allem die Division und die Potenzierung kosten Zeit bei der Durchführung. Oft können Sie die Ergebnisse schneller erzielen, wenn anstelle dieser aufwendigen Operationen weniger zeitkritische Berechnungsarten verwendet werden. Die Multiplikation kann man durch Mehrfachaddition, die Division durch Multiplikation mit Werten kleiner als Null und die Potenzierung durch Mehrfachmultiplikation vereinfachen. Diese Veränderungen zahlen sich vor allem bei kleinen Faktoren und Divisoren spürbar in einer Geschwindigkeitssteigerung aus.

Schreiben Sie statt:

```
a*4
a/4
a+3
```

besser:

```
a+a+a+a
a*0.25
a*a*a
```

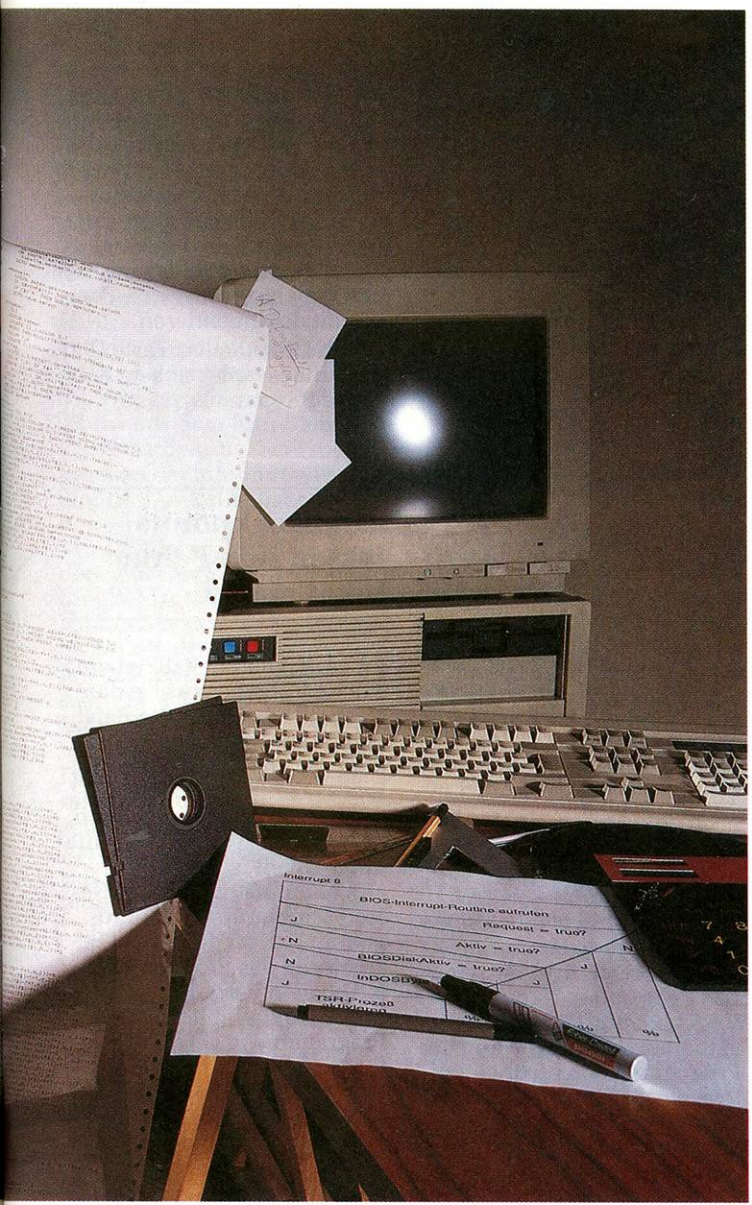
Schnell teilen und multiplizieren

Bei vielen Anwendungen werden Produkte oder Quotienten mit Potenzen von 2 gebildet. Beim Auftreten der Zweierpotenzen (zum Beispiel 2, 4, 8, 16, 32 und so weiter) als Faktoren oder Divisoren können Sie oft eine wesentlich schnellere Alternative zu den Operatoren »/« oder »*« verwenden. Mit der Shl-Anweisung (für »shift left«, also »verschiebe nach links«) wird das Binärmuster einer Zahl nach links verschoben. Der Shr-Befehl (für »shift right«, also »verschiebe nach rechts«) erledigt die Verschiebung nach rechts. Sie können daher Produkte und Quotienten, in denen Zweierpotenzen vorkommen, durch Shr- oder Shl-Kommandos ersetzen. Dazu wird den soeben beschriebenen Pascal-Befehlen der Exponent des Faktors oder Divisors zur Basis 2 übergeben. Ein Beispiel: Die Zahl 8 entspricht der Potenz 2^3. 8 ist also eine Potenz von 2. Statt »x*8« schreiben Sie daher besser »x shl 3«, statt »y/8« besser »y shr 3«. Beim Ersetzen der Division durch den Shr-Befehl müssen Sie beachten, daß der Rest bei der Division (und damit die Nachkommastellen) verlorengeht. Kommt es Ihnen nicht nur auf einen ganzzahligen Quotienten an, müssen Sie weiterhin die Standardoperatoren verwenden.

Statt:

```
alles:=y*32+x*8+z/64;
write(wert*256);
anzahl:=y*4+10;
```

schreiben Sie also besser:



Ihren Computer zur Raserei

```
alles:=(y shl 5)+(x shl 3)+(z
shr 6);
write((wert shl 8));
anzahl:=(y shl 2)+10;
```

Schnell addieren und subtrahieren

Für die Addition und Subtraktion stellt Turbo Pascal die Operatoren » + « und » – « zur Verfügung. Beide Operatoren sind dazu ausgelegt, Summen von beliebigen Zahlen zu bilden. Es liegt auf der Hand, daß die Summe zweier Bytevariablen einfacher zu errechnen ist als bei zwei Variablen in Extended-Format, da letztere intern sehr viel komplexer codiert sind als Bytewerte. Turbo Pascal benutzt bei jeder Summen- oder Differenzenbildung im Prinzip die gleiche Routine der Laufzeitbibliothek. Für Ganzzahlen (also Integer- und Bytewerte) verwenden Sie daher in den meisten Fällen besser die beiden Pascal-Kommandos »inc« und »dec«. Die Abkürzungen stehen für »increment« (was zu deutsch »erhöhen« bedeutet) und »decrement« (zu deutsch »vermindern«). Beide Befehle erhöhen oder vermindern Variableninhalte. Intern werden die beiden Kommandos direkt in Maschinencode umgewandelt. Die Prozessoren der 80x86-Serie bieten eigene Maschinenkommandos zum Inkrementieren und Dekrementieren an. Vor allem bei der Addition und Subtraktion von 1 (zum Beispiel »zaehler: = zaehler + 1;«) arbeitet die Inc-Variante gegenüber » + « wesentlich schneller. Die Inc- und Dec-Befehle erwarten neben dem Namen der zu verarbeitenden Variablen (nur Ganzzahlen sind erlaubt) die Werte, um welche die Variablen erhöht oder vermindert werden sollen. Geben Sie keinen Wert an, wird 1 addiert beziehungsweise subtrahiert.

Schreiben Sie also statt:

```
zaehler:=zaehler+1;
anzahl:=unten-30;
```

besser:

```
inc(zaehler);
dec(anzahl,30);
```

Einfache Rundungsmethode

Sollen Kommazahlen auf ihren ganzzahligen Wert abgeschnitten werden, so sollten Sie den Trunc-Befehl nur dann einsetzen, wenn der neue Wert einer Fließkommavariablen zugewiesen werden soll. Ansonsten kann einfach gerundet werden, indem die Kommazahl einer Ganzzahl zugewiesen wird.

Statt:

```
a:=1365.7866;
b:=trunc(a);
writeln(b);
```

programmieren Sie in diesem Fall:

```
a:=1365.7866:
b:=int(a);
writeln(b)
```

Verwenden Sie die richtigen Variablentypen

Mit Pascal können unterschiedliche, numerische Variablentypen verwendet werden: Ganzzahlen (»integer«), große Ganzzahlen (»longint«), Worte (»word«), Bytes (»byte«), einfachgenaue Fließkommazahlen »single«), doppeltgenaue Fließkommazahlen »double«), erweiterte Fließkommazahlen (»extended«) und Real-Zahlen (»real«). Die höchste Verarbeitungsgeschwindigkeit erlauben die Datentypen »byte« und »word«, da ihre Werte keine Vorzeichen kennen. In jedem Programm sollten Sie prüfen, welchen Wertebereich eine Variable benötigt. Durch die Begrenzung auf »kleine« Formate bei der Deklaration von Variablen können beträchtliche Geschwindigkeitsgewinne erzielt werden.

Schalten Sie die Prüfalgorithmen aus

Turbo Pascal versieht Programme in der Standardeinstellung mit einer ganzen Reihe von internen Prüfroutinen. Wenn ein fertiges Programm einwandfrei läuft und in keiner Situation Fehler auftreten, können Sie die Prüfroutinen von Turbo Pascal ausschalten. Das erreichen Sie entweder durch das Einfügen der entsprechenden Compiler-Direktiven oder durch die Änderung der Einstellungen im Optionenmenü der Turbo-Pascal-Umgebung und anschließendes Übersetzen des Quelltextes.

Binden Sie Maschinenprogramme ein

Optimalen Programmcode können Sie nur erzielen, wenn Sie direkt in Maschinensprache arbeiten. Vor allem für kurze Funktionen bietet es sich an, diese Programmteile komplett in Maschinensprache zu verfassen. Dazu gibt es zwei Verfahren: entweder das direkte Einfügen des Programmcodes über den Inline-Befehl oder die Einbindung des Programms über eine Obj-Datei, die mit einem Assembler separat erzeugt wird.

(Martin Althaus/ks)

Dem Fehler auf der Spur

Der integrierte Debugger

Seit der Version 5.0 bietet Turbo Pascal einen in die Entwicklungsumgebung integrierten Debugger, mit dem sich auf komfortable Weise Fehler finden lassen. Die Grundprinzipien der Fehlersuche im allgemeinen und die spezielle Anwendung des Debuggers sind die Themen dieses Beitrags.

Wer selbst Programme schreibt, kennt den Grundsatz: »Kein Programm ist fehlerfrei.« Bei der Verwendung einer Compiler-Sprache wie Turbo Pascal müssen grundsätzlich drei Gruppen von möglichen Fehlern unterschieden werden: syntaktische Fehler, Laufzeitfehler und logische Fehler.

Relativ einfach zu beheben sind die syntaktischen Fehler. Turbo Pascal sortiert beim Compilieren unbekannte Ausdrücke als Syntaxfehler aus. Dank der guten Hilfestellung des Systems bei der Meldung von Syntaxfehlern lassen sie sich in der Regel schnell beheben. Die zweite Gruppe von Fehlern sind jene, die beispielsweise dadurch entstehen, daß anstelle einer Zahl ein Text eingegeben wird. Diese Fehler bewirken eine Fehlermeldung während des Programmablaufs. Man bezeichnet sie daher als Laufzeitfehler. Auch diese Fehlerklasse läßt sich dank der umfangreichen Fehlercodes von Turbo Pascal meist einfach beheben. Die problematischste Gruppe der Fehler jedoch sind die logischen Fehler. Das sind Fehler, die in einem Programm auftreten, das korrekt compiliert wurde und tatsächlich abläuft, jedoch nicht die gewünschten Resultate erbringt. Der Grund dafür liegt in einem Fehler in der Progammlogik, den der Programmierer gemacht hat. Speziell für solche logischen Fehler ist der integrierte Debugger konzipiert. Er wird über die beiden Menüs »Debug« und »Break/Watch« in der Entwicklungsumgebung gesteuert. Die wesentliche Aufgabe der beiden Menüs ist es, Programme schrittweise ausführen zu können und dabei beständig den Inhalt ausgewählter Variablen zu verfolgen.

Als Beispiel soll an dieser Stelle das Listing 1 »editor1.pas« dienen. Das Programm stellt einen Mini-Editor dar, mit dem sich einzelne Textzeilen eingeben lassen. Die Textzeilen werden nicht in einem Array, sondern in einer dynamischen Datenstruktur auf dem Heap gespeichert.

Das Programm liest Bildschirmzeilen und speichert sie im Record »zeilentyp« dynamisch auf dem Heap. Geben Sie in einer Zeile das Wort »Ende« ein, so wird der Bildschirm gelöscht, und die einzelnen Zeilen gelangen der Reihe nach auf den Bildschirm. Dabei werden den einzelnen Texten Zeilennummern vorangestellt.

Das Programm realisiert die Verknüpfung der einzelnen Zeilen über ein Zeigerfeld, das immer auf den nachfolgenden Datensatz zeigt. Beim Compilieren des Programms in der Turbo-Pascal-Umgebung meldet der Compiler keinerlei Fehler. Startet man das Programm, so läuft es im großen und ganzen korrekt ab: Die einzelnen Zeilen werden eingelesen, bis »Ende« eingegeben wird, danach erfolgt die Ausgabe des Listings mit Zeilennummern. Der einzige Fehler: Auch der Text »Ende« wird ausgegeben, obwohl das eigentlich verhindert werden sollte. Genau zu diesem Zweck wird im Programm in Zeile 75 die While-Schleife eingeleitet:

```
while aktuelle<>nil do
```

Solange der Zeiger mit dem Nullzeiger identisch ist, soll die zum Datensatz gehörige Textzeile ausgegeben werden. Irgend etwas stimmt an dieser Konstruktion nicht. Es hat sich ein logischer Fehler eingeschlichen.

Um den Fehler mit dem integrierten Debugger von Turbo Pascal zu beheben, sollten Sie zunächst einen Blick auf den Programmteil werfen, in dem der Fehler wahrscheinlich steckt: die Schleife ab Zeile 70, die die Ausgabe der einzelnen Textzeilen abwickelt. Turbo Pascal bietet die Möglichkeit, bestimmte Variablen während des Programmablaufs beständig anzuzeigen. Die wichtigsten Variablen innnerhalb der While-Schleife sind »index« (die Zeilennummer) »aktuelle« (Zeiger auf Zeile), »aktuelle^.zeile« (Text der Zeile) und »aktuelle^.naechster« (Zeiger auf nächste Zeile). Diese vier Variablen soll der Debugger während eines Laufs von »editor.pas« auf dem Bildschirm darstellen. Dazu wird aus dem Break/Watch-Menü die Funktion »Add Watch« aufgerufen. Turbo Pascal fragt daraufhin nach dem Namen der Variablen, deren Inhalt anzuzeigen ist. An dieser Stelle wird zunächst »index« eingegeben. Daraufhin blendet Turbo Pascal im unteren Bildschirmbereich in einem eigenen Fenster den Namen der Variablen und dahinter den aktuellen Wert ein. Wenn das Programm noch nicht compiliert wurde, steht hier in der Regel »Unknown Identifier«. Nach der Festlegung von »index« wird dieser Vorgang für die drei weiteren Variablen wiederholt.

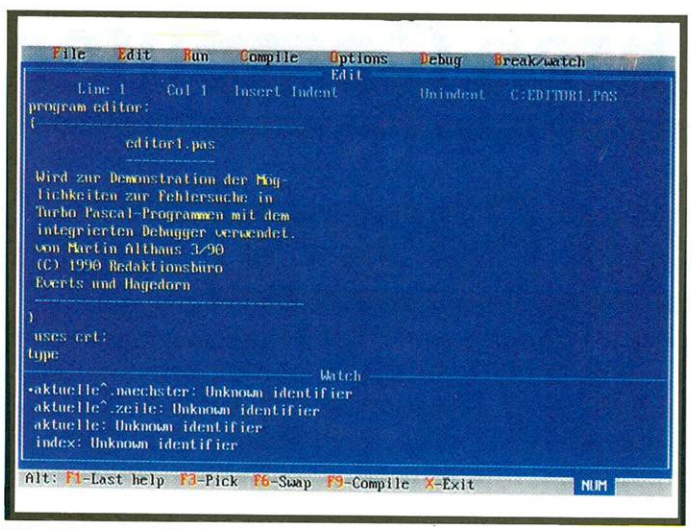
Zum Start des Programms im Debug-Modus verwenden Sie nun nicht wie sonst das Run-Kommando aus dem Run-Menü, sondern vielmehr die Step-Option aus dem gleichen Menü. Damit beginnt der Debugger nach dem Compiliervorgang die einzelnen Programmzeilen schrittweise auszuführen. Nach der Ausführung eines Kommandos wird der Bildschirm der Entwicklungsumgebung eingeblendet, um Veränderungen der Variablen im Watch-Fenster zu dokumentieren.

Durch Drücken der Taste [F7] wird die jeweils nächste Zeile des Programms ausgeführt.

Mit Breakpoints lassen sich Fehler lokalisieren

Im vorliegenden Fall hat eine solche Vorgehensweise den Nachteil, daß auch die Befehle bis zum Programmteil, in dem der Fehler wahrscheinlich steckt, alle einzeln durchlaufen werden müssen. Daher bietet Turbo Pascal die Möglichkeit, sogenannte »Breakpoints« zu setzen. Das sind Markierungen im Programmtext, an denen die Bearbeitung des Programms unterbrochen wird. Bis zu einem Breakpoint läßt sich ein Programm wie gewohnt ausführen, um dann an der unterbrochenen Stelle die Werte im Watch-Fenster zu kontrollieren oder in den Step-Modus zu wechseln.

Für den gegebenen Fall soll die Überprüfung der Inhalte der Variablen erst ab Zeile 71 beginnen. Hier steht die Anweisung:



Die Zeigervariable bei Auftreten des Fehlers

```
aktuelle:=erste;
```

Um in dieser Zeile einen Breakpoint zu setzen, bewegen Sie den Cursor auf die Zeile. Dann aktivieren Sie die Funktion »Toggle Breakpoint« aus dem Break/Watch-Menü. Turbo Pascal zeigt den gesetzten Breakpoint dadurch an, daß die Zeile in einer anderen Farbe (auf monochromen Monitoren unterstrichen) dargestellt wird.
Um die Verarbeitung des Programms zu starten, wählen Sie wie gewohnt die Option »Run« aus dem Run-Menü. Die Verarbeitung des Programms beginnt nun wie gewohnt. Der Reihe nach können einzelne Zeilen mit dem Mini-Editor eingegeben werden. Zur Demonstration der Möglichkeiten des Debuggers sollten Sie sich bei der Eingabe auf die drei folgenden Zeilen beschränken:

```
Das ist der Anfang
und das ist das
Ende
```

Diese Zeilen werden jeweils mit der Enter-Taste abgeschlossen. Nachdem Sie die Zeilen eingegeben haben, setzt Turbo Pascal die Befehlsausführung bis zu dem Punkt fort, an dem der Breakpoint gesetzt ist. Ist diese Stelle erreicht, wird die Programmausführung unterbrochen und in die Entwicklungsumgebung verzweigt. Turbo Pascal zeigt durch eine waagrechte Invertierung an, an welcher Zeile im Programmtext die Verarbeitung unterbrochen wurde.
Im Watch-Fenster zeigen sich die aktuellen Werte der Variablen. Der Wert von »index« ist immer noch undefiniert, da die Variable erst in der nächsten Anweisung initialisiert wird. Als Werte von »aktuelle« und »aktuelle^.naechster« werden die Adressen ausgegeben, auf die beide Zeiger weisen. Die Variable »aktuelle^.zeile« schließlich enthält die erste Textzeile.
Die weiteren Programmzeilen bearbeiten Sie der Reihe nach, indem Sie die Taste [F7] betätigen. Nach dem ersten Drücken von [F7] wird der nächste Befehl ausgeführt, also das Kommando:

```
index:=1;
```

Als Resultat ändert sich im Watch-Fenster der Wert von »index« auf 1. Nun betätigen Sie die Taste [F7] so lange, bis sich erneut der Wert von »index« verändert, diesmal auf 2. Das Programm wird hinter der Programmzeile

```
index:=index+1;
```

am Ende der While-Schleife unterbrochen. An dieser Stelle wurde bislang eine Textzeile auf dem Bildschirm ausgegeben.
Setzen Sie nun die Einzelschrittausführung erneut fort, bis sich der Wert von »index« abermals erhöht, diesmal auf 3. Alle auszugebenden Zeilen (das sind alle eingegebenen bis auf die Zeile »Ende«) sind auf dem Bildschirm dargestellt. Die weitere Beobachtung der Werte für die Variablen kann nun wichtige Aufschlüsse darüber geben, warum das Programm nicht bereits vor der Ausgabe der letzten Zeile abbricht. Dazu drücken Sie zweimal die Taste [F7]. Turbo Pascal steht in der Programmausführung hinter der Einleitung der While-Schleife. Die Konstellation auf dem Bildschirm ist im Bild dargestellt. An dieser Stelle wird der Grund erkennbar, warum Turbo Pascal die Schleife ein weiteres Mal durchläuft. Die Schleife ist abhängig von der formulierten Bedingung:

```
aktuelle<>nil
```

Mit einem Blick auf das Watch-Fenster läßt sich leicht feststellen, daß die Variable »aktuelle« zur Zeit tatsächlich einen Wert ungleich »nil« besitzt (der genaue Wert ist abhängig von der Speicherbelegung). Das scheinbar fehlerhafte Verhalten von Turbo Pascal ist daher nur folgerichtig. Wenn sich aber die Variable »aktuelle« nicht zur Feststellung des Feldendes verwenden läßt, welcher Ausdruck kann statt dessen verwendet werden? Ein Blick in das Watch-Fenster zeigt, daß die Variable »aktuelle^.naechste« tatsächlich den Wert »nil« besitzt. Genau hier wird der logische Fehler deutlich, der sich in das Programm eingeschlichen hat. Die Variable »aktuelle« zeigt nur auf den nächsten Datensatz. Der Zeiger zur nächsten Zeile findet sich aber erst innerhalb dieses Datensatzes. Der Fehler ist gefunden!
Wie der Durchlauf mit dem Debugger ergeben hat, war die Auswahl der Bedingung für die While-Schleife fehlerhaft. An ihrer Stelle muß in Zeile 74 stehen:

```
while aktuelle^.naechste
<>nil
```

Die entsprechende Änderung nehmen Sie sogleich im Turbo-Pascal-Quelltext vor. Danach testen Sie die verbesserte Version. Zunächst werden alle Breakpoints aus dem Listing entfernt. Das erledigt die Option »Clear All Breakpoints« im Break/Watch-Menü. Danach starten Sie das Programm wie gewohnt mit »Run«.
Geben Sie wiederum einige Zeilen ein, und markieren Sie die letzte Zeile mit dem Text »Ende«. In der nachfolgenden Auflistung des Textes tritt der Unterschied zur ersten Programmversion klar und deutlich zu Tage: Die Zeile »Ende« wird nicht mitausgegeben. Die zweite Programmversion »editor2.pas« ist in Listing 2 dargestellt.

Der Debugger wird über das Debug- und das Break/Watch-Menü gesteuert

Den alten Zustand der Entwicklungsumgebung stellen Sie wieder her, indem Sie die Einträge aus dem Watch-Fenster löschen; die Funktion »Remove all watches« im Break/Watch-Menü erledigt diese Aufgabe. Wie das kleine Beispielprogramm gezeigt hat, läßt sich der integrierte Debugger von Turbo Pascal einfach zur Fehlersuche in Programmen einsetzen. Die Möglichkeiten des Debuggers erschließen Ihnen aber auch noch andere Anwendungsgebiete.
Das Debug-Menü bietet eine Reihe von leistungsfähigen Optionen zum Umgang mit dem Debugger an. Mit der Funktion »Evaluate« lassen sich die Inhalte von Variablen (die nicht im Watch-Fenster aktiviert sein müssen) anzeigen und verändern. Dazu wird ein Variablenname eingegeben. Um die Variable zu verändern, geben Sie gleichzeitig einen neuen Wert in das New-Value-Feld des Evaluate-Fensters ein. Turbo Pascal weist der Variablen den neuen Wert zu und kann danach die Programmausführung fortsetzen. Diese Funktion ist vor allem für Programme nützlich, in denen Variablen aufgrund eines nicht lokalisierten Programmfehlers niemals die Werte annehmen, für die sie ausgelegt sind. Wird die Option »Call Stack« aus dem Debug-Menü aktiviert, zeigt der Debugger alle Prozeduren und Funktionen an, die durchlaufen werden. Die Funktion ist vor allem bei tief verschachtelten oder rekursiven Routinen nützlich, da sie einen Einblick in die Verschachtelungstiefe eines Programms erlaubt.
Die beiden Felder »Integrated Debugging« und »Standalone Debugging« aus dem Debug-Menü steuern die grundsätzliche Arbeitsweise des Debuggers. Sofern Sie »Integrated Debugging« aktivieren, arbeitet der Debugger in der Entwicklungsumgebung – das ist die Standardeinstellung. Mit dem Umschalter »Standalone Debugging« definieren Sie, ob zusätzliche Debug-Informationen in eine compilierte Exe-Datei eingebunden werden. Das ist notwendig, wenn das fertige Programm mit einem anderen Debugger (zum Beispiel Microsoft-Codeview oder Turbo Debugger von Borland) bearbeitet werden soll.
Eine wichtige Rolle spielt die Einstellung des Feldes »Display Swapping«. Hier legen Sie fest, auf welche Weise zwischen dem Editor und dem Ausgabefenster umgeschaltet werden soll.
Das Unterbinden des ständigen Umschaltens ist vor allem für Programme interessant, die viel Zeit für den Umschaltvorgang benötigen (beispielsweise Grafikprogramme).
Turbo Pascal unterscheidet für »Display Swapping« die Einstellungen »Smart«, »Always« und »Never«. Im Smart-Modus schaltet Turbo Pascal beim Debuggen nur auf den Ausgabebildschirm, wenn Routinen zur Ein- oder Ausgabe aufgerufen werden. Die Einstellung »Always« legt fest, daß der Ausgabebildschirm vor jedem Befehl aktiviert wird, wobei es egal ist, um welches Pascal-Kommando es sich dabei handelt. Die Never-Option unterbindet generell ein Umschalten auf den Ausgabebildschirm.
Mit den Fähigkeiten des integrierten Debuggers ist somit jeder Turbo-Pascal-Programmierer in der Lage, mit wenig Aufwand und in vertretbarem zeitlichen Rahmen logische Programmfehler ausfindig zu machen und sie zu beseitigen.

(Martin Althaus/zi)

```
[2T2]    1: program editor;
[DW3]    2: {--------------------------------
[3Ø2]    3:              editor1.pas
[T81]    4:              ----------
[KF5]    5:  Wird zur Demonstration der Mög-
[7T4]    6:  lichkeiten zur Fehlersuche in
[PK5]    7:  Turbo Pascal-Programmen mit dem
[3P5]    8:  integrierten Debugger verwendet.
[VT3]    9:  von Martin Althaus 3/9Ø
[4Ø4]   1Ø:  (C) 199Ø Redaktionsbüro
[L83]   11:  Everts und Hagedorn
[AL3]   12:  --------------------------------
[U7Ø]   13: }
[5H1]   14:  uses crt;
[STØ]   15: type
[8H4]   16:  zeilenzeiger=^zeilentyp;
[AY5]   17:       {Zeiger auf dynamische Struktur}
[4Ø2]   18:  zeilentyp =
[Y11]   19:  record
[Z92]   2Ø:   zeile:string;
[7Y3]   21:   naechste:zeilenzeiger;
[ZNØ]   22:  end;
[KY6]   23:  {Definition der dynmaischen Struktur}
[RNØ]   24: var
[786]   25:  erste,aktuelle,vorige:zeilenzeiger;
[EF6]   26:     {Für den Zugriff auf die Struktur}
[PA2]   27:  lesen:string;
[N94]   28:                {liest einen String ein}
[P12]   29:  index:word;
[7X5]   3Ø:        {Zählt die eingegebenen Zeilen}
[VZ1]   31:  taste:char;
[ØK5]   32:          {Für die Abfrage der Tastatur}
[MUØ]   33: begin
[DB1]   34:  clrscr;
[Y35]   35:               {Bildschirm soll frei sein}
[ØØ2]   36:  new(erste);
[4D5]   37:          {Schaffe Platz für eine Zeile}
[DV2]   38:  aktuelle:=erste;
[XS4]   39:               {Zeige auf die erste Zeile}
[P31]   4Ø:  repeat
[H82]   41:   readln(lesen);
[PB4]   42:                {Lies eine Textzeile ein}
[883]   43:   if lesen<>'Ende' then
[Ø34]   44:                   {Ende schon erreicht?}
[MUØ]   45:   begin
[764]   46:    aktuelle^.zeile:=lesen;
[YL6]   47:    {Eingabe in der Struktur speichern}
[DØ3]   48:    vorige:=aktuelle;
[A43]   49:                              {Zeiger merken}
[XE2]   5Ø:    new(aktuelle);
[XL3]   51:                    {Neue Zeile anlegen}
[NY4]   52:    vorige^.naechste:=aktuelle;
[2R5]   53:      {Mit der letzten Zeile verbinden}
[4MØ]   54:   end
[ZRØ]   55:  else
[2U4]   56:               {Ende der Eingabe erreicht}
[MUØ]   57:   begin
[E74]   58:    aktuelle^.naechste:=nil;
[YN4]   59:             {Ende der Kette markieren}
[764]   6Ø:    aktuelle^.zeile:=lesen;
[ØX4]   61:           {aktuelle Eingabe speichern}
[ZNØ]   62:   end;
[883]   63:   until lesen='Ende';
[7H4]   64:                {Folgen weitere Eingabe?}
[DB1]   65:  clrscr;
[H75]   66:               {Bildschirm wieder löschen}
[XX5]   67:  write('Es wurde der folgende Text ');
[HX3]   68:  writeln('eingegeben:');
[XF1]   69:  writeln;
[YT5]   7Ø:     {Nun folgt die Ausgabe des Textes}
[DV2]   71:  aktuelle:=erste;
[KA5]   72:         {Beginne bei der ersten Zeile}
[AE1]   73:  index:=1;
[KE5]   74:               {Den Zähler initialisieren}
[7S3]   75:  while aktuelle<>nil do
[U96]   76:  {Solange die Kette nicht beendet ist}
[MUØ]   77:  begin
[TS2]   78:   write(index,': ');
[EF4]   79:                  {Zeilennummer ausgeben}
[HF4]   8Ø:   writeln(aktuelle^.zeile);
[5N3]   81:                    {Textzeile ausgeben}
[DØ3]   82:   vorige:=aktuelle;
[TM2]   83:                              {Zeile merken}
[365]   84:   aktuelle:=aktuelle^.naechste;
[VY4]   85:             {Auf nächste Zeile zeigen}
[ST2]   86:   dispose(vorige);
[SØ3]   87:                           {Zeile freigeben}
[NG2]   88:   index:=index+1;
[NC4]   89:                    {Zeilennummer erhöhen}
[ZNØ]   9Ø:  end;
[XF1]   91:  writeln;
[C46]   92:  write('Bitte eine Taste drücken ');
[Y44]   93:  repeat until keypressed;
[8G3]   94:                   {Warte auf die Taste}
[MN5]   95:  repeat read(taste) until taste<>#Ø;
[U64]   96:                 {und lese die Taste aus}
[KNØ]   97: end.
```

Listing 1. Das Programm »editor1.pas« enthält einen logischen Fehler

```
[2T2]    1: program editor;
[DW3]    2: {--------------------------------
[3Ø2]    3:              editor1.pas
[T81]    4:              ----------
[KF5]    5:  Wird zur Demonstration der Mög-
[7T4]    6:  lichkeiten zur Fehlersuche in
[PK5]    7:  Turbo Pascal-Programmen mit dem
[3P5]    8:  integrierten Debugger verwendet.
[VT3]    9:  von Martin Althaus 3/9Ø
[4Ø4]   1Ø:  (C) 199Ø Redaktionsbüro
[L83]   11:  Everts und Hagedorn
[AL3]   12:  --------------------------------
[U7Ø]   13: }
[5H1]   14:  uses crt;
[STØ]   15: type
[8H4]   16:  zeilenzeiger=^zeilentyp;
[AY5]   17:       {Zeiger auf dynamische Struktur}
[4Ø2]   18:  zeilentyp =
[Y11]   19:  record
[Z92]   2Ø:   zeile:string;
[7Y3]   21:   naechste:zeilenzeiger;
[ZNØ]   22:  end;
[KY6]   23:  {Definition der dynmaischen Struktur}
[RNØ]   24: var
[786]   25:  erste,aktuelle,vorige:zeilenzeiger;
[EF6]   26:     {Für den Zugriff auf die Struktur}
[PA2]   27:  lesen:string;
[N94]   28:                {liest einen String ein}
[P12]   29:  index:word;
[7X5]   3Ø:        {Zählt die eingegebenen Zeilen}
[VZ1]   31:  taste:char;
[ØK5]   32:          {Für die Abfrage der Tastatur}
[MUØ]   33: begin
[DB1]   34:  clrscr;
[Y35]   35:               {Bildschirm soll frei sein}
[ØØ2]   36:  new(erste);
[4D5]   37:          {Schaffe Platz für eine Zeile}
[DV2]   38:  aktuelle:=erste;
[XS4]   39:               {Zeige auf die erste Zeile}
[P31]   4Ø:  repeat
[H82]   41:   readln(lesen);
[PB4]   42:                {Lies eine Textzeile ein}
[883]   43:   if lesen<>'Ende' then
[Ø34]   44:                   {Ende schon erreicht?}
[MUØ]   45:   begin
[764]   46:    aktuelle^.zeile:=lesen;
[YL6]   47:    {Eingabe in der Struktur speichern}
[DØ3]   48:    vorige:=aktuelle;
[A43]   49:                              {Zeiger merken}
[XE2]   5Ø:    new(aktuelle);
[XL3]   51:                    {Neue Zeile anlegen}
[NY4]   52:    vorige^.naechste:=aktuelle;
[2R5]   53:      {Mit der letzten Zeile verbinden}
[4MØ]   54:   end
[ZRØ]   55:  else
[2U4]   56:               {Ende der Eingabe erreicht}
[MUØ]   57:   begin
[E74]   58:    aktuelle^.naechste:=nil;
[YN4]   59:             {Ende der Kette markieren}
[764]   6Ø:    aktuelle^.zeile:=lesen;
[ØX4]   61:           {aktuelle Eingabe speichern}
[ZNØ]   62:   end;
[883]   63:   until lesen='Ende';
[7H4]   64:                {Folgen weitere Eingabe?}
[DB1]   65:  clrscr;
[H75]   66:               {Bildschirm wieder löschen}
[XX5]   67:  write('Es wurde der folgende Text ');
[HX3]   68:  writeln('eingegeben:');
[XF1]   69:  writeln;
[YT5]   7Ø:     {Nun folgt die Ausgabe des Textes}
[DV2]   71:  aktuelle:=erste;
[KA5]   72:         {Beginne bei der ersten Zeile}
[AE1]   73:  index:=1;
[KE5]   74:               {Den Zähler initialisieren}
[1H5]   75:  while aktuelle^.naechste<>nil do
[U96]   76:  {Solange die Kette nicht beendet ist}
[MUØ]   77:  begin
[TS2]   78:   write(index,': ');
[EF4]   79:                  {Zeilennummer ausgeben}
[HF4]   8Ø:   writeln(aktuelle^.zeile);
[5N3]   81:                    {Textzeile ausgeben}
[DØ3]   82:   vorige:=aktuelle;
[TM2]   83:                              {Zeile merken}
[365]   84:   aktuelle:=aktuelle^.naechste;
[VY4]   85:             {Auf nächste Zeile zeigen}
[ST2]   86:   dispose(vorige);
[SØ3]   87:                           {Zeile freigeben}
[NG2]   88:   index:=index+1;
[NC4]   89:                    {Zeilennummer erhöhen}
[ZNØ]   9Ø:  end;
[XF1]   91:  writeln;
[C46]   92:  write('Bitte eine Taste drücken ');
[Y44]   93:  repeat until keypressed;
[8G3]   94:                   {Warte auf die Taste}
[MN5]   95:  repeat read(taste) until taste<>#Ø;
[U64]   96:                 {und lese die Taste aus}
[KNØ]   97: end.
```

Listing 2. Mit dem integrierten Debugger ist der Fehler schnell behoben: »editor2.pas«

Benutzeroberflächen im SAA-Standard

Grundlagen und Konventionen

Die Bedienungsführung der bisher veröffentlichten Anwendungsprogramme unter MS-DOS zeichnet sich durch starke Individualität aus. Wer die Befehle von Lotus 1-2-3 beherrscht, muß die Kommandos aus MS-Word neu erlernen. Wer dBase meistert, kommt nicht daran vorbei, sich die Handhabung von Wordstar anzueignen, wenn er damit arbeiten will. Standardisierung ist notwendig. Mit dem SAA-Konzept hat IBM einen wichtigen Schritt in diese Richtung getan.

Die Standardisierung der Benutzeroberflächen von PC-Programmen ist schon seit langem fällig. Es gibt zu viele Variationen im Bildschirm- und Menüaufbau, und fast täglich kommen neue hinzu.
Die grundsätzliche Idee, die hinter der standardisierten Programmarchitektur »SAA« (Abkürzung für »System Application Architecture«) steht, ist einfach. Sie besteht darin, die Benutzerschnittstelle auf unterschiedlichen Computern möglichst konstant zu gestalten. Dieses Ergebnis erreichen die Programmierer, indem sie eine Reihe von Konzepten befolgen, die als SAA-Standards festgelegt sind.
Aufgrund dieser Vereinbarungen kann jeder Anwender sicher sein, daß sich der Computer in vergleichbaren Situationen immer gleich verhält. Wenn ein Benutzer die grundlegenden Bestandteile beherrscht und jedem Bildelement eine genau definierte Aufgabe zuteilt, fällt es ihm leicht, mit anderen standardisierten Programmen zu arbeiten, da er sich auf den Anwendungszweck konzentrieren kann und sich nicht mit der Bedienungsführung beschäftigen muß.
Der SAA-Standard besteht im allgemeinen aus drei Teilbereichen, die sich ergänzen. Der erste Abschnitt ist die physikalische Konstanz der beteiligten Hardware.
Um sinnvolle Arbeitsergebnisse zu erzielen, ist es unvermeidbar, daß der Anwender beispielsweise die Funktionstasten immer an der gleichen Stelle auf der Tastatur findet oder die Tasten einer angeschlossenen Maus mit dem Zeigefinger aktiviert.

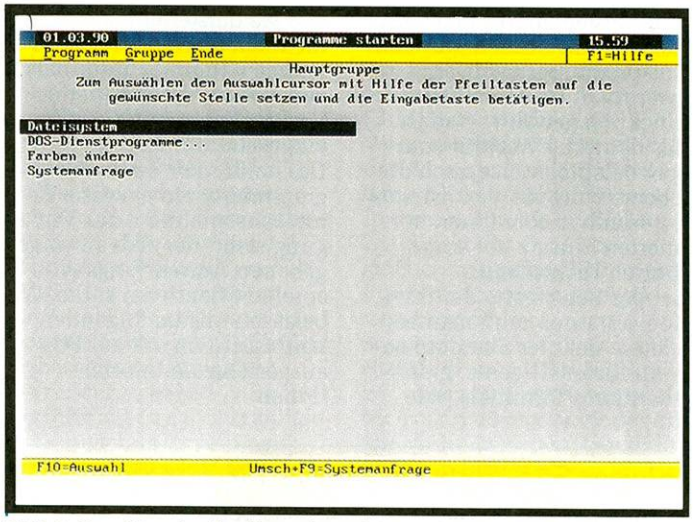


Bild 1. Der Standardaufbau eines SAA-orientierten Programms ist übersichtlich und klar strukturiert

Der SAA-Standard normiert die Bedienungsführung aller Programme

Das zweite Gebiet besteht aus syntaktischen Beständigkeit der Bildelemente. Das bedeutet, daß die auf dem Monitor erscheinenden Symbole, Fenster und Schriften einem identischen Konzept folgen. Das ist zum Beispiel der Fall, wenn alle Fensterüberschriften zentriert angeordnet sind oder Leerzeichen die einzelnen Punkte einer Menüleiste trennen. Als dritter und letzter Komplex präsentiert sich die semantische Stetigkeit, die eine einheitliche Verwendung der Sprache garantiert. Vor allem die letzten beiden Punkte haben die Entwickler des SAA-Standards konsequent und streng dokumentiert. Bevor wir aber auf den strukturellen Aufbau der Menüs und Fenster eingehen, der den Hauptteil der in den folgenden Artikeln beschriebenen SAA-Programmierung bildet, schließen sich einige Anmerkungen zur Semantik an. Für jeden englischen Fachausdruck sieht der SAA-Entwurf eine landesspezifische Übersetzung vor. Die flächendeckend erarbeiteten Sprachen reichen von chinesischen und dänischen Benennungen über holländische und italienische Bezeichnungen bis zu portugiesischen und schwedischen Definitionen. In Tabelle 1 finden Sie eine Auswahl der

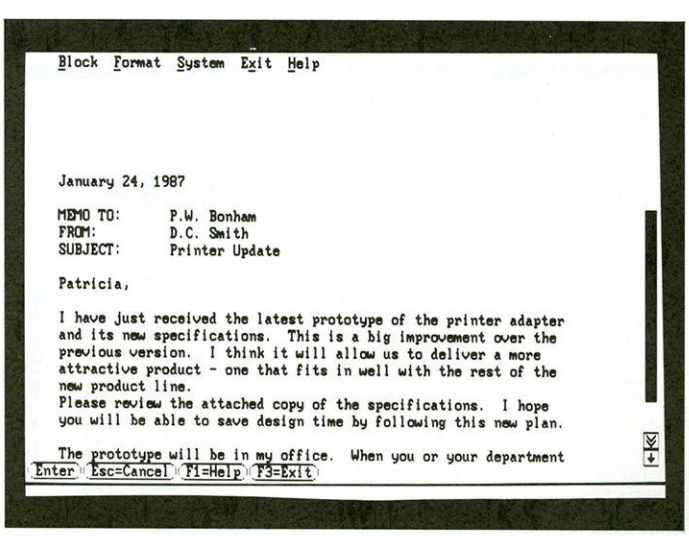


Bild 2. Der Funktionstastenbereich kann grafisch ästhetischer gestaltet sein...

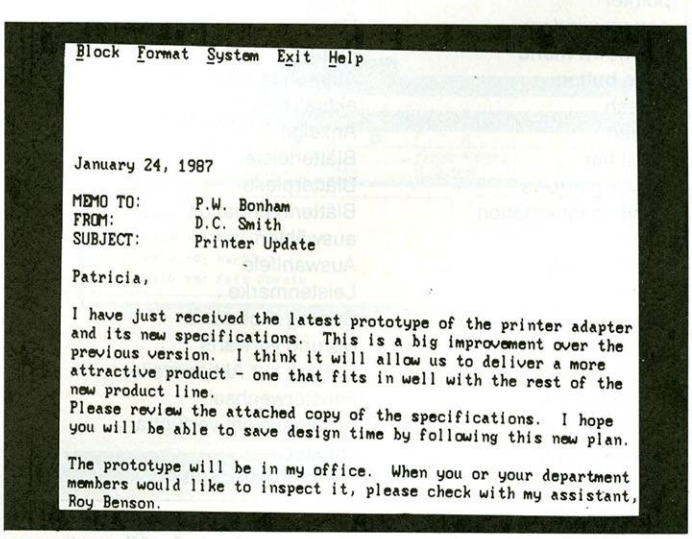


Bild 3. ...oder bei platzintensiven Anwendungen entfallen, beispielsweise bei einer Textverarbeitung

wichtigsten englischen SAA-Begriffe mit ihren deutschen Entsprechungen. Sie sind zwar nicht in jedem Fall glücklich gewählt – so ist das deutsche Wort für »cursor« beispielsweise auch die Übersetzung für »text cursor« –, erlauben aber einen normierten Einsatz der aufgeführten Fachtermini.
Bei der Benutzung der SAA-Konventionen muß man beachten, daß der Standard sowohl für intelligente, grafikorientierte Terminals (wie PCs sie darstellen) als auch für zeichenorientierte, sogenannte »dumme« Terminals gedacht ist, die bei älteren Großrechnern noch häufig eingesetzt werden.
Das heißt, daß jedem Programmierer ein gewisser Ermessensspielraum zur Verfügung steht, der sich an vorgegebenen Anwendungsbeispielen orientieren sollte. Wir befassen uns im folgenden hauptsächlich mit auf PCs ausgerichteten Informationen.

Kleines SAA-Wörterbuch (nach IBM-Vorgaben)

Englischer Begriff	Deutsche Übersetzung
action bar	Aktionszeile
action bar pull-down	Aktionsmenü
available choice	verfügbare Auswahl
backward	rückwärts
beginning of data	Datenanfang
beginning of field	Feldanfang
beginning of line	Zeilenanfang
button	Taste
cancel	abbrechen
command area	Befehlsbereich
command entry field	Befehlseingabefeld
cursor	Positionsanzeiger
default choice	Standardauswahl
display keys	Tastenanzeige
entry field	Eingabefeld
exit	verlassen
extended help	erweiterte Hilfe
fast path	Direktaufruf
forward	vorwärts
function key area	Funktionstastenbereich
group heading	Gruppenüberschrift
help panel	Hilfetextanzeige
icon	Symbol
initial field value	Anfangswert des Feldes
menu	Menü
message	Nachricht
panel	Anzeige
panel ID	Anzeige-ID
panel title	Anzeigeüberschrift
pointer	Zeiger
pop up window	Überlagerungsfenster
pull down menu	Menü
radio button	Auswahlsymbol
refresh	aktualisieren
screen	Anzeige
scroll bar	Blätterleiste
scrolling arrows	Blätterpfeile
scrolling information	Blätterinformation
select	auswählen
selection field	Auswahlfeld
slider box	Leistenmarke
switch backward	Sprung rückwärts
switch forward	Sprung vorwärts
switch to action bar	Sprung zur Aktionszeile
switch window	Fensterwechsel
unavailable choice	Auswahl nicht verfügbar
window	Fenster
window title	Fensterüberschrift

Tabelle 1. Die Beschreibung des SAA-Standards erfolgt über einen englischen Grundwortschatz, dessen deutsche Übersetzung von IBM vorgegeben ist

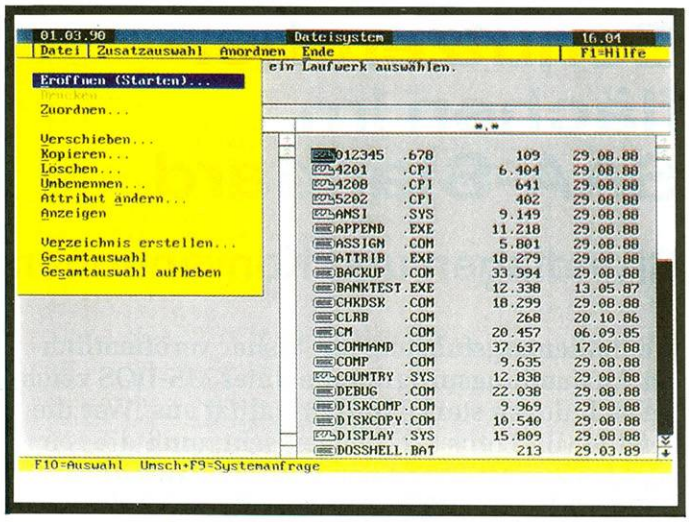


Bild 4. Die Pulldown-Menüs zeigen durch unterschiedliche Schattierungen an, welche Menüpunkte anwählbar sind

Der SAA-Standard geht prinzipiell auf drei Hauptthemen ein: die Bildgestaltung, die Dialogaufbereitung und die Fenstertechnik.
Innerhalb einer SAA-Anwendung ist der Bildschirm in drei Bereiche aufgeteilt, von denen jeder genau spezifizierte Informationen enthält (Bild 1).

Die SAA-Benutzeroberfläche besteht aus Aktionszeile, Anzeigeraum und Funktionstastenbereich

Am oberen Bildrand befindet sich die Aktionszeile, die den Zugriff auf eine Gruppe von übergeordneten Kommandos erlaubt. Wenn der Anwender einen Überbegriff aufruft, »klappt« ein Pulldown-Menü mit den anwählbaren Aktionen auf. Dieses Verfahren birgt zwei wichtige Vorteile: Einerseits sind alle Kommandos sichtbar, und jeder Befehl kann durch einfache Auswahltechniken aktiviert werden.
Aus diesem Grund muß der Anwender nicht alle Funktionen auswendiglernen, um sie effektiv und zweckorientiert nutzen zu können. Andererseits zieht der Aufruf eines Überbegriffs in keinem Fall eine unmittelbare Befehlsausführung nach sich; es erscheint immer zuerst ein Pulldown-Menü. Diese Technik schließt Fehlbedienungen weitgehend aus und vermeidet unnötige Korrekturarbeiten.
Unter der Aktionszeile befindet sich der größte Bild-

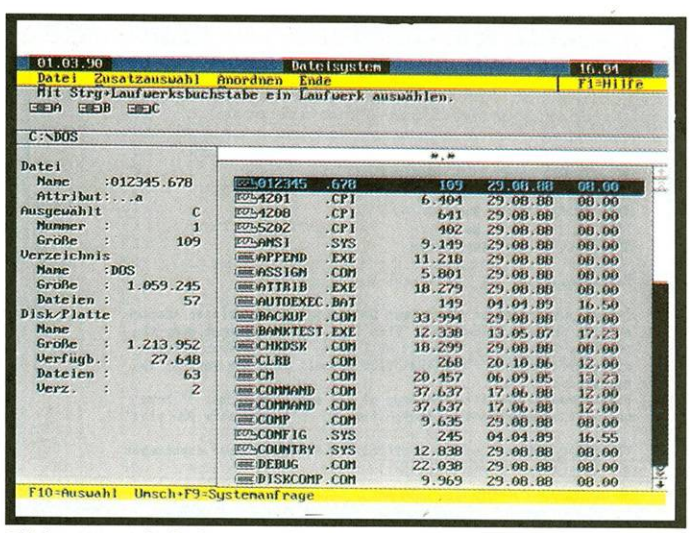


Bild 5. Der Inhalt der verschiedenen Programmfunktionen zugeordneten Fenster läßt sich leicht ändern

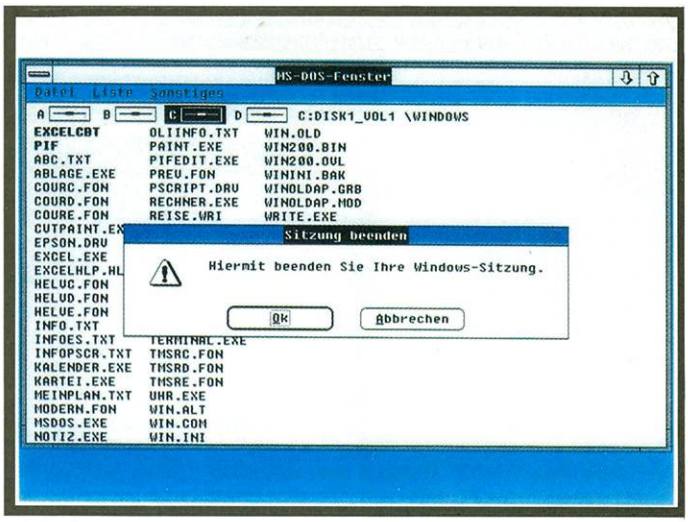


Bild 6. Sogenannte »Buttons« stellen dem Anwender genau definierte Auswahlkriterien zur Verfügung

schirmbereich: der Anzeigeraum. Er dient als eigentliche Ausgabeeinheit für die SAA-Anwendung und übernimmt die Kommunikation mit dem Anwender. Wenn das Programm mehr als eine Kategorie von Informationen gleichzeitig anzeigen soll, kann man den Anzeigeraum dem Informationsgehalt entsprechend unterteilen.

