

1 AMIGA

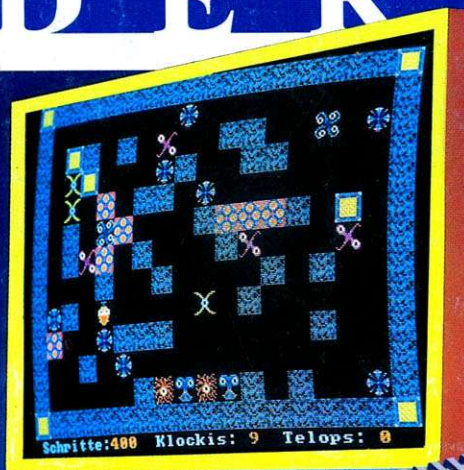
Markt & Technik

DM 16,-
ÖS 120,-/Sfr 16,-
Lit 14000/hfl 21,-/dkr 75,-

SONDERHEFT

Ausführliche Kurse

- *Amiga-Basic richtig nutzen*
- *CLI: Die zweite Ebene des Amigas*



Kurse

Sound- Digitizer im Selbstbau

- *Toller Klang für 50 Mark*

Tricks Tips

30226
Amiga sonderheft
21.00

Tips & Tricks die jeder braucht

- *Software-Hilfen für Einsteiger und Profis*

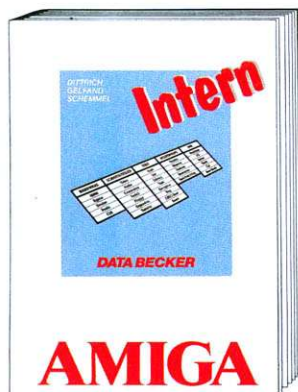
Sensationelles Spiel zum Abtippen

- *Suremosch: Die Herausforderung für jeden Strategie-Fan*



Alle Programme auch auf Diskette erhältlich

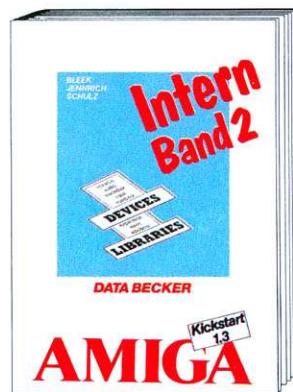
Heute lesen!
was morgen
passiert



Neuaufgabe: die Inside-Story.

Amiga Intern – ein Intern, wie man es von DATA BECKER gewohnt ist. Mit allem, was dazugehört: 68000-Prozessor, CIA, Blitter, Custom-Chips, die Strukturen von EXEC, I/O-Handhabung, Verwaltung der Resources, Erstellung eigener Devices, Exec-Base, resetteste Programme, Autoboot mit der ROMboot. library, DOS-Funktionen, interne DOS-Bibliothek, Aufbau einer Diskette, Programmierung eigener DOS-Handler.

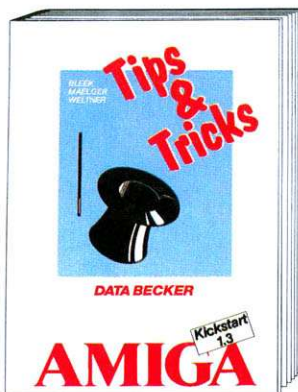
Amiga Intern
Hardcover, ca. 650 Seiten,
DM 69,-
erscheint ca. 9/88



Knallharte Informationen.

Amiga Intern Band 2 – das Buch für jeden aktiven Programmierer, der alle weiterführenden Informationen zu seiner Arbeit schnell und zuverlässig finden will: Ein- und Ausgabe über Devices, Standard-Austausch-Formate und Komprimierungsverfahren, alle Amiga-Libraries mit den dazugehörigen Strukturen, Basis- und Grundstrukturen, Preferences als Datenstruktur, Datenübermittlung von Workbench und CLI, Konventionen im Programmierstil und alles zur Version 1.3.

Amiga Intern Band 2
Hardcover, ca. 700 Seiten,
DM 69,-
erscheint ca. 9/88



Mit Programmen zaubern.

Mit Amiga Tips & Tricks läßt sich die Arbeit mit Ihrem Rechner noch effektiver gestalten – selbstverständlich bereits unter Berücksichtigung des neuen Betriebssystems (Version 1.3): Gestaltung eigener Programme, Tips & Tricks zum AmigaBASIC, Einbinden von Maschinenprogrammen in AmigaBASIC, Einsatz von DOS-Routinen, optimierende Hilfsprogramme für AmigaBASIC-Programme, Tips zur Arbeit mit der Workbench, Aufbau der Icons, die neuen Preferences.

Amiga Tips & Tricks
Hardcover, 555 Seiten, DM 49,-



Runter von der Workbench.

Rein ins AmigaDOS: Umlenken der Ein- und Ausgabe, mit RAM-Disk und CLI arbeiten, STARTUP-Sequenz, Multitasking mit dem CLI, der interne Aufbau der CLI-Befehle, eigene CLI-Befehle programmieren... Das große Buch zu AmigaDOS – mit nützlichen Batch-Dateien und einer Beschreibung der neuen CLI-Befehle und Devices unter V.1.3!

Das große Buch zu AmigaDOS
Hardcover, 370 Seiten
inkl. Diskette, DM 59,-



Alles zur Amiga-Floppy.

Brandaktuell: die zweite, erweiterte Auflage mit allem, was Bezug zur Floppy hat: Workbench, CLI, AmigaBASIC mit verschiedenen Dateitypen, Zugriff aufs Betriebssystem (mit File-Verwaltung, Trackdisk-Device, Boot-Block sowie Checksummen) und direkter Zugriff ohne DOS (MSM- und GCA-Codierung, Track lesen und schreiben, SYNC-Markierung). Dazu einen Floppyspeeder, einen Disketten-Monitor und ein schnelles, leistungsstarkes Kopierprogramm.

Das große Amiga Floppybuch
Hardcover, ca. 400 Seiten
inkl. Diskette, DM 59,-
erscheint ca. 9/88

COUPON!

HIERMIT BESTELLE ICH FÜR MEINEN AMIGA

NAME, VORNAME

STRASSE

ORT

DATA BECKER
Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

zzgl. DM 5,- Versandkosten unabhängig von der bestellten Stückzahl
☐ per Nachnahme ☐ Verrechnungsscheck liegt bei

Premiere

● Endlich ist es soweit — das erste AMIGA-Sonderheft ist auf dem Markt. Viel Arbeit hat es gekostet, aber es hat sich gelohnt. Vor allem für Sie soll es sich lohnen. Deshalb haben wir »Rund um den Amiga« eine Mischung aus tollen Listings, ausführlichen Kursen und vielen Tips & Tricks zusammengestellt.

● Ist das nicht schon im AMIGA-Magazin der Fall? Wozu dann ein Sonderheft? Die Frage läßt sich leicht beantworten. Einige Listings wären sicher irgendwann einmal im AMIGA-Magazin erschienen. Doch leider gibt es das Magazin für Amiga-Fans nur einmal im Monat. Es wäre eine nicht unerhebliche Zeit verstrichen, bis die Listings Ihnen zur Verfügung gestanden hätten. Auch der Platz im Stammheft ist eingeschränkt. Super-Listings, die etwas länger sind, haben oft gar keine Chance, im AMIGA-Magazin veröffentlicht zu werden.

● Ein weiterer Punkt: Schon nach wenigen Monaten waren die AMIGA-Ausgaben von 1987 ausverkauft, die Nachfrage blieb jedoch ungebrochen. Wer sich seit jenen Anfangszeiten des Magazins einen Amiga neu zugelegt hat, kommt an die wichtigen Informationen aus dieser Zeit nicht mehr heran. Wir haben daher grundlegende Kurse aus vielen Ausgaben für Sie neu überarbeitet und in diesem Sonderheft zusammengefaßt.

● Ein ausführlicher Basic-Kurs zeigt Ihnen, wie Sie das ohnehin sehr komfortable Amiga-Basic für Ihre eigenen Programme noch besser nutzen können. Nach einer Einführung in diese Programmiersprache erfahren Sie, wie man Menüs aufbaut, Sounds aktiviert und was sich alles mit »Bobs« und »Sprites« machen läßt. Haben Sie diesen Kurs mitgemacht, dann sind Themen wie »Sprachausgabe beim Amiga« oder »Einbindung der Systembibliothek in eigene Basic-Programme« kein Buch mit sieben Siegeln mehr. Sie lernen vieles kennen, was das Handbuch verschweigt.

● Mit dem CLI-Kurs tauchen Sie in die zweite Ebene des Amiga ein. Was kommt nach der Workbench? Wie erstellt man eine eigene Boot-Diskette? Auf diese und viele weitere Fragen erhalten Sie Antworten in dem zweiten Kurs dieser Ausgabe.

● Der Amiga ist ein fantastischer Grafik-Computer und mit einer Hardware ausgestattet, die geradezu zur Entwicklung von Spielen einlädt. Mit »Suremosch« stellen wir Ihnen ein Spiel zu Verfügung, das keinen Vergleich mit kommerziellen Produkten zu scheuen braucht. Dieses fesselnde Spiel für Leute mit »Köpfchen« fand sofort Begeisterung in unserer Redaktion.

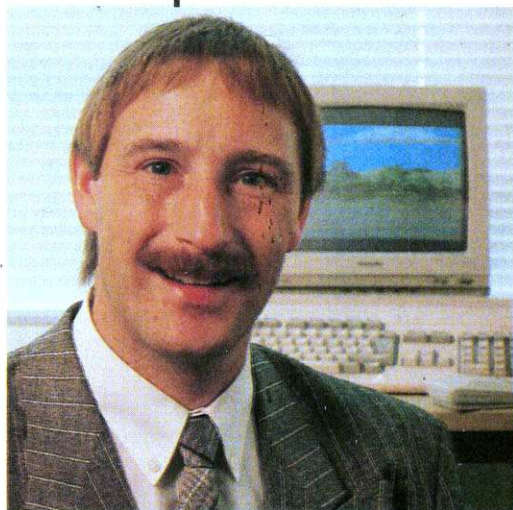
● Machen Sie Ihr eigenes Tonstudio auf. Kein Problem mit dem Digitizer, einer kleinen Hardware-Erweiterung für den Selbstbau. Die Software zu dieser leicht nachzubauenden Schaltung eröffnet Möglichkeiten, von denen Sie schon lange geträumt haben. Endlich lassen sich eigene Sounds digitalisieren und mit dem Computer weiterverarbeiten. Vier getrennte Kanäle bieten den Komfort einer Studio-Tonbandmaschine. Die einzelnen »Spuren« können Sie beliebig mischen. Auch das Verfremden der eingespeisten Klänge ist jetzt ganz einfach.

● Tips und Tricks — für viele ein Begriff mit magischer Anziehungskraft. Ihre Kennzeichen: kurz und interessant. Es ist kaum möglich, alle Kniffe auswendig zu kennen. Oft weiß man gar nicht, daß es zu einem bestimmten Problem eine ganz einfache Lösung gibt. Oder es kommt einem die schwache Erinnerung, irgendwo schon einmal etwas zu dem anstehenden Problem gelesen zu haben, nur — wo? Um diesem Mißstand ein Ende zu bereiten, haben wir viele wichtige Tips & Tricks gesammelt. Ergänzend zu den beiden Kursen finden Sie Tips & Tricks zu den Themen »Basic« und »CLI«. Wer seinen Amiga noch nicht lange besitzt, für den sind die »Tips & Tricks für Einsteiger« eine wahre Fundgrube. Aber auch die Fortgeschrittenen und Profis kommen nicht zu kurz.

● Natürlich wollen wir Ihre Meinung zu dieser Ausgabe kennenlernen. Schreiben Sie uns, was Ihnen gefallen hat und was wir nach Ihrer Ansicht noch besser machen könnten. Ebenso interessiert uns brennend, welche Themen Sie sich für zukünftige AMIGA-Sonderhefte wünschen.

● Also, behalten Sie Ihre Einstellung nicht für sich — wir freuen uns über jeden Tip. Schicken Sie Ihre Zuschriften bitte an die »Redaktion Sonderhefte«.

Ihr Gottfried Knechtel
(Leitender Redakteur)



Gottfried Knechtel

SPIELE

8 SUREMOSCH — DER BRANDNEUE DENKSPORT

Was reizt Sie besonders an einem Spiel? Die gelungene Grafik? Ein tolles Spielprinzip? »Suremosch« wird Sie in jedem Fall überzeugen. Lassen Sie sich in die Welt der »Klockis«, der »Wedongs« und der Energiefallen entführen. Der eingebaute Leveleditor gestattet es Ihnen, eigene Höhlen zu erstellen. Der Spielespaß mit Suremosch nimmt kein Ende.

»Futter« für den Amiga: Public Domain-Software bietet Alternativen zum kommerziellen Angebot.

Grafikprogramme, Spiele, Tools und Anwendungen sind im PD-Angebot zu finden.

Wir stellen die interessantesten Programme vor.

SEITE 13

PD-SOFTWARE

13 PREISWERTE SOFTWARE FÜR DEN AMIGA

»Public Domain« ist ein Schlagwort, das allen Amiga-Besitzern geläufig sein sollte. Preiswerte Software mit teilweise hervorragender Qualität versteckt sich hinter diesem Begriff. Wir erleichtern Ihnen die Auswahl der richtigen Programme und stellen die derzeit interessantesten Vertreter dieser Gattung vor.

ANWENDUNGEN

22 DCCOPY — KOPIERT MIT KOMFORT

Ein Programm sollte jeder Amiga-Anwender besitzen: ein komfortables und schnelles Kopierprogramm. »DCCopy« bietet mehr als nur schnelles Kopieren. Auf bis zu drei Laufwerke gleichzeitig kann der Inhalt der Originaldiskette vervielfältigt werden.

40 DIE BESTEN TAGE IM JAHR

Fühlen Sie sich manchmal, als könnten Sie Bäume ausreißen? Vielleicht zeigt an diesem Tag Ihr Biorhythmus Spitzenwerte. Mit unserem Listing kontrollieren Sie Ihre Leistungskurve. Planen Sie Ihren Erfolg, legen Sie entscheidende Termine einfach auf die vorausberechneten Tage.

BAUANLEITUNG

44 DIGITIZER IM SELBSTBAU — SOFTWARE INKLUSIVE

Erwecken Sie das Klangwunder Amiga zum Leben. Mit unserem Selbstbau-Digitizer und der zugehörigen Software »Digisoft Plus« tauchen Sie in die faszinierende Welt der digitalen Klänge ein. Und das alles kostet weniger als 50 Mark!

TOOLS

80 VERÄNDERUNG DES CLI

Mit dem Programm »InitCli3« verändern Sie Voreinstellungen für das CLI, beispielsweise die Farbgestaltung. Entwerfen Sie das CLI-Fenster nach eigenen Wünschen. Sogar eine bestimmte Schriftart ist mit diesem Hilfsprogramm zu wählen.

84 GENAUE DIVISION

Für verschiedene Berechnungen ist eine hohe Anzahl von Nachkommastellen unbedingt notwendig. Ein Anwendungsfall ist das Erkennen von Perioden. Mit »Division« können Sie bis zu 32000 Stellen hinter dem Komma berechnen!

Dieses Spiel begeistert jeden: »Suremosch« kämpft in der Welt der Energieverschwender gegen die »Klockis«

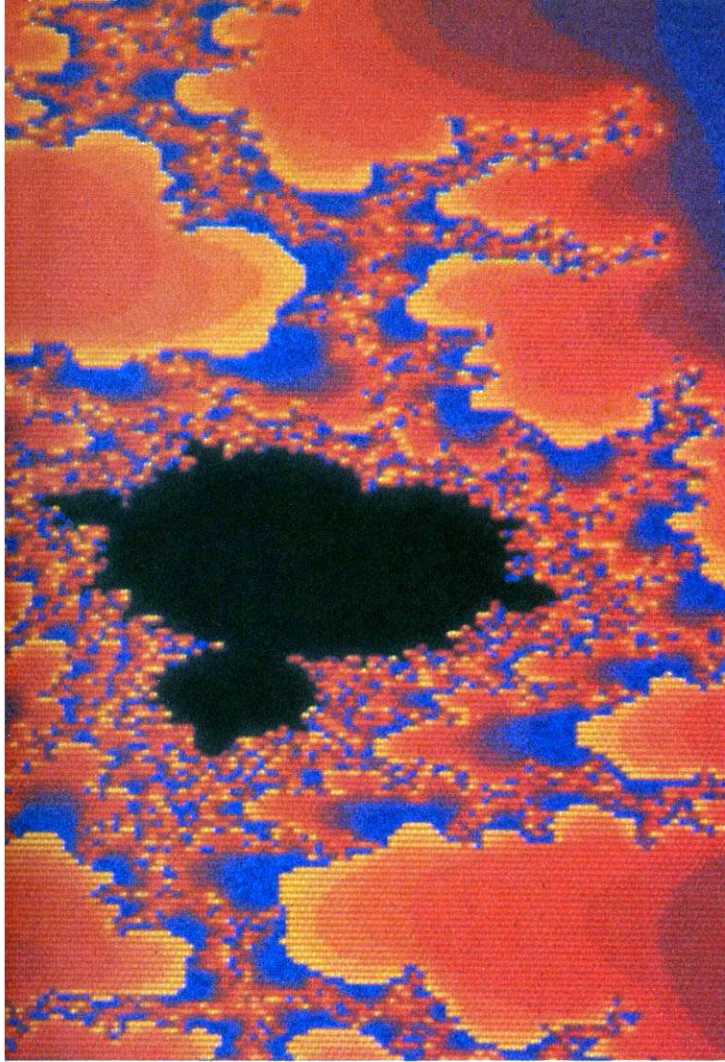
SEITE 8



Die besten Tips & Tricks haben wir für Sie zusammengestellt. Einsteiger oder Profi, Basic-Programmierer oder CLI-Insider: hier findet jeder neue Anregungen

SEITE 131





KURSE

85 BASIC FÜR AUFSTEIGER

Aller Anfang ist schwer — das gilt auch für das Programmieren des Amiga. Die Sprache, die den leichtesten Einstieg verspricht, ist das mitgelieferte Amiga-Basic. Unser ausführlicher Kurs enthüllt die Geheimnisse dieser Programmiersprache und zeigt auch »Feinheiten« wie das Aufrufen von Routinen der Systembibliothek.

114 CLI — DIE ZWEITE HAUT DES AMIGA

Die Bedienung des Amiga über die Workbench ist anfangs sehr bequem. Doch schnell sind hierbei die Grenzen erreicht, es entsteht der Wunsch nach einer effektiveren und leistungsfähigeren Benutzeroberfläche. Das CLI bietet die Lösung. Probleme beim Umgang mit dieser Oberfläche beseitigt dieser Kurs.

TIPS & TRICKS

131 TOLLE TIPS FÜR EINSTEIGER

Nach der ersten Freude über den neuen Amiga folgen häufig auch die ersten Rätsel. Warum ist auf einmal der Bildschirm dunkel, wie stelle ich die Farben ein? Zahlreiche Tips helfen bei der Lösung auftretender Rätsel.

135 DIE HEISSESTEN TIPS FÜR INSIDER

Die besten Tips von Profis für Profis: In dieser Sammlung finden auch Spezialisten zahlreiche nützliche Hinweise.

141 DIE BESTEN TIPS ZUM AMIGA-BASIC

Das Amiga-Basic besitzt interessante Fähigkeiten. Mit unseren Tips erschließen Sie noch mehr Eigenschaften dieser Programmiersprache.

144 TIPS UND TRICKS ZUM CLI

Ergänzend zum CLI-Kurs auf Seite 144 finden Sie hier die besten Tricks, die den Umgang mit der gewöhnungsbedürftigen Oberfläche zum Kinderspiel machen.

BÜCHER

147 ERSTE HILFE

Grundlagen, Amiga-DOS, Programmieren: Literatur zu diesen Themen hilft Ihnen auf die Sprünge.

EINGABEHILFEN

159 CHECKSUMMER

Wie gebe ich Programme ein? Diesen Artikel sollten Sie unbedingt lesen, wenn Sie Programme aus diesem Sonderheft abtippen möchten.

SONSTIGES

3 EDITORIAL

162 IMPRESSUM

Alle Programme aus Artikeln mit einem -Symbol finden Sie auch auf der Programmservice-Diskette zu diesem Sonderheft.

AMIGA BASIC



Der ausführliche Kurs zum Amiga-Basic verschafft Ihnen den nötigen Durchblick

SEITE 85

Mit dem Selbstbau-Digitizer erwecken Sie den Sound-Chip des Amiga zu neuem Leben

SEITE 44

DAS COMBITEC-GEWINNSPIEL!

Beantworten Sie die drei gestellten Fragen und senden Sie die Lösungen auf einer frankierten Postkarte an Combitec. Einsendeschluß ist der 1. Dezember 1988.

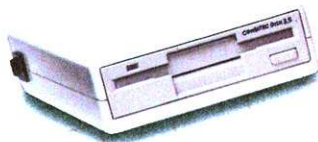
1. Welches Laufwerk wird in der Combitec Disk 3,5 verwendet?
2. Wieviel externe Diskettenlaufwerke sind von dem Amiga 500 ansprechbar?
3. Wie groß ist der serienmäßige Arbeitsspeicher des Amiga 500?

Die eingesandten Karten mit den richtigen Antworten nehmen an der Verlosung teil. Der Rechtsweg ist ausgeschlossen.

1. Preis: 1 Commodore Amiga 500 mit Monitor 1084
2. Preis: 1 Combitec S-RAM 500
3. Preis: 1 Combitec Disk 3,5

Combitec Computer GmbH, Liegnitzer Straße 6-6a, 5810 Witten,
Tel. (0 23 02) 8 80 72

GIB



Combitec Disk 3,5
Diskettenlaufwerk 3 1/2", Busdurchführung
für bis zu 3 ext. Laufwerke, Qualitätslauf-
werk TEAC Typ: FD 135 FN, Anschluß für
Track-Anzeige Typ: Combitec Track-Dis,
abschaltbarer Controller, amigafarbenes
Gehäuse
VK-Preis: DM 325,-



DEINEM



Combitec Clock 77 S(P)
Normalzeituhr DCF 77, serielle Schnittstelle
oder parallele Schnittstelle (Version P),
menügeführte Dateneingabe über Rechner
und Datenspeicherung auf Disketten,
4-Schaltausgänge, Anbindungssoftware für
Amiga 500, Atari, IBM (Preis auf Anfrage)
VK-Preis: DM 359,-
VK-Preis für Rechneranschlußkabel für o. g.
Rechner: DM 24,50

EINE

Weiteres Lieferprogramm:

Combitec 5/3

Steckernetzteil getaktet, 5 Volt,
3 Ampere

VK-Preis: **DM 115,-**

Amiga 500 VK-Preis DM 940,-
Monitor 1084 VK-Preis DM 568,-

Neu auf dem Markt!

Combitec D-RAM 2/4/8 M für A 500
Externe dynamische Speichererweiterung 2 MB, intern
aufrüstbar auf 4 bzw. 8 MB, Einsatz moderner 1 MByte-
DRAM-Module, integrierter DRAM-CONTROLLER (0 Wait-
states), Gehäuse identisch mit Combitec S-RAM 500.
Einführungsangebot: 2 MB DM 1.198,-
4 MB DM 2.298,-
8 MB DM 4.498,-

Combitec das ist Ihr neuer, kreativer Partner, der leistungsstarke Hard- und Software aus einer Hand bietet.
Schnell und zuverlässig!

Combitec führt intelligentes Zubehör rund um Ihren Amiga. Angefangen bei Teac-Qualitätslaufwerken über extrem schnelle Hard-Disk bis zu Ram-Erweiterungen in erstaunlich kompakten Gehäusen und vieles, vieles mehr.

Combitec berät Sie, hilft Ihnen weiter, hat ein offenes Ohr für Ihre Eigenentwicklungen und forscht unentwegt, um Produkte anzubieten, die Ihrem Amiga (oder anderen PC's) die Chance gibt, mehr zu leisten.

Combitec S-RAM 500 (1 M)
Speichererweiterung mit statischen RAM-Bausteinen, 512 K statischer Speicher oder 1 M Byte (Version 1 M), verwendbar als Speichererweiterung und RAM-Disk, Auto-konfigurierend (ab Version 1.3), Anschlußmöglichkeit für separates Netzteil, amigafarbenes, formschönes Gehäuse. Betrieb mit Combitec HD 20 möglich.

VK-Preis:
Combitec S-RAM 500 DM 649,-
Combitec S-RAM 1 M DM 1.048,-

Combitec Track-Disk
Track-Anzeige zur Combitec Disk 3,5, 2-stellige 7-Segmentanzeige, Steckanschluß zur Combitec Disk 3,5, amigafarbenes Gehäuse
VK-Preis: DM 49,-



AMIGA

Combitec HD 20
Harddisk mit Controller, Speicherkapazität 20 MB, Datenübertragung ca. 100 kByte/Sekunde, Autobootung von der Harddisk ab Version 1,3 möglich, eigenes Netzteil, amigafarbenes Gehäuse
VK-Preis: DM 1.215,-



Combitec HD 20 A
Adaptergehäuse für Harddisk zum Anschluß an den Amiga 500, formschönes, flaches, amigafarbenes Gehäuse

Hinweis: Die Harddisk Typ Combitec HD 20 kann nur in Verbindung mit dem Adaptergehäuse Typ Combitec HD 20 A oder einer Combitec Speichererweiterung am Amiga 500 betrieben werden.

VK-Preis: DM 184,-

CHANCE

Achtung! Für jeden Bastler unentbehrlich!

Userport-Modul für AMIGA 500, Anschluß an Expansionsport zum Ansteuern von Triac/Relaismodulen, 2 x 16 I/O-Ports (TTL-Pegel), 2 maskierbare Interrupts, softwareseitig wählbare Startadresse, externe Schaltungsrichtungsmeldung der Portleitung, Anschlußbuchse für externes Netzteil, Gehäuse wie Combitec S-RAM 500

VK-Preis DM 288,-

In Vorbereitung für Amiga 500:
Laufwerk 5,25"

8-Kanal Triac-Ausgangsstufe

8-Kanal Relaisausgangsstufe

Fragen Sie nach Prospektmaterial!

Technische Änderungen vorbehalten

COMBITEC
DIE PARTNER

☎ 02302/88072



CH Denksport

Tüfteln Sie gerne an der Lösung eines schwierigen Problems? Sind Sie begeistert von den Spielen des ungarischen Erfinders Rubik? Dann ist Suremosch für Sie genau das Richtige!

Seit vier Stunden sitze ich bereits vor dem Bildschirm. Der fünfte Versuch, die »Etwas seltsam«-Ebene zu lösen, ist gescheitert. Mein Aktionsradius ist auf 22 Schritte geschrumpft — zu wenig, um die Ebene erfolgreich zu beenden. Widerwillig füge ich mich und gebe auf. »Suremosch« heißt das Spiel, das mich gerade begeistert und meine Kombinationsfähigkeit bis an die Grenze fordert. Noch ein Versuch? Nein, mein Gewissen drängt mich dazu, weiterzuarbeiten. Schweren Herzens räume ich meinen Platz am Joystick. Sofort sitzt der nächste, von der »Suremosch-Leidenschaft« befallene Redakteur vor dem Monitor und versucht sich an meiner Ebene.

In einer anderen Welt

Suremosch führt uns in eine andere Zeit auf den Planeten Barmi in der Galaxis Aknoll. Nichts Außergewöhnliches für uns als erfahrene Star Wars-Piloten. Die Bewohner des Planeten werden seit Jahren von einer Plage heimgesucht: Klockis heißen die Quälgeister, die im Übermaß die knappe Energie von Barmi verbrauchen. Zeit für uns einzugreifen, um den Planeten zu retten. Hierzu steht uns der Suremosch zur Seite, mit dessen Hilfe wir die Klockis in ein Anti-Materie-Feld schieben (Bild 1). Dort werden sie neutralisiert und so die weitere Energieverschwendung gestoppt. Für diese schwierige Aufgabe läßt sich Suremosch, mit dem Joystick in Port B gesteuert, in den vier Himmelsrichtungen bewegen.

Nur Taktik zählt

Der Hinweis »Gehirn aktivieren!« am Beginn jeder Ebene ist ernst zu nehmen. Dies wird jedem Spieler spätestens dann bewußt, wenn er mit seiner vorgegebenen Schrittzahl am Ende ist, bevor noch der letzte Klocki in eines der blinkenden Anti-Materie-Felder geschoben wurde. Suremosch streift durch die Gewölbe (Wedongs) und Ebenen seiner zweidimensionalen Welt — immer auf der Suche nach Klockis, welche, zu allem Überfluß, auch den

Zeitfluß unserer Welt verlangsamten. Doch Vorsicht! Nicht alle Klockis reagieren gleich. Einige bewegen sich in Schubrichtung, bis sie gegen ein Hindernis prallen, andere nur jeweils ein Feld. Die dritte Art ist gutmütig und läßt sich nicht vernichten. Suremosch bleibt immer so lange in einer Ebene, bis er alle Feinde neutralisiert hat oder bis er am Ende seiner Kraft, der vorgegebenen Schrittzahl, erschöpft zusammenbricht.

Hilfe erhalten wir von den Holies. Diese Wesen stärken unseren Helden: Einmal vertilgt, erhöhen Sie die Schrittzahl oder die Anzahl der »Telops«. Mit einem Telop kann Suremosch seinen Platz mit dem eines benachbarten Objekts tauschen (ausgenommen sind die Mauersteine). Stellen Sie dazu Suremosch unmittelbar vor das entsprechende Objekt und drücken den Feuerknopf des Joysticks. Oft bleibt keine andere Möglichkeit, an alle Klockis heranzukommen,

sonst endet Ihr Versuch, wie mein letzter, mit der Meldung »Das war wohl nichts«, und auf Barmi wird weiter Energie im Übermaß verschwendet.

So startet das Vergnügen

Suremosch wird von der Workbench durch Anklicken des Icons »Suremosch.Boot« gestartet. Amiga-Basic muß sich dabei auf der gleichen Diskette befinden. Der Amiga muß sich im 80-Zeichen-Modus befinden. Im Speicher werden für das Spiel 300 KByte benötigt. Das Programm lädt die benötigten Daten und meldet sich dann mit dem Hauptmenü. Durch Anklicken mit der Maus oder den Funktionstasten wählt man einen der Punkte des Hauptmenüs (Bild 2) aus.

Erschaffen Sie als erstes einen Spieler mit Punkt 2 und Eingabe Ihres Namens. Jetzt können Sie alle Optionen des

lassen, indem Sie die rechte Maustaste festhalten und mit dem Zeiger links oben »AUFGEHEN« anwählen (Maustaste dann loslassen).

Nach einem erfolgreich beendeten Unternehmen erscheint die Meldung »Hurra, du hast es geschafft! Die Ebene ist gelöst«, ansonsten die betrübliche Mitteilung »Das war wohl nichts! Versuch's nochmal«. Klicken Sie die Meldung mit der Maus an, erscheint das Spielmenü (Bild 3). Der erste Punkt sorgt für die Rückkehr ins Hauptmenü, dabei werden alle gelösten Ebenen des aktuellen Spielers gespeichert und der Status des Spielers (Punkt 5 des Hauptmenüs) erhöht. <F4> verzweigt ebenfalls zum Hauptmenü, jedoch werden die Spielerdaten hierbei nicht gespeichert.

Jetzt geht's rund

Der vierte Punkt des Hauptmenüs gibt uns die Möglichkeit, eigene Ebenen zu entwerfen. Dazu benötigen Sie den Wedong-Editor. Die Bedienung ist einfach, da auch hier alle Funktionen mit der Maus ausgewählt werden. Durch Drücken der rechten Maustaste erscheinen drei Optionen: Spielfeld, Parameter und Projekt. Bewegen Sie den Mauszeiger auf die Option »Spielfeld«, so können Sie eine neue Ebene erstellen. Gehen Sie dazu auf den Menüpunkt »Objektwahl« und wählen eines der gezeigten Objekte aus. Dieses Objekt kann an beliebiger Stelle auf dem Spielfeld abgesetzt werden, und zwar durch Drücken der linken Maustaste. Entwerfen Sie jetzt eine eigene Ebene, indem Sie das Verfahren für jedes gewünschte Objekt wiederholen. Die Menüpunkte »Löschen« und »Leerfeld« helfen Ihnen, Fehler zu beseitigen. Wichtig: Entwerfen Sie erst einen Rahmen. Dazu dienen die blauen Randsteine (Objekt 1). Haben Sie das vergessen, so bricht das Programm einen Test (siehe unten) ab. Die Meldung »Das Spielfeld enthält einen schweren Fehler« erscheint, wenn Sie Suremosch aus dem Spielfeld steuern.

Fordern Sie andere

Vor dem ersten Test Ihrer Ebene müssen Sie folgende Parameter (»Parameter Ändern«) setzen:



Bild 1. Kann Suremosch seine Welt retten? Ein geistig anspruchsvoller Kampf, bei dem jeder Schritt zählt.

und so den Weg in den nächsten Wedong zu öffnen. Dort kann es vorkommen, daß sich die Klockies nur in einer bestimmten, anfangs nicht erkennbaren, Reihenfolge vernichten lassen. Hier muß erst eine unbekannte Anzahl einer Klocki-Spezies vernichtet sein, bevor sich die zweite Art in das Anti-Materie-Feld schieben läßt. Mit dem Spiel erhalten Sie auch einen »Wedong-Editor«. Mit diesem Editor können Sie selbst Spielfelder erstellen. Insgesamt können 1600 Ebenen gespeichert werden, so bleibt das Spiel eine ständige Herausforderung. Ich wünsche Ihnen noch viel Spaß und nehmen Sie die Warnung zum Einschalten des Gehirns ernst,

Programms nutzen. Im eingerahmten Feld steht immer der Name des aktuellen Spielers. Steht der Name auf »Gastspieler«, können Sie alle Ebenen testen.

Der Menüpunkt 3 startet das Spiel. Man wählt zunächst, durch Anklicken mit der Maus, einen Wedong aus (am Anfang: »Mein erster Wedong«). Ein einmal ausgesuchtes Gewölbe kann von einem Spieler erst dann verlassen werden, wenn er alle darin enthaltenen Ebenen gelöst hat. Erfolgreich bestandene Wedongs werden markiert und können nicht mehr ausgewählt werden. Ebenfalls mit der Maus erfolgt die Wahl des Spielfeldes. Dieses können Sie jederzeit ver-

● **Computer** ● **Hard- und Software** ● **Distribution**

Ausstellung /
Verkauf
Bochum-Innenstadt
Südring /
Ecke Rechener Str.

soyka
DATEN TECHNIK

Verwaltung:
Hattinger Str. 685
4630 Bochum 5
Tel. 02 34/4 98 25-27
Telefax 02 34/4 98 24



Overdrive-System

Zum Aufbau einer externen Copystation mit bis zu drei Drives. Wegen kabelloser Verbindung durch den **Overdrive-System-Port** ist auch beim Dritt- und Viertlaufwerk die Datensicherheit garantiert. Konstruktionsbedingt sind die Laufwerke hervorragend abgeschirmt.

Computer-Point

**Ausstellung · Beratung
Reparaturservice · Verkauf
auf 290 m²**

**Südring / Ecke Rechener Str.
4630 Bochum 1**

Wir freuen uns auf Ihren Besuch!

Flightsimulator II	DM 75,-
Western European Scenery	DM 45,-
Japan Scenery	DM 45,-
Ferrari Formula One	DM 75,-
Turbo	*DM 45,-
Bards Tale II	DM 77,-
Dungeon Master	DM 69,-
Beyond the Ice Palace	DM 69,-
Zoom	DM 57,-
Die Fugger	*DM 55,-
Menace	DM 55,-
4 x 4 Off Road Racing	*DM 69,-
Carrier Command	DM 67,-
Empire Strikes Back	DM 55,-
Ultima IV	DM 55,-

Hardware

Bootselector dfo:/df1: - dfo:/df2:	DM 17,50
Amiga 500/1000 Supra	
Hard Disk 20 MB	DM 1698,-
Amiga 2000 Supra DMA	
Hard Disk 20 MB	DM 1398,-
Impact SCSI Controller I mit 1 MB RAM O K	
ROM Sockel für V 1.3	DM 798,-
Eizo Flexscan	DM 1548,-
Amigo F1 Plus 3.5"	DM 295,-
Amigo F1 OSP 3.5"	DM 329,-
Amigo F1 M 3.5"	DM 289,-
Amigo 2000 3.5"	DM 269,-
Amigo 500 RAM	
Erw. Commodore A 501 abschaltbar und abgeschirmt auf 1 MB	DM 298,-

AKTUELL

**Auswahl aus einem Lager von über
1000 Artikeln**

Software

Aztec C dev. V 3.6	DM 444,-
Aztec C prof. V 3.6	DM 333,-
Aztec C Source Level Debugger	DM 119,-
Benchmark Modula II	DM 297,-
Lattice C V 4.0	DM 298,-
MCC Pascal V 2.0	DM 198,-
Project D	DM 77,-
Quarterback	DM 95,-
Disk to Disk	DM 75,-
Dos to Dos	DM 75,-
Diskmaster	DM 95,-
Lights, Camera, Action	DM 135,-
Deluxe Photo Lab	DM 229,-
Photon Paint (PAL)	DM 175,-
Digi View V 3.0 PAL	DM 295,-
Digi View V 3.0 Update PAL	DM 39,-
Digi Paint PAL	DM 97,-
Publisher 1000 Plus	DM 149,-
Planetarium	DM 98,-
Music X	DM 485,-
The Works	DM 289,-
Interceptor	DM 75,-
Jet	DM 75,-

Archimedes



Autorisierter
Distributor für die
Bundesrepublik
Deutschland

**32 Bit Risc-Computer
Katalog anfordern!**

Gesamtkatalog '88

Fordern Sie kostenlos und unverbindlich unseren Gesamtkatalog '88 an. U.a. mit kompletter AMIGA-Software, Video-Digitizer, Audio-Digitizer, Genlock-Interfaces, Festplattenlaufwerke, SCSI-Controller, Diskettenlaufwerke, Bausätze, RAM-Erweiterungen, Drucker, Monitore, Computer, Literatur.

Schritte: die Anfangsschrittzahl für Suremosch.

Klockis: legt die Anzahl der Klockis fest, welche zu vernichten sind, um die Ebene zu lösen.

Klockis1: ist die Zahl gelber Klockis, die zu vernichten sind, ehe das erste lila Klocki in ein Anti-Materie-Feld geschoben werden kann.

Klockis2: bestimmt den Zusammenhang zwischen lila und orangenen Klockis (siehe Klockies1).

Telops: die Anzahl der Telops zu Beginn der Ebene.

Bonus-Schritte und Bonus-Telops: der Bonus (Schritte

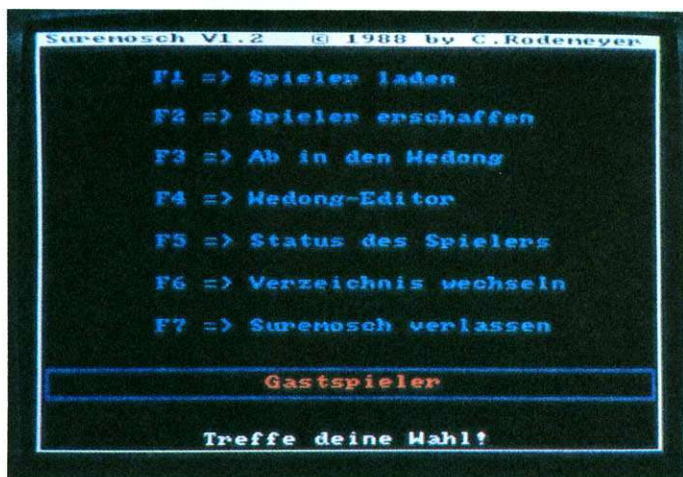


Bild 2. Der Einstieg in das Abenteuer: das Hauptmenü

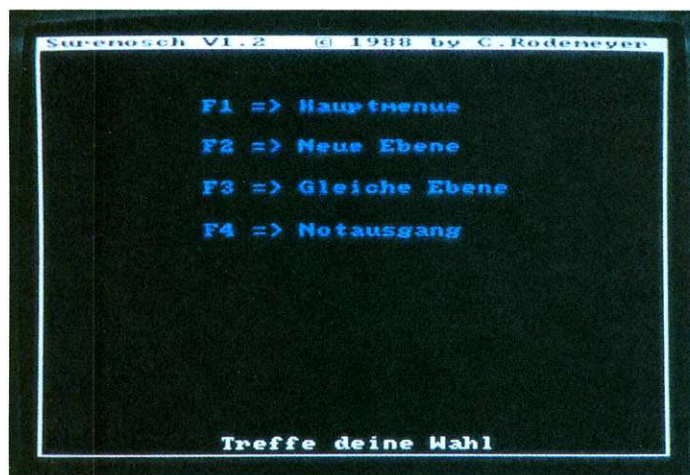


Bild 3. Sind Sie bereit für den nächsten Versuch? Das Spielmenü: Gleiten Sie in die Welt der Klockies.

oder Telops) nach Entfernen eines Bonus-Objektes.

Mit der Option »Projekt« können Sie Ihre erstellte Ebene speichern. Die Punkte »Laden«, »Speichern« und »Speichern als« beziehen sich nur auf den Editor-Speicher. Der Editor-Speicher verwaltet bis zu 40 Ebenen. Erstellen Sie mehr Ebenen, müssen Sie andere vorher in einen Wedong

ablegen. Mit der Funktion »Löschen« entfernen Sie Ebenen aus dem Editor-Speicher. Hier besteht zusätzlich die Möglichkeit, eine irrtümlich gelöschte Ebene zurückzuholen. Bei einem Löschvorgang wird nur der Name der Ebene gelöscht. Die Ebene bleibt aber bis zum Beenden des Programms im Editor-Speicher erhalten und kann erneut geladen und ge-

speichert werden. Hierzu klicken Sie nach »Laden« den Platz im Auswahlmenü an, an dem der Name vorher stand. Jetzt steht dort zwar »Leere Ebene«, das Spielfeld erscheint trotzdem und kann gespeichert werden.

Ein neues Verzeichnis wird mit »Neuer Wedong« erzeugt. Alle Ebenen, die in dem neuen Wedong untergebracht werden, sind zuerst mit der Maus auszuwählen. Nach dem Anklicken von »Weiter« gibt man den neuen Namen ein. Zu Beachten ist, daß ein Wedong nur erstellt wird, wenn mindestens zwei Ebenen für diesen ausgewählt wurden. Die Funktion »Lösche Wedong« ist eingebaut, da ein Löschen von der Workbench mittels »Discard« nicht ausreicht. Achtung: Schließen Sie immer mit dem Rücksprung ins »Hauptmenü« ab. Erst dann wird die Editor-Datei geschlossen.

Sie benötigen ein Unterverzeichnis (Schublade), in dem später das Spiel und die Daten stehen. Erzeugen Sie dafür ein solches Verzeichnis auf der Workbench: Kopieren Sie die

Schublade »Empty« von der Workbench an die gewünschte Position auf einer neuen Diskette und geben dem neuen Verzeichnis mit »Rename« den Namen »Suremosch.Spiel«. In diesem Verzeichnis erzeugen Sie drei weitere Schubladen mit den Namen »Spieler«, »Wedongs« und »Nachlader«. Die Basic-Programme »Suremosch.Boot« (Listing 1) und »Suremosch« (Listing 2) speichern Sie in der Schublade

Installations-Hinweise

»Suremosch.Spiel«. Im Verzeichnis »Nachlader« werden die Dateien »Spieler«, »Objekte« und »Editor.Dat«, worin die grafischen Informationen aller Spielfiguren hinterlegt sind, samt Info-Files erzeugt. Die Files »Mein erster Wedong« und »Wedong-Namen« kommen in die »Wedong«-Schublade.

Die Erzeugung dieser beiden Dateien erfolgt programmgesteuert: Kopieren Sie zunächst Amiga-Basic und die Basic-Programme »File.Dat-Maker« (Listing 3) und »Data-Lader« (Listing 4) in das Hauptverzeichnis Ihrer neuen Suremosch-Diskette. Starten Sie dann zuerst das Programm »Data-Lader« und danach »File-Dat-Maker«.

Bei beiden Programmen müssen Sie nun jeweils den Namen »Suremosch.Spiel« für die Hauptschublade angeben.

Christian Rodemeyer/
P. Baron/rs/sk

Wichtiger Hinweis

Die Listings (1 bis 4) finden Sie ab Seite 149 in diesem Sonderheft.

DAS AMIGA PUBLIC DOMAIN DEPOT

Seit über einem Jahr sind wir für unsere zuverlässigen und schnellen Lieferungen von PD-Software bekannt. Alle Programme werden auf hochwertigen, geprüften 2DD-Qualitätsdisks geliefert. Mit über 1500 Disks haben wir derzeit eines der größten PD-Depots. Jeder Bestellung über 10 Disks wird ein ausführliches, deutsches Handbuch zum Umgang mit Public Domain Software gratis beigelegt.

Wir liefern u.a.: alle Fish, RW, RPD, Poseidon, Panorama, RHS, Auge, Chiron, Amuse, ACS, Slideshows, Tor-Special, ES-PD, SAFE, TBAG, Faug, UK, Franz, Kickstart, SACC, etc. etc. ...

Poseidon 1-280 (und Poseidon Special 1-160)
Diese neue Superserie enthält nur das Beste vom Besten. Alle Programme sind garantiert lauffähig und lassen sich durch einfaches Anklicken starten. (Daher auch hervorragend für Einsteiger geeignet). Näheres zu dieser Serie auf unseren Katalogdisks!

2 Katalogdisks mit Kurzbeschreibung aller Programme in deutsch gegen DM 5,— anfordern (bar, Briefmarken, Scheck).
— Wir liefern als erster PD-Anbieter komplett deutsche Katalogdisks.

Einzeldiskette :	DM 6,—
ab 10 Stück :	DM 5,70
ab 20 Stück :	DM 5,50
ab 30 Stück :	DM 5,30
ab 40 Stück :	DM 5,—
ab 50 Stück :	DM 4,80
ab 100 Stück :	DM 4,50

Bei Vorkasse ist der Versand kostenlos, bei Nachnahme werden DM 6,— berechnet.

Hardwareliste:

Golem 3,5" Laufwerk	DM 299,—
- abschaltbar, durchgeschliffener Bus, helle Frontblende	
Media Diskettenbox	DM 45,—
- 3,5", stapelbar, ausziehbar, für bis zu 160 Disketten	
Mausmatte	DM 19,50
- ideale Mausunterlage in der Farbe rot	
Maushalter	DM 9,50
- wird am Monitor befestigt, hervorragender Platzschaffer	
3,5" Leerdisk 2DD 10 Stück	DM 24,—
- Qualitätsdisks eines namhaften Herstellers neutral verpackt	
Die Lieferung erfolgt umgehend ab Lager!!!	

WOLF-COMPUTERTECHNIK

Deipe Stegge 187
4420 Coesfeld

Inh.: Rainer Wolf
Tel.: 02541/2874

Die PD-Software für den Amiga gewinnt ständig an Bedeutung. Die Qualität ist ebenso rasch gestiegen wie die Menge der vorhandenen Programme. Die »PD-Flut« stellt wohl die meisten vor das Problem, die richtige Software herauszufinden. Darüber hinaus braucht jeder Anwender meist nur ein gutes Grafik- und ein Kopierprogramm. Aber wie soll ein Amiga-Besitzer heute die »Perlen« aus dem Überangebot herausfinden? Wir stellen Ihnen tolle Software für jeden Geschmack vor. Public Domain-Programme sind kostenlose Software, die jederzeit frei kopiert werden darf. Man unterscheidet dabei zwischen Shareware- und Freeware-Programmen. Unter dem Begriff Shareware sind die

Was bietet PD-Software?

Die meisten PD-Programme sind Utilities und Tools. Das sind kleine nützliche Programme, die jeder Amiga-Besitzer gebrauchen kann. In diesem Bereich ist die PD-Software am stärksten. Hier hat sie die professionellen Programme

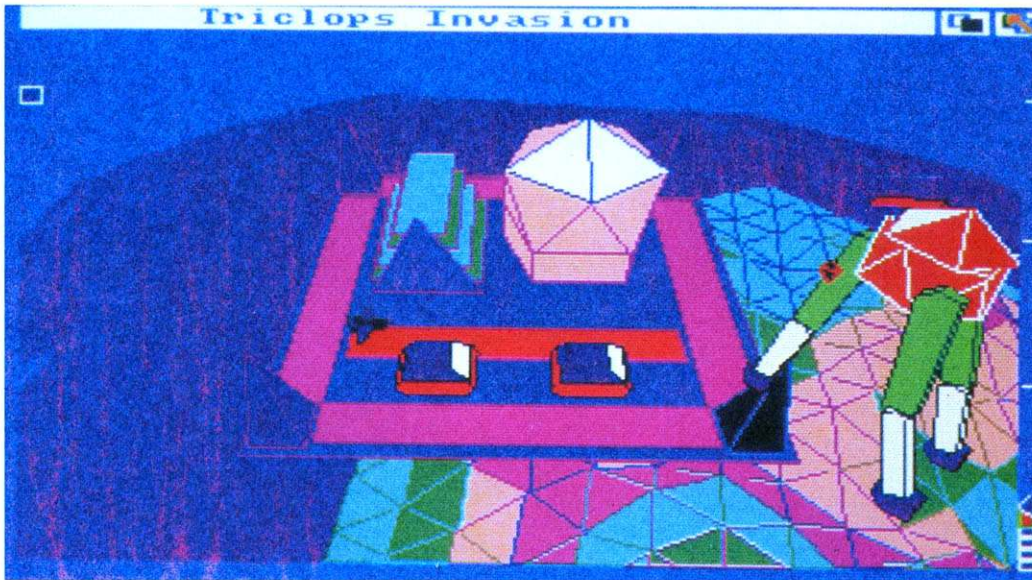
Entgelt tolle Programme schreiben. Zum einen sind dies Profis, die aus einem (bei der täglichen Arbeit benötigten) Sprite-Editor ein PD-Programm »gebastelt« haben. Diese Profis wissen, daß sich ihr kleines nützliches Programm nicht verkaufen läßt. Sie geben es somit kostenlos ab, um die Anwender auf sich oder das nächste große Pro-

Schatten zu stellen — was ihnen oft auch gelingt.

Wie oben schon gesagt, bietet PD-Software heute bei den Spielen hohe Qualität. »Tric-

Super-Spiele

lops Invasion« (Bild 1) ist ein auf Fraktalgrafik beruhendes Actionspiel. Ziel des Spiels ist es, einen Planeten vor der Zerstörung durch riesige dreibeinige Roboter (frei nach H.G. Wells »Krieg der Welten«) zu retten. Diese Dreibeiner müssen beseitigt werden, ebenso wie der Satellit, von dem die Roboter in Richtung Planeten auschwärmen. Zu diesem Zweck stehen ein Jet und zwei ferngesteuerte Panzer zur Verfügung, die bei Berührung mit einem Dreibeiner explodieren. Die Robo-



»Triclops Invasion« ist ein Action-Spiel mit toller Fraktalgrafik

Preiswerte Software

für den

Programme zusammengefaßt, die man frei testen, kopieren und weitergeben darf. Die Autoren der Shareware-Produkte verlangen aber, daß man eingetragener User des Programms wird, sofern man das Programm ständig verwenden will. Dieses Eintragen kostet Geld, man bekommt dafür jedoch in Zukunft alle Updates des Programms. Dieses Kon-

Was ist eigentlich PD-Software?

zept stammt aus den USA. Dort klappt dies auch ausgezeichnet, die Anwender haben begriffen, daß Autoren auch einen Lohn für ihre zeitaufwendige Arbeit verdient haben. In Deutschland ist diese Erkenntnis leider noch nicht vollständig bis zu allen Anwendern vorgedrungen.

Freeware ist im Gegensatz dazu kostenlos, es sind lediglich die Gebühren für Diskette und Versand beim Verreiber des Programms zu entrichten.

Programme für den Amiga sind im allgemeinen recht teuer. Eine Alternative zu kommerziellen Produkten ist Public Domain-Software, die heute teilweise eine beachtliche Qualität besitzt. Wir erleichtern Ihnen die Wahl des richtigen Programms aus dem riesigen Angebot.

noch nicht überholt, aber immerhin mit ihnen gleichgezogen. Im Spielbereich war bisher Public Domain noch nicht so leistungsfähig, die PD-Spiele sind häufig recht kompliziert und langsam und haben schlechtere Grafik als die kommerziellen Programme. Aber gerade in den letzten Monaten erscheint dieser Bereich deutlich verbessert, die Zeit der »Vier Gewinn«-Spiele ist endgültig vorbei.

Ohne Autor kein Programm

Wenn man in einem README-File einmal gelesen hat, wie lange der Autor an einem Freeware-Programm gearbeitet hat, fragt man sich, was das für Menschen sind, die ohne

gramm aufmerksam zu machen.

Die anderen sind diejenigen, deren größtes Hobby der Computer ist. Sie produzieren oft die PD-Spezialprogramme, die zum Beispiel den Sternenhimmel jedes Orts der Erde zu jedem beliebigen Zeitpunkt darstellen, oder die Struktur eines bestimmten Atoms darstellen und erklären.

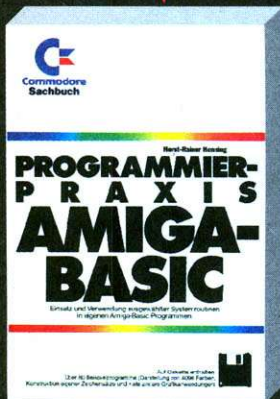
Außerdem gibt es noch die PD-Freaks, die zu Hause eine riesige Sammlung an PD-Programmen haben, und deren einziges Original heute wie damals die Workbench-Diskette ist. Sie programmieren ausschließlich auf PD-Compilern und PD-Assemblern, verachten jede Art von professioneller Software und versuchen, diese mit PD-Programmen in den

Amiga

ter werden mit einem gezielten Schuß in eines ihrer Beine vernichtet. Dabei kann man verschiedene Blickwinkel, wie zum Beispiel die Vogelperspektive, wählen. Das Besondere an diesem Spiel ist zweifellos die Fraktalgrafik, in der die Welt dargestellt wird. Leider hat dies zur Folge, daß der Spielverlauf recht langsam ist. Die Geschwindigkeit kann man jedoch erhöhen, indem statt fünf nur vier Bitplanes eingestellt werden. Diese Einstellung ist vor dem Programmstart anzugeben. Das Spiel wird über die Maus gesteuert, mit der nicht weniger als 12 verschiedene Funktionen aufgerufen werden (und das ohne Menüs). Dies ist zwar etwas gewöhnungsbedürftig, zahlt sich aber durch die einfache Steuerung aus. Leider muß das Programm neu gestartet werden, wenn man ein Spiel verloren hat.

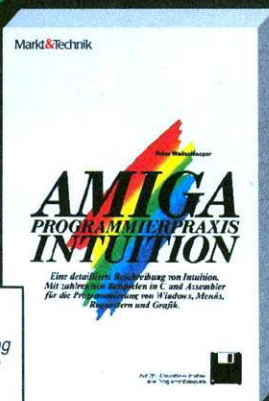
»Hack« (Bild 2) ist ein Grafik-Adventure, das vom Betriebssystem Unix auf den Amiga

Brandneue Bücher für Ihre **Amiga-** **Bibliothek**



H. R. Henning
Programmierpraxis Amiga-Basic
Einsatz und Verwendung ausgewählter Systemroutinen in eigenen Amiga-Basic-Programmen. Die beigelegte Diskette enthält über 80 Beispielprogramme.
1988, 368 Seiten, inkl. Diskette
Bestell-Nr. 90549
ISBN 3-89090-549-8
DM 59,-/sFr 54,30/öS 460,20

P. Wollschlaeger, Amiga:
Programmierpraxis Intuition
Eine detaillierte Beschreibung von Intuition. Mit zahlreichen Beispielen auf Diskette.
1988, 330 Seiten, inkl. Disk.
Bestell-Nr. 90593
ISBN 3-89090-593-5
DM 69,-/sFr 63,50/öS 538,20



A. Plenge
Amiga-3-D-Grafik und Animation
Eine leichtverständliche Anleitung für die Erstellung von dreidimensionalen Grafiken: Clipping, Perspektivische Projektion, Raytracing, Versteckte Linien, Schatten, Reflexion, 3-D-Editor.
1988, 376 Seiten, inkl. Diskette
Bestell-Nr. 90526, ISBN 3-89090-526-9
DM 69,-/sFr 63,50/öS 538,20

D. Myers, Amiga: Grafik * Musik * DFÜ
Leichtverständlicher Programmierkurs für die erfolgreiche Grafik- und Soundprogrammierung in Amiga-Basic. Die Musikfunktionen, die Stimme des Amiga, Grafik, Animation, Datenfernübertragung und viele weitere Themen werden detailliert beschrieben.
1988, 231 Seiten, inkl. Diskette
Bestell-Nr. 90579, ISBN 3-89090-579-X
DM 59,-/sFr 54,30/öS 460,20



I. Krüger, Amiga: Programmieren mit Modula 2
Leichtverständlicher Modula-2-Kurs. Mit vielen Beispielen für die systemnahe Programmierung unter der grafischen Benutzeroberfläche »Intuition«. Auf der Diskette enthalten: alle Modula-2-Beispiele für Screens, Windows etc.
1988, 362 Seiten, inkl. Diskette
Bestell-Nr. 90554, ISBN 3-89090-554-4
DM 69,-/sFr 63,50/öS 538,20



P. Wollschlaeger, Amiga-Assembler-Buch
Dieses Buch beweist, daß Assembler-Programmierung ganz einfach ist: Ein 68000er-Kurs mit vielen Beispielen. Mit ausführlichem Verzeichnis aller Systemroutinen, Anleitung für das Einbinden von Assembler-Routinen in Amiga-Basic und vielen Informationen über die Internas des Amiga-Betriebssystems. Mit Beispieldiskette.
1987, 329 Seiten, inkl. Diskette
Bestell-Nr. 90525, ISBN 3-89090-525-0
DM 59,-/sFr 54,30/öS 460,20



H. R. Henning, Programmieren mit Amiga-Basic
Eine gründliche Einführung in die Programmierung mit Amiga-Basic: Animation (bewegte Grafiken und Sprites) - Befehle zur Sprach- und Musikausgabe - Fenstertechnik - Sequentielle Dateiverwaltung - Spieleprogrammierung - viele Tips&Tricks und eine 3 1/2"-Diskette mit über 100 Programmbeispielen.
1987, 363 Seiten, inkl. Diskette
Bestell-Nr. 90434, ISBN 3-89090-434-3
DM 59,-/sFr 54,30/öS 460,20

Markt & Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computerfachgeschäften oder in den Fachabteilungen der Warenhäuser.

Irrtümer und Änderungen vorbehalten.


Markt & Technik
Zeitschriften · Bücher
Software · Schulung

Markt & Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2,
8013 Haar bei München, Telefon (089) 4613-0.

SCHWEIZ: Markt & Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656,

ÖSTERREICH: Markt & Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0,

Rudolf Lechner & Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526

Ueberreuter Media Verlagsges.mbh (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0.



Fragen Sie bei Ihrem Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Programmname: Triclops Invasion

Autor: (C) Geodesic Publication; Präsident: Mark S. Adams

Kurzbeschreibung/Sonstiges:

Fraktalgrafik; Shareware (21 Dollar)

Positiv:

- + Fraktalgrafik
- + gesamte Steuerung über die Maus
- + gut animierte Figuren

Negativ:

- etwas gewöhnungsbedürftig
- nachdem man verloren hat, muß man das Spiel neu starten
- durch die Fraktalgrafik mitunter etwas langsam

Fazit: ein gelungenes Spiel und ein absolutes Muß für jeden Fraktal-Fan

übertragen wurde. Den Datenreisenden, die einmal in der Markt & Technik-Mailbox »OIS« herumgeschnuppert haben, ist Hack wohl ebenfalls bekannt, es gehört dort zum ständigen Angebot. Die Umsetzung des Spiels auf den Amiga ist toll gelungen. Die einfachen, sehr gewöhnungsbedürftigen Zeichen, durch die ursprünglich das Spielgeschehen dargestellt wurde, sind auf dem Amiga durch Icons ersetzt. Der Reiz des Spiels wurde gerade durch die neuen Grafikelemente stark erhöht.

Stichwort Unix

Bei Public Domain-Programmen für den Amiga kommt man immer wieder mit einem Wort in Berührung, von dem viele noch nicht wissen, was es bedeutet: Unix. Unix ist ein Betriebssystem für größere Rechenanlagen. Für Unix-Systeme gibt es Unmengen von Programmen aller Art. Die Standard-Programmiersprache auf Unix-Systemen ist C, genauso wie auf dem Amiga. In dieser Vorstellung von PD-Software finden Sie mehrere Programme, die von Unix auf den Amiga übertragen wurden, wie zum Beispiel Hack, Zoo und Empire.

Bei Spielbeginn wählt man einen Abenteurer, den man von nun an durch ein Höhlensystem lenkt. Sie wählen dabei einen der Charaktere Tourist, Höhlenforscher, Kämpfer, Ritter, Höhlenbewohner oder Zauberer. Alle Charaktere besitzen bestimmte Ausrüstungsge-

genstände und spezifische Stärken und Schwächen. Nach dieser Auswahl begibt man sich auf die Suche nach einem Amulett, das sich im Besitz eines Magiers befindet. Begleiter ist ein anfangs als »little dog« bezeichneter Hund. Im Verlauf eines Abenteuers kann dieser zu einer beachtlichen Größe heranwachsen und so seinem Herrn gute Dienste erweisen. Das Spiel findet in ei-

die Menüleiste zu erreichen sind. Das ist sowohl für den Anfänger als auch für den erfahrenen Spieler von Vorteil, da Hack die deutsche Tastaturbelegung nicht akzeptiert und daher besonders bei den Sonderzeichen nicht immer klar ist, wo diese zu finden sind. Es sei abschließend darauf hingewiesen, daß dieses faszinierende Abenteuerspiel derzeit nur in einer englischen Version exi-

verbunden sind. Dabei braucht nur ein Spieler einen Amiga und Empire. Die übrigen Mitspieler benötigen lediglich ein Terminalprogramm und einen beliebigen anderen Computer. Der Amiga, auf dem Empire läuft, steuert alle Ein- und Ausgaben an die einzelnen Spieler. Aus diesem Grund kann leider keine Grafik verwendet werden.

Empire ist eine komplexe Si-

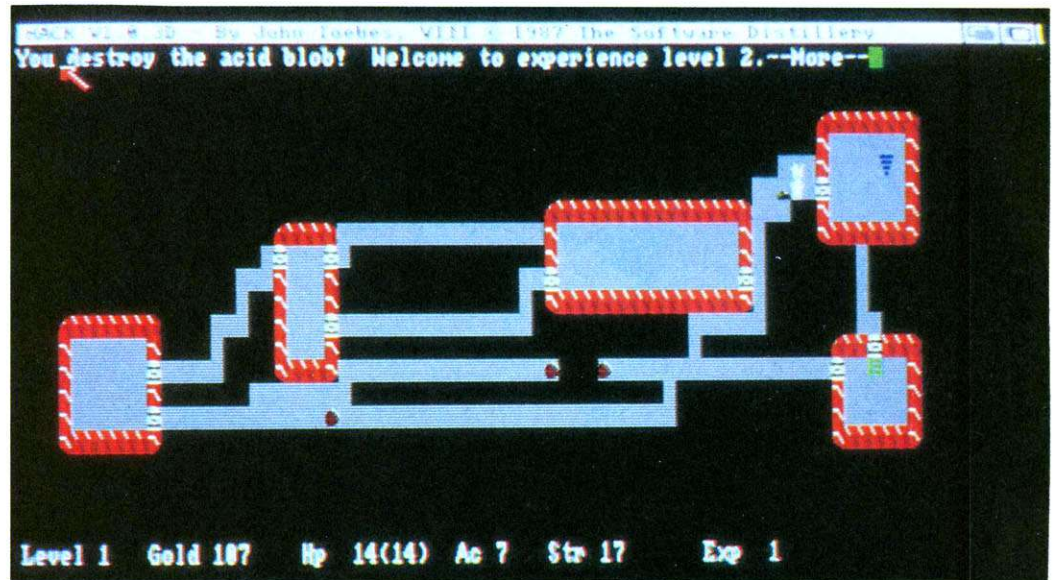


Bild 2. Schon in der ersten Ebene von »Hack« wimmelt es von Monstern

nem riesigen Höhlenkomplex statt, der aus 20 Etagen besteht. Hier wimmelt es nur so von Schätzen und Monstern. Nach dem Spielende durch den Tod der Spielfigur (Bild 3) oder Speichern eines Spielstandes muß Hack neu gestartet werden. Allerdings besteht die Möglichkeit, Hack in einer Art Trainermodus zu spielen, um sich einen Überblick zu verschaffen.

Das Abenteuer wird mit Hilfe einer Vielzahl von Befehlen gespielt, die entweder durch Tastaturkommandos oder über

stiert, Kenntnisse dieser Sprache sind unbedingt notwendig.

»Empire« ist ein etwas ungewöhnliches Spiel, das in »Draco« geschrieben ist. Draco ist eine Shareware-Programmiersprache, die einen Mittelweg zwischen Pascal und C beschreitet. Der Reiz des Taktik- und Strategiespiels liegt darin, daß es über Datenfernübertragung (DFÜ) mit Freunden gespielt wird. In der vorliegenden Version kann nur ein Anwender am Amiga spielen, während seine Gegenspieler über den seriellen Port mit dem Amiga

mulation des Weltgeschehens, bei der jeder Spieler eine Nation repräsentiert. Jeder Mitspieler kann andere Länder angreifen und mit ihnen Handel betreiben. Herrenlose Länder können besetzt werden, Soldaten einberufen, Kriegsmaschinen aufgebaut und die Wirtschaftskraft gesteigert werden. So entwickelt sich jedes Land je nach dem Handlungsschwerpunkt des jeweiligen Spielers vollkommen unterschiedlich. Insgesamt stehen den Spielern nicht weniger als 55 Befehle zur Verfügung, mit denen das Geschehen an Land, in der Luft und auf dem Wasser gesteuert wird. Darüber hinaus ist es möglich, mit einer Art »Master« Empire zu spielen. Dieser Master hat erhöhte Priorität und kann entscheidend in das Spiel eingreifen, um zum Beispiel eine Währung zu stabilisieren.

Besondere Beachtung verdient sicherlich auch die Tatsache, daß immer neue Welten geschaffen werden können, die beispielsweise in Größe, landschaftlicher Beschaffenheit und Rohstoffvorkommen unterschiedlich sind. Natürlich braucht dieses Spiel reichlich Zeit, bis es zu Ende gespielt ist. Hier ist mit ungefähr zwei Mo-

Programmname: Hack

Autor: (C) Stichting Mathematisches Centrum, Amsterdam

Übertragen auf Amiga:

John A. Toebes

Mit Hilfe von: Dave Baker,

Ed Burnette, Stan Chow,

Jay Denebiem, Gordon

Keener, Jack Rouse, Mary

Ellen Toebes, Doug Walker

Grafik: Jeff Lee

Kurzbeschreibung/Sonstiges:

Adventure mit Kultcharakter

Positiv:

- + umfangreich, komplex

+ Trainer-Mode

+ wichtige Befehle auch über Menüleiste und Maus erreichbar

+ mehrere Charaktere zur Auswahl

+ nette Grafik

Negativ:

— amerikanische Tastaturbelegung

— nachdem der Spielstand gespeichert wurde oder die Spielfigur gestorben ist, muß man das Spiel neu starten

Fazit: Für jeden Amiga-Besitzer mit Abenteuerlust bestens geeignet

naten zu rechnen, wenn täglich nicht länger als eine halbe Stunde gespielt wird. Mitspielen können je nach Größe der generierten Welt bis zu 50 Teilnehmer, die nicht alle zur gleichen Zeit spielen müssen. Gespielt werden kann mit einer Übertragungsrate von 300, 1200 oder 2400 Bit/s. Empire ist eine äußerst realistische Simulation, die kaum Wünsche offen läßt.

Programmname: Empire
Autor: Originalspiel auf Unix-System: Peter Langston
 Umgesetzt auf Amiga: Chris Gray
Kurzbeschreibung/Sonstiges: komplexe Simulation über DFÜ; in Draco geschrieben
Positiv:
 + sehr realistisch
 + komplex und umfangreich
 + zahlreiche Befehle
 + viele Teilnehmer zu verschiedenen Zeiten möglich
 + neue »Welten« erstellbar
 + 300, 1200 oder 2400 Bit/s
Negativ:
 — keine Grafik (durch DFÜ bedingt)
Fazit: ein interessantes Spiel und ein Genuß für jeden Hobbystrategen

Kopieren mit zwei Laufwerken

»QuickCopy« ist ein Kopierprogramm für zwei Laufwerke, wobei immer von DF0: nach DF1: kopiert wird. Nach dem Einlegen der Disketten und dem Anklicken des Start-Symbols beginnt QuickCopy, die Diskette zu kopieren. Dabei wird ständig die jeweilige Trackposition auf der Diskette angezeigt, die Statuszeile macht auf eventuelle Fehler aufmerksam. Nach zirka 1,5 Minuten ist der Kopiervorgang beendet, falls er nicht vorher durch Anklicken des Stop-Symbols unterbrochen wurde. Danach kann die nächste Diskette kopiert oder das Pro-

gramm verlassen werden. Somit stellt QuickCopy ein einfaches, aber zuverlässiges Werkzeug zum Kopieren von Disketten dar, das teilweise sogar geschützte Disketten — natürlich nur für Sicherheitskopien — dupliziert.

Liste aller archivierten Files erstellt werden. Diese Liste ist mit »Zoo« einfach ausgedruckt, Files mit bis zu 65535 Zeichen können kommentiert werden. Man kann das Archiv aktualisieren, einzelne Files löschen, hinzufügen und gelöschte Fi-

Verblüffendes Ray-Tracing

Bilder, die vorher nur von Großrechenmaschinen erstellt werden konnten, sind jetzt auch mit »Homecomputern« realisierbar, die auf dem



Bild 3. Das unglückliche Ende eines Abenteuers versüßt dieses Schlußbild von »Hack«

Programme unter Druck

Es ist bekannt, daß man sich von allen wichtigen Dateien und Programmen mindestens eine Sicherheitskopie anlegen sollte, da diese durch Disketten- oder Benutzerfehler zerstört werden können. Trotzdem verzichten viele auf diese Sicherung, da das Anfertigen von Sicherheitskopien lästig ist und diese große Mengen Speicherplatz auf den Disketten belegen.

Hier versucht »Zoo« zu helfen. Das Utility dient zum Archivieren von Files. Hierbei werden alle gewünschten Files um bis zu 80 Prozent komprimiert und dann in ein gemeinsames File gespeichert. Auf diese Weise werden Programme platzsparend gesichert. Außerdem bietet »Zoo« noch eine Vielzahl an weiteren Möglichkeiten.

Es kann zum Beispiel eine

les retten. Die letzte Funktion des Utilities ist das Dekomprimieren von Files. Nach Ausführung dieser Funktion hat man seine ursprünglichen Files wieder.

Für die Steuerung von Zoo stehen Anfänger- und Expertenbefehle zur Verfügung, wobei erstere immer vollständige Kommandos sind, während die Expertenbefehle nur aus einem Zeichen bestehen. So ersparen sich Fortgeschrittene nach einiger Eingewöhnungszeit mühsame Tipparbeit.

Programmname: Zoo
Autor: Autor des ursprünglichen Unix-Programms: Rahul Dhesi
 übertragen und angepaßt von: J. Brian Waters
Kurzbeschreibung/Sonstiges: Von Unix auf Amiga übertragen
Positiv:
 + komprimiert um bis zu 80 Prozent
 + Anfänger- und Expertenbefehle
Negativ:
 — nicht in Intuition eingebunden
 — etwas gewöhnungsbedürftig
Fazit: ein durchaus brauchbares Programm, um eine Diskettensammlung zu archivieren

68000er-Prozessor basieren. Der Schlüssel dazu heißt »Ray-Tracing«.

Seit kurzem gibt es entsprechende Programme auch für den Amiga. Den ersten Schritt in diese Richtung tat »Sculpt 3D«, mit dem man hervorragende Bilder auf dem Amiga berechnen konnte. Kurze Zeit später erschien »Silver«, das diese erzeugten Grafiken (die meistens im HAM-Modus dargestellt werden) auch animieren konnte. Außerdem wurden die Algorithmen verbessert, so daß die nötigen Berechnungen noch schneller wurden.

Ein erheblicher Nachteil dieser beiden Programme ist jedoch, wie so oft, ihr Preis. Hier bietet sich dem Amiga-User eine Alternative: DBW-Render.

»DBW-Render« ist ein Public Domain-Ray-Tracing-Programm, das vielseitige Möglichkeiten bietet. Der größte Unterschied zwischen den beiden Programmen Sculpt 3D und Silver ist die Eingabe der Bilder. Bei den beiden professionellen Programmen werden dem Anwender drei Fenster zur Verfügung gestellt, die die Ansicht des Objektes von vorne, von der Seite und von oben zeigen. In diesen Fenstern können nun die Objekte eingegeben werden.

Einen vollkommen anderen Weg geht DBW-Render. Hier wird dem Anwender eine ei-

Programmname: QuickCopy
Autor: Dave Devenport
Kurzbeschreibung/Sonstiges: kopiert nur mit zwei Laufwerken und nur von DF0: nach DF1:
Positiv:
 + akzeptable Kopierzeit

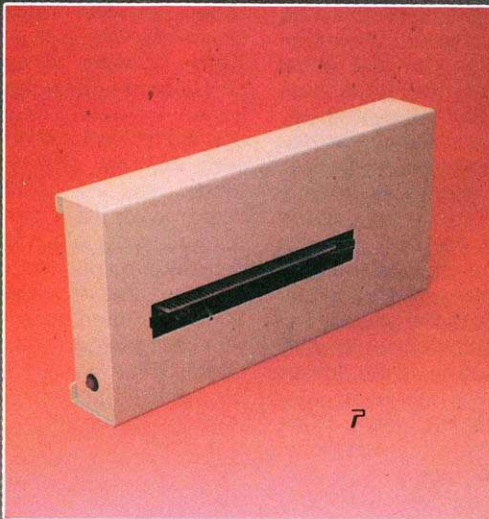
(ca. 1 min 30 s)
 + kopiert teilweise auch geschützte Disketten
Negativ:
 — es werden zwei Laufwerke benötigt
 — es kann nur von DF0: nach DF1: kopiert werden
Fazit: für Besitzer zweier Laufwerke brauchbar

KUPKE

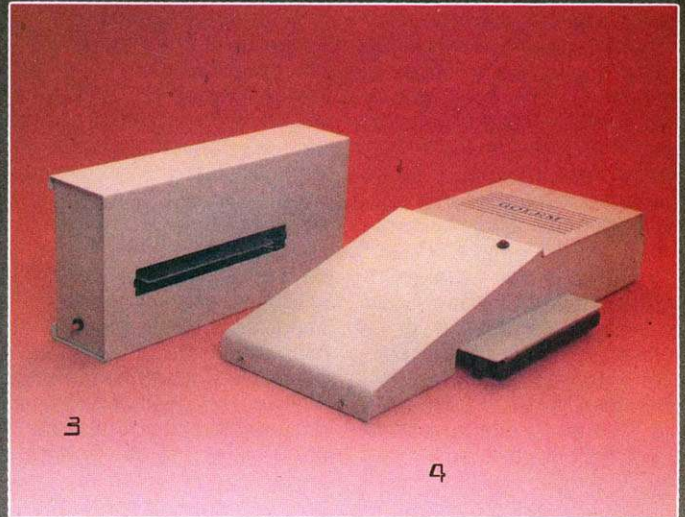


02 31/81 83 25-27
Telefax 02 31/81 74 29
D-4600 Dortmund 1
Burgweg 52 a

GOLEM

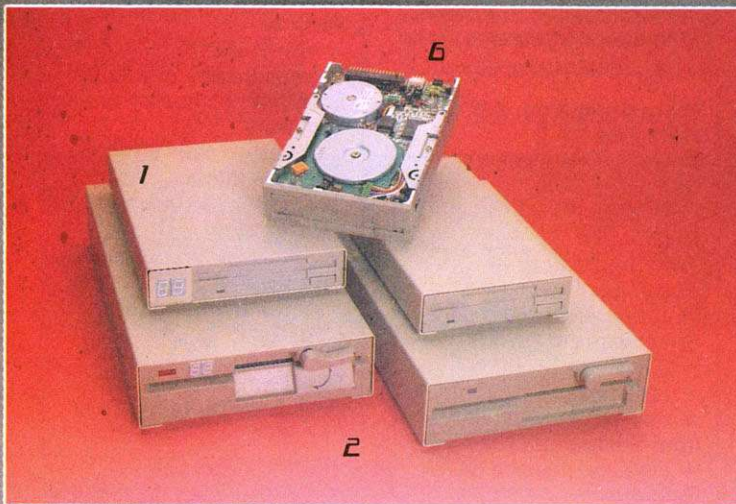


1

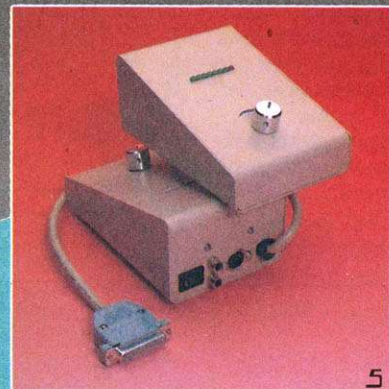


3

4



2



5

**Wir
liefern im
3-Tage-Rhythmus**

1 Golem Drive 3,5 Display

NEC 1037a mit heller Frontblende ● Amiga-farbenes Metallgehäuse ● Abschalte ● Busdurchführung bis DF3 ● Sidecar, PC 1 und PC Karten-kompatibel ● Trackdisplay zur aktuellen Spur- und Kopfanzeige

mit Display
ohne Display

DM 359.-
DM 339.-

2 Golem Drive 5,25 Display

NEC Laufwerk mit heller Frontblende ● Amiga-farbenes Metallgehäuse ● Abschalte 40/80 Track-Umschalte ● Busdurchführung bis DF3 ● PC Karten, Sidecar und PC 1 kompatibel ● Trackdisplay zur aktuellen Spur- und Kopfanzeige

mit Display
ohne Display

DM 449.-
DM 419.-

3 Golem Ram Box 2MB

2MB Speichererweiterung für dem Amiga 1000 ● ansteckbar am Systembus ● Abschalte ● Busdurchführung ● autokonfigurierend ● Betriebskontrollanzeige ● Amiga-farbenes Metallgehäuse ● erweitert den Grundspeicher auf 2,5 Megabyte

komplett
ohne Ram's

DM 1398.-
DM 449.-

4 Golem 500 Ram Box

2MB Speichererweiterung im formschönen 500'er Design ● Busdurchführung ● autokonfigurierend ● Betriebskontrollanzeige ● externer Anschluß an den Systembus ● erweitert den Grundspeicher auf 2,5 Megabyte

komplett
ohne Ram's

DM 1398.-
DM 449.-

Technische Änderungen vorbehalten

5 Golem Sound Stereo

Audio Digitizer der Spitzenklasse ● kompatibel zur meisten Samplersoftware ● DIN- und Cinch Anschluß auch für Micro geeignet ● optisches Aussteuerungsdisplay ● Stereowandlung ● umschaltbar auf Mono-Betrieb

Stereo
Mono ohne Display

DM 189.-
DM 139.-

6 Golem Drive A 2000

internes Amiga Drive ● NEC 1036a mit heller Frontblende ● einbaufertig modifiziert ● mit Staubschutzklappe ● incl. Einbauanleitung und Montagesatz

DM 200.-

7 Kickstart/Uhrenmodul

"Bitte Workbench einlegen", meldet ihr Amiga 1000 nach dem Einschalten mit dem extern ansteckbaren Kickstartmodul ● Busdurchführung ● Abschalte, so daß andere Kickstartversionen wieder gebootet werden können ● alle gängigen Kickstart-Versionen lieferbar

DM 199.-

Amiga 500/2000 kompatibles Uhrenmodul ● Akkugepuffert ● extern ansteckbar im Extragehäuse
Uhr u. Kick in einem Gehäuse

DM 149.-
DM 299.-

8 Kickstartumschaltplatine

Intern einsteckbare Umschaltplatine bestückt mit einem zusätzlichen Kickstart ● alle gängigen Versionen ● keine Lötarbeiten erforderlich ● umschaltbar auf original Kickstart

komplett
ohne Eprom's

DM 149.-
DM 59.-

gens entwickelte Hochsprache zur Verfügung gestellt, die an C angelehnt ist. Mit dieser Programmiersprache kann man Objekte in Form von »Macros« definieren. Hierbei werden zum Beispiel Farbe, Material und Brechungsindex festgesetzt. Diese Objekte können wiederum aus mehreren vorher definierten Macros bestehen. Danach kann die Position und Stellung der Objekte im Raum festgelegt werden. So ist es auch möglich, eine kurze Animation zu erstellen. Darüber hinaus können zwei verschiedene Lichtquellen (einmal diffuses Licht, einmal Spot) definiert werden. Aber die Möglichkeiten des DBW-Render sind damit noch lange nicht ausgeschöpft. Es ist beispielsweise ohne Probleme möglich, beliebige IFF-Bilder als Oberflächenmaterialien zu verwenden. Aber nicht nur bei der Erstellung der Bilder, auch in der Testphase besteht eine Vielzahl von Möglichkeiten. So kann man beispielsweise seine Grafik zuerst in einer niedrigen Auflösung berechnen lassen und gegebenenfalls Änderungen durchführen.

Für Übertragungen auf andere Systeme oder Grafikkar-

ten lassen sich sogar Bilder in einer Auflösung von bis zu 1024 x 1024 Bildpunkten mit 16 Millionen Farben berechnen, die dann auf dem normalen Amiga-Monitor (z.B. 1084) natürlich nicht dargestellt werden können. Der aktuelle Stand der Berechnungen wird automatisch zwischengespeichert, um den Schaden bei einem Systemabsturz möglichst gering zu halten. Die zugehörige Anleitung ist sehr gut und umfassend, aber leider nur in Englisch erhältlich.

Für die Zukunft ist übrigens ein grafischer Editor angekündigt, mit dem die Objekte dann ähnlich wie bei den kommerziellen Programmen eingegeben werden können.

Reset-feste RAM-Disk

Wer kennt diese Situation nicht: Man schreibt stundenlang an einem Programm, und gerade bei dem Probelauf, vor dem das Programm nicht abgespeichert wurde, stürzt das System ab. Fatale Folge ist der komplette Datenverlust — die Arbeit war umsonst. Abhilfe schafft die »ASDG-RAM-Disk«. Sie ist Reset-fest und gestattet das Testen »mit Sicherheitsnetz«.

Wie gestaltet sich die Arbeit mit der RAM-Disk in der Praxis? Nach dem Kopieren der beiden Files »Mountlist« und »asdg.vdisk.device« in den »devs«-Ordner der Startdiskette genügt ein einziger Befehl zum Initialisieren der RAM-Disk (»Mount vd0:«). Sie wird dann als »vd0:« angesprochen. Dieser Initialisierungsvorgang nimmt nur zirka fünf Sekunden in Anspruch.

Der einzige Nachteil dieses Programms ist die notwendige Reaktivierung der RAM-Disk mit dem Befehl »Mount vd0:« nach einem Reset.

Genauso wie die konventionelle RAM-Disk verwaltet die ASDG-RAM-Disk ihren Speicher dynamisch. Die RAM-Disk beansprucht also immer nur so viel Speicherplatz, wie für Daten momentan benötigt wird. Interessant ist, daß die konventionelle RAM-Disk »ram:« erhalten bleibt. So kann man mit zwei RAM-Disks gleichzeitig arbeiten.

Darüber hinaus befinden sich vier weitere Hilfsprogramme für den Gebrauch der ASDG-RAM-Disk auf der Diskette. Das Programm »CleanRamDisk« wird benötigt, um die RAM-Disk mit einer Erwei-

terung zu betreiben, die den Speicher periodisch verwaltet.

Das zweite Programm heißt »FastMem« und schaltet das Fast-RAM ein beziehungsweise aus. Dies ist bei der Benutzung von Kickstart V1.1 von Nutzen, da hier der zusätzliche Speicher unter Umständen nicht erkannt wird. Auf diese Art können auch Programme, die nur mit 512 KByte arbeiten, auf erweiterten Amigas laufen.

Das Hilfsprogramm »DeleteRamDisk« nimmt der ASDG-RAM-Disk ihren Reset-Schutz, was zur Folge hat, daß die RAM-Disk mit allen gespeicherten Daten beim nächsten Reset zerstört wird. Dieser Befehl dient dem schnellen »Aufräumen« einer vollgepackten RAM-Disk.

Das vierte und letzte Hilfsprogramm heißt »SysMon«. Dieses Programm zeigt den Zustand des Speichers, der Tasks und der CPU an. »SysMon« ist interaktiv und stellt sofort jede Veränderung numerisch und grafisch dar.

Die Programme SysMon, DeleteRamDisk und FastMem stammen von Panorama Disks.

Programmname: ASDG-RAM-Disk

Autor: Perry Kivolowitz (1987)

ASDG-RamDisk + CleanRamDisk

Kurzbeschreibung/

Sonstiges: mehrere zusätzliche Tools, Shareware (10 Dollar)

Positiv:

- + resetfest
- + kurze Installationszeit (fünf Sekunden)
- + einfache Installation (ein Befehl)
- + dynamische Speicherverwaltung
- + konventionelle RAM-Disk (»ram:«) bleibt erhalten

Negativ:

- muß nach jedem Reset neu installiert werden

Fazit: äußerst nützlich für jeden, der oft mit wichtigen Daten hantiert

»Amic PD Term« ist ein vielseitiges Terminalprogramm für den Amiga, das alle Standardfunktionen beherrscht. Zusätzlich sind einige interessante Möglichkeiten eingebaut. So findet man beispielsweise einen (ausblendbaren) Menübalken, der sich beliebig auf dem Bildschirm verschieben läßt. Weiter können die Funktionstasten mit oft gebrauchten

Texten belegt werden, über die Menüleiste läßt sich ein Editor laden, mit dem man Texte bearbeiten kann. Ansonsten beherrscht das Programm das XModem-Protokoll. Übertragungsgeschwindigkeiten zwischen 300 und 19200 Baud sind einstellbar. Mit Amic PD Term können Programme durch Up- und Download übertragen werden. Weiterhin ist der Bildschirm zu teilen. In dem Window, das zirka drei Viertel des Bildschirms bedeckt, wird die Übertragung normal dargestellt, während in dem anderen nur die Eingaben des Benutzers erscheinen. Als Zugabe können noch ein CLI-Window eröffnet und komprimierte Files wieder in ihren normalen Zustand versetzt werden.

Programmname: Amic PD Term

Autor: (C) AMIC Development

Kurzbeschreibung/Sonstiges:

Shareware (30 Dollar)

Positiv:

- + PAL-Auflösung
- + verschiebbarer Menübalken für schnelle Funktionswahl
- + F-Tasten sind belegbar
- + Editor ladbar
- + CLI-Window kann eröffnet werden
- + Files können dekomprimiert werden
- + Bildschirm ist teilbar

Negativ:

Fazit: Ein sehr nützliches Programm für jeden DFÜ-Freak

Utilities zum CLI

Jeder, der öfter im CLI arbeitet, kennt das Problem: Man hat ein Programm normal gestartet und damit das aktuelle CLI-Window belegt. Und gerade jetzt muß man einige CLI-Befehle ausführen. In diesem Fall muß das Programm beendet und mit »RUN« neu gestartet werden. Diesen Mißstand behebt das Programm »PopCLI«. Es eröffnet auf eine bestimmte Tastenkombination hin ein CLI-Window, in dem die CLI-Befehle ausgeführt werden können. Zusätzlich ist ein ScreenSaver eingebaut, der den Bildschirm nach einer einstellbaren Zeit, in der keine Eingabe über die Tastatur, die Maus beziehungsweise den Joystick erfolgt ist, ausschaltet. Das sogenannte »Einbrennen« des Monitors wird so ver-

hindert. Besonders interessant ist die Möglichkeit, die Befehlsfolge, die auf die Tastenkombination hin ausgeführt werden soll, zu verändern. Diese Befehlsfolge heißt im Normalfall »NEWCLI >NIL: CON:0/0/640/200/CLI«. Auf diese Art kann man das eine oder andere Programm bereithalten und durch Tastendruck aufrufen.

Programmname: PopCLI
Autor: Dave Baker, Ed Burnette, Jay Denebein, John Mainwaring, Gordon Keener, Jack Rouse, John Toebes, Doug Walker
Kurzbeschreibung/Sonstiges: Screen Saver implementiert
Positiv:
 + Zeit, nach der der ScreenSaver aktiv werden soll, kann bestimmt werden
 + Befehlsfolge für Aufruf läßt sich verändern
Negativ:
 —
Fazit: nützlich für jeden, der mit dem CLI arbeitet

Beim CLI ist es absolut unmöglich, Zeilen zu editieren. Es bleibt einem oft nur noch das Löschen der Zeile. Dies wird durch den »ConMan« behoben. Nach Installation des ConMan ist das Editieren von Zeilen ein Kinderspiel. Weiter kann beispielsweise das Aussehen eines CLI-Windows bestimmt werden, wodurch diese mit »CloseButton« oder ohne »Sizer« keine Utopie mehr sind. Darüber hinaus gibt es zwei verschiedene History-Funktionen, die es erlauben, ältere Zeilen (25 bis 30) zu editieren und wieder zu benutzen. Besonders interessant ist die Möglichkeit, ConMan als »Pipe« zu benutzen. Pipes geben Programmen, die neben- oder nacheinander ablaufen, die Möglichkeit, Daten auszutauschen. Dadurch müssen nicht mehr Unmengen von Dateien eröffnet werden, um Daten zwischenspeichern.

Eine ganze Reihe von Tasten und Tastenkombinationen ist mit bestimmten Funktionen belegt. So kann man zum Beispiel ein Window mit <F1> auf die kleinste Größe verkleinern.

Programmname: ConMan
Autor: William S. Hawes
Kurzbeschreibung/Sonstiges: Shareware
Positiv:
 + Editierung von CLI-Zeilen möglich
 + History-Funktion
 + als Pipe nutzbar
 + Flags der CLI-Windows lassen sich bestimmen
 + praktische Belegung der F- und Ctrl-Tasten
Negativ:
 —
Fazit: Ein Muß für jeden, der mit dem CLI arbeitet

Mit »FuncKey« ist es möglich, die wohl am wenigsten verwendeten Tasten der Amiga-Tastatur zu nutzen: die Funktionstasten. Die F-Tasten können durch FuncKey mit beliebigen Strings bis zu 79 Zeichen Länge belegt werden. Dabei läßt sich jede Taste bis zu fünfmal belegen. Erreicht wird diese Vielzahl durch die Verwendung der F-Tasten in Kombination mit <Ctrl>, <Alt>, <AMIGA-links>, <AMIGA-rechts> und ohne

Kontrolltaste. Wenn man seine Belegungen nun in dem komfortablen Eingabefeld festgelegt hat, kann man diese abspeichern, benutzen oder wieder löschen. Das Eingabefeld kann dann entweder aus dem Speicher gelöscht oder für resident erklärt werden. Wenn das Eingabefeld resident ist, kann es durch eine einfache Tastenkombination aufgerufen werden.

Programmname: FuncKey
Autor: Anson Mah
Kurzbeschreibung/Sonstiges: Shareware (5 Dollar); für 10 Dollar gibt es den Sourcecode
Positiv:
 + komfortable Eingabe
 + es ist möglich, FuncKey resident im Speicher zu behalten
 + bis zu fünf Belegungen pro F-Taste
 + Definition speicherbar
Negativ:
 —
Fazit: für jeden gut geeignet, der viel mit gleichen Strings arbeitet

RUHRSOFT PUBLIC DOMAIN SERVICE

Wir führen mit ca. 750 Disketten das z. Zt. größte Public Domain-Angebot für Ihren Amiga!

RPD	152
Fish	158
Ruhr	15
Panorama	92
Faug	51
Auge	25
TBAG	19
Amicus	26
ACS	90
Ciron Con.	115
Safe	29
CasaMiAmiga	20
UKang	45
ES	60
Kickstart	80

Kopiergebühren:
 Einzelstück je 6,50
 ab 10 Stk. je 6,-
 ab 20 Stk. je 5,50
 ab 30 Stk. je 5,-
 ab 50 Stk. je 4,50

4 randvolle Katalogdisketten gegen DM 12,- in Briefmarken oder V-Scheck anfordern. Versand erfolgt innerhalb 24 Stunden!

Jetzt NEU!

Markus Scheer Telefon
 Kapellenweg 42 0234/411958
 4630 Bochum 5

Wir kopieren auf geprüften 2DD-Disketten, garantiert errorfrei!

MS-DOS und ATARI ST Jetzt NEU! Public Domain Software Jetzt NEU!
 1100 Disketten mit ausgewählter MS-DOS-Software
 450 Disketten mit ausgewählter Atari ST-Software
 Katalog anfordern: Atari ST oder MS-DOS je 5,- DM
 Bitte Computertyp und Diskettenformat (3,5" bzw. 5,25") angeben.
 MS-DOS- und Atari ST-Bestellungen an:
 Sascha Teuber, Bochumer Str. 72, 4630 Bochum 6

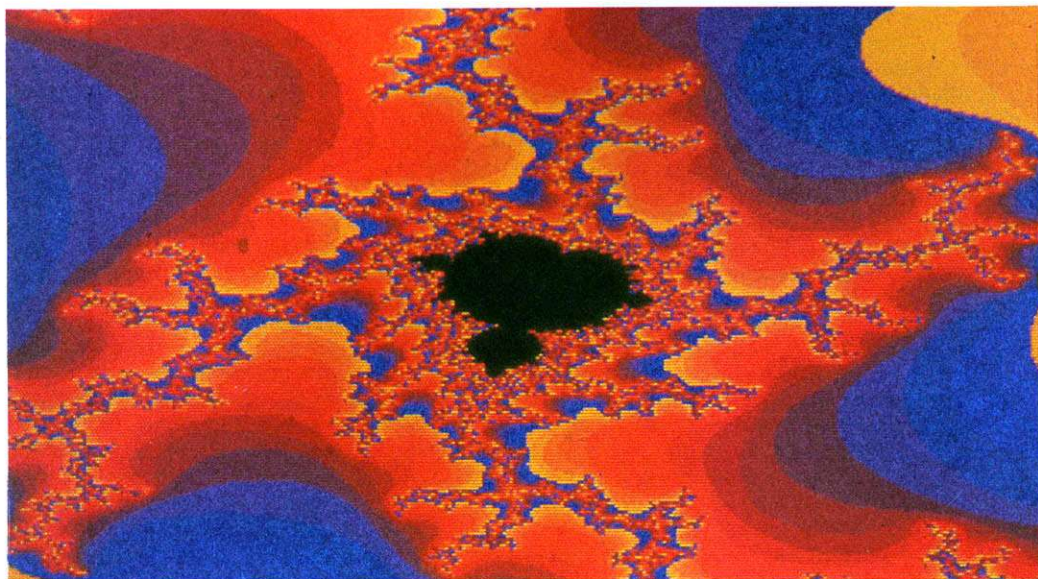


Bild 4. Apfelmännchen in voller Schönheit mit »Mand FXP«

Vielseitige Apfelmännchen

»MandFXP« ist ein Programm zur Berechnung der Mandelbrotbilder (Bild 4), die auch als »Apfelmännchen« bekannt sind. Dabei kann der zu berechnende Ausschnitt entweder von Hand eingegeben oder mit der Maus bestimmt werden. Die Steuerung des Programms erfolgt vollständig über Pull-Down-Menüs. Hier kann man zum Beispiel die Auflösung wählen (320 x 200, 320 x 400, 640 x 200, 640 x 400 Punkte). Ebenso ist die Fen-

stergröße beliebig zu verändern, die Anzahl der Farben (2 bis 32) einzustellen, die Farbpalette zu ändern, abzuspeichern und wieder zu laden. Außerdem können verschiedene Modi des Color-Cyclings eingestellt werden, was mitunter erstaunliche Effekte erzeugt. Bei all diesen Funktionen ist es schon fast selbstverständlich, daß sich die Berechnung der Farben und die Anzahl der Iterationen (Rechentiefe) verändern läßt. Beachtlich ist es, daß man zwischen der zwei- und der dreidimensionalen Darstellung wählen kann. Neben den Mandelbrotgrafiken lassen sich auch die weniger bekannten, aber nicht minder interessanten Julia-Mengen berechnen. Hier bestehen die gleichen reichhaltigen Auswahlmöglichkeiten wie bei den Mandelbrotgrafiken.

Als äußerst nützlich erweist sich die Help-Funktion, die auf Wunsch jede einzelne Funktion der Menüleiste erklärt.

Das wichtigste Bewertungskriterium der Programme zur Erzeugung von Apfelmännchen ist die benötigte Berechnungszeit für ein Bild. Hier ist MandFXP anderen Public Domain-Mandelbrot-Programmen für den Amiga weit überlegen. So dauert zum Beispiel die Berechnung eines vollständigen Apfelmännchens in einer Auflösung von 640 mal 400 Punkten in 16 Farben und 1000 Iterationen nur zirka 12 Minuten. In einer Auflösung von 320 x 200 Punkten in 32 Farben und 30 Iterationen sind es dann nur noch ungefähr 15 Sekunden. Dadurch kann man sich die interessanten Bereiche zuerst in einer niedrigen Auflösung an-

Wir sind nun am Schluß unserer Vorstellung verschiedener interessanter PD-Software angelangt. Sicher können auch Sie eines oder mehrere der beschriebenen Programme gut gebrauchen. Täglich wächst der Bestand an guter, billiger Software für den Amiga — wir werden Sie im AMIGA-Magazin und den Sonderheften ständig auf dem laufenden halten.

(Henning Wolf/
Stefan Rook/
Elke Zumbusch/rs)

sehen und sie dann gegebenenfalls in einer höheren Auflösung berechnen lassen.

MandFXP ist eine Demo-Version. Die endgültige Fassung kann gegen Einsendung von 15 Dollar von den Autoren bezogen werden. Diese Version enthält noch einige Zusatzfunktionen, die es beispielsweise erlauben, die Bilder auszudrucken, im IFF-Format zu speichern und wieder zu laden. Wünschenswert wäre die Möglichkeit, die Bilder im HAM-Modus zu berechnen.

PD-Kopierservice

Alle hier vorgestellten und getesteten Programme gibt es im Paket auf vier Disketten für 30 Mark mit Porto und Verpackung (35 Mark per Nachnahme, 40 Mark Ausland) bei:
CoWaR
Wolf und Rook GbR
Am Hang 5
2054 Geesthacht

Deluxe: Software für den Amiga

Deluxe Paint II (deutsch)/Print I

Dieses Grafikprogramm ist eines der außergewöhnlichsten auf dem Softwaremarkt. Jetzt mit Print I.

Bestell-Nr. 54114

DM 199,-* (sFr 179,-*/öS 1990,-*)

Die ideale Ergänzung zu Deluxe Paint II:

Seasons & Holidays

Bestell-Nr. 52580

DM 29,-* (sFr 26,-*/öS 290,-*)

Deluxe Art Parts II

Bestell-Nr. 52581

DM 29,-* (sFr 26,-*/öS 290,-*)

Deluxe Video 1.2 (deutsch)

Mit Deluxe Video können Sie animierte Grafiksequenzen einfach entwerfen und zusammenstellen.

Bestell-Nr. 52583

DM 249,-* (sFr 225,-*/öS 2490,-*)

Deluxe Photolab (deutsch)

Integriertes Grafikpaket und Druckprogramm mit Posterdruckfunktion und einer Vielzahl weiterer erstaunlicher Funktionen.

Bestell-Nr. 54112

DM 249,-* (sFr 225,-*/öS 2490,-*)

Für alle, die nicht auf die deutsche Version warten wollen:

Deluxe Photolab (englisch)

Bestell-Nr. 54117

DM 199,-* (sFr 179,-*/öS 1990,-*)

Deluxe Music (deutsch)

Das professionelle Musikprogramm. Jetzt mit deutscher Software.

Bestell-Nr. 52579

DM 199,-* (sFr 179,-*/öS 1990,-*)

Die ideale Ergänzung zu Deluxe Music:

It's only Rock'n'Roll

Bestell-Nr. 54115

DM 29,-* (sFr 26,-*/öS 290,-*)

Hot & Cool Jazz

Bestell-Nr. 54116

DM 29,-* (sFr 26,-*/öS 290,-*)

Deluxe Productions (englisch/NTSC)

Bestell-Nr. 54113

DM 399,-* (sFr 359,-*/öS 3990,-*)

Updates von der deutschen Version:

Paint II, Bestell-Nr. 54114U

Video 1.2, Bestell-Nr. 52583U

Photolab, Bestell-Nr. 54112U

je DM 49,-* (sFr 49,-*/öS 490,-*)

Gegen Einsendung der Originaldiskette und gegen Vorkasse.

In Vorbereitung:

Deluxe Print II (deutsch)

Bestell-Nr. 52582

DM 199,-* (sFr 179,-*/öS 1990,-*)

* Unverbindliche Preisempfehlung

Fragen Sie Ihren Händler nach weiteren Informationen.

Markt & Technik-Support:

Bei User-Registrierung rechtzeitige Update-/Upgrade-Information und Support-Unterstützung. Senden Sie uns bitte Ihre Registrierungskarte.



DELUXE:

Software für den Amiga.

Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computer-Fachgeschäften oder bei Ihrem Buchhändler.

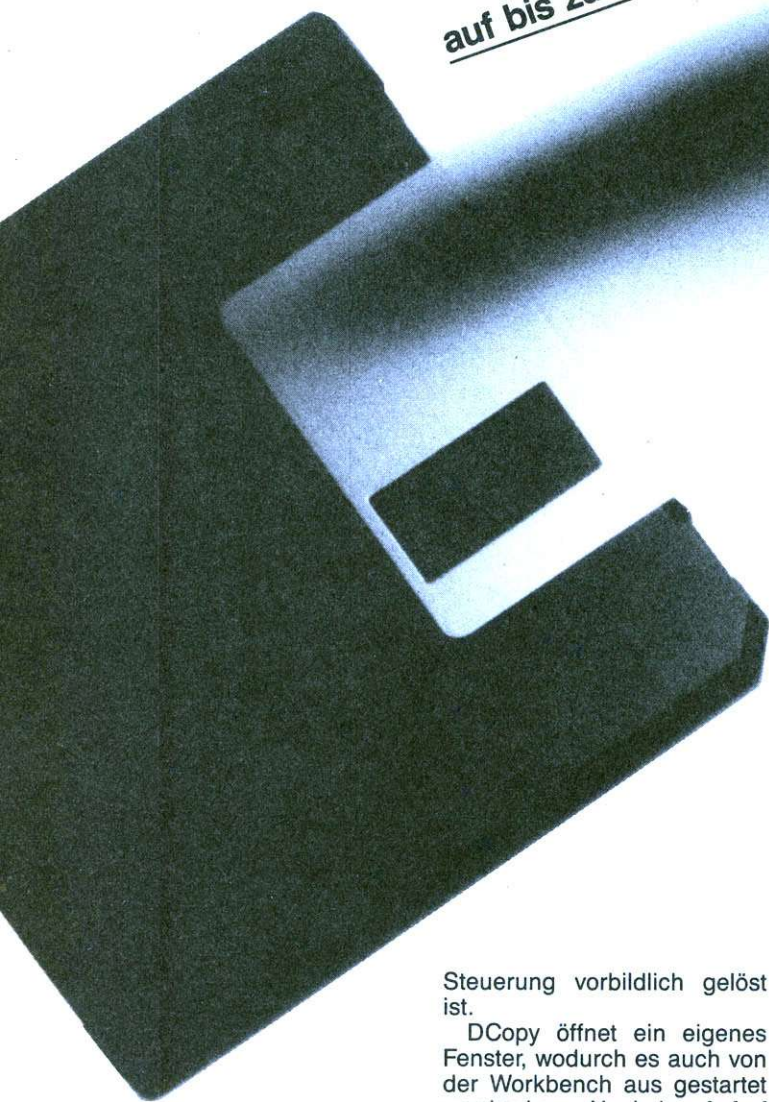
Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 46 13-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 41 56 56. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 587 1393-0; Rudolf Lechner & Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 67 75 26; Ueberreuter Media Verlagsges.m.bH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 48 15 43-0.

Das Programm »Diskcopy« von der Workbench ist zum Kopieren recht unkomfortabel. Wir bieten Ihnen eine vielseitige Alternative zu dieser Routine. Sogar das mehrfache Kopieren auf bis zu drei Laufwerke gleichzeitig beherrscht »DCopy«



D Kopieren

Darunter steht, ob das Mehrfachkopieren eingeschaltet ist (»Multiple copy OFF«). Ist dieser Modus eingeschaltet, können mehrere Kopien erstellt werden, wobei die Daten nur einmal geladen werden. Auch hierbei ist es möglich, auf mehrere Zieldisketten in einem Durchlauf zu kopieren.

Sehr interessant ist die Fähigkeit von DCopy, nicht nur die gesamte Diskette, sondern auch beliebige Teile davon zu duplizieren. »StartTrack« und »EndTrack« sind die Spuren, von beziehungsweise bis zu denen der Kopiervorgang laufen soll. Mit »Surface« gibt man an, welche Diskettenseite vervielfältigt werden soll. Auf die Bedeutung dieses Parameters gehen wir bei der Beschreibung des Untermenüs ein.

Durch Drücken von <F1> gelangen Sie in ein Untermenü, in dem die voreingestellten Werte editierbar sind. Um zu dem Parameter zu gelangen, den Sie ändern wollen, benutzen Sie die Tasten <Cursor aufwärts> und <Cursor abwärts>.

Bei »VERIFY« werden nur die Tasten <y> und <n> akzeptiert, die Verify an- beziehungsweise ausschalten.

Mit »TRIES« wird bestimmt, wie viele Schreibversuche unternommen werden, wenn ein Schreibfehler auftritt. Diese Funktion ist somit nur dann sinnvoll, wenn »VERIFY« aktiviert ist.

Für mehrfache Kopien bei nur einmaligem Lesen ist der Parameter »MULTIPLE COPY« zuständig. Genau wie bei VERIFY kann nur <y> oder <n> eingegeben werden. Positiv ist dieser Menüpunkt, wenn man nur über ein Laufwerk verfügt, da man zeitsparend mehrere Kopien nacheinander erstellen kann.

»STARTTRACK« und »ENDTRACK« geben die Start- und Endspur für den Kopiervorgang an. Falls die Startspur größer als die Endspur ist, kann man das Menü nicht verlassen. Um nur eine Spur zu kopieren, stellt man einfach beide Werte auf die gewünschte Zahl ein.

Die größtmögliche Flexibilität erhält man zusätzlich durch die Auswahl der Diskettenseite. Mit »Head« läßt sich bestimmen, welche Seite kopiert werden soll. Dabei bedeuten:

- 0 - Nur Seite 0
- 1 - Nur Seite 1

Steuerung vorbildlich gelöst ist.

DCopy öffnet ein eigenes Fenster, wodurch es auch von der Workbench aus gestartet werden kann. Nach dem Aufruf erscheinen in dem Fenster die voreingestellten Werte.

In der ersten Zeile steht nach dem Starten »Verify OFF«. Dies bedeutet, daß nach dem Schreiben der Daten keine Überprüfung auf Richtigkeit durchgeführt wird. In diesem Modus erhöht sich natürlich die Kopiergeschwindigkeit, da die Daten nicht noch einmal gelesen werden müssen.

Das in der Programmiersprache C geschriebene Programm DCopy (Listing 1) läßt beim Kopieren keine Wünsche mehr offen. Bis zu vier Laufwerke erkennt DCopy automatisch und benutzt sie auch bei Bedarf. Die gewünschten Parameter sind schnell und leicht eingestellt. Alle Eingaben erfolgen über die Tastatur, wobei die Cursor-

Copy – mit Komfort

2 - Beide Seiten

Die Angabe, wo sich die Quelldiskette befindet, erfolgt beim Menüpunkt »Source«. Die Anzahl der benutzbaren Ziellaufwerke ist abhängig von der Zahl angeschlossener Laufwerke. Maximal werden jedoch vier Laufwerke verwaltet. Die Ziellaufwerke 2 und 3 können durch die Eingabe von <n> abgeschaltet werden. Zum Einschalten gibt man einfach die Nummer des Laufwerks ein. Werden für Quell- und Ziellaufwerk dieselbe Zahl eingegeben, schaltet das Programm automatisch alle anderen Ziellaufwerke ab.

Mit <ESC> kann nach erfolgter Einstellung das Untermenü verlassen werden. Geschieht nach dem Drücken von <ESC> nichts, wurde ein Parameter falsch gewählt.

Der Start des Kopiervorgangs erfolgt mit <F10>. Nun zeigt DCopy, welche Diskette wo einzulegen ist und wartet auf das Drücken von <RETURN>. Wird mit nur einem Laufwerk gearbeitet, berechnet DCopy die nötige Anzahl der Diskettenwechsel und gibt diese auf dem Bildschirm aus.

Beim Kopieren zeigt DCopy die bearbeitete Spur an, Lese- und Schreibfehler werden mitgeteilt. Kann die Diskette nicht kopiert werden (Abbruch oder Verify-Error), erscheint eine Meldung.

Um DCopy zu verlassen, genügt ein Druck auf <ESC>. Vorher gibt DCopy jedoch noch die Anweisung, sämtliche Kopien aus den Laufwerken zu entfernen. Dies ist nötig, da die Kopien identisch zum Original sind und Amiga-DOS Probleme bekäme. Genau dasselbe tritt auf, wenn sich vor dem Start von DCopy zwei exakt

gleiche Disketten in den Laufwerken befinden. In so einem Fall müssen Sie erst das Programm starten und dann die Disketten einlegen.

Nach der Beendigung von DCopy können Sie selbstverständlich ganz normal mit Ihrem Amiga weiterarbeiten. Und das viel ruhiger, da Sie ja dann Sicherheitskopien von Ihren Programmen und Daten besitzen. (Gerald Fries/rs)

Programmname:	DCopy
Computer:	A500, A1000, A2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec-C V3.40
Aufrufe:	cc dcopy+l-s n dcopy.o-lm32-cc32

```
1 ej0 #include <devices/console.h>
2 Qf #include <intuition/intuition.h>
3 CN #include <stdio.h>
4 h4 #include <devices/trackdisk.h>
5 CH #include <exec/memory.h>
6 4r #include <libraries/dos.h>
7 Oy #include <libraries/dosextens.h>
8 Ic #define BLOCKSIZE 5632L /* BLOCKSIZE*NUMSECS (512*1
   1) */
9 T9 #define YES 1
```

Listing 1.
»DCopy«, ein komfortables
Kopierprogramm für den Amiga.
Bitte mit dem Checksummer
(Seite 159) eingeben.


```

10 bc #define NO 0
11 lj #define FAILED -10 /* operation-failed flag */
12 kc #define OK 0
13 zt struct Process *mp,*FindTask();
14 ok struct MsgPort *diskport[4];
15 xm struct IOExtTD *diskreq[4];
16 fg BYTE *erase,*puffmerk,*fastdiskpuffer[160],*diskpuffer[80],
    *AllocMem();
17 ip struct IntuitionBase *IntuitionBase,*OpenLibrary();
18 zl extern struct MsgPort *CreatePort();
19 sa extern struct IORequest *CreateExtIO();
20 ut struct NewWindow nw = {
21 ft8 0, 10, /* starting position (left,to
    p) */
22 iq 640,100, /* width, height */
23 lg -1,-1, /* detailpen, blockpen */
24 ce 0, /* flags for idemp */
25 c4 WINDOWDEPTH |
26 eq WINDOWDRAG |
27 ot NOCAREREFRESH |
28 gf ACTIVATE, /* window gadget flags */
29 n4 0L, /* pointer to 1st user gadget
    */
30 bh NULL, /* pointer to user check */
31 gn (UBYTE *) "DCopy", /* title */
32 xo NULL, /* pointer to window screen */
    /
33 qd NULL, /* pointer to super bitmap */

34 ge 0,0, /* min width, height */
35 xx 0,0, /* max width, height */
36 ix WBENCHSCREEN};
37 pv0 struct Window *w,*OpenWindow();
38 ia struct IOStdReq *WMsg; /* I/O request block
    pointer */
39 bb struct MsgPort *consoleWritePort; /* a port at which t
    o receive */
40 jd struct IOStdReq *RMsg; /* I/O request block
    pointer */
41 wx struct MsgPort *consoleReadPort; /* a port at which t
    o receive */
42 7m extern struct IOStdReq *CreateStdIO();
43 yr char readstring; /* provides a buffer even though using o
    nly one char */
44 qq char string[50]; /* provides a buffer for writing variabl
    es */
45 qa int error = 0;
46 u5 int avail = 0;
47 2p int fastavail = 0;
48 vo int anzdisk = 0;
49 jd /* verify,maxtry,multiple,starttrack,endtrack,mode,source,
    3x destination */
50 sl int flag[10] = {'n',1,'n',0,79,2,0,-1,'n','n'};
51 vd int t[4] = {-1,-1,-1,-1}; /* remember OpenDevice
    failures */
52 xt main()
53 ng {
54 sw3 int i,succcess,fail = 0;
55 qd long dos_packet();
56 yk if((IntuitionBase=OpenLibrary("intuition.library",0))==N
    ULL)
57 rk6 exit(99L);
58 uv3 consoleWritePort = CreatePort("my.con.write",0L);
59 xe if(consoleWritePort == 0)
60 un6 exit(99L);
61 vj3 WMsg = CreateStdIO(consoleWritePort);
62 ah if(consoleWritePort == 0)
63 xq6 exit(99L);
64 qy3 consoleReadPort = CreatePort("my.con.read",0L);
65 vu if(consoleReadPort == 0)
66 ot6 exit(99L);
67 kk3 RMsg = CreateStdIO(consoleReadPort);
68 mb if(RMsg == 0)
69 3w6 exit(99L);
70 re3 w = OpenWindow(&nw); /* create a window */
71 jo if(w == NULL)
72 6z6 exit(99L);
73 203 error = OpenConsole(WMsg,RMsg,w);
74 vu if(error != 0)
75 9c6 exit(99L);
76 tp /* attach a console to this window, initialize
    * for both write and read */
77 vg7

```

Listing 1. (Fortsetzung)

**BESUCHEN SIE MARKT&TECHNIK
AUF FOLGENDEN MESSEN:**
Frankfurter Buchmesse, 5.-10. Okt., Halle 6.2, Stand A 449A 451/B 440
ORGATECHNIK Köln, 20.-25. Okt., Halle 3.1/Gang MN/Stand 12/11
SYSTEC München, 25.-28. Okt., Halle 16/Stand B 18
ELECTRONICA München, 8.-12. Nov., Halle 18/Stand 18A10
Markt&Technik Verlag AG, Hans-Pinsel-Str. 2, 8013 Haar bei München

Stoff für Ihren Amiga

CLimate 1.2 für Amiga

Mit CLimate 1.2 können Sie endlich die Befehle des Command-Line-Interface benutzerfreundlich per Mausklick verwenden! Eine sehr übersichtliche Bildschirm-darstellung, die Bedienung aller Befehle mit der Maus und die Unterstützung von drei externen Laufwerken (3 1/2" oder 5 1/4"), Festplatten, RAM-Disk machen das Programm zu einer unentbehrlichen Unterstützung.

Bestell-Nr. 51653

DM 79,-* (sFr 72,-/öS 790,-*)

Devpac Assembler (deutsch) für Amiga

Ein Entwicklungspaket mit integriertem Editor/Assembler, symbolischem Debugger und schnellem Linker zum Einbinden von Hochsprachen-Modulen. Erzeugt direkt ausführbare Programme!

Bestell-Nr. 51656

DM 148,-* (sFr 134,-/öS 1480,-*)

Zing! (deutsch)

- das mächtige CLI-Werkzeug für den Amiga

Mit Zing! haben Sie endlich das gesamte File-System mit Directories und Subdirectories fest im Griff. Sie beschleunigen mit Zing! und Sie verwalten bis zu 500 Files und Subfiles und bis zu 100 Directories auf einmal. Die Bedieneroberfläche ist vom Feinsten: Pull-down-Menüs, (Click-)Icons, Funktionstasten.

Bestell-Nr. 51669

DM 99,-* (sFr 89,-/öS 990,-*)

Zing! Keys (deutsch)

- Ihr ganz persönlicher Amiga

Mit Zing! Keys machen Sie aus Ihrem Amiga das variable System, das Sie sich schon immer wünschen. Es ist Ihren eigenen Ansprüchen jederzeit anpassbar! Alle Tasten sind nach Wunsch belegbar: z.B. mit Funktionsaufrufen, Programmaufrufen, Systembefehlen und vorprogrammierten Befehlen. Die Belegung ist natürlich jederzeit abspeicherbar.

Durch die Belegung von »Hot-Keys« haben Sie mit Zing! Keys ein Multitaskingsystem par excellence!

Bestell-Nr. 51670

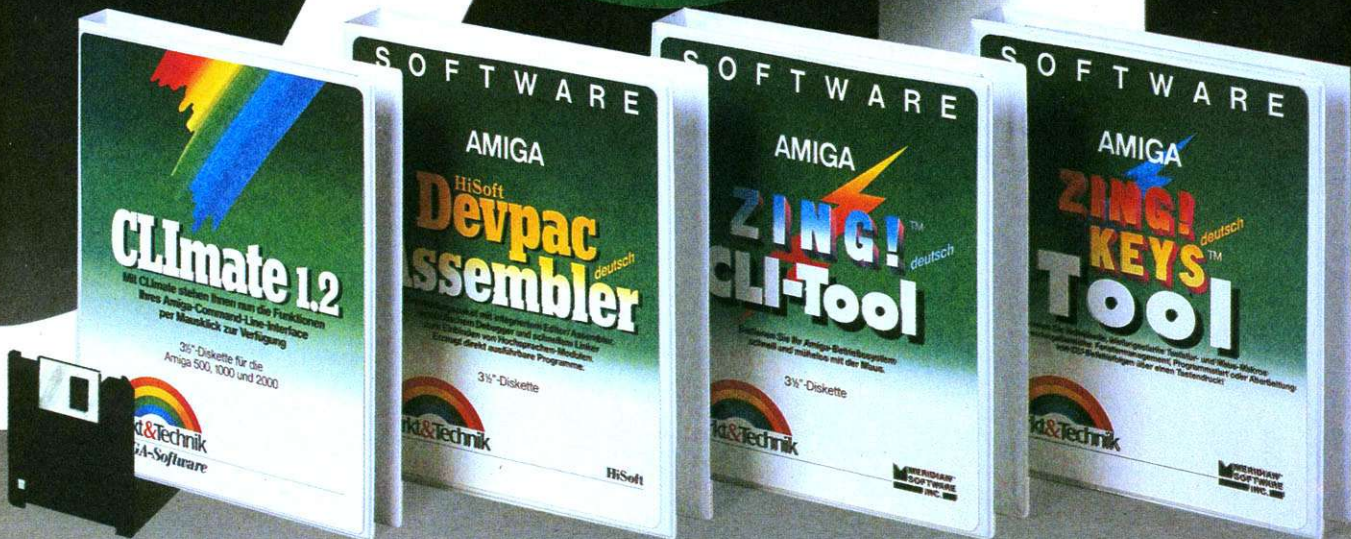
DM 79,-* (sFr 71,-/öS 790,-*)

* Unverbindliche Preisempfehlung

**Fragen Sie Ihren Händler
nach weiteren Informationen.**

Markt&Technik-Support:

Bei User-Registrierung rechtzeitige Update-/Upgrade-Information und Support-Unterstützung. Senden Sie uns bitte Ihre Registrierungskarte.



Stoff für Ihren Amiga



Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computer-Fachgeschäften oder bei Ihrem Buchhändler.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 46 13-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 41 56 56. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5 87 13 93-0; Rudolf Lechner&Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 67 75 26; Ueberreuter Media Verlagsges.m.bH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 48 15 43-0.


```

78 ID3 QueueRead(RMsg,&readstring); /* tell console w
here to
79 9fh
80 Zq
81 ru
82 Nm
83 EY3 CursorOff(); /* cursor will not b
e displayed */
84 3pJ /* How many drives are connected ? */
85 nL3 for (anzdisk=0;anzdisk<4;anzdisk++) {
86 rr6 if ((diskport[anzdisk] = CreatePort(OL,OL)) == 0) {
87 UY9 puts("\nCan't create port.\n");
88 Zf exit(OL);
89 Rw6 }
90 km diskreq[anzdisk] = (struct IOExtTD *)
91 CwE CreateExtIO(diskport[anzdisk],(long)sizeof(st
ruct IOExtTD));
92 2x6 if (OpenDevice(TD_NAME,(long)anzdisk,diskreq[anzdisk]
,OL)) {
93 tr9 DeleteExtIO(diskreq[anzdisk],(long)sizeof(struct I
OExtTD));
94 CI DeletePort(diskport[anzdisk]);
95 EC t[fail++]=anzdisk+'0';
96 Y36 }
97 Ka else {
98 UK9 if (flag[7]==-1 && anzdisk!=0)
99 FUC flag[7]=anzdisk;
100 D4K /* do not verify disk-changes */
101 CB9 mp=FindTask("File System");
102 mO dos_packet(&mp->pr_MsgPort,ACTION_INHIBIT,TRUE,OL
,OL,OL,OL,OL,OL);
103 fA6 }
104 gB3 }
105 nL anzdisk--fail; /* amount of connected drive
s */
106 qP if (anzdisk==1) /* only one drive connected
*/
107 lq6 flag[7]=0;
108 6w3 do {
109 ra6 error = 0;
110 q7 if (flag[0]=='y') {
111 Qh9 if (flag[1]==1) {
112 HOC locate (2,30);
113 DJ sprintf(string,"Verify ON with %d try",flag[1])
;
ConPutStr(WMsg,string);
114 2y }
115 rM9 }
116 dt else {
117 lOC locate (2,29);
118 3I sprintf(string,"Verify ON with %d tries",flag[1]
);
ConPutStr(WMsg,string);
119 73 }
120 wR9 }
121 xS6 }
122 jz else {
123 tu9 locate(2,35);ConPutStr(WMsg,"Verify OFF");
124 OV6 }
125 NP locate(3,32);ConPutStr(WMsg,"Multiple copy ");
126 xd if (flag[2]=='y')
127 PR9 ConPutStr(WMsg,"ON ");
128 tg6 else
129 zV9 ConPutStr(WMsg,"OFF");
130 N16 locate(4,20);
131 gy sprintf(string,"StartTrack: %d, EndTrack: %d, Surface
: %d",
flag[3],flag[4],flag[5]);
132 sYM ConPutStr(WMsg,string);
133 LH6 locate (7,7);ConPutStr(WMsg,"press <F1> to change p
arameters.");
134 hf locate (7,48);ConPutStr(WMsg,"press <F10> to begin
copy.");
135 qs locate (9,30);ConPutStr(WMsg,"press <ESC> to quit."
);
136 jf );
137 Vg i=Taste();
138 8C if (i==155)
139 VL9 i=ConGetChar(consoleReadPort,RMsg,&readstring);
140 Yo6 if (i==48) {
141 dZ9 ConGetChar(consoleReadPort,RMsg,&readstring);
142 qq success=Menu();

```

Listing 1. (Fortsetzung)

Video-Scape 3D - Berechnete Realität

Mit Video-Scape 3D können Sie dreidimensionale Objekte aus verschiedenen Blickwinkeln betrachten und durch Hinzufügen von Kamerafahrten und frei wählbarem Lichteinfall einen realistischen Computer-Videofilm erstellen.

Bestell-Nr. 51671

DM 385,-* (sFr 345,-/öS 3850,-*)

Aegis Images -

Farbenpracht leichtgemacht

Ein ideales Standard-Zeichenprogramm mit über 4000 Farben.

Bestell-Nr. 54108

DM 69,-* (sFr 62,-/öS 690,-*)

Aegis Animator und Images - Bringt Bewegung in Ihre Bilder

Aegis Animator verbindet drei Animationstechniken, damit Sie vielseitige Desktop-Video-Produktionen auf dem Amiga erzeugen können!

Bestell-Nr. 54109

DM 249,-* (sFr 225,-/öS 2490,-*)

Aegis Draw

Ein leicht zu bedienendes Zeichenprogramm.

Bestell-Nr. 54106

DM 199,-* (sFr 179,-/öS 1990,-*)

Aegis Draw Plus - CAD: Ein Traum wird erschwinglich

Computerunterstütztes Konstruieren von strukturierten Grafiken.

Bestell-Nr. 54107

DM 385,-* (sFr 345,-/öS 3850,-*)

Aegis Video-Titler -

Verwandeln Sie Ihren Amiga in eine leistungsfähige Titelmaschine

Ein Text- und Grafikgenerator.

Bestell-Nr. 54101

DM 249,-* (sFr 225,-/öS 2490,-*)

Aegis Impact - Verleihen Sie Ihrer Präsentation einen Ausdruck, der Eindruck macht

Das Desktop-Programm zur Erstellung von Präsentationsgrafiken.

Bestell-Nr. 54104

DM 149,-* (sFr 135,-/öS 1490,-*)

Aegis Sonix - Wetten, daß

auch Sie mit diesem Programm einen Hit schreiben?

Synthesizer- und Kompositionsprogramm.

Bestell-Nr. 54105

DM 149,-* (sFr 135,-/öS 1490,-*)

Aegis AudioMaster -

Das Tonstudio für den Amiga

Nachbearbeitung von digitalisierten Klängen.

Bestell-Nr. 54103

DM 99,-* (sFr 89,-/öS 990,-*)

Aegis Diga -

Der Schlüssel zur modernen Datenkommunikation

Telekommunikationssoftware.

Bestell-Nr. 54102

DM 149,-* (sFr 135,-/öS 1490,-*)

* Unverbindliche Preisempfehlung

Markt & Technik-Support:

Bei User-Registrierung rechtzeitige Update-/Upgrade-Information und Support-Unterstützung. Senden Sie uns bitte Ihre Registrierungskarte.



AEGIS: Kreative Software für den Amiga.

Markt&Technik-Produkte
erhalten Sie in den Fachabteilungen
der Warenhäuser, im Versandhandel,
in Computer-Fachgeschäften oder
bei Ihrem Buchhändler.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler
nach unserem kostenlosen Gesamtverzeichnis
mit über 500 aktuellen Computerbüchern
und Software. Oder fordern Sie es direkt
beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 41 56 56. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5 87 13 93-0; Rudolf Lechner & Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 67 75 26; Ueberreuter Media Verlagsges.m.b.H (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 48 15 43-0.


```

143 Jo6    }
144 QT      else if (i==57) {
145 Kf9      locate(1,2);
146 kr      ConPutStr(WMsg, "
                "
                ");
147 uNO      locate (7,7);ConPutStr(WMsg, "
                ");
148 MF9      locate(7,48);ConPutStr(WMsg, "
                ");
149 6q      locate(9,30);ConPutStr(WMsg, "
                ");
150 Rz      ConGetChar(consoleReadPort,RMsg,&readstring);
151 nj      success=Copy();
152 Q1      }
153 Ty6      else if (i==27)
154 zW      End();
155 FB9      CursorOff();
156 qk6      ConPutStr(WMsg, "\f");
157 BF      if (success==FAILED) {
158 6f      locate(1,25);
159 oU9      ConPutStr(WMsg, "\x9b0;33;40mATTENTION:   Copy fa
160 ox      iled !!!\x9bm");
161 b66      }
162 ZJ3      } while (1<2); /* for ever !! */
163 d80      }
164 OU      Menu()
165 b4      {
166 pj3      int zeile = 8;
167 9h      int i,k,l,spalte,j;
168 zo      int success = OK;
169 PJ      ConPutStr(WMsg, "\f");CursorOn();
170 Oc      locate(1,17);ConPutStr(WMsg, "Use UP- and DOWN-keys to se
                lect, "
                "
                "ESC to return.");
171 Y5V      locate (4,15);ConPutStr(WMsg, "VERIFY");locate(4,25);
172 rz3      if (flag[0]=='y')
173 WK      ConPutStr(WMsg, "YES");
174 B36      else
175 eR3      ConPutStr(WMsg, "NO ");
176 DF6      locate (5,16);ConPutStr(WMsg, "TRIES");
177 FT3      locate(5,25);sprintf(string, "%d", flag[1]);ConPutStr(WMsg
                ,string);
178 JK      locate (6,8);ConPutStr(WMsg, "MULTIPLE COPY");locate(6,25
                );
179 zB      if (flag[2]=='y')
180 pV      ConPutStr(WMsg, "YES");
181 IA6      else
182 LY3      ConPutStr(WMsg, "NO ");
183 KM6      locate(8,10);ConPutStr(WMsg, "START TRACK");
184 IX3      locate(8,25);sprintf(string, "%d", flag[3]);ConPutStr(WMsg
                ,string);
185 ms      locate(9,12);ConPutStr(WMsg, "END TRACK");
186 Nr      locate(9,25);sprintf(string, "%d", flag[4]);ConPutStr(WMsg
                ,string);
187 x5      locate(3,45);ConPutStr(WMsg, "<0> HEAD 0");
188 VX      locate(4,45);ConPutStr(WMsg, "<1> HEAD 1");
189 hm      locate(5,45);ConPutStr(WMsg, "<2> HEAD 0/1");
190 nM      locate(5,60);sprintf(string, "%d", flag[5]);ConPutStr(WMsg
                ,string);
191 4o      locate(7,51);ConPutStr(WMsg, "SOURCE");
192 JB      locate(7,60);sprintf(string, "df%d:", flag[6]);ConPutStr(W
                Msg,string);
193 PX      k=1;
194 Ec      do {
195 VL      locate(zeile,43);sprintf(string, "%d. DESTINATION", k);
196 Sg6      ConPutStr(WMsg,string);
197 NJ      if (k>1 && (flag[6+k]=='n' || flag[6+k]==62)) {
198 ze      locate(zeile,60);
199 IS9      ConPutStr(WMsg, "    Not wanted    ");
200 ND      }
201 Fk6      else {
202 LH      locate(zeile,60);
203 mW9      sprintf(string, "df%d:", flag[k+6]);ConPutStr(WMsg,s
                tring);
204 Us      }
205 Jo6      zeile++;
206 i1      } while (++k<anzdisk);
207 M83      /* Editor */
208 DF      i=0;
209 Lg      for (;;) {
210 6S      start:
211 vz0      switch (i) {
212 Ha6      case 0:
213 Rh9

```

```

214 qHC      zeile=4;spalte=25;break;
215 Vm9      case 1:
216 avC      zeile=5;break;
217 Zr9      case 2:
218 f1C      zeile=6;break;
219 dw9      case 3:
220 nBC      zeile=8;break;
221 h19      case 4:
222 DjC      zeile=9;spalte=25;break;
223 169      case 5:
224 wNC      zeile=5;spalte=60;break;
225 pB9      case 6:
226 AfC      zeile=7;spalte=62;break;
227 tG9      case 7:
228 F1C      zeile=8;spalte=62;break;
229 xL9      case 8:
230 KrC      zeile=9;spalte=62;break;
231 1Q9      case 9:
232 9SC      zeile=10;spalte=62;break;
233 1G6      }
234 aR      locate (zeile,spalte);
235 5w      j=ConGetChar(consoleReadPort,RMsg,&readstring);
236 mn      if (j!=155) {
237 U39      if (j==66 || j==13) /* CRSR down
                , <CR> */
238 eaC      i++;
239 WS9      else if (j==65) /* CRSR up
                */
240 suC      i--;
241 1k9      else if (j==27) { /* ESC */
242 biC      if (flag[3] > flag[4]) { /* starttra
                ck > endtrack? */
243 9YF      i=4;
244 OW      goto start;
245 xSC      }
246 CI      else if (flag[6]==flag[7]) { /* source=d
                estination */
247 H4F      for (j=2; j<4; j++) /* unselect
                other drives */
248 wRI      flag[6+j]='n';
249 MvF      return(OK);
250 2XC      }
251 cb      else if (anzdisk>2) { /* more than two dr
                ives connected */
252 KvF      for (j=6; j<6+anzdisk; j++)
253 ucI      for (l=1; l<4; l++)
254 x9L      if (flag[j]!='n' && flag[j]!=62 && l+j
                <10)
255 se0      if (flag[j]==flag[l+j]) {
256 iQR      i=l+j;
257 bj      goto start;
258 Af0      }
259 BgC      }
260 X6      return(OK);
261 Di9      }
262 zF      else {
263 nLC      if (i==1) {
264 bnF      if (j>'0' && j<='9') {
265 FWI      flag[i]=j-'0'; /* integer
                */
266 kS      sprintf(string, "%d", flag[i]);
267 VR      ConPutStr(WMsg,string);
268 KpF      }
269 Ax      else
270 MOI      i--;
271 NsC      }
272 1Q      else if (i==5) {
273 d4F      if (j>'0' && j<='2') {
274 OfI      flag[i]=j-'0'; /* integer
                */
275 tb      sprintf(string, "%d", flag[i]);
276 ea      ConPutStr(WMsg,string);
277 TyF      }
278 J6      else
279 VXI      i--;
280 W1C      }
281 ia      else if (i==6 || i==7) { /* 1st & 2nd dri
                ve */
282 JSF      if (j>'0' && j<'4' && j!=t[0] && j!=t[1] && j!=t[2]
                && j!=t[3]) {
283 XoI      flag[i]=j-'0'; /* integer
                */

```

Listing 1. (Fortsetzung)

500er Speichererweiterung

Für 512k zusätzliches RAM • alle RAM's gesockelt • selbstkonfigurierend • abschaltbar • Uhrenschaltung auf Platine mit Akku- bzw. Batteriepufferung nachrüstbar

Komplett mit 512k

Preis auf Anfrage

Superpreis mit Uhr

Preis auf Anfrage

Bauteilesatz für Uhr ohne Akku

24.-

Leiterplatte mit Stecker

*39.-

*mit Schaltplan und Bestückungsliste

Profilaufwerk 3,5"

Metallgehäuse • einstellbare Laufwerknummer mit Displayanzeige • digitale Trackanzeige • Write Protect am Laufwerk schaltbar • abschaltbar • durchgeschleifter Bus • 1 Jahr Garantie

SuperALCOMPPreis

329.-

Trackanzeige

Für DF0-DF3 einstellbar • für alle Laufwerke (3,5"/5,25") • Laufwerkbus durchgeschleift • mit Gehäuse

SuperALCOMPPreis

69.-

3,5" Laufwerk

Für alle Amiga's • einstellbare Gerätenummer • abschaltbar • Metallgehäuse • superflach • 1 Zoll (2,54 cm) • durchgeschleifter Bus • TEAC Laufwerk • 1 Jahr Garantie

komplett anschlussfertig

239.-

Amigafarbende Blende

+ 10.-

Basislaufwerke

1 Jahr Garantie

TEAC FD 135 FN 3,5" 1MB superslimline

218.-

TEAC FD 55 FR 5,25" 40/80 Tracks 1MB

239.-

TEAC FD 55GFR 5,25" 40/80 Tracks

Amigafarbende Blende

+ 10.-

1,6MB Diskchange

259.-

3,5" Gehäuse

25.-

5,25" Gehäuse

25.-

Gehäuse für "Gemischtes Doppel"

65.-

Laufwerkanschlusskabel

Zum Anschluß von Laufwerken an alle Amiga's • mit Ansteuerelektronik

Für 3,5" Laufwerk

39.-

Für 5,25" Laufwerk

49.-

Steckplatzerweiterung 3-fach für Laufwerke

Jeder Steckplatz abschaltbar und einstellbare Laufwerksnummer • Steckplatzerweiterung direkt am Amigagehäuse • Dadurch keine Kabellängenprobleme

Anschlussfertig zum SuperALCOMPPreis

49.-

Laufwerk 5,25"

40/80 Track • Laufwerksbus durchgeschleift • abschaltbar • einstellbare Adressen • MS-DOS-kompatibel • mit Diskchange

SuperALCOMPPreis

298.-

HD 1,6 MB (umschaltbar)

318.-

Amigafarbende Blende

+ 10.-

Gemischtes Doppel 3,5/5,25"

einzelne ein-/abschaltbar • einstellbare Laufwerksnummern mit Anzeige

• durchgeschleifter Bus • bei 5,25" 40/80 Tracks umschaltbar • Metallgehäuse • 1 Jahr Garantie

SuperALCOMPPreis

598.-

Userport + Experimentierkarte für Expansionport

Mit Lochraste und 2 x 6522 Ports

Leer

komplett aufgebaut

Soundsampler

Für alle Amiga's mit Software • Type bei Bestellung bitte angeben • 8-Bit Datenbreite • Betrieb am Parallelport (Druckerport) • Mit Vorverstärker für Micro-Anschluß (Cinch-Buchsen) • Musik- und Sprachdigitalisierung möglich • Arbeitet mit fast allen Digitizer-Programmen • Form-schönes Gehäuse

SuperALCOMPPreis

79.-

ausgereifte Ingenieurleistung • 14 Tage Umtauschrecht • 2 Jahre Garantie • fast alle IC's gesockelt • nur professionelle Leiterplatten • Bauteile namhafter Hersteller • mit Bedienungsanleitung

MIDI-Interface

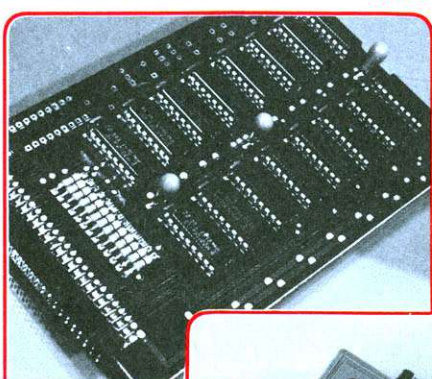
4 Kanäle einschließlich 1 Thru • Optische Datenanzeige • Formschönes Gehäuse

Wahnsinnspreis von nur

89.-

Bootselector

19.90



Kickstartumschaltung

Bauen Sie die anderen Kickstart-Versionen in Ihren Amiga 500 • Einfacher Einbau ohne Löten • für Original-Kickstart-ROM und 2 zusätzliche Versionen auf EPROM • EPROM-Programmierservice auf Anfrage

SuperALCOMPPreis

59.-

Kickstartversion auf EPROM's

129.-

Public-Domain-Disketten

Große Auswahl • schnelle Auslieferung

Einzelstück

6.-

Staffelpreise auf Anfrage

Wir suchen ständig Hardware-Entwicklungen. Wir garantieren gute Umsatzprovisionen und ehrliche Abrechnung

kostenloses Info anfordern!!!

Bestellung und Versand

ALCOMP

A. Lanfermann

Lessing Str. 46

5012 Bedburg

Tel. 0 22 72/15 80

Nachnahmeversand NN-Spesen 7.50 DM b. Vorkasse 3.- DM. Auslandsbestellungen: Nachnahmeversand NN-Spesen 10.- DM b. Vorkasse 5.- DM. Wir liefern Ihnen auf Ihre Rechnung und Gefahr zu den Verkaufs- und Lieferbedingungen des Elektronikgewerbes. Postgiroamt Köln (BLZ 370 100 50) 275 54-509

Neu!!!

Meß- und Steuerinterface

- 8 ADC-Kanäle 0-2,55V in 0,01V Stufe
- 1 DAC-Kanäle 0-255V in 0,01V Stufe
- Genauigkeit - 1,5 LSB
- 8 frei programmierbare TTL-I/O Kanäle
- Mit Gehäuse, Anschlüsse auf Schraubklemmen
- interne Referenzspannung
- Expansionsportanschluß
- einfache Programmierung in Basic möglich
- Multitasking tauglich
- DEMO-Software auf 3,5" Diskette

239.-

Amiga Eprommer

- Für A500/1000
- Expansionsportanschluß
- Für EPROM's 2701, 2702, 2704, 2708, 2716
- Alle A-Typen und CMOS-Typen
- Funktionen:
 - LEERTEST
 - VERGLEICHEN
 - AUSLESEN
 - BRENNEN
- vier Programmieralgorithmen
- 50mS/Byte - Superschnell 64K 1,5 min
- Programm zum Generieren und Brennen von Kickstarts direkt von Diskette oder aus ROM
- Mit Software + Gehäuse

225.-



Aktuell


```

284 2k      sprintf(string,"%d",flag[i]);
285 nJ      ConPutStr(WMsg,string);
286 c7F    }
287 32      else                               /* wrong i
          nput */
          i--;
288 egI    }
289 fAC    }
290 OS      else if (i>7) {                     /* 3rd & 4th
          drive */
291 sQF      if((j>='0'&&j<'4'&&j!=t[0]&&j!=t[1]&&j!=t[
          2]&&j!=t[3])
292 zsJ      || j=='n') { /* j != unconnected drive
          s */
293 KyI      flag[i]=j-'0';
294 c8      if(flag[i]==62) {
295 GOL      locate(zeile,60);
296 vI      ConPutStr(WMsg," Not wanted ");
297 nII    }
298 Zp      else {
299 K4L      locate(zeile,60);
300 B4      sprintf(string,"df%d: ( or 'n')",fl
          ag[i]);
          ConPutStr(WMsg,string);
301 3z      }
302 sNI    }
303 tOF    }
304 jY      else i--;
305 vQC    }
306 Ni      else if (i==3 || i==4) {           /* start- en
          dtrack */
307 kIF      if (j>='0' && j<='9') {
308 koI      flag[i]=10*flag[i]+j-'0';
309 Fa      if (flag[i]>NUMCYLS-1)
310 bFL      flag[i]=j-'0';
311 c5I      sprintf(string,"%d ",flag[i]);
312 EA      ConPutStr(WMsg,string);
313 3YF    }
314 46      i--;
315 5aC    }
316 r7      else {
317 bRF      if (j=='y')
318 VNI      ConPutStr(WMsg,"YES");
319 yIF      else
320 XZI      ConPutStr(WMsg,"NO ");
321 MHF      flag[i]=j;                               /* character
          */
322 ChC    }
323 zx      i++;
324 Ej9    }
325 Fk6    }
326 K5      if (i>k+5)                               /* bottom o
          f choicetable */
          i=0;
327 Pa9      if (i<0)                               /* top of c
          hoicetable */
          i=k+5;
329 ho9    }
330 Kp3    }
331 Lq0    }
332 RT      int anzdest[4];
333 jv      Copy()
334 Kn      {
335 kV3      SHORT cylinder;
336 23      int firsttime;
337 vk      int i,j,k,l,verify,drive,label8,label9;
338 MR      int anz[4];
339 vI      int head;
340 az      int changeflag = NO;
341 67      long warte;
342 2P      char meldung[17];
343 rI      CursorOff();
344 td      if (!avail) {
345 3C6      for (avail=0 ; avail<80 ; avail++) {
346 k69      diskpuffer[avail]=AllocMem(BLOCKSIZE,MEMF_CHIP);
347 yn      if (diskpuffer[avail]==0)
348 XgC      break;
349 d86    }
350 Bn      for (fastavail=0 ; fastavail<170-avail ; fastavail++
          ) {
351 fS9      fastdiskpuffer[fastavail]=AllocMem(BLOCKSIZE,MEMF_
          FAST);
352 5V      if (fastdiskpuffer[fastavail]==0)
353 cIC      break;
354 iD6    }

```

Listing 1. (Fortsetzung)

**BESUCHEN SIE MARKT & TECHNIK
AUF FOLGENDEN MESSEN:**
Frankfurter Buchmesse, 5.-10. Okt., Halle 6.2, Stand A 449A, 451/B 440
ORGATECHNIK Köln, 20.-25. Okt., Halle 3.1/Gang MN/Stand 12/11
SYSTEC München, 25.-28. Okt., Halle 16/Stand B 18
ELECTRONICA München, 8.-12. Nov., Halle 18/Stand 18A10
Markt & Technik Verlag AG, Hans-Pinsel-Str. 2, 8013 Haar bei München

Superbase Professional

Jetzt gibt es Superbase Professional! Die Profi-Version der bekannten, relationalen Datenbank Superbase mit neuen, mächtigen Features:

- Leistungsfähige Textverarbeitung mit Serienbrieffunktion
- Intelligenter Formulareditor für mehrseitige relationale Formulare mit bis zu 240 Spalten
- Mächtige, Basic-ähnliche Datenbanksprache »DML« mit Unterstützung von sämtlichen Superbase-Professional-Funktionen, Pull-down-Menüs, Eingabe-Masken, Fenstern, Scroll-Balken usw.