Der untere Bildschirmrand ist für den Funktionstastenbereich reserviert. Dieser nimmt Angaben über die situationsbezogene Funktionstastenbelegung auf und erklärt wichtige Sondertasten. Für den Funktionstastenbereich existieren die drei gültigen Darstellungsformen »kurz« (englisch »short«), »lang« (englisch »long«) und »keine Anzeige« (englisch »no display«).

Der »kurze« Funktionstastenbereich (Bild 2) ist hauptsächlich für den Einsatz einer Maus vorgesehen. In ihm sind nur die wichtigsten allgemeinen Kommandos abgebildet, die ein Benutzer über die Maus aktivieren kann, ohne die Tastatur zu benutzen. Zu diesem Zweck befinden sich die Kommandos in grafisch hervorgehobenen Feldern, die den Auswahlbereich auf einen kleinen Abschnitt des Bildschirms beschränken. Der SAA-Standard sieht vor, daß der Funktionstastenbereich die Felder »Abbrechen«, »Eingabe«, »Verlassen«, »Hilfe«, »Bedienerführung« und »Aktualisieren« einschließt, wenn sie verfügbar sind. In Hilfefenstern besteht der »kurze« Funktionstastenbereich im allgemeinen aus den Funktionen »Erweiterte Hilfe«, »Hilfeindex« und »Hilfetext für Tasten«.

Tastenbelegung im Funktionstastenbereich
Darstellungsform »kurz«

Taste/Dialogfeld	Aktion
Eingabe	Eingabe
Esc	Abbrechen
F1	Hilfe
F2	erweiterte Hilfe (nur in Hilfefenstern)
F3	Verlassen
F4	Bedienerführung (»prompt«)
F5	Aktualisieren (nur in Standardfenstern)
F9	Hilfstext für Tasten (nur in Hilfefenstern)
F11	Hilfsindex (nur in Hilfefenstern)

Darstellungsform »lang«

Taste/Dialogfeld	Aktion
Eingabe	Eingabe
Esc	Abbrechen
F1	Hilfe
F2	erweiterte Hilfe (nur in Hilfefenstern)
F3	Verlassen
F4	Bedienerführung (»prompt«)
F5	Aktualisieren
F6	Sprung vorwärts (nur in Standardfenstern)
F7	rückwärts
F8	vorwärts
F9	Zurückholen (nur in Standardfenstern)
F9	Hilfstext für Tasten (nur in Hilfefenstern)
F10	Aktionen (Sprung in die Aktionszeile)
F11	Hilfsindex (nur in Hilfefenstern)

Tabelle 2. Der Funktionstastenbereich gibt über die Funktionstastenbelegung und wichtige Sondertasten Auskunft

Der Funktionstastenbereich läßt sich in drei Versionen darstellen

Die Darstellungsform »lang« (Bild 1), die in vielen Fällen als Standardeinstellung dient, benutzen vorwiegend Anwender, die gerne mit der Tastatur arbeiten. Sie beinhaltet neben den Grundinstruktionen, die im »kurzen« Funktionstastenbereich Beachtung finden, situationsbezogene aktuelle Befehle, die der Programmentwickler für wichtig hält. Für »lange« Funktionstastenbereiche plant der SAA-Standard zusätzlich die Felder »Rückwärts« und »Vorwärts« sowie »Aktionen« ein. Letztere beinhalten normalerweise Unterprogrammaufrufe.

Am einfachsten ist die Darstellungsart »keine Anzeige« (Bild 3) zu beschreiben. Wenn sie aktiv ist, dehnt sich der Anzeigeraum auf die unteren Bildschirmzeilen aus, in denen sich bis zu diesem Zeitpunkt der Funktionstastenbereich befand. Um den Funktionstastenbereich anwenderspezifisch zuzuschneiden, schreibt der SAA-Standard vor, daß der Benutzer jederzeit über Kommando zwischen den verschiedenen Darstellungsformen wechseln kann. In vielen Fällen kommt es vor, daß die im Funktionstastenbereich aufgeführten Funktionen Befehlen aus den Pulldown-Menüs der Aktionszeile entsprechen. In dieser Situation dürfen sich die den Kommandos aus den Pulldown-Menüs zugeordneten Tasten nicht von den im Funktionsbereich genannten unterscheiden. Wenn Sie eigene SAA-Programme konzipieren, müssen Sie berücksichtigen, daß nicht nur der Hauptbildschirm in drei Teile aufgeteilt ist. Jedes Fenster enthält grundsätzlich den Funktionstastenbereich, dessen Darstellungsart der Anwender bestimmt, und den Anzeigeraum. Die gewählte Anzeigeform ist für alle Funktionstastenbereiche innerhalb einer SAA-Anwendung gleich. Das einzige wählbare Element der Benutzeroberfläche innerhalb dieses Bereichs ist die Aktionszeile, die vor allem in einfachen dialogorientierten Pop-up-Fenstern keinen sinnvollen Zweck erfüllt.

Tabelle 2 zeigt Ihnen, welche Kommandos der SAA-Stan-

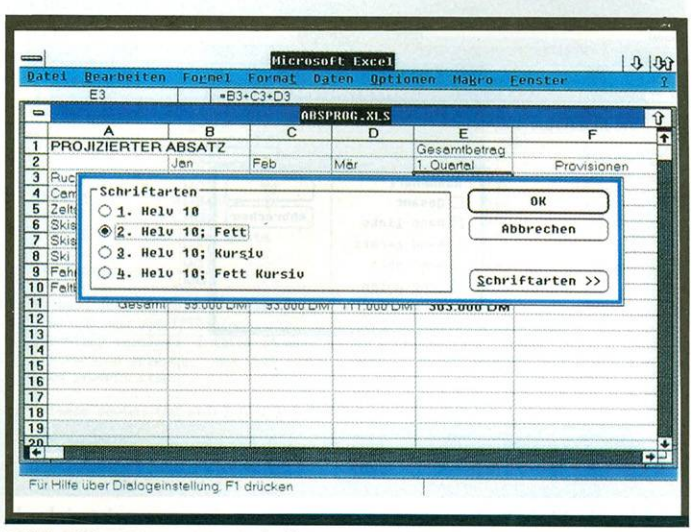


Bild 7. Ein Menü zur einfachen Auswahl (»single choice«) überläßt es dem Anwender, aus einer Liste ein Element auszuwählen

Funktion	Abkürzung
Erweiterte Hilfe	Ex Help
Rückwärts	Bkwd
Sprung rückwärts	Switch bkwd
Sprung vorwärts	Switch fwd
Sprung zur Aktionszeile	Actions
Tastenanzeige	Keys
Vorwärts	Fwd

Tabelle 3. Einige Kommandos im Funktionstastenbereich lassen sich durch englische Abkürzungen ersetzen

Anzeigefläche	Farbpalette
Primär- und Sekundärfenster mit Ausnahme von Hilfetextanzeigen	weiß
Fenster mit Hilfetexten	blau
Popup-Fenster	
– erste Ebene	türkis
– zweite Ebene	weiß
– dritte und weitere Ebenen	abwechselnd türkis und weiß

Tabelle 4. Der SAA-Standard legt bindend fest, welche Farbpaletten bei Farbbildschirmen Verwendung finden

dard für den Funktionstastenbereich vorsieht. Da der Platz am unteren Bildschirmrand nicht unbegrenzt ist, schlagen die SAA-Konventionen für einige Kommandos die in Tabelle 3 aufgelisteten englischen Abkürzungen vor.

Fenster und Pulldown-Menüs zählen zu den Grundelementen des SAA-Standards

Der Einsatz der Kürzel erweist sich allerdings als problematisch, weil er nicht in die jeweilige Landessprache eingebettet ist. Darüber hinaus sollte man andere Befehle als die erwähnten nicht abkürzen. In erster Linie bestimmen Fenster und Pulldown-Menüs die Gestaltung der SAA-Benutzeroberfläche, die durch die Kombination von Maus- und Tastaturfunktionen leicht zu bedienen ist. Aufgrund ihrer Struktur befinden sich die Pulldown-Menüs jeweils am oberen Rand eines Fensters, und die Fenster erscheinen in dem in der Mitte befindlichen Anzeigeraum.

Wie Sie aus Bild 4 ersehen können, ähnelt der Aufbau eines Pulldown-Menüs dem einer einfachen Liste, die mehrere Elemente aufführt. Der Unterschied besteht darin, daß ein Pulldown-Menü in Gruppen aufgeteilt ist, die die zur Auswahl gestellten Kommandos in groben Funktionskategorien zusammenfassen. Der Menüpunkt »Drucken« ist im Vergleich zu den anderen Befehlen mit reduzierter Intensität wiedergegeben und zeigt damit an, daß man ihn in der aktuellen Situation nicht aufrufen kann.

Neben den Pulldown-Menüs gehören die Fenster zu den wichtigsten Gestaltungsmerkmalen des SAA-Standards. Sie unterteilen den Anzeigeraum in funktions- beziehungsweise programmteilbezogene Bereiche, die unter der Kontrolle des aktiven Programms stehen. Der Anzeigeraum läßt sich, wie Bild 5 zeigt, in mehrere Abschnitte gliedern, die unterschiedliche Aufgaben erfüllen. In unserem Beispiel zeigt der rechte Teil eine Dateiliste, während der linke Bildbestandteil Statusinformationen bietet. Falls sich der Anwender zwischen mehreren Kriterien entscheiden muß, die von Anfang an feststehen, sehen die SAA-Vereinbarungen vor, daß die festgesetzten Kommandos in Kurzform in sogenannten »Buttons« erscheinen (Bild 6). Der Anwender wählt mit Hilfe eines Button über die Tastatur oder die Maus, welche der Programmfunktionen er ausführen will. Dabei spielt es eine große Rolle, daß der auswahlbezogene Kommentar – in unserem Beispiel »Hiermit beenden Sie Ihre Windows-Sitzung« – prägnant und verständlich die Aufgabe der Buttons erläutert.

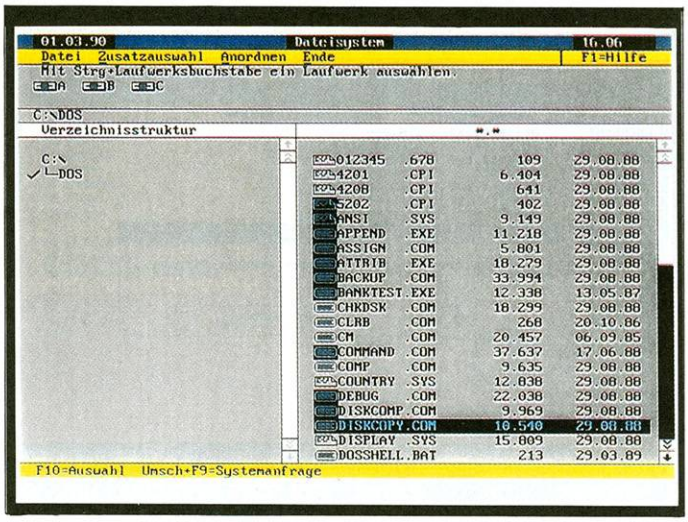


Bild 9. Das Prinzip der erweiterten Auswahl (»extended choice«) bietet sich an, wenn der Benutzer aus einer inhaltlich variablen Aufstellung einen oder mehrere Punkte markieren will

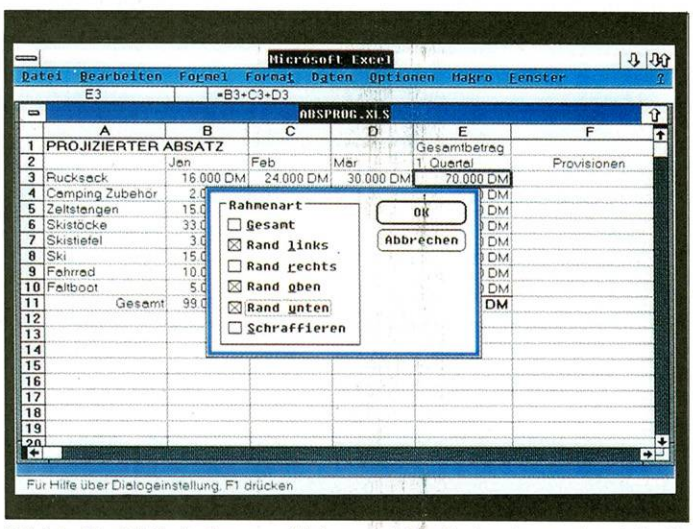


Bild 8. Die Mehrfachauswahl (»multiple choice«) erweist sich als sinnvoll, um aus einem Angebot mehrere Positionen zu aktivieren

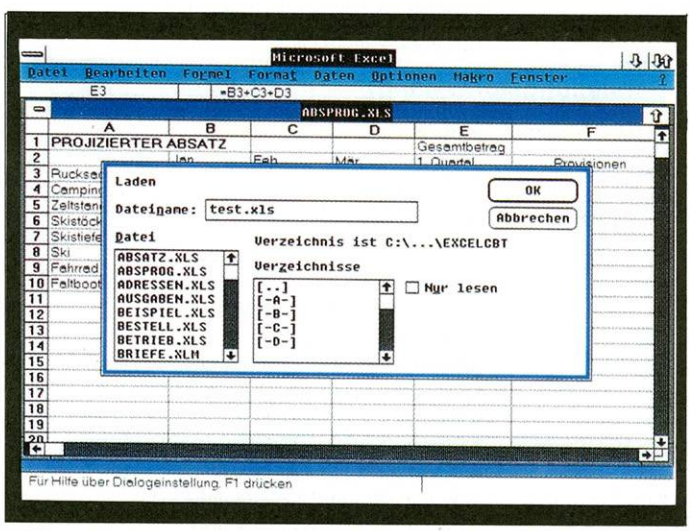


Bild 10. Die Länge eines Eingabefeldes ist nicht unbedingt begrenzt, weil es, wie beispielsweise bei Dateinamen, auch über die Fensterbreite hinausgehen kann

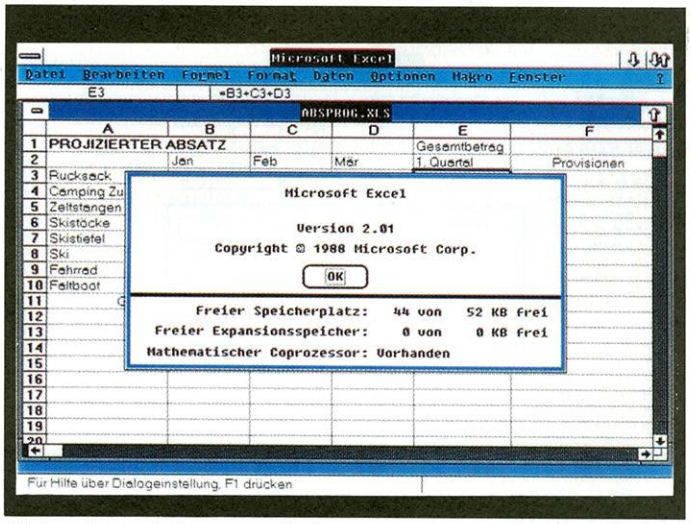


Bild 11. Ein Infofenster stellt die gewünschten Informationen ohne zusätzlichen Aufwand auf dem Monitor dar

Um Informationen auf dem Bildschirm darzustellen, bietet eine SAA-Oberfläche mehrere Anzeigearten (»panel types«), die die auszuwählenden Objekte bestimmen und die angeforderten Daten auf dem Bildschirm ausgeben.

Mehrere Anzeigearten erlauben es, die Menüs problemorientiert aufzubauen

Die wichtigsten Anzeigearten sind das Einfachauswahlmenü (»single choice menu«), das Mehrfachauswahlmenü (»multiple choice menu«), die erweiterte Auswahl (»extended choice«), das Eingabefeld (»entry«) und die Information.

Es ist bedeutsam, daß die Programmierer die Anzeigearten an die jeweils zu erfüllende Aufgabe optimal anpassen. Dabei spielt es eine große Rolle, die Auswahlkriterien soweit wie möglich zu begrenzen, damit Fehlerquellen weitgehend ausgeschaltet werden.

Als unkomplizierteste Menüart gilt die Einfachauswahl, die in Bild 7 zu sehen ist. Sie dient in diesem Fall dazu, unter der Tabellenkalkulation »Excel« die Schriftart für einen eingegebenen Text festzulegen. Ein ausgefüllter Punkt markiert den ausgesuchten Menüpunkt. Wenn der Anwender beispielsweise einen anderen Schrifttyp als »Helvetica 10 fett« anwählt, verschwindet die vorherige Markierung, und die neue Kennzeichnung erscheint.

Neben der Einfachauswahl findet die Mehrfachauswahl in vielen Situationen Verwendung. In Bild 8 legt ein Mehrfachauswahlmenü die Rahmenart für eine Zelle innerhalb der Tabellenkalkulation Excel fest. Beachten Sie bitte, daß der Anwender aus den angebotenen Eigenschaften mehrere in Anspruch nehmen kann. Es gibt aber im Vergleich zur Einfachauswahl auch Kennzeichen, die andere Merkmale löschen (in unserem Beispiel der Menüpunkt »Gesamt«).

In den vorgenannten Anzeigearten sind die Menüpunkte vorbestimmt. Selbstverständlich existieren auch Listen, deren Inhalt sich mit der Zeit ändert. In diesem Fall benutzt man die erweiterte Auswahl. Ein Beispiel aus dem Betriebssystem MS-DOS 4.01 finden Sie in Bild 9. Das rechte Fenster zählt alle Dateien auf, die sich im aktuellen Verzeichnis befinden. Alle markierten Daten sind schwarz unterlegt. Wenn die Liste einer erweiterten Auswahl über die Größe des Fensters hinausgeht, erhält der Anwender die Gelegenheit, im Inhalt zu blättern (Fachausdruck »scrollen«).

Wenn ein Programm Eingaben von Zeichenketten erwartet, um die Ausführung eines Befehls fortzusetzen, verwendet man das Eingabefeld. In Bild 10 fordert die Anwendung die Angabe eines Dateinamens. Der Benutzer kann die Dateibezeichnung entweder über Tastatur eingeben oder mit der Maus und den Pfeiltasten einen Namen aus der vorgegebenen Liste aussuchen.

Als letzte Anzeigeart kommt die Information in Frage. Sie dient letztlich dazu, programmbezogene oder befehlsorientierte Daten bekanntzugeben. Das Beispiel in Bild 10 weist auf die Version der Tabellenkalkulation Excel, die Präsenz eines mathematischen Coprozessors und den freien Speicherplatz hin.

Die vorgenannten Anzeigearten verkörpern die Grundelemente einer kommunikationsabgestimmten SAA-Oberfläche. Um den Dialog anwenderfreundlich zu gestalten, bietet es sich an, die geeigneten Komponenten geschickt zu kombinieren (Bild 11 bis 14).

Farben vereinfachen den Dialog mit dem Anwender

Farben und die grafische Hervorhebung von Teilbereichen gehören zu den wichtigsten Merkmalen der SAA-Benutzeroberfläche. Es ist kein Geheimnis, daß Farben unser Wahrnehmungs- und Auffassungsvermögen stark beeinflussen. Kräftig leuchtende Farben sind auffällig und beanspruchen sofort unsere Aufmerksamkeit, dezent anmutende Farben wirken dagegen nicht so stark. Rot oder Orange signalisieren beispielsweise Gefahr oder warnen vor etwas.

Diese Effekte machen sich auch die SAA-Vereinbarungen zunutze, die für jedes Element genau zugeordnete Farben vorsehen. Damit der Anwender sein individuelles Farbempfinden berücksichtigen kann, stehen verschiedene »Farbpaletten« zur Auswahl, die jeweils unterschiedliche, harmonierende Farben zusammenstellen.

Die Paletten für Farbbildschirme tragen die Bezeichnungen »schwarz«, »blau«, »türkis« und »weiß«. Diese Kennzeichnungen richten sich nach der vorherrschenden Hintergrundfarbe, die das Aussehen der Benutzeroberfläche maßgeblich bestimmt. Für monochrome Monitore gibt es nur die beiden Farbpaletten »weiß« und »schwarz«.

In Tabelle 4 finden Sie eine Aufstellung der Paletten für Farbbildschirme mit den zugeordneten Aufgabengebieten. Tabelle 5 differenziert diese Liste und zeigt Ihnen, welche Farbe zu welchem Bedienungselement gehört. Grundsätzlich gelten folgende Regeln in bezug auf die Farbgebung: Primär- und Sekundärfenster benutzen die weiße Palette, Hilfetextanzeigen in Sekundärfenstern die blaue Palette und Popup-Fenster abwechselnd die Paletten türkis und weiß. Des weiteren wenden Pulldown-Menüs dieselben Vorder- und Hintergrundfarben wie die Aktionszeile an. Darüber hinaus benutzt man die schwarze Palette, wenn der Bildschirm nur schwarzen Hintergrund unterstützt.

In vielen Situationen reicht die Gestaltung der Hauptanzeige einer Anwendung nicht aus, um die Kommunikation effektiv fortzuführen. In diesem Fall setzt man Popup-Fenster ein, die den Dia-

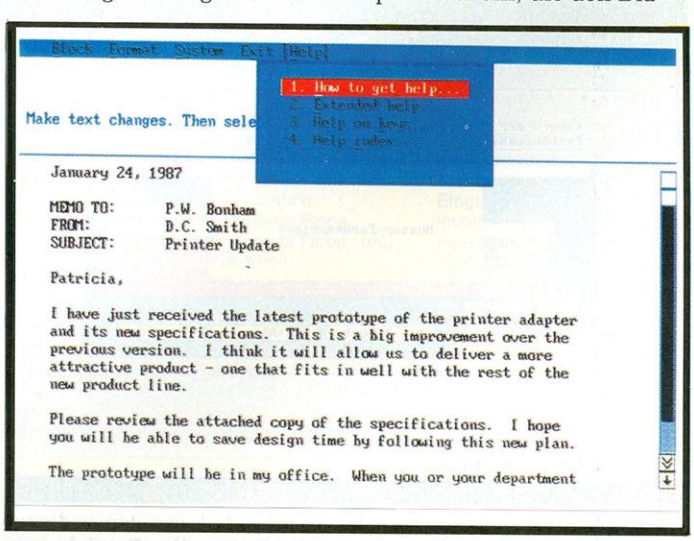


Bild 12. Die Standardfarbpalette weist jedem Bild- beziehungsweise Menüabschnitt eine bestimmte Farbe zu

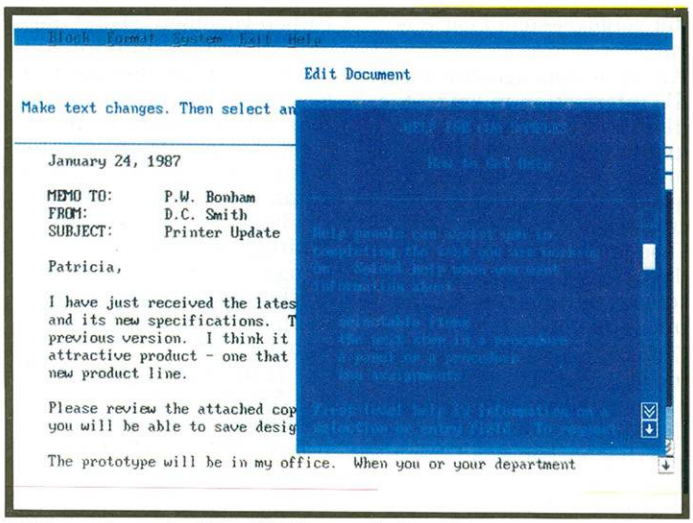


Bild 13. Die Farbe des Hilfefensters hebt sich vom Hintergrund deutlich ab

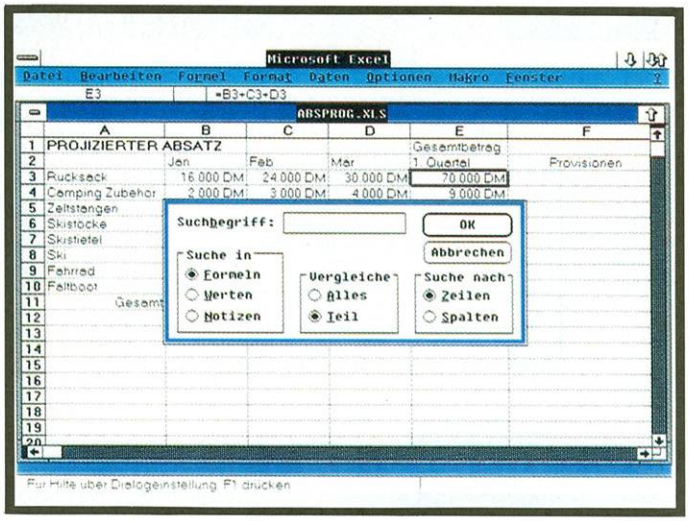


Bild 15. Ein Popup-Fenster, das man durch den Aufruf eines Menüpunktes aktiviert (»Action-Bar-Pop-Up«), bezieht sich direkt auf das Kommando und muß übersichtlich gestaltet sein

log mit dem Computer sinnvoll ergänzen.

Ein Popup-Fenster ist ein Teil des Bildschirms, der von einem Rahmen eingeschlossen ist und Informationen beinhaltet, die die Verständigung eines Anwenders mit einem anderen Fenster erweitern.

Popup-Fenster erhöhen die Übersichtlichkeit des Bildschirmaufbaus

Jedes Popup-Fenster ist mit dem darunterliegenden Fenster verknüpft und bezieht sich direkt auf ein Sprach- oder Aktionselement desselben. Wenn es sich bei dem Popup-Fenster nicht um eine Hilfetextanzeige handelt, muß der Anwender die Arbeit mit ihm beenden, bevor er mit einem anderen Popup-Fenster weiterarbeiten kann oder zum vorangegangen Bild zurückkehrt.

Die Anwendung übernimmt den Großteil der Kontrolle über das Popup-Fenster und bestimmt sowohl seinen Inhalt als auch die Gestaltung und den Aufbau. Abgesehen von der Größe des Bedienungselements legt das Programm die Position des Fensters fest. Der Benutzer hat mit Ausnahme der Eingabe und Anwahl von Daten nur die Gelegenheit, die Arbeit mit dem Fenster zu beenden und ins Hauptprogramm zurückzukehren.

Jedes Popup-Fenster besteht aus der Umrandung und dem elementbezogenen Inhalt. Die Begrenzung setzt sich in grafischen Anwendungen aus einem durch zwei Linien gebildeten Rand zusammen. Wenn es keine auf diese Weise angeordneten Zeichen gibt, benutzt man eine Linie, die um das Fenster gelegt ist. In bezug auf grafische Anwendungen ist es ratsam, alle Popup-Fenster zu schattieren. Sie erreichen diesen Effekt, indem Sie um zwei nebeneinanderliegende Begrenzungen eine zweite Linie legen.

Mit diesem Trick erwecken Sie den Anschein, daß sich ein Popup-Fenster über der darunterliegenden Fläche befindet und einen Schatten auf sie wirft. Um den Inhalt eines Popup-Fensters vom dahinterliegenden Fenster abzuheben, sollten Sie zwischen dem ausgegebenen Text und dem Rahmen Platz lassen (Bild 15 bis 17).

Jedes Popup-Fenster muß gemäß den SAA-Vereinbarungen groß genug sein, um die zu übermittelnden Angaben übersichtlich zu zeigen. Konstruieren Sie die Fenster so klein wie möglich; aber achten Sie darauf, daß die Angaben dadurch nicht unüberschaubar und ungeordnet wirken. Scheint es trotzdem einmal unvermeidlich, viele Hinweise gleichzeitig auf den Bildschirm zu bringen, dann sollte dem Anwender erlaubt sein, in den Daten zu blättern.

Der Standard gestattet es, daß ein Popup-Fenster nahezu alle SAA-Elemente wie Funktionstastenbereich, Anzeigeraum, Aktionszeile oder Eingabefelder beinhaltet. Zu bedenken ist lediglich, daß Sie

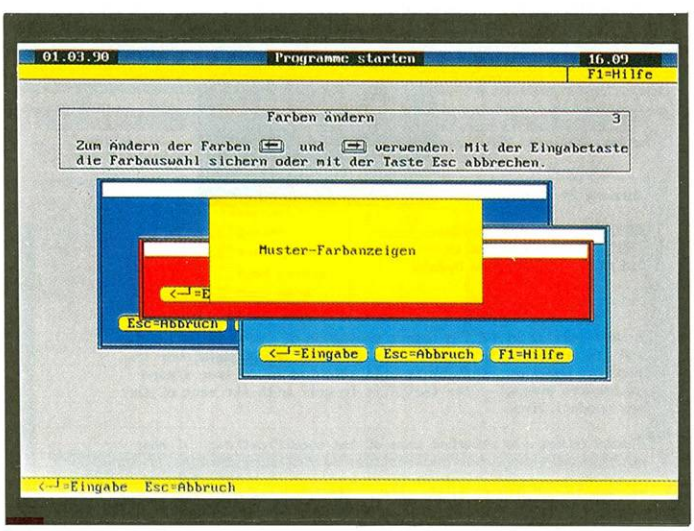


Bild 14. Jedes Popup-Fenster sollte sich farblich von den anderen unterscheiden, um die Hervorhebung des aktuellen Bereichs zu erleichtern

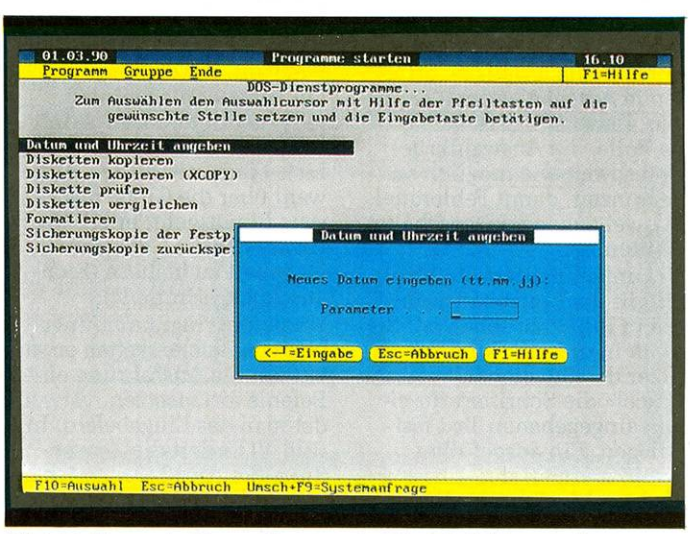


Bild 16. Wenn ein Popup-Fenster ein Eingabefeld verwaltet (»Command-Entry-Pop-Up«), ist es sinnvoll, die Form der Information vorzugeben

es auf keinen Fall mit einer Überschrift versehen dürfen. Berücksichtigen Sie bitte auch, daß Popup-Fenster eigentlich für einfache Dialoge vorgesehen sind und aus diesem Grund so schlicht wie möglich aufgebaut sein sollten.

Prinzipiell gibt es zwei Arten, ein Popup-Fenster auf dem Bildschirm zu positionieren: die objektzugeordnete und die Offset-orientierte. Die objektzugeordnete Einstellung kommt immer dann zum Einsatz, wenn sich ein Popup-Fenster auf ein Element der Basisanzeige bezieht. Stellen Sie sich zum Beispiel vor, Sie beschäftigen sich mit einer Textverarbeitung, die eine automatische Rechtschreibkorrektur bietet. In Ihrem Text befindet sich das Wort »Parallele«, und Sie wissen nicht genau, ob es richtig geschrieben ist. Also markieren Sie den zu überprüfenden Begriff und rufen die Korrekturroutine auf. Diese baut ein Popup-Fenster auf und positioniert es direkt unterhalb des zu kontrollierenden Ausdrucks. Objektzugeordnete Popup-Fenster befinden sich rechts, links, ober- oder unterhalb des zugewiesenen Elements aus der Basisanzeige. Wenn der Anwender das Popup-Fenster wieder schließt, muß das Programm den vorherigen Bildschirminhalt wiederherstellen und alle Änderungen rückgängig machen.

Offset-orientierte Popup-Fenster benutzt man hingegen, wenn zu einem auf dem Bildschirm abgebildeten Popup-Fenster ein weiteres hinzukommt. In diesem Fall ist der obere Rand des neuen Popup-Fensters unterhalb der Überschrift des alten Fensters plaziert, und der linke Rand befindet sich zwei Zeichen weiter rechts von der vorhergehenden Begrenzung.

Generell existieren drei Variationen von Popup-Fenstern: Aktionszeilen-Popups, Eingabe-Popups und Nachrichten-Popups.

Popup-Fenster sind im Gegensatz zu den anderen Fenstertypen nicht mit einer Überschrift versehen

Die Aktionszeilen-Popup-Fenster (Bild 15) reagieren auf die Auswahl eines Menüpunktes der Aktionszeile. Im dargestellten Beispiel handelt es sich um die Suche einer Zeichenkette innerhalb der Tabellenkalkulation Excel. Die Tabellenkalkulation stellt alle für die Suche wichtigen Funktionen zur Auswahl, so daß der Benutzer mit minimalem Aufwand ein optimales Resultat erhält. Aktionszeilen-Popups beschränken sich in der Regel auf eine Kommunikationsebene, damit die Ausführung des gewünschten Kommandos schnell zu einem Ergebnis führt.

Eingabe-Popup-Fenster (Bild 16) finden immer dann Verwendung, wenn der Anwender den Ablauf eines Programms durch eine Eingabe steuert oder eine Datenfolge für die Bearbeitung zur Verfügung stellt. Im abgebildeten Muster ändert der Benutzer bei Bedarf das Systemdatum. Berücksichtigen Sie bitte, daß die Routine das Format der Eingabe vorschreibt und auch auf dem Monitor ausgegeben hat.

Oberflächenelement	weiße Palette	weiße Palette (für Monochrombildschirme)	türkise Palette	schwarze Palette	schwarze Palette (für Monochrombildschirme
Aktionszeile und Pulldown-Menü					
– Hintergrund	türkis	weiß	weiß	weiß	weiß
– Auswahl	schwarz	schwarz	schwarz	schwarz	schwarz
– Auswahlbuchstaben	unterstrichen	unterstrichen	unterstrichen	unterstrichen	unterstrichen
Anzeigeraum					
– Hintergrund	weiß	weiß	türkis	schwarz	schwarz
– Systeminformation	blau	schwarz	blau	türkis	weiß
– Klammern	blau	schwarz	blau	türkis	weiß
– Spaltenüberschriften	blau	schwarz	blau	türkis	weiß
– hervorgehobener Text	braun	schwarz	gelb	gelb	weiß
– Eingabefelder	blau	schwarz	blau	türkis	weiß
– Gruppenüberschriften	blau	schwarz	blau	türkis	weiß
– Anweisungen	blau	schwarz	blau	türkis	weiß
– normaler Text	blau	schwarz	blau	türkis	weiß
– Anzeigeüberschriften	blau	schwarz	blau	türkis	weiß
– Anzeige-ID	blau	schwarz	blau	türkis	weiß
– Auswahl	schwarz	schwarz	schwarz	weiß	weiß
– nicht verfügbare Auswahl	reduzierte Helligkeit	reduzierte Helligkeit	reduzierte Helligkeit	reduzierte Helligkeit	reduzierte Helligkeit
– Auswahlbuchstaben	unterstrichen	unterstrichen	unterstrichen	unterstrichen	unterstrichen
– Inhalt des Eingabefeldes					
– normale Eingabe	schwarz	schwarz	schwarz	weiß	weiß
– hervorgehobene Eingabe	braun	schwarz	gelb	gelb	weiß
– Blätterinformation	blau	schwarz	blau	türkis	weiß
– Trennungslinie	blau	schwarz	blau	türkis	weiß
– Rand der Popup-Fenster	blau	schwarz	blau	türkis	weiß
Funktionstastenbereich					
– Hintergrund	weiß	weiß	türkis	schwarz	schwarz
– Auswahl	schwarz	schwarz	schwarz	weiß	weiß
Hervorhebung einer Auswahl					
– Auswahl-Cursor					
– Anwahlfeld	schwarz	schwarz	schwarz	weiß	weiß
– Eingabefeld	Text-Cursor im Eingabefeld	Text-Cursor im Eingabefeld	Text-Cursor im Eingabefeld	Text-Cursor im Eingabefeld	Text-Cursor im Eingabefeld
– hervorgehobene Auswahl	invertierte Farbe*	invertierte Farbe*	invertierte Farbe*	invertierte Farbe*	invertierte Farbe*
– ausgewählt	invertierte Farbe* und Kästchen	invertierte Farbe* und Kästchen	invertierte Farbe* und Kästchen	invertierte Farbe* und Kästchen	invertierte Farbe* und Kästchen
Fehlerbetonung	rot	schwarz	rot	rot	weiß
Meldungen					
– Hinweis					
– Hintergrund	weiß	weiß	weiß	weiß	schwarz
– Vordergrund	schwarz	schwarz	schwarz	schwarz	weiß
– Warnung					
– Hintergrund	gelb	weiß	gelb	gelb	schwarz
– Vordergrund	schwarz	schwarz	schwarz	schwarz	weiß
– Fehler					
– Hintergrund	rot	schwarz	rot	rot	weiß
– Vordergrund	weiß	weiß	weiß	weiß	schwarz

* Mit »invertierter Farbe« ist ein Wechsel zwischen Vorder- und Hintergrundfarbe gemeint.

Tabelle 5. Auf monochromen Bildschirmen herrscht eine andere Farbzuordnung als bei Farbsystemen

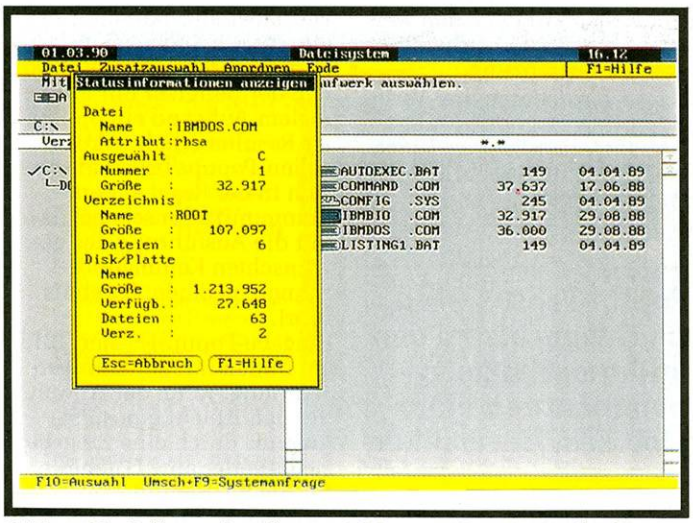


Bild 17. Ein Informationsfenster (»Message-Pop-Up«) gibt die vom Benutzer geforderten Daten prägnant und themenbezogen aus

Die Tastenbelegung im SAA-Standard

Funktion	Taste
Abbrechen	[Esc] oder [F12]
Abwärts	[Cursor abwärts]
Aktualisieren	[F5]
Aufwärts	[Cursor aufwärts]
Bedienerführung	[F4]
Befehl	[Shift-F9]
Datenanfang	[Ctrl-Home]
Datenende	[Ctrl-End]
Einfügen	[Ins]
Eingabe	[Return]
Erweiterte Hilfe	[F2]
Feldanfang	[Ctrl-Cursor links]
Feldende	[Ctrl-Cursor rechts]
Folgendes Wort	[Alt-Cursor rechts]
Hilfe	[F1]
Hilfeindex	[F11]
Hilfetext für Tasten	[F9]
Inhaltsverzeichnis der Hilfstexte	[Shift-F11]
Links	[Ctrl-PgUp] oder [Shift-F7]
Linkspfeil	[Cursor links]
Löschen	[Del]
Löschen bis Feldende	[Ctrl-Del]
Neue Zeile	[Return]
Rechts	[Ctrl-PgDn] oder [Shift-F8]
Rechtspfeil	[Cursor rechts]
Rückschritt	[Backspace]
Rückwärts	[PgUp] oder [F7]
Sperren des Blätterns	[Scroll Lock]
Sprung rückwärts	[Shift-F6]
Sprung vorwärts	[F6]
Sprung zur Aktionszeile	[F10]
Tabulatorrücksprung	[Shift-Tab]
Tabulatorsprung	[Tab]
Tastenanzeige	[Shift-F1]
Verlassen (zurück zum Vorausgegangenen)	[F3]
Verlassen (zurück zum Anfang)	[Shift-F3]
Vorhergehendes Wort	[Alt-Cursor links]
Vorwärts	[PgDn] oder [F8]
Zeilenanfang	[Home]
Zeilenende	[End]
Zurückholen	[F9]

Tabelle 6. Diese Übersicht zeigt, welche Funktionen der SAA-Standard einigen Tasten fest zuordnet

Mit dem Nachrichten-Popup (Bild 17) stellt sich die letzte Abwandlung der Popup-Fenster vor. Sie bildet die Hauptmenge der Popup-Versionen und eignet sich dazu, alle wichtigen Informationen wie Statusmeldungen, Warnungen oder Fehler zu übermitteln. Das Beispiel zeigt eine Statusmeldung des Betriebssystems in einem Fenster, das keine funktionsbezogenen Eingaben erfordert. Auf die Nachrichten und die damit zusammenhängenden Gestaltungsvarianten wird etwas später ausführlicher eingegangen.
Zunächst befassen wir uns mit einer besonderen Abwandlung der Popup-Fenster, der Hilfetextanzeige.
Die Hilfefunktion eines Programms sollte laut SAA-Standard an jeder Stelle aufrufbar sein. Sie unterstützt den Anwender in komplizierten Situationen und steht ihm zur Seite, wenn er Befehle vergessen hat oder die Bedeutung einer Mitteilung nicht versteht. Die Hilfefunktion ersetzt weder ein für Lehrzwecke gedachtes Handbuch noch einen Tutorial-Befehl.

Die kontextsensitive Hilfefunktion bietet Rat in jeder Notlage

Ein Tutorial erklärt einem neuen Benutzer, wie er ein Programm zu bedienen hat. Die Hilfefunktion erinnert ihn hingegen daran, wie eine Anwendung funktioniert. Die Hilfetextanzeige richtet sich nach dem Objekt, auf dem sich die Eingabemarke befindet (»kontextsensitive Hilfe«). Ist kein Cursor vorhanden, bietet sie pauschalen Beistand an.
Über die objektbezogenen Daten hinaus enthält die Hilfetextanzeige folgende Auswahlkriterien: ein Verzeichnis der Elemente, für die sie Hilfe bereitstellt, eine Zusammenstellung der Tastenkombinationen mit ihren Bedeutungen, erweiterte Bemerkungen über die Benutzeroberfläche sowie Bedienungshinweise zu der Hilfefunktion. Zusätzlich ist es gestattet, eine Tutorial-Funktion einzubauen, die einen auf die Anwendung bezogenen Lehrgang aufruft.
Richten Sie Ihr Augenmerk bitte darauf, daß Hilfsinformationen ausnahmslos in dem gleichen Fenster erscheinen und den jeweils vorher dargebotenen Fensterinhalt ersetzen. Damit Sie die Hilfetextanzeige sprachenabhängig formen können, sollten die Hilfstexte in einer Datei abgelegt sein.

Der Großteil der Kommunikation mit dem Anwender besteht aus Nachrichten

Nach den grundlegenden Informationen zum Bildschirmaufbau und zur Dialoggestaltung beschäftigen wir uns im folgenden mit den Nachrichten (englisch »messages«) an den Benutzer. Sie bestehen aus drei Nachrichtenkategorien: den Hinweisen (englisch »notifications«), Warnungen (englisch »warnings«) und Fehlermeldungen (englisch »criticals«).
Normale Hinweise (Bild 18) finden immer dann Verwendung, wenn eine vom Anwender aufgerufene Funktion fehlerfrei abläuft oder abgelaufen ist. Sie dienen in der Mehrzahl der Fälle dazu, dem Benutzer eine Statusmeldung zu geben. Stellen Sie sich beispielsweise vor, daß der Anwender ein Programm gestartet hat, das eine Diskette formatiert. Wenn die Formatierroutine läuft, erhält er auf dem Bildschirm die Information, daß der Vorgang läuft und wieviel Prozent der Diskette schon formatiert sind. Auf diese Weise erfährt der Anwender, wann sein Programm voraussichtlich beendet sein wird. In der abgebildeten Situation weist das System darauf hin, daß die aktivierte Formatierroutine beendet ist. Hinweise erwarten vom Anwender keine situationsbezogene Reaktion und dienen ausschließlich der Information. Für den Einsatz dieses Nachrichtentyps existieren drei Grundvoraussetzungen. Hinweise finden erstens dann Anwendung, wenn nicht genau ersichtlich ist, daß ein Vorgang stattfindet oder stattgefunden hat. Darüber hinaus gehört es zum guten Programmierstil, den Anwender über signifikante Änderungen im Anzeigeraum oder im internen Programmablauf aufzuklären.

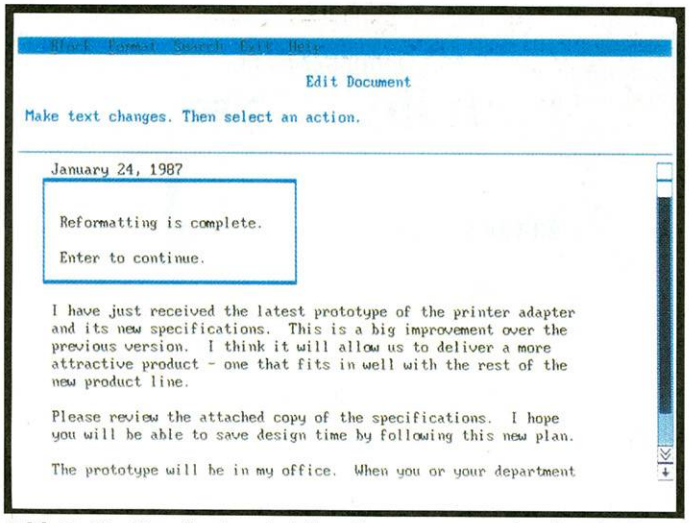


Bild 18. Ein Standardnachrichtenfenster verwendet die gleichen Farben wie der Informationsteil der SAA-Anwendung

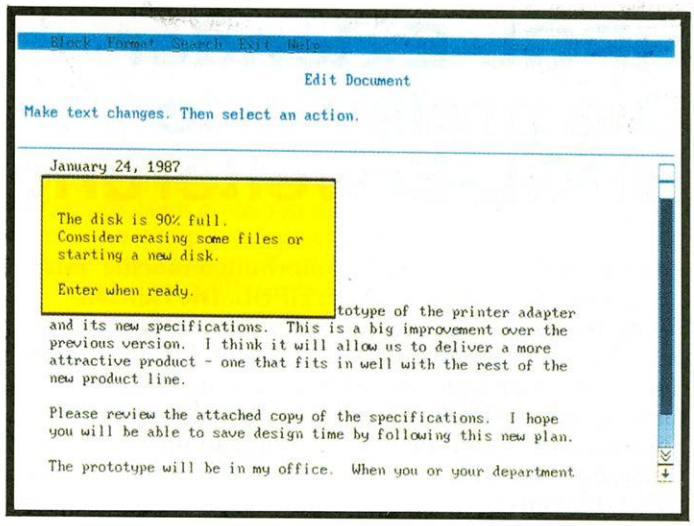


Bild 19. Das SAA-Programm hebt eine Warnung durch eine zum Hintergrund des Informationsteils verschiedene Farbe hervor

Der Einsatz eines Hinweises empfiehlt sich zweitens, wenn eine gewisse Zeitspanne vergeht, bevor die Programmausführung weitergeht. Diese Situation tritt zum Beispiel ein, wenn eine Tabellenkalkulation längere Zeit benötigt, um ein Arbeitsblatt durchzurechnen, oder eine Druckroutine einen Augenblick beansprucht, um Daten für die Ausgabe auf einem angeschlossenen Drucker vorzubereiten.