Superbase Professional ist das ideale Entwicklungswerkzeug - auch für komplexe Aufgaben!

**Superbase Professional
für Amiga
Bestell-Nr. 51672
DM 599,-***

(sFr 539,-*/öS 5990,-*)

**Superbase Professional
für Atari
Bestell-Nr. 51673
DM 599,-***

(sFr 539,-*/öS 5990,-*)

**Upgrades:
Upgrade Superbase
auf Superbase Professional
für Atari
Bestell-Nr. 51673U
DM 300,-***

(sFr 280,-*/öS 3000,-*)

**Upgrade Superbase
auf Superbase Professional
für Amiga
Bestell-Nr. 51672U
DM 300,-***

(sFr 280,-*/öS 3000,-*)

(Gegen Einsendung der Originaldiskette und gegen Vorkasse mit Verrechnungsscheck oder der abgedruckten Zahlkarte.)

* Unverbindliche Preisempfehlung

**Fragen Sie bei Ihrem Händler
nach weiteren Unterlagen.**

Markt & Technik-Support:
Bei User-Registrierung rechtzeitige Update-/Upgrade-Information und Support-Unterstützung:
Telefon 089/46 13-646 oder -205.
Senden Sie uns bitte Ihre Registrierungskarte.

SuperbaseTM PROFESSIONAL



Markt&Technik-Produkte erhalten Sie in den Fachabteilungen der Warenhäuser, im Versandhandel, in Computer-Fachgeschäften oder bei Ihrem Buchhändler.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Fragen Sie Ihren Fachhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 41 56 56. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 587 1393-0; Rudolf Lechner & Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 67 75 26; Ueberreuter Media Verlagsges.m.bH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 48 15 43-0.


```

355 jE3 }
356 MO for (k=0 ; k<4 ; k++)
357 1K6     anzdest[k]=0; /* reset amount of des
          tinations */
358 Xo3 for (cylinder=flag[3] ; cylinder<=flag[4] ; cylinder=1)
{
359 uh6     if (cylinder==flag[3]) { /* first time */
360 GT9         if (flag[6]==flag[7]) { /* only one drive */
361 YrC             if ((j=(int)(2*(flag[4]-flag[3])/(fastavail+ava
                    il+1.5)))==1)
362 FRF                 k='';
363 gTC             else
364 OVF                 k='s';
365 WJC                 locate(6,27);
366 kF                 sprintf(string,"This copy will need %d swap%c."
                    ,j,k);
367 73                 ConPutStr(WMsg,string);
368 OS                 locate(7,18);ConPutStr(WMsg,"Please insert the
                    \x9b0;33;40m"
369 NQX                     "SOURCE\x9bm disk in drive
                    \x9b0;33;40m");
370 9nC                 sprintf(string,"df%d:\x9bm",flag[6]);ConPutStr(
                    WMsg,string);
371 rE                 locate(8,32);ConPutStr(WMsg,"and press any key"
                    );
372 3b                 if ((j=Taste())==27) /* ESC */
373 47F                     return(FAILED);
374 GJC                 do { /* Is any disk inserte
                    d ? */
375 7eZ                     /* Abort if ESC is pres
                    sed */
376 OOF                     if ((j=ConMayGetChar(consoleReadPort,RMsg,&r
                    eadstring))!=27)
377 8BI                         return(FAILED);
378 8yF                         diskreq[flag[6]]->iotd_Req.io_Command = TD_
                    CHANGESTATE;
379 UV                             DoIO(diskreq[flag[6]]);
380 spC                             } while(diskreq[flag[6]]->iotd_Req.io_Actual);

381 mG                 locate(6,27) ; ConPutStr(WMsg,"
                    ");
382 ng                 locate(7,18);
383 yU                 ConPutStr(WMsg,"
                    ");
384 HO                 locate(8,32);ConPutStr(WMsg,"
                    ");
385 Di9             }
386 zF             else {
387 PhC                 locate (6,19);
388 ZS                 ConPutStr(WMsg,"Please insert the \x9b0;33;40mS
                    OURCE\x9bm disk "
389 IKR                     "in drive \x9b0;33;40m");
390 OsC                 sprintf(string,"df%d:",flag[6]);ConPutStr(WMsg,
                    string);
391 Hr                 for (k=1 ; k<anzdisk ; k++)
392 3OF                     if (flag[6+k]!=62 && flag[6+k]!='n') {
393 Y8I                         locate(6+k,23);sprintf(string,"\x9b0;33;4
                    0m%d",k);
394 YU                             ConPutStr(WMsg,string);
395 p6                             ConPutStr(WMsg,"\x9b0;33;40m. DESTINATION
                    \x9bm disk in "
396 kGX                                 "drive \x9b0;33;40m");
397 YTI                                 sprintf(string,"df%d:\x9bm",flag[6+k]);
398 cY                                 ConPutStr(WMsg,string);
399 RWF                             }
400 2gC                             locate (6+k,32) ; ConPutStr(WMsg,"and press any
                    key");
401 W4                             if ((j=Taste())==27) /* ESC */
402 XaF                                 return(FAILED);
403 SbC                                 do { /* Are disks in all drives ? *
                    /
404 qSF                                     if ((j=ConMayGetChar(consoleReadPort,RMsg,&r
                    eadstring))!=27)
405 adI                                         return(FAILED);
406 aQF                                         diskreq[flag[6]]->iotd_Req.io_Command = TD_
                    CHANGESTATE;
407 wx                                             DoIO(diskreq[flag[6]]);
408 N5C                                             }while (diskreq[flag[6]]->iotd_Req.io_Actual);

409 ux                 for (k=1 ; k<anzdisk ; k++) {
410 LgF                     if (flag[6+k]!=62 && flag[6+k]!='n') {
411 zPI                         do {
412 kLL                             if ((j=ConMayGetChar(consoleReadPort,R

```

```

          Msg,&readstring))
          ==27)
          return(FAILED);
          diskreq[flag[6+k]]->iotd_Req.io_Comma
          nd=TD.CHANGESTATE;
          DoIO(diskreq[flag[6+k]]);
          } while(diskreq[flag[6+k]]->iotd_Req.io_
          Actual);
      }
      locate (6,19);
      ConPutStr(WMsg,"
      ");
      for (k=1 ; k<anzdisk ; k++)
          if (flag[6+k] != 'n') {
              locate(6+k,23);
              ConPutStr(WMsg,"
              ");
          }
          locate (6+k,32) ; ConPutStr(WMsg,"
          ");
      }
      } else /* belongs to 'if (cylinder==
      flag[3])' */
      if (flag[6]==flag[7])
          if(WaitForDisk(flag[6],"SOURCE")==FAILED)
              return(FAILED);
          MotorOn(flag[6]);
          for (i=0,l=cylinder ; i<avail-1 ; l++) {
              if (l>flag[4]) /* end of co
              py ? */
                  break;
              /* Abort if ESC is pressed */
              if ((j=ConMayGetChar(consoleReadPort,RMsg,&readstr
              ing))!=27) {
                  MotorOff(flag[6]);
                  return(FAILED);
              }
              if (flag[5]==2) { /* both surfaces */
                  if (i==0 && fastavail>1) {
                      if((l=ReadFastTwoHeads(1))==FAILED)
                          return(FAILED);
                      i++;
                  }
                  if (l>flag[4] || i==avail-1) /* end of copy
                  ? */
                      break;
                  locate(6,30);sprintf(string,"Reading cylinder
                  %d ",l);
                  ConPutStr(WMsg,string);
                  for (head=0 ; head<2 ; i++,head++) {
                      ReadCylSec(flag[6],l,head,1);
                      if (diskreq[flag[6]]->iotd_Req.io_Error)
                          if (diskreq[flag[6]]->iotd_Req.io_Error=
                          =29) {
                              if(DiskRemoved()==FAILED)
                                  return(FAILED);
                              goto reading1;
                          }
                      else
                          if((ReadError(1,l,head,
                          diskreq[flag[6]]->iotd_Req.io_Error)
                          )==FAILED)
                              return(FAILED);
                  }
              }
          } else { /* only one surface
          */
              if (i==0 && fastavail) { /* Is there any fast
              mem ? */
                  if((l=ReadFastOneHead(1,flag[5]))==FAILED)
                      return(FAILED);
                  i++;
              }
              reading2:
              if (l>flag[4] || i==avail-1) /* e
              nd of copy ? */
                  break;
              locate(6,30);sprintf(string,"Reading cylinder

```

Listing 1. (Fortsetzung)

PROGRAMM-SERVICE

Direkt bestellen statt abtippen!

Die aktuelle Diskette zum Heft:

Amiga Sonderheft 1: Bauen Sie Ihr eigenes Tonstudio

Digisoft Plus: Die Software zu unserem Selbstbau-Digitizer, die den Amiga zum digitalen Sound-Studio macht. Das Amiga-Tonstudio arbeitet mit vier getrennten Tonspuren und besitzt neben der Digitalisierung von Klängen viele weitere Features. So können einzelne Spuren oder Teile daraus beliebig gemischt, geschnitten oder verbunden werden. Die grafische Ausgabe der digitalisierten Sounds macht die Bearbeitung zum Kinderspiel.

Suremosch: Ein Strategiespiel der neuen Dimension: Begeben Sie sich in eine ferne, fremde Welt und befreien Sie die Bevölkerung von bösen Wesen, die den Zeitfluß verlangsamen.

Dcopy, ist ein schnelles Kopierprogramm der Extraklasse. Neben hoher Geschwindigkeit bietet es die Möglichkeit, bis zu drei Kopien in einem Arbeitsgang anzufertigen.

Biorhythmus: Neben einer sehr schönen grafischen Darstellung der Kurven von Geist, Körper und Seele erlaubt das komfortable Programm auch die Ausgabe einer Jahresstatistik. Sie erfahren alles über Ihre »guten« und »kritischen« Tage.

3 1/2"-Diskette für Amiga
Bestell-Nr. 45801

DM 29,90* (sFr 24,90*/öS 299,-*)
* Unverbindliche Preisempfehlung

Amiga-Hardware-Service:

Die Wiederbelebung für die C64-Peripherie

Viele Amiga-Besitzer haben noch einen C64 mit Peripheriegeräten zu Hause stehen. Mit ein bißchen Hard- und Software können Sie diese zu neuem Leben erwecken und Ihre Daten so weiterbenutzen. Dabei ist die Bedienung wirklich einfach.

Der fertig aufgebaute IEC-Handler erlaubt es, alle C64-Geräte wie die Floppy 1541 oder 1571, Commodore-MPS-Drucker und natürlich auch den C64 (zur Datenübertragung) am Amiga zu betreiben.

Das Gesamtpaket besteht aus der fertig aufgebauten Platine mit Verbindungskabel, der Treibersoftware auf 3 1/2-Zoll-Diskette sowie einer entsprechenden Dokumentation.

Bestell-Nr. 39101

DM 79,-* (sFr 71,-*/öS 790,-*)
* Unverbindliche Preisempfehlung

Der »IEC-Handler« ist ab Ende September 1988 lieferbar. Bestellungen bitte nur gegen Vorkasse bei:

Markt&Technik Verlag AG

– Buchverlag –

IEC-Handler

Hans-Pinsel-Straße 2, 8013 Haar bei München



Weitere Angebote
auf der Rückseite!

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0

Bestellungen im Ausland bitte an: SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 41 56 56. ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 587 13 93-0; Rudolf Lechner&Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526, Ueberreuter Media Verlagsges. mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (0222) 481543-0

Sie suchen hilfreiche Utilities und professionelle Anwendungen für Ihren Computer? Sie wünschen sich gute Software zu vernünftigen Preisen? Hier finden Sie beides! Unser stetig wachsendes Sortiment enthält interessante Listing-Software für alle gängigen Computertypen. Jeden Monat erweitert sich unser aktuelles Angebot um eine weitere interessante Programmsammlung für jeweils einen Computertyp. Bei Fragen zu Bestellung und Versand der Programmservice-Disketten wählen Sie bitte:
Telefon (089) 46 13-232.

Bestellungen bitte nur gegen Vorauskasse an: Markt & Technik Verlag AG, Unternehmensbereich Buchverlag, Hans-Pinsel-Straße 2, D-8013 Haar, Telefon (089) 46 13-0.
Schweiz: Markt & Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 41 56 56.
Österreich: Markt & Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (02 22) 5 87 13 93-0.
Microcomput-ique, E. Schiller, Fasangasse 24, A-1030 Wien, Telefon (02 22) 78 56 61;
Bücherzentrum Meidling, Schönbrenner Straße 261, A-1120 Wien, Telefon (02 22) 83 31 96.
Ueberreuter Media, Verlagsges. mbH (Großhandel), Laudongasse 29, A-1082 Wien, Telefon (02 22) 48 15 43-0.
Bestellungen aus anderen Ländern bitte nur schriftlich an: Markt & Technik Verlag AG, Abt. Buchvertrieb, Hans-Pinsel-Straße 2, D-8013 Haar, und gegen Bezahlung der Rechnung im voraus.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die beigeheftete Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungsscheck mit Ihrer Bestellung. Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

Amiga 8/88: C64-Peripherie am Amiga

IEC-Handler: Mit dem IEC-Handler lassen sich C64-Peripheriegeräte wie die VC 1541 oder ein MPS-Drucker an den Amiga anschließen. Zum Betrieb ist die hier zu findende Software nötig.

Virustest: Wir rüsten zum Kampf. Damit Sie mehr Schutz vor Computerviren haben, hilft der Virustester beim Erkennen dieser lästigen Biester.

AkoTerm: Steuerungssoftware zum Betrieb des Amiga mit einem Akustikkoppler. Erschließen Sie sich die Welt der Datenfernübertragung.

Eliza: Kommunizieren Sie mit Ihrem Amiga. Mit diesem Programm führt der Amiga mit Ihnen ein Gespräch über die verschiedensten Themen. Der Psychoanalytiker daheim...

Resi: Macht Programme resetfest. So stehen Ihnen auch nach einem Warmstart noch die damit behandelten Programme zur Verfügung.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48808 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga 5/88: Vom Spiel zum nützlichen Utility

Diesmal finden Sie auf unserer Programmservice-Diskette wieder ein breites Spektrum an Listings. Von Spielen über Werkzeuge bis zu Anwendungen ist alles vorhanden.

Kniffel: Ein grafisch gut aufgemachtes Spiel für bis zu vier Teilnehmer. Kniffel wird sicher nicht langweilig. Ein Muß für alle Glücksspieler.

Manager: Verschafft Ihnen die Übersicht über Ihre Ausgaben in klarer Form und hilft somit Geld sparen. Komfortable Bedienung per Maus ist selbstverständlich.

CrossRef: Hilft Ihnen beim Analysieren von Programmen. Viele wichtige Daten von Basic-Programmen wie Labels und Variablen erhalten Sie schwarz auf weiß ausgedruckt. Ein unentbehrliches Hilfsmittel für Basic-Programmierer.

3-D-Tic-Tac-Toe: Ein gutes Auge und einen scharfen Verstand brauchen Sie für diese dreidimensionale Spielvariante.

Recover: Rettet versehentlich gelöschte Dateien von Ihrer Diskette. Auch teilweise zerstörte Dateien werden soweit als möglich restauriert.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48805 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga 4/88 3-D-Landschaften aus dem Computer

Fraktalberge: Ein Muß für alle Fans von zufallserzeugten Grafiken. Fantastisch einfach in der Bedienung und sehr schnell.

Transfer: Überträgt Bilder vom C64 auf den Amiga. Mit guter Software und leicht nachzubauender Hardware.

DiskSpy: Direktes Ändern von Daten auf der Diskette ist mit diesem Werkzeug kein Problem mehr. Es stehen viele Befehle zur Verfügung.

ColorChange: Ein Basic-Unterprogramm, mit dem Sie einfach und schnell Ihre Wunschfarben auf beliebigen Bildschirmen einstellen können.

Troof: Ein spannendes Spiel in Basic mit starker Grafik und vielen verschiedenen Levels.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48804 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga 3/88 Bildschirmfüllende Boot-Bilder mit allen Extras

BootGirl: Fantastische Bilder sofort nach dem Reset. Bis zu 32 Farben mit Color-Cycling. Die Bilder können auch bildschirmfüllend ohne Rand sein. Ein absolutes Muß für jeden Amiga-Besitzer.

CassCover: Selbstgedruckte Kassettenhüllen geben Ihnen den richtigen Überblick. Einfache Bedienung macht das Eingeben und Ausdrucken zur wahren Freude.

Command: Das Programm ermöglicht die Steuerung des Aztec-C-Compilers mit der Maus. Keine langen Eingaben per Tastatur, sondern ein einziger Mausklick startet nun die Übersetzung.

VideoText: Ein unentbehrliches Werkzeug für alle Video-Fans, die ihren eigenen Vorspann mit dem Amiga generieren wollen. Laufbänder, verschiedene Schriften und IFF-Bilder sind nur einige Stichpunkte, die das Programm so interessant machen.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48803 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga 12/87 Super-Kopierprogramm mit viel Komfort

DCopy: Unser Programm des Monats, ein Kopierprogramm, das alles bietet, was man sich nur wünschen kann. Einige Fähigkeiten: Bis zu vier Laufwerke werden verwendet, Mehrfachkopien, abschaltbares Verify und vieles mehr.

SpeedHc: Eine sehr schnelle Hardcopyroutine für Schwarzweißausdrucke mit höchster Qualität. Leicht an andere Drucker anzupassen.

Sternenhimmel: Ein unentbehrliches Werkzeug für alle Himmelsbeobachter. Das Programm zeigt alle Sterne und Planeten von jedem beliebigen Punkt der nördlichen Hemisphäre.

Checkie42: Der Checksummer für alle Programmiersprachen von Assembler über Basic bis zu C. Ab dieser Ausgabe finden Sie bei jedem Listing die Prüfziffern.

Joy: Ein sehr kurzes und schnelles C-Programm zur Abfrage des Joysticks. Es ist leicht in eigene Programme einzubinden.

Amiga-Shell: Ein C-Programm, das Komfort ins CLI bringt. Editieren der Befehlszeile, Funktionstastenbelegung und Aliasnamen sind nur einige Fähigkeiten dieses fantastischen Programms.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48705 **DM 29,90*** sFr 24,90*/öS 299,-*

Amiga 10/87 Super-Malprogramme für alle Amiga-Computer

Rainbow-Drawer: Dieses Programm des Monats bietet leistungsfähige Befehle und Funktionen, wie sie von professionellen Programmen bekannt sind: bis zu 32 Farben, alle Auflösungen, viele Befehle zum Zeichnen sowie FILL mit Mustern, BOW und anderem.

Turtle: Mit dieser Befehlsweiterung verfügen Sie über die Grafikbefehle, die bei Logo bekannt und beliebt sind.

Fractals: Dreidimensionale, realistische Gebirge mit Schattierung erzeugt dieses Programm.

Clouds: Genauso wirklichkeitsnah wie die Gebirge, aber noch erstaunlicher, sind die Wolken, die Sie mit Clouds generieren.

Apfelmännchen: Hiermit erzeugen Sie schöne Grafiken aus der beliebten Mandelbrot-Ebene.

Kudiplo: Ein gutes, unverzichtbares Werkzeug für die Kurvendiskussion stellt »Kudiplo« dar.

Senso: Testen Sie mit dieser Computer-Adaption des bekannten Spiels Ihr Gedächtnis!

Division: Bis zu 32000 Nachkommastellen können durch dieses Programm berechnet werden.

Alert: Alarme, zum Beispiel die bekannten Guru-Meditations, können Sie nun selbst programmieren. Das Programm ist in erster Linie für C-Programmierer aufschlußreich.

Border: Lassen Sie den Fensterrahmen des CLI-Fensters einfach verschwinden!

SCD: Mit diesem Utility können Sie den Pfadnamen in der Titelleiste des Fensters anzeigen.

3 1/2"-Diskette für Amiga

Bestell-Nr. 48703 **DM 29,90*** sFr 24,90*/öS 299,-*

* Unverbindliche Preisempfehlung

Übrigens: Mit den Gutscheinen aus dem DM 149,- können Sie sechs Software-Disketten Ihrer Wahl aus dem Programm-Service-Angebot der Zeitschriften

PC Magazin	Happy-Computer-Sonderheft	Computer persönlich
PC Magazin Plus	Amiga-Magazin	64'er-Magazin
Happy-Computer	Amiga-Sonderheft	64'er-Sonderheft

bestellen - egal, ob diese DM 29,90 oder DM 34,90 kosten. Das Scheckheft können Sie per Verrechnungsscheck oder mit der eingelebten Zahlkarte direkt beim Verlag bestellen.

Kennwort: Software-Scheckheft, Bestell-Nr. 39100.


```

% d ",1);
478 uq      ConPutStr(WMsg,string);
479 n1      ReadCylSec(flag[6],1,flag[5],1);
480 rE      if (diskreq[flag[6]]->iotd_Req.io_Error) /* e
error detected */
481 vTF      if (diskreq[flag[6]]->iotd_Req.io_Error==29
) {
482 P6I      if(DiskRemoved()==FAILED)
483 qTL      return(FAILED);
484 jmI      goto reading2;
485 pKF      }
486 fS      else
487 E8I      if((ReadError(i,l,head,
diskreq[flag[6]]->iotd_Req.io_Error))==
488 EkJ      FAILED)
return(FAILED);
489 wzL      return(FAILED);
490 11C      i++; /* increment memory-buffer */
491 vQ9      }
492 wR6      }
493 ha      MotorOff(flag[6]);locate(6,30);ConPutStr(WMsg,"
");
494 aE      for (k=0 ; k<4 ; k++)
495 1D9      anz[k]=anzdest[k]; /* remember the amount of
destinations */
496 wQ6      drive = flag[7]; /* actual drive-number */
497 fX      label8 = label9 = 0; /* reset multiple-copy la
bels */
498 9L      firsttime = YES; /* flag */
499 X2O change:
500 dS6      if (anzdest[drive] || flag[6]==drive){
501 4u9      if (flag[6]==drive || anz[drive]<anzdest[drive] |
| changeflag==YES){
502 GmC      changeflag = NO;
503 BU      if (firsttime==YES)
504 xuE      sprintf(meldung,"%d. DESTINATION",anz[drive]+
1);
505 y1C      else
506 rtE      sprintf(meldung,"%d. DESTINATION",anzdest[dri
ve]-anz[drive]+1);
507 aMC      if(WaitForDisk(drive,meldung)==FAILED)
508 F1F      return(FAILED);
509 k7C      MotorOn(drive);
510 E39      }
511 Fk6      }
512 5s      else
513 oB9      MotorOn(drive);
514 V7V      /* writing data */
515 cc6      for (i=0,l=cylinder ; i<avall-1 ; l++) {
516 zf9      if (l>flag[4]) /* end of copy ? */
517 GPC      break;
518 KLv      /* ESC pressed ? */
519 6B9      if ((j=ConMayGetChar(consoleReadPort,RMsg,&readstr
ing))!=27) {
520 41C      MotorOff(drive);
521 SV      return(FAILED);
522 Qv9      }
523 OC      if (flag[5]==2) { /* both surfaces */
524 3rC      if (i==0 && fastavall>1) { /* writing fast
mem-buffer */
525 6PF      if((l=WriteFastTwoHeads(l,drive))==FAILED)
526 XaI      return(FAILED);
527 2DF      else if (l==NO)
528 TaI      goto giveup;
529 JHF      i++;
530 Y3C      }
531 jX      if (l>flag[4] || i==avall-1)
532 VeF      break;
533 m9C      verify=0; /* reset amount of verif
y-errors */
534 7j      locate(6,30);
535 d2      sprintf(string,"Writing cylinder %d ",1);
536 qm      ConPutStr(WMsg,string);
537 jV      for (head=0 ; head<2 ; i++,head++) {
538 DsO writing2:
539 a4F      if(WriteCylSec(drive,l,head,1)==FAILED)
540 1oI      return(FAILED);
541 InF      if (flag[0]=='y') { /* verify ON */
542 jOI      ReadCylSec(drive,l,head,1);
543 wY      if (diskreq[drive]->iotd_Req.io_Error)
544 uAL      if (diskreq[drive]->iotd_Req.io_Error
==29) {
locate(6,22);
ConPutStr(WMsg,"Destination is \x9

```

```

0;33;40m"
547 U4d "removed\x9bm. Try it
again !");
548 WTO MotorOff(drive);
549 VN if(WaitForDisk(drive,"")==FAILED)
550 vyR return(FAILED);
551 SDO locate(6,25);
552 86 ConPutStr(WMsg,"
");
553 Sp MotorOn(drive);
554 SB goto writing2;
555 xSL }
556 6s else { /* verify-error detect
ed */
557 MfO if (++verify < flag[1]) {
558 4UR if((VerifyError(1,head,drive))==
FAILED)
559 47U return(FAILED);
560 reR else
561 ZIU goto writing2;
562 4ZO }
563 q6 else {
564 m6R anzdest[drive]--;
565 4B goto giveup;
566 8dO }
567 9eL }
568 AfF }
569 BgC }
570 Ch9 }
571 Yi else { /* only one surface
*/
572 OUC if (i==0 && fastavail) { /* copying from fastm
em to chipmem */
573 4XF if((!WriteFastOneHead(1,drive))==FAILED)
574 JMI return(FAILED);
575 ozF else if (l==NO)
576 FMI goto giveup;
577 53F i++;
578 KpC }
579 OhO writing4:
580 WKC if (l>flag[4] || i==avail-1)
581 IRF break;
582 y5C if(WriteCylSec(drive,l,flag[5],i)==FAILED)
583 SVF return(FAILED);
584 3qC if (flag[0]=='y') { /* verify */
585 PeF ReadCylSec(drive,l,flag[5],i);
586 dF if (diskreq[drive]->iotd_Req.io_Error)
587 brI if (diskreq[drive]->iotd_Req.io_Error==2
9) {
588 xfL locate(6,22);
589 y6 ConPutStr(WMsg,"Destination is \xb0;3
3;40m"
590 Bla "removed\x9bm. Try it a
gain !");
591 DAL MotorOff(drive);
592 C4 if(WaitForDisk(drive,"")==FAILED)
593 8t locate(6,25);
594 om ConPutStr(WMsg,"
");
595 8V MotorOn(drive);
596 G1 goto writing4;
597 d8I }
598 Pf else {
599 2LL if (++verify < flag[1]) {
600 8rO if((VerifyError(1,flag[5],drive))==
FAILED)
601 knR return(FAILED);
602 XKO else
603 N8R goto writing4;
604 kFL }
605 Wm else {
606 SmO anzdest[drive]--;
607 kr goto giveup;
608 oJL }
609 pKI }
610 qLC }
611 db i++;
612 sN9 }
613 tO6 }
614 uxO giveup:
615 XA6 MotorOff(drive);locate(6,30);ConPutStr(WMsg,"
");
616 CL if (flag[2]=='y') {

```

Listing 1. (Fortsetzung)


```

617 e49      if (cylinder==flag[3]) {
618 nVC          locate (6,27) ; ConPutStr(WMsg,"Another DESTINA
                TION (y/n) ?");
619 JV          j=Taste();
620 jc          locate (6,27) ; ConPutStr(WMsg,"
                ");
621 z0          if (j=='y' || j=='Y') {
622 IdF              anz[drive]++;anzdest[drive];
623 DN              changeflag = YES;
624 V4              goto change;
625 5aC          }
626 6b9          }
627 vW          else if (anz[drive]-- > 0) {
628 7mC              firsttime = NO;
629 a9              goto change;
630 Af9          }
631 Bg6          }
632 Ne          if (flag[8]!='n' && flag[8]!=62 && label8==0) {
633 Dn9              drive=flag[8];
634 SZ              label8=1;
635 gF              goto change;
636 G16          }
637 er          if (flag[9]!='n' && flag[9]!=62 && label9==0) {
638 Lw9              drive=flag[9];
639 Zh              label9=1;
640 LK              goto change;
641 Lq6          }
642 XI3      if (verify==flag[1])
643 QT6          return(FAILED);
644 Ot3      }
645 kJ          return(OK);
646 lq0      } /* end of Copy() */
647 9B0          /* exit gracefully */
648 Kp0 End()
649 Ps {
650 yU3      int i;
651 lg      if (flag[6]!=flag[7]) { /* more than one drive */
652 AE6          ConPutStr(WMsg,"f");
653 we          locate(4,24);
654 uY          ConPutStr(WMsg,"Please remove ALL destinations,");
655 8u          locate(5,27);
656 jz          ConPutStr(WMsg,"so DOS won't get confused.");
657 Gw          locate(7,33);
658 m5          ConPutStr(WMsg,"Press any key.");
659 v6          i=Taste();
660 e93      }
661 zX      for (i=0 ; i<4 ; i++)
662 oh6          if (i!=t[0]-'0' && i!=t[1]-'0' && i!=t[2]-'0' && i!=t
[3]-'0') {
663 S09              CloseDevice(diskreq[i]);
664 sc              if((mp = FindTask("File System"))!=NULL)
665 yfA                  dos_packet(&mp->pr_MsgPort,ACTION_INHIBIT,FALSE,
OL,OL,OL,OL,OL,OL);
666 FX9              DeleteExtIO(diskreq[i],(long)sizeof(struct IOExtTD
));
667 hS              DeletePort(diskport[i]);
668 mH6          }
669 a63      for (i=0;i<avail;i++)
670 p26          FreeMem(diskpuffer[i],BLOCKSIZE);
671 sP3      for (i=0;i<fastavail;i++)
672 Fu6          FreeMem(fastdiskpuffer[i],BLOCKSIZE);
673 yx3      i=CheckIO(RMsg);
674 3a      if (i==FALSE)
675 Me6          AbortIO(RMsg);
676 WA3      WaitPort(consoleReadPort);
677 43      GetMsg(consoleReadPort);
678 v2      CloseDevice(WMsg);
679 qS      CloseWindow(w);
680 t2      DeleteStdIO(RMsg);
681 SK      DeletePort(consoleReadPort);
682 FT      DeleteStdIO(WMsg);
683 KB      DeletePort(consoleWritePort);
684 Js      CloseLibrary(IntuitionBase);
685 En      exit (0L);
686 z70      } /* end of End() */
687 JY      Changenum(i)
688 a6      int i;
689 3W {
690 o23      diskreq[i]->iotd_Req.io_Command = TD_CHANGENUM;
691 kE          DoIO(diskreq[i]);
692 Af0      }
693 kp      MotorOn(i)
694 gC      int i;
695 9c {

```

```

696 7s3      diskreq[i]->iotd_Req.io_Length = 1;
697 I7      diskreq[i]->iotd_Req.io_Command = TD_MOTOR;
698 rL          DoIO(diskreq[i]);
699 Hm0      }
700 id      MotorOff(i)
701 nJ      int i;
702 GJ {
703 9t3      diskreq[i]->iotd_Req.io_Length = 0;
704 PE          diskreq[i]->iotd_Req.io_Command = TD_MOTOR;
705 yS          DoIO(diskreq[i]);
706 Ot0      }
707 w6      ReadCylSec(i,cyl,hd,puff)
708 uQ      int i;
709 k5      SHORT cyl,hd;
710 ZG      int puff;
711 Ps {
712 8B3      LONG offset;
713 sy          diskreq[i]->iotd_Req.io_Length = BLOCKSIZE;
714 UQ          diskreq[i]->iotd_Req.io_Data = (APTR)diskpuffer[puff];
715 kE          diskreq[i]->iotd_Req.io_Command = CMD_READ;
716 6S          offset = TD_SECTOR * (NUMSECS * hd + NUMSECS * NUMHEADS
* cyl);
717 OR          diskreq[i]->iotd_Req.io_Offset = offset;
718 Bf          DoIO(diskreq[i]);
719 tx          return(YES);
720 c70      }
721 Xv      WriteCylSec(i,cyl,hd,puff)
722 8e      int i;
723 yJ      SHORT cyl,hd;
724 nU      int puff;
725 d6 {
726 MP3      LONG offset;
727 LJ0      label:
728 7D3      diskreq[i]->iotd_Req.io_Length = BLOCKSIZE;
729 jf          diskreq[i]->iotd_Req.io_Data = (APTR)diskpuffer[puff];
730 5S          diskreq[i]->iotd_Req.io_Command = TD_FORMAT;
731 Lh          offset = TD_SECTOR * (NUMSECS * hd + NUMSECS * NUMHEADS
* cyl);
732 dg          diskreq[i]->iotd_Req.io_Offset = offset;
733 Qu          DoIO(diskreq[i]);
734 zL          if (diskreq[i]->iotd_Req.io_Error) {
735 Sm6              if (diskreq[i]->iotd_Req.io_Error == 28) {
736 RJ9                  locate(6,18);
737 hA                  ConPutStr(WMsg,"Destination is \x9b0;33;40mwrite-p
rotected\x9bm. "
                          "Try it again !");
738 U90                      MotorOff(i);
739 e09                      if(WaitForDisk(i,"")==FAILED)
740 k7                          return(FAILED);
741 O3C                          locate(6,18);
742 XP9                          ConPutStr(WMsg,"
");
743 mI                          }
744 Rx          MotorOn(i);
745 Zb          goto label;
746 2X6          }
747 j4          if (diskreq[i]->iotd_Req.io_Error == 29) {
748 XF9              locate(6,22);
749 qa              ConPutStr(WMsg,"Destination is \x9b0;33;40mremoved
\x9bm. "
                          "Try it again !");
750 gLO                      MotorOff(i);
751 qC9                      if(WaitForDisk(i,"")==FAILED)
752 wJ                          return(FAILED);
753 CFC                          locate(6,22);
754 dL9                          ConPutStr(WMsg,"
");
755 Ip                          }
756 d9          MotorOn(i);
757 ln          goto label;
758 EJ6          }
759 Fk3      }
760 Yc          return(YES);
761 Hm0      }
762 20      FullSeek(times,unit)
763 zs      int times,unit;
764 GJ {
765 pL3      int i;
766 tw          for (i=0 ; i<times ; i++) { /* Move head to cyl 79
then to cyl 0 */
767 T16              diskreq[unit]->iotd_Req.io_Offset=
768 eTV                  (long)((((NUMCYLS-1)*NUMSECS*
NUMHEADS-1)*512);
769 hu6              diskreq[unit]->iotd_Req.io_Command = TD_SEEK;
770 Je          DoIO(diskreq[unit]);
771 oT          diskreq[unit]->iotd_Req.io_Offset=0L;

```



```

772 kx      diskreq[unit]->iotd_Req.io_Command = TD_SEEK;
773 mh      DoIO(diskreq[unit]);
774 Ua3     }
775 V00     }
776 DS      WaitForDisk(c,text)
777 RL      int c;
778 wc      char *text;
779 Vy      {
780 Ac3      int j;
781 rf      locate(7,(80-38-strlen(text))/2);
782 z4      ConPutStr(WMsg,"Please insert the \x9b0;33;40m");
783 8H      ConPutStr(WMsg,text);
784 3n      sprintf(string,"\x9bm disk in drive \x9b0;33;40mdf%d: \x
9bm",c);
785 rn      ConPutStr(WMsg,string);
786 ma      do {
787 wj6      /* Is disk removed */
788          if((j=ConMayGetChar(consoleReadPort,RMsg,&readstring)
789             )==27) {
790              return(FAILED);
791          }
792          diskreq[c]->iotd_Req.io_Command = TD_CHANGESTATE;
793          DoIO(diskreq[c]);
794          while(!diskreq[c]->iotd_Req.io_Actual);
795          do {
796              /* Is disk inserted ? */
797              j=ConMayGetChar(consoleReadPort,RMsg,&readstring);
798              if (j==27)
799                  return(FAILED);
800              diskreq[c]->iotd_Req.io_Command = TD_CHANGESTATE;
801              DoIO(diskreq[c]);
802              while(diskreq[c]->iotd_Req.io_Actual);
803              locate(7,10);
804              ConPutStr(WMsg,"
805              " ");
806          }
807          return(YES);
808      }
809      Taste()
810      {
811          int j;
812          while((j=ConMayGetChar(consoleReadPort,RMsg,&readstring)
813             )!=-1);
814          while((j=ConGetChar(consoleReadPort,RMsg,&readstring))==
815             '\n');
816          return(j);
817      }
818      Po      locate(lin,col)
819      jt      int lin,col;
820      4X      {
821          sprintf (string, "\x9b%d;%dH",lin,col);
822          ConPutStr(WMsg,string);
823      }
824      6D      ReadFastTwoHeads(1)
825      zJ      int l;
826      Ad      {
827          int j,k,head;
828          for (k=0 ; k<fastavail-1 ; l++) {
829              if (l>flag[4])
830                  return(1);
831              if((j=ConMayGetChar(consoleReadPort,RMsg,&readstring)
832                 )==27) {
833                  MotorOff(flag[6]);
834                  return(FAILED);
835              }
836              locate(6,30);
837              sprintf(string,"Reading cylinder %d ",l);
838              ConPutStr(WMsg,string);
839              for (head=0 ; head<2 ; head++) {
840                  RRO reading3:
841                  ReadCylSec(flag[6],l,head,0);
842                  if (diskreq[flag[6]]->iotd_Req.io_Error)
843                      if (diskreq[flag[6]]->iotd_Req.io_Error==29) {
844                          if(DiskRemoved()==FAILED)
845                              return(FAILED);
846                          goto reading3;
847                      }
848                  else
849                      if((ReadError(0,l,head,
850                         diskreq[flag[6]]->iotd_Req.io_Error))=FAI
851                         LED)
852                          return(FAILED);
853                  erase=fastdiskpuffer[k++] ; puffmerk=diskpuffer[0]
854                  ;
855              }
856          }
857      }

```

```

848 5D      move.w    **#1407,d0
849 LU      move.l    _erase,a0
850 BK      move.l    _puffmerk,a1
851 g30 laber      move.l    (a1)+,(a0)+
852 47D      dbra      d0,laber
853 uzB      movem.l    (sp)+,d0/a0-a1
854 QK0    **#endasm
855 nI6      }
856 oJ3      }
857 A3      return(1);
858 qL0      }
859 dn      ReadFastOneHead(1,head)
860 nF      int l,head;
861 pI      {
862 NE3      int j,k;
863 db      for (k=0 ; k<fastavail ; l++) {
864         ap6      if (l>flag[4])
865             return(1);
866         T56      locate(6,30);
867         iJ      sprintf(string,"Reading cylinder %d ",l);
868         C8      ConPutStr(WMsg,string);
869         560 reading4:
870         H46      if((j=ConMayGetChar(consoleReadPort,RMsg,&readstring)
871            )==27) {
872             6a9      MotorOff(flag[6]);
873             7A      return(FAILED);
874         }
875         xJ      ReadCylSec(flag[6],l,head,0);
876         GK      if (diskreq[flag[6]]->iotd_Req.io_Error)
877             if (diskreq[flag[6]]->iotd_Req.io_Error==29) {
878                 mTC      if(DiskRemoved()==FAILED)
879                     return(FAILED);
880                 goto reading4;
881             }
882             else
883                 if((ReadError(0,l,head,
884                    diskreq[flag[6]]->iotd_Req.io_Error))=FAILED
885                    )
886                     return(FAILED);
887             erase=fastdiskpuffer[k++] ; puffmerk=diskpuffer[0];
888         6m0    **#asm
889         5z8      movem.l    d0/a0-a1,-(sp)
890         jr      move.w    **#1407,d0
891         z8      move.l    _erase,a0
892         py      move.l    _puffmerk,a1
893         lV0 laber0      move.l    (a1)+,(a0)+
894         kYA      dbra      d0,laber0
895         Yd8      movem.l    (sp)+,d0/a0-a1
896         4y0    **#endasm
897         R3      }
898         ng      return(1);
899         Ty0      }
900         2A      ReadError(i,track,head,errnumber)
901         PF      int i,track,head,errnumber;
902         Sv      {
903             7Z3      int j;
904             PT      locate(10,1);
905             95      ConPutStr(WMsg,"\x9b0;33;40mRead-error\x9bm at cylinder
906                 = ");
907             Ra      sprintf(string,"%d, head = %d error = %d",track,head,err
908                 number);
909             nJ      ConPutStr(WMsg,string);
910             LQ      if ((j=ConMayGetChar(consoleReadPort,RMsg,&readstring))=
911                 =27) {
912                 6A6      MotorOff(flag[6]);
913                 hk      return(FAILED);
914             }
915             erase=diskpuffer[i];
916             FN      for (j=0;j<BLOCKSIZE;j++)
917                 *erase++ = '\0';
918             6V3      locate(10,60);
919             qx      sprintf(string,"Read-errors: %d",++error);
920             xt      ConPutStr(WMsg,string);
921             7g      return(OK);
922         }
923         nI0      }
924         2a      DiskRemoved()
925         1E      {
926             24      locate(6,24);
927             zo      ConPutStr(WMsg,"Source is \x9b0;33;40mremoved\x9bm. Try
928                 it again !");
929             vP      MotorOff(flag[6]);
930             iT      if(WaitForDisk(flag[6], "")==FAILED)

```

Listing 1. (Fortsetzung)


```

924 x06      return(FAILED);
925 SC3      locate(6,24);
926 3a      ConPutStr(WMsg, "
");
927 NE      MotorOn(flag[6]);
928 GK      return(YES);
929 zU0 }
930 S3      OpenConsole(writerequest,readrequest>window)
931 RQ      struct IOStdReq *writerequest;
932 xe      struct IOStdReq *readrequest;
933 Ka      struct Window *window;
934 OT      {
935 F53      int error;
936 oO      writerequest->io_Data = (APTR) window;
937 Sq      writerequest->io_Length = sizeof(*window);
938 do      error = OpenDevice("console.device", 0L, writerequest, 0
L);
939 cI      readrequest->io_Device = writerequest->io_Device;
940 ur      readrequest->io_Unit = writerequest->io_Unit;
941 UB      /* clone required parts of the request */
942 Z9      return(error);
943 D10 }
944 Wp      /* Output a single character to a specified conso-
le */
945 e8      ConPutChar(request,character)
946 Lq      struct IOStdReq *request;
947 NX      char character;
948 Eh      {
949 Od3      request->io_Command = CMD_WRITE;
950 CF      request->io_Data = (APTR)&character;
951 Ly      request->io_Length = 1L;
952 nU      DoIO(request);
953 fj      return(YES);
954 Ot0 }
955 Ms      ConPutStr(request,string)
956 VO      struct IOStdReq *request;
957 lw      char *string;
958 Or      {
959 An3      request->io_Command = CMD_WRITE;
960 Zz      request->io_Data = (APTR)string;
961 hD      request->io_Length = -1L; /* tells console to end when
it
962 zWV      * sees a terminating zero on
963 Bw      * the string. */
964 zg3      DoIO(request);
965 rv      return(YES);
966 a50 }
967 HH      QueueRead(request,wheret0)
968 hC      struct IOStdReq *request;
969 Kf      char *wheret0;
970 a3      {
971 S63      request->io_Command = CMD_READ;
972 TI      request->io_Data = (APTR)wheret0;
973 hK      request->io_Length = 1L;
974 jV      SendIO(request);
975 l5      return(YES);
976 kFO }
977 x3      ConMayGetChar(consolePort,request,wheret0)
978 9S      struct Port *consolePort;
979 sN      struct IOStdReq *request;
980 Vq      char *wheret0;
981 lE      {
982 5w3      register temp;
983 oU      if ( GetMsg(consolePort) == NULL ) return(-1);
984 qT6      temp = *wheret0;
985 sJ3      QueueRead(request,wheret0);
986 Op      return(temp);
987 vQ0 }
988 iQ      ConGetChar(consolePort,request,wheret0)
989 2X      struct IOStdReq *request;
990 a2      struct MsgPort *consolePort;
991 gI      char *wheret0;
992 wP      {
993 FB3      register UBYTE temp;
994 4e      while((GetMsg(consolePort) == NULL)) WaitPort(consolePor
t);
995 QP      temp = *wheret0; /* get the character */
996 3U      QueueRead(request,wheret0);
997 NO      return(temp);
998 6b0 }
999 Sq      CursorOn()
1000 4X      {
1001 nP3      ConPutStr(WMsg, "\x9b\x20\x70");
1002 AF0 }
1003 Cf      CursorOff()
1004 8b      {

```

```

1005 783      ConPutStr(WMsg, "\x9b\x30\x20\x70");
1006 EJ0 }
1007 HB      long
1008 IO      dos_packet(port,type,arg1,arg2,arg3,arg4,arg5,arg6,arg7)
1009 Ar      struct MsgPort *port;
1010 xK      long type, arg1, arg2, arg3, arg4, arg5, arg6, arg7;
1011 F1      {
1012 5N3      register struct StandardPacket *sp;
1013 xy      register struct MsgPort *rp;
1014 Ta      char *zz;
1015 z7      long ret;
1016 Or      if ((rp = CreatePort(0L, 0L)) == 0)
1017 SV6      return(FAILED);
1018 F83      if ((sp = (struct StandardPacket *)
1019 U9D      AllocMem((long)sizeof(*sp), MEMF_PUBLIC|MEMF_C
LEAR)) == 0) {
1020 fU6      DeletePort(rp);
1021 WZ      return(FAILED);
1022 Uz3 }
1023 pX      zz="BBBBB";
1024 eN      sp->sp_Msg.mn_Node.ln_Name = (char *)&sp->sp_Pkt;
1025 5W      sp->sp_Pkt.dp_Link = &sp->sp_Msg;
1026 yT      sp->sp_Pkt.dp_Port = rp;
1027 8H      sp->sp_Pkt.dp_Type = type;
1028 AE      sp->sp_Pkt.dp_Arg1 = arg1;
1029 lM      sp->sp_Pkt.dp_Arg2 = arg2;
1030 WU      sp->sp_Pkt.dp_Arg3 = arg3;
1031 hc      sp->sp_Pkt.dp_Arg4 = arg4;
1032 sk      sp->sp_Pkt.dp_Arg5 = arg5;
1033 3s      sp->sp_Pkt.dp_Arg6 = arg6;
1034 EO      sp->sp_Pkt.dp_Arg7 = arg7;
1035 mn      if (arg1 == TRUE) {
1036 7n6      locate(6,20);
1037 Ip      ConPutStr(WMsg, "DOS is confused. Please reset your Am
iga.");
1038 kF3 }
1039 4I      PutMsg(port, &sp->sp_Msg);
1040 Br      locate(6,20);
1041 a6      ConPutStr(WMsg, "
");
1042 8A      WaitPort(rp);
1043 fs      GetMsg(rp);
1044 Xd      ret = sp->sp_Pkt.dp_Res1;
1045 lO      FreeMem(sp, (long)sizeof(*sp));
1046 5u      DeletePort(rp);
1047 ZQ      return(ret);
1048 uPO }
1049 DR      WriteFastTwoHeads(1,drive)
1050 YE      int 1,drive;
1051 tM      {
1052 QG3      int head;
1053 SJ      int j,k;
1054 QW      int verify=0; /* reset amount of verify-err
ors */
1055 Ir      for (k=0; k<fastavail-1; l++) {
1056 gv6      if (1>flag[4])
1057 OH9      return(1);
1058 J66      if((j=ConMayGetChar(consoleReadPort,RMsg,&readstring)
)==27) {
1059 l19      MotorOff(drive);
1060 9C      return(FAILED);
1061 7o6 }
1062 dF      locate(6,30);
1063 9Y      sprintf(string,"Writing cylinder %d ",1);
1064 MI      ConPutStr(WMsg,string);
1065 VO      for (head=0; head<2; head++) {
1066 pF9      erase=fastdiskpuffer[k++]; puffmerk=diskpuffer[0]
;
1067 lh0 ***asm
1068 Ou9      movem.l d0/a0-a1,-(sp)
1069 em      move.w **#1407,d0
1070 u3      move.l _erase,a0
1071 kt      move.l _puffmerk,a1
1072 c60 laber3      move.l (a0)+,(a1)+
1073 rDB      dbra d0,laber3
1074 TY9      movem.l (sp)+,d0/a0-a1
1075 zt0 ***endasm
1076 pT      writing1:
1077 fE9      if(WriteCylSec(drive,1,head,0)==FAILED)
1078 RUC      return(FAILED);
1079 yT9      if (flag[0]=='y') { /* verify ON */
1080 tEC      ReadCylSec(drive,1,head,0);
1081 cE      if (diskreq[drive]->iotd_Req.io_Error)
1082 aqF      if (diskreq[drive]->iotd_Req.io_Error==29)

```



```

1083 weI      {
1084 x5          locate(6,22);
                  ConPutStr(WMsg,"Destination is \x9b0;33;4
                  Om"
1085 AkX          "removed\x9bm. Try it agai
                  n !");
1086 C9I          MotorOff(drive);
1087 B3          if(WaitForDisk(drive,"")==FAILED)
1088 beL          return(FAILED);
1089 8tI          locate(6,25);
1090 om          ConPutStr(WMsg,"
                  ");
1091 8V          MotorOn(drive);
1092 4m          goto writing1;
1093 d8F          }
1094 mY          else { /* verify-error detected */
1095 2LI          if (++verify < flag[1]) {
1096 kAL          if((VerifyError(1,head,drive))==FAILED
                  )
                  return(FAILED);
                  else
                  goto writing1;
1097 knO          }
1098 XKL          else {
1099 BtO          anzdest[drive]--;
1100 kFI          return(NO);
1101 Wm          }
1102 SmL          }
1103 00          }
1104 oJI          }
1105 pKF          }
1106 qL9          }
1107 rM6          }
1108 sN3          }
1109 E7          return(1);
1110 uP0          }
1111 V1          VerifyError(1,head,drive)
1112 hV          int l,head,drive;
1113 tM          {
1114 Y03          int j;
1115 ny          locate (6,21);
1116 35          ConPutStr(WMsg," \x9b0;33;40m"
1117 7HI          "Verify Error\x9bm at cylinder ");
1118 Is3          sprintf(string,"%d, Head %d !!!",l,head);
1119 FB          ConPutStr(WMsg,string);
1120 8d0          /* Abort if ESC is pressed */
1121 ot3          if ((j=ConMayGetChar(consoleReadPort,RMsg,&readstring))=
                  =27) {
1122 mJ6          MotorOff(drive);
1123 AD          return(FAILED);
1124 8d3          }
1125 Re          FullSeek(2,drive); /* move head one time */
1126 y9          locate (6,21);
1127 kR          ConPutStr(WMsg,"
                  ");
1128 X6          return(OK);
1129 Di0          }
1130 ft          WriteFastOneHead(1,drive)
1131 rX          int l,drive;
1132 Cf          {
1133 kb3          int j,k;
1134 mB          int verify=0;
1135 wb          for (k=0 ; k<fastavail ; ) {

```

```

1136 yD6          if (l>flag[4])
1137 gZ9          return(1);
1138 bO6          if((j=ConMayGetChar(consoleReadPort,RMsg,&readstring)
                  )==27) {
1139 309          MotorOff(drive);
1140 RU          return(FAILED);
1141 Pu6          }
1142 3T          erase=fastdiskpuffer[k++]; puffmerk=diskpuffer[0];
1143 Fv0          **#asm
1144 E86          movem.l d0/a0-a1,-(sp)
1145 s0          move.w **#1407,d0
1146 8H          move.l _erase,a0
1147 y7          move.l _puffmerk,a1
1148 JMO laber4 move.l (a0)+(a1)+
1149 9WA          dbra d0,laber4
1150 hm6          movem.l (sp)+,d0/a0-a1
1151 D70          **#endasm
1152 Br          writing3:
1153 Qc6          if(WriteCylSec(drive,l,flag[5],0)==FAILED)
1154 f19          return(FAILED);
1155 G36          if (flag[0]=='y') { /* verify */
1156 wG9          ReadCylSec(drive,l,flag[5],0);
1157 qS          if (diskreq[drive]->iotd_Req.io_Error)
1158 o4C          if (diskreq[drive]->iotd_Req.io_Error==29) {
1159 ASF          locate(6,22);
1160 BJ          ConPutStr(WMsg,"Destination is \x9b0;33;40m"
1161 OyU          "removed\x9bm. Try it again !
                  ");
1162 QNF          MotorOff(drive);
1163 PH          if(WaitForDisk(drive,"")==FAILED)
1164 psI          return(FAILED);
1165 M7F          locate(6,25);
1166 20          ConPutStr(WMsg,"
                  ");
1167 MJ          MotorOn(drive);
1168 QA          goto writing3;
1169 rMC          }
1170 dt          else {
1171 GZF          if (++verify < flag[1]) {
1172 M5I          if((VerifyError(1,flag[5],drive))==FAILED
                  )
                  return(FAILED);
                  else
                  goto writing3;
1173 y1L          }
1174 LYI          else {
1175 XHK          anzdest[drive]--;
1176 yTF          return(NO);
1177 kO          }
1178 gOI          }
1179 cE          }
1180 2XF          }
1181 3YC          }
1182 4Z6          }
1183 xy          l++;
1184 6b3          }
1185 SL          return(1);
1186 8d0          }
(C) 1987 M&T

```

Listing 1. (Schluß)

LAUFWERKE

- 3 1/2" Amiga Extern**
Formsch. Metallgehäuse helle
Front, 880 KB, durchgef. Port,
mit Schraubverr. abschaltbar **279,-**
- 3 1/2" Amiga Intern**
Komplett mit Einbausatz
und Anleitung **219,-**
- 5 1/4" Amiga Extern**
Formsch. Metallgehäuse helle
Front, 40/ 80 Spur, durchgef.
Port mit Schraubverr.
abschaltbar **339,-**
- 3 1/2" Atari ST Extern**
wie oben, 720 KB, 2 x 80
Spur, eig. Netzteil **289,-**
- 5 1/4" Atari ST Extern**
wie oben, 720 KB, 40/ 80
Spur, eig. Netzteil **349,-**

Rainbow Data

SPICHERERWEITERUNGEN

- 512 KB Ram** Speicherkapazität
mit Uhr und Abschaltung a. A.
- 2 MB Box Extern**
f. Amiga 500/ 1000
Ab- und umschaltbar **1098,-**
mit durchgef. Port **1148,-**
- 20 MB Festplatte**
Amiga 500/ 1000 **949,-**
Amiga 2000 **859,-**
Atari ST **949,-**

COMPUTERLEITUNGEN

- Druckerkabel**
Amiga 500/ 1000/ 2000/ **23,-**
- Monitorkabel**
Amiga/ Scart **25,-**
- Emulatorkabel**
C 64 - Amiga **19,90**
- Bootselector**
DF 0/ DF 1 oder 2-3 **19,-**
- Mouse-Pad**
antistatisch, rutschfest **16,90**
- Weitere Angebote auf Anfrage
Preisänderungen vorbehalten

DISKETTEN

- 3 1/2" No Name 2 DD **23,90**
3 1/2" Seika 2001 2 DD **28,-**
3 1/2" Maxell 2 DD **37,50**
3 1/2" 3M DS **39,50**
5 1/4" No Name 48 TPI **8,90**
5 1/4" No Name 96 TPI **13,50**

DRUCKER

- Epson LQ 500 **1049,-**
Star LC 10 **598,-**
NEC P 6 **1198,-**
Präsident 6320 **585,-**

MONITORE

- Commodore 1084 **649,-**
Atari St SM 124 **398,-**

Erfragen Sie unsere aktuellen Tages- und Staffelpreise. Versand per Nachnahme.

Rainbow Data, Am Kalkofen 32, 5603 Wülfrath, Tel.: 02058/1366

Die besten Tage des Jahres

Der Amiga sagt Ihnen an welchen Tagen Sie Bäume ausreißen können.

Die gute grafische Darstellung und einige Besonderheiten machen dieses Biorhythmus-Programm für jeden interessant.

Die sogenannten Biorhythmen sind Sinuskurven, die am Tage der Geburt eines Menschen beim Nullpunkt starten. Die klassische Theorie des Biorhythmus kennt Kurven für den geistigen, physischen (körperlichen) und seelischen Zustand. Die Kurven besitzen verschiedene Zyklen, laufen also sehr bald nach der Geburt auseinander. Kritische Tage sind die Umkehrpunkte der Kurven sowie der Durchgang durch die Mittellinie der y-Achse. Nach dem Starten des Programmes Bio (Listing 1) erscheint das Hauptmenü mit den sieben Menüpunkten und den Daten des letzten Programmlaufs. Als erstes sollte man das aktuelle Datum eingeben, da die folgenden Angaben dieses Datum benutzen. Als nächstes ist ein Geburtsdatum einzugeben, das zur Berechnung der verschiedenen Kurven benötigt wird. Das Programm gibt aus, an welchem Wochentag man geboren wurde und wieviel Tage man alt ist. Danach wird gefragt, ob man in die Datei mit den Geburtsdaten aufgenommen werden will. Stehen die Daten schon in dieser Datei, kann man sie bei späteren Programmläufen mit dem zweiten Menüpunkt wieder laden und erspart sich die Mühe der erneuten Eingabe.

Hat man all diese Eingaben erledigt, kann man sich die Biorhythmuskurve ansehen. Außer den Kurven für Körper, Seele und Geist wird auch die Gesamtkurve gezeichnet (Bild 1). Zusätzlich werden die Werte für das aktuelle Datum noch zahlenmäßig dargestellt. Nach einem Druck auf eine beliebige Taste gelangt man wieder ins Hauptmenü. Nachdem man den fünften Menüpunkt ausgewählt hat, wird die Gesamtkurve für das laufende Jahr ge-

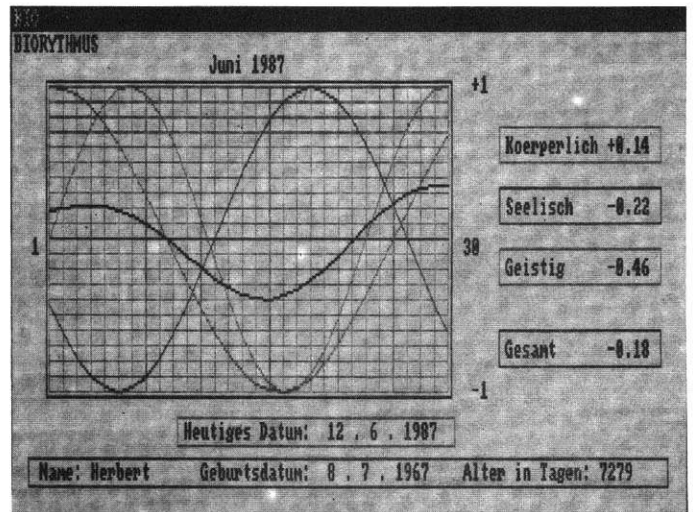


Bild 1. Die Kurven für einen Monat, hier die des Autors

zeichnet. Mit dem Menüpunkt »Extremwerte berechnen« kann man die Tage des laufenden Monats anzeigen lassen, an denen die Kurven ihr Minimum oder Maximum erreichen.

(Herbert Schwabl/rs)

Programmname:	Bio V1.1
Computer:	Amiga 500, Amiga 1000, Amiga 2000 mit Kickstart 1.2
Sprache:	Amiga-Basic

```

10 Rx4 SUB Init (n$,t,m,j,tg,mg,jg,a,Anz,Tage(),Monat$,TagNa
11 F36 me$()) STATIC
12 MP8   SHARED Datum$(),Namen$()
13 OI    SCREEN 2,640,200,4,2
14 om    WINDOW 2,"BIO", (0,0)-(631,186),0.2: WINDOW OUTPUT 2
15 in    FOR i=0 TO 9: READ r,g,b: PALETTE i,r,g,b: NEXT i
16 SN    DATA 1,.87,.73,0,0,0,.3,.5,.9,.9,.5,.57,.9,.08,.83,
17 aZ    .23,.77,0
18 S1    DATA .37,.77,.9,.9,.9,.07,1,1,1,.7,.7,.7
19 MU    DEFINT x: DIM x(256)
20 VQ    FOR i=0 TO 127: x(i)=-128: NEXT i
21 7n    FOR i=128 TO 255: x(i)=127: NEXT i: WAVE 0,x: ERASE
22 uM    x
23 VV    FOR i=1 TO 12: READ Tage (i),Monat$ (i): NEXT i
24 sH    DATA 31,"Januar",29,"Februar",31,"März",30,"April"
25 9v    DATA 31,"Mai",30,"Juni",31,"Juli",31,"August"
26 9f    DATA 30,"September",31,"Oktober",30,"November",31,"
27 CC    Dezember"
28 kX    FOR i=1 TO 8: READ TagName$ (i): NEXT i
29 eu    DATA "Montag","Dienstag","Mittwoch","Donnerstag"
30 X2B    DATA "Freitag","Samstag","Sonntag","Montag"
31 WC8    OPEN ":Namen.dat" FOR INPUT AS #2
32 6K    INPUT #2,Anz
33 1J    FOR i=1 TO Anz
        INPUT #2,Namen$ (i),Datum$ (i)
    NEXT i: CLOSE #2
    n$="Herbert": t=28: m=4: j=1987: tg=8: mg=7: jg=196
    7
    CALL Alter (tg,mg,jg,t,m,j,a)

```

```

34 b1 CALL Daten (n$,tg,mg,jg,t,m,j,a)
35 bd4 END SUB
36 y6 SUB Menue STATIC
37 t98 CALL Clr: PRINT "MENU": LINE (233,6)-(398,17),2,b
38 pH LOCATE 2,31: PRINT "B I O R Y T H M U S"
39 1F LOCATE 5,10: PRINT "1) Geburtsdatum eingeben"
40 HF LOCATE 7,10: PRINT "2) Geburtsdatum aus Datei uebe
    rnehmen"
41 Ra LOCATE 9,10: PRINT "3) Heutiges Datum eingeben"
42 xx LOCATE 11,10: PRINT "4) Biorhythmus berechnen"
43 EA LOCATE 13,10: PRINT "5) Jahreskurve berechnen"
44 Lm LOCATE 15,10: PRINT "6) Extremwerte berechnen"
45 ko LOCATE 18,10: PRINT "7) Programm beenden"
46 mo4 END SUB
47 XS SUB Eingabe (n$,tg,mg,jg,a,Anz,Namen$(),Datum$()) STATI
    C
48 lQ5 SHARED t,m,j,Anzahl,Tage(),Monat$,TagName$()
49 v68 CALL Clr: PRINT "EINGABE"
50 zh LOCATE 3,3: PRINT "Geben Sie bitte Ihren Namen ein:
    "
51 FY LOCATE 5,6: nn$=n$: INPUT n$: SOUND 1000,1
52 xk IF n$="" THEN n$=nn$: EXIT SUB
53 8b n$=LEFT$ (n$,11): LOCATE 8,3
54 Gc PRINT "Geben Sie bitte Ihr Geburtsdatum im Format '
    TT-MM-JJJJ' ein:"
55 Km Korrekt=0
56 lu WHILE (NOT Korrekt)
57 9BB LOCATE 10,6: PRINT "
    "
58 zK LOCATE 10,6: INPUT d$: SOUND 1000,1
59 FJ PRINT: PRINT "
    ";
60 WD PRINT "
    "
61 c6 CALL Kontrolle (d$,tg,mg,jg,Korrekt)
62 RF8 WEND
63 Vn CALL Alter (tg,mg,jg,t,m,j,a)
64 bF CALL Tag (tg,mg,jg,TagName$())
65 HL CALL Sternzeichen (tg,mg,jg,s$)

```



```

66 PJ CALL Clr: PRINT: PRINT " Hallo ";n$;" !"
67 RV PRINT: PRINT " Du bist am ";TagName$(0);", den
";
68 BN PRINT tg;". ";Monat$(mg);". ";jg;" geboren,"
69 XX PRINT " und bist am ";t$;" ";Monat$(m);". ";j$;" ";
a;"Tage alt !"
70 5J PRINT: PRINT " Dein Sternzeichen ist: ";s$
71 TC IF Anz<Anzahl THEN
72 MqB PRINT: i$=""
73 BI PRINT " Willst Du in die Datei aufgenommen werd
en ? (J/N)"
74 wF WHILE (i$<>"n")
75 ZFE i$=INKEY$
76 UY IF i$<>" " AND i$<>"j" AND i$<>"n" THEN
BEEP
IF i$="n" THEN SOUND 1000,1
IF i$="j" THEN
SOUND 500,1: Anz=Anz+1: i$="n"
Namens$(Anz)=n$: Datum$(Anz)=d$
CALL SpeicherDatei (Anz,Namens$(),Datum$())
END IF
WEND
END IF
IF Anz>=Anzahl THEN CALL Taste
CALL Daten (n$,tg,mg,jg,t,m,j,a)
END SUB
88 Pa SUB Datei (n$,tg,mg,jg,a,Anz,Namens$(),Datum$()) STATIC
89 055 SHARED t,m,j,Anzahl,Tage(),Monat$(),TagName$()
90 D16 CLS: PRINT "DATEI: ";Anz;"Eintraege Noch frei:"
;Anzahl-Anz
LINE (240,13)-(405,25),2,b
LOCATE 3,32: PRINT " N A M E N L I S T E"
PRINT: PRINT TAB (68);"Alter am"
PRINT " Name Geburtsdatum
";
95 7K PRINT "Sternzeichen ";STR$(t);". ";STR$(m);". ";ST
R$(j)
96 FX PRINT: s=-1
97 kO FOR i=1 TO Anz
98 oU9 IF INKEY$<>" " THEN SOUND 1000,1: s=NOT s
99 Ab IF s THEN
100 57C CALL Nummer (LEFT$(Datum$(i),2),tt,Korrekt)
101 Zn CALL Nummer (MID$(Datum$(i),4,2),mm,Korrekt)
102 Gy CALL Nummer (RIGHT$(Datum$(i),4),jj,Korrekt)
103 mB CALL Alter (tt,mm,jj,t,m,j,a)
104 xN CALL Sternzeichen (tt,mm,jj,s$)
105 I19 ELSE
106 jPC s$="": a=0
107 fY9 END IF
108 2A PRINT " ";
109 3B PRINT " ";
110 xu LOCATE CSRLIN,1: PRINT USING "_ # # _:";i;
111 Iq PRINT TAB (7);Namens$(i);TAB (35);Datum$(i);
112 No IF s THEN
113 z5C PRINT TAB (50);s$;TAB (68);USING "### # # #
";a
114 RA9 ELSE
115 DqC PRINT
116 oh9 END IF
117 Ag IF i MOD 10=0 AND i<>Anz THEN
118 8yC PRINT: PRINT
119 Gm PRINT " ----- A)uswahl N)aechste Seite -
-----";
Eingabe3:
i$=INKEY$
IF i$<>" " AND i$<>"a" AND i$<>"n" THEN
BEEP
IF i$="a" THEN SOUND 500,1: GOTO Auswahl
IF i$<>"n" THEN Eingabe3
SOUND 1000,1: LINE (8,50)-(600,180),0,bf: LOCAT
E 8,1
END IF
NEXT i
Auswahl:
LOCATE 22,4
PRINT "Bitte Nummer der gewuenschten Person einge
ben: ";
INPUT n: SOUND 1000,1: n=INT (n)
IF n<1 OR n>Anz THEN n=1
n$=LEFT$(Namens$(n),12)
CALL Nummer (LEFT$(Datum$(n),2),tg,Korrekt)
CALL Nummer (MID$(Datum$(n),4,2),mg,Korrekt)
CALL Nummer (RIGHT$(Datum$(n),4),jg,Korrekt)
CALL Alter (tg,mg,jg,t,m,j,a)

```

```

138 vb CLS: CALL Daten (n$,tg,mg,jg,t,m,j,a)
139 HJ4 END SUB
140 fo SUB EingabeDatum (t,m,j,a) STATIC
141 sj5 SHARED n$,tg,mg,jg,Tage(),Monat$(),TagName$()
142 OR8 CALL Clr: PRINT "EINGABE": LOCATE 3,3
143 qO PRINT "Geben Sie bitte das heutige Datum im Format
'TT-MM-JJJJ' ein:"
144 Tn Korrekt=0: dd$=d$
145 jL WHILE (NOT Korrekt)
146 whB LOCATE 5,6: PRINT "
"
147 xp LOCATE 5,6: INPUT d$:SOUND 1000,1
148 cZ IF d$="" THEN dd$=d$: EXIT SUB
149 vY PRINT: PRINT "
"
150 xa CALL Kontrolle (d$,t,m,j,Korrekt)
151 sg8 WEND
152 wE CALL Alter (tg,mg,jg,t,m,j,a)
153 QZ CALL Tag (t,m,j,TagName$()): CALL Clr
154 ot PRINT: PRINT " Heute ist also ";TagName$(0);",
der ";
155 4B PRINT t;". ";Monat$(m);". ";j$;" ."
156 MD PRINT " ";n$;" ist heute genau ";a;" Tage alt !"
157 dJ CALL Taste: CALL Daten (n$,tg,mg,jg,t,m,j,a)
END SUB
158 ao4 SUB Biorythmus (tg,mg,jg,t,m,j) STATIC
159 O2 SHARED Tage()
160 jQ6 CALL BioGrafik (t,m,j)
161 3x8 CALL Alter (tg,mg,jg,1!,m,j,a)
162 AQ pi=3.14151265#
163 hB pp=60*SIN ((a-23*INT (a/23))/23*2*pi)
164 Gu ee=60*SIN ((a-28*INT (a/28))/28*2*pi)
165 sP gg=60*SIN ((a-33*INT (a/33))/33*2*pi)
166 r1 ss=INT ((pp+ee+gg)/3)
167 Rn FOR i=1 TO Tage (m)-1
168 OY p=60*SIN ((a+1-23*INT ((a+1)/23))/23*2*pi)
169 eTB e=60*SIN ((a+1-28*INT ((a+1)/28))/28*2*pi)
170 s1 g=60*SIN ((a+1-33*INT ((a+1)/33))/33*2*pi)
171 cB s=INT ((p+e+g)/3)
172 Ez IF a+1>0 THEN
173 BL LINE (20+i*13,80-pp)-(33+i*13,80-p),3
174 qOE LINE (20+i*13,80-gg)-(33+i*13,80-g),6
175 yk LINE (20+i*13,80-ee)-(33+i*13,80-e),5
176 U9 LINE (20+i*13,80-ss)-(33+i*13,80-s),1
177 Tk pp=p: ee=e: gg=g: ss=s
178 9F ELSE
179 UDB pp=0: gg=0: ee=0: ss=0
180 oKE END IF
181 rkB NEXT i
182 Nd8 CALL Alter (tg,mg,jg,t,m,j,a)
183 RJ p=SIN ((a-23*INT (a/23))/23*2*pi)
184 it e=SIN ((a-28*INT (a/28))/28*2*pi)
185 6L g=SIN ((a-33*INT (a/33))/33*2*pi)
186 Yd s=(p+e+g)/3
187 Gh LOCATE 6,71: PRINT USING "+. #. #. #";p
188 NG LOCATE 9,71: PRINT USING "+. #. #. #";e
189 hS LOCATE 12,71: PRINT USING "+. #. #. #";g
190 Ou LOCATE 16,71: PRINT USING "+. #. #. #";s
191 th CALL Taste
192 qU END SUB
193 9B4 SUB Extreme (n$,tg,mg,jg,t,m,j) STATIC
194 O6 CALL ExtremGrafik (n$,m,j)
195 NP8 CALL Alter (tg,mg,jg,1!,m,j,a)
196 iy CALL Kritisch (4!,23!,a)
197 pF CALL Kritisch (10!,28!,a)
198 MQ CALL Kritisch (16!,33!,a)
199 bc CALL Taste: CLS
200 Yb CALL Daten (n$,tg,mg,jg,t,m,j,a)
201 IS END SUB
202 IK4 SUB Jahreskurve (tg,mg,jg,j,m) STATIC
203 EE CALL JahresGrafik (j,m)
204 Ce8 CALL Alter (tg,mg,jg,1!,1!,j,a)
205 81 pi=3.14151265#
206 Os pp=60*SIN ((a-23*INT (a/23))/23*2*pi)
207 xb ee=60*SIN ((a-28*INT (a/28))/28*2*pi)
208 Z6 gg=60*SIN ((a-33*INT (a/33))/33*2*pi)
209 Y1 ss=INT ((pp+ee+gg)/3)
210 8U FOR i=1 TO 365
211 jk p=60*SIN ((a+1-23*INT ((a+1)/23))/23*2*pi)
212 LAB

```

Listing 1. »Bio« zeigt Ihnen die Tage an, an denen Sie Bäume ausreißen könnten. Bitte mit dem Checksummer (Seite 159) eingeben.


```

213 ZS      e=60*SIN ((a+1-28*INT ((a+1)/28))/28*2*pi)
214 Js      g=60*SIN ((a+1-33*INT ((a+1)/33))/33*2*pi)
215 vg      s=INT ((p+e+g)/3)
216 s2      IF a+1>0 THEN
217 JzE      LINE (70+1*479/365,80-ss)-(70+(i+1)*479/365,8
              0-s),1
218 mG      ss=s
219 8rB      ELSE
220 PoE      ss=0
221 VOB      END IF
222 1H8      NEXT i
223 Lz      CALL Taste
224 eg4      END SUB
225 w1      SUB Kritisch (l,n,a) STATIC
226 Q86      SHARED m,j,Tage()
227 2X8      pi=3.14159265#: p=1: k=1
228 vS      FOR i=0 TO Tage (m)-1
229 heB      s=SIN ((a+1-n*INT ((a+1)/n))/n*2*pi)
230 rg      IF a+1>=0 THEN
231 gqE      IF s>-.98 AND ss<-.98 THEN
232 tJH      LOCATE 1,10: l=l+1
233 88      PRINT USING "# #.# #.# #.# #.# #.#";i+1;m;j
234 1bE      END IF
235 az      IF s<-.98 AND ss>-.98 THEN
236 DxH      LOCATE p,62: p=p+1
237 CC      PRINT USING "# #.# #.# #.# #.# #.#";i+1;m;j
238 mFE      END IF
239 G1      IF ABS (s)<-.15 AND ABS (ss)>-.15 THEN
240 EzH      LOCATE k,36: k=k+1
241 GG      PRINT USING "# #.# #.# #.# #.# #.#";i+1;m;j
242 qJE      END IF
243 rkB      END IF
244 Cg      ss=s
245 Oe8      NEXT i
246 024      END SUB
247 BK      SUB ExtremGrafik (n$,m,j) STATIC
248 FE6      SHARED Monat$()
249 dI8      LINE (0,0)-(600,165),0,bf
250 3D      LOCATE 1,1: PRINT "EXTREMWERTE im ";Monat$ (m);j
251 Ct      LINE (13,10)-(616,161),1,b
252 oR      LINE (213,11)-(213,160),9
253 XM      LINE (426,11)-(426,160),9
254 ez      LOCATE 20,10: PRINT "Aktive Tage"
255 zH      LOCATE 20,33: PRINT "Kritische Tage"
256 bf      LOCATE 20,60: PRINT "Passive Tage"
257 Np      LINE (16,12)-(613,52),3,b
258 FT      LOCATE 3,4: PRINT "Koepferlich"
259 ZN      LINE (16,60)-(613,100),5,b
260 k5      LOCATE 9,4: PRINT "Seelisch"
261 CA      LINE (16,108)-(613,148),6,b
262 m1      LOCATE 15,4: PRINT "Geistig"
263 HJ4      END SUB
264 EQ      SUB BioGrafik (t,m,j) STATIC
265 ik6      SHARED Monat$(),Tage()
266 ie8      CALL Clr: PRINT "BIORYTHMUS"
267 e3      LOCATE 2,24: PRINT Monat$ (m);j
268 RI      LOCATE 3,55: PRINT "+1"
269 60      LOCATE 18,55: PRINT "-1"
270 9Q      LOCATE 11,3: PRINT "1"
271 kE      LOCATE 11,23+Tage (m): PRINT STR$ (Tage (m))
272 Ry      IF m=2 AND j MOD 4=0 THEN LOCATE 11,51: PRINT STR$
              (28);" "
273 sn      xm=Tage (m)*13+20
274 cq      FOR i=33 TO xm STEP 13
275 54B      LINE (i,20)-(i,140),9
276 t98      NEXT i
277 QO      FOR i=20 TO 140 STEP 6
278 97B      LINE (33,i)-(xm,i),9
279 wC8      NEXT i
280 3g      LINE (30,18)-(xm+3,142),1,b
281 Zi      LINE (33,80)-(xm,80),2
282 MP      LINE (20+13*t,20)-(20+13*t,140),4
283 nK      LINE (458,36)-(610,50),3,b
284 c3      LOCATE 6,59: PRINT "Koerperlich"
285 7Z      LINE (458,60)-(610,74),5,b
286 IO      LOCATE 9,59: PRINT "Seelisch"
287 oJ      LINE (458,84)-(610,98),6,b
288 TU      LOCATE 12,59: PRINT "Geistig"
289 7Q      LINE (458,116)-(610,130),1,b
290 au      LOCATE 16,59: PRINT "Gesamt"
291 j14      END SUB
292 xO      SUB JahresGrafik (j,m) STATIC
293 93B      CALL Clr: PRINT "BIORYTHMUS"
294 KC      LOCATE 2,37: PRINT j

```

```

295 ye      LOCATE 3,72: PRINT "+1"
296 Sj      LOCATE 18,72: PRINT "-1"
297 h1      LOCATE 11,5: PRINT "Jan"
298 gL      LOCATE 11,72: PRINT "Dez"
299 JF      LINE (30+m*40,20)-(70+m*40,140),4,bf
300 Jf      FOR i=70 TO 550 STEP 40
301 VUB      LINE (i,20)-(i,140),9
302 JZ8      NEXT i
303 qo      FOR i=20 TO 140 STEP 6
304 MBB      LINE (70,i)-(550,i),9
305 Mc8      NEXT i
306 Z1      LINE (67,18)-(553,142),1,b
307 DU      LINE (70,80)-(550,80),2
308 024      END SUB
309 67      SUB SpeicherDatei (Anz,Namen$(),Datum$()) STATIC
310 v68      OPEN "Namen.dat" FOR OUTPUT AS #2
311 fPB      PRINT# 2,Anz
312 DT      FOR i=1 TO Anz
313 IOE      WRITE #2,Namen$ (i),Datum$ (i)
314 V1B      NEXT i
315 K88      CLOSE# 2
316 8A4      END SUB
317 Mu      SUB Kontrolle (d$,tt,mm,jj,Korrekt) STATIC
318 Hy5      SHARED Tage()
319 G48      LOCATE CSRLIN-1,1: Korrekt=-1
320 P6      IF LEN (d$)<>10 THEN Korrekt=0
321 4a      CALL Nummer (LEFT$ (d$,2),tt,Korrekt)
322 sz      CALL Nummer (MID$ (d$,4,2),mm,Korrekt)
323 9c      CALL Nummer (RIGHT$ (d$,4),jj,Korrekt)
324 i2      IF NOT Korrekt THEN
325 mnB      BEEP: PRINT "Fehlerhafte Eingabe !"
326 Pd      EXIT SUB
327 D68      END IF
328 ut      IF jj<0 OR jj>3000 THEN
329 roB      BEEP: Korrekt=0
330 U1      PRINT "Bitte nur Jahreszahlen zwischen 0000 und
              3000 eingeben !"
331 U1      EXIT SUB
332 IB8      END IF
333 XL      IF mm<1 OR mm>12 THEN
334 CNB      BEEP: PRINT "Fehlerhaftes Monat !": Korrekt=0
335 Ym      EXIT SUB
336 MF8      END IF
337 Qm      IF tt<1 OR tt>Tage (mm) THEN
338 uzB      BEEP: PRINT "Fehlerhafter Tag ! ": Korrekt=0
339 PI8      END IF
340 I1      IF mm=2 AND jj MOD 4<>0 AND tt>28 THEN
341 x2B      BEEP: PRINT "Fehlerhafter Tag ! ": Korrekt=0
342 SL8      END IF
343 Zb4      END SUB
344 6F      SUB Nummer (i$,n,Korrekt) STATIC
345 WV8      n=0
346 sS      FOR i=1 TO LEN (i$)
347 4zB      IF MID$ (i$,i,1)>"9" OR MID$ (i$,i,1)<"0" THEN
              Korrekt=0
              EXIT SUB
              END IF
              n=10*n+ASC (MID$ (i$,i,1))-48
              NEXT i
348 3VE      END SUB
349 mO      SUB Alter (tg,mg,jg,t,m,j,a) STATIC
350 aTB      IF m<3 THEN
351 hT      m1=m+12:j1=j-1
352 7N8      ELSE
353 j14      m1=m:j1=j
354 VI      END IF
355 4v      IF mg<3 THEN
356 dN6      m2=mg+12:j2=jg-1
357 M54      ELSE
358 jP6      m2=mg:j2=jg
359 jc4      END IF
360 Me      IF m2=mg+12:j2=jg-1
361 1b6      ELSE
362 RA4      m2=mg:j2=jg
363 Xw6      END IF
364 oh4      a=INT (365.25*j1)-INT (j1/100)+INT (j1/400)+INT (30
              .6001*(m1+1))+t
365 dW8      b=INT (365.25*j2)-INT (j2/100)+INT (j2/400)+INT (30
              .6001*(m2+1))+tg
              a=a-b
366 PM      END SUB
367 3Z      SUB Tag (tt,mm,jj,TagName$()) STATIC
368 yO4      CALL Alter (tt,mm,jj,0!,0!,0!,n)
369 Yy      n=ABS (n): n=n-7*INT (n/7)+2
370 BX8      TagName$ (0)=TagName$ (n)
371 1s      END SUB
372 yZ      SUB Sternzeichen (tt,mm,jj,s$) STATIC
373 354
374 TQ

```



```

375 CH8      IF mm= 1 AND tt>=21 OR mm= 2 AND tt<=19 THEN s$="
              Wassermann"
376 VG       IF mm= 2 AND tt>=20 OR mm= 3 AND tt<=20 THEN s$="
              Fische"
377 jK       IF mm= 3 AND tt>=21 OR mm= 4 AND tt<=20 THEN s$="
              Widder"
378 Ot       IF mm= 4 AND tt>=21 OR mm= 5 AND tt<=20 THEN s$="
              Stier"
379 nv       IF mm= 5 AND tt>=21 OR mm= 6 AND tt<=21 THEN s$="
              Zwillinge"
380 mn       IF mm= 6 AND tt>=22 OR mm= 7 AND tt<=22 THEN s$="
              Krebs"
381 JN       IF mm= 7 AND tt>=23 OR mm= 8 AND tt<=23 THEN s$="
              Loewe"
382 Gd       IF mm= 8 AND tt>=24 OR mm= 9 AND tt<=23 THEN s$="
              Jungfrau"
383 FF       IF mm= 9 AND tt>=24 OR mm=10 AND tt<=23 THEN s$="
              Waage"
384 lR       IF mm=10 AND tt>=24 OR mm=11 AND tt<=22 THEN s$="
              Skorpion"
385 hL       IF mm=11 AND tt>=23 OR mm=12 AND tt<=21 THEN s$="
              Schuetze"
386 lR       IF mm=12 AND tt>=22 OR mm= 1 AND tt<=20 THEN s$="
              Steinbock"
387 HJ4      END SUB
388 fF       SUB Clr STATIC
389 CM8       LINE (0,0)-(617,145),0,bf: LOCATE 1,1
390 KM4      END SUB
391 Q6       SUB Taste STATIC
392 xK8       WHILE (INKEY$="")
393 ma        WEND
394 ns        SOUND 1000,1
395 PR4      END SUB
396 o9       SUB Daten (n$,tg,mg,jg,t,m,j,a) STATIC
397 Po8       LINE (154,150)-(416,161),0,bf
398 sC        LINE (154,150)-(416,161),3,b
399 Oh        LOCATE 20,21: PRINT "Heutiges Datum: ";t;". ";m;". ";
              j

```

```

400 W6       LINE (13,166)-(616,177),0,bf
401 Vd       LINE (13,166)-(616,177),5,b
402 OZ       LOCATE 22,4: PRINT "Name: ";n$;TAB (23);"Geburtsdat
              um: ";
403 Ef       PRINT tg;". ";mg;". ";jg
404 x6       LOCATE 22,54: PRINT "Alter in Tagen: ";a
405 Zb4      END SUB
406 YR0      BEGIN:
407 D34      Anzahl=100
408 ER       DIM Tage (12), Monat$ (12), TagName$ (8)
409 uC       DIM Namen$ (Anzahl), Datum$ (Anzahl)
410 sK       ON ERROR GOTO DateiAnlegen
411 TC       OPEN "df0:Namen.dat" FOR INPUT AS #2
412 dF       CLOSE #2: GOTO Weiter
413 sb2      DateiAnlegen:
414 Ln4      Anz=1: Namen$ (1)="Herbert": Datum$ (1)="08-07-1967"
415 7M       CALL SpeicherDatei (Anz,Namen$(),Datum$())
416 ZP2      Weiter: CALL Init (n$,t,m,j,tg,mg,jg,a,Anz,Tage(),Monat$
              ()),TagName$())
417 Cj4      CALL Menue
418 wM       WHILE (i$<>"7")
419 xv8       i$=INKEY$: IF i$>="1" AND i$<="6" THEN SOUND 500,
              1
420 rI       IF i$="1" THEN CALL Eingabe (n$,tg,mg,jg,a,Anz,Name
              n$(),Datum$())
421 g9       IF i$="2" THEN CALL Datei (n$,tg,mg,jg,a,Anz,Namen$
              ()),Datum$())
422 DW       IF i$="3" THEN CALL EingabeDatum (t,m,j,a)
423 DY       IF i$="4" THEN CALL Biorythmus (tg,mg,jg,t,m,j)
424 rh       IF i$="6" THEN CALL Extreme (n$,tg,mg,jg,t,m,j)
425 OA       IF i$="5" THEN CALL Jahreskurve (tg,mg,jg,j,m)
426 y2       IF i$>="1" AND i$<="6" THEN CALL Menue
427 JQ       IF (i$<"1" OR i$>"7") AND i$<>" " THEN BEEP
428 L94      WEND
429 aT       WINDOW CLOSE 2: SCREEN CLOSE 2
430 gb0      END
(C) 1987 M&T

```

Listing 1. (Schluß)

philgerma
NEWS
pro

die
künstliche
Intelligenz
für Ihren
AMIGA

log



- ☐ schnell: 1800 Lips
- ☐ komfortabel:
DEBUG und TRACE
- leistungsfähig:
Edinburgh Standard

Preis: auf Anfrage

Demoversion
anfordern!
(DM 15,-)

philgerma
Barerstr. 32
8000 München 2
TEL. 089-281228

philgerma
NEWS

die neue
Version 4.0
bricht alle
Rekorde

LATTICE



COMPILER

für AMIGA mit
Blink Linker
Symbolic Debugger
Quellen Editor
Utilities
Libraries

Normalversion:
DM 398,-

Entwicklerversion:
DM 748,-

unerreicht schnell

Info-Material
anfordern!

philgerma
Barerstr. 32
8000 München 2
TEL. 089-281228

Wenn urplötzlich Pferdewieher aus Ihrem Wohnzimmer dringt, muß das noch lange kein Grund zur Panik sein. Vermutlich experimentiert nur jemand mit einem Amiga-Sounddemo. Das enorme Klangvolumen und die klanggetreuen Imitationen von Geräuschen und Instrumenten verblüffen immer aufs neue.

Das Geheimnis des exzellenten Amiga-Sounds liegt in der Tatsache, daß dieser Com-

Erleben Sie live die Faszination digitalisierter Klänge mit unserem Selbstbau-Digitizer und »Digi-soft plus«, dem digitalen Soundstudio zum Abtippen. Das Klangwunder ist perfekt.

Digitizer im Selbstbau Software inklusive

puter keine Tongeneratoren mit festgelegten Klangcharakteristiken besitzt, sondern jeder Ton durch Wiedergabe einer frei definierbaren Wellenform entsteht. Damit läßt sich jeder denkbare Klang generieren, es muß nur dessen Wellenform im Computer gespeichert sein.

Hier liegt nun das Problem, vor dem jeder musikbegeisterte Amiga-Fan steht, der einen tollen Sound mit dem Computer spielen oder ein Geräusch aufnehmen will: Wie bringt man die Töne in eine für den Computer verständliche Form?

Hier tritt nun unser Sound-Digitizer in Aktion, der am Amiga 1000 (Bild 1, Seite 46) und mit einer kleinen Zusatzplatine auch an Amiga 500/2000 (Bild 7, Seite 51) betrieben werden kann. Er berechnet aus einem analogen Eingangssignal (der Tonschwingung) die entsprechenden digitalen Werte und schickt diese zum Computer, der sie dann weiterverarbeitet. Die Anwendungen reichen vom Aufzeichnen von Musik über die Verwendung selbstdigitalisierter Instrumente und Geräusche in Musik- oder Spielprogrammen bis hin zum Speicheroszillografen.

Doch bevor man sich an solche Projekte heranwagen kann, muß erst einmal der Digitizer gebastelt werden. Unsere Schaltung kommt mit einem Minimum an Bauteilen aus, die zudem keine Schwierigkeiten bei der Beschaffung machen sollten. Die Kosten für den Digitizer liegen bei rund 60 Mark, wobei der Wandlerbaustein al-

leine auf 40 bis 50 Mark kommt. Das Schaltbild sehen Sie in Bild 2 (Seite 46).

Am besten bauen Sie den Digitizer auf einer Platine auf, wie sie Bild 3 (Seite 47) zeigt. Alle benötigten Bauteile sind in der Tabelle zusammengefaßt. Die Anordnung der Bauteile ist in Bild 4 (Seite 47) dargestellt. Bitte sockeln Sie unbedingt das

wertvolle Wandler-IC und schützen Sie es dadurch vor Überhitzung beim Einlöten. Die 25polige Kupplung (für den Druckerport des Amiga) löten Sie mit der oberen Reihe (13 Stifte) direkt an die Digitizer-Platine, um so eine kompakte Einheit mit kurzen Signalwegen zu schaffen. Die untere Reihe (12 Pins) ist bis auf zwei



BAUANLEITUNG



tung, die zu dem —5-Volt-Anschluß unseres Digitizers führt. Ein Tip für Bastler, die einen kleinen Amiga-Umbau nicht scheuen: Pin 24 des parallelen Druckerports ist nicht belegt und läßt sich im Amiga mit Pin 14 der seriellen Schnittstelle verbinden, worauf auch am Druckerport —5-Volt anliegen. Das Kabel und die Kupplung für die serielle Schnittstelle entfallen sodann, und die negative Spannung wird am Druckerport-Pin 24 abgegriffen. Der Amiga 500/2000 stellt leider an keinem Port —5 Volt zur Verfügung. Mit einem kleinen Trick läßt sich diese Spannung jedoch aus den vorhandenen +5 Volt erzeugen.

Digitizeranschluß		Computeranschluß
Pin-Nummer		Pin-Nummer
2	-	2
3	-	3
4	-	4
5	-	5
6	-	6
7	-	7
8	-	8
9	-	9
11	-	11
13	-	13
14	-	18
23	-	14

Tabelle 1. Anschlüsse für das Adapterkabel zum Betrieb mit Amiga 500/2000

Der 10 μ F-Kondensator zwischen Pin 2 und 4 dient zur Umladung des internen Oszillators. An Anschluß 3 liegt die Masse des Computers. Der am Ausgang (Pin 5) befindliche 10 μ F-Kondensator dient als Block- und Siebkondensator. Es ist allerdings noch zu erwähnen, daß die positive Spannung direkt von Pin 23 des 25poligen Steckers am Digitizer zur Zusatzplatine führt. Bild 6 (Seite 50) zeigt Ihnen den kompletten Stromlaufplan und wie Sie die Schaltung an die Grundschialtung anbinden. Die Schaltung wird am besten auf einer kleinen Lochrasterplatine aufgebaut.

Da Amiga 500/2000 am Druckerport eine andere Belegung aufweisen, als der Amiga 1000, benötigen Sie noch ein Adapterkabel mit zwei männlichen, 25poligen Sub-D-Steckern auf jeder Seite, das Sie nach den Angaben in der Tabelle 1 anfertigen. Alle nicht aufgeführten Anschlüsse bleiben unverbunden. Geschickte Hardware-Bastler können die Pinänderungen natürlich auch gleich auf der Grundplatine vornehmen. Die fertig umgebaute Schaltung sehen Sie in Bild 7 (Seite 51).

Nun fehlt nur noch die Verbindung des Digitizers mit der Tonquelle. Ein einadriges, abgeschirmtes Kabel wird mit der Abschirmung an Masse, mit der Signalleitung an den Toneingang des Digitizers (siehe Bild 4) gelötet. Am anderen Ende des Kabels sollte sich ein Stecker für den Kopfhörer-Ausgang ihrer Stereoanlage oder einer anderen Tonquelle befinden. Der äußere Teil eines Klin-

kensteckers für die Kopfhörer-Buchse ist mit Masse, der Innere mit der Signalleitung zu verbinden, wobei auf eventuelle

Kurzschlüsse zu achten ist.

Bevor wir uns an einen ersten Test wagen, hier für interessierte Tüftler noch einige technische Daten. Der 7574 ist ein 8-Bit-Analog-Digital-Wandler mit einer minimalen Wandlungszeit von 15 Mikrosekunden. R4 und C5 bestimmen die Wandlungszeit und können bei Timingproblemen vergrößert werden. Die Chip-Select-Leitung ist fest auf Masse gelegt, was bedeutet, daß die Read/Write-Leitung alleine den Wandlungsvorgang starten kann. Geht Read/Write nach 0 Volt, so startet die Umset-

Ausnahmen nicht belegt: zum einen muß Pin 14 des Steckers mit einem blanken Stück Draht nach außen zum Ende der breiten Masseleitung auf der Platinen-Unterseite geführt werden (diese Anschlußstelle ist auf der Platine mit einer »14« gekennzeichnet). Zum anderen muß man auch eine Leitung vom offenen Ende des Wider-

standes R1 zu Pin 23 des Steckers ziehen.

Am Druckerport des Amiga stehen alle Leitungen bereit, die unser Digitizer zum Betrieb benötigt; bis auf eine —5-Volt-Leitung. Beim Amiga 1000 liegt diese Spannung an der seriellen Schnittstelle. Wir löten daher an einen 25poligen Stecker für den seriellen Port eine Lei-

Der ICL 7660 erzeugt aus einer anliegenden positiven Spannung eine gleichwertige negative. Seine Anschlußbelegung finden Sie in Bild 5 (Seite 49). Er besitzt acht Anschlüsse, von denen nur fünf beschaltet sind. Die an Pin 8 zugeführte positive Spannung stellt der Baustein als negative Spannung an Pin 5 zur Verfügung.

zung und Busy wechselt ebenfalls nach 0 Volt. Liegt der digitalisierte Wert an den Datenregistern an, so schnellst Busy auf 5 Volt und die 8-Bit-Daten können ausgelesen werden.

Jetzt wird's ernst: Computer ausschalten, Digitizer in den Centronics-Druckerport stecken (die Bauteile müssen nach oben zeigen) und den Stecker für die Spannungsversorgung zum seriellen Port führen. Starten Sie nun den Amiga wie gewohnt.

Listing 1 (Seite 48) zeigt den Assembler-Quellcode eines einfachen Testprogramms im Format des Seka-Assemblers. Besitzer dieses Assemblers müssen nur das Listing eingeben, assemblieren und rufen

Stecker in einen passenden Ausgang an der Anlage einstecken. Am besten eignet sich für Digitalisierungen der Kopfhörer-Anschluß am Verstärker oder Receiver. Die Klangfarbe und die optimale Lautstärke für den Digitizer sind dort am leichtesten einzustellen. Doch halt! Drehen Sie bitte zunächst den Lautstärke-regler auf Null. Der Digitizer muß an die Anlage angeschlossen sein, darf aber noch keinen Ton empfangen.

Durch gefühlvolles Einstellen des Potentiometers P1 auf der Digitizer-Platine müssen Sie jetzt eine Stelle suchen, an der der zunächst dunkelblaue Bildschirm schwarz wird und nicht flackert. Haben Sie die optimale Einstellung gefun-

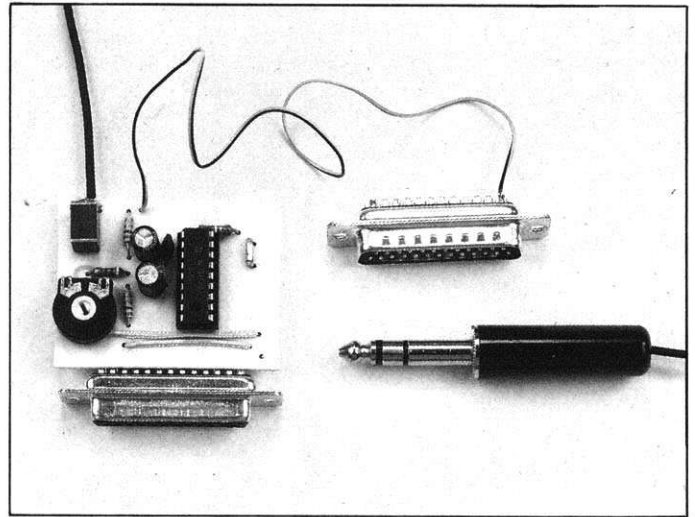
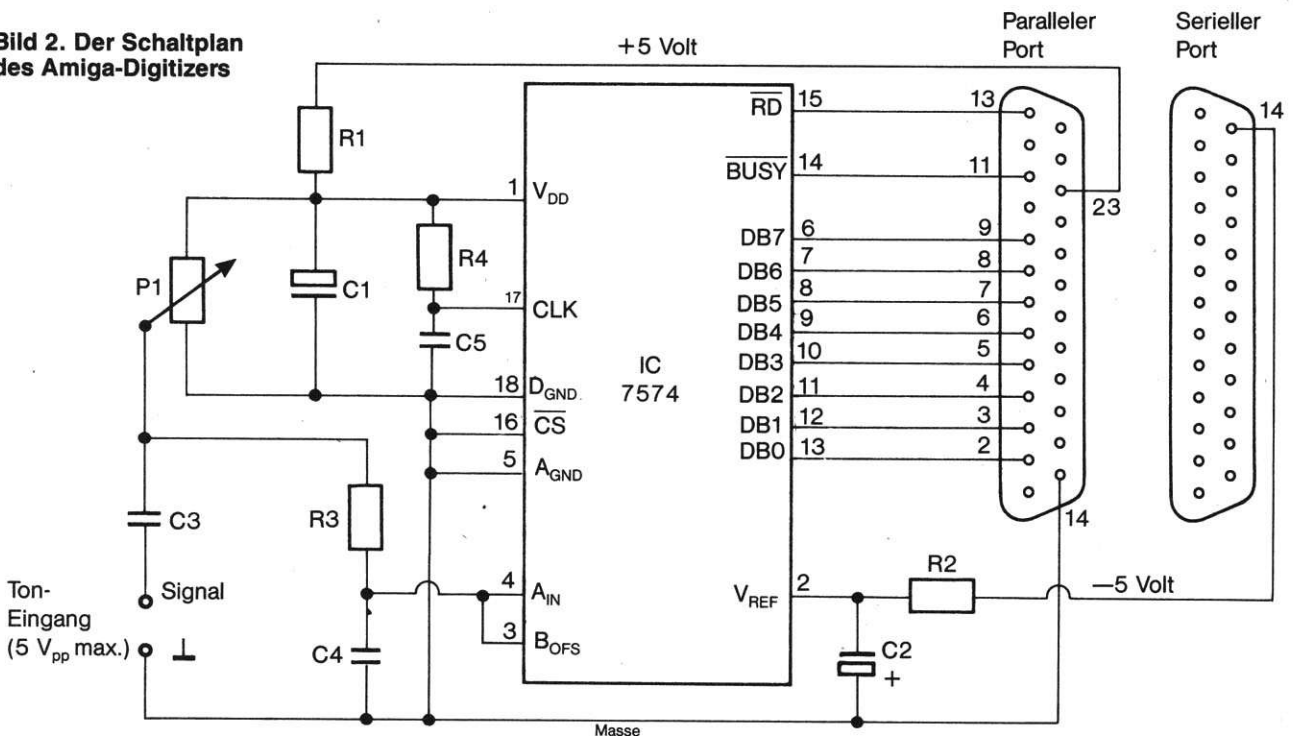


Bild 1. Der fertig aufgebaute Digitizer in der Version für Amiga 1000

Bild 2. Der Schaltplan des Amiga-Digitizers



es anschließend mit »g start« auf. Es sollten dabei rund 80 KByte Arbeitsspeicher reserviert sein (Workspace auf 80 setzen). Sind Sie nicht im Besitz eines Assemblers, so finden Sie auf der Programmserve-Diskette zu diesem Sonderheft auch das ablauffähige Programm.

Bei dem Maschinenprogramm handelt es sich um ein Testprogramm zur Überprüfung der einwandfreien Funktionstüchtigkeit des Amiga-Digitizers und zur Abstimmung der Schaltung. Zu diesem Zweck starten Sie zunächst das Programm und verbinden den Digitizer mit der Stereoanlage, indem Sie den Kopfhörer-

den, so kann der Verstärker aufgedreht werden! Während die Musik läuft, sind auf dem Bildschirm schwarze und blaue Linien zu sehen, die der Musik entsprechend flackern. Sind gelegentlich rote Striche zu sehen, so ist der Digitizer optimal angesteuert. Leiser stellen heißt es, wenn viele rote Streifen erscheinen.

Experimentieren Sie nun ausgiebig mit Lautstärke, Baß- und Höhenreglern des Verstärkers. So bekommen Sie bald ein Gefühl dafür, wie man einen optimalen Klang aus dem Digitizer zaubert. Ein langanhaltender Druck auf die linke Maustaste beendet das Programm. Unser Testprogramm

ist allerdings noch nicht auf optimale Tonqualität ausgelegt. Außerdem bietet es keinerlei Editiermöglichkeiten für die digitalisierten Klänge.

Das digitale Soundstudio

Hier greift nun die Software für unseren Digitizer ein: ein paarmal die Maustaste gedrückt, und schon ist jeder Klang in Bits und Bytes gebannt. Die komfortable Menüsteuerung und der übersichtliche Bildschirmaufbau garantieren eine unkomplizierte und dennoch schnelle Bedienung aller Funktionen. Dabei kennt

»Digisoft plus« nicht nur alle Standardbefehle wie Aufnehmen, Wiedergeben, Laden und Speichern, sondern beherrscht zudem viele Spezialfunktionen zum Mischen, Kopieren oder Rückwärts-Spielen. Die eingebaute grafische Ausgabe der Klangdaten ist dabei ständig sichtbar.

Bei »Digisoft plus« handelt es sich um eine Weiterentwicklung des bekannt-bewährten »Digisoft« mit wesentlichen Verbesserungen. Um nun in den Genuß dieses Klangwunders zu kommen, müssen die Listings 3 bis 12 abgetippt und mit dem Aztec C kompiliert werden. Eingabehinweise finden Sie auf Seite 159. Dabei ist

zu beachten, daß vor dem Compilieren der Listings 4 bis 11 das Listing 3 compiliert werden muß, da das dabei erzeugte File »dmp« in diese eingebunden wird. Listing 12 muß assembliert werden. Nach dem Laden kann das Programm vom CLI aus aufgerufen werden. Dabei sollte in den Preferences der 80-Zeichen-Modus eingestellt sein.

Nach dem Laden des Steuerprogramms sehen Sie schon die wichtigsten Bedienelemente auf dem Monitor (Bild 8, Seite 51). Am oberen Rand des Bildschirms liegt eine Menüleiste, die wie gewohnt durch Festhalten der rechten Maustaste aktiviert wird. Den Hauptteil des Bildschirms vereinnahmt die grafi-

Für jede Spur existiert ein eigener Parameterblock, in dem alle zur Aufnahme und Wiedergabe eines Kanals notwendigen Werte angezeigt werden und veränderbar sind. Unmittelbar unter der grafischen Anzeige werden die Parameter des angewählten Tracks dargestellt.

Die Maus gehört zum guten Ton

Neben der jeweiligen Spurnummer befindet sich das »Status«-Feld, in dem entweder »ON« oder »OFF« zu lesen ist. »ON« bedeutet, daß die Spur gespielt wird, »OFF« schaltet den Kanal stumm. Ein Mausklick auf dieses Feld schaltet zwischen beiden Zu-

tus auf »ON« schalten oder das »Play«-Feld anklicken.

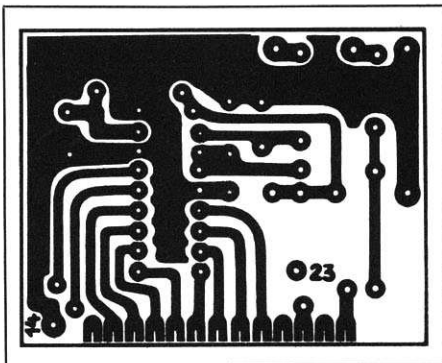
Der im »Start«-Feld angezeigte Wert gibt an, ab welcher Position der Amiga die in der jeweiligen Spur gespeicherten Klangdaten spielen soll. Steht dort der Wert »0«, wird von Anfang an gespielt. Sie ändern den Wert zum Beispiel dadurch, daß Sie das Feld mit der Maus anklicken, worauf der Cursor erscheint und Sie den neuen Wert eintippen können. Die Cursortasten, und <Backspace> erleichtern und beschleunigen die Eingabe. Wer mit der Tastatur auf Kriegsfuß steht, klickt einfach das Feld neben dem zu ändernden Wert an und erhöht oder verringert diesen sodann mit den über der unteren Gad-

Die nun folgenden Parameter lassen sich nach dem selben Prinzip einstellen. »End« bestimmt das Ende der Sounddaten und markiert zusammen mit »Start« den Sounddaten-Bereich, der gespielt oder bearbeitet werden soll. »Repeat« enthält, falls gewünscht, einen Wert, der zwischen »Start« und »End« liegt. Diese Zahl ist nur bei manchen digitalisierten Instrumenten von Belang und gibt den Beginn des Wiederholtheils (Loop) an. Verwenden Sie ein selbstdigitalisiertes Instrument in einem IFF-kompatiblen Musikprogramm, so wird nach dem Drücken einer Taste zuerst der Ton komplett von »Start« bis »End« gespielt und danach der Bereich von »One-Shot« bis »End« so lange wiederholt, bis die Taste losgelassen wird. Der Repeat-Punkt wird in der grafischen Anzeige durch eine senkrechte Linie markiert, genauso die Start- und Endpositionen. Bei »Name« kann man dem Track eine entsprechende Bezeichnung verleihen. Mit »Volume« legt man die Lautstärke zwischen 0 (Stille) und 64 (sehr laut) fest. »Period« ist der Geschwindigkeitswert für den Amiga-Soundchip und sollte zwischen minimal 124 und maximal 1000 liegen. Je niedriger der Wert, desto schneller ist die Aufnahme- und Abspielgeschwindigkeit und entsprechend erhöht sich die Klangqualität. Werte unter 256 bringen normalerweise keine hörbare klangliche Verbesserung. Indirekt proportional zu diesem Wert verhält sich »Rate«, das die Anzahl der digitalisierten Bytes pro Sekunde angibt. Dabei gilt: Je niedriger »Period«, desto höher »Rate« und desto höher damit auch der Speicherplatzbedarf bei gleicher Spieldauer.

Die Schalter »Zoom out«, »Zoom in« und »Display Range« werden weiter unten beschrieben. Die eingestellten Parameter gelten immer für den jeweils aktivierten Track. Genauso verhält es sich mit der grafischen Anzeige.

Qualität ist kein Zufall

Die großen »Play«- und »Stop«-Felder am unteren Bildschirmrand gelten für alle Stimmen und starten oder beenden die Wiedergabe aller Spuren, deren Status auf »ON« gestellt ist. Die Digitalisierung eines Klangs starten »Mouse Rec« und »Auto Rec«, »Mouse

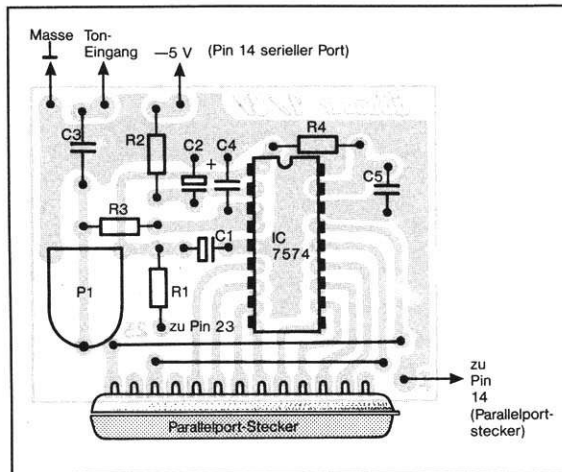


▲ Bild 3. Das Platinenlayout. Bitte achten Sie darauf, daß zwischen Rahmen und Leiterbahnen kein Kontakt entsteht.

sche Anzeige des angewählten Tracks. Über dieser befinden sich vier getrennte Blöcke mit Schaltern für die einzelnen Spuren von »Digisoft plus«. Unter der Anzeige befindet sich eine Reihe symbolisierter Gadgets für Aufnahme, Wiedergabe und vieles mehr.

Sehen wir uns nun die einzelnen Komponenten etwas genauer an. Um eine Tonspur zu wählen, muß der Schalter mit der entsprechenden Nummer am oberen Bildschirmrand angeklickt werden. Viele der per Menü aufrufbaren Funktionen und das Aufnehmen beziehen sich immer auf die gerade angewählte Spur.

Diese Spuren spielen bei unserem Steuerprogramm eine wichtige Rolle. Jedem der vier Amiga-Tonkanäle ist ein eigener Speicherbereich mit eigenen Parametern zugeordnet, so daß insgesamt vier verschiedene Klänge aufgezeichnet, manipuliert und gleichzeitig wiedergegeben werden können. In Anlehnung an ein Mehrspur-Tonbandgerät nennen wir diese vier Sound-Speicher auch Spuren.



▲ Bild 4. So wird die Platine bestückt. Achten Sie auf die Einbaueinrichtung des ICs (Kerbe).

Bauteileliste

IC	7574 (z.B. AD7574, MP7574) Hersteller: Analog Devices, Tel. 089/57 00 50
P1	Potentiometer 10 kOhm linear
R1, R2	Widerstand 1 Ohm
R3	Widerstand 910 Ohm
R4	Widerstand 100 kOhm
C1, C2	Kondensator 100 µF, 10 Volt
C3	Kondensator 680 nF
C4	Kondensator 22 nF
C5	Kondensator 120 pF
	Stecker 25polig f. RS232-Port (nur Amiga 1000)
	Kupplung 25polig f. Drucker-Port Platine, IC-Sockel, Drahtbrücken

Die folgenden Bauteile nur für Amiga 500/2000
ICL 7660
C6, C7 Kondensator 10 µF
zwei Stecker 25polig f. Adapter

ständen um. Dieser Schalter ist jedoch nur dann wirksam, wenn die Wiedergabe mit der »Play«-Taste aktiviert wurde.

Mit einem Mausklick wechseln Sie beim »Mode«-Feld zwischen »LOOP« und »ONCE«. Im »LOOP«-Modus wird die Spur laufend wiederholt, bei »ONCE« ertönt der digitalisierte Klang des jeweiligen Kanals nur einmal, wenn Sie den Sta-

get-Reihe liegenden Pfeilsymbolen. Statt der Pfeilsymbole eignet sich auch der Schieberegler zum Einstellen des gewünschten Wertes. Klicken Sie dazu das kleine, viereckige Kästchen im Schieber mit der Maus an und halten Sie die Maustaste gedrückt. Setzen Sie dieses Kästchen an die richtige Stelle und lassen Sie dort die Maustaste los.


```

; --- Amiga-Digitizer Testprogramm ---
; ----- 68000er 1/87 -----
; --Copyright 1986 Markt&Technik Verlag AG --
buffer=$40000
len=$E000 ; Musikspeicherlänge
speed=182 ; Geschwindigkeit

base1=$BFE001
base2=$BFD000
dra2=base2
drb1=base1+$100
ddra2=base2+$200
ddrb1=base1+$300

start: move.l 4,a6 ; Supervisor-Modus
lea super,a5
jmp -30(a6)
super: addq.l #8,a7
move # $2700,sr ; Interrupts sperren
move.b # $00000000,ddrb1 ; Parallel-Port
move.b ddra2,d0 ; initialisieren
andi.b # $11111010,d0
ori.b # $00000100,d0
move.b d0,ddra2

move.l #buffer,$dff0a0 ; Soundparameter
move #64,$dff0a8 ; setzen
move #speed,$dff0a6
move #len/2,$dff0a4
move.l #buffer,$dff0b0
move #64,$dff0b8
move #speed,$dff0b6
move #len/2,$dff0b4
move # $8003,$dff096
lea block,a1 ; Farben
move # $0000,block
move # $0F00,block+254
move # $0F00,block+256

schleife: lea buffer,a0
move #len,d3

digit: ori.b # $00000100,dra2 ; Digitalisierung
andi.b # $11111011,dra2 ; starten
busy: andi.b # $00000001,dra2 ; Noch beschäftigt?
beq.s busy ; Ja, warten
clr d1
move.b drb1,d1 ; Sounddaten holen

eorl.b #128,d1 ; Umrechnung
move.b d1,(a0)+ ; Wert speichern

asl #1,d1 ; Farbe berechnen
move (a1,d1),$dff180

dbra d3,digit
andi.b #64,$bfe001 ; Linke Maustaste...
bne.s schleife ; ...nicht gedrückt

move # $0003,$dff096 ; Sound-DMA aus
move #0,sr ; User-Modus
illegal ; =) Seka
even

block: blk.w 256,8

```

Listing 1. Der Assembler-Quellcode des Digitizer-Testprogramms

Rec« beginnt nach einem Druck auf die linke Maustaste mit der Aufzeichnung auf dem angewählten Kanal. Ein weiterer Mausklick beendet die Aufnahme. »Auto Rec« wartet mit der Aufnahme so lange, bis

auch wirklich ein Ton am Digitizer anliegt. Dieses Verfahren setzt allerdings einen optimal justierten Digitizer voraus.

»Listen« dient zum Abhören der momentan am Digitizer anliegenden Musik, »Adjust« zum

Abgleich des Digitizers. Bei absoluter Stille muß der Bildschirm dabei bis auf einen schmalen Streifen am oberen Rand eine flackerfreie, schwarze Farbe annehmen. Damit ist der Nullpunkt definiert. Nun den Lautstärkeregler an der Stereo-Anlage so weit aufdrehen, bis der Bildschirm schwarz und blau flackert und sich gelegentlich feine, rote Streifen zeigen. Leiser stellen heißt es bei dicken roten Streifen. Bitte beachten Sie, daß die Tonqualität bei »Test« und »Adjust« immer etwas schlechter ist als bei der Aufzeichnung.

Kommen wir nun zu den Pull-Down-Menüs. Im »Disk«-Menü erlauben die beiden Menüpunkte »Load« und »Save« das Laden und Speichern von digitalisierten Tönen und Geräuschen auf Diskette, Festplatte oder RAM-Disk. Die Eingabe des Dateinamens erfolgt ganz komfortabel über eine Datei-Auswahl-Box. Auch alle verfügbaren Volumes werden angezeigt und können ganz einfach angeklickt werden. Beim Laden wird nach einer Sicherheitsabfrage die aktuelle Spur gelöscht und automatisch genügend Speicher reserviert. Gespeichert wird immer der durch »Start« und »End« definierte Datenbereich der angewählten Spur. Sowohl beim Laden als auch beim Speichern haben Sie die Wahl zwischen drei verschiedenen Formaten: »IFF«, »Future Sound« oder »Data«. Bei »Data« handelt es sich ausschließlich um die Klangdaten, während der IFF-Standard auch die Parameter enthält. Beim »Future Sound«-Format steht vor den Daten noch die Länge der Datei (ein Longword) und die Wiedergabegeschwindigkeit (ein Word).

Im »Track«- und im »Range«-Menü verbergen sich nützliche Funktionen, die sich fast immer mit der Manipulation der gerade angewählten Spur befassen. »Clear« löscht die aktuelle Spur und gibt den dafür reservierten Speicher wieder frei, »Clear all« löscht alle Kanäle. »Reset Track« setzt die Parameter auf ihren Ursprungswert zurück. »Copy Range« kopiert alle Sounddaten der angewählten Spur von »Start« bis »End« in eine andere Spur, die nach dem Aufruf der Funktion eingegeben wird. Die Speicherreservierung für diese Spur erfolgt bei Bedarf automatisch.

»Concat Range« funktioniert im Prinzip genauso wie »Copy Range«, nur wird diesmal die

Zielspur nicht gelöscht, sondern die zu kopierenden Daten werden an die Sounddaten des Zielkanals angehängt. Damit können Sie beispielsweise einzelne Wörter aus einem in Spur 1 digitalisierten Satz herauskopieren und in Spur 2 zu einem neuen Ausspruch zusammensetzen. Mit »Paste Range« wird der aktuelle Bereich ab der Endposition des Zieltracks an diesen angehängt. »Insert Range« fügt den aktuellen Bereich beim Zieltrack an dessen Startmarkierung ein.

Im »Effects«-Menü finden sich noch weitere Extras. So kann mit »Reverse Range« der Bereich zwischen »Start« und »End« umgedreht werden. Das heißt er wird rückwärts abgespielt. Mit »Mix Range« mischen Sie den Bereich mit einer anderen Spur. »Period * 2« verdoppelt die Periode des Bereichs. Dazu werden die Daten im Speicher verdoppelt. Dies bewirkt, daß der Klang eine Oktave tiefer kommt. »Period / 2« hat genau den umgekehrten Effekt.

Das Menü »Options« erlaubt die Auswahl verschiedener Digitizer-Typen. Wenn Sie den Digitizer aus unserer Bauanleitung verwenden, stellen Sie »68000er« original ein.

Im großen Fenster in der Bildschirmmitte werden die Klangdaten der momentan angewählten Spur grafisch dargestellt. Damit sehen Sie sofort, ob die Aufnahme übersteuert ist, wo der Ton beginnt, wo er endet und vieles mehr. Drei senkrechte Linien stellen Start-, End- und Repeat-Position dar. Sie können durch einfaches Ziehen mit der Maus verschoben werden. Normalerweise wird der gesamte Track in gestauchter Form dargestellt (Bild 8, Seite 51). Wird nun ein Track abgespielt, so läuft synchron zur Musik eine senkrechte Linie über den dargestellten Bereich. Diese zeigt an, welcher Teil des Samples gerade ausgegeben wird. Klicken Sie »Display Range« an, so wird auf die »1:1-Darstellung« umgeschaltet und die Daten werden so angezeigt, wie sie im Speicher stehen (Bild 9, Seite 51). Es werden dann ab der Startmarkierung so viele Daten dargestellt, wie im Fenster Platz finden.

Mit »Zoom In« ist der zwischen Start- und Endmarkierung liegende Bereich über das ganze Display zu sehen. Es kann so stufenlos gezoomt werden. Bei »Zoom Out« wird wieder auf die ursprüngliche Darstellung umgeschaltet.

Die Titelzeile des »Digisoft plus«-Fensters zeigt den freien Speicherbereich an. Dabei werden Chip- und Fast-Memory getrennt angezeigt. »Digisoft plus« erkennt selbstständig Speichererweiterungen und nutzt so den vorhandenen Speicherbereich voll aus.

Unser Digitizer und Audiomaster

Als Tüpfchen präsentieren wir Ihnen nun auch noch eine Anpassung unseres Digitizers an den weitverbreiteten »Audiomaster«. Wenn Sie also im Besitz von AudioMaster sind und damit nicht nur Geräusche nachbearbeiten, sondern auch selbst digitalisieren wollen, dann benötigen Sie einen passenden Digitizer. Was liegt also näher, als unseren Selbstbau-Digitizer dafür zu verwenden. Dazu sind also wie gesagt einige, leicht durchführbare, Anpassungen nötig.

Nehmen Sie zunächst den Digitizer zur Hand. Auf der Oberseite der Platine sehen Sie zwei unterschiedlich lange Drahtbrücken. Die längere der beiden führt mit dem einen En-

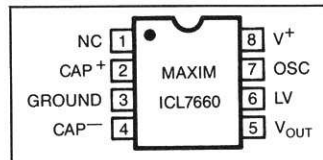


Bild 5. Die Pinbelegung des Spannung-Inverters ICL 7660

de zu Pin 13 des 25poligen Parallel-Port-Steckers. Löten Sie dieses Ende der Drahtbrücke vorsichtig aus. Führen Sie den Draht nun zu Pin 1 des Steckers, kürzen das überstehende Stück und löten ihn dort fest. Vorsicht! Es darf kein Kurzschluß zu anderen Bauteilen oder Leiterbahnen entstehen.

Mit Tuning der Hardware...

Oben haben wir beschrieben, wie Sie den Digitizer an den Amiga 500 und 2000 anpassen. Wenn Sie das dort beschriebene Adapter-Kabel verwenden, dann müssen Sie auch bei diesem die Leitung, welche normalerweise die Pins 13 der zwei Stecker des Kabels verbindet, auf beiden Seiten an den Pin 1 löten. Der Pin 1

des Digitizers muß also in jedem Fall mit dem Pin 1 des Computer-Parallel-Ports verbunden sein.

Nach diesen Änderungen funktioniert der Digitizer allerdings nicht mehr mit unserem oben beschriebenen Steuerprogramm »Digisoft plus«. Ein Umlöten der Drahtbrücke von Pin 1 zurück auf Pin 13 würde die Kompatibilität wiederherstellen. Der Einbau eines Schalters würde einfaches Umschalten ermöglichen.

Damit wären alle Umbauarbeiten abgeschlossen, welche die Hardware betreffen. Doch damit ist es leider noch nicht getan, auch die Software hat einen kleinen Tapetenwechsel nötig. Fertigen Sie sich dazu eine Sicherheitskopie Ihrer AudioMaster-Diskette an, wie im Handbuch beschrieben. Tippen Sie nun Listing 2 ein, legen die AudioMaster-Kopie in das interne Laufwerk und starten das Programm mit »RUN«.

Rund neun Minuten später, wenn sich der Amiga mit »Fertig!« meldet, ist der Software-Umbau beendet und die AudioMaster-Kopie an den Digitizer angepaßt. Bitte lassen Sie die Original-Diskette unverändert

für den Fall, daß Sie das ursprüngliche Programm irgendwann noch einmal benötigen sollten. Zudem würde beim Verändern des Originals kein Garantieanspruch mehr bestehen.

Geschafft! Dem fröhlichen Digitalisieren mit AudioMaster und unserem Amiga-Digitizer steht nichts mehr im Wege. Ganz im Gegenteil: Bei der Software-Anpassung haben wir zudem noch eine kleine Besonderheit eingebaut, über die das Originalprogramm nicht verfügt. Wenn Sie das Sampler-Window aufrufen und den Monitor-Betrieb aktivieren, dann verändert sich die Farbe des Hintergrundes in Abhängigkeit von der Lautstärke des Signals, das am Digitizer anliegt.

Zum optimalen Nullpunkt-Abgleich stellen Sie zunächst den Lautstärkereglers Ihrer Stereoanlage auf Null und suchen durch Drehen des Reglers auf der Digitizer-Platine den Nullpunkt. Der Bildschirm, normalerweise dunkelblau, muß an der richtigen Position vollkommen schwarz werden.

Starten Sie jetzt Band, Platte oder CD und stellen Sie die

» Eine neue Dimension DE LUXE SOUND V. 2. 5 « Der Audiodigitizer der Luxusklasse Exklusiv-Test im AMIGA-MAGAZIN 6/88

»Leistungsmerkmale in Stichworten«

- ★ Erzeugen aller SONIX-SOUNDS
- ★ Erzeugen aller IFF-SOUNDS (auch Instruments)
- ★ Erzeugen von DUMP-Soundfiles (Standard-Sounds)
- ★ Direct-Sampling auf bis zu 255 Disketten Nonstop
- ★ mit 2 Laufwerken (Longplay mit RECORDMAKER)
- ★ Direktes Mithören vor und während des Digitalisierens
- ★ Echo-Halleffekte in Stereo mit Standard-Sounds oder im Direct-Outputmodus ohne Digitalisierung
- ★ Klangverfremdungen (Amplituden- & Frequenzmodulation)
- ★ regelbarer hochempfindlicher Vorverstärker
- NEU Nur noch 1 Anschlußkabel (Parallel-Port) erforderlich
- NEU Wesentlich verb. DISK-Handling mit Anzeige für »FREE BYTES ON DISK«
- NEU einstellbarer Threshold-Level für Autosampling
- NEU komfortabler Schneidetisch mit Grafikanzeige
- NEU Startzeiger, Loopzeiger & Endzeiger frei einstellbar
- NEU Sound-Merging (Verketten von Einzelsounds)
- NEU SONIX & IFF-Sounds in 1-5 Oktaven speicherbar
- NEU Abspieltools (Player) in C und Assembler mit SOURCE
- NEU High-Frequency-Sampling mit allen AMIGAS bei denen der Tiefpaßfilter sich softwaremäßig abschalten läßt
- NEU LOOPING an- und abschaltbar
- NEU Umfangreiche Bedienungsanleitung
- ** Nach wie vor superschnell und kurz, da Assemblerprogramm
- ** Nach wie vor ist die HARDWARE kompatibel zu fast allen Samplerprogrammen AUDIOMASTER, PERFECT-SOUND, FUTURE-SOUND, usw.
- *** UPDATE-SERVICE-INFO unter 02381-880077

NEU AMIGA-MIDI mit Gehäuse und Anschlußkabel!!!

Unser neues AMIGA-MIDI-Interface besitzt alle wichtigen Ein- und Ausgänge: 2x MIDI-OUT, 1x MIDI-IN und 1x MIDI-THRU. Gern liefern wir Ihnen auch ein passendes MIDI-Gerät (Keyboard, Expander, Sampler). Auf Wunsch unterbreiten wir Ihnen ein individuelles Angebot nach Ihren spezifischen Angaben (z.B. bestimmter Gerätetyp) oder (Preisvorgabe). Bei Bestellung von AMIGA-MIDI bitte AMIGA-Typ angeben. Preis nur 98,- DM

NEU AMIGA-CLOCK-Echtzeituhr im Gehäuse mit Anschlußkabel. AMIGA-CLOCK ist die sinnvolle Ergänzung für alle die noch keine Echtzeituhr besitzen. Software zum Stellen und Auslesen der Uhr wird mitgeliefert. AMIGA-CLOCK ist an alle AMIGAS am JOY-PORT (durchgeschleift) anschließbar! Die erforderliche Software gehört zum Lieferumfang nur 98,- DM

Preissenkung für AMIGA-LAUFWERKE! (sehr leise)

AMIGA Einzellaufwerk 3,5" im robusten kunststofflackierten Metallgehäuse mit heller Frontblende, Busdurchführung und Abschalter kostet nur noch 278,- DM

AMIGA Einzellaufwerk 5,25" im robusten kunststofflackierten Metallgehäuse mit heller Frontblende, Busdurchführung, Abschalter, 40/80-Track-Umschaltung usw. nur noch 398,- DM

AMIGA Kombilaufwerk 5,25" & 3,5" Ausführung s.o. nur 648,- DM

WIR LIEFERN BESTMÖGLICH AB LAGER HAMM PER NACHNAHME ODER VORKASSE ZUZÜGLICH VERSANDSPESSEN.

hagenau computer
Alter Uentrop Weg 181
4700 Hamm 1
Tel: 02381-88 00 77

Unsere Produkte erhalten Sie ebenfalls bei:

Niederlande:
CAT & KORSH Int. BV
Postbus 62255
3002 Rotterdam
Tel: 010-4507696

Österreich:
MAR Computershop
Inh. Peter Rauscher
Weldengasse 41
1100 Wien
Tel: 0222-62 15 35

Schweiz:
MEGASHOP AG
Falkenplatz 7
3012 Bern
Tel: 024-4005

Schweden:
TRI-DATA
Ljungbacksv. 30
240 13 Genarp
Tel: 040-48 22 11

Norwegen:
Kreativ Teknikk
Sofies Gate 12
0170 Oslo 1
Tel: 02-460744

Lautstärke der Anlage so hoch ein, bis sich auf dem nun blau-schwarz flackernden Bildschirm gelegentlich rote Streifen zeigen. Viel Rot auf dem Bildschirm bedeutet Übersteuerung. Jetzt ist der Digitizer optimal angesteuert und die Digitalisierung kann beginnen.

...und der Software zum Erfolg

Ein Wermutstropfen für Amiga-500-Besitzer: Im Monitor-Betrieb von AudioMaster knackst es bei manchen Amiga 500-Modellen mit zu schwachem Netzteil wie bei einem Geiger-Müller-Zähler. Wenn Sie aber in den Aufnahme-Modus schalten, verschwinden diese Störungen vollständig, Sie brauchen also kein Knacken während der Digitalisierung zu befürchten. Der Grund für dieses Phänomen liegt in dem Amiga-500-Netzteil, dessen Spannung nur dann konstant ist, wenn das Amiga-Bild ausgeblendet, also die Bildschirm-DMA abgeschaltet wird. Genau dies

angesteuert. Sobald diese Leitung auf Null wechselt, beginnt der Digitizer mit dem Digitalisierungs-Prozess, der nach rund 15 Mikrosekunden abgeschlossen ist. Danach ist der digitale Amplituden-Wert an den acht Datenleitungen abrufbar. Um sicherzustellen, daß dieser Vorgang schon abgeschlossen ist, kann man vor dem Lesen des Datenregisters die Busy-Leitung des Digitizers abfragen (sie ist mit Pin 11 des Parallel-Ports verbunden). Solange dieses Signal auf Low-Pegel ist, arbeitet der Digitizer noch.

Die eben erwähnten 15 Mikrosekunden sind die kürzest mögliche A/D-Wandlungszeit.

Ein Digitalisierungs-Durchlauf sah also bislang wie folgt aus: Read-Bit löschen, Read-Bit setzen, warten, bis Busy-Bit gesetzt wird, Datenregister auslesen. Wesentlich kürzer und schneller geht dies, wenn die Read-Leitung mit Pin 1 des Parallel-Ports, dem Strobe-Signal, verbunden wird. Immer dann, wenn ein Schreibzugriff auf das Datenregister erfolgt, wechselt die Strobe-Leitung

```

1 REM AudioMaster/Amiga-Digitizer Patch
2 REM von Toni Schwaiger
3 REM Markt&Technik 68000er 1/88
4
5 SCREEN 2,950,800,1,2
6 WINDOW 2,"68000er",(0,0)-(631,10),0,2
7 w&=WINDOW(7):s&=PEEKL(w&+46)
8 a&=PEEKL(PEEKL(88+s&)+8)
9
10 OPEN "i",#1,"df0:AudioMaster"
11
12 FOR t&=a& TO a&+93863&
13 POKE t&,ASC(INPUT$(1,1))
14 NEXT t&
15
16 CLOSE #1
17
18 FOR t&=a& TO a&+39
19 READ d$
20 POKE 65976&+t&,VAL("&h"+d$)
21 NEXT t&
22
23 RESTORE
24 FOR t&=a& TO a&+39
25 READ d$
26 POKE 66048&+t&,VAL("&h"+d$)
27 NEXT t&
28 POKE 66081&+a&,212
29
30 FOR t&=a& TO a&+13
31 READ d$
32 POKE 1724&+t&,VAL("&h"+d$)
33 NEXT t&
34
35 FOR t&=a& TO a&+51
36 READ d$
37 POKE 43132&+t&,VAL("&h"+d$)
38 NEXT t&
39
40 FOR t&=a& TO a&+5

```

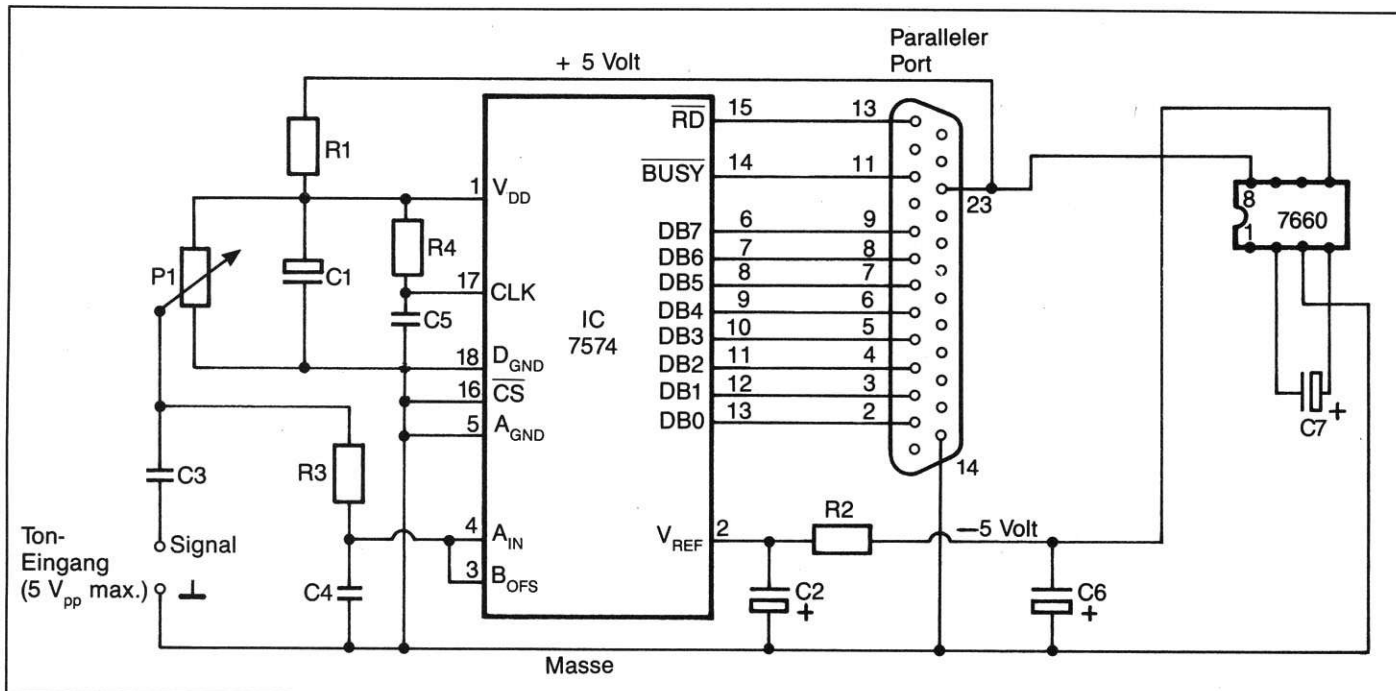


Bild 6. Der modifizierte Schaltplan des Digitizers. Betrieb am Amiga 500/2000 mit Adapterkabel.

macht AudioMaster glücklicherweise in beiden Digitalisierungs-Modi.

Abschließend noch einige Erläuterungen zu den genannten Modifikationen für alle interessierten Bastler und Amiga-Programmierer. Die Read-Leitung ging vormals zu Pin 13 des Parallel-Ports und wurde durch kurzes Löschen und darauf folgendes Setzen des entsprechenden Port-Bits

kurzzeitig auf Low und liefert damit das ideale Signal für die Read-Leitung, wie es der 7574-Chip erwartet. Der Sample-Prozess reduziert sich damit auf folgende Operationen: einen beliebigen Wert in das Datenregister schreiben, mindestens 15 Mikrosekunden warten, Datenregister auslesen. Das spart nicht nur Zeit, sondern hält auch den Programmcode klein. So klein,

daß er fast ohne Verluste in den AudioMaster-Quellcode paßt.

AudioMaster setzt voraus, daß der Digitizer einen eigenen Taktgenerator enthält und damit laufend eigenständig digitalisiert. Ein solcher Taktgenerator fehlt bei unserem Digitizer, um Kosten, Größe und Bastelarbeit möglichst klein zu halten. Zudem ist die Abstimmung exakter und damit die Klangqualität besser, wenn der

Computer das Takten übernimmt.

Das Programm wurde also in den beiden Sample-Routinen um die Operation »Schreiben in das Datenregister« (= Start der Digitalisierung) ergänzt. Die nötigen freien Bytes haben wir durch Entfernen der Befehle zum Mithören während des Sample-Vorgangs gewonnen. Die Routinen zur farblichen Darstellung von Übersteuer-

B AUANLEITUNG

```

41 READ d$
42 POKE 44128&+t&,VAL("&h"+d$)
43 NEXT t&
44
45 OPEN "o",#1,"df0:AudioMaster"
46
47 FOR t&=a& TO a&+93863&
48   PRINT #1,CHR$(PEEK(t&));
49 NEXT t&
50
51 CLOSE 1
52
53 WINDOW CLOSE 2
54 SCREEN CLOSE 2
55
56 PRINT "Fertig!"
57
58 END
59
60
61 DATA 13,FC,00,00,00,BF,E1,01,12,39,00,BF
62 DATA E1,01,04,01,00,80,10,C1,08,39,00,06
63 DATA 00,BF,E0,01,67,0A,B3,C8,62,D6,4E,71
64 DATA 4E,71,4E,71
65
66 DATA 36,38,6F,6F,6F,65,72,53,61,6D,70,6C
67 DATA 65,72
68
69 DATA 60,3C,13,FC,00,00,00,BF,E1,01,16,39
70 DATA 00,BF,E1,01,3A,3C,0F,00,0C,03,00,00
71 DATA 67,12,0C,03,00,FF,67,0C,3A,3C,00,0F
72 DATA 0C,03,00,80,66,02,42,45,33,C5,00,DF
73 DATA F1,80,4E,75
74
75 DATA 61,00,FC,1C,4E,71

```

Listing 2. Dieses Programm stimmt AudioMaster Amiga-Digitizer-freundlich

rung und Nullpunkt-Justage fanden in einer NOP-Wüste Platz, die aus unerfindlichen Gründen mitten im Programmcode ein einsames Dasein fristete. Ein abschließender Hinweis: Es kann bei der Software-Anpassung geschehen, daß das Original-AudioMaster-Icon verschwindet und statt dessen das bekannte Basic-Daten-

Icon zu sehen ist. Kopieren Sie in diesem Fall einfach das Symbol von der Original-Diskette auf die Diskette mit dem veränderten Programm. Dies geschieht mit folgendem Befehl: COPY AudioMaster.info df1: (Original-Diskette im Laufwerk 0, Kopie in Laufwerk 1).
(Toni Schwaiger/Christian Wolf/M. Greulich/M. Jobst/rs)

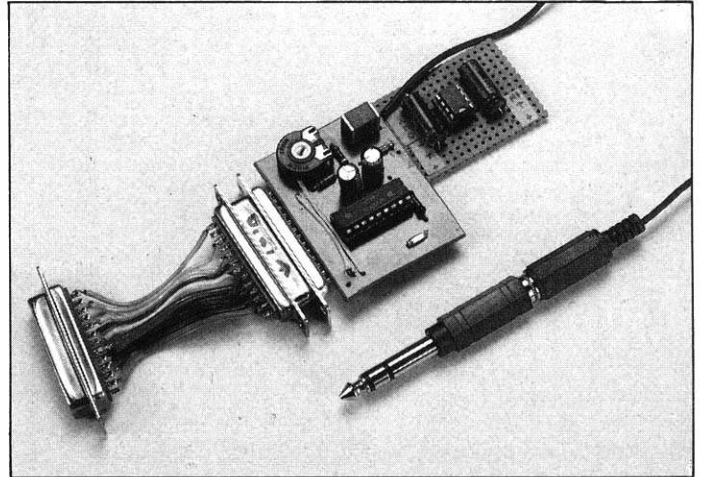


Bild 7. Der Digitizer mit Erweiterungsplatine und Adapterkabel für Amiga 500/2000

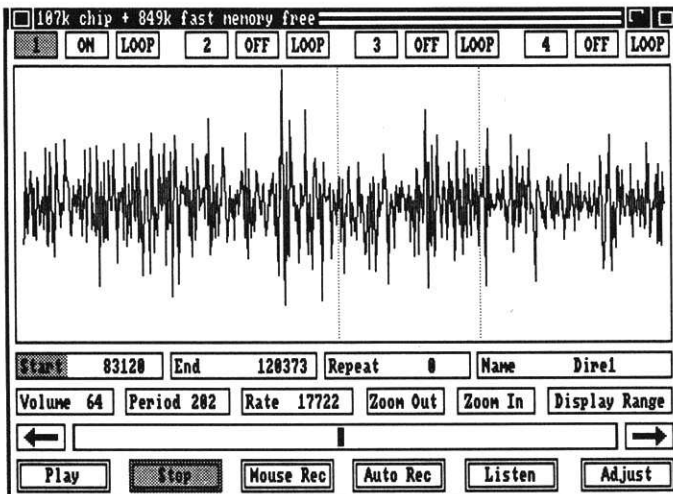


Bild 8. Die Benutzeroberfläche von »Digisoft plus«...

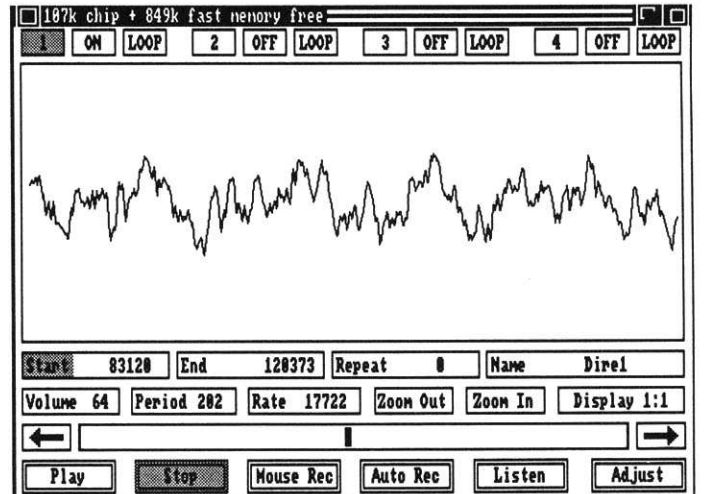


Bild 9. ...macht das Editieren der Klänge zum Kinderspiel

Public Domain
ab 2,75^{DM}

Garantie!

Wir kopieren nur auf geprüften Marken & Qualitäts-Disketten.
2DD, 135 Tpi, 100% Errorfree.

SONDERAKTION!

30 Disketten Ihrer Wahl und 2 **Gratis Disk** — mit Viruskiller, Conman, Df.c.AssigDev, Funckey, PopCli.RunBack, Turbo Backup, Virus X, Zaploon M.RBackup usw.
für 145,- DM inkl. Porto (V-Scheck/bar)

Ray-Tracing-Construction & Grafik-Paket

10 Disketten mit DBW-Render C-Light, Ray Tracing-Cons.-Set mit deutscher Anleitung, m. CAD, DTP-Grafik, Font-Edit, Fonts und vieles mehr.

für 45,- DM inkl. Porto (V-Scheck/bar)

VIRUSKILLER GRATIS

Gratis zu unseren Katalog-Disketten bekommen Sie einen Super-Viruskiller.

3 Katalog-Disketten

mit Information über den Inhalt der PD-Programme.

+ Viruskiller + CLI-Wizard
gegen 10,- DM (Bar/V-Scheck)

Fachhandel für Public Domain & Shareware

Telefon 0203/376448, Fax. 0203/359690
SCT-Datentechnik, U. Schmielewski
Postfach 10 12 64, D-4100 Duisburg

Programmname:	myincludes.h
Computer:	Amiga 500, 1000, 2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec C V3.6
Aufrufe:	cc +l -s +hdm myincludes.h
Bemerkung:	Dieses Listing muß unbedingt vor der Compilierung der Listings 4 bis 11 compiliert werden.

Programm : myincludes.h

```

1 4y0 #include <exec/types.h>
2 bQ #include <clib/macros.h>
3 Rg #include <intuition/intuition.h>
4 L9 #include <intuition/intuitionbase.h>
5 xc #include <exec/exec.h>
6 r5 #include <exec/execbase.h>
7 TZ #include <hardware/custom.h>
8 29 #include <hardware/dmabits.h>
9 Ub #include <hardware/intbits.h>
10 8v #include <libraries/dos.h>
11 42 #include <libraries/dosextens.h>
12 LW #include <stdio.h>
13 Gv #include <exec/interrupts.h>
14 pH #include <exec/nodes.h>
15 rE #include <graphics/gfxmacros.h>
16 WA #include <graphics/sprite.h>

```

(C) 1987 M&T

**Listing 3. »myincludes.h«. Bitte mit dem Checksummer
(Seite 159) eingeben.**

Programmname:	main.c
Computer:	Amiga 500, 1000, 2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec C V3.6
Aufrufe:	cc +l -s +idmp main.c
Linkeraufruf:	ln -o digitizer myfunctions.o effects.o request.o memory.o main.o mydir.o record.o displaytrack.o myints.o -lc32
Bemerkung:	Vor dem Linkeraufruf müssen die Listings 4 bis 11 compiliert und Li- sting 12 assembliert werden

Programm : main.c

```

1 hS0 /*****
2 sK /* Amiga Digitizer Software plus main */
3 uJ /* written by: Christian Wolf */
4 GT /* Matthias-Claudius-Weg 58 */
5 QE /* 2190 Cuxhaven 1 */
6 QI /* 04721/24269 */
7 nY /*****
8 ZM #define MakeID(a,b,c,d) ( (long)(a)<<24 | (long)(b)<<16
    | (c)<<8 | (d) )
9 sa #define PLAY 1
10 IL #define RECORD 2
11 6S #define STOP 3
12 nZ #define ADJUST 4
13 yD #define AUTOREC 5
14 Lr #define TEST 6
15 AS #define CONTINUE 11
16 RC #define SELECT 12
17 tJ #define DEZIMAL 13
18 97 #define NAME 14
19 ns #define STATUS1 15
20 y0 #define STATUS2 16
21 98 #define STATUS3 17
22 KG #define STATUS4 18
23 2W #define DRAW 19
24 2p #define START 20

```

**Listing 4. »main.c«. Das Hauptprogramm unserer
Sampling-Software »Digisoft plus«.**

```

25 w1 #define END 21
26 G5 #define PERIOD 22
27 p3 #define RATE 23
28 BZ #define VOLUME 24
29 PL #define ONESHOT 25
30 CN #define ONEONE 26
31 GN #define ALL 27
32 by #define ZOOMOUT 28
33 OX #define ZOOMIN 29
34 Bm #define SAMPLING 4
35 Sq #define CANCEL FALSE
36 Xg #define OK TRUE
37 LJ #define MODUS1 38
38 UU #define MODUS2 39
39 B4 #define MODUS3 40
40 KF #define MODUS4 41
41 PD #define MYPOTI 46
42 vs #define ON TRUE
43 bI #define OFF FALSE
44 LW #define RIGHT 61
45 WX #define LEFT 62
46 pg #define SELECT1 69
47 P4 #define SELECT2 70
48 aC #define SELECT3 71
49 lK #define SELECT4 72
50 g0 #define LOOP TRUE
51 vX #define ONCE FALSE
52 30 struct NewWindow NewWindow = {
53 pD7 0,0,640,234,0,1,
54 Sg CLOSEWINDOW MENUPICKI GADGETUP! GADGETDOWN,
55 zz WINDOWCLOSE! ACTIVATE! WINDOWDEPTH! WINDOWDRAG,
56 6a NULL,NULL,
57 hH "Amiga Sound-Digitizer Plus (C) 1988 by Markt & Te
    chnik",
58 8c NULL,NULL,
59 dX 640,200,640,200,
60 26 WBENCHSCREEN };
61 RG0 UBYTE coordinates[609];
62 25 USHORT LinesAll[] = { 0,0,117,0,117,12,0,12,0,0 };
63 UN USHORT Lines1[] = { 0,0,87,0,87,15,0,15,0,0 };
64 j1 USHORT Lines2[] = { 3,2,84,2,84,13,3,13,3,2 };
65 Da USHORT LinesReq1[] = { 0,0,0,0,0,39,0,39,0,0 };
66 7v USHORT LinesReq2[] = { 3,2,3,2,2,37,3,37,3,2 };
67 By USHORT LinesTrack1[] = { 0,0,23,0,23,15,0,15,0,0 };
68 I9 USHORT LinesStatus[] = { 0,0,40,0,40,12,0,12,0,0 };
69 Ex USHORT LinesStart[] = { 0,0,138,0,138,12,0,12,0,0 };
70 qc USHORT LinesDraw[] = { 0,0,621,0,621,131,0,131,0,0 };
71 y8 USHORT LinesVolume[] = { 0,0,91,0,91,12,0,12,0,0 };
72 T2 USHORT LinesPeriod[] = { 0,0,98,0,98,12,0,12,0,0 };
73 ag USHORT LinesTrackName[] = { 0,0,184,0,184,12,0,12,0,0 };
74 ac USHORT LinesLeft[] = { 0,0,45,0,45,12,0,12,0,0 };
75 D0 USHORT LinesRate[] = { 0,0,105,0,105,12,0,12,0,0 };
76 2n USHORT LinesZoom[] = { 0,0,73,0,73,12,0,12,0,0 };
77 bP USHORT Sleepy[54] = { 0x0000,0x0000,0xffff,0x0000,0xffff,0x
    0000,
78 LoM 0x4004,0x0000,0x4004,0x0000,0x600c,0x
    0000,
79 Nu 0x600c,0x0000,0x5014,0x0000,0x5014,0x
    0000,
80 Lk 0x4c64,0x0440,0x4c64,0x0440,0x46c4,0x
    0280,
81 41 0x47c4,0x0380,0x4384,0x0100,0x4284,0x
    0000,
82 UI 0x4444,0x0000,0x4444,0x0000,0x4924,0x
    0100,
83 nU 0x4824,0x0000,0x5114,0x0100,0x5014,0x
    0000,
84 tg 0x610c,0x0100,0x638c,0x0380,0x4fe4,0x
    0fe0,
85 XY 0x5ffe,0x1fff,0xffff,0x0000,0xffff,0x
    0000 };
86 ay0 USHORT LeftData[18] = { 0x00c0,0x0000,0x03c0,0x0000,0x0fc0,
    0x0000,
87 lr0 0x3fff,0xffff,0xffff,0xffff,0x3fff,
    0xffff,
88 iL 0x0fc0,0x0000,0x03c0,0x0000,0x00c0,
    0x0000 };
89 Vq0 USHORT RightData[18] = { 0x0000,0x0300,0x0000,0x03c0,0x0000
    ,0x03f0,
90 S6P 0xffff,0xfffc,0xffff,0xffff,0xffff
    ,0xfffc,
91 l1 0x0000,0x03f0,0x0000,0x03c0,0x0000
    ,0x0300 };
92 T80 struct Image LeftImage = { 15,199,32,9,1,NULL,1,0,NULL };

```



```

93 8T struct Image RightImage = { 592,199,32,9,1,NULL,1,0,NULL };
94 Ed USHORT *ChipSleepy,*c1,*c2,*ChipLeft,*ChipRight;
95 vk struct Border BorderZoom = { 0,0,1,3,JAM2,5,&LinesZoom,NULL
};
96 ru struct Border Border1 = { 0,0,1,3,JAM1,5,&Lines1,NULL };
97 Vb struct Border Border2 = { 0,0,1,3,JAM1,5,&Lines2,&Border1 }
;
98 Ed struct Border BorderReq1 = { 0,0,3,3,JAM2,5,&LinesReq1,NULL
};
99 L1 struct Border BorderReq2 = { 0,0,3,3,JAM2,5,&LinesReq2,&Bor
derReq1 };
100 2a struct Border BorderStatus1 = { 0,0,1,3,JAM2,5,&LinesStatus
,NULL };
101 3r struct Border BorderRate = { -53,-3,1,3,JAM2,5,&LinesRate,N
ULL };
102 9p struct Border BorderStart = { -74,-3,1,3,JAM2,5,&LinesStart
,NULL };
103 jT struct Border BorderModus = { 0,3,1,3,JAM2,5,&LinesStart,NU
LL };
104 86 struct Border BorderLeft = { 0,0,1,3,JAM1,5,&LinesLeft,NULL
};
105 ob struct Border BorderTrackName = { -44,-3,1,3,JAM2,5,&LinesT
rackName,NULL };
106 iE struct Border BorderDraw = { -8,-2,1,3,JAM2,5,&LinesDraw,NU
LL };
107 Nq struct Border BorderVolume = { -66,-3,1,3,JAM2,5,&LinesVolu
me,NULL };
108 sH struct Border BorderPeriod = { -62,-3,1,3,JAM2,5,&LinesPeri
od,NULL };
109 sI struct Border BorderAll = { 0,0,1,3,JAM2,5,&LinesAll,NULL }
;
110 u1 struct Image MyPotiImage;
111 zg struct IntuiText TextZoomOut = { 1,0,JAM2,3,3,NULL,"Zoom Out
",NULL };
112 8g struct IntuiText TextZoomIn = { 1,0,JAM2,7,3,NULL,"Zoom In",
NULL };
113 uM struct IntuiText TextAll = { 1,0,JAM2,7,3,NULL,"Display Rang
e",NULL };
114 Xh struct IntuiText TextOne_One = { 1,0,JAM2,15,3,NULL,"Display
1:1",NULL };
115 VQ struct IntuiText TextTest = { 1,0,JAM2,19,4,NULL,"Listen",NU
LL };
116 MH struct IntuiText TextIFF = { 0,1,JAM2,2,3,NULL,"IFF 8SVX Fil
e",NULL };
117 5n struct IntuiText TextFSound = { 0,1,JAM2,2,3,NULL,"Future So
und File",NULL };
118 4e struct IntuiText TextData = { 0,1,JAM2,2,3,NULL,"Data File",
NULL };
119 l1 struct IntuiText TextInfo = { 0,1,JAM2,2,3,NULL,"About...",N
ULL };
120 G9 struct IntuiText TextLoad = { 0,1,JAM2,2,3,NULL,"Load file",
NULL };
121 yq struct IntuiText TextSave = { 0,1,JAM2,2,3,NULL,"Save...",NU
LL };
122 f1 struct IntuiText TextRead = { 0,1,JAM2,2,3,NULL,"Read from d
isk",NULL };
123 H5 struct IntuiText TextWrite = { 0,1,JAM2,2,3,NULL,"Write to d
isk",NULL };
124 uE struct IntuiText TextDelete = { 0,1,JAM2,2,3,NULL,"Delete fi
le",NULL };
125 za struct IntuiText TextRecord = { 1,0,JAM2,9,4,NULL,"Mouse Rec
",NULL };
126 Oa struct IntuiText TextPlay = { 1,0,JAM2,27,4,NULL,"Play",NULL
};
127 aA struct IntuiText TextStop = { 1,0,JAM2,27,4,NULL,"Stop",NULL
};
128 On struct IntuiText TextAdvance = { 1,0,JAM2,19,4,NULL,"Adjust"
,NULL };
129 yC struct IntuiText TextRewind = { 1,0,JAM2,11,4,NULL,"Auto Rec
",NULL };
130 JK struct IntuiText TextAllocate = { 0,1,JAM2,2,3,NULL,"Allocat
e Track",NULL };
131 Zu struct IntuiText TextClear = { 0,1,JAM2,2,3,NULL,"Clear Trac
k",NULL };
132 Eu struct IntuiText TextCopy = { 0,1,JAM2,2,3,NULL,"Copy Range"
,NULL };
133 LX struct IntuiText TextReverse = { 0,1,JAM2,2,3,NULL,"Reverse
Range",NULL };
134 Fz struct IntuiText TextDouble = { 0,1,JAM2,2,3,NULL,"Period *
2",NULL };
135 4j struct IntuiText TextHalve = { 0,1,JAM2,2,3,NULL,"Period / 2
",NULL };
136 us struct IntuiText TextClearAll = { 0,1,JAM2,2,3,NULL,"Clear a

```

```

11 tracks",NULL };
137 9c struct IntuiText TextMix = { 0,1,JAM2,2,3,NULL,"Mix Range",N
ULL };
138 GF struct IntuiText TextConcat = { 0,1,JAM2,2,3,NULL,"Concat Ra
nge",NULL };
139 cR struct IntuiText TextPaste = { 0,1,JAM2,2,3,NULL,"Paste Rang
e",NULL };
140 Uw struct IntuiText TextZero = { 0,1,JAM2,2,3,NULL,"Zero Range"
,NULL };
141 7q struct IntuiText TextCut = { 0,1,JAM2,2,3,NULL,"Cut Range",N
ULL };
142 yM struct IntuiText TextInsert = { 0,1,JAM2,2,3,NULL,"Insert Ra
nge",NULL };
143 CK struct IntuiText TextDisplay = { 0,1,JAM2,2,3,NULL,"Display
track",NULL };
144 xf struct IntuiText TextReset = { 0,1,JAM2,2,3,NULL,"Reset Trac
k",NULL };
145 Od struct IntuiText TextStart = { 1,0,JAM2,3,2,NULL,"Start",NUL
L };
146 xd struct IntuiText TextEnd = { 1,0,JAM2,3,2,NULL,"End",NULL };
147 3S struct IntuiText TextPeriod = { 1,0,JAM2,3,2,NULL,"Period",N
ULL };
148 4X struct IntuiText TextRate = { 1,0,JAM1,3,2,NULL,"Rate",NULL
};
149 m3 struct IntuiText TextVolume = { 1,0,JAM2,3,2,NULL,"Volume",N
ULL };
150 Ra struct IntuiText TextModus = { 1,0,JAM1,28,59,NULL,"Mode",NU
LL };
151 fy struct IntuiText TextTrackName = { 1,0,JAM1,-39,0,NULL,"Nam
e",NULL };
152 np struct IntuiText TextOneShot = { 1,0,JAM1,3,2,NULL,"Repeat",
NULL };
153 Ge struct IntuiText TextCurrent[4] = { 1,0,JAM2,17,3,NULL,"1",N
ULL,
1,0,JAM2,17,3,NULL,"2",N
ULL,
1,0,JAM2,17,3,NULL,"3",N
ULL,
1,0,JAM2,17,3,NULL,"4",N
ULL };
154 7rZ
155 Av
156 Yt
157 eVO struct IntuiText TextOn = { 1,0,JAM2,0,0,NULL,"ON",NULL };
158 10 struct IntuiText TextOff = { 1,0,JAM2,0,0,NULL,"OFF",NULL };
159 KK struct IntuiText TextLoop = { 1,0,JAM2,5,3,NULL,"LOOP",NULL
};
160 cu struct IntuiText TextOnce = { 1,0,JAM2,5,3,NULL,"ONCE",NULL
};
161 P5 struct IntuiText TextProbe = { 1,0,JAM2,0,0,NULL,"000000",NU
LL };
162 Fj struct IntuiText TextPrefer = { 0,1,JAM2,2,3,NULL,"Prefer",N
ULL };
163 of struct IntuiText TextChip = { 0,1,JAM2,2,3,NULL,"Chip",NULL
};
164 nD struct IntuiText TextFast = { 0,1,JAM2,2,3,NULL,"Fast",NULL
};
165 Sy struct IntuiText TextHardware = { 0,1,JAM2,2,3,NULL,"Hardwar
e",NULL };
166 an struct IntuiText TextDig3 = { 0,1,JAM2,2,3,NULL," Future Sou
nd",NULL };
167 cf struct IntuiText TextDig2 = { 0,1,JAM2,2,3,NULL," Perfect So
und",NULL };
168 Nm struct IntuiText TextDig1 = { 0,1,JAM2,20,3,NULL,"68000er mo
difid",NULL };
169 Ok struct IntuiText TextDig0 = { 0,1,JAM2,20,3,NULL,"68000er or
iginal",NULL };
170 EF struct MenuItem MenuIFF = { 0,40,0,144,13,ITEMTEXT HIGHCOMP
ITEMENABLED,
0,&TextIFF,NULL,0,NULL,0 };
171 wuS
172 KYO struct MenuItem MenuFSound = { &MenuIFF,40,13,144,13,ITEMTE
XT HIGHCOMPI ITEMENABLED,
0,&TextFSound,NULL,0,NULL,0 };
173 tCS
174 Q60 struct MenuItem MenuData = { &MenuFSound,40,26,144,13,ITEMT
EXT HIGHCOMPI ITEMENABLED,
0,&TextData,NULL,0,NULL,0 };
175 qWS
176 4D0 struct MenuItem MenuInfo = { NULL,0,39,160,13,
ITEMTEXT HIGHCOMPI ITEMENABLED,
COMMSEQ,
0,&TextInfo,NULL,'I',NULL,0 };
177 nMT
178 1S
179 U00 struct MenuItem MenuDelete = { &MenuInfo,0,26,160,13,

```

Listing 4. (Fortsetzung)


```

180 8AV      ITEMTEXT HIGHCOMPI COMMSEQ I
181 wH      EMENABLED,
           0,&TextDelete,NULL,'D',NULL,
           0 };
182 4d0 struct MenuItem MenuSave = {
183 wBT      &MenuDelete,0,13,160,13,
           ITEMTEXT HIGHCOMPI ITEMENABLED,
           0,&TextSave,NULL,0,&MenuData,0
184 FO      };
185 rY0 struct MenuItem MenuLoad = {
186 wVT      &MenuSave,0,0,160,13,
           ITEMTEXT HIGHCOMPI ITEMENABLED I
           COMMSEQ,
187 qX      0,&TextLoad,NULL,'L',NULL,0 };
188 qA0 struct MenuItem MenuHalve = {
189 HJU      NULL,0,39,181,13,
           ITEMTEXT HIGHCOMPI COMMSEQ I TE
           MENABLED,
           0,&TextHalve,NULL,'/',NULL,0
190 mI      };
191 5s0 struct MenuItem MenuDouble = {
192 KMS      &MenuHalve,0,26,181,13,
           ITEMTEXT HIGHCOMPI COMMSEQ I TE
           MABLED,
           0,&TextDouble,NULL,'8',NULL,0 }
193 wU      };
194 lq0 struct MenuItem MenuMix = {
195 NPS      &MenuDouble,0,13,181,13,
           ITEMTEXT HIGHCOMPI COMMSEQ I TE
           MABLED,
           0,&TextMix,NULL,'M',NULL,0 };
196 Dc      struct MenuItem MenuReverse = {
197 VHO      &MenuMix,0,0,181,13,
           ITEMTEXT HIGHCOMPI COMMSEQ I
           TEMENABLED,
           0,&TextReverse,NULL,'R',NUL
198 QSW      L,0 };
199 PT      struct MenuItem MenuInsert = {
200 FCO      NULL,0,39,152,13,ITEMTEXT HI
           ITEMENABLED,0,&TextInsert,NU
201 6eV      LL,0,NULL,0 };
202 cJ0 struct MenuItem MenuPaste = {
203 u5S      &MenuInsert,0,26,152,13,ITEMT
           EXT HIGHCOMPI
           ITEMENABLED,0,&TextPaste,NULL,0
204 tw0      ,NULL,0 };
205 sRV      struct MenuItem MenuConcat = {
           &MenuPaste,0,13,152,13,ITEMT
           EXT HIGHCOMPI
           ITEMENABLED,0,&TextConcat,NU
206 NUO      LL,0,NULL,0 };
207 fFt      struct MenuItem MenuCopy = {
           &MenuConcat,0,0,152,13,ITEMTEX
           T HIGHCOMPI
           ITEMENABLED,0,&TextCopy,NULL,0
208 k30      ,NULL,0 };
209 JsU      struct MenuItem MenuReset = {
           NULL,0,26,160,13,
           ITEMTEXT HIGHCOMPI ITEMENABLED
           I COMMSEQ,
210 JJ      0,&TextReset,NULL,'R',NULL,0
           };
211 Sk0 struct MenuItem MenuClearAll = {
212 20X      &MenuReset,0,13,160,13,ITE
           MTEXT HIGHCOMPI
           ITEMENABLED,0,&TextClearAl
213 az0      1,NULL,0,NULL,0 };
214 KCU      struct MenuItem MenuClear = {
           &MenuClearAll,0,0,160,13,ITEM
           TEXT HIGHCOMPI
           ITEMENABLED,0,&TextClear,NULL
215 8Q0      ,0,NULL,0 };
216 9WT      struct MenuItem MenuDig1 = {
           NULL,72,13,152,13,
           CHECKIT ITEMTEXT HIGHCOMPI ITEM
           ENABLED,
217 mU      0xfffffff,0,&TextDig1,NULL,0,NU
           LL,0 };
218 qH0 struct MenuItem MenuDig0 = {
219 BGT      &MenuDig1,72,0,152,13,
           CHECKIT CHECKED ITEMTEXT HIGHC
           OMP ITEMENABLED,
220 pX      0xffffffe,&TextDig0,NULL,0,NU
           LL,0 };
221 nu0 struct MenuItem MenuHardware = {
222 yYX      NULL,0,0,80,13,ITEMTEXT HI
           GHCOMPI
           ITEMENABLED,0,&TextHardwar
           e,NULL,0,&MenuDig0,0 };
223 Ps0 struct Menu MenuOptions = {
224 OK      NULL,304,0,64,8,MENUENABLED,"Op
           tions",&MenuHardware };
225 Y1      struct Menu MenuEffects = {
           &MenuOptions,224,0,64,8,MENUENA
           BLED,"Effects",&MenuReverse };
226 Vu      struct Menu MenuRange = {
           &MenuEffects,152,0,48,8,MENUENABL
           ED,"Range",&MenuCopy };
           struct Menu MenuTrack = {
           &MenuRange,80,0,48,8,MENUENABLED,
           "Track",&MenuClear };

```

```

227 pp      struct Menu MenuProject = {
           &MenuTrack,0,0,64,8,MENUENABLED
           ,"Project",&MenuLoad };
228 RU      struct Gadget GGPlay = {
           NULL,8,214,88,16,GADGHCOMP,RELVERI
           FY,
           BOOLGADGET,&Border2,0,&TextPlay,0,
229 DoP      0,PLAY,0 };
230 U00 struct Gadget GGRecord = {
231 MpP      &GGPlay,220,214,88,16,GADGHCOMP,
           RELVERIFY,
           BOOLGADGET,&Border2,0,&TextRecord,
           0,0,RECORD,0 };
232 TA0 struct Gadget GGStop = {
233 6oP      &GGRecord,114,214,88,16,GADGHCOMP,
           RELVERIFY,
           BOOLGADGET,&Border2,0,&TextStop,0,
           0,STOP,0 };
234 4I0 struct Gadget GGAutoRec = {
235 aIS      &GGStop,326,214,88,16,GADGHCOMP
           ,RELVERIFY,
           BOOLGADGET,&Border2,0,&TextRewi
           nd,0,0,AUTOREC,0 };
236 bX0 struct Gadget GGAdvance = {
237 ONP      &GGAutoRec,543,214,88,16,GADGHC
           OMP,RELVERIFY,
           BOOLGADGET,&Border2,0,&TextAdvance
           ,0,0,ADJUST,0 };
238 ux0 struct Gadget GGTest = {
239 MEP      &GGAdvance,433,214,88,16,GADGHCOMP
           ,RELVERIFY,
           BOOLGADGET,&Border2,0,&TextTest,0,
           0,TEST,0 };
240 NP0 struct Gadget GGStatus1 = {
241 GXU      &GGTest,57,11,41,13,GADGHCOMP,R
           ELVERIFY,
           BOOLGADGET,&BorderStatus1,0,N
           ULL,0,0,STATUS1,0 };
242 ur0 struct Gadget GGStatus2 = {
243 NFU      &GGStatus1,218,11,41,13,GADGHCO
           MP,RELVERIFY,
           BOOLGADGET,&BorderStatus1,0,N
           ULL,0,0,STATUS2,0 };
244 PRO struct Gadget GGStatus3 = {
245 UnU      &GGStatus2,379,11,41,13,GADGHCO
           MP,RELVERIFY,
           BOOLGADGET,&BorderStatus1,0,N
           ULL,0,0,STATUS3,0 };
246 Ey0 struct Gadget GGStatus4 = {
247 bVU      &GGStatus3,540,11,41,13,GADGHCO
           MP,RELVERIFY,
           BOOLGADGET,&BorderStatus1,0,N
           ULL,0,0,STATUS4,0 };
248 uh0 char BufferEnd[8] = "empty",BufferPeriod[4] = "358",BufferS
           tart[8] = "begin";
249 I9      char BufferVolume[3] = "64",BufferRate[6] = "10000",BufferO
           neShot[8] = "0";
250 OD      char Not_Allocated[] = "Track 1 is not allocated";
251 QI      char Out_of_Memory[] = "Sorry there's no more memory";
252 u0      char BufferTrackName[4][60] = { "untitled","untitled","unti
           tled","untitled" };
253 h4      char MyTitle[40];
254 Tb      struct StringInfo EndInfo = { BufferEnd,0,0,8,0,0,0,10,27
           ,0,0,0 };
255 uq      struct StringInfo PeriodInfo = { BufferPeriod,0,0,4,0,0,0,0
           ,10,27,0,0,0 };
256 Sd      struct StringInfo StartInfo = { BufferStart,0,0,8,0,0,1,6,1
           0,27,0,0,0 };
257 OH      struct StringInfo RateInfo = { BufferRate,0,0,6,0,0,5,4,10,
           27,0,0,0 };
258 Le      struct StringInfo VolumeInfo = { BufferVolume,0,0,3,0,0,3,2
           ,10,27,0,0,0 };
259 OI      struct StringInfo OneShotInfo = { BufferOneShot,0,0,8,0,0,3
           ,2,10,27,0,0,0 };
260 qI      struct StringInfo TrackNameInfo = { BufferTrackName[0],0,0,
           49,0,0,0,0,10,27,0,0,0 };
261 mv      struct PropInfo MyPotInfo;
262 rL      struct Gadget GGDrawHand = {
           &GGStatus4,16,29,608,128,GADGH
           NONE,
           RELVERIFY GADGIMMEDIATE,BOOLGA
           DGET,
263 84T      &BorderDraw,0,NULL,0,NULL,DRAW
           ,0 };
264 MU      struct Gadget GGZoomIn = {
           &GGDrawHand,426,180,74,13,GADGHC
           OMP,RELVERIFY,
           BOOLGADGET,&BorderZoom,0,&TextZo
           omIn,
265 wo0      0,0,ZOOMIN,0 };
266 dqR      struct Gadget GGAll = {
           &GGZoomIn,512,180,118,13,GADGHCOMP,
           GADGIMMEDIATE RELVERIFY,
           BOOLGADGET,&BorderAll,0,&TextAll,0,
267 oI      0,ALL,0 };
268 tCO      struct Gadget GGStart = {
           &GGAll,82,166,64,11,GADGHCOMP,
           RELVERIFY STRINGCENTER LONGINT GA
269 B30      270 R7
271 qv0      272 DzQ

```



```

273 Lr          DGIMMEDIATE,STRGADGET,
                &BorderStart,0,NULL,0,&StartInfo,
                START,0};
274 go0 struct Gadget GGSelectStart = { &GGStart,9,164,48,11,GADGHC
OMP,
275 bsW          GADGIMMEDIATE,BOOLGADGET,
276 RF          NULL,0,&TextStart,0,0,START
                ,0};
277 sJ0 struct Gadget GGEnd = { &GGSelectStart,228,166,64,11,GADGHC
OMP,
278 J50          RELVERIFY1 STRINGCENTER1 LONGINT1 GADG
IMMEDIATE,STRGADGET,
279 C1          &BorderStart,0,NULL,0,&EndInfo,END,
                0};
280 Ae0 struct Gadget GGSelectEnd = { &GGEnd,155,164,33,11,GADGHCOM
P,GADGIMMEDIATE,
281 68W          BOOLGADGET,NULL,0,&TextEnd,
                0,0,END,0};
282 07
283 230 struct Gadget GGPeriod = { &GGSelectEnd,173,183,32,11,GADGH
COMP,
284 UXR          RELVERIFY1 STRINGRIGHT1 LONGINT1 GA
DGIMMEDIATE,STRGADGET,
285 oM          &BorderPeriod,0,NULL,0,&PeriodIn
fo,PERIOD,0};
286 lW0 struct Gadget GGSelectPeriod = { &GGPeriod,112,181,55,11,GA
DGHCOMP,
287 mLX          GADGIMMEDIATE,BOOLGADGET,N
ULL,0,&TextPeriod,
288 Gx          0,0,PERIOD,0};
289 c70 struct Gadget GGRate = { &GGSelectPeriod,274,183,48,11,GADG
HCOMP,
290 adP          RELVERIFY1 STRINGRIGHT1 LONGINT1 GADG
IMMEDIATE,STRGADGET,
291 v2          &BorderRate,0,NULL,0,&RateInfo,RAT
E,0};

```

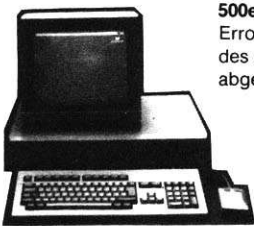
```

292 Gt0 struct Gadget GGSelectRate = { &GGRate,222,181,39,11,GADGHC
OMP,
293 97V          GADGIMMEDIATE,BOOLGADGET,NUL
L,0,&TextRate,
294 P9          0,0,RATE,0};
295 R00 struct Gadget GGVolume = { &GGSelectRate,74,183,24,11,GADGH
COMP,
296 gJR          RELVERIFY1 STRINGRIGHT1 LONGINT1 GA
DGIMMEDIATE,STRGADGET,
297 02          &BorderVolume,0,NULL,0,&VolumeIn
fo,VOLUME,0};
298 9T0 struct Gadget GGSelectVolume = { &GGVolume,9,181,54,11,GADG
HCOMP,
299 wKX          GADGIMMEDIATE,BOOLGADGET,N
ULL,0,&TextVolume,
300 i0          0,0,VOLUME,0};
301 7f0 struct Gadget GGZoomOut = { &GGSelectVolume,340,180,74,13,G
ADGHCOMP,
302 JZS          RELVERIFY,BOOLGADGET,
303 4P          &BorderZoom,0,&TextZoomOut,0,NU
LL,ZOOMOUT,0};
304 cd0 struct Gadget GGModus1 = { &GGZoomOut,105,11,41,13,GADGHCOM
P,
305 McR          RELVERIFY,BOOLGADGET,
306 p6          &BorderStatus1,0,&TextLoop,0,NUL
L,MODUS1,0};
307 yM0 struct Gadget GGModus2 = { &GGModus1,266,11,41,13,GADGHCOMP
,
308 PfR          RELVERIFY,BOOLGADGET,
309 yB          &BorderStatus1,0,&TextLoop,0,NUL
L,MODUS2,0};
310 OK0 struct Gadget GGModus3 = { &GGModus2,427,11,41,13,GADGHCOMP
,
311 S1R          RELVERIFY,BOOLGADGET,

```

Listing 4. (Fortsetzung)

Comptec Aktuell



500er-Tuning

Errow-Gehäuse Bausatz zum Umbau des A 500 in ein PC-Metallgehäuse mit abgesetzter Tastatur und Aufnahme von 2 x 3,5" Laufwerken, internes Netzteil, 5,25" Festplatte, sowie 4 x A 2000 Steckplätze, davon 3 als AT/XT Amiga möglich. Gehäusebausatz

ab 398,- DM

Gehäusekombinationen:

Gehäuse + A 500 eingebaut	1288,- DM
Gehäuse + A 500 jedoch mit 2. Laufwerk	1492,- DM
Gehäuse + 20 MB < 65 ms	1185,- DM
Gehäuse + 40 MB < 40 ms	1480,- DM
Gehäuse + A 2000 Adapterplatine	698,- DM

Laufwerke:

3,5" intern, für Gehäuse mit Interface	222,- DM
3,5" extern, Metallgehäuse, abschaltbar, Bus	255,- DM
3,5" extern, wie oben, jedoch NEC 1037	278,- DM
5,25" extern, Metallgehäuse, 40/80 Track, abschaltbar	336,- DM
5,25" extern, jedoch TEAC 55	366,- DM
20 MB externe Festplatte für A 500	978,- DM

PD; Speichererweiterungen;
weitere Infos sind gegen
Freiumschlag erhältlich.
Lieferung gegen Nachnahme
oder Vorauskasse

Comptec Versand
+ Ladenverkauf

Obermörnterstraße 1 - nahe Reeser Rheinbrücke
4192 Kalkar 4 (Niedermörnter)
Tel.: 0 28 24 / 38 67

NEUE AMIGA SOFTWARE !

SPRITE - ANIMATOR

Dieser neue Sprite-Editor weist bisher noch nie dagewesene Eigenschaften auf. Mit Hilfe dieses Sprite-Editors können Sie animierte Sprites mit bis zu 16 Bewegungsphasen kinderleicht erstellen. Weiterhin stellt dieser Sprite-Editor umfangreiche und komfortable Funktionen zum Erstellen und Animieren von Sprites zu Verfügung.

Die daraus entstehenden Sprites können im Assembler-, C-, Basic- oder IFF-Format abgespeichert werden. Durch Programmierung in Assembler ist der Sprite-Editor auch in seiner Verarbeitungsgeschwindigkeit bisher unübertroffen. Weiterhin wird die volle Pal-Auflösung des Bildschirm genutzt. Diesen Sprite-Editor sollte jeder haben, der mit Sprites zu tun hat.

Am besten gleich bestellen:

SPRITE ANIMATOR Best.-Nr. A 07 001 88

Preis: 109,- DM zuzügl. 5,- DM Versandkosten
Versand per Nachnahme oder gegen V-Scheck!

Gegen 1,30 in Briefmarken erhalten Sie unsere Info-
blätter über unser derzeitiges Angebot an Amiga-Soft-
ware. Bestellungen unter:

Software 2000, Lange Straße 19, 2320 PLÖN
Telefon: 04522 / 13 79

HÄNDLERANFRAGEN ERWÜNSCHT !


```

312 7G      &BorderStatus1,0,&TextLoop,0,NUL
          L,MODUS3,0;
313 mBO struct Gadget GGModus4 = {
          &GGModus3,588,11,41,13,GADGHCOMP
          ,
314 vLR      RELVERIFY,BOOLGADGET,
          &BorderStatus1,0,&TextLoop,0,&Te
315 y5      xtLoop,MODUS4,0;
316 EeO struct Gadget GGMYPoti = {
          &GGModus4,61,197,517,13,GADGHCOM
          P,
317 irR      RELVERIFY GADGIMMEDIATE,PROPGADG
          ET,
318 zn      &MyPotiImage,0,NULL,0,&MyPotiInf
          o,MYPOTI,0;
319 3yO struct Gadget GGOOneShot = {
          &GGMYPoti,374,166,64,11,GADGHCO
          MP,
320 zLS      RELVERIFY STRINGCENTER LONGINT
          GADGIMMEDIATE,STRGADGET,
321 Hk      &BorderStart,0,NULL,0,&OneShotI
          nfo,ONESHOT,0;
322 S9O struct Gadget GGSelectOneShot = {
          &GGOOneShot,301,164,56,11,
          GADGHCOMP,
323 42Y      GADGIMMEDIATE,BOOLGADGET,
          NULL,0,
324 M4      &TextOneShot,0,0,ONESHOT,
          0;
325 L5O struct Gadget GGTrackName = {
          &GGSelectOneShot,489,166,140,
          11,GADGHCOMP,
326 XGU      RELVERIFY STRINGCENTER,STRGAD
          GET,
327 3S      &BorderTrackName,0,&TextTrack
          Name,
328 OH      0,&TrackNameInfo,NULL,0;
329 GGO struct Gadget GGLeft = {
          &GGTrackName,8,197,46,13,GADGHCOMP
          ,
330 D9P      RELVERIFY GADGIMMEDIATE,BOOLGADGET
          ,
331 xp      &BorderLeft,0,NULL,0,NULL,LEFT,0;
332 LZO struct Gadget GGRight = {
          &GGLeft,585,197,46,13,GADGHCOMP,
333 GCQ      RELVERIFY GADGIMMEDIATE,BOOLGADGE
          T,
334 X1      &BorderLeft,0,NULL,0,NULL,RIGHT,0
          };
335 tDO struct Gadget GGSelect1 = {
          &GGRight,9,11,41,13,GADGHCOMP,R
          ELVERIFY,
336 BXS      BOOLGADGET,&BorderStatus1,0,&Te
          xtCurrent[0],
337 4x      0,0,SELECT1,0;
338 ZHO struct Gadget GGSelect2 = {
          &GGSelect1,170,11,41,13,GADGHCO
          MP,RELVERIFY,
339 HeS      BOOLGADGET,&BorderStatus1,0,&Te
          xtCurrent[1],
340 93      0,0,SELECT2,0;
341 KQO struct Gadget GGSelect3 = {
          &GGSelect2,331,11,41,13,GADGHCO
          MP,RELVERIFY,
342 NLS      BOOLGADGET,&BorderStatus1,0,&Te
          xtCurrent[2],
343 E9      0,0,SELECT3,0;
344 q1O struct Gadget GGSelect4 = {
          &GGSelect3,492,11,41,13,GADGHCO
          MP,RELVERIFY,
345 TsS      BOOLGADGET,&BorderStatus1,0,&Te
          xtCurrent[3],
          0,0,SELECT4,0;
346 JF      BYTE CharacterBuffer[108];
347 hOO struct Gadget *GadgetTable[6]=
          {
349 S14      &GGSelectStart,&GGSelectEnd,&GGSelectPeriod,
          &GGSelectRate,&GGSelectVolume,&GGSelectOneShot;
350 j86 struct Window *OpenWindow(),*MyWindow;
352 9c struct IntuitionBase *IntuitionBase;
353 wQ struct IntuiMessage *MyMessage,*GetMsg();
354 AZ struct Interrupt *OldInterrupt,*SetIntVector(),*Paula;
355 V8 struct GfxBase *GfxBase;
356 XL struct Gadget *Pre_Gadget;
357 gS struct SimpleSprite MySprite;
358 e1 USHORT *MySpriteData;
359 4q UWORD SpriteGot;
360 fO USHORT code,ItemNum,MenuNum,SubNum,ID,Number;
361 do APTR msgaddress;
362 Sn ULONG NewPosition,OldPosition,RangeStart[4],RangeEnd[4],re
          gstart,regend,posbackup;
363 s1 ULONG oneshot[4],startpos[4],class,ChunkID,ChunkLength,cou
          ntpos;
364 sd UBYTE i,Current=0,track;
365 n2 UWORD SpritePosition,sample,rest,x1,y1;

```

```

366 HD UBYTE hi,current,currenty[4];
367 6k UWORD rate,CurrentInt[4],IntCount[4];
368 nZ UWORD mask[4] = { 1,2,4,8 };
369 7s UWORD AudioBits=0;
370 Ip BOOL Skip=FALSE,Resident=TRUE,DisplayModus=FALSE,PLAYING,
          Modus[4];
371 A7 ULONG start[4],end[4],length[4],allocated[4],MemoryType[4]
          ,MinLength;
372 tY ULONG volume[4],period[4],samplingrate[4],pre_period[4],pr
          e_oneshot[4];
373 Yq ULONG ChipStart,ChipEnd,ChipPointer;
374 ku ULONG FastStart,FastEnd,FastPointer;
375 ys APTR DirBuffer;
376 WN ULONG Min,Max,Null1,Null2;
377 BU ULONG *parameter[6],*parapointer;
378 EI ULONG PotiStart[6] = { 0,0,124,3583,0,0 };
379 ad ULONG MaxiPart[6] = { 1,1,875,25287,64,1 };
380 cv ULONG Step[6] = { 2,2,1,31,1,2 };
381 hi LONG MyFile,lock;
382 MS FILE *MYFILE;
383 8d struct FileInfoBlock *FileInfoBlock;
384 NO struct RastPort *RastPort;
385 z1 char BufferFileName[108];
386 N1 extern struct ExecBase *SysBase;
387 KD struct FastIntVariables
          {
388 d57     ULONG start;
389 8b9     ULONG MemCounter;
390 GQ     ULONG end;
391 8m     APTR CopyBuffer1;
392 Et     APTR CopyBuffer2;
393 Sr     APTR RestBuffer;
394 rm     UWORD RestLength;
395 7U     BOOL *Modus;
396 4x     APTR Next; };
397 boO extern struct FastIntVariables FastIntVariables[4],RepeatVa
          riables[4];
398 O6 struct VHDR {
          ULONG Length;
399 OVE     ULONG OneShotHiSamples;
400 GI     ULONG RepeatPart;
401 Ke     ULONG SamplesPerHiCycle;
402 sq     UWORD SamplesPerSecond;
403 cf     UBYTE Octaves;
404 1F     UBYTE Packed;
405 O3     ULONG Volume; };
406 P6O ULONG FORM_ID,_8SVX_ID,VHDR_ID,BODY_ID;
407 ff struct FSound {
          ULONG length;
408 RIG     UWORD samplingrate; };
409 zeO struct BODY {
          ULONG BODY_ID;
410 WxE     ULONG Length; };
411 KPO struct IFF {
          ULONG FORM_ID;
412 ggD     ULONG Length;
413 yg     ULONG _8SVX_ID;
414 Ub     ULONG VHDR_ID;
415 hY     struct VHDR VHDR;
416 pF     struct BODY BODY; };
417 I6O struct FSound FSound;
418 kb struct VHDR VHDR;
419 yd struct IFF *IFF;
420 kB extern USHORT Hardware;
421 Cr extern void SetStart(),SetEnd(),SetPeriod(),SetRate(),SetVo
          lume(),SetOneShot();
422 F2 extern struct StringInfo StrRampInfo1,StrRampInfo2;
423 8d void (*function[]) () = { &SetStart,&SetEnd,&SetPeriod,&Set
          Rate,&SetVolume,&SetOneShot };
424 Xt main()
425 nG {
426 LM1 IntuitionBase=(struct IntuitionBase *)OpenLibrary("intuiti
          on.library",0);
427 e6 if (! IntuitionBase) GoodBye();
428 vZ GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0
          );
429 OS if (! GfxBase) GoodBye;
430 bK MyPotiInfo.Flags=AUTOKNOB FREEHORIZ;
431 zv MyPotiInfo.HorizBody=512;
432 sE MyPotiInfo.HorizPot=0;
433 Sv IFF=AllocMem(sizeof(struct IFF),MEMF_PUBLIC|MEMF_CLEAR);
434 9V FileInfoBlock=AllocMem(sizeof(struct FileInfoBlock),MEMF_P
          UBLIC|MEMF_CLEAR);
435 zz Paula=(struct Interrupt *)AllocMem(sizeof(struct Interrupt
          t),MEMF_PUBLIC);
436 DO c2=ChipSleepy=AllocMem(sizeof(Sleepy),MEMF_CHIP);
437 tq c1=Sleepy;
438 3r for (i=0;i<54;i++) *c2++=*c1++;
439 Cq c2=ChipLeft=AllocMem(sizeof(LeftData),MEMF_CHIP);

```



```

440 8h LeftImage.ImageData=(USHORT *)c2;
441 cs c1=LeftData;
442 Bz for (i=0;i<18;i++) *c2++=*c1++;
443 fh c2=ChipRight=AllocMem(sizeof(RightData),MEMF_CHIP);
444 4U RightImage.ImageData=(USHORT *)c2;
445 7v c1=RightData;
446 F3 for (i=0;i<18;i++) *c2++=*c1++;
447 aS MySpriteData=c2=c1=AllocMem(516,MEMF_CHIP|MEMF_CLEAR);
448 AU *c2+=0x0000;
449 BV *c2+=0x0000;
450 Ya for (i=0;i<128;i++) *c2+=0x8000,*c2+=0x8000;
451 nf if (! (DirBuffer=AllocMem(2000,MEMF_PUBLIC|MEMF_CLEAR))) Go
odBye();
452 LV NewWindow.FirstGadget=&GGSelect4;
453 pA MyWindow=OpenWindow(&NewWindow);
454 CE if (! MyWindow) GoodBye();
455 Wz RastPort=MyWindow->RPort;
456 1D SetAPen(RastPort,0);
457 ny SetMenuStrip(MyWindow,&MenuProject);
458 gB DrawImage(RastPort,&LeftImage,0,0);
459 Ym DrawImage(RastPort,&RightImage,0,0);
460 eM DisplayTrack(0,FALSE);
461 4s Stop();
462 vc SetPointer(MyWindow,ChipSleepy,26,16,0,0);
463 tB for (i=0;i<=3;i++)
464 Qt6 {
465 nq7 SetStatus(i);
466 YI period[i]=pre_period[i]=358;
467 q9 samplingrate[i]=10000;
468 Ob MemoryType[i]=0;
469 J4 volume[i]=64;
470 u1 IntCount[i]=0;
471 CQ CurrentInt[i]=0;
472 Cm Modus[i]=LOOP;
473 DI currenty[i]=0;
474 Nr FastIntVariables[i].CopyBuffer1=AllocMem(64,MEMF_CHI
P);
475 Oy RepeatVariables[i].CopyBuffer1=FastIntVariables[i].C

```

```

476 Rw opyBuffer1;
FastIntVariables[i].CopyBuffer2=AllocMem(64,MEMF_CHI
P);
477 26 RepeatVariables[i].CopyBuffer2=FastIntVariables[i].C
opyBuffer2;
478 I9 FastIntVariables[i].RestBuffer=AllocMem(128,MEMF_CHI
P);
479 tQ RepeatVariables[i].RestBuffer=AllocMem(128,MEMF_CHIP
);
480 kF6 }
481 QQ1 parameter[0]=startpos;
482 X1 parameter[1]=end;
483 T1 parameter[2]=period;
484 Ty parameter[3]=samplingrate;
485 MT parameter[4]=volume;
486 I4 parameter[5]=oneshot;
487 Vy InitMemory();
488 EJ SetAPen(RastPort,0);
489 aJ SetDrMd(RastPort,COMPLEMENT);
490 1l RectFill(RastPort,9,11,49,23);
491 Xr RectFill(RastPort,114,214,201,229);
492 1X InversGadget(GadgetTable[0]);
493 hd FORM_ID=MakeID('F','O','R','M');
494 8H _BSVX_ID=MakeID('S','S','V','X');
495 O4 VHDR_ID=MakeID('V','H','D','R');
496 6y BODY_ID=MakeID('B','O','D','Y');
497 Mw ClearPointer(MyWindow);
498 aH FOREVER
499 zS {
500 Lx2 if (PLAYING && (AudioBits & mask[Current]) && !DisplayMod
us)
501 1U6 {
502 XP7 SpriteGot=GetSprite(&MySprite,1);
503 rJ MySprite.x=7;
504 mU MySprite.y=29;
505 IM MySprite.height=128;

```

Listing 4. (Fortsetzung)

Super-Software Abonnement:

Wußten Sie, daß Sie die Disketten zu den Listings aus diesem Heft im Abonnement bestellen können? Für nur **DM 298,-** jährlich erhalten Sie jeden Monat die Diskette aus einer dieser Zeitschriften: 64'er, Amiga Magazin, PC Magazin Plus oder ST-Magazin.

Für Schüler und Studenten gibt's Ermäßigung: **DM 278,-** jährlich (bitte Kopie des Ausweises beifügen). Sie können vierteljährlich, halbjährlich oder jährlich bezahlen.

Sie sparen sich durch ein Abo mehr als **DM 50,-** und die Zeit für die Bestellabwicklung - deshalb: am besten gleich den Coupon ausschneiden und ausgefüllt an die genannte Adresse schicken!

Das Abo kann innerhalb von 8 Tagen widerrufen werden.



Markt&Technik Verlag AG
Unternehmensbereich Buchverlag
Hans-Pinsel-Str. 2, 8013 Haar bei München



Name

Telefon

Adresse

Abo für Zeitschrift/
ab Ausgabe

Datum/Unterschrift

Zahlungsweise $\frac{1}{4}$ $\frac{1}{2}$ $\frac{1}{1}$

Unterschrift des Erziehungs-
berechtigten bei Jugend-
lichen unter 18 Jahren



C O U P O N


```

506 5Z      ChangeSprite(0,&MySprite,MySpriteData);
507 JZ      while (CheckPort(MyWindow->UserPort))
508 8bE      {
509 ZJF      struct FastIntVariables *IntData;
510 1n      ULONG *StartPtr=&RangeStart[Current],*EndPtr
=&RangeEnd[Current];
511 Kv      IntData=(SysBase->IntVects[7+Current]).iv_D
ata;
512 w7      NewPosition=IntData->MemCounter-start[Curre
nt];
513 VQ      if (NewPosition>=*StartPtr && NewPosition<
=*EndPtr)
514 opJ      SpritePosition=7+304*(NewPosition - *Sta
rtPtr)/(*EndPtr - *StartPtr);
515 cHF      else SpritePosition=0;
516 1S      if (SpritePosition != OldPosition)
517 HkJ      {
518 NpK      MoveSprite(0,&MySprite,SpritePosition,2
9);
519 bp      OldPosition=SpritePosition;
520 OtJ      }
521 PuE      }
522 VS7      FreeSprite(SpriteGot);
523 Rw6      }
524 oB2      else WaitPort(MyWindow->UserPort);
525 8a      if (MyMessage=GetMsg(MyWindow->UserPort))
526 Qt5      {
527 h1      class=MyMessage->Class;
528 X5      code=MyMessage->Code;
529 Cs      msgaddress=MyMessage->IAddress;
530 8J      switch(class)
531 Vy6      {
532 Mz7      case MOUSEMOVE:
533 yrC      FreeHand(Current);
534 yx      Skip=TRUE;
535 Yh      break;
536 4G7      case CLOSEWINDOW:
537 gqC      GoodBye();
538 bk      break;
539 wg7      case MENUPICK:
540 OJC      MenuNum=MENUNUM(code);
541 Ek      ItemNum=ITEMNUM(code);
542 9s      SubNum=SUBNUM(code);
543 s4      switch(MenuNum)
544 1BD      {
545 n3E      case 0:
546 kZ      switch(ItemNum)
547 1EF      {
548 s9G      case 1:
549 eD      track=Current;
550 dV      if (!AllocationCheck(track)) continue;
551 a6      if (!GetFileName(BufferFileName,CharacterBu
ffer,"Save track",DirBuffer,2000,Resident))
552 arK      { Resident=TRUE;
553 qzM      break;
554 wRK      }
555 tYG      if (! (MYFILE=fopen(CharacterBuffer,"w")))
556 Q9K      { SetWindowTitles(MyWindow,"Sorry, I ca
n't open this file!",-1);
557 IsM      Resident=TRUE;
558 73      continue; }
559 UBG      SetPointer(MyWindow,ChipSleepy,26,16,0,0);
560 K2      Resident=FALSE;
561 R7      switch(SubNum)
562 OTH      {
563 5LI      case 0:
564 zy      WriteBody(1);
565 2B      break;
566 AR      case 1:
567 wn      fwrite(&length[track],1,4,MYFILE);
568 k5      fwrite((UBYTE *)&samplingrate[track])+2,
1,2,MYFILE);
569 43      WriteBody(1);
570 7G      break;
571 HZ      case 2:
572 Au      if (length[track] < 32767)
573 BeM      {
574 9QN      SaveIFF(5);
575 IL      WriteBody(5);
576 InM      }

```

```

577 4KI      else {
578 t6O      SaveIFF(1);
579 ED      WriteBody(1);
580 MrN      }
581 IRI      break;
582 OtH      }
583 kKG      ClearPointer(MyWindow);
584 hE      EmptyPort();
585 MV      break;
586 Wo      case 2:
587 vX      if (!GetFileName(BufferFileName,CharacterBu
ffer,"Delete file",DirBuffer,2000,Resident)
)
588 ARK      { Resident=TRUE;
589 QZM      break;
590 W1K      }
591 P3G      if (DisplayRequest(4,"Warning!")==CANCEL) b
reak;
592 8y      DeleteFile(CharacterBuffer);
593 rZ      Resident=FALSE;
594 rO      EmptyPort();
595 Wf      break;
596 cs      case 0:
597 1r      if (!GetFileName(BufferFileName,CharacterBu
ffer,"Load track",DirBuffer,2000,Resident))
598 KbK      { Resident=TRUE;
599 wTM      EmptyPort();
600 bk      break;
601 hCL      }
602 1bG      Resident=TRUE;
603 W5      track=Current;
604 Du      SetPointer(MyWindow,ChipSleepy,26,16,0,0);
605 1p      if (!LoadFile(CharacterBuffer))
606 VvK      { ClearPointer(MyWindow);
607 4bM      EmptyPort();
608 vr      continue; }
609 AkG      ClearPointer(MyWindow);
610 9X      pre_period[track]=period[track];
611 IQ      pre_oneshot[track]=oneshot[track];
612 oJ      DisplayTrack(Current,TRUE);
613 oQ      SetStatus(Current);
614 B1      EmptyPort();
615 qz      break;
616 2L      case 3:
617 Dr      DisplayRequest(0,"About program");
618 Fm      EmptyPort();
619 u3      break;
620 OVf      }
621 w5E      break;
622 4L      case 1:
623 zo      switch(ItemNum)
624 OTF      {
625 5LG      case 0:
626 97      if (!AllocationCheck(Current)) continue;
627 rO      if (DisplayRequest(5,"Warning!")==CANCEL )
break;
628 bI      SetPointer(MyWindow,ChipSleepy,26,16,0,0);
629 pA      ClearTrack(Current);
630 V5      ClearPointer(MyWindow);
631 6F      break;
632 EV      case 1:
633 Hn      if (DisplayRequest(6,"Warning!")==CANCEL) b
reak;
634 hO      SetPointer(MyWindow,ChipSleepy,26,16,0,0);
635 pR      for (track=0;track<=3;track++)
636 4m      if (start[track]) ClearTrack(track);
637 cC      ClearPointer(MyWindow);
638 DM      break;
639 NF      case 2:
640 NL      if (!AllocationCheck(Current)) continue;
641 aD      ResetTrack(Current);
642 Ht      SetStatus(Current);
643 IR      break;
644 OtF      }
645 KTE      break;
646 Um      case 2:
647 NC      switch(ItemNum)
648 OrF      {
649 TJG      case 0:
650 aq      if (!AllocationCheck(Current)) continue;

```



```

651 nw      track=(UBYTE)DisplayRequest(2,"Copy a track
652 qx      ");
653 rg      if (!track) break;
654 eM      track--;
655 VyK      if (!start[track])
656 SyL      {
        if (!Memory(track,length[Current])) co
        ntinue;
        SetPeriod(track,period[Current]);
        oneshot[track]=oneshot[Current]-startp
        os[Current];
        }
659 d8K      if (length[Current] > length[track]) MinLe
660 SF6      ngth=length[track];
        else MinLength=length[Current];
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        SetWindowTitles(MyWindow,"Copying track..."
        ,-1);
664 NL      FastCopy(start[Current]+startpos[Current],s
        tart[track]+startpos[track],MinLength);
        ClearPointer(MyWindow);
        EmptyPort();
        break;
        case 1:
        if (!AllocationCheck(Current)) continue;
        track=DisplayRequest(3,"Concat Range");
        if (!track) break;
        track--;
        if (!AllocationCheck(track)) continue;
        SetWindowTitles(MyWindow,"Concat the tracks
        ...",-1);
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        if (!Concat(Current,track))
        { SetWindowTitles(MyWindow,Out_of_Memor
        y,-1);
        ClearPointer(MyWindow);
        continue; }
        ClearPointer(MyWindow);
        RangeStart[track]=0;
        RangeEnd[track]=(allocated[track]-1)/608*60
        8+607;
        if (track==Current) DisplayTrack(Current,FA
        LSE);
        EmptyPort();
        break;
        case 2:
        if (!AllocationCheck(Current)) continue;
        track=DisplayRequest(7,"Paste Range");
        if (!track) break;
        track--;
        if (!AllocationCheck(track)) continue;
        SetWindowTitles(MyWindow,"Pasting range..."
        ,-1);
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        if (!Paste(Current,track))
        { SetWindowTitles(MyWindow,Out_of_Memor
        y,-1);
        ClearPointer(MyWindow);
        continue; }
        ClearPointer(MyWindow);
        RangeStart[track]=0;
        RangeEnd[track]=(allocated[track]-1)/608*60
        8+607;
        if (track==Current) DisplayTrack(Current,FA
        LSE);
        EmptyPort();
        break;
        case 3:
        if (!AllocationCheck(Current)) continue;
        track=DisplayRequest(7,"Insert Range");
        if (!track) break;
        track--;
        if (!AllocationCheck(track)) continue;
        SetWindowTitles(MyWindow,"Inserting range..
        ...",-1);
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        if (!InsertRange(Current,track))
        { SetWindowTitles(MyWindow,Out_of_Memor
        y,-1);
        ClearPointer(MyWindow);
        continue; }
674 rRM
675 ea

```

```

716 tTG      ClearPointer(MyWindow);
717 XO      RangeStart[track]=0;
718 vS      RangeEnd[track]=(allocated[track]-1)/608*60
        8+607;
        if (Current==track)
        {
        SetStart(Current,startpos[Current]);
        SetEnd(Current,end[Current]);
        SetOneShot(Current,oneshot[Current]);
        DisplayTrack(Current,FALSE);
        }
725 hCK      EmptyPort();
726 zWG      break;
727 en      }
728 kFF      break;
729 gpE      case 3:
730 sB      switch(ItemNum)
731 jY      {
732 kDF      case 0:
733 p5G      track=Current;
734 dC      if (!AllocationCheck(track)) continue;
735 cU      SetWindowTitles(MyWindow,"Reversing track..
        ...",-1);
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        Reverse(track);
        ClearPointer(MyWindow);
        DisplayTrack(Current,FALSE);
        EmptyPort();
        break;
        case 1:
        if (!AllocationCheck(Current)) continue;
        track=DisplayRequest(7,"Mix two tracks");
        if (!track) break;
        track--;
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        if (!AllocationCheck(track)) continue;
        SetWindowTitles(MyWindow,"Mixing the tracks
        ...",-1);
        Mix(track);
        ClearPointer(MyWindow);
        EmptyPort();
        break;
        case 2:
        if (!AllocationCheck(Current)) continue;
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        SetWindowTitles(MyWindow,"Period *= 2 ...",
        -1);
        if (!Double(Current))
        { SetWindowTitles(MyWindow,Out_of_Memor
        y,-1);
        ClearPointer(MyWindow);
        continue; }
        ClearPointer(MyWindow);
        EmptyPort();
        break;
        case 3:
        if (!AllocationCheck(Current)) continue;
        SetWindowTitles(MyWindow,"Period /= 2 ...",
        -1);
        SetPointer(MyWindow,ChipSleepy,26,16,0,0);
        Halve(Current);
        ClearPointer(MyWindow);
        EmptyPort();
        break;
        }
774 UzF      case 4:
775 dxE      switch(ItemNum)
776 SH      {
777 TwF      case 0:
778 YoG      switch(SubNum)
779 xd      {
780 WzH      case 0:
781 brI      Hardware=0;
782 WN      break;
783 Yh      case 1:
784 gx      Hardware=1;
785 fS      break;
786 bk      }
787 hCH      }
788 dmG      break;

```

Listing 4. (Fortsetzung)


```

789 JEF      }
790 foE      break;
791 LGD      }
792 hqA      break;
793 6B      case GADGETDOWN:
794 yF      ID=((struct Gadget *)msgaddress)->GadgetID;
795 bX      if ( ID==DRAW )
796 mFE      {
797 E7F      FreeHand(Current);
798 ED      Skip=TRUE;
799 ox      break;
800 uPE      }
801 qhA      if ( (ID>=START) && (ID<=ONESHOT) )
802 sLG      {
803 B5H      InversGadget(GadgetTable[currenty[Current]
            ]);
804 SX      currenty[Current]=ID-START;
805 D7      InversGadget(GadgetTable[currenty[Current]
            ]);
806 Ff      if (currenty[Current]<2 && DisplayModus)
            DisplayTrack(Current,FALSE);
807 w5      break;
808 2XG      }
809 p8A      if ( ID==RIGHT )
810 OTF      {
811 qLG      current=currenty[Current];
812 xF      parapointer=parameter[current];
813 oF      if (current<2 || current==5) SoundOFF(Cur
            rent);
814 UK      do {
815 r9      function[current](Current,parapointer[Curre
            nt]+Step[current]);
816 3M      } while(!(64 & *((UBYTE *)0xbfe001)));
817 IZ      if (current<2 || current==5) SetStatus(Cur
            rent);
818 pC      if (current<2 && DisplayModus) DisplayTrac
            k(Current,FALSE);
819 8H      break;
820 EjF      }
821 QfA      if ( ID==LEFT )
822 CfF      {
823 2XG      current=currenty[Current];
824 9R      parapointer=parameter[current];
825 7J      if (current<2 || current==5) SoundOFF(Curre
            nt);
826 gW      do {
827 9T      function[current](Current,parapointer[Curre
            nt]-Step[current]);
828 FY      } while(!(64 & *((UBYTE *)0xbfe001)));
829 Ul      if (current<2 || current==5) SetStatus(Cur
            rent);
830 10      if (current<2 && DisplayModus) DisplayTrac
            k(Current,FALSE);
831 KT      break;
832 QvF      }
833 FkA      if ( ID==MYPOTI )
834 OrF      {
835 BYG      ULONG SetValue;
836 Fk      current=currenty[Current];
837 98      MaxiPart[0]=MaxiPart[1]=MaxiPart[5]=allocat
            ed[Current]-1;
838 Pm      if (currenty[Current]<2 || currenty[Curren
            t]==5) SoundOFF(Current);
839 tj      do {
840 ro      SetValue=(MyPotiInfo.HorizPot>>7)*MaxiPar
            t[current];
841 GO      SetValue /= 511;
842 u6      SetValue += PotiStart[current];
843 gI      function[current](Current,SetValue);
844 Vo      } while(!(64 & *((UBYTE *)0xbfe001)));
845 w9      if (currenty[Current]<2 || currenty[Curren
            t]==5) SetStatus(Current);
846 He      if (current<2 && DisplayModus) DisplayTrac
            k(Current,FALSE);
847 aj      break;
848 gBF      }
849 c1A      break;
850 HJ      case GADGETUP:
851 tA      ID=((struct Gadget *)msgaddress)->GadgetID;
852 jd      switch(ID)
853 hAF      {

```

```

854 G6G      case ZOOMOUT:
855 Ec      RangeStart[Current]=0;
856 LG      RangeEnd[Current]=(allocated[Current]-1)/60
            8*608+607;
857 OM      DisplayTrack(Current,FALSE);
858 lu      break;
859 bs      case ZOOMIN:
860 oJ      DisplayTrack(Current,TRUE);
861 ox      break;
862 GD      case ALL:
863 iC      if (DisplayModus)
864 sLK      {
865 PDL      DisplayModus=FALSE;
866 2o      PrintIText(RastPort,&TextAll,512,180);

            DisplayTrack(Current,FALSE);
867 YW      }
868 OVK      }
869 qdG      else
870 yRK      {
871 4KL      DisplayModus=TRUE;
872 Qv      SetAPen(RastPort,0);
873 n6      SetDrMd(RastPort,JAM1);
874 oU      RectFill(RastPort,513,181,513+115,191)
            ;
875 ds      PrintIText(RastPort,&TextOne_One,512,1
            80);
876 hf      DisplayTrack(Current,FALSE);
877 9eK      }
878 5EG      break;
879 KA      case PLAY:
880 OB      if (!PLAYING)
881 9cK      {
882 v4L      SetDrMd(RastPort,COMPLEMENT);
883 b6      SetAPen(RastPort,0);
884 Ot      RectFill(RastPort,8,214,95,229);
885 tD      RectFill(RastPort,114,214,201,229);
886 InK      }
887 CpG      PLAYING=TRUE;
888 x1      Stop();
889 Jr      Play();
890 HQ      break;
891 JO      case RECORD:
892 lp      Stop();
893 E5      SetWindowTitles(MyWindow,"Waiting for mouse
            click...",-1);
894 we      Test();
895 FJ      RecordTrack(Current,0);
896 JC      pre_period[Current]=period[Current];
897 kH      EmptyPort();
898 QL      DisplayTrack(Current,TRUE);
899 QZ      break;
900 KQ      case STOP:
901 ig      if (PLAYING)
902 UxK      {
903 GPL      SetDrMd(RastPort,COMPLEMENT);
904 wR      SetAPen(RastPort,0);
905 DX      RectFill(RastPort,114,214,201,229);
906 MF      RectFill(RastPort,8,214,95,229);
907 d8K      }
908 y9G      PLAYING=FALSE;
909 I6      Stop();
910 bk      break;
911 61      case AUTOREC:
912 L9      Stop();
913 xe      SetWindowTitles(MyWindow,"Waiting for Sound
            ...",-1);
914 ds      WaitSound();
915 6G      RecordTrack(Current,"Auto Recording...");
916 id      DisplayTrack(Current,TRUE);
917 ir      break;
918 ht      case ADJUST:
919 SG      Stop();
920 Sv      SetWindowTitles(MyWindow,"Adjust your digit
            izer...",-1);
921 jd      Adjust();
922 9g      EmptyPort();
923 ox      break;
924 bb      case TEST:
925 YM      Stop();
926 ul      SetWindowTitles(MyWindow,"Testing quality..
            .",-1);

```



```

927 TB      Test();
928 Fm      EmptyPort();
929 xM      /*
930 y7      if (!Listen((UWORD)period[Current]))
931 T7K      { Listening=FALSE;
932 xbM      break; }
933 aRG      Listening=TRUE;    /*
934 z8      break;
935 5aF      }
936 bhA      if (ID==START)
937 3WF      {
938 nIG      SoundOFF(Current);
939 I1      SetStart(Current,StartInfo.LongInt);
940 5h      SetStatus(Current);
941 6F      break;
942 ChF      }
943 ZsA      if ( (ID>=STATUS1) && (ID<=STATUS4) )
944 AdF      {
945 UnG      Number=ID-STATUS1;
946 PB      if (!start[Number]) break;
947 ab      AudioBits ^= mask[Number];
948 Rb      SetStatus(Number);
949 EN      break;
950 KpF      }
951 cmA      if ( ID==PERIOD )
952 I1F      {
953 L5G      SetPeriod(Current,PeriodInfo.LongInt);
954 JS      break;
955 PuF      }
956 EZA      if ( ID==ONESHOT )
957 NqF      {
958 7LG      SoundOFF(Current);
959 HM      SetOneShot(Current,OneShotInfo.LongInt);
960 P1      SetStatus(Current);
961 QZ      break;
962 W1F      }
963 seA      if ( ID==RATE )
964 UxF      {
965 UyG      SetRate(Current,RateInfo.LongInt);
966 Ve      break;
967 b6F      }
968 bJA      if ( (ID>=SELECT1) && (ID<=SELECT4) )
969 Z2G      {
970 smH      InversGadget(GadgetTable[currenty[Current]
971 MS      ]);
972 uo      SelectTrack(ID-SELECT1);
973 c1      InversGadget(GadgetTable[currenty[Current]
974 iDG      ]);
975 KGA      break;
976 g9F      }
977 J7G      if ( ID==VOLUME )
978 hq      {
979 nIF      SetVolume(Current,VolumeInfo.LongInt);
980 BkA      break;
981 LEF      }
982 NsG      if ( (ID>=MODUS1) && (ID<=MODUS4) )
983 ZB      {
984 oHK      Number=ID-MODUS1;
985 P1L      if (Modus[Number]==LOOP)
986 Rb      {
987 vQK      Modus[Number]=ONCE;
988 IYG      PrintIText(RastPort,&TextOnce,105+161*
989 tMK      Number,11);
990 WSL      }
991 8d      else
992 OVK      {
993 w5G      Modus[Number]=LOOP;
994 2XF      PrintIText(RastPort,&TextLoop,105+161*
995 FwA      Number,11);
996 OTF      }
997 kyG      break;
998 cv      }
999 2e      if ( ID==END )
1000 3C      {
1001 9eF      SoundOFF(Current);
1002 5EA      SetEnd(Current,EndInfo.LongInt);
1003 Bg9      SetStatus(Current);
          break;
      }

```

```

1004 Ch4      }
1005 f12      ReplyMsg(MyMessage);
1006 rB      if (Skip) Skip=FALSE;
1007 4r      else
1008 Cf5      {
1009 z86      for (i=0;i<40;i++) MyTitle[i]=0;
1010 Kh      Number_to_String(MyTitle,(2+ChipEnd-ChipPointer)/1024
1011 iw      ,2);
1012 VZ      strcat(MyTitle,"k chip + ");
1013 Ok      Number_to_String(MyTitle+strlen(MyTitle),(2+FastEnd-F
1014 sF      astPointer)/1024,3);
1015 Ed      strcat(MyTitle,"k fast memory free");
1016 Ot5      SetWindowTitles(MyWindow,MyTitle,-1);
1017 Pu1      RefreshOneShot(Current);
1018 Qv0      }
(C) 1987 M&T

```

Listing 4. (Schluß)

Programmname: displaytrack.c

Computer: Amiga 500, 1000, 2000 mit Kickstart 1.2

Sprache: C

Compiler: Aztec C V3.6

Aufrufe: cc +l -s +idmp displaytrack.c

Programm : displaytrack.c

```

1 cb0 #define abs(x) ((x)<0?-(x):(x))
2 7L extern struct Window *MyWindow;
3 Bh extern struct RastPort *RastPort;
4 oE extern UBYTE coordinates[4];
5 iJ extern ULONG start[4],startpos[4],end[4],allocated[4],length[4],oneshot[4];
6 pc extern BOOL DisplayModus;
7 V1 extern ULONG RangeStart[4],RangeEnd[4];
8 r8 extern UBYTE currenty[4];
9 H3 extern char *MyPlane;
10 Q5 CalculateAll(Track)
11 FY1 UBYTE Track;
12 8b0 {
13 bY1 UWORD fast,part;
14 Us UBYTE *value;
15 op UBYTE sampledvalue;
16 x7 ULONG sum,endpos;
17 IE part=(RangeEnd[Track]-RangeStart[Track])/608+1;
18 PP value=start[Track]+RangeStart[Track]+part;
19 oB endpos=start[Track]+allocated[Track]-1;
20 Oy for (fast=1;fast<609;fast++)
21 Hk6 {
22 zP7 if (endpos > value) sampledvalue=*value;
23 JK else sampledvalue=0;
24 Cv sampledvalue^=128;
25 3e sampledvalue=sampledvalue>>1;
26 FG coordinates[fast-1]=156-sampledvalue;
27 bp value += part;
28 Sx6 }
29 8q1 coordinates[608]=92;
30 Uz0 }
31 qT NextPosition(Position,track)
32 Iq1 ULONG Position;
33 9y UBYTE track;
34 Ux0 {
35 qm1 register UWORD fast;
36 qE UBYTE *value;
37 AB UBYTE sampledvalue;
38 Db ULONG MemoryStart;
39 Yk ULONG MemoryEnd;
40 oV fast=0;
41 i4 MemoryStart=start[track]+Position;
42 mt MemoryEnd=start[track]+allocated[track];

```

Listing 5. »displaytrack.c«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

43 Aa   while (fast<609)
44 e77   {
45 8y8       value=(UBYTE *) (MemoryStart+fast);
46 Wr       sampledvalue=*value;
47 n0       if (value > MemoryEnd) sampledvalue=0;
48 S2       sampledvalue ^= 128;
49 R2       sampledvalue=sampledvalue>>1;
50 SQ       coordinates[fast]=156-sampledvalue;
51 5r       fast++;
52 qL7     }
53 rM0     }
54 EG   DisplayTrack(Number,Flag)
55 fV1   UBYTE Number,Flag;
56 qJ0   {
57 7i1     UWORD i;
58 An     if (!start[Number])
59 tM5     {
60 XR6       for (i=0;i<609;i++) coordinates[i]=92;
61 9v       DrawIt();
62 1a       return;
63 1W5     }
64 OY1   if (Flag)
65 zS5     {
66 6b6       RangeStart[Number]=startpos[Number];
67 UZ       RangeEnd[Number]=startpos[Number]+(end[Number]-startpos[Number])/608*608+607;
68 6b5     }
69 uO1   if (DisplayModus)
70 4X5     {
71 wO6       if (currenty[Number]==1 && length[Number]>608)
72 gXB         NextPosition(end[Number]-608,Number);
73 Ja6       else NextPosition(startpos[Number],Number);
74 M8       DrawIt();
75 En       return;
76 Ej5     }
77 4r1   else
78 Cf5     {
79 vP6       CalculateAll(Number);
80 SE       DrawIt();
81 oU       DrawLine(Number,startpos[Number]);
82 h8       DrawLine(Number,end[Number]);
83 xF       if (oneshot[Number]) DrawLine(Number,oneshot[Number]);
84 Mr5     }
85 Ns0   }
86 Ni   DrawIt()
87 Lo   {
88 La1     MyPlane=MyWindow->RPort->BitMap->Planes[0];
89 9S     SetDrMd(RastPort,JAM1);
90 oJ     SetAPen(RastPort,0);
91 K2     RectFill(RastPort,9,28,628,157);
92 Ip     DrawLines(coordinates,607,0);
93 V00   }
94 KO   #asm
95 d8A     public _DrawLines
96 hN0   _DrawLines:
97 IrA     movem.l d0-d7/a0-a6,-(a7)
98 EF     clr.l d0
99 RK     clr.l d2
100 xA     move.l 64(a7),a0 ;_MemStart
101 Dq     move.l 72(a7),d0 ;_PixelPos
102 Z6     move.l _MyPlane,a1
103 eX     divu #16,d0
104 qZ     asl.w #1,d0
105 g7     add.w d0,a1
106 Qj     clr.w d0
107 IR     swap d0
108 RY     move.l #$80000000,d1
109 oe     lsr.l d0,d1
110 ZD     move.b (a0),d2
111 eC     mulu #80,d2
112 zI     add.w d2,a1
113 1i     add.w #2,a1
114 kv     clr.w d2
115 UH     move.b (a0)+,d2
116 70     move.w #80,d4
117 UW     move.w #2,d5
118 Ta     move.w #1,d6
119 zP     move.l 68(a7),d0
120 p2     move.l #$80000000,d7

```

```

121 C50 nextbit: cmp.b (a0),d2
122 rSA     beq equal
123 Yq     bcc 12
124 V7     bcs 11
125 yR0 equal: or.l d1,(a1)
126 f7A     lsr.l d6,d1
127 Ya     add.w d6,a0
128 Un     bra.s 13
129 Uc0 11: sub.b (a0),d2
130 HhA     neg.b d2
131 UQ     asr.b d6,d2
132 oF     bcc even1
133 29     or.l d1,(a1)
134 OV     adda.w d4,a1
135 Vx0 even1: move.w d2,d3
136 DVA     bra lstart1
137 CH0 14: or.l d1,(a1)
138 SZa     adda.w d4,a1
139 Lr0 lstart1: dbra d2,14
140 tLA     lsr.l d6,d1
141 Oc     bra lstart2
142 KQ0 15: or.l d1,(a1)
143 Xea     adda.w d4,a1
144 d70 lstart2: dbra d3,15
145 FQA     clr.w d2
146 zm     move.b (a0)+,d2
147 d4     bra 13
148 qz0 12: sub.b (a0),d2
149 mLA     asr.b d6,d2
150 Ae     bcc even2
151 KR     or.l d1,(a1)
152 xb     suba.w d4,a1
153 t40 even2: move.w d2,d3
154 hrA     bra lstart3
155 ah0 16: or.l d1,(a1)
156 iFA     suba.w d4,a1
157 rR0 lstart3: dbra d2,16
158 BdA     lsr.l d6,d1
159 sy     bra lstart4
160 iQ0 17: or.l d1,(a1)
161 6kA     suba.w d4,a1
162 9h0 lstart4: dbra d3,17
163 XIA     clr.w d2
164 H4     move.b (a0)+,d2
165 X90 13: tst.w d1
166 CKA     beq.s nextbyte
167 AH     add.w d5,a1
168 Qw     move.l d7,d1
169 mk0 nextbyte: dbra d0,nextbit
170 25A     movem.l (a7)+,d0-d7/a0-a6
171 pq     rts
172 QK1     #endasm
173 Qs0 FreeHand(Current)
174 wq1 UBYTE Current;
175 lEO {
176 tJ1 UBYTE *MyPtr=start[Current]+startpos[Current];
177 ML   SHORT x,y;
178 fR   ULONG *StartPtr=&RangeStart[Current],*EndPtr=&RangeEnd[Current];
179 yt   if (!AllocationCheck(Current)) return;
180 qe   if (!DisplayModus)
181 rK5   {
182 dI6     SHORT LineStart,LineEnd,LineRepeat;
183 hn     ULONG NewStart,NewEnd,NewRepeat;
184 dr     SoundOFF(Current);
185 LB     do {
186 8p     x=MyWindow->MouseX;
187 p6     if (x<16 || x>625) continue;
188 nE     LineStart=608*(startpos[Current] - *StartPtr)/(*EndPtr - *StartPtr);
189 E5     LineEnd=608*(end[Current] - *StartPtr)/(*EndPtr - *StartPtr);
190 5M     LineRepeat=608*(oneshot[Current] - *StartPtr)/(*EndPtr - *StartPtr);
191 Cv     NewStart=abs(LineStart-(x-16));
192 G8     NewEnd=abs(LineEnd-(x-16));
193 G1     NewRepeat=abs(LineRepeat-(x-16));
194 TO     if ( NewStart < NewEnd && NewStart <= NewRepeat )
195 5YA     {
196 9oB     NewStart = (x-16)*(*EndPtr - *StartPtr)/608 + *StartPtr;

```



```

197 wp      SetStart(Current,NewStart);
198 ChA      }
199 S36      else if ( NewEnd > NewRepeat)
200 AdA      {
201 OeB      NewRepeat=(x-16)*(*EndPtr - *StartPtr)/608 + *St
artPtr;
202 ly      SetOneShot(Current,NewRepeat);
203 HmA      }
204 7u6      else
205 FiA      {
206 3cB      NewEnd=(x-16)*(*EndPtr - *StartPtr)/608 + *Start
Ptr;
207 Ch      SetEnd(Current,NewEnd);
208 MrA      }
209 GZ6      } while(!(64 & *((UBYTE *)0xbfe001)));
210 Jv      SetStatus(Current);
211 Pu5      }
212 Qv0      }
213 I1 DrawLine(Track,Position)
214 Wp1 UBYTE Track;
215 Fn      ULONG Position;
216 Qt0 {
217 nK1 ULONG LinePosition;
218 Y1 ULONG *StartPtr=&RangeStart[Track],*EndPtr=&RangeEnd[Track
];
219 1B      if (Position>=*StartPtr && Position<=*EndPtr)
220 Ux5      {
221 OQ6      LinePosition=608*(Position - *StartPtr)/(*EndPtr - *S
tartPtr);
222 HQ      SetDrMd(RastPort,COMPLEMENT);
223 I1      Move(RastPort,LinePosition+16,28);
224 5W      Draw(RastPort,LinePosition+16,156);
225 d85      }
226 e90      }
(C) 1987 M&T

```

Listing 5. (Schluß)

Programmname: mydir.c

Computer: Amiga 500, 1000, 2000 mit Kickstart 1.2

Sprache: C

Compiler: Aztec C V3.6

Aufrufe: cc -I -s -idmp mydir.c

Programm : mydir.c

```

1 hS0 /*****
2 tf /* Subroutine to get a filename from disk */
3 uJ /* written by: Christian Wolf */
4 GT /* Matthias-Claudius-Weg 58 */
5 QE /* 2190 Cuxhaven 1 */
6 QI /* 04721/24269 */
7 nY /*****
8 9o #define toAPTR(b) ((b)<<2)
9 Q8 #define toBPTR(a) ((a)>>2)
10 DV extern struct DosLibrary *DOSBase;
11 iC static struct RootNode *Root;
12 xy static struct DosInfo *DosInfo;
13 2z static struct DeviceList *DevList;
14 pB static struct NewWindow NewSpecialWindow = {
15 ex7 170,25,300,147,0,1,
16 7r CLOSEWINDOW GADGETUP GADGETDOWN DISKINSERTED DISKREM
OVED,
17 NN WINDOWCLOSEI ACTIVATEI WINDOWDEPTHI WINDOWDRAG,
18 Uy NULL,NULL,
19 u9 NULL,
20 W0 NULL,NULL,
21 1v 640,200,640,200,
22 QU WENCHSCREEN };
23 zZ0 static struct IntuiText SubTextLoad = {1,0,JAM1,3,2,NULL,"D
o it!",NULL };
24 rH static struct IntuiText TextVolumes = {1,0,JAM1,3,2,NULL,"S
how Volumes",NULL };
25 Gg static struct IntuiText TextParent = {1,0,JAM1,11,2,NULL,"P
arent Dir",NULL };

```

```

26 r9 static struct IntuiText TextDrawer = {1,0,JAM1,12,106,NULL,
"Drawer",NULL };
27 zq static struct IntuiText TextFile = {1,0,JAM1,12,120,NULL,"F
ile",&TextDrawer };
28 Oy static struct IntuiText TextFileName = {1,0,JAM2,0,0,NULL,N
ULL,NULL };
29 DP static USHORT SubLinesCancel[] = { 0,0,55,0,55,10,0,10,0,0
};
30 OF static USHORT LinesMyUp[] = { 0,0,19,0,19,14,0,14,0,0 };
31 hf static USHORT LinesDrawer[] = { 0,0,281,0,281,10,0,10,0,0 }
;
32 ly static USHORT LinesMyCursorUp[] = { 6,12,13,12,13,7,16,7,10
,2,9,2,3,7,6,7,6,12 };
33 80 static USHORT LinesMyCursorDown[] = { 6,2,13,2,13,7,16,7,10
,12,9,12,3,7,6,7,6,2 };
34 1Y static USHORT LinesParent[] = { 0,0,103,0,103,10,0,10,0,0 }
;
35 d1 static USHORT CursorData[] = { 0x0180,0x0180,0x0180,0x0180,
0x0180,0x3ffc,0x1fff8,
36 6YV 0x0fff0,0x07e0,0x03c0,0x0180,
37 6L 0x0180,0x03c0,0x07e0,0x0fff0,
0x1fff8,0x3ffc,0x0180,
38 mM 0x0180,0x0180,0x0180,0x0180
};
39 BK0 static struct Image CursorUpImage = { 2,2,16,11,1,NULL,1,0,
NULL };
40 Hq static struct Image CursorDownImage = { 2,2,16,11,1,NULL,1,
0,NULL };
41 iL static struct Border BorderMyCancel = { 0,0,1,3,JAM1,5,SubL
inesCancel,NULL };
42 z3 static struct Border BorderMyUp = { 0,0,1,3,JAM1,5,LinesMyU
p,NULL };
43 rt static struct Border BorderMyDown = { 0,0,1,3,JAM1,5,LinesM
yUp,NULL };
44 SH static struct Border BorderDrawer = { -58,-2,1,3,JAM1,5,Lin
esDrawer,NULL };
45 zU static struct Border BorderParent = { 0,0,1,3,JAM1,5,LinesP
arent,NULL };
46 aL static char BufferDrawer[108]="",CurrentDirectory[108];
47 7o static char BufferFileName[108]="";
48 qR static struct PropInfo MyPropInfo;
49 EB static struct Image MyPropImage;
50 MJ static struct StringInfo DrawerInfo = { BufferDrawer,0,0,10
8,0,0,0,0,10,27,0,0,0 };
51 hg static struct StringInfo FileNameInfo = { BufferFileName,0,
0,108,0,0,0,0,10,27,0,0,0 };
52 Wd #define LOAD 1
53 BX static struct Gadget GGSUBLoad = { NULL,234,132,56,11,GADGH
COMP,
54 12Z GADGIMMEDIATEI RELVERIFY,
BOOLGADGET,
55 JO &BorderMyCancel,0,&SubTe
xtLoad,0,0,LOAD,0 };
56 9G0 #define UP 3
57 G1 static struct Gadget GGSUBUp = { &GGSUBLoad,270,11,20,15,GA
DGHCOMP,
58 pX GADGIMMEDIATEI RELVERIFY,BO
OLGADGET,
59 U1 &BorderMyUp,0,NULL,0,0,UP,
0 };
60 Rz0 #define DOWN 4
61 Fo static struct Gadget GGSUBDown = { &GGSUBUp,270,86,20,15,GA
DGHCOMP,
62 tAZ GADGIMMEDIATEI RELVERIFY,
BOOLGADGET,
63 3S &BorderMyDown,0,NULL,0,0
,DOWN,0 };
64 jZ0 #define GETVOLUMES 5
65 G2 static struct Gadget GGGetVolumes = { &GGSUBDown,8,132,104,
11,GADGHCOMP,GADGIMMEDIATEI RELVERIFY,
66 udc BOOLGADGET,&BorderPar
ent,0,&TextVolumes,0,
0,GETVOLUMES,0 };
67 pr0 #define PARENT 6
68 i3 static struct Gadget GGParentDir = { &GGGetVolumes,121,132,
104,11,GADGHCOMP,GADGIMMEDIATEI RELVERIFY,
69 FwB BOOLGADGET,&BorderPare

```

Listing 6. »mydir.c«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

nt,0,&TextParent,0,0,P
ARENT,0];
70 Kv0 #define MYPROP 8
71 1F static struct Gadget GGMypProp = { &GGParentDir,270,29,20,54
,GADGHCOMP,
72 luY RELVERIFY1 GADGIMMEDIATE,P
ROPGADGET,
73 Mb &MyPropImage,0,NULL,0,&My
PropInfo,MYPROP,0];
74 Iv0 #define DRAWER 9
75 Je static struct Gadget GGDrawer = { &GGMypProp,66,106,220,10,G
ADGHCOMP,
76 VEY RELVERIFY1 STRINGCENTER,ST
RGADGET,
77 bG &BorderDrawer,0,NULL,0,&D
rawerInfo,DRAWER,0];
78 TI0 #define FILENAME 10
79 K3 static struct Gadget GGFileName = { &GGDrawer,66,120,220,10
,GADGHCOMP,
80 Z1a RELVERIFY1 STRINGCENTER,
STRGADGET,
81 OH &BorderDrawer,0,NULL,0,
&FileNameInfo,FILENAME,
0];
82 w10 #define FILESELECT 11
83 32 static struct Gadget GGFileSelect[9];
84 a5 #define A_DIRECTORY 1
85 ba #define NOT_A_DIRECTORY 0
86 LH #define PARENTDIR -1
87 AE #define VOLUME 2
88 5U static struct FileInfoBlock *FileInfoBlock;
89 TF static struct MemHeader MyMemHeader;
90 zb static struct MemChunk *MyMemChunk;
91 uU struct Name { char *EntryName;
92 dRE BOOL dir;
93 Hr struct Name *NextOne;
94 kz struct MsgPort *HandlerPort; };
95 dV0 static struct RastPort *SpecialRastPort;
96 O1 static struct Window *SpecialWindow;
97 22 static SHORT Number_of_Files,Number_of_Positions,Position=0
,OldPosition=0;
98 IJ static UBYTE Depth=0;
99 p1 static char parentdir[] = "parent",StringName[108];
100 1J static BOOL FileSelect,FirstTime=FALSE;
101 FC static ULONG MyLock,FileNumber;
102 bS static struct Name *FirstName=0,*CurrentName=0,*NextName,*M
yFindName();
103 O1 static USHORT *DataPointer;
104 1p BOOL GetFileName(FileName,FullName,Mode,MyBlock,Size,Reside
nt)
105 E11 char *FileName,*FullName,*Mode;
106 kG APTR MyBlock;
107 43 ULONG Size;
108 43 BOOL Resident;
109 ha0 {
110 x11 struct Gadget *TempGadget;
111 16 BYTE Count;
112 Zo struct IntuiMessage *SpecialMessage;
113 TX ULONG Class,ID;
114 pE BOOL End=TRUE,GetNext=FALSE;
115 sK USHORT *s;
116 gW if (!(FileInfoBlock=AllocMem(sizeof(struct FileInfoBlock),
MEMF_PUBLIC))) return FALSE;
117 UY NewSpecialWindow.Title=Mode;
118 OZ TempGadget=&GGFileName;
119 CN for (Count=0;Count<9;Count++)
120 j26 { GGFileSelect[Count].NextGadget=TempGadget;
121 I48 GGFileSelect[Count].GadgetType=BOOLGADGET;
122 QG GGFileSelect[Count].Activation=RELVERIFY;
123 gs GGFileSelect[Count].Flags=GADGHNONE;
124 Ux GGFileSelect[Count].Width=256;
125 fb GGFileSelect[Count].Height=10;
126 3H GGFileSelect[Count].LeftEdge=10;
127 26 GGFileSelect[Count].TopEdge=11+Count*10;
128 md GGFileSelect[Count].GadgetID=FILESELECT+Count;
129 1A TempGadget=&GGFileSelect[Count]; }
130 XV1 MyPropInfo.Flags=FREEVERT1 AUTOKNOB;
131 v1 MyPropInfo.VertPot=0;
132 Cc MyPropInfo.VertBody=65535;
133 1p NewSpecialWindow.FirstGadget=&GGFileSelect[8];
134 Ri if (!(SpecialWindow=OpenWindow(&NewSpecialWindow))) return

```

```

(FALSE);
135 Gy SpecialRastPort=SpecialWindow->RPort;
136 M1 DataPointer=s=AllocMem(sizeof(CursorData),MEMF_CHIP);
137 tH CursorDownImage.ImageData=DataPointer;
138 70 CursorUpImage.ImageData=DataPointer+11;
139 9J for (Count=0;Count<22;Count++) *s+=CursorData[Count];
140 CI DrawImage(SpecialRastPort,&CursorUpImage,270,11);
141 kC DrawImage(SpecialRastPort,&CursorDownImage,270,86);
142 Md PrintIText(SpecialRastPort,&TextFile,0,0);
143 3K while (GetMsg(SpecialWindow->UserPort));
144 oB if ((!FirstName || !Resident || !Depth))
145 HK5 {
146 KB6 MyMemHeader.mh_Lower=MyBlock;
147 rX MyMemHeader.mh_First=MyBlock;
148 E1 MyMemHeader.mh_Upper=MyBlock+Size;
149 vV MyMemHeader.mh_Free=Size-sizeof(struct MemChunk);
150 VK MyMemChunk=MyBlock;
151 OX MyMemChunk->mc_Next=0;
152 wP MyMemChunk->mc_Bytes=MyMemHeader.mh_Free;
153 Ty5 }
154 J61 else
155 Ru5 {
156 L16 if (MyLock=Lock(CurrentDirectory) )
157 TWA {
158 qFB Unlock(MyLock);
159 Pb MyPropInfo.VertBody=65535/(Number_of_Files/9+1);
160 PW if (Number_of_Files > 9 )
161 swF MyPropInfo.VertPot=(65535*Position)/(Number_
of_Files-9);
162 HcB OnlyRefresh(&GGMypProp,SpecialWindow);
163 CQ DisplayFiles(Position,FirstName);
164 1D WaitPort(SpecialWindow->UserPort);
165 29 goto msgarrived;
166 gBA }
167 hc5 }
168 Gx1 FOREVER
169 f82 {
170 CA3 FileSelect=TRUE;
171 8U FileNumber=255;
172 UD Position=0;
173 b7 if (!(MyLock=Lock(CurrentDirectory))) Depth=0;
174 KN else Unlock(MyLock);
175 n2 if (!FirstTime)
176 8g7 { FirstTime=TRUE;
177 iZ9 Depth=0; }
178 JH3 MyPropInfo.Flags=FREEVERT1 AUTOKNOB;
179 hU MyPropInfo.VertPot=0;
180 yO MyPropInfo.VertBody=65535;
181 av OnlyRefresh(&GGMypProp,SpecialWindow);
182 Ue ClearDisplay();
183 1L Number_of_Files=0;
184 QY GetNext=TRUE;
185 Kt FirstName=NextName=Allocate(&MyMemHeader,sizeof(struct N
ame));
186 4W if (Depth)
187 kC7 { NextName->EntryName=Allocate(&MyMemHeader,strlen(
parentdir));
188 189 strcpy(NextName->EntryName,parentdir);
189 AH NextName->dir=PARENTDIR;
190 bV NextName->NextOne=Allocate(&MyMemHeader,sizeof(st
ruct Name));
191 4H Number_of_Files++;
192 6L NextName=NextName->NextOne;
193 7h MyLock=Lock(CurrentDirectory);
194 C2 Examine(MyLock,FileInfoBlock); }
195 PF3 else { UWORD 1;
196 gHa char *VolName;
197 Go Root = DOSBase -> dl_Root;
198 z4 DosInfo = toAPTR(Root->rn_Info);
199 qn DevList = toAPTR(DosInfo->di_DevInfo);
200 aN while (DevList)
201 O9G { if(DevList->dl_Type == DLT_VOLUME && Dev
List->dl_Task)
202 yZL { VolName=toAPTR((BPTR)DevList->dl_Na
me)+1;
203 GTN Number_of_Files++;
204 CY l=strlen(VolName);
205 JX NextName->EntryName=Allocate(&MyMem
Header,1);
206 2u strcpy(NextName->EntryName,VolName)
;

```



```

207 12      NextName->dir=VOLUME;
208 HA      NextName->HandlerPort=DevList->dl_
           Task;
209 uo      NextName->NextOne=Allocate(&MyMemHe
           ader,sizeof(struct Name));
           NextName=NextName->NextOne; }
           DevList=toAPTR(DevList->dl_Next); }
210 tdm      GetNext=FALSE;
211 KhH      NextName->EntryName=0;
212 Jz9      NextName->NextOne=0;
213 nt      DisplayFiles(Position=0,FirstName);
214 r9      }
215 yn      End=TRUE;
216 Uz8      while (End)
217 yj3      {
218 GW          if (GetNext)
219 Tw4              {
220 CG5                  if (GetNext=ExNext(MyLock,FileInfoBlock))
221 Vy9                      {
222 c4A                          register UWORD l;
223 XOE                          Number_of_Files++;
224 VyF                          l=strlen(FileInfoBlock->fib_FileName);
225 ep                          NextName->EntryName=Allocate(&MyMemHeader,l
226 Wb                              );
227 ft                          strcpy(NextName->EntryName,FileInfoBlock->
228 zB                              fib_FileName);
229 fl                          if (FileInfoBlock->fib_EntryType>0) NextNa
230 HY                              me->dir=A_DIRECTORY;
231 GA                              else NextName->dir=NOT_A_DIRECTORY;
232 kz                              NextName->NextOne=Allocate(&MyMemHeader,siz
233 lGE                              eof(struct Name));
234 mH9                              NextName=NextName->NextOne;
235 Jy5                      }
236 AG                      else WaitPort(SpecialWindow->UserPort);
237 EW                      NextName->EntryName=0;
238 AC                      NextName->NextOne=0;
239 Gg                      if (Number_of_Files < 10) DisplayFiles(Position,First
240 oH9                          Name);
241 jvA                      if ((Number_of_Files % 10)==0)
242 BF                          {
243 av                              MyPropInfo.VertBody=65535/(Number_of_Files/9+1);
244 wR9                              MyPropInfo.VertPot=(65535*Position)/(Number_of_Fi
245 vR5                              les-9);
246 Nt                              OnlyRefresh(&GGMypProp,SpecialWindow);
247 v06                      }
248 K47                      msgarrived:
249 31                      if (SpecialMessage=GetMsg(SpecialWindow->UserPort))
250 YD                          {
251 zS8                              Class=SpecialMessage->Class;
252 Ug9                              ReplyMsg(SpecialMessage);
253 KJ                              switch(Class)
254 CV                                  {
                                  case CLOSEWINDOW:
                                  FreeMem(FileInfoBlock,sizeof(struct FileInfoBlock)
                                  );
                                  FreeMem(DataPointer,sizeof(CursorData));

```

```

255 iw      CloseWindow(SpecialWindow);
256 nM      return(FALSE);
257 4D      break;
258 HB      case DISKINSERTED:
259 ua      case DISKREMOVED:
260 3u      Depth=0;
261 eA      strcpy(BufferDrawer,"");
262 xo      strcpy(CurrentDirectory,"");
263 sz      OnlyRefresh(&GGDrawer,SpecialWindow);
264 MG      FreeList(FirstName);
265 jc      End=FALSE;
266 rc      strcpy(BufferFileName,"");
267 W9      OnlyRefresh(&GGFileName,SpecialWindow);
268 FO      break;
269 ej      case GADGETDOWN:
270 8l      ID=SpecialMessage->IAddress->GadgetID;
271 MG      switch(ID)
272 KnA      {
273 qIB          case DOWN:
274 sq              FileSelect=TRUE;
275 oA              FileNumber=255;
276 oe              do {
277 O9F                  if ((Position==Number_of_Files-9) || (Number
278 12                      _of_Files<10)) break;
279 4I                      Position += 1;
280 PiB                      DisplayFiles(Position,FirstName);
281 EL                      } while(!(64 & *((UBYTE *)0xbfe001)));
282 ptF                      if (Number_of_Files != 9)
283 EZB                          MyPropInfo.VertPot=(65535*Position)/(Number_
284 Ve                              of_Files-9);
285 XA                      OnlyRefresh(&GGMypProp,SpecialWindow);
286 42                      break;
287 OM                      case UP:
288 Oq                          FileSelect=TRUE;
289 kGF                          FileNumber=255;
290 6I                          do {
291 GU                              if (!Position) break;
292 buE                              Position -= 1;
293 QXB                              DisplayFiles(Position,FirstName);
294 15F                              } while(!(64 & *((UBYTE *)0xbfe001)));
295 QlB                      if (Number_of_Files != 9)
296 hq                          MyPropInfo.VertPot=(65535*Position)/(Number_
297 ig                              of_Files-9);
298 GE                          OnlyRefresh(&GGMypProp,SpecialWindow);
299 CY                          break;
300 C2                          case MYPROP:
301 WEF                              FileSelect=TRUE;
302 D6                              FileNumber=255;
303 ll                              do {
                                  Position=MyPropInfo.VertPot*(Number_of_Files
                                  -9)/65535;
                                  if (Position+8>Number_of_Files) Position=Nu
                                  mber_of_Files-9;
                                  if (Position<0) Position=0;

```

Listing 6. (Fortsetzung)

STEFAN OSSOWSKI

Public-Domain

STEFAN OSSOWSKI

Ca. 900 Disketten lieferbar:
Fish 1-156, Panorama 1-78, Faug 1-51,
Amicus 1-26, Auge 4000 1-28, Taifun 1-70,
RPD 1-135, Kickstart 1-80, Chiron
Conception 1-79 und viele andere!

Einzeldisk	DM 7.-
ab 10 Stück	DM 6.50
ab 20 Stück	DM 6.-
ab 30 Stück	DM 5.50
ab 50 Stück	DM 5.-
ab 100 Stück	DM 4.70
ab 200 Stück	DM 4.50

Wir kopieren selbstverständlich auf
2DD-Disketten mit doppeltem Verify.

Bei Bestellung von mindestens 10 Disketten
wird die PD-Disk **CLI-Help** - unentbehrlich für
Anfänger und Einsteiger - **kostenlos**
mitgeliefert! Stichwort: CLI

10 % Abo-Rabatt!

5 DM kosten unsere **beiden**
Katalogdisketten mit
Kurzbeschreibung fast aller
Programme in **Deutsch**.
Mit V-Scheck oder in Briefmarken
anfordern!
Garantie: Versand erfolgt am Tag
des Bestelleinganges!

• Taifun • Taifun
Super-PD-Software

Taiun: Auslese der besten PD-Programme
Eigenentwicklung - Exklusivvertrieb

Sonderangebot:

1-30	DM 155.-
1-50	DM 240.-
1-70	DM 330.-

Testen auch Sie unsere Stärken

- Zuverlässigkeit
- Schnelligkeit
- Service

Ihr PD-Spezialist:

STEFAN OSSOWSKI

Veronikastraße 33, 4300 Essen 1
Telefon: 0201/788778


```

304 1a      if (OldPosition!=Position) DisplayFiles(Posi
305 w2      tion,FirstName);
306 p8E      OldPosition=Position;
307 s1B      } while(!(64 & *((UBYTE *)0xbfe001)));
308 yTA      break;
309 u39      }
310 z1      break;
311 nQ      case GADGETUP:
312 lv      ID=SpecialMessage->IAddress->GadgetID;
313 zSA      switch(ID)
314 t7B      {
315 Qu      case DRAWER:
316 2VF      if (!(MyLock=Lock(BufferDrawer,FileInfoBlock)))
317 Q3G      {
318 J4      strcpy(BufferDrawer,CurrentDirectory);
319 mt      DrawerInfo.BufferPos=strlen(BufferDrawer);
320 5E      OnlyRefresh(&GGDrawer,SpecialWindow);
321 BgF      break;
322 JVB      }
323 mD      else UnLock(MyLock);
324 uU      strcpy(CurrentDirectory,BufferDrawer);
325 91F      if (BufferDrawer[strlen(BufferDrawer)-1]!=':' ||
326 5sB      strlen(BufferDrawer)==0)
327 DgF      Depth=1;
328 F8G      else
329 cp      {
330 bv      Depth=2;
331 HkM      Count=0;
332 NtN      while(BufferDrawer[Count])
333 rh      {
334 OtM      if (BufferDrawer[Count]=='/') Depth+
335 PuF      Count++;
336 WQB      }
337 tm      FreeList(FirstName);
338 NW      End=FALSE;
339 uN      break;
340 xR      case FILENAME:
341 IY      case LOAD:
342 eA      strcpy(FullName,CurrentDirectory);
343 jK      if (Depth>1) strcat(FullName,"/");
344 AZ      strcat(FullName,BufferFileName);
345 fy      strcpy(FileInfo,BufferFileName);
346 po      FreeMem(DataPointer,sizeof(CursorData));
347 CQ      FreeMem(FileInfoBlock,sizeof(struct FileInfoBloc
348 bG      k));
349 Yn      CloseWindow(SpecialWindow);
350 cC      return(TRUE);
351 jB      break;
352 JaF      case PARENT:
353 92H      if (Depth)
354 Lm      { FreeList(FirstName);
355 jEF      End=FALSE;
356 foB      GetParentDir();
357 wR      }
358 sm      break;
359 F8      case GETVOLUMES:
360 fW      FreeList(FirstName);
361 kt      End=FALSE;
362 qLA      Depth=0;
363 U99      break;
364 oHD      }
365 PdE      if ((ID>=FILESELECT) && (ID<=FILESELECT+8))
366 WI      {
367 GD      if ((ID-FILESELECT+Position)>=Number_of_Files) break;
368 sLI      CurrentName=MyFindName(Position,ID-FILESELECT
369 odJ      ,FirstName);
370 OU      if (CurrentName->dir==A_DIRECTORY)
371 sJ      {
372 Jw      if (Depth>1) strcat(CurrentDirectory,"/");
373 Cx      Depth++;
374 fm      strcat(CurrentDirectory,CurrentName->En
375 93      tryName);
376 WP      strcpy(BufferDrawer,CurrentDirectory);
377      DrawerInfo.BufferPos=strlen(BufferDrawer);
378      OnlyRefresh(&GGDrawer,SpecialWindow);
379      FreeList(FirstName);
380      End=FALSE;

```

```

377 09      break;
378 6bI      }
379 yuE      if (CurrentName->dir==VOLUME)
380 4XI      {
381 sCJ      struct DeviceList *l;
382 Fn      Root = DOSBase -> dl_Root;
383 y3      DosInfo = toAPTR(Root->rn_Info);
384 pm      DevList = toAPTR(DosInfo->di_DevInfo);
385 Vj      l=CurrentName->HandlerPort;
386 Eo      while(DevList->dl_Next)
387 BeP      {
388 dzQ      if (DevList->dl_Task==1 && DevLi
389 CZ      st->dl_Type==DLT_DEVICE) break;
390 InP      DevList=toAPTR(DevList->dl_Next)
391 LpJ      ;
392 qb      }
393 u4      Depth++;
394 fI      strcpy(CurrentDirectory,toAPTR((BPTR)Dev
395 YJ      List->dl_Name)+1);
396 18      strcat(CurrentDirectory,".");
397 VP      strcpy(BufferDrawer,CurrentDirectory);
398 sl      DrawerInfo.BufferPos=strlen(BufferDrawer);
399 MV      OnlyRefresh(&GGDrawer,SpecialWindow);
400 SxI      FreeList(FirstName);
401 53E      End=FALSE;
402 QtI      break;
403 8ZJ      if (CurrentName->dir==PARENTDIR)
404 cW      {
405 zs      GetParentDir();
406 Tc      FreeList(FirstName);
407 Z4I      End=FALSE;
408 81E      break;
409 XOI      }
410 YZJ      if (FileSelect || (FileNumber!=ID-FILESELECT)
411 Jo      )
412 DF      {
413 sV      FileSelect=FALSE;
414 bk      FileNumber=ID-FILESELECT;
415 hCI      strcpy(BufferFileName,CurrentName->Entr
416 TjE      yName);
417 B9K      OnlyRefresh(&GGFileName,SpecialWindow);
418 Xn      break;
419 tP      }
420 yZ      else {
421 Po      FileSelect=TRUE;
422 uD      strcpy(FullName,CurrentDirectory);
423 43      if (Depth>1) strcat(FullName,"/");
424 Rf      strcat(FullName,BufferFileName);
425 qV      strcpy(FileInfo,BufferFileName);
426 nw      FreeMem(DataPointer,sizeof(CursorData));
427 tOJ      FreeMem(FileInfoBlock,sizeof(struct Fil
428 pyE      eInfoBlock));
429 vQD      CloseWindow(SpecialWindow);
430 rO8      return(TRUE);
431 xS6      break;
432 yT4      }
433 zu2      }
434 OV1      }
435 lW0      }
436 Wn      struct Name *MyFindName(Position,Offset,FirstName)
437 RD1      struct Name *FirstName;
438 Wr      SHORT Position,Offset;
439 lUO      {
440 1e1      struct Name *CurrentName;
441 4M      SHORT i;
442 2k      CurrentName=FirstName;
443 bW      for (i=0;i<Position;i++) CurrentName=CurrentName->NextOne;
444 3o      for (i=Position;i<Position+Offset;i++) CurrentName=Curren
445 By      tName->NextOne;
446 Ch0      return(CurrentName);
447 2p      DisplayFiles(Position,FirstName)

```



```

448 c01 struct Name *FirstName;
449 L1 SHORT Position;
450 Cf0 {
451 gL1 BOOL End=FALSE;
452 H7 SHORT i=0;
453 Ep struct Name *CurrentName;
454 Ew CurrentName=FirstName;
455 AS while (i<Position)
456 I18 {
457 4X9 if (!CurrentName->EntryName)
458 KnD {
459 dgE Position=i;
460 LU break;
461 RxD }
462 529 CurrentName=CurrentName->NextOne;
463 FD i++;
464 U28 }
465 sQ1 SetDrMd(SpecialRastPort,JAM2);
466 Gd for (i=0;i<=8;i++)
467 Tw6 {
468 zX7 SetAPen(SpecialRastPort,0);
469 7D SetSoftStyle(SpecialRastPort,FS_NORMAL,255);
470 rq if (CurrentName->NextOne)
471 X0B {
472 adC strcpy(StringName,CurrentName->EntryName);
473 x0 if (strlen(StringName)>25) StringName[25]=0;
474 aa if (CurrentName->dir==A_DIRECTORY || CurrentName->dir==PARENTDIR)
475 W4G { strcat(StringName," (dir)");
476 ORI SetSoftStyle(SpecialRastPort,FSF_ITALIC,255); }
477 t7C TextFileName.IText=StringName;
478 BD PrintIText(SpecialRastPort,&TextFileName,10,12+i*10);
479 bM RectFill(SpecialRastPort,10+strlen(StringName)*8,10+i*10,260,10+i*10+9);
480 NK CurrentName=CurrentName->NextOne;
481 lGB }
482 mH6 }
483 nI0 }
484 ke ClearDisplay()
485 LE {
486 D11 SetDrMd(SpecialRastPort,JAM2);
487 Iq SetAPen(SpecialRastPort,0);
488 xA RectFill(SpecialRastPort,10,11,260,100);
489 t00 }
490 O2 FreeList(FirstName)
491 J51 struct Name *FirstName;
492 sL0 {
493 jJ1 struct Name *NextOne;
494 KA do {
495 p45 if (FirstName->EntryName)
496 Qf9 Deallocate(&MyMemHeader,FirstName->EntryName,strlen(FirstName->EntryName));
497 5E5 NextOne=FirstName->NextOne;
498 T2 Deallocate(&MyMemHeader,FirstName,sizeof(struct Name));
499 J1 ;
500 5F4 FirstName=NextOne;
501 5a0 } while (NextOne);
502 oF OnlyRefresh(MyGadget,WhatWindow)
503 s31 struct Gadget *MyGadget;
504 Wd struct Window *WhatWindow;
505 5Y0 {
506 M11 struct Gadget *TempGadget2;
507 eJ TempGadget2=MyGadget->NextGadget;
508 S3 MyGadget->NextGadget=0;
509 aH RefreshGadgets(MyGadget,WhatWindow,0);
510 Pt MyGadget->NextGadget=TempGadget2;
511 Fk0 }
512 HS GetParentDir()
513 Dg {
514 wz1 char *MyPtr;
515 Ec BYTE Count;
516 LG Count= strlen(CurrentDirectory);
517 J3 MyPtr=CurrentDirectory+Count-1;
518 T0 if (Depth>2)
519 Jm5 {
520 2j6 while (*MyPtr != '/')
521 LoD {
522 6LE *MyPtr-- = 0;

```

```

523 5z Count--;
524 SxD }
525 y86 *MyPtr=0;
526 U25 }
527 K71 else
528 Sv5 {
529 D56 while (*MyPtr != ':')
530 UxD {
531 lRE if (!Count--) break;
532 GV *MyPtr-- = 0;
533 b6D }
534 c75 }
535 pN1 Depth--;
536 xa strcpy(BufferDrawer,CurrentDirectory);
537 IP OnlyRefresh(&GDrawer,SpecialWindow);
538 gB0 }
(C) 1987 M&T

```

Listing 6. (Schluß)

Programmname: memory.c

Computer: Amiga 500, 1000, 2000 mit Kickstart 1.2

Sprache: C

Compiler: Aztec C V3.6

Aufrufe: cc +l-s +idmp memory.c

Programm : memory.c

```

1 kb0 extern ULONG ChipStart,ChipEnd,ChipPointer,FastStart,FastEnd,FastPointer;
2 E9 extern ULONG start[4],allocated[4],MemoryType[4];
3 xz InitMemory()
4 OT {
5 kP1 ULONG FreeMem;
6 AX Forbid();
7 eo FreeMem=AvailMem(MEMF_CHIP|MEMF_LARGEST)-70000;
8 RF Permit();
9 rr if (!(ChipStart=AllocMem(FreeMem,MEMF_CHIP)))
10 6Z5 {
11 CM7 GoodBye();
12 Ch5 }
13 Y11 ChipEnd=ChipStart+FreeMem-2;
14 3a ChipPointer=ChipStart;
15 Jg Forbid();
16 AM FreeMem=AvailMem(MEMF_FAST|MEMF_LARGEST);
17 aO Permit();
18 yX if (!(FastStart=AllocMem(FreeMem,MEMF_FAST)))
19 gc5 { FastEnd=2;
20 Ie7 FastPointer=0;
21 Mv return;
22 Mr5 }
23 Wc1 FastEnd=FastStart+FreeMem-2;
24 f1 FastPointer=FastStart;
25 Pu0 }
26 GK BOOL InsertTrack(Track,Amount,StartPosition)
27 Vo1 UBYTE Track;
28 IS ULONG Amount;
29 Ir ULONG StartPosition;
30 Qt0 {
31 Wg1 UBYTE Count;
32 34 ULONG Start=start[Track]+StartPosition;
33 Un UWORD *Source,*Dest,*i;
34 Q4 if (MemoryType[Track] == MEMF_CHIP)
35 UD6 { if (Amount > (2 + ChipEnd) - ChipPointer) return(FALSE);
36 5f8 else { i = ChipPointer - Start;
37 aRF ChipPointer += Amount;
38 xz Dest = ChipPointer - 2;
39 mG Source = Start + (ULONG)i - 2;
40 e9D }
41 fA6 }

```

Listing 7. »memory.c«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

42 VI1 else
43 Ex6 { if (Amount > (2 + FastEnd) - FastPointer) return(F
      ALSE);
      else { i = FastPointer - Start;
            FastPointer += Amount;
            Dest = FastPointer - 2;
            Source = Start + (ULONG)i - 2;
            }
      }
44 6W8 }
45 RSF }
46 oC }
47 u0 }
48 mHD }
49 nI6 }
50 b6I for (Count=0;Count<4;Count++)
51 Tz6 { if (Count != Track)
52 FQC { if (start[Count])
53 pHI { if (MemoryType[Track] == MemoryType[Count])
      { if (start[Count] > start[Track])
        { if (start[Count] > start[Track])
          start[Count] += Amount;
        }
      }
    }
  }
54 laO }
55 VXU }
56 uPO }
57 vQI }
58 wRC }
59 xS6 }
60 F2I while (i--)*Dest-- = *Source--;
61 yd return(TRUE);
62 OVO }
63 la Free(Track,Amount,StartPosition)
64 6P1 UBYTE Track;
65 t3 ULONG Amount;
66 cS ULONG StartPosition;
67 lUO {
68 7r1 UBYTE Count;
69 ef ULONG Start=start[Track]+StartPosition;
70 5I UWORD *Source,*Dest,*i;
71 if if (MemoryType[Track] == MEMF_CHIP)
72 XT6 { if (Amount > allocated[Track]) return(FALSE);
73 gG8 else { i = ChipPointer - Start;
74 JCF ChipPointer -= Amount;
75 pv Source = Start;
76 D7 Dest = Start - Amount;
77 FKD }
78 G16 }
79 6t1 else
80 fb6 { if (Amount > allocated[Track]) return(FALSE);
81 h78 else { i = FastPointer - Start;
82 ADF FastPointer -= Amount;
83 x3 Source = Start;
84 LF Dest = Start - Amount;
85 NsD }
86 Ot6 }
87 Ch1 for (Count=0;Count<4;Count++)
88 F06 { if (start[Count])
89 PHC { if (MemoryType[Track] == MemoryType[Count])
90 LAI { if (start[Count] > start[Track])
91 FJP start[Count] -= Amount;
92 UzI }
93 VOC }
94 W16 }
95 G5I while (i--)*Dest++ = *Source++;
96 Hf return TRUE;
97 Z40 }
98 qt ULONG GetMem(Track,Amount)
99 fy1 UBYTE Track;
100 Sc ULONG Amount;
101 Z20 {
102 5M1 ULONG Start;
103 UA if (Amount > (2 + ChipEnd) - ChipPointer)
104 Dw5 { if (Amount > (2 + FastEnd) - FastPointer) return(F
      LSE);
      else { Start = FastPointer;
            FastPointer += Amount;
            MemoryType[Track] = MEMF_FAST;
            }
        }
105 WX7 }
106 QRE }
107 QC }
108 kFC }
109 lG5 }
110 sj1 else { Start = ChipPointer;
111 md8 ChipPointer += Amount;
112 mO MemoryType[Track] = MEMF_CHIP;
113 pK6 }
114 lG1 return(Start);
115 rMO }
(C) 1987 M&T

```

Listing 7. (Schluß)

Programmname: effects.c

Computer: Amiga 500, 1000, 2000 mit Kickstart 1.2

Sprache: C

Compiler: Aztec C V3.6

Aufrufe: cc +l -s +idmp effects.c

Programm : effects.c

```

1 ny0 extern ULONG start[4],end[4],startpos[4],length[4],allocate
d[4],oneshot[4];
2 Vx extern UBYTE Current;
3 83 extern ULONG samplingrate[4],RangeStart[4],RangeEnd[4];
4 dN Reverse(Track)
5 9S1 UBYTE Track;
6 2V0 {
7 jF1 register UBYTE *s,*e;
8 QR register UBYTE swap;
9 o8 s=(UBYTE *) (start[Track]+startpos[Track]);
10 KL e=(UBYTE *) (start[Track]+end[Track]);
11 CZ do
12 8b3 {
13 dm4 swap=*s;
14 N4 *s++=*e;
15 ku *e--=swap;
16 G13 }
17 yR1 while (e>s);
18 In0 }
19 gr Mix(Track)
20 Oh1 UBYTE Track;
21 Hk0 {
22 AE1 register UBYTE *s1,*s2;
23 fp register ULONG e;
24 fY s1=(char *) (start[Current]+startpos[Current]);
25 TH s2=(char *) (start[Track]+startpos[Track]);
26 o9 e=MIN(length[Current],length[Track]);
27 o0 while (e--)*s2++=( ( (*s2 ^ 128) + ( *s1++ ^ 128 ) ) >
> 1 ) ^ 128;
28 Sx0 }
29 GS BOOL Concat(Source,Dest)
30 za1 UBYTE Source,Dest;
31 Ru0 {
32 OE1 if (!(InsertTrack(Dest,length[Source],allocated[Dest]))) r
eturn(FALSE);
33 M1 FastCopy(start[Source]+startpos[Source],start[Dest]+alloca
ted[Dest],length[Source]);
34 Nn allocated[Dest] += length[Source];
35 YD return(TRUE);
36 a50 }
37 qL BOOL Paste(Source,Dest)
38 7i1 UBYTE Source,Dest;
39 Z20 {
40 gD1 if (!(InsertTrack(Dest,length[Source],end[Dest]+1))) retur
n(FALSE);
41 ok FastCopy(start[Source]+startpos[Source],start[Dest]+end[De
st]+1,length[Source]);
42 Vv allocated[Dest] += length[Source];
43 gL return(TRUE);
44 lD0 }
45 nj BOOL InsertRange(Source,Dest)
46 Fq1 UBYTE Source,Dest;
47 ha0 {
48 tT1 if (!(InsertTrack(Dest,length[Source],startpos[Dest]))) re
turn(FALSE);
49 T7 FastCopy(start[Source]+startpos[Source],start[Dest]+startp
os[Dest],length[Source]);
50 d3 allocated[Dest] += length[Source];
51 Sh end[Dest]+=length[Source];
52 Wq startpos[Dest]+=length[Source];
53 Kw if (oneshot[Dest]) oneshot[Dest]+=length[Source];
54 rW return(TRUE);
55 t00 }
56 F3 BOOL Double(Track)
57 zI1 UBYTE Track;
58 sL0 {
59 NB1 char *s=start[Track]+allocated[Track];
60 zk char *e=start[Track];
61 xd UWORD *d;
62 n2 if (!(InsertTrack(Track,allocated[Track],allocated[Track])
)) return FALSE;

```



```

63 Q1 allocated[Track] *=2;
64 yx d=start[Track]+allocated[Track];
65 rX #asm
66 z8A move.l -4(a5),a0
67 Ma move.l -8(a5),a1
68 tm move.l -12(a5),a2
69 Dd0 double: move.b -(a0),d0
70 t1A move.b d0,d1
71 1H asl.w #8,d1
72 v3 move.b d0,d1
73 OD move.w d1,-(a2)
74 rh cmpa.l a0,a1
75 yZ bne.s double
76 sm1 #endasm
77 S4 SoundOFF(Track);
78 PY oneshot[Track] *=2;
79 ed SetStart(Track,startpos[Track]*2);
80 vP SetEnd(Track,end[Track]*2);
81 8w SetOneShot(Track,oneshot[Track]);
82 BV SetStatus(Track);
83 AQ RangeEnd[Track] *=2;
84 5T return TRUE;
85 Ns0 }
86 zD Halve(Track)
87 Tm1 UBYTE Track;
88 Mp0 {
89 87I char *s=start[Track];
90 hd char *d=s;
91 Dn char *e=start[Track]+allocated[Track];
92 Iy0 #asm
93 Q2A move.l -4(a5),a0
94 n1 move.l -8(a5),a1
95 KD move.l -12(a5),a2
96 j70 halve: clr.w d0
97 PJA move.b (a0)+,d0
98 WM move.b (a0)+,d1
99 ma add.w d0,d1
100 JY asr.w #1,d1
101 fJ move.b d0,(a1)+
102 MD cmpa.l a0,a2
103 Xe bne.s halve
104 KEO #endasm
105 Gy1 Free(Track,allocated[Track]/4*2,allocated[Track]);
106 3n allocated[Track] = allocated[Track]/4*2;
107 wY SoundOFF(Track);
108 SH SetStart(Track,startpos[Track]/4*2);
109 Zw SetOneShot(Track,oneshot[Track]/4*2);
110 W1 SetEnd(Track,(end[Track]+1)/4*2-1);
111 qr RangeEnd[Track]=(RangeEnd[Track]+1)/4*2-1;
112 fz SetStatus(Track);
113 pK0 }
(C) 1987 M&T

```

Listing 8. »effects.c«. Bitte mit dem Checksummer (Seite 159) eingeben.

Programmname: request.c

Computer: Amiga 500, 1000, 2000
mit Kickstart 1.2

Sprache: C

Compiler: Aztec C V3.6

Aufrufe: cc -I -s +idmp request.c

Programm : request.c

```

1.4Q0 #define CANCEL 0
2 Sq #define OK 6
3 ww #define TRACK1 1
4.46 #define TRACK2 2
5 CG #define TRACK3 3
6 KQ #define TRACK4 4
7 vO #define CONTINUE 5
8 Pk #define NOBORDER NULL
9 uK struct RequestData
10 8W7 { SHORT Left,Top,Width,Height;

```

```

11 hf9 struct Gadget *FirstGadget;
12 jx struct IntuiText *MyText;
13 J9 SHORT OKLeft,OKTop,CancelLeft,CancelTop; };
14 UDO static struct IntuiText
15 Uk6 InfoText10 = { 1,0,JAM2,12,123,NULL,"for their very us
16 wG InfoText9 = { 1,0,JAM2,40,112,NULL,"Manx Software Sys
17 Ut InfoText8 = { 1,0,JAM2,40,101,NULL," and to
18 fJ InfoText7 = { 1,0,JAM2,32,90,NULL,"for this wonderful
19 Ol InfoText6 = { 1,0,JAM2,40,79,NULL," Commodore Am
20 Uj InfoText5 = { 1,0,JAM2,40,68,NULL," Thanks to
21 YL InfoText4 = { 1,0,JAM2,40,57,NULL," 1987 All rights
22 hc InfoText3 = { 1,0,JAM2,40,46,NULL," 2190 Cuxhave
23 Nm InfoText2 = { 1,0,JAM2,40,35,NULL," Matthias-Claudi
24 ww InfoText1 = { 1,0,JAM2,40,24,NULL," Christian W
25 49 InfoText0 = { 1,0,JAM2,32,13,NULL," This program was
26 9Y EraseText = { 1,0,JAM2,8,13,NULL,"Loading a new sound
27 9B TextOK = { 1,0,JAM2,8,4,NULL,"OK",NULL };
28 XL TextCancel = { 1,0,JAM2,8,4,NULL,"Cancel",NULL };
29 qm Text1 = { 1,0,JAM2,8,4,NULL,"1",NULL };
30 3r Text2 = { 1,0,JAM2,8,4,NULL,"2",NULL };
31 Gw Text3 = { 1,0,JAM2,8,4,NULL,"3",NULL };
32 T1 Text4 = { 1,0,JAM2,8,4,NULL,"4",NULL };
33 Mn TextMemory = { 1,0,JAM2,7,13,NULL,"Sorry, but there's
34 uP File = { 1,0,JAM2,7,13,NULL,"Sorry, but I can't load
35 RT TextCheck = { 1,0,JAM2,7,13,NULL,"Sorry, this track is
36 bp TextContinue = { 1,0,JAM2,7,4,NULL,"Continue",NULL };
37 EM TextSure = { 1,0,JAM2,32,13,NULL,"You really want to
38 8Y ConcatTrack = { 1,0,JAM2,8,13,NULL,"Concat current tr
39 E8 DeleteAll = { 1,0,JAM2,20,13,NULL,"All tracks will be
40 aM DeleteTrack = { 1,0,JAM2,8,13,NULL,"The recorded data
41 Ue MixWave = { 1,0,JAM2,24,13,NULL,"Destination track:",
42 UO CopyTrack = { 1,0,JAM2,8,13,NULL,"Copy current track
43 Hn0 USHORT LinesOK[] = { 0,0,31,0,31,15,0,15,0,0 };
44 td USHORT LinesCancel[] = { 0,0,63,0,63,15,0,15,0,0 };
45 Kz USHORT LinesTrack[] = { 0,0,23,0,23,15,0,15,0,0 };
46 SJ USHORT LinesCont[] = { 0,0,79,0,79,15,0,15,0,0 };
47 dt struct Border BorderCont = { 0,0,1,3,JAM1,5,&LinesCont,NULL
48 r1 struct Border BorderOK = { 0,0,1,3,JAM1,5,&LinesOK,NULL };
49 ky struct Border BorderCancel = { 0,0,1,3,JAM1,5,&LinesCancel,
50 la struct Border BorderTrack = { 0,0,1,3,JAM1,5,&LinesTrack,NU
51 NR struct Gadget GGOK = { NULL,16,25,32,16,GADGHCOMP,RELVERIFY
52 5ON BOOLGADGET,&BorderOK,0,&TextOK,0,0,0
53 Xa0 struct Gadget GGCcancel = { &GGOK,288,25,64,16,GADGHCOMP,REL
54 loR BOOLGADGET,&BorderCancel,0,&Text
55 QK0 struct Gadget GG4 = { NULL,168,25,24,16,GADGHCOMP,RELVERIFY
56 o1M BOOLGADGET,&BorderTrack,0,&Text4,0,0,
57 LA0 struct Gadget GG3 = { &GG4,120,25,24,16,GADGHCOMP,RELVERIFY

```

Listing 9. »request.c«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

58 juM          BOOLGADGET,&BorderTrack,0,&Text3,0,0,
                TRACK3,0};
59 if0 struct Gadget GG2 = { &GG3,68,25,24,16,GADGHCOMP,RELVERIFY,

60 enM          BOOLGADGET,&BorderTrack,0,&Text2,0,0,
                TRACK2,0};
61 Bz0 struct Gadget GG1 = { &GG2,16,25,24,16,GADGHCOMP,RELVERIFY,

62 ZgM          BOOLGADGET,&BorderTrack,0,&Text1,0,0,
                TRACK1,0};
63 MLO struct RequestData MyRequest[] =
64 Mv6          { { 170,25,300,150,0,&InfoText0,16,25,288,25 },
65 OW8          { 136,50,368,50,&GGCancel,&EraseText,16,25,288,25 }
                ,
66 Sk          { 212,50,216,50,&GG1,&CopyTrack,16,25,288,25 },
67 Vs          { 212,50,216,50,&GG1,&ConcatTrack,16,25,288,25 },
68 qs          {
69 JO          { 136,50,368,50,&GGCancel,&DeleteTrack,16,25,288,25
                },
70 A1          { 136,50,368,50,&GGCancel,&DeleteAll,16,25,288,25
                },
71 TE          { 212,50,216,50,&GG1,&MixWave,16,25,288,25 } };
72 h20 struct NewWindow NewRequestWindow = {
73 9X7          0,0,640,234,0,1,
74 IP          CLOSEWINDOW GADGETUP! GADGETDOWN! SIMPLE_REFRESH,
75 JJ          WINDOWCLOSE! ACTIVAT! WINDOWDEPTH! WINDOWDRAG,
76 Qu          NULL,NULL,
77 q5          NULL,
78 Sw          NULL,NULL,
79 xr          640,200,640,200,
80 MQ          WENCHSCREEN };
81 Ds0 struct Window *RequestWindow;
82 X3 UWORD DisplayRequest(Number,Title)
83 O81 UBYTE Number;
84 CL char *Title;
85 Jm0 {
86 u81 struct IntuiMessage *ThisMsg;
87 gc ULONG ID,Class;
88 Ra NewRequestWindow.FirstGadget=MyRequest[Number].FirstGadget
      ;
89 VP NewRequestWindow.LeftEdge=MyRequest[Number].Left;
90 Y3 NewRequestWindow.TopEdge=MyRequest[Number].Top;
91 n8 NewRequestWindow.Width=MyRequest[Number].Width;
92 Vs NewRequestWindow.Height=MyRequest[Number].Height;
93 uq NewRequestWindow.Title=Title;
94 86 GGOK.LeftEdge=MyRequest[Number].OKLeft;
95 Zh GGOK.TopEdge=MyRequest[Number].OKTop;
96 v9 GGCancel.LeftEdge=MyRequest[Number].CancelLeft;
97 ew GGCancel.TopEdge=MyRequest[Number].CancelTop;
98 Jk if (!(RequestWindow=OpenWindow(&NewRequestWindow))) return
      (FALSE);
99 ia PrintIText(RequestWindow->RPort,MyRequest[Number].MyText,
      0,0);
100 Ar FOREVER
101 Z2 {
102 JS2 WaitPort(RequestWindow->UserPort);
103 Yw if (ThisMsg=GetMsg(RequestWindow->UserPort))
104 c56 {
105 2t7 Class=ThisMsg->Class;
106 Et switch(Class)
107 f88 {
108 AM9 case CLOSEWINDOW:
109 lf CloseWindow(RequestWindow);
110 FA return(0);
111 ir break;
112 js break;
113 Oq case GADGETUP:
114 J1 ID=(struct Gadget *) (ThisMsg->IAddress)->GadgetID;
115 qk switch(ID)
116 oHA {
117 toB case CANCEL:
118 Pr case OK:
119 AP case TRACK1:
120 DT case TRACK2:
121 GX case TRACK3:
122 Jb case TRACK4:
123 zt CloseWindow(RequestWindow);
124 Ok return(ID);
125 w5 break;
126 2XA }
127 y79 break;

```

```

128 4Z8 }
129 OS7 ReplyMsg(ThisMsg);
130 6b6 }
131 7c1 }
132 8d0 }
(C) 1987 M&T

```

Listing 9. (Schluß)

Programmname:	myfunctions.c
Computer:	Amiga 500, 1000, 2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec C V3.6
Aufrufe:	cc +l -s +idmp myfunctions.c

Programm : myfunctions.c

```

1 eF0 #define SAMPLING 4
2 HE #define ON TRUE
3 xe #define OFF FALSE
4 7T #define CANCEL 0
5 u2 extern struct NewWindow NewRequestWindow,NewWindow;
6 Wc extern struct Window *RequestWindow,*MyWindow;
7 9d extern struct Gadget GGContinue;
8 LZ extern struct IntuiText TextWindow;
9 N7 USHORT *ChipSleepy,*c1,*c2,*ChipLeft,*ChipRight,*MySpriteData;
10 o1 extern struct FileInfoBlock *FileInfoBlock;
11 oq extern char BufferFileName[],Out_of_Memory[];
12 an extern ULONG RangeStart[4],RangeEnd[4];
13 vb extern APTR DirBuffer;
14 tX extern FastIntHandler(),fast0(),fast1(),fast2(),fast3();
15 92 extern void chip0(),chip1(),chip2(),chip3();
16 vX extern struct Interrupt *Paula1,*OldInterrupt;
17 if void (*fastint[4])() = { fast0,fast1,fast2,fast3 };
18 a6 void (*chipint[4])() = { chip0,chip1,chip2,chip3 };
19 OH struct FastIntVariables
20 h97 { ULONG start;
21 Cf9 ULONG MemCounter;
22 KU ULONG end;
23 Cq APTR CopyBuffer1;
24 Ix APTR CopyBuffer2;
25 Wv APTR RestBuffer;
26 vq UWORD RestLength;
27 BY BOOL *Modus;
28 81 APTR Next; };
29 UYO struct FastIntVariables FastIntVariables[4],RepeatVariables
      [4];
30 Mq extern ULONG NewPosition,OldPosition,regstart,regend,posba
      ckup;
31 hA extern ULONG oneshot[4],startpos[4],class,ChunkID,ChunkLen
      gth,countpos;
32 ye extern UBYTE i,Current,track,hi,current,x,y,currenty[4],Cu
      rrent;
33 NS extern UWORD sample,rest,x1,y1,rate,CurrentInt[4],IntCount
      [4];
34 oO extern UWORD mask[4],AudioBits;
35 Sq extern BOOL DisplayModus,PLAYING,Modus[4];
36 Ax extern ULONG start[4],end[4],length[4],allocated[4],Memory
      Type[4];
37 JQ extern ULONG ChipStart,ChipEnd,ChipPointer,FastStart,FastE
      nd,FastPointer;
38 IR extern ULONG volume[4],period[4],samplingrate[4],pre_perio
      d[4],pre_oneshot[4];
39 tI extern ULONG MyFile;
40 Tw extern FILE *MYFILE;
41 nJ extern struct RastPort *RastPort;
42 II extern ULONG *parameter[6],*parapointer,PotiStart[6],MaxiPa
      rt[6];
43 JR extern char BufferStart[8],BufferEnd[8],BufferPeriod[4],Buf
      ferRate[6];
44 22 extern char BufferOneShot[8],BufferVolume[3],Not_Allocated[
      ];
45 Pe extern char BufferTrackName[4][60];

```



```

46 PC extern struct StringInfo TrackNameInfo;
47 XZ extern struct IntuiText TextFreeChip,TextFreeFast,TextOn,Te
xtOFF,TextCurrent[4];
48 dN extern ULONG FORM_ID,_8SVX_ID,VHDR_ID,BODY_ID;
49 IT struct VHDR { ULONG Length;
50 NeE     ULONG OneShotHiSamples;
51 df     ULONG RepeatPart;
52 h1     ULONG SamplesPerHiCycle;
53 FD     UWORD SamplesPerSecond;
54 z2     UBYTE Octaves;
55 8c     UBYTE Packed;
56 lQ     ULONG Volume; };
57 l10 struct FSound { ULONG length;
58 neG     UWORD samplingrate; };
59 Ly0 struct BODY { ULONG BODY_ID;
60 sJE     ULONG Length; };
61 gl0 struct IFF { ULONG FORM_ID;
62 22D     ULONG Length;
63 K2     ULONG _8SVX_ID;
64 qx     ULONG VHDR_ID;
65 3u     struct VHDR VHDR;
66 Bb     struct BODY BODY; };
67 FK0 extern struct FSound FSound;
68 dL extern struct VHDR VHDR;
69 73 extern struct IFF *IFF;
70 zA extern struct GfxBase *GfxBase;
71 hh extern struct IntuitionBase *IntuitionBase;
72 BY struct Custom *CustomBase=0xdff000;
73 qr extern struct PropInfo MyPotiInfo;
74 jq extern struct Gadget GGMypoti,GGStart,GGEnd,GGVolume,GGRate
,GGPeriod;
75 Ro extern struct Gadget GGTrackName,GGOneShot;
76 y0 extern UBYTE coordinates[];
77 FK char *MyPlane;
78 EK EmptyPort()
79 Dg {
80 gY1 struct IntuiMessage *MyMsg;
81 BD while (MyMsg=GetMsg(MyWindow->UserPort)) ReplyMsg(MyMsg);

82 Kp0 }
83 jM Number_to_String(Buffer,number,len)
84 cn1 char *Buffer;
85 WY ULONG number;
86 Er UBYTE len;
87 Lo0 {
88 2F1 UBYTE count,count1;
89 mq for (count=0;count<=len;count++)
90 Or6 {
91 Bx7     Buffer[len-count]=48+number%10;
92 tq     number = number/10;
93 V06 }
94 ra1 count=0;
95 Yn while ((Buffer[0]==48) && (count<len))
96 Ux8 {
97 Uf9     for (count1=0;count1<=len-count;count1++) Buffer[
count1]=Buffer[count1+1];
98 6S     count++;
99 b68 }
100 c70 }
101 mY ResetTrack(Track)
102 i11 UBYTE Track;
103 b40 {
104 Hc1 startpos[Track]=RangeStart[Track]=0;
105 Tw end[Track]=RangeEnd[Track]=allocated[Track]-1;
106 G9 length[Track]=allocated[Track];
107 kH SetStart(Track,startpos[Track]);
108 Zs SetEnd(Track,end[Track]);
109 5d SetOneShot(Track,pre_oneshot[Track]);
110 3g period[Track]=pre_period[Track];
111 mS SetPeriod(Track,period[Track]);
112 qa DisplayTrack(Track,TRUE);
113 pK0 }
114 HL BYTE Memory(channel,amount)
115 HL1 UBYTE channel;
116 Gw ULONG amount;
117 pIO {
118 9p1 amount = amount / 2 * 2;
119 x8 if (!(start[channel]=GetMem(channel,amount)))
120 sL4 {
121 hk5     SetWindowTitles(MyWindow,Out_of_Memory,-1);
122 Z0     AudioBits &= (mask[channel]^15);
123 5o     SetStatus(channel);

```

```

124 fE     return(FALSE);
125 lW4 }
126 7D1 length[channel]=allocated[channel]=amount;
127 hx end[channel]=amount-1;
128 In RangeStart[channel]=startpos[channel]=0;
129 sj RangeEnd[channel]=end[channel]/608*608+607;
130 zf InitTrack(channel);
131 61 return(TRUE);
132 8d0 }
133 UK InitTrack(Track)
134 EX1 UBYTE Track;
135 7a0 {
136 CR1 Number_to_String(BufferStart,startpos[Track],6);
137 3s OnlyRefresh(&GGStart,MyWindow);
138 vx Number_to_String(BufferEnd,end[Track],6);
139 HQ OnlyRefresh(&GGEnd,MyWindow);
140 G10 }
141 5o ClearTrack(Channel)
142 A11 UBYTE Channel;
143 F10 {
144 771 if (AudioBits & mask[Channel])
145 7w5 { AudioBits &= ( mask[Channel] ^ 15 );
146 Q87     SetStatus(Channel); }
147 s11 if (start[Channel]) Free(Channel,allocated[Channel],alloca
ted[Channel]);
148 xY allocated[Channel]=start[Channel]=end[Channel]=startpos[Ch
annel]=length[Channel]=0;
149 59 period[Channel]=358;
150 RB samplingrate[Channel]=10000;
151 UF oneshot[Channel]=0;
152 wK volume[Channel]=64;
153 la strcpy(BufferTrackName[Channel],"untitled");
154 kD OnlyRefresh(&GGTrackName,MyWindow);
155 5K EmptyDisplay();
156 A9 DisplayTrack(Channel,FALSE);
157 X20 }
158 rK BOOL LoadIFF(ThisFile)
159 Vy {
160 cc1 ULONG Length;
161 uw UBYTE nothing;
162 c7 ULONG ChunkID;
163 9D Read(ThisFile,&ChunkID,4);
164 ZO if (ChunkID != FORM_ID)
165 b44 {
166 G95     Close(ThisFile);
167 Mv     return(FALSE);
168 iD4 }
169 aP1 Read(ThisFile,&Length,4);
170 GK Read(ThisFile,&ChunkID,4);
171 u0 if (ChunkID != _8SVX_ID)
172 iB4 {
173 NG5     Close(ThisFile);
174 T2     return(FALSE);
175 pK4 }
176 rE1 do
177 nG3 {
178 OS4     Read(ThisFile,&ChunkID,4);
179 EJ if (ChunkID == VHDR_ID) Read(ThisFile,&VHDR,sizeof(stru
ct VHDR));
180 8G if ((ChunkID != VHDR_ID) && (ChunkID != BODY_ID) )
181 rK7 {
182 nc8     Read(ThisFile,&Length,4);
183 zb     if (Length % 2) Length+=1;
184 TD     for (i=0;i<Length;i++) Read(ThisFile,&nothing,1);
185 zU7 }
186 OV3 }
187 IK1 while (ChunkID != BODY_ID);
188 2X0 }
189 IV BOOL LoadFile(FileName)
190 y11 char *FileName;
191 lU0 {
192 Bo1 ULONG ThisFile,ThisLock,SeekPosition;
193 Pq if (!(ThisLock=Lock(FileName,ACCESS_READ)))
194 aJ5 { SetWindowTitles(MyWindow,"Sorry, I can't open this f
ile",-1);
195 Ph7     return FALSE; }
196 2j1 Examine(ThisLock,FileInfoBlock);

```

Listing 10. »myfunctions.c«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

197 q7 if (FileInfoBlock->fib_EntryType>0)
198 sM5 { SetWindowTitles(MyWindow,"Can't load a directory!",-
      1);
199 T17 return FALSE; }
200 Df1 if (start[track])
201 Be5 {
202 Jx6 if (DisplayRequest(1,"Warning!")==CANCEL) return FALSE;
      E;
203 9a ClearTrack(track);
204 In5 }
205 Jk1 strcpy(BufferTrackName[track],BufferFileName);
206 8y TrackNameInfo.BufferPos=strlen(BufferFileName);
207 E6 TrackNameInfo.Buffer=BufferTrackName[track];
208 c5 OnlyRefresh(&GGTrackName,MyWindow);
209 kI UnLock(ThisLock);
210 yV ThisFile=Open(FileName,MODE_OLDFILE);
211 iA if (LoadIFF(ThisFile))
212 Mp5 {
213 wH6 Read(ThisFile,&ChunkLength,4);
214 Zm ChunkLength=VHDR.OneShotHiSamples+VHDR.RepeatPart;
215 sm i=(UBYTE)(VHDR.Octaves-1);
216 pw SeekPosition=0;
217 86 while (i)
218 SvD {
219 VqE SeekPosition+=ChunkLength;
220 aF ChunkLength*=2;
221 Zb i--;
222 h2 VHDR.OneShotHiSamples*=2;
223 b6D }
224 iU6 if (!Memory(track,ChunkLength))
225 lnA { Close(ThisFile);
226 uCC return FALSE; }
227 Bi6 SetRate(track,VHDR.SamplesPerSecond);
228 PL SetOneShot(track,VHDR.OneShotHiSamples);
229 8X Seek(ThisFile,SeekPosition,0);
230 0a SetWindowTitles(MyWindow,"Loading IFF File...",-1);
231 56 LoadBody(ThisFile);
232 Tr return TRUE;
233 lG5 }
234 Mt1 ThisFile=Open(FileName,MODE_OLDFILE);
235 mt Read(ThisFile,&FSound,sizeof(struct FSound));
236 i4 if (FileInfoBlock->fib_Size == (FSound.length+6))
237 lE5 {
238 5F6 if (!Memory(track,FSound.length))
239 i5A { ClearTrack(track);
240 SLC Close(ThisFile);
241 9R return FALSE; }
242 nu6 SetRate(track,FSound.samplingrate);
243 yI SetWindowTitles(MyWindow,"Loading Future Sound File..",-1);
244 iJ LoadBody(ThisFile);
245 g4 return TRUE;
246 yT5 }
247 ZS1 Close(ThisFile);
248 a7 ThisFile=Open(FileName,MODE_OLDFILE);
249 iu if (!Memory(track,FileInfoBlock->fib_Size))
250 tG5 { ClearTrack(track);
251 dW7 Close(ThisFile);
252 Kc return FALSE; }
253 XY1 SetWindowTitles(MyWindow,"Loading Raw Data File...",-1);
254 ST LoadBody(ThisFile);
255 qE return TRUE;
256 8d0 }
257 AN SaveIFF(Octaves)
258 HK1 UBYTE Octaves;
259 7a0 {
260 Hf1 UBYTE i;
261 go ULONG BodyLength;
262 3V BodyLength=length[track]/32*32;
263 d2 IFF->VHDR.Length=20;
264 jK if (oneshot[track]>startpos[track])
265 Dg5 {
266 tJ6 IFF->VHDR.OneShotHiSamples=(oneshot[track]-startpos[track])>>(Octaves-1);
      IFF->VHDR.RepeatPart=(end[track]+1-oneshot[track])>>(Octaves-1);
267 HG }
268 Kp5 else
269 Ax1 {
270 iI5 IFF->VHDR.OneShotHiSamples=length[track]>>(Octaves-1);
271 DN6 }

```