Des weiteren benutzt man Hinweise, um Benutzer von Multitasking-Systemen darauf aufmerksam zu machen, daß eine im Hintergrund ablaufende Anwendung beendet ist.

Fehler und Warnungen sollten sich von den Standardmeldungen abheben

Neben den Hinweisen gibt es noch die Kategorie der Warnungen (Bild 19). Sie weisen darauf hin, daß das Programm mit einer Unklarheit konfrontiert ist, auf die der Benutzer eventuell reagieren muß. Im dargestellten Beispiel teilt das Programm mit, daß die Diskette zu 90 Prozent belegt ist. Wenn der Anwender es für erforderlich hält, löscht er Dateien auf dem Datenträger. Vertritt er hingegen die Meinung, daß der freie Speicherplatz ausreicht, kann er mit dem Programm weiterarbeiten. Als letzter Nachrichtentyp sind die Fehlermeldungen von Bedeutung (Bild 20). Sie verlangen eine problemorientierte Antwort, die die Ursache der Fehlermeldung beseitigt. Das Programm gibt eine Fehlermeldung aus, wenn der Anwender eine Funktion falsch oder zu einem regelwidrigen Zeitpunkt aufgerufen hat oder wenn das System eine Sachlage antrifft, die eine Reaktion erfordert.

Als Beispiel dient die in Bild 20 angenommene Situation, in der eine Routine Daten von einem Laufwerk lesen will, in dem sich kein Datenträger befindet. Sie müssen nun entweder eine Diskette einlegen und die Aktion wiederholen oder die Funktion abbrechen und ins Hauptprogramm zurückkehren.

Alle Nachrichten sollten sich in Popup-Fenstern befinden, die allerdings keine Fensterüberschrift in Anspruch nehmen. Des weiteren begleitet ein Piepton jede Warnung und jede Fehlermeldung, wenn der Anwender dieses akustische Signal nicht über ein Kommando abgeschaltet hat.

Beachten Sie bitte in bezug auf die Kolorierung, daß bei Farbbildschirmen die einzelnen Nachrichtentypen unterschiedlich eingefärbt sein sollten. Zur Veranschaulichung wählen wir eine Oberfläche, die im allgemeinen schwarze Schrift auf weißem Grund benutzt. Während Hinweise auf dieselbe Art und Weise aufgebaut sind, stellt die Applikation Warnungen leicht hervorgehoben dar (beispielsweise durch einen gelben Hintergrund) und unterlegt Fehlermeldungen mit einer auffälligen Farbe. In dem zuletzt angeführten Fall eignet sich besonders ein gut sichtbares Rot.

Wie Sie aus den vorgenannten Erläuterungen ersehen, ordnet der Standard bestimmten Tasten und Tastenkombinationen verbindlich Funktionen zu. Diese Tastaturbelegung bleibt in einer SAA-Anwendung zu jeder Zeit bestehen und darf nicht verändert werden. Tabelle 6 zeigt Ihnen alle auf PCs reservierten Kombinationen in alphabetischer Reihenfolge.

Mit den SAA-Vereinbarungen sorgt erstmals in der PC-Welt ein Standard für geordnete Verhältnisse. Wenn sich die SAA-Konventionen durchsetzen, erhalten die Anwender eine normierte Benutzeroberfläche, die ihnen den Umgang mit Standardapplikationen und problemorientierten Anwendungen wesentlich erleichtert.

Fenster, Pulldown-Menüs und Mausunterstützung sorgen für übersichtliche und leicht zu bedienende Programme, die hohen Ansprüchen gerecht werden.

(Markus Zietlow/ks)

Literaturhinweis:
»Common User Access Panel Design and User Interaction«, IBM, Florida, USA

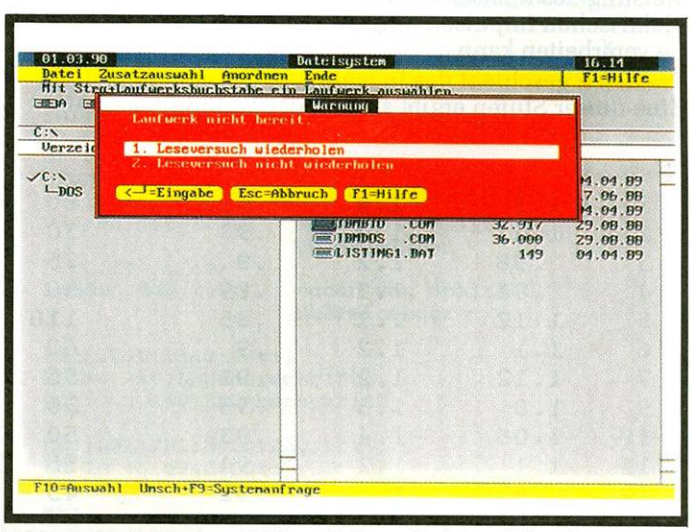


Bild 20. Ein kritischer Fehler erhält optisch Nachdruck, indem man ihn mit einer im Vergleich zum restlichen Hintergrund sehr auffälligen Farbe unterlegt

HPGL-Extender – Die preiswerte HPGL-Erweiterung

Wer sich mit seinem Plotter unterhalten möchte, muß auch seine Sprache sprechen: HPGL. Die Hewlett-Packard-Graphic-Language, wie sie eigentlich heißt, enthält eine Vielzahl von Anweisungen für den Plotter, um Symbole und grafische Elemente zu zeichnen. Allerdings ist der Umfang von HPGL nicht immer ganz ausreichend. Damit der Plotter noch komplexere Anweisungen versteht, erweitern Sie HPGL mit Hilfe des HPGL-Extender.

Es gibt eine ganze Reihe von Grafikprogrammen, mit denen sich auf meist recht einfache Weise eindrucksvolle und schöne Bilder produzieren lassen.

Einfache Handhabung bedeutet aber auch fast immer eine Einschränkung in der Gestaltungsfreiheit. Was können Sie tun, wenn einige Zeichenelemente anders aussehen sollen, als sie von Ihrem Grafikprogramm angefertigt werden, und welche Möglichkeiten haben Sie, den Aufbau einer Zeichnung aktiv zu bestimmen?

Ein Zeichenprogramm kann noch so gut durchdacht und ausgefeilt sein – optimal für alle Anwendungen kann es dennoch niemals sein.

Wer außergewöhnliche Anforderungen an seine Zeichnungen stellt, schreibt sich sein Zeichenprogramm am besten selbst. Der HPGL-Extender hilft Ihnen dabei.

Dieses Programm ist ein aus der Praxis heraus entwickeltes Werkzeug und enthält die grundlegenden Funktionen zur Herstellung von Zeichnungen. Es kann und soll aber auch als Gerüst dienen, in das Sie Ihre persönlichen Wünsche mit einbeziehen.

Wenn Sie mit Hilfe von maschinellen Zeichengeräten, wie beispielsweise einem Plotter, eine Grafik zeichnen wollen, müssen Sie dem Gerät in geeigneter Form mitteilen, was es machen soll. »Zeichne eine Koordinatenachse« wäre zum Beispiel eine für einen Menschen verständliche Anweisung. Eine Zeichenmaschine erkennt aber nur eine Reihe von elektrischen Impulsen als Anweisung. Die verbale Anweisung des Menschen muß also in eine Sequenz von elektrischen Impulsen umgesetzt werden, die eine Maschine verarbeiten kann.

Praktisch geschieht das in mehreren Umsetzungsstufen. Eine dieser Stufen ergibt sich an der Verbindungsstelle

```
1     1.1      1.2      .95       20
2     1.2      1.3      .93       30
3      .98     1.2      .9        35
4      .88     1.1      .89       50
5     1.12     1.1      .85       110
6     1.1      1.2      .9        60
7     1.12     1.2      .91       55
9     1.0      1.3      .95       30
10    1.05     1.4      .93       50
11    1.1      1.2      .94       55
12     .95     1.1      .91       45
```

Bild 2. Die exemplarische Datei »demo.dat« enthält Werte zur Darstellung eines Kurvenzugs

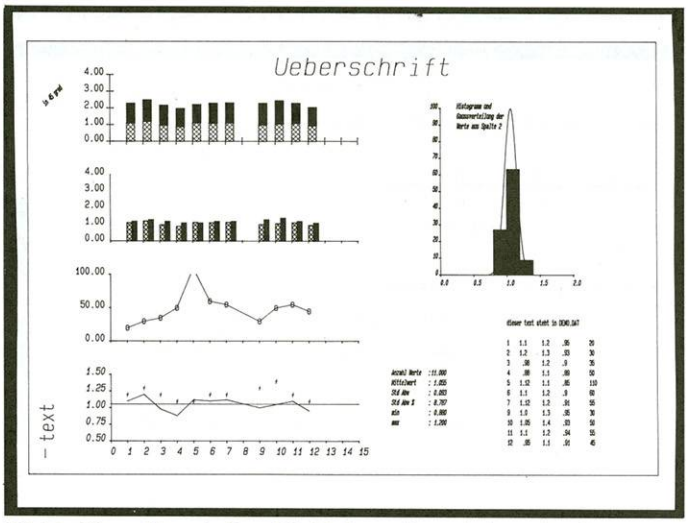


Bild 1. Wenn Sie mit dem HPGL-Extender arbeiten, ist diese Zeichnung im Handumdrehen entworfen

zwischen Rechner und Grafikausgabegerät. Mit dieser Stufe befaßt sich der HPGL-Extender. Hardware-mäßig ist sie meistens als serielle oder parallele Schnittstelle ausgelegt.

Auf der Software-Seite ist sie bei vielen Geräten als HPGL definiert. In dieser Sprache wird einem Grafikausgabegerät mitgeteilt, was es wissen und tun muß, um eine Zeichnung anzufertigen. Prinzipiell besteht HPGL aus über fünfzig Befehlen, die nach bestimmten Regeln aufgebaut sind. Die ersten beiden Buchstaben sind ein Kürzel für die Funktion, die ausgeführt werden soll. Dann folgen Daten, deren Reihenfolge von der jeweiligen Funktion abhängig ist. Die Daten sind duch Kommata getrennt.

Der komplette Befehl wird durch ein Semikolon abgeschlossen. So holt zum Beispiel »SP2;« (select pen 2) den Plotter-Stift 2 aus der Halterung. »PA1200,2500;« (plot absolut x, y) bewegt den Stift zur angegebenen Position. Das Kommando »PD;« (pen down) senkt den Stift auf die Zeichenunterlage ab.

Mit HPGL-Dateien ist der Austausch von Zeichnungen zwischen Grafikprogrammen kein Problem

Eine Reihe von Grafikprogrammen gibt neben der direkten Plotter-Ansteuerung auch die Ansteuerungssequenzen in Textdateien aus. Solche HPGL-Dateien können dann von anderen Programmen gelesen werden. Auf diese Weise wird ein Transfer von Zeichnungen zwischen verschiedenen Programmpaketen möglich.

Mit HPGL kann im Prinzip jede Art von Grafik formuliert werden. Das ist bereits durch die grundlegenden Funktionen Stiftauswahl und Stiftbewegung gegeben. Etwas komplexere Funktionen, beispielsweise für die Balken- und Textausgabe, heben den Komfort enorm. Damit ist allerdings der Befehlsumfang von HPGL schon fast erschöpft.

Es ist gar nicht besonders kompliziert, mit HPGL »von Hand« eine statistische Präsentationsgrafik mit Koordinatensytem, Achsenbeschriftung und Wertelinien oder Balkendiagramm zu zeichnen. Sie benötigen zuerst ein Skizzenblatt, auf dem Sie alle Linien und Texte an den richtigen Positionen vorzeichnen. Als nächstes brauchen Sie nur noch eine Menge Geduld, um alles, was sich vom weißen Papier abhebt, in HPGL-Text zu übersetzen. Eine einfache Präsentationsgrafik erreicht schnell einen Umfang von mehreren KByte HPGL-Text.

HPGL als direkter Draht zwischen Mensch und Computer ist eben doch nicht so optimal. Für diesen Zweck wurde HPGL allerdings auch nicht entwickelt. Für diese Aufgabe sind eine höhere Sprache und ein Übersetzungsprogramm, das den Befehl "Zeichne eine Koordinatenachse" in HPGL umsetzt, notwendig. Diese Aufgabe erfüllt - zumindest ansatzweise - der HPGL-Extender.

Die Anweisung »Zeichne eine Koordinatenachse« reicht auch für einen Menschen nicht ganz aus. Für eine Maschine natürlich ebensowenig. Beide brauchen weitere Angaben über Papiergröße, Achsenbemaßung und Lage der Achse. Abgesehen davon soll die Syntax, also die sprachliche Form der Anweisung, so aussehen, daß sie logisch aufgebaut und damit leicht erlernbar ist. Für die Syntax des HPGL-Extenders trifft dies zu (Tabelle).

Hängen Sie sich bei der Entwicklung von eigenen Sprachelementen an den Sprachformalismus von HPGL an, so hat das den Vorteil, daß Sie original HPGL-Befehle mit in eigene Befehlslisten einbauen können. Dadurch werden sie vom Übersetzerprogramm einfach durchgeschleift - also in derselben Form weitergeleitet. In gewisser Weise sind die neuen Sprachelemente als Erweiterung zur HPGL zu

```
 demo.plo
NP;                 not to plotter
NM;                 not to monitor
XP380;YP260;        papiergroesse
IN;
 SP1;               hole stift
COsw,rt,gn,bl,ge,or,hb,vi;
                    aktuelle farbbelegung
SPsw;
MO10,10;PL10,15;    strich
RE0,0,380,260;      rechteck

LW4;LH8;LS87;LD90; beschriftungsaussehen
MO15,20;            position
LBtext;             textausgabe

LW1;LH3;LS90;LD0;   beschriftungsaussehen
MO290,90;           position
LBdieser text steht in DEMO.DAT;
LB ;                leerzeile
LFdemo.dat;         schreibe text aus datei

SPsw;               koordinaten block------k
TL0,1;              tic-länge              k
XC50;YC20;          koordinatennullpunkt   k
                                           k
LW2;LH4;LS80;           x                  k
DE6,0;                  x                  k
XB0;XE15;XS1;XT10;      x                  k
XA7;                    x                  k
                                           k
LW2;LH4;LS80;           y                  k
DE5,2;                  y                  k
YB.5;YE1.5;YS.25;YT10;  y                  k
YA7;                    y                  k

SPor;               linie aus der ersten
LNdemo.dat,1,1;     spalte nur als linie

MO220,60;
LH3;LW1;
DE6,3;
STdemo.dat,1,1;     statistic + ausdruck

LM;                 mittelwertlinie

SPbl;               linie aus der ersten
LNdemo.dat,2,2;     spalte nur als
                    punktemarkierung


SPsw;               koordinaten block------k
XC250;YC120;                               k
DE5,1;                  x                  k
XB0;XE2;XS.5;XT20;      x                  k
XA7;                    x                  k
DE5,0;                  y                  k
YB0;YE100;YS10;YT10;    y                  k
YA7;                    y                  k


MO260,220;
LBHistogramm und;
LBGaussverteilung der;
LBWerte aus Spalte 2;
```

```
SPge;
HIdemo.dat,1,10;    histogramm datei,
                    spalte, klassen
SPrt;
GA;                 gauss verteilung

SPsw;               koordinaten block------k
XC50;YC80;          koordinaten nullpunkt  k
LW2;LH4;LS80;           x                  k
DE6,0;                  x,                 k
XB0;XE15;XS1;XT10;      x                  k
XA3;                    x                  k
LW2;LH4;LS90;           y                  k
DE5,2;                  y                  k
YB0;YE100;YS50;YT20;    y                  k
YA7;                    y                  k

SYo;
SPrt;
LNdemo.dat,4,7;

                    balkendiagramm von
                    datei demo.dat
SPsw;               koordinaten block------k
YC140;                                     k
XA3;                    x                  k
YB0;YE4;YS1;YT10;       y                  k
YA5;                    y                  k

BW.3;               balkenbreite einheiten

SPor;
FT4,20,45;          balkenaussehen schraffur
                    abstand, winkel
PBdemo.dat,1,2;     daten von spalte 1,
                    nebeneinander

SPbl;
FT3,5,0;
PBdemo.dat,2,2;     daten von spalte 2,
                    nebeneinander

SPsw;               koordinaten block------k
YC200;                                     k
TL2,2;                                     k
XA3;                    x                  k
YB0;YE4;YS1;YT10;       y                  k
YA7;                    y                  k

BW.5;
SPgn;
FT4,20,45;
PBdemo.dat,1,1;     daten von spalte 1,
                    übereinander
SPrt;FT3,5,45;
PBdemo.dat,2,1;     daten von spalte 2,
                    übereinander

LH10;LW6;LS80;SPvi;
MO150,240;LBUeberschrift;
MO150.3,240;LBUeberschrift;

LH3;LW1;SP1;MO10,220;LD45;
LB in 45 grad;LD0;

ex;
```

Bild 3. Der Inhalt der Datei »demo.plo« stellt die Grundlage für die Zeichnung aus Bild 1 dar

verstehen. An der Entstehung der Demonstrationszeichnung »demo.plo« (Bild 1, 2 und 3) werden nun die Elemente des Extenders erklärt. Für den Aufbau einer Extender-Datei gelten einige Konventionen. Befehle beginnen am Anfang einer Zeile und enden mit einem Semikolon. In einer Zeile dürfen mehrere Befehle hintereinander stehen. Das Semikolon ist nicht Bedingung, wenn der Befehl der letzte einer Zeile ist. Zeilen oder Zeichenfolgen, die nicht mit einem Buchstaben beginnen, sind Kommentare. Durch das Einfügen von Leerzeichen am Zeilenanfang wird also der Rest der Zeile nicht mehr bearbeitet.

Mit einfachen Elementen wird die Hewlett-Packard-Graphic-Language sinnvoll erweitert

Zu Beginn der Zeichnungsdatei wird mitgeteilt, ob der Plotter, der im Normalfall direkt angesteuert wird, mitlaufen soll. »NP;« (no plotter) setzt den betreffenden Schalter im Programm. Das gleiche gilt für »NM;« (no monitor) in bezug auf die Bildschirmausgabe. Der erzeugte HPGL-Text befindet sich in »hpgl.plo«, von wo aus er weitergegeben wird. »hpgl.plo« läßt sich auch mit dem MS-DOS-Befehl »print« in einer Warteschlange an den Plotter senden.

Befehl	Bezeichnung	Erläuterung
NP;	no plotter	keine Ausgabe zum Plotter
NM;	no monitor	keine Ausgabe an den Monitor
IN;	initialize	Plotter- und Programm-initialisierung
XP<Real>; YP<Real>;	papersize	Papiergröße in mm
XB<Real>; YB<Real>;	begin	Achsenbeginn in Einheiten
XE<Real>; YE<Real>;	end	Achsenende in Einheiten
XS<Real>; YS<Real>;	step	Schrittweite in Einheiten
XT<Real>; YT<Real>;	tic	Tic-Abstand in mm
XC<Real>; YC<Real>;	coordinate	Koordinatenverschiebung in mm
XA<Int>; YA<Int>;	axis	zeichnet Achse (Bitschalter)
MO<Real>,<Real>;	move	bewegt Stift nach (X,Y)
PL<Real>,<Real>;	plot	zeichnet Linie nach (X,Y)
LW<Real>;	label width	Beschriftungszeichen, Breite in mm
LH<Real>;	label height	Beschriftungszeichen, Höhe in mm
LS<Real>;	label slant	Beschriftungszeichen, Neigung in Grad
LD<Real>;	label direction	Beschriftungsrichtung in Grad
DE<Int>,<Int>;	decimal	Dezimalzahl (Gesamtstellen, Nachkommastellen)
LB<text>;	label	schreibt Text
LF<String>;	Label from File	schreibt Text von Datei <String>
TL<Real>,<Real>;	tic length	Tic-Länge in mm (vor und hinter der Achse)
SY<Character>;	symbol	Symbolausgabe an Punktstelle
LN<String>,<Int>,<Int>;	line	Liniendiagramm (Datei, Spalte, Bitschalter)
PB<String>,<Int>,<Int>;	bar	Balkendiagramm (Datei, Spalte, Bitschalter)
ST;	statistics	schreibt statistische Daten
LM;	line meanvalue	Mittelwertslinie
CO<String>;	color	Farbe der Plotter-Stifte
SP<String>;	select pen	holt Stift mit Farbe <String>
RE<Real>,<Real>,<Real>,<Real>;	rectangle	Rechteck (x1, y1, x2, y2 in mm)
BA<Real>,<Real>,<Real>,<Real>;	bar absolute	Balken absolut (x1, y1, x2, y2 in mm)
BR<Real>,<Real>,<Real>,<Real>;	bar relative	Balken relativ (x1, y1, x2, y2 in Einheiten)
BW<Real>;	bar width	Balkenbreite in Einheiten
ST<String>,<Int>,<Int>;		Statistik (Datei, Spalte, Bitschalter)
HI<String>,<Int>,<Int>;		Histogramm (Datei, Spalte, Klassen)
GA;		Gaußsche Kurve der letzten Einstellung
EX;	exit	Ende der zu beachtenden Befehle

Tabelle. Die Befehle des HPGL-Extenders im Überblick

Die Anwendungsfläche des Papiers ist im Fall von DIN-A3-Millimeterpapier 380 mm in X- und 280 mm in Y-Richtung. Dies erfährt der Extender über die Anweisung »XP380;YP280;«.

Der Befehl »IN;« ist zwar original HPGL-Text (er initialisiert den Plotter), löst aber auch im Extender eine Reihe von Null- und Voreinstellungen aus.

Den Stift 1 nimmt der Plotter auf den Befehl »SP1;« (select pen) aus der Halterung. Der Extender läßt den Befehl auch in der Form »SPblack;« zu. Der schwarze Stift muß dann natürlich als Nummer 1 eingesteckt sein. Die Zuordnung »black« zu Stift Nummer 1 wird im Programm als Voreinstellung festgelegt. Sie kann aber über den Extender-Befehl »COschwarz,rot,grün,blau;« (color farbreihe) an beliebiger Stelle verändert werden. Danach versteht der Extender den Befehl »SPschwarz;«.

Elementar sind die beiden folgenden Funktionen: »MO10,10;« (move) und »PL10,20;« (plot). »MO« bewegt den Stift zu der in Millimetern angegebenen X/Y-Position, während »PL« nach dem Absenken des Stifts eine Linie zur neuen angegebenen Position zeichnet.

Möchten Sie ein Rechteck zeichnen, sind ein Move- und vier Plot-Befehle nötig. Die Anweisung »RE0,0,380,260;« (rectangle x1,y1,x2,y2) führt ebenfalls zu einem Rechteck, macht aber vier von fünf Befehlen überflüssig.

Eine ganze Befehlsgruppe ist der Ausführung von Beschriftungen gewidmet: »LW1;LH3;LS87;LD0;« (label width, label height) bestimmen Zeichenbreite und Zeichenhöhe in Millimetern sowie (label slant, label direction) Zeichenneigung und Beschriftungsrichtung in Grad. »LBtext;« (label) schreibt den anhängenden Text und hängt die Zeichenfolge CR/LF (carriage return/line feed) an. »LFdemo.dat;« schreibt den Text, der in der angegebenen Datei enthalten ist.

Doch nun zu einem anderen Anwendungsbereich. Zur Darstellung eines Kurvenzuges tragen Sie X- und Y-Werte in ein Koordinatensystem ein. Als X-Werte sollen die Daten aus der ersten Spalte der Datei »demo.dat« (Bild 2) und als Y-Werte die Daten aus der zweiten Spalte gelten. Zuvor müssen durch Analyse der Daten die Parameter der Achsen in sinnvoller Weise ausgewählt werden.

Die Definitionen erreichen Sie mit der Befehlsfolge »XC50;XB0;XE15;XS1;XT10;«. »XC« ist die Verschiebung des Koordinatennullpunktes in X-Richtung vom Papierrand, angegeben in Millimetern. »XB«, »XE« und »XS« sind Beginn-, End- und Schrittwert der X-Achse in Werteeinheiten. »XT« gibt den Tic-Abstand in Millimetern an, das ist der Abstand zwischen zwei Schritten. Sinngemäß gilt das gleiche auch für die Y-Achse. Die Darstellung der Tics stellen Sie mit »TL0,1;« ein. Die erste Zahl ist die Länge der Tics in Millimetern über der X-Achse und rechts von der Y-Achse, die zweite Zahl die Tic-Länge zu den anderen Seiten der Achsen. Mit »DE6,0;« bestimmen Sie die Dezimalzahlendarstellung. Die erste Zahl betrifft die Anzahl an dargestellten Ziffern ohne Dezimalpunkt, die zweite Zahl ist die Anzahl der Nachkommastellen.

Der HPGL-Extender ermittelt automatisch statistische Daten

Das Zeichnen der Achse übernimmt der Befehl »XA7;«, wobei die Zahl als Summe von Bitschaltern zu sehen ist. Ist Bit 1 in der Zahl enthalten, wird der Achsenstrich gezeichnet, ist Bit 2 gesetzt, kommen die Tics dazu. Das Bit 4 bewirkt das Beschriften.

In das vorbereitete Koordinatensystem soll jetzt die Kurve eingezeichnet werden. Der Befehl dazu lautet »LNdemo-

.dat,1,1;« (line). Der erste Parameter »demo.dat« ist die Datei, in der sich die Daten befinden, der zweite Parameter gibt die Spalte der Y-Werte an. Der dritte Parameter bestimmt über Bitschalter das Aussehen des Kurvenzuges. Ist das erste Bit gesetzt, wird die Verbindungslinie zwischen den Punkten gezeichnet. Ist Bit 2 enthalten, kommen die Punktmarkierungen dazu.
Mit dem ähnlich aufgebauten Befehl »STdemo.dat,1,1;« ermittelt der HPGL-Extender statistische Daten. Die zweite Zahl (0 oder 1) gibt an, ob die Ergebnisse ebenfalls mit vorbereiteten Bedingungen ausgegeben werden sollen.
Wie Sie Richtlinien einfügen, zeigt der Beispielbefehl »LM;«, womit der soeben berechnete Mittelwert als Linie über der gesamten X-Achse eingezeichnet wird.

Balkendiagramme lassen sich mit dem HPGL-Extender problemlos realisieren

In eine neue Koordinatenecke soll eine Häufigkeitsverteilung in Balkenform gezeichnet werden. Der entsprechende Befehl lautet »HIdemo.dat,1,10;«. Für das Histogramm aus den Werten der Y-Spalte 1 in der Datei »demo.dat« soll die X-Achse in zehn Klassen eingeteilt werden. Die zuständige Prozedur zählt die Häufigkeiten aus und zeichnet die Balken als Häufigkeitsprozente. Eine andere Form der Darstellung von Häufigkeitsverteilungen ist die Gaußsche Kurve (»GA;«).
Mit »SYo;« wird dafür gesorgt, daß, wenn eine Punktemarkierung erwünscht ist, das Symbol »o« an der Punktposition gezeichnet wird. In der vierten Y-Wertereihe überschreitet ein Wert den Maximalwert der Y-Achse. Durch Einsatz der HPGL-Funktion »InputWindow« wird dieser Wert abgeschnitten.
Das Zeichnen von Balkendiagrammen mit »PBdemo .dat,1,2;« (plot bar) ist dem Linienzeichnen sehr verwandt. Hier dient der Bitschalter jedoch dazu, zu bestimmen, ob die Balken nebeneinander oder aufeinander gestapelt gezeichnet werden sollen. Die Balkenbreite wird mit »BW.3;« in Einheiten eingestellt. Beachten Sie in diesem Zusammenhang den FT-Befehl, der als original HPGL-Befehl das Füllmuster bestimmt und ohne Beachtung weitergegeben wird.
Das Programm »hpglext.pas« (Listing) selbst arbeitet auf folgende Weise: Im »Startup« wird bestimmt, woher die eigentliche Zeichnungsanweisung kommen soll. In einer Datei »plo.plo« steht als Text »DEMO.PLO«, also der Name der Datei, die die Zeichenbefehle enthält. Diese Maßnahme hat bei Batch-Operationen den Sinn, daß der Extender ohne Dialogeingabe ablaufen kann. Die Hauptroutine »Work« setzt eine Befehlszeile zusammen, gibt sie zum Interpreter-Teil weiter und fragt nach Tastatureingaben. Nach dem Drücken einer beliebigen Taste außer [Esc] pausiert das Programm, wird [Esc] gedrückt, bricht das Programm ab.

Kleine Tippfehler in den Plotter-Anweisungen korrigiert der HPGL-Extender völlig selbständig

Der Interpreter-Teil bestimmt anhand der ersten beiden Zeichen in der Befehlszeile, was zu tun ist. Kommentare werden zurückgehalten, unbekannte Befehle ohne Beachtung weitergegeben. Kleinere »Unebenheiten« in der Eingabe werden nach dem folgenden Muster ausgebügelt: Kleinbuchstaben und Leerzeichen sind erlaubt, fehlende Trenn- und EOF(end of file)-Zeichen werden ergänzt.

MaskEdit PLUS 5.0

Jetzt auch für Turbo Basic und Quick Basic

Professioneller Maskengenerator

Nicht nur Profis sind begeistert

Auch viele Hobbyprogrammierer haben jetzt die Vorzüge eines professionellen Produktes entdeckt. Zudem ist durch die **kontextbezogene Hilfe** auf Tastendruck und die frei konfigurierbare Bedienung jeder in der Lage, sofort mit **MaskEdit Plus** konstruktiv zu arbeiten.

Zaubern auch Sie in Sekundenschnelle **komfortable Bildschirmmasken** für Menü- und Untermenüsysteme, Vorder- und Hintergrundmasken usw. auf den Monitor, und binden Sie danach den von **MaskEdit Plus** im gewählten Ausgabeformat erzeugten Quellcode einfach in Ihr Programm ein.

Geld sparen durch Modulsystem

Durch den modularen Aufbau von **MaskEdit Plus** können Sie gezielt und kostensparend einkaufen: Standard-Dateiausgabeformate wie ASCII, Binär, dBase und ANSI sind bereits im Grundpaket enthalten, durch **spezielle Maskentreiber** bedienen Sie Ihre bevorzugte Programmiersprache.

Professionelle Leistungen für jedermann

Neben acht Rahmentypen sind bis zu 16 Hintergrundfarben, Schattenwurf und Cursorkoordinaten darstellbar, und beim Rahmenzeichnen werden die **Kreuzungspunkte automatisch** angepaßt. Eine integrierte ASCII-Tabelle macht zusammen mit einem **universellen Maustreiber** gehörig Tempo.

Mit über 50 Funktionen können Sie auch **Teile einer Maske** kopieren, löschen, verschieben, zentrieren, färben oder füllen. Dabei wird von **MaskEdit Plus** der komplette IBM-Zeichensatz bis 255 Zeichen, die gleichzeitige Bearbeitung von zwei Masken und die Definition von Ein-/Ausgabefeldern unterstützt.

Programmier-Highlights der Maskentreiber in Kürze

Sämtliche Treiber werden als Sourcecode geliefert * Eigene Routinen können zur Kontrolle von Eingaben, Hilfetexten etc. problemlos in den Steuerungsablauf eingebunden werden * Integration von Datum und aktueller Uhrzeit und automatische Typüberprüfung bei der Eingabe, auch für den zusätzlichen Datentyp DATUM * Aufruf der generierten Masken durch einen einzigen Befehl und Steuerung der Ein-/Ausgabefelder durch den Maskentreiber * Wahlweise Ausschluß von Teilmengen des Zeichensatzes und automatische Eingabe in Großbuchstaben inklusive der Umlaute * Bei Änderungen von definierten Feldern kann die Kontrolle an das eigene Programm übergeben werden * Jedes Eingabefeld kann einzeln gefärbt werden, ist voll editierbar und läßt sowohl rechts- als auch linksbündige Ausgabe zu * Bewegung des Cursors innerhalb der Eingabemaske durch die Pfeil- und TAB-Tasten und Verlassen der Eingabemaske durch Funktionstasten-Betätigung.

Treiber:	MaskEdit Plus
Turbo Pascal 3/4/5	mit einem Treiber DM 99,–*
Microsoft C, Turbo C	Paketpreis,
JPI Topspeed Modula-2	mit zwei Treibern DM 149,–*
Turbo Basic	Paketpreis,
Quick Basic ab 4.0	mit drei Treibern DM 199,–*

Demonstrationsdiskette DM 5,–

* Unverbindliche Preisempfehlung. Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung.

– Bitte benutzen Sie die Bestellkarte –

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

Am Anfang des Programms finden Sie die Grundfunktionen. Besondere Beachtung verdient die Prozedur »Send«. Hier werden die Informationen an die zweite parallele Schnittstelle weitergeleitet. Ist der Plotter-Puffer voll, nimmt der Plotter keine Zeichen mehr an, und es kommt zu Wartezeiten. Sind diese Zeiten länger als zehn Sekunden, entsteht der Fehler Nummer 152. Dann geht die letzte Eingabe verloren und muß wiederholt werden. Damit dies nicht unkontrolliert mitten in einer Befehlszeile geschieht, müssen die Zeichen einzeln abgeschickt werden. Sollte Ihr Plotter an einer seriellen Schnittstelle angeschlossen sein, müssen Sie diesen Ausgabeteil anpassen und in »InitPlot« die Schnittstelle »lpt2« gegen »com1« austauschen.

Im Mittelteil des Programms befinden sich die komplexeren Funktionen. Hier ist der eigentliche Schwerpunkt für die Funktionen des HPGL-Extenders.

Abschließend noch ein Tip: Es ist einfach, Funktionen und Kurvenfelder anzufertigen, wenn Sie mit einem Hilfsprogramm die entsprechenden Datenfelder in eine Textdatei schreiben. Ist X kontinuierlich steigend, so können Sie nach einer Veränderung des Programms getrost auf die X-Spalte der Datei verzichten. Y-Punkte, die nicht gezeichnet werden sollen, könnten beispielweise mit »-1« markiert werden. Es ist aber auch denkbar, ohne Aufwand eine Funktion ähnlich der Gaußschen Kurve als Prozedur einzubauen. Der Extender ist ein wirkungsvolles Werkzeug für die Plotter-Steuerung und für viele Erweiterungen offen.

(Reinhold Wegert/bk/zi)

```
[AN2]    1: { HPGL Extender
[6U2]    2:   Reinhold Wegert
[YZØ]    3:   (c) 199Ø
[AF5]    4:   Redaktionsbüro Everts & Hagedorn
[U7Ø]    5: }
[VØ3]    6: program   HPGLExt;
[VN2]    7: uses      Crt, Dos;
[6XØ]    8: const
[YK4]    9:           Etx    = #3;        {end of text}
[A44]   1Ø:           Esc    = #27;            {escape}
[8T4]   11:           T      = ';';   {HPGL terminator}
[U64]   12:           C      = ',';         {separator}
[NØ5]   13:           MaxPen = 8;         {plotterpens}
[RNØ]   14: var
[6B1]   15:           WorkOn,
[SV1]   16:           ToPlotter,
[R13]   17:           ToMoni   : boolean;
[1Y4]   18:           Color    : array[1..8] of string;
[LG1]   19:           HelpFile,
[XH2]   2Ø:           HPGLFile : file;
[GG1]   21:           DataFile,
[ZF1]   22:           Plotter,
[WN2]   23:           OutFile  : text;
[AV3]   24:           TxtStr   : string[128];
[9V4]   25:           Txt      : array[1..128] of byte;
[8XØ]   26:           TxtP,
[ØC3]   27:           TxtStrP  : pointer;
[EGØ]   28:           Ch,
[GE2]   29:           SYmbol   : char;
[7M4]   3Ø:           HelpFileName, HPGLFileName,
[FD4]   31:           OutFileName, DataFileName,
[8U2]   32:           CmdLine, Command,
[592]   33:           Msg, DataLine,
[XU2]   34:           st1, st2 : string;
[G52]   35:           nB, B, R : word;
[KG2]   36:           XLabel, YLabel,
[ZU2]   37:           XLabelD, YLabelD,
[7E1]   38:           LineCon,
[CM2]   39:           LineConD : byte;
[A53]   4Ø:           PUnitsX, PUnitsY,
[UF3]   41:           PUnitsXD, PUnitsYD,
[UP2]   42:           DecAll, DecAfter,
[4S2]   43:           Column, ColumnD,
[9K1]   44:           Grid, Pen,
[6L2]   45:           Cn       : integer;
[NP2]   46:           PSizeX, PSizeY,
[D13]   47:           PSizeXD, PSizeYD,
[AL2]   48:           XCoord, YCoord,
[9G4]   49:           X, XBegin, XEnd, XStep, XTic,
[4K3]   5Ø:           XFactor, GX, BXBegin,
[KG4]   51:           Y, YBegin, YEnd, YStep, YTic,
[EK3]   52:           YFactor, GY, BYBegin,
[M24]   53:           LabelWidth, LabelHeight,
[CF3]   54:           LabelSlant, LabelDir,
[644]   55:           TicPos, TicNeg, BarWidth,
[PH2]   56:           YBarSum, Rl, K,
[S35]   57:           Sum, SumQ, NValues, Mini, Maxi,
[VN4]   58:           MeanValue, StdDev, StdDevPrz,
[7K3]   59:           x1, y1, x2, y2        : real;
[EØ6]   6Ø: {basic components ************************}
[585]   61: {-----------------------------------------}
[CN5]   62: procedure Change(var R1, R2: real);
[Ø52]   63: var RL: real;
[MUØ]   64: begin
[DX3]   65:   RL:= R1;  R1:= R2;  R2:= RL;
[ZNØ]   66: end;
[RG7]   67: {----change coordinate point into mm point}
[KB5]   68: function  mmX(var x: real): real;
[MUØ]   69: begin
[LB6]   7Ø:   mmX:= XCoord +(x*XFactor) -(GX*XFactor);
[ZNØ]   71: end;
[585]   72: {-----------------------------------------}
[SB5]   73: function  mmY(var y: real): real;
[MUØ]   74: begin
[3C6]   75:   mmY:= YCoord +(y*YFactor) -(GY*YFactor);
[ZNØ]   76: end;
[TB6]   77: {-------------------------extracts spaces}
[8Y5]   78: procedure NoSpace(var Str: string);
```

```
[8Y1]   79: var b: byte;
[MUØ]   8Ø: begin
[7D6]   81:   b:= pos(' ', Str);    {where is a space?}
[CY2]   82:   while b > Ø do begin
[ØL2]   83:     delete(Str, b, 1);
[H56]   84:     b:= pos(' ', Str);     {another space?}
[ZNØ]   85:   end;
[ZNØ]   86: end;
[CL6]   87: {----------------------cuts trailing zeros}
[YØ6]   88: procedure NoNulls(var Str: string);
[DA2]   89: var s: string;
[MUØ]   9Ø: begin
[P31]   91:   repeat
[AS4]   92:     s:= Copy(Str, length(Str), 1);
[H86]   93:       {last character of string possible Ø}
[6M5]   94:     if s = 'Ø' then              {shorten Str}
[A55]   95:       Str:= Copy(Str, 1, length(Str)-1);
[B92]   96:   until s <> 'Ø';
[3X5]   97:   if s = '.' then       {cut decimal point}
[A55]   98:     Str:= Copy(Str, 1, length(Str)-1);
[ZNØ]   99: end;
[2U6]  1ØØ: {-----------------change integer to string}
[UZ6]  1Ø1: function  IntStr(var I: integer): string;
[DA2]  1Ø2: var s: string;
[MUØ]  1Ø3: begin
[5N3]  1Ø4:   str(I, s);  IntStr:= s;
[ZNØ]  1Ø5: end;
[2M6]  1Ø6: {--------------------change real to string}
[RN6]  1Ø7: function  RealStr(var R: real): string;
[DA2]  1Ø8: var s: string;
[MUØ]  1Ø9: begin
[ZC4]  11Ø:   str(R:12:6, s);  RealStr:= s;
[ZNØ]  111: end;
[9W6]  112: {-------------for positioning in paper mms}
[GM6]  113: function  GridStr(var R: real): string;
[DA2]  114: var s: string;
[MUØ]  115: begin
[T43]  116:   str(R * Grid:12:6, s);
[Ø14]  117:   NoSpace(s);   NoNulls(s);
[H32]  118:   GridStr:= s;
[ZNØ]  119: end;
[2Ø7]  12Ø: {----------change real to formatted string}
[PN6]  121: function  FormStr(var R: real): string;
[DA2]  122: var s: string;
[MUØ]  123: begin
[FH6]  124:   str(R:DecAll:DecAfter, s);  FormStr:= s;
[ZNØ]  125: end;
[AY6]  126: {----------send message to plotter on lpt2}
[7R4]  127: procedure Send(Msg: string);
[A13]  128: var i, io: integer;
[MUØ]  129: begin
[KC3]  13Ø:   if Msg <> '' then begin
[5A2]  131:     if ToMoni then
[FC6]  132:       writeln(Msg);  {ctrl line to monitor}
[E63]  133:     write(OutFile, Msg);
[UY3]  134:     if ToPlotter then begin
[966]  135:     {$I-}              {enable error handling}
[2C5]  136:     for i:= 1 to length(Msg) do begin
[865]  137:                  {single character transmit}
[ZØ4]  138:       write(Plotter, Msg[i]);
[FA2]  139:       io:=IOResult;
[VH6]  14Ø:     {<> Ø if character not reached plotter}
[RL5]  141:       if io <> Ø then begin   {if IO error}
[273]  142:         if io = 16Ø then begin
[LN4]  143:           writeln('turn plotter on ');
[TY4]  144:           sound(1ØØØ); delay(1ØØ); nosound;
[RØ4]  145:           writeln('IOResult ', io,
[9R3]  146:                   ', continue with key');
[Y44]  147:           repeat until keypressed;
[U22]  148:           Ch:= ReadKey;
[VF4]  149:           if Ch = #Ø then  Ch := ReadKey;
[ZØ4]  15Ø:           write(Plotter, Msg[i]);
[ZNØ]  151:         end;
[BD3]  152:         while io = 152 do begin
[8A5]  153:                {on plotter response pause}
[ZØ4]  154:           write(Plotter, Msg[i]);
[PG3]  155:                    {try again and again}
[FA2]  156:           io:=IOResult;
```

```
[DV5]  157:          end;   {other errors not checked !}
[ZNØ]  158:        end;
[ZNØ]  159:      end;
[766]  16Ø:      {$I+}            {enable error handling}
[ZNØ]  161:      end;
[ZNØ]  162:    end;
[ZNØ]  163: end;
[HH6]  164: {--------------------pause or halt program?}
[T63]  165: procedure TestKbd;
[MUØ]  166: begin
[HZ3]  167:   if keypressed then begin
[U22]  168:     Ch:= ReadKey;
[D94]  169:     if Ch = #Ø then Ch := ReadKey;
[2P3]  17Ø:                          {trap function key}
[Y44]  171:     repeat until keypressed;
[H57]  172:   {pause, wait for another key to continue}
[U22]  173:     Ch:= ReadKey;
[D94]  174:     if Ch = #Ø then Ch := ReadKey;
[XW4]  175:     if Ch = #27 then  WorkOn:= false;
[1X2]  176:                                {exit on escape}
[ZNØ]  177:   end;
[ZNØ]  178: end;
[UL7]  179: {---next continous string from commandline}
[GF4]  18Ø: function  NextStr: string;
[VT2]  181: var N, NS: string;
[MUØ]  182: begin
[621]  183:   NS:= '';
[CG6]  184:   N:= CmdLine[Cn];           {next character}
[GR2]  185:   while (N <> ',') and
[VM1]  186:         (N <> ';') and
[ØM1]  187:         (N <> ' ') and
[MØ5]  188:         (Cn <= length(CmdLine))  do begin
[3N6]  189:     NS:= NS + N;      {complete next string}
[ZY5]  19Ø:     inc (Cn);              {character counter}
[GE6]  191:     N:= CmdLine[Cn];         {next character}
[ZNØ]  192:   end;
[DC6]  193:   NextStr:= NS;            {extracted string}
[SC1]  194:   inc (Cn);
[D95]  195:   while CmdLine[Cn] = ' ' do inc(Cn);
[MB5]  196:                 {skip spaces in DataFile line}
[ZNØ]  197: end;
[F47]  198: {--------get next integer from commandline}
[CP4]  199: function  NextInt: integer;
[UK5]  2ØØ: var NI, Er: integer; Str: string;
[MUØ]  2Ø1: begin
[WG2]  2Ø2:   Str:= NextStr;
[PU3]  2Ø3:   if Str = '' then Str:= '-1';
[N73]  2Ø4:                                {flag for default}
[DU2]  2Ø5:   val(Str, NI, Er);
[RE4]  2Ø6:   if Er <> Ø then begin  writeln
[Z66]  2Ø7:     ('translation error with:  ',CmdLine);
[DUØ]  2Ø8:     halt;
[ZNØ]  2Ø9:   end;
[P72]  21Ø:   NextInt:= NI;
[ZNØ]  211: end;
[X17]  212: {-----------get next real from commandline}
[G84]  213: function  NextReal: real;
[CF6]  214: var NR: real; Er: integer; Str: string;
[MUØ]  215: begin
[WG2]  216:   Str:= NextStr;
[PU3]  217:   if Str = '' then Str:= '-1';
[HV2]  218:   val(Str, NR, Er);
[RE4]  219:   if Er <> Ø then begin  writeln
[9X5]  22Ø:     ('translation error with ',CmdLine);
[DUØ]  221:     halt;
[ZNØ]  222:   end;
[G92]  223:   NextReal:= NR;
[ZNØ]  224: end;
[2H6]  225: {--------------------limits plotting area}
[DZ3]  226: procedure InputWindow;
[MUØ]  227: begin
[KR5]  228:   x1:= PUnitsX * mmX(XBegin) / PSizeX;
[XR5]  229:   y1:= PUnitsY * mmY(YBegin) / PSizeY;
[DB5]  23Ø:   x2:= PUnitsX * mmX(XEnd) / PSizeX;
[PB5]  231:   y2:= PUnitsY * mmY(YEnd) / PSizeY;
[US5]  232:   Send('IW' +RealStr(x1) +C  +RealStr(y1)
[SH5]  233:       +C +RealStr(x2) +C  +RealStr(y2) +T);
[ZNØ]  234: end;
[585]  235: {-----------------------------------------}
[R82]  236: procedure XF;
[MUØ]  237: begin
[6Y3]  238:   XFactor:= XTic / XStep;
[7Z3]  239:   if XBegin = Ø then  GX:= Ø;
[HY6]  24Ø:   {GX = helpvar if axis end left from zero}
[ØX4]  241:   if XBegin > Ø then  GX:= XBegin;
[K45]  242:   if (XBegin < Ø) and (XEnd >= Ø) then
[R31]  243:      GX:= Ø;
[X15]  244:   if (XBegin < Ø) and (XEnd < Ø) then
[YM1]  245:      GX:= XEnd;
[ZNØ]  246: end;
[V82]  247: procedure YF;
[MUØ]  248: begin
[FY3]  249:   YFactor:= YTic / YStep;
[BZ3]  25Ø:   if YBegin = Ø then  GY:= Ø;
[7X4]  251:   if YBegin > Ø then  GY:= YBegin;
[R45]  252:   if (YBegin < Ø) and (YEnd >= Ø) then
[U31]  253:      GY:= Ø;
[225]  254:   if (YBegin < Ø) and (YEnd < Ø) then
[4N1]  255:      GY:= YEnd;
```

```
[ZNØ]  256: end;
[7N6]  257: {--------------------plot with millimeters}
[PT4]  258: procedure PlotTo(x, y: real);
[MUØ]  259: begin
[VE4]  26Ø:   Send('PD;PA' + GridStr(x) + C +
[WW6]  261:     GridStr(y) + T); {change to grid units}
[ZNØ]  262: end;
[LM6]  263: {--------------------move with millimeters}
[ES4]  264: procedure MoveTo(x, y: real);
[MUØ]  265: begin
[CF4]  266:   Send('PU;PA' + GridStr(x) + C +
[HF2]  267:     GridStr(y) + T);
[ZNØ]  268: end;
[B16]  269: {group commands ************************}
[NV6]  27Ø: {=======================================}
[DG3]  271: procedure InitPlot;
[MUØ]  272: begin
[UY3]  273:   if ToPlotter then begin
[7U3]  274:     assign(Plotter, 'com1');
[U56]  275:        {plotter connected as lineprinter 2}
[X23]  276:     rewrite(Plotter);
[ZNØ]  277:   end;
[NB6]  278:   Send(Esc + '.(');  {plotter attention!}
[KR1]  279:   Send('IN;');
[YR5]  28Ø:          {1. HPGL command after Plotter on}
[ØT4]  281:   Send('SCØ,' + GridStr(PSizeX) +
[BA4]  282:         ',Ø,' + GridStr(PSizeY) + T);
[ØK7]  283:   PUnitsX:= round(PUnitsXD*PSizeX/PSizeXD);
[FK7]  284:   PUnitsY:= round(PUnitsYD*PSizeY/PSizeYD);
[NZ4]  285:   Send('IPØ,Ø,' + IntStr(PUnitsX) +
[5W3]  286:                C + IntStr(PUnitsY) + T);
[CØ6]  287:   Send('PAØ,Ø;');          {Plotter alive?}
[WD6]  288:   LabelWidth:= 1;    {character size in mm}
[6U6]  289:   LabelHeight:= 3;  {character width in mm}
[VZ5]  29Ø:   Send('SI.1,.3;');     {but send it in cm}
[DK2]  291:   LabelSlant:= 9Ø;
[VG6]  292:   Send('SLØ;');    {character slant in tan}
[LY1]  293:   LabelDir:= Ø;
[XV5]  294:   Send('DI1,Ø;');          {label direction}
[RU1]  295:   TicPos:= Ø;
[ØT1]  296:   TicNeg:= 1;
[KL5]  297:   Send('TLØ,1;');                {tic length}
[WY5]  298:   XLabel:= 7;        {bit: 1=line, 2=tics,}
[9K6]  299:   YLabel:= 7;    {3=label  on axis drawing}
[2V6]  3ØØ:   LineCon:= 1;  {1=line, 2=symbol on curve}
[LG6]  3Ø1:   DecAll:= 12;      {decimals bevore/after}
[NM6]  3Ø2:   DecAfter:= 6;   {decimalpoint in axlabel}
[ZNØ]  3Ø3: end;
[585]  3Ø4: {-----------------------------------------}
[Ø97]  3Ø5: {in same way: limits, specifications,  ...}
[GD3]  3Ø6: procedure MeanLine;
[MUØ]  3Ø7: begin
[YA2]  3Ø8:   Y:= MeanValue;
[SF4]  3Ø9:   MoveTo(mmX(XBegin), mmY(Y));
[G64]  31Ø:   PlotTo(mmX(XEnd), mmY(Y));
[ZNØ]  311: end;
[F26]  312: {-------------------------come in with mm}
[W46]  313: procedure BarAbs(x1, y1, x2, y2: real);
[MUØ]  314: begin
[T92]  315:   MoveTo(x1, y1);
[T74]  316:   Send('RA' + GridStr(x2) + C +
[FN2]  317:                GridStr(y2) + T);
[ZNØ]  318: end;
[SK6]  319: {-----------------come in with coordinates}
[M66]  32Ø: procedure BarRel(x1, y1, x2, y2: real);
[MUØ]  321: begin
[Z76]  322:   x1:= mmX(x1); y1:= mmY(y1);{change to mm}
[M14]  323:   x2:= mmX(x2); y2:= mmY(y2);
[LV5]  324:   BarAbs(x1, y1, x2, y2);  {make bar in mm}
[ZNØ]  325: end;
[F26]  326: {-------------------------come in with mm}
[AC6]  327: procedure RectAbs(x1, y1, x2, y2: real);
[MUØ]  328: begin
[T92]  329:   MoveTo(x1, y1);
[ZT4]  33Ø:   PlotTo(x2, y1);  PlotTo(x2, y2);
[TT4]  331:   PlotTo(x1, y2);  PlotTo(x1, y1);
[ZNØ]  332: end;
[SK6]  333: {-----------------come in with coordinates}
[3E6]  334: procedure RectRel(x1, y1, x2, y2: real);
[MUØ]  335: begin
[Z76]  336:   x1:= mmX(x1); y1:= mmY(y1);{change to mm}
[M14]  337:   x2:= mmX(x2); y2:= mmY(y2);
[TA6]  338:   RectAbs(x1, y1, x2, y2);{make rect in mm}
[ZNØ]  339: end;
[247]  34Ø: {------------move to right position bevore}
[584]  341: procedure LabelFromFile;
[MUØ]  342: begin
[Ø77]  343:   DataFileName:= NextStr;  {in CommandLine}
[865]  344:   assign(DataFile, DataFileName);
[7M2]  345:   reset(DataFile);
[HA2]  346:   Command:= 'LB';
[615]  347:   while not Eof(DataFile) do begin
[CE4]  348:     readln(DataFile, DataLine);
[LC5]  349:     Send('LB' +DataLine +#13 +#1Ø +Etx +T);
[ZNØ]  35Ø:   end;
[YK2]  351:   close(DataFile);
[ZNØ]  352: end;
```