```

272 jG IFF->VHDR.RepeatPart=0;
273 Pu5 }
274 jO1 IFF->VHDR.SamplesPerHiCycle=8;
275 5p IFF->VHDR.SamplesPerSecond=samplingrate[track];
276 Pb IFF->VHDR.Octaves=Octaves;
277 y3 IFF->VHDR.Packed=0;
278 Qo IFF->VHDR.Volume=volume[track]*1024;
279 aC IFF->FORM_ID=FORM_ID;
280 GA IFF->_8SVX_ID=_8SVX_ID;
281 hs IFF->VHDR_ID=VHDR_ID;
282 iL IFF->BODY.BODY_ID=BODY_ID;
283 qF IFF->BODY.Length=BodyLength;
284 kH for (i=0;i<(Octaves-1);i++)
285 X06 {
286 u57 BodyLength/=2;
287 Em IFF->BODY.Length+=BodyLength;
288 e96 }
289 oR1 IFF->Length=sizeof(struct IFF)+IFF->BODY.Length-8;
290 cs fwrite(IFF,sizeof(struct IFF),1,MYFILE);
291 hC0 }
292 7l GoodBye()
293 f8 {
294 NB1 Stop();
295 yL if (MyWindow) { ClearMenuStrip(MyWindow);
296 WDH CloseWindow(MyWindow); }
297 sG1 if (GfxBase) CloseLibrary(GfxBase);
298 oY if (IntuitionBase) CloseLibrary(IntuitionBase);
299 FI if (ChipStart) FreeMem(ChipStart,ChipEnd-ChipStart+2);
300 7N if (FastStart) FreeMem(FastStart,FastEnd-FastStart+2);
301 hs if (DirBuffer) FreeMem(DirBuffer,2000);
302 DW if (FastIntVariables[0].CopyBuffer1)
303 pI5 {
304 RH6 for (i=0;i<4;i++)
305 rKB {
306 ODC FreeMem(FastIntVariables[i].CopyBuffer1,64);
307 UK FreeMem(FastIntVariables[i].CopyBuffer2,64);
308 fw FreeMem(FastIntVariables[i].RestBuffer,128);
309 yI FreeMem(RepeatVariables[i].RestBuffer,128);
310 OVb }
311 iW5 }
312 Tk1 if (FileInfoBlock) FreeMem(FileInfoBlock,sizeof(struct FileInfoBlock));
313 Co if (ChipSleepy) FreeMem(ChipSleepy,54);
314 Ry if (ChipLeft) FreeMem(ChipLeft,18);
315 Nd if (ChipRight) FreeMem(ChipRight,18);
316 BG if (MySpriteData) FreeMem(MySpriteData,516);
317 GM exit(0L);
318 8d0 }
319 Ka InitFast(Channel)
320 2a2 UBYTE Channel;
321 7a0 {
322 gw1 UWORD Rest;
323 dD ULONG Start=start[Channel]+startpos[Channel],Length;
324 Q1 UWORD i;
325 Im UBYTE *c1,*c2;
326 BX FastIntVariables[Channel].start=Start;
327 tO FastIntVariables[Channel].MemCounter=Start;
328 pN FastIntVariables[Channel].Modus=&Modus[Channel];
329 Vj Rest = ( length[Channel] % 64 ) + 64;
330 fV if (Rest >= length[Channel])
331 Hk5 {
332 GL6 c1=FastIntVariables[Channel].end=Start;
333 bR FastIntVariables[Channel].RestLength=length[Channel];
334 Ot5 }
335 E11 else
336 Mp5 {
337 jL6 FastIntVariables[Channel].RestLength=Rest;
338 9D c1=FastIntVariables[Channel].end=Start+length[Channel]/64*64 - 64;
339 Ty5 }
340 yj1 c2=FastIntVariables[Channel].RestBuffer;
341 YT for (i=0;i<Rest;i++) *c2++=*c1++;
342 mV if (oneshot[Channel]>startpos[Channel])
343 Tw5 {
344 ZF6 Start=start[Channel]+oneshot[Channel];
345 Hb RepeatVariables[Channel].start=Start;
346 HJ RepeatVariables[Channel].MemCounter=Start;
347 hF RepeatVariables[Channel].Modus=&Modus[Channel];
348 jM Length=end[Channel]+1-oneshot[Channel];
349 YK Rest=( Length % 64 ) + 64;

```



```

350 3C      if (Rest >= Length)
351 b4A      {
352 LOB          c1=RepeatVariables[Channel].end=Start;
353 z0          RepeatVariables[Channel].RestLength=Length;
354 IDA      }
355 YL7      else
356 g9A      {
357 WAB          RepeatVariables[Channel].RestLength=Rest;
358 VJ          c1=RepeatVariables[Channel].end=Start+Length/64*
              64 - 64;
359 nIA      }
360 Hr7      FastIntVariables[Channel].Next=&RepeatVariables[Chan
nel];
361 2P      RepeatVariables[Channel].Next=&RepeatVariables[Chann
el];
362 N1      c2=RepeatVariables[Channel].RestBuffer;
363 up      for (i=0;i<Rest;i++) *c2++=*c1++;
364 sN5      }
365 zG1      else FastIntVariables[Channel].Next=&FastIntVariables[Chan
nel];
366 uP0      }
367 Sm      SoundOn(Channel)
368 oM1      UBYTE Channel;
369 tM0      {
370 QH1      CustomBase->aud[Channel].ac_vol=(UWORD)volume[Channel];
371 ra      CustomBase->aud[Channel].ac_per=(UWORD)period[Channel];
372 wv      *((ULONG *)0x70)=FastIntHandler;
373 UZ      if (MemoryType[Channel] == MEMF_CHIP) Paula1->is_Code=chi
pint[Channel];
374 70      else Paula1->is_Code=fastint[Channel];
375 hA      Paula1->is_Data=&FastIntVariables[Channel];
376 Yy      OldInterrupt=SetIntVector(INTB_AUDIO+Channel,Paula1);
377 Oh      InitFast(Channel);
378 EO      CustomBase->intreq=INTF_SETCLR(mask[Channel]<<7);
379 Eg      CustomBase->intena=INTF_SETCLR(mask[Channel]<<7);
380 mJ      CustomBase->dmacon=DMAF_MASTER DMAF_SETCLR mask[Channel];

381 9e0      }
382 B1      SoundOFF(Channel)
383 3b1      UBYTE Channel;
384 8b0      {
385 hr1      CustomBase->dmacon = mask[Channel];
386 3Z      CustomBase->intena=mask[Channel]<<7;
387 5v      CustomBase->intreq=mask[Channel]<<7;
388 G10      }
389 w3      Play()
390 Eh      {
391 qg1      for (i=0;i<4;i++)
392 GJ6          {
393 F47              if(AudioBits&mask[i]) SoundOn(i);
394 Mr6          }
395 Ns0      }
396 i5      Stop()
397 Lo      {
398 kf1      for (i=0;i<4;i++) SoundOFF(i);
399 Rw0      }
400 Ot      SetStatus(Track)
401 Xq1      UBYTE Track;
402 Qt0      {
403 FZ1      SetDrMd(RastPort,JAM2);
404 7b      if (AudioBits & mask[Track])
405 Tw5          {
406 uP6              SetAPen(RastPort,0);
407 c2              RectFill(RastPort,66+Track*161,14,66+Track*161+24,20)
;
408 pZ              PrintIText(RastPort,&TextOn,69+Track*161,14);
409 A7              if (PLAYING) SoundOn(Track);
410 c75          }
411 SF1      else
412 a35          {
413 QX6              PrintIText(RastPort,&TextOFF,66+Track*161,14);
414 P1              if (PLAYING) SoundOFF(Track);
415 hC5          }
416 ID0      }
417 Ss      WriteBody(Octaves)
418 ru1      UBYTE Octaves;
419 hA0      {
420 Lr1      ULONG Count;
421 6V      char i,DataPart;
422 oh      SHORT sum;
423 7F      char *DataStart;

```

```

424 TD      ULONG DataLength;
425 3I      SetWindowTitles(MyWindow,"Saving track...",-1);
426 45      DataLength=length[track];
427 18      DataPart=1<<(Octaves-1);
428 iJ      DataLength>>=(Octaves-1);
429 SE      while (DataPart)
430 sL8          {
431 U89              DataStart=start[track]+startpos[track];
432 Gh              for (Count=0;Count<DataLength;Count++)
433 vOE                  {
434 WLF                      sum=0;
435 Wy                      for (i=0;i<DataPart;i++) sum += *DataStart+
;
436 Li                          putc((char)(sum/DataPart),MYFILE);
437 3YE                      }
438 zN9              DataPart>>=1;
439 k4              DataLength<<=1;
440 6b8          }
441 tJ1      fclose(MYFILE);
442 8d0      }
443 Ku      LoadBody(ThisFile)
444 RC1      ULONG ThisFile;
445 7a0      {
446 bn1      Read(ThisFile,start[track],length[track]);
447 ng      Close(ThisFile);
448 T7      AudioBits |= mask[track];
449 Fk0      }
450 6K      BOOL AllocationCheck(Channel)
451 9h1      UBYTE Channel;
452 Eh0      {
453 j11          if (!start[Channel])
454 GJ4              {
455 WY5                  for (i=0;i<=5;i++) BufferStart[i]="begin"[i];
456 C1                  OnlyRefresh(&GGStart,MyWindow);
457 Mf                  for (i=0;i<=5;i++) BufferEnd[i]="empty"[i];
458 QZ                  OnlyRefresh(&GGEnd,MyWindow);
459 gt                  Not_Allocated[6]=49+Channel;
460 GI                  SetWindowTitles(MyWindow,Not_Allocated,-1);
461 6f                  return(FALSE);
462 Sx4              }
463 S71          return(TRUE);
464 Uz0      }
465 QW      SetVolume(Number,Volume)
466 ZJ1      UBYTE Number;
467 Wk      ULONG Volume;
468 Ux0      {
469 he1          if (Volume>64 || (!start[Number]))
470 Wz4              {
471 YK5                  Number_to_String(BufferVolume,volume[Number],1);
472 Ey                  OnlyRefresh(&GGVolume,MyWindow);
473 eD                  return;
474 e94              }
475 c31          volume[Number]=Volume;
476 dP          Number_to_String(BufferVolume,volume[Number],1);
477 J3          OnlyRefresh(&GGVolume,MyWindow);
478 7R          CustomBase->aud[Number].ac_vol=(UWORD)volume[Number];
479 jEO      }
480 EG      SetEnd(Number,End)
481 oY1      UBYTE Number;
482 3o      ULONG End;
483 jCO      {
484 O71          if (!AllocationCheck(Number)) return;
485 7R          if ( End<=startpos[Number] ) End=startpos[Number]+1;
486 cX          if ( End>allocated[Number] ) End=end[Number];
487 Ty          if (!DisplayModus) DrawLine(Number,end[Number]);
488 hY          end[Number]=End/2*2+1;
489 VO          if (!DisplayModus) DrawLine(Number,end[Number]);
490 N5          length[Number]=end[Number]-startpos[Number]+1;
491 Yy          Number_to_String(BufferEnd,end[Number],6);
492 y7          OnlyRefresh(&GGEnd,MyWindow);
493 bA          if (oneshot[Number] >= end[Number]) SetOneShot(Number,0);
494 yTO      }
495 ZD      SetOneShot(Number,OneShot)
496 3n1      UBYTE Number;
497 vX      ULONG OneShot;
498 yR0      {
499 dM1          if (!AllocationCheck(Number)) return;
500 Ba          if (OneShot<startpos[Number] || OneShot>=end[Number]) On
eShot=0;

```

Listing 10. (Fortsetzung)


```

501 dx OneShot=OneShot/2*2;
502 3D if (OneShot && !oneshot[Number] && !DisplayModus) DrawLine
      (Number,OneShot);
503 nq if (!OneShot && oneshot[Number] && !DisplayModus) DrawLine
      (Number,oneshot[Number]);
504 uf if (OneShot && oneshot[Number] && !DisplayModus)
      {
505 5Y5 DrawLine(Number,oneshot[Number]);
506 eh6 DrawLine(Number,OneShot);
507 mJ }
508 Ch5 oneshot[Number]=OneShot;
509 8e1 Number_to_String(BufferOneShot,oneshot[Number],6);
510 We OnlyRefresh(&GGOneShot,MyWindow);
511 sZ }
512 G10 }
513 K2 SetStart(Number,Start)
514 L51 UBYTE Number;
515 k1 ULONG Start;
516 GJ0 {
517 ve1 if (!AllocationCheck(Number)) return;
518 DK Start /= 2;
519 z1 Start *= 2;
520 an if (Start>=end[Number]) Start=startpos[Number];
521 4e if (!DisplayModus) DrawLine(Number,startpos[Number]);
522 T3 startpos[Number]=Start;
523 K1 if (!DisplayModus) DrawLine(Number,Start);
524 vd length[Number]=end[Number]-startpos[Number]+1;
525 z0 Number_to_String(BufferStart,startpos[Number],6);
526 K9 OnlyRefresh(&GGStart,MyWindow);
527 KY if (oneshot[Number]<startpos[Number] && oneshot[Number])
      SetOneShot(Number,0);
528 W10 }
529 hR SetPeriod(Number,Period)
530 bL1 UBYTE Number;
531 xo ULONG Period;
532 Wz0 {
533 yJ1 if (Period<124 || Period>999)
      {
534 Y14 {
535 JU5 Number_to_String(BufferPeriod,period[Number],2);
536 AP OnlyRefresh(&GGPeriod,MyWindow);
537 gF return;
538 gB4 }
539 St1 period[Number]=Period;
540 xU samplingrate[Number]=3580000/Period;
541 qI Number_to_String(BufferRate,samplingrate[Number],4 );
542 5H OnlyRefresh(&GGRate,MyWindow);
543 rc Number_to_String(BufferPeriod,period[Number],2);
544 IX OnlyRefresh(&GGPeriod,MyWindow);
545 nw CustomBase->aud[Number].ac_per=(UWORD)period[Number];
546 oJ0 }
547 Z2 SetRate(Number,Rate)
548 td1 UBYTE Number;
549 9t ULONG Rate;
550 oH0 {
551 2m1 if (Rate>28870 || Rate<3583)
      {
552 qJ5 {
553 Su6 Number_to_String(BufferRate,samplingrate[Number],4);
554 HT OnlyRefresh(&GGRate,MyWindow);
555 yX return;
556 yT5 }
557 VN1 samplingrate[Number]=Rate;
558 vv period[Number]=3580000/samplingrate[Number];
559 7s Number_to_String(BufferPeriod,period[Number],2);
560 Yn OnlyRefresh(&GGPeriod,MyWindow);
561 Ac Number_to_String(BufferRate,samplingrate[Number],4 );
562 Pb OnlyRefresh(&GGRate,MyWindow);
563 5E CustomBase->aud[Number].ac_per=(UWORD)period[Number];
564 6b0 }
565 Sy RefreshOneShot(Number)
566 Bv1 UBYTE Number;
567 5Y0 {
568 sF1 ULONG SetValue;
569 t5 current=current[Number];
570 ms MaxiPart[5]=MaxiPart[0]=MaxiPart[1]=allocated[Number]-1;
571 4M parapointer=parameter[current];
572 3n SetValue = parapointer[Number];
573 lp SetValue -= PotiStart[current];
574 Tw SetValue *= 511;
575 Wv if (MaxiPart[current]) SetValue /= MaxiPart[current];
576 wH else SetValue=0;
577 Gw SetValue=SetValue<<7;
578 Bz MyPotiInfo.HorizPot=(UWORD)SetValue;
579 fU OnlyRefresh(&GGMyPoti,MyWindow);

```

```

580 Mr0 }
581 LK EmptyDisplay()
582 Kn {
583 lL1 strcpy(BufferEnd,"empty");
584 Sb OnlyRefresh(&GGEnd,MyWindow);
585 Cw strcpy(BufferStart,"begin");
586 I7 OnlyRefresh(&GGStart,MyWindow);
587 y8 strcpy(BufferRate,"10000");
588 p1 OnlyRefresh(&GGRate,MyWindow);
589 Ba strcpy(BufferPeriod,"358");
590 2H OnlyRefresh(&GGPeriod,MyWindow);
591 GH strcpy(BufferVolume,"64");
592 Au OnlyRefresh(&GGVolume,MyWindow);
593 Jp strcpy(BufferOneShot,"0");
594 Du OnlyRefresh(&GGOneShot,MyWindow);
595 6b0 }
596 Mg SelectTrack(Number)
597 gQ1 UBYTE Number;
598 a30 {
599 MV1 SetDrMd(RastPort,COMPLEMENT);
600 Cf RectFill(RastPort,9+Current*161,11,49+Current*161,23);
601 GJ Current=Number;
602 Eh RectFill(RastPort,9+Current*161,11,49+Current*161,23);
603 WI if (start[Number])
      {
604 g95 SetStart(Number,startpos[Number]);
605 DF6 SetEnd(Number,end[Number]);
606 lB Number_to_String(BufferVolume,volume[Number],1);
607 kW OnlyRefresh(&GGVolume,MyWindow);
608 QA SetOneShot(Number,oneshot[Number]);
609 vK SetPeriod(Number,period[Number]);
610 v5 }
611 rM5 }
612 HJ1 else EmptyDisplay();
613 Mf TrackNameInfo.BufferPos=strlen(BufferTrackName[Number]);
614 8h TrackNameInfo.Buffer=BufferTrackName[Number];
615 Be OnlyRefresh(&GGTrackName,MyWindow);
616 YO DisplayTrack(Number,FALSE);
617 xS0 }
618 cp InversGadget(MyGadget)
619 kv1 struct Gadget *MyGadget;
620 wP0 {
621 Mp1 SHORT x=MyGadget->LeftEdge;
622 TP SHORT y=MyGadget->TopEdge;
623 kt SetDrMd(RastPort,COMPLEMENT);
624 Qv SetAPen(RastPort,0);
625 oK RectFill(RastPort,x,y,x+MyGadget->Width-1,y+MyGadget->Height-1);
626 6b0 }
(C) 1987 M&T

```

Listing 10. (Schluß)

Programmname: record.c

Computer: Amiga 500, 1000, 2000 mit Kickstart 1.2

Sprache: C

Compiler: Aztec C V3.6

Aufrufe: cc +l -s +idmp record.c

Programm : record.c

```

1 OU0 extern UWORD rate;
2 TI extern ULONG regstart,regend,period[4],start[4],startpos[4],end[4];
3 17 extern ULONG allocated[4],length[4],MemoryType[4],volume[4];
4 Xm extern UBYTE Current;
5 oe extern ULONG ChipStart,ChipEnd,ChipPointer;
6 s0 extern ULONG FastStart,FastEnd,FastPointer;
7 ak extern USHORT *ChipSleepy;
8 DR extern struct Window *MyWindow;
9 If extern void MyRec0(),MyRec1(),MyTest0(),MyTest1(),MyAd0(),MyAd1();

```

Listing 11. »record.c«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

10 qf extern void MyWait0(),MyWait1();
11 EW extern struct DosLibrary *DOSBase;
12 gd void (*Recording[2])() = { &MyRec0,&MyRec1 };
13 li void (*Adjusting[2])() = { &MyAd0,&MyAd1 };
14 Sr void (*Testing[2])() = { &MyTest0,&MyTest1 };
15 oc void (*Waiting[2])() = { &MyWait0,&MyWait1 };
16 Q6 USHORT Hardware=0;
17 t8 ULONG Record(start,end,Text)
18 ki1 ULONG start,end;
19 hX char *Text;
20 GJO {
21 hN1 rate=(UWORD)period[Current]/2;
22 8o regstart=start;
23 qU regend=end;
24 Cs4 #asm
25 7ZB movem.l d0-d6/a0-a6,-(a7)
26 SO move.l _regstart,-(a7)
27 A3 move.l _regend,-(a7)
28 Mf move.w _rate,$dff0a6
29 HH move.l $4,a6
30 9T jsr -132(a6)
31 gC lea _Recording,a0
32 E3 clr.l d0
33 Qv move.w _Hardware,d0
34 F7 asl.w #2,d0
35 mu move.l (a0,d0),$70
36 Xi move.l (a7)+,a5
37 Vf move.l (a7)+,a4
38 5y move.w #64,$dff0a8
39 zJ move.b #00000000,$bfe301
40 ct move.b $bfd200,d0
41 WW andi.b #011111010,d0
42 NO ori.b #00000110,d0
43 lm move.b d0,$bfd200
44 a2 lea $bfe101,a0
45 Vv lea $bfd000,a1
46 7F lea $dff0aa,a2
47 PA lea $dff09c,a3
48 Mn move.w $dff01c,d2
49 HZ and.w #0111111111111111,d2
50 Xd move.w d2,$dff09a
51 oX move.w $dff002,d3
52 Qe and.w #0111111111111111,d3
53 9Z move.w d3,$dff096
54 mx clr.w d4
55 Pn bset #2,(a1)
56 w9 bclr #2,(a1)
57 5J move.w #11000000010000000,$dff09a
58 EJ move.w #10000000010000000,$dff09c
59 X80 click: andi.b #64,$bfe001
60 AyB bne click
61 T60 newpc: move.w #00000000010000000,$dff09a
62 FJB move.w #00000000010000000,$dff09c
63 ZR0 click1: andi.b #64,$bfe001
64 NOB beq.s click1
65 2A move.l a4,d7
66 Gm or.w #10000000000000000,d2
67 ou move.w d2,$dff09a
68 Oq or.w #10000000000000000,d3
69 Pp move.w d3,$dff096
70 Bb jsr -138(a6)
71 t0 movem.l (a7)+,d0-d6/a0-a6
72 g6 bclr #0,d7
73 op move.l d7,_regend
74 7v bra.s exit
75 AR public _MyRec0
76 pm0 _MyRec0: move.w #00000000010000000,(a3)
77 kRB cmpa.l a4,a5
78 7g bne.s sample0
79 qc lea newpc,a0
80 BW move.l a0,2(a7)
81 TG rte
82 yg0 sample0: move.b (a0),d0
83 rFB bset #2,(a1)
84 Ob bclr #2,(a1)
85 PH eori.b #128,d0
86 jL move.b d0,d1
87 NM asl.w #8,d1
88 lN move.b d0,d1
89 WH move.w d1,(a2)
90 YO move.b d0,(a4)+
91 dQ rte

```

```

92 T1 public _MyRec1
93 970 _MyRec1: move.w #00000000010000000,(a3)
94 liB cmpa.l a4,a5
95 Q0 bne.s sample1
96 7t lea newpc,a0
97 Sn move.l a0,2(a7)
98 kX rte
99 KW0 sample1: clr.b (a0)
100 ozB move.b (a0),d0
101 fX eori.b #128,d0
102 zb move.b d0,d1
103 dc asl.w #8,d1
104 ld move.b d0,d1
105 mX move.w d1,(a2)
106 oG move.b d0,(a4)+
107 tg rte
108 sn0 exit: move.l d7,_regend
109 PJ4 #endasm
110 KJ1 return(regend);
111 nIO }
112 Gd RecordTrack(Current,Text)
113 xr1 UBYTE Current;
114 E4 char *Text;
115 nG0 {
116 391 register UWORD *s1,*s2,*e;
117 OE if (start[Current])
118 KA5 Record(start[Current]+startpos[Current],start[Current]
+end[Current]+1,Text);
119 kX1 else
120 pD3 { if (FastPointer <= FastEnd && (FastEnd+2-FastPointer)
> (ChipEnd+2-ChipPointer))
121 XL9 { start[Current]=FastPointer;
122 jrB FastPointer=Record(FastPointer,FastEnd+2,Text);
123 9t if (FastPointer == start[Current])
124 zPG { start[Current]=0;
125 XbI return; }
126 2EB allocated[Current]=FastPointer-start[Current];
127 y4 if ((e=allocated[Current]) < ChipEnd+2-ChipPointer)
128 EQF { FastPointer = start[Current];
129 PmH s2=start[Current]=ChipPointer;
130 MC s1=FastPointer;
131 ek ChipPointer += allocated[Current];
132 bI SetPointer(MyWindow,ChipSleepy,26,16,0,0);
133 lW FastCopy(FastPointer,start[Current],allocated[Current]);
134 V5 ClearPointer(MyWindow);
135 v3 goto chipalloc; }
136 WYB MemoryType[Current]=MEMF_FAST;
137 lB length[Current]=allocated[Current];
138 M1 startpos[Current]=0;
139 is end[Current]=allocated[Current]-1;
140 y2 InitTrack(Current); }
141 6t else
142 cBG { if (ChipPointer <= ChipEnd)
143 AcM { start[Current]=ChipPointer;
144 fsO ChipPointer=Record(ChipPointer,ChipEnd+2,Text);
145 hW if (ChipPointer == start[Current])
146 lLS { start[Current]=0;
147 txU return; }
148 arO allocated[Current]=ChipPointer-start[Current];
149 Zt0 chipalloc: MemoryType[Current]=MEMF_CHIP;
150 E00 length[Current]=allocated[Current];
151 ZE startpos[Current]=0;
152 v5 end[Current]=allocated[Current]-1;
153 Bk InitTrack(Current);
154 UzM }
155 kPI else SetWindowTitles(MyWindow,"Out of memory!",-1);
156 wIH }
157 X2D }
158 Y30 }
159 GJ Adjust()
160 Wz {
161 oM1 rate=period[Current]/2;
162 F3 Stop();

```

Listing 11. (Fortsetzung)


```

163 R73      # asm
164 RuB      movem.l  d0-d7/a0-a6,-(a7)
165 c5        move.w  _rate,$dff0a6
166 C4        lea      Block,a1
167 QK        move.l  #0,d0
168 hf2      set:     move.w  #8,(a1,d0)
169 k2B       add.w   #2,d0
170 qa        cmp.w   #512,d0
171 Mu        bne.s   set
172 CM        move    #0,Block
173 xJ        move    #0f00,Block+254
174 8W        move    #0f00,Block+256
175 dd        move.l  $4,a6
176 Vp        jsr     -132(a6)
177 ze        lea      _Adjusting,a0
178 aP        clr.l   d0
179 mH        move.w  _Hardware,d0
180 bT        asl.w   #2,d0
181 8G        move.l  (a0,d0),$70
182 5f        lea      Block,a4
183 QJ        move.w  #64,$dff0a8
184 K4        move.b  #00000000,$bfe301
185 xE        move.b  $bfd200,d0
186 rr        andi.b  #11111010,d0
187 iJ        ori.b   #00000110,d0
188 M7        move.b  d0,$bfd200
189 vN        lea      $bfe101,a0
190 qG        lea      $bfd000,a1
191 Sa        lea      $dff0aa,a2
192 kV        lea      $dff09c,a3
193 GJ        lea      $dff180,a5
194 i9        move.w  $dff01c,d2
195 dI        move.w  #0000000010000000,d3
196 AN        move.w  #1,d4
197 wO        move.l  #2,d5
198 wC        move.l  #128,d6
199 KW        move.l  #8,d7
200 iO        and.w   #0111111111111111,d2
201 y4        move.w  d2,$dff09a
202 Qe        move.w  #1100000010000000,$dff09a
203 Ze        move.w  #1000000010000000,$dff09c
204 yuO      click5:  andi.b  #64,$bfe001
205 XzB      bne      click5
206 NZ        move.w  #0000000010000000,$dff09a
207 ae        move.w  #0000000010000000,$dff09c
208 Y4        or.w    #1000000000000000,d2
209 6C        move.w  d2,$dff09a
210 2xO      click4:  andi.b  #64,$bfe001
211 2TB      beq.s   click4
212 Tt       jsr     -138(a6)
213 ou        bra.s   exit2
214 mn        public  _MyAd0
215 PTO      _MyAd0: move.w  d3,(a3)
216 GNB      clr.w   d0
217 hs        move.b  (a0),d0
218 Qf        bset    d5,(a1)
219 x1        bclr    d5,(a1)
220 x9        eor.b   d6,d0
221 uW        move.b  d0,d1
222 oa        asl.w   d7,d1
223 wY        move.b  d0,d1
224 hS        move.w  d1,(a2)
225 gO        asl.w   d4,d0
226 p1        move.w  (a4,d0),(a5)
227 pe        rte
228 63        public  _MyAd1
229 fkO      _MyAd1: move.w  d3,(a3)
230 UbB      clr.w   d0
231 b8        clr.b   (a0)
232 w7        move.b  (a0),d0
233 AM        eor.b   d6,d0
234 7J        move.b  d0,d1
235 iN        asl.w   d7,d1
236 91        move.b  d0,d1
237 uF        move.w  d1,(a2)
238 tB        asl.w   d4,d0
239 2y        move.w  (a4,d0),(a5)
240 2p        rte
241 AW        global  Block,512
242 Ih3      exit2:  movem.l  (a7)+,d0-d7/a0-a6
243 ZT        #endasm
244 wRO      }
245 AR      Test()

```

```

246 uN      {
247 Ck1      rate=period[Current]/2;
248 dR      Stop();
249 pV4      # asm
250 kCB      movem.l  d0-d6/a0-a6,-(a7)
251 xG        move.w  _rate,$dff0a6
252 ss        move.l  $4,a6
253 k4        jsr     -132(a6)
254 um        lea      _Testing,a0
255 pe        clr.l   d0
256 lW        move.w  _Hardware,d0
257 q1        asl.w   #2,d0
258 NV        move.l  (a0,d0),$70
259 eX        move.w  #64,$dff0a8
260 YI        move.b  #00000000,$bfe301
261 BS        move.b  $bfd200,d0
262 55        andi.b  #11111010,d0
263 wx        ori.b   #00000110,d0
264 aL        move.b  d0,$bfd200
265 9b        lea      $bfe101,a0
266 4U        lea      $bfd000,a1
267 go        lea      $dff0aa,a2
268 yJ        lea      $dff09c,a3
269 vM        move.w  $dff01c,d2
270 q8        and.w   #0111111111111111,d2
271 6C        move.w  d2,$dff09a
272 Ym        move.w  #1100000010000000,$dff09a
273 hm        move.w  #1000000010000000,$dff09c
274 OtO      click2:  andi.b  #64,$bfe001
275 N1B      bne      click2
276 Vh        move.w  #0000000010000000,$dff09a
277 im        move.w  #0000000010000000,$dff09c
278 gC        or.w    #1000000000000000,d2
279 EK        move.w  d2,$dff09a
280 82O      click3:  andi.b  #64,$bfe001
281 4ZB      beq.s   click3
282 b1        jsr     -138(a6)
283 JQ        movem.l  (a7)+,d0-d6/a0-a6
284 sx        bra.s   exit1
285 Cp        public  _MyTest0
286 dEO      _MyTest0: move.w  #0000000010000000,(a3)
287 pOB      move.b  (a0),d0
288 AY        bset    #2,(a1)
289 hu        bclr    #2,(a1)
290 ia        eori.b  #128,d0
291 2e        move.b  d0,d1
292 gf        asl.w   #8,d1
293 4g        move.b  d0,d1
294 pa        move.w  d1,(a2)
295 v1        rte
296 Q4        public  _MyTest1
297 sUO      _MyTest1: move.w  #0000000010000000,(a3)
298 gDB      clr.b   (a0)
299 iC        move.b  (a0),d0
300 lW        eor.b   #128,d0
301 Co        move.b  d0,d1
302 qp        asl.w   #8,d1
303 Eq        move.b  d0,d1
304 zk        move.w  d1,(a2)
305 5s        rte
306 lQO      exit1:  move.l  d7,_regend
307 bV4      #endasm
308 yTO      }
309 ZN      WaitSound()
310 wP      {
311 5S1      Forbid();
312 M7       Disable();
313 rXO      # asm
314 xKA        lea      $bfe101,a0
315 TJ        lea      _Waiting,a2
316 k1        clr.l   d0
317 CW        move.w  _Hardware,d0
318 O3        asl.w   #2,d0
319 eh        move.l  (a2,d0),a1
320 Eh        move.b  #00000000,$bfe301
321 3K        move.b  $bfd200,d0
322 Tn        andi.b  #11111010,d0
323 Je        ori.b   #00000110,d0
324 eJ        move.b  d0,$bfd200
325 sUO      wait1:  clr.w   d0
326 s5A      jsr     (a1)
327 lu        andi.b  #127,d0
328 u1        asr.b   #2,d0

```



```

329 zM      beq.s    wait1
330 m0      cmp.b    #31,d0
331 10      beq.s    wait1
332 ze      bra.s    quit
333 NC      public   _MyWait0
334 DG0     _MyWait0: ori.b    #00000100,$bfd000
335 IHA      andi.b   #11111011,$bfd000
336 tA0     wait:    andi.b   #00000001,$bfd000
337 r5A     beq      wait
338 3p      move.b   (a0),d0
339 XY      rts
340 WM      public   _MyWait1
341 KM0     _MyWait1: clr.b    (a0)
342 7tA     move.b   (a0),d0
343 bc      rts
344 Ho0     quit:
345 D7      #endasm
346 th1     Permit();
347 oh      Enable();
348 c70     }
(C) 1987 M&T

```

Listing 11. (Schluß)

Programmname: myints.asm

Computer: Amiga 500, 1000, 2000
mit Kickstart 1.2

Sprache: Assembler

Assembler: Aztec-Assembler V3.6

Aufrufe: as myints.asm

Programm : myints.asm

```

1 3aE      public   _FastIntHandler
2 090     _FastIntHandler:
3 4eE      move.l   d0/a0-a1/a5,-(a7)
4 Yz      move.w   $dff01e,d0
5 jd      and.w    $dff01c,d0
6 VQ      move.l   $4,a1
7 jZ      btst     #7,d0
8 Vg      beq.s    cont0
9 nV      move.l   172(a1),a5
10 ue     jmp      (a5)
11 AM0     cont0:   btst     #8,d0
12 eqE     beq.s    cont1
13 zk      move.l   184(a1),a5
14 yi      jmp      (a5)
15 QU0     cont1:   btst     #9,d0
16 nOE     beq.s    cont2
17 Bz      move.l   196(a1),a5
18 2m      jmp      (a5)
19 se0     cont2:   btst     #10,d0
20 mHE     beq.s    fuck
21 9m      move.l   208(a1),a5
22 6q      jmp      (a5)
23 mv0     fuck:    movem.l   (a7)+,d0/a0-a1/a5
24 YLE     rts
25 tz      public   _fast0
26 SA0     _fast0   and.w    #0000000010000000,d0
27 GME     move.w   d0,$dff09c
28 D20     again0:  move.l   168(a1),a1
29 8OE     move.l   4(a1),a5
30 qR      cmp.l    8(a1),a5
31 72      beq.s    endreached0
32 4b      bcc.s    checkloop0
33 eo      move.w   #31,d0
34 OI      move.l   12(a1),a0
35 H40     copy0:   move.w   (a5)+,(a0)+
36 GkE     dbra     d0,copy0
37 aw      move.l   a5,4(a1)
38 SM      move.l   12(a1),a0
39 HD      move.l   a0,$dff0a0
40 Xb      move.w   #32,$dff0a4
41 Mw      move.l   16(a1),12(a1)
42 Rc      move.l   a0,16(a1)

```

```

43 cb      bra.s    end_int0
44 m10     endreached0: move.l   20(a1),$dff0a0
45 RGE     clr.l    d0
46 FL      move.w   24(a1),d0
47 uD      add.l    d0,4(a1)
48 pm      asr.w    #1,d0
49 PO      move.w   d0,$dff0a4
50 j1      bra.s    end_int0
51 yEO     checkloop0: move.l   26(a1),a5
52 OgE     tst.w    (a5)
53 Dg      beq.s    noloop0
54 VI      move.l   (a1),4(a1)
55 B0      move.l   30(a1),a5
56 JE      move.l   $4,a1
57 7T      move.l   a5,168(a1)
58 fN      bra.s    again0
59 o60     noloop0:  lea      StopIt0(pc),a0
60 lKE     move.l   4,a1
61 WC      move.l   a0,172(a1)
62 Iy0     end_int0 movem.l   (a7)+,d0/a0-a1/a5
63 ByE     rts
64 nJ      public   _chip0
65 C40     _chip0   and.w    #0000000010000000,d0
66 tzE     move.w   d0,$dff09c
67 270     again00:  move.l   168(a1),a1
68 11E     move.l   4(a1),a5
69 T4      cmp.l    8(a1),a5
70 GR      beq.s    endreached00
71 Fk      bcc.s    checkloop00
72 yz      move.l   a5,$dff0a0
73 4a      add.l    #64,4(a1)
74 59      move.w   #32,$dff0a4
75 AP      bra.s    end_int00
76 S60     endreached00: move.l   a5,$dff0a0
77 xmE     clr.l    d0
78 lr      move.w   24(a1),d0
79 QJ      add.l    d0,4(a1)
80 LI      asr.w    #1,d0
81 vu      move.w   d0,$dff0a4
82 HW      bra.s    end_int00
83 Qw0     checkloop00: move.l   26(a1),a5
84 WCE     tst.w    (a5)
85 VE      beq.s    noloop00
86 lo      move.l   (a1),4(a1)
87 hW      move.l   30(a1),a5
88 pk      move.l   $4,a1
89 dz      move.l   a5,168(a1)
90 hf      bra.s    again00
91 mK0     noloop00:  lea      StopIt0(pc),a0
92 HqE     move.l   4,a1
93 21      move.l   a0,172(a1)
94 620     end_int00 movem.l   (a7)+,d0/a0-a1/a5
95 hUE     rts
96 1U0     StopIt0:  and.w    #0000000010000000,d0
97 OUE     move.w   d0,$dff09c
98 FJ      move.w   d0,$dff09a
99 QW      move.w   d0,$dff09c
100 ru     asr.w    #7,d0
101 pC      move.w   d0,$dff096
102 pJ      movem.l   (a7)+,d0/a0-a1/a5
103 pc      rts
104 GI      public   _fast1
105 kT0     _fast1   and.w    #0000000010000000,d0
106 XdE     move.w   d0,$dff09c
107 S20     again1:  move.l   180(a1),a1
108 PFE     move.l   4(a1),a5
109 71      cmp.l    8(a1),a5
110 UL      beq.s    endreached1
111 Qy      bcc.s    checkloop1
112 v5      move.w   #31,d0
113 fZ      move.l   12(a1),a0
114 eN0     copy1:   move.w   (a5)+,(a0)+
115 a5E     dbra     d0,copy1
116 rD      move.l   a5,4(a1)
117 jd      move.l   12(a1),a0
118 cZ      move.l   a0,$dff0b0
119 ty      move.w   #32,$dff0b4
120 dD      move.l   16(a1),12(a1)
121 it      move.l   a0,16(a1)
122 ww      bra.s    end_int1

```

Listing 12. »myints.asm«. Bitte mit dem Checksummer (Seite 159) eingeben.


```

123 750 endreached1: move.l 20(a1),$dff0b0
124 1XE clr.l d0
125 Wc move.w 24(a1),d0
126 BU add.l d0,4(a1)
127 63 asr.w #1,d0
128 kk move.w d0,$dff0b4
129 33 bra.s end_int1
130 LX0 checkloop1: move.l 26(a1),a5
131 HxE tst.w (a5)
132 W0 beq.s noloop1
133 mZ move.l (a1),4(a1)
134 SH move.l 30(a1),a5
135 aV move.l $4,a1
136 mg move.l a5,180(a1)
137 2g bra.s again1
138 ETO noloop1: lea StopIt1(pc),a0
139 2bE move.l 4,a1
140 4d move.l a0,184(a1)
141 dK0 end_int1: movem.l (a7)+,d0/a0-a1/a5
142 SFE rte
143 A2 public _chip1
144 UNO _chip1 and.w #00000000100000000,d0
145 AGE move.w d0,$dff09c
146 KB0 again11: move.l 180(a1),a1
147 2IE move.l 4(a1),a5
148 KL cmp.l 8(a1),a5
149 fn beq.s endreached11
150 h9 bcc.s checkloop11
151 JL move.l a5,$dff0b0
152 Lr add.l #64,4(a1)
153 RW move.w #32,$dff0b4
154 Yp bra.s end_int11
155 rY0 endreached11: move.l a5,$dff0b0
156 E3E clr.l d0
157 28 move.w 24(a1),d0
158 h0 add.l d0,4(a1)
159 cZ asr.w #1,d0
160 GG move.w d0,$dff0b4
161 fw bra.s end_int11
162 pIO checkloop11: move.l 26(a1),a5
163 nTE tst.w (a5)
164 re beq.s noloop11
165 I5 move.l (a1),4(a1)
166 yn move.l 30(a1),a5
167 61 move.l $4,a1
168 IC move.l a5,180(a1)
169 61 bra.s again11
170 Gm0 noloop11: lea StopIt1(pc),a0
171 Y7E move.l 4,a1
172 a9 move.l a0,184(a1)
173 WU0 end_int11: movem.l (a7)+,d0/a0-a1/a5
174 y1E rte
175 Ko0 StopIt1: and.w #00000000100000000,d0
176 f1E move.w d0,$dff09c
177 Wa move.w d0,$dff09a
178 7A asr.w #7,d0
179 5S move.w d0,$dff096
180 5Z movem.l (a7)+,d0/a0-a1/a5
181 5s rte
182 ca public _fast2
183 110 _fast2 and.w #00000000100000000,d0
184 nTE move.w d0,$dff09c
185 uTO again2: move.l 192(a1),a1
186 fvE move.l 4(a1),a5
187 Ny cmp.l 8(a1),a5
188 qd beq.s endreached2
189 1K bcc.s checkloop2
190 BL move.w #31,d0
191 vp move.l 12(a1),a0
192 OF0 copy2: move.w (a5)+,(a0)+
193 tPE dbra d0,copy2
194 7T move.l a5,4(a1)
195 zt move.l 12(a1),a0
196 wu move.l a0,$dff0c0
197 EK move.w #32,$dff0c4
198 tT move.l 16(a1),12(a1)
199 y9 move.l a0,16(a1)
200 FG bra.s end_int2
201 RRO endreached2: move.l 20(a1),$dff0c0
202 ynE clr.l d0
203 ms move.w 24(a1),d0
204 Rk add.l d0,4(a1)

```

```

205 MJ asr.w #1,d0
206 45 move.w d0,$dff0c4
207 MN bra.s end_int2
208 hp0 checkloop2: move.l 26(a1),a5
209 XDE tst.w (a5)
210 oJ beq.s noloop2
211 2p move.l (a1),4(a1)
212 1X move.l 30(a1),a5
213 q1 move.l $4,a1
214 J6 move.l a5,192(a1)
215 Oy bra.s again2
216 dp0 noloop2: lea StopIt2(pc),a0
217 IrE move.l 4,a1
218 b3 move.l a0,196(a1)
219 xF0 end_int2: movem.l (a7)+,d0/a0-a1/a5
220 iVE rte
221 WK public _chip2
222 lF0 _chip2 and.w #00000000100000000,d0
223 QWE move.w d0,$dff09c
224 pg0 again22: move.l 192(a1),a1
225 IYE move.l 4(a1),a5
226 Ob cmp.l 8(a1),a5
227 38 beq.s endreached22
228 8X bcc.s checkloop22
229 dg move.l a5,$dff0c0
230 b7 add.l #64,4(a1)
231 ms move.w #32,$dff0c4
232 vE bra.s end_int22
233 Fz0 endreached22: move.l a5,$dff0c0
234 UJE clr.l d0
235 IO move.w 24(a1),d0
236 xG add.l d0,4(a1)
237 sp asr.w #1,d0
238 ab move.w d0,$dff0c4
239 2L bra.s end_int22
240 Dd0 checkloop22: move.l 26(a1),a5
241 3JE tst.w (a5)
242 Cz beq.s noloop22
243 YL move.l (a1),4(a1)
244 E3 move.l 30(a1),a5
245 MH move.l $4,a1
246 pc move.l a5,192(a1)
247 UM bra.s again22
248 jD0 noloop22: lea StopIt2(pc),a0
249 oNE move.l 4,a1
250 7Z move.l a0,196(a1)
251 vv0 end_int22: movem.l (a7)+,d0/a0-a1/a5
252 E1E rte
253 e70 StopIt2: and.w #00000000100000000,d0
254 v1E move.w d0,$dff09c
255 mq move.w d0,$dff09a
256 NQ asr.w #7,d0
257 Li move.w d0,$dff096
258 Lp movem.l (a7)+,d0/a0-a1/a5
259 L8 rte
260 ys public _fast3
261 I30 _fast3 and.w #00000000100000000,d0
262 39E move.w d0,$dff09c
263 Tg0 again3: move.l 204(a1),a1
264 vBE move.l 4(a1),a5
265 dE cmp.l 8(a1),a5
266 Cv beq.s endreached3
267 6g bcc.s checkloop3
268 Rb move.w #31,d0
269 B5 move.l 12(a1),a0
270 Mx0 copy3: move.w (a5)+,(a0)+
271 CJE dbra d0,copy3
272 NJ move.l a5,4(a1)
273 F9 move.l 12(a1),a0
274 GF move.l a0,$dff0d0
275 Zg move.w #32,$dff0d4
276 9j move.l 16(a1),12(a1)
277 EP move.l a0,16(a1)
278 Ya bra.s end_int3
279 ln0 endreached3: move.l 20(a1),$dff0d0
280 E3E clr.l d0
281 28 move.w 24(a1),d0
282 h0 add.l d0,4(a1)
283 cZ asr.w #1,d0
284 OQ move.w d0,$dff0d4
285 fh bra.s end_int3
286 370 checkloop3: move.l 26(a1),a5

```



```

287 nTE      tst.w      (a5)
288 6c      beq.s      noloop3
289 I5      move.l     (a1),4(a1)
290 yn      move.l     30(a1),a5
291 61      move.l     $4,a1
292 6d      move.l     a5,204(a1)
293 kG      bra.s      again3
294 2B0 noloop3: lea      StopIt3(pc),a0
295 Y7E      move.l     4,a1
296 0a      move.l     a0,208(a1)
297 H00 end_int3: movem.l (a7)+,d0/a0-a1/a5
298 y1E      rte
299 sc      public     _chip3
300 2x0 _chip3 and.w     #0000010000000000,d0
301 gmE      move.w     d0,$dff09c
302 Rx0 again3: move.l     204(a1),a1
303 YoE      move.l     4(a1),a5
304 Gr      cmp.l      8(a1),a5
305 RT      beq.s      endreached33
306 Zv      bcc.s      checkloop33
307 x1      move.l     a5,$dff0d0
308 rN      add.l      #64,4(a1)
309 7E      move.w     #32,$dff0d4
310 Id      bra.s      end_int33
311 dQ0 endreached33: move.l a5,$dff0d0
312 kZE      clr.l      d0
313 Ye      move.w     24(a1),d0
314 DW      add.l      d0,4(a1)
315 85      asr.w      #1,d0
316 uw      move.w     d0,$dff0d4
317 Pk      bra.s      end_int33
318 by0 checkloop33: move.l 26(a1),a5
319 JzE      tst.w      (a5)
320 XM      beq.s      noloop33
321 ob      move.l     (a1),4(a1)
322 UJ      move.l     30(a1),a5
323 cX      move.l     $4,a1

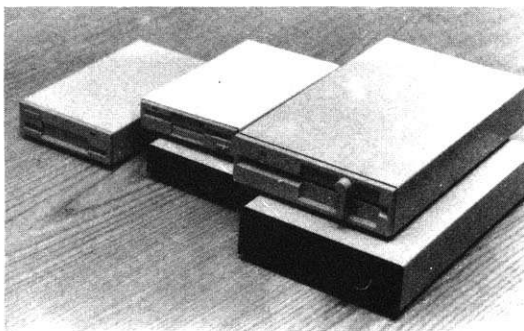
```

```

324 e9      move.l     a5,204(a1)
325 sh      bra.s      again33
326 0e0 noloop33: lea      StopIt3(pc),a0
327 4dE      move.l     4,a1
328 u6      move.l     a0,208(a1)
329 KM0 end_int33: movem.l (a7)+,d0/a0-a1/a5
330 UHE      rte
331 uQ0 StopIt3: and.w     #0000010000000000,d0
332 BHE      move.w     d0,$dff09c
333 26      move.w     d0,$dff09a
334 dg      asr.w      #7,d0
335 by      move.w     d0,$dff096
336 b5      movem.l     (a7)+,d0/a0-a1/a5
337 b0      rte
338 PE      public     _FastCopy
339 Ba2 _FastCopy: movem.l a0/a1/d0,-(a7)
340 qSE      move.l     24(a7),d0
341 L3      lsr.l      #1,d0
342 RH      move.l     20(a7),a1
343 uK      move.l     16(a7),a0
344 bg3 FastLoop: move.w (a0)+,(a1)+
345 YSE      subq.l     #1,d0
346 TC      bne      FastLoop
347 uF      movem.l     (a7)+,a0/a1/d0
348 gh      rte
349 1W      public     _CheckPort
350 1F1 _CheckPort: move.l 4(sp),a0
351 onE      move.l     $14(a0),a0
352 KL      clr.l      d0
353 Dv      tst.l      (a0)
354 FR      bne.s      nomsg
355 Ro      addq.l     #1,d0
356 XV3 nomsg:  rts
(C) 1987 M&T

```

Listing 12. (Schluß)



PROFILAUFWERKE für Ihren AMIGA

2 Jahre Garantie, 14 Tage Umtauschrecht, professionelle Leiterplatten, fast alle ICs gesockelt, Bedienungsanleitung, auf Wunsch vollständiges Manual mit allen Daten zu den Laufwerken lieferbar, 2tägiger Liefer-Rhythmus.

Für alle Laufwerke gilt:

- voll kompatibel zur vorhandenen Soft- und Hardware,
- komplett anschlussfertig,
- amigafarbenes Metallgehäuse,
- abschaltbar (intelligente Abschaltung),
- Kapazität 880 KB,
- korrekte LED-Ansteuerung,
- erkennen Disk-Change,
- kein separates Netzteil nötig (Stromversorgung über AMIGA)
- an alle AMIGA-Modelle anschließbar.

Für unsere 5.25"-Laufwerke gilt zusätzlich:

- alle umschaltbar 40/80 Tracks

Alle Laufwerke sind auch mit Busdurchführung lieferbar und sind dann mit einer automatischen Laufwerkserkennung ausgestattet, so daß beim Anschluß eines weiteren Laufwerkes an unser Laufwerk, das Fremdlaufwerk auf die nächsthöhere Laufwerksadresse als unser Laufwerk gesetzt wird. Aufpreis: 25,- DM

SDN 3.5" - 1037 A 249,-

- zusätzlich: - Superslimline, nur 25,4 mm hoch
- nur noch 5V Spannungsversorgung
- sehr niedriger Stromverbrauch

SDN 3.5" - 1036 A 269,-

- zusätzlich: - extrem robuste Mechanik
- Standardbauhöhe 32 mm

SDN 3.5" Digital - 1037 A 289,-

- zusätzlich: - durchgeführter Bus bis df3: mit automat. Laufwerkserkennung
- Digitale Trackanzeige mit Helligkeitsregulierung

SDN 5.25" - TEAC FD 55 FR 299,-

- zusätzlich: - schwarze Frontblende
- unformatiert 1 MB Kapazität

SDN 5.25" - NEC 1157C 309,-

- zusätzlich: - helle Frontblende
- Diskettenauswurf durch Feder
- unformatiert 1,67 MB Kapazität

SDN 5.25" Digital - 1157C 339,-

- zusätzlich: - durchgeführter Bus bis df3: mit automat. Laufwerkserkennung
- Digitale Trackanzeige mit Helligkeitsregulierung

SDN 3.5" intern 219,-

- für Einbau in A2000
- komplett mit Einbauanleitung und Montagematerial
- helle Frontblende

Rohlaufwerke

(unmodifiziert, ohne Gehäuse u. Kabel):

NEC 1036A	195,-
NEC 1037A	195,-
NEC 1157C	229,-
TEAC FD 55FR	229,-
Gehäuse (NEC 1036, 1037)	19,-
Gehäuse (NEC 1157, TEAC FD 55)	22,-
AMIGA 2000 & 1084	2350,-
XT-Karte	890,-
AT-Karte	auf Anfrage
NEC P2200	879,-
NEC P6	1199,-
Star LC10	588,-
Star LC10 Color	750,-
Eizo Flexscan	1499,-
Mitsubishi EUM-1471A	1398,-
Festplatte 30 MB - 5.25"	849,-
- für A2000 intern	
Festplatte 20 MB - 3.5"	949,-
- für A2000 intern	
Festplatte 30 MB - 3.5"	1049,-
- für A2000 intern	
Festplatte 30 MB	949,-
- für A500/1000 extern	
Golem 2MB für A1000	a. Anfr.
Profex 2MB für A500	a. Anfr.
Bootselektor	19,-
Farbband NEC P6	17,-
Farbband NEC P2200	17,-

WIR FÜHREN GÜNSTIG UND SCHNELL REPARATUREN AN ALLEN AMIGA-MODELLEN AUS.

Stalter Computerbedarf · Gartenstr. 17 · 6670 St. Ingbert · Tel. 06894/35231

Veränderung des CLI

Will man den Amiga programmieren, muß man sich an das CLI gewöhnen. Neben den vielen Vorteilen, die CLI-Fenster bieten, gibt es aber auch einige Nachteile. Mit InitCli3 (Listing 1) kann das Fenster nach eigenen Wünschen verändert werden.

Es erscheint wenig sinnvoll, im CLI, das ja eine Programmierumgebung darstellt, vier Farben zur Verfügung zu haben. Vor allem, wenn man weiß, daß man dadurch 20 KByte Speicher mehr benötigt als bei zweifarbiger Darstellung. Dieses Problem kann mit InitCli3 leicht beseitigt werden, wenn man das Programm folgendermaßen aufruft:

initcli -S

Durch diesen Befehl wird eine der zwei Bit-Ebenen ausgeschaltet, was zur Folge hat, daß nur noch zwei Farben dargestellt werden. Will man später zur vierfarbigen Darstellung zurückkehren, genügt die Eingabe von:

initcli -s

Wer den Monochrom-Modus benutzen will und keinen Schalter an seinem Monitor besitzt, kann dies auch softwaremäßig tun. Mit **initcli -C**

wird dieser Modus eingeschaltet. Der Aufruf, um wieder die voreingestellten Farben aus Preferences zu benutzen, lautet:

initcli -c

Ein weiterer interessanter Befehl von InitCli dient dazu, das CLI-Fenster auf die maximale Größe zu bringen. Das heißt, daß nach **initcli -m**

das Fenster den gesamten Bildschirm belegt. Einen Befehl zur Wiederherstellung der alten Größe gibt es jedoch nicht, aber dies läßt sich dann mit der Maus erledigen.

Sehr ärgerlich ist es, wenn Text mit 80 Zeichen pro Zeile dargestellt werden soll, aber wegen des Fensterrahmens nicht alle Zeichen in eine Zeile passen. Dies umgeht man mit dem Aufruf:

initcli -B

Nach diesem Befehl wird der Rahmen des CLI-Fensters ausgeschaltet. Dadurch stehen für Text 80 Spalten und 32 Zeilen zur Verfügung. In diesem Modus kann das Fenster weder verschoben noch in der Größe verändert werden. Die Vorder- und Hintergrundschalter funktionieren aber, obwohl unsichtbar, immer noch. Um wieder ein Fenster mit Rahmen zu erhalten, ruft man InitCli3 einfach so auf:

initcli -b

Mit »Preferences« kann man Voreinstellungen für Workbench und CLI vornehmen. Noch mehr Parameter lassen sich mit »InitCli3« verändern.

Mit »Preferences« kann man zwar einstellen, ob der Workbenchscreen im Interlace-Modus (320 x 512 Punkte) arbeiten soll, aber diese Umstellung wird erst nach dem nächsten Booten des Amiga vorgenommen. InitCli3 stellt sofort nach dem Aufruf

initcli -l

den Interlace-Modus ein und mit

initcli -i

wieder aus.

Eine weitere interessante, bis jetzt noch nicht genutzte, Fähigkeit des Amiga ist es, verschiedene Zeichensätze auch in einem CLI-Fenster zu benutzen. Der Aufruf muß hierfür so aussehen:

initcli -F/<Größe>

Für muß ein Zeichensatzname eingegeben werden, der im Directory »Fonts« existiert. Der Name muß unmittelbar hinter dem »F« stehen. Mit Größe gibt man die gewünschte Zeichengröße an, die natürlich in einem Subdirectory mit dem Fontnamen stehen muß. Um wieder den normalen System-Zeichensatz zu benutzen, lautet der Aufruf:

initcli -f

Die letzte Funktion dient dazu, den ursprünglichen Zustand wiederherzustellen:

initcli -d

Danach sieht das Fenster aus wie vor allen Veränderungen.

Sollte man die Parameter vergessen, erhält man mit

initcli ?

eine kurze Anleitung mit allen Parametern.

Um nun das Fenster ohne Rahmen und mit maximaler Größe darzustellen, muß man aber nicht InitCli3 zweimal aufrufen. Es können nämlich mehrere Parameter, durch Leerzeichen getrennt, angegeben werden, beispielsweise:

initcli -B -S

Die Reihenfolge der Parameter spielt dabei keine Rolle.

InitCli3 ist durch all seine Fähigkeiten ein Programm, daß sicher immer wieder angewandt wird. (Torsten Jürgeleit/rs)

Programmname:	InitCli3
Computer:	Amiga 500, 1000, 2000 mit Kickstart 1.2
Sprache:	C
Compiler:	Aztec C Version 3.40A
Aufrufe:	cc InitCli3 -s In InitCli3.c -lc

```
1 0s3  /*****
2 ih   /*      InitCLI  Version 1.3  05/87      von Torsten Ju
3 4s   /*
4 eu   /*      Inklusive einer verbesserten Version von SOLOBITMAP (
5 6u   /*
6 yb   /*      (Version fuer Aztec C Compiler 3.4a und KickSta
7 Uy   /*      rt 1.2) */
8 Of0  /*****
9 mr   #include <exec/exec.h>
10 hq  #include <devices/console.h>
11 yA  #include <graphics/gfxbase.h>
12 wu  #include <graphics/display.h>
13 bq  #include <graphics/text.h>
14     #include <intuition/intuition.h>
```

```
14 Cz  #include <libraries/dos.h>
15 86  #include <libraries/dosextens.h>
16 Vm  #include <libraries/diskfont.h>
17 rV  #include <functions.h>
18 Re  #include <stdio.h>
19 v2  #define VERSION      "1.3"
20 d0  #define REV          0L
21 S4  #define NAMESIZE     40
22 jg  #define PSIZE        200L
23 BE  #define EOS          '\0'
24 XI  #define SYSTEMGADGETS ( WINDOWDRAG | WINDOWSIZING )
25 3D  #define BUF_NAME     "Border"
26 MF  /* wichtige Deklarationen und Definitionen von globalen Var
iablen */
27 95  struct GfxBase      *GfxBase      = NULL;
28 kw  struct ViewPort    *vp;
29 Ja  struct ColorMap    *cm;
30 70  struct BitMap       *b;
31 fk  struct IntuitionBase *IntuitionBase = NULL;
32 YV  struct Preferences *p;
33 44  struct Window       *w          = NULL;
34 o8  struct DiskFontBase *DiskFontBase = NULL;
35 yW  struct MsgPort iorp = { { 0, 0, NT_MSGPORT, 0, 0 }, 0, -1, 0
36 MTN ,
37 BCP { (struct Node *)&iorp.mp_MsgList.lh
Tail, 0,
(struct Node *)&iorp.mp_MsgList.lh
Head, 0, 0 } };
38 in0 struct IOStdReq ior = { { { 0, 0, 0, 0, 0 }, &iorp, 0 }, 0 };
39 ud  UWORD mono_cm[] = { 0x000, 0x0f0, 0x000, 0x000 }; /* Farbwe
```



```

rte fuer Mono- *
40 JLO * chrom-Darstellung : *
41 hHO * schwarz, gruen, *
42 7h * schwarz, schwarz * /
43 fq0 VOID exit_program(), print_usage (), solobitmap_off(), solo
    bitmap_on(),
44 ZA5    monochrom(), default_colors(), max_window(), border_on
    (), border_off(),
45 Jo    exit_border(), interlace_on(), interlace_off(), new_fo
    nt(),
46 8P    default_font(), get_font();
47 iDO struct Window *get_window();
48 x4 SHORT check_flags();
49 QxF /* Haupt-Programm */
50 NFO VOID
51 DM main(argc, argv)
52 2L3    SHORT argc;
53 xZ    BYTE *argv[];
54 oHO {
55 KL3    REGISTER SHORT i;
56 1A    if (!(GfxBase = (struct GfxBase *)
57 mkJ        OpenLibrary("graphics.library", REV))) e
    xit_program(20);
58 Dx3    if (!(IntuitionBase = (struct IntuitionBase *)
59 dOJ        OpenLibrary("intuition.library", REV)))
    exit_program(21);
60 ed3    if (!(w = get_window())) exit_program(22);
61 ja    vp = &w->WScreen->ViewPort;
62 uR    b = &w->WScreen->BitMap;
63 bs    if (argc == 1) {
64 g96        printf("Flags fehlen !\n\n");
65 eS        exit_program(23);
66 4Z3    }
67 Sm    if (*argv[1] == '?') print_usage();
68 Dy    for (i=1; i<argc; i++) {
69 yU6        if (*argv[i] != '-') {
70 Ea9            printf("Vermisse '-' vor dem Flag '%c' !\n\n", *(a
    rgv[i]+1));
71 rv6        } else {
72 MG9            switch(*(argv[i]+1)) {
73 c5C                case 's' :
74 7FF                    solobitmap_off();
75 8H                        break;
76 74C                case 'S' :
77 36F                    solobitmap_on();
78 BK                        break;
79 w9C                case 'c' :
80 4mF                    default_colors();
81 EN                        break;
82 R8C                case 'C' :
83 VIF                    monochrom();
84 HQ                        break;
85 WtC                case 'm' :
86 wxF                    max_window();
87 KT                        break;
88 2EC                case 'b' :
89 IXF                    border_on();
90 NW                        break;
91 XDC                case 'B' :
92 23F                    max_window();
93 Yn                    border_off();
94 Ra                        break;
95 UnC                case 'i' :
96 1YF                    interlace_off();
97 Ud                        break;
98 zmC                case 'I' :
99 PUF                    interlace_on();
100 Xg                        break;
101 RhC                case 'f' :
102 T9F                    default_font();
103 aj                        break;
104 wgC                case 'F' :
105 AwF                    new_font(argv[i] + 2);
106 dm                        break;
107 RfC                case 'd' :
108 bqF                    border_on();
109 go                    solobitmap_off();
110 YG                    default_colors();
111 LM                    max_window();
112 Ho                    interlace_off();
113 eK                    default_font();
114 lu                        break;
115 ztC                default :
116 X2F                    printf("Unbekanntes Flag '%c' !\n\n", *(arg

```

```

v[i]+1));
break;
117 ox
118 uP9    }
119 vQ6    }
120 wR3    }
121 hy    exit_program(0);
122 yTO }
123 xHF /* Programm verlassen */
124 ZR0 VOID
125 il    exit_program(error)
126 Ye3    SHORT error;
127 zS0 {
128 a43    if (ior.io_Device) {
129 2U6        if (iorp.mp_SigBit != -1) {
130 rt9            FreeSignal(iorp.mp_SigBit);
131 7c6        }
132 BO            CloseDevice(&ior);
133 9e3    }
134 Os    if (GfxBase) CloseLibrary(GfxBase);
135 Ju    if (IntuitionBase) CloseLibrary(IntuitionBase);
136 MT    if (DiskfontBase) CloseLibrary(DiskfontBase);
137 LD    exit(error);
138 EJ0 }
139 geF /* Bedienungshinweise */
140 ph0 VOID
141 ly    print_usage()
142 Eh {
143 2L3    printf("\n\x9b1;33mInitCLI\x9b0;3;31m Version %s 05/87"
    , VERSION);
144 R1    printf("\x9b0m von \x9b4mTorsten Juergeleit\x9b0m\n\n"
    );
145 kv    printf("\x9b7;42m Eingabeformat : \x9b0;40m\n\n");
146 DQ    printf(" InitCLI [-s] [-S] [-c] [-C] [-m] [-b] [-B]
    [-i] [-I] [-f]");
147 k8    printf(" [-Ffont/size]\n\n");
148 aj    printf(" [-d]\n\n");
149 sG    printf("\x9b7;42m Bedeutung der Flags : \x9b0;40m\n\n");
150 Y6    printf(" -s = SoloBitmap ausschalten\n\n"
    );
151 Op    printf(" -S = SoloBitmap einschalten\n\n"
    );
152 Nb    printf(" -c = Farben aus Preferences ein
    schalten\n\n");
153 ut    printf(" -C = Monochrom-Modus einschalte
    n\n\n");
154 Do    printf(" -m = Maximale Window-Groesse\n\n"
    );
155 fD    printf(" -b = Window-Rahmen einschalten\
    n\n");
156 Lu    printf(" -B = Window-Rahmen ausschalten\
    n\n");
157 VV    printf(" -i = Interlace Modus ausschalte
    n\n\n");
158 xE    printf(" -I = Interlace Modus einschalte
    n\n\n");
159 nO    printf(" -f = Standard ROM-Zeichensatz e
    inschalten\n\n");
160 Tk    printf(" -F[font]/[size] = Neuen Zeichensatz von Disk
    laden\n\n");
161 ea    printf(" -d = Standardeinstellungen eins
    chalten\n\n\n");
162 Md    exit_program(0);
163 d80 }
164 6gF /* Task mit dazugehoerigem Window suchen -
    *
165 nrG * entnommen aus 'SetFont 1.1' von Commodore
    -AMIGA */
166 Lv0 struct Window *
167 Fj    get_window()
168 e7 {
169 4V3    struct MsgPort *con;
170 mD    struct StandardPacket *packet = NULL;
171 hm    struct InfoData *id = NULL;
172 X9    struct Window *window = NULL;
173 vB    if ((OpenDevice("console.device", -1L, &ior, OL)) != NUL
    L) exit_program(24);
174 YF    if ((iorp.mp_SigBit = AllocSignal(-1L)) < 0) exit_progr
    am(25);
175 cn    iorp.mp_SigTask = (struct Task *)FindTask(NULL);
176 Eq    if (iorp.mp_SigTask->tc_Node.ln_Type == NT_PROCESS) {

```

Listing 1. Der Quellcode von InitCli3 für den Aztec-C-Compiler V3.40A und Kickstart 1.2. Bitte mit dem Checksummer (Seite 159) eingeben.


```

177 KJ6      if (con = (struct MsgPort *)
178 qaG        ((struct Process *)iorp.mp_SigTask)->pr_Co
179 219        nsoleTask) {
180 IQM        if (packet = (struct StandardPacket *)
181 caC          AllocMem((ULONG)sizeof(*packet), MEMF_
182 hQL          _CLEAR)) {
183 UHF        if (id = (struct InfoData *)
184 sM          AllocMem((ULONG)sizeof(*id), MEMF_CLEA
185 X1          R)) {
186 M1          packet->sp_Msg.mn_Node.ln_Name = (BYTE *)&
187 wz          packet->sp_Pkt;
188 UF          packet->sp_Pkt.dp_Link = (struct Message *)
189 vu          &(packet->sp_Pkt);
190 b7          packet->sp_Pkt.dp_Port = &iorp;
191 kb          packet->sp_Pkt.dp_Type = ACTION_DISK_INFO;
192 bbC        packet->sp_Pkt.dp_Arg1 = ((ULONG)id) >> 2
193 hX          ; /* BPTR !@#$$%&! */
194 8d9        PutMsg(con, packet);
195 9e6        WaitPort(&iorp);
196 Af3        window = (struct Window *) (id->id_VolumeNod
197 UB          e);
198 JI          FreeMem(id, (ULONG)sizeof(*id));
199 Di0        }
200 kWF        FreeMem(packet, (ULONG)sizeof(*packet));
201 og0 VOID    }
202 Co          solobitmap_on()
203 Dg {
204 AF3      if (b->Depth == 2) {
205 Fm6        b->Depth = 1;
206 c2        FreeRaster(b->Planes[1], 640L, (ULONG)vp->DHeight);
207 b1        RemakeDisplay();
208 Mr3      }
209 Ns0 }
210 fuF        /* Zweite Bitplane einschalten */
211 yq0 VOID
212 2j          solobitmap_off()
213 Nq {
214 1R3      UWORD cm[4];
215 xB          PLANEPTR planeptr;
216 JN          if (b->Depth == 1) {
217 fp6          if (planeptr = AllocRaster(640L, (ULONG)vp->DHeight)
218 JC9            ) {
219 Yd            b->Depth = 2;
220 Of            b->Planes[1] = planeptr;
221 pw            BitClear(planeptr, (ULONG)RASSIZE(640, vp->DHeight
222 1l            t), 0L);
223 8sQ            RemakeDisplay();
224 mTC            if (p = GetPrefs((struct Preferences *)
225 wf              AllocMem(PSIZE, MEMF_CLEAR), PSIZ
226 91              E)) {
227 It              cm[0] = (UWORD)GetRGB4(vp->ColorMap, 0L);
228 EB              cm[1] = (UWORD)GetRGB4(vp->ColorMap, 1L);
229 bo              cm[2] = p->color2;
230 7S              cm[3] = p->color3;
231 dI              FreeMem(p, PSIZE);
232 kF9              FreeColorMap(vp->ColorMap);
233 1G6              vp->ColorMap = GetColorMap(4L);
234 mH3              LoadRGB4(vp, cm, 4L);
235 nIO          }
236 nLF        }
237 OGO VOID    /* Monochrom-Modus einschalten */
238 h3          monochrom()
239 nG {
240 mz3      FreeColorMap(vp->ColorMap);
241 ka          vp->ColorMap = GetColorMap((ULONG)(2 * b->Depth));
242 WN          LoadRGB4(vp, mono_cm, (ULONG)(2 * b->Depth));
243 VQ0 }
244 A3F        /* Urspruengliche Farben aus Preferences ein
245 WOO VOID    schalten */
246 R1          default_colors()
247 vO {
248 6U3      UWORD cm[4];

```

```

249 JC      if (p = GetPrefs((struct Preferences *)
250 ZJK        AllocMem(PSIZE, MEMF_CLEAR), PSIZE)) {
251 In6        cm[0] = p->color0;
252 Ry          cm[1] = p->color1;
253 a9          cm[2] = p->color2;
254 jK          cm[3] = p->color3;
255 fc          FreeMem(p, PSIZE);
256 2F          FreeColorMap(vp->ColorMap);
257 Oq          vp->ColorMap = GetColorMap((ULONG)(2 * b->Depth));
258 Wg          LoadRGB4(vp, cm, (ULONG)(2 * b->Depth));
259 Bg3      }
260 Ch0 }
261 z6F        /* Window auf Maximalgroesse bringen */
262 nf0 VOID
263 3d          max_window()
264 Cf {
265 Of3      ULONG deltax, delay;
266 S4          deltax = - (ULONG)w->LeftEdge;
267 oV          delay = - (ULONG)w->TopEdge;
268 aA          MoveWindow(w, deltax, delay);
269 dC          deltax = (ULONG)w->WScreen->Width - (ULONG)w->Width;
270 kh          delay = (ULONG)w->WScreen->Height - (ULONG)w->Height
271 5R          ;
272 t6          SizeWindow(w, deltax, delay);
273 Pu0        Delay(10L); /* warten, bis Window vergroessert
274 SbF        wurde */
275 Os0 VOID    /* Window-Rahmen einschalten */
276 Oo          border_on()
277 Ps {
278 5L3      BYTE *pointer_buf;
279 YU          LONG *pointer;
280 aA          pointer_buf = w->UserData;
281 bA          pointer = (LONG *) (pointer_buf + 6);
282 Ug          if (!strcmp(pointer_buf, BUF_NAME)) {
283 pL6          fprintf(stdout, "\x0c\x1b\x1b\n"); /* Bildschirm loesch
284 1Td          en und */
285 Rw6        /* Console-Device re
286 zk          initialisieren */
287 1H          w->UserData = NULL;
288 Ql          w->FirstGadget->NextGadget->NextGadget = (struct G
289 fJ          adget *)pointer;
290 9C          w->Title = (UBYTE *)++pointer;
291 nJ          w->Flags |= SYSTEMGADGETS; /* System-Gadgets e
292 iD3        inschalten */
293 jEO        w->Flags &= ~BORDERLESS; /* Rahmen einschalt
294 yvF        en */
295 KCO VOID    RefreshWindowFrame(w);
296 5t          FreeMem(pointer_buf, 14L);
297 jC {
298 1f3      BYTE *pointer_buf;
299 y8          LONG *pointer;
300 xw          BYTE columns[4], lines[4];
301 2G          if (w->UserData == NULL) {
302 QR6          max_window();
303 RJ          if (! (pointer_buf = (BYTE *)AllocMem(14L, MEMF_CLEAR)
304 nP6          ))
305 2S6          exit_program(26);
306 r1          strcpy(pointer_buf, BUF_NAME);
307 1K          pointer = (LONG *) (pointer_buf + 6);
308 Ay          *pointer = (LONG)w->FirstGadget->NextGadget->Nex
309 HB          tGadget;
310 em          *++pointer = (LONG)w->Title;
311 IT          w->UserData = pointer_buf;
312 nU          w->FirstGadget->NextGadget->NextGadget = NULL;
313 a4          w->Flags |= SYSTEMGADGETS; /* System-Gad
314 Xa          ets ausschalten */
315 xp          w->Flags |= BORDERLESS; /* Rahmen auss
316 31          chalten */
317 04          RefreshWindowFrame(w);
318 AG          fprintf(stdout, "\x9b0y\x9b0x\n"); /* Li
nke obere Ecke */
sprintf(&columns[0], "%d", (BYTE)((UWORD)640/w->IFont
->tf_XSize));
fprintf(stdout, "\x9b%su\n", &columns[0]);
sprintf(&lines[0], "%d", (BYTE)((UWORD)w->WScreen->H
eight/

```



```

319 Wr      w->IFont->tf_YSize));
320 hE6      fprintf(stdout, "\x9b%st\x0c\n", &lines[0]);
321 Bg3      }
322 Ch0      }
323 WsF      /* Interlace Modus einschalten */
324 nF0 VOID
325 RG      interlace_on()
326 Cf      {
327 qV3      GfxBase->system_bplcon0 != INTERLACE;
328 Yf      RemakeDisplay();
329 Jo0      }
330 sIF      /* Interlace Modus ausschalten */
331 um0 VOID
332 x3      interlace_off()
333 Jm      {
334 AX3      GfxBase->system_bplcon0 &= ~INTERLACE;
335 fm      RemakeDisplay();
336 Qv0      }
337 UXF      /* Neuen Zeichensatz von Disk laden */
338 It0 VOID
339 JX      new_font(name)
340 jK3      BYTE *name;
341 Ru0      {
342 m13      BYTE      font[NAMESIZE], *size;
343 yz      REGISTER SHORT i;
344 Om      if (*name == EOS) {
345 VV6      printf("Zeichensatzname und -groesse fehlen !\n\n");
346 f9      exit_program(28);
347 b63      }
348 Op      size = name;
349 eo      for (i=0; (*size != '/') && (*size != EOS); i++) font[i]
          = *size++;
350 GQ      if (!i) {
351 uV6      printf("Zeichensatzname fehlt !\n\n");
352 rH      exit_program(29);
353 hC3      }
354 dD      if ((*size == EOS) || (*++size == EOS)) {
355 wx6      printf("Zeichensatzgroesse fehlt !\n\n");
356 89      exit_program(30);
357 lG3      }
358 qK      if (!atoi(size)) {
359 qz6      printf("Falsche Zeichensatzgroesse !\n\n");
360 IF      exit_program(31);
361 pK3      }
362 IT      font[i] = EOS;
363 bV      strcat(font, ".font");
364 yx      get_font(font, (UWORD)atoi(size), (UBYTE)FPF_DISKFONT);
365 t00      }
366 oVF      /* Standard Zeichensatz aus dem ROM (topaz)
          einschalten */
367 UMO VOID
368 9Z      default_font()
369 tM      {
370 iU3      if (!p = GetPrefs((struct Preferences *)
371 g2K      AllocMem(PSIZE, MEMF_CLEAR, PSIZE))) e
          xit_program(32);
372 Gf3      get_font("topaz.font", (UWORD)p->FontHeight, (UBYTE)FPF
          _ROMFONT);
373 ZW      FreeMem(p, PSIZE);

```

```

374 2X0      }
375 BGF      /* Zeichensatz umschalten */
376 dV0 VOID
377 IG      get_font(font, size, flag)
378 ES3      BYTE *font;
379 ZV      UWORD size;
380 DF      UBYTE flag;
381 5Y0      {
382 tB3      struct TextAttr *ta, *old_ta;
383 6K      struct TextFont *tf, *old_tf;
384 bD      BYTE      columns[4], lines[4];
385 ks      if (!DiskFontBase) {
386 Pf6      if (!DiskFontBase = (struct DiskFontBase *)
387 HUS      OpenLibrary("diskfont.library",
          REV))) {
388 aA9      printf("'diskfont.library' fehlt !\n\n");
389 xm      exit_program(33);
390 In6      }
391 Jo3      }
392 4L      if (!(ta = (struct TextAttr *)
393 MyE      AllocMem((ULONG)sizeof(*ta), MEMF_CLEAR))) ex
          it_program(34);
394 lX3      ta->ta_Name = (STRPTR)font;
395 OX      ta->ta_YSize = size;
396 gr      ta->ta_Style = FS_NORMAL;
397 mv      ta->ta_Flags = flag;
398 89      if (flag == FPF_DISKFONT) {
399 Vs6      tf = (struct TextFont *)OpenDiskFont(ta);
400 AE3      } else {
401 rD6      tf = (struct TextFont *)OpenFont(ta);
402 Uz3      }
403 Tb      if (!tf) {
404 Jg6      printf("Kann Zeichensatz '%s/%d' nicht finden !\n\n",
          font, size);
405 FJ3      } else {
406 xk6      old_ta = w->WScreen->Font;
407 M5      w->WScreen->Font = ta;
408 hP      ta = old_ta;
409 9k      old_tf = w->RPort->Font;
410 PH      if (!SetFont(w->RPort, tf)) old_tf = tf;
411 nd      CloseFont(old_tf);
412 Is      printf("\033c"); /* Bildschirm loeschen */
413 fA3      }
414 P1      FreeMem(ta, (ULONG)sizeof(*ta));
415 lF      if (!(strcmp(w->UserData, BUF_NAME))) {
416 aS6      fprintf(stdout, "\x9b0yx9b0x\n"); /* Li
          nke obere Ecke */
417 Mn      sprintf(&columns[0], "%d", (BYTE)((UWORD)640/tf->tf_X
          Size));
418 dh      fprintf(stdout, "\x9b%su\n", &columns[0]);
419 nt      sprintf(&lines[0], "%d", (BYTE)((UWORD)w->WScreen->H
          eight/
          tf->tf_YSize));
420 Fs      fprintf(stdout, "\x9b%st\x0c\n", &lines[0]);
421 Kr6      }
422 oJ3      }
423 pK0      }
(C) 1987 M&T

```

Listing 1. (Schluß)

3.5" Colordisketten: rot, gelb, grün, orange	je Stück	2,99 DM
3.5" No Name Disks, nur 2DD, 135 tpi, 1. Wahl	Staffelpreise	
3.5" Markendisks, sehr gute Qualität	je 10 Stück von 28,- bis 23,- DM	
Reinigungsset (naß, trocken)	5,25" 15,- 3,5" nur 16,- DM	
Monitorreinigungssset (naß, trocken), 20 PADS	nur 19,- DM	
Diskettenbox, doppelreihig, Schubladensystem, für 150 3.5"	45,- DM	
Markt & Technik-Bücher ab Lager lieferbar, speziell für Amiga	auf Anfrage	
Tastaturschacht für A2000, ermöglicht das Reinschieben der Tastatur		
inkl. Kabel & Mouse, unter den A2000 (ähnli. A1000)	129,- DM	
diverse Drucker-, Monitor-, DFÜ-, Anschluß-, Adapterkabel	ab 19,- DM	
Druckerstände, Plexi, sehr stabil, Papierablage	59,- DM	
NEC 1036A als Ersatzlaufwerk für A500, 1000, 2000, Industrieversion	249,- DM	
NEC 1037A 199,- DM NEC 1157C 258,-	alle NEC-Laufwerke + Ersatzteile	
diverse Spezialstecker & Buchsen für Amiga, z.B. 23pol. + Gehäuse	7,- DM	
selbstverständlich auch Ersatzteile (8520 ...)	auf Anfrage	
Amiga 500-2000, Tech. Reference Manual, Original Commodore	auf Anfrage	
NEC 1037A Laufwerk, 100% kompatibel, abschaltbar, mit/ohne Bus	auf Anfrage	
NEC 1037A Doppellaufwerk, tech. Daten wie vor, bes. günstig	auf Anfrage	
A2000 Int. NEC Spitzenlaufwerk, Einbaumat., deutsche Anleitung	249,- DM	
Interessante Festplatten mit/ohne Controller für die Amigas, z.B. den		
Testieger aus Amiga 9/88: NEC D 3146H 3.5"	auf Anfrage	
A2090/2 Commodore SCSI/ST 506 Controller für den A2000 + A500		
mit Adapter, inkl. Software + Anschlußkabel	798,- DM	

A500 Speichererw., 512 K auf 1 MB, inkl. akkugepufferter Echtzeituhr, hardwarem. abschaltbar, 1. ab. dtsh. Modell auf Anfrage

Abdeckhauben: A500 22,- DM, A2000 Tast. 22,- A 1081/4, NEC 43,- DM

Amiga Public Domain auf 3.5" Disketten z.B. Fish, RPD, CC, ab 4,50 DM

Faug, Panorama, Auge, ACS ab 11 St. 3,50 DM ab 25 St. nur 3,- DM

NEC P6+, P7+, div. Zubehör: Colorsatz, Einzelblatteinzug auf Anfrage

Star LC-10 oder LC-10-Color, Commodore MPS 1500 ab Lager lieferbar

Kickstartumschaltplatine für A500, 1000, 2000 bis max. 4 KS auf Anfrage

Eprombrennservice, Kick 1.3, Viruskick auf Anfrage

Stereobausatz für den 1081 (+ bgl.) Monitore, 100% Stereo 79,- DM

Wordperfect, die Textverarbeitung für den Amiga nur 359,- DM

NEC Multisync I, kein Import, dtsh. Version, anschlußf. 1398,- DM

Eizo 8060S, 0.28 dots, Spitzenmonitor mit Fuß, anschlußf. 1439,- DM

komplettes NEC- + Eizo-Programm, anschlußfertig an AMIGA!

*** Wir liefern Ihnen alle Hard-, Software für das 1. Amiga-Sonderheft, Preise auf Anfrage

*** Informationen + 3 PD-V Disks gegen 10,- DM nur in Bar, keine Schecks

*** per Vorauskasse ohne Versandkosten!

Versand: UPS-Nachnahme + anteilige Versandkosten

Achtung, viele unserer Preise standen bei Drucklegung nicht genau fest bzw. sind ständig in Bewegung. Bei Bedarf bitte anfragen!

AHS-AMEGAS Hard & Software Vertriebs GmbH

Postfach 100248 · Ladenverkauf: Kaiserstraße 82 · 6360 Friedberg 1 · Telefon 06031/61950 (nicht aufgeben!)

Programmname:	Division
Computer:	Amiga 500, 1000, 2000 mit Kickstart 1.2
Sprache:	Basic

Programm : Division

```

1 UwO REM *****
2 gH REM *          Division          *
3 Qb REM *   written by Roland Keßler   *
4 zi REM * (c) 1987 by Markt & Technik *
5 YO REM *****
6 EO REM Demonstration des Unterprogramms --Division--
7 2c Eingabe:
8 gm2 CLS
9 t1 WIDTH 77
10 Ws INPUT "Dividend ";Dividend
11 cs INPUT "Divisor ";Divisor
12 Zv INPUT "Länge ";Laenge
13 ZC PRINT
14 S9 PRINT "Sind Sie sicher (j/n) ?"
15 kYO Abfrage1:
16 Ok2 Eingabe$=INKEY$
17 rI IF Eingabe$="n" THEN RUN
18 us IF Eingabe$<>"j" THEN Abfrage1
19 rz CALL Division(Dividend,Divisor,Laenge)
20 gJ PRINT
21 9N PRINT Dividend;" ";Divisor;"=";Ergebnis$
22 iL PRINT
23 L8 PRINT "Ausgabe auf Drucker (j/n) ?"
24 xm0 Abfrage2:
25 9t2 Eingabe$=INKEY$
26 Cq IF Eingabe$="n" THEN WeiterJN
27 93 IF Eingabe$<>"j" THEN Abfrage2
28 Et LPRINT Dividend;" ";Divisor;"="

```

```

29 z0 FOR Lauf=1 TO LEN(Ergebnis$) STEP 77
30 P64 LPRINT MID$(Ergebnis$,Lauf,77)
31 Hc2 NEXT Lauf
32 vk LPRINT
33 Yn0 WeiterJN:
34 uX2 PRINT
35 fp PRINT "Noch eine Rechnung (j/n) ?"
36 D30 Abfrage3:
37 L52 Eingabe$=INKEY$
38 JG IF Eingabe$="n" THEN END
39 RH IF Eingabe$<>"j" THEN Abfrage3
40 nC RUN
41 DSO REM SUB-Programm --Division--
42 70 SUB Division(DD,DS,LA) STATIC
43 YD2 SHARED Ergebnis$
44 nv Dividend=DD
45 7R Divisor=DS
46 3Q Laenge=LA
47 kI IF SGN(Dividend)<>SGN(Divisor) THEN Ergebnis$="-"
48 tz Dividend=ABS(Dividend)
49 os Divisor=ABS(Divisor)
50 3T WHILE Dividend<>INT(Dividend) OR Divisor<>INT(Divisor)
51 VC4 Dividend=Dividend*10
52 Vz Divisor=Divisor*10
53 I62 WEND
54 FN FOR Lauf=1 TO Laenge
55 aG4 Stellen=FIX(Dividend/Divisor)
56 tb Ergebnis$=Ergebnis$+RIGHT$(STR$(Stellen),LEN(STR$(Stellen))-1)
57 xo Dividend=10*(Dividend-Stellen*Divisor)
58 9s IF Komma=0 THEN Ergebnis$=Ergebnis$+"," : Komma=1
59 j42 NEXT Lauf
60 020 END SUB
(C) 1987 M&T

```

Listing 1. Das Demoprogramm zu »Division«.
Bitte mit dem Checksummer (Seite 159) eingeben.

Genauere Division

Manchmal ist es sinnvoll, beim Teilen zweier Zahlen mehr als 16 Nachkommastellen zu erhalten. Mit »Division« können Sie bis zu 32 000 Stellen hinter dem Komma berechnen lassen.

Ein Anwendungsfall ist das Erkennen von Perioden (das ist eine Ziffernfolge eines Bruchs, die sich ständig wiederholt). Man benötigt dazu mindestens die zweifache Periodenlänge, bei dem Bruch 100/233 immerhin 464 Ziffern nach dem Komma. Selbst bei doppelter Genauigkeit erhält man in Basic höchstens 16 Nachkommastellen. »Division« ist jedoch in der Lage bis zu 32 000 Stellen hinter dem Komma zu berechnen.

Das Programm Division (Listing 1) besteht aus zwei Teilen, dem Unterprogramm und dem Demonstrationsteil, der nur die Anwendung des Unterprogramms veranschaulicht. In den Zeilen 8 bis 14 wird der Bildschirm gelöscht, die Zeilenlänge auf 80 Zeichen eingestellt und die Eingabe der drei Variablen »Dividend«, »Divisor« und »Laenge« verlangt. Die Variable Laenge beinhaltet die gewünschte Länge des Ergebnis-Strings. In den Zeilen 16 bis 21 fragt das Programm ab, ob die Eingabe richtig war. Zugelassen sind die Tasten <n> und <j>. Die SUB-Routine »Division« wird in der Zeile 20 aufgerufen, die Variablen Dividend, Divisor und Laenge werden dabei an das Unterprogramm übergeben. Nachdem vom Unterprogramm die Variable »Ergebnis\$« mit der gewünschten Anzahl der Ziffern des Ergebnisses gefüllt worden ist, wird die Rechnung und das Ergebnis auf dem Bildschirm ausgegeben (Zeilen 21 bis 23). Nun wird der Benutzer in den Zeilen 24 bis 28 gefragt, ob eine Ausgabe auf dem Drucker gewünscht wird. Falls ja, wird in den Zeilen 30 bis 34 die Rechnung und das Ergebnis auf dem Drucker ausgegeben. Dabei wird der Ergebnis-String durch den MID\$-Befehl in Teile zu je 77 Zeichen aufgeteilt, die einzeln an den Drucker geschickt werden. So erreicht man eine formatierte Ausgabe. Ohne diesen Befehl kann es Probleme mit der Preferences-Einstellung der Druckzeilen-Breite geben. Danach

wird gefragt, ob noch eine Rechnung durchgeführt werden soll. Wenn ja, wird das Programm neu gestartet, sonst beendet. In den Zeilen 47 bis 66 befindet sich das Unterprogramm »Division«. Der Algorithmus hält sich an die Methode, die beim Dividieren »per Hand« angewandt wird. Auf einem Blatt Papier kann man das gut nachvollziehen. Die Zeilen 47 und 48 sind der Kopf der Routine. Es werden die drei Variablen aus dem Hauptprogramm erwartet, die hier allerdings die Namen DD, DS und LA haben, um eine Veränderung der Variablen Dividend und Divisor zu verhindern. Durch die SHARED-Anweisung hat auch das Hauptprogramm Zugriff auf Ergebnis\$. Um die SUB-Routine verständlicher zu gestalten, werden die Variablen DD, DS und LA sofort wieder in Dividend, Divisor und Laenge umbenannt, wobei dies allerdings lokale Variablen sind. Falls Dividend und Divisor unterschiedliche Vorzeichen haben, erhält das Ergebnis als erste Stelle ein »-« (Zeile 53). In den Zeilen 54 und 55 werden die Vorzeichen von Dividend und Divisor entfernt, und dann werden beide Variablen so lange mit zehn multipliziert, bis sie keine Nachkommastellen mehr haben. Damit sind die Vorbereitungen abgeschlossen.

Genauigkeit ist Trumpf

In der FOR...NEXT-Schleife wird der Dividend durch den Divisor geteilt und die Stellen vor dem Komma des Ergebnisses an die Variable Ergebnis\$ angehängt. Durch den RIGHT\$-Befehl wird von dieser Zahl eine Stelle von links abgeschnitten, weil diese entweder ein »-« oder ein Leerzeichen enthält (Zeile 62).

In der Zeile 63 wird der Rest der Division berechnet. Nun müßte man die nächste Stelle »herunterholen«, da aber im ersten Durchgang der FOR...NEXT-Schleife schon mit dem ganzen Dividend gerechnet wird, muß nur eine Null angehängt werden, was durch das »10*...« auch geschieht. Da, wie schon erwähnt, im ersten Durchgang mit dem ganzen Dividend gerechnet wird, muß auch im ersten Durchgang das Komma gesetzt werden (Zeile 64).

Wenn Sie eine Division mit großer Genauigkeit benötigen, müssen Sie die Zeilen 47 bis 66 an Ihr Programm anhängen. Der Aufruf geschieht wie oben beschrieben. (Roland Keßler/rs)

Basic für Aufsteiger

Basic-Programmierung. Schöpfen Sie die tollen Fähigkeiten Ihres Computers endlich voll aus.

Das mit dem Amiga zusammen ausgelieferte Basic gehört zu einer neuen Generation von Interpretern: Programmzeilen benötigen keine Zeilennummern mehr, lange Variablennamen machen ein Programm wesentlich lesbarer. Neue Befehle, die Zusammenfassung von Programmzeilen zu Blöcken und alphanumerische Ansprungsmarkern gestatten, verbessern den Ruf einer Programmiersprache, in der strukturierte Programmierung bisher ein Fremdwort war. Die Grafik-Befehle eröffnen neue Dimensionen in der optischen Gestaltung von Bildschirmausgaben. Mit den Sound-Befehlen zaubern Sie vierstimmigen Stereosound aus dem Computer. Ihrer Kreativität sind keine Grenzen gesetzt.

sondern auch künstlerische Fähigkeiten anspruchsvoll umgesetzt werden können aber auch Einsteiger profitieren von diesem umfangreichen Basic-Kurs.

Basic strukturiert

Zunächst soll erklärt werden, was unter strukturierter Programmierung eigentlich gemeint ist und wie Amiga-Basic diese Programmierphilosophie unterstützt. Die Meinungen über die strukturierte Programmierung sind durchaus nicht einstimmig. Während sie bei Profis zur Selbstverständlichkeit gehört, ist sie bei Freizeitprogrammierern nach wie vor umstritten. Natürlich ist es viel leichter, zur Erstellung

Amiga-Basic unterstützt die fantastischen Fähigkeiten der Hardware mit mächtigen Befehlen. Vorbei sind die Zeiten, in denen Grafik- und Musikkunstwerke mit PEEK und POKE programmiert wurden. Begeben Sie sich mit uns in neue Dimensionen der Basic-Programmierung. Schöpfen Sie die tollen Fähigkeiten Ihres Computers endlich voll aus.

rierten Gegenstände. Soll nach einiger Zeit Ihr Programm um eine »Kleinigkeit« erweitert werden, so fällt Ihnen das als Autor natürlich nicht schwer: einfach nach Zeile x die Anweisung a, b und c einfügen — fertig. Aber was ist das: Plötzlich spielt der Programmteil d vollkommen verrückt. Aber für Sie als Programmierer ist das ja kein Problem: einfach (?)...

Einfacher wäre es jedenfalls, wenn das Programm eine klare, leicht durchschaubare Struktur besäße, also strukturiert programmiert worden wäre. Die Vorteile der strukturierten Programmierung wollen wir Ihnen anhand eines, bei der Einführung in diese Technik oft verwendeten, Beispiels demonstrieren: Die Lösung folgender quadratischer Gleichung:

$$a * x^2 + b * x + c = 0$$

Eine solche Gleichung hat in der Regel zwei Lösungen:

$$x1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Keine Sorge — wir werden dieses Problem lösen, ohne Sie mit komplizierten mathematischen Beweisführungen zu belasten. Vorher sollten wir uns allerdings noch ein paar grundsätzliche Gedanken zur Programmierung von Computern machen.

Programmieren — was ist das?

Wenn Sie sich entschlossen haben, ein bestimmtes Problem mit dem Computer zu lösen, so sollten Sie erst einmal die Frage beantworten: Ist das Problem überhaupt auf einem Computer lösbar? Dann muß ein (Programmier-)Verfahren gefunden werden, mit dem sich das Problem lösen läßt. Ein solches Verfahren nennt man auch Algorithmus, und das ist nichts anderes als eine Lösungsvorschrift, die es dem Computer gestattet, das Problem eindeutig und in angemessener Zeit zu lösen. Unterschätzen Sie den Faktor Zeit hierbei nicht, denn es gibt auch Lösungsverfahren, die zwar theoretisch in der Lage sind, ei-

Was ist ein Interpreter?

Wenn ein Basic-Programm mit einem Interpreter eingegeben und gestartet wird, muß er für jeden Befehl nachsehen, was er tun soll. Dafür existiert im Interpreter für jede Anweisung ein Maschinenprogramm. Dieses muß der Interpreter finden und ausführen.

Haben Sie ein einfaches Programm wie

```
PRINT "Hallo";
GOTO 100
```

schaut der Interpreter bei »PRINT« erst nach, ob er den Befehl überhaupt kennt. Ist dies der Fall, so startet er das Programm, das für diesen Befehl zuständig ist. Ebenso verläuft es bei der »GOTO«-Anweisung.

Bei dieser Methode geht viel Zeit verloren. Ein Compiler umgeht diese Zeitspanne, indem er das Programm ein einziges Mal beim Start komplett in Maschinensprache übersetzt. So kann sich der Prozessor bei compilierten Programmen mit seiner eigentlichen Aufgabe beschäftigen und muß nicht ständig nach Befehlsprogrammen suchen.

Compiler haben jedoch einen entscheidenden Nachteil: Soll in einem Programm ein Fehler behoben werden, so muß der Compiler gestartet, das Programm geladen, korrigiert und wieder compiliert werden. Bei einem Interpreter hingegen kann das Programm nach Korrekturen sofort wieder gestartet werden.

Sie stehen schon mit einem Bein in der Programmierung und können mit Begriffen wie FOR...NEXT oder IF...THEN umgehen? Dieser Kurs hilft Ihnen, festen Fuß in einer Umgebung zu fassen, in der es nicht mehr nur auf Logik ankommt,

eines Programms einfach drauffloszutippen und die Ablauflogik nebenher während der Programmierung zu entwickeln.

Derartige Programme funktionieren nach der Fertigstellung ebenso wie ihre struktu-

KURSÜBERSICHT

1. Strukturierte Programmierung

Programmentwicklung, Top-Down-Design, Struktogramme, optische Gestaltung von Quelltexten, »IF...THEN...ELSE...ENDIF«, »WHILE...WEND« (Seite 85)

2. Unterprogramme

Funktionen, GOSUB-Unterprogramme, echte Unterprogramme, lokale und globale Variablen, Parameterübergabe (Seite 90)

3. Daten und Dateien

Programmdateien, serielle Dateien, relative Dateien, indexsequentieller Dateizugriff (Seite 92)

4. Grafik

Setzen von Punkten, Screens und Windows, Rechtecke und Ellipsen, Muster, Kopieren und Scrollen von Bildschirmausschnitten (Seite 95)

5. Animation

Bobs und Sprites, Bewegung von Objekten, Kollisionen (Seite 99)

6. Geräusche

Spracherzeugung, Tonleitern, Klangcharakteristik, Wellenformen (Seite 105)

7. Unterbrechungsverarbeitung

Mausereignisse, Menüauswahl, Fehlerbehandlung, Zeitmessung (Seite 106)

8. Externe Funktionen

Einbinden von Maschinenprogrammen, Aufruf von Library-Funktionen, BMAP-Format (Seite 110)

ne eindeutige Lösung zu liefern, jedoch dafür zuviel Zeit benötigen. Bei unserem Beispiel scheint es in dieser Hinsicht keine Schwierigkeiten zu geben, da als Lösungsalgorithmus die oben angegebenen Formeln zur Verfügung stehen. Der programmierte Algorithmus für x1 lautet also:

```
x1 = (-b + SQR( b*b -
4*a*c )) / (2*a).
```

Nicht selten gilt jedoch ein Algorithmus zwar für die meisten Variationen eines Problems, bei Sonderfällen aber versagt er. So ist in unserem Beispiel der Lösungsalgorithmus nicht brauchbar, falls der Koeffizient »a« gleich Null ist. Aus der Ursprungsgleichung ergibt sich dann »b*x + c = 0« (eine sogenannte lineare Gleichung). Diese Gleichung besitzt, falls b von Null verschieden ist, die Lösung »x = -c/b«. Ebenfalls mit diesem Algorithmus nicht lösbar ist der Fall, wenn der Wert unter der Wurzel negativ ist. Berücksichtigen Sie bei der Programmierung diese Sonderfälle nicht, und ein Anwender Ihres Programmes will eine derartige Gleichung lösen, so führt dies im günstigsten Fall zum Programmabbruch, oder schlimmer: Der Anwender erhält ein falsches Ergebnis. So verhalten sich nur schlecht durchdachte, minderwertige Programme.

Unser erstes Programm (Listing 1) zeigt Ihnen einen Lösungsvorschlag, wie er bei Basic-Programmen häufig anzutreffen ist. Allerdings bedarf es doch schon einiger Anstrengung, um dem Listing zu ent-

nehmen, was dieses Programm nun eigentlich an einer bestimmten Stelle macht.

Von oben nach unten

Zum Glück gibt es einige Hilfsmittel, die es gestatten, das Erstellen von Programmen übersichtlicher zu gestalten: Zum einen sind dies »Struktogramme« genannte grafische Darstellungen der logischen Programmstruktur (siehe Tabelle 1), zum anderen ist es die Programmiermethode des »Top-Down-Design«. Bei der letztgenannten Methode zerlegt man ein Problem erst grob in die zur Lösung notwendigen Schritte. Dann werden diese Bausteine schrittweise so verfeinert, bis ein ablauffähiges Programm entsteht. Auch wir werden nach dieser Methode vorgehen.

Zuerst Überlegen wir uns also, in welche Teile man das Lösen der quadratischen Gleichung zerlegen könnte. Grob gesehen sind das zunächst drei Varianten:

1. Eingabe der Werte für die Koeffizienten a, b und c
2. Lösen der linearen Gleichung
3. Lösen der quadratischen Gleichung

Die Punkte 2 und 3 schließen sich gegenseitig aus, das heißt, es wird entweder nur die eine oder die andere Tätigkeit ausgeführt. Durch Überprüfung des Wertes der Variablen »a« wird entschieden, welche Gleichung zu lösen ist. Versuchen wir diesen Zusammenhang einmal grafisch darzustellen.

Wir verwenden für eine solche übersichtliche Darstellung dazu die in Tabelle 1 gezeigten Strukturblocke. Aus diesen werden sogenannte Struktogramme aufgebaut. Ein für unsere Zwecke geeigneter Block ist die »zweiseitige Alternative«. Die zu überprüfende Bedingung lautet »a = 0«. Ist diese Bedingung wahr, so soll der mit »ja« gekennzeichnete Block (die lineare Gleichung), im anderen Fall der mit »nein« bezeichnete Block (quadratische Gleichung) ausgeführt werden (Bild 1).

Übersicht durch Struktogramme

Würden wir dieses Struktogramm in ein Programm übersetzen, so wäre der Programmablauf nach der Ausgabe der Ergebnisse beendet. Eine wei-

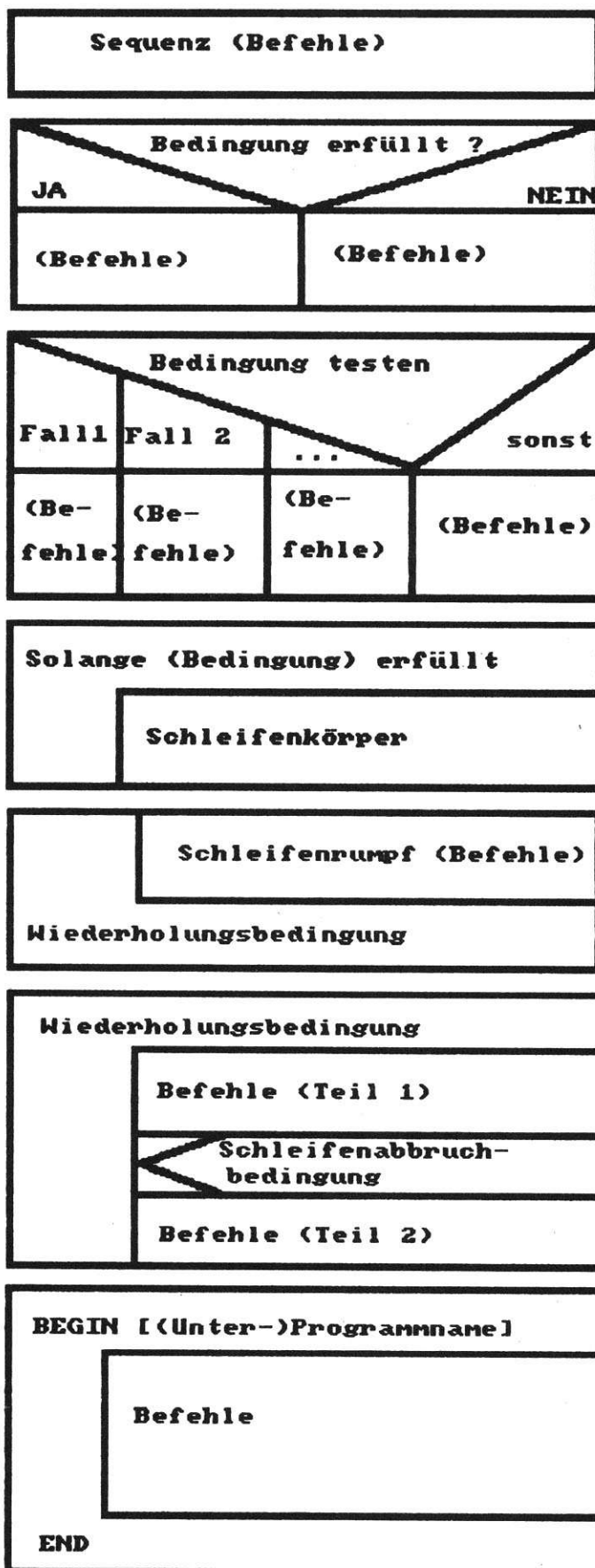


Tabelle 1. Die Strukturblocke und ihre Bedeutung: Unterhalb der Verzweigungsblöcke befindet sich der erste Schleifenkörper. Die Befehle im Schleifenrumpf werden mindestens einmal abgearbeitet, man spricht daher von einer nicht-abweisenden Schleife. Darunter befindet sich eine Schleife mit Abbruchbedingung. Ganz unten sehen Sie den Struktogramm-Rahmen eines Unterprogramms.

```
100 INPUT "a,b,c"; a,b,c
110 IF a = 0 GOTO 190
120 d = b*b - 4*a*c
130 IF d < 0 GOTO 260
140 IF d = 0 GOTO 280
150 x1 = (-b + SQR(d))/2/a
160 x2 = (-b - SQR(d))/2/a
170 PRINT x1,x2
180 GOTO 100
190 IF b = 0 GOTO 230
200 x0 = -c/b
210 PRINT x0
220 GOTO 100
230 IF c = 0 GOTO 310
240 PRINT "Keine Lösung"
250 GOTO 100
260 PRINT "Komplexe Lösungen
kann ich nicht"
270 GOTO 100
280 x0 = -b/2/a
290 PRINT x0
300 GOTO 100
310 END
```

Listing 1. So sieht ein unstrukturiertes Programm aus

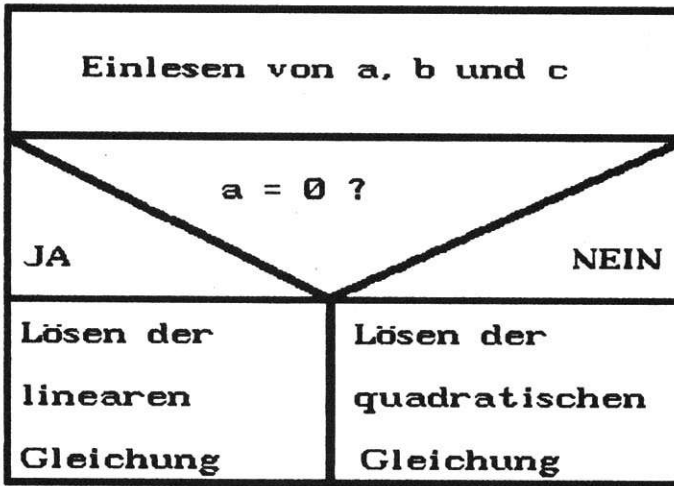


Bild 1. Problemendarstellung durch Struktogramme

tere Berechnung könnte erst durch einen Neustart des Programms erfolgen. Da dies etwas unkomfortabel ist, ergänzen wir also die Aufgabenstellung so, daß eine beliebige Anzahl von Berechnungen durchgeführt werden kann. Wir wollen den Anwender entscheiden lassen, wann er das Programm beenden möchte. Oft lassen sich für solche Zwecke »unsinnige Eingaben« ausnutzen. Eine solche ist zum Beispiel die Eingabe von Null für alle drei Koeffizienten.

Der Übersicht (Tabelle 1) können Sie die Form des Strukturblocks für Schleifenanweisungen entnehmen. Oberhalb des Schleifenkörpers wird die Bedingung festgelegt, bei der die Anweisungen desselben ausgeführt beziehungsweise wiederholt werden. Für unser Beispiel ist der Schleifenkörper

per das in Bild 1 gezeigte Struktogramm und die Wiederholungsbedingung lautet »a oder b oder c ungleich Null«. Damit man erkennen kann, daß es sich bei einem Struktogramm um ein komplettes Programm und nicht um einen Ausschnitt handelt, wird um das Diagramm ein Struktogrammrahmen gelegt (Bild 2).

Das sichere Baugerüst

Nachdem nun das grobe Programmgerüst entwickelt ist, wenden wir uns den einzelnen Bausteinen zu. Das Struktogramm zur Lösung der linearen Gleichung (Bild 3) macht uns keine größeren Schwierigkeiten. Durch eine weitere zweiseitige Alternative wird der Fall, daß auch »b« den Wert

Null hat, abgefangen. Bei der quadratischen Gleichung (Bild 4) muß zunächst bestimmt werden, ob der Wert innerhalb der Wurzel (im Struktogramm mit »d« bezeichnet) negativ ist. Ist dies der Fall, gibt der Computer eine entsprechende Meldung aus. Nun bleiben nur noch zwei Möglichkeiten übrig: »d« ist Null oder größer als Null. Diese beiden Fälle werden unterschiedlich behandelt.

Aufgrund der von uns gewählten Struktogrammgröße ist für die hierzu notwendigen Schritte kein Platz mehr in der Grafik. Daher wird an dieser Stelle der Grafik vermerkt, daß ein weiteres Struktogramm (dieses sehen Sie in Bild 5) eingefügt werden muß. Nach der Fertigstellung der einzelnen Bausteine werden diese in das Programmgerüst eingefügt.

So entsteht das komplette Struktogramm für die Lösung unseres Problems (Bild 6).

Der Vorteil von Struktogrammen gegenüber den bekannten Programmablaufplänen ist ihre größere Übersichtlichkeit und die Möglichkeit, daß jedes Struktogramm als Block wieder in einem anderen enthalten sein kann; das heißt man kann Struktogramme beliebig tief verschachteln. Der besseren Übersichtlichkeit halber sollten Sie sich jedoch an die Regel halten, ein Struktogramm niemals größer als eine normale Schreibmaschinen-seite zu machen. Bei Platzproblemen lagern Sie Struktogramme auf weitere Seiten aus und machen dies mit einem Vermerk an entsprechender Stelle kenntlich, wie in obigem Beispiel gezeigt.

Lineare Gleichung

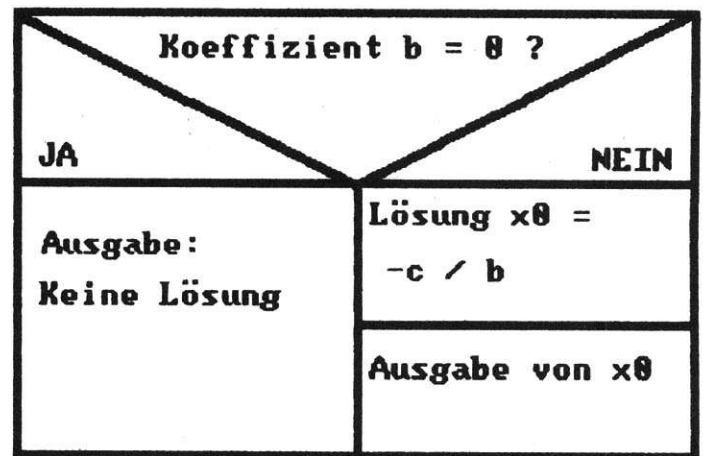


Bild 3. Die Lösung einer linearen Gleichung

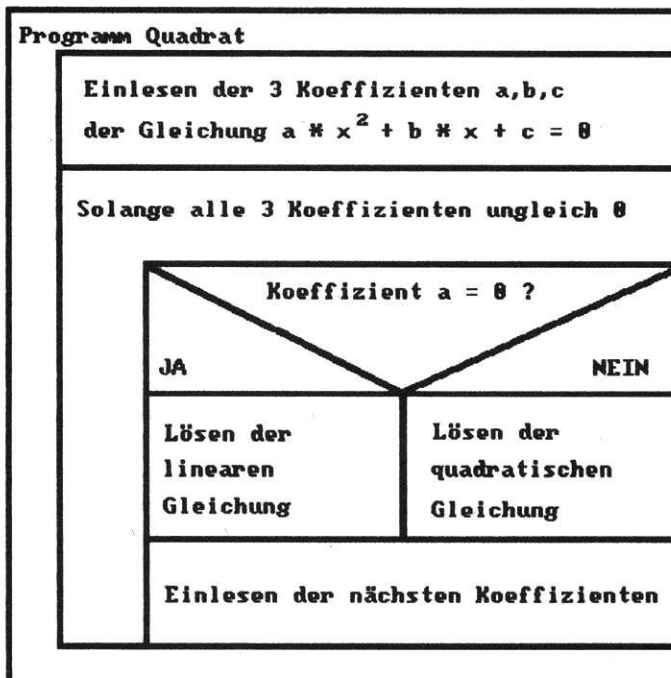


Bild 2. Struktogramm eines kompletten Programms

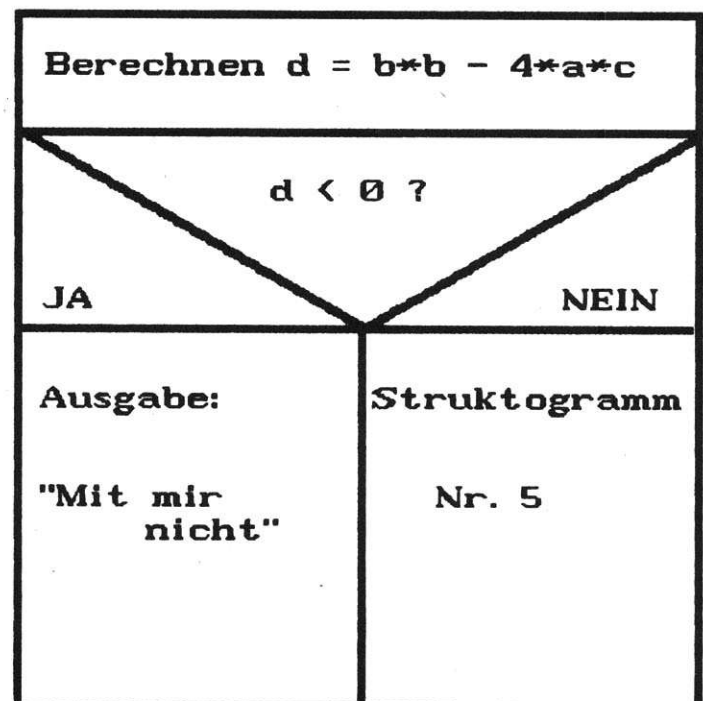
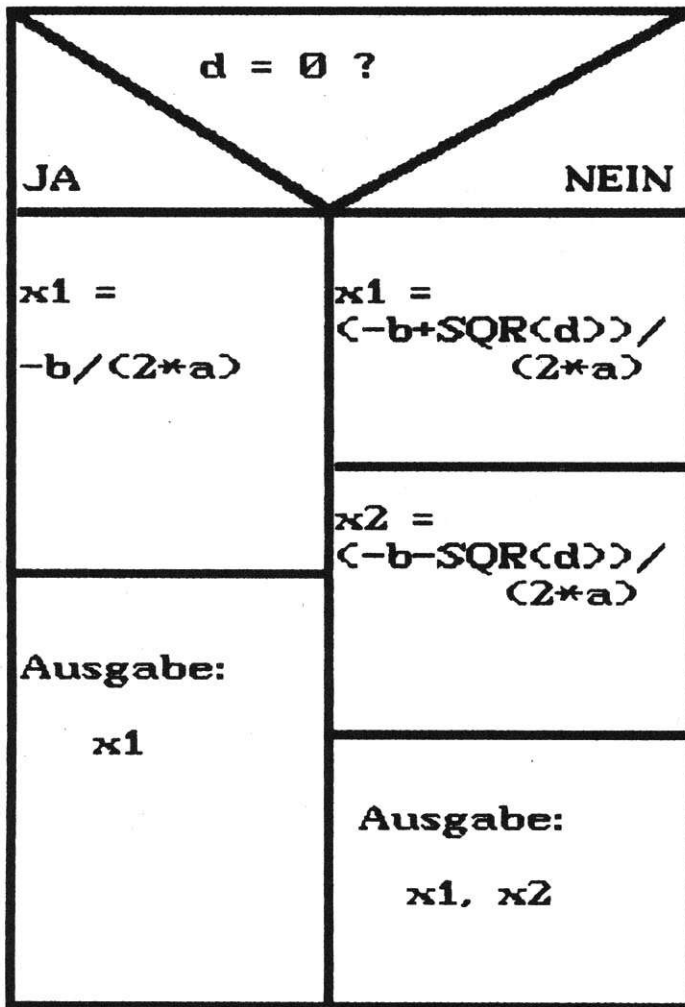


Bild 4. Lösung einer quadratischen Gleichung



Reine Routinesache

Mit der Entwicklung der Struktogramme ist die Problemaufbereitung abgeschlossen. Das Eingeben des Programms ist nur noch Routine-sache. Die Formulierungen im Struktogramm erinnern nur wenig an die Sprache Basic. In der Tat lassen sich solche Diagramme in jede beliebige Programmiersprache übersetzen. In unserer Übersicht (Tabelle 1) finden Sie für die jeweiligen Strukturblocke die entsprechenden Basic-Anweisungen.

Eine geeignete Vorgehensweise ist das Übersetzen jedes einzelnen Strukturblockes, um diesen dann getrennt von den anderen Bausteinen auf Fehler zu untersuchen. Beginnen wir mit dem Programmgerüst (Listing 2). Aus Ihrer Programmierpraxis wissen Sie vielleicht, daß Wiederholungen mit Programmschleifen realisiert werden. Die Anwendung einer »FOR...NEXT«-Schleife empfiehlt sich hier jedoch nicht, da hier die Anzahl der Wiederholungen von vornherein festgelegt wird. Mit »WHILE...WEND« bietet Ami-

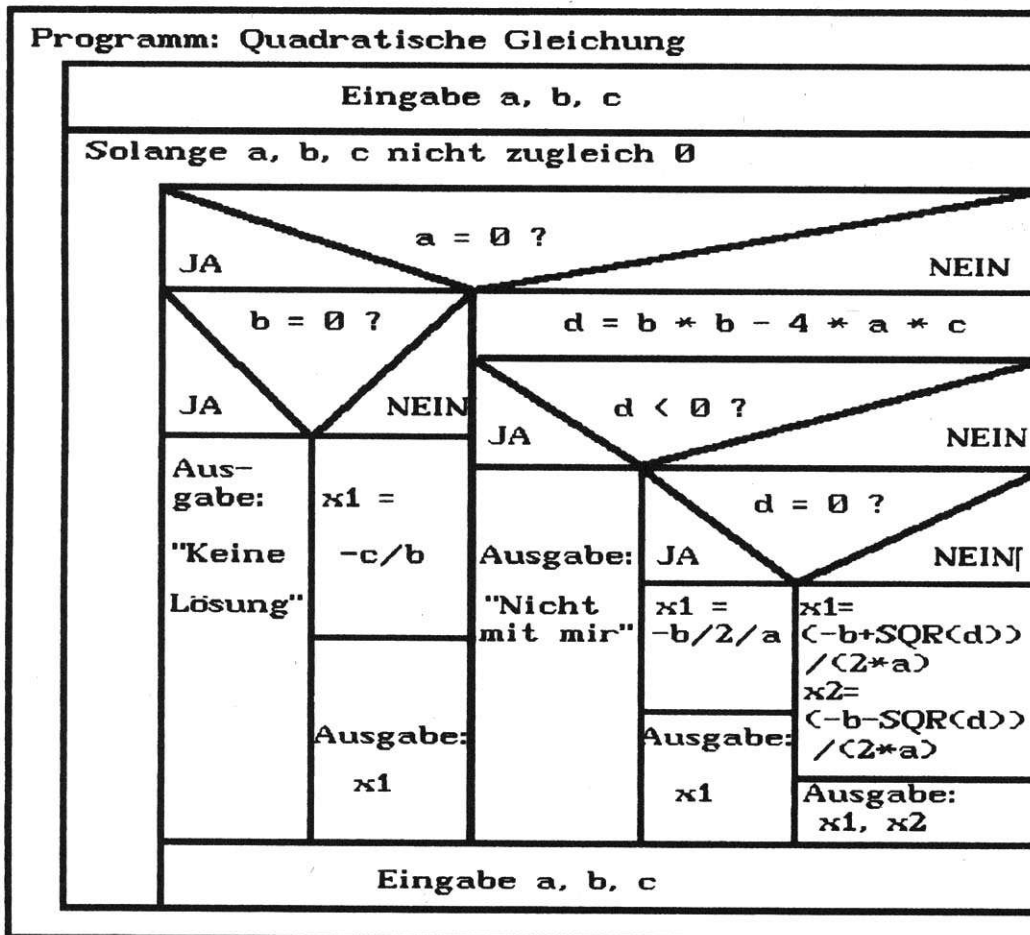
```

main:
  INPUT "a,b,c"; a,b,c
  WHILE a <> 0 OR b <> 0 OR c <> 0
    IF a = 0 THEN
      PRINT "Lösung der linearen Gleichung !"
    ELSE
      PRINT "Lösung der quadratischen Gleichung !"
    END IF
    INPUT "a,b,c"; a,b,c
  WEND
END
  
```

Listing 2. Das Programmgerüst unseres Problems

Bild 5.
Das ausgelagerte
Struktogramm Nr. 5

Bild 6.
Struktogramm eines
kompletten Programms



ga-Basic jedoch eine weitere Möglichkeit der Schleifenprogrammierung: Gelangt der Computer während des Programmablaufs an eine WHILE-Anweisung, so wird überprüft, ob die dem WHILE folgende Aussage zutrifft (hier: a,b,c ungleich Null). Ist dies der Fall (die Aussage ist wahr), so wird das Programm mit der Abarbeitung der zwischen »WHILE« und »WEND« stehenden Befehle fortfahren. Mit der Ausführung von »WEND« springt das Programm zurück zum entsprechenden »WHILE« und überprüft dort erneut die angegebene Bedingung. Dies wird solange fortgeführt, bis die Bedingung falsch ist. In diesem Fall fährt das Programm mit den Befehlen nach dem »WEND«-Befehl fort. Ist die Bedingung gleich bei der ersten Überprüfung falsch, so wird der zwischen »WHILE« und »WEND« stehende Programmteil niemals durchlaufen. Daher wird die »WHILE«-Schleife auch abweisende Schleife genannt. Vielleicht haben Sie erkannt, daß es sich bei der »FOR...NEXT«-Schleife um einen Spezialfall der »WHILE«-Schleife handelt, denn die Anweisungsfolge

```

I = 1
WHILE I <= 10
  ...
  (Befehle)
  ...
  I = I + 1
  
```



```
REM Input nur zum Test

INPUT "b,c"; b,c
IF b = 0 THEN
  PRINT "Keine Lösung"
ELSE
  x0 = -c/b
  PRINT x0
END IF
END
```

Listing 3. Programmteil lineare Gleichung

WEND
entspricht genau
FOR I = 1 TO 10
...
(Befehle)
...
NEXT I

Die Programmteile für das Lösen der Gleichungen (Listing 3 und 4) sind schon einfacher zu durchschauen. Die beiden »Zweige« der zweiseitigen Alternative (ja und nein) werden programmtechnisch mit »IF...THEN...ELSE...END IF« realisiert. Ist die zwischen dem »IF«- und dem »THEN«-Befehl stehende Bedingung erfüllt, so werden die zwischen »THEN« und »ELSE« stehenden Befehle abgearbeitet. Falls kein »ELSE«-Befehl vorhanden ist, werden die Befehle zwischen »THEN« und »END IF« ausgeführt. Ist die angegebene Bedingung jedoch

```
REM INPUT-Anweisung nur zum Test

INPUT "a,b,c"; a,b,c
d = b*b - 4*a*c
IF d > 0 THEN
  x1 = (-b + SQR(d))/2/a
  x2 = (-b - SQR(d))/2/a
  PRINT x1,x2
ELSEIF d = 0 THEN
  x0 = -b/2/a
  PRINT x0
ELSE
  PRINT "Mit mir nicht!"
END IF
END
```

Listing 4. Programmteil quadratische Gleichung

falsch, so werden die Befehle zwischen »ELSE« und »END IF« ausgeführt, falls kein »ELSE« vorhanden ist, wird die Befehlsausführung nach dem »END IF«-Befehl fortgesetzt.

Tritt die »ELSE«-Alternative überhaupt nicht auf, so wäre es Platzverschwendung, wenn man im zugehörigen Struktogramm diesem nicht vorhandenen Teil ebensoviel Platz einräumen würde, wie dem »THEN«-Zweig. Daher hat es sich eingebürgert, in diesem Fall den Strukturblock nach Bild 7 abzuwandeln.

```
Variable:
a = 0 ' Koeffizienten in der quadratischen
b = 0 ' Gleichung a * x*x + b * x + c = 0
c = 0
d = 0 ' Zwischenergebnis
x0 = 0 ' Endergebnis bei linearer Gleichung
x1 = 0 ' Endergebnisse bei
x2 = 0 ' quadratischer Gleichung

main:
INPUT "a,b,c"; a,b,c
WHILE a <> 0 OR b <> 0 OR c <> 0
  IF a = 0 THEN
    IF b = 0 THEN
      PRINT "Keine Lösung"
    ELSE
      x0 = -c/b
      PRINT x0
    END IF
  ELSE
    d = b*b - 4*a*c
    IF d > 0 THEN
      x1 = (-b + SQR(d))/2/a
      x2 = (-b - SQR(d))/2/a
      PRINT x1,x2
    ELSEIF d = 0 THEN
      x0 = -b/2/a
      PRINT x0
    ELSE
      PRINT "Mit mir nicht!"
    END IF
  END IF
  INPUT "a,b,c"; a,b,c
WEND
END
```

Listing 5. Die Lösung des Problems als strukturiertes Programm

Für die Testphase haben wir in die Beispielprogramme zwei Eingabeanweisungen eingefügt. Funktionieren alle Programmteile zufriedenstellend, so entfernen Sie die Testbefehle, fügen alle Teile zusammen und Ihre Problemlösung ist fertig (Listing 5). Damit Verschachtelungen im Programm leicht erkennbar sind, ist es vorteilhaft, den Programmtext der entsprechenden Strukturblocke einzurücken (siehe Listing 5). Sie haben jetzt die wichtigsten Techniken der strukturierten Programmierung kennengelernt. Sehen wir uns nun die weniger gebräuchlichen Strukturen an. In Tabelle 1 können Sie neben der abweisenden Schleife auch eine nicht-abweisende Schleife er-

kennen. In Amiga-Basic gibt es dafür keinen direkt entsprechenden Befehl. Sie können diese Schleifenart, bei der der im »Schleifenkörper« liegende Programmteil mindestens einmal abgearbeitet wird, aber mit folgender Befehlskombination simulieren:

```
begin:
  (Schleifenkörper)
IF (Bedingung) GOTO be
gin
```

Diese Simulation ist übrigens einer der wenigen Ausnahmefälle der sauberen strukturierten Programmierung, der die Benutzung des »GOTO«-Befehles erlaubt. Ansonsten ist der »GOTO«-Befehl nur in Sonderfällen zur Beendigung des Programms gestattet.

tet. Aber keine Sorge, wenn Sie sauber strukturiert programmieren, werden Sie den »GOTO«-Befehl nicht vermissen.

Schlechte Zeiten für »GOTO«

Auch auf den Befehl »IF...GOTO« können Sie (abgesehen von obiger Ausnahme) in Zukunft verzichten. Die Befehlsfolge »IF...THEN...ELSE...END IF« bietet Ihnen dafür mehr als einen vollwertigen Ersatz.

Somit verbleiben noch drei Struktogrammtypen aus der Tabelle 1 zu besprechen. Die Bedeutung des Struktogrammrahmens wird sich später, bei der Besprechung von Unterprogrammen, automatisch ergeben. Kommen wir also zur mehrseitigen Alternative. Die zugehörige Befehlsfolge in Amiga-Basic lautet:

```
IF (Bedingung 1) THEN
  (Befehlsblock 1)
ELSEIF (Bedingung 2)
  THEN (Befehlsblock 2)
ELSEIF (Bedingung 3)
  THEN (Befehlsblock 3)
.
.
.
ELSE
  (Befehlsblock n)
END IF
```

Hier wird nun zuerst die Bedingung 1 geprüft. Trifft diese zu, so wird Befehlsblock 1 abgearbeitet und danach wird mit der Befehlsausführung nach dem »END IF« fortgefahren. Trifft die Bedingung dagegen nicht zu, so wird Bedingung 2 überprüft und entsprechend der Befehlsblock 2 ausgeführt oder nicht. Trifft keine der angegebenen Bedingungen zu, so wird die nach dem »ELSE«-Befehl stehende Befehlsfolge zur Ausführung gebracht. In der mehrseitigen Alternative können beliebig viele »ELSEIF«-Anweisungen stehen und die »ELSE«-Anweisung kann, muß aber nicht vorhanden sein. Um Rechenzeit zu sparen, ist es zu empfehlen, die Bedingungen nach der Wahrscheinlichkeit ihres Eintretens zu ordnen. Denn wird die wahrscheinlichste Bedingung als erste Bedingung gesetzt, ist es auch am wahrscheinlichsten, daß der erste Befehlsblock ausgeführt wird. Und da danach die Befehlsausführung nach dem »END IF« fortgesetzt wird, wird die Zeit für die nachfolgenden »ELSEIF«-Abfragen gespart.

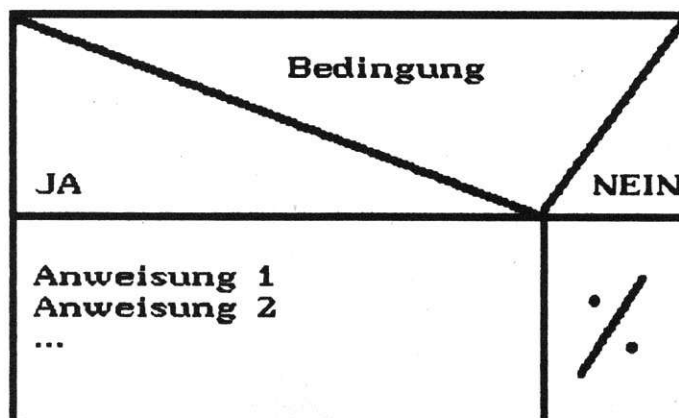


Bild 7. Der modifizierte Verzweigungsblock

»Abweisende« Schleifen

Zuletzt nun zur (abweisen- den) Schleife mit Abbruchbe- dingung. Auch aus einer Schleife kann man im allge- meinen nur mit Hilfe eines »GOTO«-Befehles springen. Entsprechend dem oben Ge- sagten sollte dies nur in Aus- nahmefällen geschehen. Eine der akzeptablen Möglichkeiten ist wieder der gewollte Pro- grammabbruch im Falle eines Fehlers:

```
WHILE (Bedingung)
    (Befehlsblock 1)
IF (Fehlerbedingung)
    GOTO Marke
    (Befehlsblock 2)
WEND
(weitere Befehle)
Marke:
(Fehlerbehandlung)
```

Günstiger als das »IF (Feh- lerbedingung) GOTO Marke«- Konstrukt ist der Sprung zur Fehlerbehandlung mit »ON ERROR GOTO«, dazu aber später.

Vielleicht fragen Sie sich, warum dem »GOTO«-Befehl hier so wenig Bedeutung zuge- messen wird. Die Antwort ist einfach: man braucht ihn nicht. Das Konzept der strukturierten Programmierung erlaubt, wie Sie auch an unserem Beispiel sehen konnten, einzelne Pro- grammtteile unabhängig von- einander zu entwickeln, zu tes- ten und erst dann zu einem vollständigen Programm zu- sammenzusetzen. Will man nun einen dieser Programmtile verbessern, so muß auch wirklich nur dieser abgeändert werden, während Sie alle an- deren Teile unverändert las- sen.

Dieser unbestreitbare Vorteil wird jedoch im allgemeinen

durch die Benutzung des »GO- TO«-Befehles zunichte ge- macht, da es dieser gestattet, von einem Strukturblock in ei- nen anderen zu springen. Da- her werden die einzelnen Blöcke durch den »GOTO«- Befehl so miteinander verwor- ben, daß normalerweise bei je- der Programmänderung das gesamte Programm umge- schrieben werden muß. Außer- dem trägt das »wilde« Umher- springen in einem Programm nicht gerade zu besseren Übersichtlichkeit bei.

Reine Äußerlichkeiten

Wie baut man nun ein Basic- Programm idealerweise auf? Zuerst sollte damit begonnen werden, den im Hauptpro- gramm verwendeten Variablen Anfangswerte zuzuweisen. Der Sinn dieser Maßnahme liegt darin, alle benutzten Va- riablen dem Interpreter vor der eigentlichen Programmaus- führung bekannt zu machen, um so Rechenzeit zu sparen. Zusätzlich erhalten Sie da- durch schnell eine Übersicht über alle verwendeten Varia- blen.

Weiterhin ist es vorteilhaft, einzelne Variablen als Spei- cher für Konstanten zu verwen- den. Ein Beispiel: Nehmen wir an, Sie haben ein Programm geschrieben, das formatierte Ausgaben auf dem Bildschirm liefert. Das Programm benötigt an verschiedenen Stellen die Information, wie viele Zeichen in eine Zeile passen. Wenn Sie einen hochauflösenden Moni- tor besitzen, werden dies 80 Zeichen pro Zeile sein. Doch was passiert, wenn nun die Ausgabe auf einem Moni- tor/Fernseher erfolgen soll, der nur 60 Zeichen pro Zeile dar- stellen kann? Sie müssen das

Programm nach jenen Stellen durchforsten, wo dem Pro- gramm die darstellbare Zeilen- breite mitgeteilt wird, und dort mühsam entsprechend än- dern. Wesentlich einfacher wird die gesamte Operation, wenn Sie am Anfang des Pro- gramms eine Variable mit der entsprechenden Zeilenbreite belegen. Bei einer Änderung braucht später nur deren Wert angepaßt werden.

Nach den Variablen sollten die Funktionen definiert wer- den. Genauer dazu später. Anschließend kann das eigent- liche Programm folgen. Dieses sollte am besten mit einer Sprungmarke (Label), zum Beispiel »Main«, eingeleitet werden.

Nach der »END«-Anweisung des Hauptprogramms folgen dann die Unterprogramme. Ein typisches Programmgerüst sieht also wie in Listing 6 aus.

Unterprogramme als logische Einheiten

Wobei handelt es sich nun eigentlich bei Unterprogram- men. Bei der Entwicklung eines Programmes mit Hilfe des Top-Down-Design beschrei- ben Sie den Programmablauf erst grob und verfeinern diesen Ansatz dann immer mehr. Häu- fig erweist es sich als günstig, zusammengehörige Pro- grammtteile als eine logische Einheit aufzufassen. Um diese logische Zusammengehörig- keit zu betonen, ist es möglich diese Programmtteile als soge- nannte Unterprogramme aus- zuführen. In Amiga-Basic gibt es drei derartige Zusammen- fassungen: Funktionen, »unechte« und »echte« Unterpro- gramme:

Einer Funktion in Basic kann eine Reihe von Werten überge- ben werden, mit denen sie Be- rechnungen zuführt und dann ein eindeutiges Ergebnis lie- fert. Dabei ist zu beachten, daß die Funktionsdefinition nicht länger als eine Programmzeile (80 Zeichen) sein darf. Neh- men wir als einfaches Beispiel eine Prozentrechnung: Sie ha- ben einen gewissen Verdienst und wollen wissen, wieviel Sie davon als Steuer dem Finanz- amt in den Rachen werfen dür- fen. In Ihrer Lohnsteuertabelle finden Sie leider nur den Ver- dienst und die Steuer in Pro- zent. Also schreiben Sie ein- fach eine Basic-Funktion, die für Sie rechnet:

```
DEF FNsteuer(Einkommen,
Prozent) = Einkommen *
Prozent / 100.0
```

Mit dem Befehl »DEF« teilen Sie dem Basic-Interpreter mit, daß jetzt die Definition einer Funktion folgt. Dabei muß jede von Ihnen definierte Funktion mit den beiden Buchstaben »FN« beginnen. Nach dem Na- men der Funktion folgen, in Klammern eingeschlossen und durch Kommata getrennt, die Parameter der Funktion. Die Namen dieser Platzhalter sind dabei Nebensache, es empfiehlt sich jedoch, aussa- gekräftigte Namen zu wählen. Nach der Aufzählung der Platzhalter geben Sie nach ei- nem Gleichheitszeichen an, was innerhalb der Funktion mit diesen Platzhaltern gesche- hen soll.

In Ihrem Programm können Sie dann etwa angeben:

```
MeinVerdienst = 30000
Abzug = FNsteuer(MeinVer-
dienst, 47.5)
```

Da Funktionen in Amiga-Ba- sic leider auf eine Zeile be- schränkt sind, muß man häufig auf ihre Anwendung verzich- ten. Hier wird dann auf Unter- programme ausgewichen.

Die »unechten« Unterpro- gramme, die mit Hilfe von »GO- SUB Label« aufgerufen wer- den, dürften Ihnen bekannt sein. Diese bieten die Möglich- keit, logisch zusammengehöri- ge Programmtteile an separater Stelle niederzuschreiben (sie- he auch Bild 8):

```
label:
    (Unterprogramm)
RETURN
```

Findet nun der Basic-Inter- preter im Programmablauf den Befehl »GOSUB label«, so ver- zweigt er zu dieser Marke, führt die danach folgenden Befehle aus, bis er auf den Befehl »RE- TURN« stößt. Dieser veranlaßt ihn, zu dem Befehl, von dem aus er das Label angesprun- gen hatte, zurückzuspringen und mit dem »GOSUB« folgen- den Befehl fortzufahren.

Mit »GOSUB« aufgerufene Unterprogramme haben je- doch einen entscheidenden Nachteil: Sie besitzen nicht die für »echte« Unterprogramme notwendige Unabhängigkeit. Ein Unterprogramm sollte von dem Programm, das es aufruft, so unabhängig wie nur irgend möglich sein. Dies ist notwen- dig, damit es einerseits auch in anderen Programmen sinnvoll eingesetzt und andererseits ohne Anpassungsschwierig- keiten durch ein neues, besse-

```
Variablendeklaration:
Monate% = 12 'Anzahl der Monate im Jahr
Wochen% = 52 'Anzahl der Wochen pro Jahr
PI      = 4 * ATN(1) 'Kreiszahl pi
DIM MonatsNamen$(12)
MonatsName$(1) = "Januar"
...
Funktionen:
DEF FNsinh(x) = 0.5 * ( exp(x) - exp(-x) ) 'sinus hyperbolicus
main:
    'Dieses Programm berechnet zu jedem Datum den Wochentag
    '(Programm)
END
Unterprogramm1:
    '(Anweisungen)
RETURN
SUB UnterProgramm2(x,y,z) STATIC
    '(Anweisungen)
END SUB
```

Listing 6. Ein typisches Programmgerüst

Hauptprogramm

Unterprogramm

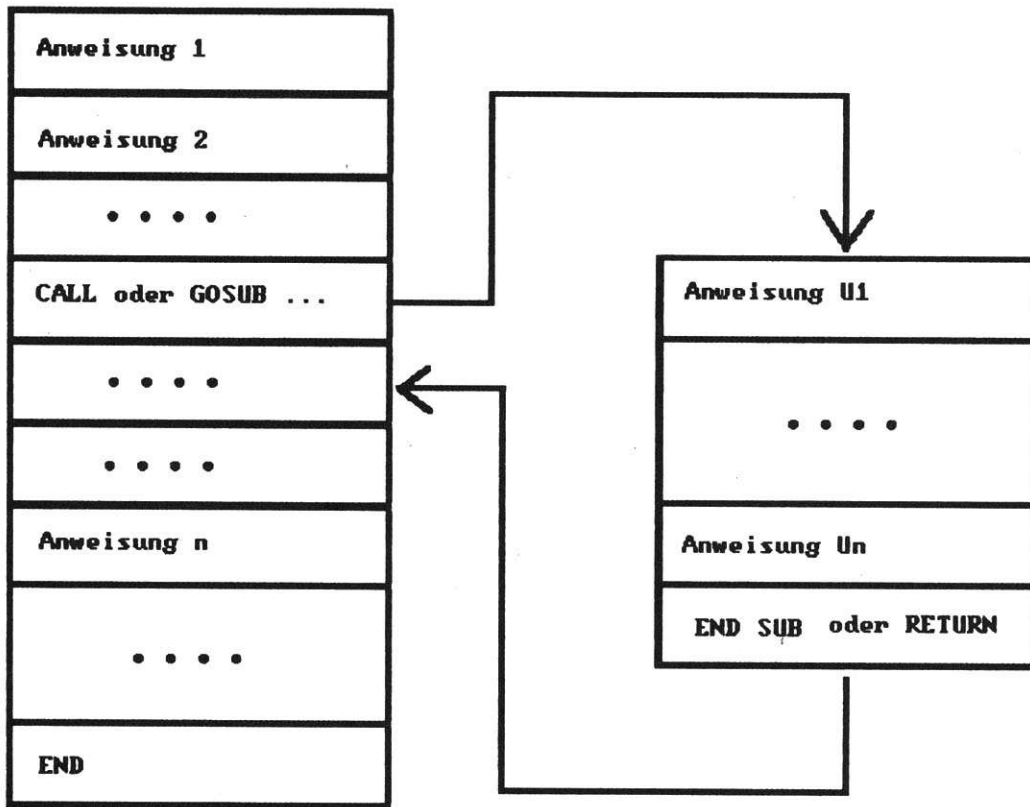


Bild 8. Der Aufbau von Unterprogrammen

res Unterprogramm ersetzt werden kann. Zur Veranschaulichung dient Listing 7. Es enthält zwei Unterprogramme: ein unechtes mit dem Namen »pseudo« und ein echtes mit dem Namen »echt«.

Die Unabhängigkeit vom »Hauptprogramm« ist bei mit »GOSUB« aufgerufenen Routinen nicht erfüllt, denn alle Variablen des Unterprogramms sind auch im übergeordneten Hauptprogramm bekannt und umgekehrt. Das hat folgende

Konsequenzen: Erstens: Bei der Entwicklung des Unterprogramms muß der Programmierer darauf achten, daß er nicht unbeabsichtigt Variablenamen verwendet, die schon im Hauptprogramm benutzt werden. Das kann fatale Folgen haben: ein Unterprogramm ändert Variablenwerte, die im Hauptprogramm für eine andere Aufgabe vorgesehen sind. Die Fehlersuche in solchen Fällen ist in der Regel sehr aufwendig. Zweitens: Die Variablenübergabe an solche Unterprogramme ist sehr umständlich.

Ein einfaches Beispiel: Mit dem Unterprogramm

```
printxy:
    LOCATE Zeile%,Spalte%
    PRINT Text$
RETURN
```

lassen sich beliebige Texte an einer beliebigen Stelle des Bildschirms ausgeben. Vor dem Aufruf der Routine müssen den Variablen Zeile%, Spalte% und Text\$ natürlich noch entsprechende Werte zugewiesen werden. Das an die Variablenamen angehängte »%«-Zeichen klassifiziert diese als ganzzahlig (kurze Ganzzahl), das angehängte »\$« steht für Stringvariable. Näheres hierzu finden Sie im Amiga-Basic-Handbuch.