Listing. Der HPGL-Extender »hpglext.pas« erweitert die bewährte Plotter-Sprache

```
[WS6]  353: {-------------------set new LabelDirection}
[NC4]  354: procedure LabelDirection;
[MUØ]  355: begin
[7B3]  356:   LabelDir:= NextReal;
[8Y1]  357:                                st1:='DI1,Ø;';
[XM5]  358:   if LabelDir >  22.5 then  st1:='DI1,1;';
[7N5]  359:   if LabelDir >  67.5 then  st1:='DIØ,1;';
[ZU5]  36Ø:   if LabelDir > 112.5 then  st1:='DI-1,1;';
[AV5]  361:   if LabelDir > 157.5 then  st1:='DI-1,Ø;';
[BY5]  362:  if LabelDir > 2Ø2.5 then  st1:='DI-1,-1;';
[TV5]  363:   if LabelDir > 247.5 then  st1:='DIØ,-1;';
[SV5]  364:   if LabelDir > 292.5 then  st1:='DI1,-1;';
[KR5]  365:   if LabelDir > 337.5 then  st1:='DI1,Ø;';
[YN1]  366:   Send(st1);
[ZNØ]  367: end;
[585]  368: {---------------------------------------}
[NH3]  369: procedure SelectPen;
[CC4]  37Ø: var S: string; Er: integer;
[MUØ]  371: begin
[ØH6]  372:   S:= NextStr;       {pen nr or color text}
[3R3]  373:   if (S = '') or (S = 'Ø') then
[T95]  374:     Send('SPØ;')                {reset pen}
[WP1]  375:   else begin
[GP6]  376:     val(S, Pen, Er);  {test for val or str}
[DØ2]  377:     if Er = Ø then
[HU5]  378:       Send(CmdLine)        {it was a number}
[PH6]  379:     else begin      {test S for color name}
[CU3]  38Ø:     for Pen:= 1 to MaxPen do
[ZL3]  381:       if Color[Pen] = S then
[Ø54]  382:         Send('SP'+ IntStr(Pen) + T);
[2W4]  383:     end;  {if color not found,  }
[1B5]  384:   end;    {no message, no change}
[ZNØ]  385: end;
[585]  386: {---------------------------------------}
[RM3]  387: procedure DrawXAxis;
[374]  388: var NR: real; Er: integer;
[MUØ]  389: begin
[ML6]  39Ø:   XLabel:= NextInt;  {line?, tics?, label?}
[BP5]  391:   if XLabel = -1 then XLabel:= XLabelD;
[VS1]  392:                                    {default}
[ØA3]  393:   if XBegin > XEnd then
[S63]  394:                     Change(XBegin, XEnd);
[UY5]  395:   if (XLabel and 1) = 1 then begin   {line}
[MH4]  396:     MoveTo(mmX(XBegin), YCoord);
[PA4]  397:     PlotTo(mmX(XEnd), YCoord);
[ZNØ]  398:   end;
[FY5]  399:   if (XLabel and 2) = 2 then begin   {tics}
[HW1]  4ØØ:     K:= XBegin;
[ZP3]  4Ø1:     while K <= XEnd do begin
[CZ4]  4Ø2:       MoveTo(mmX(K), YCoord + TicPos);
[AX4]  4Ø3:       PlotTo(mmX(K), YCoord - TicNeg);
[952]  4Ø4:       K:= K + XStep;
[ZNØ]  4Ø5:     end;
[ZNØ]  4Ø6:   end;
[S66]  4Ø7:   if (XLabel and 4) = 4 then begin  {label}
[HW1]  4Ø8:     K:= XBegin;
[ZP3]  4Ø9:     while K <= XEnd do begin
[LR2]  41Ø:       st1:= FormStr(K);
[DR2]  411:            val(st1, NR, Er);
[US5]  412:             {reject minus sign bevore zero}
[RE4]  413:            if Er <> Ø then begin  writeln
[Y85]  414:           ('translation error in X label');
[8K1]  415:             halt; end;
[UY4]  416:            if NR = Ø then st1:=FormStr(NR);
[H52]  417:       NoSpace(st1);
[MP3]  418:       MoveTo(mmX(K), (YCoord -
[4U2]  419:                       (LabelHeight * 2)));
[HT3]  42Ø:       Send('LB' + st1 + Etx + T);
[952]  421:       K:= K + XStep;
[ZNØ]  422:     end;
[ZNØ]  423:   end;
[ZNØ]  424: end;
[585]  425: {---------------------------------------}
[VM3]  426: procedure DrawYAxis;
[374]  427: var NR: real; Er: integer;
[MUØ]  428: begin
[PL6]  429:   YLabel:= NextInt;  {line?, tics?, label?}
[HP5]  43Ø:   if YLabel = -1 then YLabel:= YLabelD;
[VS1]  431:                                    {default}
[3A3]  432:   if YBegin > YEnd then
[U63]  433:                     Change(YBegin, YEnd);
[WY5]  434:   if (YLabel and 1) = 1 then begin   {line}
[RE4]  435:     MoveTo(XCoord, mmY(YBegin));
[964]  436:     PlotTo(XCoord, mmY(YEnd));
[ZNØ]  437:   end;
[HY5]  438:   if (YLabel and 2) = 2 then begin   {tics}
[LW1]  439:     K:= YBegin;
[ØR3]  44Ø:     while K <= YEnd do begin
[955]  441:       MoveTo(XCoord + TicPos, mmY(K));
[735]  442:       PlotTo(XCoord - TicNeg, mmY(K));
[B52]  443:       K:= K + YStep;
[ZNØ]  444:     end;
[ZNØ]  445:   end;
[U66]  446:   if (YLabel and 4) = 4 then begin  {label}
[LW1]  447:     K:= YBegin;
[ØR3]  448:     while K <= YEnd do begin
[LR2]  449:       st1:= FormStr(K);
[DR2]  45Ø:            val(st1, NR, Er);
[US5]  451:             {reject minus sign bevore zero}
[RE4]  452:            if Er <> Ø then begin  writeln
[Z85]  453:           ('translation error in Y label');
[8K1]  454:             halt; end;
[UY4]  455:            if NR = Ø then st1:=FormStr(NR);
[H52]  456:       NoSpace(st1);
[235]  457:       MoveTo((XCoord - (length(st1)+1)
[R24]  458:               * LabelWidth * 1.5), mmY(K));
[HT3]  459:       Send('LB' + st1 + Etx + T);
[B52]  46Ø:       K:= K + YStep;
[ZNØ]  461:     end;
[ZNØ]  462:   end;
[ZNØ]  463: end;
[Y87]  464: {--------get specified value from textline}
[UA4]  465: procedure DecodeDataLine;
[RK2]  466: var u: integer;
[MUØ]  467: begin
[TU5]  468:   Cn:= 1;                 {character counter}
[U77]  469:   CmdLine:= DataLine; {use in proc NextStr}
[166]  47Ø:   X:= NextReal;      {1. column is X value}
[216]  471:   YBarSum:= Ø;            {for stapeld bars}
[GH4]  472:   for u:= 1 to Column do begin
[E37]  473:     Y:= NextReal; {Y from specified column}
[3N3]  474:     YBarSum:= YBarSum + Y;
[HM5]  475:   end;                 {returns with X and Y}
[ZNØ]  476: end;
[585]  477: {---------------------------------------}
[8R3]  478: procedure Statistics;
[2R3]  479: var WriteSt: integer;
[MUØ]  48Ø: begin
[TD6]  481:   Sum:= Ø;          {init statistic values}
[VB1]  482:   SumQ:= Ø;
[2Y1]  483:   NValues:= Ø;
[DR1]  484:   Mini:= 1e2Ø;
[XV1]  485:   Maxi:= -1e2Ø;
[Ø77]  486:   DataFileName:= NextStr;  {in CommandLine}
[SK5]  487:   Column:= NextInt;              {2. param}
[7W5]  488:   if Column = -1 then Column:= ColumnD;
[NT5]  489:   WriteSt:= NextInt;             {3. param}
[T35]  49Ø:   if WriteSt = -1 then WriteSt:= Ø;
[865]  491:   assign(DataFile, DataFileName);
[7M2]  492:   reset(DataFile);
[615]  493:   while not Eof(DataFile) do begin
[CE4]  494:     readln(DataFile, DataLine);
[NM2]  495:     DecodeDataLine;
[R52]  496:     Sum:= Sum + Y;
[4Y2]  497:     SumQ:= SumQ + Y*Y;
[TH3]  498:     NValues:= NValues + 1;
[GZ3]  499:     if Y < Mini then Mini:= Y;
[EZ3]  5ØØ:     if Y > Maxi then Maxi:= Y;
[ZNØ]  5Ø1:   end;
[YK2]  5Ø2:   close(DataFile);
[PC4]  5Ø3:   MeanValue:= Sum / NValues;
[5D5]  5Ø4:   if NValues = 1 then StdDev:= Ø else
[ZY4]  5Ø5:     StdDev:= sqrt(abs((SumQ - Sum *
[254]  5Ø6:                MeanValue) / (Nvalues - 1)));
[SU5]  5Ø7:   StdDevPrz:= 1ØØ * StdDev / MeanValue;
[YU3]  5Ø8:   if WriteSt > Ø then begin
[MB3]  5Ø9:                          {WriteStatistics}
[WT5]  51Ø:              {move to right position bevore}
[9L5]  511:     Send('LBAnzahl Werte    :' + FormStr
[4A4]  512:           (NValues) + #1Ø + #13 + Etx +T);
[WK5]  513:     Send('LBMittelwert      :' + FormStr
[SF4]  514:           (MeanValue) + #1Ø + #13 +Etx +T);
[A65]  515:     Send('LBStd Abw         :' + FormStr
[K24]  516:            (StdDev) + #1Ø + #13 + Etx + T);
[M65]  517:     Send('LBStd Abw %       :' + FormStr
[LM4]  518:           (StdDevPrz) + #1Ø + #13 +Etx +T);
[1U4]  519:     Send('LBmin             :' + FormStr
[RK3]  52Ø:           (Mini) + #1Ø + #13 + Etx + T);
[HU4]  521:     Send('LBmax             :' + FormStr
[1K3]  522:           (Maxi) + #1Ø + #13 + Etx + T);
[ZNØ]  523:   end;
[ZNØ]  524: end;
[585]  525: {---------------------------------------}
[HF3]  526: procedure PlotLine;
[MUØ]  527: begin
[Ø77]  528:   DataFileName:= NextStr;  {in CommandLine}
[WX2]  529:   Column:= NextInt;
[7W5]  53Ø:   if Column = -1 then Column:= ColumnD;
[353]  531:   LineCon:= NextInt;
[576]  532:   if LineCon = -1 then LineCon:= LineConD;
[586]  533:   InputWindow;                 {set plot range}
[Ø24]  534:   if (LineCon and 2) = 2 then
[NA6]  535:     Send('SM' + Symbol + T);    {symbol on}
[865]  536:   assign(DataFile, DataFileName);
[7M2]  537:   reset(DataFile);
[RB6]  538:   readln(DataFile, DataLine);     {1. line}
[NM2]  539:   DecodeDataLine;
[YL6]  54Ø:   MoveTo(mmX(X), mmY(Y));  {to 1. position}
[615]  541:   while not Eof(DataFile) do begin
[CE4]  542:     readln(DataFile, DataLine);
[NM2]  543:     DecodeDataLine;
[SR3]  544:     if LineCon > Ø then begin
[V14]  545:       if (LineCon and 1) = 1 then
[AH3]  546:                     PlotTo(mmX(X), mmY(Y));
[S14]  547:       if (LineCon and 1) = Ø then
[UH3]  548:                     MoveTo(mmX(X), mmY(Y));
[ZNØ]  549:     end;
[ZNØ]  55Ø:   end;
[4Y2]  551:   Send('IW;SM;PU;');
[E66]  552:     {input window off, symbol off, pen up}
```

```
[YK2]  553:   close(DataFile);
[ZNØ]  554: end;
[585]  555: {-----------------------------------------}
[LF3]  556: procedure PlotBars;
[MUØ]  557: begin
[Ø77]  558:   DataFileName:= NextStr;  {in CommandLine}
[WX2]  559:   Column:= NextInt;
[7W5]  56Ø:   if Column = -1 then Column:= ColumnD;
[353]  561:   LineCon:= NextInt;
[576]  562:   if LineCon = -1 then LineCon:= LineConD;
[586]  563:   InputWindow;              {set plot range}
[865]  564:   assign(DataFile, DataFileName);
[7M2]  565:   reset(DataFile);
[615]  566:   while not Eof(DataFile) do begin
[CE4]  567:     readln(DataFile, DataLine);
[NM2]  568:     DecodeDataLine;
[FX4]  569:     if (LineCon and 1) = 1 then begin
[WP2]  57Ø:                               {stapeld bars}
[W41]  571:       x1:= X;
[KK2]  572:       y1:= YBarSum - Y;
[LØ3]  573:       x2:= x1 + BarWidth;
[N52]  574:       y2:= YBarSum;
[4MØ]  575:     end
[WP1]  576:     else begin
[XV3]  577:                          {bars stand on x axis}
[KN4]  578:       x1:= X + (Column-1) * BarWidth;
[611]  579:       y1:= Ø;
[LØ3]  58Ø:       x2:= x1 + BarWidth;
[451]  581:       y2:= Y;
[ZNØ]  582:     end;
[G23]  583:     BarRel(x1, y1, x2, y2);
[3G3]  584:     RectRel(x1, y1, x2, y2);
[ZNØ]  585:   end;
[HV6]  586:   Send('IW;PU;');{input window off, pen up}
[YK2]  587:   close(DataFile);
[ZNØ]  588: end;
[SY7]  589: {----run Statistics first without printing}
[7R3]  59Ø: procedure Histogramm;
[KR5]  591:           {be shure to set new axis bevore}
[R33]  592:                          {change x with y !!}
[1P3]  593: const  MaxClasses = 5Ø;
[P95]  594: var    i, Class, Classes: integer;
[FD4]  595:        RL, XClass, Prozent: real;
[VN5]  596:        Counter: array[Ø..MaxClasses] of
[HF1]  597:                 integer;
[MUØ]  598: begin
[Ø77]  599:   DataFileName:= NextStr;  {in CommandLine}
[WX2]  6ØØ:   Column:= NextInt;
[7W5]  6Ø1:   if Column = -1 then Column:= ColumnD;
[A43]  6Ø2:   Classes:= NextInt;
[V25]  6Ø3:   if Classes = -1 then Classes:= 1Ø;
[8H6]  6Ø4:   for i:= Ø to Classes do  Counter[i]:= Ø;
[EØ3]  6Ø5:                            {reset counters}
[SL5]  6Ø6:   XClass:= (XEnd - XBegin) / Classes;
[RZ2]  6Ø7:   BarWidth:= XClass;
[865]  6Ø8:   assign(DataFile, DataFileName);
[7M2]  6Ø9:   reset(DataFile);
[847]  61Ø:   {second reading pass, count y in classes}
[615]  611:   while not Eof(DataFile) do begin
[CE4]  612:     readln(DataFile, DataLine);
[NM2]  613:     DecodeDataLine;
[3V4]  614:     RL:= ((Y - XBegin) / XClass) + Ø.5;
[9W2]  615:     Class:=round(RL);
[2A6]  616:     Counter[Class]:= Counter[Class] + 1;
[KR4]  617:     Counter[Ø]:= Counter[Ø] + 1;
[ZNØ]  618:   end;
[KP4]  619:   for i:= 1 to Classes do begin
[4E4]  62Ø:     x1:= XBegin + (XClass * (i-1));
[611]  621:     y1:= Ø;
[LØ3]  622:     x2:= x1 + BarWidth;
[RM5]  623:     y2:= 1ØØ * Counter[i] / Counter[Ø];
[Z83]  624:     if y1 <> y2 then begin
[G23]  625:       BarRel(x1, y1, x2, y2);
[3G3]  626:       RectRel(x1, y1, x2, y2);
[ZNØ]  627:     end;
[ZNØ]  628:   end;
[ZNØ]  629: end;
[NL6]  63Ø: {---------------------run statistics first}
[8N3]  631: procedure PlotGauss;
[T32]  632: var FG: real;
[1L3]  633: function Gauss: real;
[MUØ]  634: begin
[685]  635:   Y:= ( -1 *((((X -MeanValue) / StdDev)
[MZ4]  636:       * ((X - MeanValue) / StdDev )) / 2));
[7W1]  637:   Y:= exp(Y);
[HL5]  638:   Y:= (Ø.39894 / (StdDev * XFactor)) * Y;
[KG1]  639:   Gauss:= Y
[ZNØ]  64Ø: end;
[MUØ]  641: begin
[WA2]  642:   X:= MeanValue;
[6U2]  643:   FG:= YEnd / Gauss;
[FX3]  644:   X:= XBegin;  Y:= YBegin;
[YL6]  645:   MoveTo(mmX(X), mmY(Y));  {to 1. position}
[P31]  646:   repeat
[3Ø6]  647:     X:= X + XStep / 5Ø;        {resolution}
[7C2]  648:     Y:= Gauss * FG;
[AH3]  649:     PlotTo(mmX(X), mmY(Y));
[FL2]  65Ø:   until X >= XEnd;
[ZNØ]  651: end;
[D47]  652: {-----insert here your own procedures-----}
[YF7]  653: {============DECODE EXTENDED HPGL COMMANDS}
[MN5]  654: procedure InterpretCommandLine;
[MUØ]  655: begin
[2H4]  656:   Command:= Copy(CmdLine, 1, 2);
[5G5]  657:   Command[1]:= upcase(Command[1]);
[9G5]  658:   Command[2]:= upcase(Command[2]);
[U21]  659:   Cn:= 3;
[VT6]  66Ø:     {character counter for begin next item}
[H96]  661:   if Command <> 'LB' then NoSpace(CmdLine);
[G86]  662:   if Command = 'NP' then ToPlotter:= false
[3N6]  663:   else if Command ='NM' then ToMoni:= false
[5U5]  664:   else if Command = 'IN' then InitPlot
[G94]  665:   else if Command = 'XP' then
[TX6]  666:     PSizeX:= NextReal     {papersize in mm}
[H94]  667:   else if Command = 'YP' then
[KZ2]  668:     PSizeY:= NextReal
[M84]  669:   else if Command = 'XB' then
[PE6]  67Ø:     XBegin:= NextReal         {x axis begin}
[N84]  671:   else if Command = 'YB' then
[WE6]  672:     YBegin:= NextReal         {y axis begin}
[U84]  673:   else if Command = 'XE' then
[SR5]  674:     XEnd:= NextReal             {x axis end}
[V84]  675:   else if Command = 'YE' then
[WR5]  676:     YEnd:= NextReal             {y axis end}
[A55]  677:   else if Command = 'XS' then begin
[LT6]  678:     XStep:= NextReal; XF;  {step in x axis}
[4MØ]  679:   end
[B55]  68Ø:   else if Command = 'YS' then begin
[VT6]  681:     YStep:= NextReal; YF;  {step in y axis}
[4MØ]  682:   end
[C55]  683:   else if Command = 'XT' then begin
[HG6]  684:     XTic:=  NextReal;  XF;   {tic distance}
[4MØ]  685:   end
[D55]  686:   else if Command = 'YT' then begin
[MG6]  687:     YTic:=  NextReal;  YF;   {tic distance}
[4MØ]  688:   end
[P84]  689:   else if Command = 'XC' then
[GW6]  69Ø:     XCoord:= NextReal     {coordinate zero}
[R84]  691:   else if Command = 'YC' then
[WX2]  692:     YCoord:= NextReal
[DY5]  693:   else if Command = 'XA' then DrawXAxis
[GY5]  694:   else if Command = 'YA' then DrawYAxis
[R45]  695:   else if Command = 'MO' then begin
[8F4]  696:     X:= NextReal;  Y:= NextReal;
[C32]  697:     MoveTo(X, Y);
[4MØ]  698:   end
[M45]  699:   else if Command = 'PL' then begin
[8F4]  7ØØ:     X:= NextReal;  Y:= NextReal;
[W22]  7Ø1:     PlotTo(X, Y);
[4MØ]  7Ø2:   end
[655]  7Ø3:   else if Command = 'LW' then begin
[AM3]  7Ø4:                          {set new LabelWidth}
[4T3]  7Ø5:     LabelWidth:= NextReal;
[YA3]  7Ø6:     x1:= LabelWidth * Ø.1;
[CE3]  7Ø7:     x2:= LabelHeight * Ø.1;
[T64]  7Ø8:     Send('SI' + RealStr(x1) + C +
[4M2]  7Ø9:                  RealStr(x2) + T);
[4MØ]  71Ø:   end
[845]  711:   else if Command = 'LH' then begin
[YR3]  712:                          {set new LabelHeight}
[724]  713:     LabelHeight:= NextReal;
[YA3]  714:     x1:= LabelWidth * Ø.1;
[CE3]  715:     x2:= LabelHeight * Ø.1;
[T64]  716:     Send('SI' + RealStr(x1) + C +
[4M2]  717:                  RealStr(x2) + T);
[4MØ]  718:   end
[Y45]  719:   else if Command = 'LS' then begin
[NM3]  72Ø:                          {set new LabelSlant}
[TT3]  721:     LabelSlant:= NextReal;
[RU4]  722:     x1:= ArcTan(((9Ø - LabelSlant) * 2
[WF1]  723:                                * pi / 36Ø));
[9A4]  724:     Send('SL' + RealStr(x1) + T);
[4MØ]  725:   end
[Ø45]  726:   else if Command = 'LD' then begin
[L87]  727:     LabelDirection;{set new LabelDirection}
[4MØ]  728:   end
[U35]  729:   else if Command = 'DE' then begin
[KN6]  73Ø:     DecAll:= NextInt;  {format of decimals}
[U93]  731:     DecAfter:= NextInt;
[4MØ]  732:   end
[W35]  733:   else if Command = 'LB' then begin
[NØ5]  734:                  {with LaBel add EndOfText?}
[956]  735:     st1:= Copy(CmdLine, length(CmdLine),1);
[GY6]  736:     {last character of string possible etx}
[N54]  737:     if st1 = Etx then  CmdLine:=
[VP5]  738:        Copy(CmdLine, 1, length(CmdLine)-1);
[9R2]  739:                                {excludes etx}
[NS4]  74Ø:     Send(CmdLine + #13 + #1Ø + Etx + T);
[4MØ]  741:   end
[445]  742:   else if Command = 'LF' then begin
[VE6]  743:     LabelFromFile;       {text from a file}
[4MØ]  744:   end
[S45]  745:   else if Command = 'TL' then begin
```

Listing. Der HPGL-Extender »hpglext.pas« erweitert die bewährte Plotter-Sprache

```
[H76]  746:     TicPos:= NextReal;          {tic length}
[L13]  747:     TicNeg:= NextReal;
[NA6]  748:      {don't pass through, don't use XT, YT}
[4MØ]  749:   end
[X94]  75Ø:   else if Command = 'SY' then
[ZL6]  751:     Symbol:= CmdLine[3]         {set symbol}
[M45]  752:   else if Command = 'LN' then begin
[6C6]  753:     PlotLine;        {draw a line from file}
[4MØ]  754:   end
[Ø45]  755:   else if Command = 'PB' then begin
[ØD6]  756:     PlotBars;          {draw bars from file}
[4MØ]  757:   end
[K45]  758:   else if Command = 'LM' then begin
[Ø66]  759:     MeanLine;       {draw a mean value line}
[4MØ]  76Ø:   end
[D45]  761:   else if Command = 'CO' then begin
[8R3]  762:                        {new color name list}
[CU3]  763:     for Pen:= 1 to MaxPen do
[LU3]  764:                       Color[Pen]:= NextStr;
[4MØ]  765:   end
[Z45]  766:   else if Command = 'SP' then begin
[YT1]  767:     SelectPen;
[4MØ]  768:   end
[845]  769:   else if Command = 'RE' then begin
[8C2]  77Ø:                                 {rectangle}
[PS4]  771:     x1:= NextReal;  y1:= NextReal;
[WS4]  772:     x2:= NextReal;  y2:= NextReal;
[MF3]  773:     RectAbs(x1, y1, x2, y2);
[4MØ]  774:   end
[G35]  775:   else if Command = 'BA' then begin
[VM2]  776:                              {bar absolute}
[PS4]  777:     x1:= NextReal;  y1:= NextReal;
[WS4]  778:     x2:= NextReal;  y2:= NextReal;
[E23]  779:     BarAbs(x1, y1, x2, y2);
[4MØ]  78Ø:   end
[K45]  781:   else if Command = 'BR' then begin
[1N2]  782:                              {bar relative}
[PS4]  783:     x1:= NextReal;  y1:= NextReal;
[WS4]  784:     x2:= NextReal;  y2:= NextReal;
[G23]  785:     BarRel(x1, y1, x2, y2);
[4MØ]  786:   end
[W45]  787:   else if Command = 'BW' then begin
[1E3]  788:     BarWidth:= NextReal;
[4MØ]  789:   end
[G76]  79Ø:   else if Command = 'ST' then Statistics
[U56]  791:   else if Command = 'HI' then Histogramm
[LY5]  792:   else if Command = 'GA' then PlotGauss
[NH7]  793: {-insert here your own commands and calls-}
[D94]  794:   else if Command = 'EX' then
[AX6]  795:     WorkOn:= false  {EXit = use for break?}
[263]  796:   else Send(CmdLine);
[CL6]  797:      {original HPGL commands pass through}
[ZNØ]  798: end;
[KK9]  799: {xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}
[983]  8ØØ: procedure Startup;
[ZH2]  8Ø1: var i: integer;
[MUØ]  8Ø2: begin
[DB1]  8Ø3:   clrscr;
[4P6]  8Ø4:   TxtP:= @Txt;      {pointer for blockread}
[EF7]  8Ø5:   TxtStrP:= @TxtStr[1]; {pntr to stringbeg}
[2G6]  8Ø6:   TxtStr[Ø]:= #128;  {string has 128 chars}
[6U5]  8Ø7:   B:= 1;             {read one block at time}
[Ø27]  8Ø8:   OutFileName:= 'HPGL.PLO'; {out hpgl code}
[ØØ7]  8Ø9:   assign(OutFile, OutFileName);   {to file}
[723]  81Ø:   rewrite(OutFile);
[G54]  811:   HelpFileName:= 'plo.plo';
[C64]  812:                      {file with HPGLFileName}
[A95]  813:   assign(HelpFile, HelpFileName);
[SN2]  814:   reset(HelpFile);
[TC5]  815:   BlockRead (HelpFile, TxtPˆ, B, R);
[7VØ]  816:   i:= 1;
[DV2]  817:   HPGLFileName:= '';
[U94]  818:   while (Txt[i] > 45) do begin
[YH3]  819:                            {hope filename is ok}
[7E2]  82Ø:     HPGLFileName:=
[WG4]  821:                  HPGLFileName + chr(Txt[i]);
[EY4]  822:                {file with HPGL information}
[951]  823:     inc(i);
[ZNØ]  824:   end;
[FM2]  825:   close(HelpFile);
[9W3]  826:   writeln(HPGLFileName);
[A57]  827:   {writeln('press key when plotter ready');
[Y44]  828:   repeat until keypressed;
[U22]  829:   Ch:= ReadKey;
[1T4]  83Ø:   if Ch = #Ø then Ch := ReadKey;}
[ØA5]  831:   assign(HPGLFile, HPGLFileName);
[DN2]  832:   reset(HPGLFile);
[F94]  833:   nB:= FileSize(HPGLFile) + 1;
[RY2]  834:                             {count of blocks}
[PB6]  835:   ToPlotter:= true;           {plot direct}
[DN6]  836:   ToMoni:= true;     {echo HPGL on monitor}
[672]  837:   PSizeXD:= 4Ø4;
[4L6]  838:      {default, fits to plotter units in IP}
[KE2]  839:   PSizeYD:= 275.5;
[MØ2]  84Ø:   PSizeX:= 4Ø4;
[ZC2]  841:   PSizeY:= 275.5;
[2P2]  842:   PUnitsXD:= 16158;
[7G6]  843:      {plotter units for default papersize}
```

```
[DN2]  844:   PUnitsYD:= 11Ø4Ø;
[TE1]  845:   Grid:= 1Ø;
[K36]  846:      {Grid Size (reachable points) 1/1Ø mm}
[ZW1]  847:   ColumnD:= 1;
[HM4]  848:                 {plot 1. column with LINE}
[ØV1]  849:   Column:= 1;
[H12]  85Ø:   LineConD:= 1;
[285]  851:           {bit: 1=line, 2=symbol on LINE}
[EX1]  852:   LineCon:= 1;
[CY1]  853:   Symbol:= '+';
[YY3]  854:                      {point marker on LINE}
[E12]  855:   BarWidth:= 1;
[NU1]  856:   XCoord:= Ø;
[ET1]  857:   XBegin:= Ø;
[CA1]  858:   XEnd:= Ø;
[LN1]  859:   XStep:= 1;
[5B1]  86Ø:   XTic:= 1;
[BV1]  861:   XLabelD:= 1;
[RU1]  862:   YCoord:= Ø;
[GT1]  863:   YBegin:= Ø;
[EA1]  864:   YEnd:= Ø;
[NN1]  865:   YStep:= 1;
[7B1]  866:   YTic:= 1;
[DV1]  867:   YLabelD:= 1;
[R97]  868:   {write here your default color positions}
[WY6]  869:   {use same syntax later in extHPGL cmdfil}
[Ø23]  87Ø:   Color[1]:= 'black';
[YN2]  871:   Color[2]:= 'red';
[W33]  872:   Color[3]:= 'green';
[6X2]  873:   Color[4]:= 'blue';
[193]  874:   Color[5]:= 'yellow';
[BX2]  875:   Color[6]:= 'pink';
[W53]  876:   Color[7]:= 'brown';
[G63]  877:   Color[8]:= 'orange';
[ZNØ]  878: end;
[585]  879: {----------------------------------------}
[XS7]  88Ø: {read unknown structured HPGL file blkwise}
[RS2]  881: procedure Work;
[AT2]  882: var i, j: integer;
[MUØ]  883: begin
[6X1]  884:   CmdLine:= '';
[3A5]  885:   j:= Ø;                      {block counter}
[L26]  886:   while (j <= nB) and (WorkOn = true)  do
[MUØ]  887:   begin
[4Ø6]  888:     BlockRead (HPGLFile, TxtStrPˆ, B, R);
[1G2]  889:                                 {read 1 block}
[196]  89Ø:     i:= 1;          {reset character counter}
[PX5]  891:     while (i <= 128) and (WorkOn = true) do
[V35]  892:     begin                     {go on decoding}
[TS3]  893:       if (TxtStr[i] = ';') or
[B43]  894:          (TxtStr[i] = #13) or
[TC4]  895:          (TxtStr[i] = #1Ø) then begin
[LW2]  896:                               {end of Command}
[RP4]  897:         if CmdLine[1] > #64 then begin
[V46]  898:          {reject line begin with unliteral}
[BV3]  899:           InterpretCommandLine;
[ZNØ]  9ØØ:         end;
[LT4]  9Ø1:         CmdLine:= '';                {reset}
[4M5]  9Ø2:         TestKbd;         {for halt or pause}
[4MØ]  9Ø3:       end
[WP1]  9Ø4:       else begin
[UX4]  9Ø5:         CmdLine:= CmdLine + TxtStr[i]
[N54]  9Ø6:                    {completes command line}
[ZNØ]  9Ø7:       end;
[951]  9Ø8:       inc(i);
[PF5]  9Ø9:     end;                {while in one block}
[B51]  91Ø:     inc(j);
[A85]  911:   end;                        {no more blocks}
[YM2]  912:   WorkOn:= false;
[ZNØ]  913: end;
[585]  914: {----------------------------------------}
[F23]  915: procedure Shutup;
[MUØ]  916: begin
[4M2]  917:   Close(HPGLFile);
[DL5]  918:   Send('SPØ;');                  {put pen away}
[SM6]  919:   MoveTo(PSizeX, PSizeY);         {free paper}
[DM5]  92Ø:   Send(Esc + '.)');              {Plotter Off}
[HM5]  921:   if ToPlotter then  Close(Plotter);
[7L2]  922:   Close(OutFile);
[FG4]  923:   if ToMoni = true then begin
[F44]  924:     write('press a key to end');
[KB4]  925:                    {hold last monitor infos}
[Y44]  926:     repeat until keypressed;
[ZNØ]  927:   end;
[DB1]  928:   ClrScr;
[ZNØ]  929: end;
[NV6]  93Ø: {=======================================}
[MUØ]  931: begin
[UG1]  932:   Startup;
[FF2]  933:   WorkOn:= true;
[P31]  934:   repeat
[4XØ]  935:     Work;
[4Z2]  936:   until not WorkOn;
[CD1]  937:   Shutup;
[KNØ]  938: end.
```

Listing. Der HPGL-Extender »hpglext.pas« erweitert die bewährte Plotter-Sprache

Wie wär's mal mit Komfort?

SAA-Blockbefehle für Turbo Pascal und Co.

Bei allen Editoren, die an das SAA-Konzept von IBM angelehnt sind, markieren, kopieren und löschen Sie auf ganz einfache Weise Blöcke mit den Cursor-Tasten. Mit Wordstar-kompatiblen Editoren allerdings ist der Aufwand erheblich größer. Damit Sie die Vorzüge des SAA-Konzeptes auch bei Wordstar-kompatiblen Editoren nutzen können, gibt es nun das Utillity »Saaws«.

Wordstar, der »Oldie« unter den Textverarbeitungsprogrammen, hat mit seinen Control-Sequenzen einen Standard gesetzt, dem sich viele Software-Häuser angeschlossen haben und ihm bis heute treu geblieben sind. Auch die bekannten Programmiersprachen aus dem Hause Borland zählen dazu.

So markieren Sie zum Beispiel in Turbo Pascal Anfang und Ende eines Blocks mit [Ctrl-KB] und [Ctrl-KK]. Die Konkurrenzprodukte von Microsoft dagegen bauen auf dem SAA-Konzept auf und lassen sich um vieles einfacher bedienen. Hierbei werden Blöcke mit den Cursor-Tasten markiert, wobei Sie zusätzlich die Shift-Taste drücken müssen. Dies hat große Ähnlichkeit mit dem Markieren mittels einer Maus und wirkt deshalb auf den Anwender auch viel logischer und plausibler.

Das Programm »saaws.asm« (Listing 1) bringt die SAA-Blockbefehle auch Wordstar-kompatiblen Editoren bei. Die Grundidee ist, daß ein Hintergrundprogramm den Tastatur-Interrupt auf sich umleitet und bei jeder Anfrage nachsieht, ob eine Cursor-Taste in Verbindung mit [Shift] betätigt wurde. Falls ja, dann gibt das Hintergrundprogramm nicht den Code der Taste zurück, sondern eine vordefinierte Control-Sequenz. Wenn Sie zum Beispiel [Shift-End] betätigen, dann bekommt das Vordergrundprogramm (der Editor) die Tastenfolge [Ctrl-KB] [End] [Ctrl-KK] geliefert. Als Folge wird daher der Bereich von der aktuellen Cursor-Position bis zum Ende der Zeile markiert, wobei der Cursor an das Zeilenende springt.

Leider können nicht alle SAA-Befehle eins zu eins übernommen werden. Dadurch, daß das Blockende immer hinter dem Blockanfang liegen muß, ist es nicht möglich, einen Block »vom Ende her« zu definieren. Ein weiterer Unterschied tritt bei den Verschiebe- und Kopierbefehlen auf. SAA-Editoren arbeiten mit einem Puffer, den es bei Wordstar-kompatiblen Editoren nicht gibt. Mit dem Schalter FILE können Sie daher die Puffer-Emulation einstellen, die das Löschen einer einzelnen Zeile mit [Ctrl-Y] jedoch nicht unterstützt. Steht der Schalter auf 0, dann wird der Befehl [Shift-Del] zum Löschen in den Puffer erst ausgeführt, wenn Sie den Pufferinhalt mit [Shift-Ins] an anderer Stelle in den Quelltext einfügen. Dies kann zur Verwirrung beitragen, weil Sie auf dem Bildschirm noch Bereiche sehen können, die eigentlich schon gelöscht sind. Außerdem geht der Inhalt des Puffers verloren, wenn Sie einen neuen Block definieren. Steht der Schalter dagegen auf 1, dann wird der Puffer als Block im aktuellen Verzeichnis unter dem Namen »$« gespeichert. Diese Einstellung ist vor allem für Festplattenbesitzer zu empfehlen. Noch schneller geht es allerdings, wenn Sie eine RAM-Disk verwenden. Hierzu müssen Sie die Konstante »FNAME« entsprechend ändern. Wenn Ihre RAM-Disk also zum Beispiel als Laufwerk G angesprochen wird, dann muß die Definition so lauten:

```
FNAME Equ <"g",":","$",".">
```

Dieses etwas verwirrende Aussehen der Zeichenkette ist dadurch zu erklären, daß sie gleichzeitig als Byte- und als Wort-Folge benötigt wird.

Die dritte Konstante am Anfang des Quelltextes stellt die Taste dar, die das Überschreiben eines Blocks bestätigt. Normalerweise wird dies »Y« sein, es wäre aber auch ein »J« für deutsche Programmversionen denkbar. Wenn Ihr Editor keine Rückfrage beim Überschreiben mit [Ctrl-KW] stellt, dann setzen Sie »OVERWR« einfach auf 0. Außerdem können Sie dann auf den Programmteil zum Anlegen eines Dummy-Blocks vor dem Programmstart verzichten.

Eine weitere Einschränkung betrifft den Befehl [Shift-Ctrl-Right] zum Markieren eines Wortes. Während bei SAA-Editoren der Cursor mit [Ctrl-Right] und [Shift-Ctrl-Right] auf jeweils verschiedene Positionen springt, kann der Emulator nur auf den normalen Wort-Sprungbefehl des Editors zurückgreifen. Deshalb gibt es den Schalter »MARK«. Steht dieser auf 1, dann übergibt Saaws nicht den Code für [Ctrl-Right], sondern sieht direkt im Bildschirmspeicher nach, wo das Wort zu Ende ist. Für jedes gefundene Zeichen, das Bestandteil des Wortes ist, wird der Code für [Right] in den Tastaturpuffer geschrieben, so daß die Markierung zum Schluß am Ende und nicht am Anfang des nächsten Wortes steht. Dieses Verfahren hat allerdings zwei Grenzen. Zum einen kann das Programm bei überlangen Zeilen natürlich nicht über den rechten Bildschirmrand »hinaussehen«. Deshalb greift es auf den normalen Sprungbefehl [Ctrl-Right] zurück, wenn das Wort bis an den Bildschirmrand reicht. Dieses Verfahren funktioniert jedoch nicht mit den Editoren von Borland, weil diese die achtzigste Spalte immer frei lassen. Vermutlich sollte hier einmal ein Statuszeichen für »überlange Zeile« erscheinen, was die Programmierer dann aber vergessen zu haben scheinen. Deshalb bewirkt die Kombination [Shift-Ctrl-Right] in diesem Fall dasselbe wie [Shift-Right]. Ähnliche Schwierigkeiten gibt es am Ende einer Zeile. Auch hier wird wieder auf das normale [Ctrl-Right] zurückgegriffen, so daß sich die Emulation vom SAA-Editor unterscheidet.

Das zweite Problem taucht beim Aufnehmen eines Makros auf. Da ein Makro nicht rekursiv definiert werden kann, zählt der Emulator die Zeichen des aktuellen Wortes und schreibt eine entsprechende Anzahl von Cursor-Right-Anschlägen in den Puffer. Beim Abspielen des Makros bleibt diese Zahl natürlich konstant, so daß sich unterschiedlich lange Worte nicht markieren lassen. Um dem vorzubeugen, müssen Sie entweder mit »MARK Equ 0« die erweiterte Sprung-Emulation abschalten oder in Makros nur mit den herkömmlichen, Wordstar-kompatiblen Markierungsbefehlen arbeiten.

Die letzte kleine Unschönheit tritt schließlich noch auf, sobald Sie einen neuen Block markieren, der im Text vor dem letzten Block liegt. Weil Anfang und Ende eines

Info: Zur besseren Lesbarkeit wurde bei der Darstellung der Tastenkombinationen von der sonst üblichen Schreibweise abgewichen. Die Tastenbezeichnungen im einzelnen:

[Home]	[Pos1]
[End]	[Ende]
[PgUp]	[Bild aufwärts]
[PgDn]	[Bild abwärts]
[Up]	[Cursor aufwärts]
[Down]	[Cursor abwärts]
[Left]	[Cursor links]
[Right]	[Cursor rechts]

Blocks – außer mit [Ctrl-KT] – nicht gleichzeitig definiert werden können, sehen Sie in diesem Fall den Rest der Zeile kurzzeitig aufblitzen.
Normalerweise werden residente Programme beim Einschalten des Rechners in den Arbeitsspeicher geladen und verbleiben dort bis zum Ausschalten. Das hier verwendete Utility »saaws.com« geht jedoch einen zweiten Weg, der nur selten benutzt wird. Als eine Art »Huckepack-Programm« ist es nur so lange aktiv, wie auch die zugehörige Anwendung läuft. Dies hat den Vorteil, daß keinerlei Speicherplatz verschwendet wird und Sie zudem die »geshifteten« Cursor-Tasten in anderen Programmen normal benutzen können. Ein Nachteil ist allerdings, daß sich der Aufruf in der Kommandozeile ändert. Wenn Sie zum Beispiel bis jetzt Ihren Editor mit »turbo test« aufgerufen haben, so müssen Sie ihn nun mit

```
saaws c:\pas\tp5\turbo.exe test
```

starten. Dies ist jedoch nur auf den ersten Blick ein Nachteil, denn schließlich gibt es unter MS-DOS ja die Möglichkeit, Befehle mit Stapeldateien abzukürzen. Zudem hat eine Stapeldatei den Vorteil, daß Sie das Programm aus den verschiedensten Unterverzeichnissen heraus aufrufen können. Dies ist zum einen schneller, als die Pfad-Variable von MS-DOS auf jedes Unterverzeichnis mit ausführbaren Programmen zu erweitern, zum anderen ist es übersichtlicher, als alle Programme in einem einzigen Verzeichnis unterzubringen – zumal oft mehrere Programme den gleichen Namen haben. Eine für den vorliegenden Fall passende Stapeldatei ist zum Beispiel folgende:

```
@echo off
saaws c:\pas\tp5\turbo.exe %1 %2 %3
```

Wenn Sie dieser Stapeldatei den Namen »turbo.bat« geben und sie in einem Verzeichnis mit Pfad-Zugriff speichern, rufen Sie wie gewohnt den Editor mit dem Kommando »turbo« auf.
Sämtliche neu hinzugekommenen Tastenkombinationen können Sie nochmals aus der Tabelle ersehen. Als zusätzliche Erweiterung ist ein kleiner Makrorecorder eingebaut. Er läßt sich mit den Tastenkombinationen [Ctrl-M] und [Alt-M] steuern und ist als Hilfe für sich ständig wiederholende Eingaben gedacht. Die Zeichenfolge 8 x [Alt-177] kürzen Sie beispielsweise folgendermaßen ab:

```
[Ctrl-M] [Alt-177] (8 x) [Alt-M]
```

Das wiederholte Eingeben von [Alt-M] ist dabei aufgrund der automatischen Tastenwiederholung nicht so schlimm, wie es aussieht. Die Aufnahmekapazität des Makrorecorders beträgt in dieser Version etwa 200 Anschläge. Sollte Ihnen das nicht ausreichen, reservieren Sie im Assemblerprogramm an der angegebenen Stelle mit »DW n Dup (?)« mehr Speicher für den Puffer.
Zusätzlich zu diesem dynamischen Makro lassen sich feste Makros im Quelltext definieren. Wenn Sie eigene Makros definieren wollen, müssen Sie nur die Tabelle »Makros« um den Scancode der Aufruftaste sowie Adresse und Länge des Ersatz-Strings ergänzen. Beachten Sie aber bitte, daß alle Angaben als Worte abgelegt werden, so daß Zeichenketten etwas umständlich als einzelne Buchstaben angegeben werden müssen.

(Wolfgang Lorenz/bk/zi)

Taste	Funktion
Cursor-Taste mit [Shift]	Schreibmarke bewegen und Ende des markierten Blocks neu setzen. Vorher Blockanfang festlegen, falls nötig. Funktioniert nicht mit den grauen Cursor-Tasten und bei eingeschaltetem [Num-Lock]. Vorsicht: Keine Rückwärtsmarkierung möglich!
[Ctrl-Home/End]	Cursor an den Anfang der ersten/letzten Zeile, entspricht [Ctrl-PgUp/PgDn].
[Ctrl-PgUp/PgDn]	Fenster um eine Zeile nach oben/unten scrollen. Falls möglich, behält der Cursor seine Position im Text bei. Entspricht [Ctrl-W/Z].
[Shift-Ctrl-Left/Right]	Um ein Wort nach links/rechts gehen und neues Blockende festlegen. Steht der Schalter »MARK« auf 1, dann wird das Ende des Wortes aus dem Bildschirmspeicher gelesen.
[Shift-Ctrl-Home/End], [Shift-Ctrl-PgUp/PgDn]	An den Anfang der ersten/letzten Textzeile gehen und Blockende neu setzen.
[Tab/Shift-Tab]	Block ein-/ausrücken. Die Anzahl der Spalten wird in der Konstanten »INDENT« festgelegt. Funktioniert nur bei Editoren, die den Befehl [Ctrl-KI] beziehungsweise [Ctrl-KU] kennen.
[Ctrl-Ins]	Markierten Block verbergen und zum Kopieren freigeben. Im File-Modus Block auf Diskette sichern.
[Shift-Del]	Markierten Block verbergen und zum Verschieben freigeben. Vorsicht: Beim nächsten [Shift-Ins] wird der Block gelöscht, falls zwischenzeitlich nicht ein neuer Block definiert wurde! Im File-Modus Block auf Diskette sichern und löschen.
[Shift-Ins]	Freigegebenen Block an die aktuelle Cursor-Position kopieren/verschieben. Danach wird er wieder zum Kopieren freigegeben, so daß Vervielfältigung mit wiederholtem [Shift-Ins] möglich ist. Im File-Modus Block von Diskette einlesen.
[Del]	An den Blockanfang gehen und markierten Block löschen.
Darstellbares Zeichen	An den Blockanfang gehen, markierten Block löschen und Zeichen ausgeben.
Nicht darstellbare Steuertaste	Markierten Block verbergen. Dieser kann mit [Ctrl-KH] wieder sichtbar gemacht und dann auf herkömmliche Weise verarbeitet werden.
[Ctrl-M]	Makrorecorder einschalten, vorherige Definition löschen. Alle folgenden Tasten werden so lange mitprotokolliert, bis der Puffer voll ist oder die Aufnahme mit [Alt-M] abgebrochen wird. In der Standardversion ist etwa für 200 Tasten Platz. SAA-Blockbefehle werden zuerst umgewandelt und dann gespeichert (jede Cursor-Bewegung entspricht drei Tasten, plus einmal [Ctrl-KB] am Anfang).
[Alt-M]	Bei eingeschaltetem Recorder Aufnahme beenden, ansonsten Makro ausführen.
[Ctrl-S/A]	Die französischen Anführungszeichen (») und («), wie sie auch bei Microsoft Word definiert sind.
[Alt-A]	Zeile einfügen und Anfang eines Pascal-Kommentars
[Alt-S]	Ende eines Pascal-Kommentars gemäß den Konventionen der DOS International (43 Zeichen pro Zeile)
[Alt-Minus]	Kommentarzeile mit Minuszeichen einfügen, dient zur optischen Abtrennung von Programmblöcken.
[Alt-P]	Das Schlüsselwort »Procedure« und ein nachfolgendes Leerzeichen
[Alt-B]	Neuen Block mit »Begin« und »End;« eröffnen.