Original und Fälschung

Um die Probleme, die durch gleiche Variablenamen entstehen können, zu beseitigen, haben die Entwickler des Amiga-Basic die Besonderheit vorgesehen, auch echte Unterprogramme mit »lokalen« Variablen zu verwenden. Die Bezeichnung »lokal« bedeutet in diesem Zusammenhang, daß diese Variablen nur dem Unterprogramm, nicht jedoch dem Hauptprogramm bekannt sind. So werden Namenskonflikte vermieden. Sie können sowohl in Ihrem Hauptprogramm als auch im Unterprogramm eine Variable mit dem Namen »x« definieren. Während »x« im Hauptprogramm beispielsweise den Wert Null hat, kann »x« im Unterprogramm Eins enthalten. Mit der Verwendung von lokalen Variablen können Sie sich eine Sammlung oft benötigter Unterprogramme aufbauen (sogenannte Programmbibliotheken) und diese nach Bedarf in Ihre Programme einfügen, ohne darauf achten zu müssen, ob diese Variablen mit gleichnamigen im Hauptprogramm kollidieren. Erst diese Arbeitsweise ermöglicht effizientes Programmieren.

Wie bringen Sie Ihrem Amiga nun bei, daß Sie fortan nur noch »echte« Unterprogramme schreiben möchten? Dazu genügt nicht mehr die Angabe einer Sprungmarke, wie bei den mit »GOSUB« aufgerufenen Unterprogrammen. Ein echtes Unterprogramm muß mit der Anweisung

```
SUB Unterprogrammname
( Parameterliste ) STATIC
```

beginnen. Obwohl die Variablen echter Unterprogramme in der Regel nichts mit denen des Hauptprogramms zu tun haben, müssen auf irgend eine Weise zu verarbeitende Werte übergeben werden. Dazu dient die Parameterliste. Sie enthält, ebenso wie bei den Funktionen, eine Aufzählung von »formalen« Variablen, an die die von Ihnen gewünschten Werte übergeben werden können. Der Name »formale Variable« rührt von der Tatsache her, daß diesen Variablen im Unterprogramm nur insofern Bedeutung zukommt, als sie als Platzhalter für die dann tatsächlich übergebenen aktuellen Variablen des Hauptprogramms dienen, mit denen die Berechnungen dann durchgeführt werden.

Entsprechend der »RETURN«-Anweisung bei den unechten Unterprogrammen bewirkt bei den echten die Anweisung »END SUB« das Beenden des Unterprogramms. Ein echtes Unterprogramm kann zusätzlich noch an beliebiger Stelle mit Hilfe des Befehles »EXIT SUB« verlassen werden.

Aufgerufen werden echte Unterprogramme mittels des Befehls »CALL«. Mit

```
CALL UprogName
( Parameterliste )
```

veranlassen Sie Ihren Computer, die Programmausführung mit dem genannten Unterprogramm fortzusetzen. Die formalen Parameter des Unterprogramms erhalten die Werte der in der Liste aufgeführten aktuellen Parameter.

Hier wird es nun notwendig, sich mit der Art der Parameterübergabe etwas genauer auseinanderzusetzen. Es sind hier zwei Arten der Parameterübergabe denkbar: zum einen die Übergabe durch Referenz, zum anderen die Übergabe als Wert. Diese Übergabeformen haben unterschiedliche Auswirkungen auf die Werte, die die übergebenen Parameter nach der Ausführung des Unterprogramms annehmen können. Besprechen wir also zunächst die von Amiga-Basic

```
Variable:
a = 1 : b = 2 : c = 3
x = 4 : y = 5 : z = 6
main:
PRINT a,b,c,x,y,z : PRINT
GOSUB pseudo      : PRINT
CALL echt(a,b,z)  : PRINT
GOSUB pseudo      : PRINT
x = 400 : y = 500 : c = 300
echt z,x,(x)      : PRINT
GOSUB pseudo      : PRINT
END
pseudo:
PRINT a,b,c,x,y,z
RETURN
SUB echt(x,y,z) STATIC
    SHARED a,c
    b = 20
    z = 30
    PRINT a,b,c,x,y,z
    a = 10
END SUB
```

Listing 7. Lokale und globale Variablen

praktizierte Methode: die Übergabe durch Referenz.

Bei diesem Verfahren wird dem Unterprogramm nicht der Inhalt der übergebenen Variable mitgeteilt, sondern die Speicherstelle, in der sich dieser Wert befindet. Dies hat zwei Konsequenzen: erstens können derartige Unterprogramme nicht rekursiv (aus sich selbst heraus) aufgerufen werden.

Der Ruf nach Unterprogrammen

Daher muß in Amiga-Basic ein Unterprogramm immer vom Typ »STATIC« sein. Die zweite — und wichtigere — Konsequenz ist, daß das Unterprogramm durch diese Übergabeart in die Lage versetzt wird, die aktuellen Parameter zu verändern; das heißt die aktuellen Parameter können nach dem Unterprogrammaufruf einen anderen Wert haben als vorher. Dagegen wird bei der direkten Übergabe des Wertes Wert in eine neue, vom Unterprogramm bereitgestellte Speicherzelle geschrieben. Das bedeutet, daß sich alle Änderungen des Variablenwertes auf diese Speicherzelle beziehen und die Speicherzelle, in der die übergebene Variable im Hauptprogramm gespeichert ist, nicht beeinflusst wird. Also haben nach einem Unterprogrammaufruf, bei dem die Parameter als Wert übergeben wurden, die Parameter nach dem Aufruf die gleichen Werte wie vorher.

Basic überlistet

In einigen Fällen kann es Ihnen durchaus wünschenswert erscheinen, die Parameter als Wert und nicht durch Referenz zu übergeben. Hier läßt sich der Basic-Interpreter nun durch einen einfachen Trick überlisten. Es ist Ihnen vielleicht schon aufgefallen, daß Sie an die eingebauten Basic-Funktionen nicht unbedingt eine Variable übergeben müssen. Es ist auch möglich, einen Ausdruck, wie etwa »a+b«, an eine Funktion zu übergeben. Auch bei den von Ihnen geschriebenen Unterprogrammen ist dies möglich. Die Vorgehensweise von Amiga-Basic ist dabei wie folgt: Es wird zuerst der Wert des Ausdruckes berechnet (in unserem Beispiel also »a+b«) und nachdem dieser Wert von Ihnen an keine Variable übergeben wird, rich-

tet Amiga-Basic eine temporäre Speicherstelle ein, in der dieser Wert abgelegt wird. Die Adresse dieser Speicherstelle wird dann an das Unterprogramm übergeben. Genau hier können Sie einhaken, indem Sie die zu übergebende Variable in Klammern einschließen. So gaukeln Sie Amiga-Basic vor, daß sich innerhalb der Klammern ein zu berechnender Ausdruck befindet. Für diesen Ausdruck wird dann eine temporäre Speicherstelle reserviert, in die der berechnete Wert eingelesen wird.

Nachdem in unserem Fall aber nichts zu berechnen ist, wird dorthin einfach der Wert der Variable geschrieben. Anstatt der richtigen Adresse der Variablen wird im Unterprogramm nun diese temporäre Adresse benutzt, mit der Folge, daß der Wert der eigentlichen Variable vom Unterprogramm nicht verändert werden kann.

Der unbestreitbare Vorteil der echten Unterprogramme sind lokale Variablen — trotzdem sind diese nicht immer erwünscht. Um dem Programmierer die Möglichkeit zu geben, bewußt Variablen zu definieren, die sowohl im Hauptprogramm, als auch im Unterprogramm bekannt sind, ist in Amiga-Basic der Befehl »SHARED« implementiert. In Computer-Chinesisch heißen derartige Variablen »global«, um ihren Gegensatz zu den lokalen Variablen herauszustellen. Geben Sie also in Ihrem Unterprogramm (am besten gleich nach der »SUB«-Anweisung) das Schlüsselwort »SHARED«, gefolgt von einer Variablenliste an, so besitzen die genannten Variablen im Unterprogramm denselben Wert wie auch im Hauptprogramm. Listing 7 soll dazu dienen, die bisher gewonnenen Kenntnisse an einem praktischen Beispiel zu erproben. Vor dem Lauf sollten Sie in einem »Trockentest« ermitteln, welche Werte auf dem Bildschirm ausgegeben werden. Gelingt Ihnen dies, dann haben Sie die Prinzipien von lokalen und globalen Variablen sowie von der Datenübergabe an Unterprogramme als Wert und durch Referenz verstanden. Wie Ihnen in diesem Listing sicher auffällt, haben wir den »CALL«-Befehl bei den Unterprogrammaufrufen weglassen. Dies können Sie immer dann gefahrlos tun, wenn keine Verwechslungen mit Sprungmarken zu befürchten sind. Allerdings darf die Parameterliste dann auch nicht in Klammern stehen.

Von Daten und Dateien

Sie haben nun Ihr erstes Standbein in Amiga-Basic erworben. Doch bekanntlich läuft's sich mit einem Bein ziemlich schlecht. Daher sollten Sie sich für weitere Fortschritte noch Kenntnisse über den Umgang mit Dateien aneignen. Aufbauend auf diesen beiden Grundpfeilern werden Sie sich dann auch sehr schnell mit den speziell auf die Hardware des Amiga zugeschnittenen Eigenschaften von Amiga-Basic anfreunden können.

Die Befehle »DATA«, »READ« und »RESTORE« sind Ihnen vielleicht schon bekannt, so daß wir sie hier nur der Vollständigkeit halber erwähnen (andernfalls lesen Sie bitte im Basic-Handbuch nach). Etwas interessanter sind schon die Befehle zum Umgang mit dem Dateisystem des Amiga. Der Befehl »FILES« zeigt Ihnen im Ausgabefenster die Files im aktuellen Directory an, während Sie mit dem Befehl »CHDIR <Directory>« das aktuelle Directory ändern können. Mit »NAME <Datei1> AS <Datei2>« weisen Sie dem File <Datei1> den neuen Namen <Datei2> zu. Dagegen löschen Sie mit »KILL <Datei>« ein File von der Diskette. Dabei ist zu beachten, daß alle Argumente der genannten Befehle wie Strings behandelt werden. Insbesondere müssen die Argumente daher in Anführungszeichen eingeschlossen werden.

Im wesentlichen müssen Sie drei Arten von Dateien unterscheiden: Programmdateien, sequentielle und relative Dateien. Diese drei Arten sollen nun in dieser Reihenfolge besprochen werden. Am kürzesten fällt dabei die Besprechung der Programmdateien aus: Diese können mit Hilfe des »LOAD«-Befehles geladen und mit »RUN« zur Ausführung gebracht werden. Auf Diskette gesichert werden diese Dateien mit Hilfe des »SAVE«-Befehles, wobei es aber einige Kleinigkeiten zu beachten gilt. Gibt man nur »SAVE <Dateiname>« an, so wird das Basic-Programm in einem speziellen Format abgespeichert. Wollen Sie dies verhindern, um etwa das Basic-Programm auch mit einem normalen Textverarbeitungsprogramm bearbeiten zu können, so müssen Sie »SAVE <Dateiname>,A« angeben. Das »A« steht dabei für ASCII.

Haben Sie ein Programm einmal im ASCII-Format gespeichert, so merkt dies der Interpreter bei Programmänderungen, das heißt beim Abspeichern eines im ASCII-Format vorliegenden Programmes ist die Angabe des »A« überflüssig. Geben Sie statt »A« ein »P« an, so läßt sich das Programm bei erneutem Laden nicht mehr listen. Dies bietet einen einfachen Programmschutz. Alle hier angegebenen Befehle zur Behandlung von Programmdateien können Sie im Befehlsfenster angeben, aber wahrscheinlich ziehen Sie die Anwahl dieser Befehle im »Project«-Menü mit Hilfe der Maus vor.

Daten in Serie

Kommen wir also nun zu den sequentiell oder auch seriell genannten Dateien, denn schließlich wollen Sie nicht nur Programme auf Diskette speichern, sondern auch Daten, mit denen die Programme arbeiten können. Der Name »serielle« Datei rührt von der Tatsache her, daß bei diesen Dateien die vorhandenen Daten nur nacheinander zu lesen sind. Sie können sich dies etwa wie bei einem Tonband vorstellen: Wenn sich Ihr Lieblingslied in der Mitte des Bandes befindet, dann müssen Sie eben gegebenenfalls bis dorthin vorspulen (Bild 9). Das erklärt auch, weshalb Sie mit seriellen Dateien im wesentlichen nur drei Dinge erledigen können: Die Informationen in der Datei nacheinander einlesen, überschreiben oder aber Daten an das Ende der Datei anhängen.

Dagegen ist es nicht möglich, irgendwelche Informationen irgendwo in der Mitte der Datei einzufügen oder zu löschen, ohne die gesamte Datei in den Speicher zu holen und nach erfolgter Änderung wieder vollständig neu auf Diskette zu schreiben. Wie Sie sich vorstellen können, ist dies ein zeitaufwendiges Unterfangen. Zugriffen wird auf serielle Dateien mit Hilfe des »OPEN«-Befehles, der folgende Syntax besitzt:

```
OPEN <Dateiname> FOR  
<Modus> AS <Datei  
nummer>
```

Damit können Sie Ihrem Amiga erst einmal die von Ihnen gewünschte Datei bekanntgeben. Dabei ist <Dateiname> der Name des entsprechenden Files, für <Modus> können Sie eines der drei Schlüsselwörter (ohne

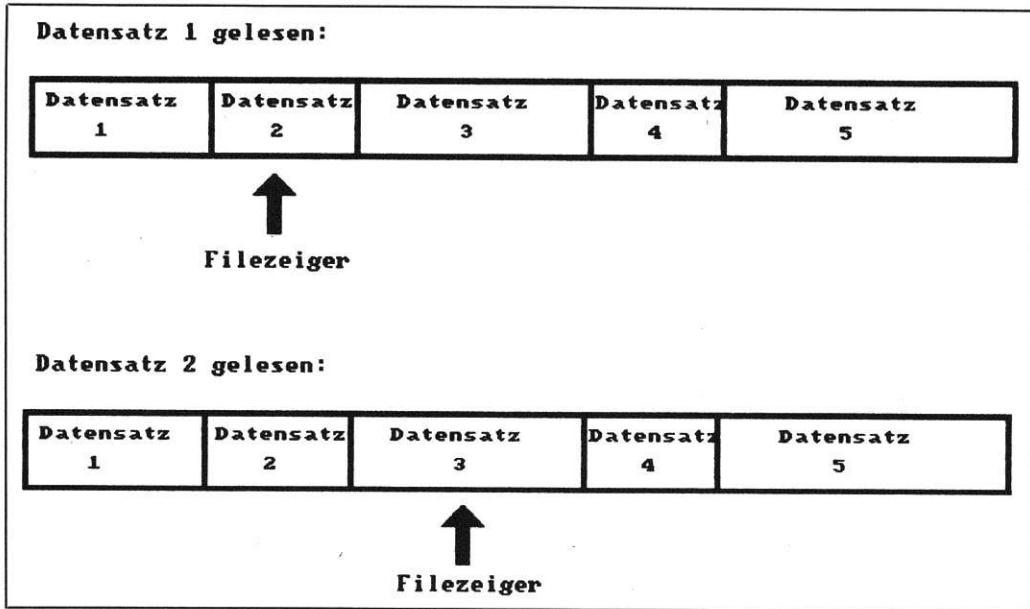


Bild 9. Das Lesen von seriellen Dateien

Anführungszeichen) einsetzen: »INPUT« falls Sie von der Datei Daten einlesen wollen, »OUTPUT«, falls Sie eine neue Datei eröffnen oder eine alte Datei überschreiben wollen und schließlich »APPEND«, falls Sie Daten an eine bestehende Datei anhängen wollen. Sollte im letzten Fall die Datei noch nicht existieren, so wird eine Datei mit dem entsprechenden Namen angelegt. Die weiteren, noch zu besprechenden Ein- und Ausgabebefehle erfordern die Angabe einer Dateinummer, die Sie hier im »OPEN«-Befehl einer bestimmten Datei zuweisen.

Der Unterschied dieser Ein- und Ausgabebefehle zu den entsprechenden Befehlen zur Ausgabe auf den Bildschirm und zur Eingabe von der Tastatur besteht eigentlich nur darin, daß Sie zusätzlich angeben müssen, welche Datei von den Befehlen betroffen sein soll. Dazu folgt dem jeweiligen Bildschirm- oder Tastaturbefehl einfach die im OPEN-Befehl zugeordnete Dateinummer. Um diese Nummer von einer normalen Zahl unterscheiden zu können, wird sie durch ein vorausgehendes Doppelkreuz gekennzeichnet.

Während also zum Beispiel der Befehl »PRINT 2,a\$« die Zahl 2, gefolgt vom Textstring »a\$« auf den Bildschirm ausgibt, schreibt der Befehl »PRINT #2,a\$« den Textstring »a\$« in die Datei, der durch den »OPEN«-Befehl die Nummer 2 zugeordnet wurde.

Bis jetzt haben wir klar getrennt zwischen einer Datei, die ein Basic-Programm enthält und einer sequentiellen Datei, die Daten enthält. Falls

Sie Ihre Programmdateien im ASCII-Format abgespeichert haben, ist diese Unterscheidung künstlich, wie Ihnen Listing 8 verdeutlicht. Da Sie dieses Beispielprogramm nur zu eigenen Experimenten anregen soll, werden wir es nur kurz erläutern. Das Programm öffnet eine serielle Datei in der RAM-Disk und schreibt in diese eine Reihe von Strings. Diese Strings ergeben wiederum ein Basic-Programm. Der »CHAIN«-Befehl am Ende unseres Programms bewirkt, daß dieses durch das in der seriellen Datei gespeicherte überschrieben wird. Nun enthält aber auch dieses Programm am Ende einen »CHAIN«-Befehl, mit dessen Hilfe sich dieses Programm selbst erneut in den Speicher lädt und startet. Ein Zähler zeigt an, wie oft dies bereits geschehen ist. Somit haben Sie hier ein schönes Beispiel für eine Endlosschleife. Die wichtige Erkenntnis, die aus diesem Beispiel zu ziehen

ist, ist jedoch folgende: Es besteht kein wesentlicher Unterschied zwischen sequentiellen Dateien und im ASCII-Format abgespeicherten Programmdateien. Somit ist es möglich,

programmgesteuert Programme zu erstellen oder zu verändern (Für Profis: Mit Hilfe dieser Methode sollte zum Beispiel ohne größere Schwierigkeiten ein Preprozessor im Stil der Sprache C realisierbar sein). Etwas Wichtiges können Sie aus Listing 8 allerdings noch entnehmen: Jede Datei, die Sie mit »OPEN« geöffnet haben, sollten Sie mit Hilfe des »CLOSE«-Befehles auch wieder schließen. Dazu geben Sie nach dem »CLOSE« einfach die zugeordnete Dateinummer an. Geben Sie keine Dateinummer an, dann werden alle zur Zeit geöffneten Dateien geschlossen.

Ist alles relativ?

Eine anspruchsvollere Art Daten zu verwalten sind relative Dateien. Im Gegensatz zu seriellen Dateien ist es bei diesen möglich, auf Daten an beliebiger Stelle eines Files direkt zuzugreifen. Haben wir bei den seriellen Daten zur Veranschaulichung das Beispiel ei-

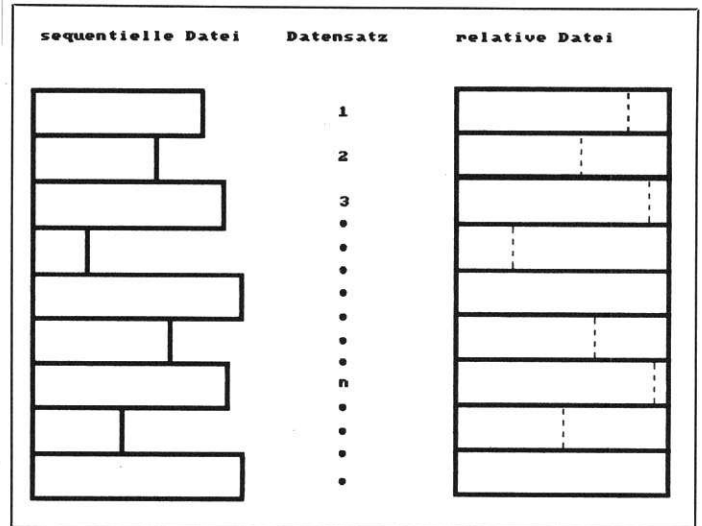


Bild 11. Die unterschiedlichen Datensatzlängen bei sequentiellen und relativen Dateien

```
Variable:
COMMON anzahl
anzahl = 0
wohin$ = "ram:katastrophe"
cr$ = CHR$(13) : REM jede Basic-Programmzeile muss mit
               : REM Wagenruecklauf abgeschlossen werden.

Variable$ = "COMMON anzahl"+cr$
ausgabe$ = "PRINT anzahl"+cr$
katastrophe$ = "CHAIN "+CHR$(34)+wohin$+CHR$(34)+cr$
wieoft$ = "anzahl = anzahl + 1"+cr$

main:
OPEN wohin$ FOR OUTPUT AS 1
PRINT #1,Variable$
PRINT #1,wieoft$
PRINT #1,ausgabe$
PRINT #1,katastrophe$
CLOSE #1
CHAIN wohin$ : REM Achtung! Endlosschleife
END : REM zu diesem Befehl gelangt Basic nie
```

Listing 8. Der Zugriff auf serielle Dateien

nes Tonbandes benutzt, so entspricht einer relativen Datei am ehesten eine Schallplatte (Bild 10): Wenn Sie hier Ihr Lieblingslied hören wollen, so bringen Sie den Tonarm einfach in die richtige Position und lassen dann das Lied abspielen. Sie müssen nur wissen, das wievielte Lied auf der Schallplatte Sie hören wollen. Ganz ähnlich verhält es sich mit den relativen Dateien: Den Liedern entsprechen »Datensätze«, die eine festgelegte Länge besitzen. Derartige Datensätze mit bestimmter, vorgegebener Länge wollen wir in Zukunft als »Record« bezeichnen. Um nun auf einen Record

zugreifen zu können, ist einfach dessen Nummer anzugeben. Bild 11 zeigt die unterschiedlichen Datensatzlängen bei sequentiellen und relativen Dateien. Unter einem Record können Sie sich ein Feld (Array) mit einer bestimmten Länge (in Byte) vorstellen.

Im Unterschied zu normalen Arrays, die entweder nur Integer-Zahlen, Strings oder Gleitpunkt-Zahlen enthalten können, können diese Datentypen in einem Record theoretisch gemischt sein. Die Betonung liegt hier auf »theoretisch«, denn bei Amiga-Basic funktioniert das nicht so einfach — aber dazu später mehr.

Gehen wir einmal davon aus, daß eine relative Datei eröffnet werden soll. Dies geschieht wieder mit dem »OPEN«-Befehl, allerdings ohne die Angabe eines Modus:

```
OPEN <Dateiname> AS
<Dateinummer> LEN=<
Recordlänge>
```

Dabei ist die Angabe der Recordlänge von besonderer Wichtigkeit, denn Amiga-Basic muß sich einen Datenpuffer dieser Länge anlegen, um einen vollständigen Record lesen oder schreiben zu können. Wie oben gesagt, kann ein Record theoretisch die unterschiedlichsten Datentypen (lange Ganzzahl, String...) enthalten. Dagegen wird der von Amiga-Basic angelegte Datenpuffer immer als String angesehen. Um also etwa eine doppelt genaue Gleitpunktzahl in einem Record abzuspeichern, muß diese erst einmal in eine Zeichenkette umgewandelt werden. Dabei ist es allerdings unnötig, jede einzelne Ziffer getrennt umzuwandeln, wie dies bei der »STR\$«-Funktion geschieht. Um die Recordlänge richtig angeben zu können, genügt es vollkommen, zu wissen, daß eine doppelt genaue Gleitpunktzahl in der Amiga-internen Darstellung 8 Byte benötigt (eine einfache genaue Gleitpunktzahl und eine doppelt genaue Ganzzahl benötigen jeweils 4 Byte, eine kurze Ganzzahl 2 Byte). Die Funktionen »MKI\$(<kurze Ganzzahl>«), »MKL\$(<lange Ganzzahl>«), »MKS\$(<einfache Gleitpunktzahl>« und »MKD\$(<doppelte Gleitpunktzahl>« interpretieren ihr Argument jeweils als String. Die umgekehrten Funktionen (»CVI«, »CVL«, »CVS«, »CVD«), bewirken das Gegenteil. Sie sollten sich unbedingt die Unterschiede zu den »STR\$«- und »VAL\$«-Funk-

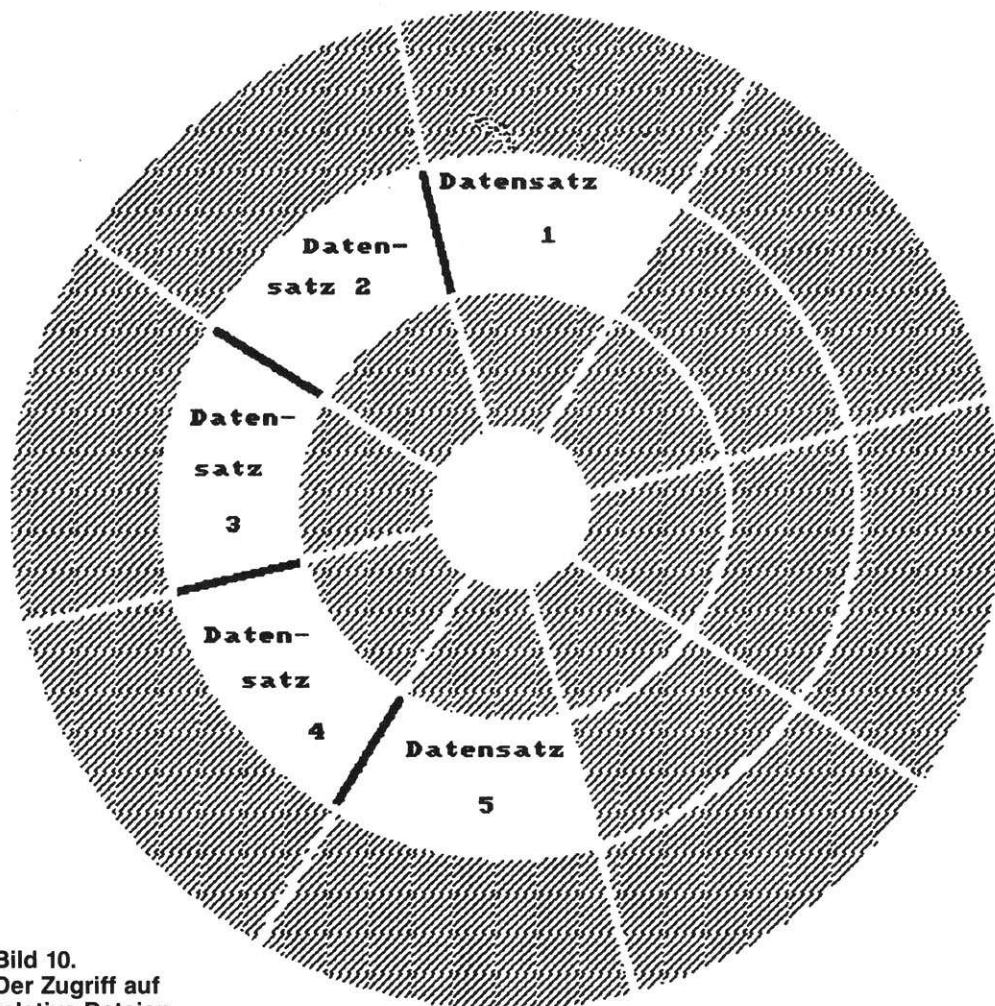


Bild 10.
Der Zugriff auf
relative Dateien

tionen vor Augen führen: Während diese beiden Funktionen ihre Argumente tatsächlich in einen String oder eine Zahl

umwandeln, werden bei den »MKX\$«- und »CVX«-Funktionen die Argumente nicht verändert, sondern nur uminter-

pretiert. Dies erkennen Sie auch daran, daß die mit der »STR\$«-Funktion in eine Zeichenkette umgewandelte Zahl »12345« eine Länge von 5 Byte besitzt, während das mit der »MKI\$«-Funktion erhaltene Ergebnis hingegen nur 2 Byte benötigt.

Nun wissen Sie zwar, wie Sie Zahlen als Strings interpretieren können, doch wie bringt man diese Strings in den Datenpuffer? Dazu ist es zunächst nötig, den Datenpuffer in unterschiedliche Bereiche aufzuteilen. Diesen Bereichen werden dann gezielt Strings und »Zahlen« zugeordnet. Dies geschieht mit Hilfe der »FIELD«-Anweisung, die Sie am besten immer direkt nach dem »OPEN«-Befehl angeben. Als Beispiel (siehe Listing 9) nehmen wir eine Personalkarte, die folgende Daten enthalten soll: Personalnummer (lange Ganzzahl), Nachname (maximal 20 Byte), Vorname (maximal 20 Byte), Gehalt (einfach genaue Gleitpunktzahl).

```
Variable:
Kennung% = 0      : REM Personalnummer, lange Ganzzahl = 4 Byte
Name$ = ""       : REM Nachname, String maximal 20 Byte
Vorname$ = ""    : REM Vorname, String maximal 20 Byte
Gehalt = 0       : REM Einkommen, Gleitkommazahl = 4 Byte
Recordnummer% = 1 : REM Schleifenvariable zum Recordzaehlen
Anzahl% = 2      : REM Anzahl der einzulesenden Records

main:
OPEN "RAM:Personal" AS 1 LEN=48 : REM Laenge = 4 + 20 + 20 + 4
FIELD #1, 4 AS PersID$, 20 AS NachNam$, 20 AS VorNam$, 4
AS Einkommen$
FOR Recordnummer% = 1 TO Anzahl%
INPUT "Kennnummer ";Kennung%
INPUT "Nachname ";Name$
INPUT "Vorname ";Vorname$
INPUT "Gehalt ";Gehalt
PRINT
LSET PersID$ = MKL$(Kennung%) : REM in Dateipuffer
schreiben
LSET NachNam$ = Name$
LSET VorNam$ = Vorname$
LSET Einkommen$ = MKS$(Gehalt)
PUT #1,Recordnummer% : REM Schreiben auf relative Datei
NEXT Recordnummer%
FOR Recordnummer% = 1 To Anzahl%
GET #1,Recordnummer% : REM Lesen von relativer Datei
PRINT "Kennnummer : ";CVL(PersID$)
PRINT "Nachname : ";NachNam$
PRINT "Vorname : ";VorNam$
PRINT "Gehalt : ";CVS(Einkommen$)
PRINT
NEXT Recordnummer%
CLOSE #1
END
```

Listing 9. Der Umgang mit relativen Dateien

```
FIELD #1, <Länge1>
AS <Name1$>,...,<
LängeN> AS <NameN$>
```

teilt also den Eingabepuffer in N Abschnitte auf, die dann unter den Namen <Name1\$> bis <NameN\$> als Strings angesprochen werden können. Diese Namen erwecken zwar auf den ersten Blick den Eindruck, als seien sie ganz normale Stringvariablen, jedoch sind beim Umgang mit ihnen einige Besonderheiten zu beachten: Da sie mit dem Datenpuffer der relativen Datei verbunden sind, dürfen ihnen keine Strings mit Hilfe des normalen Gleichheitszeichens zugewiesen werden. Tut man dies dennoch, so verlieren sie ihre Zuordnung zum Datenpuffer und werden zu normalen Stringvariablen.

Zuweisungen an Datenpuffervariablen haben daher mit Hilfe des »LSET«-Befehles oder des »RSET«-Befehles zu erfolgen. Der Unterschied zwischen diesen beiden Befehlen ist der, daß der erste die Daten linksbündig in den Puffer schreibt, während der zweite die Daten dem Puffer rechtsbündig zuweist.

Nachdem der Datensatz dann vollständig in den Puffer übernommen worden ist, kann Amiga-Basic mit Hilfe des »PUT«-Befehls, dem als Argumente die Dateinummer und die Recordnummer zu übergeben sind, den Record in die Datei schreiben. Umgekehrt kann mit Hilfe des »GET«-Befehles ein bestimmter Record aus einer Datei in den Datenpuffer eingelesen werden.

Nun wissen wir, wie man Daten in relative Dateien schreibt und wieder zurückholt. Doch wie findet man bestimmte Daten wieder? Die einzige Informationsquelle, die bis jetzt zur Verfügung steht, ist die Recordnummer. Um aber etwa in unserer Personaldatei den Namen Müller zu finden, erweist sich diese Nummer als wenig hilfreich. Auch die Records der Reihe nach einzulesen, erscheint wenig sinnvoll, denn dazu hätte eine serielle Datei auch genügt. Und dennoch sind serielle Dateien die Lösung unseres Problems. Der Trick dabei ist, sich eine serielle Datei anzulegen, die nicht den gesamten Record enthält, sondern nur ein Schlüsselwort aus jedem Record und die dazugehörige Recordnummer (ein sogenannter Index). Durch die Beschränkung auf ein Schlüsselwort, wie etwa den Nachnamen, kann die serielle Datei im Regelfall so kurz gehalten werden, daß sie vollständig in einen Array eingelesen werden kann. In einen

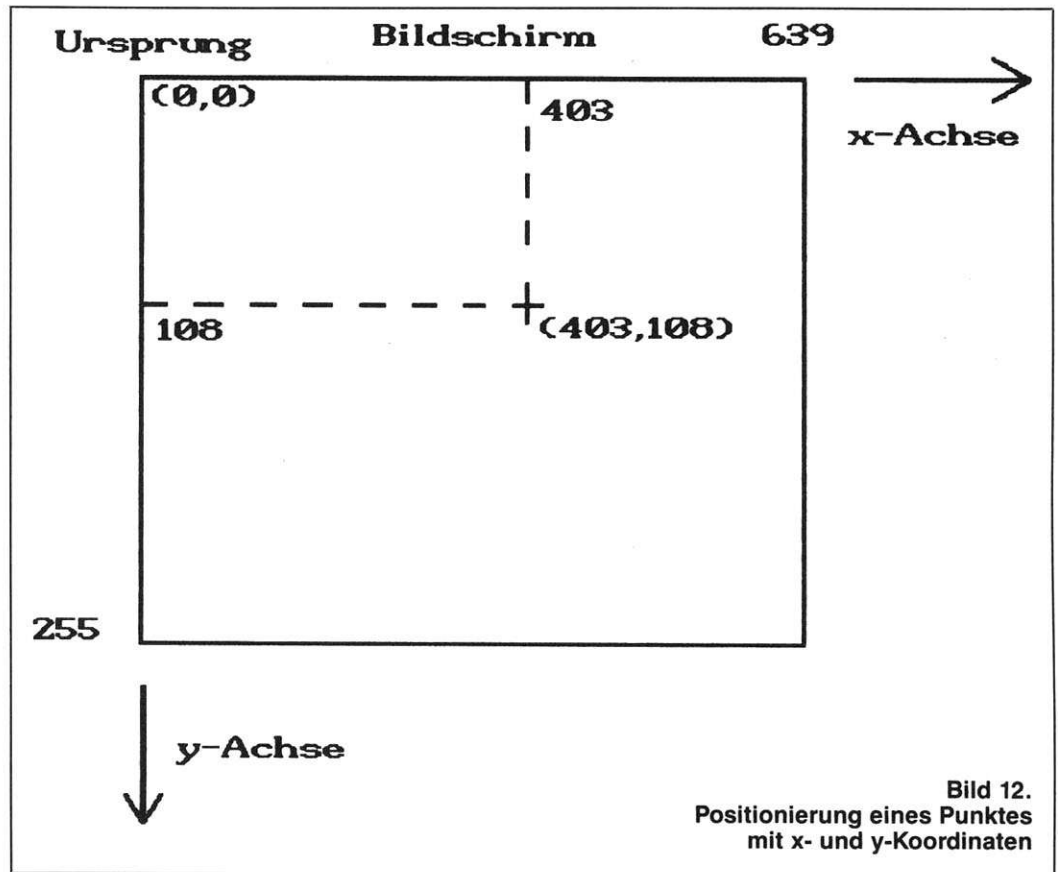


Bild 12.
Positionierung eines Punktes
mit x- und y-Koordinaten

zweiten Array werden gleichzeitig die zugehörigen Recordnummern eingelesen.

Und so wird der zu einem Nachnamen gehörige Record aufgefunden: Öffnen Sie die Index-Datei und lesen Sie diese in zwei Arrays (Index und Recordnummer) ein. Suchen Sie im Index-Array nach dem fraglichen Namen und merken Sie die Position des Namens im Array vor. Nun lesen Sie aus der relativen Datei den Record ein, der im zweiten Feld (dem mit den Recordnummern) an der gleichen Position wie der gefundene Name steht. Dieses Zugriffsverfahren heißt index-sequentiell, da der relative Datensatz gefunden wird, indem die Recordnummer mit Hilfe eines Schlüsselfeldes (der Arrayposition), in einer sequentiellen Datei bestimmt wird. Vielleicht ist Ihnen auch schon einmal die zugehörige (englischsprachige) Abkürzung »ISAM« über den Weg gelaufen.

Nun bleibt nur noch zu klären, wie viele Records in der relativen Datei enthalten sind, denn schließlich muß ja das Feld, in das die Schlüsselwörter eingelesen werden sollen, dimensioniert, also dessen Größe festgelegt werden. Hier kommt Ihnen die Tatsache zu Hilfe, daß Records eine bestimmte Länge haben. Nachdem die relative Datei geöffnet

wurde (als Datei mit der Nummer N), erhalten Sie mit Hilfe der Funktion »LOF(N)« die Länge der Datei in Byte und somit nach der Division dieser durch die Recordlänge die Anzahl der Records.

Grafik Punkt für Punkt

Nach den Basic-Befehlen der trockeneren Sorte wird es nun erheblich bunter. Wir kommen zu den zahlreichen »Bonbons«, die Amiga-Basic zu bieten hat. Bekannt geworden ist der Amiga vor allem durch seine Grafikeigenschaften. Diese sind auch bei Amiga-Basic entsprechend berücksichtigt und werden mit einer Reihe von Befehlen unterstützt.

Um eine grafische Darstellung auf den Bildschirm zu zeichnen, müssen die einzelnen Punkte des Bildschirms gesetzt werden. Der einfachste Befehl hierfür ist »PSET«. Die Koordinaten werden bezogen auf die linke obere Bildschirmcke angegeben (Bild 12). In Zukunft werden wir, entsprechend dem mathematischen Sprachgebrauch, die horizontale Bildschirmrichtung als x-Richtung und die vertikale Bildschirmrichtung als y-Richtung bezeichnen. Die linke, obere Ecke heißt dann »Ursprung des Koordinatensystems« und hat

die x-Koordinate 0 und die y-Koordinate 0 (0,0).

Mit PSET »(200,170)« können wir also einen Punkt setzen. Entfernen können wir diesen wieder mit PRESET (200,170). Durch das Setzen der entsprechenden Punkte können beliebige, einfarbige Grafiken erzeugt werden.

Eine mehrfarbige Grafik ist wesentlich ansprechender als eine einfarbige. Wie Sie vielleicht wissen, kann der Amiga selbst auf der Workbench vier Farben darstellen: drei Vordergrundfarben und den Hintergrund. Daher ist es möglich, sowohl beim »PSET«- als auch beim »PRESET«-Befehl nach der Koordinatenangabe, durch ein Komma getrennt, die Farbe anzugeben, mit der der Punkt gezeichnet werden soll. Sie können also angeben: »PSET (200,170), <Farbe>«, dabei ist <Farbe> eine ganze Zahl zwischen 0 und 3, entsprechend den vier auf der Workbench darstellbaren Farben. Durch den Befehl »POINT (<x>, <y>)« erhalten Sie die Farbe des Punktes mit den Koordinaten (x,y).

Jetzt wird's bunt

Mit diesen einfachen Befehlen lassen sich bereits sehr ansprechende Bilder erzeugen (Listing 10). Wir nehmen eine Bildschirmzeile mit einer Brei-

in der Titelleiste des Fensters. Werden keine Koordinaten angegeben, so besitzt das Fenster die Größe des Bildschirms, auf den man sich mit <Screennummer> bezieht. Der Parameter <Typ> gibt an, welchen Einfluß Mausbefehle auf das Fenster haben. Sie können damit festlegen, ob das Fenster mit der Maus geschlossen, in den Hintergrund gebracht, oder seine Größe verändert werden kann. Die unterschiedlichen Einstellungen schlagen Sie bitte in Ihrem Amiga-Basic-Handbuch nach.

Je nach dem gewählten Wert <Bitplanes> steht Ihnen eine mehr oder minder große Anzahl möglicher Farben zur Verfügung. Bis jetzt haben wir aber noch nicht festgelegt, welche Farben das sein sollen. Diese Angaben teilen wir dem Computer mit dem »PALETTE«-Befehl mit. Die Farben erhalten Kennnummern von 0 bis maximal 31 (Einschränkung: In der hohen Auflösung 640 x 512 sind nur 16 Farben möglich).

Die Farben auf dem Bildschirm werden im Prinzip genauso festgelegt, wie bei einem gewöhnlichen Fernseher: Aus den drei Grundfarben Rot, Grün und Blau (daher der Name RGB-Monitor) können durch Mischung nahezu beliebige Farben hergestellt werden. Könnte man nur wählen, ob etwa die rote, grüne oder blaue Farbe vorhanden sein soll oder nicht, so gäbe es nur acht verschiedene Farben. Der Amiga bietet jedoch die Möglichkeit, die Intensität der einzelnen Farben in 16er-Schritten zu variieren. Dies ergibt ei-

ne Anzahl von $16 \times 16 \times 16 = 4096$ möglichen Farben. Die Intensitätswerte für eine kleine Auswahl von Farben sind im Basic-Handbuch bei der Beschreibung des »PALETTE«-Befehls dargestellt. Die Syntax des Befehls ist demnach also:

PALETTE <Farbnummer> ,
<Rotanteil> , <Grünanteil> , <Blauanteil>

Mit dem Befehl

COLOR <Vordergrund> ,
<Hintergrund>

können Sie bestimmen, welche der mit »PALETTE« definierten Farben für Vorder- und Hintergrund des Bildschirms verwendet werden soll.

Noch mehr Grafikbefehle

Der einfachste der höheren Grafikbefehle kann mehr, als sein Name verheißt. Denn der »LINE«-Befehl zeichnet außer Linien auch Rechtecke, die zusätzlich mit einer Farbe ausgefüllt werden können. Mit der Anweisung

LINE (<Startpunkt>) -
(<Endpunkt>) , <Farbe>

zeichnet der Computer eine Gerade vom angegebenen Startpunkt (x1,y1) zum Endpunkt (x2,y2) in der angegebenen Farbe. Wird keine Farbe angegeben, dann wird die aktuelle Vordergrundfarbe benutzt. Werden keine Startkoordinaten angegeben, wird vom letzten gesetzten Punkt aus gezeichnet. Ergänzend kann

man noch, durch ein Komma von der Farbe getrennt, entweder den Parameter »B« (Box) oder den Parameter »BF« (Box filled) angeben. In diesem Fall werden die Anfangskoordinaten als linke obere und die Endkoordinaten als rechte untere Ecke eines Rechteckes interpretiert. Bei der Angabe von »B« wird nur ein Rechteckrahmen gezeichnet, mit »BF« wird das Rechteck zusätzlich mit der angegebenen Farbe ausgefüllt.

Aus Ellipsen werden Kreise

Das Zeichnen von Kreisen wird schon etwas komplizierter. Man erkennt das schon an der Anweisungssyntax:

CIRCLE <Mittelpunkt> ,
<Radius> , <Farbe> ,
<Start> , <Ende> ,
<Bildverhältnis>

Bevor wir auf die Feinheiten eingehen, sollte zunächst einmal die Bedeutung der Parameter klar sein. <Start> und <Ende> bezeichnen die Winkel im Bogenmaß (360 Grad = 2, Zählrichtung entgegen dem Uhrzeigersinn), bei dem das Zeichnen des Kreises begonnen und beendet werden soll.

So ist es möglich, auch beliebige Kreissegmente zu zeichnen. Sind die beiden Winkelangaben positive Zahlen, so wird nur der Kreisbogen gezeichnet. Bei negativen Winkelangaben wird dagegen der Kreisbogen noch mit einem Radius zum Mittelpunkt versehen. Dies ist wichtig, um gezeichnete Kreisbögen mit Farben füllen zu können.

Nun wird es etwas schwieriger: Die Punkte des Amiga-Bildschirmes sind keine Quadrate, sondern Rechtecke, deren Seiten im Verhältnis 2.25 : 1 stehen (das heißt, das Bildverhältnis beträgt 0,44). Dieser Wert ist auch für das Bildverhältnis voreingestellt, doch er gilt nur in der Auflösung mit 640 x 256 Bildpunkten, zudem ist er auch noch vom Monitor abhängig.

Arbeitet man in einer anderen Auflösung, so erhält man keine Kreise, sondern Ellipsen. Dies ist einerseits ganz nützlich, da man so durch die Angabe eines anderen Bildverhältnisses gezielt Ellipsen zeichnen kann, andererseits enthalten Grafiken so in der einen Auflösung Kreise und in einer anderen Ellipsen. Wenn Sie sich mit dieser Problematik näher auseinandersetzen wollen, so bietet Ihnen Listing 11 einige

```
Variable:
links = -10 : rechts = 10 : oben = -10 : unten = 10
radius = 0 'Kreisradius = Seitenlaenge des gleichseitigen Dreiecks
Hoehe = 0 'Hoehe des gleichseitigen Dreiecks
BVH = 0 'Bildverhaeltnis
phi = 4! * ATN(1!) / 3! 'Winkel 60 Grad = pi / 3
x1 = 0 : y1 = 0 : x2 = 0 : y2 = 0 : x3 = 0 : y3 = 0 'Dreieck
Eckpunkte
```

Listing 11. So werden Kreise immer rund

NEU!
auch als Bausatz

1.8MB

AMIGA 500

Die 1.8-MB-Karte ist supereinfach einzubauen:

- als 0.5-MB-Erweiterung auf 1 MB intern: AMIGA umdrehen, Speichereinbauelemente öffnen und Karte einstecken – die Garantie bleibt erhalten.
- als 1.8-MB-Erweiterung auf 2.3 MB intern: Gehäuse öffnen, GARY-Chip herausnehmen, Adapter in den GARY-Sockel einsetzen, GARY aufstecken – fertig!

DM 1197,-

Die 1.8-MB-Karte gibt es jetzt auch

- als Bausatz mit allen Teilen, Schaltplan und Bestückungsliste, jedoch **ohne** 1-MegaBit-Chips (511000) **DM 228,-**
- als Bausatz wie oben, jedoch fertig gelötet mit allen Teilen, ebenfalls **ohne** 1-MegaBit-Chips (511000) **DM 342,-**

NEU!

3.8MB

AMIGA 500

Für Unersättliche: Die 3.8-MB-Karte für 4.3 MB Mammutspeicher intern:

- als 1.8-MB-Erweiterung wie oben (bereits fertig mit 1-MegaBit-Chips bestückt) – zusätzlich mit weiteren 16 RAM-Chips (511000) selbst bestücken, einbauen und dann den zusätzlichen Speicher mit „Add-Mem“ (40000 – 5FFFF) einbinden – fertig!
- komplett bestückt mit 3.8 MB

DM 1698,-
DM 2428,-

Alle Karten sind bis 1.8 MB autokonfigurierend und mit gesockelten ICs (Ausnahme: Bausatz) sowie einer Echtzeituhr (akkugepuffert) versehen. Sie arbeiten bereits auch unter Workbench 1.3.

Aufgrund der enormen Nachfrage nach 1-MegaBit-Chips auf dem Weltmarkt liefern wir in der Reihenfolge der Bestellungen aus. **Ordern Sie also bitte rechtzeitig.** (Änderungen vorbehalten)

Die gigantischen Speicherkarten erhalten Sie nur im guten Fachhandel oder bei

NEU!

1.8MB

AMIGA 1000

Endlich! Die 1.8-MB-Karte für den 1000er:

- wird ganz einfach in den vorderen Speichererweiterungsschacht des Rechners gesteckt. Die bisherige 256 KB-Erweiterung wird abgezogen, die neue 1.8 MB-Karte wird aufgesteckt, dann einen Adapter in den CPU-Sockel, CPU aufstecken – fertig. 512 KB bleiben nach dem Abschalten erhalten.
- läuft absolut problemlos mit Sidecar und Festplatte

DM 1311,-

Gigatron G. Preuth, R. Tiedeken
(Entwicklung, Service & Versand)
Resthauser Str. 128, 4590 Cloppenburg
Telefon 04471/3070

FreeCom Wolfgang F.W. Paul
(Auslieferung & Service Raum Hamburg)
Bismarckstraße 2, 2000 Hamburg 20
Telefon 040/49 59 90


```

main:
begin:
  PRINT "Geben Sie die Seitenlaengen eines Rechteckes ein."
  INPUT "x,y "; x%,y%
  IF (x% > 640 OR y% > 512) OR (x% < 70 OR y% < 70) GOTO begin
  Initialize x%, y%, BVH
  Dreieck x%, y%
  Segment 2,5,3,4,x1+links,y1+unten,1
  Segment 5,0,4,1,5,0,x2+rechts,y2+unten,2
  Segment 1,2,1,2,0,3,x3,y3+oben,4
  Segment 1,2,2,3,3,4,x1,y3,5
  Segment 5,0,3,4,4,5,x3,y1+radius/2,3
  Segment 0,1,1,2,5,0,x2,y3,6
  Segment 0,1,2,3,4,5,x3,y1,7
  LINE (0,0)-(x%,y%),7,b
  Finalize
END
SUB Initialize( Breite%, Hoehe%, Bild ) STATIC
  aus = 0 : ein = 1 : Bild = 0
  hoch% = (Breite% \ 321) + 1 'hoch% ist 1 oder 2
  interlace% = 2 * (Hoehe% \ 257) 'interlace% ist 0 oder 2
  modus% = hoch% + interlace% 'modus% ist 1, 2, 3, oder 4
  ScreenX% = hoch% * 320 'x-Aufloesung ist 320 oder 640
  ScreenY% = (1 + .5 * interlace%) * 256 'y-Aufloesung 256 oder 512
  IF modus% = 1 THEN
    Bild = 2 * .5 'halb so viele Punkte in x-Richtung
  ELSEIF modus% = 2 THEN
    Bild = 1 * .5 'Normalaufloesung, Monitorabhän-
    gig, in den USA 0.44
  ELSEIF modus% = 3 THEN
    Bild = 4 * .5 'halb so viele Punkte in x, doppelt so viele in y
  ELSE
    Bild = 2 * .5 'doppelt so viele Punkte in y-Richtung
  END IF
  SCREEN 2,ScreenX%,ScreenY%,3,modus%
  WINDOW 2,"RGB",0,2 : WINDOW OUTPUT 2
  'Alle Kombinationen mit Rot, Gruen und Blau aus und ein
  PALETTE 0,aus,aus,aus : PALETTE 1,aus,aus,ein : PALETTE 2,aus,
    ein,aus
  PALETTE 3,aus,ein,ein : PALETTE 4,ein,aus,aus : PALETTE 5,ein,
    aus,ein
  PALETTE 6,ein,ein,aus : PALETTE 7,ein,ein,ein
END SUB
SUB Dreieck( Breite%, Hoehe% ) STATIC
  SHARED x1,y1,x2,y2,x3,y3,radius,BVH
  x3 = Breite% / 2!
  y1 = Hoehe% * 7! / 12!
  IF Breite% > Hoehe% * BVH THEN
    radius = Hoehe% / BVH / 4!
    h = SQR(3!) * radius / 2! * BVH
    x2 = x3 + radius / 2!
    x1 = x3 - radius / 2!
  ELSE
    radius = Breite% / 4!
    h = SQR(3!) * radius / 2!
    x2 = x3 + radius / BVH / 2!
    x1 = x3 - radius / BVH / 2!
  END IF
  y2 = y1
  y3 = y2 - h
END SUB
SUB Segment( von1%,bis1%,von2%,bis2%,von3%,bis3%,fx,fy,farbe% ) STATIC
  SHARED x1,y1,x2,y2,x3,y3,radius,BVH,phi
  CIRCLE (x1,y1),radius,farbe%,von1%*phi,bis1%*phi,BVH : REM Zeichne
    eine
  CIRCLE (x2,y2),radius,farbe%,von2%*phi,bis2%*phi,BVH : REM Flaechen,
    die
  CIRCLE (x3,y3),radius,farbe%,von3%*phi,bis3%*phi,BVH : REM aus
    drei
  PAINT (fx,fy),farbe% : REM Kreis
    segmenten besteht
END SUB
SUB Finalize STATIC
  WHILE NOT MOUSE(0) : SLEEP : WEND
  WINDOW CLOSE 2 : REM Warte auf Mausklick, dann schliesse Ausgabe
    fenster
  SCREEN CLOSE 2
END SUB

```

Listing 11. (Schluß)

Denkanstöße. Zum Verständnis unseres Kurses ist dies jedoch nicht erforderlich, daher wird hier auf eine Besprechung verzichtet.

Flächenfüllend

Nicht immer sind die Flächen, die Sie zeichnen wollen, Rechtecke und Ellipsen. Um

auch Flächen zeichnen zu können, die etwas unregelmäßiger aussehen, wurde der »AREA«-Befehl implementiert. Durch mehrere »AREA (<Punkt>)«-Anweisungen nacheinander legen Sie die Ecken eines Gebildes (mathematisch: eines Polygonzuges) fest, das mit Hilfe der »AREAFILL«-Anweisung mit der Vordergrundfarbe

ausgefüllt werden kann. Sie müssen sich allerdings nicht auf eine Füllfarbe beschränken, sondern Sie können sich mit Hilfe des »PATTERN«-Befehles auch eigene Füllmuster definieren. Genauso erlaubt der »PATTERN«-Befehl selbst-erstellte Linienmuster. Beim normalen Ziehen von Linien sind alle Punkte gesetzt. Es ist jedoch möglich, ein 16 Punkte langes Muster zu entwerfen, das zum Ziehen von Linien benutzt wird. Dabei gehen Sie am besten entsprechend Bild 14 vor. Jeder zu setzende Punkt besitzt eine bestimmte Wertigkeit. Addieren Sie einfach die Wertigkeiten der Punkte, die Sie setzen wollen, auf. Beachten Sie dabei, daß der erste Punkt eine negative Wertigkeit besitzt. Die errechnete Gesamtwertigkeit des Musters weisen Sie dann an eine kurze Ganzzahl zu, die Sie dem »PATTERN«-Befehl als Parameter übergeben. Ein Füllmuster definieren Sie, indem Sie mehrere Linien untereinander zeichnen und deren einzelne Wertigkeiten berechnen. Es gibt allerdings die Einschränkung: die Anzahl der Linien muß eine Potenz von 2 sein, daher sind nur 1, 2, 4, 8, 16, 32, und so weiter als Linienanzahl erlaubt.

Dimensionieren Sie ein entsprechend großes Feld aus kurzen Ganzzahlen und übergeben Sie es dem »PATTERN«-Befehl als zweiten Parameter:

```

PATTERN <Linie%>,
<Füllmuster%>

```

Die bis jetzt besprochenen Grafikbefehle dienten alle zum Zeichnen neuer Bilder, die folgenden bearbeiten dagegen bereits vorhandene Bilder. Da ist zum einen der »SCROLL«-Befehl:

```

SCROLL (<linke obere
Ecke>) - (<rechte
untere Ecke>),<delta
x>,<delta y>

```

Mit diesem Befehl wird der im angegebenen Rechteck liegende Bildausschnitt um <delta x> Punkte nach rechts und um <delta y> Punkte nach unten verschoben. Alle in Schieberichtung aus dem definierten Rechteck hinausgeschobenen Zeilen und/oder Spalten verschwinden damit vom Bildschirm. Durch geschickte Programmierung können mit diesem Verfahren Grafiken über den Bildschirm geschoben werden:

```

FOR i = 0 TO 100
  SCROLL (50+i,50)

```

```

-(150+i,150),1,0
NEXT i

```

Ein Nachteil dieser Technik ist das Löschen derjenigen Teile des Hintergrundes, über die der Bildausschnitt verschoben wird. Ist diese Wirkung nicht erwünscht, empfiehlt sich eine Befehlskombination aus »PUT« und »GET«. Beachten Sie, daß dies die Grafikversionen dieser Befehle sind, die mit den »PUT«- und »GET«-Befehlen relativer Dateien nichts zu tun haben. »PUT« und »GET« sind die wohl mächtigsten Grafikbefehle des Amiga-Basic. Um mit diesen Befehlen arbeiten zu können, wird eine Feldvariable benötigt, die genügend Elemente besitzt, um die komplette Bildinformation eines Ausschnittes zu speichern. Daran erkennt man schon, daß dieses Verfahren für ganze Bildschirmseiten nicht geeignet ist. Die Größe des benötigten Integer-Feldes läßt sich nach folgender Formel berechnen:

$$\text{<Größe>} = \text{<Höhe>} * \text{INT}((\text{<Breite>} + 16) / 16) * \text{<Anzahl der Bitplanes>} + 3$$

Nachdem Sie ein entsprechend großes Feld definiert haben, können Sie mit

```

GET (<linke obere Ecke>) - (<rechte untere
Ecke>), <Feldname%>

```

die Bildinformation in die Feldvariable einlesen und mit

```

PUT (<linke obere Ecke>), <Feldname%>

```

den Ausschnitt an einer beliebigen Stelle des Bildschirms plazieren. Schreiben Sie ihn jedoch wieder an die alte Stelle zurück, so verschwindet überraschenderweise das Bild. Das liegt daran, daß hinter dem Feldnamen beim »PUT«-Befehl noch ein Schlüsselwort angegeben werden kann, das bestimmt, in welcher Weise das gespeicherte Bild mit dem Bildschirm verknüpft wird. Voreingestellt ist hier der Verknüpfungstyp »XOR« (exklusives oder), der bewirkt, daß alle Bildpunkte, die zugleich im abzubildenden Ausschnitt und im Fenster gesetzt sind, gelöscht werden. Die anderen möglichen Typen für diese Einstellung sind zum einen »AND« und »OR«, zum anderen können Sie auch »PSET« und »PRESET« angeben. Die Wirkungsweise dieser Schlüsselworte können Sie anhand von Tabelle 2 ermitteln, wobei zu beachten ist, daß diese Operationen für jede Bitplane separ-

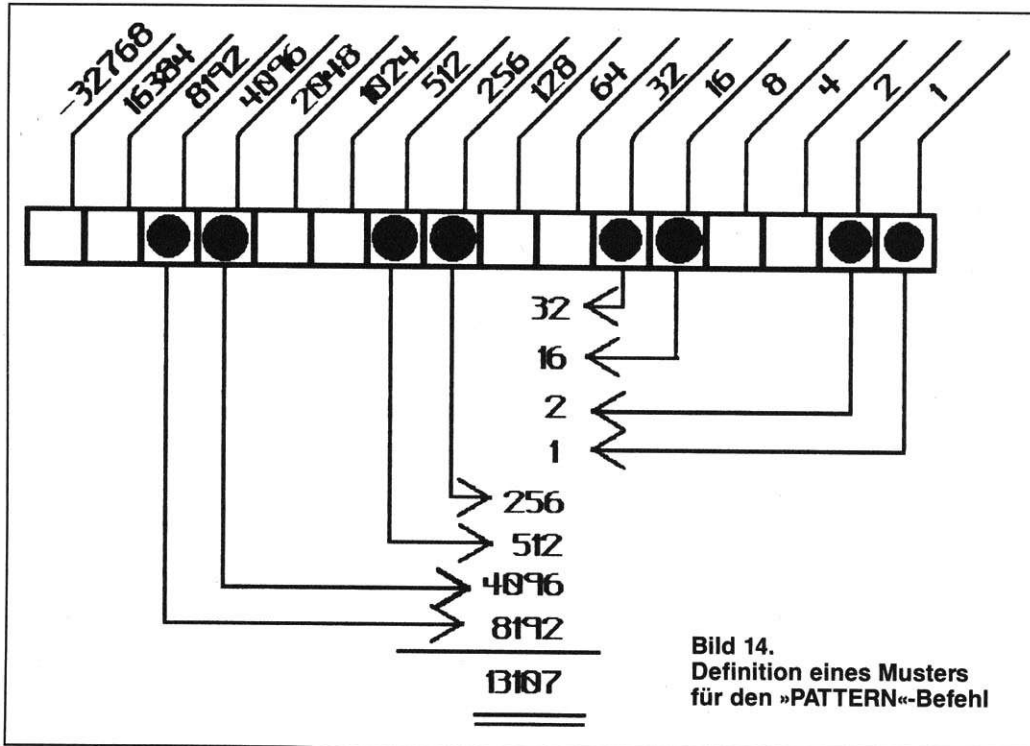


Bild 14.
Definition eines Musters
für den »PATTERN«-Befehl

rat durchgeführt werden. Da wir nun schon Zeichnungen in Feldvariablen abgespeichert haben, bietet es sich an, diese Feldvariablen und damit auch die Bilder auf Diskette zu speichern. Auch hier können wieder die »MKI\$«- und »CVI«-Befehle (siehe oben) von Nutzen sein:

```
OPEN <File> FOR OUTPUT
AS <Nummer>
FOR i = 0 TO <Größe>
  PRINT # <Nummer>
  >,MKI$(<Feldname%>
  >(i));
NEXT i
CLOSE <Nummer>
```

speichert das Bild auf Diskette, während

```
OPEN <File> FOR INPUT
AS <Nummer>
DIM <Feldname%> (LOF
(<Nummer>/2 - 1)
FOR i = 0 TO LOF(<
Nummer> / 2 - 1
  <Feldname%>
  (i) = CVI(INPUT$(2,
  <Nummer>))
NEXT i
CLOSE <Nummer>
```

ein ausreichend großes Feld dimensioniert und das Bild dann einliest. Der Befehl »INPUT\$(<Anzahl>,<Nummer>)« liest dabei aus der Datei mit der Kennzahl <Nummer> genau die Menge <Anzahl> Byte ein.

Listing 12 bietet Ihnen eine Demonstration der Wirkung der bis jetzt besprochenen Grafikbefehle.

Einfach animierend

Mit Hilfe des »SCROLL«-Befehles und der »GET«- und »PUT«-Anweisungen lassen sich bereits einfache Bewegungen auf dem Bildschirm darstellen. Doch wofür hat unser Amiga schließlich seine Spezialchips, die unter anderem eben auch auf die Darstellung von bewegten Objekten spezialisiert sind.

Damit Sie mit dem im folgenden erworbenen Wissen auch

gleich experimentieren können, sollten Sie sich mit dem Objekt-Editor des Amiga-Basic auskennen. Ist dies nicht der Fall, so lesen Sie bitte zunächst das Kapitel 7 Ihres Basic-Handbuches durch. Der Objekt-Editor »ObjEdit« wird dort ausführlich beschrieben. Wir wollen hier nur kurz auf seine Arbeitsweise eingehen: Nach dem Aufruf des Editors müssen Sie sich entscheiden, ob Bobs oder Sprites editiert werden sollen. Haben Sie Ihre Wahl getroffen, so können Sie die gewünschte Größe des Ob-

jektes mit der Maus einstellen. Die vorhandenen Funktionen erlauben ein komfortables und einfaches Erstellen der Objekte. Nach der Fertigstellung sollten Sie Ihr Objekt auf Diskette speichern. Die Datei enthält dann alle Informationen, wie Größe, Form und Farbe, die für die Darstellung benötigt werden.

Falls Sie den Unterschied zwischen Bobs und Sprites noch nicht kennen: Ein Bob (Blitter Object) ist praktisch ein Teil des Grafikspeichers, der vom Blitter (einem der oben erwähnten Spezialchips) blitzschnell von einer Position auf die andere kopiert werden kann. Dabei wird der jeweilige Hintergrund in einen anderen Speicherbereich gerettet. Durch das schnelle Kopieren ergibt sich der Eindruck einer fließenden Bewegung. Dadurch, daß Bobs ein Teil des Bildschirmspeichers sind, sind sie in ihrer Größe und Farbe theoretisch nicht eingeschränkt. Praktisch dagegen können zu große Bobs nicht mehr schnell genug umkopiert werden, so daß die bewegten Objekte zu flimmern beginnen.

Im Gegensatz zu Bobs sind Sprites Objekte, die von einem Spezialchip auf den Bildschirm eingeblendet werden. Sie benötigen daher keinen Platz im Bildspeicher und können sich wesentlich schneller als Bobs bewegen. Dafür sind Sprites allerdings in der Größe auf eine Breite von 16 Pixel beschränkt, und auch die Farbauswahl ist eingeschränkt.

Operation	Bild		
		XOR	
Bit-plane	Resultat		
		OR	
		AND	
		PSET	
		PRESET	

Tabelle 2. Wahrheitstafeln zum »PUT«-Befehl


```
Variable:
true = 1
breite% = 620 ' Seiten eines Rechtecks
hoehe% = 240
n% = 4 ' Unterteile Rechteck in 2 * 4 Gitter
m% = 2
radius% = breite% \ (2*n%+1) ' Zum Zeichnen in Untergitter
h% = hoehe% \ (2*m%+1)
phi = 4! * ATN(1!) / 3! ' Winkel 60 Grad = pi / 3
speicher% = (h%+1) * INT((radius% + 16)/16) * 3 + 3 ' Bilder merken
DIM image1%(speicher%)
DIM image2%(speicher%)
DIM fill%(7) ' Fuellmuster im PATTERN-Befehl
fill%(0) = -31807 : fill%(1) = 7800 : fill%(2) = 28686 : fill%(3) =
-32383
fill%(4) = -32383 : fill%(5) = 28686 : fill%(6) = 7800 : fill%(7) =
-31807
DIM default%(1) : default%(0) = -1 : default%(1) = -1
DEF FNmitteX( i ) = (i-.5)/n%*breite% ' x-Mittelwert eines
Untergitters
DEF FNmitteY( i ) = (i-.5)/m%*hoehe% ' y-Mittelwert eines
Untergitters

main:
Initialize
Gitter n% , m% , 7
PATTERN 13107
LINE ( FNmitteX(.75),FNmitteY(.75) )-(FNmitteX(1.25),
FNmitteY(1.25)),3,b
PATTERN -1
CIRCLE(FNmitteX(2),FNmitteY(1),radius%,1,,.5
PAINT (FNmitteX(2),FNmitteY(1)),1
CIRCLE(FNmitteX(3),FNmitteY(1),radius%,2,3*phi,phi,.75
CIRCLE(FNmitteX(4),FNmitteY(1),radius%,4,-3*phi,-phi,.25
PATTERN -1,fill%
COLOR 5,6
AREA (FNmitteX(1),FNmitteY(1.75)) : AREA STEP (radius%,h%)
AREA STEP (-2*radius%,0) : AREA FILL
PATTERN -1,default%
COLOR 7,0
x% = INT(FNmitteX(2)) : y% = INT(FNmitteY(2))
LINE (x%,y%)-(x%+radius%,y%+h%),2,bf
GET (x%,y%)-(x%+radius%,y%+h%),image1%
LINE (x%-radius%,y%-h%)-(x%,y%),6,bf
GET(x%-radius%,y%-h%)-(x%,y%),image2%
x% = INT(FNmitteX(3))
PUT (x% - radius%/2,y%-h%/2),image2%,PSET
PUT (x%,y%),image1%,XOR
PUT (x%-radius%,y%-h%),image1%,OR
PUT (x%,y%-h%),image2%,AND
x% = INT(FNmitteX(3.75))
WHILE true ' warte auf Mausklick
PUT (x%,y%-h%),image1%,PRESET
PUT (x%,y%-h%),image2%,AND
FOR i = 1 TO 1.5*radius%
IF MOUSE(0) THEN true = 0
SCROLL (x%,y%-h%)-(x%+1.5*radius%,y%+h%),1,1
NEXT i
WEND
Finalize
END

SUB Initialize STATIC
aus = 0
ein = 1

SCREEN 2,640,256,3,2
WINDOW 2,"Grafik-Demo",,0,2 : WINDOW OUTPUT 2
'Alle Kombinationen mit Rot, Gruen und Blau aus und ein
PALETTE 0,aus,aus,aus : PALETTE 1,aus,aus,ein
PALETTE 2,aus,ein,aus : PALETTE 3,aus,ein,ein
PALETTE 4,ein,aus,aus : PALETTE 5,ein,aus,ein
PALETTE 6,ein,ein,aus : PALETTE 7,ein,ein,ein
END SUB

SUB Gitter( n% , m% , farbe% ) STATIC
SHARED breite% , hoehe%
1 = 0

LINE (1,1)-(breite%,hoehe%),7,b
FOR i = 1 TO breite% STEP breite%/n%
LINE (1,1)-(1,hoehe%),7
NEXT i
FOR i = 1 TO hoehe% STEP hoehe%/m%
LINE (1,i)-(breite%,i),7
NEXT i
END SUB

SUB Finalize STATIC
WINDOW CLOSE 2
SCREEN CLOSE 2
END SUB
```

Listing 12. Unsere Grafikbefehle im Einsatz

Objekte einlesen

Vielleicht juckt es jetzt schon in Ihren Fingern — wir wollen Sie nicht länger auf eine erste Demonstration warten lassen. In der Schublade »BasicDemos« Ihrer »ExtrasD«-Diskette existiert eine Datei mit dem Namen »Ball«. Sie enthält ein ebensolches Objekt, und dieses wollen wir nun mit einem Programm (Listing 13) laden und abbilden. Mit

```
OPEN "ExtrasD:Basic
Demos/Ball" FOR INPUT
AS 1
OBJECT.SHAPE 1,INPUT$(
LOF(1),1)
```

wird die Datei geöffnet und das Objekt mit der Nummer 1 belegt. Natürlich können Sie die Werte des Objekts auch einer Stringvariablen zuweisen, und dann mit

```
OBJEKT.SHAPE <Nummer>
1>,<String$>
```

das Objekt definieren. Dies hat den Vorteil, daß Sie aus dem <String\$> durch die Anweisungen

```
<Tiefe$> = CVI(MID$(
<String$>,11,2))
<Breite$> = CVI(MID$(
<String$>,15,2))
<Höhe$> = CVI(MID$(
<String$>,19,2))
```

Informationen über die Breite, Höhe und Anzahl der benutzten Bitplanes erhalten können. Vor allen Dingen auf die Anzahl werden wir später etwas näher eingehen, da sie für die Farbdarstellung von Bobs (nicht jedoch von Sprites) eine wesentliche Rolle spielt. Wollen Sie

mehreren Objekten die gleiche Form geben, so können Sie natürlich die »OBJECT.SHAPE«-Anweisung nochmals in derselben Art wie oben benutzen. Es spart jedoch Speicherplatz, wenn Sie stattdessen angeben:

```
OBJECT.SHAPE <Nummer>
2>,<Nummer1>
```

<Nummer1> bezeichnet dabei ein bereits definiertes Objekt bezeichnet.

Bis jetzt haben Sie dem Amiga nur ein neues Animationsobjekt bekannt gemacht — sehen können Sie es auf dem Bildschirm allerdings noch nicht. Bevor wir es jedoch einschalten, sollte erst mal die Position des Objektes festgelegt werden. Mit den Anweisungen

```
OBJECT.X <Nummer> ,
<x-Position>
OBJECT.Y <Nummer> ,
<y-Position>
```

legen Sie den Abstand vom Koordinatenursprung fest. Mit

```
OBJECT.ON <Nummer>
```

wird Ihr Objekt nun sichtbar. Doch jetzt wird es interessant: wir bringen Bewegung ins Spiel. Durch

```
OBJECT.VX <Nummer> ,
<x-Geschwindigkeit>
OBJECT.VY <Nummer> ,
<y-Geschwindigkeit>
```

legen Sie die Geschwindigkeit der Bewegung der Bewegung in Bildpunkten pro Sekunde fest. Eine positive <x-Geschwindigkeit> bewegt das Objekt von links nach rechts; eine negative in umgekehrter Richtung. Ebenso läßt sich mit

```
WINDOW 4,"Animation",(310,95)-(580,170),15
ON COLLISION GOSUB abprallen
COLLISION ON
OPEN "Ball" FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,240
OBJECT.Y 1,50
OBJECT.VX 1,30
OBJECT.VY 1,30
OBJECT.ON
OBJECT.START
WHILE 1 : SLEEP : WEND

abprallen:
merkf = WINDOW(1)
WINDOW 4
i = COLLISION(0)
IF i=0 THEN RETURN
j = COLLISION(1)
IF j=-2 THEN OBJECT.VX i,ABS(OBJECT.VX(i))
IF j=-4 THEN OBJECT.VX i,-ABS(OBJECT.VX(i))
IF j=-1 THEN OBJECT.VY i,ABS(OBJECT.VY(i))
IF j=-3 THEN OBJECT.VY i,-ABS(OBJECT.VY(i))
OBJECT.START
WINDOW merkf
RETURN
```

Listing 13. Ein einfaches Programm zur Sprite-Darstellung

UNLIMITED

Spiele

AARGH	54
ALIEN FIRES	64
AMEGAS	9
ARAZOK's TOMB	49
ARENA & BRATACASS	58
ARKANOID	57
BACKGAMMON	29
BACKLASH	44
BAD CAT	53
BALANCE OF POWER	59
BALL RAIDER	29
BALLYHOO MYSTERY	74
BARBARIAN (Psygnosis)	54
BATTLESHIPS	49
BLASTERBALL	29
BONE CRUNCHER	49
BORROWED TIME	65
BRAINSTORM	29
BUBBLE BOBBLE	47
BUREAUCRACY	68
CHAMPIONSHIP BASEBALL	59
CHAMPIONSHIP BASKETBALL	59
CHAMPIONSHIP FOOTBALL	73
CHAMPIONSHIP GOLF	59
CHESSMASTER 2000	61
CLEVER & SMART	48
COMPUTER BASEBALL	68
COMPUTER HITS	74
COOGANS RUN	44
DARK CASTLE	59
DEEP SPACE	57
DEFENDER OF THE CROWN	63
DEJA VU	85
DETONATOR	39
DIABLO	39
EARL WEAVER BASEBALL	95
ECO	54
ENCHANTER	57
ENFORCER	29
FAERY TALE	67
FEUD	19
FINAL MISSION	47
FIREPOWER	44
FLIP-FLOP	26
GALACTIC INVASION	49
GALAXY FIGHT	44
GARRISON	44
GARRISON II	49
GOLDEN PATH	49
GOLDEN PYRAMID	39
GRAND SLAM TENNIS	72
GRIDIRON	111
GROßMEISTER	54
GUILD OF THIEVES	59
HACKER II	75
HEX	89
HITCHHIKERS GUIDE	57
HOLLYWOOD HUNX	59
INTO THE EAGLES NEST	59
JEWELS OF DARKNESS	59
JIGSAW MANIA	25
JINXTER	69
JUMP JET	39
KAMPFGROUPE	89
KARATE KID II	57
KING OF CHICAGO	61
KNIGHT ORC	65
LAS VEGAS	25
LEADER BOARD GOLF	57
LEADER BOARD TOURNAMENT	29
LEATHER GODESSES	79
LEVIATHAN	49
LITTLE COMPUTER PEOPLE	69
LURKING HORROR	59
LIBYANS IN SPACE	59
MAGICIANS DUNGEONS	79
MINDBREAKER	25
MINDSHADOW	69
MISSION ELEVATOR	39
MOEBIUS	57
MOONMIST	77
OBLITERATOR	57
OGRE	69
PACK BOY	25
PHALANX	25
PHANTASIE II	59
PLUTOS	47
POWERPLAY	52
REISENDE IM WIND	59
RETURN TO ATLANTIS	69
ROADWAR 2000	69
ROADWARS	49
ROCKY	29
SECONDS OUT	69
SHANGHAI	59
SHOOTING STAR	24
SILICON DREAMS	69
SINBAD & THRONES OF THE F.	64
SKYBLASTER	49
SPACE BALLER	19
SPACE BATTLE	19
SPACEPORT	49
SPEED	19
STATIONFALL	64
STRANGE NEW WORLD	45
STRIKE FORCE HARRIER	59
STRIP POKER II	39
TASS TIMES IN TONETOWN	79
TERRAMEX	44
TERRORPODS	55

Wer mehr bezahlt hat selber Schuld!

Schlag auf Schlag fällt die Hochburg um; die Elefanten stehen traurig rum; die versunkene Stadt bleibt unterm Meer: **UNLIMITED muß her!**

Animation

AEGIS Animator	198
AEGIS Videoscape 3D 2.0 dt.	289
AEGIS Videoscape 3D Update	49
AEGIS Videotiler	149
APPRENTICE DISNEY 3D jr.	98
APPRENTICE DISNEY 3D Animator	389
APPRENTICE Libraries: GEO	39
APPRENTICE Libraries: LETTERS	39
DELUXE Productions	349
DELUXE Video 1.2 deutsch	69
Pageflipper deutsch	69
Pageflipper plus F/X	298
Sculpt 3-D	145
Silver	195

Bücher

Amiga Jahrbuch 1988	12
Amiga User's Guide	39
Das grosse Public Domain Buch #1	45
Developers Reference Guide	49

Deutsche Handbücher

Balance of Power	29
Calligrapher	29
Flugsimulator II	29
Jet	29
Kampfgruppe	29

Datenbank

Micro Fiche Filer deutsch	169
Superbase Personal deutsch	197
Superbase Professional deutsch	549

Disketten

3 1/2 Zoll 2DD Maxell 10er	39
3 1/2 Zoll 2DD No Name	23
Diskettenreinigungssatz 3 1/2	15
Diskettenreinigungssatz 5 1/4	12
Diskettentasche Stoff 3 1/2	19

Diverses

AEGIS Diga!	109
-------------	-----

Drucker

Citizen 120D	498
Druckerkabel A-500/2000 Centr.	15
Hewlett Packard Paint Jet	3398
MPS 1500C Farbdrucker	698
NEC P-2200 24 Nadeln	895
STAR Laserprinter 8	5495
STAR LC 24/10	949
STAR LC-10	639
STAR LC 10 Color	748
STAR NB 24/10	1498
STAR NB 24/15	1998

Erweiterung

HURRICANE 68030 Optionboard	795
HURRICANE Speicherboard 2M/32	1495
HURRICANE Turboboards A2000 o.P.	1495
SCSI & RAM 2 MByte OK	685
TV-HF Modulator A-500/2000	57
XT Erweiterung für A-2000	995

Noch mehr Spiele

TESTDRIVE	59
TETRIS	48
THAI BOXING	29
THE ART OF CHESS	59
THE PAWN	59
THREE STOOGES	64
TIME BANDIT	53
TOLTEKA	59
TRINITY	69
ULTIMA III	69
UNINVITED	67
VADER	25
VAMPIRES EMPIRE	49
VIDEO VEGAS	59
VYPER	47
WINTER GAMES	59
WINTER OLYMPICS 88	44
WIZARDS CROWN	99
WORLD GAMES	59
XR-35	22
ZOOM!	44
ZORK TRILOGY	115

Diese Liste ist bedingt durch Anzeigenvorlauf nicht immer aktuell. Gerne nennen wir Ihnen telefonisch die letzten Neuheiten auf dem Spielmarkt.

Farbbänder

Citizen 120D	14
MPS 1500 Color	29
MPS 1500 sw.	25
NEC P2200 sw.	25
NEC P6 sw.	12
NEC P6 Color	49
NEC P7 Color	69
NEC P7 sw.	19
STAR LC 24/10	19
STAR LC-10	19
STAR LC 10 Color	29
STAR NB 24/10	19
STAR NB 24/15	25
STAR NL-10 sw.	12
Farbpatrone DIABLO 150 4 gem.	24
Farbpatrone DIABLO 150 einz.	6

Festplatten

20 MB SCSI f. A-2000 kompl.	995
SCSI Hardcard 20 MB A-2000	1395
SCSI Hardcard 40 MB A-2000	1895
SCSI Hardcard 80 MB A-2000	2895

Grafik

AEGIS Draw	169
AEGIS Draw plus	298
AEGIS Images	57
AEGIS Impact	111
Art Gallery #1	49
Art Gallery #2	49
Butcher 2.0 PAL deutsch	85
Calligrapher deutsch	169
DELUXE Paint II	185
DELUXE Photolab	199
DELUXE Print & Data #1	149
Digi Paint PAL deutsch	95
EASYL 1000 Zeichentablett	598
EASYL 2000 Zeichentablett	698
EASYL 500 Zeichentablett	589
Fonts & Borders	89
Funktion Graphenzeichner	89
Intro Cad	108
Photon Paint Expansion Disk	65
Photon Paint	149
Pixmate	94
Printmaster plus	69
Prism plus	99
TV-Text 3D	149

Kabel

AMIGA Scart Kabel 2 mtr.	29
--------------------------	----

Kalkulation

Analyze 2.0	222
Logistik	277
Maxiplan 500	335
Maxiplan plus	293

Laufwerke

Laufwerk 3 1/2 extern	277
Laufwerk 3 1/2 intern	199
Laufwerk 5 1/4 extern	333

Lernen

Länder dieser Erde	39
--------------------	----

Monitore

Farbmonitor 1084	648
Monitorständer 14 Zoll/Metall	39

Musik

AEGIS Audiomaster	77
DELUXE Music Construction dtsh.	177
Drum Studio	49
Future Sound II	333
Instant Music	85
Midi Interface A-1000	87
Midi Interface A-500/2000	87
Music Student	128
Music Studio 2.0	77
Perfect Sound mit Digitizer	139
Pro Midi Studio	266
AEGIS Sonix	111
Sound Sampler A-1000	111
Sound Sampler A-500/2000	111
Studio Magic	198

Simulation

Flightsimulator II	79
Galileo Planetarium	88
Interceptor	79
Jet	79
Scenery Disk #11	49
Scenery Disk #7	49
Scenery Disk Europe	41

Sprachen

AC Basic Compiler	289
Aztec C Developers	439
Aztec C Professional	319
Lattice C Compiler Companion	198
Lattice C Compiler 4.0	349
Lisp Metacomco	344
Macro Assembler Metacomco	144
Modula 2 Commercial	388
Modula 2 Developer	284
Modula 2 Regular	166
Pascal 2.0 (ISO) Metacomco	244
Pascal UCSD	199

Tools

CLI Mate	80
Demonstrator deutsch	79
DISCOVERY Disk Editor deutsch	188
Disk to Disk	88
DiskMaster dtsh.	98
Dos to Dos	98
Fast Lightning	79
Floppy Accelerator II	48
Grabbit	49
Marauder	55
Mirror	98
Mirror Hacker Package	98
Project D	79
Quarterback 1.4 deutsch	111
Shell Metacomco	94
Toolkit Metacomco	82
TxEd plus Editor	72
Zing Keys	72

Video

Digi-View 3.0 Digitizer	298
PAL Video Karte f. A-2000	111

Zubehör

Abdeckhaube AMIGA 500 Acryl	19
Basic Keyboard Overlay A-1000	19
DOS Keyboard Overlay A-1000	19
Jitter Rid Filterscheibe	29
Joystick Competition Pro	25
Joystick Competition Pro E	40
Joystick Competition Pro T	35
Konzepthalter schwenkbar	14
Mouse House Max grau	15
Mouse House Millie rosa	15
Mouse House Stoffmaus braun	15
Mouse Pad 27 x 23 cm	14
Trackball	79

BESTELLSERVICE

Rund um die Uhr **06121/543848**

Wir liefern nur Originalprogramme zu knallhart kalkulierten Preisen. Bestellen Sie schriftlich oder unter obiger Telefonnummer. Lieferung solange Vorrat reicht gegen Vorkasse (+4,- DM Porto) oder Nachnahme (+6,- DM Porto). Mindestbestellwert 50,- DM. Preisänderungen und Irrtümer vorbehalten.

UNLIMITED

M. Hottenbacher, Kehrstraße 23, 6200 Wiesbaden

negativen Werten für die <y-Geschwindigkeit> die Bewegungsrichtung bestimmen. Selbst nach der Eingabe dieser Befehle verharrt unser Objekt aber immer noch regungslos auf dem Bildschirm. Erst mit

```
OBJECT.START <Nummer>
```

beginnt ein Bewegungsablauf, der durch

```
OBJECT.STOP <Nummer>
```

wieder angehalten werden kann. Damit Sie während der Bewegung nicht den Überblick verlieren, läßt sich die augenblickliche Position mit den Funktionen

```
OBJECT.X (<Nummer>)
```

```
OBJECT.Y (<Nummer>)
```

ermitteln. Nicht immer bleibt die Bewegung eines Objektes konstant: Durch Brems- und Beschleunigungsmanöver wird die Programmierung erst richtig interessant. Die Befehle

```
OBJECT.AX <Nummer> ,
```

```
<x-Beschleunigung>
```

```
OBJECT.AY <Nummer> ,
```

```
<y-Beschleunigung>
```

erlauben es, die Objekte zu beschleunigen. Dabei wird bei positiven Werten nach rechts beziehungsweise nach unten beschleunigt, bei negativen Werten jeweils umgekehrt. Um ein Objekt abzubremsen, müssen Sie es also einfach entgegen seiner momentanen Bewegung beschleunigen. Selbstverständlich gibt es auch die zur Geschwindigkeit analogen Befehle, die augenblickliche Beschleunigung abzufragen.

Geschwindigkeit ist alles

Ihr Objekt läuft bis jetzt immer der Nase nach und stößt so irgendwann an den Rand des Fensters. Wenn Sie ihm nicht beibringen, was es in einem solchen Fall tun soll, bleibt es prompt wie angewurzelt stehen. Für solche Zusammenstöße hat Amiga-Basic die Anweisung

```
ON COLLISION GOSUB <Marke>
```

Dabei ist <Marke> entweder eine Zeilennummer oder eine alphanumerische Sprungmarke. Sie geben also das (unechte) Unterprogramm an, das nach einer Kollision abgearbeitet werden soll. Voraussetzung dafür ist aber, daß Sie diesen Mechanismus mit

```
COLLISION ON
```

aktiviert haben. Diese beiden Zeilen sollten in Ihrem Programm am besten zwischen den Variablendeklarationen und dem Beginn des Hauptprogramms stehen. In diesem »Kollisions-Unterprogramm« können Sie dann entsprechend auf den Zusammenstoß reagieren. In unserem Fall kann das Objekt ja nur mit dem Rand des Fensters kollidieren. Das muß aber nicht immer so sein. Mit der Funktion

```
COLLISION(0)
```

erhalten Sie die Nummer des Objekts, welches zuletzt einen Zusammenstoß hatte. Rufen Sie diese Funktion erneut, mit der soeben ermittelten Nummer als Argument auf, so erhalten Sie die Nummer des Kollisionspartners. Ist diese Angabe negativ, so handelt es sich jeweils um den oberen (-1), linken (-2), unteren (-3) oder rechten (-4) Rand des Bildschirmfensters. Sie können sich allerdings mit Hilfe des Befehles

```
OBJECT.CLIP (<linke  
obere Ecke>) - (<  
rechte untere Ecke>)
```

selbst das Rechteck definieren, dessen Ränder eine Kollision auslösen sollen. Unser Programmbeispiel verwendet diese Technik, um den Ball vom Rand des Fensters abprallen zu lassen. Bei diesem Programm handelt es sich um eine verbesserte Version des in Ihrem Basic-Handbuch beim »OBJECT.SHAPE«-Befehl angegebenen, fehlerhaften Programms. Den Fehler dieses Programms entdecken Sie sehr leicht, indem Sie es einfach abtippen und einige Zeit laufen lassen: der Ball verläßt das Fenster! Die wesentliche Änderung in unserem Programm betrifft das Unterprogramm »Abprallen«. Anstatt lediglich zu überprüfen, ob das Objekt mit den senkrechten oder waagrechten Fensterbegrenzungen kollidiert und dann die y-beziehungsweise x-Geschwindigkeit einfach umzudrehen, werden in unserer Version alle Seiten einzeln abgefragt und das Objekt unter Verwendung der »ABS«-Funktion wieder auf den rechten Weg geschickt. Wo liegen nun die Ursachen für das absonderliche Verhalten der Urversion? Da wir in den Basic-Interpreter nicht »hineinschauen« können, ist die sichere Bestimmung des Fehlers nur schwer möglich. Trotzdem können Überlegungen dazu bei Ihren Programmerversuchen zur Animation von Nutzen sein:

Nach dem Starten des Bewegungsablaufes bewegen sich die Animationsobjekte unabhängig von Ihrem Basic-Programm. Erst nach einer Kollision wird das Programm aus seinem »Schlaf« geweckt und es verzweigt in ein eigens für diesen Zweck geschriebenes Unterprogramm, welches sich um solche Zusammenstöße kümmert. Um dorthin zu gelangen, braucht der Amiga natürlich etwas Zeit und währenddessen bewegt sich das Objekt weiter.

Um die Ecke gebracht

Stößt das Objekt nun genau in die Ecke des Fensters, so merkt sich der Amiga die Kollision mit den beiden Fensterbegrenzen (er kann sich bis zu 16 Zusammenstöße merken). Nehmen wir einmal an, das Objekt stößt an die linke obere Ecke und die Urversion dreht nun zuerst die waagrechte Bewegungsrichtung um. Wieder zurück im Hauptprogramm merkt der Amiga, daß noch eine zweite Kollisionsmeldung vorliegt und dreht nun auch die senkrechte Bewegung um. Beide Unterprogrammaufrufe haben jedoch »viel« Zeit gekostet und so hat sich das Objekt inzwischen weiter nach oben bewegt und eine dritte Kollisionsmeldung verursacht. Auf diese wird auch prompt reagiert und der Effekt ist ein er-

neutes Umdrehen der Geschwindigkeitskomponente — das Objekt bewegt sich also weiter nach oben und damit aus dem Fenster hinaus! Die Ursache des Problems ist also die mangelnde Geschwindigkeit des Interpreters. Wenn sich dies allerdings bereits bei einem Objekt auswirkt, wie soll das erst mit mehreren werden? In einem solchen Fall sind natürlich eine ganze Reihe von Kollisionen zu erwarten und alle sollten auch verarbeitet werden. Das Objekt, dessen Kollision bearbeitet wird, hält zwar an, die anderen laufen jedoch weiter und können dabei auch mit anderen Objekten zusammenstoßen. Dies kann dazu führen, daß die maximale Anzahl von 16 Kollisionen schnell überschritten wird. Alle Objekte, deren Zusammenstöße mit dem Rand sich der Amiga nicht mehr merken kann, verschwinden vom Bildschirm. Einen einfachen Trick zur Lösung dieses Problems zeigt unser Programm (Listing 14). Im Unterprogramm »Abprallen« werden außer dem Objekt, das eine Kollision erfahren hat, auch die Zusammenstöße der anderen registriert und bearbeitet. Durch das Streichen der beiden Zeilen mit dem »FOR«- und dem »NEXT«-Befehl und dem Einfügen von »BEEP« direkt unter der Sprungmarke »Abprallen:« läßt sich beobachten, was passiert, wenn Sie nicht mit dieser Technik arbeiten: In-

```
anzahl = 6
SCREEN 2,640,255,2,2
WINDOW 4,"Animation",,15,2
ON COLLISION GOSUB Abprallen
COLLISION ON
RANDOMIZE TIMER
OPEN "ball" FOR INPUT AS 1
form$ = INPUT$(LOF(1),1)
CLOSE 1
FOR n = 1 TO anzahl
  OBJECT.SHAPE n,form$
  OBJECT.X n,300 * RND + 100
  OBJECT.Y n,80 * RND + 50
  OBJECT.VX n,200 * RND - 100
  OBJECT.VY n,200 * RND - 100
NEXT n
OBJECT.ON
OBJECT.START
WHILE 1 : SLEEP : WEND
Abprallen:
  i = COLLISION(0)
  IF i = 0 THEN RETURN
  BEEP : FOR i = 1 TO anzahl
    j = COLLISION(i)
    IF j=-2 THEN OBJECT.VX i,ABS(OBJECT.VX(i))
    IF j=-4 THEN OBJECT.VX i,-ABS(OBJECT.VX(i))
    IF j=-1 THEN OBJECT.VY i,ABS(OBJECT.VY(i))
    IF j=-3 THEN OBJECT.VY i,-ABS(OBJECT.VY(i))
    IF j> 0 THEN BEEP
  NEXT i
  OBJECT.START
RETURN
```

Listing 14. Kollisionsabfrage für Sprites

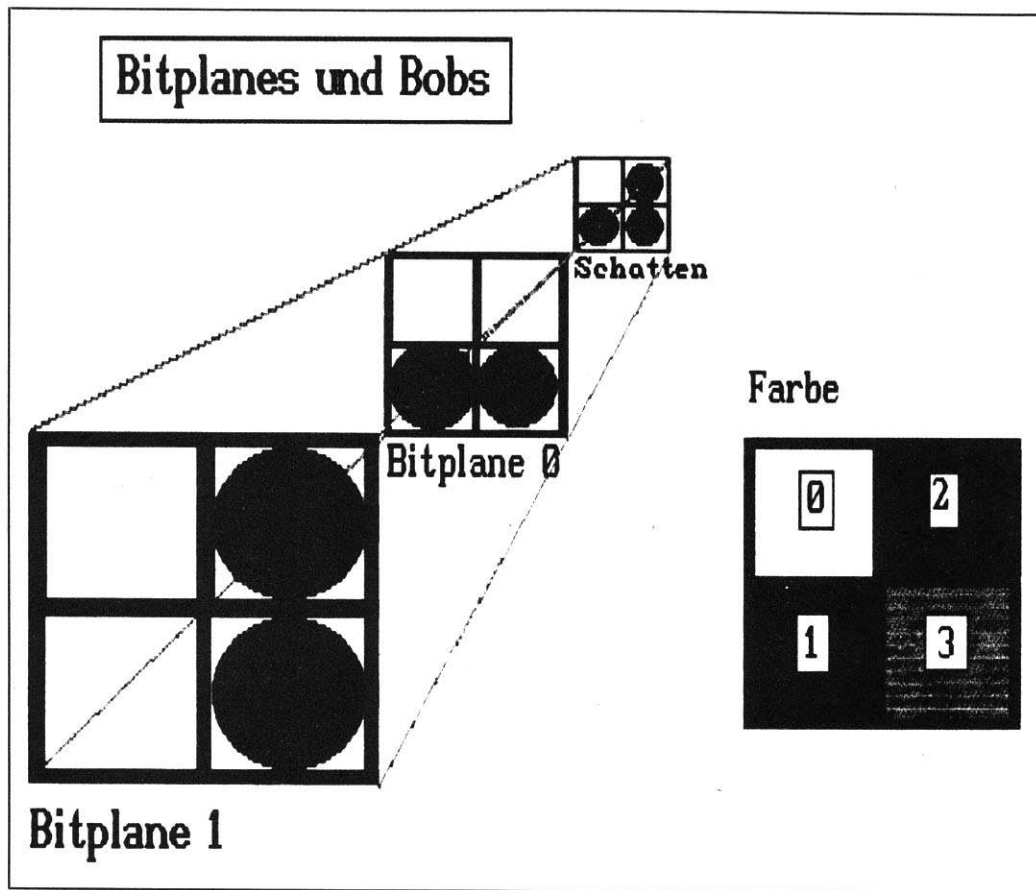


Bild 15. Das Zeichnen von Bobs in Bitplanes

nerhalb kurzer Zeit wird aus dem kurzen Pfeifton ein Dauerton — Amiga-Basic kommt mit der Verarbeitung der Kollisionen nicht mehr nach.

Die bis jetzt besprochenen Animationsbefehle erlauben Ihnen den einfachen Umgang mit Objekten, unabhängig davon, ob es sich um Bobs oder Sprites handelt. Die nun folgenden, höheren Animationsbefehle finden zum Großteil nur auf Bobs Anwendung und erfordern zudem ein etwas detaillierteres Wissen um die Art, wie Ihr Amiga Farben auf dem Bildschirm darstellt. Kenntnisse des binären und sedezimalen Zahlensystems sind zwar vorteilhaft, jedoch kommen Sie zur Not auch ohne diese aus. Wenn Sie jedoch tiefer in die Geheimnisse Ihres Amiga eindringen wollen, kann Ihnen nur empfohlen werden sich, durch entsprechende Übung, mit diesen Zahlensystemen vertraut zu machen.

Wie Sie noch vom »PALETTE«-Befehl her wissen, können Sie bis zu 32 Farben (0 bis 31) gleichzeitig darstellen. Wenn Sie nochmals einen Blick auf die Wertigkeitstabelle des »PATTERN«-Befehles werfen, werden Sie feststellen, daß die Zahlen 0 bis 31 entsprechend den Wertigkeiten durch fünf (übereinanderliegende) Punk-

te dargestellt werden können. Die vier Farben der Workbench können durch die vier Punktkombinationen (0,0);(0,1);(1,0);(1,1) (siehe Bild 15) dargestellt werden. Um diese vier Farben auf dem ganzen Bildschirm darzustellen, werden also zwei übereinanderliegende Ebenen (Bitplanes) benötigt. Zur Darstellung der Bildschirmfarben sieht Ihr Amiga also im Speicher in den jeweiligen Bitplanes nach, welche übereinanderliegenden Punkte gesetzt sind und wählt dann, nach den Punktkombinationen zugeordneten Wertigkeiten, die entsprechende, mit »PALETTE« eingestellte Farbe.

Bobs und Bitplanes

Ein Bob wird nun definiert, indem für jede Bitplane angegeben wird, welche Punkte gesetzt werden und welche nicht. Dies muß allerdings nicht für die ganze Bitplane geschehen, sondern nur für einen durch die Breite und Höhe des Bobs gegebenen Bereich. Um die folgenden Erörterungen nicht zu umfangreich werden zu lassen, beschränken wir uns auf die Workbencheinstellung von zwei Bitplanes. Die DATA-Zeilen in Listing 15 definieren einen Bob, der aus vier

Rechtecken in den vier Workbenchfarben besteht (Bild 15). Im Normalfall wird dieser Bob auch so auf dem Bildschirm erscheinen, doch was passiert, wenn Sie einen eigenen Screen, etwa mit einer anderen Anzahl von Bitplanes, definieren? Dazu ist in Amiga-Basic der Befehl »OBJECT.PLANES« vorgesehen, der — neben der obligatorischen Objektnummer — zwei Argumente besitzt. Besprechen wir zunächst das erste Argument. Dieses Argument ist eine »Maske« (eine 1 Byte lange Ganzzahl), die es gestattet

zu spezifizieren, in welche Bitplanes des Bildschirms die einzelnen Bitplanes des Bobs eingetragen werden sollen. Dazu tippen Sie am besten Listing 15 ab und starten das Programm. Die für Sie jetzt interessanten Bilder befinden sich in der ersten Spalte: Das unterste Bild zeigt den normalen Bob, denn das erste Argument von »OBJECT.PLANES« war hier 3, was auf beiden Bitplanes gesetzten Punkten entspricht. Das oberste Bild zeigt gar nichts, denn

Programmierer zählen »anders«

gar nichts, denn hier war das Argument 0, womit der Bob auch auf keiner Bitplane gezeichnet wird. In den Bildern in den beiden mittleren Zeilen sieht der Bob dagegen merkwürdig verändert aus. Hier wurden nur die Anteile des Bobs gezeichnet, die in der Bitplane 0 (Argument = 1) beziehungsweise in der Bitplane 1 (Argument = 2) liegen. Lassen Sie sich bitte nicht von der bei Programmierern üblichen Zählweise verwirren: Es liegen zwei Bitplanes vor, die mit 0 und 1 numeriert sind. Ebenso besteht ein Byte aus 8 Bits, die von 0 bis 7 gezählt werden. Die restlichen drei Spalten zeigen die Auswirkungen des zweiten Arguments des »OBJECT.PLANES«-Befehles. Zu diesen Erläuterungen müssen wir noch etwas weiter ausholen. Jedem Bob ist außer seinen Bitplanes auch noch ein »Schatten« zugeordnet (siehe Bild 15).

Stellen Sie sich die Bitplanes als durchsichtige Glasscheibe vor. An den Stellen der Bitplanes, an denen Punkte gesetzt sind, sind diese Scheiben geschwärzt und somit undurchsichtig. Lassen Sie jetzt

```
main:
  Erzeuge Bob$
  Beschriftung
  Zeichne Bob$
END
SUB Erzeuge(BobShape$) STATIC
  READ Farbe& : BobShape$ = MKL$(Farbe&) : REM nicht benutzt
  READ Datensatz& : BobShape$ = BobShape$ + MKL$(Datensatz&) : REM
    nicht benutzt
  READ Tiefe& : BobShape$ = BobShape$ + MKL$(Tiefe&)
  READ Breite& : BobShape$ = BobShape$ + MKL$(Breite&)
  READ Hoehe& : BobShape$ = BobShape$ + MKL$(Hoehe&)
  READ Flags% : BobShape$ = BobShape$ + MKI$(Flags%)
  READ Auswahl% : BobShape$ = BobShape$ + MKI$(Auswahl%) : REM
    nicht benutzt
  READ AnAus% : BobShape$ = BobShape$ + MKI$(AnAus%) : REM
    nicht benutzt
  Grenze% = Hoehe& * Tiefe& * ((Breite& + 15)\16)
  FOR i = 1 TO Grenze% : REM Schleifengrenze muss kurze Ganzzahl sein
```

Listing 15. Dieses Programm zeigt, wie Bobs dargestellt werden

penden Bobs Auswirkungen darauf, welcher Bob von welchen anderen Bobs überdeckt werden kann. Die Anweisungssyntax lautet

OBJECT.PRIORITY <Bob nummer>, <Priorität>

<Priorität> ist dabei ein kurze Ganzzahl.

Der Amiga wird geschwätzig

Die Möglichkeiten des Amiga zur Tonerzeugung wurden von den Basic-Entwicklern leider etwas stiefmütterlich behandelt. Den einfachsten Befehl zur Erzeugung von Geräuschen kennen Sie schon: »BEEP« dient nur dazu, die Aufmerksamkeit eines Benutzers auf den Amiga zu lenken. Anwendung findet dieser Befehl zur akustischen Unterstützung der Fehlermeldungen von Amiga-Basic.

Interessanter ist da schon die Sprachausgabe des Amiga. Diese ist durchaus komfortabel und einfach zu bedienen, vorausgesetzt, es handelt sich um englische Texte. Hier wurde leider auf die Bedürfnisse der deutschen Amiga-Anwender keine Rücksicht genommen.

Der Befehl »TRANSLATE\$« übersetzt einen beliebigen Text in einen dem Amiga verständlichen Phonemcode (eine Art Lautschrift). Da aber von einem englischsprachigen Text ausgegangen wird, erhalten deutsche Texte einen geradezu zungenbrecherischen Phonemcode. Entsprechend hört sich dann auch ein mit Hilfe des »SAY«-Befehles wieder-gegebener deutscher Text an. Deutsch mit starkem englischen Akzent ist ein milder Ausdruck dafür. Im Anhang H Ihres Amiga-Basic-Handbuches finden Sie zwar unter anderem auch einige Vorschläge zur Behandlung deutscher Texte, aber die Ergebnisse klingen nicht sehr überzeugend. Wer Wert auf eine saubere deutsche Aussprache legt, wird sich wohl noch gedulden müssen. Oder er übersetzt die deutschen Texte per Hand mittels der Phoneme-Tabelle in den Phoneme-Code.

Der »SAY«-Befehl erwartet also als erstes Argument den Phonemcode des auszusprechenden Textes. Als zweites Argument kann der Name eines Ganzzahlfeldes, das die Eigenschaften der Stimme festlegt, übergeben werden.

Dieses besitzt neun Einträge, daher dimensionieren (»DIM«) Sie es am besten als »<Stimme%>(8)«. Die einzelnen Feldeinträge werden in folgender Tabelle beschrieben:

Element	Funktion	Zulässige Werte
0	Grundfrequenz der Stimme	65 bis 320
1	Silbenbetonung	0,1
2	Sprechgeschwindigkeit	40 bis 400
3	Geschlecht der Stimme	0,1
4	Stimmqualität	5000 bis 28000
5	Lautstärke	0 bis 65
6	Ausgabekanäle	0 bis 9
7	Synchronisation	0 bis 2
8	Asynchron-Modus	0 bis 2

Element 0 beschreibt die Grundfrequenz der dargestellten Stimme. Hier sind Werte zwischen 65 und 320 Hertz (Schwingungen pro Sekunde) möglich. Die Voreinstellung ist eine normale Stimme mit 110 Hertz. Element 1 beschreibt die Silbenbetonung: Voreingestellt ist 0, die der Betonung von Silben entspricht, während 1 eine monotone, roboterhafte Sprache erzeugt. Das Element 2 bestimmt die Sprechgeschwindigkeit in Worten pro Minute. Voreingestellt ist ein Wert von 150, Sie können jedoch Werte von 40 bis 400 angeben. Element 3 bestimmt, ob der Sprecher männlich (0) oder weiblich (1) sein soll.

Die Stimmqualität wird durch das vierte Element beeinflusst. Es können Werte zwischen 5000 und 28000 eingestellt werden. Dabei ist jedoch zu empfehlen, den voreingestellten Wert von 22000 nicht zu verändern. Element 5 steuert die Lautstärke zwischen 0 und dem voreingestellten Maximum von 65. Die Ton-Kanäle zur Sprachausgabe können durch das Element 6 ausgewählt werden. Wie Sie vielleicht wissen, besitzt der Amiga vier Ton-Kanäle, die unabhängig voneinander angesteuert werden können. Dabei sind die Kanäle 0 und 3 der linken, die Kanäle 1 und 2 der rechten Seite zugeordnet. 0 bis 3 wählt den entsprechenden Ton-Kanal. 4 wählt die Kanäle 0 und 1, 5 wählt die Kanäle 0 und 2, 6 wählt die Kanäle 3 und 1 und 7 wählt die Kanäle 3 und 2. Die Einstellung 8 wählt hingegen jeden verfügbaren linken und 9 wählt jeden verfügbaren rechten Kanal. Voreingestellt ist der Wert 10, der jedes verfügbare Paar aus linkem und rechtem Kanal wählt. Schließlich wählt ein Wert von 11 irgendeinen verfügbaren Kanal aus.

Mit Feldelement Nummer 7 kann die Synchronisation der Sprachausgabe mit dem Basic-Programm erfolgen. Voreingestellt ist hier der Wert 0, der Basic veranlaßt, so lange die Bearbeitung weiterer Anweisungen einzustellen, bis der »SAY«-Befehl beendet ist. Ein Wert von 1 dagegen führt die Sprachausgabe im Hintergrund aus, während Basic mit der Befehlsbearbeitung fortfährt. Das letzte Feldelement ist nur dann von Bedeutung, wenn die asynchrone Sprachausgabe aktiviert ist. In diesem Fall bewirkt 0 (voreingestellt) eine normale Verarbeitung, was heißt, daß eine weitere »SAY«-Anweisung erst durchgeführt wird, wenn die vorausgehende beendet ist. Bei einem Wert von 1 wird die Sprachausgabe abgebrochen, während bei einem Wert von 2

die aktuelle »SAY«-Anweisung abgebrochen und die neue »SAY«-Anweisung bearbeitet wird.

The sound of music...

Bevor wir uns nun den beiden einzigen Befehlen des Amiga-Basic, die zur Tonerzeugung zur Verfügung stehen, zuwenden, erst einmal zu theoretischen Grundlagen der Tonerzeugung. Ein Ton entsteht durch Schwingungen der uns umgebenden Luft, wobei unser Ohr Schwingungen zwischen etwa 15 und 20000 Hz wahrnehmen kann. Der wichtigste Ton in den musikalischen Notensystemen ist der Kammerton »a«, der eine Frequenz von 440 Hertz besitzt. Die C-Dur Grundtonleiter enthält neben diesem Ton noch sechs weitere und ist wie folgt aufgebaut:

Note	Frequenz
c	261.630
d	293.660
e	329.630
f	349.230
g	392.000
a	440.000
h	493.880

Diese sieben Töne umfassen eine Oktave, der nächste Ton in diesem System ist wieder ein »c«. Dessen Frequenz ist allerdings mit 523,260 Hertz genau doppelt so groß wie die des ersten. Diese Regel hat Allgemeingültigkeit: Mit jeder Oktave verdoppeln sich die Frequenzen der zugrundeliegenden Töne, beziehungsweise bei tieferen Oktaven halbieren sich die Frequenzen. Nehmen wir als Beispiel den Ton »a«: Der Kammerton hat eine Frequenz von 440 Hertz, das »a« in der nächst höheren Ok-

Dur-Tonleiter	Frequenzverhältnis	Beispiel
Prim (Grundton)	1 : 1	c
Sekunde	9 : 8	d
Große Terz	5 : 4	e
Quarte	4 : 3	f
Quinte	3 : 2	g
Große Sexte	5 : 3	a
Große Septime	15 : 8	b
Oktave	2 : 1	c
Moll-Tonleiter	Frequenzverhältnis	Beispiel
Prim (Grundton)	1 : 1	a
Sekunde	9 : 8	b
Kleine Terz	6 : 5	c
Quarte	4 : 3	d
Quinte	3 : 2	e
Kleine Sexte	8 : 5	f
Kleine Septime	9 : 5	g
Oktave	2 : 1	a

Tabelle 3. Dur- und Moll-Tonleitern

tave hat eine Frequenz von 880 Hertz, noch eine Oktave höher beträgt die Frequenz des »a« 1760 Hertz. Das »a« eine Oktave unter dem Kammerton hat dagegen nur eine Frequenz von 220 Hertz.

Ausgehend vom Grundton lassen sich die Tonleitern auch durch Frequenzverhältnisse charakterisieren, wobei zwischen Dur- und Moll-Tonarten unterschieden werden muß (Tabelle 3).

Doch nun wollen wir dem Amiga erste Töne entlocken. Mit der Anweisung

```
SOUND <Frequenz>,
<Dauer>, <Lautstärke>,
<Kanal>
```

können Sie Ihre Lautsprecher zum Leben erwecken. Dabei können Sie bei Ihren ersten Versuchen die beiden letzten Argumente sogar weglassen, da Amiga-Basic voreingestellt die Töne mit mittlerer Lautstärke auf Kanal 0 wiedergibt. Die Frequenz als erstes Argument sollte nach den vorstehenden Erläuterungen keine Schwierigkeiten mehr machen. Das zweite Argument gibt an, wie lange der Ton gespielt werden soll. Dabei bezieht sich Ihr Amiga aber leider nicht auf Sekunden, sondern auf Zeittakte. Sie können dabei eine Dauer zwischen 0 und 77 Zeittakten angeben, wobei 18,2 Zeittakte etwa einer Sekunde entsprechen. Wenn Sie wollen, können Sie für den Kanal eine Nummer zwischen 0 und 3 angeben, während die Lautstärke ein Wert zwischen 0 und 255 sein muß.

Wie Sie wissen, verfügt der Amiga über vier Soundkanäle. Soll nun ein mehrstimmiges Stück abgespielt werden und Sie geben den einzelnen Kanälen nacheinander die »SOUND«-Anweisungen, so

hat dies ein heilloses Durcheinander zur Folge. Zum Wohl Ihrer Ohren empfehlen wir Ihnen daher, den Befehl »SOUND WAIT« einzusetzen. Dieser sammelt alle nachfolgenden »SOUND«-Anweisungen auf und erst der Befehl

SOUND RESUME

gibt sie zur Ausgabe frei (Listing 16). Zu einem Kanal gehörende Töne werden nacheinander ausgegeben. Zu unterschiedlichen Kanälen gehörige Folgen beginnen gleichzeitig. Dabei kann sich allerdings ein Problem ergeben: Wollen Sie eine bestimmte Tonfolge zunächst über Kanal 0 wiedergeben und nach einiger Zeit etwa Kanal 1 zuschalten, so muß vom Wirken des »SOUND RESUME« bis zum Einsatz von Kanal 1 über diesen eine beliebige Tonfolge mit der Lautstärke 0 ausgegeben werden. Das Abspielen von Tönen mit der Lautstärke 0 ist ein guter Trick, um die Tonausgabe auf verschiedenen Kanälen zu synchronisieren.

Die Welle bestimmt den Klang

Der zweite Befehl zur Beeinflussung von Tönen ist die »WAVE«-Anweisung. Mit

```
WAVE <Kanal>, <Wellenform%>
```

kann die Wellenform der erzeugten Töne beeinflusst werden. Was haben Töne nun mit Wellen zu tun? Zur Klärung dieser Frage unternehmen wir einen weiteren kleinen Ausflug in die Theorie. Den Tönen, die Sie mit den bisherigen Befehlen erzeugen konnten, lag eine einfache Sinuswelle (Bild 16a) zugrunde. So treten beim Kam-

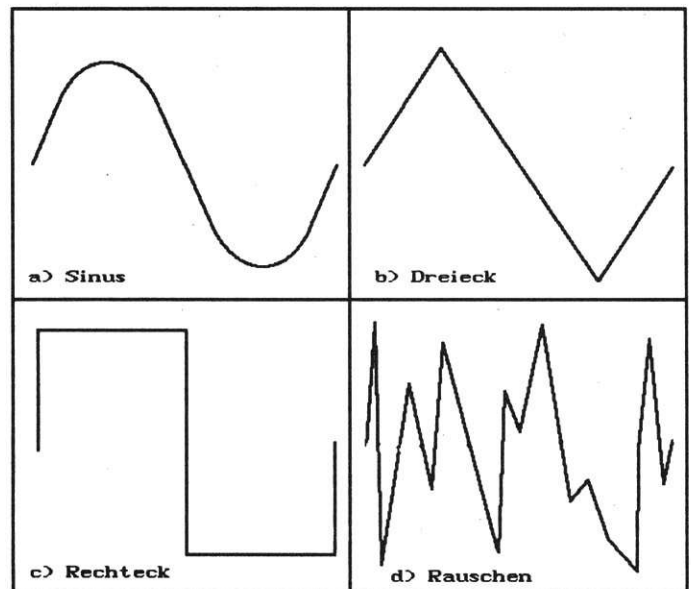


Bild 16. Die vier Standard-Wellenformen zur Klang-erzeugung

merton »a« 440 Schwingungen pro Sekunde auf — und jede einzelne sieht so aus, wie die in Bild 16a gezeigte. Vielleicht haben Sie beim Ausprobieren der Befehle bemerkt, daß die erzeugten Töne nicht unbedingt natürlich, sondern eher künstlich klingen. Der Grund dafür ist, daß Sinusschwingungen in der Natur normalerweise nicht in der reinen Form vorkommen. Um nun die Klangcharakteristik eines bestimmten Musikinstrumentes simulieren zu können, benötigt man andere Wellenformen. Genau diese können Sie mit dem WAVE-Befehl dem Amiga bekanntgeben. Das Argument <Wellenform%> ist dabei ein mindestens 256 Einträge großes Ganzzahlfeld, dessen Elemente Werte von -128 bis 127 sein dürfen. Auf diese Weise können Sie beliebige Schwingungsformen definieren.

Neben der Sinusschwingung gibt es noch andere, bei der Tonerzeugung oft verwendete Wellenformen: die Dreiecks- (Bild 16b) und Rechtecksschwingungen (Bild 16c), sowie das Rauschen (Bild 16d). Aus unserem Programm (Listing 17) können Sie erkennen, wie diese Formen programmiert werden. Für die Umstellung eines Kanals zurück auf die Standardeinstellung, die Sinuswelle, genügt folgende Anweisung:

```
WAVE <Kanal>, SIN
```

Kenner des C 64 oder anderer soundfähiger Computer werden jetzt sicherlich noch etwas vermissen. Die Klangcharakteristik eines Instrumentes wird nämlich nicht nur durch

die Wellenform, sondern auch durch die »Hüllkurve« bestimmt. Sie beschreibt den Anschlag, also die Art und Weise, wie schnell ein Ton die maximale Lautstärke erreicht, wie lange er dann auf dem Maximalpegel verbleibt und schließlich das Ausklingen des Tones. In Amiga-Basic gibt es leider keinen Befehl, mit dessen Hilfe derartige Hüllkurven erstellt werden können. Also müssen Sie solche Toneigenschaften selbst programmieren. Listing 18 demonstriert eine Möglichkeit, dies zu tun. Als kleines Bonbon bildet es die Hüllkurve auch noch auf dem Bildschirm ab.

Achtung: Unterbrechung

Als Anwender von Amiga-Basic-Programmen oder Programmierer dieser Sprache kennen Sie sicher den Requester mit einer Fehlermeldung und der kleinen »OK«-Box am oberen Bildschirmrand: Ein Fehler ist aufgetreten, der Programmablauf wird unterbrochen. Eventuell vorher eingegebene Daten sind verloren. Ärgerlich? In den meisten Fällen ja. Amiga-Basic kennt daher einen Mechanismus, mit dem nach Auftreten eines Fehlers die Programmausführung in einen bestimmten Teil des Programms verzweigt. Hier könnten dann zum Beispiel vor dem Programmieren wichtige Daten auf die Diskette gespeichert werden. Vielleicht läßt sich aber auch die Ursache des Fehlers durch Änderungen des Programms beheben. Das

```
DIM Freq(3),Dauer(3),Lautst(3)
SOUND WAIT
Leseschleife:
FOR i = 0 TO 3
  READ Freq(i),Dauer(i),Lautst(i)
  IF Freq(0)=-1 THEN SOUND RESUME : END
  SOUND Freq(i),Dauer(i),Lautst(i),i
NEXT i
GOTO Leseschleife
REM Frequenz Dauer Lautstaerke
DATA 440.0 , 36.4 , 255
DATA 440.0 , 36.4 , 0
DATA 440.0 , 36.4 , 0
DATA 440.0 , 36.4 , 0
DATA 261.63, 72.8 , 100
DATA 261.63, 72.8 , 100
DATA 329.63, 72.8 , 100
DATA 392.00, 72.8 , 100
DATA -1 , -1 , -1
```

Listing 16. Ein einfaches Programm zur Sound-Ausgabe

Programm kann dann ohne einen Abbruch am Ort der Unterbrechung seine Tätigkeit wieder aufnehmen. Damit ist das Stichwort für diesen Abschnitt unseres Kurses gefallen: Unterbrechungen. Sie funktionieren ähnlich wie ein Unterprogrammssprung mit »GOSUB«. Im Unterschied zu dieser Technik wird eine Verzweigung jedoch nicht durch einen Programmbefehl, sondern durch ein bestimmtes Ereignis hervorgerufen. Solche Ereignisse können das Betätigen der Maus, ein Menüclick oder aber das Auftreten eines Fehlers sein. In dem jeweiligen Unterprogramm wird dann auf das entsprechende Ereignis reagiert. Auf diese Weise läßt sich die von der Workbench gewohnte Benutzeroberfläche einfach in Basic-Programme integrieren. Aber eben auch das Abfangen von Software-Fehlern und eine Reaktion darauf durch Befehle des Programmierers sind mit diesen Funktionen einfach zu realisieren.

Dem Fehler auf der Spur

Fehlerlos ist sicherlich kein Programm, zumindest solange es sich noch in der Entwicklung befindet. Als Programmierer müssen Sie schließlich alle Eventualitäten von Anfang an mit in Ihr Programm einplanen. Dies gilt besonders für die Fehler, die bei der Bedienung des Programms gemacht werden. Um Programmentwicklern die Arbeit zu erleichtern,

```
main:
  Wellenformen
  Spielen
  WHILE NOT MOUSE(0) : SLEEP : WEND
END
SUB Wellenformen STATIC
Variable:
  DIM dreieck%(255),rechteck%(255),rauschen%(255)
startsub:
  RANDOMIZE TIMER
  FOR i = 0 TO 63
    dreieck%(i) = 127 * i / 64
    dreieck%(i + 64) = 127 * (64-i)/64
    dreieck%(i + 128) = -127 * i / 64
    dreieck%(i + 192) = -127 * (64-i)/64
  NEXT i
  FOR i = 0 TO 127
    rechteck%(i) = 127
    rechteck%(i + 128) = -128
  NEXT i
  FOR i = 0 TO 255
    rauschen%(i) = 255 * RND - 128
  NEXT i
  WAVE 0,SIN
  WAVE 1,dreieck%
  WAVE 2,rechteck%
  WAVE 3,rauschen%
END SUB
SUB Spielen STATIC
  FOR k = 0 TO 3
    SOUND WAIT
    FOR i = 440 TO 880 STEP 40
      SOUND i,3.6,,k
    NEXT i
    SOUND RESUME
  NEXT k
END SUB
```

Listing 17. Die Programmierung verschiedener Wellenformen

bietet Amiga-Basic einerseits die Möglichkeit, Fehlermeldungen des Interpreters abzufangen, andererseits lassen sich auch neue mögliche Fehler definieren. Mit Hilfe des Befehls

ON ERROR GOTO Marke

können Sie dann auf das Auftreten eines dieser Fehler entsprechend reagieren. Doch

woher wissen Sie, welcher Fehler aufgetreten ist? Dazu dient die vordefinierte Variable »ERR«. Jedem Fehler wird von Basic eine bestimmte Nummer zugeordnet. Diese Zuordnung können Sie dem Anhang B Ihres Basic-Handbuches entnehmen. Wenn Sie dort jedoch nachschlagen, werden Sie feststellen, daß von den möglichen 255 Fehlercodes nur ein kleiner Teil eine Bedeutung hat, während den restlichen Fehlernummern der Fehler »Unprintable Error« zugeordnet ist. Genau hier können Sie eingreifen: der Befehl

ERROR Nummer

gestattet es Ihnen, jeden Fehler zu simulieren. Falls Sie zum Beispiel in Ihrem Programm irgendwo die Zeile »ERROR 2« plaziert haben, dann wird Ihr Programm genau dort mit der Fehlermeldung »Syntax Error« abgebrochen. Haben Sie nun aber zuvor irgendwo in Ihrem Programm die Befehlszeile »ON ERROR GOTO« Marke stehen, so verzweigt Ihr Programm erst zu der angegebenen Routine. Innerhalb dieser Routine ist es zweckmäßig, die Variable »ERR« auf Ihrem Wert zu überprüfen (in unserem Beispiel hat sie den Wert zwei). Abhängig von diesem Wert können Sie nun versuchen,

den Fehler zu beheben, oder aber zumindest eine Fehlermeldung ausgeben. Der Programmteil nach der durch den »ON ERROR GOTO«-Befehl angesprungenen Marke wird so lange abgearbeitet, bis Amiga-Basic auf den Befehl

RESUME

trifft. Den »RESUME«-Befehl gibt es in drei Variationen: Sie können »RESUME« alleine angeben — dann kehrt Basic genau zu dem Befehl zurück, der den Fehler ausgelöst hat, und führt ihn nochmals durch. Sie können aber auch eine Sprungmarke hinter »RESUME« angeben. In diesem Fall springt Basic zu dieser Marke. Dieses Verfahren ist jedoch mit größter Vorsicht zu genießen.

Wo ist der Übeltäter?

Es gibt eigentlich nur einen Fall, in dem es angewendet werden sollte: den Sprung an das Programmende. Bei der dritten Variation geben Sie nach dem »RESUME« das Schlüsselwort »NEXT« an. Dieser Zusatz bewirkt, daß die Programmausführung direkt nach dem Befehl fortfährt, der den Fehler verursachte.

Als Beispiel für die Fehlerverarbeitung soll Ihnen Listing 19 dienen. Mit dem Programm lassen sich Texte in eine sequentielle Datei speichern. Für den Fall, daß die angegebene Datei bereits auf der Diskette existiert, sollen die in ihr enthaltenen Texte auf dem Bildschirm ausgegeben und weitere Texte daran angehängt werden. Zunächst wird die Fehlerunterbrechung aktiviert. Um nach der Eingabe des Dateinamens zu verhindern, daß die Datei mit einem Leerstring eröffnet wird, wird in diesem Fall ein »Fehler« erzeugt. Dieser hat die Nummer 42 und tritt immer dann auf, wenn bei der Eingabe lediglich die Taste <Return> betätigt wird. Natürlich kann dieser Fall auch einfacher abgefangen werden. Unser Beispiel soll Ihnen lediglich den Nutzen dieser Funktion erläutern.

Tritt beim Öffnen der Datei zum Lesen der Fehler Nummer 53 auf, dann ist die angegebene Datei nicht vorhanden. Das Programm kann in diesem Fall mit einer erneuten Eröffnung zum Schreiben und der Eingabe der Texte fortfahren. Tritt dieser Fehler allerdings nicht auf, werden alle Sätze der Datei gelesen und auf dem Bild-

```
Variable:
  DIM huellkurve%(255)
main:
  REM 1. schnelles Ansteigen
  FOR i = 0 TO 30
    huellkurve%(i) = 255 * i / 30
  NEXT i
  REM 2. erstes abklingen
  FOR i = 31 TO 60
    huellkurve%(i) = 255 * (1 - (i-31)/100)
  NEXT i
  REM 3. halten
  FOR i = 61 TO 100
    huellkurve%(i) = huellkurve%(60)
  NEXT i
  REM 4. abklingen
  FOR i = 101 TO 255
    huellkurve%(i) = huellkurve%(100) * (1 - (i-101)/155)
  NEXT i
  PSET (4,190 - .7*huellkurve%(0))
  FOR i = 1 TO 255
    LINE -(2*i,190 - .7*huellkurve%(i))
  NEXT i
  FOR i = 0 TO 255
    SOUND 440,.5,huellkurve%(i)
  NEXT i
  WHILE NOT MOUSE(0) : SLEEP : WEND
END
```

Listing 18. Ein Programm zur Hüllkurvenbestimmung

schirm ausgegeben. Der in diesem Sonderheft benutzte Checksummer benutzt übrigens eine solche Routine.

Zur Fehlerbearbeitung bleibt jetzt nur noch eine — allerdings wenig nützliche — Systemvariable zu besprechen. Die Variable »ERL« enthält beim Auftreten eines Fehlers die letzte Zeilennummer, die vor dem Auftreten des Fehlers abgearbeitet wurde. Leider muß diese Zeilennummer auch wirklich eine Nummer sein. Label werden nicht erfaßt. Da Sie aber wegen der mächtigen strukturierten Befehle des Amiga-Basic normalerweise auf Zeilennummern verzichten können, lassen sich nur wenige Einsatzmöglichkeiten für diese Variable finden.

Der Programmierer als Fallensteller

Hinter der besprochenen Methode der Fehlerbearbeitung steckt ein allgemeines Programmierprinzip, das wir mit dem deutschen Wort Unterbrechungsverarbeitung vorgestellt haben. Wie üblich ist der entsprechende amerikanische Fachausdruck »event trapping« wesentlich anschaulicher. Denn dies ist genau die Absicht: Ein Programm zu schreiben, welches darauf wartet, daß ihm ein bestimmtes Ereignis in die Falle geht. Sobald dieses Ereignis eintritt, läßt Ihr Amiga alles stehen und liegen, um dieses sofort zu bearbeiten. Ist er damit fertig, kehrt er wieder zum eigentlichen Programm zurück, um es an der Stelle fortzusetzen, an der es unterbrochen wurde. Es gibt viele Ereignisse, die eine derartige Bearbeitung erfordern. Zwei davon sind Ihnen schon bekannt: Zum einen die oben besprochene Fehlerbehandlung, zum anderen die bei den Animationsbefehlen besprochene Verarbeitung von Zusammenstoßen grafischer Objekte. Weitere mögliche Ereignisse sind das Verstreichen einer gewissen Zeitspanne oder der Druck auf eine Maustaste.

Besonders für die Mausbedienung bietet Amiga-Basic mächtige Funktionen, die eine einfache Maus- und Menüsteuerung erlauben. Wie Sie ja schon von der Workbench gewohnt sind, besitzen die beiden Maustasten unterschiedliche Funktionen. Bekanntlicherweise dient die rechte Maustaste zur Auswahl von

```
Variable:
  dn$ = "" ' Dateiname
  Num = 0 ' Fehlernummer
ON ERROR GOTO Fehler
main:
  CLS
  Anfang:
    Num = 0
    INPUT "Dateiname";dn$
    PRINT
    IF dn$ = "" THEN ERROR 42
    IF Num = 42 THEN Anfang
    OPEN dn$ FOR INPUT AS #1
    IF Num = 53 THEN
      CALL OpenDatei(dn$)
    ELSE
      CALL ReadDatei
    END IF
  END
END
SUB OpenDatei(datei$) STATIC
  s$ = "" ' Zeile zum Schreiben in Datei
  PRINT
  PRINT " <*> beendet die Eingabe "
  PRINT
  OPEN datei$ FOR APPEND AS #1
  LINE INPUT "Satz: ";s$
  WHILE s$ <> ""
    PRINT #1,s$
    LINE INPUT "Satz: ";s$
  WEND
END SUB
SUB ReadDatei STATIC
  e$ = "" ' enthaelt eingelesene Zeile
  WHILE NOT EOF(1)
    LINE INPUT #1,e$
    PRINT e$
  WEND
END SUB
Fehler:
  Num = ERR
  IF Num = 42 THEN
    CLS
    PRINT "Keine leere Eingabe"
  ELSEIF Num <> 53 THEN
    PRINT "Programmfehler!"
    PRINT "Programm beendet"
    RESUME Ende
  END IF
  RESUME NEXT
```

Listing 19. Dieses Programm zeigt, wie Fehler abgefangen werden können. Es soll den Nutzen der Funktionen zur Fehlerbehandlung in Amiga-Basic verdeutlichen.

Pull-Down-Menüs, während mit der linken Maustaste Programme geladen beziehungsweise Icons oder Gadgets selektiert werden. Diese unterschiedliche Bedeutung der beiden Maustasten kommt auch in Amiga-Basic zum Tragen: Die beiden Befehle

ON MOUSE GOSUB Marke
ON MENU GOSUB Marke

machen dem Interpreter die Zeilennummer bekannt, zu der er nach dem entsprechenden Ereignis verzweigen soll. Wir wollen uns zunächst dem »MOUSE«-Befehl zuwenden, der sich auf die linke Maustaste bezieht. Falls Sie in Ihrem Programm am Anfang den Befehl »ON MOUSE GOSUB Maus« gefolgt von »MOUSE ON« setzen, wird jedesmal, wenn Sie die linke Maustaste betätigen, zur Sprungmarke »Maus« verzweigt. Hier müssen nun diejenigen Befehle stehen, die die Verarbeitung dieses Ereignisses übernehmen sollen.

MOUSE (n) Funktion

0	Atualisierung der Koordinaten
1	x-Position
2	y-Position
3	x-Position beim Drücken der Maustaste
4	y-Position beim Drücken der Maustaste
5	x-Position beim Loslassen der Taste
6	y-Position beim Loslassen der Taste

Die wichtigste dieser Funktionen ist »MOUSE(0)«. Damit wird für die restlichen sechs Funktionen »MOUSE(1)« bis »MOUSE(6)« die Position des Mauszeigers auf den aktuellen Stand gebracht. Dabei erhalten die ungeradzahlgigen »MOUSE«-Funktionen die x-Koordinate und die geradzahlgigen »MOUSE«-Funktionen jeweils die y-Koordinate des Mauszeigers. Doch wodurch unterscheidet sich zum Beispiel die mit »MOUSE(1)« ermittelte x-Position von der mit »MOUSE(3)« ermittelten? Am besten führen wir dazu ein kleines Experiment durch. Geben Sie Listing 20 ein und starten Sie das Programm. Bewegen Sie jetzt Ihren Mauszeiger über den Bildschirm, ohne dabei eine Maustaste zu betätigen. Wie Sie sehen, verändern sich nur die Werte in der ersten Spalte, während alle anderen Werte gleich bleiben. Damit haben wir auch schon unser erstes Ergebnis. Die Funktionen MOUSE(1) und MOUSE(2) zeigen die bei einem MOUSE(0)-Aufruf vorliegende x-beziehungsweise y-Position des Mauszeigers, unabhängig davon, ob Sie eine Maustaste gedrückt haben oder nicht.

Nun betätigen Sie die linke Maustaste und halten sie fest. Jetzt nehmen alle drei Spalten (beinahe) die gleichen Werte

Mit Hilfe der Funktion
MOUSE (n)

können Sie sich, je nach dem Wert von n, über den augenblicklichen Zustand der Maus informieren. In folgender Tabelle finden Sie die Ergebnisse, die bei den einzelnen n-Werten dieser Funktion geliefert werden.

```
Variable:
  f$ = "###" ' Formatstring zur PRINT USING - Anweisung
  dummy = 0 ' Hilfsvariable, nimmt MOUSE(0) Wert an
main:
  LOCATE 1,7 : PRINT "MOUSE(1/2)"
  LOCATE 1,27 : PRINT "MOUSE(3/4)"
  LOCATE 1,47 : PRINT "MOUSE(5/6)"
  LOCATE 2,1 : PRINT "x = "
  LOCATE 3,1 : PRINT "y = "
  WHILE 1
    dummy = MOUSE(0)
    LOCATE 2,10 : PRINT USING f$;MOUSE(1)
    LOCATE 3,10 : PRINT USING f$;MOUSE(2)
    LOCATE 2,30 : PRINT USING f$;MOUSE(3)
    LOCATE 3,30 : PRINT USING f$;MOUSE(4)
    LOCATE 2,50 : PRINT USING f$;MOUSE(5)
    LOCATE 3,50 : PRINT USING f$;MOUSE(6)
  WEND
END
```

Listing 20. So ermitteln Sie die Mausposition

an. Also haben wir herausgefunden, daß die Funktionen MOUSE(3) bis MOUSE(6) auf das Betätigen der Auswahl-taste reagieren. Ziehen Sie nun die Maus mit festgehaltener Auswahl-taste über den Bildschirm. Wie erwartet, ändert sich die erste Spalte. Aber auch in der dritten Spalte werden neue Werte angezeigt, während die zweite Spalte unverändert bleibt. Wir müssen das so interpretieren: Die »MOUSE«-Funktionen 3 und 4 geben die Koordinaten an, bei denen die Auswahl-taste gedrückt wurde, während die »MOUSE«-Funktionen 5 und 6 die Koordinaten bezeichnen, bei denen entweder die Auswahl-taste losgelassen wurde, oder aber — falls die Auswahl-

taste noch gedrückt ist — an welcher Position sich die Maus augenblicklich befindet.

Die Maus in der Falle

Damit sind wir soweit, um uns an Listing 21 wagen zu können. In der ersten Zeile befehlen wir Amiga-Basic, daß es sich bei jeder Betätigung der Auswahl-taste sofort an die mit dem Label »Maus« bezeichnete Stelle begeben soll. Die Aktivierung dieser Unterbrechungsverarbeitung haben wir durch den Befehl

MOUSE ON

programmiert. Ebenso wird mit dem Timer verfahren, wobei das Argument (hier eins) an-

gibt, in welchem Sekundenintervall dieser Aufruf durchgeführt werden soll. Dann soll ein Kreis mit dem Radius »r« und dem Mittelpunkt (xpos,ypos) gezeichnet werden. Falls Sie sich über die etwas seltsame »WHILE...WEND«-Schleifenkonstruktion wundern: Hinter dem »WHILE«-Befehl steht normalerweise eine logische Bedingung wie etwa »x < 100« und die »WHILE«-Schleife wird so lange ausgeführt, bis diese Bedingung unwahr ist. Nun ist es aber eine Eigenschaft von Basic, allen von Null verschiedenen Zahlen den logischen Wert »wahr« zuzuordnen, während die Zahl Null logisch »falsch« entspricht.

Damit ist der Wert eins gleichbedeutend mit »wahr« und so haben wir eine Endlosschleife konstruiert. Wäre Ihr Amiga ein Home-Computer ohne die Fähigkeiten des Multitasking, wäre der im Schleifenkörper enthaltene Befehl »SLEEP« sinnlos, da sich Basic im Moment sowieso in einer Endlosschleife befindet. Der »SLEEP«-Befehl führt jedoch dazu, daß Ihr Basic-Programm so lange angehalten wird, bis ein Unterbrechungsereignis eintritt. Gegenüber der normalen Endlosschleife hat dies den Vorteil, daß Ihr Programm anderen, gleichzeitig laufenden Programmen, mehr Laufzeit zur Verfügung stellt.

Der Amiga schläft

Das Basic-Programm wartet in diesem Zustand also nur darauf, daß Sie die Maustaste bedienen. Welche Funktion hat das Programm nun? In der ersten Zeile nach dem Label »Maus:« wird der augenblickliche Zustand der Maustaste überprüft. Das Programm unterscheidet zwei Zustände. Erstens: die Maustaste wurde ein-, zwei- oder dreimal kurz betätigt. Zweitens: die Maustaste bleibt längere Zeit gedrückt. Da Ihr Amiga ein sehr schneller Computer ist, werden Sie keine Zeit haben, den Finger von der Maus zu nehmen: schon wird angezeigt, daß die Auswahl-taste gegenwärtig gedrückt ist. Um Ihnen die Zeit zu geben, den Finger von der Maus zu nehmen, warten wir eine Sekunde. Danach wird die Maustaste mit »MOUSE(0)« nochmals abgefragt. An dieser Stelle sollten Sie nun wissen, was die Zahlenwerte, die Sie durch diese Funktion erhalten, bedeuten. Liefert die

Funktion MOUSE(0) als Ergebnis eine Null, so heißt dies, daß die Maus seit dem letzten Aufruf von MOUSE(0) nicht betätigt wurde. Mit einer Eins teilt uns der Interpreter einen einmaligen Tastendruck mit. Die Auswahl-taste war jedoch zur Zeit des Aufrufs nicht mehr gedrückt. Analog bedeutet eine Zwei, daß die Maustaste zweimal betätigt wurde und im Moment nicht aktiv ist. Eine Drei gibt an, daß die Taste mehr als zweimal betätigt, danach aber losgelassen wurde.

Negative Zahlen informieren ebenso über die Anzahl der Mausklicks. Allerdings mit dem Unterschied, daß die Taste auch im Moment des Aufrufs noch niedergehalten wurde. Wegen Sie die Maus nun an irgendeine Stelle des Bildschirms. Halten Sie die Maustaste niedergedrückt. Der Kreis verändert seine Position und befindet sich nun unterhalb des Mauszeigers. Durch Verschieben der Maus läßt sich der Kreis an eine beliebige Stelle des Bildschirms plazieren. Die Bewegung stoppt mit Loslassen der Auswahl-taste. Durch eine kurze Betätigung derselben Taste können Sie die Farbe beim nächsten Zeichenvorgang bestimmen. Ein einmaliger Klick bedeutet, daß der Kreis in Weiß gezeichnet wird; zwei- und dreimalige Klicks verändern die Farbe in Schwarz beziehungsweise Rot (vorausgesetzt, die Farben auf der Workbench besitzen die Standardeinstellung). Das Programm endet selbsttätig, wenn Sie zehn Sekunden die Maus nicht betätigt haben. Hierfür befindet sich in der Unterbrechungs-routine für den Timer ein Zähler, der bei jedem Einsprung um eins erhöht wird. Hat er den Wert zehn erreicht, endet das Programm. Jede Mausbetätigung muß diesen Zähler natürlich wieder auf Null setzen. Damit ist das Programmende genau zehn Sekunden nach dem letzten Bedienereingriff gewährleistet.

Das Menü zum Dessert

Die rechte Taste der Maus, die sogenannte Menütaste, hat eine gänzlich andere Aufgabe. Wenn Sie die Menüleiste mit dem Mauszeiger anfahren und dann die Menütaste drücken, erscheint die Menüleiste. Wählen Sie nun eine dieser Überschriften mit einem Mausklick aus, so klappt eine Liste von Auswahlpunkten in den Bild-

```
ON MOUSE GOSUB Maus
ON TIMER(1) GOSUB Uhr
Variable:
xpos   = 300 ' Setzen des Kreises
ypos   = 100
r       = 10
Farbe   = 1
Status% = 0 ' MOUSE(0)-Status
Flag%   = 0 ' Flag fuer Maus-Betaetigung
Zeit    = 1 ' Programm beenden, wenn Zeit = 0
i%      = 0 ' Schleifenzaeher

main:
CIRCLE (xpos,ypos),r,Farbe
MOUSE ON ' Unterbrechungsverarbeitung ein
TIMER ON
WHILE Zeit
    SLEEP ' Basic schlaeft jetzt
WEND
END
Maus:
Flag% = 0
Status% = MOUSE(0)
LOCATE 2,10
PRINT "Status 1: ";Status%
FOR i% = 1 TO 250 : NEXT i% ' Verzoeigerungsschleife
Status% = MOUSE(0)
LOCATE 3,10
PRINT "Status 2: ";Status%
IF Status% < 0 THEN
    WHILE Status% < 0
        CIRCLE (xpos,ypos),r,0
        xpos = MOUSE(1)
        ypos = MOUSE(2)
        CIRCLE (xpos,ypos),r,Farbe
        Status% = MOUSE(0)
        LOCATE 3,10
        PRINT "Status 2: ";Status%
        Flag% = 0
    WEND
ELSE
    Farbe = Status%
END IF
RETURN
Uhr:
LOCATE 2,60 : PRINT DATE$
LOCATE 3,61 : PRINT TIME$
Flag% = Flag% + 1
IF Flag% = 10 THEN
    LOCATE 10,10
    PRINT " ... Zeit abgelaufen "
    Zeit = 0
END IF
RETURN
```

Listing 21. Ein einfaches Programm mit Maussteuerung

schirm. Genau dieses Verhalten können Sie auch von Basic aus steuern. Als Beispiel steht Ihnen Listing 22 zur Verfügung. Zunächst muß der Interpreter wissen, wie viele Menüs Sie anlegen wollen. Die Anzahl der Menüs ist auf zehn beschränkt, mehr Überschriften passen in der Regel auch nicht in die Menüleiste. Eine weitere Beschränkung gilt für die Anzahl der Unterpunkte. Ein Menü darf maximal 19 davon haben. Die Unterpunkte selbst dürfen in Basic keine weiteren Untermenüs enthalten.

Das Programm aus Listing 22 beschränkt sich auf zwei Hauptmenüs. Wir haben sie einfach »Menü 1« und »Menü 2« genannt. Eingerichtet werden diese Menüs durch den »MENU«-Befehl. Mit der ersten Nummer im Anschluß an das Schlüsselwort machen Sie dem Basic-Interpreter bekannt, an welcher Stelle der Menüleiste das Menü stehen soll. Mit einer Eins an dieser Stelle legt der Interpreter dieses an die linke Seite und mit einer zehn wird es an der rechten Seite platziert. Dazwischen liegende Werte bestimmen die Position entsprechend zwischen den beiden Enden der Menüleiste. Der zweiten Zahl nach dem »MENU«-Befehl kann Ihr Amiga entnehmen, an welcher Stelle innerhalb des Hauptmenüs der angegebene Unterpunkt platziert werden soll. Eine Null an dieser Stelle besagt, daß es sich bei der angegebenen Zeichenkette um den Menütitel selbst handelt. Mit Werten von 1 bis 19 kann nun jeder Menüunterpunkt im Menü positioniert werden. Der Sinn des dritten Wertes wird von vielen Programmierern noch unterschätzt. Mit ihm läßt sich der entsprechende Menüpunkt aktivieren oder deaktivieren. Setzen Sie hier eine Eins ein, erscheint in Ihrem Menü der entsprechende Schriftzug in normaler Schrift.

Ihre Auswahl bitte...