Tabelle. Die zusätzlich verfügbaren Tastenkombinationen

```
[4G4]     1: ;****************************************
[5Z2]     2: ;     Name: SAAWS.ASM
[247]     3: ;  Aufgabe: Emulation von SAA-Blockbefehlen
[CR4]     4: ;           bei Wordstar-Editoren
[3P5]     5: ;Assembler: Masm 5.Ø, Tasm 1.Ø, Quick
[ZY4]     6: ;           Assembler 2.Ø1 (Manuelles
[SD4]     7: ;           Linken erforderlich)
[D16]     8: ;           Nach dem Binden mit Exe2Bin in
[446]     9: ;           eine COM-Datei übersetzen oder
[ØT5]    1Ø: ;           Schalter /t bei TLink angeben
[KZ6]    11: ;Copyright: 199Ø Redaktionsbüro Everts &
```

```
[HP5]    12: ;           Hagedorn, DOS International
[CH4]    13: ;    Autor: Wolfgang Lorenz
[4G4]    14: ;****************************************
[6B6]    15: ;Diesen Schalter je nach gewünschter
[V66]    16: ;Pufferemulation auf Ø oder 1 stellen
[RK1]    17: FILE Equ 1
[XR5]    18: ;Name für die Pufferdatei als Folge
[X67]    19: ;einzelner Zeichen, je kürzer desto besser
[CL2]    2Ø: FNAME Equ <"$",".">
[596]    21: ;Zeichen, das als Bestätigung für das
[5G6]    22: ;Überschreiben von Blöcken eingegeben
```

```
[DN2]    23: ;werden muß
[A62]    24: OVERWR Equ "y"
[3L6]    25: ;Schalter für die Emulation von »Wort
[7A3]    26: ;markieren rechts«
[BL1]    27: MARK Equ 1
[M17]    28: ;Anzahl der Spalten, um die mit <Tab> bzw.
[4P5]    29: ;<Shift-Tab> eih/ausgerückt wird
[1W1]    30: INDENT Equ 2
[CA4]    31: Code Segment Public 'CODE'
[EA2]    32:     Assume CS:Code
[A61]    33:     Org 100h
[201]    34: Begin:
[VL1]    35:     Jmp Start
[NF6]    36: ;=========================================
[LA2]    37: ; Datenbereich
[NF6]    38: ;=========================================
[W06]    39: UseMsg DB 13,10,"SAAWS Version 1.1 - SAA-"
[V45]    40:     DB "Blockbefehle mit Wordstar-"
[8Y4]    41:     DB "kompatiblen Editoren",13,10
[6E6]    42:     DB "Aufruf: SAAWS <Programmname.{EXE|"
[KL5]    43:     DB "COM}> <Programmparameter>",13,10
[U83]    44: UseLen Equ $-UseMsg
[796]    45: ErrorMsg DB "SAAWS: Das Programm »"
[514]    46: ErrorLen Equ $-ErrorMsg
[9L6]    47: Err2Msg DB "« wurde nicht gefunden",13,10
[EH3]    48: Err2Len Equ $-Err2Msg
[M06]    49: Err8Msg DB "« ist zu groß für den "
[P24]    50:     DB "Arbeitsspeicher",13,10
[6K3]    51: Err8Len Equ $-Err8Msg
[FV6]    52: ErrXMsg DB "« konnte nicht ausgeführt "
[XA2]    53:     DB "werden",13,10
[6P3]    54: ErrXLen Equ $-ErrXMsg
[HK5]    55: Tmp1Msg DB "SAAWS: Die temporäre "
[X13]    56:     DB "Pufferdatei »"
[4L3]    57: Tmp1Len Equ $-Tmp1Msg
[NC7]    58: Tmp2Msg DB "« konnte nicht angelegt werden"
[XY0]    59:     DB 13,10
[8L3]    60: Tmp2Len Equ $-Tmp2Msg
[ER0]    61:     Even
[MS7]    62: ;DTA im PSP und sämtliche Meldungen dürfen
[7P6]    63: ;als Makrospeicher genutzt werden; einen
[0F7]    64: ;größeren Makrospeicher erreichen Sie mit
[MM5]    65: ;»DW n Dup (?)« an dieser Stelle
[268]    66: ;Sie können ständig verfügbare Makros durch
[T77]    67: ;Anhängen von Scancode, Adresse und Länge
[WB6]    68: ;an diese Tabelle definieren (Ende mit 0)
[NS3]    69: Makros DW 1F13h,QuoteL,1
[ZM2]    70:     DW 1E01h,QuoteR,1
[D14]    71:     DW 1E00h,ComStart,ComSLen
[7U3]    72:     DW 1F00h,ComEnd,ComELen
[L73]    73:     DW 3500h,Line,LineLen
[H54]    74:     DW 1900h,Procedure,ProcLen
[9N3]    75:     DW 3000h,Block,BlockLen
[4N4]    76: ;Keine weiteren Makros mehr
[XH0]    77:     DW 0
[065]    78: ;Anführungszeichen auf (CtrlS)
[PY1]    79: QuoteL DW 175
[CY4]    80: ;Anführungszeichen zu (CtrlA)
[6Z1]    81: QuoteR DW 174
[4E6]    82: ;Beginn eines Pascal-Kommentars (AltA)
[7N4]    83: ComStart DW 4700h,310Eh,"{"," "
[HZ4]    84: ComSLen Equ ($-ComStart) Shr 1
[987]    85: ;Ende eines Pascal-Kommentars in Spalte 43;
[4C7]    86: ;die negative Zahl steht für Wiederholung,
[H05]    87: ;also 42 mal 4D00 (Cursor rechts)
[K45]    88: ComEnd DW 4700h,-42,4D00h,1011h,2C79h
[SC2]    89:     DW "}",4700h,5000h
[3A4]    90: ComELen Equ ($-ComEnd) Shr 1
[R26]    91: ;Kommentarzeile mit Strichen (Alt-)
[ZP5]    92: Line DW 4700h,310Eh,"{"," ",-39,"-"," ","}"
[6V1]    93:     DW 4700h,5000h
[L44]    94: LineLen Equ ($-Line) Shr 1
[2U3]    95: ;»Procedure « (AltP)
[KN5]    96: Procedure DW "P","r","o","c","e","d","u"
[LL1]    97:     DW "r","e"," "
[945]    98: ProcLen Equ ($-Procedure) Shr 1
[2Y5]    99: ;Block mit »Begin« und »End« (AltB)
[CC5]   100: Block DW "B","e","g","i","n",-4,13,"E","n"
[XD3]   101:     DW "d",";",-2,4B00h,-3,4800h
[274]   102: BlockLen Equ ($-Block) Shr 1
[BB6]   103: ;Hier fangen die normalen Variablen an
[CU1]   104: EnvSeg DW ?
[ZS1]   105: CmdOfs DW ?
[XR1]   106: CmdSeg DW ?
[A82]   107: FCB1Ofs DW 5Ch
[BW1]   108: FCB1Seg DW ?
[C82]   109: FCB2Ofs DW 6Ch
[CW1]   110: FCB2Seg DW ?
[WE2]   111: Terminator DB 0
[0002]  112: NameOfs DW ?
[3Z1]   113: NameLen DW ?
[TD2]   114: AlterInt09 DD ?
[KD2]   115: AlterInt16 DD ?
[2S1]   116: IF FILE EQ 1
[W43]   117: TempName DB FNAME,0
[034]   118: TempLen Equ $-TempName-1
[TU0]   119: ENDIF
[NF6]   120: ;=========================================
[KL5]   121: ; Neuer Tastaturinterrupt 09hex
[NF6]   122: ;=========================================
[CZ6]   123: ;Nummern der Tasten, wie sie die Tastatur
[UD1]   124: ;liefert
[0S5]   125: ScanCodes DB 48h,50h,4Bh,4Dh,47H,4Fh,49h
[1W1]   126:     DB 51h,52h,53h
[5L3]   127: ScanNr Equ $-ScanCodes
[ER0]   128:     Even
[MF6]   129: ;Nummern der Tasten, die in den Puffer
[8E6]   130: ;geschrieben werden (0000h = Ignorieren)
[875]   131: Shift DW 0148h,0150h,014Bh,014Dh,0147h
[464]   132:     DW 014Fh,0149h,0151h,0152h,0153h
[185]   133: Ctrl DW 1117h,2C1Ah,7300h,7400h,8400h
[R24]   134:     DW 7600h,8400h,7600h,0100h,0000h
[445]   135: SCtrl DW 0000h,0000h,0173h,0174h,0184h
[R34]   136:     DW 0176h,0184h,0176h,0000h,0000h
[7M2]   137: NeuerInt09 Proc
[PC1]   138:     Push AX
[WC1]   139:     Push CX
[3B1]   140:     Push DI
[EC1]   141:     Push ES
[AC6]   142: ;BIOS-Variable »Shift-Status« lesen
[GH1]   143:     Xor AX,AX
[3H1]   144:     Mov ES,AX
[RS2]   145:     Mov AH,ES:[0417h]
[L95]   146: ;Auf <NumLock> und <Alt> prüfen
[KW1]   147:     Test AH,28h
[F62]   148:     Jne ExecOld09
[MD5]   149: ;Auf <Shift> und <Ctrl> prüfen
[BW1]   150:     Test AH,07h
[ZU1]   151:     Je ExecOld09
[9G4]   152: ;Taste in der Tabelle suchen
[VF1]   153:     Mov DI,CS
[8G1]   154:     Mov ES,DI
[H24]   155:     Mov DI,Offset ScanCodes
[382]   156:     Mov CX,ScanNr
[H71]   157:     In AL,60h
[UL0]   158:     Cld
[XZ1]   159:     Repne Scasb
[EY1]   160:     Je KeyFound
[BP1]   161: ExecOld09:
[5U6]   162: ;Taste nicht gefunden, zum alten Tastatur-
[6G3]   163: ;Interrupt springen
[G31]   164:     Pop ES
[G21]   165:     Pop DI
[V31]   166:     Pop CX
[R31]   167:     Pop AX
[CF2]   168:     Jmp AlterInt09
[WM1]   169: KeyFound:
[TH5]   170: ;Puffercode der Taste bestimmen
[C54]   171:     Sub DI,Offset ScanCodes+1
[Y66]   172: ;Taste zusammen mit <Ctrl> gedrückt?
[2W1]   173:     Test AH,04h
[5X1]   174:     Je CheckKey
[R22]   175:     Add DI,ScanNr
[MD7]   176: ;Taste zusammen mit <Shift-Ctrl> gedrückt?
[ZV1]   177:     Test AH,03h
[5X1]   178:     Je CheckKey
[R22]   179:     Add DI,ScanNr
[DG1]   180: CheckKey:
[SL7]   181: ;Puffercode holen und auf Gültigkeit prüfen
[K81]   182:     Shl DI,1
[BG3]   183:     Mov AX,ES:[Shift+DI]
[7E1]   184:     And AX,AX
[122]   185:     Je Ungueltig
[YM6]   186: ;Puffercode in den BIOS-Tastaturpuffer
[MS1]   187: ;eintragen
[6U1]   188:     Mov DI,0040h
[8G1]   189:     Mov ES,DI
[Z96]   190: ;Ist überhaupt noch Platz im Puffer?
[8U2]   191:     Mov DI,ES:[001Ch]
[CL2]   192:     Lea CX,[DI-001Ch]
[TU1]   193:     And CX,001Eh
[VT1]   194:     Add CX,001Eh
[TT2]   195:     Cmp CX,ES:[001Ah]
[122]   196:     Je Ungueltig
[217]   197: ;Puffercode schreiben und Zeiger erhöhen
[KK2]   198:     Mov ES:[DI],AX
[6S2]   199:     Mov ES:[001Ch],CX
[8U1]   200: Ungueltig:
[C17]   201: ;Mit der Taktleitung den Empfang der Taste
[392]   202: ;bestätigen
[K71]   203:     In AL,61h
[H81]   204:     Or AL,80h
[YN1]   205:     Out 61h,AL
[NK1]   206:     Jmp Delay
[811]   207: Delay:
[A42]   208:     And AL,Not 80h
[YN1]   209:     Out 61h,AL
[G97]   210: ;Interruptbestätigung an den Controller
[5M1]   211:     Mov AL,61h
[LN1]   212:     Out 20h,AL
[497]   213: ;Register vom Stapel und Interrupt beenden
[G31]   214:     Pop ES
[G21]   215:     Pop DI
[V31]   216:     Pop CX
[R31]   217:     Pop AX
```

Listing 1. Das Assemblerprogramm »saaws.asm« erlaubt SAA-Blockbefehle bei Wordstar-kompatiblen Editoren

```
[GRØ]  218:      Iret
[7M2]  219: NeuerIntØ9 Endp
[NF6]  22Ø: ;=======================================
[LL5]  221: ; Neuer Tastaturinterrupt 16hex
[NF6]  222: ;=======================================
[ERØ]  223:      Even
[323]  224: CtrlKB DW 25ØBh,3Ø62h
[AD1]  225: CodeØ DB Ø
[TE1]  226: Code1 DB ?
[XY1]  227:      DW 25ØBh,256Bh
[UP4]  228: CtrlQB DW 1Ø11h,3Ø62h,25ØBh,2C79h
[GF1]  229: Code2 DW ?
[233]  23Ø: CtrlKH DW 25ØBh,2368h
[KF1]  231: Code3 DW ?
[YØ3]  232: CtrlR1 DW 25ØBh,3Ø62h
[Y44]  233: CtrlR2 DW ?,4DØØh,25ØBh,256Bh
[8W5]  234: CtrlKI DW INDENT Dup (25ØBh,1769h),1Ø11h
[V41]  235:      DW 256Bh
[9X5]  236: CtrlKU DW INDENT Dup (25ØBh,1675h),1Ø11h
[V41]  237:      DW 256Bh
[2S1]  238: IF FILE EQ 1
[ØØ6]  239: CtrlKW DW 25ØBh,1177h,FNAME,1CØDh,OVERWR
[7G3]  24Ø:      DW 1Ø11h,3Ø62h,25ØBh,2C79h
[NW5]  241: CtrlKR DW 25ØBh,1372h,FNAME,1CØDh,25ØBh
[F31]  242:      DW 2368h
[ZRØ]  243: ELSE
[YS5]  244: CtrlKV DW 25ØBh,2368h,25ØBh,2F76h,25ØBh
[F31]  245:      DW 2368h
[VP5]  246: CtrlKC DW 25ØBh,2368h,25ØBh,2E63h,25ØBh
[F31]  247:      DW 2368h
[TUØ]  248: ENDIF
[BØ2]  249: ReadPtr DW ?
[SY1]  25Ø: ReadCnt DW Ø
[M32]  251: MovePtr DW ?
[L62]  252: MacroPtr DW Ø
[P42]  253: MacroCnt DW Ø
[XA2]  254: RepeatKey DW ?
[F82]  255: RepeatCnt DW Ø
[G62]  256: BlockFlag DB Ø
[L32]  257: MoveFlag DB Ø
[R93]  258: NeuerInt16 Proc Far
[NK6]  259: ;Anzahl der Zeichen im internen Puffer
[3B1]  26Ø:      Push DI
[WC2]  261:      Mov DI,ReadCnt
[BG8]  262: ;Prüfen auf Funktion »Warte und lese Taste«
[RB1]  263:      And AH,AH
[KL1]  264:      Je ReadKey
[ØK1]  265:      Cmp AH,1Øh
[KL1]  266:      Je ReadKey
[7R7]  267: ;Prüfen auf Funktion »Hole Puffer-Status«
[1K1]  268:      Cmp AH,Ø1h
[642]  269:      Je FunktionØ1
[2K1]  27Ø:      Cmp AH,11h
[A62]  271:      Jne ExecOld16
[PZ1]  272: FunktionØ1:
[E27]  273: ;Wenn ein Wiederholungscode gesetzt ist,
[KZ6]  274: ;dann »Taste gedrückt« zurückgeben
[9M2]  275:      Cmp RepeatCnt,Ø
[LH2]  276:      Jne KeyPressed
[6L5]  277: ;Bei leerem internen Puffer alte
[VA5]  278: ;Interruptroutine ausführen
[5C1]  279:      And DI,DI
[ØV1]  28Ø:      Je ExecOld16
[932]  281: KeyPressed:
[G95]  282: ;Zero-Flag auf dem Stapel löschen
[5G1]  283:      Mov DI,SP
[915]  284:      And Byte Ptr SS:[DI+6],Not 4Øh
[PD4]  285: ;Dummy-Taste zurückgeben
[BV1]  286:      Mov AX,ØØ21h
[G21]  287:      Pop DI
[GRØ]  288:      Iret
[6P1]  289: ExecOld16:
[HN6]  29Ø: ;Zum alten Tastaturinterrupt springen
[G21]  291:      Pop DI
[5F2]  292:      Jmp AlterInt16
[YD1]  293: ReadKey:
[Y26]  294: ;Es soll eine Taste gelesen werden,
[B75]  295: ;Wiederholungszähler prüfen
[RU2]  296:      Mov AX,RepeatCnt
[7E1]  297:      And AX,AX
[RV1]  298:      Je CheckBuf
[NR7]  299: ;Zähler erniedrigen und Taste zurückgeben
[DØ1]  3ØØ:      Dec AX
[RU2]  3Ø1:      Mov RepeatCnt,AX
[GU2]  3Ø2:      Mov AX,RepeatKey
[5F2]  3Ø3:      Jmp CheckMacro
[ZG1]  3Ø4: CheckBuf:
[L45]  3Ø5: ;Internen Puffer überprüfen
[5C1]  3Ø6:      And DI,DI
[AØ2]  3Ø7:      Je IgnoreKey
[RF2]  3Ø8:      Jmp ReadBuffer
[KT1]  3Ø9: IgnoreKey:
[FC6]  31Ø: ;Interner Puffer ist leer, Taste holen
[ØF1]  311:      Xor AH,AH
[ZXØ]  312:      Pushf
[HK2]  313:      Call AlterInt16
[ET5]  314: ;Falls ein Block an ist, dann Makro-
[GF7]  315: ;Ausführung übergehen (Rekursionsproblem)
[ZK2]  316:      Cmp BlockFlag,Ø
[A72]  317:      Jne NotFound
[M67]  318: ;Mit den vordefinierten Makros vergleichen
[8F3]  319:      Mov DI,Offset Makros
[UR1]  32Ø: SearchKey:
[EU6]  321: ;Ist das Ende der Makro-Tabelle erreicht?
[ØU3]  322:      Cmp Word Ptr CS:[DI],Ø
[PY1]  323:      Je NotFound
[3S4]  324: ;Wurde der Scancode gefunden?
[RF2]  325:      Cmp AX,CS:[DI]
[3Ø2]  326:      Je CodeFound
[9R6]  327: ;Nein, mit nächstem Scancode vergleichen
[761]  328:      Add DI,6
[792]  329:      Jmp SearchKey
[TS1]  33Ø: CodeFound:
[W75]  331: ;Pufferzeiger auf Makro setzen
[SN2]  332:      Mov AX,CS:[DI+2]
[TM2]  333:      Mov DI,CS:[DI+4]
[ZA2]  334:      Jmp SetBuffer
[CN1]  335: NotFound:
[ZB6]  336: ;Auf Cursortaste mit <Shift> prüfen
[1K1]  337:      Cmp AH,Ø1h
[3K2]  338:      Jne JmpNoShift
[S64]  339: ;Ist es die Escape-Taste?
[WV1]  34Ø:      Cmp AX,Ø11Bh
[AR2]  341:      Jne CheckInsDel
[T32]  342: JmpNoShift:
[C52]  343:      Jmp NoShift
[K32]  344: CheckInsDel:
[D15]  345: ;Puffercode der Taste sichern
[XZ1]  346:      Mov Code1,AL
[654]  347: ;Auf <Shift-Del> prüfen
[CU1]  348:      Cmp AX,Ø153h
[332]  349:      Jne CheckIns
[WX6]  35Ø: ;Bei unmarkiertem Block Taste ignorieren
[ZK2]  351:      Cmp BlockFlag,Ø
[AØ2]  352:      Je IgnoreKey
[FC6]  353: ;Block-Flag löschen, Move-Flag setzen
[HM2]  354:      Mov BlockFlag,Ø
[1M2]  355:      Mov MoveFlag,-1
[2S1]  356: IF FILE EQ 1
[C17]  357: ;Block mit <Ctrl-KW> speichern und löschen
[AF3]  358:      Mov AX,Offset CtrlKW
[VN2]  359:      Mov DI,TempLen+8
[ZRØ]  36Ø: ELSE
[NA6]  361: ;Block mit <Ctrl-KH> unsichtbar machen
[VE3]  362:      Mov AX,Offset CtrlKH
[4A1]  363:      Mov DI,2
[SR6]  364: ;Sequenz für <Shift-Ins> auf <Ctrl-KV>
[WG4]  365:      Mov MovePtr,Offset CtrlKV
[TUØ]  366: ENDIF
[ZA2]  367:      Jmp SetBuffer
[5H1]  368: CheckIns:
[HØ4]  369: ;Auf <Ctrl-Ins> prüfen
[HT1]  37Ø:      Cmp AX,Ø1ØØh
[852]  371:      Jne ShiftIns
[WX6]  372: ;Bei unmarkiertem Block Taste ignorieren
[ZK2]  373:      Cmp BlockFlag,Ø
[AØ2]  374:      Je IgnoreKey
[XU2]  375: ;Move-Flag setzen
[1M2]  376:      Mov MoveFlag,-1
[2S1]  377: IF FILE EQ 1
[CT6]  378: ;Block mit <Ctrl-KW> auf Diskette sichern
[AF3]  379:      Mov AX,Offset CtrlKW
[RN2]  38Ø:      Mov DI,TempLen+4
[ZRØ]  381: ELSE
[6Z4]  382: ;Block mit <Ctrl-KH> verbergen
[VE3]  383:      Mov AX,Offset CtrlKH
[4A1]  384:      Mov DI,2
[WH3]  385: ;Block-Flag löschen
[HM2]  386:      Mov BlockFlag,Ø
[KP6]  387: ;Sequenz für <Shift-Ins> auf <Ctrl-KC>
[PF4]  388:      Mov MovePtr,Offset CtrlKC
[TUØ]  389: ENDIF
[ZA2]  39Ø:      Jmp SetBuffer
[AL1]  391: ShiftIns:
[774]  392: ;Auf <Shift-Ins> prüfen
[8U1]  393:      Cmp AX,Ø152h
[AF2]  394:      Jne CheckRight
[YM6]  395: ;Bei nicht gesichertem Block ignorieren
[MC2]  396:      Cmp MoveFlag,Ø
[5W2]  397:      Jne NoIgnoreKey
[ZA2]  398:      Jmp IgnoreKey
[D82]  399: NoIgnoreKey:
[2S1]  4ØØ: IF FILE EQ 1
[WH3]  4Ø1: ;Block-Flag löschen
[HM2]  4Ø2:      Mov BlockFlag,Ø
[5T4]  4Ø3: ;Block mit <Ctrl-KR> einlesen
[5F3]  4Ø4:      Mov AX,Offset CtrlKR
[SN2]  4Ø5:      Mov DI,TempLen+5
[ZRØ]  4Ø6: ELSE
[KP6]  4Ø7: ;Sequenz für <Shift-Ins> auf <Ctrl-KC>
[PE3]  4Ø8:      Mov AX,Offset CtrlKC
[1K7]  4Ø9: ;Alte Shift-Ins-Sequenz in Puffer schreiben
[RV2]  41Ø:      Xchg AX,MovePtr
[8A1]  411:      Mov DI,6
[TUØ]  412: ENDIF
[ZA2]  413:      Jmp SetBuffer
[Ø12]  414: CheckRight:
[3T1]  415: IF MARK EQ 1
```

```
[9L5]  416: ;Auf <Shift-Ctrl-Right> prüfen
[PU1]  417:     Cmp AX,Ø174h
[582]  418:     Jne CheckCrsr
[Ø16]  419: ;Adresse des Cursors berechnen, lade
[MV7]  42Ø: ;Segmentregister für Zugriff auf BIOS-Daten
[WC1]  421:     Push CX
[BC1]  422:     Push DS
[GH1]  423:     Xor AX,AX
[1H1]  424:     Mov DS,AX
[9V6]  425: ;Nummer der aktuellen Seite mal 2 nach DI
[US2]  426:     Mov DI,DS:[Ø462h]
[HV1]  427:     And DI,ØØFFh
[K81]  428:     Shl DI,1
[YU6]  429: ;Position des Cursors auf der Seite holen
[393]  43Ø:     Mov CX,DS:[Ø45Øh+DI]
[4Ø7]  431: ;Zeichen pro Zeile mal Cursorzeile nach AX
[LU2]  432:     Mov AL,DS:[Ø44Ah]
[XM1]  433:     Xchg AL,CH
[721]  434:     Mul CH
[K66]  435: ;Aktuelle Cursorspalte dazuaddieren
[FB1]  436:     Add AL,CL
[M51]  437:     Adc AH,Ø
[UG6]  438: ;Mal zwei, ergibt Adresse im Bildschirm
[T91]  439:     Shl AX,1
[VS6]  44Ø: ;Plus Anfangsadresse der aktuellen Seite
[6S2]  441:     Add AX,DS:[Ø44Eh]
[2Z5]  442: ;Offsetadresse im Bildschirm nach DI
[3G1]  443:     Mov DI,AX
[4E4]  444: ;Segmentadresse berechnen
[XZ1]  445:     Mov AX,ØBØØØh
[X75]  446: ;Ist es ein Monochrom-Adapter?
[XN4]  447:     Cmp Byte Ptr DS:[Ø463h],ØB4h
[622]  448:     Je Monochrom
[NA6]  449: ;Nein, dann Segment für Farbe setzen
[5W1]  45Ø:     Mov AH,ØB8h
[BU1]  451: Monochrom:
[Y56]  452: ;Datensegment auf Bildschirmspeicher
[1H1]  453:     Mov DS,AX
[SS6]  454: ;Anzahl der verbleibenden Zeichen in der
[Z72]  455: ;Zeile nach CH
[PD1]  456:     Sub CH,CL
[KZ5]  457: ;Anzahl der Zeichen initialisieren
[8D1]  458:     Mov CL,-1
[DT4]  459: ;Nach einem Buchstaben suchen
[W82]  46Ø: SearchLetter:
[GØ2]  461:     Call Letter
[LD2]  462:     Jc LetterFound
[MU4]  463: ;Weitersuchen, bis Zeilenende
[XB1]  464:     And CH,CH
[UU2]  465:     Jne SearchLetter
[BK3]  466:     Jmp Short OldWRight
[G62]  467: LetterFound:
[876]  468: ;Bei Zeilenende normales <Ctrl-Right>
[XB1]  469:     And CH,CH
[ZØ2]  47Ø:     Je OldWRight
[KD6]  471: ;Suchen, bis kein Buchstabe mehr kommt
[GØ2]  472:     Call Letter
[LD2]  473:     Jc LetterFound
[147]  474: ;Zielposition gefunden, Anzahl der Zeichen
[3S3]  475: ;nach rechts eintragen
[6F1]  476:     Xor CH,CH
[N11]  477:     Neg CX
[K72]  478:     Mov CtrlR2,CX
[UK6]  479: ;Gesicherte Register wiederherstellen
[E31]  48Ø:     Pop DS
[V31]  481:     Pop CX
[ZN5]  482: ;Puffer auf Zeichenkette setzen
[1F3]  483:     Mov AX,Offset CtrlR2
[6A1]  484:     Mov DI,4
[XS4]  485: ;Ist bereits ein Block aktiv?
[ZK2]  486:     Cmp BlockFlag,Ø
[P92]  487:     Jne SetBuffer
[2Y6]  488: ;Nein, dann zusätzlich <Ctrl-KB> ausgeben
[ØF3]  489:     Mov AX,Offset CtrlR1
[MG3]  49Ø:     Jmp Short SetCtrlKB
[2T1]  491: OldWRight:
[WN5]  492: ;Normaler Wortsprung nach rechts
[E31]  493:     Pop DS
[V31]  494:     Pop CX
[TUØ]  495: ENDIF
[1S1]  496: CheckCrsr:
[7R5]  497: ;Auf bereits aktiven Block prüfen
[H73]  498:     Mov AX,Offset CodeØ
[5A1]  499:     Mov DI,3
[ZK2]  5ØØ:     Cmp BlockFlag,Ø
[P92]  5Ø1:     Jne SetBuffer
[9T6]  5Ø2: ;Bei nicht aktivem Block muß der Code für
[X87]  5Ø3: ;»Blockanfang setzen« geschrieben werden
[NE3]  5Ø4:     Mov AX,Offset CtrlKB
[LU1]  5Ø5: SetCtrlKB:
[BØ1]  5Ø6:     Inc DI
[BØ1]  5Ø7:     Inc DI
[623]  5Ø8: ;Block-Flag setzen
[PN2]  5Ø9:     Mov BlockFlag,-1
[1S1]  51Ø: IF FILE EQ Ø
[U33]  511: ;Move-Flag löschen
[6E2]  512:     Mov MoveFlag,Ø
[TUØ]  513: ENDIF
[KT1]  514: SetBuffer:
```

```
[R74]  515: ;Pufferzeiger neu setzen
[MK2]  516:     Mov ReadPtr,AX
[4Ø2]  517: ReadBuffer:
[DT6]  518: ;Anzahl der Tasten im Puffer verringern
[6ZØ]  519:     Dec DI
[9G2]  52Ø:     Mov ReadCnt,DI
[HW5]  521: ;Taste aus dem internen Puffer lesen
[FD2]  522:     Mov DI,ReadPtr
[9H2]  523:     Mov AX,CS:[DI]
[DV5]  524: ;Zeiger auf nächste Taste setzen
[BØ1]  525:     Inc DI
[BØ1]  526:     Inc DI
[N17]  527: ;Ist es ein negativer Wiederholungscode?
[FS1]  528:     Cmp AX,-256
[B32]  529:     Jbe NoRepeat
[RM4]  53Ø: ;Ja, dann Anzahl - 1 speichern
[341]  531:     Not AX
[RU2]  532:     Mov RepeatCnt,AX
[F37]  533: ;Zu wiederholende Taste lesen und sichern
[9H2]  534:     Mov AX,CS:[DI]
[SV2]  535:     Mov RepeatKey,AX
[CX6]  536: ;Anzahl der Zeichen im Puffer erniedrigen
[YZ1]  537:     Dec ReadCnt
[UX6]  538: ;Zeiger auf das nächste Zeichen setzen
[BØ1]  539:     Inc DI
[BØ1]  54Ø:     Inc DI
[HK1]  541: NoRepeat:
[WB5]  542: ;Pufferzeiger zurückschreiben
[DH2]  543:     Mov ReadPtr,DI
[HF3]  544:     Jmp Short CheckMacro
[BG1]  545: NoShift:
[EC6]  546: ;Es war keine geshiftete Cursortaste
[ZK2]  547:     Cmp BlockFlag,Ø
[N12]  548:     Je CheckMacro
[GD5]  549: ;Auf <Tab> und <Shift-Tab> prüfen
[ZV1]  55Ø:     Cmp AX,ØF Ø9h
[DZ1]  551:     Je IsIndent
[VU1]  552:     Cmp AX,ØFØØh
[912]  553:     Jne CheckDel
[385]  554: ;Block mit <Ctrl-KU> ausrücken
[8F3]  555:     Mov AX,Offset CtrlKU
[3G3]  556:     Jmp Short ExecIndent
[MK1]  557: IsIndent:
[A55]  558: ;Block mit <Ctrl-KI> einrücken
[WE3]  559:     Mov AX,Offset CtrlKI
[U12]  56Ø: ExecIndent:
[PG7]  561: ;Anzahl der Spalten holen und Puffer setzen
[C83]  562:     Mov DI,INDENT Shl 1+2
[ZA2]  563:     Jmp SetBuffer
[BF1]  564: CheckDel:
[XY7]  565: ;Kennzeichen für »Block gesetzt« löschen
[HM2]  566:     Mov BlockFlag,Ø
[3Ø4]  567: ;Auf Taste <Del> prüfen
[6A1]  568:     Mov DI,4
[UT1]  569:     Cmp AX,53ØØh
[L92]  57Ø:     Je DeleteBlock
[6Y4]  571: ;Auf normales Zeichen prüfen
[DK1]  572:     Cmp AL,2Øh
[M52]  573:     Jb SteuerCode
[UY6]  574: ;Bei normalem Zeichen Block löschen und
[HW2]  575: ;Zeichen ausgeben
[BØ2]  576:     Mov Code2,AX
[BØ1]  577:     Inc DI
[432]  578: DeleteBlock:
[DF3]  579:     Mov AX,Offset CtrlQB
[ZA2]  58Ø:     Jmp SetBuffer
[512]  581: SteuerCode:
[U57]  582: ;Bei Steuerzeichen Block unsichtbar machen
[DØ2]  583:     Mov Code3,AX
[6ZØ]  584:     Dec DI
[VE3]  585:     Mov AX,Offset CtrlKH
[ZA2]  586:     Jmp SetBuffer
[KZ1]  587: CheckMacro:
[GØ7]  588: ;Flag für Aufzeichnung lesen und löschen
[EF1]  589:     Xor DI,DI
[7T2]  59Ø:     Xchg DI,MacroPtr
[XA4]  591: ;Wurde <Ctrl-M> gedrückt?
[6W1]  592:     Cmp AX,32ØDh
[R22]  593:     Jne NoCtrlM
[5N5]  594: ;Ja, dann Aufzeichnung einschalten
[NP2]  595:     Mov MacroPtr,128
[NC2]  596:     Mov MacroCnt,Ø
[A92]  597: ReturnDummy:
[PD4]  598: ;Dummy-Taste zurückgeben
[3V1]  599:     Mov AX,Ø1ØØh
[W13]  6ØØ:     Jmp Short Zurueck
[ZE1]  6Ø1: NoCtrlM:
[Ø34]  6Ø2: ;Wurde <Alt-M> gedrückt?
[PT1]  6Ø3:     Cmp AX,32ØØh
[US1]  6Ø4:     Jne NoAltM
[676]  6Ø5: ;Ist die Aufzeichnung ausgeschaltet?
[5C1]  6Ø6:     And DI,DI
[2Z1]  6Ø7:     Je ExecMacro
[7M3]  6Ø8: ;Nein, dann abschalten
[KE2]  6Ø9:     Mov MacroPtr,Ø
[BY2]  61Ø:     Jmp ReturnDummy
```

Listing 1. Das Assemblerprogramm »saaws.asm« erlaubt SAA-Blockbefehle bei Wordstar-kompatiblen Editoren

```
[GR1]  611: ExecMacro:
[F87]  612: ;Makro bei mindestens einer Taste im Puffer
[Ø12]  613: ;ausführen
[1S2]  614:     Mov DI,MacroCnt
[5C1]  615:     And DI,DI
[FG2]  616:     Je ReturnDummy
[Z86]  617: ;Pufferzeiger auf Makropuffer setzen
[PL1]  618:     Mov AX,128
[ZA2]  619:     Jmp SetBuffer
[GA1]  62Ø: NoAltM:
[CC5]  621: ;Ist die Makro-Aufzeichnung an?
[5C1]  622:     And DI,DI
[XP1]  623:     Je Zurueck
[V66]  624: ;Ist noch genügend Platz im Puffer?
[PD3]  625:     Cmp DI,Offset Makros
[U22]  626:     Jae Zurueck
[BV5]  627: ;Ja, dann Taste in Puffer schreiben
[EK2]  628:     Mov CS:[DI],AX
[HM5]  629: ;Pufferzeiger und Anzahl erhöhen
[BØ1]  63Ø:     Inc DI
[BØ1]  631:     Inc DI
[KT2]  632:     Mov MacroPtr,DI
[P22]  633:     Inc MacroCnt
[9G1]  634: Zurueck:
[R34]  635: ;Taste in AX zurückgeben
[G21]  636:     Pop DI
[GRØ]  637:     Iret
[ØM2]  638: NeuerInt16 Endp
[3T1]  639: IF MARK EQ 1
[SD6]  64Ø: ;Zeichen auf Buchstaben überprüfen
[TP2]  641: Letter Proc Near
[X17]  642: ;Zeichen aus dem Bildschirmspeicher holen
[6G2]  643:     Mov AL,DS:[DI]
[UX6]  644: ;Zeiger auf das nächste Zeichen setzen
[BØ1]  645:     Inc DI
[BØ1]  646:     Inc DI
[8W6]  647: ;Anzahl der gelesenen Zeichen vermerken
[KØ1]  648:     Inc CL
[1ZØ]  649:     Dec CH
[RC4]  65Ø: ;Prüfen auf Unterstrich
[9H1]  651:     Cmp AL,"_"
[XY1]  652:     Je IsLetter
[RW2]  653: ;Prüfen auf Zahl
[BE1]  654:     Cmp AL,"Ø"
[HY1]  655:     Jb NoLetter
[XE1]  656:     Cmp AL,"9"
[S42]  657:     Jbe IsLetter
[GP4]  658: ;Ist es ein Großbuchstabe?
[DF1]  659:     Cmp AL,"A"
[HY1]  66Ø:     Jb NoLetter
[ZG1]  661:     Cmp AL,"Z"
[S42]  662:     Jbe IsLetter
[2V4]  663: ;Ist es ein Kleinbuchstabe?
[DF1]  664:     Cmp AL,"a"
[HY1]  665:     Jb NoLetter
[ZG1]  666:     Cmp AL,"z"
[S42]  667:     Jbe IsLetter
[277]  668: ;Prüfen auf Akzente oder deutsche Umlaute
[ØH1]  669:     Cmp AL,128
[HY1]  67Ø:     Jb NoLetter
[UG1]  671:     Cmp AL,154
[S42]  672:     Jbe IsLetter
[GG1]  673:     Cmp AL,16Ø
[HY1]  674:     Jb NoLetter
[ZG1]  675:     Cmp AL,165
[S42]  676:     Jbe IsLetter
[N56]  677: ;Ist es ein griechischer Buchstabe?
[NG1]  678:     Cmp AL,224
[HY1]  679:     Jb NoLetter
[UG1]  68Ø:     Cmp AL,235
[S42]  681:     Jbe IsLetter
[WL1]  682: NoLetter:
[V56]  683: ;Kein Buchstabe, Carry-Flag löschen
[PLØ]  684:     Clc
[3PØ]  685:     Ret
[YL1]  686: IsLetter:
[6U6]  687: ;Es ist ein Buchstabe, Carry-Flag setzen
[ENØ]  688:     Stc
[3PØ]  689:     Ret
[SZ1]  69Ø: Letter Endp
[TUØ]  691: ENDIF
[NF6]  692: ;=========================================
[DS2]  693: ; Hauptprogramm
[NF6]  694: ;=========================================
[FA2]  695:     Assume DS:Code
[DU1]  696: Anleitung:
[6B7]  697: ;Anleitungstext ausgeben und Programmende
[1H3]  698:     Mov SI,Offset UseMsg
[D72]  699:     Mov DI,UseLen
[CS6]  7ØØ: ;Ausgabe einer Fehlermeldung und Abbruch
[8Z1]  7Ø1: FehlerEnde:
[DB1]  7Ø2:     Mov BX,2
[4H1]  7Ø3:     Mov DX,SI
[LG1]  7Ø4:     Mov CX,DI
[NL1]  7Ø5:     Mov AH,4Øh
[381]  7Ø6:     Int 21h
[TB5]  7Ø7: ;Programm mit Fehlercode beenden
[512]  7Ø8:     Mov AX,4CFFh
[381]  7Ø9:     Int 21h
```

```
[ØA6]  71Ø: ;Einsprungpunkt für Programmstart
[E21]  711: Start:
[XØ6]  712: ;Vom COM-Programm nicht benötigten
[EZ6]  713: ;Speicherplatz freigeben, 512 Bytes Stapel
[ZH7]  714:     Mov SP,(Offset Ende-Offset Begin+3ØFh)
       And ØFFFØh
[LT6]  715:     Mov BX,(Offset Ende-Offset Begin+3ØFh)
       Shr 4
[RM1]  716:     Mov AH,4Ah
[381]  717:     Int 21h
[K95]  718: ;Länge der Kommandozeile holen
[EP2]  719:     Mov CL,DS:[128]
[6F1]  72Ø:     Xor CH,CH
[2D5]  721: ;Leerzeichen vor dem Dateinamen
[TH2]  722: ;überspringen
[KK1]  723:     Mov DI,129
[XE1]  724:     Mov AL," "
[AR1]  725:     Repe Scasb
[K97]  726: ;Bei fehlendem Parameter Anleitung ausgeben
[7T1]  727:     Jne Weiter
[TG2]  728:     Jcxz Anleitung
[XA1]  729: Weiter:
[WH5]  73Ø: ;Anfang des Dateinamens sichern
[6ZØ]  731:     Dec DI
[P11]  732:     Inc CX
[6G1]  733:     Mov SI,DI
[KL4]  734: ;Ende des Dateinamens suchen
[XZ1]  735:     Repne Scasb
[252]  736:     Cmp [DI-1],AL
[P12]  737:     Jne NoSpace
[6ZØ]  738:     Dec DI
[P11]  739:     Inc CX
[YD1]  74Ø: NoSpace:
[G36]  741: ;Zwischen Programmnamen und dessen
[B66]  742: ;Kommandozeile 2 Bytes Platz schaffen
[WC1]  743:     Push CX
[LG1]  744:     Mov CX,DI
[GF1]  745:     Sub CX,SI
[GH2]  746:     Mov NameLen,CX
[UØ2]  747:     Lea DI,[SI-2]
[2H2]  748:     Mov NameOfs,DI
[9M1]  749:     Rep Movsb
[5E5]  75Ø: ;Dateinamen mit Ø abschließen
[MF1]  751:     Xor AL,AL
[NXØ]  752:     Stosb
[V25]  753: ;Adresse der Kommandozeile des
[MC4]  754: ;Unterprozesses sichern
[762]  755:     Mov CmdSeg,CS
[P62]  756:     Mov CmdOfs,DI
[WZ5]  757: ;Länge der Kommandozeile eintragen
[R31]  758:     Pop AX
[NXØ]  759:     Stosb
[B56]  76Ø: ;Segmentadresse des Environment an
[KA4]  761: ;Unterprozeß übergeben
[1U2]  762:     Mov AX,DS:[2Ch]
[Y82]  763:     Mov EnvSeg,AX
[P17]  764: ;Segmentadressen der FCBs im aktuellen PSP
[MY1]  765: ;übergeben
[9E2]  766:     Mov FCB1Seg,CS
[AE2]  767:     Mov FCB2Seg,CS
[2S1]  768: IF FILE EQ 1
[LY6]  769: ;Pufferdatei öffnen, um Rückfrage beim
[4L4]  77Ø: ;Überschreiben zu erhalten
[6W3]  771:     Mov DX,Offset TempName
[PH1]  772:     Xor CX,CX
[UM1]  773:     Mov AH,3Ch
[381]  774:     Int 21h
[VE2]  775:     Jnc DateiOffen
[A12]  776: TempFehler:
[557]  777: ;Pufferdatei konnte nicht geöffnet werden
[DB1]  778:     Mov BX,2
[AL3]  779:     Mov DX,Offset Tmp1Msg
[ZC2]  78Ø:     Mov CX,Tmp1Len
[NL1]  781:     Mov AH,4Øh
[381]  782:     Int 21h
[XX4]  783: ;Name der Pufferdatei ausgeben
[6W3]  784:     Mov DX,Offset TempName
[2G2]  785:     Mov CX,TempLen
[NL1]  786:     Mov AH,4Øh
[381]  787:     Int 21h
[YK3]  788:     Mov SI,Offset Tmp2Msg
[SB2]  789:     Mov DI,Tmp2Len
[VE2]  79Ø:     Jmp FehlerEnde
[TØ2]  791: DateiOffen:
[EL6]  792: ;Zur Kontrolle 1 Byte in Datei schreiben
[CH1]  793:     Mov BX,AX
[EB1]  794:     Mov CX,1
[NL1]  795:     Mov AH,4Øh
[381]  796:     Int 21h
[352]  797:     Jc TempFehler
[RS4]  798: ;Fehler bei voller Diskette
[UF1]  799:     Cmp AX,CX
[MF2]  8ØØ:     Jne TempFehler
[PA6]  8Ø1: ;Temporäre Datei wieder schließen
[YM1]  8Ø2:     Mov AH,3Eh
[381]  8Ø3:     Int 21h
[352]  8Ø4:     Jc TempFehler
[TUØ]  8Ø5: ENDIF
[4L7]  8Ø6: ;Tastaturinterrupts sichern und auf eigene
```

```
[ZP4]  8Ø7: ;Behandlungsroutine setzen
[9W1]  8Ø8:     Mov AX,3516h
[381]  8Ø9:     Int 21h
[WH4]  81Ø:     Mov Word Ptr AlterInt16,BX
[YR4]  811:     Mov Word Ptr AlterInt16+2,ES
[KW1]  812:     Mov AX,35Ø9h
[381]  813:     Int 21h
[5K4]  814:     Mov Word Ptr AlterIntØ9,BX
[7S4]  815:     Mov Word Ptr AlterIntØ9+2,ES
[8C1]  816:     Push CS
[G31]  817:     Pop ES
[Y44]  818:     Mov DX,Offset NeuerInt16
[8W1]  819:     Mov AX,2516h
[381]  82Ø:     Int 21h
[X44]  821:     Mov DX,Offset NeuerIntØ9
[HW1]  822:     Mov AX,25Ø9h
[381]  823:     Int 21h
[DZ3]  824: ;Unterprozeß aufrufen
[3F3]  825:     Mov BX,Offset EnvSeg
[YE2]  826:     Mov DX,NameOfs
[9W1]  827:     Mov AX,4BØØh
[381]  828:     Int 21h
[ØD6]  829: ;Fehlercode der Exec-Funktion sichern
[ZXØ]  83Ø:     Pushf
[PC1]  831:     Push AX
[6E7]  832: ;Alte Tastaturinterrupts wiederherstellen
[GC3]  833:     Lds DX,CS:AlterIntØ9
[HW1]  834:     Mov AX,25Ø9h
[381]  835:     Int 21h
[HC3]  836:     Lds DX,CS:AlterInt16
[8W1]  837:     Mov AX,2516h
[381]  838:     Int 21h
[8C1]  839:     Push CS
[E31]  84Ø:     Pop DS
[2S1]  841: IF FILE EQ 1
[Y95]  842: ;Temporäre Pufferdatei löschen
[6W3]  843:     Mov DX,Offset TempName
[RL1]  844:     Mov AH,41h
[381]  845:     Int 21h
[TUØ]  846: ENDIF
[ZD7]  847: ;Fehlercode der Exec-Funktion zurückholen
[R31]  848:     Pop AX
[LSØ]  849:     Popf
[8E2]  85Ø:     Jnc KeinFehler
[WF6]  851: ;Unterprozeß konnte nicht aufgerufen
[8X6]  852: ;werden, passende Fehlermeldung ausgeben
[BH3]  853:     Mov SI,Offset Err2Msg
[TB2]  854:     Mov DI,Err2Len
[N81]  855:     Cmp AL,2
[U22]  856:     Je ErrorExit
[XH3]  857:     Mov SI,Offset Err8Msg
[HC2]  858:     Mov DI,Err8Len
[V81]  859:     Cmp AL,8
[U22]  86Ø:     Je ErrorExit
[5W6]  861: ;Unbekannter Fehler, allgemeine Meldung
[XM3]  862:     Mov SI,Offset ErrXMsg
[HG2]  863:     Mov DI,ErrXLen
[9W1]  864: ErrorExit:
[N47]  865: ;Ersten Teil der Meldung über Fehlerkanal
[6H1]  866: ;ausgeben
[DB1]  867:     Mov BX,2
[CW3]  868:     Mov DX,Offset ErrorMsg
[9U2]  869:     Mov CX,ErrorLen
[NL1]  87Ø:     Mov AH,4Øh
[381]  871:     Int 21h
[VP5]  872: ;Name des Unterprozesses ausgeben
[YE2]  873:     Mov DX,NameOfs
[8E2]  874:     Mov CX,NameLen
[NL1]  875:     Mov AH,4Øh
[381]  876:     Int 21h
[VE2]  877:     Jmp FehlerEnde
[6Ø2]  878: KeinFehler:
[3X7]  879: ;Returncode des Unterprozesses zurückgeben
[XM1]  88Ø:     Mov AH,4Dh
[381]  881:     Int 21h
[VM1]  882:     Mov AH,4Ch
[381]  883:     Int 21h
[5H2]  884: Ende Label Byte
[TF1]  885: Code Ends
[SF1]  886:     End Begin
```