Das Menü ist aktiviert und kann durch einen Mausklick ausgewählt werden. Es gibt allerdings Situationen während des Programmablaufs, bei denen der Programmierer den Bediener an der Auswahl eines Unterpunktes oder Menüs hindern möchte. Dies kann er durch eine Null als dritten Wert erreichen. Das Menü erscheint dann in der Ihnen sicher bekannten »Geisterschrift«. Ein

derart gekennzeichnete Menüpunkt wird zwar noch angezeigt, die Unterbrechungssteuerung reagiert allerdings nicht mehr auf seine Auswahl. Wird das Hauptmenü selbst so gekennzeichnet, dann ist das gesamte Menü nicht mehr verfügbar. Eine weitere Hilfe für den Bediener ist die Wahl des Wertes zwei an dieser Stelle. Der Amiga setzt dann vor den Menütext einen Haken. Dabei sollten Sie allerdings beachten, daß dieser Haken Platz beansprucht. Falls Sie einen Haken vor den Menüpunkt setzen wollen, sollten Sie daher zwei Leerzeichen vor den eigentlichen Text setzen. Sinn ergibt diese Hakentechnik bei Menüpunkten, die eine Schalter-

```
DATA "Menu 1", " Punkt 1.1",
      " Punkt 1.2", " Punkt 1.3",
      " Ende",*
DATA "Menu 2", " Punkt 2.1",
      " Punkt 2.2",
      " Punkt 2.3",*
DATA #

main:
  i% = 1
  READ mt$
  WHILE mt$ <> "*"
    MENU i%,0,1,mt$
    j% = 1
    READ mup$
    WHILE mup$ <> "*"
      MENU i%,j%,1,mup$
      Flag(i%,j%) = 1
      j% = j% + 1
      READ mup$
    WEND
    i% = i% + 1
    READ mt$
  WEND

  ON MENU GOSUB Wahl
  ON BREAK GOSUB Ende
  MENU ON
  BREAK ON
  WHILE 1 : SLEEP : WEND ' Ba
    sic schläft

Ende:
MENU RESET
END

Wahl:
  m% = MENU(0) : p% = MENU(1)
  ON m% GOSUB Menu1,Menu2
  RETURN
Menu1:
  ON p% GOSUB Menu11,Menu12,
    Menu13,Ende
  RETURN
Menu2:
  ON p% GOSUB Menu21,Menu22,
    Menu23
  RETURN

Menu11:
Menu12:
Menu13:
Menu21:
Menu22:
Menu23:
  IF Flag(m%,p%) = 1 THEN
    Flag(m%,p%) = 2
  ELSE
    Flag(m%,p%) = 1
  END IF
  MENU m%,p%,Flag(m%,p%)
  RETURN
```

Listing 22. So programmiert man eigene Menüs

funktion haben. In einer Textverarbeitung lassen sich so beispielsweise aktivierte Schriftattribute (fett, kursiv, unterstrichen, usw.) kennzeichnen.

Das Programm (Listing 22) beginnt mit zwei DATA-Zeilen. Diese enthalten die einzelnen Menütexte. Am Anfang der Zeile befindet sich der Menütitel, es folgen die Unterpunkte. Jedes Menü wird mit einem »*« abgeschlossen. Das Ende der Daten zeigt »#« an. Die folgende »WHILE«-Schleife aktiviert die Menüs. Durch eine Änderung der DATA-Zeilen (Hinzufügen/Löschen von Texten) lassen sich ohne Veränderung der Programmanweisungen die Menüs einfach umstellen. Nach einer Aktivierung der Unterbrechungsverarbeitung schicken wir den Amiga wie gehabt in den »Schlaf«. Mit den Anweisungen

```
ON BREAK GOSUB Ende
BREAK ON
```

befehlen wir dem Interpreter, auf die Marke »Ende:« zu verzweigen, wenn der Bediener das Programm mit den Tastenkombinationen <Amiga .> oder <CTRL C> beendet. Damit ist gewährleistet, daß auch in diesen Fällen mit

```
MENU RESET
```

die Menüs wieder aus der Menüleiste entfernt werden. Der Befehl »ON BREAK...« muß jedoch mit Vorsicht gehandhabt werden. Sie können damit leicht eine Endlosschleife konstruieren, aus der Sie nur durch die »beliebte« Tastenkombination <Ctrl Amiga> wieder herauskommen.

Nach Auswahl eines Menüpunktes verzweigt der Interpreter in die mit »Wahl:« gekennzeichnete Routine. Dort ermittelt das Programm mit der Funktion »MENU(0)« die Nummer des angewählten Menüs und durch »MENU(1)« den entsprechenden Unterpunkt. Die Verzweigung auf die zu jedem Menüpunkt gehörende Unterbrechungsroutine ist mit zwei »ON GOSUB«-Anweisungen realisiert. In unserem Programm münden aber alle Verzweigungen wieder in einem Programmteil zusammen. Es soll uns ja nur die Menüauswahl demonstrieren. Mit Hilfe des Feldes »Flag%« wird dort lediglich ein Schalter umgelegt und das Menü entsprechend mit Haken versehen oder der Haken gelöscht. Die besprochene Thematik bietet sehr viel Raum für eigene Experimente.

Sie werden beim Ausprobieren dieser Technik begeistert feststellen, wie einfach sich Programme damit strukturieren lassen.

Basic ruft Assembler

Kommen wir nun zu einem komplizierteren Thema: dem Einbinden von Maschinenprogrammen in Basic. Wir halten uns dabei im wesentlichen an das in Ihrem Basic-Handbuch in Kapitel 6.2 angegebene Beispiel. Das dort abgedruckte Programm enthält allerdings einen Druckfehler und zwei Ungenauigkeiten, die in Listing 23 behoben sind. Aufgabe dieses Programms ist es, in einer beliebigen Zeichenkette alle Kleinbuchstaben in Großbuchstaben umzuwandeln. Da das im Basic-Handbuch angegebene Assemblerprogramm ausreichend kommentiert ist, können wir uns hier eine weitere Dokumentation sparen. Wir gehen davon aus, daß Ihnen ein Assembler zur Verfügung steht, um das Programm in ausführbaren Maschinencode zu übersetzen. Das in Listing 23 abgedruckte Programm wurde mit dem zum Lattice C-Compiler 4.0 mitgelieferten Makro-Assembler »asm« übersetzt und danach mit dem Programm »blink« gelinkt. Einen Hexdump des erzeugten, ausführbaren Files sehen Sie in Bild 17. Was sofort auffällt, ist eine gewisse Diskrepanz in der Dateilänge zur Länge des eigentlichen Maschinenprogramms. Während des Assemblierens wurde die Länge des Maschinenprogramms mit 56 Byte angegeben. Das vom Linker erzeugte File ist aber 92 Byte lang. Die zusätzlichen Byte enthalten Informationen für Adreßumrechnungen. Da dies aber nur notwendig ist, wenn

```
CSECT CODE
movem.l a0/d0-d1,-(sp)
move.l 16(sp),d0
move.l 20(sp),a0
clr.l d1
bra test

start:
move.b 0(a0,d0.w),d1
cmp.b #$61,d1
blt test
cmp.b #$7a,d1
bgt test
and.b #($ff-$20),
0(a0,d0.w)

test:
dbf d0,start
movem.l (sp)+,a0/d0-d1
rts
END
```

Listing 23. Assemblercode zur Stringumwandlung

```

0000: 000003F3 00000000 00000001 00000000 .....
0010: 00000000 0000000E 000003E9 0000000E .....
0020: 48E7C080 202F0010 206F0014 42816000 H... /.. o..B..
0030: 001C1230 00000C01 00616000 00100C01 ...0....am....
0040: 007A6E00 00080230 00DF0000 51C8FFE4 .zn...0....Q...
0050: 4CDF0103 4E750000 000003F2 l...Nu.....

```

Bild 17. Hex-Dump des fertigen Programms

im Assemblerprogramm absolute Adressen verwendet werden — und dies ist in unserem Beispiel nicht der Fall — können wir diese Informationen überlesen. Die für uns interessante Stelle wird durch die Hex-Zahl \$000003E9 gekennzeichnet.

Diese Zahl teilt dem Amiga mit, daß das nächste Langwort (vier Byte) die Länge des Maschinencodes in Langworten enthält und der Maschinencode selbst im übernächsten Langwort beginnt. Damit ist unsere Strategie zum Einlesen des Maschinencodes klar vorgezeichnet: Das gesamte vom Linker erzeugte File wird in eine Stringvariable gelesen und die Hex-Zahl \$000003E9 gesucht. Dann wird die Länge des Maschinencodes bestimmt und dieser in eine neue Stringvariable eingelesen. Hier muß allerdings ein kleiner Trick angewandt werden: Es scheint, als ob Strings, die eine gerade Anzahl von Byte enthalten, immer bei einer ungeraden Adresse beginnen. Maschinenprogramme müssen dagegen immer bei einer geraden Adresse starten. Daher ist es unbedingt notwendig, daß der String, der das Maschinenprogramm enthält, eine ungerade Länge erhält — wir addieren zur errechneten Stringlänge eine 1. Das in Listing 24 angegebene Unterprogramm »ReadCode« führt genau diese Aktionen aus. Eine Einschränkung besteht noch für die hier angegebene Methode, Maschinenprogramme einzulesen: Da Strings nicht mehr als 32 KByte Zeichen enthalten können, muß auch das Maschinenprogramm kürzer als 32 KByte sein.

Wir haben das Maschinenprogramm nun gelesen. Wie wird es aber nun zur Ausführung gebracht? Hier hilft uns wieder der »CALL«-Befehl. Nach »CALL« muß nicht unbedingt der Name eines Unterprogramms stehen, sondern es kann ebenso eine Variable angegeben werden, die die Anfangsadresse eines Maschinenprogramms enthält. Diese Anfangsadresse unseres Strings, der das Maschinenprogramm enthält, erhalten wir mit der Funktion »SADD«.

Die vom Unterprogramm benötigten Parameter sind wie gewohnt innerhalb der Klammern anzuführen. Sie werden auf dem Stack abgelegt, von wo sie vom Maschinenprogramm geholt werden können. Der Stack ist eine Art Zwischenspeicher, in welchem Daten gestapelt abgelegt und nach dem sogenannten LIFO-Prinzip abgearbeitet werden. »LIFO« steht für »Last in — first out« und bedeutet, daß beim Ablegen von neuen Daten auf den Stapel, die am Fuße des Stapels liegenden Daten abgearbeitet werden. Bei der Übergabe der Parameter an das Unterprogramm gibt es eine Einschränkung zu beachten: Es können nur kurze und lange Ganzzahlen übergeben werden.

Von den anderen Variablentypen (Arrays, Strings...) müssen Sie die Adresse übergeben. Diese läßt sich einfach mit »VARPTR« oder »SADD« ermitteln.

Wenn Sie beim Aufruf von Maschinenprogrammen von Basic aus Systemabstürze vermeiden wollen, müssen Sie unbedingt eine Eigenart von

```

ReadCode "Code",McProg$
PRINT "Geben Sie einen String
ein: "
INPUT st$
UPPER st$
PRINT st$
END
SUB UPPER(argument$) STATIC
  SHARED McProg$
  adr% = SADD(argument$)
  laeng% = LEN(argument$)
  entry% = SADD(McProg$)
  CALL entry%(laeng%,adr%)
END SUB
SUB ReadCode(filename$,
  AsmCode$) STATIC
  OPEN filename$ FOR INPUT
  AS #1
  file$ = INPUT$(LOF(1), #1)
  CLOSE #1
  HunkCode% = &H3E9
  i = 1
  code% = CVL(MID$(file$,
    1,4))
  WHILE (code% <> HunkCode%)
    i = i + 4 : REM Nächste
    es Langwort
    code% = CVL(MID$(file$,i,
    4))
  WEND
  CodeLength% = 4*CVL(MID$(
    file$,i+4,4))+1
  AsmCode% = MID$(file$,i+8,
    CodeLength%)
END SUB

```

Listing 24. Beispiel für das Einbinden assemblierter Maschinenprogramme

Basic berücksichtigen: Sobald Sie im laufenden Programm eine neue Variable benutzen, werden sich die Anfangsadressen aller Strings ändern. Sie sollten daher die Startadresse Ihres Maschinenprogramms erst unmittelbar vor dem Unterprogrammaufruf ermitteln. Listing 24 gibt Ihnen auch hierzu ein Beispiel.

Es geht auch ohne Assembler

Vielleicht haben Sie keinen Assembler und ärgern sich jetzt, weil wir Ihnen etwas schmackhaft gemacht haben, das Sie nicht realisieren können. Keine Angst — zur Not und mit etwas Handarbeit geht es auch ohne Assembler. Die Lösung finden Sie in Listing 25. Kurz zur Funktionsweise dieses Programms: Jedem in Listing 23 angegebenen Assemblerbefehl entspricht (mindestens) eine kurze Ganzzahl. Die unserem Maschinenprogramm entsprechenden Zahlen sehen Sie im Assemblerlisting in Ihrem Basic-Handbuch am linken Rand. Diese Zahlen kann man in »DATA«-Zeilen schreiben und diese dann mit Hilfe des »READ«-Befehles lesen. In diesem Fall muß man allerdings für die Dimensionierung im voraus wissen, wie lang das Maschinenprogramm ist. Kennt das Basic-Programm erst einmal die Anzahl der Maschinenbefehle, so kann ein entsprechendes Ganzzahlfeld dimensioniert werden. In dieses Feld wird nun das Maschinenprogramm eingelesen. Die Anfangsadresse erhält man durch die Anwendung der »VARPTR«-Funktion auf das erste Feldelement.

Bibliotheken sparen Zeit

Die Organisation des Betriebssystems in Libraries (Bibliotheken) sichert einen einfachen Zugriff auf die Routinen der Systemsoftware. Die Software-Entwickler des Amiga haben aber nicht nur für das Betriebssystem, sondern auch für Programmsammlungen, die man nicht direkt dem Betriebssystem zuordnen kann, diese Organisationsform gewählt. Ein Beispiel dafür ist eine Library zur Berechnung der auch als »transzendente« Funktionen bezeichneten Winkelfunktionen (Sinus, Kosinus, Tangens, und andere). Leider können Sie die Library-Funktionen nicht direkt an-

```

DEFINT a-Z : REM Alle Variablen
Integer
READ CodeLength
DIM McProg(CodeLength)
FOR i = 0 TO CodeLength - 1
  READ McProg(i)
NEXT i
PRINT "Geben Sie einen String
ein: "
INPUT st$
UPPER st$
PRINT st$
END

SUB UPPER(argument$) STATIC
  SHARED McProg$
  adr% = SADD(argument$)
  laeng% = LEN(argument$)
  entry% = VARPTR(McProg(0))
  CALL entry%(laeng%,adr%)
END SUB

DATA 27
DATA &h48e7f,&hc080
DATA &h202f,&h0010
DATA &h206f,&h0014
DATA &h4281
DATA &h6000,&h001c
DATA &h1230,&h0000
DATA &h0c01,&h0061
DATA &h6400,&h0010
DATA &h0c01,&h007a
DATA &h6e00,&h0008
DATA &h0230,&h00df,&h0000
DATA &h51c8,&h8ffe4
DATA &h4cdf,&h0103
DATA &h4e75

```

Listing 25. Einbinden von Maschinencode in Basic-Programme

sprechen, da sich die Parameterübergabe des Amiga-Basic von der der Libraries unterscheidet. Deshalb müssen vorher noch einige Vorbereitungen getroffen werden. Um die Library »mathtrans.library« benutzen zu können, müssen Sie dem Interpreter erst mitteilen, welche Funktionen in dieser Library vorhanden sind und in welchem Prozessorregister die Parameter übergeben werden sollen. Auf Ihrer Extras-Diskette finden Sie eine Schublade mit dem Namen »FD1.2« (sofern Ihr Amiga unter Kickstart 1.2 läuft). Um sich die in dieser Schublade enthaltenen Files betrachten zu können, müssen Sie allerdings den CLI-Editor »ed« verwenden. Das für uns interessante File heißt »mathtrans.lib.fd« (siehe Bild 18). In diesem File wird genau beschrieben, welche Funktionen in der Bibliothek vorhanden sind und in welchen Registern die Parameter übergeben werden müssen. Amiga-Basic kann mit der Form der Daten in dieser Datei allerdings nichts anfangen. Deshalb befindet sich in der Schublade »BasicDemos« das Programm »ConvertFD«, welches die Daten der FD-Datei in ein für Basic verständliches Format umwandelt. Die vom Programm generierte Datei


```
##base _MathTransBase
##bias 30
##public
SPAtan(float) (D0)
SPSin(float) (D0)
SPCos(float) (D0)
SPTan(float) (D0)
SPSincos(leftFloat,rightFloat) (D1,D0)
SPSinh(float) (D0)
SPCosh(float) (D0)
SPTanh(float) (D0)
SPExp(float) (D0)
SPLog(float) (D0)
SPPow(leftFloat,rightFloat) (D1,D0)
SPSqrt(float) (D0)
SPTieee(float) (D0)
SPFieee(integer) (D0)

*-----
*
* New functions added for Release 1.1
*
*-----
SPAsin(float) (D0)
SPAcos(float) (D0)
SPLog10(float) (D0)
##end
```

Bild 18. Die FD-Dateien enthalten die Funktionsnamen und Parameterinformationen einer Bibliothek

heißt »mathtans.bmap«. Wie sieht nun das Format eines solchen ».bmap«-Files aus? Dazu sollten wir uns erst einmal überlegen, welche Informationen der Basic-Interpreter eigentlich benötigt, um die jeweilige Funktion aufrufen zu können. Dies wäre zunächst einmal der Name der Funktion selbst. Außerdem muß die Sprungadresse, also die Stelle, an der sich die Routine innerhalb der Library befindet, bekannt sein.

Schließlich muß der Interpreter noch wissen, welche Parameter die Funktion benötigt und in welches der Prozessorregister diese übergeben werden. Genau diese Informationen stehen in der ».bmap«-Datei.

Für jede Funktion — und eine Funktion ist ja nichts anderes als ein Unterprogramm — steht dort der von einem Null-Byte abgeschlossene Funktionsname als ASCII-Code, gefolgt von der Position dieser Funktion in der Library. Dabei wird die Position relativ zum Anfang der Library als 16-Bit-Offset ausgedrückt. (Ein Offset ist ein Wert, der zu einer Basisadresse — in diesem Fall diejenige der Library — addiert werden muß.) Danach folgt für jedes Register, in dem ein Wert übergeben wird, ein Byte. Dieses Byte enthält eine Zahl, die angibt, welches Register betroffen ist. Dabei stehen die Zahlen 1 bis 8 für die Register D0 bis D7, die Zahlen 9 bis 13 für A0 bis A4. Nach der letzten

Registerangabe folgt ein Null-Byte, um Ihrem Amiga mitzuteilen, daß die Beschreibung dieses Funktionsaufrufes beendet ist.

Für Funktionen, die sich an die Konventionen der Parameterübergabe der Sprache C halten, ist die Registerparameterliste leer, da die Parameter dann auf dem Stack übergeben werden.

Das FFP-Format

Bei einem Aufruf einer Library-Funktion müssen Sie genau unterscheiden, ob diese Funktion ein Resultat zurückgibt oder nicht. Falls keiner Ihrer Funktionsaufrufe einen Wert an Ihr Basic-Programm zurückgibt, genügt es, die Library mit dem Befehl **LIBRARY <Library-Name>** dem Basic-Interpreter bekannt zu machen. In unserem Beispiel (Listing 26) ist dies nicht der Fall und wir müssen Basic mitteilen, welchen Variablentyp die Funktionen zurückliefern. Eine FFP-Variable ist nun gerade so lang, wie eine lange Ganzzahl. Deshalb haben wir alle Funktionen mit Hilfe von

```
DECLARE FUNCTION <Funktionsname> LIBRARY
```

als solche definiert. Die in Klammern angegebenen Funktionsargumente müssen Sie nicht angeben, sie bieten aber eine wertvolle Gedächtnisstütze. Nach der Erzeugung der ».bmap«-Dateien treffen wir hier auf unser zweites Pro-

blem: Wie bringen wir unsere normalen Gleitpunktzahlen in das FFP-Format? Die Zuweisung »ffp% = Gleitzahl« löst unser Problem sicher nicht, da bei einer derartigen Zuweisung der Nachkommaanteil einfach abgeschnitten wird. Das liegt aber nicht in unserem Sinne. Auch die einfache Benutzung einer normalen Gleitpunktzahl ist nicht der richtige Weg, da sich das FFP-Format leider von dem, bei den normalen Gleitpunktzahlen verwendeten, IEEE-Format unterscheidet. Wir kommen aber zum Ziel, wenn wir einen kleinen Umweg über Strings in Kauf nehmen. Mit Hilfe der Funktion »MK\$(<Gleitpunktzahl>)<« wird ein vier Byte langer String produziert, aus dem mit der Funktion »CVL(<String>)<« wieder eine lange Ganzzahl gewonnen werden kann. Nun können Sie diese Ganzzahl an die Funktion »SPFieee« übergeben, die daraus eine FFP-Zahl macht. Da dies während solcher Problemstellungen öfter vorkommen kann, haben wir für Sie die Funktion »FNftoffp%« definiert, die eine Gleitzahl in das FFP-Format überführt. Die danach definierte Funktion »FNfptof« leistet genau das Umgekehrte: Sie wandelt eine FFP-Zahl in eine Gleitpunktzahl um.

In unserem Beispiel wird nun der IEEE-Ganzzahl »ieee« der Wert 0.5 zugewiesen, dieser Wert dann in das FFP-Format konvertiert und danach eine Reihe von transzendenten Funktionen aufgerufen. Mit dem Befehl

LIBRARY CLOSE

lassen sich die geöffneten Libraries wieder schließen. Mit dieser sicher nicht einfachen Thematik beschließen wir unseren Basic-Kurs. Wenn Sie diesen aufmerksam durchgearbeitet haben, besitzen Sie jetzt nicht nur eine Menge neues Wissen, sondern auch eine Vielzahl von Anregungen für eigene Experimente.

Setzen Sie Ihre Ideen in die Tat um. Bauen Sie das vorhandene Wissen durch »Learning by Doing« aus. Behalten Sie jedoch Ihre Kunstwerke nicht für sich, sondern senden Sie uns diese für eine Veröffentlichung zu, damit auch andere von Ihren Ideen profitieren können. Wir freuen uns über jedes eingesandte Listing. Abgedruckte Listings werden natürlich entsprechend honoriert. Vielleicht steht dann eines Tages Ihr Name unter dem Programm des Monats im AMIGA-Magazin.

(Jürgen Singer/Martin Jobst/rs)

```
DECLARE FUNCTION SPAsin% (ffp%) LIBRARY
DECLARE FUNCTION SPAcos% (ffp%) LIBRARY
DECLARE FUNCTION SPAtan% (ffp%) LIBRARY
DECLARE FUNCTION SPSin% (ffp%) LIBRARY
DECLARE FUNCTION SPCos% (ffp%) LIBRARY
DECLARE FUNCTION SPTan% (ffp%) LIBRARY
DECLARE FUNCTION SPSinh% (ffp%) LIBRARY
DECLARE FUNCTION SPCosh% (ffp%) LIBRARY
DECLARE FUNCTION SPTanh% (ffp%) LIBRARY
DECLARE FUNCTION SPExp% (ffp%) LIBRARY
DECLARE FUNCTION SPLog% (ffp%) LIBRARY
DECLARE FUNCTION SPSqrt% (ffp%) LIBRARY
DECLARE FUNCTION SPTieee% (ffp%) LIBRARY
DECLARE FUNCTION SPFieee% (long%) LIBRARY
```

```
LIBRARY "mathtans.library"
```

```
DEF FNftoffp(float) = SPFieee%(CVL(MK$(float)))
DEF FNfptof(ffp%) = CWS(MKL$(SPTieee%(ffp%)))
```

```
ieee = .5
ffp% = FNftoffp(ieee)
ffpAsin% = SPAsin%(ffp%) : PRINT "asin(";ieee;") = "; FNfptof(ffpAsin%)
ffpAcos% = SPAcos%(ffp%) : PRINT "acos(";ieee;") = "; FNfptof(ffpAcos%)
ffpAtan% = SPAtan%(ffp%) : PRINT "atan(";ieee;") = "; FNfptof(ffpAtan%)
ffpSin% = SPSin%(ffp%) : PRINT "sin(";ieee;") = "; FNfptof(ffpSin%)
ffpCos% = SPCos%(ffp%) : PRINT "cos(";ieee;") = "; FNfptof(ffpCos%)
ffpTan% = SPTan%(ffp%) : PRINT "tan(";ieee;") = "; FNfptof(ffpTan%)
ffpSinh% = SPSinh%(ffp%) : PRINT "sinh(";ieee;") = "; FNfptof(ffpSinh%)
ffpCosh% = SPCosh%(ffp%) : PRINT "cosh(";ieee;") = "; FNfptof(ffpCosh%)
ffpTanh% = SPTanh%(ffp%) : PRINT "tanh(";ieee;") = "; FNfptof(ffpTanh%)
ffpExp% = SPExp%(ffp%) : PRINT "exp(";ieee;") = "; FNfptof(ffpExp%)
ffpLog% = SPLog%(ffp%) : PRINT "log(";ieee;") = "; FNfptof(ffpLog%)
ffpLog10% = SPLog10%(ffp%) : PRINT "log10(";ieee;") = "; FNfptof(ffpLog10%)
ffpSqrt% = SPSqrt%(ffp%) : PRINT "sqrt(";ieee;") = "; FNfptof(ffpSqrt%)
```

```
LIBRARY CLOSE
END
```

Listing 26. Einbinden der transzendenten FFP-Funktionen

WIEVIELE NEIN AKZEPTIEREN SIE?

	JA	NEIN
Deutsche Anleitung?	☒	☐
Deutscher Zeichensatz?	☒	☐
Freies Plazieren von farbiger Grafik und Text?	☒	☐
Für Nadel-, Laser- und Tintenstrahldrucker?	☒	☐
Für Postscriptdrucker auch?	☒	☐
Workbench 1.3 Treiber?	☒	☐
Ladbare Zeichensätze?	☒	☐
Kopierschutz?	☐	☒
IFF-Grafiken einbinden?	☒	☐
Beliebig lange Texte verarbeiten?	☒	☐
Großzügige Update-Regelung?	☒	☐



Shakespeare ist von

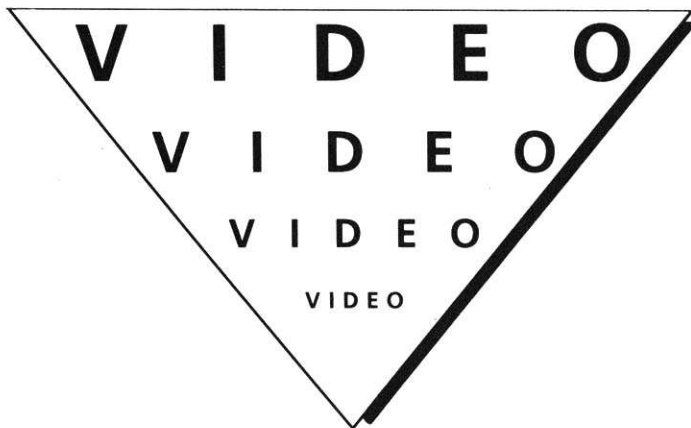
INFINITY

Infinity Software, Inc

—HS&Y

Herausgeber der deutschen Versionen von:
MaxiPlan, Microfiche Filer,
Digi-Paint, Digi-View, Videoeffects 3D,
Butcher 2.0, Sculpt-Animate 3D,
Shakespeare u.a.

EMPFOHLENER
VERKAUFSPREIS
DM 348,—



Video Effects 3-D

Spezialeffekte zur Betitelung von Videofilmen mit Texten und Logos, die zuvor mit Deluxe Paint oder TV * Text in HIRES entworfen wurden.

- NEU!**
- Motive im Raum bewegen, Zoomen, Verkleinern, Spiraldrehung
 - Überblenden, Schatteneffekte, Perspektiv- und Rotationseffekte
 - Überlagerung mit Vorrangkontrolle
 - Drahtgittermodell für Echtzeitanimation
 - Wiedergabe mit bis zu 60 Bildern pro Sekunde
 - deutsche Anleitung
- DM 498,—

Video Page

Professionelles Betitelungsprogramm für den Amiga

- NEU!**
- deutsche Version
 - verschiedene Schriftsätze und -größen
 - bequeme Editierfunktionen (WYSIWYG)
 - ausgereifte Kombination von Maus- und Funktionstastensteuerung
 - Funktionen für Rolltitel, Fade-Out, Cut-In etc.
 - Modul für Untertitel
 - integrierbare Module in Vorbereitung für: vordefinierte Untertexte, Zeit- und Kostenermittlung der Session, Grafik, Ergebnistabellen, einspielbare Hintergrundgeräusche (Gelächter, Applaus, Buhs, etc.)

Sculpt-Animate 3-D 68020/68881 UPDATE

ab sofort lieferbar DM 148,—

Sculpt-Animate 3-D Super-Fonts

- NEU!**
- Sorgfältig erarbeitete 3-dimensionale Schriftsammlungen für Sculpt-Animate 3-D.
- DM 148,—

HS & Y – Hurricane Board

- 68020/68881 Karte für A1000 und A2000
- 32-Bit RAM bis zu 4 MB
- z.B. für A2000/68020/68881 DM 2398,—
- A1000/68020/68881 DM 1898,—

HS & Y – Speedy Board für Amiga 1000 und 2000

- NEU!**
- 14,32 MHz 68000 Prozessor abschaltbar (!)
 - Socket für 68881 Co-Prozessor
 - Geschwindigkeitszuwachs um 170 %

Infos anfordern bei:

HS & Y

Heinrichson Schneider & Young
Herderstraße 94 · 5000 Köln 41
Tel.: 02 21/43 16 87
Fax: 02 21/43 65 69

und:
TV * Text, ProVideo CG1,
ProVideo Plus, diverse Fonts,
Animationsprogramme.

Der Amiga ist nicht nur über die Benutzeroberfläche der Workbench zu bedienen. Es gibt noch eine zweite Ebene, mit der sich viel mehr anfangen läßt, als dies mit der Workbench möglich ist: das CLI.

Der Amiga bietet dem Benutzer mit der Workbench eine komfortable Benutzeroberfläche. Programme können durch »anklicken« mit der Maus gestartet werden. Auch lassen sich einfache Operationen wie beispielsweise Löschen oder Umbenennen von Files leicht durchführen. Über die Workbench-Ebene können jedoch keine tiefergreifenden Manipulationen auf der Diskette vorgenommen werden. So läßt sich über die Workbench beispielsweise nicht feststellen, wie viele Dateien tatsächlich in welchen Schubladen (Subdirectories) auf der Diskette vorhanden sind. Oder wie wollen Sie ein Programm starten, das zwar auf der Diskette vorhanden ist, aber über kein Icon (Bildsymbol) verfügt? Doch es geht auch anders: Mit dem CLI (Command-Line-Interface, zu deutsch etwa: Kommando-Zeilen-Übergeber).

Doch was ist das CLI, wo ist es zu finden? Im Amiga-Handbuch stehen nicht viele Informationen zu diesem Programm, das sich (manchmal unsichtbar) auf jeder Workbench-Diskette befindet. Bevor wir aber das CLI erklären, machen wir das Command-Line Interface für den Anwender sichtbar!

Dazu fertigen Sie bitte zuerst eine Arbeitskopie der Workbench-Diskette an und legen dann diese Kopie in das interne Laufwerk. Starten Sie den Amiga erneut und rufen dann das Programm »Preferences« auf. Im linken unteren Viertel des Preferences-Menüs sehen Sie ein kleines Feld, in dem »CLI ON/OFF« steht. Bei einer nicht modifizierten Workbench steht das Feld (der »Schalter«) auf »CLI OFF«. Klicken Sie nun das

Wort »ON« an, speichern die Änderung mit SAVE wieder auf Diskette und aktivieren den Schreibschutz. Der Amiga ist anschließend neu zu starten. Klicken Sie das Symbol der Workbench-Diskette an und, nachdem das erste Fenster sichtbar ist, die Schublade mit dem Namen »System«. Durch die Änderung der Preferences findet sich nun ein neues Symbol in diesem Ordner, das den Namen »CLI« trägt. Bevor wir uns jetzt näher mit dem CLI befassen, soll zuvor geklärt werden, was dieses Programm eigentlich macht und wie es einzusetzen ist.

Neue Wege mit dem CLI

Von der Workbench aus kann der Benutzer nur mit Dateien umgehen, die über ein Icon verfügen. Andere Dateien können von der Workbench aus nicht beeinflußt oder aufgerufen werden. Hier greift das CLI ein: Es ermöglicht die direkte Beeinflussung aller auf einer Diskette vorhandenen Dateien. (Für alle Leser, die schon einmal mit Personal Computern gearbeitet haben: Das CLI ist in etwa vergleichbar mit COMMAND.COM unter MS-DOS.)

Die Befehle werden nicht (wie bei der Workbench) durch Anklicken eines Icons ausgeführt, sondern durch Kommandos, die der Anwender über die Tastatur eingibt. Versuchen wir doch gleich, anhand von praktischen Beispielen den Umgang und die Möglichkeiten, die uns das CLI in die Hand gibt, kennenzulernen. Rufen Sie dazu das Programm durch zweimaliges Anklicken des

CLI-Icons in der Schublade »System« auf. Nach kurzer Ladezeit erscheint auf dem Bildschirm ein neues Fenster. Dies ist das Arbeitsfenster des CLI. Hier erfolgen alle Ein- und Ausgaben, die das Command-Line-Interface betreffen. Es

oder »System.info«). Zu jedem dieser Icon-Files gehört auch das eigentliche Programm, welches durch das Icon aufgerufen wird (bei unserem Beispiel »Preferences« oder »System«).

Falls Sie über weitere Diskettenlaufwerke verfügen, können Sie sich über einen Zusatzparameter auch die Directories dieser Floppy-Stationen anzeigen lassen. Dazu ist hinter dem DIR-Befehl der Name des entsprechenden Laufwerks anzugeben. (Die vier möglichen Diskettenlaufwerke tragen die Bezeichnungen »df0« bis »df3«, die beiden anschließbaren Festplatten »dh0« und »dh1«.) Dabei ist nur zu beachten, daß der Laufwerksname durch ein Leerzeichen vom Befehl getrennt ist und daß am Ende ein Doppelpunkt steht. Das Directory des ersten externen Laufwerks würden Sie also mit folgendem Befehl erhalten:

DIR df1:

Wenn kein externes Laufwerk angeschlossen ist, erfolgt

CLI – Die Haut c

empfehlenswert, das Fenster zu vergrößern, so daß es fast den ganzen Bildschirm ausfüllt.

Anhand eines einfachen Befehls soll Ihnen nun der Unterschied zur Workbench erklärt werden. Lassen Sie dazu die Kopie der Workbench-Diskette im internen Laufwerk eingelegt und geben Sie das Kommando »DIR« ein, das mit der RETURN-Taste abzuschließen ist. Nach kurzer Zeit erscheinen viele einzelne Namen in zwei Spalten auf dem Bildschirm. Dieser erste Befehl, den wir kennenlernen, hat die Aufgabe, das Directory (Inhaltsverzeichnis) der eingelegten Diskette anzuzeigen. Sie werden feststellen, daß nach Eingabe des DIR-Befehls plötzlich viel mehr Dateinamen auf dem Bildschirm erscheinen, als dies im Symbolfenster auf der Workbench der Fall wäre. Dort sehen Sie nur Dateien, die über ein Icon verfügen. Im CLI-Fenster wären das die Programme, die die Endung ».info« aufweisen (auf der Workbench-Diskette beispielsweise die Files »Preferences.info«

eine entsprechende Fehlermeldung.

Da es manchmal vorkommen kann, daß mehr Dateinamen im Directory stehen, als auf dem Bildschirm angezeigt werden können, läßt sich die Ausgabe mit der Leertaste anhalten. Durch Druck auf <RETURN> oder die Backspace-Taste wird die Ausgabe fortgesetzt.

Zusätze zum DIR-Befehl

Das Directory einer Diskette besteht im allgemeinen aber nicht nur aus Dateinamen, sondern auch noch aus Subdirectories (Unterverzeichnissen). Ein Subdirectory kann man sich als Directory im Directory vorstellen. Im Hauptverzeichnis, in dem Sie sich gerade befinden, sehen Sie beispielsweise die Namen »c«, »s«, »devs« oder »Trashcan«. Hinter diesen Namen verborgen sich Subdirectories, was durch die angeschlossene Kennung »(dir)« ersichtlich ist.

Um den Inhalt eines Subdirectories anzusehen, ist ein weiterer Zusatzparameter beim DIR-Befehl nötig, nämlich der Name des Unterverzeichnisses. Möchten Sie sich etwa den Inhalt des Subdirectorys »c« ansehen, ist folgendes einzugeben:

```
DIR df0:c
```

Die Angabe des Laufwerknamens ist, da sowieso gerade diese Floppy-Station ausgewählt ist, nicht unbedingt nötig. Es ist aber empfehlenswert, immer das Laufwerk mitanzugeben, da man bei mehreren angeschlossenen Laufwerken leicht den Überblick verlieren kann.

Der DIR-Befehl bietet aber noch mehr Möglichkeiten, als nur das komplette Directory anzuzeigen. Dazu existiert der Zusatz »opt«, der in Zusammenhang mit ihm folgenden Buchstaben den DIR-Befehl verändert. Durch Angabe des Buchstabens »a« hinter dem Zusatz »opt« zeigt der Amiga sämtliche Dateien auf der Dis-

die Eingabe von »e« in dieses gewechselt werden. <RETURN> überspringt den Dateinamen. Wurde jedoch in das Subdirectory gewechselt und man möchte wieder in das Hauptdirectory zurück, ist »b« einzugeben. Das CLI wechselt auf die nächsthöhere Ebene zurück und macht dort weiter, wo der Einsprung in das Unterverzeichnis stattfand. »b« hinter einem Filenamen im Hauptdirectory beendet die Ausgabe.

Falls hinter dem Fragezeichen die Zeichenkette »del« (>delete« = löschen) eingetippt wird, löscht dieser Befehl die betreffende Datei. Auch Dateiverzeichnisse lassen sich auf diese Weise entfernen. Dabei ist nur zu beachten, daß zuerst aus dem betreffenden Subdirectory sämtliche Dateien entfernt sein müssen.

Der Buchstabe »t« hinter »opt« schließlich erlaubt die Ausgabe des Inhalts der betreffenden Datei. Dies hat allerdings nur Sinn, wenn es sich

Unterverzeichnis. Auf dem Bildschirm sollte jetzt »startup-sequence ?« sichtbar sein. Drücken Sie »t« und schließen die Eingabe mit <RETURN> ab. Blitzschnell erscheinen viele Zeilen, die wie ein Programm aussehen. Wenn Sie sich die einzelnen Zeilen ansehen, erkennen Sie einige Befehle (beispielsweise »ECHO«, »SETMAP« oder »LOADWB«), die Sie ebenfalls vom CLI aus benutzen können. Auf diese Befehle soll aber erst später eingegangen werden.

Viele Möglichkeiten

Die Eingabe von »q« beendet jederzeit die Ausführung des Befehls »DIR opt«. Durch die Tastenkombination <CTRL C> läßt sich ebenfalls ein Abbruch erzwingen.

Sie haben jetzt einen wichtigen Befehl kennengelernt, mit dem sich schon eine Menge anfangen läßt. Der DIR-Befehl zeigt uns aber nicht alles an. Es gibt noch weitere Informationen zu Dateien, die ein anderer Befehl viel detaillierter aufführt: Das Kommando »LIST«.

Der DIR-Befehl zeigte uns zwar alle Dateien auf der Diskette an, konnte uns jedoch keine Auskunft über deren Länge (Anzahl der Bytes) oder das Datum des letzten Updates (der letzten Speicherung) geben. (Der Amiga speichert zu jeder Datei auch die aktuelle Uhrzeit und das Datum. So läßt sich erkennen, wann die Datei das letztmal geändert wurde.) Um an diese »versteckten« Informationen zu gelangen, setzt man den Befehl »LIST« ein.

Versteckte Informationen

Geben Sie versuchsweise »LIST« gefolgt von <RETURN> ein. Zu jeder Datei des aktuellen Directories gibt das CLI jetzt zusätzliche Informationen aus. Zuerst erfährt man, wie viele Bytes in einer Datei gespeichert sind. Findet sich hier keine Angabe von Zahlen, sondern die Zeichenfolge »DIR«, ist dies der Name eines Dateiverzeichnisses. Die folgenden Buchstaben »rwd« geben an, welche Operationen mit dieser Datei durchführbar sind. Dabei gilt:

»r« — die Datei kann gelesen werden
»w« — das File läßt sich überschreiben

»e« — die Datei kann als Kommando aufgerufen werden
»d« — die Daten lassen sich löschen

Im Regelfall sind alle vier Schalter auf »an«, das bedeutet, alle vier Kennbuchstaben sind sichtbar.

Die letzten beiden Zeichenblöcke geben Auskunft darüber, wann die Datei zum letztenmal gespeichert wurde (Datum und Uhrzeit). Allerdings sollten die letzten Angaben nicht zu genau genommen werden, da für eine genaue Angabe eine im Amiga eingebaute Echtzeituhr erforderlich ist, die batteriegepuffert arbeitet. Da dies aber nicht zur seriemäßigen Ausstattung des Amiga gehört, müssen diese Daten nicht unbedingt mit den Zeiten übereinstimmen, während denen die Datei gespeichert wurde.

Der LIST-Befehl erlaubt wie der DIR-Befehl die Angabe zu-

KURSÜBERSICHT

Der CLI-Kurs erläutert Ihnen in mehreren Folgen den Umgang mit der Amiga-Benutzerschnittstelle CLI (Command-Line-Interface) der Workbench 1.2 bis Version 33.60. Am Ende haben Sie alle wichtigen Befehle kennengelernt und können die Kommandos einsetzen. Die Kursübersicht zeigt Ihnen, ab welchen Seiten welche CLI-Kommandos erklärt werden:

Seite 115 behandelt Befehle zum Anzeigen des Directories und Stellen der Uhr (dir, list, cd, date, endcli)

Seite 117 erklärt Ihnen, wie Sie Dateien kopieren, löschen, umbenennen und Disketten formatieren können (filenote, delete, relabel, rename, type, copy, diskcopy, info, format, install)

Seite 120 zeigt unter anderem auf, wie Texte und Startdateien erstellt und bearbeitet werden (makedir, ed, prompt, wait, loadwb, setclock, execute)

Seite 124 befaßt sich mit Befehlen, die sich zum Einbau in eine Batch-Datei anbieten (assign, echo, failat, if endif, then, else, stack, protect, bind-drivers)

Seite 126 setzt die Erklärung von Kommandos für eine Batch-Datei fort (run, edit, join, lab, quit, skip, addmem und break)

Seite 128 geht auf neue Spezialbefehle für Zusatzgeräte wie etwa Festplatten ein (search, newcli, sort, why, dimount, dpformat, mount, addbuffers, diskchange)

zweite es Amiga

kette an, einschließlich der Namen in allen Unterverzeichnissen. Geben Sie dazu als Beispiel folgendes ein:

```
DIR df0: opt a
```

Es werden sämtliche auf der Diskette enthaltene Dateinamen angezeigt. Beachten Sie bitte auch bei diesem Befehl die Schreibweise: Die einzelnen Leerzeichen dürfen nicht vergessen werden!

Gibt man anstelle des Buchstabens »a« ein »d« ein, erfolgt die Ausgabe aller Dateiverzeichnisse, also aller Namen, hinter denen die Kennung »(dir)« steht.

Ein besonderer Befehl ist der folgende:

```
DIR df0: opt i
```

Tippt man diese Zeichenfolge ein, gibt der Amiga jeweils nur einen Dateinamen aus, auf den ein Fragezeichen folgt. Bei Druck auf die RETURN-Taste übergeht das CLI die entsprechende Datei und gibt den Namen der folgenden Datei aus. Stellt der angezeigte Name ein Subdirectory dar, kann durch

dabei um eine Textdatei handelt. Ein Beispiel:

Sie möchten den Inhalt einer bestimmten Datei im Klartext auf dem Bildschirm ausgeben.

Nehmen wir dazu die Datei »Startup-Sequence«, die sich im Subdirectory »s« der Workbench-Diskette befindet. Diese Datei ist das erste File, das nach dem Einlegen der Workbenchdiskette geladen und gestartet wird (die Datei ist vergleichbar mit einer Autostart-Datei (AUTOEXEC.BAT) unter MS-DOS). In dieser Datei stehen alle Befehle und Parameter, die der Amiga abarbeitet, bis nach einiger Wartezeit endlich die Workbench erscheint. Diese Startdatei besteht aus ASCII-Zeichen, ist also eine Textdatei.

Vergewissern Sie sich, daß Ihre Workbench-Kopie eingelegt ist und geben Sie folgendes ein:

```
DIR df0: opt i
```

Drücken Sie nun so lange <RETURN>, bis »s(dir) ?« erscheint. Mit der Eingabe von »e« gelangen Sie nun in dieses

sätzlicher Parameter. Geben Sie versuchsweise folgendes ein:

```
LIST df0: p s#?
```

Das CLI zeigt Ihnen nur Dateien an, die als ersten Buchstaben ein »s« besitzen. Mit dem Parameter »p« läßt sich also eine Selektion der Dateien erzielen. Es werden nur Dateien angezeigt, die in die Maske nach dem Parameter »p« passen. Die Zeichen »#?« ersetzen den Rest des Namens, ein »?« steht für einen einzelnen Buchstaben. Einige Beispiele für zulässige Muster:

```
LIST df0: p sy#?
LIST df0: p sy??em
LIST df0: p #?.info
```

Das erste Beispiel würde alle Dateien anzeigen, deren Anfangsbuchstaben »sy« sind, also etwa »system« und »system.info«. Beim zweiten Befehl gibt das CLI nur Dateien auf dem Bildschirm aus, deren erste und letzten Buchstaben »sy« und »em« wären. Die beiden mittleren Zeichen unterliegen keinen Beschränkungen, könnten also Buchstaben, Zahlen oder Zeichen sein.

Der dritte Befehl zeigt alle Dateien an, deren letzte fünf Zeichen ».info« lauten (beispielsweise »system.info«, »trashcan.info« oder »preferences.info«). Diese Maskierung läßt sich auch auf den DIR-Befehl anwenden. Zwei Beispiele:

```
DIR df0: #?.info
DIR df0: ex#?
```

Ein weiterer Parameter beim LIST-Befehl ist der Buchstabe »s«. Es werden nur Dateinamen ausgegeben, deren Namen den hinter »s« angegebenen Text enthalten.

```
LIST df0: s refer
```

würde bei eingelegter Workbench-Diskette die Dateinamen »preferences« und »preferences.info« auf dem Bildschirm ausgegeben.

Sollen nur Dateien angezeigt werden, die man nach einem bestimmten Datum gespeichert hat, ist folgende Schreibweise anzuwenden:

```
LIST df0: since Tag-
Monat-Jahr
```

Dabei müssen die jeweiligen Daten in der Form eingegeben werden, wie sie auch bei der LIST-Ausgabe auf dem Bildschirm erscheinen. Ein Beispiel für die korrekte Schreibweise:

```
LIST df0: since
11-jan-87
```

zeigt alle Dateien an, die nach dem 11. Januar 1987 gespeichert wurden. Das Gegenteil des Parameters »since« ist die Angabe von »upto«. In diesem Fall zeigt das CLI nur Dateien an, die bis zu einem bestimmten Datum angelegt wurden.

```
LIST df0: upto 11-jan-87
```

gibt alle Dateien auf dem Bildschirm aus, die vor dem 11. Januar 1987 gespeichert wurden.

Der LIST-Befehl kennt noch drei weitere Optionen, die man jederzeit angeben kann:

Der Zusatz »keys« zeigt zusätzlich noch die Nummer des Datei-Startblocks (File-Header) der jeweiligen Datei auf der Diskette an. Dies ist aber eine Option, die nur dann sinn-

parameter ein komplexer Befehl. Es ist also auch folgende verwirrende Kombination möglich:

```
LIST df0: p e#? s ref
keys since 11-mar-1986
upto 17-jan-87 nodates
quiet
```

Sie sehen, der LIST-Befehl erfordert etwas Training, bis man ihn beherrscht.

Die »Ebenen« der Diskette

Wie Sie am Anfang dieses Artikels gelesen haben, ist der Inhalt einer Diskette meist auf verschiedene Subdirectories verteilt. Sie können inzwischen

begeben, um dort zu arbeiten. Das CLI stellt für diese Aufgabe den Befehl »CD« (change directory = wechsele das Verzeichnis) zur Verfügung. Gibt man dem Befehl »CD« ohne Parameter ein, so zeigt das CLI den Namen des aktuellen Dateiverzeichnisses an. Probieren wir dies gleich aus. Geben Sie bitte ein:

```
CD
```

In Ihrem CLI-Fenster müßte nun der Name der Diskette, der auch unter dem Diskettensymbol auf der Workbench zu sehen ist, erscheinen (Workbench, Workbench 1.x oder ähnlich). Wechseln Sie nun versuchsweise das aktuelle Directory. Nehmen wir an, Sie

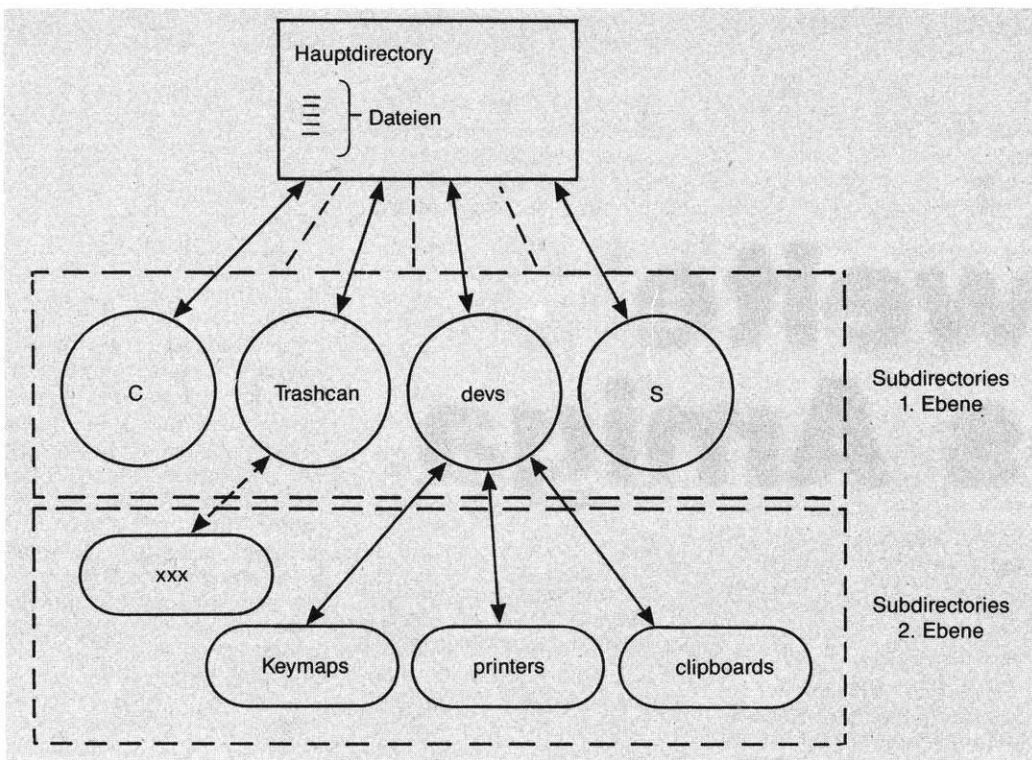


Bild 1. An diesem Beispiel läßt sich die Struktur der Subdirectories besser verstehen

voll einzusetzen ist, wenn man über einen Diskettenmonitor und genügend Kenntnis über den Dateiaufbau auf der Diskette verfügt.

Der zweite Zusatz »nodates« unterdrückt die Ausgabe des Datums und der Uhrzeit hinter jeder Datei. Es werden also nur die Länge der Datei und die Stellung der Schalter »rwd« ausgegeben.

Der Parameter »quick« schließlich unterbindet die Ausgabe aller Zusätze. Das CLI gibt nur die Dateinamen aus, entspricht mit dem Zusatz »quick« also weitgehend dem DIR-Befehl.

Der LIST-Befehl ist durch die Vielzahl seiner möglichen Pa-

das Directory von Haupt- und Unterverzeichnissen lesen. Aber können Sie sich auch die Struktur eines Aufbaus mit Haupt- und Dateiverzeichnissen vorstellen? Deshalb zuerst eine kleine Erläuterung (betrachten Sie dazu bitte auch Bild 1):

Dieses Bild stellt einen kleinen Ausschnitt der Workbench-Diskette schematisch dar. Im Bild oben sehen Sie das Hauptverzeichnis, von dem aus wir uns bis jetzt die Unterverzeichnisse mit dem DIR- und LIST-Befehl angesehen haben. Wir hielten uns aber immer im Hauptverzeichnis auf. Zuweilen ist es jedoch nötig, daß wir uns in ein Subdirectory

möchten in das Subdirectory »devs (dir)« überwechseln. Der Befehl dazu lautet:

```
CD devs
```

Überprüfen Sie mit
CD

ob Sie sich auch im richtigen Subdirectory befinden: In Ihrem CLI-Fenster müßte nun der Name der Diskette und, durch einen Doppelpunkt abgetrennt, der Name des Dateiverzeichnisses erscheinen, in diesem Fall »xxx:devs«. Wenn Sie Bild 1 betrachten, befinden wir uns jetzt im Kreis in der mittleren Reihe, in welchem »devs« steht. Versuchen wir den Befehl »DIR«. Wir stellen fest, daß nun andere Dateien erschei-

nen als im Hauptdirectory. Hier finden sich drei Unterverzeichnisse mit den Namen »keymaps«, »printers« und »clipboards« (die untere Reihe in Bild 1). Mit dem Kommando

```
CD keymaps
```

würden Sie jetzt das Subdirectory »keymaps« zum aktuellen Verzeichnis erklären und befänden sich in der unteren Reihe von Bild 1. Soweit wollen wir aber nicht gehen. Wie gelangt man aber jetzt wieder nach »oben« ins Hauptdirectory? Dazu dient der Befehl

```
CD /
```

der Sie immer in das nächsthöhere (!) Directory zurückbringt. Wenn Sie sich beispielsweise in einem Subdirectory aufhielten, das vielleicht in der vierten Ebene angelegt ist, kämen Sie mit dem Befehl »CD /« nur zurück in die dritte Ebene. Um in das Hauptdirectory zu gelangen, wäre folgender Befehl nötig:

```
CD ////
```

Doch das geht auch einfacher: Mit

```
CD df0:
```

erreichen Sie immer das Hauptdirectory des internen Laufwerks, egal, in welcher Tiefe Sie sich gerade aufhalten. (Sollten Sie auf einem anderen Laufwerk als »df0« arbeiten, ist natürlich immer das aktuelle Laufwerk anzugeben. »CD df0:« bringt Sie nicht ins Hauptdirectory von »df1«, wenn gerade auf »df1« gearbeitet wird.)

Selbstverständlich kann auch zwischen einzelnen Directory-Zweigen gewechselt werden. Dabei muß allerdings immer der Umweg über das Hauptdirectory genommen werden. Wenn Sie sich gerade im Dateiverzeichnis »devs« aufhalten, gelangen Sie nicht mit »CD trashcan« in dieses Nebendirectory. Die korrekte Schreibweise wäre in diesem Fall

```
CD /trashcan
```

oder

```
CD df0:trashcan
```

Befindet man sich sogar eine Ebene tiefer (etwa im Subdirectory »printers«, Bild 1), wäre folgender Befehl richtig:

```
CD //trashcan
```

oder

```
CD df0:trashcan
```

Nehmen wir an, im Dateiverzeichnis »trashcan« wäre ein weiteres Subdirectory mit dem Namen »XXX« und wir wollen

vom Unterverzeichnis »printers« nach »XXX« überwechseln. Dies ist etwas schwerer nachvollziehbar:

```
CD //trashcan/XXX
```

oder, etwas einfacher

```
CD df0:trashcan/XXX
```

Der CD-Befehl dient aber nicht nur zum Wechseln der Directory-Ebenen, sondern auch dem Anwählen des gewünschten Laufwerks. Nehmen wir an, das interne Laufwerk »df0« ist derzeit das aktuelle Laufwerk. Sie möchten aber mit einer Diskette arbeiten, die sich in der externen Floppy-Station befindet, also in »df1«. Das Überwechseln geschieht mit folgendem Befehl:

```
CD df1:
```

Schon arbeiten Sie mit der Diskette in »df1«. Natürlich ist es auch möglich, direkt in bestimmte Verzeichnisse zu springen:

```
CD df1:XXX
```

wobei XXX der Verzeichnisname oder eine Kombination aus Verzeichniswegen sein darf. Ein Beispiel: Das aktuelle Laufwerk ist »df1«. In »df0« befindet sich die Workbench-Diskette, von der in das gedachte Verzeichnis »XXX« im Verzeichnis »devs/printers« gewechselt werden soll. Der Aufruf sieht dann folgendermaßen aus:

```
CD df0:devs/printers/XXX
```

Schon befindet man sich im Subdirectory »XXX« im Verzeichnis »devs/printers«.

Die Zeit läuft mit . . .

Wie bereits weiter oben erwähnt, besitzt der Amiga eine Uhr, bei der auch das Datum mit gespeichert wird. Da der Amiga normalerweise nicht über eine (nötige!) batteriegepufferte Echtzeituhr verfügt, holt sich das Betriebssystem beim Laden der Workbench das Datum der letzten Modifikation von der Diskette und verwendet diese Daten als aktuelle Zeit. Diese Uhr kann aber auch von Ihnen gestellt werden. Dazu findet sich im CLI der Befehl »DATE«. Ohne Parameter einzugeben erfährt man die »aktuelle« Zeit und das Datum. Sollte es nötig sein, diese Daten zu aktualisieren, muß der DATE-Befehl folgendermaßen eingesetzt werden:

```
DATE 15-apr-88
```

um das Datum zu stellen. Die

Uhrzeit läßt sich genauso leicht einstellen:

```
DATE 01:42:13
```

Diese Schreibweise ist auch kombinierbar. Mögliche Formen:

```
DATE 15-apr-88 12:32:00
```

```
DATE 14:12 13-mar-99
```

Die Reihenfolge spielt keine Rolle. Das CLI unterscheidet Zeit und Datum anhand des Doppelpunktes und Bindestriches.

Beim nächsten Schreibzugriff auf Diskette wird die nun aktualisierte Zeit gespeichert und ist (besitzt man keine gepufferte Uhr) die Basis beim nächsten Neustart.

Es gibt aber noch einen kleinen Trick beim Setzen des Datums: Sie können das aktuelle Datum relativ leicht um einen Wochentag erhöhen oder erniedrigen. Wird beispielsweise regelmäßig jeden Tag mit einer bestimmten Diskette (idealerweise der Workbench-Kopie) gearbeitet, erhöht folgendes Kommando automatisch das aktuelle Datum um einen Tag:

```
DATE tomorrow
```

Schon wurde aus einem »11-06-88« das Datum »12-06-88«. Ebenso leicht kann ein Tag abgezogen werden:

```
DATE yesterday
```

Bindet man diesen Befehl in die »Startup-Sequence« ein (wie das gemacht wird, erfahren Sie in einem späteren Abschnitt dieses Kurses), aktualisiert der Amiga bei jedem Start das Datum.

Damit wollen wir den ersten Abschnitt des CLI-Kurses abschließen. Um die Arbeit mit dem CLI zu beenden, geben Sie abschließend den Befehl »ENDCLI« ein. Danach befinden Sie sich wieder auf der Workbench. Sie haben bereits einige Befehle kennengelernt, die Ihnen einen Eindruck von der Leistungsfähigkeit des CLI vermitteln. Sollten Sie noch Schwierigkeiten bei der Schreibweise haben, hilft die Eingabe eines durch ein Leerzeichen abgetrennten Fragezeichens hinter jedem Befehl weiter. Das CLI zeigt danach die möglichen Parameter und deren Schreibweise an.

Nachdem Befehle behandelt wurden, die dem Betrachten des Directories (Inhaltsverzeichnis) der Diskette dienen, setzen wir die Reihe mit Befehlen fort, die gezielt Dateien beeinflussen können. Da wir natürlich keine wichtigen Dateien beschädigen wollen, er-

stellen wir eine kleine Testdatei. Geben Sie dazu bitte folgende Befehle ein (die Erklärung folgt später):

```
COPY * TO df0:Test
```

Der Cursor springt nun in eine neue Zeile, liefert aber kein Prompt (1>) mehr. Drücken Sie jetzt zwei- bis dreimal <RETURN> und anschließend <CTRL \> (dies ist der Schrägstrich, der auf der Tastatur in der obersten Reihe rechts vor dem Linkspfeil steht).

Die Floppystation läuft kurz an und erzeugt eine leere Datei mit dem Namen »Test«. Im folgenden werden wir uns bei Diskettenoperationen auf diese Datei beschränken, um keine falschen Dateien zu beschädigen!

Das DOS (Diskettenbetriebssystem) des Amiga erlaubt es, an jede Datei auf der Diskette einen Kommentar von 80 Zeichen anzufügen. Dieser Kommentar kann dazu dienen, bestimmte Informationen zu einer Datei festzuhalten. Beispielsweise, um welche Version des Programms es sich dabei handelt oder welche Dateien noch zu dem Programm gehören. Der Kommentar läßt sich mit dem LIST-Befehl betrachten. Dabei wird in einer zusätzlichen Zeile der Kommentar, falls vorhanden, auf dem Bildschirm ausgegeben. Der Text kann auch von der Workbench-Ebene aus betrachtet werden, und zwar mit dem Kommando »INFO« aus dem ersten Pull-Down-Menü.

Der Kommentar verfügt über einige Eigenschaften, die möglicherweise nicht sofort verständlich sind. So wird er beispielsweise beim Kopieren einer Datei nicht mit übertragen. Die kopierte Datei, die ein identisches Abbild der ersten Datei sein sollte, enthält also keinen Kommentar!

Ein weiterer Punkt ist folgender: Wird eine Datei mit einem neuen Inhalt überschrieben, bleibt der Kommentar der alten Datei erhalten (auch wenn dieser nun nicht mehr zutrifft).

Doch nun zum eigentlichen Befehl, der das Anlegen eines Kommentars ermöglicht: der Befehl »FILENOTE«. Mit diesem Kommando kann ein Text zu einer Datei eingegeben werden. Der Text kann aber mit diesem Befehl nicht angesehen werden. Dies ist nur mit dem LIST-Befehl machbar.

Versuchen wir gleich einmal, eine Datei mit einem Kommentar auszustatten. Vergewissern Sie sich zuerst, daß die einge-

legte Diskette nicht schreibgeschützt ist.

Als Beispiel wollen wir der Datei »Test« einen Kommentar anfügen. Der Befehl dazu lautet folgendermaßen:

```
FILENOTE Test "Text"
```

Dabei sollte der Kommentar immer in Anführungszeichen gesetzt werden, damit auch Leerzeichen oder Sonderzeichen korrekt übernommen werden. Kontrollieren wir gleich, ob der Befehl korrekt ausgeführt wurde. Geben Sie nun ein:

```
LIST Test
```

Auf dem Bildschirm sollten jetzt die vom LIST-Befehl her bekannten Ausgaben erscheinen. Zusätzlich findet sich in einer neuen Zeile, durch einen Doppelpunkt eingeleitet, der Kommentar, den wir eben angefügt haben.

Der Text kann auch wieder gelöscht werden. Dazu ist nur anstelle des Kommentars ein leerer String zu übergeben. Nach Eingabe von

```
FILENOTE Test ""
```

ist die Datei wieder vom Kommentar befreit. Prüfen Sie dies mit

```
LIST Test
```

nach. Die Kommentarzeile, die vorher durch einen Doppelpunkt eingeleitet wurde, ist nun verschwunden.

Schnelle Auskunft

Ein weiterer, wichtiger Befehl des CLI ist das Kommando »INFO«. Der Befehl gibt Auskunft über die angeschlossenen Disketten- und Festplattenlaufwerke. Nach Eingabe des Befehls

```
INFO
```

wird ein Statusreport über alle angeschlossenen Laufwerke und die RAM-Floppy ausgegeben (Bild 2).

Dieser Befehl dient hauptsächlich dazu, einen schnellen Überblick über die momentan verfügbaren Disketten- und Festplattenspeicher sowie deren Kapazitäten zu geben. Er eignet sich auch zum Einbinden in die »Startup-Sequenz«. In diese Datei eingebaut, erhält der Anwender gleich beim Start des Amiga die Information über die angeschlossenen Laufwerke.

Die nächsten Befehle, mit denen wir uns befassen, nehmen direkten Einfluß auf Dateien oder die gesamte Diskette.

Diese Befehle bewirken in den meisten Fällen eine direkte Veränderung einzelner Bytes, Dateien oder der kompletten Diskette. Sie sind deshalb mit Vorsicht zu genießen, da durch falsche Handhabung leicht eine Datei oder der Disketteninhalt zerstört werden kann.

Der erste Befehl dieser Art, der besprochen wird, dient dem Neu- beziehungsweise Umbenennen einer Datei oder eines Dateiverzeichnisses (Subdirectory). Es ist dies der Befehl »RENAME«.

Die Schreibweise des Befehls sieht folgendermaßen aus:

```
RENAME alter Name TO  
neuer Name
```

Benennen wir doch gleich einmal eine Datei um. Nehmen wir als Beispiel das Programm »Test«, das den neuen Namen »Abfall« erhalten soll. Der Befehl dazu sieht wie folgt aus:

```
RENAME Test TO Abfall
```

Eine Kontrolle mit dem DIR-Befehl ergibt, daß die ehemalige Datei »Test« nun den Namen

ry anzeigen lassen, werden Sie feststellen, daß die Datei »Test« nicht mehr zu sehen ist. Sie ist aber nur aus dem Hauptdirectory verschwunden. »Test« befindet sich jetzt mit dem neuen Namen »Test1« im Subdirectory »Trashcan«, würde also unter gleichzeitiger Änderung des Namens in ein anderes Directory verlagert.

Auch diesen Vorgang wollen wir wieder rückgängig machen. Die Kommandofolge

```
CD df0:
```

```
RENAME Trashcan/Test1 TO  
Test
```

bringt die Datei wieder ins Hauptdirectory zurück und gibt ihr den alten Namen, wie eine Kontrolle mit »DIR« beweist.

Ein weiteres Kommando zum Ändern eines Namens ist der Befehl »RELABEL«. Allerdings können damit keine Dateien oder Subdirectories umbenannt werden. Dieser Befehl wirkt nur auf den Namen einer Diskette. Möchten Sie also den Namen Ihrer Diskette (der auch unter dem Diskettensymbol in der Workbench er-

legte Diskette nun den Namen »Meine Workbench«. Dies können Sie leicht überprüfen, wenn Sie den Befehl »INFO« eingeben.

Löschen und Kopieren

Von Zeit zu Zeit kommt es vor, daß man einige Dateien oder Unterverzeichnisse auf seiner Diskette löschen möchte, um beispielsweise Platz für neue Programme zu schaffen. Dazu dient der Befehl »DELETE«, der aber sehr mit Vorsicht zu genießen ist, da die betreffenden Dateien endgültig verloren sind (es existieren zwar schon einige Möglichkeiten, die Dateien wieder zu retten, doch ist dafür ein genaues Wissen über den Diskettenaufbau sowie der Besitz eines Diskettenmonitors nötig). Die folgenden Beispiele sollten also nicht ausprobiert werden, wenn Sie nicht sicher sind, daß die betreffenden Dateien noch auf anderen Disketten existieren. Da wir aber mit einer Kopie (!) der Workbench-Diskette arbeiten, sollte uns das nicht weiter stören, da ja noch ein Original vorhanden ist.

Schreiten wir auch hier gleich wieder zur Praxis und versuchen, eine Datei zu löschen. Hier bietet sich das File »Test« an, welches keine »lebensnotwendigen« Daten enthält. Gelöscht wird die Datei mit folgendem Befehl:

```
DELETE Test
```

oder, falls Sie sich in einem Unterverzeichnis oder einer anderen Diskette befinden, mit

```
DELETE df0: Test
```

Vergessen Sie nicht die Leerzeichen zwischen den einzelnen Parametern.

Es können auch mehrere Dateien (maximal 10) mit einem einzigen Befehl gelöscht werden. Dazu sind nur die einzelnen Namen durch ein Leerzeichen getrennt in die gleiche Befehlszeile zu schreiben. Das nächste Beispiel würde fünf verschiedene Dateien löschen:

```
DELETE df0: Test1 Test2  
Test3 Test4 Test5
```

Selbstverständlich können auch Pfadnamen oder Masken verwendet werden:

```
DELETE df0: P#?  
DELETE df0: Devs/  
Keymaps/USA2  
DELETE df0: System/  
Er#?t
```

Unit	Size	Used	Free	Full	Errs	Status	Name
JH1:	340K	7	671	1%	0	Read/Write	JH1
JH0:	20M	14349	26449	35%	0	Read/Write	JH0
DF2:	880K	1670	88	94%	0	Read/Write	ExtrasD
DF1:	880K	1744	14	99%	0	Read Only	WB A2000-1.6.87
DF0:	880K	1742	16	99%	0	Read/Write	Workdisk DM
RAM:	1K	1	0	100%	0	Read/Write	

Volumes available:
ExtrasD [Mounted]
WB A2000-1.6.87 [Mounted]
Workdisk DM [Mounted]
RAM Disk [Mounted]
JH0 [Mounted]
JH1 [Mounted]

Bild 2. Ein mit »INFO« erstellter Report

»Abfall« trägt. Da wir aber den ursprünglichen Namen behalten wollen, benennen wir das Unterverzeichnis wieder um:

```
RENAME Abfall TO Test
```

(Die Angabe von »TO« wäre nicht unbedingt nötig, erleichtert aber die Lesbarkeit des Befehls.) Sollen in einem der beiden Namen Sonderzeichen oder Leerzeichen verwendet werden, sind die Namen in Anführungszeichen zu setzen. Der RENAME-Befehl kann aber noch mehr, als nur eine Datei umbenennen! Geben Sie probierhalber diesen Befehl ein:

```
RENAME Test TO Trashcan/  
Test1
```

Der Befehl mag Ihnen vielleicht etwas merkwürdig erscheinen, doch in dieser Weise wirkt der RENAME-Befehl wie eine Move-Anweisung. Wenn Sie sich mit »DIR« das Directo-

scheint) ändern, ist folgende Form zu verwenden (bis zu 30 Zeichen sind erlaubt):

```
RELABEL Laufwerk: "neuer  
Name"
```

Die Laufwerksangabe ist nicht zwingend erforderlich, empfiehlt sich aber der Übersichtlichkeit halber. Auf die Praxis angewendet würde die Befehlsfolge für Ihre im internen Laufwerk eingelegte Workbench-Kopie so aussehen:

```
RELABEL df0: "Meine  
Workbench"
```

Zu beachten ist hierbei, daß die Leerzeichen nicht vergessen werden dürfen. Des weiteren muß ein Ausdruck, der Leerzeichen oder Sonderzeichen enthält, in Anführungszeichen gestellt werden. Wenn Sie das obige Beispiel eingegeben haben, trägt die einge-

Im ersten Beispiel würden alle Dateien gelöscht, die im Hauptverzeichnis von Laufwerk 0 stehen und mit »P« beginnen. Der zweite Fall löscht das Programm »USA2«, das sich im Subdirectory »Devs/Keymaps« befindet. Der dritte Befehl schließlich löscht alle Dateien, die sich im Unterverzeichnis »System« befinden, mit »Er« beginnen und als letzten Buchstaben ein »t« besitzen.

Der DELETE-Befehl gibt bei mehr als einer gelöschten Datei eine Meldung aus. Wer diese Meldung unterbinden möchte, kann am Schluß der Befehlszeile »QUIET« oder abgekürzt »Q« eingeben. Dieser Zusatz unterdrückt die Ausgabe der Meldung. Ein Beispiel:

```
DELETE Test1 Test2 Test3
QUIET
```

oder abgekürzt

```
DELETE Test1 Test2 Test3
Q
```

Auch beim DELETE-Befehl kann mit Jokern und Maskierung gearbeitet werden (#?, ??).

»DELETE« läßt außer »Quiet« noch den Parameter »All« zu, der, am Schluß der Befehlszeile angehängt, folgende Wirkung hat: Wenn Sie beispielsweise die Kommandozeile

```
DELETE df0: De#? ALL
```

eingeben würden, so wären anschließend alle Dateien und auch Subdirectories verschwunden, deren erste beiden Buchstaben mit »De« begonnen hätten. Man kann die Maskierung auch noch weiter treiben. Der Befehl

```
DELETE #?/#?E#?
```

würde sämtliche Dateien auf der gesamten (!) Diskette löschen, die irgendwo in ihrem Namen ein »e« enthalten (auch alle Subdirectories). Mit diesem Befehl sollte also sehr vorsichtig umgegangen werden, da man leicht Dateien unrettbar löschen kann, die noch gebraucht werden. Außerdem erfolgt vor der Befehlsausführung keine Sicherheitsabfrage, ob die Dateien wirklich entfernt werden sollen.

Eines ist beim Delete-Befehl noch anzumerken: Es kann kein Subdirectory gelöscht werden, wenn in ihm noch Dateien enthalten sind. Bevor das Unterverzeichnis gelöscht werden kann, müssen also zuerst alle (!) Dateien aus diesem entfernt werden.

Des weiteren können sich Dateien auch Lösversuchen

widersetzen. Jede Datei enthält vier Flags (Schalter), die bestimmte Manipulationen mit der Datei zulassen oder verbieten. Den Zustand dieser vier Flags (»r-w-e-d«) können Sie sich auch mit dem LIST-Befehl ansehen. Nur wenn das »d«-Flag gesetzt ist, läßt sich die Datei entfernen. Ändern lassen sich diese Flags mit dem »PROTECT«-Befehl.

Vorbereiten einer Diskette

Um Daten auf eine Diskette speichern zu können, ist nötig, daß die betreffende Diskette formatiert ist. Formatieren bedeutet in diesem Fall, daß die Magnetscheiben in konzentrische Kreise und die Kreise in einzelne Sektoren eingeteilt werden. Es soll hier nicht näher auf die Aufteilung einer Diskette eingegangen werden. Wichtig ist hier nur zu wissen, daß Disketten auf jeden Fall formatiert sein müssen, bevor sie genutzt, das heißt Daten auf ihnen gespeichert werden können. Beim Amiga dient dazu der Befehl »FORMAT«. Dieser Befehl ermöglicht es allerdings nur, Disketten zu formatieren. Festplatten oder 5 1/4-Zoll-Laufwerke können mit diesem Befehl nicht vorbereitet werden. Der Befehl benötigt folgende Schreibweise:

```
FORMAT DRIVE Laufwerk:
NAME Name NOICONS
```

Unterstützende Literatur:
Das Amiga-Handbuch, Markus Breuer,
Markt & Technik-Verlag, 49 Mark

Es ist auf korrekte Schreibweise zu achten. Leerzeichen dürfen nicht vergessen werden! Außerdem sind die Formalparameter »DRIVE« und »NAME« immer mit anzugeben. Der Parameter »NOICONS« kann, muß aber nicht eingetippt werden. Läßt man ihn weg, installiert das DOS auf der Diskette das Subdirectory »Trashcan« und das zugehörige Icon (Symbol). Gibt man dagegen »NOICONS« an, werden die »Trashcan«-Dateien nicht erzeugt. Um beispielsweise eine Diskette im externen Laufwerk (df1) zu formatieren, müßte folgende Befehlszeile eingegeben werden (versuchen Sie nicht, diesen Befehl nachzuvollziehen, wenn Sie sich nicht sicher sind, ob Sie die Daten in dem betreffenden Laufwerk noch benötigen. Der Amiga verlangt zwar zuerst eine Sicherheitsbestätigung, doch wie leicht kann man sich vertippen... Nach der

Formatierung sind diese Daten unrettbar verloren):

```
FORMAT DRIVE df1: NAME
"Leere Diskette"
```

Auch hier muß der letzte Ausdruck wieder in Anführungszeichen gesetzt werden, wenn Leerzeichen oder Sonderzeichen enthalten sind (zulässig sind insgesamt 30 Zeichen). Wenn die Diskette nicht schreibgeschützt war, besitzt sie nun den Diskettenamen »Leere Diskette« und kann beschrieben werden.

Eine so vorbereitete Diskette kann in diesem Stadium allerdings nur zur reinen Datenspeicherung verwendet werden. Wie Sie sicher wissen, verlangt der Amiga nach dem Einschalten beziehungsweise Laden des Kickstarts eine Workbench-Diskette. Der Amiga würde die eben formatierte Diskette aber nicht als Startdiskette anerkennen, sondern solange nach einer Workbench-Diskette verlangen, bis eine mit einem Startsektor versehene Diskette eingelegt wird. Dies erledigt der nächste Befehl: »INSTALL« (nicht bei Festplatten oder 5 1/4-Zoll-Laufwerken).

Um beispielsweise eine Diskette in Laufwerk 1 (df1) startfähig zu machen, reicht die Eingabe der nächsten Zeile:

```
INSTALL df1:
```

Das Laufwerk läuft kurz an und installiert den Startsektor. Nach diesem Vorgang akzeptiert der Amiga diese Diskette als bootfähige (Workbench-) Diskette.

Kopieren von Dateien

Jeder Anwender muß von Zeit zu Zeit Dateien von einem Gerät auf ein anderes kopieren. Sei es, um seinen Druckertreiber oder ein anderes Programm auf eine andere Diskette zu übertragen oder, im Fall des Amiga, eine Datei auszudrucken.

Zum Kopieren einer einzelnen Datei dient der Befehl »COPY«. Dieses Kommando dupliziert den Inhalt einer angegebenen Datei auf ein anderes Gerät. Diese können Diskettenlaufwerke, Festplatten, die RAM-Disk, Drucker oder im einfachsten Fall der Bildschirm sein.

Besprechen wir also der Reihe nach die verschiedenen Möglichkeiten, wie man mit dem COPY-Befehl Dateien vielfältigen kann:

Die erste Art bezieht sich natürlich auf diejenigen Geräte, wo dieser Befehl bevorzugt eingesetzt wird: den Disketten- oder Festplatten-Laufwerken und die RAM-Disk.

Die Schreibweise dafür lautet allgemein:

```
COPY Quell-Datei TO
Ziel-Datei/-Gerät
```

Erklären wir dies gleich an einem Beispiel: Sie möchten die Datei »Clock« von Ihrer Workbench-Diskette im internen Laufwerk (df0) auf eine zweite Diskette im ersten angeschlossenen Laufwerk (df1) kopieren. Die Datei befindet sich im Hauptdirectory und soll auch auf der zweiten Diskette im Hauptdirectory stehen. Der Befehl dazu würde lauten:

```
COPY Clock TO df1:
```

oder, noch exakter

```
COPY df0:Clock TO df1:
```

(Das »TO« ist nicht zwingend erforderlich, erhöht aber die Lesbarkeit.) Wenn wir davon ausgehen, daß wir uns von Anfang an im Hauptdirectory der im internen Laufwerk (df0) eingelegten Diskette befinden haben, sind beide Befehle richtig. Ist aber eine andere Diskettenstation das aktuelle Laufwerk oder befindet man sich in einem Subdirectory, wäre nur der zweite Befehl richtig.

Im nächsten Fall soll die gleiche Datei vom selben Laufwerk auf df1 kopiert werden. Die Kopie erhält aber einen anderen Namen. Die Schreibweise:

```
COPY df0:Clock TO df1:
"Neuer Name"
```

Sie können also gleich bei der Befehlseingabe den neuen Namen der Zieldatei festlegen und müssen sie nicht erst später umbenennen. Dies funktioniert auch mit Pfadangaben, um in Unterverzeichnisse zu gelangen. Beispiel: Die Datei »List«, die sich im »C«-Directory von df0 befindet, soll auf df1 in das Unterverzeichnis »XXX« übertragen werden. Gleichzeitig erhält die Datei den neuen Namen »Test«. Die Kommandozeile

```
COPY df0:C/List TO df1:
Xxx/Test
```

erledigt diese Aufgabe. Natürlich kann der Copy-Befehl auch maskiert werden (maskieren = Platzhalter einsetzen).

```
COPY df0:C/A#? TO df1:xx
```

Dieser mit Jokern arbeitende Befehl überträgt beispielsweise alle Dateien, die sich im

C-Directory von Laufwerk #0 befinden und mit »A« beginnen, in den Ordner »XX« von Laufwerk #1.

Auch beim COPY-Befehl können Sie, wie beim DELETE-Befehl, zwei Parameter am Schluß der Befehlszeile eingeben. Der erste ist »QUIET« oder »Q«, der die schon vom DELETE-Kommando her bekannte Funktion hat, nämlich die Unterdrückung von Textausgaben auf dem Bildschirm. Beispiel:

```
COPY df0:C TO df1: QUIET
```

Das C-Directory (df0) wird in das Hauptdirectory von df1 kopiert. Dabei werden Bestätigungsmeldungen des Copy-Befehls unterbunden.

Der zweite Parameter ist das Wort »ALL«. Damit sagen Sie dem COPY-Befehl, daß, falls Sie ein Directory kopieren wollen, auch alle Unterverzeichnisse dieses Ordners mitkopiert werden sollen (normalerweise werden nur die Dateien, nicht (!) die Subdirectories kopiert). Um dies zu verdeutlichen, folgendes Beispiel:

```
COPY df0:Devs TO df1:
```

Es würden alle Dateien übertragen, die sich im Verzeichnis »Devs« befinden, nicht aber die Unterverzeichnisse.

Folgende Befehlszeile würde auch noch die Unterverzeichnisse mitkopieren und gleichzeitig auf der Zieldiskette die entsprechenden Ordner erzeugen:

```
COPY df0:Devs TO df1:ALL
```

Dies würde auch mit Festplattenlaufwerken (dh0/dh1 beziehungsweise beim Amiga 2000 jh0 etc.) und dem RAM funktionieren. Beispiele:

```
COPY df0:c TO dh0: ALL
```

```
COPY df1:c TO RAM:
```

Wie weiter oben erwähnt, läßt sich die Ausgabe einer Datei auch auf den Drucker umlenken. Der Drucker würde also eine schriftliche Kopie einer Datei ergeben. Unter AmigaDOS wird ein angeschlossener Drucker durch »prt:« (für Printer) angesprochen. Legen wir als Beispiel die Ausgabe einer Datei um, beziehungsweise kopieren die Datei auf ein Blatt Papier. Dafür benutzen wir die Startdatei des Amiga (Startup-Sequence), die sich im S-Verzeichnis befindet und von der wir wissen, daß sie lesbaren ASCII-Text enthält. Die Befehlszeile sieht so aus:

```
COPY df0:S/startup-
sequence TO prt:
```

Schon wird ein Ausdruck dieser Datei erzeugt.

Auf diese Weise können beispielsweise auch Daten über die serielle »SER:« oder parallele »PAR:« Schnittstelle gesendet werden.

Ein weiteres Gerät ist der Bildschirm »CON:«.

Wenn Sie, wie auch im Amiga-DOS-Handbuch beschrieben, nachstehende Zeile abtippen, werden alle Eingaben von Tastatur (die Tastatur wird durch den Stern »*« gekennzeichnet) auf ein neu eröffnetes Bildschirmfenster umgeleitet. Sie tippen also etwas ein, das anschließend direkt in einem anderen Fenster, das den Titel »Ausgabefenster« trägt, ausgegeben wird (Abbruch mit <CTRL C> oder <CTRL =>, eingegeben im CLI-Fenster).

```
COPY * TO CON:10/10/200/
100/Ausgabefenster
```

Vielleicht verstehen Sie jetzt auch die Anweisung am Anfang dieses Abschnitts, mit der wir die Test-Datei erzeugt haben. Diese Anweisung macht nichts anderes, als Tastatureingaben in eine Datei mit dem Namen »Test« umzuleiten und nach Druck auf <CTRL \> auf Diskette zu speichern.

Diskettenkopie anfertigen

Ein Befehl, der (wenn keine Kopierprogramme vorhanden sind) das Kopieren kompletter Disketten erlaubt, ist das Kommando »DISKCOPY«. Damit lassen sich originalgetreue Kopien von 3½-Zoll-Disketten anfertigen. Kaputte Sektoren oder Tracks werden damit aber nicht übertragen. DISKCOPY führt die selbe Funktion aus, als wenn Sie auf der Workbench-Ebene zwei Diskettensymbole übereinander legen. Dabei ist es nicht nötig, die Zieldiskette vorher zu formatieren. Dies wird von »DISKCOPY« erledigt. (Es empfiehlt sich, die Quelldiskette mit Schreibschutz zu versehen.) Das Format des Befehls sieht so aus:

```
DISKCOPY Quell-Laufwerk
TO Ziel-Laufwerk
```

Ein Beispiel: Es soll von der Diskette, die sich im internen Laufwerk (df0) befindet, eine Sicherungskopie auf eine Diskette erstellt werden, die sich im df1 befindet:

```
DISKCOPY df0: TO df1:
```

Nach Eingabe dieses Be-

fehls wird nachgefragt, ob die Diskette wirklich kopiert werden soll. Nach Bestätigung befindet sich nach Abschluß des Vorgangs ein identisches Abbild der Diskette im Ziel-Laufwerk. Die Zieldiskette trägt zur Unterscheidung den Namen »COPY OF xxx«, wobei »xxx« der Name der Ursprungsdiskette ist. Soll die Zieldiskette einen anderen Namen erhalten, kann auch noch der Formalparameter »NAME« am Schluß der Befehlszeile angehängt werden. Dies sieht dann so aus:

```
DISKCOPY df0: TO df1:
NAME "Neuer Name"
```

Die Diskette erhält auf diese Weise gleich beim Kopieren einen neuen Namen.

Es kann auch mit nur einem Laufwerk dupliziert werden. Wenn Sie

```
DISKCOPY df0: TO df0:
```

eingeben, werden Sie (wenn es an der Zeit ist) vom Programm aufgefordert, die jeweils benötigte Diskette (Quelle oder Ziel) einzulegen.

Der nächste Befehl, den wir nun besprechen, dient zur Abwechslung nicht der Datei- oder Diskettenmanipulation, sondern »nur« zum Betrachten von Dateien, geeigneterweise ASCII-Dateien, also beispielsweise Textfiles. Es ist dies das Kommando »TYPE«, das die Inhalte angegebener Dateien auf dem Bildschirm ausgibt. »TYPE« kann auch wieder in mehreren Variationen verwendet werden. Die einfachste Form ist die Ausgabe einer Datei auf den Bildschirm. Nehmen wir wieder die Datei »Startup-Sequence« im Ordner »S«, von der wir ja schon wissen, daß sie lesbaren Text im ASCII-Format enthält:

```
TYPE S/startup-sequence
```

Der Inhalt des Files wird auf dem Bildschirm angezeigt. Sollte der Text aus dem CLI-Fenster scrollen, hält ein Druck auf die Leertaste die Ausgabe an. <RETURN> oder <BACKSPACE> (der Linkspfeil rechts oben auf der Tastatur) setzt die Anzeige fort. <CTRL C> bricht die ganze Ausgabe ab.

Sie können aber auch die Ausgabe einer Datei auf ein anderes Gerät lenken. Der nächste Befehl gibt beispielsweise die Datei auf einen angeschlossenen Drucker aus:

```
TYPE S/startup-sequence
TO prt:
```

Auch beim TYPE-Kommando lassen sich Zusatzparame-

ter mitangeben. Der Parameter »OPT N«, am Schluß der Befehlszeile eingegeben, bewirkt, daß vor jede ausgegebene Zeile eine Zeilennummer gesetzt wird (nur für die Ausgabe gültig). Das obige Beispiel, diesmal mit Zeilennummern:

```
TYPE S/startup-sequence
TO prt: OPT N
```

Der Parameter »OPT H« hat zur Folge, daß die betreffende Datei nicht im Klartext, sondern als Hex-Dump (Ausgabe von Hexadezimal-Zahlen) angezeigt wird. Dies dürfte aber nur für Programmierer von Interesse sein.

Anzeigen von ASCII-Dateien

Bevor wir uns mit dem umfangreichen Befehl »ED« beschäftigen, werden zuerst einige leichter zu verstehende Kommandos des CLI behandelt. Der nächste Befehl ist »MAKEDIR«.

Im Regelfall arbeitet man bei einer Diskette oder Festplatte in der obersten Directory-Ebene, dem sogenannten »Root-Directory«, in welches Sie immer mit dem Befehl

```
CD :
```

gelangen (korrekterweise sollte »CD Laufwerk:« geschrieben werden, um sicherzugehen, daß auch das richtige Laufwerk angesprochen wird). Sie können in dieses Verzeichnis alle Programme speichern, die Sie möchten. Wenn Sie sich aber nun vorstellen, daß Sie (etwa bei Festplatten) bis zu mehrere hundert Programme in dieses Verzeichnis speichern können, kommt Ihnen sicher der Gedanke, daß beispielsweise die Directory-Anzeige dieser Diskette/Festplatte sehr unübersichtlich wird. Die einzelnen Einträge passen gar nicht alle auf den Bildschirm. Außerdem können weitere Probleme auftreten, wenn man etwa alle Dateien, die zu einem Programm gehören, kopieren möchte und diese erst mühsam zusammensuchen muß, da nicht klar ersichtlich ist, welche dies sind. Zu diesem Zweck schufen findige Entwickler Subdirectories. Dies sind Unterverzeichnisse, die einem anderen Directory untergeordnet sind. Damit ist es möglich, für bestimmte Anwendungen eigene »Ordner« auf der Diskette/Festplatte zu schaffen, in denen dann zu einem Gebiet gehörige Programme gespeichert werden

(Bild 3). Bestes Beispiel dafür ist die Workbench-Diskette. Sie enthält eine Vielzahl verschiedener Subdirectories, in denen teilweise noch weitere Unterverzeichnisse eingebettet sind. Von diesen Ordnern sind aber auf der Workbench-Ebene nur die zu sehen, zu denen ein entsprechendes *.info-File existiert, welches beispielsweise die Grafikdaten für das Bildsymbol zur Verfügung stellt.

Der »c«-Ordner enthält nur Programme, die etwas mit dem CLI zu tun haben. Im Ordner »devs« dagegen befinden sich nur Programme, die etwas mit der Tastaturanpassung, Druckern oder anderen Gerätetreibern zu tun haben.

Auch Sie werden früher oder später in Situationen kommen, in denen Sie der Übersichtlichkeit halber Dateiverzeichnisse erstellen müssen. Das CLI stellt dafür den Befehl »MAKEDIR« zur Verfügung. Die Schreibweise für dieses Kommando lautet allgemein:

MAKEDIR (Verzeichnis)
Name

»Verzeichnis« muß nicht angegeben werden, wenn Sie in dem Directory, in dem Sie sich gerade befinden, ein Unterverzeichnis anlegen möchten. Soll aber ein Verzeichnis generiert werden, das beispielsweise zwei Ebenen tiefer liegt (Bild 3, untere Reihe), ist der komplette Pfad anzugeben. Zwei Beispiele:

```
MAKEDIR df0:Anwendung
MAKEDIR df0:Anwendung/
Grafik
```

Im ersten Fall wird im aktuellen Verzeichnis das Unterverzeichnis »Anwendung« erstellt. Im zweiten Fall dagegen entsteht der Ordner »Grafik« im Verzeichnis »Anwendung«, das bereits dem aktuellen Directory untergeordnet ist. Das Directory »Grafik« ist also in der zweiten Verzeichnisebene angesiedelt (siehe dazu auch das Bild 3). Die Angabe des Laufwerks ist nicht unbedingt erforderlich, erhöht aber die Lesbarkeit. Ebenfalls ist darauf zu achten, daß der Name des zu erzeugenden Verzeichnisses in Anführungszeichen gesetzt wird, wenn Leerzeichen oder Sonderzeichen enthalten sind.

Aktivieren der Echtzeituhr

Der Amiga 2000 enthält fest eingebaut eine batteriegepufferte Echtzeituhr, die auch

nach dem Ausschalten des Computers weiterläuft und somit beim nächsten Einschalten die aktuelle Uhrzeit zur Verfügung stellt. Auch ein Amiga 500, der mit der intern einzubauenden Speichererweiterungskarte (Erweiterung auf 1 MByte) ausgerüstet ist, verfügt über eine solche Echtzeituhr. Leider kann diese Uhr nicht mit »DATE« gestellt und ausgelesen werden. Dazu existiert auf neueren Workbench-Versionen (ab Versionsnummer 33.4xx) der Befehl »SET-CLOCK«. Dieser Befehl ermöglicht das Übertragen der Amiga-Systemuhr von und zur

Dies funktioniert aber nur dann, wenn Sie tatsächlich über die batteriegepufferte Uhr verfügen. Ansonsten meldet das Amiga-DOS (Betriebssystem) einen Fehler!

Damit dieser letztgenannte Befehl nicht jedesmal von Hand eingegeben werden muß, bietet er sich geradezu dafür an, in die Startdatei der Workbench-Diskette eingebunden zu werden, die jedesmal beim Neustart des Amiga abgearbeitet wird. Wie das gemacht wird, erfahren Sie etwas später im Kurs, wenn der »ED«-Befehl besprochen wird. Ein ebenfalls leicht zu verstehen-

müssen Sie nur den Parameter »%N« einfügen. Für unser Beispiel sähe dies so aus:

PROMPT "Hallo, User %N>

Das Ergebnis auf dem Bildschirm hätte die Form »Hallo, User 1>«. Den normalen, gewohnten Prompt bringt Ihnen folgender Befehl zurück:

PROMPT %N>

Ein weiterer Befehl, der eigentlich nur sehr selten zum Einsatz kommt, ist das Kommando »WAIT«. Es ist im Regelfall nur für Besitzer eines Amiga 2000 mit PC-/AT-Karte oder eines Sidecar interessant, die eine Festplatte besitzen und bei denen der Controller in einem PC-Steckplatz steckt. Hier muß der Amiga so lange mit Festplattenzugriffen warten (etwa 40 Sekunden), bis der PC nach dem Booten die Platte nicht mehr benötigt (sonst kommen sich beide Systeme bei Zugriffen auf die Platte ins Gehege).

Warten mit Variationen

Der WAIT-Befehl kann jedoch durch seine vielfältigen Möglichkeiten auch für andere Zwecke eingesetzt werden. Beispielsweise, wenn eine mit »ECHO« ausgegebene Meldung länger am Bildschirm sichtbar bleiben soll und erst danach das Startprogramm weiter abgearbeitet wird. Im einfachsten Fall wartet der Amiga eine Sekunde, bis er mit der Abarbeitung der Startdatei fortfährt. Dies erreicht man durch Eingabe von »WAIT« ohne weitere Parameter. Soll länger gewartet werden, muß, durch ein Leerzeichen getrennt, die Zeitdauer in Sekunden (SEC) oder Minuten (MIN) angegeben werden. Zwei Beispiele:

```
WAIT 32sec
WAIT 3min
```

Im ersten Fall stoppt das System für 32 Sekunden, beim zweiten Beispiel wartet der Amiga drei Minuten. Es ist aber auch möglich, den Computer bis zum Erreichen einer bestimmten Uhrzeit anzuhalten. Das nachstehende Beispiel würde den Amiga warten lassen, bis die angegebene Uhrzeit erreicht wird:

```
WAIT UNTIL 12:00
```

Ein dagegen sehr wichtiger Befehl, der eigentlich in jede Startdatei gehört, ist »LOADWB«. Erst dieser Befehl ermöglicht es dem Anwender,

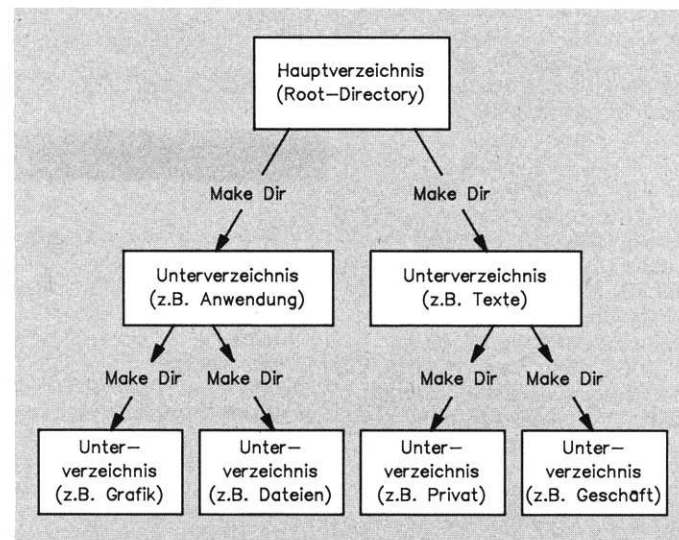


Bild 3. Ein Beispiel für eine Diskette/Festplatte, auf der nach Themen geordnet Subdirectories erzeugt wurden

batteriegepufferten Echtzeituhr.

Das Stellen dieser Uhr geht relativ einfach vonstatten: zuerst ist mit dem »DATE«-Befehl die Systemuhr des Amiga zu stellen.

Anschließend muß dieses Datum in die Echtzeituhr übertragen werden. Der Befehl dazu sieht folgendermaßen aus:

```
SETCLOCK opt save
```

Das Datum (und damit auch die Uhrzeit) sind nun in die Echtzeituhr übertragen und bleiben auch nach dem Ausschalten des Amiga erhalten.

Nach einem erneuten Einschalten des Computers befindet sich das Datum noch in der Echtzeituhr, nicht aber in der Systemuhr des Amiga. Die Zeit muß also bei jedem Neustart wieder aus der Echtzeituhr ausgelesen und in die Systemuhr übertragen werden. Auch hierzu dient wieder der SET-CLOCK-Befehl, diesmal nur mit anderen Parametern:

```
SETCLOCK opt load
```

der Befehl, der sich auch zum Einbinden in die Startdatei eignet, ist das Kommando »PROMPT«.

Doch was ist ein »Prompt«? Wie Sie sicher bereits festgestellt haben, steht im CLI-Fenster immer die Kennung des jeweiligen Command-Line-Interfaces (Sie können mehrere CLI-Fenster gleichzeitig öffnen) vor dem Cursor (1>). Sie können diese Meldung aber auch ändern, wenn eine »eigene« Bereitschaftsanzeige gewünscht wird. Nehmen wir an, der normale Prompt soll so geändert werden, daß er eine persönlichere Note erhält:

```
PROMPT "Hallo, User> "
```

Nach Eingabe dieses Kommandos würde nun im aktuellen CLI-Fenster anstelle des »1>«-Prompts die Bereitschaftsanzeige »Hallo, User>« erscheinen.

Wenn Sie aber zusätzlich zu dieser neuen Meldung auch noch die Nummer des aktuellen CLI-Tasks sehen möchten,

daß er mit den Icons (Bildsymbolen) auf der Workbench-Ebene arbeiten kann.

LOADWB erlaubt noch die Angabe eines speziellen Zusatzparameters:

```
LOADWB -debug
```

Hiermit wird eine Spezialfunktion (für »Eingeweihte«: ROMWack) des Amiga aktiviert, die nur für Programmierer interessant ist, die sich sehr gut mit dem Amiga-Betriebssystem auskennen und auch Maschinensprache beherrschen müssen. Deshalb soll auf diese Zusatzfunktion nicht näher eingegangen werden, da bei falscher Anwendung die Gefahr eines »endgültigen Systemabsturzes« besteht.

Nach dieser »Aufwärmübung« besprechen wir nun einen der umfangreichsten Befehle, die uns das CLI zur Verfügung stellt: den Texteditor »ED«. Er ermöglicht das Erzeugen und Bearbeiten von Texten, die im ASCII-Format (!) vorliegen müssen. Ein Beispiel für eine solche Textdatei ist die in diesem Kurs bereits öfter erwähnte Startdatei »Startup-Sequence«, welche sich im »s«-Directory befindet. Bei jedem Neustart des Amiga wird diese Datei geladen und alle in ihr enthaltenen Befehle der Reihe nach abgearbeitet. Sie können sich den Inhalt dieser Datei beispielsweise mit dem Befehl

```
TYPE :s/startup-sequence
```

ansetzen. Bearbeiten läßt sie sich jedoch nur mit dem Editor.

Der Aufruf von ED erfolgt sehr einfach. Es ist nur der Name der zu bearbeitenden Datei sowie, wenn erforderlich, die Größe des zu reservierenden Textspeichers anzugeben. Wird keine Größe angegeben, reserviert ED von sich aus 40000 Byte Textspeicher, was für durchschnittlich lange Texte völlig genügt.

Versuchen wir doch gleich einmal, eine Datei zu bearbeiten beziehungsweise zu erzeugen. Geben Sie folgendes ein:

```
ED :Test-Ed SIZE 50000
```

Die Angabe von »size 50000« wäre nicht unbedingt erforderlich gewesen, sondern soll nur die Parameterübergabe zur Größenbestimmung des zu reservierenden Textspeichers demonstrieren. Ebenso ist auch der Doppelpunkt vor dem Dateinamen nicht nötig. Er stellt nur sicher, daß die Datei im Root-Directory (Hauptdirectory) des aktuellen Laufwerks erzeugt wird.

Da die Datei »Test-Ed« noch nicht existiert, gibt ED eine Meldung aus, die besagt, daß eine neue Datei erzeugt wird. Doch widmen wir uns nun den Funktionen des Editors:

Im normalen Betrieb des Editors, in dem Sie sich nach dem Start befinden, können Sie ganz normal Texte eingeben. Da ED ein »Fullscreen-Editor« ist, läßt sich der Cursor ohne Probleme im ganzen Fenster bewegen. Ausnahme ist die letzte Zeile des Fensters, die für Editor-Meldungen reserviert ist.

Außer der Eingabe des ASCII-Textes beherrscht ED eine Vielzahl von Befehlen, die dem Laden, Speichern und Bearbeiten des Textes dienen. Alle diese Befehle (von Commodore als »erweiterter Modus« bezeichnet) werden durch einen Druck auf die ESC- und CTRL-Taste eingeleitet. Da die einzelnen Steuerbefehle im mitgelieferten Handbuch ausführlich genug beschrieben sind, soll Ihnen die Tabelle 1 zusammenfassend einen Überblick über die wichtigsten Funktionen geben.

Doch wozu kann man nun den Editor verwenden, werden Sie sich vielleicht fragen? Mit einer Textverarbeitung kann doch viel komfortabler gearbeitet werden! Doch dazu dient ED ja auch gar nicht. Vielmehr ist er dazu da, kleine ASCII-Files wie beispielsweise die Startdatei oder ähnliche, ausführbare Dateien, sogenannte Batch-Dateien, zu erstellen.

Das Betriebssystem des Amiga erlaubt es, verschiedene Befehle des CLI zu einem kleinen Programm zusammenzufassen, das den Sinn hat, einzelne Kommandofolgen, die häufig gebraucht werden, durch einen einzigen Befehl zu ersetzen. Versuchen wir doch gleich einmal, eine solche ausführbare Datei zu erzeugen! Da wir uns bereits im Editor befinden, können wir gleich loslegen. Als Beispiel werden wir mit den Befehlen, die wir bis jetzt kennengelernt haben, eine Batch-Datei erstellen.

Folgende Befehle, die natürlich auch im Direktmodus eingegeben werden können, eignen sich gut für diese Datei:

```
DIR
INFO
LOADWB
SETCLOCK opt load
```

Der letzte Befehl ist natürlich nur dann sinnvoll, wenn Sie über die batteriegepufferte Echtzeituhr verfügen.

Geben Sie nun die drei beziehungsweise vier Befehle im ED-Fenster ein und schließen jeden Befehl durch <RETURN> ab. Wenn Sie anschließend <ESC> <X> <RETURN> eingeben, wird der Editor beendet, nachdem die kleine Batch-Datei unter dem Namen »Test-Ed« in das Hauptdirectory des aktuellen Laufwerks gespeichert wurde. Sie können sich den Inhalt dieser ASCII-Datei noch einmal mit dem Befehl

```
TYPE :Test-Ed
```

anzeigen lassen. Jedoch können Sie ohne weiteres nichts mit dieser Datei anfangen. Zum Aufruf eines Batchfiles stellt das CLI einen weiteren Befehl zur Verfügung: das Kommando »EXECUTE«.

Nach dem Aufruf des EXECUTE-Befehls, hinter dem der Name der auszuführenden Datei angegeben werden muß, startet der Amiga mit der Ausführung der Batch-Datei. Die Abarbeitung kann, wenn erforderlich, mit <CTRL D> abgebrochen werden.

Um die soeben erzeugte Datei zu starten, geben Sie also folgendes ein:

```
EXECUTE :Test-Ed
```

Der Amiga wird anschließend das Directory des aktuellen Laufwerks anzeigen, danach den Zustand und die Belegung der angeschlossenen Geräte (Laufwerke) ausgeben und schließlich nochmals die Workbench laden. Wenn Sie den vierten Befehl auch noch eingetippt haben, zeigt der

Alle Befehle des Editors

<CTRL A>	— an der aktuellen Cursorposition wird eine Zeile eingefügt
<CTRL B>	— die Zeile, in der sich der Cursor gerade befindet, wird gelöscht
<CTRL D>	— der Text im ED-Fenster wird um einige Zeilen nach unten gescrollt
<CTRL E>	— setzt den Cursor entweder an die obere oder untere Begrenzung des ED-Fensters
<CTRL F>	— damit läßt sich der Buchstabe, auf dem sich der Cursor befindet, von Kleinschrift auf Großschrift und umgekehrt wechseln
<CTRL G>	— das letzte ESC-Kommando wird noch einmal ausgeführt
<CTRL H>	— das Zeichen, das links vom Cursor steht, wird gelöscht (Funktion wie die Backspace-Taste)
<CTRL I>	— der Cursor springt auf die nächste Tabulatormarke (normalerweise drei Zeichen, Funktion wie TAB-Taste)
<CTRL O>	— Löscht ein Wort oder Leerstellen bis zum Beginn des nächsten Wortes oder Leerzeichens
<CTRL R>	— bewegt den Cursor auf den Schluß des vorhergehenden Wortes
<CTRL T>	— bewegt den Cursor auf den ersten Buchstaben des nächsten Wortes
<CTRL U>	— der Text im ED-Fenster wird um einige Zeilen nach oben gescrollt
<CTRL V>	— das ED-Fenster wird neu gezeichnet (»refresh«)
<CTRL Y>	— der dem Cursor folgende Text wird bis zum Zeilenende gelöscht
<CTRL J>	— positioniert den Cursor auf den Anfang oder das Ende der aktuellen Zeile
<ESC> <A> "Text"	— der in den Anführungszeichen stehende Text wird in der nächsten Zeile eingefügt
<ESC> 	— der Cursor springt an das Ende des gesamten Textes
<ESC> <E>	— die aktuelle Cursorposition wird als das Blockende markiert
<ESC> <S>	— die aktuelle Cursorposition wird als Blockanfang markiert
<ESC> <D>	— die Zeile, in der sich der Cursor gerade befindet, wird gelöscht
<ESC> <D> 	— der mit »BS« und »BE« markierte Block wird gelöscht
<ESC> <E> "Text1" "Text2"	— der Text »Text1« wird durch »Text2« ersetzt
<ESC> <E> <Q> "Text1" "Text2"	— wie »E«, nur wird vorher eine Bestätigung verlangt
<ESC> <E> <X>	— die rechte Randmarkierung wird ignoriert, es

Tabelle 1. Der Texteditor »ED« beherrscht alle Funktionen, die z

Amiga nochmals die aktuelle Zeit aus der batteriegepufferten Uhr an. Der Befehl EXECUTE kann zwar noch viel mehr als nur eine Batch-Datei aufrufen, doch diese weiteren Möglichkeiten werden erst besprochen, wenn Sie noch weitere Befehle kennengelernt haben, die sich zur Einbindung in eine Batch-Datei eignen.

Sie können sich natürlich auch die Startdatei der Workbench-Diskette mit dem Kommando

```
Ed :s/startup-sequence
```

ansetzen und mit ihr herumexperimentieren. Achten Sie aber vorher darauf, daß die Diskette schreibgeschützt ist, da Sie sonst unter Umständen schwer zu behebbende Fehler verursachen könnten, wenn

die möglicherweise veränderte Datei auf die Diskette zurückgeschrieben wird. Deshalb sollte in diesem Fall der Editor mit <ESC> <Q> verlassen werden, um ein Zurückschreiben der Batch-Datei zu verhindern.

Starten mit Komfort

Das Betriebssystem des Amiga kontrolliert bei jedem Neustart, ob das »s«-Verzeichnis auf Diskette vorhanden ist und ob sich in diesem Directory die Startdatei, die »Startup-Sequence«, befindet. Ist diese vorhanden, wird sie geladen und die einzelnen Befehle, die in diesem Programm enthalten sind, der Reihe nach abgear-

beitet. Wie dies in etwa funktioniert, zeigt Ihnen Bild 4.

Wir beschäftigen uns jetzt mit CLI-Befehlen, die sich hervorragend zum Einbau in eine Batch-Datei eignen, auch wenn die meisten dieser Befehle genauso gut im Direktmodus eingegeben werden können.

Beginnen wir zuerst mit einem der einfachsten Befehle, mit ECHO, das der Ausgabe eines Textes dient. ECHO ist eigentlich nur sinnvoll, wenn es in eine Batch-Datei eingebunden wird (Wer will schon im Direktmodus einen Text ausgeben?). Ein Beispiel für einen Einsatz von ECHO: Sie möchten gerne aktuell darüber informiert sein, was der Amiga gerade in der Startdatei bearbeitet und wünschen eine Ausgabe eines betreffenden Textes im CLI-Fenster. Dazu wird einfach der ECHO-Befehl vor jeden Befehl in der Batch-Datei gesetzt. Erstellen wir uns gleich eine Stapeldatei. Geben Sie dazu bitte folgendes ein:

```
ED RAM:Test
```

Damit erstellen Sie im RAM eine kleine Stapeldatei, die später mit

```
EXECUTE RAM:Test
```

ausgeführt wird. Da wir nur den ECHO-Befehl kennenlernen wollen, genügt eine kleine Befehlsfolge. Zu beachten ist bei ECHO nur, daß der auszugebende Text in Anführungszeichen stehen und vom Rest durch ein Leerzeichen getrennt sein muß. Tippen Sie bitte folgende Zeilen im ED ein und beenden den Editor anschließend mit <ESC X> <RETURN>:

```
ECHO "Lese Directory"
ECHO ""
DIR df0:
ECHO "Erstelle Info"
ECHO ""
INFO
ECHO "Lade Workbench"
ECHO ""
LOADWB
```

Die ersten beiden Zeilen geben auf dem Bildschirm den Text »Lese Directory« und eine Leerzeile aus. Nach Ausgabe dieser Meldung liest der Amiga das Inhaltsverzeichnis der in das interne Laufwerk (df0) eingelegten Diskette und zeigt es am Bildschirm an. Als nächstes gibt der Computer wieder einen Text aus: »Erstelle Info«, gefolgt von einer Leerzeile. Der darauf folgende Befehl INFO informiert Sie über die angeschlossenen Laufwerke und

Geräte. Nach der dritten Textausgabe »Lade Workbench« wird noch einmal die Workbench geladen und die Diskettensymbole »erneuert«. Der ECHO-Befehl kann jederzeit in eine Batch-Datei eingebunden werden, um beispielsweise aktuelle Tätigkeiten oder benutzereigene Meldungen (etwa »Hallo User ...«) anzuzeigen.

Ebenfalls zum Einbinden in eine Batch-Datei ist FAILAT gedacht. FAILAT dient dem Anheben des »Fehlerniveaus« beim Abarbeiten einer Stapeldatei: Erläutern wir dies genauer:

Jedesmal, wenn bei der Ausführung eines Stapeldatei-Befehls ein Fehler auftritt, liefert Amiga-DOS je nach Schwere der Fehlfunktion einen Wert zurück. Je nach Höhe des eingestellten Fehlerniveaus wird entweder auf den Fehler reagiert, oder er wird ignoriert. Standardeinstellung bei der Ausführung einer Batch-Datei ist der Wert 10. Dies bedeutet, daß die weitere Bearbeitung der Stapeldatei unterbrochen wird, wenn ein Fehler mit einem größeren Wert als 10 gemeldet wird. Es gibt beispielsweise leichte Fehler mit dem Wert 5, bei denen vom Amiga-DOS Warnungen ausgegeben werden. Will man aber nicht, daß die Weiterführung, aus welchen Fehlergründen auch immer, abgebrochen wird, kann man das Fehlerniveau erhöhen. So sichert etwa

```
FAILAT 30
```

oder ein höher angesetztes Fehlerniveau, daß die Abarbeitung der Batch-Datei nicht gestoppt wird. Nach Beendigung der Befehlsausführung der Stapeldatei wird der Wert von Amiga-DOS automatisch wieder auf 10 gesetzt. Das Abbruchniveau ist also nur solange gültig, wie die Stapeldatei abgearbeitet wird.

Einbindung des PC-Systems

Ein weiterer Befehl, der aber nur für Besitzer von PC-Erweiterungen (etwa der Janus-PC/AT-Karte) Bedeutung besitzt, ist BINDDRIVERS. Dieser Befehl läßt sich sowohl im Direktmodus wie auch in Stapeldateien verwenden.

BINDDRIVERS, das ohne zusätzliche Parameter eingegeben wird, ermöglicht erst die Einbindung des PC-Systems in das Amigasystem. Es ist quasi das Kommando, den PC/AT zum Leben zu erwecken und seine Tätigkeit als Task in

»ED« auf einen Blick

kann also über das eingestellte rechte Randende hinausgeschrieben werden

<ESC> <F> "Text" — von der Cursorposition ab vorwärts wird der Text nach einer bestimmten Zeichenfolge durchsucht

<ESC> <I> "Text" — vor der Zeile, in der sich der Cursor gerade befindet, wird ein Text eingefügt

<ESC> <I> — der mit »BS« und »BE« markierte Block wird ab der, der Cursorposition folgenden Zeile nochmals eingefügt

<ESC> <I> <F> "Name" — dieser Befehl gestattet es dem Anwender, in der nach dem Cursor folgenden Zeile die Datei »Name« einzufügen

<ESC> <L> <C> — bei der Suche nach einem Text (»F«) soll zwischen Groß- und Kleinschreibung unterschieden werden

<ESC> <N> — der Cursor wird an den Anfang der nächsten Zeile gesetzt

<ESC> <P> — der Cursor wird an den Anfang der vorhergehenden Zeile gesetzt

<ESC> <Q> — der Editor wird verlassen, ohne daß der bearbeitete Text gespeichert wird. Vorher erfolgt aber eine Sicherheitsabfrage.

<ESC> <R> <P> — der nach RP eingegebene Befehl wird so lange wiederholt, bis ein Fehler auftritt oder das Textende erreicht ist. Beispiel: RP D

<ESC> <S> <A> "Name" — der Text wird gespeichert, ohne den Editor zu beenden. Wird »Name« weggelassen, erfolgt die Speicherung unter dem Namen, der beim Aufruf von ED angegeben wurde.

<ESC> <S> — der mit »BS« und »BE« markierte Block wird im ED-Fenster angezeigt

<ESC> <S> <H> — mit diesem Kommando kann man sich die aktuellen Einstellungen des Editors anzeigen lassen

<ESC> <S> <L> x — Der linke Randsteller wird auf Spalte »x« gesetzt

<ESC> <S> <R> x — Der rechte Randsteller wird auf Spalte »x« gesetzt

<ESC> <S> <T> x — dieser Befehl dient zum Einstellen der Tabulatorstopps (der Normalwert beträgt x=3)

<ESC> <T> — der Cursor wird an den Textanfang gesetzt

<ESC> <U> — damit werden Änderungen an der aktuellen Zeile rückgängig gemacht. Dies funktioniert aber nur, wenn noch nicht <RETURN> gedrückt wurde.

<ESC> <U> <C> — bei der Suche nach Text (»F«) wird nicht zwischen Groß- und Kleinschreibung unterschieden

<ESC> <W> "Name" — der mit »BS« und »BE« markierte Block wird in die Datei »Name« gespeichert

<ESC> <X> "Name" — mit diesem Befehl wird der Text gespeichert und ED beendet. Wird dabei »Name« nicht angegeben, so benutzt ED den Namen, der beim Editor-Aufruf mit angegeben wurde.

das Amigasystem einzuflechten. Dieser Befehl sollte, wenn er benötigt wird, immer am Anfang der auszuführenden Batch-Datei stehen (nach FAIL-AT, wenn vorhanden).

Ebenso leicht zu erklären ist die Anweisung STACK. Jedem CLI-Befehl wird vor seiner Ausführung ein bestimmter Speicherbereich (Stapelspeicher) reserviert, den er, falls zur Befehlsausführung nötig, verwenden kann. Im Normalfall sind dies 4000 Byte. Es gibt aber einige CLI-Befehle, die mehr Platz für die Ablage von Daten benötigen (etwa DIR oder SORT). Um diesen Befehlen mehr Stapelspeicher zu gewähren, dient STACK. Will man die Größe des aktuellen Stapelspeichers erfahren, ist STACK ohne Parameter einzugeben. Soll dagegen der Stapel verändert werden, muß man, durch ein Leerzeichen getrennt, die Größe des neuen Bereichs angeben (diese Größe sollte aber mindestens 4000 Byte betragen). Ein Beispiel:

```
STACK 8000
```

Nach Eingabe dieses Kommandos stehen den CLI-Befehlen nun 8000 Byte Stapelspeicher zur Verfügung.

Gegen Diskettenwechsel

Gerade für Besitzer von nur einem Laufwerk oder einer zusätzlichen Festplatte ist folgender Befehl geeignet: ASSIGN. Doch um den Befehl zu verstehen, ist erst eine Erklärung über einen Teil des Amiga-Betriebssystems nötig.

Für das Amiga-DOS existieren verschiedene Standardverzeichnisse (Directories), in denen nach bestimmten Dateien gesucht wird. So sucht das Betriebssystem beispielsweise jeden CLI-Befehl im »c«-Directory und Zeichensätze im »fonts«-Ordner. Beim Start mit einer Workbench-Diskette sind diese Verzeichnisse, wenn nicht anders angegeben, auf die Startdiskette fixiert. Dies bedeutet: auch wenn Sie die Startdiskette entnehmen und eine andere Diskette einlegen, auf der die gleichen Dateien enthalten sind, wird der Amiga, wenn man beispielsweise einen CLI-Befehl eingibt, immer das Einlegen der Startdiskette verlangen. Das bedeutet, daß er es nicht registriert, wenn auch auf der eingelegten Diskette der benötigte Befehl im »c«-Directory vorhanden ist. Geben Sie bitte zum weiteren

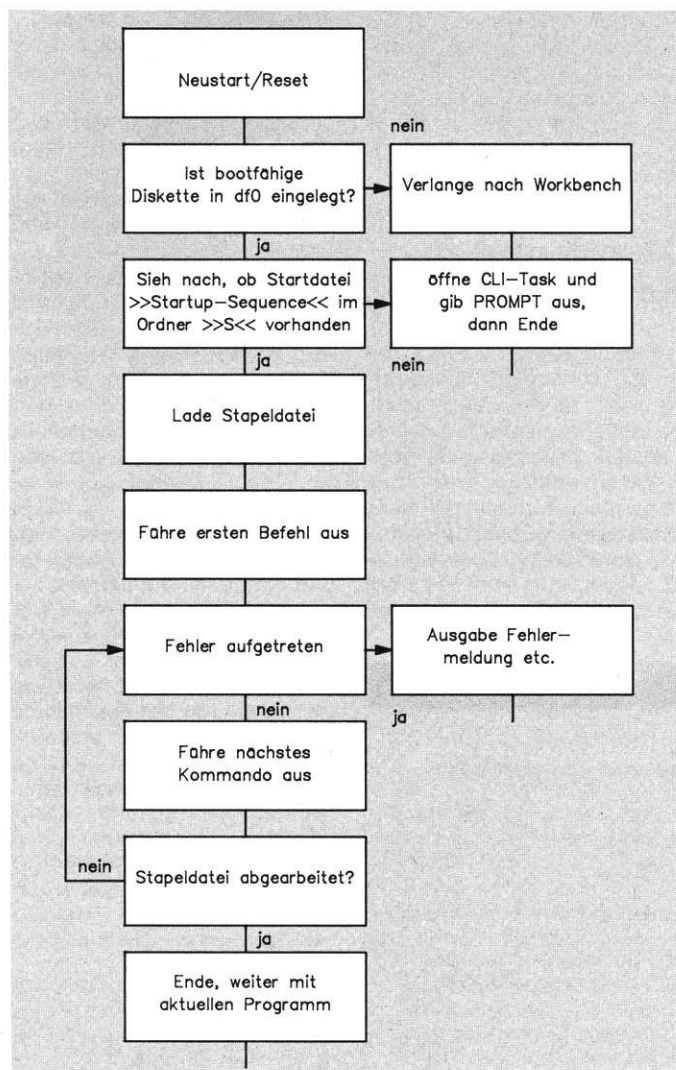


Bild 4. Die Startdatei »Startup-Sequence« wird vom Amiga-DOS beim Start in dieser Art und Weise behandelt

Verständnis folgendes ein:
ASSIGN

Es wird eine Liste ausgegeben, anhand der man erkennen kann, welche Ordner (Verzeichnisse) auf welches Gerät fixiert sind. Im Normalfall sind alle Verzeichnisse auf die Startdiskette festgelegt!

Besitzer von nur einem Laufwerk kennen sicher das Problem: Sie möchten etwa von einer anderen Diskette das Directory lesen, aber der Amiga verlangt immer wieder die Startdiskette und nichts klappt. Abhilfe schafft nun folgendes:

Sie kopieren die Befehle, die häufig benötigt werden (beispielsweise DIR, LIST, INFO) ins RAM und lassen die Befehle von dort aus ausführen. Bei allen Befehlen, die sich dann im RAM befinden, wird der Amiga nun nicht mehr das Einlegen der Startdiskette verlangen. Alles, was zum Gelingen dieser Prozedur nötig ist, ist außer dem Vorhandensein der Befehle im RAM die Mitteilung an den Amiga, wo diese Befeh-

le nun zu finden sind. Dies macht ASSIGN. Probieren wir dies nun im kleinen Rahmen durch. Geben Sie bitte die folgenden Befehle ein, die natürlich auch in einer Batch-Datei stehen können:

```
MAKEDIR RAM:c
COPY c/dir RAM:c
COPY c/list RAM:c
COPY c/info RAM:c
COPY c/assign RAM:c
```

Nun befinden sich die Befehle DIR, LIST, INFO und ASSIGN im RAM, und zwar im zuvor mit MAKEDIR erzeugten »c«-Verzeichnis. Doch der Amiga greift noch immer auf die Startdiskette zu, da wir ihm noch nicht mitgeteilt haben, daß die Befehle jetzt im RAM stehen. Wenn Sie als Abschluß

```
ASSIGN c: RAM:c
```

eingeben, werden die CLI-Befehle von jetzt ab aus dem RAM geholt (natürlich nur diejenigen, die ins RAM kopiert wurden). Es empfiehlt sich, immer ASSIGN in das neue Ver-

zeichnis mitzukopieren, da mit
ASSIGN c: df0:c

wieder auf ein anderes Gerät fixiert werden kann. Und das ist leider nicht mehr so einfach möglich, wenn sich ASSIGN nicht im aktuellen Ordner befindet.

Ebenso können natürlich auch alle anderen Verzeichnisse wie etwa »devs« oder »libs« neu fixiert werden. Wie viele Dateien Sie auf andere Geräte festlegen, ist nur durch den Speicherbereich der betreffenden Einheiten eingeschränkt.

Für Festplattenbesitzer ist der ASSIGN-Befehl natürlich noch interessanter. Sie können sämtliche Standard-Verzeichnisse (etwa devs, l, libs, fonts und c) auf ihre Festplatte installieren und in die Startdatei der Workbench-Diskette einige Zeilen einbauen, die sämtliche Ordner auf die Festplatte fixieren. Danach wird (sofern sich der betreffende Befehl wirklich im »C«-Ordner der Platte befindet) nicht mehr das Einlegen der Startdiskette verlangt, was häufigen Diskettenwechsel erspart. Folgendes, in die Startdatei eingebaute Beispiel würde einige Standardverzeichnisse (soviel Sie wollen) auf die Festplatte »dh0« festlegen (für Amiga 2000-Besitzer mit PC-Festplatte wäre dies das Laufwerk »jh0«):

```
...
ASSIGN c: dh0:c
ASSIGN devs: dh0:devs
ASSIGN l: dh0:l
ASSIGN libs: dh0:libs
ASSIGN fonts: dh0:fonts
ASSIGN sys: dh0:sys
...
```

Wichtig bei ASSIGN ist die richtige Syntax, also die Schreibweise. Es muß immer der Quellordner zuerst angegeben werden. Danach folgt das Zielgerät und ohne Leerzeichen dazwischen der Zielordner.

Schleifenprogrammierung

Was nun folgt, ist ein umfangreiches und äußerst nützliches Werkzeug des CLI, mit dem man Batch-Dateien erstellen kann, die abhängig von Entscheidungen verschiedene Dinge erledigen können. Das Werkzeug ist die Schleifenprogrammierung, die man mit den Befehlen IF, THEN, ELSE und ENDIF verwirklichen kann. Diese vier Befehle lassen sich

nur in Batch-Dateien verwenden, im Direktmodus führen sie zu keinem Ergebnis.

IF prüft auf ein Ereignis. Ist dieses wahr, so wird eine vorher bestimmte Befehlsfolge ausgeführt, die im Regelfall hinter THEN steht. THEN muß nicht unbedingt angegeben werden. Fehlt THEN, werden die folgenden Anweisungen so lange ausgeführt, bis das Betriebssystem auf die Befehle ELSE oder ENDIF trifft.

Stimmt das Ergebnis nicht, erfolgt die Ausführung einer

Zur Erklärung des Beispiels: Es wird geprüft, ob ein Fehler mit dem Wert ≥ 10 aufgetreten ist. Wenn ja, führt der Amiga »(Kommando 1)« aus. Ist dagegen die Bedingung nicht erfüllt worden, fährt der Amiga mit der normalen Abarbeitung der Befehlsfolge fort. Wenn es Sie wundert, daß zwei verschiedene Möglichkeiten zur Auswahl stehen, so können wir Sie beruhigen. Im obigen Fall kann die ELSE-Zeile auch weggelassen werden, wenn nicht zwingend eine Reaktion bei

```
IF (Datei 1) EQ (Datei 2) THEN (Kommando 1)
ELSE (Kommando 2)
ENDIF
```

Stimmen die Dateien »(Datei 1)« und »(Datei 2)« überein, wird »Kommando 1« abgearbeitet. Ansonsten gilt das gleiche wie bereits gesagt.

Da aber möglicherweise auf eine Nichtübereinstimmung geprüft werden könnte, läßt sich auch ein Teil der Abfrage negieren, das heißt, umkehren. Die Bedingung dafür lautet NOT.

```
IF (Datei 1) NOT EQ
(Datei 2) THEN
(Kommando 1)
ELSE (Kommando 2)
ENDIF
```

(Kommando 2) würde in diesem Fall nur ausgeführt werden, wenn die Dateien gleich sind (Bild 7).

Die IF-THEN-ELSE-ENDIF-Konstruktionen können auch ineinander verschachtelt werden, was eine komplexe Batch-Programmierung ermöglicht. Nach dieser etwas schwerer verdaulichen Befehlskombination zum Abschluß dieses Kursteils noch ein Kommando, welches leicht zu verstehen und sehr nützlich ist: PROTECT. Damit können Dateien gegen Schreiben und Löschen, Lesen und Aufrufen geschützt werden. Wenn Sie sich mit dem LIST-Befehl ein Directory ausgeben lassen, können Sie nach dem Datum vier verschiedene Buchstaben in jeder Zeile sehen, die den Status der Datei anzeigen. Diese vier Buchstaben (r-w-e-d) haben folgende Bedeutung:

»r« — die Datei kann gelesen werden,
 »w« — das File läßt sich überschreiben,
 »e« — die Datei kann als Kommando aufgerufen werden,
 »d« — die Daten lassen sich löschen.

Datenschutz eingebaut

Mit dem PROTECT-Befehl können nun gezielt die einzelnen Flags (Schalter) beeinflusst werden. Dabei ist die Schreibweise wie folgt:

```
PROTECT NAME Datei
FLAGS xxxxx
```

Es werden nur die Flags gesetzt, die auch angegeben sind. Um also ein Flag der Datei zu löschen, müssen die verbleibenden drei Flags gesetzt

werden, das vierte wird automatisch gelöscht:

```
PROTECT NAME df1:Test
FLAGS re
```

würde die Datei »Test« gegen Löschen und Überschreiben schützen, da nur die Flags »r« und »e« gesetzt werden.

Bei der PROTECT-Methode ist allerdings eines zu beachten. Wenn Sie mit »Snapshot« die Lage einzelner Icons in einem Window dauerhaft festhalten wollen, dürfen die »info«-Files nicht schreibgeschützt sein! Dies liegt daran, daß in den Icon-Dateien die Lage des später auf dem Workbench-Screens erscheinenden Bildsymbols eingetragen wird.

Um eine flexible Ausführungsdatei programmieren zu können, stellt das CLI verschiedene Befehle zur Verfügung wie etwa IF.THEN..ELSE..ENDIF-Konstruktionen, die besprochen wurden. Es existiert noch eine zweite Befehlskombination, die in etwa mit dem von Basic her bekannten Befehl GOTO gleichzusetzen ist. Es sind dies die Befehle SKIP und LAB.

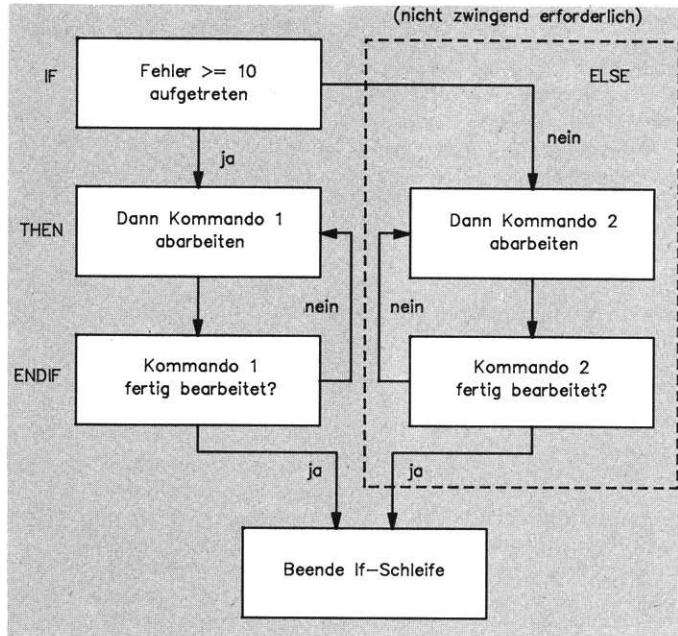


Bild 5. Wenn in der Batch-Datei ein Fehler ≥ 10 aufgetreten ist, wird Kommando 1 abgearbeitet (IF-THEN)

anderen Befehlsfolge, die hinter ELSE beginnt und durch ENDIF abgeschlossen wird.

Um nun die einzelnen Ereignisse genauer zu spezifizieren, werden hinter IF verschiedene Bedingungen angegeben. Drei Schlüsselworte stehen für die Abfrage des aufgetretenen Fehlerwertes (siehe FAILAT). Diese drei Bedingungen sind WARN für einen Fehlerwert ≥ 5 , ERROR bei einem Fehlerlevel ≥ 10 und FAIL, wenn ein Wert ≥ 20 zurückgemeldet wurde (dies funktioniert aber nur, wenn der Fehlerlevel mit FAILAT auf die gewünschten Gegebenheiten angepaßt wurde). Ein Beispiel (siehe dazu auch Bild 5):

```
IF Error THEN (Kommando 1)
ELSE (Kommando 2)
ENDIF
```

oder

```
IF Error THEN (Kommando 1)
ENDIF
```

nicht erfüllter Bedingung gewünscht wird. Falls in diesem Fall normal weitergemacht werden soll, entfällt die ELSE-Zeile.

Es gibt aber noch weitere Bedingungen: Eine davon ist EXISTS. Bei Angabe dieses Schlüsselworts prüft der Amiga, ob auf einer Diskette eine bestimmte Datei vorhanden ist. Ein weiteres Beispiel (siehe auch Bild 6):

```
IF EXISTS Preferences
THEN (Kommando 1)
ELSE (Kommando 2)
ENDIF
```

Ist die Datei Preferences auf der aktuellen Diskette vorhanden, wird »(Kommando 1)« ausgeführt. Wenn nicht, arbeitet der Amiga »(Kommando 2)« ab. Auch hier gilt: Wenn nicht zwingend eine Reaktion auf ein Nichtvorhandensein von »Preferences« nötig ist, kann die ELSE-Zeile entfallen. Die dritte Bedingung ist EQ. Bei Angabe dieser Bedingung prüft AmigaDOS, ob zwei Dateien gleich sind (Bild 7):

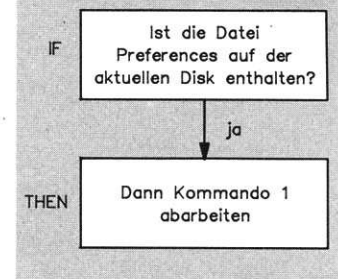


Bild 6. Um festzustellen, ob eine Datei auf der aktuellen Diskette enthalten ist, dient die IF-Bedingung EXISTS (den Rest des Bildes entnehmen Sie bitte Bild 5)

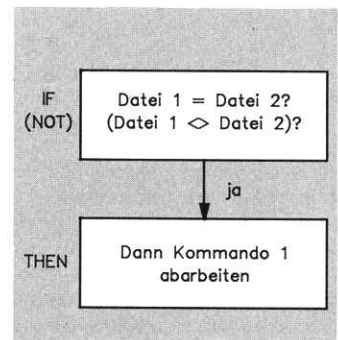


Bild 7. Durch eine gezielte IF-Abfrage läßt sich auch feststellen, ob zwei Dateien den gleichen Inhalt besitzen (den Rest des Bildes entnehmen Sie bitte Bild 5)

LAB dient dazu, eine Sprungmarke zu fixieren, zu der mit SKIP gesprungen werden kann. Ein Beispiel:

```
BINDDRIVERS
LOADWB
...
...
SKIP Marke1
...
...
LAB Marke1
...
```

Alle Befehle, die auf die LAB-Zeile folgen, werden nach dem Sprung zu der definierten Marke weiterbearbeitet. Es müssen nur noch die Bedingungen definiert werden, bei denen ein Sprung erfolgen soll (siehe Schema in Bild 8). Eine Möglichkeit dafür ist eine IF-Abfrage:

```
...
...
IF EXISTS Preferences
THEN setze Abarbeitung
fort
ELSE SKIP Marke1
ENDIF
...
...
ENDCLI > NIL:
LAB Marke1
ECHO "Datei nicht vorhanden"
ENDCLI > NIL:
```

Bei unserem Beispiel wird zuerst geprüft, ob die Datei »Preferences« auf dem aktuellen Laufwerk vorhanden ist. Ist dies der Fall, arbeitet der Amiga die Batch-Datei weiter ab. Ist die Datei aber nicht vorhanden, springt das Betriebssystem mit SKIP zu der mit LAB definierten Marke und führt die Befehle aus, die der LAB-Zeile folgen. Anschließend wird das CLI-Fenster geschlossen. Diese Kombination kann also eingesetzt werden, um mehr Struktur in eine Stapeldatei zu bringen. Allerdings ist es wohl in den seltensten Fällen nötig, diese Kombination zu verwenden, da sich ein solches Problem auch anders lösen läßt.

Ein weiterer, allerdings selten gebrauchter Befehl ist QUIT. Dieser Befehl dient zum Beenden von Stapeldateien und Kommandofolgen und kann beispielsweise anstelle von ENDCLI stehen. Im Unterschied zu ENDCLI wird bei QUIT das CLI-Fenster nicht geschlossen. Verändern wir doch das obige Beispiel, indem wir QUIT einbauen:

```
...
...
IF EXISTS Preferences
THEN setze Abarbeitung
fort
ELSE SKIP Marke1
ENDIF
...
...
QUIT
LAB Marke1
ECHO "Datei nicht vorhanden"
QUIT
```

Die Datei tut nichts anderes als oben beschrieben. Einziger Unterschied ist, daß nach der Abarbeitung das CLI-Fenster nicht geschlossen wird, was bei ENDCLI der Fall wäre.

Quit kann aber noch mehr: Wenn hinter dem Befehl ein Parameter angegeben wird, setzt Amiga-DOS diesen Wert nach Beendigung des Befehls automatisch als neues Fehler-niveau fest:

```
QUIT 40
```

Nach Beendigung des QUIT-Befehls würde das Fehler-niveau also den Wert 40 aufweisen. Der Befehl, der wie auch SKIP und LAB nur in Stapeldateien verwendet werden kann, ist sicher nicht dafür geeignet, dauernd eingesetzt zu werden. In speziellen Situationen kann er aber eine nützliche Anwendung finden.

Batchen mit Komfort

Mit Batch-Dateien läßt sich also eine Menge anfangen. Um aber ganz flexibel damit zu arbeiten, kann man beim Aufruf der Stapeldatei gleich einen Parameter übergeben, mit dem in der Befehlsdatei gearbeitet wird. Das könnte beispielsweise der Name eines Programms sein. Für das folgende Beispiel nehmen wir an, daß eine Befehlsfolge programmiert werden soll, die eine beliebige Datei mit einer bestimmten Folge von Befehlen »bearbeitet«. Wir möchten, daß im Beispiel die (beliebige) Datei zuerst einmal in die RAM-Disk kopiert wird, von dort auf Bildschirm und Drucker ausgegeben und auf eine in »df1« eingelegte Diskette kopiert wird. Dieser Vorgang ergibt zwar keinen rechten Sinn, doch dient er hervorragend zur Demonstration über den Aufbau einer flexiblen Stapeldatei.

Wir möchten erreichen, daß nur einmalig beim Aufruf der Batch-Datei der Name des Pro-

gramms übergeben wird, mit dem wir den Vorgang durchführen. Da in der Befehlsdatei der Name des Programms sicher öfter verlangt wird, ist es das einfachste, einen Platzhalter dafür einzusetzen. Genau dies erreicht man mit der Definition »KEY Name«.

Bei Angabe dieser Definition am Beginn der Datei wird dem Amiga mitgeteilt, daß hinter dem Namen der Batch-Datei noch ein weiterer Parameter folgt. Hinter dem Namen kann, durch einen Schrägstrich getrennt, stehen, ob der Parameter nötig ist (/A). Ein Beispiel:

```
.key name/a
...
LOADWB
```

Nehmen wir an, der Name der Batch-Datei wäre »Test« und der des zu bearbeitenden Programms »xyz«. Über die Tastatur müssen Sie zum Aufruf dieser Stapeldatei

```
EXECUTE Test xyz
```

eingeben. Ab jetzt würde überall in der Batch-Datei, wo <Name> steht, der Platzhalter »xyz« eingesetzt. Für unser Beispiel sieht die zu erstellende Stapeldatei so aus:

```
.KEY name/a
COPY <NAME> TO ram:
TYPE ram:<Name>
COPY ram:<Name> TO prt:
COPY ram:<Name> TO df1:
```

Anstelle von <Name> findet sich bei der Abarbeitung der mit dem Aufruf übergebene Name der zu bearbeitenden Datei. Es kann aber vorkommen, daß eine Befehlsfolge programmiert werden soll, bei der nicht unbedingt ein Name übergeben werden soll. Wir wollen annehmen, daß in diesem Fall ein vorher bestimmter Platzhalter eingesetzt wird. Die Definition »DEF Name2« bestimmt, daß, wenn kein Name beim Aufruf übergeben wird, ein definierter Platzhalter Verwendung findet. In diesem Fall muß das »/A« hinter .KEY entfallen, damit nicht zwingend ein Übergabeparameter angefordert wird:

```
.KEY name
.DEF name xxx
COPY <NAME> TO ram:
TYPE ram:<Name>
COPY ram:<Name> TO prt:
COPY ram:<Name> TO df1:
```

Gibt der Anwender jetzt beim Aufruf mit EXECUTE keinen Namen mehr an, setzt der Amiga anstelle von <Name> die mit .DEF bestimmte Zeichenkette »xxx« ein.

Speziell für den Amiga 1000 sind einige RAM-Erweiterungen zum internen oder externen Anschluß erhältlich, die teilweise den Nachteil aufweisen, daß sie nicht autokonfigurierend sind. Das bedeutet, daß sie vom Betriebssystem beim Initialisieren nicht automatisch erkannt werden. Um diese Speichererweiterungen dem Amiga doch noch zugänglich zu machen, gibt es den sowohl im Direkt- wie auch im Programmmodus einsetzbaren Befehl ADDMEM.

Das Zusatz-RAM wird aktiv

Er teilt dem Amiga mit, daß neuer Speicher einzubinden ist und an welcher Speicheradresse er beginnen soll. Dazu muß die Startadresse des neuen Speicherbereichs, also wo man ihn gerne haben will, und die Länge des RAM-Bereichs +1 angegeben werden. Zum Beispiel könnte der Aufruf für 1 MByte mehr Speicher, beginnend ab \$200000, so lauten:

```
ADDMEM 200000 300000
```

Nach Bestätigung des Befehls mit <RETURN> steht dem Anwender der neue RAM-Bereich sofort zur Verfügung.

Von Zeit zu Zeit kann es nötig werden, daß mehrere Einzeldateien (etwa Textfiles) zu einem einzigen File verbunden werden müssen. Damit die Sache nicht zu umständlich wird, stellt das CLI den Befehl JOIN zur Verfügung. Er erlaubt, maximal fünfzehn Einzeldateien zu einer neuen Gesamtdatei »zusammenschweißen«. Die einzige Bedingung beim Aufruf ist, daß die neue Datei einen anderen Namen als eine der Einzeldateien erhalten muß. Die zu verbindenden Dateien müssen in der Reihenfolge, wie sie verbunden werden sollen, dem Befehl folgen:

```
JOIN Text1 Text2 ASText3
```

Die Dateien »Text1« und »Text2« werden zu der neuen Datei »Text3« vereint.

Tasks auf Trab

Einer der wohl am häufigsten gebrauchten Befehle im CLI ist RUN. Dieses äußerst nützliche Programm erlaubt es, vom CLI aus ein Programm (einen Task/Prozeß) zu starten, das abläuft, ohne das CLI zu blockieren. Um dies verständlicher zu machen, geben Sie bitte folgendes ein:

```
Preferences <RETURN>
```

Sobald der Preferences-Screen erschienen ist, legen Sie ihn mittels des Hintergrundschalters oben rechts nach hinten, so daß das CLI wieder sichtbar ist. Versuchen Sie, im CLI irgend einen Befehl einzugeben. Sie werden feststellen, daß das CLI keine Eingaben mehr verarbeitet. Holen Sie Preferences wieder in den Vordergrund und beenden das Programm mit dem Schließesalter (Gadget) links oben.

Nun probieren wir folgenden Befehl aus:

```
RUN Preferences
<RETURN>
```

Wiederholen Sie die Prozedur und geben einen Befehl im CLI ein. Sie sehen, das eben

```
RUN COPY df0:Disk.info
TO df0:Test.info+
DELETE df0:Test.info+
ECHO "Fertig!"
```

Während der Anwender nach dem Aufruf ungestört mit dem CLI weiterarbeiten kann, kopiert das Betriebssystem die Datei »Disk.info« in das File »Test.info« und löscht die Kopie anschließend. Erst durch die Meldung »Fertig!« bekommt der Benutzer mit, daß der Vorgang abgeschlossen ist.

Ein selten benutzter Befehl des CLI ist »EDIT«. EDIT ist ein wenig komfortabler Zeileneditor, der nur gestattet, vorhandene, sequentiell aufgebaute Dateien zu bearbeiten. Es ist nicht möglich, von EDIT aus

Kommen wir nun zu den Befehlen des CLI, die etwas mit Zusatzgeräten wie beispielsweise Festplatten oder Zusatzlaufwerken zu tun haben.