Listing 1. Das Assemblerprogramm »saaws.asm« erlaubt SAA-Blockbefehle bei Wordstar-kompatiblen Editoren

```
[8F5]    1: E9 CD Ø5 ØD ØA 53 41 41 57 53 2Ø 56 65 72 73 69
[6H5]    2: 6F 6E 2Ø 31 2E 31 2Ø 2D 2Ø 53 41 41 2D 42 6C 6F
[BH5]    3: 63 6B 62 65 66 65 68 6C 65 2Ø 6D 69 74 2Ø 57 6F
[EH5]    4: 72 64 73 74 61 72 2D 6B 6F 6D 7Ø 61 74 69 62 6C
[KH5]    5: 65 6E 2Ø 45 64 69 74 6F 72 65 6E ØD ØA 41 75 66
[SC5]    6: 72 75 66 3A 2Ø 53 41 41 57 53 2Ø 3C 5Ø 72 6F 67
[WM5]    7: 72 61 6D 6D 6E 61 6D 65 2E 7B 45 58 45 7C 43 4F
[KL5]    8: 4D 7D 3E 2Ø 3C 5Ø 72 6F 67 72 61 6D 6D 7Ø 61 72
[2D5]    9: 61 6D 65 74 65 72 3E ØD ØA 53 41 41 57 53 3A 2Ø
[WE5]   1Ø: 44 61 73 2Ø 5Ø 72 6F 67 72 61 6D 6D 2Ø AF AE 2Ø
[XC5]   11: 77 75 72 64 65 2Ø 6E 69 63 68 74 2Ø 67 65 66 75
[VK5]   12: 6E 64 65 6E ØD ØA AE 2Ø 69 73 74 2Ø 7A 75 2Ø 67
[YB5]   13: 72 6F E1 2Ø 66 81 72 2Ø 64 65 6E 2Ø 41 72 62 65
[5H5]   14: 69 74 73 73 7Ø 65 69 63 68 65 72 ØD ØA AE 2Ø 6B
[EF5]   15: 6F 6E 6E 74 65 2Ø 6E 69 63 68 74 2Ø 61 75 73 67
```

```
[3E5]   16: 65 66 81 68 72 74 2Ø 77 65 72 64 65 6E ØD ØA 53
[9D5]   17: 41 41 57 53 3A 2Ø 44 69 65 2Ø 74 65 6D 7Ø 6F 72
[P95]   18: 84 72 65 2Ø 5Ø 75 66 66 65 72 64 61 74 65 69 2Ø
[2M5]   19: AF AE 2Ø 6B 6F 6E 6E 74 65 2Ø 6E 69 63 68 74 2Ø
[NF5]   2Ø: 61 6E 67 65 6C 65 67 74 2Ø 77 65 72 64 65 6E ØD
[UB5]   21: ØA 9Ø 13 1F 6E Ø2 Ø1 ØØ Ø1 1E 7Ø Ø2 Ø1 ØØ ØØ 1E
[B75]   22: 72 Ø2 Ø4 ØØ ØØ 1F 7A Ø2 Ø8 ØØ ØØ 35 8A Ø2 ØA ØØ
[E85]   23: ØØ 19 9E Ø2 ØA ØØ ØØ 3Ø B2 Ø2 ØF ØØ ØØ ØØ AF ØØ
[ZE5]   24: AE ØØ ØØ 47 ØE 31 7B ØØ 2Ø ØØ ØØ 47 D6 FF ØØ 4D
[EB5]   25: 11 1Ø 79 2C 7D ØØ ØØ 47 ØØ 5Ø ØØ 47 ØE 31 7B ØØ
[AD5]   26: 2Ø ØØ D9 FF 2D ØØ 2Ø ØØ 7D ØØ ØØ 47 ØØ 5Ø 5Ø ØØ
[Y55]   27: 72 ØØ 6F ØØ 63 ØØ 65 ØØ 64 ØØ 75 ØØ 72 ØØ 65 ØØ
[Z95]   28: 2Ø ØØ 42 ØØ 65 ØØ 67 ØØ 69 ØØ 6E ØØ FC FF ØD ØØ
[HH5]   29: 45 ØØ 6E ØØ 64 ØØ 3B ØØ FE FF ØØ 4B FD FF ØØ 48
[K15]   3Ø: ØØ ØØ ØØ ØØ ØØ ØØ 5C ØØ ØØ ØØ 6C ØØ ØØ ØØ ØØ ØØ
[Ø35]   31: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 24 2E ØØ 48 5Ø
[2E5]   32: 4B 4D 47 4F 49 51 52 53 48 Ø1 5Ø Ø1 4B Ø1 4D Ø1
[N85]   33: 47 Ø1 4F Ø1 49 Ø1 51 Ø1 52 Ø1 53 Ø1 17 11 1A 2C
[E45]   34: ØØ 73 ØØ 74 ØØ 84 ØØ 76 ØØ 84 ØØ 76 ØØ Ø1 ØØ ØØ
[555]   35: ØØ ØØ ØØ ØØ 73 Ø1 74 Ø1 84 Ø1 76 Ø1 84 Ø1 76 Ø1
[195]   36: ØØ ØØ ØØ ØØ 5Ø 51 57 Ø6 33 CØ 8E CØ 26 8A 26 17
[XU5]   37: Ø4 F6 C4 28 75 16 F6 C4 Ø7 74 11 8C CF 8E C7 BF
[8R5]   38: EE Ø2 B9 ØA ØØ E4 6Ø FC F2 AE 74 Ø9 Ø7 5F 59 58
[WS5]   39: 2E FF 2E E3 Ø2 81 EF EF Ø2 F6 C4 Ø4 74 ØB 83 C7
[BN5]   4Ø: ØA F6 C4 Ø3 74 Ø3 83 C7 ØA D1 E7 26 8B 85 F8 Ø2
[7N5]   41: 23 CØ 74 22 BF 4Ø ØØ 8E C7 26 8B 3E 1C ØØ 8D 4D
[4H5]   42: E4 83 E1 1E 83 C1 1E 26 3B ØE 1A ØØ 74 Ø8 26 89
[GL5]   43: Ø5 26 89 ØE 1C ØØ E4 61 ØC 8Ø E6 61 EB Ø1 9Ø 24
[9L5]   44: 7F E6 61 BØ 61 E6 2Ø Ø7 5F 59 58 CF ØB 25 62 3Ø
[M75]   45: ØØ ØØ ØB 25 6B 25 11 1Ø 62 3Ø ØB 25 79 2C ØØ ØØ
[D75]   46: ØB 25 68 23 ØØ ØØ ØB 25 62 3Ø ØØ ØØ ØØ 4D ØB 25
[MB5]   47: 6B 25 ØB 25 69 17 ØB 25 69 17 11 1Ø 6B 25 ØB 25
[M85]   48: 75 16 ØB 25 75 16 11 1Ø 6B 25 ØB 25 77 11 24 ØØ
[NB5]   49: 2E ØØ ØD 1C 79 ØØ 11 1Ø 62 3Ø ØB 25 79 2C ØB 25
[195]   5Ø: 72 13 24 ØØ 2E ØØ ØD 1C ØB 25 68 23 ØØ ØØ ØØ ØØ
[U55]   51: ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ 57 2E 8B 3E
[8K5]   52: 1E Ø4 22 E4 74 2D 8Ø FC 1Ø 74 28 8Ø FC Ø1 74 Ø5
[MG5]   53: 8Ø FC 11 75 18 2E 83 3E 28 Ø4 ØØ 75 Ø4 23 FF 74
[WV5]   54: ØC 8B FC 36 8Ø 65 Ø6 BF B8 21 ØØ 5F CF 5F 2E FF
[XE5]   55: 2E E7 Ø2 2E A1 28 Ø4 23 CØ 74 ØC 48 2E A3 28 Ø4
[CK5]   56: 2E A1 26 Ø4 E9 A8 Ø1 23 FF 74 Ø3 E9 2C Ø1 32 E4
[VN5]   57: 9C 2E FF 1E E7 Ø2 2E 8Ø 3E 2A Ø4 ØØ 75 1E BF 42
[1K5]   58: Ø2 2E 83 3D ØØ 74 15 2E 3B Ø5 74 Ø5 83 C7 Ø6 EB
[5L5]   59: FØ 2E 8B 45 Ø2 2E 8B 7D Ø4 E9 FA ØØ 8Ø FC Ø1 75
[ZD5]   6Ø: Ø5 3D 1B Ø1 75 Ø3 E9 21 Ø1 2E A2 C1 Ø3 3D 53 Ø1
[4D5]   61: 75 1D 2E 8Ø 3E 2A Ø4 ØØ 74 B4 2E C6 Ø6 2A Ø4 ØØ
[HP5]   62: 2E C6 Ø6 2B Ø4 FF B8 FA Ø3 BF ØA ØØ E9 C7 ØØ 3D
[XE5]   63: ØØ Ø1 75 17 2E 8Ø 3E 2A Ø4 ØØ 74 92 2E C6 Ø6 2B
[YM5]   64: Ø4 FF B8 FA Ø3 BF Ø6 ØØ E9 AB ØØ 3D 52 Ø1 75 1A
[UK5]   65: 2E 8Ø 3E 2B Ø4 ØØ 75 Ø3 E9 73 FF 2E C6 Ø6 2A Ø4
[ØF5]   66: ØØ B8 ØE Ø4 BF Ø7 ØØ E9 8C ØØ 3D 74 Ø1 75 6E 51
[5V5]   67: 1E 33 CØ 8E D8 8B 3E 62 Ø4 81 E7 FF ØØ D1 E7 8B
[6H5]   68: 8D 5Ø Ø4 AØ 4A Ø4 86 C5 F6 E5 Ø2 C1 8Ø D4 ØØ D1
[1M5]   69: EØ Ø3 Ø6 4E Ø4 8B F8 B8 ØØ BØ 8Ø 3E 63 Ø4 B4 74
[ØT5]   7Ø: Ø2 B4 B8 8E D8 2A E9 B1 FF E8 1E Ø1 72 Ø6 22 ED
[WN5]   71: 75 F7 EB 27 22 ED 74 23 E8 ØF Ø1 72 F7 32 ED F7
[YN5]   72: D9 2E 89 ØE DA Ø3 1F 59 B8 DA Ø3 BF Ø4 ØØ 2E 8Ø
[LK5]   73: 3E 2A Ø4 ØØ 75 2Ø B8 D6 Ø3 EB 13 1F 59 B8 CØ Ø3
[7H5]   74: BF Ø3 ØØ 2E 8Ø 3E 2A Ø4 ØØ 75 ØB B8 BC Ø3 47 47
[ØM5]   75: 2E C6 Ø6 2A Ø4 FF 2E A3 1C Ø4 4F 2E 89 3E 1E Ø4
[LL5]   76: 2E 8B 3E 1C Ø4 2E 8B Ø5 47 47 3D ØØ FF 76 14 F7
[8M5]   77: DØ 2E A3 28 Ø4 2E 8B Ø5 2E A3 26 Ø4 2E FF ØE 1E
[YF5]   78: Ø4 47 47 2E 89 3E 1C Ø4 EB 45 2E 8Ø 3E 2A Ø4 ØØ
[6P5]   79: 74 3D 3D Ø9 ØF 74 ØA 3D ØØ ØF 75 ØD B8 EE Ø3 EB
[7T5]   8Ø: Ø3 B8 E2 Ø3 BF Ø6 ØØ EB AD 2E C6 Ø6 2A Ø4 ØØ BF
[TC5]   81: Ø4 ØØ 3D ØØ 53 74 Ø9 3C 2Ø 72 ØA 2E A3 CE Ø3 47
[WP5]   82: B8 C6 Ø3 EB 91 2E A3 D4 Ø3 4F B8 DØ Ø3 EB 87 33
[ZK5]   83: FF 2E 87 3E 22 Ø4 3D ØD 32 75 13 2E C7 Ø6 22 Ø4
[DA5]   84: 8Ø ØØ 2E C7 Ø6 24 Ø4 ØØ ØØ B8 ØØ Ø1 EB 3A 3D ØØ
[GH5]   85: 32 75 1C 23 FF 74 Ø9 2E C7 Ø6 22 Ø4 ØØ ØØ EB E9
[GL5]   86: 2E 8B 3E 24 Ø4 23 FF 74 EØ B8 8Ø ØØ E9 47 FF 23
[EK5]   87: FF 74 15 81 FF 42 Ø2 73 ØF 2E 89 Ø5 47 47 2E 89
[8P5]   88: 3E 22 Ø4 2E FF Ø6 24 Ø4 5F CF 8A Ø5 47 47 FE C1
[GK5]   89: FE CD 3C 5F 74 32 3C 3Ø 72 2C 3C 39 76 2A 3C 41
[SF5]   9Ø: 72 24 3C 5A 76 22 3C 61 72 1C 3C 7A 76 1A 3C 8Ø
[KH5]   91: 72 14 3C 9A 76 12 3C AØ 72 ØC 3C A5 76 ØA 3C EØ
[VM5]   92: 72 Ø4 3C EB 76 Ø2 F8 C3 F9 C3 BE Ø3 Ø1 BF 86 ØØ
[GU5]   93: BB Ø2 ØØ 8B D6 8B CF B4 4Ø CD 21 B8 FF 4C CD 21
[2T5]   94: BC ØØ ØA BB AØ ØØ B4 4A CD 21 8A ØE 8Ø ØØ 32 ED
[FR5]   95: BF 81 ØØ BØ 2Ø F3 AE 75 Ø2 E3 CF 4F 41 8B F7 F2
[US5]   96: AE 38 45 FF 75 Ø2 4F 41 51 8B CF 2B CE 89 ØE E1
[US5]   97: Ø2 8D 7C FE 89 3E DF Ø2 F3 A4 32 CØ AA 8C ØE D4
[HL5]   98: Ø2 89 3E D2 Ø2 58 AA A1 2C ØØ A3 DØ Ø2 8C ØE D8
[TN5]   99: Ø2 8C ØE DC Ø2 BA EB Ø2 33 C9 B4 3C CD 21 73 2Ø
[3R5]  1ØØ: BB Ø2 ØØ BA FF Ø1 B9 22 ØØ B4 4Ø CD 21 BA EB Ø2
[BM5]  1Ø1: B9 Ø2 ØØ B4 4Ø CD 21 BE 21 Ø2 BF 2Ø ØØ E9 7Ø FF
[VP5]  1Ø2: 8B D8 B9 Ø1 ØØ B4 4Ø CD 21 72 D5 3B C1 75 D1 B4
[ZM5]  1Ø3: 3E CD 21 72 CB B8 16 35 CD 21 89 1E E7 Ø2 8C Ø6
[VL5]  1Ø4: E9 Ø2 B8 Ø9 35 CD 21 89 1E E3 Ø2 8C Ø6 E5 Ø2 ØE
[EL5]  1Ø5: Ø7 BA 2C Ø4 B8 16 25 CD 21 BA 34 Ø3 B8 Ø9 25 CD
[1P5]  1Ø6: 21 BB DØ Ø2 8B 16 DF Ø2 B8 ØØ 4B CD 21 9C 5Ø 2E
[KN5]  1Ø7: C5 16 E3 Ø2 B8 Ø9 25 CD 21 2E C5 16 E7 Ø2 B8 16
[4N5]  1Ø8: 25 CD 21 ØE 1F BA EB Ø2 B4 41 CD 21 58 9D 73 36
[WK5]  1Ø9: BE 9E Ø1 BF 18 ØØ 3C Ø2 74 1Ø BE B6 Ø1 BF 27 ØØ
[2L5]  11Ø: 3C Ø8 74 Ø6 BE DD Ø1 BF 22 ØØ BB Ø2 ØØ BA 89 Ø1
[GL5]  111: B9 15 ØØ B4 4Ø CD 21 8B 16 DF Ø2 8B ØE E1 Ø2 B4
[Ø35]  112: 4Ø CD 21 E9 CA FE B4 4D CD 21 B4 4C CD 21
```

Listing 2. Das ausführbare Programm »saaws.com« zum Eingeben mit SCheck

Die Speicherverwaltung unter MS-DOS

Dieser Artikel widmet sich der Speicherverwaltung von PCs. Sie erfahren wichtige Grundlagen über die Speicheraufteilung eines MS-DOS-Computers und über die diversen »Tricks«, mit denen sich der Programmierer über die 640-KByte-Grenze hinwegsetzen kann.

Wer über MS-DOS 4.0 verfügt, kennt vielleicht den neuen Befehl »mem«. Die Kommandos »mem /program« und »mem /debug« geben einen ziemlich differenzierten Überblick über die aktuelle Speicherbelegung Ihres PC. Dahinter steckt natürlich eine Speicherorganisation, die zum Teil charakteristisch für das Konzept eines IBM-kompatiblen PC, zum Teil aber auch charakteristisch für MS-DOS ist. Ein Teil der hier wiedergegebenen Informationen ist nicht in den technischen Unterlagen von PCs oder von MS-DOS dokumentiert, sondern beruht auf Untersuchungen, die von MS-DOS-Experten angestellt wurden.

Mit der Präsentation des ersten PC fällte IBM zwei heute noch wirksame Grundsatzentscheidungen: die Entscheidung für Intel-Prozessoren und die Entscheidung für MS-DOS. MS-DOS ist ein Betriebssystem für den Intel 8088/8086-Mikroprozessor, woraus sich einige unangenehme Einschränkungen ergeben. Eine der wichtigsten ist: Der Prozessor kann nur einen Arbeitsspeicher von maximal 1 MByte adressieren. Zwar besteht diese Schranke bei den Nachfolgern 80286, 80386 und 80486 nicht mehr, jedoch zwingt MS-DOS diese Prozessoren in einen Betriebszustand (»Real Mode«), in dem sie einen 8086 simulieren und daher ihre erweiterten Fähigkeiten nicht einsetzen können. Also gilt auch für die Nachfolger des 8086-Mikroprozessors: Unter MS-DOS ist der Arbeitsspeicher auf 1 MByte beschränkt. Hiervon sind aber nur 640 KByte als freier Arbeitsspeicher nutzbar, denn die Adressen oberhalb der 640-KByte-Grenze sind grundsätzlich für Bildschirmspeicher und andere Hardware-Einheiten reserviert.

Der Speicherbereich von 0 beginnend bis 1 MByte trägt den Namen Basisspeicher. Als »Extended Memory« bezeichnet man verfügbaren Speicher, der oberhalb der 1-MByte-Grenze eingerichtet ist. Dieser Speicher ist unter MS-DOS nicht unmittelbar adressierbar. Besitzt der PC tatsächlich einen solchen Speicher, so kann er nur im sogenannten »Protected Mode« der 80286-, 80386- und 80486-Prozessoren genutzt werden. Natürlich ist vorausgesetzt, daß der PC kein 8086- oder 8088-System ist.

Die 640-KByte-Grenze ist für viele Anwendungen zu eng. Daher haben sich vor mehreren Jahren die Firmen Lotus, Intel und Microsoft (diese Gruppe gab sich den Namen »LIM«) auf ein Konzept der Speichererweiterung geeinigt, das heute zunehmend von Software-Herstellern akzeptiert wird. Es trägt den Namen EMS (»Expanded Memory Specifications«). Das Prinzip ist einfach: Man blendet 16-KByte-Blöcke des Erweiterungsspeichers, jetzt »Expanded Memory« genannt, in den Adreßbereich des 8086-Prozessors ein. Sie sehen also sozusagen durch ein Fenster im Bereich bis 1 MByte auf ein Stück des Erweiterungsspeichers. Die Verwaltung des Erweiterungsspeichers weicht natürlich von den üblichen MS-DOS-Methoden ab. Daher kann nicht jedes Programm etwas mit diesem Speicher anfangen. Es muß schon entsprechend programmiert sein. MS-DOS unterstützt ab der Version 4.0 dieses Konzept offiziell.

SEGM OFFS	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
02B8:0048	00	00	5E	0C	04	80	5D	18	63	18	4E	55	4C	20	20	20
											N	U	L			
02B8:0058	20	20	00	13	88	5B	96	B8	02	FF	FF	FF	FF	0F	00	00

Bild 1. Speicherauszug mit dem Kopf des Nul-Treibers

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Wert:	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Hex:	8				0				0				4			

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Wert:	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Hex:	8				0				0				4			

Bild 2. Beispiel eines Attributbytes

Die Speicheraufteilung variiert von Installation zu Installation

Die Fragezeichen in Tabelle 1 deuten an, daß die Adressen nicht festgelegt sind und von System zu System variieren können. Das hat einen einfachen Grund. Bei der Installation des Rechners legen Sie in der Datei »config.sys« fest, wie viele Puffer (jeweils zur Aufnahme eines Disk-Sektors) reserviert, welche Treiber installiert werden sollen und so weiter. Sie bestimmen auch, welche Programme speicherresident sind. Daher hängen die Adressen von Ihrer individuellen Installation ab.

Sie sehen, daß der freie Arbeitsspeicher, also der Bereich bis zur Adresse A000:0000hex, tatsächlich nicht völlig frei ist, sondern wesentliche Teile des Betriebssystems aufnimmt. Der für ein Anwenderprogramm verfügbare Platz hängt davon ab, wie groß Sie verschiedene Pufferbereiche gewählt haben und wie umfangreich speicherresident installierte Programme sind.

MS-DOS muß diese Adressen verwalten. Dazu organisiert das Betriebssystem die Puffer, die Treiber und die speicherresidenten Programme in verketteten Listen. Wenn Sie wissen wollen, was wo gespeichert ist, müssen Sie den Anfang der jeweiligen Liste finden und sich bis zum Ende durcharbeiten. Für die Verkettung selbst setzt MS-DOS unterschiedliche Methoden ein. Treiber werden über einen Zeiger im Kopf des Treibers verkettet, residente Programme über sogenannte »Memory-Control-Blocks« (MCB). Ein Beispiel für Treiber und MCBs wird Ihnen verdeutlichen, wie das System arbeitet.

Die Verwaltung des Speichers wird auch in den technischen Handbüchern zu MS-DOS nicht vollständig beschrieben. Dokumentiert ist, wie man von einem Treiber oder einem Memory-Control-Block zum nächsten kommt, nicht jedoch, wie man den Anfang der Treiber- und der Control-Block-Liste findet. MS-DOS verwendet dazu interne Strukturen, zum Beispiel den »DOS-Informationsblock« (DIB). Das ist ein Speicherbereich, in dem wichtige Systemadressen stehen. Seinen Aufbau finden Sie in Tabelle 2.

Liste der Gerätetreiber

Nr.	Seg:Offs Treiber	Attribut	Routinen Offset Strategie	Interrupt	Name/Einheit
1	02B8:0048	8004	185D	1863	NUL
2	0C5E:0000	4040	00DA	00E5	1 Laufwerk
3	0B44:0000	C053	00C1	00CC	CON
4	0070:016E	8013	062A	0635	CON
5	0070:0180	8000	062A	063B	AUX
6	0070:0192	A040	062A	0656	PRN
7	0070:01A4	8008	062A	067C	CLOCK$
8	0070:01B6	0842	062A	0682	3 Laufwerke
9	0070 01CA	8000	062A	063B	COM1
10	0070:01DC	A040	062A	065C	LPT1
11	0070:01EE	A040	062A	0664	LPT2
12	0070:0200	A040	062A	066C	LPT3
13	0070:0212	8000	062A	0641	COM2
14	0070:0224	8000	062A	0647	COM3
15	0070:0236	8000	062A	064D	COM4

Bild 3. Beispiel für eine Treiberkette

Adresse Segment:Offset	Verwendung
0000:0000	Interrupt-Vektortabelle
0040:0000	ROM-Kommunikationsbereich
0050:0000	DOS-Kommunikationsbereich
????:0000	»io.sys« beziehungsweise »ibmio.com«
????:0000	»msdos.sys« beziehungsweise »ibmdos.com«
????:0000	MS-DOS-Puffer, Kontrollbereiche, Gerätetreiber
????:0000	residenter Teil des »command.com«
????:0000	Anwenderprogramm (Com- oder Exe-Datei)
????:0000	Stapelbereich für Com-Dateien
????:0000	transienter Teil des »command.com«
A000:0000	reserviert für System

Tabelle 1. Die Speicheraufteilung unter MS-DOS

Diese Struktur ist, wie gesagt, nicht dokumentiert, und es besteht daher auch keine Garantie dafür, daß sie in allen MS-DOS-Versionen so erhalten bleibt. Zum Beispiel scheint der Zugang zu MS-DOS-Puffern in der Version 4.0 verändert worden zu sein, so daß dort die Tabelle 2 nur begrenzt anwendbar ist.

Wichtig ist die Information über den ersten MCB, denn über diesen können Sie die Kette der resident installierten Programme verfolgen. An den DIB schließt sich der erste Treiber in der Kette der Treiber an, der Nul-Treiber. Daher wissen Sie auch, wo die Treiberkette startet.

Geträtetreiber dienen der Peripheriesteuerung. MS-DOS unterscheidet zwei Typen: Zeichentreiber und Blocktreiber. Zeichentreiber bearbeiten Zeicheneingabe und Zeichenausgabe. Der Standardtreiber »con« (für Konsole) ist zum Beispiel für die Tastatureingabe und Bildschirmausgabe zuständig. Blocktreiber bedienen Medien wie Disketten, Platten, RAM-Disks und Streamer. Sie bearbeiten ganze Datenblöcke auf einmal, in der Regel den Inhalt eines Sektors. Näheres erfahren Sie in den technischen Handbüchern zu MS-DOS.

MS-DOS richtet beim Systemstart fünf Treiber standardmäßig ein. Sie tragen die Namen »nul«, »con«, »aux«, »prn« und »clock$«. Den Start der Treiberkette bildet immer der Nul-Treiber. Zusätzliche Gerätetreiber oder Treiber, die vorhandene ersetzen sollen, werden über einen Befehl der Art »device = ...« in der Datei »config.sys« installiert. Zum Beispiel wird der Con-Treiber über den Befehl

```
DEVICE=ANSI.SYS
```

durch den Ansi-Treiber ersetzt.

Jeder Treiber besitzt einen Treiberkopf (Tabelle 3). Steht in den Byte 0 bis 1 der Wert -1 (FFFFhex), so ist der Treiber der letzte in der verketteten Liste. Den Typ des Treibers können Sie aus dem Attribut ersehen. Tabelle 4 gibt Auskunft über die Bedeutung der einzelnen Bits des Attributbytes (hier für MS-DOS 4.0). Bei Blocktreibern finden Sie in Byte 10 normalerweise die Anzahl der unterstützten Geräte.

Die Analyse der Treiberkette macht wenig Mühe. Den Anfang der Treiberkette finden wir über die MS-DOS-Funktion 52hex (siehe Tabelle 2). Jeder Treiberkopf enthält einen Zeiger auf den nächsten Treiberkopf, so daß die Kette leicht zu verfolgen ist.

Betrachten wir ein Beispiel (Bild 1). Die Adresse 02B8:0048hex wird als Beginn der Treiberkette ausgewiesen. Wir erwarten dort den Treiberkopf des Nul-Treibers. Bild 1 zeigt einen Speicherauszug. Die ersten vier Byte enthalten den Zeiger auf den nächsten Treiber, hier (gemäß Intel-Konvention von hinten nach vorne zu lesen) 0C5E:0000 hex. Das Attribut des Nul-Treibers finden Sie in den Byte 4 und 5, nämlich 8004hex (wieder von hinten nach vorne). Interpretieren Sie das Attribut mit Hilfe von Tabelle 4 oder Bild 2. Sie können den Treiber anhand des Zustands von Bit 2 als Nul-Treiber identifizieren. Bit 15 klas-

```
                                  Platz für 6 Paragraphen
    1CB4:0000  4D BC 1C 06 00 00 BF 0A-03 A5 A5 B8 03 03 89 44   M..............D

->  1CB4:0010  43 4F 4D 53 50 45 43 3D-63 3A 5C 64 6F 73 5C 63   COMSPEC=c:\dos\c
    1CB4:0020  6F 6D 6D 61 6E 64 2E 63-6F 6D 00 50 41 54 48 3D   ommand.com.PATH=
    1CB4:0030  43 3A 5C 44 4F 53 3B 43-3A 5C 53 59 53 54 45 4D   C:\DOS;C:\SYSTEM
    1CB4:0040  5C 55 54 49 4C 00 00 01-00 43 3A 5C 53 59 53 54   \UTIL....C:\SYST
    1CB4:0050  45 4D 5C 55 54 49 4C 5C-53 41 56 45 41 4C 4C 2E   EM\UTIL\SAVEALL.
    1CB4:0060  43 4F 4D 00 0E 00 EB D3-E8 98 F9 FE C7 EB CE E8   COM.............

    1CB4:0070  4D BC 1C 65 00 0D E8 8A-53 41 56 45 41 4C 4C 00   M..e....SAVEALL.

->  1CB4:0080  CD 20 21 1D 00 9A F0 FE-1D F0 47 01 AC 16 56 01   . !.......G...V.
    1CB4:0090  AC 16 BF 05 AC 16 AC 16-01 01 01 00 02 FF FF FF   ................

    1CB4:00A0  FF FF FF FF FF FF FF FF-FF FF FF FF B5 1C E6 FF   ................

    1CB4:00B0  BC 1C 14 00 18 00 BC 1C-FF FF FF FF 00 00 00 00   ................
```

Bild 4. Beispiel eines Memory-Control-Blocks

Byte	Verwendung
00 – 03	Zeiger auf ersten Memory-Control-Block
04 – 07	Zeiger auf ersten Disk-Parameter-Block (Byte 04 wird durch Interrupt 21hex, Funktion 52hex adressiert)
08 – 11	Zeiger auf letzen DOS-Puffer
12 – 15	Zeiger auf Uhrtreiber »clock$«
16 – 19	Zeiger auf Konsolentreiber »con«
20 – 21	maximale Sektorlänge (beträgt normalerweise 512 Byte)
22 – 25	Zeiger auf ersten MS-DOS-Puffer
26 – 29	Zeiger auf Pfad-Tabelle
30 – 33	Zeiger auf System-File-Tabelle
34 – 35	reserviert
ab 36	Zeigerkette von Gerätetreibern mit Start beim Nul-Treiber

Tabelle 2. Der MS-DOS-Informationsblock (DIB)

```
ADR   TYP       PSP-ADR ENV-ADR Länge Name

0B42  BIOS      0008            0647
118A  frei      0000            0003
118E  Umgebung  1616            0008 \DOS\MOUSE.COM
1197  Programm  1198            1195 SHARE
1322  Programm  1323    14B9    0177 COMMAND (22 2E)
149A  Umgebung  1323            0010 Umgebung von COMMAND
14AB  Programm  14AC    118F    0169 KEYB (09)
1615  Programm  1616    118F    026B MOUSE (10 22 74)
1881  Umgebung  188B            0008 C:\SYSTEM\UTIL\DOSEDIT.COM
188A  Programm  188B    1882    00A2 DOSEDIT
192D  Umgebung  1937            0008 C:\SYSTEM\UTIL\SAVEALL.COM
1936  Programm  1937    192E    0065 SAVEALL (7F)
199C  Umgebung  19A7            0009 C:\PASCAL\MCB.EXE
19A6  Programm  19A7    199D    8619 MCB

Gesamtkapazität: 654336 Byte
Restkapazität  : 549328 Byte
Größter freier Speicherblock: 589280 Byte
```

Bild 5. Beispiel für eine MCB-Liste

sifiziert ihn als Zeichentreiber. Außerdem finden Sie in Bild 1 den Namen des Treibers in den folgenden acht Bytes ab Byte 10. Wenn das Attribut ausgewertet wird, liefert das zum Beispiel für den Treiber 3 in Bild 3 folgende Information:

```
Der Treiber
- ist ein Zeichentreiber
- unterstützt »output until busy«
- ist Standardausgabetreiber
- ist Standardeingabetreiber
- verarbeitet Ioctl-Control-Strings
- unterstützt generische Ioctl-Funktionsaufrufe
```

Es handelt sich hier übrigens um den Ansi-Treiber. Er wurde in der Kette vor dem von MS-DOS eingerichteten Standardtreiber für »con« eingegliedert und setzt daher diesen außer Kraft.

Die Unit »Memory« (Listing 1) stellt eine ganze Reihe nützlicher Funktionen für die Speicherverwaltung zur Verfügung. Unter anderem »FirstDriverAdr« und »NextDriverAdr«, die einen Zeiger für den ersten und jeweils nächsten Treiberkopf liefern. Bemerkenswert sind auch die Hilfsfunktionen »PtrSeg« und »PtrOfs«, die den Segment- beziehungsweise Offset-Anteil eines Zeigers als Datenwort isolieren. Sie werden im weiteren Verlauf dieses Artikels noch auf eine Reihe weiterer, wichtiger Funktionen stoßen, die allesamt in »Memory« eingebunden sind.

Das Laden von Programmen, das Ausführen von Stapeldateien, die Verwaltung von speicherresidenten Programmen und so weiter setzt voraus, daß für die entsprechenden Zwecke Speicherplatz reserviert wird. Der Programmierer braucht sich darum nicht zu kümmern, denn die Exec-Funktion von MS-DOS übernimmt diese Aufgabe. Das Betriebssystem benutzt hierfür Memory-Control-Blocks (MCBs). Ein MCB ist ein 16-Byte-Feld, das dem reservierten Speicherbereich unmittelbar vorangeht. MCBs beginnen grundsätzlich an einer Paragraphengrenze. Ein Paragraph ist ein Block von 16 Byte, der an einer durch 16 teilbaren Speicheradresse beginnt. Diese Adresse läßt sich in der Intel-Konvention immer in der Form ????:0000hex angeben, wird also durch eine Segmentadresse eindeutig beschrieben. Tabelle 5 erläutert den Aufbau eines MCB.

Die Zeichen »M« und »Z« sind offenbar die Initialen von Mark Zbikowski, einem der MS-DOS-Entwickler, der sich auch im Exe-Datei-Header verewigt hat. PSP steht für »Programm-Segment-Präfix«, einen 256-Byte-Bereich, der jedem geladenen Programm unmittelbar vorangeht. Wenn ein MCB ein solches Programm verwaltet, so wird über Byte 1 und 2 die Segmentadresse des PSP angegeben. Beim Laden eines Programms richtet der Kommandoprozessor eine Kopie des MS-DOS-Umgebungsbereichs für das Programm ein. Auch für diese Kopie wird über einen MCB ein Speicherbereich reserviert. Die PSP-Adresse dieses MCB zeigt dann (in der Regel) auf das PSP des zugeordneten Programms. Die Verwendung von Byte 8 bis 15 ist nicht offiziell geregelt. Wenn ein MCB ein Programm kontrolliert, werden Sie (MS-DOS ab 3.3) dort in der Regel den Programmnamen finden.

Die MCBs enthalten keine Zeiger auf den nächsten MCB. Jedoch ist die Position des nächsten MCB über die Länge des reservierten Speicherbereichs zu berechnen. Der nächste MCB schließt sich nämlich unmittelbar an den vorangehenden Speicherblock an. Im Speicher entstehen also keine Lücken. Bild 4 zeigt ein Beispiel.

An der Adresse 1CB4:0000hex befindet sich der MCB für den Umgebungsbereich des Programms »saveall«. Es schließt sich der MCB für das Programm selbst an. Die Beziehung zwischen beiden wird in Bild 4 durch Pfeile erläutert.

Analysieren wir das Beispiel. Der MCB ab Adresse 1CB4:0000hex beginnt mit dem ASCII-Wert 4Dhex, also dem Zeichen »M«. Es folgt daher ein weiterer MCB. Die Byte 3 und 4 des MCB enthalten 06hex und 00hex, also die Zahl 6 (gemäß Intel-Konvention kommt das niederwertige Byte zuerst). Der MCB verwaltet also die auf ihn folgenden sechs Paragraphen (in Bild 4 die folgenden sechs

```
MCB von »autoexec.bat« (MS-DOS 4.0)

4D 00 00 03 00 05 01 36-81 26 F4 05 FF 00 36 83   M......6.&....6.

Speicherblock von »autoexec.bat«

00 01 00 00 00 00 00 AE 00 00 00 FF FF FF FF FF   ................
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 43   ...............C
3A 5C 41 55 54 4F 45 58 45 43 2E 42 41 54 00 0D   :\AUTOEXEC.BAT..
```

Bild 6. Beispiel einer Stapeldatei unter MS-DOS 4.0

```
MCB von »autoexec.bat« (MS-DOS 3.2)

4D 00 00 03 00 CD 21 86 E0 3D 14 03 72 05 3D 14   M....=!åα=..r.=.

Speicherblock von »autoexec.bat«

00 C3 01 00 00 FF FF FF FF FF FF FF FF FF FF FF   .├..............
FF FF FF FF FF FF FF FF FF 43 3A 5C 41 55 54 4F   .........C:\AUTO
45 58 45 43 2E 42 41 54 00 .....................  EXEC.BAT........
```

Bild 7. Beispiel einer Stapeldatei bei MS-DOS 3.2

Zeilen). Der nächste MCB beginnt direkt dahinter, also an der Adresse 1CB4:0070hex, was identisch mit 1CBB:0000hex ist. Er gehört zu dem Programm »saveall«. Der PSP-Zeiger zeigt auf die Adresse 1CBC:0000hex, also auf den direkt folgenden Paragraphen.

Wir analysieren in dem Programm »mcbliste.pas« (Listing 2) die Kette der MCBs und versuchen herauszufinden, von welchem Typ die kontrollierten Speicherblöcke sind. Bild 5 zeigt ein Beispiel. Ab MS-DOS 4.0 können Sie hierfür das neue Utility »mem« einsetzen, das Sie mit »mem /program« aufrufen. Sie erhalten dort allerdings keine Auskunft über die PSP- beziehungsweise Umgebungsadresse, die man aus dem MCB beziehungsweise PSP ermitteln kann.

Betrachten Sie das speicherresidente Programm »saveall« (Bild 5). Die Segmentadresse des zugehörigen Umgebungsbereichs ist 192Ehex. Der zum Umgebungsbereich gehörende MCB liegt genau einen Paragraph davor, hat also die Adresse 192Dhex. Die Länge des Umgebungsbereichs ist 8 (Paragraphen). Daher beginnt der nächste MCB bei 192Dhex + 1hex + 8hex, also bei 1936hex. Dort finden Sie den MCB für »saveall«. Das Programm selbst, genauer sein PSP, beginnt einen Paragraph dahinter, also bei 1937hex. In Byte 2Chex und 2Dhex des PSP finden Sie die Segmentadresse des zugeordneten Umgebunsgbereichs. Das ist hier 192Ehex und stimmt mit dem vorher ermittelten Wert überein.

Der MCB an der Stelle 118Ahex war, wie eine direkte Analyse des Arbeitsspeichers zeigte, der »autoexec.bat« zugeordnet, wurde aber nach Beendigung der Stapeldatei von MS-DOS wieder freigegeben. Die Restkapazität ist die Summe der Kapazitäten aller freien Speicherblöcke und dem von dem Programm »mcbliste.pas« selbst in Anspruch genommenen Speicherplatz. Das MCB-Programm ist so compiliert, daß es den Rest des Arbeitsspeichers belegt. Nach der Beendigung von »mcbliste.pas« ist dieser Platz wieder frei. Da ein Programm nicht zerstückelt werden kann, ist die Maximalkapazität für ein eventuell noch zu ladendes Programm die Größe des größten freien Speicherblocks.

Die Nummern in Klammern hinter einigen Programmnamen sind die Nummern von Interrupts, die sich auf diese Programme beziehen. Sie können diese folgendermaßen ermitteln: Vergleichen Sie die Segmentadressen der Interrupts gemäß der Interrupt-Vektortabelle mit den PSP-Adressen der Programme. Stimmen sie überein, so hat das Programm bei seiner Installation die betreffenden Interrupts neu eingerichtet oder vorhandene umgeleitet. Wir gehen darauf weiter unten noch einmal ein.

Bei der Reservierung von Speicher wird nicht mitgeteilt, welchem Zweck sie dient. Es gibt auch keine dokumentierten Regeln über die Identifikation der Speicherblöcke. Man ist also auf eine Analyse von Beispielen angewiesen.

Byte	Verwendung
00 – 03	Zeiger auf nächsten Treiber (Segment:Offset)
04 – 05	Attribut
06 – 07	Zeiger auf Strategieroutine (Offset)
08 – 09	Zeiger auf Interrupt-Routine (Offset)
10 – 17	Name oder Kennzeichnung des Treibers

Tabelle 3. Aufbau eines Treiberkopfes

MS-DOS-Umgebungsbereiche können anhand der ersten Eintragung im Speicherblock identifiziert werden

Der erste MCB kontrolliert alle über »config.sys« installierten Gerätetreiber und sonstigen Einstellungen. Er ist also dem BIOS zuzurechnen. Bis MS-DOS 3.3 verwaltet der erste MCB die Treiber direkt. Ab MS-DOS 4.0 wird für die Speicherverwaltung innerhalb dieses Speicherblocks eine ähnliche Struktur verwendet, wie sie die MCBs selbst darstellen.

Ein MCB, dessen PSP-Adresse Null ist, gehört zu einem ehemals belegten Speicherblock, der inzwischen von MS-DOS wieder freigegeben wurde. Er wird gemäß der Belegungsstrategie bei der nächsten sich bietenden Gelegenheit wieder verwendet.

Umgebungsbereiche lassen sich durch ihre erste Eintragung identifizieren. Sie beginnen stets mit »comspec = « oder mit »path = «. Wir benutzen das in der Funktion »Test_Env«. Sie liefert den Wert »true« (wahr), wenn der Speicherblock mit »comspec = « oder »path = « beginnt.

Ist ein Speicherblock als Umgebungsbereich identifiziert, so suchen Sie den Namen des Programms, zu dem der Umgebungsbereich gehört. Jede Eintragung im Umgebungsbereich wird durch 00hex abgeschlossen. Der gesamte Umgebungsbereich wird darüber hinaus mit 00hex beendet. Am Schluß finden Sie also die Folge »00 00«. Der Speicherblock eines Umgebungsbereichs enthält (in der Regel) daran anschließend die Folge »01 00«. Insgesamt endet er also mit »00 00 01 00«. Hier schließt sich ab MS-DOS 3.0 Pfad und Name des Programms an, zu dem der Umgebungsbereich gehört. Bild 4 zeigt im oberen Teil ein Beispiel dazu.

Beim Umgebungsbereich des Originalkommandoprozessors sieht die Sache anders aus. Hier wird an den Umgebungsbereich nicht der Name des Kommandoprozessors angehängt. In unserem Programm identifizieren wir den Originalkommandoprozessor über diese Eigenschaft.

Ein Programm kann ein anderes Programm aufrufen. Zum Beispiel ruft normalerweise der Kommandoprozessor ein Anwendungsprogramm auf. Man nennt dann das aufru-

Bit	Bedeutung bei Bitwert 1, falls nicht anders angegeben
15	Bitwert 0: Blocktreiber, Bitwert 1: Zeichentreiber
14	Treiber unterstützt Ioctl-Funktionen (DOS-Funktion 44hex)
13	Blocktreiber: Treiber bedient Medium mit Nicht-IBM-Format Zeichentreiber: Treiber unterstützt »output until busy« nicht
12	reserviert
11	Treiber unterstützt Wechselmedium (zum Beispiel Diskette)
10 – 7	reserviert
6	Treiber unterstützt generische Ioctl-Funktionen
5 – 4	reserviert
3	Uhr-Treiber
2	Nul-Treiber
1	Zeichentreiber: Treiber ist aktueller Standardausgabetreiber Blocktreiber: Treiber unterstützt 32-Bit-Sektornummern
0	Zeichentreiber: Treiber ist aktueller Standardeingabetreiber

Tabelle 4. Belegung des Attributbytes im Treiberkopf

fende Programm Elternprogramm, das aufgerufene Kindprogramm. Unter MS-DOS wird ein Programm mit Hilfe der Exec-Funktion (MS-DOS-Funktion 4Bhex) aktiviert. Diese Funktion arbeitet mit einem Parameterblock, der auch spezifiziert, woher der Umgebungsbereich für das Kindprogramm stammen soll. Normalerweise wird eine Kopie des Originalumgebungsbereichs angelegt. Es kann aber auch ein anderer Umgebungsbereich explizit angegeben werden. Daher ist es möglich, daß verschiedene Programme mit demselben Umgebungsbereich arbeiten. Bei MS-DOS 4.0 ist das zum Beispiel bei den über »install = « installierten Programmen so.

Der Name des zu einem Umgebungsbereich gehörenden Programms wird in der Funktion »EnvirName« bestimmt. Die Funktion »Scan_Name« trennt den eigentlichen Namen vom eventuell vorangestellten Pfad.

Schwieriger ist die Identifikation von Programmen. Betrachten Sie dazu wieder Bild 4. Wir arbeiten mit zwei Tests. Der erste prüft, ob die PSP-Adresse im MCB mit der Segmentadresse des Speicherblocks übereinstimmt, auf den sich der MCB bezieht. Es wird also geprüft, ob die PSP-Adresse um genau 1 größer ist als die Segmentadresse des MCB. In Bild 4 beginnt der MCB bei 1CB4:0070hex oder, was das gleiche ist, bei 1CBB:0000hex. Der PSP-Zeiger enthält dort die Segmentadresse 1CBChex, zeigt also auf den unmittelbar folgenden Paragraphen.

Das ist allerdings noch kein vollständig verläßliches Indiz. Als zweites prüfen Sie, ob die beiden ersten Bytes des Speicherblocks CDhex / 20hex oder CDhex / 27hex sind. Das einem Programm vorgelagerte PSP beginnt nämlich in der Regel mit dem Code des Befehls »int 20h«, also CDhex (für »int«) und 20hex. In seltenen Fällen steht statt dessen dort »int 27h« mit dem Code CDhex 27hex. Fällt dieser Test positiv aus, so liefert die Funktion »Test_PSP« den Wert »true«. Diese Methode ist nicht absolut verläßlich. Ein zu einem Programm gehörender Speicherblock kann zum Beispiel auch mit einem Overlay geladen sein. In diesem Fall kann der PSP überschrieben werden.

Wurde ein MCB als zu einem Programm gehörig erkannt, bestimmen Sie mit der Funktion »MCBName« den Namen des Programms. Sie gehen dabei folgendermaßen vor: In den Byte 8 bis 15 des MCB findet man oft den Programmnamen. Falls er kürzer als der verfügbare Platz ist, wird er mit 00hex abgeschlossen. Es erfolgt nur ein Plausibilitätstest. Dabei wird geprüft, ob dort nur Zeichen stehen, die in einem Programmnamen vorkommen können (Funktion »Ist_Name«). Fällt der Test positiv aus, können Sie sich damit begnügen.

Fällt der Test nicht positiv aus, setzen Sie den Namen zunächst auf einen Leer-String. Wenn alle MCBs analysiert sind, versuchen Sie in einem zweiten Anlauf, den Namen zu bestimmen, diesmal über die Zuordnung zu einem Umgebungsbereich. Die Bytes 2Chex und 2Dhex des PSP eines Programms enthalten, wie bereits erwähnt, einen Zeiger auf den zugeordneten Umgebungsbereich. Im Beispiel von Bild 4 sind es die Bytes B5hex und 1Chex, was dem Wert 1CB5hex entspricht. Durchsuchen Sie die Liste der MCBs nach diesem Umgebungsbereich, und prüfen Sie, ob der PSP-Zeiger des Umgebungsbereichs nun umgekehrt auf den PSP des Programms zeigt. Ist dies der Fall, ist dieser Umgebungsbereich dem Programm zugeordnet, und der zum Umgebungsbereich gehörende Name ist der Programmname. Mit der Funktion »Scan_Name« tilgen wir etwa vorhandene Pfadbezeichnungen aus dem Namen.

Byte	Verwendung
00	ASCII-Zeichen »M«, wenn noch ein MCB folgt ASCII-Zeichen »Z«, wenn dies der letzte Block ist
01 – 02	PSP-Adresse
03 – 04	Länge des Speicherblocks in 16-Byte-Paragraphen
08 – 15	bei Programmen oft der Name

Tabelle 5. Aufbau des Memory-Control-Blocks (MCB)

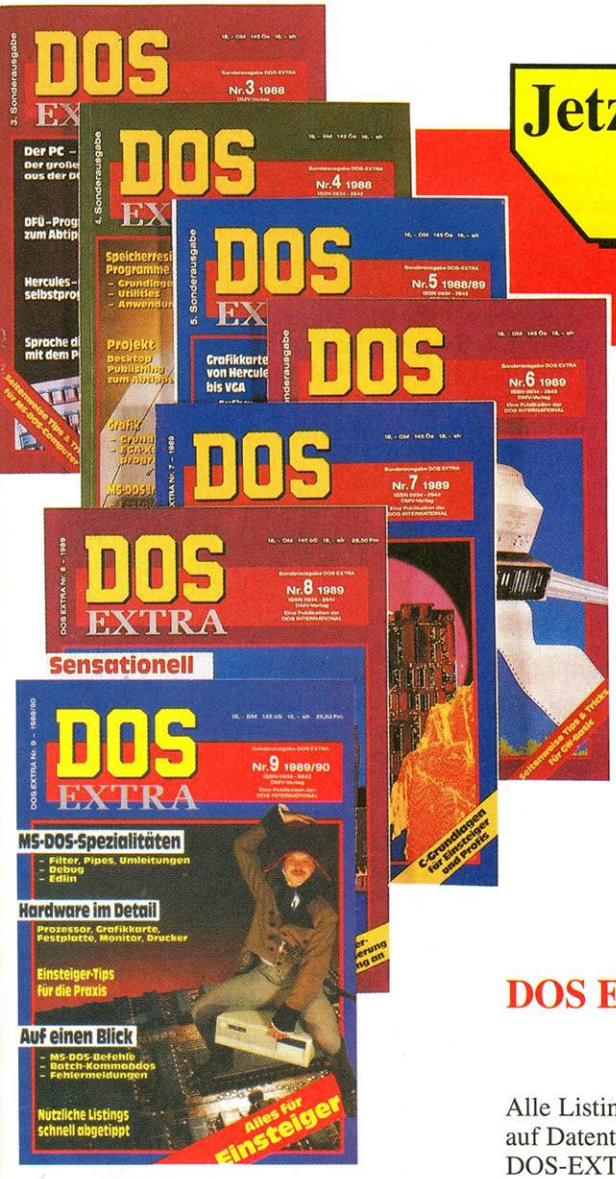
Die letzte Gruppe identifizierbarer MCBs gehört zu den Stapeldateien. Auch hier sind Sie auf Beobachtungen angewiesen. Bild 6 und 7 zeigen jeweils ein Beispiel für MS-DOS 4.0 und MS-DOS 3.2.

Die Situation ist nun etwas kompliziert. Die Speicherblöcke für Stapelprogramme unterscheiden sich in ihrem Aufbau in den Versionen bis MS-DOS 3.2 und ab 3.3. Das hat einen einfachen Grund: Ab MS-DOS 3.3. kann eine Stapeldatei eine andere über »call« aufrufen. Daher muß eine Stapeldatei nun zusätzlich zu anderen Informationen auch die Rücksprungadresse (»yy yy« in Tabelle 6) speichern. Bekannt ist weiterhin, daß MS-DOS eine Stapeldatei über einen Bytezähler verwaltet. Dieser zeigt immer auf das nächste noch nicht bearbeitete Byte der Stapeldatei. MS-DOS speichert lediglich diesen Zähler als Programmstatus im Speicherblock (»xx xx xx xx« in Tabelle 6).

Wir haben nicht alles entschlüsseln können, was in einem solchen Speicherblock steht. Die Informationen in Tabelle 6 stimmen jedenfalls in einer Vielzahl überprüfter Beispiele. Nicht immer ist die Folge der FF-Bytes lückenlos. Fol-

```
Adreß-Byte      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0B43 1. BCB    44 44 0B 14 00 9A F0 FE 58 4D 41 45 4D 20 20 20
                D                    X  M  A  E  M
0B58 2. BCB    44 59 0B B7 04 00 F9 0B 58 4D 41 32 45 4D 53 20
                D                    X  M  A  2  E  M  S
```

Bild 8. Beispiel für BIOS-Control-Blocks



Jetzt auch im Abo

DOS EXTRA

Immer wieder aktuell

3 Der PC glasklar
4 MS-DOS intern
5 Von Basic bis BIOS
6 Alles über Basic
7 Power für Programmierer
8 Assembler von Anfang an
9 Alles für Einsteiger
10 MS-DOS 4.01

DOS Extra je Ausgabe **DM 18,—***

Alle Listings der DOS-EXTRA-Ausgaben sind auf Datenträger als DATABOX erhältlich. Alle DOS-EXTRA-Ausgaben und DOS-EXTRA-DATABOXEN bitte direkt beim Verlag mit beiliegender Bestellkarte anfordern.

Databox-Inhalte der DOS Extra Nr. 10

- Units-Bibliotheken für Turbo Pascal: Datum, Sound, Bildschirmausgabe
- Prüflisting für die Leistungsdaten der Hardware
- Snapshot-Utility, Version 2.7
- TSR-Datei-Manager
- DOS-Assembler SAM III, Version 1.1
- Schnelles Kopierprogramm mit 115200 Bit pro Sekunde
- Komfortable Dateiumleitung
- Hercules-Grafik in Basic programmiert
- Patch für die BGI-Treiber von Turbo Pascal
- Verbesserungen zu SBasic

DOS TASCHENBÜCHER

Ihre PC-Informationen im Westentaschenformat - platzsparend und immer griffbereit!

Band 1:
Von Assembler bis MS-DOS

Die Neuauflage des legendären, seit langem vergriffenen Sonderheftes DOS EXTRA 1: Die wichtigsten Interna des PC als Beiträge, Tabellen und Listings. Wissen, das jeder PC-Besitzer haben muß!

Band 2:
Word 3.0 und 4.0

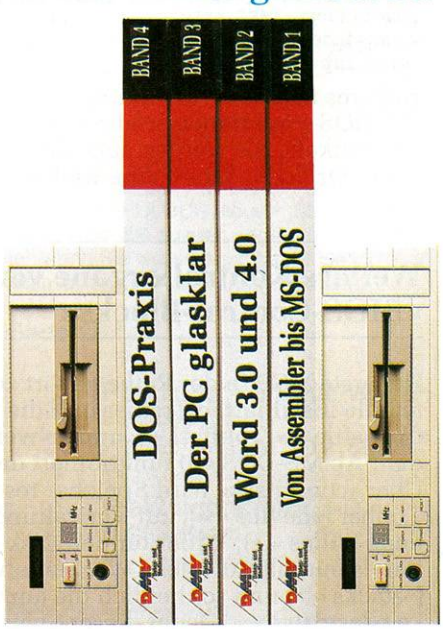
Mit Tips, Tricks und gezieltem Know-how über das Textverarbeitungsprogramm Word sparen Sie viel Zeit und Ärger. Sie lernen den Umgang mit Druckformatvorlagen, Textbausteinen, Makros etc. Ein Muß für jeden Word-Anwender.

Band 3:
Der PC glasklar

Sie erhalten einen Einblick in das geheimnisvolle Innere der PC-Hardware und die Funktionen des PC-Betriebssystems. Wichtige Begriffe wie BIOS, Interrupt und EMS-Speicher werden leicht verständlich erklärt. Das Buch für den interessierten PC-Einsteiger.

Band 4:
DOS-Praxis

Wenn es um die Modifikation der »Config.sys«, das Schreiben von Batch-Dateien, Datensicherung bei Festplatten, Gerätetreiber oder die Beherrschung der DOS-Kommandos geht — dann ist dieses Taschenbuch genau das Richtige für Sie!



PC-Wissen kompakt: DOS-Taschenbücher

DOS-Taschenbücher
Alle Bände ca. 270 Seiten, mit vielen Abbildungen, Hardcover broschiert — jeweils DM **29,80***

DATABOX
Alle Listings, Beispiele und Programme zu den DOS-Taschenbüchern Band 1, 3 und 4 erhalten Sie auch auf Datenträger zum günstigen DATABOX-Preis — jeweils DM **24,—***

* Unabhängig von der Anzahl der bestellten Produkte berechnen wir für das Inland DM 4,— bzw. für das Ausland DM 6,— Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

MS-DOS bis Version 3.2 – xx xx xx xx = Bytezähler	
Byte	00 01 02 03 04 05 06 07..0A 0B .. 18 19..?? ??+1
Inhalt	00 xx xx xx xx ?? ?? FF..FF FF .. FF Name 00
MS-DOS ab Version 3.3 – yy yy = Rücksprungzeiger	
Byte	00 01 02 03 04 05 06 07. 08 09 0A 0B .. 1E 1F .. ?? ??+1
Inhalt	00 01 00 yy yy 00 00 xx xx xx xx FF .. FF Name 00

Tabelle 6. Aufbau des Speicherblocks einer Stapeldatei

gende Prüfung wird in der Prozedur »Test_Batch« durchgeführt: Das erste Byte in einem Stapelprogramm-Speicherblock hat den Wert 00hex. Bis zu MS-DOS 3.2 beginnt der Name ab Byte 26. Wir lesen die Bytefolge, bis wir 00hex finden, und prüfen, ob die letzten vier Byte die Zeichenfolge ».bat« enthalten. Ab MS-DOS Version 3.3 ist der Name ab Byte 31 gespeichert.

Die Hauptarbeit des Programms »mcbliste.pas« wird in den Prozeduren »GetFirstMCB«, »GetNextMCB« und »MCB_Analyse« geleistet. »GetFirstMCB« ortet den ersten MCB mit Hilfe der (undokumentierten) MS-DOS-Funktion 52hex.

Alle MCBs werden in einem Array »L« gespeichert. Die Elemente des Arrays sind Datensätze vom Typ »MCBtyp«, der in der Unit »Memory« vereinbart wird. Jeder Record speichert das identifizierende Zeichen »M« oder »Z«, die PSP-Adresse, die Segmentadresse der zugehörigen Umgebung, wenn der MCB zu einem Programm gehört, die Segmentadresse des MCB selbst und den Typ, soweit er identifiziert werden kann.

Der erste MCB wird in »L[1]« gespeichert. Ist »L[n]« bekannt und noch nicht der letzte MCB der Kette, so bestimmen Sie »L[n + 1]« mit der Prozedur »GetNextMCB«. Die Segmentadresse des MCB ergibt sich aus der des Vorgängers und der Länge des zum Vorgänger gehörenden Speicherblocks. Dann ist es kein Problem, die PSP-Adresse und die Länge des neuen Speicherblocks zu bestimmen. Es schließt sich die Bestimmung des Typs an. Dazu verfahren Sie folgendermaßen:

```
Wenn PSP-Adresse Null ist,
dann ist Speicherblock frei
sonst: wenn »Test_PSP« positiv ausfällt,
dann handelt es sich um ein Programm
sonst: wenn »Test_Env« positiv ausfällt,
dann handelt es sich um eine Umgebung
sonst: wenn »Test_Batch« positv ausfällt,
dann handelt es sich um eine Stapeldatei
sonst ist der Typ unbekannt.
```

Näheres über die Verwaltung des Arbeitsspeichers über MS-DOS-Funktionen erfahren Sie im weiteren Verlauf dieses Artikels. Sie werden auch erfahren, warum manche Speicherblöcke nicht ohne weiteres identifizierbar sind.

Weitere Kontrollorgane von MS-DOS: die »BIOS-Control-Blocks«

Der erste MCB in der Reihe gehört zum BIOS und verwaltet alle installierten Treiber und die Speicherbereiche, die über »file = «, »buffers = « und so weiter eingerichtet werden. Ab MS-DOS 4.0 funktioniert das folgendermaßen: Über »config.sys« wird Speicher reserviert. Dieser ist für Treiber (»device = «), zur Verwaltung von geöffneten Dateien (»files = «), Puffer für den Disk-Zugriff (»buffers = «), Dateisteuerblöcke (»fcbs = «), Stapelverwaltung (»stacks = «) und schließlich die anzusteuernden Laufwerke (»lastdrive = «). Jeder dieser Speicherbereiche wird über einen 16 Byte großen Block verwaltet, der »BIOS-Control-Block« (BCB) genannt wird. Jeder BCB geht unmittelbar dem Bereich voran, den er kontrolliert. Der erste BCB schließt sich direkt an den ersten MCB an (Tabelle 7).

Die Verkettung der Blöcke ist ebenso wie bei den MCBs indirekt. Aus der Adresse des BCB und der Länge des kontrollierten Blocks können Sie die Adresse des nächsten Blocks berechnen. Ein Beispiel dazu sehen Sie in Bild 8.

Byte 0 enthält jeweils den ASCII-Wert 44hex (Zeichen »D«). Es werden also Treiber verwaltet. Die Namen finden Sie ab Byte 8. Byte 1 und 2 geben die Segmentadresse des kontrollierten Blocks an. Block 1 mit der Adresse 0B44hex (»44 0B«) beginnt also unmittelbar hinter seinem BCB. Die Länge können Sie Byte 3 und 4 entnehmen. Bei Block 1 ist sie 14B0hex (»0B 14«). Addieren Sie diese Länge auf die Startadresse des Blocks, also 0B44hex, so erhalten Sie die Adresse des BCB des zweiten Blocks, also 0B58hex.

Den Start der Kette finden Sie genauso wie den ersten MCB, denn der erste BCB schließt sich an den ersten MCB an. Der Aufbau im Speicher ist immer der gleiche: Zuerst werden die Treiber behandelt, dann »files«, »fcbs«, »buffers«, »lastdrive« und schließlich »stacks«. Das Ende der Kette ist daher am Kennzeichner »S« für »stacks« erkennbar.

Das Programm MCB gibt Ihnen insgesamt einen weitgehenden Überblick über die variable Belegung des Basisspeichers. Es liefert in einigen Teilen das gleiche Resultat wie das Mem-Kommando von MS-DOS 4.0. MS-DOS unterstützt die Speicherverwaltung auf der Seite des Programmierers durch mehrere Funktionen, also Unterfunktionen des Interrupt 21hex. Analoge Aufrufe verwendet »command.com« beim Plazieren von Programmen im Arbeitsspeicher. Folgende vier davon sind wichtig:

48hex: Speicherblock zuordnen
49hex: Speicherblock freigeben
4Ahex: Größe eines Speicherblocks ändern
58hex: Zuordnungsstrategie festlegen (undokumentiert)

Interessant ist, wie MS-DOS vorgeht. Angenommen, Sie laden über den Kommandoprozessor ein Programm. Das Betriebssystem muß dann zwei Speicherblöcke reservieren, einen für die Kopie des Umgebungsbereichs und einen für das eigentliche Programm, die zugehörigen Daten und den Stapelspeicher. Zunächst muß MS-DOS ermitteln, wieviel Speicher benötigt wird. Das hängt aber vom Typ des Programms ab. Normalerweise beansprucht ein Com-Programm den gesamten freien Rest des Arbeitsspeichers. Minimal braucht es soviel, wie der PSP (256 Byte), der eigentliche Programmcode und ein Stapel von mindestens zwei Byte ergeben. Bei Exe-Programmen ist das etwas anders. Deren Programmkopf enthält zwei Felder mit den Namen »MinAlloc« und »MaxAlloc«. MinAlloc gibt an, wieviel Platz (gemessen in Paragraphen) das Programm auf jeden Fall über den unmittelbaren Platzbedarf für Code und Daten hinaus braucht. MaxAlloc beschreibt, wieviel Platz nach Möglichkeit reserviert werden soll. Der Ladevorgang wird abgebrochen, wenn bereits MinAlloc

Byte	Bedeutung
00	Kennzeichen »D«: »device = « »F«: »files = « »X«: »fcbs = « »B«: »buffers = « »L«: »lastdrive = « »S«: »stacks = «
01 – 02	Segmentadresse des Speicherblocks
03 – 04	Länge in Paragraphen
08 – 0F	Name, nur bei Treibern

Tabelle 7. Aufbau des BIOS-Control-Blocks (BCB)

zu groß ist. MaxAlloc wird in der Regel auf FFFFhex (das entspricht 1 MByte) gesetzt. Wenn der über MinAlloc errechnete Platzbedarf zugeordnet werden kann, wird soviel Speicher reserviert, wie MaxAlloc angibt beziehungsweise, falls dies nicht möglich ist, noch verfügbar ist.
Zurück zu den MCBs. Der freie Arbeitsspeicher kann fragmentiert sein, denn es ist möglich, einen Speicherbereich wieder freizugeben. Wie soll MS-DOS vorgehen, wenn erneut Speicher zu reservieren ist? Prinzipell stehen drei Strategien zur Verfügung, die über die Funktion 58hex festgelegt werden können:
1. »first fit«: Suche den ersten passenden freien Bereich
2. »best fit« : Suche den kleinsten passenden freien Bereich
3. »last fit« : Suche den letzten passenden freien Bereich
Die Methode »first fit« hat sich als die effektivste herausgestellt und wird daher von MS-DOS als Standardmethode eingesetzt. Wenn nun ein Speicherblock gewählt wird, der größer ist als der angeforderte, bleibt Platz übrig. In diesem Fall wird der Speicherblock in zwei Blöcke aufgeteilt: einen für die Anforderung und einen für den freien Rest. Beide werden dann über jeweils einen MCB koordiniert. Wenn Speicher angefordert wird, muß das Betriebssystem einen zusammenhängenden Speicherbereich finden, der ausreichend groß ist. Aufteilen ist nicht erlaubt, deshalb bietet sich folgende Methode an: Insgesamt ist genügend Speicherplatz vorhanden, aber kein Block ist groß genug. MS-DOS meldet dann: Nicht genug Arbeitsspeicher. Das verwendete Programm »mcbliste.pas« gibt Auskunft über die Summe des freien Speichers und über den größten verfügbaren Speicherblock.

MS-DOS stellt alle wichtigen Hilfsmittel für die Speicherverwaltung zur Verfügung

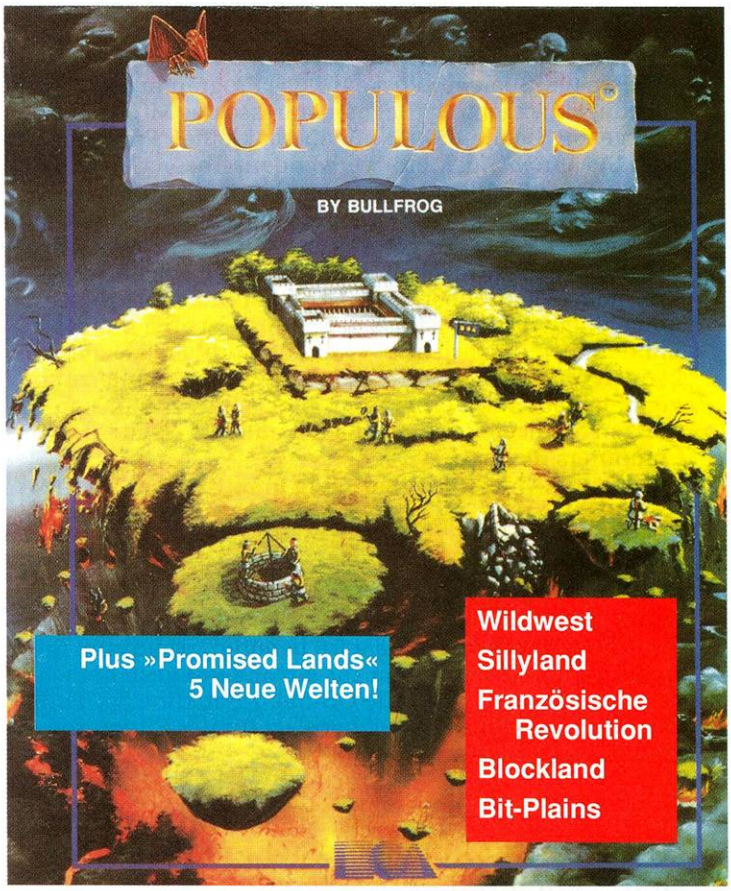
Es seien nun die Funktionen zur Speicherverwaltung etwas genauer erläutert:
Funktion 48hex: (Speicher belegen)
Aufruf:
Register AH: 48hex
Register BX: Anzahl der zu reservierenden Paragraphen
Nach erfolgreichem Aufruf enthält das Register AX die Segmentadresse des zugeordneten Speicherblocks. Im Fehlerfall wird das Carryflag gesetzt, und AX enthält eine Fehlernummer. BX gibt dann an, wieviel Speicherplatz maximal verfügbar ist. Die Turbo-Pascal-Prozedur »AllocMem« in der Unit »Memory« reserviert Speicher mit dieser MS-DOS-Funktion. Wird Platz auf diese Weise reserviert, klassifiziert »mcbliste.pas« den Speichertyp als »unbekannt«, da es sich bei dem reservierten Block weder um ein Programm, einen Umgebungsbereich noch um eine Stapeldatei handelt. Sie können sich den Speicherblock mit Hilfe des Programms »debug« auch manuell ansehen. Wird als Segmentadresse des Speicherblocks zum Beispiel 1BC8hex ausgegeben, so finden Sie den zugehörigen MCB ab 1BC7hex. Geben Sie also nach dem Starten des Debuggers ein:

```
d 1BC7:0000
```

Belegte Speicherblöcke können mit der MS-DOS-Funktion 49hex wieder freigegeben werden:
Funktion 49hex: (Belegten Speicher wieder freigeben)
Aufruf:
Register AH: 49hex
Register ES: Segmentadresse des freizugebenden Blocks
Die Funktion »FreeAllocMem« erfüllt den gleichen Zweck in Turbo-Pascal-Programmen. Sie ist, wie »AllocMem« in der Unit »Memory« verfügbar.
Sie können mit den beiden eben beschriebenen Prozeduren (oder auch den Funktionen) experimentieren und sich

DOS Spielekiste

Kennen Sie schon?



Spielen Sie Gottheit, Herrscher und Eroberer in einem

Haben Sie nicht auch schon davon geträumt, König oder Staatsmann zu sein und die Geschicke eines ganzen Volkes zu lenken? Mit Ihrem PC und dem Spiel POPULOUS stehen Sie im Mittelpunkt der Geschehnisse: Gewinnt die Stärke des Lichts oder die Macht der Finsternis?
Die Entscheidung liegt bei Ihnen!

In neun verschiedenen Welten mit jeweils 100 Schwierigkeitsstufen können Sie bei POPULOUS Ihre Anhänger um sich scharen und durch geschicktes Vorgehen zur Weltherrschaft führen. Sie lösen Erdbeben aus, lassen Vulkane ausbrechen, geben den Anstoß zum Bau von Städten, Sie bilden Ritter heran und geben das Zeichen zum Kampf gegen den Rivalen. Dabei berechnet der Computer laufend Ihre Machtfülle nach dem Stand der Zivilisation Ihrer Anhänger.

Und POPULOUS kann noch viel mehr! Neben den fünf zusätzlichen Welten in unserem DOS-Leserangebot können Sie über 60 Spieleparameter menügesteuert verändern oder mit den Paint-Map-Funktionen beliebige, eigene Welten konstruieren. POPULOUS steuern Sie über Tastatur, Joystick oder perfekt mit der Maus. Sie spielen gegen den Computer oder gegen einen Spielpartner, wenn Sie wollen auch per Modem.
Ein Spiel - das jetzt endlich auch für den PC zur Verfügung steht!

Review in DOS 6/90, Seite 267

Nur solange der Vorrat reicht

Populous + 5 Neue Welten
Electronic Arts,
PC-5,25-Zoll-Disk. DM **69,95** *
und aus unserem laufenden Programm
Fraktalgenerator 3-D
DMV-Software,
PC-5,25-Zoll-Disk. DM **69,–** *

* Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

DMV - Verlag • Postfach 250 • 3440 Eschwege

DMV Daten- und Medienverlag

einen konkreten Einblick in die Speicherreservierung verschaffen. In Zusammenarbeit mit »mcbliste.pas« und »FreeAllocMem« dürfte es für Sie kein Problem sein, beispielsweise speicherresidente Programme aus dem Arbeitsspeicher zu entfernen. Sie müssen dabei allerdings »Handarbeit« leisten. Ermitteln Sie mit »mcbliste.pas« die Adresse der Speicherblöcke für das Programm und die zugeordnete Umgebung (Segmentadresse des MCB plus 1), und geben Sie die Blöcke mit einem kleinen Programm, in dem Sie »FreeAllocMem« verwenden, frei.
Ein einziges Problem können Sie damit allerdings nicht lösen: Wenn ein Programm Interrupts verändert, so können Sie diese Änderungen allein mit einer Speicherfreigabe nicht rückgängig machen. Das Programm »mcbliste.pas« informiert Sie darüber, ob Interrupts durch ein Programm umgeleitet wurden. Ist das der Fall, so ist eine Freigabe mit der oben beschriebenen Methode problematisch. Sie kennen nämlich die Adressen der betroffenen Interrupts vor dem Laden des Programms nicht. Im schlimmsten Fall müssen Sie nach der Freigabe mit einem Absturz Ihres Computers rechnen.

Funktion: 4Ahex (Blockbelegung modifizieren)
Aufruf:
Register AH: 4Ahex
Register ES: Segmentadresse des Blocks
Register BX: Anzahl der angeforderten Paragraphen
In der Unit »Memory« finden Sie die entsprechende Prozedur »ModiAllocMem«. Sie wird benötigt, wenn zum Beispiel von einem Programm aus ein anderes gestartet werden soll (beispielsweise die OS-Shell-Funktion bei Turbo Pascal) und daher der Speicherplatz für das Elternprogramm verkleinert werden muß.

Auch nichtdokumentierte Funktionen leisten manchmal gute Dienste

Mit der nichtdokumentierten MS-DOS-Funktion 58hex kann die Speicherverwaltungsmethode festgelegt beziehungsweise erfragt werden:

Funktion: 58hex (Speicherverwaltungsstrategie setzen/abfragen)
Aufruf:
Register AH: 58hex
Register AL: 00hex Ermitteln der Strategie
01hex Festlegen der Strategie
Register BX: 00hex: »first fit«
01hex: »best fit«
02hex: »last fit«

Bei der Festlegung (AX enthält 01hex) wird über BX die gewünschte Strategie eingegeben, bei der Ermittlung ausgegeben. Dazu dienen die Funktion »GetAllocStrat« und die Prozedur »SetAllocStrat« in der Unit »Memory«. Experimentieren Sie ein wenig mit den vorgestellten Hilfsmitteln. Sie werden auf diese Weise einen sehr plastischen Eindruck von der MS-DOS-Speicherverwaltung erhalten.

Zum Abschluß dieses Überblicks über Speicherverwaltung beschäftigen wir uns noch kurz mit dem »Expanded Memory«. Alle diesbezüglichen Funktionen werden über den Interrupt 67hex abgewickelt, der über eine Vielzahl von Unterfunktionen verfügt. Wir konzentrieren uns auf die Funktionen, die einen Überblick über die augenblickliche Belegung geben. Sie finden in der Turbo-Pascal-Unit »Memory« entsprechende Routinen.

Zunächst müssen Sie feststellen, ob überhaupt EMS-Speicher installiert ist. Ein Indiz dafür ist, daß der Interrupt 67hex belegt ist. Er könnte allerdings auch für eine andere Funktion verwendet worden sein. Prüfen Sie daher, ob der Interrupt-Handler die Bezeichnung »EMMXXXX0« ausweist (die Zeichenfolge »XXXX« steht hier nicht für einen Platzhalter, sondern als tatsächlicher Teil der Bezeichnung). Das ist der offizielle Name des EMS-Treibers. Die Nummer der EMS-Version bekommen Sie über die Unterfunktion 46hex des EMS-Interrupts.

EMS-Speicher wird in 16-KByte-Seiten (Pages) verwaltet. Sie können nun ermitteln, wie viele Seiten überhaupt zur Verfügung stehen und wie viele momentan belegt sind. Dafür ist die Unterfunktion 42hex des EMS-Interrupts vorgesehen. In der Unit »Memory« finden Sie dazu passend die Funktion »EMS—Seiten—gesamt« und »EMS—Seiten—frei«. Eine Seite des Expanded Memory muß in den Basisspeicher eingeblendet werden. Dafür muß ein entsprechender Speicher, der sogenannte »Page Frame« reserviert werden. Sie finden ihn über die Unterfunktion 41hex des EMS-Interrupts, auf der die Unit-Funktion »EMS—Page—Frame» beruht. Das sind schon die wichtigsten Funktionen, die Informationen über das Expanded Memory geben. Der Vollständigkeit halber finden Sie in der Unit »Memory« noch eine Funktion »EXT—Memory«, die über die Größe des Extended Memory berichtet. Sie beruht auf einer erweiterten Funktion des Interrupt 15hex, die für 8086/8088-Systeme allerdings nicht existieren.

(Joachim Jäger/ks)

```
[P7Ø]    1: {
[PW3]    2: ***************************************
[ØD4]    3: *       U N I T   M E M O R Y       *
[PW3]    4: ***************************************
[FT5]    5: * Das Unit stellt Funktionen und     *
[DK6]    6: * Prozeduren zur Speicherverwaltung  *
[ZS4]    7: * zur Verfügung:                     *
[WX5]    8: * - Ermittlung der Treiberadressen   *
[WM5]    9: * - Ermittlung und Einlesen der      *
[ØZ4]   1Ø: *   Memory Control Blocks            *
[ØD6]   11: * - Routinen zur Speicherverwaltung  *
[Z15]   12: *   mit MS-DOS-Funktionen            *
[HH6]   13: * - darüberhinaus allgemein nützliche*
[E44]   14: *   Prozeduren                       *
[PW3]   15: ***************************************
[BG5]   16: * Autor: Dr. Joachim Jäger           *
[PW3]   17: ***************************************
[U7Ø]   18: }
[762]   19: Unit Memory;
[ØØØ]   2Ø:
[SK1]   21: interface
[ØØØ]   22:
[S62]   23: uses Dos,Crt;
[ØØØ]   24:
[GB5]   25: type Driver_Ptr=^Driver_Head;
[ZG3]   26:      Driver_Head = record
[Z35]   27: {Aufbau eines Treiberkopfes}
[ØB3]   28:      Zeiger:Driver_Ptr;
[ØL2]   29:      Attribut:Word;
[614]   3Ø:      StratOfs,IntrOfs:Word;
```

```
[G74]   31:      Name:array[Ø..7] of Byte;
[ZNØ]   32:      end;
[TB3]   33:      String9=String[9];
[FK2]   34:      MCBTyp = record
[6R4]   35: {MCB = Memory Control Block}
[EY1]   36:            Ident:Char;
[7K4]   37:            PSPSeg,ENVSeg,Laenge:Word;
[U52]   38:            Name:String;
[V12]   39:            MCBSeg:Word;
[G72]   4Ø:            Typ:String9;
[ZNØ]   41:            end;
[TH2]   42:      BCBTyp = record
[512]   43:                        BCBSeg:Word;
[EY1]   44:                        Ident:Char;
[PA2]   45:                        BIOSSeg:Word;
[S12]   46:                        Laenge:Word;
[E92]   47:                        Name:String9;
[8L1]   48:                        S:String;
[ZNØ]   49:                end;
[AR2]   5Ø:     AllocRec = record
[9S2]   51:                  Paragraph:Word;
[TB2]   52:                  Segment:Word;
[E32]   53:                  Fehler:Byte;
[ZNØ]   54:                end;
[ØØØ]   55:
[7H3]   56: var Regs:Registers;
[ØØØ]   57:
[PM6]   58: {allgemein nützliche Funktionen und}
[DC2]   59: {Prozeduren}
[ØØØ]   6Ø:
```

```
[2N5]   61: function Bit(w:word;p:Byte):Byte;
[XY4]   62: function Hex(x:Word):String;
[B88]   63: function fill(Anzahl:Word;Zeichen:Char):Str
            ing;
[UK4]   64: procedure Titel(t:String);
[8Y2]   65: procedure ClrEop;
[1P4]   66: procedure Seitenschaltung;
[9P5]   67: function PtrSeg(p:Pointer):Word;
[MR5]   68: function PtrOfs(p:Pointer):Word;
[165]   69: function DOS_Version:String;
[M5B]   7Ø: procedure ReadBlock(Segm,Offs:Word;Anzahl:B
            yte;var Block:String);
[ØØØ]   71:
[166]   72: {spezielle Funktionen für Treiber}
[ØØØ]   73:
[2V5]   74: function FirstDriverAdr:Pointer;
[X38]   75: function NextDriverAdr(p:Driver_Ptr):Pointe
            r;
[ØØØ]   76:
[676]   77: {spezielle Routinen für Memory}
[413]   78: {Control Blocks}
[ØØØ]   79:
[NK6]   8Ø: procedure GetFirstMCB(var MCB:MCBTyp);
[AH7]   81: procedure GetNextMCB(var Vorg,Nachf:MCBTyp)
            ;
[ØØØ]   82:
[UV5]   83: {spezielle Prozeduren für BIOS}
[413]   84: {Control Blocks}
[ØØØ]   85:
[HE6]   86: procedure GetFirstBCB(var BCB:BCBTyp);
[NL7]   87: procedure GetNextBCB(var Vorg, Nachf: BCBTy
            p);
[ØØØ]   88:
[RR6]   89: {Routinen zur Speicherallokation mit}
[423]   9Ø: {DOS-Funktionen}
[ØØØ]   91:
[ND6]   92: procedure AllocateMem(var A:AllocRec);
[ØH6]   93: procedure FreeAllocMem(var A:AllocRec);
[3K6]   94: procedure ModiAllocMem(var A:AllocRec);
[K15]   95: function GetAllocStrat:Word;
[F27]   96: procedure SetAllocStrat(Strategie:Byte);
[ØØØ]   97:
[3D6]   98: {Routinen für EMS, Extended Memory}
[VM4]   99: function EMS_Test:Boolean;
[VN4]  1ØØ: function EMS_Status:Word;
[W55]  1Ø1: function EMS_Version:String;
[9V5]  1Ø2: function EMS_Seiten_gesamt:Word;
[AF5]  1Ø3: function EMS_Seiten_frei:Word;
[785]  1Ø4: function EMS_Page_Frame:Word;
[BN4]  1Ø5: function EXT_Memory:Word;
[ØØØ]  1Ø6:
[4L2]  1Ø7: implementation
[ØØØ]  1Ø8:
[2N5]  1Ø9: function Bit(w:word;p:Byte):Byte;
[WT5]  11Ø: {ermittelt den Wert von Bit p in}
[AK1]  111: {Word w}
[MUØ]  112: begin
[T24]  113:   Bit:=(w and (1 shl p)) shr p;
[ZNØ]  114: end;
[ØØØ]  115:
[XY4]  116: function Hex(x:Word):String;
[7G6]  117: {wandelt Word in Hexdarstellung um}
[FB3]  118: var tØ,t1,t2,t3:Byte;
[P12]  119:     HZ: String;
[MUØ]  12Ø: begin
[DB3]  121:   HZ:='Ø123456789ABCDEF';
[LR1]  122:   tØ:=x and $F;
[433]  123:   t1:=(x and $FØ) shr 4;
[C83]  124:   t2:=(x and $FØØ) shr 8;
[UV1]  125:   t3:=x shr 12;
[RW6]  126:   Hex:=HZ[t3+1]+HZ[t2+1]+HZ[t1+1]+HZ[tØ+1];
[ZNØ]  127: end;
[ØØØ]  128:
[B88]  129: function fill(Anzahl:Word;Zeichen:Char):Str
            ing;
[CF7]  13Ø: {füllt einen String mit gegebener Anzahl}
[UY5]  131: {von Exemplaren eines Zeichens}
[Z92]  132: var s:String;
[MUØ]  133: begin
[GR4]  134:   FillChar(s,Anzahl+1,Zeichen);
[7X2]  135:   s[Ø]:=Chr(Anzahl);
[UA1]  136:   Fill:=s
[ZNØ]  137: end;
[ØØØ]  138:
[UK4]  139: procedure Titel(t:String);
[BB5]  14Ø: {erzeugt eine zentrierte und}
[3L4]  141: {invertierte Titelzeile}
[VY1]  142: var i:Byte;
[H72]  143:     z:String[8Ø];
[MUØ]  144: begin
[TØ3]  145:   TextBackground(Ø);
[DB1]  146:   ClrScr;
[VE3]  147:   i:=(8Ø-Length(t)) div 2;
[NA4]  148:   z:=Fill(i,' ')+t+Fill(i-1,' ');
[SB5]  149:   TextBackground(7);Textcolor(Ø);
[5H1]  15Ø:   Write(z);
[ZB5]  151:   TextBackground(Ø);TextColor(7);
[PV1]  152:   GotoXY(1,3);
[ZNØ]  153: end;
[ØØØ]  154:
[8Y2]  155: procedure ClrEop;
[TK5]  156: {Löscht den Bildschirm ab der}
[EL2]  157: {Cursorzeile}
[N64]  158: var i,Spalte,Zeile:Byte;
[MUØ]  159: begin
[CR2]  16Ø:   Spalte:=WhereX;
[5E2]  161:   Zeile:=WhereY;
[5B1]  162:   ClrEol;
[9K3]  163:   for i:=Zeile+1 to 25 do
[LM4]  164:   begin GotoXY(1,i); ClrEol end;
[XG3]  165:   GotoXY(Spalte,Zeile)
[ZNØ]  166: end;
[ØØØ]  167:
[1P4]  168: procedure Seitenschaltung;
[7U6]  169: {bereitet die nächste Bildschirmseite}
[Ø41]  17Ø: {vor}
[K13]  171: var Antwort:Char;
[MUØ]  172: begin
[7Ø2]  173:   GotoXY(1,25);
[AK6]  174:   Write('Bitte eine Taste drücken ... ');
[223]  175:   Antwort:=ReadKey;
[WV1]  176:   GotoXY(1,5);
[541]  177:   ClrEop
[ZNØ]  178: end;
[ØØØ]  179:
[9P5]  18Ø: function PtrSeg(p:Pointer):Word;
[BS6]  181: {ermittelt das Segment eines Pointers}
[MUØ]  182: begin
[EV4]  183:   PtrSeg:=MemW[Seg(p):Ofs(p)+2]
[ZNØ]  184: end;
[ØØØ]  185:
[MR5]  186: function PtrOfs(p:Pointer):Word;
[EK6]  187: {ermittelt den Offset eines Pointers}
[MUØ]  188: begin
[CV4]  189:   PtrOfs:=MemW[Seg(p):Ofs(p)]
[ZNØ]  19Ø: end;
[ØØØ]  191:
[165]  192: function DOS_Version:String;
[RD6]  193: {bestimmt die Versionsnummer in der}
[BS3]  194: {Form 32Ø für DOS 3.2Ø}
[MX2]  195: var hn,un:String;
[MUØ]  196: begin
[LX1]  197:   Regs.AH:=$3Ø;
[3Ø2]  198:   MsDos(Regs);
[956]  199:   Str(Regs.AL:1,hn); {hn: Hauptnummer}
[R86]  2ØØ:   Str(Regs.AH:2,un); {un: Unternummer}
[VZ3]  2Ø1:   DOS_Version:=hn+'.'+un;
[ZNØ]  2Ø2: end;
[ØØØ]  2Ø3:
[M5B]  2Ø4: procedure ReadBlock(Segm,Offs:Word;Anzahl:B
            yte;var Block:String);
[P36]  2Ø5: {überträgt einen Speicherblock}
[VM4]  2Ø6: {(max 255 Zeichen) in String}
[MUØ]  2Ø7: begin
[SU4]  2Ø8:   if Anzahl>255 then Anzahl:=255;
[YL3]  2Ø9:   Block[Ø]:=Chr(Anzahl);
[4E6]  21Ø:   Move(Mem[Segm:Offs],Block[1],Anzahl)
[ZNØ]  211: end;
[ØØØ]  212:
[2V5]  213: function FirstDriverAdr:Pointer;
[676]  214: {erzeugt einen Pointer, der auf die}
[DH6]  215: {Adresse des ersten Treibers zeigt}
[UG2]  216: var P:Pointer;
[MUØ]  217: begin
[XX1]  218:   Regs.AH:=$52;
[1Z6]  219: {Funktion für DOS-Informationsbereich}
[3Ø2]  22Ø:   MsDos(Regs);
[PA5]  221:   MemW[Seg(P):Ofs(P)]:=Regs.BX+$22;
[176]  222: {Adresse des Zeigers manipulieren}
[F75]  223:   MemW[Seg(P):Ofs(P)+2]:=Regs.ES;
[Z73]  224:   FirstDriverAdr:=P;
[ZNØ]  225: end;
[ØØØ]  226:
[X38]  227: function NextDriverAdr(p:Driver_Ptr):Pointe
            r;
[VU6]  228: {ermittelt die nächste Treiberadresse}
[MUØ]  229: begin
[BD4]  23Ø:   NextDriverAdr:=P^.Zeiger;
[ZNØ]  231: end;
[ØØØ]  232:
[NK6]  233: procedure GetFirstMCB(var MCB:MCBTyp);
[2W3]  234: {liest den ersten MCB}
[MUØ]  235: begin
[XX1]  236:   Regs.AH:=$52;
[SF4]  237: {DIB-Adresse ermitteln}
[3Ø2]  238:   MsDos(Regs);
[YY1]  239:   with MCB do
[MUØ]  24Ø:   begin
[XX1]  241:     if Regs.BX<2
[HF6]  242:     then MCBSeg:=MemW[Regs.ES-1:Regs.BX+6]
[586]  243:     else MCBSeg:=MemW[Regs.ES:Regs.BX-2];
[394]  244:     Ident:=Chr(Mem[MCBSeg:Ø]);
[Z54]  245:     PSPSeg:=MemW[MCBSeg:1];
[K24]  246:     Laenge:=MemW[MCBSeg:3];
[TV1]  247:     if Ident<>'M'
[ZN1]  248:     then begin
[667]  249:            Writeln('Fehler bei Suche nach e
            rstem MCB');
[DUØ]  25Ø:          Halt;
[ZNØ]  251:        end;
[ØØØ]  252:
```

Listing 1. Eine Sammlung nützlicher Funktionen: die Unit »Memory« im Listing »memory.pas«

```
[ZNØ]  253:   end;
[ZNØ]  254: end;
[ØØØ]  255:
[AH7]  256: procedure GetNextMCB(var Vorg,Nachf:MCBTyp)
       ;
[Y76]  257: {liest den jeweils nächsten MCB ein}
[XT4]  258: {und ermittelt seinen Typ}
[482]  259: var i,p:Word;
[MUØ]  26Ø: begin
[MN6]  261:   Nachf.MCBSeg:=Vorg.MCBSeg+1+Vorg.Laenge;
[386]  262:   Nachf.Ident:=Chr(Mem[Nachf.MCBSeg:Ø]);
[Z56]  263:   Nachf.PSPSeg:=MemW[Nachf.MCBSeg:1];
[246]  264:   Nachf.Laenge:=MemW[Nachf.MCBSeg:3];
[7M6]  265:   if (Nachf.Ident<>'M') and (Nachf.Ident<>'
       Z')
[ZN1]  266:   then begin
[6W7]  267:          writeln('Fehler beim Lesen eines F
       olgeblocks');
[DUØ]  268:          halt;
[ZNØ]  269:        end;
[ZNØ]  27Ø: end;
[ØØØ]  271:
[HE6]  272: procedure GetFirstBCB(var BCB:BCBTyp);
[FT5]  273: {liest ersten BIOS-Control-Block}
[ZS2]  274: var MCB: MCBTyp;
[7A1]  275:     i:Word;
[MUØ]  276: begin
[SX2]  277:   GetFirstMCB(MCB);
[XX1]  278:   with BCB do
[MUØ]  279:   begin
[N93]  28Ø:     BCBSeg:=MCB.MCBSeg+1;
[S84]  281:     Ident:=Chr(Mem[BCBSeg:Ø]);
[L84]  282:     BIOSSeg:=MemW[BCBSeg:1];
[H14]  283:     Laenge:=MemW[BCBSeg:3];
[W91]  284:     Name:='';
[DP7]  285:     for i:=8 to 15 do Name:=Name+Chr(Mem[BC
       BSeg:i]);
[ZNØ]  286:   end;
[ZNØ]  287: end;
[ØØØ]  288:
[NL7]  289: procedure GetNextBCB(var Vorg, Nachf: BCBTy
       p);
[NM6]  29Ø: {liest nächsten BIOS-Control-Block}
[DY1]  291: var i: Word;
[MUØ]  292: begin
[P22]  293:   with Nachf do
[MUØ]  294:   begin
[XM5]  295:     BCBSeg:=Vorg.BIOSSeg+Vorg.Laenge;
[S84]  296:     Ident:=Chr(Mem[BCBSeg:Ø]);
[L84]  297:     BIOSSeg:=MemW[BCBSeg:1];
[H14]  298:     Laenge:=MemW[BCBSeg:3];
[W91]  299:     Name:='';
[DP7]  3ØØ:     for i:=8 to 15 do Name:=Name+Chr(Mem[BC
       BSeg:i]);
[ZNØ]  3Ø1:   end;
[ZNØ]  3Ø2: end;
[ØØØ]  3Ø3:
[ND6]  3Ø4: procedure AllocateMem(var A:AllocRec);
[SF6]  3Ø5: {Alloziert Speicher gemäß Angaben im
[Z71]  3Ø6:  Record A
[EU5]  3Ø7:  Aufruf: A.Paragraph: Anzahl der zu
[CL4]  3Ø8:  reservierenden Paragraphen
[ØS3]  3Ø9:  Fehler 7: MCB zerstört
[NL5]  31Ø:  Fehler 8: nicht genügend Platz}
[MUØ]  311: begin
[MZ2]  312:   with Regs do begin
[Y31]  313:     AH:=$48;
[ØR2]  314:     BX:=A.Paragraph;
[3Ø2]  315:     MsDos(Regs);
[LE2]  316:     if Flags and 1 = Ø
[ZN1]  317:     then begin
[ØB2]  318:            A.Segment:=AX;
[ET1]  319:            A.Fehler:=Ø
[4MØ]  32Ø:          end
[WP1]  321:     else begin
[HK4]  322:            A.Fehler:=AX;   {Fehlercode}
[EP2]  323:            A.Paragraph:=BX
[BC5]  324: {Maximal verfügbarer Platz}
[4MØ]  325:          end
[4MØ]  326:   end
[ZNØ]  327: end;
[ØØØ]  328:
[ØH6]  329: procedure FreeAllocMem(var A:AllocRec);
[BB3]  33Ø: {gibt Speicher frei
[TB6]  331:  Aufruf:  A.Segment: Segmentadresse des
[A25]  332:  freizugebenden Speicherblocks
[ØS3]  333:  Fehler 7: MCB zerstört
[ØX5]  334:  Fehler 8: ungültige Blockadresse}
[MUØ]  335: begin
[RØ2]  336:   with Regs do
[MUØ]  337:   begin
[241]  338:     AH:=$49;
[917]  339: {DOS-Funktion 49H: Freigabe von Speicher}
[NB2]  34Ø:     ES:=A.Segment;
[3Ø2]  341:     MsDos(Regs);
[LE2]  342:     if Flags and 1 = Ø
[ØM2]  343:     then  A.Fehler:=Ø
[GB5]  344:     else A.Fehler:=AX;  {Fehlercode}
[4MØ]  345:   end
[ZNØ]  346: end;
[ØØØ]  347:
```

```
[3K6]  348: procedure ModiAllocMem(var A:AllocRec);
[8B7]  349: {modifiziert bereits allozierten Speicher
[2R7]  35Ø:  Aufruf: A.Paragraph: neue Speichergröße in
[V22]  351:  Paragraphen
[ZT6]  352:  A.Segment: Segmentadresse eines bereits
[Z83]  353:  allozierten Blocks
[ØS3]  354:  Fehler 7: MCB zerstört
[145]  355:  Fehler 8: nicht genügend Platz
[1X5]  356:  Fehler 9: ungültige Blockadresse}
[MUØ]  357: begin
[RØ2]  358:   with Regs do
[MUØ]  359:   begin
[ØF4]  36Ø:     AH:=$4A; {DOS-Funktion 4AH:
[MY6]  361:  Modifizieren von alloziertem Speicher}
[NB2]  362:     ES:=A.Segment;
[ØR2]  363:     BX:=A.Paragraph;
[3Ø2]  364:     Msdos(Regs);
[LE2]  365:     if Flags and 1 = Ø
[GH2]  366:     then A.Fehler:=Ø
[WP1]  367:     else begin
[HK4]  368:            A.Fehler:=AX;   {Fehlercode}
[EP2]  369:            A.Paragraph:=BX
[BC5]  37Ø: {Maximal verfügbarer Platz}
[4MØ]  371:          end
[4MØ]  372:   end
[ZNØ]  373: end;
[ØØØ]  374:
[K15]  375: function GetAllocStrat:Word;
[K54]  376: {ermittelt die aktuelle
[KN3]  377:  Allokationstrategie:
[6W1]  378:  Ø: First fit
[WV1]  379:  1: Best fit
[XZ1]  38Ø:  2: Last fit}
[MUØ]  381: begin
[3Y1]  382:   Regs.AH:=$58;
[1V1]  383:   Regs.AL:=Ø;
[3Ø2]  384:   MsDos(Regs);
[LØ4]  385:   GetAllocStrat:=Regs.AX
[ZNØ]  386: end;
[ØØØ]  387:
[F27]  388: procedure SetAllocStrat(Strategie:Byte);
[HL6]  389: {setzt eine neue Allokationsstrategie
[6W1]  39Ø:  Ø: First fit
[WV1]  391:  1: Best fit
[XZ1]  392:  2: Last fit}
[MUØ]  393: begin
[3Y1]  394:     Regs.AH:=$58;
[4V1]  395:     Regs.AL:=1;
[TE3]  396:     Regs.BX:=Strategie;
[8Y1]  397:     MsDos(Regs)
[ZNØ]  398: end;
[ØØØ]  399:
[VM4]  4ØØ: function EMS_Test:Boolean;
[1Y5]  4Ø1: {Testet, ob EMS installiert ist}
[HW2]  4Ø2: var Name:String;
[481]  4Ø3:     i:Byte;
[MUØ]  4Ø4: begin
[RØ2]  4Ø5:   with Regs do
[MUØ]  4Ø6:   begin
[647]  4Ø7: {Test, ob Interrupt-Vektor 67H belegt ist}
[F31]  4Ø8:     AH:=$35;
[C41]  4Ø9:     AL:=$67;
[3Ø2]  41Ø:     MsDos(Regs);
[UD1]  411:     if ES+BX=Ø
[ØH3]  412:     then EMS_Test:=false
[2R6]  413:     else {Chance, daß Interruptvektor mit
[UG4]  414:           EMS-Interrupt belegt ist}
[MUØ]  415:       begin
[CF5]  416:         {Name des Interrupt-Handlers ab
[WØ2]  417:          ES:ØA lesen}
[W91]  418:         Name:='';
[B47]  419:         for i:=1Ø to 17 do Name:=Name+Chr(M
       em[ES:i]);
[FY2]  42Ø:         if Name='EMMXXXXØ'
[454]  421: {Test, on Name korrekt}
[EF3]  422:         then EMS_Test:=true
[LN3]  423:         else EMS_Test:=false;
[4MØ]  424:       end
[4MØ]  425:   end
[ZNØ]  426: end;
[ØØØ]  427:
[VN4]  428: function EMS_Status:Word;
[YØ7]  429: {Ermittelt Fehlerstatus bei EMS, Ø: kein
[VH1]  43Ø:  Fehler}
[MUØ]  431: begin
[RX1]  432:   Regs.AH:=$4Ø;
[ZC2]  433:   Intr($67,Regs);
[UH3]  434:   EMS_Status:=Regs.AH;
[ZNØ]  435: end;
[ØØØ]  436:
[W55]  437: function EMS_Version:String;
[W96]  438: {Bestimmt die EMS-Versionsnummer als
[5U3]  439:  dreistelligen String}
[YD4]  44Ø: var Hauptnr,Nebennr:Char;
[MUØ]  441: begin
[XX1]  442:   Regs.AH:=$46;
[ZC2]  443:   Intr($67,Regs);
[DW1]  444:   if Regs.AH=Ø
[2RØ]  445:   then
[MUØ]  446:     begin
[T85]  447:       Hauptnr:=Char(Regs.AL shr 4 + 48);
```

```
[695]   448:        Nebennr:=Char(Regs.AL and $ØF + 48);
[8N5]   449:        EMS_Version:=Hauptnr+'.'+Nebennr;
[4MØ]   45Ø:      end
[AH3]   451:    else EMS_Version:='';
[ZNØ]   452: end;
[ØØØ]   453:
[9V5]   454: function EMS_Seiten_gesamt:Word;
[LT6]   455: {Ermittelt die Gesamtzahl der EMS-Seiten
[Y82]   456:  (je 16 KByte)}
[MUØ]   457: begin
[TX1]   458:   Regs.AH:=$42;
[ZC2]   459:   Intr($67,Regs);
[5Ø5]   46Ø:   EMS_Seiten_gesamt:=Regs.DX;
[ZNØ]   461: end;
[ØØØ]   462:
[AF5]   463: function EMS_Seiten_frei:Word;
[VF6]   464: {Ermittelt die noch freie Zahl von EMS-
[W93]   465:  Seiten (je 16 KByte)}
[MUØ]   466: begin
[TX1]   467:   Regs.AH:=$42;
[ZC2]   468:   Intr($67,Regs);
[1F4]   469:   EMS_Seiten_frei:=Regs.BX;
[ZNØ]   47Ø: end;
[ØØØ]   471:
[785]   472: function EMS_Page_Frame:Word;
[AN6]   473: {Ermittelt die Segmentadresse des Page-
[TC4]   474:  Frame, d.h. die Adresse des
[XF6]   475:  Segmentes im Speicher bis 1 MByte, in das
[EF5]   476:  EMS-Speicher eingeblendet wird.}
[MUØ]   477: begin
[SX1]   478:   Regs.AH:=$41;
[ZC2]   479:   Intr($67,Regs);
[B74]   48Ø:   EMS_Page_Frame:=Regs.BX
[ZNØ]   481: end;
[ØØØ]   482:
[BN4]   483: function EXT_Memory:Word;
[YY7]   484: {Ermittelt die Größe des Extended-Memory}
[MUØ]   485: begin
[FY1]   486:   Regs.AH:=$88;
[9C2]   487:   Intr($15,Regs);
[2L3]   488:   EXT_Memory:=Regs.AX
[ZNØ]   489: end;
[ØØØ]   49Ø:
[KNØ]   491: end.
```