Der erste dieser Spezialbefehle ist DJMOUNT. Dieser Befehl kann nur dann eingesetzt werden, wenn Sie einen Amiga 2000 mit PC-/AT-Karte besitzen, in den eine Festplatte eingebaut ist, die über einen Controller auf der MS-DOS-Seite gesteuert wird. Dieses Kommando sagt dem Amiga, daß er die für ihn eingerichtete Partitionierung auf der PC-Festplatte benutzen soll. Dazu eine kurze Erklärung: Bei der Partitionierung wird eine Festplatte (Hard-Disk) in einzelne Bereiche aufgeteilt.

```
ASSIGN devs: jh0:devs
ASSIGN sys: jh0:
ASSIGN c: jh0:c
...
```

In dieser Befehlsfolge wird zuerst der MS-DOS-Teil aktiviert. Der Amiga wartet dann so lange, bis der PC fertig gebootet hat und teilt Amiga-DOS anschließend mit, daß eine Festplatte (jh0:) zur Verfügung steht. Daraufhin werden die Standard-Dateiverzeichnisse so installiert, daß der Amiga von nun an alle Befehle, Fonts, Devices oder Libraries auf der Festplatte sucht.

Doch um die PC-Platte für den Amiga zugänglich zu machen, muß zuerst die Partition eingerichtet und die Platte formatiert werden. Dazu existieren die Befehle FDISK, ADISK (beide zu finden auf der MS-DOS-Systemdiskette) und DPFFORMAT. Das Einrichten geschieht folgendermaßen: — Zuerst mit BINDDRIVERS den PC/AT booten. Dann mit FDISK und ADISK die einzelnen Bereiche der Platte einrichten (beispielsweise 10 MByte für den PC und den Rest für den Amiga). — Anschließend vom CLI aus DJMOUNT eingeben. Die auftretende Fehlermeldung kann ignoriert werden.

Um die Amiga-Partition zu formatieren, ist folgendes einzutippen:

```
DPFORMAT DRIVE jh0:
NAME Name
```

Der Amiga formatiert und validiert seinen Bereich, der im Anschluß vom Amiga aus als »jhx:« benutzt werden kann.

Sollten Sie aber einen SCSI-Controller (etwa den von Commodore) mit der dazugehörigen Festplatte besitzen, muß zum Formatieren ein anderer Befehl verwendet werden: PREP. Nachdem die Platte mit MOUNT (näheres dazu später) aktiviert wurde, muß eine Routine aufgerufen werden, die dem Festplatten-Controller die Daten der Platte übermittelt.

Auch für SCSI ist gesorgt

Dies erledigt PREP. Für die erste Hard-Disk sieht der Aufruf so aus:

```
PREP Laufwerk:
```

Es erscheinen der Reihe nach verschiedene Abfragen, die die Platte betreffen. Zuerst will PREP den Festplattentyp wissen. Für eine gebräuchli-

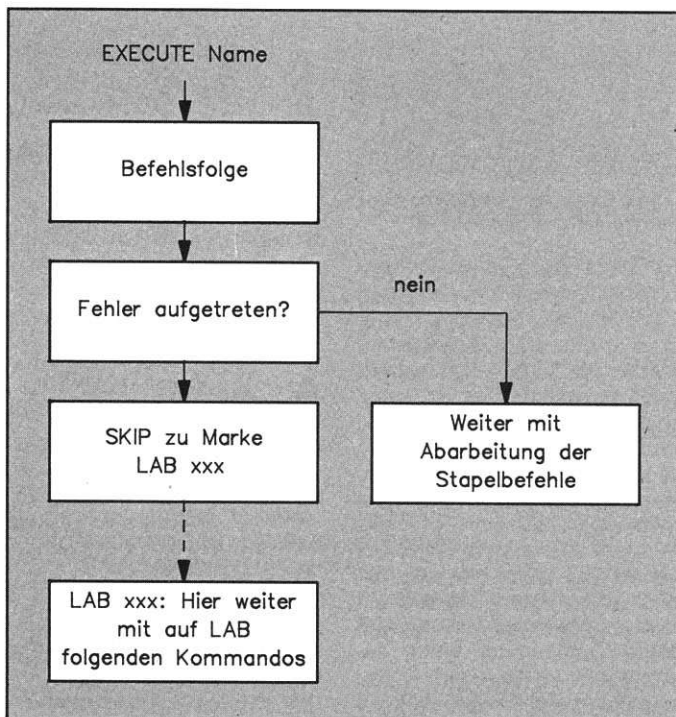


Bild 8. Die SKIP/LAB-Kombination funktioniert in etwa wie der von Basic her gewohnte Sprungbefehl GOTO

gestartete Preferences-Programm arbeitet im Hintergrund, während das CLI weiterhin Eingaben annimmt und verarbeitet. Der Befehl RUN startet also vom CLI aus ein Programm und gibt die Kontrolle anschließend zurück, so daß das Command-Line-Interface nicht blockiert ist. RUN ist ein äußerst nützlicher Befehl, der sowohl im Direkt- wie auch im Programmodus eingegeben werden kann und das Multitaskingsystem nicht an seiner Entfaltung hindert.

Aber RUN kann noch mehr: Mit einem einzigen Aufruf lassen sich auch mehrere Prozesse starten. Dazu muß nur zwischen jedem neuen Befehl ein »+« gesetzt werden.

neue Dateien zu erzeugen. Da dieser Editor außerdem nur mit Binärdateien etwas anfangen kann und auch sonst relativ selten eingesetzt wird, gehen wir in diesem Kurs nicht näher darauf ein. Hauptsächlich gedacht ist dieses zeilenorientiert arbeitende Programm wohl für Programmierer, die beispielsweise Maschinenprogramme oder ähnliches editieren wollen. Der nächste interessante CLI-Befehl ist VERSION. Der Befehl ist ohne Parameter anzugeben. Er zeigt die Versionsnummern der Kickstart- und der Workbench an. So können Sie jederzeit erfahren, ob Sie die aktuellen Versionen besitzen (derzeit Kickstart 33.180/Workbench 33.60).

Das Einrichten der Festplatte

Jeder Bereich kann von einem anderen Betriebssystem belegt werden. Angenommen, Sie besitzen eine 20-MByte-Festplatte mit zwei Platten und vier Köpfen, wobei jede Platte 612 Spuren hat (0 bis 611). Es wird gewünscht, daß jeweils 10 MByte für MS-DOS und 10 MByte für Amiga-DOS zur Verfügung stehen. Also wird der Festplatte (genauer: dem Festplatten-Controller) mitgeteilt, daß die Spuren 0 bis 305 für MS-DOS und die Spuren 306 bis 611 für den Amiga reserviert sind. Die Platte besitzt also zwei Partitions (Bild 9).

Fremde Partitions werden nicht erkannt. Das bedeutet, daß für den Amiga die ganze Platte nur 306 Spuren groß ist, genauso, wie der PC/AT auch nur 306 Spuren der Platte für sich beansprucht.

Der Befehl DJMOUNT darf aber erst aufgerufen werden, nachdem der PC/AT fertig gebootet hat und keine Plattenzugriffe innerhalb von AUTO-EXEC.BAT mehr macht. Gegebenenfalls muß mit WAIT genügend lange gewartet werden (etwa WAIT 40), da sonst das System abstürzen kann. Dies ist logisch, da nicht zwei Systeme gleichzeitig auf die Platte zugreifen können.

Nachdem der Befehl aufgerufen und ausgeführt wurde, steht dem A2000-Besitzer (mit Brückenkarte und PC-Festplatte) der für Amiga-DOS reservierte (partitionierte) Bereich zur Verfügung. In einer Startdatei könnte das so aussehen:

```
BINDDRIVERS
WAIT 40
DJMOUNT
```


che 20-MByte-ST506-Festplatte mit zwei Platten und vier Köpfen wählt man am besten die Seagate ST225. Bei der folgenden Abfrage möchte der Amiga wissen, ob die Platte automatisch nach einigen Sekunden Inaktivität die Köpfe parken soll. Die Antwort hängt von Ihrem Geschmack ab. Das Parken der Plattenköpfe schont Ihre Festplatte, wenn bei der Arbeit mit dem Amiga plötzlich der Strom abgeschaltet wird.

Sollten Sie die Hard-Disk partitionieren wollen (geschieht in einem Eintrag in der MountList), ist anschließend anzugeben, bis wohin die erste Partition reichen soll. Die nächste Frage bezieht sich darauf, wieviel Pufferspeicher der Platte zugestanden wird (näheres darüber später bei ADDBUFFERS).

Bevor die Vorbereitungen abgeschlossen werden, will Amiga-DOS noch wissen, ob defekte Sektoren der Hard-Disk markiert werden sollen. Nach Beantwortung dieser Frage wird die Platte zur Formatierung vorbereitet.

Formatiert wird die Hard-Disk anschließend wie gewohnt mit dem bekannten CLI-FORMAT-Befehl.

Wie bereits erwähnt, muß die an SCSI-Controller hängende Platte in das System eingebunden werden. Dies erledigt MOUNT. Insofern in der MountList (nähere Informationen darüber erhalten Sie im neuen Amiga-DOS-Handbuch und im Manual des SCSI-Controllers) die Platte eingetragen ist, genügt der Befehl

MOUNT dh0:

oder richtiger

MOUNT res0:

damit die Hard-Disk sofort dem Amiga als »dh0:« zur Verfügung steht.

Ebenfalls kurz erwähnt wurde ADDBUFFERS. Dieser hilfreiche Befehl bringt dem Anwender große Vorteile:

Normalerweise reserviert Amiga-DOS beim Einschalten oder einem Reset automatisch etwa 17 KByte Speicher für jedes angeschlossene Laufwerk (Sektor-Cache-Speicher). Vergrößert der Anwender nun diesen Pufferspeicher, können in diesem Puffer mehr Daten der Floppystation stehen, es muß also nicht so oft auf die Diskette zugegriffen werden. Damit wird der Datentransport etwas schneller. Um das zu erreichen, muß ADDBUFFERS zusammen mit einem Parameter eingegeben werden, der die

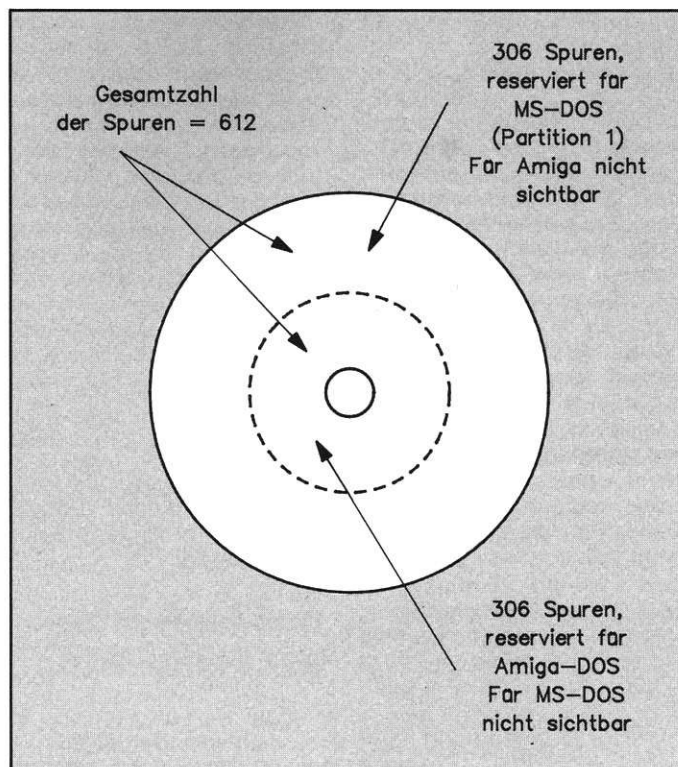


Bild 9. Diese Grafik verdeutlicht, wie eine Festplatte in Partitions, das heißt einzelne Teilbereiche, aufgeteilt ist

Größe des zu reservierenden Puffers angibt:

ADDBUFFERS df1: 40

reserviert 40 zusätzliche Pufferspeicher (jeder Pufferspeicher hat etwa eine Größe von 500 Byte). Größer sollte der Pufferspeicher aber nicht gemacht werden, da die Zugriffszeitvorteile ab einer bestimmten Puffergröße nicht mehr so groß sind und unnötig Speicher verbraucht wird.

Ein 5¼-Zoll-Laufwerk erkannt

Besitzen Sie ein 5¼-Zoll-Diskettenlaufwerk, das nicht automatisch einen Diskettenwechsel erkennt? Dann hilft DISKCHANGE weiter. Mit diesem Befehl teilen Sie Amiga-DOS mit, daß im 5¼-Zoll-Laufwerk die Diskette gewechselt wurde. Angenommen, das Laufwerk ist »df2:«, dann lautet der komplette Befehl:

DISKCHANGE df2:

Wenn im Amiga-Modus nicht mit 5¼-Zoll-Diskettenlaufwerken gearbeitet wird, kann dieser Befehl getrost gelöscht oder vergessen werden.

Folgende Situation ist jedem Amiga-Besitzer bekannt: Sie geben einen Befehl ein, der mit einer lapidaren Fehlermeldung abgebrochen wird. Meist sind aber diese Fehlermeldungen nicht gerade aussagekräftig.

Wenn Sie etwas mehr darüber erfahren möchten, geben Sie WHY ein. Der Amiga sagt Ihnen dann etwas detaillierter, warum die Ausführung des Befehls fehlgeschlagen ist.

Möchten Sie gerne wissen, ob eine bestimmte Zeichenkette in einer Datei enthalten ist, oder wissen Sie nicht mehr, wie oft in einer Datei ein Begriff vorkommt? Dann hilft SEARCH weiter. Der Befehl durchsucht das angegebene Verzeichnis beziehungsweise die angegebene Datei nach einer bestimmten, vorgegebenen Zeichenkette. Als Beispiel soll die Startdatei im aktuellen Laufwerk nach dem Befehl RUN durchsucht werden:

SEARCH startup-sequence
"RUN"

SEARCH durchsucht nun die Startdatei und zeigt alle Zeilen mit Zeilennummern und deren Inhalt an, in denen der Befehl RUN vorkommt. Wenn Sie nicht genau wissen, in welcher Datei der Suchtext steht, geben Sie einfach als Zusatzparameter ALL an. Suchen wir doch im gedachten Verzeichnis »Texte« nach dem Namen Müller:

SEARCH Texte "Müller"
ALL

Selbstverständlich können in den Dateinamen auch Joker, Platzhalter und Wildcards eingesetzt werden. Drei weitere Beispiele:

SEARCH startup-sequence
"r"

SEARCH df1: "ma" ALL

SEARCH jh0:tex#?

"Brief"

Im ersten Fall wird in der Startup-Sequence nach allen Begriffen gesucht, die den Buchstaben »r« enthalten. Die entsprechenden Zeilen werden dann aufgelistet.

Im zweiten Beispiel sucht der Amiga auf der Diskette im Laufwerk df1, und zwar in allen Dateien (Parameter »ALL«). Der Suchbegriff enthält die Buchstaben »ma«.

Im letzten Beispiel wird auf der Festplatte mit der Kennung jh0 in allen Verzeichnissen mit dem Namensanfang »tex« nach dem Begriff »Brief« gesucht.

Im gesuchten Begriff sind keine Joker zugelassen. Es wird unabhängig davon gesucht, ob die Zeichenkette allein steht oder nicht.

Zu beachten ist, daß SEARCH nicht zwischen Groß- und Kleinschreibung unterscheidet. Es ist also egal, ob der Suchtext Müller groß- oder kleingeschrieben ist.

SORTieren Sie ASCII-Dateien

Nützlich ist auch der Befehl SORT. Angenommen, Sie möchten eine ASCII-Datei, die vielleicht Namen oder ähnliches enthält, alphabetisch sortieren. Dazu können Sie natürlich auch den ED aufrufen und mit seinen Hilfsfunktionen umständlich den Text neu sortieren. Oder Sie verwenden SORT. Der Befehl sortiert eine Textdatei in aufsteigender alphabetischer Ordnung. Dabei werden nur komplette Zeilen, die mit Return abgeschlossen sind, berücksichtigt. Außerdem muß die komplette Datei auf einmal in den Speicher passen. SORT unterscheidet nicht zwischen Groß- und Kleinschreibung.

Einen Haken hat SORT: Es muß genügend Platz auf dem Stapelspeicher vorhanden sein, in dem SORT Daten zwischenspeichern kann. Oftmals reichen aber die 4000 Byte, die normalerweise jedem CLI-Befehl zur Verfügung stehen, nicht aus. Damit der Amiga beim Sortieren nicht abstürzt, ist es zweckmäßig, den Stapelspeicher mit dem STACK-Befehl auf etwa 10000 bis 15000 Byte zu vergrößern.

Beim SORT-Befehl ist anzugeben, welche Datei sortiert

werden (Quelldatei) und wie die sortierte Datei heißen soll (Zieldatei). SORT läßt die alte Datei unberührt und erstellt eine neue, sortierte Datei. Ein Beispiel: Wir möchten die Textdatei »Test« alphabetisch in aufsteigender Ordnung sortieren. Die neue Datei soll den Namen »Test.sort« erhalten:

```
SORT test TO test.sort
```

Wenn nicht gewünscht wird, daß ab der ersten Spalte einer jeden Zeile sortiert wird, läßt sich ein Zusatzparameter angeben, der festlegt, in welcher Spalte das erste zu sortierende Zeichen steht:

```
SORT test TO test.sort  
COLSTART 8
```

Nach dieser Änderung wird erst ab dem achten Buchstaben jeder mit RETURN abgeschlossenen Zeile sortiert.

Ergänzend wieder ein kurzer, leicht verständlicher Befehl, der äußerst hilfreich ist. Es handelt sich dabei um NEWCLI. Angenommen, Sie möchten parallel zwei verschiedene CLI-Tätigkeiten ausführen oder benötigen mehrere offene CLI-Fenster (maximal 20). Geben Sie NEWCLI ein. Sofort besitzen Sie ein zweites CLI-Window, in dem genau wie im ersten gearbeitet werden kann. Der Prompt »2>« zeigt Ihnen die Nummer des geöffneten Fensters. So läßt sich beispielsweise ein Text in einem Window ausgeben, während im anderen ein ähnlicher Text steht. Die zwei Texte lassen sich direkt vergleichen. Ein größerer Vorteil ist es aber, daß, falls ein Programm, das vom ersten CLI aus aufgerufen wurde, abstürzt, jederzeit ein weiteres CLI offen ist, in dem weitergearbeitet werden kann. Wie auch immer Sie die Möglichkeit, mehrere CLI-Fenster offen haben zu können, einsetzen, nützlich kann es sicher von Zeit zu Zeit sein.

Befehle neuer Versionen

Auf neueren Workbench-Versionen sind einige Befehle dazugekommen, die teilweise recht nützliche Funktionen erfüllen und die Befehlsvielfalt des mächtigen Instruments CLI erweitern. So zum Beispiel SETDATE. Dieser Befehl dient dazu, nachträglich das Erstellungsdatum und die Erstellungsuhrzeit einer Disketten-datei zu ändern. (Jede Datei auf Diskette oder im Speicher enthält zusätzliche Daten dar-

über, wann und um welche Uhrzeit das File erstellt wurde. Dies dient dazu, den Überblick darüber zu behalten, welche etwaigen Programmversionen älter oder neuer sind.) Angenommen, Sie besitzen ein paar Programme, bei denen Daten eingetragen sind, die nicht stimmen (etwa, weil bisher keine Echtzeituhr vorhanden war). Diese Daten sollen nun aktualisiert, also dem neuesten Stand angepaßt werden. Im Normalfall ist dies ohne einen Diskettenmonitor nicht möglich, da die Zeitdaten in bestimmten Diskettenblöcken stehen, die so ohne weiteres nicht beeinflußt werden können. SETDATE behebt dieses Manko. Der Befehl erlaubt die jederzeitige Änderung eines Erstellungsdatums und der Erstellungsuhrzeit (die Uhrzeit kann beispielsweise mit LIST angezeigt werden).

Fehler werden entschlüsselt

Ein Beispiel: Die Datei »Preferences« wurde mit einem falschen Datum gespeichert. Nachträglich soll das Programm auf den 27.11.1987, 12 Uhr datiert werden:

```
SETDATE Preferences  
27-nov-87. 12:00
```

Bei dem Befehl ist zu beachten, daß die Monate mit drei Buchstaben angegeben werden und die Schreibweise der Monate englisch ist (also MAY und nicht MAI).

Sicher kennen auch Sie diesen Effekt: Bei irgendeiner Dateioperation, sei es nun laden, speichern oder löschen, ist ein Fehler aufgetreten (etwa Datei nicht vorhanden) und das Betriebssystem gibt in der Menüzelle der Workbench-Oberfläche einen Fehlercode aus, der aus einer Zahl besteht. Doch was bedeuten diese Zahlen? Nicht immer ist das AmigaDOS-Handbuch griffbereit, aus dem man die Fehlertexte ablesen kann. Hier hilft FAULT weiter. Dieser Befehl fordert den Amiga auf, den Klartext zu der jeweiligen Fehlermeldung auszugeben (jedem Fehlercode ist auch ein Fehlertext zugeordnet). Ein Beispiel: Beim Öffnen einer Datei, sei es vom CLI oder von der Workbench aus, ist ein Fehler aufgetreten, das Programm wird nicht ausgeführt. In der Menüzelle sehen wir, daß ein Fehler mit dem Code 205 aufgetreten ist. Wir möchten erfahren, welche Bedeutung der Fehlercode 205 hat, also geben wir ein:

```
FAULT 205
```

Der Amiga gibt anschließend den Klartext des Fehlers aus: »Fault 205: object not found« — Datei nicht gefunden.

Sie können bis zu zehn Fehlercodes in einer Zeile angeben. Als Ergebnis erhalten Sie die Texte der Reihe nach aufgelistet. Ein Beispiel:

```
FAULT 103 202 203 204  
205 213
```

Als Ergebnis erscheint:
Fault 103: insufficient free store

Fault 202: object in use
Fault 203: object already exists

Fault 204: directory not found
Fault 205: object not found
Fault 213: disk not validated

Dieser Befehl ist sicher eine nützliche Hilfe, wenn es darum geht, Fehlermeldungen zu verstehen. Wie diese einzelnen Fehler nun zu beheben sind, können Sie einschlägiger Fachliteratur oder dem AmigaDOS-Handbuch entnehmen. Um den Kurs nicht unnötig zu verlängern, verzichten wir an dieser Stelle auf das Thema »Fehlerbeseitigung«.

Erste Hilfe

Der Amiga ist in Sachen Disketten leider sehr empfindlich. Wie schnell passiert es doch, daß man eine Diskette aus einem Laufwerk nimmt, bevor der letzte Zugriff abgeschlossen ist. Schon ist es passiert: Die Diskette ist »geschädigt«, der Inhalt unrettbar (?) verloren. Dies kann sich beispielsweise dadurch äußern, daß der Amiga beanstandet, daß die Diskette nicht validiert, also nicht gültig ist oder daß sie nicht mehr gelesen werden kann. Um die Daten (so gut es geht) wieder zurückzugewinnen zu können, stellt das CLI den Befehl DISKDOCTOR zur Verfügung. Er hilft, Ordnung in die Datenstrukturen zu bringen und die Daten zu retten. Allerdings restauriert DISKDOCTOR nur die Dateien, die noch genügend Informationen besitzen, die zur Wiederherstellung dienen. Das bedeutet, daß defekte Dateien verlorengehen.

Im Anschluß an die Behandlung durch DISKDOCTOR sollte die gesamte Diskette kopiert werden, um mögliche Fehler auf der ursprünglichen Diskette auszuschließen. Der Aufruf von DISKDOCTOR erfolgt beispielsweise für eine in »df1« eingelegte, geschädigte Diskette folgendermaßen:

```
DISKDOCTOR df1:
```

Nachdem die anschließende Sicherheitsabfrage mit <RETURN> bestätigt wurde, beginnt Amiga-DOS, die Struktur der Diskette einzulesen und wieder zu restaurieren. Dieser Befehl sollte aber als die letzte Möglichkeit angesehen werden, defekte Disketten zu retten, da unter Umständen Daten verlorengehen.

Wie Sie sicher wissen, ist der Amiga ein Multitasking-Computer. Das bedeutet, daß er scheinbar mehrere Pro-

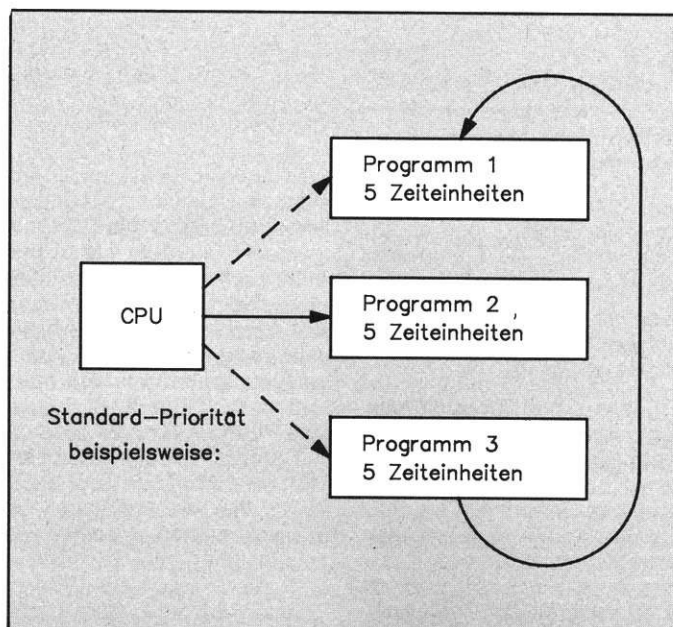


Bild 10. Im Normalfall erhält jeder Prozeß vom System prozentual die gleiche Prozessor-Rechenzeit zugeteilt

gramme gleichzeitig bearbeiten kann. Dies ist aber nur bedingt richtig. Genaugenommen wird jedes Programm eine bestimmte Zeit abgearbeitet und dann zum nächsten gesprungen (Bild 10). Das bedeutet, daß die Rechenzeit aufgeteilt wird. Je mehr Programme also »parallel« arbeiten, desto langsamer wird jeder einzelne Prozeß. Eine nützliche Mög-

Gerechte Verteilung

lichkeit ist es, daß der Benutzer jedem Programm eine bestimmte Rechenleistung zuteilen kann. Es ist sicher nicht sinnvoll, einem Prozeß wie etwa dem Hintergrunddrucken genausoviel Rechenzeit zur Verfügung zu stellen wie einem sehr rechenintensiven Programm. Deshalb kann mit CHANGETASKPRI die Re-

so gebremst werden, daß das System nicht mehr sinnvoll funktioniert. Im Normalfall sind Werte zwischen -10 und 10 noch vertretbar. Ein Beispiel:

```
CHANGETASKPRI 10
```

Ab jetzt werden alle Prozesse, die der Anwender von diesem CLI aus startet, bevorzugt behandelt, also schneller abgearbeitet. Dieser Befehl ist also durchaus sinnvoll, wenn es darum geht, den Amiga effektiver arbeiten zu lassen.

Um »die Kontrolle« über das System zu behalten, kann auch der nächste Befehl dienen: STATUS. Er dient dazu, den Benutzer darüber zu informieren, welche vom CLI aus aktivierten Prozesse gerade im Hintergrund arbeiten. Wenn STATUS ohne weitere Parameter eingegeben wird, zeigt der Amiga die Nummern der CLI-Tasks sowie die Namen, mit denen sie aktiviert wurden. Ange-

entnehmen Sie bitte dem »Technischen Amiga-DOS-Handbuch«. Eine Erklärung an dieser Stelle würde unnötigerweise zu viel Platz verschlingen. Dieser Parameter dient wohl hauptsächlich dazu, zu erfahren, mit welcher Rechenleistung, also welcher Priorität, jeder CLI-Task arbeitet.

Der zweite Parameter, der erlaubt ist, heißt CLI oder ALL. Beide Zusätze haben dieselbe Bedeutung. Laut Handbuch zeigen diese Zusätze die Sektionsnamen eines geladenen Prozesses an. Jedoch konnte dieses Resultat bei uns bisher noch nicht beobachtet werden, so daß wohl dieser Parameter entfallen kann.

Dagegen ist der dritte und letzte Formalparameter sinnvoll: FULL. Wie der Name schon sagt, gestattet FULL, daß zu jedem Task alle möglichen Angaben ausgegeben, also daß TCB und CLI/ALL zusammen dargestellt werden. Für das obige Beispiel sähe das Resultat so aus:

```
Task 1: stk 3200, gv
150, pri 0, Loaded as
command: status
```

Soll nur ein bestimmter Prozeß angezeigt werden, ist nach STATUS die Nummer des Tasks anzugeben, also beispielsweise STATUS 1 FULL.

Zusatz für PC-Karte

Nur für PC-/AT-Karten-Besitzer ist der letzte, nur auf neuesten Workbench-Versionen enthaltene Befehl SETPREFS. Der Befehl informiert Sie darüber, mit welchen Grundeinstellungen die PC-Karte konfiguriert ist. Sie erfahren, an welchen Stellen im Speicher sich die einzelnen Emulationen und Schnittstellen befinden und welche Grafikmodi aktiviert sind. Da dies aber genauso gut auch über die PC-Seite gemacht werden kann und wohl auch nicht für jeden interessant ist, wird der Befehl hier nur kurz erwähnt. Sollten Sie nähere Informationen dazu benötigen, ziehen Sie das neue Amiga-DOS-Handbuch und das PC-Handbuch zu Rate.

Damit ist der CLI-Kurs beendet. Sie sollten nun über genügend Wissen verfügen, um sinnvoll mit den einzelnen Befehlen umgehen zu können und auch Ihre eigenen Batch-Dateien zu schreiben, damit der Amiga in Zukunft effektiver arbeiten kann.

(Dieter Maier/Reiner Kunz/rs)

Gestalten Sie Ihr Sonderheft mit

Nach langem Warten halten Sie in diesem Augenblick das erste Amiga-Sonderheft in den Händen. Die Arbeit an diesem Heft hat allen Redakteure sehr viel Spaß bereitet. Wir haben natürlich lange und intensiv überlegt, welche Themen Sie, die Leser, besonders interessieren. Überzeugt Sie das Ergebnis? Sind Sie mit der Qualität und Länge der Listings einverstanden? Oder sollen wir längere Listings lieber nur auf der Programmservice-Diskette veröffentlichen?

Schreiben Sie uns Ihre Meinung zu diesem ersten Sonderheft.

Wir versuchen natürlich auch weiterhin, die interessantesten Themen ausführlich im Heft zu behandeln. Im Sonderheft stehen mehr Seiten für einen Schwerpunkt zur Verfügung, als das im Amiga-Magazin der Fall ist. Bei der Auswahl der Schwerpunkte können Sie mitwirken. Welche Themen wünschen Sie für die folgenden Hefte? Sind Sie an Kursen interessiert, und wenn, zu welchen Bereichen? Oder wünschen Sie mehr und längere Listings? Sind Spiele das Thema, das Sie am meisten reizt, oder fehlen Ihnen Anwendungslistings? Schreiben Sie uns Ihre Wünsche, am besten auf eine Postkarte, und adressieren Sie diese an:

Markt & Technik Verlag AG
Redaktion Sonderhefte
»Themen Amiga«
Hans-Pinsel-Str. 2
8013 Haar b. München

Ihr

Ralf Sablowski

Ralf Sablowski
(Redakteur)

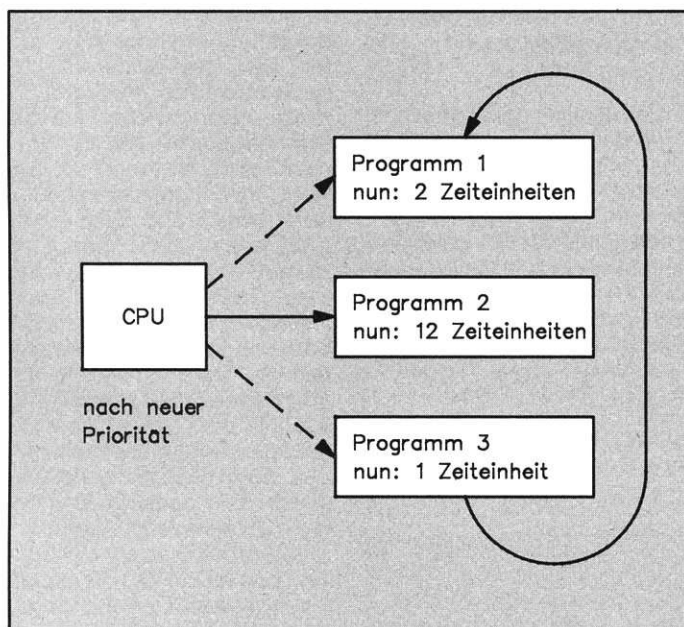


Bild 11. Nach einer Änderung erhalten rechenintensivere Programme mehr Prozessor-Rechenzeit zugeteilt

chenzeit eingestellt werden (Bild 11). Die neue Rechenzeit gilt allerdings nur für die Programme, die vom gleichen CLI aus aufgerufen werden, in dem auch CHANGETASKPRI verwendet wurde. Alle anderen Programme, Tasks und Prozesse, die direkt von der Workbench oder einem anderen CLI-Fenster aus aktiviert werden, laufen mit der normalen Rechengeschwindigkeit ab.

Für den Befehl können Werte zwischen -128 und 127 angegeben werden. Allerdings sollten keine zu großen Werte eingegeben werden, da sonst möglicherweise zum Betrieb des Amiga wichtige Prozesse

nommen, Sie haben ein offenes CLI-Fenster, in dem Sie den Befehl STATUS eingeben. Als Meldung wird erscheinen:

```
Task 1: Loaded as
command: Status
```

Aber STATUS erlaubt die Eingabe von Zusatzparametern. Eines dieser Schlüsselwörter ist TCB. Wenn Sie diesen Zusatz anwenden, erscheinen Informationen über die Stackgröße zu jedem Task, der Priorität, mit der der Task bearbeitet wird sowie über die Größe die Globalvektoren. Eine Erklärung des letzten Begriffes, der wohl nur für Programmierer interessant ist,

Guru verliert seinen Schrecken

Wenn der gefürchtete Requester »Software Error« erscheint, ist der GURU nicht mehr weit. Wohl dem, der noch ein weiteres CLI-Fenster offen, oder wenigstens die Workbench geladen hat. Bei dem Requester ist nämlich der Computer noch gar nicht abgestürzt. Erst wenn Sie »Cancel« anklicken, passiert's. Ignorieren Sie den Requester einfach und arbeiten Sie in dem CLI-Fenster weiter. So können Sie in Ruhe wertvolle Dateien von der RAM-Disk auf einer Diskette retten. (Franz Dimbeck/rs)

Checkie 42 meldet sich

Mit dem Checksummer »Checkie 42« lassen sich Listings einfacher eingeben, wenn der Amiga bei einer falsch eingetippten Zeile einen Warnton erzeugt. Die Zeile 17 des Checksummers sollte folgendermaßen geändert werden:

```
17 IF FSumme = falsch THEN BEEP: GOTO Wiederholung
```

Diese Funktion ist nützlich, wenn der Leser beim Eintippen gar nicht mehr auf den Bildschirm schaut, sondern sich auf die Tastatur und das Listing konzentriert. (Jürgen Schulze/rs)

Datei oder nicht Datei?

Sie können auch ohne die Bibliotheksfunktionen prüfen, ob eine Datei vorhanden ist. Schreiben Sie einfach:

```
OPEN File$ FOR APPEND AS 1
L=LOF(1)
CLOSE 1
IF L=0 THEN
  PRINT "Sorry Kid"
  KILL FILE$
  KILL FILE$+".info"
END
END IF
```

War die Datei nicht vorhanden, ist die Länge gleich Null. Es wurde aber in diesem Fall die Datei und eine Info-Datei erzeugt. Diese löschen Sie mit KILL. Diese Lösung nach einer Datei hat den Vorteil, daß Sie keine bmap-Dateien brauchen. (Eike Cornelius/rs)

Begrüßung durch den AMIGA

Der Amiga ist in der Lage, jeden Anwender nach Einlegen der Workbench-Diskette individuell zu begrüßen. Sie brauchen nur die »Startup-Sequence« des Amiga entsprechend zu verändern. Mit Hilfe des Befehls

```
ECHO "text"
```

können Sie Begrüßungstexte auf dem Bildschirm ausgeben, oder Sie lassen den Amiga mit SAY ein paar nette Worte sprechen. Folgende Schritte sind nötig, um den Amiga bei jedem Systemstart zum Sprechen zu bringen:

1. Legen Sie auf der Workbench-Diskette mit

```
COPY * to df0:name
```

eine Textdatei an, in die Sie über die Tastatur die gewünschte Begrüßung schreiben. Beenden Sie die Eingabe mit Hilfe der Tastenkombination <CTRL \>.

2. Starten Sie den Editor mit

```
ED df0:s/startup-sequence
```

um die Startup-Sequence zu ändern.

3. Schreiben Sie eine Zeile vor LOADWB den Befehl

```
SAY -x name
```

4. Speichern Sie die neue Startup-Sequence mit <ESC x>. Wenn Sie nun Ihren Amiga neu starten, werden Sie das Ergebnis Ihrer Arbeit hören. Sollte Ihnen die Begrüßung durch den Amiga einmal nicht mehr gefallen, so können Sie diese einfach variieren, indem Sie die entsprechende Textdatei mit dem Editor ändern.

(Christoph Huwe/rs)

Umstieg leichtgemacht

Viele Besitzer eines C 64 sind auf den Amiga umgestiegen. Die meisten wollen ihren C 64 behalten. Mit wenig Aufwand können die Aufsteiger ihren alten Computer auch noch am Amiga-Monitor verwenden. Auf der Rückseite des 1081- und auch des 1084-Monitors befinden sich zwei Cinch-Buchsen: Audio und CVBS

Verbinden Sie Pin 3 der Buchse des C64 zum Anschluß eines Monitors mit der Audio-Buchse. Pin 4 liefert das Farbsignal für den Anschluß an die CVBS-Buchse. Jetzt nur noch die Massen der beiden Cinch-Buchsen am Monitor mit Pin 2 der C64-Buchse verbinden und alles stimmt. Um das Bild des C64 auf dem Monitor zu sehen, schalten Sie diesen von RGB auf CVBS um. Der Schalter befindet sich unter der Klappe vorne am Bildschirm.

(Jochen Hauck/rs)

Die Hintertür für Banditen

Der Byte-Bandit ist ein Virus, der allen Amiga-Besitzern Kummer bereitet (siehe AMIGA 5/88, Seite 96 unten rechts). Immer wenn Ihr Amiga von dem Erreger befallen ist, färbt sich nach einiger Zeit der Bildschirm dunkel und der Computer versinkt in Lethargie. Hier gibt es aber einen einfachen Trick, um den Bildschirm wieder einzublenden. Drücken Sie nacheinander von links nach rechts die Tasten <ALT>, <Commode>, <entspricht <Amiga links>), <Space>, <Amiga> und <Alt>. Halten Sie dabei alle Tasten gedrückt. Und siehe da: der Bildschirm blendet sich wieder ein. Es handelt sich bei dieser Tastenkombination vermutlich um ein Hintertürchen, das sich die Programmierer offen gelassen haben, um nicht Opfer ihrer eigenen Seuche zu werden. Auf jeden Fall sollten Sie den Virus aus dem Boot-Block löschen.

(Sven Galanska/rs)

Tolle Tips für Einsteiger

Der Amiga ist ein leistungsfähiger, schneller Computer. Doch die Leistungsfähigkeit hat ihren Preis: Viele Vorteile bleiben gerade Anfängern verborgen. Unsere Tips und Tricks sollen Ihnen helfen, die ersten Klippen im Umgang mit dem Amiga erfolgreich zu umschiffen.

Was tun bei »Task held«?

Wer einen schwerwiegenden Programmierfehler gemacht und den Amiga dadurch zum Absturz gebracht hat, kennt die Meldung »Task held, finish all Disc-activity«. Was viele aber nicht wissen: Diese Meldung besagt, daß lediglich dieser eine Task angehalten wurde. Wenn Sie zu diesem Zeitpunkt noch einen weiteren Task laufen haben, zum Beispiel ein zweites CLI, können Sie mit diesem problemlos weiterarbeiten. Vor allem Einsteiger schalten bei dieser Meldung nämlich den Computer aus und wieder ein, da sie meinen, er wäre vollständig abgestürzt. (Teut Weidemann/rs)

Alles auf einen Schlag

Ein weiterer Tip nutzt den Einsatz der <Shift>-Taste bei der Auswahl von Icons aus. Immer wenn die <Shift>-Taste gedrückt wird, können Sie mit der Maus mehrere Objekte auf der Workbench gleichzeitig anwählen. Ein Beispiel wäre es, mehrere Programme gleichzeitig zu aktivieren oder wie oben gezeigt auf der Workbench zu verschieben beziehungsweise in andere Fenster zu kopieren.

Ein besonderer Trick ist die gleichzeitige Auswahl von Projekten

und Werkzeugen. Jedes Icon eines Projekts erhält in seiner Info-Datei einen Namen eines Programms, das beim Aktivieren des Programms automatisch geladen werden soll.

Ein Beispiel: Jedes Amiga-Basicprogramm erhält als »Default-Tool« den Namen »Amiga-Basic«. Häufig steht in diesem Namen aber gleichzeitig der Name eines bestimmten Verzeichnisses. Was nun, wenn Amiga-Basic sich in einem anderen Verzeichnis befindet. Dann erscheint nach dem Aktivieren des Programms die Meldung: »Error while opening Amiga-Basic«.

Der Weg dies zu umgehen, wäre den Namen in der Info-Datei zu ändern. Dies geht über das Menü »Info« der Workbench. Eine andere Möglichkeit ist, erst Amiga-Basic zu starten. Über »OPEN« kann der Programmierer dann sein Programm laden. Der dritte Weg ist praktischer: Klicken Sie bei gehaltener Shift-Taste erst das Icon des Programms und dann das Piktogramm von Amiga-Basic an. Dabei spielt es keine Rolle, ob es sich im selben Ordner oder irgendeinem anderen Fenster befindet.

(Dieter Bendrich/rs)

Please replace Workbench ...

Vielleicht haben Sie sich über die gut gemeinte Aufforderung auch schon geärgert, wenn Sie ein auf einer anderen Diskette gespeichertes Programm starten, aber beim besten Willen nicht das auf der Workbench befindliche Betriebssystem und zugleich das gewünschte Programm im Laufwerk haben können, weil Sie kein externes Laufwerk besitzen. Ein zweites Laufwerk ist daher eine der wichtigsten Erweiterungen, die man seiner Freundin (sprich: Amiga) gönnen sollte. Das billigste, schnellste und leiseste Laufwerk auf dem Markt ist die RAM-Disk. Sie belegt allerdings kostbaren Speicherplatz. Wie wird man eine RAM-Disk, wenn man sie nicht mehr benötigt, wieder los?

Praktisch wäre das Einrichten einer solchen Disk und beliebiges Entfernen dieser vom CLI aus. Mit den beiden folgenden Programmen läßt sich dies auch realisieren:

Legen Sie sich mit der Anweisung »MAKEDIR df0:ram« auf der Diskette ein Verzeichnis mit dem Namen »ram« an. Aktivieren Sie dann den Editor mit »ED df0:ram/on« und tippen die folgenden Zeilen ein:

```
makedir ram:c
copy :c/copy ram:c/copy
assign c: ram:c
copy ...
```

Welche und wieviel Betriebssystem-Befehle in das Verzeichnis »ram:c« kopiert werden, liegt bei Ihnen; auf jeden Fall sollten sich die Befehle DIR und CD darin befinden.

Mit <ESC x> verlassen Sie den Editor und haben soeben eine Batch-Datei erstellt. Mit der Anweisung EXECUTE ram/on werden nun die in der Datei befindlichen CLI-Befehle der Reihe nach ausgeführt. Mit dem Befehl INFO können Sie sich davon überzeugen, daß Ihnen nun zwei Laufwerke zur Verfügung stehen. Da der DIR-Befehl sich jetzt hoffentlich in der RAM-Disk befindet, können Sie sich ohne Probleme Inhaltsverzeichnisse beliebiger Disketten ausgeben lassen oder Programme aufrufen, die selbst kein Betriebssystem auf der Diskette enthalten. Mit einer weiteren Batch-Datei schalten sie die RAM-Disk wieder ab. Nennen wir Sie »ram/off« und geben nach dem Starten des Editors

```
ASSIGN c: df0:c
DELETE ram:c all quiet
```

ein. Nach dem Verlassen des Editors mit <ESC x> kann die RAM-Disk nun mit »Execute ram/off« wieder vollkommen entleert werden. Sie beansprucht dann nur noch etwa 1 KByte Ihres Speichers. Die RAM-Disk selbst läßt sich nur durch einen Neustart des Systems beseitigen.

(Wilhelm Holtmeier/rs)

Dr. Amiga repariert Disketten

Haben Sie auch schon mal versehentlich eine Diskette dem Laufwerk entnommen, während die rote Kontrollampe leuchtete?

Nicht selten ist diese Diskette danach unbrauchbar. Ein Zugriff darauf wird vom Amiga nur mit einem »read/write-error« quittiert. Aber noch ist nicht alles verloren. Geben Sie ein: »discdoctor Drive« (setzen Sie bitte für Drive das Laufwerk ein, in dem sich die defekte Diskette befindet). Der Doctor beschäftigt sich eine Weile

mit der Diskette und fordert dann den Anwender auf, die Dateien auf eine neue Diskette zu kopieren und danach die alte neu zu formatieren. Dann kann man nur hoffen, daß nicht allzuviel zerstört worden ist.

(Roland Keßler/rs)

Einfacher Schutz vor Viren

Wer sich durch eine Unachtsamkeit oder aus Unwissenheit einen Virus auf seine Diskette kopiert hat, kann diesem mit dem CLI-Befehl INSTALL zu Leibe rücken. Dieser Befehl überschreibt die beiden ersten Blocks auf der Diskette (Bootblock), in denen sich die meisten Viren versteckt halten. Im Bootblock befindet sich die Information für das Betriebssystem, ob eine Diskette eine Startdiskette ist, sowie ein Assemblerprogramm, das der Amiga beim Booten ausführt. INSTALL macht aus jeder formatierten Diskette eine normale Startdiskette. Der Befehl löscht also auch alle mit dem »BootBoy« erstellten Vorspanne. Jedoch seien Sie vorsichtig. Wenden Sie den Befehl INSTALL nicht wahllos auf allen Ihren Disketten an, um sich vorsorglich vor Computerviren zu schützen. Es gibt zahlreiche Programme, wie zum Beispiel »Archon«, die Ihre eigene Startroutine besitzen. Diese würden Sie unwiderruflich löschen. Das ganze Spiel wäre unbrauchbar.

(Ralf Dittmer/rs)

File-COPY mit einem Laufwerk

Mit diesem Utility können Sie mit nur einem Laufwerk komfortabel Ihre Files kopieren. Das Batch-File »FILECOPY« besteht nur aus CLI-Befehlen und arbeitet in der RAM-Disk:

.key file

```
ECHO "Bitte Source-Disk in Drive df0:"
ECHO "und ENTER drücken"
DATE >nil: ?
fail 10
cc df0:<file> ram:
IF NOT EXISTS <file>
    ECHO " Datei <file> existiert nicht"
    ECHO "Rufen Sie die Kopierroutine erneut auf"
QUIT 10
ENDIF
ECHO " Bitte Ziel-Disk in Drive df0: "
ECHO " und ENTER drücken"
DATE >nil: ?
cc ram:<file> df0:
ECHO "ready"
```

Zur Initialisierung der Kopierroutine werden die benötigten CLI-Befehle mit Hilfe des Batch-Files »CLIRAM« in die RAM-Disk kopiert:

```
COPY df0:c/COPY ram:cc
COPY df0:c/ECHO ram:
COPY df0:c/FAILAT ram:fail
COPY df0:c/IF ram:
COPY df0:c/ENDIF ram:
COPY df0:c/QUIT ram:
COPY df0:c/DIR ram:
COPY df0:c/DATE ram:
COPY df0:c/ED ram:
COPY df0:c/EXECUTE ram:ex
COPY df0:c/MAKEDIR ram:mdir
COPY df0:FILECOPY ram:
CD ram:
DIR
```

Gehen Sie wie folgt vor: Geben Sie mit dem Editor beide Batch-files ein und speichern diese auf Ihrer Work-Disk (Workbench-Kopie oder CLI-Diskette). Initialisieren Sie das Kopierprogramm mit dem Befehl:

```
EXECUTE CLIRAM
```

Zum Kopieren eines Files auf eine andere Diskette dient der Befehl:

```
ex FILECOPY Name
```

Das Programm fordert die benötigten Disketten automatisch an. Sie müssen nur die Disketten wechseln und mit < RETURN > quittieren. (Johannes Mayr/rs)

(Anmerkung der Redaktion: Das Batchfile speichert einige CLI-Befehle unter abgekürzten Namen in der RAM-Disk. Dies ist praktisch, erspart Tipparbeit, kann aber zu Verwechslungen führen. Grundsätzlich sollten Sie die CLI-Befehle nicht umbenennen.)

Grüße aus dem Herzen des Amiga

Das Betriebssystem des Amiga befindet sich im Kickstart-RAM (\$FC0000 - \$FFFFFF beziehungsweise 16580608 - 16777216). Basic-Programmierer können mit Hilfe des Befehls PEEK einen Blick in die Systemroutinen werfen. Schauen Sie sich in diesem Bereich um. An manchen Stellen ist auch ohne Disassembler etwas erkennbar. Mit dem folgenden Basic-Programm finden Sie eine besondere Nachricht:

```
REM Werte für Kickstart 1.1
Start = 16653596 : Ziel = 16653672
REM Werte für Kickstartversion 1.2
REM Start = 16649670 : Ziel = 16649747
CLS
WIDTH 67
PRINT
FOR i = Start TO Ziel
  PRINT CHR$(i);
NEXT i
```

(Rolfdieter Schiedrum/rs)

Ein neues Geheimnis

Wieder ist einer der versteckten Scherze des Amiga-Betriebssystems entdeckt:

Diesmal handelt es sich um einen Trick im Preference-Window: — Drücken Sie dort die Knöpfe der beiden abgebildeten Mäuse nacheinander; viermal von links nach rechts. — Jetzt betätigen Sie im Auswahlfenster »ChangePrinter« den Pfeil nach oben. Wenn Sie ganz oben angelangt sind, lesen Sie dort

Klar, daß wir das nicht abdrucken. (Siad Ardroumli/rs)

Noch mehr Geheimnisse

Durch Drücken mehrerer Tasten können Sie dem Amiga so manche geheimnisvolle Botschaft entlocken.

Eine interessante »Fingerübung« ist die folgende:

Drücken Sie zunächst die linke < Shift >- und < Alt >-Taste. Nicht loslassen. Als nächstes betätigen Sie die < Alt >- , < Shift >- und < Amiga >-Taste auf der rechten Seite. Zum guten Schluß müssen Sie noch eine beliebige sechste Taste drücken; schon versteckt sich die Maus in der rechten unteren Ecke des Bildschirms. Befindet sich die Maus allerdings vor der Fingerakrobatik in der Menüleiste, rutscht der gesamte Workbench-Screen nach unten. Der eigentliche Sinn dieser Kombinationen ist unklar, oder kennen Sie eine Anwendung? Was auch immer dahintersteckt, es verbergen sich bestimmt noch mehr solcher geheimer Tastenkombinationen im Amiga. Machen Sie sich auf die Suche. (Mario Himmel/rs)

Einen haben wir noch

Laden Sie die Workbench. Drücken Sie nun die Kombination der Tasten < Shift >, < Alt >, < Shift >, < Alt > und eine Funktionstaste. So, fünf Finger sind jetzt belegt. Vermutlich brauchen Sie auch beide Hände — wer hat schon so große Hände?. Versuchen Sie dennoch einmal, die Diskette aus dem Laufwerk zu ziehen. Jetzt erscheint oben im Screen eine Nachricht:

(Ein Schelm wer Arges dabei denkt)

Und zu guter Letzt noch ein Scherz — vermutlich. In den Preferences versteckt sich an der linken Seite des Fensters ein Gadget. Kennen Sie dessen Bedeutung? Wer sie bereits weiß, soll sich bitte melden. Wenn das wieder ein Scherz ist, heißt es dann bald »Einen, einen haben wir noch.....«. (Siad Ardroumli/rs)

Ein interessantes Projekt

Es gibt unterschiedliche Arten von Piktogrammen, unter anderem die »Tools« und die »Projects«. Tools sind Programme. Durch zweimaliges Anklicken werden sie gestartet. Zwei Beispiele hierfür sind die »Preferences« und die Uhr.

Projects dagegen sind Dateien, die von anderen Programmen erzeugt wurden. Sie können nicht eigenständig geladen werden. Sie sind abhängig von einem Tool. Jedes Basic-Programm stellt ein solches Projekt dar. Dennoch lassen sich die Projekte durch zweimaliges Anklicken aktivieren. In der Info-Datei des Piktogramms ist der Name des zugehörigen Tools vermerkt. Sie verändern ihn mit Hilfe des Menüpunktes »Info« auf der Workbench:

Angenommen, Sie haben mit einem Editor einen Text verfaßt und möchten, daß dieser beim Anklicken des Datei-Piktogramms auf dem Bildschirm erscheint. Kopieren Sie zunächst das Tool »More« von der Extras-Diskette in die RAM-Disk. Als Mutterprogramm tragen Sie »ram:more« in der Info-Datei Ihres Textes ein. Nun müssen Sie nur noch das Text-Icon anklicken.

Viele Editoren erzeugen zu den gespeicherten Dateien allerdings keine Piktogramme. Ein Beispiel ist »MicroEmacs« auf der Extras-Diskette. Doch der Workbench-Benutzer kann zu diesen Texten ein bereits existierendes Icon verwenden. Er darf sogar ein Icon wählen, dessen Gattung nicht dem geforderten Projekt-Typ entspricht:

Laden Sie in diesem Fall »IconEd« sowie das gewünschte Bild. Klicken Sie einen zweiten Rahmen an, in den Sie ein beliebiges Piktogramm laden, dessen Typ aber dem gewünschten entspricht — in diesem Fall ein Projekt. Löschen Sie den zweiten Kasten mit »Clear Image«, wählen Sie »Merge with Frame« und bewegen die Maus auf das erste Feld. Wenn Sie nun die Maus loslassen, befindet sich im aktivierten zweiten Quadrat das gewünschte Image. Dieses speichern Sie unter dem Namen des Textes als Info-Datei. (Ralph Benzinger/rs)

Wer hat den Amiga entwickelt?

Kennen Sie die Entwickler des Amiga. Interessenten können deren Namen ermitteln. Aktivieren Sie die Workbench. Drücken Sie gleichzeitig beide < Shift >-Tasten sowie beide < ALT >-Tasten — wenn Sie jetzt auch noch eine der 10 Funktionstasten betätigen können, erscheinen am oberen Bildschirmrand die gesuchten Namen. Die Liste der Namen reicht von — halt, schauen Sie am besten selbst nach. Besonders die Taste F10 ist recht amüsant. (Marco Trauner/rs)

Amerikanische Tastatur

Besitzer des Amiga 1000 kennen das Problem: Ein Programm benutzt den deutschen Zeichensatz und Sie haben nur eine amerikanische Tastatur. Wo liegt der Doppelpunkt, wo das Semikolon?

Entweder Sie lernen die deutsche Tastaturbelegung auswendig oder benutzen Tastaturaufkleber. Die dritte Möglichkeit ist mit wesentlich weniger Aufwand verbunden:

Drücken Sie das gewünschte Zeichen auf der amerikanischen Tastatur zusammen mit der < ALT >-Taste. (Marco Trauner/rs)

Gleiche Icons für RAM-Disk und Workbench

Wollen Sie für die Workbench und die RAM-Disk eigene Icons verwenden? Mit einem kleinen Kniff ist dies zu schaffen.

Sobald Sie die RAM-Disk mit dem Befehl »DIR ram:« initialisieren, erscheint immer das Standard-Icon auf der Workbench, auch wenn Sie für die Workbench-Diskette bereits ein eigenes Icon entworfen haben. Möchten Sie die RAM-Disk ebenfalls durch Ihr eigenes Icon darstellen, so erreichen Sie dies mit Hilfe der Initialisierung:

COPY disk.info to ram:

Workbench-Icon und RAM-Icon sind nun gleich. Soll die RAM-Disk ein ganz besonderes Icon besitzen, so müssen Sie dieses auf der Diskette speichern und die RAM-Disk durch den Befehl:

COPY spezialicon.info to ram:

aufrufen.

(Klaus Rupp/rs)

Selbstgestrickte Icons schützen

Mit Hilfe des CLI-Befehls PROTECT können Sie Ihre selbstgestellten Icons schützen. Dies ist vor allem für Basic-Programmierer interessant.

Wenn Sie für Ihr Programm mühsam ein eigenes Icon erstellt haben, dann das Programm ändern und erneut speichern, löscht Amiga-Basic brutal Ihr Icon. Deshalb sollten Sie das Icon beziehungsweise die zugehörige Info-Datei mit diesem Befehl sichern:

```
PROTECT Programmtitel.info R
```

Jetzt hat Amiga-Basic keine Chance mehr. Dieses Icon kann nur noch gelesen werden. (Rolfdieter Schiedrum/rs)

Schneller Wechsel der Druckertreiber

Dieser Tip richtet sich an diejenigen, die zwei oder mehr Drucker haben und denen der Druckertreibertausch mit »Preferences« zu umständlich ist. Es wurde für Kickstart 1.2 entwickelt und sollte außerdem nicht mit der Original-Workbench nachvollzogen werden. Um die Schritte zu verdeutlichen, verwenden wir im Beispiel die Treiber für den Epson JX-80 und den Diablo C-150.

1. Starten Sie den Amiga wie gewohnt (Amiga 1000 mit Kickstart) und Workbench. Sobald jetzt der Text »Workbench-Diskette. Version ...« erscheint, drücken Sie bitte <Ctrl-D>. Sie befinden sich nun im Fenster des CLI (Command Line Interface).

2. Geben Sie nun folgende Befehle ein und lösen Sie diese, wie alle anderen, mit <Return> aus: »system/SETMAP d« und »LOADWB«. Sie befinden sich nun auf der Workbench.

3. Lassen Sie sich mit »DIR devs/printers« die auf der Diskette befindlichen Treiber ausgeben und merken sich den Namen derjenigen, die Sie öfter verwenden wollen.

4. Kopieren Sie die Treiber in das Hauptdirectory. Beispiel: »COPY devs/printers/Epson__JX-80 TO Epson__JX-80«
»COPY devs/printers/Diablo__C-150 TO Diablo__C-150«

5. Nun schaffen Sie erst mal Platz auf der Diskette. Das können Sie mit der Anweisung »DIR devs/printers OPT I«. Es erscheinen die einzelnen Namen der Treiber und Sie können durch Eingabe des Wortes »del« hinter dem Fragezeichen alle Treiber, die Sie nicht mehr brauchen (normalerweise alle), löschen.

6. Leihen Sie sich mit »COPY clock.info to Epson__JX-80.info« das Icon der Uhr für den ersten Treiber. Führen Sie diese Anweisung für jeden der vorher kopierten Treiber durch.

7. Mit der Anweisung »ENDCLI« gelangen Sie nun wieder auf den Bildschirm der Workbench. Öffnen Sie die Workbench-Disk, wählen dort die Empty-Schublade und im Menü den Punkt Duplicate aus. Im Fenster erscheint eine Schublade namens »Copy of Empty«. Geben Sie ihr mit »Rename« aus dem Menü den Namen »Drucker«. Nach dem Öffnen dieser Schublade können Sie das Uhren-Icon mit den Treibernamen in die neue Schublade ziehen. Wenn Sie jetzt mal wieder den Treiber wechseln wollen, so reicht ein Anklicken in der Workbench, um ihn auszuwählen.

Wenn Sie sich mit dem Icon-Editor auskennen, können Sie auch das Icon ändern. (Peter Riedlberger/rs)

Alles Feste ins RAM

Mit der resetfesten RAM-Disk von ASDG können Sie sich vor unerwarteten Systemabstürzen schützen. Die ASDG-Disk ist unter anderem nützlich, um die CLI-Befehle darin zu sichern — besonders, wenn sie nur mit einem Laufwerk arbeiten. Trotz eines Resets stehen Ihnen dann sofort alle Befehle zur Verfügung, um im CLI zu arbeiten. Sie sparen Zeit. Die »Startup-Sequence« auf Ihrer Diskette sollte etwa so aussehen:

```
MOUNT VDO:
IF EXISTS VDO:c
ELSE
ECHO "Kopiere CLI das erste mal"
MAKEDIR VDO:c
COPY c/DIR| ECHO| LIST| COPY| ASSIGN| CD TO VDO:c
ASSIGN C: vd0:c
PATH df0:c ADD
```

```
ENDIF
PATH SYSTEM ADD
SETMAP d
LOADWB
```

Welche Befehle Sie in der resetfesten RAM-Disk unterbringen, richtet sich nach Ihren Anforderungen. Entscheidend ist, daß Sie beim nächsten Start nicht eine Ewigkeit warten müssen, bis der Amiga eine neue RAM-Disk mit den CLI-Befehlen einrichtet.

(Wolfgang Hettmer/rs)

Starkes Duo: CLI und Workbench

Geht es Ihnen auch so: Gerade sind Sie mitten drin im Arbeiten auf der Workbench, plötzlich benötigen Sie einen CLI-Befehl. Als erstes müssen Sie nun die System-Schublade öffnen, das CLI laden und auch noch das Fenster vergrößern. Das geht einfacher. Fügen Sie einen kurzen Befehl in der »Startup-Sequence« ein:

```
NEWCLI con: 540/200/60/50/CLI-Fenster
```

Binden Sie den Befehl vor dem Befehl LOADWB ein. Nach dem Booten finden Sie auf der Workbench unten rechts ein kleines CLI-Fenster. Es steht Ihnen bei Bedarf sofort zur Verfügung.

Wollen Sie die Startup-Sequence im großen Stil ändern, ist die Kenntnis der zahlreichen CLI-Befehle unerlässlich. Im Kurs ab Seite 114 finden Sie viele Hinweise zum Command-Line-Interface. Außerdem gibt's noch Tips zum CLI ab Seite 144.

(Peter Riedlberger/rs)

INSTALL mit nur einem Laufwerk

Gerade im Zusammenhang mit der Beseitigung von Viren aus dem Boot-Block hat der Befehl INSTALL Berühmtheit erlangt. Er weist allerdings eine Schwäche auf. Mit ihm können Sie nicht, wenn er von der Arbeitsdiskette geladen wird, eine Diskette im Laufwerk df0: installieren. Um den gewünschten Effekt zu erzielen, sind folgende Schritte nötig:

```
COPY c/INSTALL to ram:
```

```
CD ram:
```

```
INSTALL df0:
```

Der Amiga kopiert zunächst die Routine in die RAM-Disk. Sollte sich der INSTALL-Befehl nicht im C-Ordner befinden, müssen Sie ihn im Ordner »System« suchen:

```
DIR df0:system
```

Befindet sich der Befehl dort, geben Sie folgendes ein:

```
COPY System/INSTALL to ram:
```

Mit der kopierten Routine sind Sie dann jederzeit in der Lage, eine Diskette im internen Laufwerk zu bearbeiten, um sich beispielsweise einen Virus vom Hals zu schaffen.

(Boris Neumann/rs)

MERGE ohne lästige Frage

Wenn ein Basic-Programm mit dem Befehl CHAIN MERGE arbeitet, um Veränderungen im Programm hervorzurufen, tritt ein Problem auf: Bei mehrmaligem Aufruf von CHAIN MERGE erscheint die lästige Abfrage, ob das alte Programm vor der Veränderung gespeichert werden soll. Diese muß der Anwender immer erst beantworten und erst dann läuft das Programm weiter (siehe auch AMIGA 3/88, Seite 95). Verwenden Sie folgende Variante:

```
DELETE Neuer Bereich
SAVE "Programmname"
CHAIN MERGE "ram:Neue Befehlszeilen", 1000
```

In diesem Beispiel wurde die Zeile 1000 eingefügt, da als Anfangspunkt keine alphanumerische Sprungmarke erlaubt ist. Unter den Befehlszeilen, die eingefügt werden, muß sich in der ersten Zeile die Marke »NeuerBereich« finden. Diese wird dann ab Zeile 1000 eingefügt. Vor einem erneuten Aufruf von CHAIN MERGE wird dieser Bereich wieder gelöscht. Wenn das Programm sich dann quasi selbst speichert, entfällt die Sicherheitsabfrage.

(Eike Cornelius/rs)

Hallo Amiga-User

Der Amiga ist für viele Anwender der erste Computer mit Multitasking-Fähigkeiten. Dies erfordert ein Umdenken beim Programmieren. Einige Regeln sind zu beachten, um ein einwandfreies Multitasking zu gewährleisten:

1. Das Betriebssystem kann nicht den unsachgemäßen Zugriff auf die Routinen der Libraries kontrollieren. Ist etwas im Programm des Anwenders angemeldet oder zugewiesen worden, muß dies vor dem Beenden des Programms wieder rückgängig gemacht werden. Wird zum Beispiel Speicher reserviert oder eine Bibliothek geöffnet, muß die Library auch wieder geschlossen beziehungsweise der Speicher wieder freigegeben werden.

2. Fehlermeldungen von den Routinen des Betriebssystems müssen in jedem Fall kontrolliert werden. Sie dürfen nicht einfach davon ausgehen, daß zum Beispiel die mit »AllocMem« erfolgte Reservierung von Speicher immer erfolgreich verläuft. Mehrere Tasks können vielleicht RAM anfordern. Schon geht Ihr eigenes Programm leer aus. Auch bei einem Fehler ist zu beachten, daß das Programm alle bis dahin geöffneten Ressourcen vor dem Rücksprung wieder an das System zurückgibt.

3. Die Register D0,D1,A0,A1 sind »Scratch-Register«. Sie enthalten Parameter beim Aufruf von Systemroutinen. Sie dürfen in keinem Fall annehmen, daß die Register von den Routinen unberührt bleiben. Wenn wichtige Werte in den betreffenden Registern stehen, müssen Sie diese vor dem Aufruf einer System-Routine retten. Die übrigen Register sind im allgemeinen vor einem Überschreiben sicher.

4. Daten, die von den Custom-Chips (Copper, Blitter ...) verwendet werden, sind immer im Chip-RAM zu speichern. Die Chips können nur auf diesen Speicherbereich zugreifen. Oft kommt es vor, daß ein Programm auf einem Amiga mit 512 KByte läuft, jedoch seine Dienste in Zusammenarbeit mit einer Speichererweiterung verweigert. Da hat ein Programmierer nicht aufgepaßt. Im Zweifelsfall sollten Sie daher alle Daten immer ins Chip-RAM legen. Allerdings müssen Sie aufpassen: Bei mehreren gleichzeitig laufenden Programmen kann dieser Speicher schnell knapp werden.

5. Besitzer des Seka-Assemblers sollten sich bewußt sein, daß der Objektcode des Assemblers immer zuerst in das Fast-RAM geladen wird. Befinden sich im Objektcode Daten für die Custom-Chips, freut sich der Guru.

6. Damit ein Programm auch in naher Zukunft auf Rechnern mit 680xx-Prozessoren läuft, sollte der Befehl SR,<ea> durch die Funktion »GetCC()« aus der Exec-Library ersetzt werden.

7. Ein Programm sollte niemals Adreßzeiger mit nur 24 Bit verwenden. Auch wenn der 68000er im Amiga nur einen 24 Bit breiten Adreßbus besitzt. 32-Bit-Zeiger sichern Sie vor Überraschungen auf dem Hardware-Sektor.

8. Die einzige absolute Adresse beim Amiga ist \$00000004. Dort befindet sich der Zeiger auf die Exec-Library. Bei dieser absoluten Adresse sollte es ein guter Programmierer belassen.

9. Oft wird vergessen, daß sich mehrere gleichzeitig laufende Programme die Rechenzeit teilen müssen. Die Ausführung einer Routine kann zugunsten einer anderen unterbrochen werden. Dies kann zu Konflikten führen, wenn Programme das gleiche wollen, zum Beispiel Speicher allozieren oder auf ein Laufwerk zugreifen. Dies ist zwar selten, aber Sie müssen sich darauf einstellen: Fordert Ihr Programm einen Teil der Ressourcen des Amiga, müssen Sie dafür sorgen, daß kein anderer Task gleichzeitig dasselbe will. Dies erreichen Sie durch den Aufruf von Forbid() beziehungsweise Permit(). Forbid sperrt alle weiteren Tasks, während Permit den alten Zustand herstellt. Beide Routinen liegen in der Exec-Library.

So, das reicht. Dies sind zumindest einige wichtige Punkte, die Sie bei der Programmierung des Amiga berücksichtigen sollten — Es lohnt sich. Der Amiga ist ein so fantastischer Computer, er verdient auch entsprechend behandelt zu werden.

(Wolf-Jürgen Faust/rs)

RAM-Erweiterung abschalten

Einige Programme für den Amiga 500 laufen nicht mit der Speichererweiterung. Auch das Programm »NoFastMem« hilft in manchen Fällen nicht. Sie müssen die RAM-Erweiterung schon komplett abschalten. Hierzu ist lediglich der Einbau eines Schalters erforderlich:

— Nehmen Sie zunächst die Erweiterungsplatine und betrachten sie von der Unterseite. Trennen Sie auf der Platine die eingezeichnete Leiterbahn direkt an Pin 32 durch und überbrücken die Unterbrechung mit einem Schalter. Vor jedem Aus- und Einschalten der RAM-Erweiterung schalten Sie auf jeden Fall den Amiga aus.

Warten Sie mit dem Umschalten etwa 10 Sekunden, bis alle Signalpegel im Rechner abgebaut werden.

Die Umbauanleitung bezieht sich auf die Original-Speichererweiterung von Commodore, das Prinzip ist aber bei allen Speichererweiterungen dasselbe: Sie müssen die Verbindung zwischen den Leitungen EXRAM Pin(32) und GROUND (unter anderem Pin 33) durchtrennen und mit einem Schalter überbrücken. Nur wenn EXRAM auf low liegt, erkennt der Amiga die RAM-Erweiterung. Dies ist der Normalfall bei eingebaute Speichererweiterung. Ist die Verbindung bei eingebaute Erweiterung jedoch unterbrochen, steht der zusätzliche Speicher nicht zur Verfügung. (Anmerkung der Redaktion: Bei ausgeschalteter Erweiterung ist die Verbindung zwischen EXRAM und GROUND unterbrochen. Nur das zusätzliche RAM wird beim Starten nicht aktiviert. Eine auf der Erweiterung eingebaute Uhr arbeitet normal.)

Denken Sie bitte daran, daß Sie durch Arbeiten an der Hardware jegliche Garantieansprüche verlieren und Ihren wertvollen Computer zerstören könnten. Eine Umbauanleitung, bei der Sie sich das Löten auf der Platine ersparen, finden Sie in der 68000er Ausgabe 11/87 auf Seite 101).

(Frank Ahlhorn/rs)

Es geht auch ohne Mäuse

Gehören Sie zu den Anwendern, die die Maus auch mal beiseite legen und nur mit dem CLI arbeiten? Insbesondere C- und Assemblerprogrammierer benötigen keine Workbench-Umgebung. Dann werden Sie sicherlich eine Diskette anlegen, bei der das AmigaDOS-Fenster nach dem Booten gar nicht erst geschlossen wird. Am einfachsten geht dies, indem Sie den Befehl LOADWB in der »Startup-Sequence« entfernen. Dabei entsteht das Problem, daß das Fenster nach dem Start mit der Maus auf die volle Größe gebracht werden muß, um die PAL-Auflösung zu nutzen. Doch wo ist die Maus? Wieder müssen Sie dieselbe aus der Schreibtischschublade hervorkramen. Aber halt, es geht auch einfacher. Dieses C-Programm, in die »Startup-Sequence« eingebunden, schafft Abhilfe:

```
#include <exec/types.h>
#include <intuition/intuition.h>
```

Die heißesten Tips für Insider

Tips und Tricks gehören zur Lieblingslektüre aller Computerfans. Unsere Zusammenstellung wertvoller kleiner und großer Tips hilft Profis und Fortgeschrittenen, Probleme, die mit dem Amiga auftreten, effektiver zu lösen.


```

struct IntuitionBase *IntuitionBase;
main()
{
    IntuitionBase = (struct IntuitionBase*)
        OpenLibrary ("intuition.library",0);
    if (IntuitionBase == 0) exit (0);
    SizeWindow (IntuitionBase -> ActiveWindow, 0,
        56);
    CloseLibrary (IntuitionBase);
}

```

In den ersten beiden Zeilen werden die benötigten Include-Files eingebunden. Der Zeiger auf die IntuitionBase wird eingerichtet und in main() die Intuition-Bibliothek geöffnet. Die Funktion »SizeWindow« verändert die Größe eines Windows:

SizeWindow (Zeiger auf Fensterstruktur, dy, dx)

In diesem Fall wird das aktuelle Fenster, also das CLI-Fenster um 56 Pixel in der Vertikalen vergrößert. Zum Schluß wird die Library geschlossen und zurück zum CLI gesprungen. Eine einfache, aber wirkungsvolle Sache. (Dirk Brammerts/rs)

»J« oder »y« keine Frage

Der neue Befehl des CLI, ASK, ist eine feine Sache. Leider erwartet er ein »y« für die Antwort »Ja«. Da bietet es sich an, den Befehl einzudeutschen. An vier Stellen muß das »y« durch »j« ersetzt werden. Ein Basic-Programm führt alle Änderungen durch:

```

CHDIR "SYS:C"
OPEN "ASK" FOR INPUT AS #8
ASK$ = INPUT$(LOF(8), #8)
CLOSE #8
IF LEN(ASK$) <> 1580 THEN FEHLER
IF MID$(ASK$,1156,1) <> "y" THEN FEHLER
IF MID$(ASK$,1180,1) <> "Y" THEN FEHLER
IF MID$(ASK$,1218,1) <> "Y" THEN FEHLER
IF MID$(ASK$,1228,1) <> "y" THEN FEHLER
MID$(ASK$,1156,1) = "j"
MID$(ASK$,1180,1) = "J"
MID$(ASK$,1218,1) = "J"
MID$(ASK$,1228,1) = "j"
OPEN "askd" FOR OUTPUT AS #6
PRINT #6, ask$;
CLOSE #6
END
Fehler:
PRINT "Da stimmt was nicht"

```

Nach erfolgreicher Konvertierung löschen Sie die Datei »ask.info«. Damit der alte ASK-Befehl erhalten bleibt, heißt der neue Befehl »ASKD«. Wenn Sie ausschließlich mit der deutschen Version arbeiten möchten, löschen Sie ASK, um Speicherplatz zu sparen. (Harald Zacharias/rs)

Mit Basic ins System

Wußten Sie, daß Sie einen Screen auf dem Bildschirm mit einem Basic-Programm verschieben können? Wissen Sie, wie ein Overscan-Screen geöffnet wird? Dies ist nicht sonderlich schwer. Setzen Sie die Werte für die Größen Width und Height auf 704 beziehungsweise auf 282. Aber ist es wirklich erforderlich, das Bild mit den Preferences nach links oben zu verschieben? Diese Arbeit übernimmt von nun an der Amiga:

```

LIBRARY "intuition.library"
prefs$ = STRING$(120,0)
CALL GetPrefs&(SADD(prefs$),120)
x = PEEK(SADD(prefs$)+118)
y = PEEK(SADD(prefs$)+119)
POKE SADD(prefs$)+118, x-16 AND 255
POKE SADD(prefs$)+119, x-13 AND 255
CALL SetPrefs&(SADD(prefs$),120,1)
SCREEN 1,704,282,1,2
WINDOW 1,,(0,0)-(694,268),0,1

```

```

LINE (0,0)-(694,268),1
LINE (0,0)-(693,267),1
a$=""
WHILE a$ = "" : a$ = INKEY$ : WEND
WINDOW CLOSE 1
SCREEN CLOSE 1
POKE SADD(prefs$)+118,x
POKE SADD(prefs$)+119,y
CALL SetPrefs&(SADD(prefs$),120,1)
LIBRARY CLOSE
END

```

Zunächst öffnet das Programm die Intuition-Library. Die zugehörige Bitmap-Datei muß sich auf der Basic-Diskette befinden. Da die Angaben über die Position des Screens an 118ter und 119ter Stelle der Preferences gespeichert sind, müssen Sie 120 Byte reservieren. Dies geschieht durch die Definition eines Strings mit dieser Länge. In diesen kopieren Sie die benötigten Zeichen aus den Preferences. Dies erledigt die Funktion »GetPrefs«. Anschließend wird die neue Position des Screens in den verantwortlichen Stellen des Strings eingetragen und mit der Funktion »SetPrefs« gesetzt. Ein nun geöffneter Overscan-Screen erscheint in der Mitte des Bildschirms. Damit alles seine Ordnung hat, werden am Ende des Programms die alten Werte in die Preferences eingetragen. Wenn Sie das Listing entsprechend modifizieren, können Sie auf weitere Werte in den Preferences zugreifen. Anwendungen gibt es genug. (Friedhof Siebert/rs)

Reservieren bestimmter Speicherbereiche

Die EXEC-Funktion »AllocAbs« ist sehr nützlich, wenn bestimmte Speicherbereiche reserviert werden sollen, um dort zum Beispiel eine Grafik abzulegen. Diese Funktion teilt dem Betriebssystem mit, daß der gewünschte Bereich im Speicher belegt ist. Schreibt man seine Daten in einen nicht reservierten Bereich, kann es passieren, daß das Betriebssystem diesen neu belegt. Mit dem folgenden Assembler-Programm wird zum Beispiel ein Speicherblock von \$50000-\$60000 reserviert:

```

BelegeAbs:
    move.l    $04,a6 ; AbsExecBase
    move.l    #$50000,a1 ; Startadresse
    move.l    #$10000,d0 ; Anzahl der Bytes
    jsr      -204(a6) ; AllocAbs

```

(Christian Seiler/rs)

Direkt in die Bitmap

Für besonders schnelle Assembler-Programme ist es oft wünschenswert, Punkte auf dem Bildschirm direkt durch Adressierung der Bitplanes anzusprechen. Eine Möglichkeit hierzu bieten die zwei Unterroutinen »PSet« und »PClear«:

```

PClear:
    move.b    #$b5, label+1
PSet:
    divu      #8, d0
    mulu      #40, d1
    add.w     d0, d1
    swap      d0
    move.b    #, d2
    sub.b     d0, d2
label:
    bset      d2, (a5,d1)
    move.b    #$f5, label+1
    rts

```

Die Unterprogramme sind mit dem Seka-Assembler geschrieben. Um einen Punkt zu setzen oder zu löschen, rufen Sie »PSet« beziehungsweise »PClear« auf. Übergeben Sie folgende Werte in den genannten Registern: X-Koordinate des Punktes in d0

Y-Koordinate des Punktes in d1
Zeiger auf Bitplane in a5.

Die Startadressen der Bitplanes eines Screens sind leicht zu finden. Die Intuition-Funktion »OpenScreen« liefert den Zeiger auf die Screendaten. Addieren Sie zu diesem Wert # \$C0. An der berechneten Stelle befindet sich die Adresse der ersten Plane. Der Pointer auf die zweite und alle weiteren Planes liegt jeweils vier Byte höher.

Diese direkte Programmierung der Bitplanes ist etwas für Tüftler und birgt viele Fehlerquellen. Sie können Punkte auch durch einfache Angabe der Koordinaten mit Hilfe der Systemroutinen in der »graphics.library« setzen und löschen. In Basic steht Ihnen hierfür der Befehl PSET zur Verfügung. Allerdings sind diese Befehle ein wenig langsamer.
(David Czock/rs)

Keine unnötige Arbeit mit den Preferences

Mit Preferences kann man bekanntlich seine Workbench individuell gestalten (beispielsweise Farben oder Mauszeiger). Viele Amiga-Programme bringen jedoch eine Workbench mit, die mit anderen Voreinstellungen arbeitet. Um diese Voreinstellungen seinen Wünschen anzupassen, kann man die Parameter einer eigenen Workbench verwenden, ohne die »Preferences« aufrufen zu müssen. Die benötigten Daten befinden sich im File »devs/system-configuration«. Sie können die Konfiguration auf Ihrer Workbench mit folgendem Befehl einfach auf jede andere Startdiskette kopieren:

Copy Quelldiskette:devs/system-configuration to
Zieldiskette:devs/system-configuration

(Anmerkung der Redaktion: Bei zahlreichen Programmen ist die Veränderung der Preferences nicht unbedenklich, da die betreffenden Programme zum Beispiel nur mit einer bestimmten Farbeinstellung arbeiten können. Sie sollten dies bedenken, wenn Sie die gesamte Systemkonfiguration auf die beschriebene Art ändern.)
(Wilhelm Holtmeier/rs)

Druckermanagement

Versucht man, seinem Drucker über die Druckerschnittstelle mehr als nur Text, also auch Steuerzeichen zu übermitteln, so kann man schnell verzweifeln ... denn da kommt nichts an!

Der Grund liegt in der speziellen Druckersteuerung des Amiga. Es wurde nämlich — um die 100prozentige Kompatibilität aller Programme zu gewährleisten — ein spezieller Druckercode entwickelt, der dann von dem mit Preferences gewählten Druckertreiber für den jeweiligen Drucker umgesetzt wird. Das ist eine sehr intelligente Idee, denn so erspart man sich die umständliche Installation von neuen Programmen.

Im Handbuch wird nur kurz — durchaus üblich für Commodore-Dokumentationen — auf diesen Umstand eingegangen. Hier nun eine Auswahl der meistverwendeten Drucker-Steuer-Codes (ESC steht für CHR\$(27); die übrigen Zeichen sind in Anführungsstriche zu setzen; es können auch mehrere Befehle kombiniert werden, diese sind dann durch einen Strichpunkt zu trennen):

```
ESC[2w - Elite ein
ESC[1w - Elite aus
ESC[4w - Schmalschrift ein
ESC[3w - Schmalschrift aus
ESC[6w - Breitschrift ein
ESC[5w - Breitschrift aus
ESC[2"z - NLQ ein
ESC[1"z - NLQ aus
ESC[2p - Proportionalischrift ein
ESC[1p - Proportionalischrift aus
ESC[2v - Superscript ein
ESC[1v - Superscript aus
ESC[4v - Subscript ein
ESC[3v - Subscript aus
ESC[1m - Fettdruck ein
ESC[22m - Fettdruck aus
ESC[4m - Unterstreichen ein
ESC[24m - Unterstreichen aus
```

Ein Beispiel für eine Textausgabe in Fettdruck:

```
LPRINT CHR$(27); "[1m"; "Hallo Welt!"
```

Ein weiteres für die Übergabe des Anführungszeichens:

```
LPRINT CHR$(27); "["; chr$(34); "z"; "Diesmal in NLQ"
```

Wer diese Art der Druckersteuerung in seinen Programmen verwendet, gewährleistet dadurch, daß sein Programm auch auf anderen Druckern die gleichen Ergebnisse liefert. Wenn diese Art der Programmierung jedoch zu umständlich ist, der kann den Drucker ohne die Umleitung über »prt:« auch direkt ansprechen, indem er die Schnittstellen »par:« oder »ser:« verwendet. Alle Steuerzeichen werden dann »unzensuriert« übermittelt. Im »ROM Kernel Manual: Libraries and Devices« befindet sich im Kapitel 15 (Printer Device) eine komplette Liste dieser Drucker-Kontrollfunktion. Leser mit C-Kenntnissen erhalten hier auch detaillierte Informationen über den Umgang des Amiga mit einem Drucker.
(Michael Baas/rs)

Mit beiden Armen tief im System

Hier noch ein paar kleine Routinen für die Assembler-Freaks: Unterroutine für die Initialisierung einer eigenen Copperliste:

```
set_user_copperlist:
    MOVE.L 4,A6
    LEA Gfxname,A1
    JSR -408(A6)
    MOVE.L D0,A5
    ADD.L #$32,A5
    MOVE.L A5,savecop2
    MOVE.L (A5),savecop
    MOVE.L #COPLIST,(A5)
    RTS
```

Unterroutine für das Entfernen einer eigenen Copperliste:

```
recall_old_copperlist:
    MOVE.L savecop,A5
    MOVE.L savecop2(A5)
    RTS
```

Label, die von den Copper-Routinen gebraucht werden:

```
gfxname: dc.b "graphics.library",0
even
savecop: dc.l 0
savecop2: dc.l 0
```

(Warnung der Redaktion: Sie sollten wissen was Sie tun, wenn Sie in dieser oder anderer Form direkt auf die Hardware des Amiga zugreifen.)
(Christian Zaenker/rs)

Basic-Ausgabe im großen Stil

Das Ausgabefenster des Amiga-Basic hat eine Größe von 200 Punkten in der Vertikalen. Auch mit der Maus läßt es sich nicht auf die volle PAL-Höhe ausdehnen. Der Interpreter ist mit der falschen Maximalhöhe ausgestattet. Doch diese läßt sich durch einen kleinen Patch korrigieren: Stellen Sie zunächst fest, ob Sie die richtige Basic-Version für den folgenden Patch besitzen. Prüfen Sie die Länge des Programms mit Hilfe des CLI-Befehls LIST:

```
LIST AmigaBasic
```

Als Ergebnis sollten Sie 103484 oder 103500 erhalten. In diesem Fall schauen Sie sich das Amiga-Basic genauer an:

```
TYPE AmigaBasic opt h
```

Sobald am linken Bildschirmrand die Adresse \$5800 auftaucht, drücken Sie die Leertaste. So halten Sie das Listing an. Mit der BackSpace-Taste setzen Sie die Ausgabe fort. In der Zeile mit der Adresse \$57e0 finden Sie den gesuchten Wert für die Fenstergröße:

```
57E0: 000E4E75 2F0A303C 0280323C 00C80C43 ..Nu/.0<
      ..2<...C
```

Die Parameter \$0280 und \$00C8 entsprechen der maximalen

Breite beziehungsweise Höhe. Diese Werte können Besitzer eines Diskettenmonitors direkt auf der Diskette ändern. Doch nicht jeder hat dieses Werkzeug. Sie können den »Patch« auch mit dem folgenden Basic-Programm durchführen:

```
CLEAR , 50000&
NeueHöhe$ = CHR$(1)+CHR$(0) 'erste 1=256

OPEN "AmigaBasic" FOR INPUT AS #1 LEN=4096
OPEN "NeuesBasic" FOR OUTPUT AS #2 LEN=4096

PRINT #2, INPUT$( &H57EC, #1); 'Anfang kopieren
PRINT #2, NeueHöhe$; 'neue Höhe
a$=INPUT$(2, #1) 'alte Höhe ignorieren

PRINT #2, INPUT$( &H7000, #1); 'Rest kopieren
PRINT #2, INPUT$( &H7000, #1);
PRINT #2, INPUT$( &H5C00, #1);
WHILE NOT EOF(1) 'spart mühsames Berechnen
PRINT #2, INPUT$(1, #1); 'der verbleibenden Bytes
WEND

CLOSE #1, #2
```

Wenn Sie Basic vom CLI aus aufgerufen haben, können Sie nach Ablauf des Programms gleich weiterarbeiten. Stellen Sie zunächst sicher, daß das geänderte Basic die gleiche Länge besitzt wie das ursprüngliche:

```
LIST NeuesBasic
```

Bei einem anderen Wert als 103484 beziehungsweise 103500 müssen Sie den »Patcher« überprüfen und von vorne beginnen. Die Funktion des neuen Basic können Sie ausprobieren:

```
RUN NeuesBasic
```

Hat alles geklappt? Wenn alles gut geht, bleiben noch einige Aufräumarbeiten. Sie können mit DELETE die Info-Datei des gepatchten Basic löschen und umbenennen:

```
DELETE NeuesBasic.info
RENAME AmigaBasic AltesBasic
RENAME NeuesBasic AmigaBasic
```

Natürlich sollten Sie diese Arbeiten nicht mit der Original-Extras-Diskette ausführen. Arbeiten Sie mit einer Sicherheitskopie. Dann machen Sie nichts kaputt, wenn Ihnen ein Fehler unterläuft. Es ist zu empfehlen, daß Sie das neue Basic unter dem Originalnamen speichern. Ansonsten müßten Sie die Default-Tool-Namen aller bereits gespeicherten Basic-Programme und den entsprechenden Toolnamen, den Amiga-Basic allen neu gespeicherten Programmen zuordnet, ändern. Doch diese unnötige Arbeit wird sich jeder Programmierer ersparen wollen.

(Peter Pörner/rs)

Basic-Patch für andere Versionen

Das große Ausgabefenster für Amiga-Basic ist der Hit. Für die ältere Basic-Version 1.00 mit der Länge von 93156 Bytes sind folgende Änderungen im Patch-Programm notwendig:

- Die Zahl &H57EC muß durch &H55E8,
- die Zahl &H5C00 durch &H2800 ersetzt werden.

Mit diesen kleinen Veränderungen können Sie dann auch die älteren Basic-Versionen patchen, um in den Genuß des großen Ausgabefensters zu kommen.

(Alexander Wagner/rs)

Tolles Tool mit Anleitung

Der Editor »MicroEmacs« schlägt den ED um Längen. Allerdings nur wenige wissen, daß sich dieses hervorragende Werkzeug auf der ExtrasD-Diskette von Commodore befindet. Schauen Sie nach. Das Programm hat seinen Platz in der Schublade »Tools«. Und auch eine Anleitung ist auf der Diskette gespeichert. Sie verbirgt sich hinter dem Piktogramm »MicroEmacs.doc«. Es existieren drei Wege die Anleitung zu lesen:

— Klicken Sie das Piktogramm »MicroEMACS.doc« zweimal an. Mit einem System-Requester fordert der Amiga Sie nun auf, das Original der ExtrasD-Diskette einzulegen. Sofern Sie nur ein

Laufwerk besitzen, gelangen Sie nach zwei weiteren Diskettenwechseln in das erste Bild der Anleitung.

— Wer die Anleitung schneller lesen möchte, lädt das Utility »More«. Es dient dazu, ASCII-Dateien auf dem Bildschirm auszugeben. Nach der Frage »Filename?«, tippen Sie »MicroEMACS.doc« und drücken <Return>.

— Der elegante Weg ist, eine <Shift>-Taste zu drücken und mit der Maus die Piktogramme MicroEMACS.doc und More anzuklicken. Danach wählen Sie in der Menüzeile der Workbench »Open«. Wer es noch einfacher möchte, klickt bei gehaltener <Shift-Taste> MicroEMACS.doc an und aktiviert dann MORE mit einem Doppelklick.

Nach dem Laden sehen Sie auf dem Bildschirm die erste Seite der Anleitung. Durch Drücken von <h> gelangen Sie in einen Screen mit Hilfsanweisungen zu More. Mit <Space> blättern Sie in der Anleitung weiter. Sie umfaßt 70 Seiten und ist in Englisch geschrieben. Am Ende befindet sich eine alphabetische Liste aller »Shortcuts« (abgekürzte Befehle) von MicroEMACS. Sie verlassen die Anleitung mit <q>. Wer den gesamten Text intensiv studieren möchte, kann ihn auch mit PrintFiles ausdrucken lassen.

(Nick Seggelke/rs)

Mehr für MORE: Printfiles

Das Tool MORE auf der Extras-Diskette ist schon eine feine Sache, um Textdateien auf dem Bildschirm auszugeben. Im Gegensatz zu TYPE stoppt die Ausgabe auf dem Bildschirm immer, wenn dieser voll ist. Allerdings kann es nicht verwendet werden, um Textdateien auf dem Drucker auszugeben. Wer also TYPE durch MORE ersetzt, sollte auch das Programm »PrintFiles« im C-Ordner seiner Workbench aufnehmen. Dieses Utility befindet sich ebenfalls auf der Extras-Diskette:

```
COPY ExtrasD:TOOLS/PrintFiles TO StartDisk:C
```

Noch komfortabler als MORE ist BLITZ von der Fish-Disk 60. Dies sollten Sie sich auch einmal anschauen.

(Torsten Radtke/rs)

Laufwerk: eine brennende Sache

Nachdem in der Ausgabe 2/88, Seite 93, das Ein- und Ausschalten der Power-LED gezeigt wurde, ist nun die Leuchtdiode des Laufwerks dran. Das Aufleuchten erreichen Assemblerprogrammierer mit diesen Befehlen:

```
LED: EQU $BFD100
      move.b #127,LED
      move.b #119,LED
      move.b #0,LED +512
```

Auch wer in Basic programmiert, kann die LED aufleuchten lassen:

```
LED = 12570880&
POKE LED, 127
POKE LED, 119
POKE LED + 512,0
```

Um die LED wieder auszuschalten, braucht man lediglich die letzte 0 mit dem Wert 255 auszutauschen. Das kleine Programm kann jeder gut brauchen, der zum Beispiel dem Anwender zeigen möchte, daß er die Finger vom Laufwerk lassen soll. So verhindern Sie, daß jemand eine Diskette zu früh aus dem Laufwerk nimmt. »Entwarnung« ist erst, wenn die Lampe erlischt.

(Armin Hegglin/rs)

Joystickabfrage in Assembler

Für Programmierer von Actionspielen ist die Verwendung von Assembler eine Alternative zu Basic oder C, um schnelle Programme zu erzeugen. Um von einem Maschinenprogramm den Joystick abzufragen, gibt es prinzipiell zwei Verfahren:

1. Sie können das »Gameport-Device« verwenden. Der Haken an der Sache: Ihr Programm wird aufgebläht und langsam.
2. Als Alternative bleibt, den Joystickport direkt abzufragen. Greifen Sie mittels weniger Assemblerbefehle direkt auf die Hardwareregister zu. Das folgende Programm kehrt nach seinem Auf-

ruf mit einem Wert im Register d0 für den Joystick in Port 0 zurück. Den entsprechenden Wert für Port 1 finden Sie in d1:

```
JOYODAT = $DFF00A
JOY1DAT = $DFF00C
JOYTEST = $DFF036
LeseJoystick:
  clr.w JOYTEST      ; Löschen der Mausregister
  move.w JOYODAT,d0 ; JoyPort 0 lesen
  move.w JOY1DAT,d1 ; JoyPort 1 lesen
  rts
```

In den Registern d0 und d1 stehen je nach Stellung des angeschlossenen Joysticks diese Werte:

```
$0300 links
$0200 links oben
$0201 links unten
$0100 oben
$0000 Mitte
$0001 unten
$0103 rechts oben
$0003 rechts
$0002 rechts unten
```

Einfacher und schneller geht es nicht. (Jörg Schmidt/rs)

Der Herr der Fenster

Basic-Programmierer können mit Hilfe der »Intuition-Library« jedes Fenster auf vielfältige Weise beeinflussen. Die Befehle in dieser Bibliothek sind um einiges leistungsfähiger als die üblichen Basic-Befehle. Die »Intuition-Library« ist unter anderem für die Verwaltung der Windows und Screens verantwortlich. Öffnen Sie zunächst diese Bibliothek:

```
LIBRARY "intuition.library"
```

Natürlich muß sich die Datei »intuition.bmap« im aktuellen Dateiverzeichnis befinden. Folgende Befehle stehen dem Programmierer nun zur Verfügung:

```
CALL WindowLimits (WINDOW(7),minx,miny,maxx,maxy)
verändert die Minimal- und Maximalwerte des aktuellen Fensters.
```

```
CALL SizeWindow (WINDOW(7), Breite, Höhe)
```

wechselt die Größe eines Windows um die angegebenen Werte.

```
CALL WindowToFront (Window(7))
```

```
CALL WindowToBack (WINDOW(7))
```

bringen ein Fenster in den Vorder- oder Hintergrund.

```
CALL MoveWindow (WINDOW(7),X,Y)
```

Damit läßt sich ein Fenster auf dem Bildschirm positionieren. Mit ähnlichen Befehlen können Sie Screens beeinflussen. Sie müssen nur einen Zeiger auf den Screen-Datensatz festlegen:

```
LET ScreenHdl& = PEEKL (WINDOW(7)+46)
```

Die Befehle für einen Screen lauten:

```
CALL ScreenToFront(ScreenHdl&)
```

```
CALL ScreenToBack (ScreenHdl&)
```

```
CALL MoveScreen (X,Y)
```

Wer so seine Fenster und Screens kontrolliert, wird feststellen, daß er viele neue leistungsfähige Funktionen einsetzen kann. Er muß nur aufpassen, daß für die Breite oder Länge Integervariablen verwendet werden. Außerdem werden die übergebenen Werte nicht auf Gültigkeit überprüft. (Tobias Helge Kosuch/rs)

Basic-Programme beschleunigen

Basic-Programme lassen sich beschleunigen. Hierzu muß der Programmierer die Funktion »SetTaskPri« aus der exec.library aufrufen und die Priorität des zum Basic-Programm gehörenden Tasks erhöhen:

```
DECLARE FUNCTION FindTask& LIBRARY
LIBRARY "exec.library"
task& = FindTask&(task&, zahl)
```

```
CALL SetTaskPri& (task&, zahl)
```

```
...
```

```
REM Programm
```

```
...
```

```
CALL SetTaskPri& (task&,0)
```

Durch den ersten Aufruf erhöhen Sie die Priorität eines Programms. Das Betriebssystem stellt dem Programm nun mehr Rechenzeit zur Verfügung. Die »Zahl n« kann sich in den Grenzen 1 bis 127 bewegen, um eine Beschleunigung zu bewirken. Je höher der Wert, desto schneller läuft das Programm. Der zweite Aufruf setzt den Zeitfaktor wieder auf den ursprünglichen Wert. Dies sollten Sie nicht vergessen, da sonst die Betriebssystemroutinen, die normalerweise vorrangig bearbeitet werden, nicht mehr korrekt ablaufen können. Die Beschleunigung macht sich vor allem bezahlt, wenn viele Berechnungen anfallen oder eine komplizierte Grafik aufgebaut wird. (Klaus van Eerd/rs)

Nanu — Fenster ohne Rahmen

Mit der Routine »WRAND« schalten Sie die Umrandung eines Fensters aus oder an. Das folgende Programm zeigt die Subroutine und eine Anwendung als Beispiel::

```
LIBRARY "intuition.library"
WINDOW 2, "test", (100,50)-(300,150),26
WRAND
PAINT (1,1),2
COLOR 2,3:PRINT "AMIGA-Fenster"
WHILE WINDOW(0)=2
  IF MOUSE(0) THEN CALL WRAND
WEND
WINDOW CLOSE 2
LIBRARY CLOSE
END
SUB WRAND STATIC
wi&=WINDOW(7)
'Adresszeiger Intuition: Fenster-Datensatz
POKEL(wi&+24),PEEKL(wi&+24) XOR 2048
'Borderless-Flag setzen/löschen
CALL RefreshWindowFrame(wi&) 'Rahmen erneuern
END SUB
```

Wie bei allen Programmen, die Routinen einer Systembibliothek verwenden, muß sich natürlich die Bitmap-Datei der verwendeten Library im aktuellen Dateiverzeichnis befinden. In diesem Fall brauchen Sie also die Datei »Intuition.bmap«.

(Michael Padberg/rs)

Debugging-Hilfe für C-Programme

Jeder, der schon mit Compilern gearbeitet hat, weiß ein Lied davon zu singen: das Programm steckt voller Fehler, die der Compiler auch brav ausgibt, aber viel zu schnell. Ehe sich der Programmierer versieht, sind die ersten Meldungen bereits vom Bildschirm verschwunden. Doch zum Glück kann der Amiga-User Dateien umlenken. Die Sequenz:

```
cc >Errout Optionen
```

ruft den Compiler auf und lenkt die Ausgabe der Fehlerliste auf eine Diskettendatei um. Diese können Sie dann mit TYPE jederzeit ausgeben und anschauen. Natürlich können Sie die Fehlerliste auch auf einem Drucker ausgeben. (Jörg Nowak/rs)

Schnelle Schleifen in C

Bei Schleifen lohnt es sich, Zeit zu sparen:

```
main ()
int x
for ( x=0; x < 5000000 ; x++)
```

Diese Schleife mit 5 Millionen Durchläufen benötigt etwa 46 Sekunden, wenn Sie das Programm von der RAM-Disk starten. Wesentlich schneller wird die Schleife durchlaufen, wenn Sie Registervariablen verwenden:


```
main()
register int x;
for ( x=0; x < 5000000; x++)
```

Dieser »Loop« benötigt nur 23 Sekunden. Eine vergleichbare Assembler-Schleife ist noch schneller:

```
Start:
  move.l #5000000,d2
loop:
  dbra d1, loop
rts
```

Eine solche Schleife benötigt nur noch 7 Sekunden.

(Andreas Moshhammer/rs)

NoFastMem für den Amiga 2000

In der AMIGA 8/9 veröffentlichten wir in der Rubrik Tips und Tricks eine Umbauanleitung, um die Speichererweiterung beim Amiga 2000 abzuschalten. Hier ist eine einfachere Version. Da auf der Erweiterungskarte im Amiga 2000 keine Uhr berücksichtigt zu werden braucht, kann die Abschaltung des Speichers durch eine Unterbrechung der Spannungsversorgung erfolgen. Nehmen Sie hierzu die Platine; die Bauteileseite bitte dabei auf Sie zugerichtet.

Trennen Sie auf der Erweiterungskarte die Leiterbahn zwischen den Kontaktfahnen an Pin 5 und 6 durch. Diese Trennstelle müssen Sie nun mit einem Schalter überbrücken. Den Schalter können Sie außen am Amiga befestigen.

Achtung: Vor jedem Umschalten muß der Amiga ausgeschaltet werden. Bedenken Sie bitte, daß Sie durch Arbeiten an der Hardware jegliche Garantieansprüche verlieren. Für eventuelle Schäden, die beim Umbau entstehen, wird keine Haftung übernommen.

(Dieter Schlingmann/rs)

Die Lösung für den neuen Amiga

Viele Besitzer des Amiga 2000 stellen uns die Frage, welche Unterschiede zwischen dem Amiga A2000 und B2000 bestehen.

Differenzen bestehen in der Hardware. Verschiedene Bauteile des alten Amiga 2000 sind durch »FAT-Agnus« und »Garry« ersetzt worden. Der B2000 enthält im Gegensatz zum A2000 1 MByte auf der Hauptplatine. Die oberen 512 KByte sind durch einen Jumper abschaltbar. Damit wird der Einbau eines Schalters, um das FastRam hardwaremäßig abzuschalten, noch einfacher. Allerdings ist eine Erweiterung auf 1,5 MByte nicht mehr so leicht durchführbar wie zuvor.

Eine weitere Änderung: Der Anschluß des Lightpen wurde auf den zweiten Maus-Port gelegt. Mit Hilfe eines Jumpers läßt sich diese Einstellung ändern. Die Entwickler haben auch den Video-Teil durch einen 36poligen Slot erweitert.

Die wichtigen Jumper im einzelnen:

— J101: Mit diesem wird das höchste Adreßbit der Speichererweiterung eingestellt. Momentan ist es A23. Es wird benötigt, um den Speicherbereich ab \$C00000 zu verwalten. In der zweiten Position ist es Bit A19. Es erlaubt die Nachrüstung mit einem erweiterten VideoChip. Dieser wird also 1 MByte verwalten können.

— J200: Der Jumper regelt den Lightpen-Port.

— J300: Er ist zuständig für den Takt des CIA Timer Chip. In der momentanen Position wird der Timer mit Netzfrequenz, das heißt 50 Hertz, getaktet. In der zweiten Position bestimmt der Takt des vertikalen Synchronisierimpulses das Geschehen.

— J301: Dieser Jumper ist für den Anschluß eines zweiten internen Laufwerks entscheidend.

— J500: Sind die oberen 512 KByte des 1 MByte großen RAM-Bereichs unerwünscht, können Sie diesen Jumper auftrennen. Dann besitzt der Amiga 2000 nur noch 512 KByte Chip-RAM.

(Jörg Koch/rs)

Quelle: Technical Reference Manual 500/2000

Externes Laufwerk zum Booten

Mit einer Änderung der »Startup-Sequence« können Sie den Bootvorgang auf ein externes Laufwerk umleiten. Dies ist interessant für die Besitzer eines 5¼-Zoll-Laufwerks, die ein Programm

auf 5¼-Zoll-Diskette kopiert haben, das sich aber nur durch Booten starten läßt.

Für die Änderung müssen Sie eine Startdiskette präparieren, die den Bootvorgang auf das externe Laufwerk umlenkt. Am einfachsten verwenden Sie eine normale Workbenchkopie.

Auf dieser ändern Sie die »Startup-Sequence«:

```
ASSIGN S:      df1:s
FAILAT 21
ASSIGN L:      df1:l
ASSIGN FONTS:  df1:fonts
ASSIGN LIBS:   df1:libs
ASSIGN devs:   df1:devs
ASSIGN SYS:    df1:
ASSIGN C:      df1:c
df0:c/CD SYS:
df0:c/EXECUTE Startup-Sequence
```

Probieren Sie das Programm aus. Legen Sie Ihre »externe Bootdiskette« in Laufwerk df1: und die modifizierte Workbench im Laufwerk df0: ein. Jetzt starten Sie den Amiga erneut.

Diese Methode hilft oft, aber nicht immer. Sie versagt zum Beispiel bei Programmen, die das interne Laufwerk direkt ansprechen, um dort Daten zu lesen.

(Angela Schmidt/rs)

CLI-Mate ohne »fastdir«

Mit Hilfe eines Basic-Programms läßt sich eine der schlechten Angewohnheiten von »CLI-Mate« vermeiden. »CLI-Mate« ist ein leistungsfähiges Tool, mit dem Sie Ihre Disketten verwalten können. Das Programm erlaubt es, Files aus jedem Dateiverzeichnis in andere zu kopieren. Des weiteren erzeugt das Programm für jedes Dateiverzeichnis auf einer Diskette ein »Fast-Directory«. Dieses wird automatisch auf jede Diskette kopiert, deren Schreibschutz nicht aktiviert ist.

Der große Vorteil dieser Prozedur: Jedesmal wenn Sie mit »CLI-Mate« ein Directory von der Diskette lesen, das bereits als »fastdir« vorhanden ist, zeigt der Amiga prompt das Inhaltsverzeichnis an.

Der Nachteil: Diese zusätzlichen Files nehmen Platz weg. Sie stören auf Disketten, die Sie einmal mit »CLI-Mate« erstellt, aber nie wieder mit dem Programm bearbeiten möchten.

Das Programm »Modify.CLI-Mate« schafft Abhilfe:

```
DEFINT a-z
OPEN "r", #1, "df0:c/climate", 1

REM Befindet sich das Programm in einem anderen
REM Pfad, so muß dieser eingesetzt werden.
FIELD #1,1 as d$
h$="nil:"+string$(4,0)
REM h$="fastdir" * für alten Zustand*
Loop:
  READ: IF s=-1 then CLOSE : END
  FOR i = s to s+7
    LSET d$=MID$(H$,i-s+1,1) : PUT #1,I
  NEXT I
  GOTO Loop

DATA &H083B
DATA &H0F5E
DATA &H0F8A
DATA &H1780
DATA &H17A9
DATA &H438D
DATA &H4552
DATA &H4BD3
DATA &H4C91
DATA &H4DCB
DATA &H -1
```

Um ein modifiziertes Programm wieder in den alten Zustand zu versetzen, müssen Sie die im Listing angegebene Zeile von »Modify.CLI-Mate« ersetzen.

(H. Rückeshäuser/rs)

Hello, Amiga is calling

Subroutinen in Amiga-Basic müssen mit CALL aufgerufen werden, wenn der Aufruf am Anfang einer Zeile steht und keine Parameter besitzt. Diese Maßnahme ist erforderlich, da der Interpreter den Aufruf sonst mit einer Sprungmarke verwechselt. Doch wird dies auch verhindert, wenn Sie statt CALL einen Doppelpunkt verwenden. Das verkürzt nicht nur die Zeile, sondern trägt auch zur Übersichtlichkeit bei.

(Jürgen Brendel/rs)

Störenfriede im Basic-Programm

Wenn Sie einmal in Basic auf Syntaxfehler stoßen, für die Sie keine Erklärung finden, kontrollieren Sie einmal alle Zeilen mit einer »END IF«-Anweisung. Sobald hinter IF noch ein Blank (Leerzeichen) steht, spielt der Interpreter häufig nicht mehr mit. Am besten bringen Sie den Cursor hinter die Anweisung und drücken <Return>. Dann sind Sie sicher, daß keine störende Zeichen folgen.

René Beaupoil/rs)

Wohin mit den .bmaps

Um zu gewährleisten, daß Amiga-Basic die .bmap-Dateien auch immer findet, sollten diese ins Verzeichnis »libs« Ihrer System-Diskette kopiert werden. So können Sie Ihr Basic-Verzeichnis beliebig mit CHDIR wechseln, ohne eine LIBRARY-Anweisung mit »File not found« quittiert zu bekommen. Außerdem brauchen Sie die .bmap-Dateien nun nur noch auf einer Diskette, egal wie viele Basic-Disketten Sie haben.

(Angela Schmidt/rs)

Es gibt nichts zu bereuen

Haben Sie schon einmal Änderungen in einer Zeile beim Editieren eines Programms in Amiga-Basic bereut. Jetzt wollen Sie die »Untat« wieder rückgängig machen, aber wie sah die Zeile doch gleich noch aus...? Eine einfache Hilfe ist die UNDO-Funktion: Drücken Sie, ohne die Zeile vorher zu verlassen, die rechte <Amiga>-Taste und <L>. Sofort erscheint die alte Zeile wieder.

(Jens Spitzczok/rs)

Das gibt's doch gar nicht

In der Ausgabe 3/88 des AMIGA-Magazins (Seite 106), wurde beschrieben, wie man in einem Basic-Programm feststellt, ob eine Datei vorhanden ist. Hier ist eine weitere Möglichkeit, die auch ohne die Systembibliotheken auskommt:

```
OPEN "Pfad/Name" FOR APPEND AS #1
IF LOF(1) = 0 THEN
  PRINT "Datei nicht vorhanden"
ELSE
  PRINT "Datei vorhanden"
END IF:
CLOSE #1
```

Als erstes wird die zu prüfende Datei zum sequentiellen Anfügen geöffnet. Jetzt wird die Länge der Datei ermittelt. Ist sie gleich Null, ist die Datei nicht auf der Diskette. Ist die Datei nicht vorhanden, wird sie auch nicht angelegt, wie das beim normalen Öffnen

der Fall wäre — ein großer Vorteil dieser Methode. Selbst leere Dateien werden erkannt, da sie immer ein Abschlußzeichen enthalten.

(Roland Nöll/rs)

Fensternamen

In der »Intuition.library« gibt es eine Funktion namens »SetWindowTitles«. Sie ändert den Namen eines Fensters. Mit einer Unter-routine läßt sich dies von Basic aus realisieren:

```
SUB Titel (WT$,ST$) STATIC
  WT$=WT$+CHR$(0) : ST$=ST$+CHR$(0)
  CALL SetWindowTitles
  (WINDOW(7),SADD(WT$),SADD(ST$))
END SUB
```

Die Subroutine ändert sowohl den Namen des aktuellen Fensters als auch des zugehörigen Screens. Übergeben Sie die neuen Namen beim Aufruf:

```
CALL Titel ("Neuer Fenstername", "Neuer
Screentitel")
```

Beachten Sie, daß Sie zuvor die »Intuition.library« mit dem Befehl »LIBRARY "intuition.library"« öffnen müssen.

(Tobias Helge Kosuch/rs)

Die besten Tips zum Amiga- Basic

Amiga-Basic ist einer der leistungsfähigsten Basic-Dialekte, die es überhaupt gibt. Aber holen Sie auch wirklich alles aus dieser Sprache heraus? Mit den tollen Tips auf den folgenden Seiten erweitern Sie Ihren Sprach-schatz.

Eigene Icons für Amiga-Basic

Wer hat sich noch nicht darüber geärgert, daß die Icons (Bildsymbole) der Amiga-Basic-Dateien alle gleich aussehen? Ändern der Icons mit dem »IconEd« lohnt sich nicht, denn kaum hat Amiga-Basic die Datei wieder bearbeitet, ist das Standard-Bildsymbol wieder da. Ein kurzes Basic-Programm schafft hier Abhilfe. Es läßt sich einfach in bestehende Programme einbauen. Zunächst muß einmalig mit dem Programm »ConvertFD« aus dem Verzeichnis »BasicDemos« der ExtrasD-Diskette die »bmap«-Datei für die Icon-Bibliothek erzeugt werden. Dieser Vorgang ist im Amiga-Basic-Handbuch ausführlich beschrieben. Anschließend können Sie das gewünschte Icon

mit dem auf der Workbench-Diskette befindlichen Symbol-Editor (Icon-Ed) zeichnen. Eine Beschreibung der Bedienung des Editors ist in dieser Ausgabe zu finden. Beim Test des Listings wurde das erstellte Icon in einem mit »MAKEDIR DBOBJ« erzeugten Verzeichnis unter dem Namen »DBIcon.info« abgelegt. Der Zusatz »info« braucht nicht angegeben zu werden. Er wird von Put/Get-DiskObjekt automatisch generiert. Die »bmap«-Datei wurde in das Verzeichnis »basiclibs« gelegt. Die neue Routine weist das DBIcon allen Programmen zu, deren Namen eingegeben wird. Der Filename »ende« bricht die Verarbeitung ab. Eventuell auftretende Fehlermeldungen haben folgende Bedeutung:

150 — Datei nicht vorhanden oder kein Piktogramm

151 — Fehler bei der Piktogrammerstellung

Bei Verwendung in eigenen Programmen muß die Routine hinter der Close-Anweisung für die jeweilige Datei stehen, da sonst wieder das Einheits-Icon auftaucht.

```
LIBRARY "basiclibs/icon.library"
DECLARE FUNCTION PutDiskObjekt& LIBRARY
DECLARE FUNCTION GetDiskObjekt& LIBRARY
```



```

filnam$=" :DBObj/DBIcon"+chr$(0)
diskobj$=GetDiskObjekt$(SADD(filnam$))
IF diskobj$=0 THEN ERROR 150
ls1f:
INPUT "Filename: ";file$
IF file$="ende" THEN SYSTEM
file$=file$+chr$(0)
zw$=PutDiskObjekt$(SADD(file$),diskobj$)
IF zw$=0 THEN ERROR 151
GOTO ls1f

```

(Hans Keutgen/rs)

Schnelle Textausgabe in Basic

Da die Geschwindigkeit der Bildschirmausgabe beim Amiga-Basic ja direkt zum Abgewöhnen ist, haben wir die Libraries (Library: Routinensammlung im Kickstart-ROM oder auf Diskette) nach einer schnelleren Routine durchforstet und sind in der »graphics.library« bei »text« fündig geworden. Um diese Routine im Basic zu verwenden, stellen wir eine kleine Sub-Routine vor, die mit »Prnt(a\$)« oder »Prnt(".....")« aufgerufen werden kann. Dabei ist jedoch zu beachten, daß hinter dem Text kein Zeilenvorschub gemacht wird.

```

SUB Prnt(a$) STATIC
CALL text(Window(8),SADD(a$),LEN(a$))
END SUB

```

Am Anfang des Hauptprogramms muß die Library mit »LIBRARY "graphics.library"« eröffnet werden. (Stefan Riegel/rs)

Copy und Paste in Basic

Beim Entwickeln verschiedener Programme mit ähnlichen Strukturen kann es vorkommen, daß sich viele Programmteile untereinander gleichen. Eine einfache Methode, Programmteile von einem Programm in das andere zu übertragen, ist die Verwendung der COPY-Funktion aus der Menü-Leiste. Laden Sie das Programm, in dem der gewünschte Programmteil steht, markieren Sie ihn und aktivieren danach die COPY-Funktion. Das Segment wird nun im »Clipboard« (Zwischenspeicher) gespeichert. Nun können Sie das Programm laden, in dem dieser Programmteil benötigt wird, den Cursor positionieren und mit PASTE das Segment in den Programmspeicher übernehmen.

(Uwe Bäumler/rs)

Mehr Speicher für Amiga-Basic

Mit dem Basic-Kommando CLEAR können Sie den verfügbaren Programm- und Variablenspeicher neu dimensionieren (siehe auch Amiga-Basic-Handbuch). Dennoch gibt es Schwierigkeiten mit dem Befehl:

CLEAR liefert häufig keinen Block, obwohl FRE(-1) noch genügend Platz anzeigt. Die Erklärung können Sie leicht experimentell ermitteln:

Laden Sie vor Aufruf des Basic-Interpreters das Utility »Free Memory« von der »ExtrasD-Diskette«. Dieses Programm öffnet am unteren Bildrand ein Fenster und gibt eine grafische Darstellung der momentanen Speicherbelegung. Laden Sie nun Amiga-Basic und Ihr Testprogramm. Das Zusatzfenster zeigt wie der freie Systemspeicher zusammenschumpft.

Starten Sie ein Basic-Programm, das am Anfang eine CLEAR-Anweisung ausführt. Diese soll beispielsweise das Daten- und Programmsegment der voreingestellten 25000 Bytes vergrößern.

```
CLEAR , 50000
```

Das geht nur dann gut, wenn im Speicher ein weiterer zusammenhängender Block der angeforderten Größe neben dem bereits bestehenden Bereich zur Verfügung steht. »Free Memory« zeigt Ihnen sehr eindrucksvoll, wie das aktuelle Datensegment bei diesen Operationen hin- und hergeschaltet wird. Erst wird ein neuer Block reserviert, dann das alte Segment freigegeben. Das Ergebnis dieser Versuche lautet:

— Der momentane Programm- und Datenspeicher wird bei der Ermittlung eines neuen nicht einbezogen. Dies gilt auch, wenn der aktuelle Speicher nur zum Teil genutzt wird.

— Der neue Speicherbereich muß zusammenhängend sein.

Es gibt zwei Wege, um ein möglichst großes Datensegment zugewiesen zu bekommen:

1. Im Direkt-Modus verkleinern Sie das Datensegment soweit wie möglich:

```
CLEAR , 1024
```

Nun können Sie ein großes wählen. Zum Beispiel:

```
CLEAR , 100000
```

2. Im Programm erfolgt die Reservierung folgendermaßen.

```
CLEAR , 25000 - FRE(0)
```

REM tatsächliche Programmlänge bleibt reserviert

```
CLEAR, FRE(-1) -30000
```

REM verbleibender Speicher abzüglich einer Reserve

Die Reserve, die Sie dem Betriebssystem zur Verfügung stellen müssen, hängt davon ab, ob Sie im Programm Befehle verwenden, die Systemspeicher anfordern:

- SOUND
- WAVE
- WINDOW
- SCREEN
- LIBRARY

Besonders das Öffnen eines neuen Screens läßt den verfügbaren Speicher rasch dahinschmelzen. (D. Siedentop/rs)

Joystickabfrage

Die Basicfunktion »STRIG(n)« liefert je nach Argument (n) den Status des Feuerknopfes eines wählbaren Joysticks. Die Funktion übergibt eine »-1« oder »0« als Rückgabewert, je nachdem, ob die getestete Bedingung erfüllt ist oder nicht. In der Beschreibung im Basic-Handbuch hat sich der Guru eingeschlichen. Dort stehen die Rückgabewerte »1« beziehungsweise »0«.

(Jochen Kümmel/rs)

Ein neuer Befehl für Basic

Der Befehl CENTER erlaubt Ihnen, einen Text in einem Fenster zu zentrieren. Zahlreiche Basic-Versionen, zum Beispiel Simons Basic auf dem C 64, kennen dieses Statement. Es läßt sich in Amiga-Basic als Subroutine verwirklichen:

```

SUB Center (a$) STATIC
POKEW WINDOW (8) + 36, WINDOW (2)/2-LEN (a$) * 4
PRINT a$
END SUB

```

Wenn Sie nun Überschriften ausgeben möchten, schreiben Sie in Ihrem Programm:

```
CENTER "genau in der Mitte"
```

Paßt dieser Text nicht in das Fenster, wird die Ausgabe unterdrückt. Ansonsten erscheinen alle Zeilen — »genau in der Mitte«. (Tobias Helge Kosuch/rs)

Sag mir deinen Namen

Nach langer Nacht ist das Programm endlich fertig — fast. Alles was der Programmierer noch braucht, ist eine Funktion, die durch einen bestimmten Tastendruck ausgelöst wird. »Wie war noch mal der Code von dieser.... Wo ist das Handbuch? Oje, da stehen gar nicht alle Werte und dann auch noch von einer anderen Tastatur. Verd...«

Wenn Sie alle Tastencodes Ihres Keyboards wissen möchten, schreiben Sie sich lieber eine Liste, die Sie immer verwenden können:

```

10 REM Tastenkombinationen
20 WHILE 1
30   a$=""

```

```

40 WHILE a$=""
50   a$=INKEY$
60 WEND
70 FOR I=1 TO LEN(a$)
80   PRINT ASC(MID$(a$,I,1))
90 NEXT
100 PRINT
110 WEND

```

Mit diesem Basic-Programm können Sie alle Tastatur-Codes ermitteln. Vor allem, wenn dieses verflixte Handbuch wieder einmal verschwunden ist. (Klaus Blaschek/rs)

Alternative zu MOUSE

Mit Amiga-Basic lassen sich die Koordinaten der Maus nur in Verbindung mit MOUSE(0) abfragen. Eine Alternative:

```

PEEKW(WINDOW(7)+12)
PEEKW(WINDOW(7)+14)

```

Der erste Aufruf liefert die Y-, der zweite die X-Koordinate. Die Werte beziehen sich auf den Rahmen des Fensters. Um die Koordinaten relativ zur Zeichenebene zu erhalten, müssen Sie 11 vom Y- und 3 vom X-Wert abziehen. Ein Beispiel:

```

wind& = WINDOW(7)
WHILE 1
  LOCATE 1,1
  PRINT PEEKW(wind&+14), PEEKW(wind&+12)
WEND

```

Das Programm zeigt ständig die Mauskoordinaten relativ zum Rahmen des Fensters. Wenn Sie in der ersten Zeile des Listings `a&=PEEKW(WINDOW(7)+46)`

schreiben und in der vierten Zeile

```
PRINT PEEKW(a&+18), PEEKW(a&+16)
```

erhalten sie die Angaben bezüglich des Screens.

(Tobias Helge Kosuch/rs)

Scrollen im Listfenster

Das Listfenster des Amiga-Basic läßt sich auf mehrere Arten scrollen:

- Am bekanntesten ist das Scrollen mit Hilfe der Cursortasten.
- Seitenweise läßt sich das Fenster durch gleichzeitiges Drücken der Shift-Taste und der Cursortasten nach oben beziehungsweise unten verschieben.

- Die dritte Methode ist hingegen nicht so bekannt. Das Listfenster können Sie auch mit der Maus bewegen.

Um eine bestimmte Zeile auf dem Bildschirm anzuzeigen, brauchen Sie nur die linke Maustaste drücken und bei gedrückter Taste mit der Maus nach unten oder oben aus dem Fenster zu fahren. Das Listing bewegt sich so lange in die geforderte Richtung, bis Sie den Mauszeiger wieder ins Fenster bringen, oder die Maustaste loslassen.

Diese Funktion dient eigentlich zum Markieren eines Blocks, der nachfolgend mit »COPY, CUT und PASTE« bearbeitet werden soll. Um den selektierten Block normal darzustellen, genügt ein Mausklick. Alles ist wie gehabt, nur der Cursor ist genau dort, wo Sie es möchten. (Heinz Löscher/rs)

Lieber schlafen als lange warten

Das Basic-Statement SLEEP erscheint nur selten in Programmen. Es könnte in der Kombination mit der Maus nützlich angewendet werden. Wenn ein Programm warten soll, bis Sie die Maustaste anklicken, ist diese Subroutine geeignet:

```

Warte_auf_Klick1:
  WHILE ( MOUSE(0)=0)
  WEND
RETURN

```

Diese Routine hat den Nachteil, daß der Prozessor viel Zeit verschwendet. Immer wieder durchläuft er die WHILE-Schleife. Die

Alternative hierzu ist eine Routine, die mit SLEEP arbeitet:

```

Warte_auf_Klick2:
  ON MOUSE GOSUB Geklickt
  MOUSE ON
  Nochwach = 1
  WHILE (Nochwach)
    SLEEP
  WEND
RETURN
Geklickt:
  Nochwach = 0 : RETURN

```

Diese Mausabfrage ist besser, da der Prozessor andere Aufgaben wahrnehmen kann. Der Amiga ist eine Multitasking-Maschine. Wenn ein Task nur auf die Maus wartet, so sollte er auch keine wertvolle Prozessorzeit beanspruchen. Erst der Druck einer Maustaste im zum Task gehörenden Fenster reißt die Subroutine aus dem Schlaf. (Tis Veugen/rs)

Ein riesiges Basic-Fenster

Wie schon im Leserforum im AMIGA-Magazin, Ausgabe 11/87, erwähnt, ist das normale Basic-Fenster des Amiga nur maximal 187 Punkte groß. Möchten Sie ein größeres Window? Möchten Sie den gesamten PAL-Bildschirm nutzen? Hier ist die Lösung:

Als erstes müssen Sie einen neuen Screen für das Fenster öffnen. Die im Basic-Handbuch angegebenen Höchstgrenzen für den SCREEN-Befehl sind falsch. Selbst auf einem Standard-Amiga lassen sie sich geringfügig überschreiten. Um ein großes Fenster zu programmieren, muß ein Trick angewendet werden:

Verschieben Sie erst mit den »Preferences« den Bildschirm nach links oben. Dabei müssen Sie in Kauf nehmen, daß der Text in der Menüleiste bei manchen Monitoren schlechter zu lesen ist. Experimentieren Sie mit den Einstellungen, bis Sie einen guten Kompromiß gefunden haben. Geben Sie anschließend in Basic dieses Listing ein:

```

SCREEN 1,673,269,1,2
WINDOW 1,"Grafik",(0,0)-(664,255),10,1
WINDOW OUTPUT 1
FOR y=1 to 255
  w=w+1
  FOR x=1 TO w
    PSET (664*RND(1),y)
  NEXT
NEXT y

```

Wenn Sie dieses Programm starten, so sehen Sie eine Grafik-Demonstration mit 664*255 genutzten Punkten — diese Zahl berücksichtigt noch nicht einmal den Rand des Fensters.

(Andreas Imhoff/rs)

Basic-Programme verändern sich

Wollen Sie ein Programm schreiben, um Funktionen zu plotten? So richtig komfortabel arbeitet eine solche Routine, wenn der Benutzer eine gewünschte Funktion während des Programmablaufs über die Tastatur eingeben kann. Immer wieder lassen sich neue Formeln eintippen und anschließend in einem x-y-Diagramm betrachten. Hierzu ist ein Programm erforderlich, das die Funktion als String liest und in den Programmcode einbaut. Für Amiga-Basic ist dies ein Leichtes:

- Lesen Sie die Funktion mit Hilfe des INPUT-Befehls
- Öffnen Sie eine sequentielle Datei
- Speichern Sie die Funktion als ASCII-Datei
- Verknüpfen Sie die Datei und Ihr Programm mit MERGE

```

10 INPUT "Funktion"; FU$
20 FU$ = "500 DEF FNxy(X) = "+ FU$ +":GOTO 80"
30 OPEN "0",#2,"ram:Datei"
40 PRINT #2, FU$
50 CLOSE 2
60 CHAIN MERGE "ram:Datei", 500, ALL
70 GOTO 500
80 INPUT "x";x

```



```

90 PRINT "F("x")="; FNxy(x)
100 DELETE 500
110 GOTO 10

```

Anhand des Beispiels kann jeder Programmierer diese Technik studieren. Die neue Datei ist am besten in der RAM-Disk aufgehoben. Dies erspart Diskettenzugriffe. Die angehängte Zeile können Sie mit DELETE löschen und anschließend durch eine neue Funktion ersetzen.

Bei dieser Methode müssen Sie eines beachten: Speichern Sie das Programm vor dem ersten Durchlauf. Sobald das Programm zum zweiten Mal die MERGE-Anweisung durchführt, meldet sich Amiga-Basic mit der Meldung: »Current Programm is not saved. Do you want to save it before proceeding«. Sie sollten dann in jedem Fall »No« anklicken. (Torsten Kerschat/rs)

Hardcopy-Routine im Programm

Auf der Diskette »Amiga Extras 1.2« befindet sich in der Schublade »BasicDemos« das Programm »ScreenPrint«. Dieses können Sie in Ihren eigenen Basic-Programmen verwenden.

— Entfernen Sie zunächst den oberen Programmteil bis zur Zeile »Screendump«.

— Löschen Sie auch alle REM-Zeilen; das spart Speicherplatz. Lesen Sie die Kommentare in den REM-Zeilen jedoch einmal durch. Hier finden Sie wertvolle Erläuterungen zu »Screendump«.

— Speichern Sie das Listing mit »SAVE " <name> ".a «.

— Mit »MERGE " <name> " « läßt sich die Hardcopy-Routine in jedes Basic-Programm einbinden.

Der Aufruf erfolgt entweder über ein Pull-Down-Menü oder mit Hilfe des Befehls »GOSUB Screendump«. Der Amiga beginnt danach sofort mit dem Ausdruck des gerade sichtbaren Bildschirms. Wichtig: Wenn Sie »ScreenPrint« auf eine andere Diskette kopieren, muß sich auf dieser die Datei »exec.bmap« befinden.

(Ralf Thomas Barding/rs)

Stop! Mauskontrolle

Eine kurzes Listing demonstriert Ihnen, wie Sie die Maus von Basic aus programmieren. Das Programm zeigt die horizontale und vertikale Position der Maus auf dem Bildschirm an, sobald Sie die linke Maustaste betätigen:

```

WHILE 1
  x=MOUSE(2)
  y=MOUSE(3)
  IF MOUSE(0) > 0 THEN PRINT x;y
WEND

```

Diese Schleife können Sie durch Drücken der Tasten <CTRL> und <C> verlassen. (Eike Cornelius/rs)

Wenn Farben rotieren

Die Farben einer Grafik lassen sich auch nach dem Zeichnen mit PALETTE ändern. Erzeugen Sie beispielsweise durch einen zyklischen Farbwechsel den Eindruck einer Bewegung:

```

SCREEN 2,320,200,5,1
WINDOW 2, "Grafik", (0,0)-(310,310),0,2
FOR x=-150 TO 100 STEP 1
  FOR y=-100 TO 100 STEP 1
    IF y < > 0 THEN f=ABS((x^2/y):ELSE f=1
    test: IF f>31 THEN f=f/2: GOTO TEST
    PSET (x+150,y+100),f
  NEXT y
NEXT x
TauscheFarben:
FOR t=1 TO 29
  PALETTE f+2, RND, RND, RND
NEXT t
GOTO TauscheFarben

```

Diese Demo verwendet Zufallszahlen zur Berechnung der neuen Farben. Berechnen Sie den Farbwechsel doch einmal durch einen eigenen Algorithmus. Verwenden Sie auch selbstgemalte Bilder. (Volker Hauße/rs)

Tips und Tricks zum CLI

Die Arbeit mit dem CLI (Command Line Interface) ist gegenüber der Workbench effektiver und leistungsfähiger. Doch hat dies seinen Preis: Die Befehlseingabe ist komplizierter. Unsere Tips rund um das CLI erleichtern Ihnen den Umgang mit dieser Befehlsebene.

Vollgas durch ADDBUFFERS

Einen Geschwindigkeitsvorteil bei Diskettenzugriffen erhalten Anwender auch durch den Befehl »ADDBUFFERS«. Arbeitet man mit einer Diskette und möchte von Zeit zu Zeit einmal das Directory ansehen, ist dies mit viel Wartezeit verbunden. Schneller geht es mit dem Befehl ADDBUFFERS. Er überläßt dem angegebenen Laufwerk mehr Speicher. Wird DIR eingegeben, liest der Amiga zuerst die Daten von der Disk. Gibt man nun erneut DIR ein, werden die Daten aus dem Puffer gelesen. Günstige Puffergrößen für den Befehl liegen zwischen 50 und 100. Bei Arbeiten mit der Workbench werden auch Icons beim zweiten Mal wesentlich schneller angezeigt. ADDBUFFERS installiert eine Art RAM-Disk für das/die Laufwerk(e). Syntax:

AddBuffers Laufwerk: 100

Dem angegebenen Laufwerk werden zirka 50 KByte mehr Speicher zugewiesen. Der Amiga gibt den vergebenen Speicher nur nach einem Reset zurück. (Norbert Cohen/rs)

Nur ein Zylinder

Wenn man sich mit dem CLI beschäftigen möchte, aber nur ein Laufwerk hat, ist es ratsam die CLI-Befehle, die der Amiga normalerweise aus dem Verzeichnis »C« liest, in der RAM-Disk abzulegen. Hierzu schreibt man eine Befehlsdatei (Batch-File), in der alle notwendigen Anweisungen stehen. Dies geschieht mit der Anweisung: »ED RAM-Disk« <RETURN>. Dieser Befehl startet den Editor. Geben Sie nun nacheinander folgende Befehle ein. (RETURN nach jeder Zeile nicht vergessen):

```

PROMPT "RAM %n> "  Ändert die CLI-Bereitschaftsanzeige
MAKEDIR ram:c      ' Richtet C-Directory im RAM ein
COPY c/cd to ram:c ' Kopiert den Befehl CD in die RAM-Disk
COPY c/dir to ram:c ' Kopiert den Befehl DIR in die RAM-Disk
COPY c/...          ,

```

Jeder sollte in der Liste die Befehle ergänzen, von denen er meint, daß sie benötigt werden. Dabei muß man sich darüber im Klaren sein, daß jeder überflüssige Befehl im RAM kostbaren Speicher verbraucht.

PATH ram:c

Der letzte Befehl veranlaßt den Amiga, Befehle zuerst in der RAM-Disk zu suchen. Wer viele Befehle kopiert, faßt mehrere in einem Arbeitsgang zusammen:

```
COPY df0:c/(COPY | ECHO | TYPE | DIR | LIST) ram:c

```

Der Editor wird durch die Tastenkombination <ESC> <X> <RETURN> verlassen. Ist die Diskette nicht schreibgeschützt, wird die Datei »RAM-Disk« gespeichert. Wenn die RAM-Disk nun durch »Execute RAM-Disk« aktiviert wird, ist der Anwender in der Lage, sich auch auf anderen Disketten umzusehen. Er ist nicht mehr von der Arbeits-Diskette abhängig. Der Aufruf der RAM-Disk kann natürlich gleich in der »Startup-Sequence« untergebracht werden.

Sollten Sie einmal einen Befehl aufrufen, der nicht im RAM vorhanden ist, sucht der Amiga im C-Directory der Startdiskette. Ist diese gerade nicht eingelegt, macht der Amiga durch einen Requester auf das Versäumnis aufmerksam. (rs)

Der schnellste Weg

Drücken Sie während des Bootens Ihrer Workdisk <CTRL D>. Das unterbricht die Ausführung der »Startup-Sequence«. Der Amiga arbeitet nämlich nach dem Starten automatisch im CLI. Er führt nach dem Booten diese spezielle Befehls-Datei aus, an deren Ende ein Sprung in die Workbench-Ebene steht. Wer einmal im CLI arbeitet, kann die Datei anschauen. Sie steht im Verzeichnis »S«:

```
TYPE s/startup-sequence
```

TYPE ist einer der Befehle des CLI. Er gibt eine Textdatei auf dem Bildschirm aus. Wenn der Ausstieg aus der Startsequenz allerdings zu früh erfolgte, arbeitet der Amiga noch mit dem englischen Tastaturrechner. Dies läßt sich durch die Eingabe eines weiteren Befehls im CLI ändern:

```
System/Setmap D
```

Eine allgemeine Beschreibung der CLI-Befehle und des wichtigen Bildschirmditors ED finden Sie im großen CLI-Kurs in diesem Heft oder dem AmigaDOS-Handbuch. (rs)

Einsteiger-Tips ...

Das erste, womit der Amiga-Anwender beim Laden einer Workbench- oder einer anderen, boot-fähigen Diskette konfrontiert wird, ist ein Fenster (Window) mit der Startmeldung. Erst später erscheint die Benutzeroberfläche mit den Diskettensymbolen. Die Workbench selbst ist nämlich auch nur ein Programm, das mit dem CLI-Befehl LOADWB geladen wird. Dieser Befehl steht in der »s/Startup-Sequence«-Datei, die sich auf jeder bootfähigen Diskette befindet. Drücken Sie beim Boot-Vorgang einmal die Tastenkombination <CTRL D>, sobald das Window auf dem Bildschirm erscheint. Es wird »***Break***« und das sogenannte Prompt ausgegeben. In unserem Fall ist das ein »1>«. Wir befinden uns im Command Line Interface (CLI) des Amiga.

Das Prompt zeigt dem Anwender die Bereitschaft des Computers zu Tastatureingaben. Zusätzlich steht im Prompt vor dem Größerzeichen noch die Nummer des laufenden CLI-Task. Wir haben es bekanntlich mit einem Multitasking-System zu tun. Tippen Sie nun den Befehl NEWCLI ein — ein zweites Window erscheint, diesmal mit einem Prompt »2>«. Sie können jetzt mit der Maus zwischen beiden CLIs wechseln (fahren Sie mit der Maus in das gewünschte CLI-Fenster und drücken Sie die linke Maus-Taste). Das ist sehr praktisch, wenn man beispielsweise mit dem DIR-Befehl das Disketteninhaltsverzeichnis auf das erste CLI-Window gelegt hat, während im zweiten CLI die nötigen Files mit »Copy« in eine RAM-Disk kopiert werden können.

Tippen Sie nun LOADWB ein. Damit haben Sie per CLI (das auch nur ein Teilprogramm des Amiga-DOS ist) das Workbench-Programm gestartet. Die Workbench wiederum ist ein Programm, das die Routinen der Benutzeroberfläche »Intuition« und der Systembibliothek »Icon« nutzt. Nach dem Aufruf der Workbench haben Sie (wenn Sie sich die Windows nach Wunsch entsprechend auf dem Bildschirm einrichten) nun die volle Leistungsfähigkeit eines Kommando-Interpreters und einer grafischen Benutzeroberfläche gleichzeitig. Diskettennamen ändern geht per Workbench schneller und komfortabler, wohingegen das CLI für verschiedene andere Arbeiten einfacher zu bedienen ist. Bleiben wir erst einmal beim CLI, doch ohne die Workbench zu vernachlässigen. Sehen wir uns einmal mit TYPE S/STARTUP-SEQUENCE an, was beim Booten passiert. Auf dem Bildschirm erscheint eine Reihe von Befehlen, unter anderem auch LOADWB und ENDCLI. Die

Befehle, die in der Startup-Sequence stehen, werden beim Booten der Reihe nach ausgeführt. Natürlich können wir jederzeit die Befehlsfolge ändern, die beim Laden ausgeführt werden soll.

Mit der Befehlsfolge ED S/STARTUP-SEQUENCE rufen wir den Texteditor ED auf und finden in einem großen Window unsere Befehlsfolge wieder. Hier können wir nach Belieben Befehle ändern, hinzufügen oder entfernen. Entfernen wir doch einmal nur den ENDCLI-Befehl, indem wir den Cursor auf seine Zeile steuern und <CTRL B> (Zeile löschen) drücken. Mit der Escape-Taste (ESC) kommen wir in eine Befehlszeile, in der wir — nach Entfernung des Schreibschutzes von der Diskette — ein »X« eingeben und <RETURN> drücken. Die so geänderte Startup-Sequenz löscht uns bei einem Neubooten der Workbench nicht jedesmal den CLI, so daß wir immer gleich die Vorteile beider Benutzerschnittstellen gleichzeitig verwenden können. Ein Window mit Icons sagt uns nämlich nicht immer, welche Programme sich noch auf einer Diskette befinden. Oder hätten Sie als Einsteiger gewußt, daß es den Editor ED auf Ihrer Workbench-Diskette gibt?

Der erste Befehl, den man im Command-Line-Interface verwendet, ist in aller Regel der DIR-Befehl. Mit DIR OPT A kann man sogar das lästige einzelne Listen der Subdirectories vermeiden. Nur manchmal stört es ziemlich, wenn zusammengehörige Files ungeordnet — eines links, eines rechts — im Bild erscheinen. LIST QUICK schafft Abhilfe, alle Dateien erscheinen schön geordnet untereinander. (Manfred Kohlen/rs)

... und Profi-Tips

Haben Sie schon mal überlegt, warum Sie nach dem falschen Eintippen einer Zeile im CLI Ihre Backspace-Taste laufend so strapazieren? Wenn man schon früher gewußt hätte, daß <CTRL X> genügt, um die ganze Zeile zu löschen!

Und wenn Sie gleich eine ganze Reihe von — richtig eingetippten — Befehlen ausführen lassen wollen, brauchen Sie nicht unbedingt alles einzeln eintippen und auf die Bearbeitung jedes Befehls zu warten. Für eine automatische Abarbeitung der CLI-Befehle gibt es nämlich drei Möglichkeiten — eine davon findet man in keinem Handbuch, nicht einmal in der Entwickler-Dokumentation.

Möglichkeit 1: Häufig benutzte Befehlsfolgen legt man mit dem Editor ED in einem File ab, welches mit dem EXECUTE-Befehl aufgerufen werden kann und dann selbständig seine Arbeit verrichtet. Bei sehr häufiger Verwendung des EXECUTE-Befehls empfiehlt sich, den Befehlsnamen im C-Directory per RENAME zu kürzen. Profis rufen eine Datei zum Kopieren der DOS-Befehle ins RAM nur noch mit »XY« auf (X für EXECUTE und Y als Filenamen). Die Amiga-DOS-Datei Y sollte dann alle Befehle zum Kopieren der wichtigsten Befehle ins RAM enthalten.

Möglichkeit 2: Die zweite Methode, Befehlsfolgen aufzurufen, findet sich im englischsprachigen DOS-Manual oder dem deutschen Amiga-Handbuch (Markt&Technik Verlag): Mit RUN wird ein Task gestartet, der die Befehle ausführt, die man per Pluszeichen und RETURN voneinander trennt. Das sieht dann so aus:
RUN COPY Y TO PRT+
LIST+
NEWCLI

Das CLI überprüft die Eingabe auf das Pluszeichen und startet den Task erst, wenn ein Befehl ohne Pluszeichen eingegeben wird.

Möglichkeit 3: Jedesmal das RUN und die Pluszeichen einzugeben, ärgert sehr, wenn man nicht unbedingt einen Task dafür braucht. Abhilfe schafft die Tastenkombination <CTRL J>. Tippen Sie einfach den ersten Befehl der Folge (ohne RUN), drücken dann aber anstatt der RETURN-Taste <CTRL J>. Der Cursor ist in der nächsten Zeile, der Befehl wird aber noch nicht ausgeführt. Jetzt den nächsten Befehl eingeben, besagte Kontrollfunktion, und so weiter, bis zum letzten Befehl. Erst jetzt <RETURN> drücken. Probieren Sie es aus!

Nun stellen Sie sich einmal vor, in der obersten Zeile des CLI-Windows steht ein wichtiger Text, zum Beispiel ein jetzt benötigter Dateiname oder ähnliches. Sie müssen aber unbedingt noch etwas eintippen, sind dummerweise in der letzten Zeile des CLI-Windows angekommen und würden den obenstehenden Text bei einer weiteren Befehlseingabe verlieren. Wenn Sie Glück haben und Ihr CLI-Window noch nicht die volle Größe hat, können Sie dadurch ein Wegscrollen des Textes verhindern. Ansonsten gibt es

noch die Tastenkombination <CTRL K>: Der Cursor wandert eine Zeile nach oben. Auch wenn in der jetzt neuen Eingabezeile schon etwas stehen sollte, wird vom Amiga-DOS nur das übernommen, was Sie jetzt eingeben. Natürlich können Sie damit nicht zum Beispiel ein paar Zeilen höher gehen, um dort eine falsche Eingabe zu verbessern. Das CLI ist und bleibt ein Zeileneditor. Mit <CTRL J> wandert der Cursor übrigens genauso wieder nach unten, wie er das aufwärts auch tat — ohne Texte mit neuen darübergetippten Befehlen unkenntlich zu machen.