Listing 1. Eine Sammlung nützlicher Funktionen: die Unit »Memory« im Listing »memory.pas«

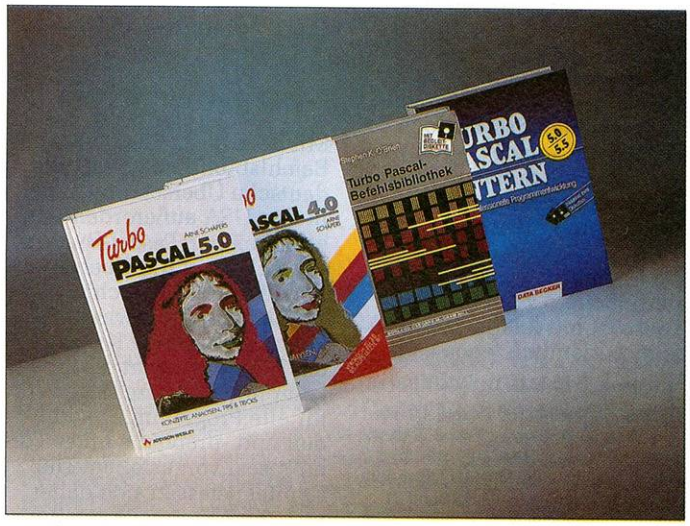
```
[P7Ø]     1: {
[U94]     2: ****************************************
[U44]     3: *                MCBLISTE                *
[U94]     4: ****************************************
[AD6]     5: * Aufgabe: Das Programm analysiert die   *
[YK6]     6: * verkettete Liste der Memory Control    *
[LA6]     7: * Blocks und gibt sie aus. Es bestimmt   *
[PR6]     8: * die Gesamtkapazität und die Restkapa-  *
[HZ4]     9: * zität des RAM.                         *
[U94]    1Ø: ****************************************
[BU5]    11: * Autor: Dr. Joachim Jäger              *
[U94]    12: ****************************************
[U7Ø]    13: }
[113]    14: program MCBListe;
[ØØØ]    15:
[ZG3]    16: uses Crt, Dos, Memory;
[ØØØ]    17:
[W26]    18: type IntArray=array[Ø..255] of word;
[ØØØ]    19:
[ZS2]    2Ø: var MCB: MCBTyp;
[Y44]    21:     L:array[1..1ØØ] of MCBTyp;
[ØC2]    22:     INT:IntArray;
[CS5]    23:     Segment,i,j,Nr,Anzahl,Start:Word;
[N75]    24:     Geskap,Restkap,Maxkap:LongInt;
[ØØØ]    25:
[717]    26: function Ist_Name(var s:String):Boolean;
[AC7]    27: {testet, ob ein String einen Programmnamen
[TT2]    28:  darstellen kann}
[ZU4]    29: const VerbZeich:set of Char=
[AP3]    3Ø: {verbotene Zeichen}
[Z38]    31: [#Ø..#$1F,'.','"','/','\','[',']',':','|','
             <','>','+','=',',',#123..#254];
[VY1]    32: var i:Byte;
[MUØ]    33: begin
[8U3]    34:   for i:=1 to length(s) do
[9A3]    35:   if s[i] in VerbZeich
[ZN1]    36:   then begin
[PV2]    37:          Ist_Name:=false;
[ASØ]    38:          exit
[4MØ]    39:        end
[AF3]    4Ø:   else Ist_Name:=true
[ZNØ]    41: end;
[ØØØ]    42:
[M5B]    43: procedure ReadBlock(Segm,Offs:Word;Anzahl:B
             yte;var Block:String);
[YY6]    44: {überträgt einen Speicherblock (max 255
[NN3]    45:  Zeichen) in String}
[MUØ]    46: begin
[SU4]    47:   if Anzahl>255 then Anzahl:=255;
[YL3]    48:   Block[Ø]:=Chr(Anzahl);
[EN4]    49: {Stringlänge eintragen}
[4E6]    5Ø:   Move(Mem[Segm:Offs],Block[1],Anzahl)
```

```
[ZNØ]    51: end;
[ØØØ]    52:
[2H6]    53: function EnvirName(MCB:MCBTyp):String;
[US6]    54: {ermittelt den Namen eines Programms, zu
[WC4]    55:  dem die Umgebung gehört}
[C12]    56: var p:Word;
[633]    57:     Name,Block:String;
[MUØ]    58: begin
[E26]    59: {sucht nach Zeichenfolge ØØ ØØ Ø1 ØØ am
[6H6]    6Ø:  Ende eines Umgebunsgbereichs und liest
[HØ5]    61:  den dahinter stehenden Namen}
[777]    62: ReadBlock(MCB.MCBSeg+1,Ø,MCB.Laenge*16,Bloc
             k);
[YE3]    63:   p:=Pos(#Ø+#Ø+#1+#Ø,Block);
[ØUØ]    64:   if p>Ø
[ZN1]    65:   then begin
[G31]    66:          p:=p+4;
[W91]    67:          Name:='';
[P31]    68:          repeat
[4G3]    69:            Name:=Name+Block[p];
[731]    7Ø:            p:=p+1;
[BK4]    71:          until (Block[p]=#Ø) or (p=256)
[4MØ]    72:        end
[BF5]    73:   else Name:='Kommandoprozessor';
[6L5]    74: {falls kein Name gefunden wird}
[BR2]    75:   EnvirName:=Name
[ZNØ]    76: end;
[ØØØ]    77:
[CC6]    78: function MCBName(MCB:MCBTyp):String;
[FU6]    79: {ermittelt den Namen eines Programms aus
[8U3]    8Ø:  dem zugehörigen MCB}
[YØ2]    81: var i:Word;
[U52]    82:     Name:String;
[MUØ]    83: begin
[VL6]    84: {übernimmt Byte 8 bis 15 des MCB ggf. als
[KYØ]    85:  Name}
[W91]    86:   Name:='';
[VPØ]    87:   i:=1;
[P31]    88:   repeat
[EW5]    89:     Name:=Name+Chr(Mem[MCB.MCBSeg:i+7]);
[951]    9Ø:     inc(i);
[N76]    91:   until (i=9) or (Mem[MCB.MCBSeg:i+7]=$ØØ);
[UT3]    92:   if Ist_Name(Name)=true
[W13]    93:   then MCBName:=Name
[UH2]    94:   else MCBName:=''
[ZNØ]    95: end;
[ØØØ]    96:
[TD6]    97: function BatchName(MCB:MCBTyp):String;
[AS6]    98: {ermittelt den Namen des Batch, zu dem ein
[N92]    99:  MCB gehört}
[YØ2]   1ØØ: var i:Word;
[U52]   1Ø1:     Name:String;
[MUØ]   1Ø2: begin
[W91]   1Ø3:   Name:='';
[VT6]   1Ø4:   if DOS_Version<'3.3Ø' then i:=$29 else i:
             =$2F;
[V76]   1Ø5: {Name beginnt bei Byte 19H bis DOS 3.2 und
[PT3]   1Ø6:  bei Byte 2FH ab DOS 3.3}
[P31]   1Ø7:   repeat
[ZK5]   1Ø8:     Name:=Name+Chr(Mem[MCB.MCBSeg:i]);
[951]   1Ø9:     inc(i);
[EH4]   11Ø:   until Mem[MCB.MCBSeg:i]=Ø;
[WM2]   111:   BatchName:=Name
[ZNØ]   112: end;
[ØØØ]   113:
[TE6]   114: function Scan_Name(s:String):String;
[756]   115: {zieht Name aus Pfad & Name heraus}
[C12]   116: var p:Word;
[9W6]   117: begin {sucht letztes Auftreten von \ und
[MP3]   118:       übernimmt den Rest}
[P31]   119:   repeat
[312]   12Ø:     p:=Pos('\',s);
[L64]   121:     s:=Copy(s,p+1,Length(s)-p)
[LN1]   122:   until p=Ø;
[T12]   123:   Scan_Name:=s
[ZNØ]   124: end;
[ØØØ]   125:
[NR6]   126: function Test_PSP(MCB:MCBTyp):Boolean;
[SV6]   127: {Test, ob MCB zu einem Programm gehört}
[ZE2]   128: var Code:Word;
[MUØ]   129: begin
[E54]   13Ø:   if MCB.PSPSeg=MCB.MCBSeg+1
[TG7]   131: {zeigt PSP-Zeiger auf nächsten Paragraph ?}
[S7A]   132:   then if (MemW[MCB.PSPSeg:Ø]=$2ØCD) or (Me
             mW[MCB.PSPSeg:Ø]=$27CD)
[AH3]   133:        then Test_PSP:=true
[TL3]   134:        else Test_PSP:=false
[FR3]   135:   else Test_PSP:=false;
[ZNØ]   136: end;
[ØØØ]   137:
[GP6]   138: function Test_Env(MCB:MCBTyp):Boolean;
[UH6]   139: {Test, ob MCB zu einem Umgebungsbereich
[AN1]   14Ø:  gehört}
[VY1]   141: var i:Byte;
[8L1]   142:     s:String;
[MUØ]   143: begin
[Ø16]   144: {beginnt Block mit COMSPEC= oder mit
[EA1]   145:  PATH= ?}
[EVØ]   146:     s:='';
```

Listing 2. Das Programm »mcbliste.pas« analysiert Memory-Control-Blocks

```
[KH7]  147:      for i:=Ø to 7 do s:=s+Chr(Mem[MCB.MCBse
           g+1:i]);
[H16]  148: if (s='COMSPEC=') or (Copy(s,1,5)='PATH=')
[5G3]  149:      then Test_Env:=true
[MK3]  15Ø:      else Test_Env:=false
[ZNØ]  151: end;
[ØØØ]  152:
[UØ7]  153: function Test_Batch(MCB:MCBTyp):Boolean;
[SB7]  154: {überprüft, ob MCB zu einem Batch gehört}
[MU4]  155: var Block,Test,Name:String;
[264]  156:     p,i,Version,Start:Word;
[MUØ]  157: begin
[VW3]  158:   if Mem[MCB.MCBSeg+1:Ø]<>Ø
[VP4]  159: {Block beginnt nicht mit ØØH}
[2RØ]  16Ø:   then
[MUØ]  161:     begin
[343]  162:       Test_Batch:=false;
[ASØ]  163:       exit
[ZNØ]  164:     end;
[UE3]  165:   Name:=BatchName(MCB);
[LD5]  166:   Test:=Copy(Name,Length(Name)-3,4);
[UM4]  167: {.BAT vom Namen abtrennen}
[ZP4]  168:   if Test<>'.BAT' { und testen}
[FV3]  169:   then Test_Batch:=false
[9Y3]  17Ø:   else Test_Batch:=true;
[ZNØ]  171: end;
[ØØØ]  172:
[SL6]  173: procedure MCB_Analyse(var MCB:MCBTyp);
[MUØ]  174: begin
[YY1]  175:   with MCB do
[MUØ]  176:   begin
[L71]  177:     TYP:='';
[W91]  178:     Name:='';
[HD2]  179:     if MCB.PSPSeg=Ø
[VT4]  18Ø: {dann ist Block freigegeben}
[2RØ]  181:     then
[MUØ]  182:       begin
[8A2]  183:         Typ:='frei     ';
[ØWØ]  184:         exit;
[ZNØ]  185:       end;
[WM3]  186:     if Test_PSP(MCB)=true
[ZN1]  187:     then begin
[8U2]  188:            Typ:='Programm ';
[X43]  189:            Name:=MCBName(MCB);
[WV4]  19Ø:            EnvSeg:=MemW[MCB.PSPSeg:$2C]
[234]  191: {Segment der Umgebung}
[4MØ]  192:          end
[DH4]  193:     else if Test_Env(MCB)=true
[ZN1]  194:          then begin
[5S2]  195:                 Typ:='Umgebung ';
[VD3]  196:                 Name:=EnvirName(MCB)
[4MØ]  197:               end
[MR4]  198:          else if Test_Batch(MCB)=true
[ZN1]  199:               then begin
[LE2]  2ØØ:                      Typ:='Batch    ';
[4B3]  2Ø1:                      Name:=BatchName(MCB)
[4MØ]  2Ø2:                    end
[WP1]  2Ø3:               else begin
[HZ2]  2Ø4:                      Typ:='unbekannt';
[661]  2Ø5:                      Name:=''
[ZNØ]  2Ø6:                    end;
[4MØ]  2Ø7:   end
[ZNØ]  2Ø8: end;
[ØØØ]  2Ø9:
[Ø6C]  21Ø: procedure MCB_Interrupt(var INT: IntArray;
           var Anz: Word; MCB: MCBTyp);
[4G5]  211: {ermittelt die Interrupts, deren
[ST6]  212:  Segmentadresse in der Interrupt-Vektor-
[A97]  213:  tabelle mit der PSP-Adresse des Programms
[LV2]  214:  übereinstimmt}
[K72]  215: var n,i:Word;
[D52]  216:     Vektor:Word;
[MUØ]  217: begin
[WF3]  218:   if MCB.Typ='Programm '
[2RØ]  219:   then
[MUØ]  22Ø:     begin
[8Ø2]  221:       Anz:=Ø; n:=Ø;
[9U2]  222:       for i:=Ø to 255 do
[MUØ]  223:         begin
[BF4]  224:           if MemW[Ø:i*4+2]=MCB.PSPSeg
[AK3]  225: {Segment des Intr. i}
[2RØ]  226:           then
[MUØ]  227:             begin
[131]  228:               n:=n+1;
[HM1]  229:               INT[n]:=i
[4MØ]  23Ø:             end
[ZNØ]  231:         end;
[K31]  232:       Anz:=n
[4MØ]  233:     end
[ZNØ]  234: end;
[ØØØ]  235:
[3H6]  236: procedure MCB_Ausgabe(var MCB:MCBTyp);
[N25]  237: {gibt die Daten eines MCB aus}
[YS2]  238: var Anz,i:word;
[MUØ]  239: begin
[5D9]  24Ø:   Write(Hex(MCB.MCBSeg),'    ',MCB.Typ,'  '
           ,Hex(MCB.PSPSeg),'    ');
[WF3]  241:   if MCB.Typ='Programm '
[RZ4]  242:   then Write(Hex(MCB.ENVSeg),'    ')
[823]  243:   else Write('        ');
[4R5]  244:   Write(Hex(MCB.Laenge),'   ',MCB.Name);
[WF3]  245:   if MCB.Typ='Programm '
[2RØ]  246:   then
[MUØ]  247:     begin
[3P4]  248:       MCB_Interrupt(INT,Anz,MCB);
[R71]  249:       if Anz>Ø
[2RØ]  25Ø:       then
[MUØ]  251:         begin
[YL1]  252:           Write(' (');
[K28]  253:           for i:=1 to Anz do Write(copy(hex
           (INT[i]),3,2),' ');
[SV1]  254:           Writeln(')')
[4MØ]  255:         end
[A62]  256:       else Writeln
[4MØ]  257:     end
[ØA2]  258:   else Writeln;
[ZNØ]  259: end;
[ØØØ]  26Ø:
[MP2]  261: procedure Kopf;
[VV5]  262: {Überschrift auf dem Bildschirm}
[YØ2]  263: var i:Word;
[MUØ]  264: begin
[327]  265:   Titel('Analyse der Memory-Control-Blocks'
           );
[779]  266:   Writeln('MCB-ADR Typ        PSP-ADR Env-A
           DR Länge  Name');
[FB3]  267:   Writeln(fill(79,'-'));
[ZNØ]  268: end;
[ØØØ]  269:
[VX3]  27Ø: begin  {Hauptprogramm}
[ZVØ]  271:   Kopf;
[7F3]  272:   {Einlesen des 1. MCB}
[SX2]  273:   GetFirstMCB(MCB);
[UZ2]  274:   MCB.Typ:='BIOS     ';
[NY1]  275:   MCB.Name:='';
[1N1]  276:   L[1]:=MCB;
[ØØØ]  277:
[115]  278:   {Einlesen der folgenden MCBs}
[VPØ]  279:   i:=1;
[8G3]  28Ø:   Restkap:=Ø;Maxkap:=Ø;
[P31]  281:   repeat
[951]  282:     Inc(i);
[914]  283:     GetNextMCB(L[i-1],L[i]);
[U43]  284:     MCB_Analyse(L[i]);
[EG3]  285:     if L[i].Typ='frei     '
[2RØ]  286:     then
[MUØ]  287:       begin
[945]  288:         Restkap:=Restkap+L[i].Laenge;
[4V7]  289:         if L[i].Laenge>Maxkap then Maxkap:=
           L[i].Laenge
[4MØ]  29Ø:       end
[HH3]  291:   until L[i].Ident='Z';
[HØ6]  292: { ... bis letzter Block gelesen ist}
[8G6]  293:   Anzahl:=i; {Anzahl der MCBs speichern}
[8F5]  294:   Restkap:=Restkap+L[i].Laenge+1;
[P25]  295: {Platz von MCBListe addieren}
[G28]  296:   if L[i].Laenge>Maxkap then Maxkap:=L[i].L
           aenge;
[ØØØ]  297:
[K86]  298: {Programmname ggf. über Beziehung zu
[KR3]  299:  Umgebung berichtigen}
[9E3]  3ØØ:   for i:=1 to Anzahl do
[MUØ]  3Ø1:   begin
[YV6]  3Ø2:     if (L[i].Typ='Programm ') and (L[i].Nam
           e='')
[2RØ]  3Ø3:     then
[MUØ]  3Ø4:       begin
[XPØ]  3Ø5:         j:=1;
[P31]  3Ø6:         repeat
[Ø89]  3Ø7:   if (L[j].Typ='Umgebung ') and (L[j].PSPSe
           g=L[i].MCBSeg+1)
[LX6]  3Ø8:   and (Memw[L[i].PSPSeg:$2c]=L[j].MCBSeg+1)
[U86]  3Ø9:   then L[i].Name:=Scan_Name(L[j].Name);
[B51]  31Ø:          inc(j);
[YL2]  311:         until j > Anzahl
[4MØ]  312:       end
[ZNØ]  313:   end;
[ØØØ]  314:
[VU2]  315:   {Daten ausgeben}
[WV1]  316:   GotoXY(1,5);
[9E3]  317:   for i:=1 to Anzahl do
[MUØ]  318:   begin
[F43]  319:     MCB_Ausgabe(L[i]);
[YH5]  32Ø:     if WhereY=24 then Seitenschaltung
[ZNØ]  321:   end;
[XF1]  322:   Writeln;
[ER5]  323:   if WhereY>2Ø then Seitenschaltung;
[TZ2]  324:   With L[Anzahl] do
[MUØ]  325:     begin
[1V6]  326:       GesKap:=MCBSeg+Laenge+1;GesKap:=GesKa
           p*16;
[UK6]  327:       RestKap:=RestKap*16; Maxkap:=Maxkap*1
           6;
[GH7]  328:       Writeln('Gesamtkapazität:',GesKap,' B
           ytes');
[ØG7]  329:       Writeln('Restkapazität  :',RestKap,'
           Bytes');
[Z2A]  33Ø:       Writeln('Größter freier Speicherblock
           : ',Maxkap,' Bytes');
[4MØ]  331:     end
[KNØ]  332: end.
```

Listing 2. Das Programm »mcbliste.pas« analysiert Memory-Control-Blocks



Profiwissen für Turbo Pascal

Der PC und die Programmiersprache Turbo Pascal sind schon seit einiger Zeit zu einem festen Bestandteil der Mikrocomputeranwendungen geworden. Turbo Pascal hat bei seiner Veröffentlichung auf dem Gebiet der Personalcomputer neue Maßstäbe gesetzt und sich schnell etabliert. Die Sprache ist leicht anzuwenden und besitzt in den späteren Versionen einen umfangreichen Befehlssatz.

Das vorliegende »Turbo-Pascal Profibuch« von Martin Althaus beschäftigt sich mit dem Pascal-Dialekt und dem damit verbundenen Grundlagenwissen für die professionelle Programmierung. Nach einer kurzen Einführung in das Gebiet der Speicher- und Variablenverwaltung gibt der Autor eine Anleitung für den direkten Hardware-Zugriff. Die einzelnen Funktionen werden in der Reihenfolge vorgestellt, in der sie in der Praxis vorkommen. Obwohl vor allem der Befehlswortschatz der neuen Versionen von Turbo Pascal sehr umfangreich ist und die angeführten Beispiele nicht unstrukturiert erscheinen sollen, behandelt der Verfasser alle wichtigen Themen genau und ausführlich.

Das Buch ist hauptsächlich in die zwei größeren Themenkomplexe Systeminformationen und professionelle Programmentwicklung unterteilt. Im ersten Teil beschäftigt sich der Autor mit dem internen Aufbau der Programmiersprache.

Der zweite Teil des Buches beinhaltet Ausführungen über die fortgeschrittene Programmierung, mit deren Hilfe man problemorientierte und bedienungsfreundliche Programme realisieren kann. Die praxisorientierten Anwendungsbeispiele zu den Themen objektorientierte Programmierung, Fenstertechnik, Pulldown-Menüs, speicherresidente Programme, Grafik für Fortgeschrittene und Speicherverwaltung erleichtern das Verständnis für die vermittelten Lerninhalte.

Trotz des hohen Preises von 98 Mark (eine Diskette befindet sich im Lieferumfang) gehört das »Turbo-Pascal Profibuch« zu den besten programmspezifischen Begleitmaterialien, die auf dem Markt verfügbar sind. Die detaillierten Ausführungen machen die Beschäftigung mit den einzelnen Kapiteln zu einer Entdeckungsreise.

(Markus Zietlow/ks)

Martin Althaus, »Das Turbo-Pascal Profibuch«, Sybex-Verlag, ISBN 3-88745-776-5, zirka 800 Seiten, 98 Mark

Turbo Pascal und dBase III Plus

Die Programmiersprache Turbo Pascal und das Datenbanksystem dBase III Plus gehören zu den meistverkauften Programmpaketen auf dem Gebiet der elektronischen Datenverarbeitung. Bisher ist es leider nicht möglich gewesen, die Vorteile der einzelnen Programme zu verbinden.

Dieses Ziel verfolgt und erreicht »Tools für dBase III Plus: Turbo-Pascal«. Denn obwohl die veröffentlichte Schnittstelle zwischen Turbo Pascal und dBase III Plus überzeugen kann, stellt das Buch mit der beiliegenden Diskette im Grunde genommen eine arithmetische Standardbibliothek dar. Die trigonometrischen sowie statistischen Funktionen und die Anwendungsbeispiele aus der Finanzmathematik eignen sich trotz des gelungenen Aufbaus für Hobbyprogrammierer weniger.

»Tools für dBase II Plus: Turbo-Pascal« ist eine Arithmetikbibliothek für das dBase-System, die einen umfangreichen Befehlswortschatz zur Verfügung stellt. Da das Einsatzgebiet aufgrund der stark spezifizierten Funktionen eingeschränkt ist, eignet sich das Buch hauptsächlich für Anwendungen auf dem wissenschaftlich-technischen Gebiet. Obwohl sich das Werk an einen spezifischen Anwenderkreis wendet, ist der Preis von 98 Mark trotz der mitgelieferten Diskette etwas zu hoch.

(Markus Zietlow/ks)

J.T. Cooper & Company, »Tools für dBase III Plus: Turbo-Pascal«, Verlag Markt & Technik AG, ISBN 3-89090-478-5, zirka 300 Seiten, 98 Mark

Ausführliche Turbo-Pascal-Referenz

Michael Tischer ist vielen PC-Kennern sicher ein vertrauter Name. Stammt aus seiner Feder mit »PC Intern« doch ein bekanntes Buch zur PC-Programmierung. In seinem Buch »Turbo Pascal Intern« hat sich Data-Becker-Autor Tischer den Details der Programmiersprache Turbo Pascal gewidmet. Auf knapp 900 Seiten stellt Tischer dar, wie sich professionelle Programmiermethoden in Turbo Pascal umsetzen lassen. Das Spektrum reicht von der Beschreibung interner Details von Turbo Pascal bis hin zur Realisierung von TSR- und Multitasking-Programmen. Alle Listings des Buches werden als Quelltext auf einer Diskette mitgeliefert. Unter anderem befinden sich darunter eine komplette SAA-Bibliothek, Treiber für den Mauszugriff und die Programmierung von Extended Memory sowie Module zum Erstellen von TSR-Programmen.

Unbedingte Voraussetzung für das Verständnis des Buches sind umfassende Kenntnisse der Programmierung des Betriebssystems MS-DOS und der Maschinensprache des 8086/80286. Wer den PC allgemein und Turbo Pascal speziell bereits sehr gut kennt, für den hält das Buch einige eindrucksvolle Einblicke und gute Tricks bereit.

Obwohl »Turbo Pascal Intern« mit 98 Mark nicht gerade preiswert ist, kann es jedem Turbo-Pascal-Experten empfohlen werden, der noch gründlicher mit seiner Programmiersprache arbeiten möchte. Wer jedoch eine gemütliche Bettlektüre sucht, für den ist das Buch sicher kein guter Kauf: Sein Verständnis verlangt konzentriertes Lesen.

(Martin Althaus/ks)

Michael Tischer, »Turbo Pascal Intern«, Data Becker, ISBN 3-89011-374-5, zirka 900 Seiten, 98 Mark

Die Arbeitsweise von Turbo Pascal durchleuchtet

Mit den beiden Bänden »Turbo Pascal 4.0/5.0 – Konzepte, Analysen, Tips & Tricks« gewährt der Autor Arne Schäpers tiefe Einblicke in die Arbeitsweise des Turbo-Pascal-Compilers. Als Autor der deutschen Originaldokumentation zu Turbo Pascal bestätigt Schäpers seit Jahren, daß Computerbücher sowohl verständlich als auch sachlich ansprechend verfaßt werden können. Mit seinem zweibändigen Buch »Konzepte, Analysen, Tips & Tricks« demonstriert der Autor, daß sich ein verständlicher Schreibstil auch bei komplizierten Themenstellungen realisieren läßt.

Welchen Code Turbo Pascal bei der Compilierung von Programmen erzeugt, wie die Routinen der Laufzeitbibliothek in Pascal-Programmen aktiviert werden und auf welche Weise Turbo Pascal den Speicher organisiert, sind nur einige der angesprochenen Themen. In Band 1 (zirka 500 Seiten) widmet sich der Autor einigen internen Details von Turbo Pascal, dem arithmetischen Coprozessor, speicherresidenten Programmen, dem »Innenleben« der Unit »graph« und dem Gebiet »Gerätetreiber«. Band 2 (zirka 400 Seiten) untersucht Zugriffsverfahren auf EMS-Speicher, die Möglichkeiten von Overlays und der Programmierung der seriellen Schnittstelle. Für Pascal-Interessierte, die ihrem Com-

Impressum

Herausgeber
Christian Widuch
Leitung des Redaktionsbüros
Volker Everts (ev), Andreas Hagedorn (hg)
Chefredakteurin
Eva-Maria Hierlmeier (hi)
Leitende Redakteure
Thomas Röder (tr), Johann Zieger (zi)
Redaktion
Alois Kneisle (al), Benjamin Kleinke (bk),
Dieter Mayer (dm), Werner Niedermeier (wn),
Andrea Scheding (as), Karsten Schramm (ks)
Schlußredaktion
Maria Mollenhauer, Susanne Raffin
Redaktionsassistenz
Claudia Beer, Susanne Bienert, Dorothea Lehner
Koordination
Stefan Ritter
Produktionsleitung
Gerd Köberich
Bereichsleitung
Claudia Ebbrecht (Fotosatz/Lektorat)
Margarete Schenk, Helmut Skoupy (Montage/Reprografie)
Layout
Patricia Reifenhausen
Fotografie
Heike Reifer
Fotosatz
Silvia Führer
Lektorat
Dagmar Wilhelm-Ballhaus
Montage/Reprografie
Manuela Eska, Andrea Gundlach
Werbegestaltung
Mohamed Hawa, Petra Küch
Anzeigenverkaufsleitung
Wolfgang Brill
Anzeigenverkauf für PLZ 1,4,5
Gerlinde Rachow (0 56 51) 8 09 - 3 90
Sylvia Stephani (0 56 51) 8 09 - 3 80
Karina Ehrlich (0 56 51) 8 09 - 3 71
Bernd Heckmann (0 56 51) 8 09 - 3 81
Anzeigenverkauf für PLZ 2 + 3
DMV-Verlagsbüro Hamburg
Ohlsdorfer Straße 34, 2000 Hamburg 60
Sylvia Ehrenpfordt, Tel. (0 40) 46 12 33, Telefax: (0 40) 47 43 10
Anzeigenverkauf für PLZ 6 – 8
DMV-Verlagsbüro München
Zaunkönigweg 2c, 8000 München 82,
Telefon: (0 89) 4 39 10 87, Telefax: (0 89) 4 39 10 80
Leitung: Britta Fiebig
Anzeigenverkauf: Monika Schöbel, Peter Schätzle, Michael Hofmann
Anzeigenverwaltung und Disposition
Andrea Giese, Karina Ehrlich, Beate Kranz, Christina Füllgraf
Anschrift Redaktion:
Redaktionsbüro Everts & Hagedorn
Redaktion DOS International,
Hans-Stießberger-Str. 3, 8013 Haar/München
Telefon: (0 89) 46 10 01 -0
Telefax: (0 89) 46 10 01-90
Anschrift Verlag:
DMV Daten und Medien-Verlag
Widuch GmbH & Co. KG
Fuldaer Straße 6, 3440 Eschwege
Telefon: (0 56 51) 8 09-0
Telefax: (0 56 51) 80 9-3 33

Vertrieb
Verlagsunion Erich Pabel-Arthur Moewig KG (VPM)
Friedrich-Bergius-Straße 20
6200 Wiesbaden
Druck
Druckerei Jungfer, 3420 Herzberg
Bezugspreise
DOS Extra erscheint viermal jährlich.
Einzelpreis DM 18, –, 145 öS, 18, – sfr., Fmk 55, –
Abonnementpreise
Die Preise verstehen sich grundsätzlich einschließlich Porto und Verpackung.
Inland: Bezugspreis 4 Ausgaben: DM 60, –
Europäisches Ausland: Bezugspreis 4 Ausgaben: DM 70, –
Außereuropäisches Ausland: Bezugspreis 4 Ausgaben: DM 80, –
Bankverbindungen:
Postscheck Frankfurt/M: Kto.-Nr.: 23043-608
Raiffeisenbank Eschwege:
BLZ: 522 603 85, Kto.-Nr.: 245 7008
Die Abonnementbestellung kann innerhalb einer Woche nach Auftrag beim DMV-Verlag, Postfach 250, 3440 Eschwege, schriftlich widerrufen werden. Zur Wahrung der Frist reicht der Poststempel. Das Abonnement verlängert sich automatisch um 4 Ausgaben, wenn es nicht mindestens 6 Wochen vor Ablauf beim Verlag schriftlich gekündigt wird.
Manuskripte, Programme:
bitte an das Redaktionsbüro Everts & Hagedorn, Redaktion DOS International, Hans-Stießberger-Straße 3, 8013 Haar/München, senden.
Für unverlangt eingesandte Manuskripte und Datenträger sowie Fotos übernehmen der Verlag und die Redaktion keine Haftung. Die Zustimmung zum Abdruck wird vorausgesetzt.

Das Urheberrecht für veröffentlichte Manuskripte liegt ausschließlich beim Verlag. Nachdruck sowie Vervielfältigung oder sonstige Verwertung von Texten nur mit schriftlicher Genehmigung des Verlages. Für veröffentlichte Programme und Basteleien übernehmen wir weder Gewähr noch Haftung. Schaltungen und verwendete Bezeichnungen müssen nicht frei von gewerblichen Schutzrechten sein.
Namentlich gekennzeichnete Fremdbeiträge geben nicht in jedem Fall die Meinung der Redaktion wieder.

piler auch hinter die Kulissen folgen möchten, lassen sich beide Bände uneingeschränkt empfehlen. Der große Nachteil der Bücher soll jedoch auch nicht verschwiegen werden: Wer die Software in eigenen Programmen nutzen möchte und nicht viel Zeit zum Abtippen hat, kommt nicht umhin, die Diskette zu den Büchern zu bestellen. Diese ist mit knapp 100 Mark jedoch ziemlich teuer.

(Martin Althaus/ks)

Arne Schäpers, »Turbo Pascal 4.0/5.0 – Konzepte, Analysen, Tips & Tricks«, Addison-Wesley, ISBN 3-89319-130-5 (Band 1), ISBN 3-89319-191-7 (Band 2), 58 Mark (Band 1), 58 Mark (Band 2)

Ein ausführlicher Kurs für Turbo Pascal 5.0

Als umfassendes Lehrbuch zu Turbo Pascal ist »Arbeiten mit Turbo Pascal 5.0« konzipiert, das als deutsche Übersetzung eines amerikanischen Originals im Sybex-Verlag erschienen ist. Als ehemaliger Lehrer versteht es der Autor, die Programmiersprache Pascal von Grund auf zu vermitteln. Dabei sind sowohl die Turbo-Pascal-Programmierumgebung als auch die Elemente eines Pascal-Programms Thema des Buches. Neben der Vermittlung der Sprachelemente von Pascal liegt ein Schwerpunkt des Buches in der praxisorientierten Erläuterung fundamentaler Datenstrukturen und Algorithmen. Alle im Buch entwickelten Programme sind auf einer beiliegenden Diskette gespeichert. Bemerkenswert ist, daß viele gerade für Anfänger schwer verständliche Themen wie Zeiger und Rekursionen erstens nicht ausgespart und zweitens verständlich dargestellt werden.

Auf diese Weise ist das Buch für zwei Gruppen von Anwendern geeignet: zum einen für Leser, die sich schrittweise in die Programmierung mit Pascal einarbeiten möchten, und zum anderen als Nachschlagewerk für Programmierer, die mit Pascal bereits erfolgreich arbeiten. Mit einem Preis von 69 Mark für über 620 Seiten liegt »Arbeiten mit Turbo Pascal 5.0« im attraktiven Preisbereich.

Douglas Herget, »Arbeiten mit Turbo Pascal 5.0«, Sybex-Verlag, ISBN 3-88745-579-7, zirka 620 Seiten, 69 Mark

Ein Nachschlagewerk für den Programmierer

Mit dem Buch »Turbo Pascal Befehlsbibliothek« liegt die deutsche Übersetzung eines in den USA außerordentlich erfolgreichen Pascal-Buches vor. Leider ist die Übersetzung stellenweise schlecht geglückt und fehlerhaft. Das zugrundeliegende Konzept des Buches besteht darin, ein komplettes Nachschlagewerk für alle Programmiermethoden und -probleme unter Turbo Pascal vorzustellen. Die Kapitel befassen sich daher mit Themen wie Programmstruktur, Datentypen, Zeiger, Dateien, Rekursionen und Kontrollstrukturen. Aber auch Profithemen wie externe Prozeduren, Inline-Code und Interrupt-Aufrufe werden nicht ausgespart. In der übersetzten Ausgabe ist das Buch für Turbo Pascal in der Version 3.0 verfaßt. Die Änderungen auf die Versionen 4.0 und 5.0 wurden jedoch nicht in den vorhandenen Text eingearbeitet, sondern als Zusatzabschnitte an das Ende des jeweiligen Kapitels gestellt.

Aufgrund der großen konzeptionellen Unterschiede zwischen Turbo Pascal 3.0 und den Nachfolgeversionen ab 4.0 entsteht so die Situation, daß vieles, was zu Beginn eines Kapitels für Turbo Pascal 3.0 gesagt wird, am Schluß des Kapitels für die neueren Versionen wieder revidiert werden muß. Hier hätte dem Buch eine grundsätzliche Überarbeitung aller Kapitel auf die aktuellste Version (Version 5.5 ist im Buch nicht erwähnt) gutgetan. Schade, daß unter solchen Ungenauigkeiten auch eigentlich gut dargestellte Themen wie beispielsweise die Erstellung residenter Programme mit Turbo Pascal, leiden. Mit einem Preis von 96 Mark für ungefähr 800 Seiten ist die »Turbo Pascal Befehlsbibliothek« zudem nicht preiswert. Wer immer noch mit Turbo Pascal in der Version 3.0 programmiert, macht mit dem Buch sicher keinen Fehlkauf.

Ansonsten kann das Werk nicht bedenkenlos empfohlen werden.

(Martin Althaus/ks)

Stephen K. O'Brien, »Turbo Pascal Befehlsbibliothek«, McGraw-Hill, ISBN 3-89028-155-9, zirka 800 Seiten, 96 Mark

»Gesucht und gefunden«

Adreßverwaltung für jedermann, leistungsstark und universell

Adressen praxisgerecht archivieren, sortieren, auflisten, suchen und per Tastendruck in Ihre Texte übernehmen - das ist ConText-Adreßverwaltung. In Zusammenarbeit mit ConText PRO und ConText EASY können Ihre Adressen auch direkt für Serienbriefe verarbeitet werden.

Die ConText-Adreßverwaltung kann aus ASCII-Dateien Adressen importieren, bietet kontextbezogene Hilfe bei allen Funktionen, sucht auch mit Wildcards in allen Feldern, ist wahlweise speicherresident oder als normales Programm installierbar und...

Die ConText-Adreßverwaltung paßt nicht nur zu ConText PRO und ConText EASY, sondern auch zu anderen Anwendungen. Überzeugen Sie sich!

ConText-Adreßverwaltung - durchdacht wie alle ConText-Produkte.

»Ganz nebenbei und aufmerksam«

Schnell und pfiffig mit Warnton, das Ideale auch für Sie!

Eine gute und praktische Idee: Während Ihre Hände und Augen mit der Texteingabe beschäftigt sind, warnt Sie die ConText-Rechtschreibhilfe durch akustische Signale. Sofort und ohne Rücklauf können Sie gleich an der Cursorposition korrigieren: Das spart Zeit und Nerven.

Die ConText-Rechtschreibhilfe arbeitet mit einem auswechselbaren RAM-residenten Lexikon, das um Ihre jeweils neuen Begriffe erweitert wird. Die Lexika können editiert, auf Massenspeicher ausgelagert und automatisch aus Texten für Ihre individuelle Anwendung ganz neu erstellt werden.

Den Fehler-Streß beim Eintippen schaffen Sie übrigens nicht nur für ConText PRO und Context EASY ab - die ConText-Rechtschreibhilfe ist universell einsetzbar und funktioniert sogar auf der DOS-Ebene!

»Gestalten, ansehen und ausdrucken«

Neue Zeichen in ConText PRO und EASY für Ihre individuelle Anwendung!

Sprengen Sie den IBM-ASCII-Zeichensatz! Mit dem ConText-Zeicheneditor können Sie beliebige eigene Zeichen zur WYSIWYG-Bildschirmdarstellung von ConText PRO und ConText EASY erzeugen. Fremdsprachige oder wissenschaftliche Sonderzeichen - von nun ab kein Problem mehr.

Auch an den Ausdruck wurde gedacht! In einem Extra-Druckerzeichen-Editor passen Sie Ihre neuen Zeichen an Ihren Drucker an. Ein raffiniertes Download-Programm und ein residenter Druckertreiber holen zur Darstellung alles aus Ihrem Drucker heraus, was technisch möglich ist.

Zu kompliziert für Sie? Eine ausführliche Anleitung hilft Ihnen über die Anfangsschwierigkeiten hinweg und macht Sie zu einem ConText-Profi - und das alles zu einem absoluten Low-Cost-Preis!

»Einschalten und loslegen«

Textverarbeitung jetzt auch für Sie, zum Einsteigerpreis!

Vergessen Sie die Angst des Anfängers vor dem Computer oder das nervende Wälzen von langen Handbüchern: Mit ConText EASY können Sie »einschalten und loslegen«, denn ConText EASY ist eine speziell auf Einsteiger und Hobbyanwender zugeschnittene Textverarbeitung. ConText EASY verbindet alle bewährten Vorzüge der professionellen Version ConText PRO mit einem eingängigen und praxisgerechten Bedienungskomfort.

Ohne große Vorkenntnisse und Computererfahrung kann jeder, also auch Sie, mit ConText EASY ansprechende Ergebnisse erzielen. Alle Bedienungsfunktionen sind klar in Pulldown-Menüs gegliedert und werden durch Hilfe-Fenster unterstützt. Mit wahlweiser Ansteuerung über eine Maus oder über Tastaturkürzel sowie der Darstellung von Fett-, Breit-, Kursivschrift etc. im WYSIWYG-Prinzip »live« auf dem Bildschirm macht ConText EASY den Einstieg in die Textverarbeitung auf dem Computer auch für Sie zu einem Erfolgserlebnis.

Und wenn Ihnen die Möglichkeiten von ConText EASY nicht mehr genügen sollten, können Sie problemlos auf die professionelle Version ConText PRO umsteigen: Alle Funktionen von ConText EASY finden Sie dort ebenfalls vor, und alle Texte, Layoutmasken und Druckereinstellungen können übernommen werden. Dazu bieten wir Ihnen zu einem günstigen Preis ein Update von ConText EASY auf ConText PRO an. Trauen Sie sich:
ConText EASY - »Einschalten und loslegen«

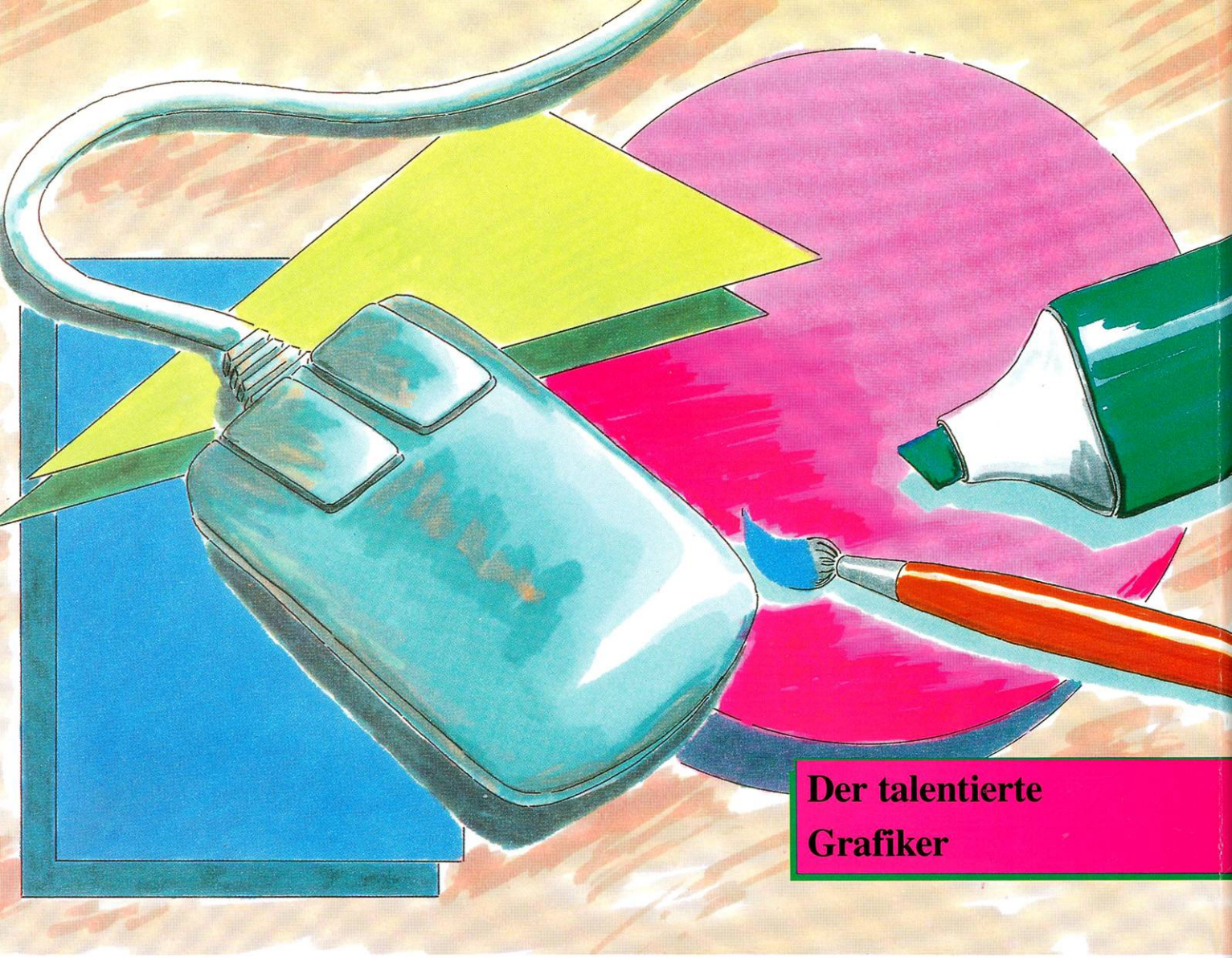
ConText EASY	**DM 99,–***
ConText Adreßverwaltung	**DM 49,–***
ConText Rechtschreibhilfe	**DM 69,–***
ConText Zeicheneditor	**DM 99,–***

* Unverbindliche Preisempfehlung. Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung.

Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag



Für repräsentative Dokumente ConText PRO

Sie werden kaum Erfolg haben, wenn Ihre Texte eintönig und langweilig gestaltet sind. Mit ConText PRO zaubern Sie einfach, schnell und elegant Dokumente, die wirklich ankommen. Breit-, Fett-, Kursivschrift und Unterstreichung sind Ihre vielfältigen Möglichkeiten, die Sie alle in Echtbild-Darstellung auf dem Monitor sehen.

Für Linien und Rahmen, Blockgrafiken, Schemata und einfache Logos bietet Ihnen ConText PRO einen komfortablen Blockgrafik-Editor. Sie wählen zwischen mehreren Linien- und Blockarten und Weiß als »Radiergummi« für Korrekturen. Ecken und Verzweigungen in Ihrer Linienführung paßt der Editor automatisch an.

Ihre Arbeitsgeschwindigkeit steigern Sie durch eine Maus um ein Vielfaches, bei ConText PRO gibt's die Spezialsoftware dafür gratis dazu. Und wenn Ihnen der ASCII-Zeichensatz nicht mehr genügt, rüsten Sie doch preiswert auf: Mit dem ConText-Zeicheneditor gestalten Sie eigene Zeichen für Bildschirmdarstellung und Ausdruck.

ConText PRO Der talentierte Grafiker

CONTEXT PRO

ConText PRO
Das Textprogramm für den professionellen Einsatz **DM 199,–***

DMV-Power-Pack
ConText PRO, ERGO 3.0 und ConText ADREVA **DM 349,–***

* Unverbindliche Preisempfehlung. Unabhängig von der Anzahl der bestellten Programme berechnen wir für das Inland 4,– DM bzw. für das Ausland 6,– DM Porto und Verpackung. Bitte benutzen Sie die Bestellkarte.

DMV-Verlag · Postfach 250 · 3440 Eschwege

DMV Daten- und Medienverlag

Coupon

Ja, ich interessiere mich für ConText PRO! Schicken Sie mir bitte unverbindlich und kostenfrei Ihr Informationsmaterial. Meine Adresse:

Name _____

Straße _____

PLZ _____ Ort _____