Am Ende solcher Kontrolloperationen und vor einer neuen Befehlseingabe empfiehlt sich übrigens dringend die Tastenkombination <CTRL X>. Manchmal versteht das CLI verschiedene Kombinationen mit der CTRL-Taste nämlich als Buchstaben, und einen Befehl namens »<CTRL G>-DIR« gibt es bekanntlich nicht. (Manfred Kohlen/rs)

CLI im Rausch der Farben

Vielleicht haben Sie sich schon einmal gewünscht, die verschiedenen Schrifttypen (Kursiv, Fett, Invers und Unterstrichen) in den Bildschirmausgaben der CLI-Befehle verwenden zu können? Das Einstellen der Vordergrund- und Hintergrundfarbe für Ausgaben des CLI wäre auch von Vorteil.

Commodore erwähnt in dem ohnehin nicht üppigen DOS-Manual mit keiner Silbe, daß so etwas möglich ist. Man muß lediglich in die Ausgabebefehle (ECHO, PROMPT, FILENOTE und andere) eine Escape-Sequence einbauen und schon kann man alle Schrifttypen in den verschiedensten Farben benutzen. Der Aufbau einer solchen Sequenz sieht so aus:

SEZ Schrift [: Vordergrund [: Hintergrund]:]m
Dabei bedeutet SEZ die Sequenz-Einführungszeichen (*,e,)]. Hinter den Einführungszeichen wird die Schriftart gewählt:

- 0 = normal
- 1 = Fettdruck
- 3 = Kursiv
- 4 = Unterstrichen
- 7 = Invers

Für die Vordergrundfarbe sind folgende Parameter erlaubt:

- 30 = normal
- 31 = weiß
- 32 = schwarz
- 33 = orange

Und schließlich für die Hintergrundfarbe:

- 40 = normal
- 41 = weiß
- 42 = schwarz
- 43 = orange

Als Endkennzeichen fungiert das Zeichen <m>. Auch die Kombination von verschiedenen Schriftarten ist durch mehrfache Angabe solcher Sequenzen möglich.

Für die Ausgabe eines unterstrichenen kursiven Textes in Orange auf weißem Grund ist also einzugeben:

ECHO "e[3;4;33;41m Test-Text *e[0m"

Wie im Beispiel sollte man die Schriftart am Schluß wieder mit der Folge »*e[0m« auf normal einstellen. Übrigens wirken diese Einstellungen auch auf den Drucker, wenn man die Ausgabe mit >prt: (Drucker am Parallelport) oder >ser: (serielle Schnittstelle) in derselben Anweisung umlenkt. (Michael Baas/rs)

Hilfe bei vollen Disketten

Die Disketten von Amiga haben zwar eine Kapazität von 880 KByte, aber hin und wieder wird auch diese Grenze erreicht. Für Besitzer eines zweiten Laufwerks lassen sich mit Hilfe folgender Vorgangsweise einige KBytes wiedergewinnen:

1. Formatieren Sie eine neue Diskette (hier im Laufwerk 1) mit Hilfe des CLI-Befehls »format drive df1: name DiskName noicons« (setzen Sie bitte für DiskName den gewünschten Diskettennamen ein).

2. Kopieren Sie alle Dateien mit der Anweisung »copy df0: to df1: all« auf die frisch formatierte Diskette.

Sie werden nach der Anwendung von »info df1:« sehen, daß wesentlich weniger Speicherplatz auf Ihrer Kopie belegt ist. Obendrein werden die Directory-Einträge noch schneller gefunden.

(Paul Scholda/rs)

»CLI« reagiert auf Kontrollsequenzen

Welche Funktionen lösen sogenannte Kontrollsequenzen (Taste <CTRL> gleichzeitig mit einer anderen Taste drücken) bei der Arbeit mit dem CLI aus? Wir haben eine Auswahl für Sie zusammengestellt:

CTRL-B: Entspricht einem Anklicken des Cancel-Feldes im Auto-requester

CTRL-C: Abbruch eines Befehls, zum Beispiel DIR

CTRL-D: Ausführung einer Batchdatei (Startup-Sequence) abbrechen

CTRL-G: Bildschirmblitz (kann auch mit einer ECHO-Anweisung ausgelöst werden)

CTRL-H: Delete

CTRL-J: Tabulator

CTRL-K: Carriage Return (keine ausführende Wirkung!)

CTRL-L: löscht den Bildschirm

CTRL-M: Entspricht der Enter- oder Return-Taste

CTRL-N: Aufruf des alternativen Zeichensatzes. Dieser bleibt bis <RETURN> oder <CTRL-Q>.

CTRL-Q: Ausschalten des alternativen Zeichensatzes

CTRL-V: Entspricht dem Anklicken des Retry-Feldes im Auto-requester

CTRL-X: löscht die aktuelle Zeile

(Lutz Beyert/rs)

Kurz und bündig: Fenster ohne Rahmen

Für das Problem »CLI-Fenster ohne Rahmen« gibt es eine einfache und kurze Lösung.

Um statt der normalen 77 die vollen 80 Zeichen Breite zu nutzen, läßt sich der Rahmen eines CLI-Fensters einfach mit den ESC-[Befehlen ausschalten. Escape-Befehle werden normalerweise zur Druckersteuerung verwendet. Beim Amiga werden sie vor der Ausgabe an den in den Preferences eingestellten Drucker angepaßt. Diese Anpassung wird aber auch bei der Ausgabe im CLI-Fenster durchgeführt. Wird beispielsweise von einem Programm in einem CLI-Fenster die Sequenz <ESC[4m> ausgegeben, erscheinen danach alle Texte unterstrichen. Dies ist auch über die Tastatur möglich. Drücken Sie nacheinander die Tasten <ESC>, <[>, <4> und <m>. Jetzt ist Unterstreichen eingeschaltet. Die heißesten Tips für Insider finden Sie unter der Überschrift »Druckeransteuerung« einen Teil der weiteren Befehle. Es gibt aber nicht nur Befehle für die Einstellung der verschiedenen Darstellungsarten, sondern auch für die Formatsteuerung. Hierzu einige Beispiele:

Tastenkombination	Bedeutung von n
ESC [n u	Breite in Zeichen
ESC [n x	Linker Rand in Pixeln
ESC [n y	Abstand zur Oberkante des Fensters
ESC [n t	Anzahl der Zeilen
ESC c	normales Format

Wie Sie aus der Tabelle sehen, kann der gesamte Bildschirm durch wenige Eingaben genutzt werden. Bringen Sie zunächst das CLI-Fenster mit der Maus auf die Größe des Bildschirms. Mit dem Befehl »ESC [80u« wird die Zeilenbreite auf 80 Zeichen gestellt und der linke Rand mit »ESC [0x auf die äußerste linke Position gesetzt. Um in dem großen CLI-Fenster aufzuräumen, kann es mit <CNTL L> gelöscht werden. Sofort verschwindet auch der Rand des Fensters. Schließlich sollten Sie noch <Return> drücken, damit die Formatbefehle nicht als Teil des nächsten CLI-Befehls interpretiert werden.

Ab jetzt wird bei der Textausgabe das gesamte Fenster benutzt. Wünschen Sie wieder das alte Format, genügen die Tasten <ESC c> <Return>. Jeder, der seinen Bildschirm häufig auf diese Art vergrößern will, kann die benötigten Escape-Sequenzen auch in zwei Textdateien »Rahmen-Aus« beziehungsweise »Rahmen-Ein« zusammenfassen. Dann genügt der einfache Aufruf »TYPE Rahmen-Aus«

(Winfried Stappert/rs)



Amiga Basic

Basic ist noch immer einer der schnellsten und einfachsten Wege zur Programmierung. Doch Umsteiger von anderen Heimcomputern werden merken, daß Microsoft Basic für den Amiga leistungsfähiger ist. Und da das von Commodore mitgelieferte Basic-Handbuch eigentlich nur weiter fortgeschrittenen Basic-Programmieren von Nutzen ist, wünscht sich der Anfänger ein verständliches Buch, das ihn Schritt für Schritt mit diesem Basic-Dialekt vertraut macht. Das sehr umfangreiche Buch enthält fast alles, was Einsteiger und Aufsteiger wissen sollten. Das Spektrum reicht von Grafik bis hin zu Sprach- und Musikausgabe. Der Leser wird dabei durch eine verständliche und humorvolle Sprache bei guter Laune gehalten. Die braucht er auch, denn es kommen auf ihn zahlreiche und vor allem umfangreiche Listings zu. Doch für »Profis« sind die Beispielprogramme oft zu umfangreich. Dagegen ist der über 200 Seiten starke Anhang des Buches wirklich jedem zu empfehlen. Neben Erklärungen und Abhilfen zu allen Fehlermeldungen ist ein umfangreicher Basic-Referenzteil vorhanden. Die Autoren haben die einzelnen Kapitel des Buches gut miteinander verknüpft. Daten aus einer Dateiverwaltung können beispielsweise im darauffolgenden Abschnitt grafisch dargestellt werden. Die Verfasser setzen sich dabei erfreulicherweise stark für den IFF-Standard ein.

Der positive Eindruck wird abgerundet durch ein kleines Fachwortlexikon, das dem Einsteiger mit Sicherheit einige Fragen beantworten wird.

(Bernhard Carli/rs)

Rügheimer, Spanik, AMIGA BASIC, Data Becker GmbH, 774 Seiten, ISBN 3-89011-209-9, 59 Mark

Das können AMIGA 500 & 2000

Dieses Buch bietet sich für den Einsteiger an, der mehr über den Amiga wissen will. Durch ein gutes Konzept findet der Leser schnell die Stellen im Buch, die ihn interessieren. Auf den ersten zehn Seiten sind die einzelnen Kapitel durch Kurzbeschreibungen vertreten, in denen jeweils nur das Wichtigste zu den Themen steht. Diese Erklärungen sind in zwei Spalten aufgeteilt, wobei links die Erläuterung für Einsteiger und rechts die für Profis steht. Diese Seiten können später zum Nachschlagen verwendet werden, aber der eigentliche Sinn liegt in den Seitenangaben am Ende der Spalte, die auf die ausführlichen Kapitel verweisen.

In den folgenden Abschnitten werden nun die Fähigkeiten des Amiga erläutert, ohne



allerdings auf dessen Programmierung einzugehen. Durch die zahlreichen Hinweise und Beschreibungen von vorhandenen Programmen zeichnet sich das Buch als Kaufhilfe aus, die noch Unentschlossene unterstützen soll. Auch die vielen Bilder sind eine gute Hilfestellung bei der Auswahl des richtigen Programms.

Von Anwenderprogrammen für Grafik, Animation und Musik über Programmiersprachen bis zu Hardware-Erweiterung findet der Leser viele Beispiele, die ihm bei seiner Kaufentscheidung helfen.

Alles in allem ein Buch, das für denjenigen geeignet ist, der sich einen Computer kaufen will und noch nicht recht weiß, ob der Amiga der richtige Computer für ihn ist oder nicht.

(rs)

Stein, Das können AMIGA 500 & 2000, Data Becker GmbH, 190 Seiten, ISBN 3-89011-178-5, 29 Mark



Amiga Programmier-Handbuch

Die Autoren dieses Handbuchs sind offensichtlich mit dem Ziel angetreten, Ordnung in die große Menge der Funktionen und Strukturen der Amiga-Bibliotheken zu bringen. Sie stellen deshalb die einzelnen Funktionen der Bibliotheken in der Reihenfolge Syntax (Format des Funktionsaufrufes), Funktion (was macht die Funktion), Parameter (welche Parameter werden benötigt), Ergebnis (welche Werte gibt die Funktion zurück), Datentyp (welche Variablen müssen wie deklariert werden), Sonstiges (zusätzliche Bemerkungen, Verweise auf andere Funktionen) vor. Zusätzlich sind die besprochenen Funktionen und Strukturen in Themenbereiche gegliedert. So finden sich alle Funktionen und Strukturen, die mit »Screens« verknüpft sind, unter dem Titel »Der Screen«. Weitere Themenbereiche sind: Windows, Zeichnen in Screens und Windows, Programmbedienung, Menüs, Gadgets, Systemmeldungen, Ein- und Ausgabe, Drucker, Workbench, Sprachausgabe, Multitasking, Mathematik-Libraries und das IFF-Format. Die Autoren verdeutlichen die vorgestellten Funktionen anhand vieler Beispielprogramme. Bei der Vielfalt der Themenbereiche können Spezialbegriffe wie etwa Multitasking nur kurz dargestellt werden.

Durch die übersichtliche Struktur ist dieses Handbuch als Nachschlagewerk gut geeignet. Es kann allen Anwendern empfohlen werden, die sich mit der Programmierung ihres Amiga auseinandersetzen.

(Ingolf Krüger/rs)

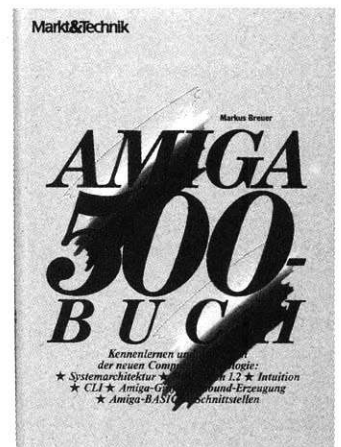
Frank Kremser/Jörg Koch, Amiga Programmier-Handbuch, Markt und Technik Verlag AG, ISBN 3-89090-491-2, Preis: 69 Mark

Amiga 500-Buch

Das »Amiga 500-Buch« läßt sich grob in drei Teile gliedern. Im ersten Teil des Buches beschreibt der Autor die wesentlichen Grundzüge der Bedienung des Amiga. Durch diese Anleitung und praktische Übungen wird der Leser vertraut gemacht mit der grafisch orientierten Bedienung, der Anwendung der Maus und allen wichtigen Handgriffen, die er später tagtäglich benötigt. Die einfache Bedienung des Amiga ist im wesentlichen eine Leistung der Workbench. Und so beschäftigt sich der erste Teil auch ausführlich mit diesem Programm.

Der zweite Buch-Teil befaßt sich mit dem sogenannten CLI. Diese zweite Benutzerschnittstelle des Computers hat trotz der umständlicheren Handhabung ihre Vorteile. Sie wird meist von den Programmierern bei intensiver Nutzung der Betriebssystemkommandos verwendet.

Der dritte und letzte Teil wendet sich schließlich an diejenigen Leser, die neben Grundlagenwissen tiefer in die Materie einsteigen wollen. Der Autor beschreibt hier die Hardware des Amiga. Er zeigt viele Einsatzmöglichkeiten und beschreibt die Kontakte des Computers zur Außenwelt (Anschlüsse für Drucker und andere Zusatzgeräte). Die Programmierer unter den Lesern — und solche die es noch werden wollen — lernen anhand von ein paar kleinen Beispielprogrammen in Amiga-Basic,



was man schon mit dieser einfachen und leicht erlernbaren Sprache, die ja jeder Besitzer beim Kauf mitgeliefert bekommt, auf dem Amiga anfangen kann. (Joschy Polierer/rs)

Markus Breuer, Amiga 500-Buch, Markt & Technik Verlag AG, 490 Seiten, ISBN 3-89090-522-6, Preis: 49 Mark

Das Amiga 2000 Buch

Markus Breuer, bereits bekannt durch seine Bücher zum Amiga 500/1000, hat sein Werk über den Amiga 2000 ganz dem Commodore-Flaggschiff gewidmet. Gegenüber seinem ersten Werk ist dieses Buch im Umfang um rund 150 Seiten auf insgesamt über 600 Seiten angewachsen. In die vier Themenbereiche Workbench, Amiga-DOS, MS-DOS und die Programmstrukturierung gegliedert, gibt der Autor sowohl dem Anfänger als auch Fortgeschrittenen ausführliche Hilfestellung für den Umgang mit ihrem Computer.

Wenn Sie als Neuling auf diesem Gebiet Ihren Amiga gerade erst erworben haben, finden Sie im ersten Teil zahlreiche grundlegende Informationen über Ihr neues Werkzeug. Nach kurzer Zeit sind Begriffe wie Intuition, Workbench, OSA, RAM und DOS keine Fremdworte mehr. Dieser erste Teil des Buches entspricht in weiten Teilen allerdings dem Einführungskapitel im »Amiga 1000 Buch«. Einige Einzelheiten sind neu hinzugekommen oder nochmals überarbeitet worden.

Nach dem Einstieg in die Bedienung des Amiga über die Workbench folgt mit der Betriebssystemebene eine Einführung in die Kommandoeingabe über das CLI. Den größten Teil dieses Kapitels nimmt die alphabetische Auflistung

und Erklärung der DOS-Befehle ein. Es läßt sich gut als Nachschlagewerk nutzen. Im Anschluß daran erhalten Sie eine Vielzahl nützlicher Tipps und Tricks zum CLI. Wie beim Buch zum Amiga 500 ist den ausführlich beschriebenen Ausbaumöglichkeiten des Amiga 2000 ein weiteres Kapitel gewidmet. Hier steht Wissenswertes über Zubehör und Steckkarten. Eine Entscheidungshilfe also für alle diejenigen, die an den Ausbau ihres Amiga denken.

Die Beschreibung der Zusammenarbeit zwischen dem Amiga und der PC-beziehungsweise MS-DOS-Steckkarte dürfte auch alle Besitzer des Amiga 1000 mit Sidecar interessieren. In diesem Teil geht es um den Datentransfer zwischen den beiden Betriebssystemen.

Mit nützlichen Informationen am Ende — der Erklärung verschiedener Hardwarebegriffe, Fehlermeldungen und Vorstellung nützlicher Programmiersprachen — schließt dieses Buch. Es sollte bei jeder Neuanschaffung einem Amiga 2000 beiliegen. Solange dies nicht der Fall ist, lohnt sich für den engagierten Amiga-Anwender, selbst wenn er »nur« an den Ausbau des Modells 1000 mit dem Sidecar oder der Zorro-Box denkt, in jedem Fall eine Anschaffung.

(Joschy Polierer/rs)

Markus Breuer, Amiga-2000-Buch, Markt & Technik Verlag AG, 570 Seiten, ISBN 3-89090-574-9, Preis 59 Mark

Commodore Sachbuch: Programmieren mit Amiga-Basic

Wie in der Computerbranche fast schon üblich, fällt beim Amiga die dem Computer mitgelieferte Dokumentation äußerst mager aus. Sie enthält zwar eine Beschreibung der Basic-Befehle, eine für den Einsteiger gerechte Einführung in die Programmierung sucht man jedoch vergeblich. In diese Lücke stößt »Programmieren mit Amiga-Basic« aus dem Verlag Markt & Technik. Mit Hilfe dieses Buches gewinnt der Leser recht schnell einen Einblick in den Aufbau und die Arbeitsweise des Amiga-Basic.

Nach einem einführenden Teil, der sich mit der Bedienung des Editors auseinandersetzt, lernt der Leser im ersten Kapitel grundsätzliche Techniken der Programmierung. Das erstmalige Auftreten eines Be-



fehlswortes wird durch einen grau unterlegten Kasten, der Kurzinformationen über Format und Funktion des Befehls enthält, hervorgehoben. Dies ist eine nützliche Hilfe für den fortgeschrittenen Leser, wenn

er bereits gelesene Informationen noch einmal nachschlagen möchte.

Ein weiteres Kapitel muß sich natürlich mit einer Stärke des Amiga auseinandersetzen: der Grafik. Ein komplettes Programmbeispiel der in diesem Abschnitt beschriebenen Techniken ist das Malprogramm »PAINT-Amiga«. Wie auch alle anderen Beispiele befindet sich das Programm auf der beigelegten Diskette. Das erspart dem experimentierfreudigen Leser ein langwieriges und fehlerträchtiges Eintippen der Listings.

Für die Freunde von Datenlisten und Adreßverzeichnissen ist das Kapitel »Dateien« besonders interessant. Aber auch das Format der IFF-Grafikdateien wird hier ausführlich beschrieben. Natürlich darf die Behandlung von Sprites, Bobs und Sound in einem solchen Grundlagenbuch nicht fehlen. Gerade hier wird die Nützlichkeit der mitgelieferten Diskette deutlich: Denn für die Erstellung von Sprites und Bobs werden entsprechende Dateien benötigt, die durch die mitgelieferten DATA-Dateien nur sehr zeitaufwendig eingetippt werden können.

Mit einer Übersicht aller Basic-Befehle, einer Liste aller im Buch dargestellten Programme und einer vollständigen Dateiverwaltung einschließlich Dokumentation, Programmablaufplan und Variablenliste schließt dieses Werk, das dem Einsteiger in die Programmierung des Amiga mit ruhigem Gewissen empfohlen werden kann. Die ersten Schritte auf diesem Gebiet fallen mit dieser Unterstützung leichter.

(Wolf-Gideon Bleek/rs)

Horst-Rainer Henning, Programmieren mit Amiga-Basic, Commodore Sachbuch (Markt & Technik Verlag), 363 Seiten, ISBN 3-89090-434-3, Preis: 59 Mark

Das große Buch zu Amiga-DOS

Dieses Werk bietet eine ausführliche Einführung in den Umgang mit dem CLI, der zweiten Benutzerschnittstelle des Amiga, sowie Informationen für den fortgeschrittenen DOS-Anwender.

In der Einführung des in acht Teile gegliederten Buches gehen die Autoren auf die Aufgabe des Amiga-DOS, das Öffnen eines CLI-Fensters und die Help-Funktionen ein. Der zweite Teil befaßt sich mit den einzelnen CLI-Kommandos. Hinweise zum täglichen Umgang

mit dem CLI bietet der dritte Teil. Die Teile vier bis sechs behandeln die Themen Batchprocessing, Multitasking und interne DOS-Bibliotheken.

Eine Reihe neuer DOS-Befehle zum Abtippen befindet sich in Form von DATA-Listings im siebten Abschnitt. Der achte Teil schließlich enthält eine Kurzerklärung der DOS- und Editor-Kommandos zum Nachschlagen.

Der Neuauflage des Data-Becker-Buches (1. Auflage 1988) liegt nun auch eine Diskette mit den vorgestellten Beispiel-Listings bei. Diese Auflage enthält als eine der ersten Veröffentlichungen Hinweise zu der von Commodore möglicherweise in Zukunft auslieferbaren Kickstart-Version 1.3.

Zu jedem DOS-Befehl wird eine Unterscheidung der alten Version 1.2 und der zukünftigen Änderung beim Kickstart 1.3 gegeben.

Die Autoren erklären die einzelnen Befehle anhand vieler Beispiele. Hier unterscheidet sich das Amiga-DOS-Buch von anderen Büchern zum selben Thema. Die Befehle werden ausführlich erklärt und nicht in der sonst üblichen, tabellarischen Form beschrieben. Dies ist für den Leser sehr angenehm, kann man dieses Buch dadurch wirklich »lesen« und nicht nur als reines Nachschlagewerk verwenden.

Im von vielen Autoren des Becker-Verlages gewohnten, lässig saloppen Schreibstil springen die Autoren von Thema zu Thema. Der Einsteiger gerät dabei in Gefahr, einen aufgenommenen Faden zu verlieren. Ungünstig gewählte Beispiele tragen dazu bei. So wird die Funktion der Unterverzeichnisse und Pfade am Beispiel einer Festplatte erklärt, die selten gleich mit dem Amiga zusammen erworben wird. Viele der angeschnittenen Probleme und Themen können von den Autoren selbst nicht ausreichend erklärt werden oder sind nur unzureichend erprobt. Beim Lesen der Kapitel Multitasking und Multiuserbetrieb kann man sich manchmal dem Eindruck nicht erwehren, daß es sich hierbei um noch unerforschte Gebiete handelt. Dennoch bietet das Buch für experimentierfreudige Computerfans eine Reihe interessanter Anregungen.

(Joschy Polierer/rs)

Das große Buch zu Amiga-DOS, Kerkhof/Tornsdorff/Zoller, Data Becker GmbH, 310 Seiten, Preis 59 Mark, ISBN 3-89011-306-0

Fortsetzung von Seite 12

Programname:	Suremosch.boot
Computer:	Amiga 500, 1000, 2000 mit Kickstart 1.2
Sprache:	Amiga-Basic

```

1 9u0 ' Suremosch-Lader, schafft freien Speicher und laedt das
      Hauptprogramm
2 VF  CLEAR ,100000&
3 hv  SCREEN 2,318,246,4,1
4 5g  WINDOW 1,"Suremosch V1.2 (C) 1988 by C.Rodemeyer",(0,0)-(30
      9,232),0,2
5 ey  WIDTH 80,20
6 fo  RESTORE Palette.daten
7 lJ  FOR n=0 TO 15
8 jq2  READ r,g,b
9 EO  PALETTE n,r,g,b
10 l60 NEXT n
11 WT  LOCATE 12,9:PRINT "Suremosch wird geladen!"
12 yB  CHAIN "Suremosch"
13 Pt  Palette.daten:
14 l0  DATA 0,0,0,1,1,1,.26,.4,.26,.4,.73,.4,.73,1,.73,.86,.13,.13
      ,1,0,.53,.66,0,1,.2
15 lV  DATA .73,.93,.2,.4,1,.33,.4,.53,.13,.13,.86,1,.4,.13,.4,.53
      ,.66,.13,.26,.33,.66,.86,1
(C) 1987 M&T

Listing 1. »Suremosch.Boot«.
Das Ladeprogramm für Suremosch.

```

Programname:	Suremosch
Computer:	Amiga 500,1000,2000 mit Kickstart 1.2
Sprache:	Amiga-Basic

```

1 NU0 ' *****
2 oU  ' * Suremosch V1.2 *
3 Pq  ' * *
4 nV  ' * Copyright C 1988 by *
5 Rs  ' * Markt & Technik AG *
6 Rt  ' * Christian Rodemeyer *
7 2e  ' * Goldregenstraße 28 *
8 FQ  ' * 4370 Marl - Sinsen *
9 Vc  ' *****
10 xJ  DEFINT a-z
11 5J  DIM SHARED xy,True,False,jx,jy,Obj,Feldx,Feldy,B$,Vrz$,Fehl
      er,Tr$,Skor
12 a0  True=-1:False=0:Obj=17
13 iC  PicEl=2+Obj*INT((Obj+15)/16)*4
14 Lk  jx=2:jy=1:Feldx=17:Feldy=12
15 yd  MaxObj=10:Skor=29
16 ma  PfadE$="Nachlader/Editor.dat":Le$="Leere Ebene! "
17 pf  Wpfad$="Wedongs/":Vrz$=Wpfad$+"Wedong-Namen"
18 yE  Spname$="Gastspieler":Tr$=CHR$(10)
19 Et  DIM SHARED Spieler(PicEl,3,2),Objekt(PicEl,MaxObj),Feld(Fel
      dx,Feldy),Status(9)
20 su  DIM SHARED Fpos(1),r(1),Hpos(1),Npos(1),Apos(1)
21 sD  DIM SHARED EdFeld(Feldx,Feldy),Buffer(3000),Editfiles$(39)
22 Ot  DIM WahlEd(39),Solved(39),Wedongs$(39),Ebenen$(39),Edbuffer
      (17682)
23 UU  DEF FNx(x)=x*Obj+jx
24 lP  DEF Fny(y)=y*Obj+jy
25 Ho  DEF FN Pc(x,y,x1,y1,x2,y2)=x>x1 AND x<x2 AND y>y1 AND y<
      y2
26 lJ  DEF FN Syntax(File$)=INSTR(File$,".")=0 AND INSTR(File$,"/"/
      )=0
27 gU  GOSUB SoundInit:GOSUB PicLoad:GOSUB FileInit:GOSUB BlinkIni
      t
28 aE  ON ERROR GOTO IO.Error
29 C5  Hauptmenue:
30 s32  COLOR 1,0:CLS:GOSUB Titel
31 7z  WHILE MOUSE(0)<>0:WEND
32 vn0  Tloop:
33 D12  IF MOUSE(0)<>0 THEN
34 xq4  g=(MOUSE(2)+4)/8
35 99  IF (g MOD 3)=0 THEN g=g\3 ELSE g=0
36 C4  WHILE MOUSE(0)<>0:WEND
37 Cv2  ELSE
38 ue4  g=ASC(INKEY$+CHR$(0))-128
39 ZS2  END IF

```

40 YJ	IF g>0 THEN ON g GOTO SetSpieler,Erschaffe,Spielstart,Ed itor,Rank,CD,WB
41 uLO	GOTO Tloop
42 aZ	WB:SYSTEM
43 3h	Rank:
44 R62	IF Spname\$="Gastspieler" THEN BEEP:GOTO Tloop
45 pM	OPEN "Spieler/"+Spname\$ FOR INPUT AS 1
46 uJ	COLOR 8,0:CLS
47 Km	B\$=INPUT\$(299,1):Aktpfad\$=LEFT\$(B\$,INSTR(B\$,Tr\$)-1)
48 5V	IF Aktpfad\$<>" " THEN
49 4n4	CALL Zentri(2,"Der Spieler"):COLOR 12:CALL Zentri(4,Spn ame\$)
50 64	COLOR 8:CALL Zentri(6,"befindet sich zur Zeit im Wedong ")
51 qq	n=1
52 Lu	WHILE INSTR(n,Aktpfad\$,"/")<>0:n=INSTR(n,Aktpfad\$,"/")+1:WEND
53 Gd	Wedong\$=MID\$(B\$,n,INSTR(n,Aktpfad\$,".Wedong\$")-n)
54 iM	COLOR 12:CALL Zentri(8,Wedong\$)
55 xy	COLOR 8:CALL Zentri(11,"Er hat dort bereits"+STR\$(ASC(M ID\$(B\$,259,1)+CHR\$(0))+" Ebenen gelöst.")
56 vv	GOSUB Wm
57 Y92	ELSEIF LOF(1)<300 THEN
58 Cj4	CALL Zentri(10,"Absoluter Neuling!")
59 yV	GOSUB Wm
60 un2	END IF
61 TP	COLOR 9:CLS
62 5z	IF LOF(1)>299 THEN
63 kt4	CALL Zentri(2,"Gelöste Wedongs des Spielers")
64 By	COLOR 8:CALL Zentri(4,Spname\$)
65 YD	n=-1:COLOR 12
66 VV	WHILE NOT EOF(1)
67 Zd6	n=n+1
68 rw	INPUT #1,g\$
69 cQ	IF (n MOD 2)=0 THEN x=2 ELSE x=19
70 8U	LOCATE 7+n\2,x:PRINT g\$
71 Ps	IF ((n+1) MOD 40)=0 THEN
72 nY8	GOSUB Wm:COLOR 12
73 B1	LINE (0,55)-(318,217),0,BF
74 KX	n=-1
75 926	END IF
76 FT4	WEND
77 GG	GOSUB Wm
78 C52	END IF
79 5p	CLOSE 1
80 rJ0	GOTO Hauptmenue
81 NA	Wm:CALL Message("Taste,Mausklick oder Feuer!"):CALL Reactio n:RETURN
82 WE	SetSpieler:
83 6w2	CALL Message("Welcher Spieler soll geladen werden?")
84 fn	g\$="":CALL Inline(2,25,g\$,36)
85 72	IF g\$<>" " AND FN Syntax(g\$) AND UCASE\$(g\$)<>"GASTSPIE LER" THEN
86 mg4	CALL Checkfile("Spieler/"+g\$)
87 r6	IF Fehler=53 THEN BEEP:CALL Message("Diesen Spieler gib t es nicht!"):CALL Reaction ELSE Spname\$=g\$
88 MF2	END IF
89 cN0	GOSUB Titel:GOTO Tloop
90 Qr	Erschaffe:
91 5s2	CALL Message("Wie soll der Spieler heißen?")
92 nv	g\$="":CALL Inline(2,25,g\$,36)
93 j3	IF g\$<>" " AND UCASE\$(g\$)<>"GASTSPIELER" AND FN Syntax (g\$) THEN
94 A94	Spname\$=g\$
95 ua	OPEN "Spieler/"+Spname\$ FOR OUTPUT AS 1
96 yr	PRINT #1,Tr\$;STRING\$(298,CHR\$(0));:CLOSE 1
97 VO2	END IF
98 lW0	GOSUB Titel:GOTO Tloop
99 91	CD:
100 fW2	COLOR 15,0:CLS
101 Cp	CALL Zentri(4,"Gebe den Pfad zum Verzeichnis ein!")
102 Kd	COLOR 12:LOCATE 6,1:WIDTH 39:LINE INPUT:g\$:WIDTH 80
103 CR	CALL Checkfile(g\$+"Wedong-Namen")
104 c6	IF Fehler=0 THEN
105 gG4	Wpfad\$=g\$+"/":Vrz\$=Wpfad\$+"Wedong-Namen"
106 zu	GOSUB GetWedongs
107 FY2	END IF
108 eR	IF Fehler<>0 THEN
109 TU4	BEEP:COLOR 15
110 jL	PRINT Tr\$Tr\$ "Das ist kein Wedong-Verzeichniss!"
111 ne	CALL Reaction
112 kd2	END IF
113 Oq0	GOTO Hauptmenue

Listing 2. »Suremosch«. Das Hauptprogramm des Spiels


```

114 v5 Alles:
115 rY2 COLOR 15,7:CLS:RESTORE KeineWedongsBotschaft
116 ZP CALL Zentri(4,"Du bist ein echter Suremoscher!")
117 s2 COLOR 14
118 ef FOR n=0 TO 5
119 qZ4 READ g$
120 6V CALL Zentri(8+n*2,g$)
121 Yt2 NEXT n
122 yp CALL Reaction
123 Lx0 RETURN
124 NH KeineWedongsBotschaft:
125 602 DATA "Du hast alle Wedongs in diesem","Verzeichniss gelöst, echt Spitze!"
126 19 DATA "Nachdem du so viele Ebenen gelöst hast,","bist du nun sicherlich in der Lage"
127 Tf DATA "selbst gute Wedongs zu entwerfen.", "Los, gehe in den Editor und fang an."
128 jI0 Spielstart:
129 nn2 IF Spname$="Gastspieler" THEN IF Wanz>=0 THEN GOSUB Wochoice:GOTO GastIn ELSE BEEP:CALL Message("Hier gibts keine Wedongs!") :CALL Reaction:GOTO Abort
130 Go OPEN "Spieler/"+Spname$ FOR INPUT AS 2
131 Gx B$=INPUT$(LOF(2),2):CLOSE 2
132 9I Aktpfad$=LEFT$(B$,INSTR(B$,Tr$)-1)
133 Pq0 NeuW:
134 RN2 Flag=False
135 iX IF Aktpfad$="" THEN
136 mq4 FOR n=0 TO Wanz
137 Zk6 IF INSTR(300,B$,Wedongs$(n)+Tr$)<>0 THEN Solved(n)=1 ELSE Solved(n)=0:Flag=True
138 pA4 NEXT n
139 CC IF Flag THEN GOSUB Wochoice:Flag=False ELSE Flag=True
140 9A IF Wanz<0 THEN Flag=True
141 sb2 ELSE
142 Yw4 FOR n=0 TO 39
143 DQ6 Solved(n)=ASC(MID$(B$,260+n,1)+CHR$(0))
144 vG4 NEXT n
145 Ok n=1:Sanz=ASC(MID$(B$,259,1)+CHR$(0))
146 rQ WHILE INSTR(n,Aktpfad$,"/")<>0:n=INSTR(n,Aktpfad$,"/")+1:WEND
147 m9 Wedong$=MID$(B$,n,INSTR(n,Aktpfad$,".Wedong")-n)
148 KD2 END IF
149 rJ IF Flag THEN GOSUB Alles:GOTO Abort
150 h60 GastIn:
151 652 OPEN Aktpfad$ AS 1 LEN=272
152 Q2 FIELD #1,200 AS dummy$
153 M3 GET #1,1:mes$=dummy$
154 5P FIELD #1,18 AS Eb$,254 AS dummy$
155 ng Ebanz=0
156 xx WHILE NOT EOF(1)
157 mv4 GET #1
158 ow Ebenen$(Ebanz)=Eb$
159 Sf Ebanz=Ebanz+1
160 ip2 WEND
161 eu Ebanz=Ebanz-2
162 nF0 EbenenWahl:
163 YL2 CALL Verzeichnis(Ebenen$(),Solved(),Ebanz)
164 it COLOR 5,0:CALL Zentri(2,Wedong$)
165 gZ COLOR 8:LOCATE 4,1:h=1
166 ow FOR n=1 TO 3:h2=INSTR(h,mes$,Tr$):PRINT MID$(mes$,h,h2-h):h=h2+1:NEXT
167 aK CALL Message("Wähle eine Ebene!")
168 S5 Ebene=40
169 vx WHILE Ebene>Ebanz
170 Bx4 CALL GetKlick(Ebenen$(),Ebanz,Ebene,Solved())
171 zr IF Ebene<=Ebanz THEN IF Solved(Ebene)=1 THEN Ebene=40
172 D12 WEND
173 qu0 JumpEbene:
174 Zj2 COLOR 9,0:CLS:CALL Zentri(13,"Achtung, Gehirn aktivieren!")
175 X6 CALL ReadEbene(1,Ebene+1,Feld(),Status())
176 jc MENU 1,0,1,"Optionen":MENU 1,1,1,"Aufgeben"
177 x1 ON MENU GOSUB Optionen:MENU ON
178 EZ GOSUB Game
179 fN MENU OFF:MENU 1,0,0,""
180 41 CALL Rahmen(64,75,189,58):COLOR 1,0
181 nX IF Status(3)<>0 THEN
182 QW4 LOCATE 12,11:PRINT "Das war wohl nichts!"
183 Uy LOCATE 15,12:PRINT "Versuchs nochmal!"
184 ZI2 ELSE
185 lm4 LOCATE 12,17:PRINT "Hurra,"
186 8c LOCATE 13,10:PRINT "du hast es geschafft!"
187 qx IF Spname$<>"Gastspieler" THEN
188 Pz6 Solved(Ebene)=1

```

```

189 r6 Sanz=Sanz+1:MID$(B$,259)=CHR$(Sanz)
190 iy MID$(B$,260+Ebene)=CHR$(1)
191 Vs IF Sanz=Ebanz+1 THEN
192 xB8 B$=B$+Wedong$+Tr$
193 h9 Aktpfad$=""
194 W4 MID$(B$,1)=Tr$
195 JC LOCATE 15,10:PRINT "Der Wedong ist gelöst."
196 yi CLOSE 1
197 mV6 ELSE
198 4Y8 LOCATE 15,10:PRINT "Die Ebene ist gelöst."
199 926 END IF
200 pY4 ELSE
201 7b6 LOCATE 15,10:PRINT "Die Ebene ist gelöst."
202 C54 END IF
203 D62 END IF
204 I9 CALL Reaction.
205 g80 Spielmenue:
206 Ig2 COLOR 8:CLS
207 gR LOCATE 5,11:PRINT "F1 => Hauptmenue"
208 iz LOCATE 8,11:PRINT "F2 => Neue Ebene"
209 Rk LOCATE 11,11:PRINT "F3 => Gleiche Ebene"
210 IF LOCATE 14,11:PRINT "F4 => Notausgang"
211 6P CALL Message("Treffe deine Wahl")
212 Os0 Uwait:
213 7c2 IF MOUSE(0)<>0 THEN
214 YK4 g=MOUSE(2)\8-1
215 33 IF (g MOD 3)=0 THEN g=g\3 ELSE g=0
216 5o2 ELSE
217 nX4 g=ASC(INKEY$+CHR$(0))-128
218 SL2 END IF
219 LP IF STRIG(2)<>0 THEN Gleiche
220 Bq IF g>0 THEN ON g GOTO Spielende,Neue,Gleiche,Abort
221 NQ0 GOTO Uwait
222 BX Neue:IF Aktpfad$="" THEN NeuW ELSE EbenenWahl
223 og Gleiche:IF Status(3)=0 THEN Neue ELSE GOTO JumpEbene
224 Ed Spielende:
225 F42 CLOSE 1:IF Spname$="Gastspieler" THEN GOTO Hauptmenue
226 1h OPEN "Spieler/"+Spname$ FOR OUTPUT AS 1
227 jP PRINT #1,B$;:CLOSE 1
228 Fn0 GOTO Hauptmenue
229 d4 Abort:CLOSE 1:GOTO Hauptmenue
230 zI Wochoice:
231 aY2 CALL Verzeichnis(Wedongs$(),Solved(),Wanz)
232 YN COLOR 5,0:CALL Zentri(2,"Wähle einen Wedong!")
233 4H h=40
234 Hq WHILE h>Wanz
235 n14 CALL GetKlick(Wedongs$(),Wanz,h,Solved())
236 SU IF h<=Wanz THEN IF Solved(h)=1 THEN h=40
237 G42 WEND
238 Qn Wedong$=Wedongs$(h)
239 Cr Aktpfad$=Wpfad$+Wedong$+"Wedong"
240 OH MID$(B$,1)=Aktpfad$+Tr$
241 pq FOR n=0 TO 39:Solved(n)=0:MID$(B$,260+n)=CHR$(0):NEXT
242 XI Sanz=0:MID$(B$,259)=CHR$(0)
243 Ht0 RETURN
244 DV IO.Error:
245 Hu2 Fehler=ERR
246 8r IF ERR=53 THEN RESUME NEXT
247 360 ON ERROR GOTO 0
248 ej ' Hier beginnen die Routinen, die nur vom Editor gebraucht werden
249 9N ' Wer glaubt, den Editor nicht zu brauchen, braucht auch diesen Teil nicht
250 1u ' abzutippen. Statt dessen muß folgende Zeile eingefügt werden:
251 Fn 'Editor:BEEP:CALL Message("Der Editor ist nicht vorhanden") :CALL Reaction:GOTO Hauptmenue
252 Zj ' ***** Anfang des Editorteils *****
253 cJ EdmenuDat:
254 oB2 DATA Spielfeld,ObjektWahl,Löschen,LeerFeld,Startposition,Testen,+
255 fP DATA Parameter,Ansehen,Ändern,+
256 Do DATA Projekt,Laden,Speichern,Speichern als,Löschen,Umbenennen,Neuer Wedong,Lösche Wedong,"Hauptmenü",+
257 xx0 DATA @
258 RQ Edmenu:ON MENU(0) GOTO Editieren,Parameter,Projekt
259 3H Editieren:ON MENU(1) GOSUB ObjektWahl,Loeschen,LeerFeld,Startposition,Testen
260 A02 GOTO EndMenu
261 rL0 Parameter:ON MENU(1) GOSUB Ansehen,Aendern
262 C22 GOTO EndMenu
263 Uo0 Projekt:ON MENU(1) GOSUB Laden,Speichern,Speichern.als,Fdel,Rename,Wedong,KiWe,Back

```

```

264 E42 GOTO EndMenu
265 020 BEEP:STOP
266 GX EndMenu:
267 or2 CALL Message("Entwerfe dein eigenes Spielfeld!")
268 RY dummy=MOUSE(0):dummy=0
269 vg MENU 1,0,1:MENU 2,0,1:MENU 3,0,1:MENU ON
270 iK0 RETURN
271 XK Rename:
272 w02 GOSUB Minaktiv:GET (jx,jy)-STEP(305,220),Edbuffer
273 nC CALL Verzeichnis(Editfiles$(),WahlEd(),39)
274 dm COLOR 9,0:CALL Zentri(2,"Klicke die umzubennenden Wedongs
an!")
275 o7 CALL Klion("Abbruch")
276 3o FIELD #1,18 AS io1$
277 Yy CALL GetKlick(Editfiles$(),39,h,WahlEd())
278 F1 WHILE h<=39
279 x24 IF (h MOD 2)=0 THEN x=2 ELSE x=21
280 e9 CALL Message("Gebe den Namen der Ebene ein!")
281 Ff COLOR 12:y=h/2+8:Nam$="":CALL Inline(x,y,Nam$,18)
282 WK IF Nam$<>" " THEN
283 od6 GET #1,h+1
284 NM LSET io1$=Nam$
285 fA CALL Message(" "):Editfiles$(h)=Nam$
286 v9 PUT #1,h+1
287 ZS4 END IF
288 j9 CALL GetKlick(Editfiles$(),39,h,WahlEd())
289 6u2 WEND
290 Dg PUT (jx,jy),Edbuffer,PSET
291 jf0 RETURN
292 GS KiWe:
293 Hj2 GOSUB Minaktiv:GET (jx,jy)-STEP(305,220),Edbuffer
294 t5 CALL Verzeichnis(Wedongs$(),WahlEd(),Wanz)
295 jU COLOR 9,0:CALL Zentri(2,"Klicke die zu löschenden Wedongs
an!")
296 9S CALL Klion("Abbruch")
297 fJ CALL GetKlick(Wedongs$(),Wanz,h,WahlEd())
298 vJ WHILE h<=Wanz
299 zN4 IF WahlEd(h)=0 THEN
300 6b6 KILL Wpfd$+Wedongs$(h)+".Wedong"
301 w7 Wedongs$(h)=" Gelöscht! ":WahlEd(h)=1
302 oh4 END IF
303 lP CALL GetKlick(Wedongs$(),Wanz,h,WahlEd())
304 L92 WEND
305 UJ B$="":Flag=False
306 Wa FOR n=0 TO Wanz
307 XT4 IF WahlEd(n)=0 THEN B$=B$+Wedongs$(n)+CHR$(10) ELSE Fla
g=True
308 Zu2 NEXT n
309 qb CLOSE 2
310 2P CALL Message("Reinitialisierung des Verzeichnisses!")
311 Qy IF Flag THEN OPEN Vrz$ FOR OUTPUT AS 2:PRINT #2,B$;:CLOSE
2
312 JE GOSUB GetWedongs
313 a3 PUT (jx,jy),Edbuffer,PSET
314 IT FOR n=0 TO 39:WahlEd(n)=0:NEXT
315 R30 RETURN
316 4y Wedong:
317 9z2 IF Wanz>=40 THEN BEEP:RETURN
318 g8 GOSUB Minaktiv:GET (jx,jy)-STEP(305,220),Edbuffer
319 Xw CALL Verzeichnis(Editfiles$(),WahlEd(),39)
320 sm COLOR 9,0:CALL Zentri(2,"Wähle die Ebenen für den Wedong!
")
321 tR CALL Klion("Weiter")
322 Hh CALL GetKlick(Editfiles$(),39,h,WahlEd())
323 Zk h2=0
324 z1 WHILE h<=39
325 8c4 IF WahlEd(h)=0 AND Editfiles$(h)<>Le$ THEN
326 606 h2=h2+1:WahlEd(h)=1
327 sb4 ELSE
328 Ms6 IF WahlEd(h)=1 THEN h2=h2-1:WahlEd(h)=0
329 F84 END IF
330 Pp CALL GetKlick(Editfiles$(),39,h,WahlEd())
331 ma2 WEND
332 Ly IF h2>1 THEN
333 Ko4 COLOR 15,0:CLS:Wedong$="":mes$=""
334 mM CALL Zentri(2,"Gebe den Namen des Wedongs ein!")
335 QN COLOR 12:CALL Inline(12,4,Wedong$,18)
336 aH IF FN Syntax(Wedong$) THEN CALL Checkfile(Wpfd$+Wedong
$) ELSE Fehler=99
337 Iv IF Fehler=53 THEN
338 Ke6 OPEN Vrz$ FOR APPEND AS 3
339 Tm COLOR 15:CALL Zentri(6,"Gebe eine Botschaft ein!"):LO
CATE 8,1
340 Zz PRINT #3,Wedong$:CLOSE 3:COLOR 12

```

```

341 CC FOR n=1 TO 3
342 jw8 g$="":CALL Inline(1,8+n,g$,38)
343 rn mes$=mes$+g$+Tr$
344 9U6 NEXT n
345 26 CALL Message("Der Wedong wird erzeugt!")
346 3J OPEN Wpfd$+Wedong$+".Wedong" AS #2 LEN=272
347 Qh FIELD #2,200 AS Botschaft$
348 wR LSET Botschaft$=mes$
349 W5 PUT #2,1:h=2
350 0a FIELD #1,272 AS io1$:FIELD #2,272 AS io2$
351 vJ FOR n=0 TO 39
352 tG8 IF WahlEd(n)=1 THEN GET #1,n+1:LSET io2$=io1$:PUT #
2,h
353 bT h=h+1
354 Je6 NEXT n
355 aL CLOSE 2
356 FV Wanz=Wanz+1:Wedongs$(Wanz)=Wedong$
357 M54 ELSE
358 cA6 CLS:BEEP:COLOR 15,0
359 x1 IF Fehler=0 THEN CALL Zentri(10,"Dieser Name existier
t bereits") ELSE CALL Zentri(10,"Illegaler Filename!"
)
360 of CALL Reaction
361 le4 END IF
362 mf2 END IF
363 EN COLOR 0,0:CLS:PUT (jx,jy),Edbuffer,PSET
364 6H FOR n=0 TO 39:WahlEd(n)=0:NEXT
365 Fr0 RETURN
366 tX Back:EndFlag=True:RETURN
367 Xu Fdel:
368 Uw2 GOSUB Minaktiv:GET (jx,jy)-STEP(305,220),Edbuffer
369 lK CALL Verzeichnis(Editfiles$(),WahlEd(),39)
370 2R COLOR 9,0:CALL Zentri(2,"Klicke die zu löschende Ebene an
!")
371 Mf CALL Klion("Abbruch")
372 5V CALL GetKlick(Editfiles$(),39,h,WahlEd())
373 bQ FIELD #1,18 AS n$,254 AS dummy$
374 nZ WHILE h<=39
375 3d4 GET #1,h+1
376 J8 LSET n$=Le$:Editfiles$(h)=n$
377 98 PUT #1,h+1
378 Bb CALL GetKlick(Editfiles$(),39,h,WahlEd())
379 YM2 WEND
380 f8 PUT (jx,jy),Edbuffer,PSET
381 V70 RETURN
382 ZH Minaktiv:
383 TN2 MENU OFF
384 Q4 MENU 1,0,0:MENU 2,0,0:MENU 3,0,0
385 ZB0 RETURN
386 d1 Ansehen:
387 EC2 GOSUB Minaktiv
388 Q1 GOSUB Zeigen:CALL Reaction
389 hd PUT (88,80),Buffer,PSET
390 eG0 RETURN
391 P0 Zeigen:
392 4I2 GET (88,80)-STEP(150,70),Buffer
393 nv CALL Rahmen(90,82,146,66)
394 NG x=13:TIMER ON
395 oh LOCATE 12,x:PRINT "Schritte :";USING"###";
Edstatus(2)
396 lM LOCATE 13,x:PRINT "Klockis :";USING"###";Edstatus(3)
397 pd LOCATE 14,x:PRINT "Klockis1 :";USING"###";Edstatus(5)
398 4q LOCATE 15,x:PRINT "Klockis2 :";USING"###";Edstatus(6)
399 SN LOCATE 16,x:PRINT "Telops :";USING"###";Edstatus(8)
400 Nv LOCATE 17,x:PRINT "BonusSchritte:";USING"###";Edstatus(4)
401 XT LOCATE 18,x:PRINT "BonusTelops :";USING"###";Edstatus(9)
402 qS0 RETURN
403 Ck Aendern:
404 Qb2 GOSUB Minaktiv:GOSUB Zeigen
405 SL A1:CALL Para(26,12,Edstatus(2),3):IF Edstatus(2)=0 THEN B
EEP:GOTO A1
406 p1 A2:CALL Para(27,13,Edstatus(3),2):IF Edstatus(3)=0 THEN B
EEP:GOTO A2
407 Az5 CALL Para(27,14,Edstatus(5),2)
408 J5 CALL Para(27,15,Edstatus(6),2)
409 UE CALL Para(27,16,Edstatus(8),2)
410 T5 CALL Para(27,17,Edstatus(4),2)
411 kN CALL Para(27,18,Edstatus(9),2)
412 ZX2 PUT(88,80),Buffer,PSET
413 Vb Saved=False
414 2e0 RETURN

```

Listing 2. (Fortsetzung)


```

415 IA Testen:
416 lv2 GET (jx,jy)-STEP(305,220),Edbuffer
417 GH FOR x=0 TO Feldx
418 PN4 FOR y=0 TO Feldy
419 Am6 Feld(x,y)=EdFeld(x,y)
420 z72 NEXT y,x
421 Vy FOR n=0 TO 9:Status(n)=Edstatus(n):NEXT
422 n1 GOSUB Minaktiv
423 RQ MENU 1,0,1:FOR n=1 TO 4:MENUE 1,n,0:NEXT
424 yQ MENU 1,5,1,"Abbrechen":ON MENU GOSUB Aufgeben:MENUE ON
425 uX Fehler=0:GOSUB Game
426 rB CALL Rahmen(59,95,200,40)
427 CD LOCATE 15,12:COLOR 6
428 ih IF Fehler=255 THEN
429 4a4 BEEP
430 cx CALL Zentri(14,"Das Spielfeld enthält")
431 Le CALL Zentri(16,"einen schweren Fehler!")
432 aj2 ELSEIF Status(3)=0 THEN
433 Lq4 CALL Zentri(14,"Die Ebene wurde")
434 l1 CALL Zentri(16,"erfolgreich gelöst!")
435 cL2 ELSE
436 MU4 CALL Zentri(14,"Die Ebene wurde nicht")
437 8J CALL Zentri(16,"gelöst!")
438 Ot2 END IF
439 5w CALL Reaction
440 M1 MENU 1,5,1,"Testen"
441 gm ON MENU GOSUB Edmenu
442 f8 PUT (jx,jy),Edbuffer,PSET
443 Ud ObNr=1:FOR n=1 TO 4:MENUE 1,n,1:NEXT
444 W80 RETURN
445 5R Laden:
446 B92 GOSUB Minaktiv
447 A7 IF NOT Saved THEN
448 5t4 COLOR 12,0:CLS
449 X9 CALL Zentri(8,"Die aktuelle Ebene ist noch nicht")
450 bQ CALL Zentri(10,"gespeichert! Soll trotzdem eine neue")
451 LV CALL Zentri(12,"Ebene geladen werden?")
452 NB CALL JaNein(15,16,Saved)
453 F82 END IF
454 Tb IF Saved THEN
455 j84 CALL Verzeichnis(Editfiles$()),WahlEd(),39)
456 gI COLOR 9,0:CALL Zentri(2,"Wähle die zu ladende Ebene!")
457 k3 CALL Klion("Abbruch")
458 vt CALL GetKlick(Editfiles$(),39,File,WahlEd())
459 vN IF File<=39 THEN
460 HQ6 CALL Message("Die Ebene wird geladen.")
461 oc CALL ReadEbene(1,File,EdFeld(),Edstatus())
462 D3 LedFlag=True
463 PI4 END IF
464 QJ2 END IF
465 aF CALL Spielfeld(EdFeld(),Edstatus(0),Edstatus(1))
466 sU0 RETURN
467 hl Speichern.als:
468 6X2 GOSUB Minaktiv:GET (jx,jy)-STEP(305,220),Edbuffer
469 xM CALL Verzeichnis(Editfiles$()),WahlEd(),39)
470 j9 COLOR 9,0
471 mv CALL Zentri(2,"Wo soll die Ebene gespeichert werden?")
472 zI CALL Klion("Abbruch")
473 AB CALL GetKlick(Editfiles$(),39,File,WahlEd())
474 Ac IF File<=39 THEN
475 tY4 IF Editfiles$(File)<>Le$ THEN
476 dz6 COLOR 5,0
477 N3 CALL Zentri(2,"Soll die Ebene überschrieben werden? ")
478 d1 CALL JaNein(15,5,Flag)
479 K34 ELSE
480 4s6 Flag=True
481 ha4 END IF
482 N9 IF Flag THEN
483 k86 IF (File MOD 2)=0 THEN x=2 ELSE x=21
484 wR CALL Message("Gebe den Namen der Ebene ein!")
485 7h y=File\2+8:COLOR 12
486 fN Nam$="":CALL Inline(x,y,Nam$,18)
487 pd IF Nam$<>" " THEN
488 Qg8 CALL Message("Die Ebene wird gespeichert.")
489 DI CALL WriteEbene(1,File,Nam$,EdFeld(),Edstatus())
490 Vb Editfiles$(File)=Nam$+SPACE$(18-LEN(Nam$)):Saved=Tr
ue
491 rk6 END IF
492 s14 END IF
493 tm2 END IF
494 Vy PUT (jx,jy),Edbuffer,PSET
495 Lx0 RETURN

```

```

496 6M Speichern:
497 E22 IF File>39 THEN Speichern.als
498 vr IF Editfiles$(File)=Le$ THEN Speichern.als
499 20 GOSUB Minaktiv
500 CF CALL Message("Die Ebene wird gespeichert!")
501 pz CALL WriteEbene(1,File,Editfiles$(File),EdFeld(),Edstatus
())
502 r0 Saved=True
503 T50 RETURN
504 ut LeerFeld:ObNr=0:RETURN
505 fX Editor:
506 vd2 RESTORE EdmenuDat
507 b5 Saved=True:GOSUB Loeschen
508 Kg READ m$:x=1:y=0
509 aV WHILE m$<>"@"
510 ch4 IF m$="+" THEN x=x+1:y=0 ELSE MENU x,y,1,m$:y=y+1
511 MB READ m$
512 hv2 WEND
513 ip ObNr=1:EndFlag=False
514 4V Apos(0)=0:Apos(1)=0
515 NM OPEN PfadE$ AS 1 LEN=272
516 ps CALL Message("Entwerfe dein eigenes Spielfeld!")
517 O3 ON MENU GOSUB Edmenu:MENUE ON
518 bJ WHILE NOT EndFlag
519 A14 dummy=MOUSE(0)
520 u7 IF dummy<>0 THEN EdFeld(Apos(0),Apos(1))=ObNr:Saved=F
alse
521 M9 Npos(0)=(MOUSE(1)-Obj/2-2)/Obj:Npos(1)=(MOUSE(2)-Obj/2-
2)/Obj
522 GZ IF (Npos(0)<>Apos(0) OR Npos(1)<>Apos(1)) AND Npos(
0)<=Feldx AND Npos(0)>=0 AND Npos(1)<=Feldy AND Npos
(1)>=0 THEN
523 k36 IF Apos(0)=Edstatus(0) AND Apos(1)=Edstatus(1) THEN
524 3X8 EdFeld(Apos(0),Apos(1))=0
525 IO PUT (FNx(Apos(0)),FNy(Apos(1))),Spieler(0,0,0),PSET
526 5o6 ELSE
527 on8 PUT (FNx(Apos(0)),FNy(Apos(1))),Objekt(0,EdFeld(Apo
s(0),Apos(1))),PSET
528 SL6 END IF
529 LQ Apos(0)=Npos(0):Apos(1)=Npos(1)
530 pf PUT (FNx(Npos(0)),FNy(Npos(1))),Objekt(0,ObNr),PSET
531 V04 END IF
532 1p2 WEND
533 YV IF NOT Saved THEN
534 rU4 MENU OFF:COLOR 8,0:CLS
535 UQ CALL Zentri(4,"Die Ebene ist noch nicht gespeichert!")
536 tI CALL Zentri(7,"Soll sie gespeichert werden?")
537 YO CALL JaNein(15,14,Flag)
538 XY IF Flag THEN GOSUB Speichern
539 dW2 END IF
540 WG CLOSE 1
541 Ik0 GOTO Hauptmenue
542 TO Startposition:
543 ki2 GOSUB Minaktiv
544 Zm PUT (FNx(Edstatus(0)),FNy(Edstatus(1))),Objekt(0,0),PSET
545 MH CALL Message("Gebe die Startposition ein!")
546 bS dummy=MOUSE(0)
547 OT WHILE MOUSE(0)=0
548 Qp4 Npos(0)=(MOUSE(1)-Obj/2)/Obj:Npos(1)=(MOUSE(2)-Obj/2)/O
bj
549 WL IF (Npos(0)<>Apos(0) OR Npos(1)<>Apos(1)) AND Npos(
0)<=Feldx AND Npos(1)<=Feldy THEN
550 BA6 PUT (FNx(Apos(0)),FNy(Apos(1))),Objekt(0,EdFeld(Apos(
0),Apos(1))),PSET
551 hm Apos(0)=Npos(0):Apos(1)=Npos(1)
552 pu PUT (FNx(Npos(0)),FNy(Npos(1))),Spieler(0,0,0),PSET
553 rk4 END IF
554 NB2 WEND
555 LA Edstatus(0)=Npos(0):Edstatus(1)=Npos(1)
556 Kw0 RETURN
557 ei Loeschen:
558 xu2 IF NOT Saved THEN
559 Oy4 GOSUB Minaktiv
560 l5 CALL Message("Willst du die Ebene wirklich löschen?")
561 Yr CALL JaNein(4,3,Saved)
562 Ot2 END IF
563 EM IF Saved THEN
564 Os4 FOR x=0 TO Feldx:FOR y=0 TO Feldy:EdFeld(x,y)=0:NEXT y,
x
565 zc COLOR 1,0:CLS:File=40
566 2y Edstatus(0)=0:Edstatus(1)=0
567 m7 PUT (FNx(0),FNy(0)),Spieler(0,0,0),PSET

```

```

568 6z2 END IF
569 X90 RETURN
570 4K ObjektWahl:
571 CA2 GOSUB Minaktiv
572 N1 GET (7,7)-STEP(120,54),Buffer
573 pa CALL Rahmen(9,9,116,50)
574 xx FOR n=0 TO 4
575 934 PUT (15+n*(Obj+5),15),Objekt(0,n+1),PSET
576 Tw PUT (15+n*(Obj+5),20+Obj),Objekt(0,n+6),PSET
577 uF2 NEXT n
578 20 dummy=MOUSE(0):CALL Message("Wähle ein Objekt!")
579 Ju WHILE MOUSE(0)=0:WEND
580 Oz x=MOUSE(1)-10:y=MOUSE(2)-10
581 qH IF x>0 AND x<117 AND y>0 AND y<49 THEN
582 YK4 ObNr=INT(x/(Obj+5))+1+5*INT(y/(Obj+5))
583 LE2 END IF
584 2u WHILE MOUSE(0)<>0:WEND
585 7f PUT (7,7),Buffer,PSET
586 oQ0 RETURN
587 C8 SUB Para(x,y,Zahl,Stellen) STATIC
588 QJ2 LINE (x*8-8,y*8-1)-STEP(Stellen*8,0),5
589 VH P:g$=INKEY$:IF g$="" THEN P
590 8M LINE (x*8-8,y*8-1)-STEP(Stellen*8,0),0
591 2Q IF g$<>CHR$(13) THEN
592 f14 CALL Inline(x,y,g$,Stellen)
593 Qn Zahl=VAL(g$)
594 WP2 END IF
595 dF0 END SUB
596 9d SUB JaNein(x,y,Flag) STATIC
597 LH2 BEEP:LOCATE y,x:x=x*8-16:y=y*8-16:COLOR 1,0
598 G4 GET (x-8,y-8)-STEP(107,38),Buffer
599 6j CALL Rahmen(x-7,y-7,105,36)
600 yk CALL Rahmen(x,y,44,22):CALL Rahmen(x+47,y,44,22)
601 O1 PRINT PTAB(x+12)"JA"PTAB(x+54)"NEIN"
602 nG Flag=MOUSE(0)
603 hI WHILE MOUSE(0)=0:WEND
604 5k Flag=FNPC(MOUSE(1),MOUSE(2),x+4,y+4,x+44,y+18)
605 Bp PUT (x-8,y-8),Buffer,PSET
606 OG WHILE MOUSE(0)<>0:WEND
607 pr0 END SUB
608 1J SUB Klion(Tx$) STATIC
609 UG2 CALL Rahmen(118,24,75,23)
610 Ch COLOR 8,0:LOCATE 5,17:PRINT Tx$
611 tv0 END SUB
612 LY ' ***** Ende des Editorteils *****
***
613 jS Playgame:
614 sd2 LOCATE Skor,11:PRINT USING"## #";Status(2);
615 tx LOCATE Skor,24:PRINT USING"## #";Status(3);
616 Sd LOCATE Skor,35:PRINT USING"## #";Status(8);
617 Sr IF Status(2)=0 OR Status(3)=0 OR Aufgabe THEN MENU OFF:RE
TURN
618 fR IF STRIG(2) AND Status(8)>0 THEN Tcheck
619 AQ r(1)=0
620 Gm r(0)=STICK(2):IF r(0)<>0 THEN MoveX
621 aO r(1)=STICK(3):IF r(1)<>0 THEN MoveY
622 Sk0 GOTO Playgame
623 Lq Tcheck:
624 vF2 r(1)=0:r(0)=STICK(2):IF r(0)<>0 THEN TcheckObj
625 rS r(1)=STICK(3):IF r(1)<>0 THEN TcheckObj
626 Wo0 GOTO Playgame
627 Aa TcheckObj:
628 UA2 Hpos(0)=Fpos(0)+r(0):Hpos(1)=Fpos(1)+r(1)
629 OF ObNr=Feld(Hpos(0),Hpos(1))
630 xn IF ObNr>2 AND ObNr<8 THEN Tausche
631 bt0 GOTO Playgame
632 1z Tausche:
633 J42 Status(8)=Status(8)-1
634 YV SWAP Feld(Fpos(0),Fpos(1)),Feld(Hpos(0),Hpos(1))
635 yr CALL Effekt(.7,1,Tel())
636 VQ PUT(Hpos(0)*Obj+jx,Hpos(1)*Obj+jy),Spieler(0,Richtung,0),
PSET
637 kc FOR n=0 TO 100:NEXT
638 Zn PUT(Fpos(0)*Obj+jx,Fpos(1)*Obj+jy),Objekt(0,ObNr),PSET
639 Sz Fpos(0)=Hpos(0):Fpos(1)=Hpos(1)
640 k20 GOTO Playgame
641 zI MoveX:
642 fF2 xy=0
643 AJ IF r(xy)=1 THEN Richtung=3 ELSE Richtung=2
644 Sw0 GOTO Move
645 90 MoveY:
646 oP2 xy=1
647 49 IF r(xy)=1 THEN Richtung=1 ELSE Richtung=0

```

```

648 W00 GOTO Move
649 NC Move:
650 TC2 x=Fpos(0)*Obj+jx:y=Fpos(1)*Obj+jy
651 m1 PUT(x,y),Spieler(0,Richtung,1),PSET
652 iJ GOSUB Check
653 jx PUT(x,y),Spieler(0,Richtung,0),PSET
654 1t FOR n=0 TO 100:NEXT
655 vB PUT(x,y),Spieler(0,Richtung,2),PSET
656 6z FOR n=0 TO 200:NEXT
657 GF IF Hflag THEN MoveEnd
658 a9 CALL Del(Fpos(0),Fpos(1))
659 51 Fpos(xy)=Fpos(xy)+r(xy)
660 yX Status(2)=Status(2)-1
661 7Z0 GOTO MoveEnd
662 81 MoveEnd:
663 YJ2 PUT(Fpos(0)*Obj+jx,Fpos(1)*Obj+jy),Spieler(0,Richtung,0),
PSET
664 Yz IF ObNr=0 THEN CALL Effekt(.5,0,Huepf())
665 9R0 GOTO Playgame
666 Jo Check:
667 oO2 Hflag=False
668 9T Hpos(0)=Fpos(0)+r(0):Hpos(1)=Fpos(1)+r(1):ObNr=Feld(Hpos(
0),Hpos(1))
669 41 ON ObNr GOTO Mauer,Mauer,Push,Push,Push,Flutsch,Push,BonS
chri,BonTel,Lfeld
670 S2 FOR w=0 TO 199:NEXT
671 Bn0 RETURN
672 NU BonTel:
673 WF2 Status(8)=Status(8)+Status(9)
674 bW IF Status(8)>99 THEN Status(8)=99
675 rN CALL Effekt(.7,1,Btel())
676 Y8 FOR w=0 TO 199:NEXT
677 Ht0 RETURN
678 kx Lfeld:
679 vJ2 CALL Effekt(.4,1,Hdick())
680 cC FOR w=0 TO 199:NEXT
681 Lx0 RETURN
682 WO Flutsch:
683 4L2 n=400
684 V1 WHILE Feld(Hpos(0)+r(0),Hpos(1)+r(1))=0
685 Qh4 CALL MoveObj(Hpos(),ObNr)
686 Du FOR w=0 TO 100:NEXT w
687 HT n=n+30
688 FV SOUND n,.5,250
689 YM2 WEND
690 CJ IF Feld(Hpos(0)+r(0),Hpos(1)+r(1))=2 THEN CALL ZielDel(Hp
os())
691 78 Hflag=True
692 W80 RETURN
693 pU BonSchri:
694 1D2 Status(2)=Status(2)+Status(4)
695 p2 IF Status(2)>1000 THEN Status(2)=1000
696 av CALL Effekt(.7,1,Bschri())
697 tT FOR w=0 TO 199:NEXT
698 cEO RETURN
699 sJ Mauer:FOR w=0 TO 300:NEXT:Hflag=True:RETURN
700 49 Shinguz:
701 T02 IF Feld(Hpos(0)+r(0),Hpos(1)+r(1))=2 THEN
702 2h4 Status(5)=Status(5)-1
703 zt CALL ZielDel(Hpos())
704 Pz Hflag=False
705 JC2 END IF
706 kM0 RETURN
707 nd Norm2:
708 md2 IF Feld(Hpos(0)+r(0),Hpos(1)+r(1))=2 AND Status(5)<=0 TH
EN
709 Hy4 Status(6)=Status(6)-1
710 60 CALL ZielDel(Hpos())
711 W6 Hflag=False
712 QJ2 END IF
713 rTO RETURN
714 Om Norm3:
715 yq2 IF Feld(Hpos(0)+r(0),Hpos(1)+r(1))=2 AND Status(6)<=0 TH
EN
716 WF4 Status(7)=Status(7)-1
717 D7 CALL ZielDel(Hpos())
718 dD Hflag=False
719 XQ2 END IF
720 ya0 RETURN
721 fT KeW1:RETURN
722 zx Push:

```

Listing 2. (Fortsetzung)


```

723 LG2 IF Feld(Hpos(0)+r(0),Hpos(1)+r(1))=0 THEN
724 3K4 CALL MoveObj(Hpos(),ObNr)
725 gs CALL Effekt(.4,0,Stos())
726 J22 ELSE
727 h14 Hflag=True
728 8m ON ObNr-2 GOSUB Shinguz, Norm2, Norm3, KeWi
729 ha2 END IF
730 8k0 RETURN
731 gp Optionen:ON MENU(1) GOTO Aufgeben
732 go Aufgeben:Aufgabe=True:RETURN
733 ba 2 ' Initialisierungsroutinen
734 y7 BlinkInit:
735 Q72 ON TIMER(.45) GOSUB Blink
736 Vb TIMER ON
737 Fr0 RETURN
738 mZ Blink:
739 ye2 Cycle=Cycle+1:IF Cycle>3 THEN Cycle=1
740 M1 ON Cycle GOTO Phase1,Phase2,Phase3
741 Mv0 Phase1:
742 U02 PALETTE 2,1,1,.2:PALETTE 3,.4,.6,.1:PALETTE 4,.4,.6,.1:RE
TURN
743 Q00 Phase2:
744 B32 PALETTE 2,.4,.6,.1:PALETTE 3,1,1,.2:PALETTE 4,.4,.6,.1:RE
TURN
745 U50 Phase3:
746 4m2 PALETTE 2,.4,.6,.1:PALETTE 3,.4,.6,.1:PALETTE 4,1,1,.2:RE
TURN
747 R00 Titel:
748 MG2 MENU OFF
749 sW FOR n=1 TO 4:MENU n,0,0,"":NEXT
750 h7 RESTORE TitelBild
751 Dc COLOR 8,0
752 vx FOR n=0 TO 6
753 s04 READ B$
754 OG LOCATE 3+n*3,8:PRINT B$
755 m72 NEXT n
756 UM COLOR 12:CALL Zentri(25,SPACE$(38)):CALL Zentri(25,Spname
$)
757 cb LINE (0,188)-STEP(309,16),9,B
758 9T CALL Message("Treffen deine Wahl!")
759 bD0 RETURN
760 LI TitelBild:
761 382 DATA "F1 => Spieler laden", "F2 => Spieler erschaffen"
762 rJ DATA "F3 => Ab in den Wedong", "F4 => Wedong-Editor"
763 as DATA "F5 => Status des Spielers", "F6 => Verzeichnis wec
hseln"
764 sh DATA "F7 => Suremosch verlassen"
765 6n0 FileInit:
766 RK2 OPEN "r", #1, PfadE$, 272
767 so FIELD #1, 18 AS EbeneEd$, 254 AS dummy$
768 e2 FOR n=0 TO 39
769 nT4 GET #1, n+1
770 IX Editfiles$(n)=EbeneEd$
771 2N2 NEXT n
772 G0 CLOSE 1
773 Fc0 GetWedongs:
774 v02 OPEN Vrz$ FOR INPUT AS 9
775 11 Wanz=-1
776 JD WHILE NOT EOF(9)
777 NM4 Wanz=Wanz+1
778 3d LINE INPUT #9, Wedongs$(Wanz)
779 Oo2 WEND
780 me CLOSE 9
781 xZ0 RETURN
782 HU PicLoad:
783 qJ2 OPEN "Nachlader/Spieler" FOR INPUT AS 1
784 IH FOR n=0 TO 3
785 fW4 FOR n2=0 TO 2
786 pp6 FOR n1=0 TO PicEl
787 Aa8 Spieler(n1,n2)=CVI(INPUT$(2,#1))
788 8k6 NEXT n1
789 Cp4 NEXT n2
790 Lg2 NEXT n
791 ZJ CLOSE 1
792 p2 OPEN "Nachlader/Objekte" FOR INPUT AS 1
793 kP FOR n1=0 TO MaxObj
794 OY4 FOR n=0 TO PicEl
795 UY6 Objekt(n,n1)=CVI(INPUT$(2,#1))
796 Rm4 NEXT n
797 Ht2 NEXT n1
798 gQ CLOSE 1
799 Fr0 RETURN
800 M1 Game:

```

```

801 Fy2 Aufgabe=False
802 Fd Fpos(0)=Status(0):Fpos(1)=Status(1)
803 KJ CALL Spielfeld(Feld(),Fpos(0),Fpos(1))
804 MN Richtung=0
805 xk LOCATE Skor,1:COLOR 15,0
806 tD PRINT " Schritte: Klockis: Telops:";
807 s0 COLOR 12
808 4T ON ERROR GOTO Spielfehler
809 UD GOSUB Playgame
810 Cq ON ERROR GOTO IO.Error
811 R30 RETURN
812 vZ Spielfehler:Aufgabe=True:Fehler=255:RESUME NEXT
813 29 SoundInit:
814 K12 DIM Welle(255)
815 kf Welle(0)=RND(1)
816 T4 FOR n=0 TO 255:Welle(n)=COS(n/50)*RND*127:NEXT n
817 KG WAVE 0,Welle:WAVE 1,SIN:ERASE Welle
818 Y4 RESTORE Sounddaten
819 FX READ anz:DIM SHARED Ding(anz,1):FOR n=0 TO anz:READ Ding(
n,0),Ding(n,1):NEXT
820 ef READ anz:DIM Huepf(anz,1):FOR n=0 TO anz:READ Huepf(n,0),
Huepf(n,1):NEXT
821 ou READ anz:DIM Hdick(anz,1):FOR n=0 TO anz:READ Hdick(n,0),
Hdick(n,1):NEXT
822 a4 READ anz:DIM Stos(anz,1):FOR n=0 TO anz:READ Stos(n,0),St
os(n,1):NEXT
823 1m READ anz:DIM Bschr(anz,1):FOR n=0 TO anz:READ Bschr(n,0
),Bschr(n,1):NEXT
824 4u READ anz:DIM Btel(anz,1):FOR n=0 TO anz:READ Btel(n,0),Bt
el(n,1):NEXT
825 yR READ anz:DIM Tel(anz,1):FOR n=0 TO anz:READ Tel(n,0),Tel(
n,1):NEXT
826 gI0 RETURN
827 sD Sounddaten:
828 Je2 DATA 6,300,125,700,255,500,200,400,220,350,170,300,130,25
0,100
829 OT DATA 3,250,90,280,160,260,255,230,100
830 2v DATA 3,320,90,340,160,340,255,310,100
831 ap DATA 6,250,80,280,160,260,240,250,90,280,160,260,255,230,
100
832 Us DATA 5,300,180,300,200,200,180,400,250,350,180,300,100
833 jh DATA 5,500,100,1050,200,500,250,750,200,650,100,900,50
834 s9 DATA 6,400,250,450,200,500,150,550,100,500,150,450,200,40
0,250
835 CU0 ' Modulare Unterprogramme
836 YD SUB Del(x,y) STATIC
837 RY2 Feld(x,y)=0
838 nT PUT (x*Obj+jx,y*Obj+jy),Objekt(0,0),PSET
839 Zb0 END SUB
840 Uw SUB MoveObj(P(),ObNr) STATIC
841 zm2 CALL Del(P(0),P(1))
842 JD P(xy)=P(xy)+r(xy)
843 jU PUT(P(0)*Obj+jx,P(1)*Obj+jy),Objekt(0,ObNr)
844 wU Feld(P(0),P(1))=ObNr
845 fh0 END SUB
846 Bj SUB ZielDel(P()) STATIC
847 712 Status(3)=Status(3)-1
848 1S CALL Effekt(.8,2,Ding())
849 7u CALL Del(P(0),P(1))
850 km0 END SUB
851 zJ SUB Effekt(Dauer!,Kanal,Pegel(2)) STATIC
852 hW2 FOR n=0 TO UBOUND(Pegel)
853 eD4 SOUND Pegel(n,0),Dauer!,Pegel(n,1),Kanal
854 N12 NEXT n
855 pr0 END SUB
856 Ux SUB Spielfeld(F(2),spx,spy) STATIC
857 Oa2 TIMER OFF
858 VS COLOR 1,0:CLS:xpos=jx:ypos=jy
859 j5 CALL Message("Die Ebene wird aufgebaut.")
860 ED FOR y=0 TO Feldy:FOR x=0 TO Feldx:PUT(xpos,ypos),Objekt(0
,F(x,y)),PSET:xpos=xpos+Obj:NEXT:ypos=ypos+Obj:xpos=jx:NE
XT
861 Or PUT (spx*Obj+jx,spy*Obj+jy),Spieler(0,0,0),PSET
862 Xd TIMER ON
863 xZ0 END SUB
864 kJ SUB Message(Tx$) STATIC
865 Df2 COLOR 1,0:LOCATE Skor,1:PRINT SPACE$(40);
866 7u CALL Zentri(Skor,Tx$)
867 130 END SUB
868 Xn SUB Inline(x,y,In$,MaxChar) STATIC
869 Sj2 g$=In$:z=0:In$=""
870 dg LOCATE y,x:PRINT SPACE$(MaxChar);:LOCATE y,x
871 pM WHILE g$<>CHR$(13)

```

```

872 ch4 IF g$ >= CHR$(32) AND z < MaxChar THEN
873 D16 In$=In$+g$:z=z+1
874 Kb PRINT g$;
875 n54 ELSEIF g$=CHR$(8) AND z>0 THEN
876 wQ6 z=z-1
877 mW IF z=0 THEN In$="" ELSE In$=LEFT$(In$,z)
878 jf LOCATE y,x+z:PRINT " ";
879 xL LOCATE y,x+z
880 814 END IF
881 7A LINE ((x+z-1)*8,y*8-1)-STEP(7,0),15
882 WA g$=INKEY$
883 gU2 WEND
884 UR PRINT " ";
885 JLO END SUB
886 1b SUB GetKlick(Inhalt$(),Maxnum,Nummer,Revers()) STATIC
887 vn2 WHILE MOUSE(0) < >0:WEND
888 ty WHILE MOUSE(0)=0
889 194 IF FNPC(MOUSE(1),MOUSE(2),0,56,317,216) THEN
890 b66 Nummer=INT((MOUSE(2)-7*8)/8)*2+INT(MOUSE(1)/(20*8))
891 yH4 ELSE
892 gF6 Nummer=Maxnum+1
893 LE4 END IF
894 rn IF Alt < > Nummer AND Alt <= Maxnum THEN
895 Og6 IF Revers(Alt)=1 THEN COLOR 6,11 ELSE COLOR 1,0
896 jQ CALL Position(Alt,Inhalt$(Alt))
897 PI4 END IF
898 4m IF Nummer <= Maxnum THEN COLOR 14,15:CALL Position(Numme
r,Inhalt$(Nummer))
899 Og Alt=Nummer
900 xL2 WEND
901 Zb0 END SUB
902 QT SUB Position(Nummer,Tx$) STATIC
903 Rs2 IF (Nummer MOD 2)=0 THEN x=2 ELSE x=21
904 HL LOCATE Nummer\2+8,x:PRINT Tx$;SPACE$(18-LEN(Tx$))
905 dF0 END SUB
906 Lc SUB Verzeichnis(Inhalt$(),Rev(),MaxEl) STATIC
907 SA2 COLOR 1,0:CLS
908 sa FOR n=0 TO MaxEl
909 m54 IF Rev(n)=1 THEN COLOR 6,11 ELSE COLOR 1,0
910 81 CALL Position(n,Inhalt$(n))
911 Id2 NEXT n
912 km0 END SUB
913 6K SUB ReadEbene(Nr,Satz,Sp(2),St()) STATIC
914 Lx2 FIELD #Nr,18 AS io1$,234 AS io2$,20 AS io3$
915 EF GET #Nr,Satz+1
916 nn n=1
917 KL FOR x=0 TO Feldx
918 TR4 FOR y=0 TO Feldy
919 tN6 Sp(x,y)=ASC(MID$(io2$,n)+CHR$(0))
920 KO n=n+1
921 4C2 NEXT y,x
922 mD FOR x=0 TO 9
923 114 St(x)=CVI(MID$(io3$,2*x+1))
924 pK2 NEXT x
925 xz0 END SUB
926 ch SUB WriteEbene(Nr,Satz,Nam$,Sp(2),St()) STATIC
927 YA2 FIELD #Nr,18 AS io1$,234 AS io2$,20 AS io3$
928 1v LSET io1$=Nam$:h$=""
929 WX FOR x=0 TO Feldx
930 fd4 FOR y=0 TO Feldy
931 hq6 h$=h$+CHR$(Sp(x,y))
932 FN2 NEXT y,x
933 XW LSET io2$=h$:h$=""
934 yP FOR x=0 TO 9
935 hL4 h$=h$+MKI$(St(x))
936 1W2 NEXT x
937 dm LSET io3$=h$
938 f5 PUT #Nr,Satz+1
939 BDO END SUB
940 m3 SUB Checkfile(File$) STATIC
941 H22 Fehler=0
942 eC OPEN File$ FOR INPUT AS 8:CLOSE 8
943 FHO END SUB
944 WO SUB Reaction STATIC
945 1U2 WHILE INKEY$ < > "" OR MOUSE(0) < > 0 OR STRIG(2) < > 0:WEND
946 QO WHILE INKEY$="" AND STRIG(2)=0 AND MOUSE(0)=0:WEND
947 nW WHILE INKEY$ < > "" OR MOUSE(0) < > 0 OR STRIG(2) < > 0:WEND
948 KM0 END SUB
949 12 SUB Zentri(Zeile,Tx$) STATIC
950 8k2 LOCATE Zeile,(41-LEN(Tx$))\2
951 p2 PRINT Tx$;

```

```

952 OQ0 END SUB
953 Ts SUB Rahmen(x1,y1,x2,y2) STATIC
954 172 TIMER ON
955 GE LINE (x1-1,y1-1)-STEP(x2+2,y2+2),0,BF
956 w7 LINE (x1,y1)-STEP(x2,y2),3,B
957 56 LINE (x1+2,y1+2)-STEP(x2-4,y2-4),3,B
958 UW0 END SUB
959 nQ ' Autor: Christian Rodemeyer, Goldregenstraße 28, 4370 Marl
-Sinsen
960 Jv ' Programmname: Suremosch (Denk&Strategiespiel in reinem Ba
sic)
(C) 1987 M&T

```

Listing 2. «Suremosch» (Schluß)

Programmname:	File-Dat-Maker
Computer:	Amiga 500,1000,2000 mit Kickstart 1.2
Sprache:	Amiga-Basic
Bemerkung:	Erzeugt Datendateien für Suremosch

```

1 Gv0 ' Suremosch Daten Erzeuger
2 Js ' Dieses Programm muß einmal fehlerlos durchlaufen sein, be
vor Suremosch
3 GU ' das erste mal gestartet wird. Es erzeugt in den verschied
enen
4 bX ' Verzeichnissen die entsprechenden Files.
5 JQ '
6 9r PRINT "Verzeichniss, in dem das Suremosch Hauptprogramm lie
gt"
7 wY INPUT V$
8 Qc ' Erstellung der Ebenen-Datei für den Editor
9 n2 OPEN "r",#1,V$+"/Nachlader/Editor.dat",272
10 67 FIELD #1,18 AS Ebene$,234 AS Spielfeld$,20 AS Status$
11 nQ LSET Ebene$="Leere Ebene!"
12 TR LSET Spielfeld$=STRING$(234,0)
13 XV LSET Status$=STRING$(20,0)
14 OH FOR n=1 TO 40
15 H92 LOCATE 3,3:PRINT "Writing ...";n
16 BB PUT #1,n
17 sD0 NEXT n
18 6q CLOSE 1
19 o4 ' Kennzeichnung des Verzeichnisses Wedongs als Wedong-Verze
ichniss
20 Oj OPEN V$+"/Wedongs/Wedong-Namen" FOR OUTPUT AS 1
21 Iu PRINT #1,"Mein erster Wedong"
22 Au CLOSE 1
(C) 1987 M&T

```

Listing 3. «File-Dat Maker» erzeugt die nötigen Daten-Dateien für Suremosch

Programmname:	Data-Lader
Computer:	Amiga 500,1000,2000 mit Kickstart 1.2
Sprache:	Amiga-Basic
Bemerkung:	Erzeugt Objekt-Dateien für Suremosch

```

1 aI0 ' Dieser Data-Lader erzeugt die nötigen
2 8b ' Binär-Dateien für das Suremosch-PRG.
3 ws ' Dazu muß das Verzeichniss angegeben
4 TN ' werden, in dem das Programm liegt.
5 IT DIM SHARED dat$
6 OM DEF FN Byte(z)=VAL("&h"+MID$(dat$,z,2))
7 BZ INPUT "Verzeichniss des Suremosch-Programms:";Vrz$
8 Dp CHDIR Vrz$
9 sc RESTORE SpDat:CALL Rdat("Nachlader/Spieler","SpDat")
10 QX RESTORE ObDat:CALL Rdat("Nachlader/Objekte","ObDat")

```

Listing 4. «Data-Lader» erzeugt weitere Dateien für Suremosch


```

11 JU RESTORE WeDat:CALL Rdat("Wedongs/Mein erster Wedong.Wedong"
12 wr END
13 hn SUB Rdat(File$,D$) STATIC
14 UF2 OPEN File$ FOR OUTPUT AS 1 LEN=1000
15 Xp z=1:READ dat$:Check=0
16 08 WHILE MID$(dat$,z,1) <> "*"
17 l24 IF MID$(dat$,z,1)="-" THEN
18 ff6 z=z+1:w=FN Byte(z):z=z+2
19 GV c=FN Byte(z):Check=Check+w*c*z
20 k6 PRINT #1,STRING$(c,CHR$(w));
21 wf4 ELSE
22 4h6 w=FN Byte(z):Check=Check+w-z
23 xS PRINT #1,CHR$(w);
24 KD4 END IF
25 9X z=z+2
26 R3 IF z > LEN(dat$) THEN READ dat$:z=1
27 sg2 WEND
28 U9 READ w:IF w <> Check THEN BEEP:PRINT "Fehler in den Datas
von ";D$;"!":STOP
29 H1 CLOSE 1
30 WYO END SUB
31 YI SpDat:
32 2Y DATA 001100110004-0004,03E000000080000000,80000001C0-0006,0
630000009480000
33 KL DATA 0F78-000603E00000,03E0000001C00000,01C000000080-0022,0
630-002E03E0000000
34 rJ DATA 8000000080000003,E0000007F000000F,F800001084000010,840
0001FFC00000F
35 lJ DATA F800000FF8000007,F0000007F0000003,E0000001C0-000A,03E0
0000080000000
36 rc DATA 80000003E0000007,F000001FFC000010,840000108400001F,FC0
Q000C1800000C
37 Y5 DATA 180000063000000E,3800000360000001,C0-000711001100,04-0
00403E0000000
38 eP DATA 8000000080000001,C0-000606300000,09480000F78-0006,03F
0000003F00000
39 6J DATA 01E0000000E0000000,20-00220630-002E,03E000000080000000,
80000003E0000007
40 SB DATA F000000FF8000010,840000108400001F,FC00000FF800000F,F80
00007F8000003
41 x6 DATA F8000001F0000000,70-000A03E0000000,8000000080000003,E0
000017F000001F
42 IA DATA F800001084000010,8400001FFC00000C,0800000C0C000006,1C0
0000718000009
43 sI DATA D000000070-0007,1100110004-0004,03E000000080000000,800
00001C0-0006
44 bB DATA 0630000009480000,0F78-000607E00000,07E0000003C00000,03
80000002-0023
45 U2 DATA 0630-002E03E0000000,8000000080000003,E0000007F000000F,
F800001084000010
46 Pg DATA 8400001FFC00000F,F800000FF800000F,F000000FE0000007,C00
00007-000B03
47 b4 DATA E000000080000000,80000003E0000007,F400000FFC000010,840
000108400001F
48 PR DATA FC0000081800001818,00001C3000000C70,000005C8000007-000
8,1100110004-0009
49 H4 DATA 80000001C0000001,C0000003E0000003,E0-00060F780000,0948
00000630-0006
50 lv DATA 01C000000080000000,80000003E0-002E,0630-001E01C00000,0
3E0000007F00000
51 VI DATA 07F000000FF80000,0FF800001FFC0000,1084000010840000,0FF
8000007F00000
52 jc DATA 03E000000080000000,80000003E0-000A,01C0000003600000,0E
38000006300000
53 h6 DATA 0C1800000C180000,1FFC000010840000,108400001FFC0000,07F
0000003E0000000
54 rx DATA 8000000080000003,E0-000711001100,04-000920000000,E0000
001E0000003
55 cM DATA F0000003F0-0006,0F78000009480000,0630-000601C0000000,8
000000080000003
56 dV DATA E0-002E0630-001F,70000001F0000003,F8000007F800000F,F80
0000FF800001F
57 hz DATA FC00001084000010,8400000FF8000007,F0000003E0000000,800
0000080000003
58 G7 DATA E0-000B70000009,D000000718000006,1C00000C0C00000C,0800
001FFC000010
59 35 DATA 840000108400001F,F8000017F0000003,E000000080000000,800
00003E0-0007
60 k9 DATA 1100110004-0008,0200000003800000,03C0000007E00000,07E0
-00060F780000
61 Fd DATA 094800000630-0006,01C000000080000000,80000003E0-002E,0
630-001E07000000

```

```

62 ce DATA 07C000000FE00000,0FF000000FF80000,0FF800001FFC0000,108
4000010840000
63 k3 DATA 0FF8000007F00000,03E000000080000000,80000003E0-000A,07
00000005C80000
64 FD DATA 0C7000001C300000,1818000008180000,1FFC000010840000,108
400000FFC0000
65 Gy DATA 07F4000003E0000000,8000000080000003,E0-000711001100,04
-001001800000
66 Xm DATA 0280000042B00000,49BC00000783E0000,49BC000042B00000,028
000000180-0026
67 Vu DATA 0200000002-000F,0200000002-0023,01C0000002700000,067C0
0004E7E0000
68 JS DATA 4E7F000007FFF0000,4E7F00004E7E0000,067C000002700000,01C
0-001A03C00000
69 MS DATA 02740000067C0000,4E4E00004E430000,7FC100004E430000,4E4
E0000067C0000
70 oc DATA 0274000003C0-000F,1100110004-0010,0180000002800000,42B
0000049B80000
71 I5 DATA 783C0000049BC0000,42BE000002B00000,0180-002602000000,02
-000F02000000
72 F5 DATA 02-002301C00000,0270000006780000,4E7C00004E7E0000,7FFE
00004E7F0000
73 Vh DATA 4E7F0000067F0000,027C000001C0-001A,07C0000002720000,06
7C00004E4C0000
74 6f DATA 4E4600007FC20000,4E4300004E410000,064F0000027C0000,01D
8-000F110011
75 ey DATA 0004-001001800000,02B0000042BE0000,49BC00000783C0000,49
B8000042B00000
76 CX DATA 028000000180-0026,0200000002-000F,0200000002-0023,01C0
0000027C0000
77 nR DATA 067F00004E7F0000,4E7F00007FFE0000,4E7E00004E7C0000,067
8000002700000
78 rh DATA 01C0-001A01D80000,027C0000064F0000,4E4100004E430000,7F
C200004E460000
79 sK DATA 4E4C0000067C0000,0272000007C0-000F,1100110004-0011,C00
0000A0000006
80 t9 DATA A100001EC9000003E,0F00001EC9000006,A100000A00000000,C0-
002720000000
81 y6 DATA 20-000F20000000,20-002201C00000,072000001F300000,3F390
0007F390000
82 iX DATA 7FFF000007F390000,3F3900001F300000,0720000001C0-001A,01
E0000017200000
83 BO DATA 1F30000039390000,6139000041FF0000,6139000039390000,1F3
0000017200000
84 NO DATA 01E0-000F110011,0004-0011C0000000,A0000006A100000E,C90
0001E0F00001E
85 oU DATA C900003EA1000006,A0000000C0-0027,2000000020-000F,20000
00020-0022
86 Wg DATA 01C0000007200000,0F3000001F390000,3F3900003FFF0000,7F3
9000007F390000
87 7A DATA 7F3000001F200000,01C0-001A01F00000,272000001F300000,19
39000031390000
88 KI DATA 21FF000061390000,4139000079300000,1F2000000DC0-000F,11
00110004-0011
89 pA DATA C0000006A000003E,A100001EC900001E,0F00000EC9000006,A10
0000A0000000
90 wp DATA C0-002720000000,20-000F20000000,20-002201C00000,1F2000
007F300000
91 PJ DATA 7F39000007F390000,3FFF00003F390000,1F3900000F300000,072
0000001C0-001A
92 rs DATA 0DC000001F200000,7930000041390000,6139000021FF0000,313
9000019390000
93 Wx DATA 1F30000027200000,01F000000000000000000000000000000000* 94
5e DATA 61864
95 mE ObDat:
96 wR DATA 001100110004-00FF,-00121100110004,84C080-0009123C,0000
3B60000004C0
97 Mg DATA 000004A000008620,0000055E80000013,0000B8A1000013EA,800
0175E00001218
98 4j DATA 0000100000000018080,00008C80007B3F0000,DFDF8000B37C8000
,ED408000C0968000
99 rP DATA B9358000B9538000,681F8000D2810000,FDAC8000475E8000,EC1
50000A8818000
100 Az DATA E9A78000AD3F8000,B84D0000FF730000,F5DC8000DA880000,814
88000133C0000
101 R1 DATA FB648000A4E40000,A4F18000A6220000,855E8000CC938000,BEA
F800097FF8000
102 X8 DATA 175F0000D23F8000,1C02000001808000,B1CC8000FFFF8000,DFD
F8000B37C8000
103 9q DATA FF7C8000FBF68000,BDF58000BDF38000,EE3F8000D7DF8000,FDB
F8000FFFF8000
104 1P DATA FFFF8000BDF8000,FBFB8000BD3F8000,B9CD8000FFFF800000,1
100110004DFFD80

```

```

105 A6 DATA 00C1C180-000584,9080009040800089,24800084908000D2,0980
00C1258000C4
106 tI DATA 9180009240800089,2480008490800092,4880-0005E1C180,00DF
FD8000E00380
107 ts DATA 00BE3E8000FFFFF80,0076DB000071470000,6DB7000076D30000,3
A2E000025B60000
108 dI DATA 369A000073670000,6DB7000076D30000,7B7F0000FFFF8000,9E3
E8000E0038000
109 G1 DATA DFFD8000C1C18000,3E3E0000A9268000,AEB8000B24A8000,A92
E8000E5D38000
110 BJ DATA DA498000E9678000,AC9A8000B24A8000,A92E8000A4828000,3F7
E0000E1C18000
111 p5 DATA DFFD8000FFFF8000,FFFF8000DFFD8000,E0038000EAA8000,E00
38000E80B8000
112 6M DATA E1438000E80B8000,E1438000E80B8000,E0038000E92B8000,E00
38000DFFD8000
113 vd DATA FFFF8000FFFF800000,1100110004-0008,1C0E-000A0120000000
,C000000120000001
114 wH DATA 20000000C0000001,20-000A1C0E-0012,700380001C0E-0006,03
3000000120-0012
115 f5 DATA 012000000330-0006,1C0E0000700380-0015,0618-00220618-00
27,C000000120000002
116 Me DATA 1000000210000001,20000000C0-001B,1100110004-0008,18000
00024000000
117 Wo DATA 2400000018000000,0320000003C00000,01E0000000F00000,013
000000006000000
118 Cf DATA 0900000090000000,06-000E38030000,6404800042080000,4210
000026300000
119 eG DATA 1F60000003E00000,01E0000001F00000,01BE000003190000,021
0800004108000
120 gT DATA 480980003007-000A,3803000064048000,4208000042100000,26
3000001F600000
121 h4 DATA 03E0000001E00000,01F00000001BE0000,0319000002108000,041
0800048098000
122 W0 DATA 3007-004B110011,0004-000818060000,2409000024090000,1A1
6000003F00000
123 hD DATA 01E0000000C0-000F,C000000120000001,20000000C0-004E,1A1
6000025290000
124 wC DATA 42D0800042108000,273900001FFE0000,03F0000044C88000,48C
4800048C48000
125 gG DATA 21E1000021210000,121200000A140000,0738000000C0-0006,1A
16000025290000
126 G1 DATA 42D0800042108000,252900001C0E0000,0210000044088000,48C
4800048C48000
127 JI DATA 21E1000021210000,121200000A140000,0738000000C0000000,1
100110004-0004
128 wL DATA 0406000024190000,4224000058240000,2519800025820000,19B
8000000F00000
129 L1 DATA 03C0000007660000,1069000066290000,0906800009108000,260
900001808-0006
130 Df DATA 0406000020010000,4000000040-0005,80-000680000000,E0000
001C00000000
131 Nb DATA 40-000640-0005,8000000080002001,00001808-000604,060000
2001000040
132 nP DATA 00000040-000580,-001940-000580,00000800020010000,1808
-00061C060000
133 DY DATA 2601000043008000,4100800001818000,0183000001BE000000,F
8000007C000001F
134 IS DATA 600000306000006060,0000402080004030,800020190000180E,0
000001100110004
135 o9 DATA -000401C0000003,E000001084-0006,0410000022A20000,61C30
00073E70000
136 XH DATA 61C3000022A20000,0410-000610840000,03E0000001C0-000A,0
1C0000003E00000
137 SN DATA 1084-000A22220000,61C3000071C70000,61C300002222-000A,1
084000003E00000
138 WX DATA 01C0-000A01C00000,03E000001084-000A,2002000060030000,7
007000060030000
139 k5 DATA 2002-000A10840000,03E0000001C0-000E,0360000011440000,0
9C8000004900000
140 yq DATA 22A2000039CE0000,0F78000039CE0000,22A2000004900000,09C
8000011440000
141 aX DATA 0360-000B110011,0004000400001208,00000E7000002340,0000
194E00003DB0
142 sT DATA 0000427C000005A2,00001A5800007248,000025A60000404A,000
00EA800001294
143 Of DATA 0000144000003438,000002-00472106,0000324800000E74,0000
63410000194E
144 GN DATA 00003DB10000C3FC,0000066200009C39,0000742800002666,000
041CA00004EA9
145 lJ DATA 0000129400001442,0000353800002204,0000210200002040,000
0000400004001

```

```

146 mi DATA -00070100008180,0000024000008421,0000042000000240,0000
018000004001
147 RG DATA -000702000001000000,2004000000110011,0004-00080E380000
,7F7F000067F30000
148 85 DATA 67F300007FFF0000,3DDE000001400000,0140-000601C00000,01
C0000010840000
149 rw DATA 0C98000007F0-000E,0808000004100000,0360000001C0000000,
8000000080000000
150 ya DATA 8000000080000000,80-000602200000,077000000360-0032,014
0000002A00000
151 iW DATA 06B000001EBC0000,0888000007F0-000A,1E3C00003F7E0000,67
F3000043E10000
152 RT DATA 43E1000067F30000,3DDE000001C00000,01C0000001C00000,03E
0000007F00000
153 rJ DATA 1FFC00000FF80000,07F0-0007110011,0004FFFF8000FE3F,8000
EC9B8000C551
154 4u DATA 8000EC9B8000FE3F,8000C7F180009364,8000AA2A80009364,800
0C7F18000FE3F
155 Qh DATA 8000EC9B8000C551,8000EC9B8000FE3F,8000FFFF80008220,800
044110000280A
156 CA DATA -0006280A000044,1100008220800001,C0000001C0000001,C000
008220800044
157 9X DATA 110000280A-0006,080A000004110000,02208000FC1F8000,F9CF
8000E3E38000
158 HI DATA C3E18000C3E18000,81C08000380E0000,7C9F00007DDF0000,7C9
F0000380E0000
159 IU DATA 81C08000C3E18000,C3E18000E3E38000,F9CF8000FC1F8000,03E
0000007F00000
160 pS DATA 1F7C00003EBE0000,3F7E00007FFF0000,FFFF8000EF7B8000,D63
58000EF7B8000
161 NW DATA FFFF80007FFF0000,3F7E00003EBE0000,3F7C000047F00000,83E
00000*
162 yu DATA 103418
163 Hu WeDat:
164 Nq DATA 4469657365722057,65646F6E6720656E,7468E46C74204562,656
E656E20616C6C
165 RD DATA 65720A5363687769,657269676B656974,7367726164652C20,766
F6E2073656872
166 e8 DATA 206C65696368740A,626973207A69656D,6C69636820736368,776
5722E20566965
167 y6 DATA 6C20537061DF210A,-20607363682056,312E32-20052A0000,220
03AAFE8202A-2017
168 rd DATA 2A20000021003AAF,E8202A2020436F70,7972696768740003,C7B
C4C616E672C20
169 lX DATA 61626572206C6569,63687420-010A02,010101-000B0101,00-01
04-0005010101
170 qz DATA 000003-00040100,0102-01040001000000,0100010A0101000000
,010000000100010A
171 jy DATA 010100010101000000,0100010A01010001,0000070000010005,0
A010100010301000000
172 SE DATA 0101010A0101-0009,010A01010001000000,03000000010A0101,
000101000300030000
173 j5 DATA 010A010100010003,0003000300010A0101,000100000300030000
,0106010100010003
174 Ct DATA 090308030000000101,06-01040A-0104,060101-000B-010E,000
1000102BC0010
175 61 DATA 00630000-000503,000246FC7220482E,472E57656C6C73-2005,0
102-010C080A08
176 bj DATA 0A080A010A080A08,01010A080A080A08,0A010A080A010108,0A0
80A080A080A01
177 02 DATA 0A0801010A080A08,04040A080A010A0101,080A080A0404080A,0
80A0101010A080A
178 nm DATA 0804040A080A010A,0101080A080A080A,080A010A0801010A,080
A080A080A010A
179 ry DATA 080A01010109010A,080A010A080A08010101,0001080A010A080A
,080A01010000-010B
180 VP DATA 000101-0007010101,0006-0009010100,06-00060100010101,00
06-000501010000
181 lq DATA 0101-0006010200,0300-010E001000,0B015E000A-0009,050001
5365687220
182 RW DATA 65696E6661636820,6F6465723F04-010B,040101-0009010101,0
006-000706000101
183 rh DATA -000B0101-000B,0101-000B0101000000,-0A050000000101,000
0000A030A030A
184 KZ DATA 0000000101010000,0A0A020A0A0000-0104,00000A0A020A0A000
0,0101010000000A03
185 Oz DATA 0A030A0000000101,000000-0A05000000,0101-000B0101-000B,
0101-000B010100

```

Listing 4. (Fortsetzung)


```

186 YT DATA 06-00070600010101,-0009010104-010B,040001000600FA00,08
-000901000045
187 RU DATA 696E66616368206C,E4636865726C6963,68-0005010101-0009,0
101000101-0007
188 zb DATA 01010000000101-0005,0101-00050101000000,0101-000701010
0,0101-0009010101
189 DK DATA 03-000903010101,0000000908000000,-01050009000200,080
0-0106000900
190 nv DATA 02000800-0105000000,090908000000010101,03-000903010101
,-00090101000101
191 Ok DATA -00070101000000,01-00060101-0005,01010000000101-0007,0
101030101-0009
192 hZ DATA 010101-00060100,06000A00050014-0009,014B6F6D62696E69,6
572652067757421
193 is DATA 2020200A0101010A,010A0A01010A010A0A,010A0A010A04010A0A
,010A01010A0A030A
194 N8 DATA 010108-0A040101,0A0A0A08010A0401,0A0A0A010A010901,-0A0
4010A0A0A0101
195 Uq DATA 0505-0A07010A01,0A05010A01-0A05,010A0101070A0A0A,01010
A080A0A0A0101
196 gu DATA 0A0A0A0102010A01,0A03010A010A0A07,010A0102010A0A0A,010
1-0A05010108
197 19 DATA 0A010A01010A01-0A08,010A0A010A0A0A0101,0A0A0A010A01010
A,000A0105-0A060101
198 Ws DATA -0A05010A010A0A0A,01010A0A030A010A,01050A060A010101,0A
0A010A070A0A01
199 hI DATA 080A010A0A01010A,010A01010A01010A,000D000200280006,002
D00030001-0005
200 d5 DATA 01496E2052656968,20756E6420476C69,656420-0109-0204,01-
000401-0006
201 su DATA 0201-000406000000,0400000201-0004,01-0004040002010101,
04010101-00050101
202 Ua DATA 000000030001-0005,01010000000300-0104,0001010100000400
00,01-000501010003
203 p7 DATA 0000000100050005,0001010003000000,010000050000-0104,04
-010405000500
204 g1 DATA 01010A0A0A080908,01-000401010A0A0A,-010500-0104-0A06,0
1-00040102-0A06
205 G1 DATA 01000300000102-0A07,000003000102-0A06,01-000401-0204,-
01090001000201
206 Lf DATA 2C00110063000600,05-00064D697474,656C736368776572,-200
6-010E020000
207 X8 DATA 01020000000A0A0A,0101000005010101,00010A0A0A010101,050
0050000030007
208 dL DATA 0A0A010101000005,000300010A0A09010101,050000000100010A
0A,0101010000050100
209 8K DATA 010000010A000101,0006-0004010001,030001010000010101,00
00060000000101
210 w3 DATA 000100000001-0005,0101010000030000,01-000401010000,030
0010000010000
211 N6 DATA 0701010003000100,0100000100000101,0000010003090100,010
0010101000401
212 oC DATA 0000010000010000,0101-0004020800,0101010701010000,0100
0001-0005-010E
213 Km DATA 0002000201400010,00630004-000701,4E756C6C2050726F,626C
656D6F-2005
214 m0 DATA 02-010B0201-0007,070A000001010002,030301010001000000,0
101000102000004
215 np DATA 00000001000101000000,0A060A0100070000,0101000100000A00
0000,0107000101000000
216 zH DATA 01000100000100010101,0004070100000100,0100080101070001
,-0007090101000000
217 ZP DATA 0701000003000008,0101000100000A01,-0004090101000101,00
0A0400010000010101
218 nX DATA 000001000A0001-0004,0101-000401-0005,0101010006-0004,0
10001000000101-0004
219 1h DATA 0204-0005010100,07-0008070102-010B,0200040008019000,09
00090003-0008
220 sz DATA 46FC722064656E20,416E66E46E676572,20200A-010B0A01,0007
-020507000000
221 q0 DATA 0101070707020202,0707070000010109,07000702000007000000
,0101-000407-0006
222 eu DATA 0101-000408-0006,01010008010101-0006,-01050001-0004,05
00010100000500
223 FJ DATA 01000000-0105-0004,010000000100000101,-000903080101-00
08,0100000101-0008
224 GC DATA 09-0104-0007-0A04,01010A060000030000,0A06060A0101-0007
,0A06060A010109-0006
225 15 DATA -0A04010A-010B,0A0001000B00C800,090019-00090152,FC636B
6B6F70706C
226 Rh DATA 756E67-20060102,-010C0001-0005,0100000001010003,000003
0100060000

```

```

227 BS DATA 01010100010001000000,0101000001010000,0300070A01000100
,0901010000000100
228 Na DATA 0100000100010101,0001010001080807,0000000101-0005,0100
00010100010101
229 QU DATA 00010003000001000000,0101000007000001,00000100010101-0
004,01010200000300010101
230 yQ DATA 00010108-010504,010100000100070000,010000000101-0004,0
1-000501010100
231 gB DATA 030103000000001000000,0101-0004030100,0100020001010700,
01-0008-010E00
232 Em DATA 06000400C8000900,3200630063-0005,0141756368206E69,6368
7420FC62656C
233 nD DATA 212020-010E-0008,010000010202030202,090100000100000101
,0000000100050001
234 YJ DATA 0000030101040900,0100010700030000,0101000000010001,000
0010300010107
235 Ir DATA 010100040001-0004,0101000001000004,0001000303010101,00
00010000040001
236 RE DATA 0000010108000600,010000040001020101,080000030001000000
,05000101080001000000
237 gq DATA 0101010000010101,000001070007000000,010101000007-0005,
01010201010001000000
238 TR DATA 0100030003000101,0001000701000501,0000000101000000,010
2-0006-010E
239 kJ DATA 0008000A012C000E,003200080063000000,0400014574776173,2
073656C7473616D
240 qQ DATA 21-200401-020B,0101020201020200,02020102020101-0205,00
-02050101-0204
241 aF DATA 070007-02040101,020207070703070707,02020101-070401,030
1-07040101070707
242 2g DATA 0101030101070707,0101070707010405,0401070707010108,070
3030509050303
243 CV DATA 0708010107070701,040504010707070101,0707070101030101,0
707070101-0704
244 qb DATA 010301-07040101,0707070301030103,0707070101070703,0001
020100030707
245 Z1 DATA 0101070300000102,0100000307010103,000000020202000000,0
3010100000020202
246 SL DATA 0102020200000101,-020B0100010006,0064000E00320063,-000
50A00144D61
247 n1 DATA 6C20776173206C65,6963687465732020,-010E-0007040000,030
2010A0100000010101
248 6z DATA 000100000101040A,01-000701010102,0A0A010003010301,0000
010107010A0A
249 LG DATA 01-000406000101,0A0A010A0A010501,03000001010A0A0A,010A
010A010000010101
250 hb DATA 0A060A0A0A010A01,0001000101010A010101,0A0A0A0100000101
,0A0A010A00040A00
251 cI DATA 01000001010A0A01,0A00040A-00040101,0A0A0A02070A0101,00
04000101000004
252 z4 DATA 0700000001010000,0101-000601000000,0101010A040A01-0A04
,00040001010A0A0A
253 aS DATA 010A010A0A010009,-010E0001000101,AE000F0000000400,08-0
0502457273
254 hJ DATA 7465722053746170,656C-2005-010B,020101000103010A,050A-
00040201-0004
255 8u DATA 0A0A01000001000101,000105010A0A01-0004,0101000104010A0
A0A,0000010001010000
256 ry DATA 04-0104-00040101,05-000601010000,0101000500010101,0000
00-01040000
257 Eq DATA 010908000101000000,01010201090901000000,01020001010000
09,01000006-00040101
258 k7 DATA 030007000101000101,0000010100030001,-0A050100010103,00
0001-06050100
259 DF DATA 0101000000010A01,0A010A0100010100,0701090A070A070A,070
00101000001-0006
260 wM DATA 0700-0107020102,-01040010000202,580010000F000400,02-00
0501*
261 Ck DATA 75223

```

(C) 1987 M&T

Listing 4. »Data-Lader« (Schluß)

Dieses Programm ist unentbehrlich beim Abtippen unserer Listings. Es hilft, Tippfehler zu vermeiden und spart viel Zeit.

Ein längeres Listing ohne Fehler abzutippen ist (fast) unmöglich. Aus diesem Grund haben wir in Ausgabe 3/88 des AMIGA-Magazins eine Eingabehilfe — den Checksummer »Checkie 42« veröffentlicht. Die hier vorgestellte Version 1.1 enthält erweiterte Funktionen und bietet mehr Komfort. Damit möglichst viele unserer Leser dieses Programm auch tatsächlich anwenden, haben wir es möglichst kurz gehalten und in einer Sprache programmiert, die alle Abtipper besitzen: Amiga-Basic.

Die Form der Listings

Die Listingzeilen bestehen aus einer bis zu vierstelligen Zeilennummer, der zwei- beziehungsweise dreistelligen Prüfsumme und der eigentlichen Programmzeile. Beispiel:

```
10 TT0 print "Hallo!"  
   |  
   |— Prüfcode  
   |— Zeilennummer
```

Nach einer Leerstelle im Anschluß an die Zeilennummer stehen bis zu drei Zeichen Prüfcode. Die einzelnen Zeichen können sein eine Ziffer (»0« bis »9«), ein kleiner Buchstabe (»a« bis »z«) oder ein Großbuchstabe (»A« bis »Z«). Die ersten beiden Zeichen der Prüfsumme sind der eigentliche Prüfcode. Im dritten Zeichen ist die Spaltenposition der ersten »Nicht-Leerstelle« verschlüsselt. Das ist für diejenigen Anwender interessant, welche die Struktur des Listings, also die Einrückungen durch Leerzeichen, übernehmen wollen. Ist dies nicht Ihre Absicht, können Sie die Eingabe der Checksumme schon nach den ersten beiden Zeichen mit <Return> abschließen. Bei Checkie 42 muß die Groß- und Kleinschreibung so wie im Listing abgedruckt übernommen werden.

Eingabehinweis:

Geben Sie »Checkie 42« (Version 1.1) bitte mit Checkie 42 aus der AMIGA-Ausgabe 3/88 oder 12/87 ein. Sollten Sie die alte Eingabehilfe nicht besitzen, so tippen Sie das Listing für die Version 1.1 im normalen Basic-Editor ohne Prüfsummen und Zeilennummern ab.

Der Umgang mit Checkie 42

Nach dem Start fragt das Programm nach einem Dateinamen. Unter dem angegebenen Namen speichert Checkie 42 die eingegebenen Listingzeilen ab. Existiert bereits eine Datei mit diesem Namen auf der Diskette, so haben Sie mit der Abfrage »Nur Checksummer ausgeben?« zwei Möglichkeiten:

- <j> Ausgabe der Datei mit Checksumme auf den Bildschirm oder den Drucker.
- <n> Einlesen der Programmzeilen aus der vorhandenen Datei und Eingabe der Checksumme mit der Tastatur.

Beide Alternativen sind gedacht für Anwender, die ein Listing nicht mit dem Zeileneditor des Checkie, sondern mit einem schnelleren und/oder komfortableren Editor ihrer Wahl — zum Beispiel dem Editor von Amiga-Basic (mit »..«, »a« speichern) erfaßt haben.

Checkie 42 errechnet nach der Eingabe <j> die Prüfsummen Ihres Textes und Sie können diese dann mit dem Listing im AMIGA-Magazin vergleichen. Bei der Ausgabe auf den Bildschirm schreibt das Programm die Programmzeilen inklusive Checksummen zusätzlich in eine Datei auf Diskette mit dem Zusatz ».chk«. Diese können Sie später zum Beispiel mit dem CLI-Befehl TYPE ohne erneute Berechnung der Prüfsumme noch einmal ausgeben.

Haben Sie »Nur Checksumme ausgegeben?« mit »n« beantwortet, dann können Sie dem Programm den Vergleich überlas-

sen, in dem die Frage »Eingabe aus Datei« mit »j« beantwortet wird. Dann brauchen Sie nur noch die Checksummen eingeben. Der Checksummer holt sich die Zeile aus der angegebenen Datei statt von der Tastatur. Entspricht die eingegebene Prüfzahl nicht der errechneten, kann die Zeile gleich korrigiert werden.

Beantworten Sie obige Frage mit »n«, zählt Checkie die in der Datei vorhandenen Zeilen und wartet mit der Zeilennummer »Anzahl+1« auf die Eingabe einer neuen Zeile. Alle weiteren Eingaben hängt das Programm an die bestehende Datei an. Diese Funktion ist sinnvoll, wenn Sie ein Listing in mehreren Teilen abtippen wollen.

So tippen Sie Listings ab

Haben Sie anfangs einen Dateinamen eingegeben oder Sie hängen neue Zeilen an eine bestehende Datei an, dann arbeiten Sie im normalen Eingabemodus. Checkie 42 schlägt dabei eine Zeilennummer vor und wartet auf die Prüfsumme. Nach Eingabe derselben taucht der Cursor zwischen den zwei Trennstrichen auf. Dort muß nun die Zeile »ohne« Zeilennummer und Prüfsumme eingegeben werden. Nach Betätigen der Taste <Return> berechnet Checkie die Prüfsumme. Leerstellen vor und hinter der Programmanweisung werden ignoriert. Stimmen Programmzeile und Prüfsumme mit derjenigen im Listing überein, speichert der Checksummer die Eingabe ab und wartet auf die nächste Zeile.

Einfügemodus: Wahrscheinlich wird eine abgetippte Zeile mal einen Fehler enthalten. Checkie 42 positioniert den Cursor dann an den Anfang der Zeile und wartet auf die korrekte Eingabe. Korrekturen lassen sich mit der Backspace- oder Delete-Taste durchführen. Um Zeichenfolgen einzufügen, kann kurzfristig mit <F2> der Einfügemodus eingeschaltet werden. Dieser Modus sollte allerdings nach der Fehlerkorrektur wieder ausge-

schaltet werden, da er die Eingabe verlangsamt.

Sonderfall-Prüfsumme ignorieren: Möchten Sie zum Beispiel eine Kommentarzeile

nicht »original« übernehmen, läßt sich trotz einer falschen Prüfsumme eine Übernahme der Zeile mit der Funktionstaste <F6> erzwingen. Sie können damit aber auch falsche Programmzeilen übernehmen. Verwenden Sie deshalb die Taste <F6> nicht gewohnheitsmäßig. Der Checksummer teilt Ihnen nach Beenden des Programms mit, wieviel Zeilen er ungeprüft übernommen hat.

Prüfsumme und Zeilennummer ändern: Natürlich kann es auch vorkommen, daß die Programmzeile zwar richtig abgetippt wurde, sich bei der Prüfsumme aber ein Fehler eingeschlichen hat. Nach Betätigen von <F1> kann die Prüfsumme korrigiert werden. Während der Eingabe der Prüfsumme läßt sich mit <F7> die vom Programm vorgeschlagene Zeilennummer verändern. Damit können Sie gezielt nur bestimmte Teile eines Listings übernehmen.

Haben Sie eine mit einem anderen Editor geschriebene Programmdatei überprüft und nur in wenigen Zeilen Fehler festgestellt, lassen sich durch Vorgabe der Nummern diese Zeilen gezielt ändern. Bei Angabe der Zeilennummer in aufsteigender Reihenfolge benötigt das Programm übrigens erheblich weniger Zeit für die Suche der Zeilen in der jeweiligen Datei. Um die versehentliche Übernahme fehlerhafter Zeilen zu verhindern, sperrt das Programm bei fehlender Übereinstimmung der Prüfsummen die Taste <F7> (Änderung der Zeilennummer).

Fehlerfrei abtippen

Eingabe beenden: Die Kombination <Ctrl-e> beendet den Programmlauf nach vollständiger Eingabe des Listings oder für eine Unterbrechung.

Am Schluß noch ein Tip für alle Leser, denen unser Basic-Editor zu langsam ist. Die Berechnung der Prüfsummen erfolgt im Unterprogramm »CalcSumme«. Dieser Teil ist sehr einfach in schnellere Sprachen, wie beispielsweise C, umsetzbar.

Wer schon einmal Fehler in einem abgetippten Listing gesucht hat, der weiß, wie frustrierend diese Arbeit sein kann. Nutzen Sie deshalb den »Checkie 42«. Sie sparen viel Zeit und müssen sich nicht dauernd auf die Suche nach tückischen Fehlern begeben.

(Dieter Behlich/kn)


```

180 LB2 ELSEIF e=8 THEN
181 4h4 IF i>1 THEN
182 sZ6 i=1-1 : REM <BS>
183 XY LOCATE sy,sx+1 : PRINT "."
184 un4 END IF
185 Tn2 ELSEIF e=13 THEN
186 KZ4 IF i=AnzCsZ THEN i=AnzCsZ+1 : REM <CR>
187 cL2 ELSE
188 ra4 IF e>47 AND e<58 THEN
189 v16 e=e-48 : REM 0-9
190 A14 ELSEIF e>64 AND e<91 THEN
191 Rv6 e=e-55 : REM A-Z
192 jX4 ELSEIF e>96 AND e<123 THEN
193 Zx6 e=e-61 : REM a-z
194 jS4 ELSE
195 yz6 GOTO blinken : REM weder noch
196 6z4 END IF
197 JY PRINT e$;
198 ve cs(i)=e
199 D7 i=i+1
200 A32 END IF
201 wn IF i<=AnzCsZ THEN blinken
202 F50 ESEnde:
203 2K2 COLOR 1,0
204 Oh LOCATE sy,sx-15
205 c0 PRINT "Checksumme:"
206 gI RETURN
207 aE0 NeuZeile:
208 dU2 IF FZok = wahr THEN
209 P44 NeuZeile=0
210 nn WHILE e<>13 OR NeuZeile=0
211 Ux6 LOCATE zy,1:PRINT USING "####";NeuZeile;
212 oF e=ASC(INPUT$(1))
213 gP IF e>47 AND e<58 THEN NeuZeile=NeuZeile*10+e-48
214 Jn IF NeuZeile > 9999 THEN e=8
215 in IF e=8 THEN NeuZeile=INT(NeuZeile/10)
216 vJ4 WEND
217 kh IF Checkfile THEN
218 tQ6 IF NeuZeile < Zeile THEN
219 y38 WHILE NOT EOF(1)
220 VGA LINE INPUT #1,e$
221 cd PRINT #2,e$
222 Ip8 WEND
223 MH CLOSE 1 : CLOSE 2
224 fY GOSUB backup
225 Xa OPEN dn$+".bak" FOR INPUT AS #1
226 Tl OPEN dn$ FOR OUTPUT AS #2
227 9t Zeile=1
228 cV6 END IF
229 Yp WHILE (NeuZeile > Zeile) AND (NOT EOF(1))
230 fQ8 LINE INPUT #1,e$
231 mn PRINT #2,e$
232 l7 Zeile=Zeile+1
233 C06 WEND
234 cR IF EOF(1) THEN
235 bL8 CLOSE 1
236 8i NeuZeile=Zeile
237 uN LOCATE zy,1:PRINT USING "####";NeuZeile;
238 9G Checkfile=0
239 ng6 END IF
240 oh4 END IF
241 vV Zeile=NeuZeile
242 qJ2 END IF
243 Ht0 RETURN
244 1H EingabeZeile:
245 n72 x=cs(AnzCsZ)
246 BWO weiter:
247 ga2 cy=zy+INT(x/LBZeile):cx=zx+(x MOD LBZeile)
248 vy LOCATE cy,cx
249 o6 COLOR 0,1
250 mI PRINT CHR$(z(x));
251 y1 LOCATE cy,cx
252 68 IF x>apos THEN apos=x
253 xx IF Checkfile AND FZok THEN
254 wL4 IF EOF(1) THEN
255 OR6 Checkfile=0 : CLOSE 1
256 jS4 ELSE
257 ix6 e$=INPUT$(1,1)
258 6z4 END IF
259 mV2 ELSE
260 Q24 e$=INKEY$
261 922 END IF
262 gp4 IF e$="" THEN weiter
263 OI COLOR 1,0
264 OW PRINT CHR$(z(x));
265 CF LOCATE cy,cx
266 8G e=ASC(e$)
267 UR2 IF ((e AND 127)<32) OR e=127 THEN Controlcode
268 op IF imode THEN GOSUB insert
269 I6 PRINT e$
270 K1 z(x)=e : e=30

```

```

271 3D0 Controlcode:
272 gF2 IF e=13 OR e=10 THEN
273 lN4 RETURN
274 092 ELSEIF e=30 THEN
275 2p4 a=1
276 Ea2 ELSEIF e=29 THEN
277 034 a=LBZeile
278 6G2 ELSEIF e=31 THEN
279 DD4 a=-1
280 Gb2 ELSEIF e=28 THEN
281 Rg4 a=-LBZeile
282 9s2 ELSE
283 UX4 GOTO noCrS
284 WP2 END IF
285 hz x=x+a
286 5Z IF x>=0 AND x<LZeile THEN weiter
287 tD x=x-a
288 HB GOTO weiter
289 kE0 noCrS:
290 dN2 IF e=8 THEN
291 lS4 IF x>0 THEN
292 Kk6 x=x-1
293 AM LOCATE zy+INT(x/LBZeile),zx+(x MOD LBZeile)
294 Bb FOR i=x TO apos
295 788 z(i)=z(i+1)
296 Ja PRINT CHR$(z(i));
297 Z0 IF i MOD LBZeile=59 THEN PRINT:PRINT TAB(zx);
298 FV6 NEXT i
299 pr z(apos)=32 : PRINT " "
300 yR apos=apos-1
301 ng4 END IF
302 H12 ELSEIF e=127 THEN
303 Kk4 FOR i=x TO apos
304 GH6 z(i)=z(i+1)
305 SJ PRINT CHR$(z(i));
306 i9 IF i MOD LBZeile=59 THEN PRINT:PRINT TAB(zx);
307 Oe4 NEXT i
308 y0 z(apos)=32 : PRINT " "
309 7a apos=apos-1
310 Vy2 ELSEIF e=129 THEN
311 oI4 GOSUB EingabeSumme
312 sC x=cs(AnzCsZ)
313 9U2 ELSEIF e=130 THEN
314 513 imode=imode XOR 1
315 tU5 LOCATE 7,28
316 y23 IF imode THEN
317 IT5 PRINT "aus"
318 jS3 ELSE
319 YV5 PRINT "ein"
320 6z3 END IF
321 Kg2 ELSEIF e=131 THEN
322 L54 GOSUB loeschen
323 3N x=cs(AnzCsZ)
324 7Y2 ELSEIF e=134 THEN
325 bD4 RETURN
326 PR2 ELSEIF e=5 THEN
327 ZT4 FEnde=wahr
328 eG RETURN
329 F82 END IF
330 xr GOTO weiter
331 If0 insert:
332 oJ2 IF apos>x THEN
333 3o4 FOR i=apos TO x STEP -1
334 Qt6 z(i+1)=z(i)
335 q64 NEXT i
336 PH z(x)=32
337 Ny apos=apos+1
338 Kp IF apos=LZeile THEN apos=apos-1:z(LZeile)=32
339 uK FOR i=x TO apos
340 lI6 PRINT CHR$(z(i));
341 H1 IF i MOD LBZeile=59 THEN PRINT:PRINT TAB(zx);
342 xD4 NEXT i
343 yA LOCATE zy+INT(x/LBZeile),zx+(x MOD LBZeile)
344 UN2 END IF
345 vX RETURN
346 cX0 CalcSumme:
347 3s2 a=0 : b=0 : c=0
348 z2 IF e=134 THEN
349 pv4 FZok=wahr
350 WX FF6=FF6+1
351 Gz2 ELSE
352 hv4 WHILE z(apos)=32 AND apos>0
353 pI6 apos=apos-1
354 9x4 WEND
355 N1 IF apos>0 THEN
356 O36 WHILE z(c)=32

```

Listing. Der verbesserte »Checkie 42«. Bitte mit der ersten Version von »Checkie 42« oder ohne Prüfsummen und Zeilennummern eingeben.


```

357 Bt8      c=c+1
358 D16      WEND
359 Jc4      END IF
360 EJ       FOR i=c TO apos
361 Y26      j=(i-c) MOD AnzFak
362 Qs      k=(i+1-c) MOD AnzFak
363 s1      a=a+((z(i) AND 127)-32)*Faktor(j)
364 2E      b=b+((z(i) AND 127)-32)*Faktor(k)
365 Ka4      NEXT i
366 pA      cs(4)=a+Zeile-INT((a+Zeile)/62)*62
367 4N      cs(5)=b+Zeile-INT((b+Zeile)/62)*62
368 4Q      FZok=(cs(1)=cs(4)) AND (cs(2)=cs(5))
369 tm2      END IF
370 Kw      RETURN
371 pM0      Uebernahme:
372 xD2      FOR i=0 TO apos
373 ih4      PRINT#2,CHR$(z(i));
374 Tj2      NEXT i
375 VZ4      PRINT#2,""
376 5R2      Zeile=Zeile+1
377 R3      RETURN
378 N90      fertig:
379 MJ2      IF Checkfile THEN
380 Z24      WHILE NOT EOF(1)
381 6r6      LINE INPUT#1,e$
382 DE      PRINT#2,e$
383 cQ4      WEND
384 Ok      CLOSE 1
385 922      END IF
386 5q      CLOSE 2
387 nt      CLS
388 Rg      LOCATE 12,35
389 WL      PRINT "F E R T I G !!!"
390 I9      LOCATE 20,1
391 D1      IF FF6<>0 THEN
392 C14      PRINT "ACHTUNG!!! ";
393 9H      PRINT FF6;" Zeile(n) wurde(n) ungeprüft gespeichert."
394 IB2      END IF
395 jL      RETURN
(C) 1988 M&T

```

Listing. Der verbesserte »Checkie 42« (Schluß)

Inserentenverzeichnis

Alcomp	29
Bittendorf	83
Combitec	6/7
CompTec	55
Data Becker	2, 163, 164
Gigatron	97
Hagenau Computer	49
HS & Y	113
Kupke Computertechnik	17
Markt & Technik Buchverlag	14, 20/21, 24/25, 26/27, 30/31, 57
Ossowski	65
Philgerma	43
Rainbow Data	39
Ruhrsoft	19
Schmielewski	51
Software 2000	55
Soyka Datentechnik	11
Stalter, J.M.	79
Unlimited	101
Wolf, Soft- und Hardwareversand	12

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Albert Absmeier

Leitender Redakteur: Gottfried Knechtel – verantwortlich für den redaktionellen Teil

Chef vom Dienst: Susanne Kirmaier

Redaktion: Klaus Schrödl, Ralf Sablowski

Redaktionsassistent: Andrea Kaltenhauser, Brigitte Bobenstetter, Helga Weber (202)

Mitarbeiter der Redaktion: Martin Jobst, Elke Zumbusch, Christian Ewald, Rainer Kunz

Alle Artikel sind mit dem Kennzeichen des Redakteurs
(kn = Gottfried Knechtel, sk = Klaus Schrödl, rs = Ralf Sablowski)
und/oder mit dem Namen des Autors/Mitarbeiters gekennzeichnet

Art-director: Friedemann Porscha

Layout: Erich Schulze (Cheflayout), Bernd Wiehl

Fotografie: Jens Jancke, Sabine Tennstaedt,

Titelgestaltung: Friedemann Porscha, Rolf Boyke

Spritzgrafik: Ewald Standke, Norbert Raab

Computergrafik: Werner Nienstedt

Auslandsrepräsentation:

Schweiz: Markt & Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug,
Tel. 042-41 5656, Telex: 862329 mut ch

USA: M&T Publishing Inc., 501 Galveston Drive Redwood
City, CA 94063,
Telefon: (415) 366-3600, Telex 752-351

Österreich: Markt & Technik Ges. mbH
Hermann Raniger, Große Neugasse 28,
A 1040-Wien, Tel. 0043-222-8579455, Telex: 047-132532

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten worden sein, muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt & Technik Verlag AG verlegten Publikationen und dazu, daß Markt & Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Produktionsleiter: Klaus Buck (180)

Anzeigenverkaufsleitung: »Populäre Computerzeitschriften«: Alexander Narings (780)

Anzeigenleitung: Alicia Clees (313) – verantwortlich für Anzeigen

Anzeigenformate: 1/2 Seite ist 266 Millimeter hoch und 185 Millimeter breit (2 Spalten à 86 Millimeter oder 4 Spalten à 43 Millimeter). Vollformat 297 x 210 Millimeter.

Anzeigenverwaltung und Disposition: Lisa Landthaler (233)

Anzeigen-Auslandsvertretung: England: F. A. Smyth & Associates Limited, 23a, Aylmer Parade, London, N2 0PQ. Telefon: 00 44/1/340 5058, Telefax: 00 44/1/341 9602

Taiwan: Third Wave Publishing Corp., 1-4 Fl. 977 Min Shen E. Road, Taipei 10581, Taiwan, R.O.C., Tel. 0086/2/763 0052, Telefax: 0086/2/76 58 767, Telex: 0785 29 335

Vertriebsleiter: Helmut Grünfeldt (189)

Leiter Vertriebs-Marketing: Benno Gaab (740)

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1,

Bezugsmöglichkeiten: Leser-Service: Telefon (089) 46 13-366. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen.

Preis: Das Einzelheft kostet DM 16,-

Druck: SOV Graphische Betriebe, Laubanger 23, 8600 Bamberg

Urheberrecht: Alle in diesem Heft erschienenen Beiträge sind urheberrechtlich geschützt. Alle Rechte, auch Übersetzungen, vorbehalten. Reproduktionen, gleich welcher Art, ob Fotokopie, Mikrofilm oder Erfassung in Datenverarbeitungsanlagen, nur mit schriftlicher Genehmigung des Verlages. Anfragen sind an Michael Scharfenberger zu richten. Für Schaltungen, Bauanleitungen und Programme, die als Beispiele veröffentlicht werden, können wir weder Gewähr noch irgendwelche Haftung übernehmen. Aus der Veröffentlichung kann nicht geschlossen werden, daß die beschriebenen Lösungen oder verwendeten Bezeichnungen frei von gewerblichen Schutzrechten sind. Anfragen für Sonderdrucke sind an Benno Gaab (740) zu richten.

© 1988 Markt & Technik Verlag Aktiengesellschaft

Redaktion »AMIGA«

Redaktionsdirektor: Michael M. Pauly

Vorstand: Otmar Weber (Vors.), Bernd Balzer, Werner Brodt

Leiter Unternehmensbereich »Populäre Computerzeitschriften«: Michael Scharfenberger

Redaktionskoordination »Populäre Computerzeitschriften«: Hans-Günther Beer

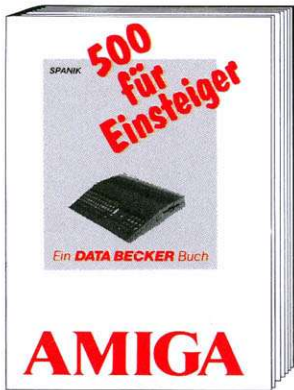
Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 46 13-0, Telex 5-22052

Telefon-Durchwahl im Verlag: Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089/46 13 und dann die Nummer, die in den Klammern hinter dem jeweiligen Namen angegeben ist.

Mitteilung gemäß Bayerischem Pressegesetz:

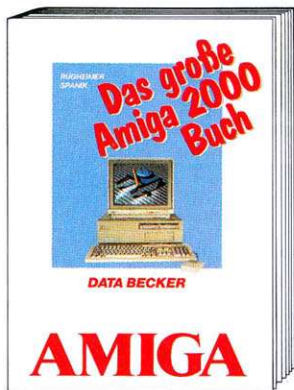
Aktionäre, die mehr als 25% des Kapitals halten: Otmar Weber, Ingenieur, München; Carl-Franz von Quadt, Betriebswirt, München; Aufsichtsrat: Carl-Franz von Quadt (Vorsitzender), Dr. Robert Dissmann (stellv. Vorsitzender), Eduard Heilmayr

AMIGA BUCHHITS



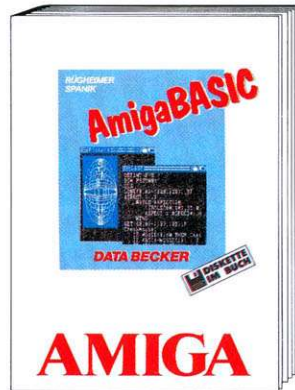
Gleich loslegen.
Bei „500 für Einsteiger“ heißt es aufstellen, anschließen und sofort loslegen. Schnell und für jedermann verständlich zeigt Ihnen dieses Buch, was Sie mit Ihrem neuen Rechner so alles anstellen können. Workbench, AmigaBASIC, CLI und DOS – schon bald wissen Sie, worauf es ankommt.

Amiga 500 für Einsteiger
343 Seiten, DM 39,-



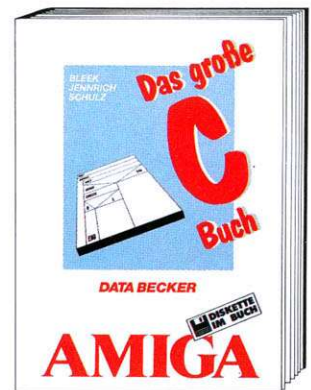
Ihr ständiger Begleiter.
Haben Sie einmal mit dem großen Amiga-2000-Buch gearbeitet, so wird es sicherlich seinen festen Platz neben Ihrem Amiga behalten. Denn dieses Buch bietet Ihnen mehr als eine detaillierte Einführung. Vom Laufwerkseinbau bis hin zum Kickstart im RAM wird hier echtes Profi-Wissen vermittelt.

Das große Amiga-2000-Buch
Hardcover, 684 Seiten
DM 59,-



Amiga BASIC komplett.
Dieses Buch zeigt auf über 770 Seiten, worauf es beim Programmieren ankommt. Natürlich mit jeder Menge interessanter Programmbeispiele, die auch gleich auf Diskette mitgeliefert werden. Ebenfalls im Buch: eine detaillierte Beschreibung des AC-BASIC-Compilers.

AmigaBASIC
Hardcover, 775 Seiten
inkl. Diskette, DM 59,-



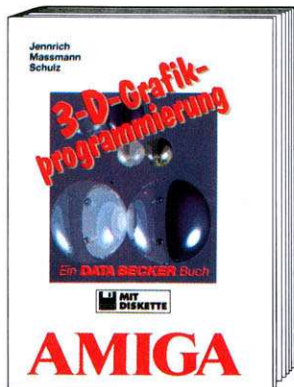
Profi-Programme in C.
Wenn Sie an C Spaß gefunden haben, gibt Ihnen das große C-Buch den letzten Schliff. Denn hier erfahren Sie nicht nur, wie ein C-Compiler arbeitet und wie Sie selbst die schwierigsten Probleme in C lösen, sondern auch, wie Sie eine optimale Benutzeroberfläche entwickeln.

Das große C-Buch
zum Amiga
Hardcover, 682 Seiten
inkl. Diskette, DM 69,-



Das Buch zum Thema Nr. 1.
Grafik auf dem Amiga – mit dem entsprechenden Know-how ist hier fast alles möglich. Dieses Buch bietet es Ihnen: Nutzung der Libraries, die Register der Grafik-Chips, Aufbau und Programmierung von Screens, Windows, Halfbrite und Interlace aus BASIC und C u.v.a.m.

Amiga Supergrafik
Hardcover, 686 Seiten
inkl. Diskette, DM 59,-



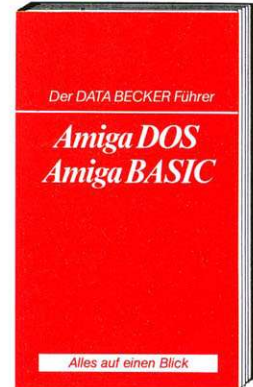
Licht und Schatten.
Dieses Buch öffnet Ihnen den Weg in eine faszinierende Welt. Anhand ausgefeilter Grafikalgorithmik erzeugen Sie phantastische Bilder: naturgetreue Spiegelungen, Licht und Schatten. Alles absolut realistisch und automatisch berechnet. In allen Auflösungen mit bis zu 4096 Farben!

Amiga 3-D-Grafikprogrammierung
Hardcover, 283 Seiten
inkl. Diskette, DM 59,-



Da steckt Musik drin.
Hier werden Sie zu einem Komponisten ausgebildet, der nicht nur die notwendigen Grundbegriffe der Musiktheorie beherrscht, sondern auch modernste Technik einzusetzen weiß:

Musikprogramme wie Sonix oder Audio Master, Sampler, MIDI-Interface und und und.
Amiga-Musikbuch
Hardcover, ca. 300 Seiten
DM 49,-
erscheint ca. 9/88



Der DATA BECKER Führer zu AmigaDOS & -BASIC
269 Seiten
DM 24,80

DATA BECKER

Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

BESTELL-COUPON
Einsenden an: DATA BECKER · Merowingerstr. 30 · 4000 Düsseldorf 1
Bitte senden Sie mir für meinen AMIGA
zzgl. DM 5,- Versandkosten
unabhängig von der bestellten Stückzahl
☐ per Nachnahme ☐ Verrechnungsscheck liegt bei
Name _____ Straße _____ Ort _____



Die Datenbank mit Profil.

DATAMAT hat ganz gehörig an Profil gewonnen: Die neue professionelle Version des beliebten Amiga-Programms läßt sich problemlos bedienen wie eine Dateiverwaltung, bietet aber die enormen Vorteile einer relationalen Datenbank.

Ein Programm mit Profil eben: Profil heißt die über 200 Befehle und Funktionen starke Interpretersprache, die das Programmieren so komfortabel macht.

Anfänger finden mit bequemen Pulldown-Menüs und bis zu acht offenen Fenstern den schnellen Einstieg in die Dateiverwaltung, fortgeschrittene Anwender nutzen die an BASIC angelehnte, strukturierte Programmiersprache. Die ist nicht nur in der leichten Erlernbarkeit ganz groß: Mit „Profil“ entstehen eigene Benutzeroberflächen so professionell wie komplexe Auswertungen. Verbindungen zwischen Dateien sind auch ohne Programmierung möglich und – besonders praktisch – eine Änderung dieser Relationen erfordert keine Reorganisation der Dateien.

Das neue Programm beweist zudem im wörtlichen Sinn Größe: Die Dateien können bis zu zwei Milliarden Zeichen umfassen. Bei zwei Milliarden stößt auch erst die Datensatz-Menge an ihre Grenze; die maximale Datensatzgröße beträgt 64.000 Zeichen. Vollkommen unbeschränkt ist die Zahl der Datenfelder (max. Feld-

größe 32.000 Zeichen). Und wer aus diesen Unmengen von Daten rasch eine bestimmte Angabe benötigt, der hat sie – bei bis zu 80 Indexfeldern – in Sekunden-schnelle zur Hand. Verlage-eigene Tests beweisen: Gesuchte Werte oder Texte können in höchstens zwei Sekunden gefunden werden, ausgewählt aus sage und schreibe 10.000 Datensätzen.

Die Vorteile von DATAMAT Professional lassen sich also leicht ausrechnen. Schwierigere Operationen dürfen Sie getrost dem Programm überlassen: Über 100 Funktionen und Konstanten stehen zur Verfügung, um Felder auch mit elaborierten Formeln zu verknüpfen. Was wieder ein schönes Stück zum neuen DATAMAT-Profil beiträgt.

Fordern Sie ganz unverbindlich unser kostenloses Info an.

DATAMAT Professional AMIGA DM 498,-

COUPON

- ☐ Bitte senden Sie mir ganz unverbindlich Ihr kostenloses Info zu
DATAMAT Professional AMIGA
☐ Hiermit bestelle ich für DM 498,- DATAMAT Professional AMIGA
☐ per Nachnahme ☐ Verrechnungsscheck liegt bei

NAME, VORNAME

STRASSE

ORT

DATA BECKER
Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010