

1

SONDERHEFT

AMIGA

Markt & Technik

DM 9,80

öS 80,-
sfr 9,80TIPS &
TRICKSMit dabei als Ratgeber:
Hermann der User
von Karl BihlmeierÜber 200 Kniffe
für Amiga-Besitzer

● Die tollsten Kniffe für Programmierer in Basic, C und Assembler ● Die besten Tips zur Workbench, zum CLI und zur Shell ● Nützliche Hilfen für Anwender: Holen Sie alles heraus aus Textverarbeitungen, Datenbanken, Grafik- und Animationsprogrammen ● Wichtige Tricks für Einsteiger; harte Nüsse für Profis; und vieles mehr zum Amiga, seiner Hardware, OS 2.0 und, und, und ...

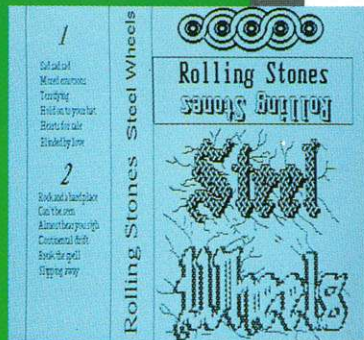
AMIGA TIPS & TRICKS

SUPER-SOFTWARE FÜR DEN AMIGA



DTP zum Sparpreis

»Multitext« verleiht Glückwunschkarten, Einladungen, Visitenkarten ect. ein ansprechendes Aussehen. Die Gestaltung der Dokumente erfolgt denkbar einfach mit Mausunterstützung. Multitext bietet die Fähigkeiten professioneller DTP-Pakete zum Sparpreis. Zum Leistungsspektrum des Programms gehören unter anderem: Ganzheitsstellung, beliebig viele Schriften, Schattieren von Objekten, Füllmuster, vielfältige Editierfunktionen. Gestalten wie die Profis mit Multitext.



Außerdem:

»C-A-O-S«: Eine faszinierende Simulation, die das Gravitationsverhalten verschiedener Planeten zueinander in Echtzeit-Animation veranschaulicht.

... und

»Real-Time«: Die voll lauffähige Demo-Version des professionellen Pakets für dreidimensionale Echtzeit-Animationen. Mit drei Beispiel-Filmen.

... und

»Tatort«: Wer bringt seine Meute zuerst in den sicheren Hafen? Ein Würfelspaß für bis zu vier Spieler und 16 Spielsteine.

... und

für Labyrinth-Fans: Viel Spaß mit "Amaze".

jetzt in

POWER
DISC

**NUR
19,80 DM**



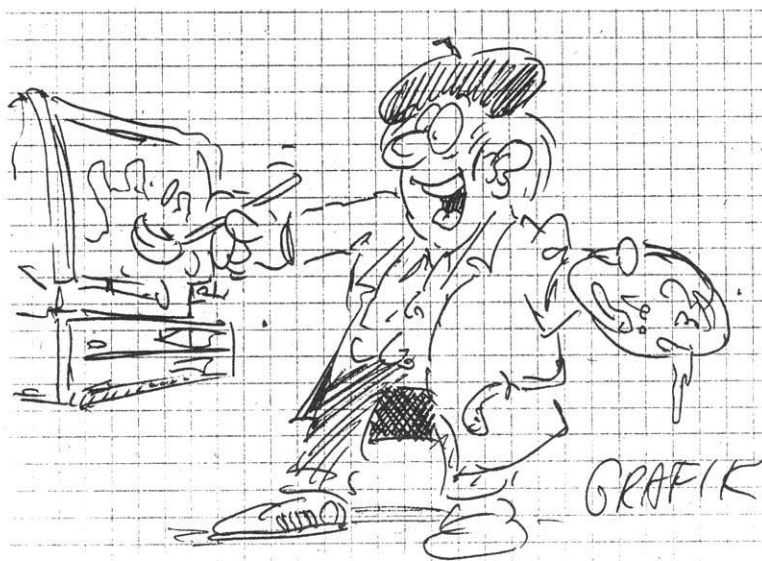
**AMIGA
POWER-DISC 8:
ab 16. Oktober
am Kiosk**

oder bestellen Sie
direkt: CSJ Markt
& Technik

Leserservice
089/ 20 25 15 28

PROGRAMMIEREN**AmigaBASIC***Damit fängt jeder an – auch das Sonderheft* **4****Gfa-BASIC***Beliebter BASIC-Dialekt für Fortgeschrittene* **19****Zaubersprache***Alles Assembler – oder was?* **24****C wie Cäsar***»Die Sprache« des Amiga* **33****Hokus Pokus***Trickkiste für alle Programmierer* **84****WORKBENCH UND CLI****Bedienung bitte!***Bekommen Sie den Amiga in den Griff* **48****3000er und OS 2.0***Erste Kniffe zum neuen Betriebssystem* **76****SOFTWARE****öffentliche Hilfe***Public-Domain-Ratgeber* **68****In der Praxis***Software-Hilfe für Anwender* **74****Schreiberlinge***Ideen zu Beckertext & Co* **80****HARDWARE****Vom Drucker bis Festplatte***Für kleine und große Bastler* **71****UND DIE SPEZIELLE »SCHATZKISTE«**

Für alle Freunde der Tips & Tricks:
 buntgemischte Seiten mit Ideen, Kniffen,
 Ratschlägen zu allem, was mit Amiga zu tun
 hat; hier ist für jeden etwas dabei;
 hier sollten Sie so richtig rumwühlen

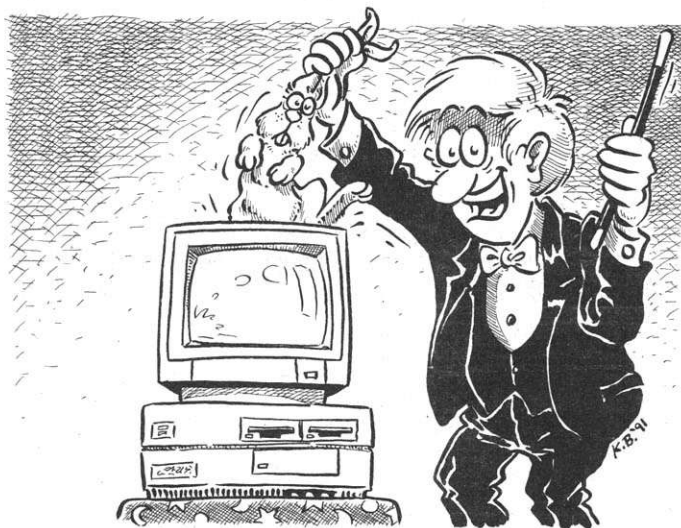
58**AMIGA-FANS AUFGEPAßT**

Ein komplettes Magazin voller Tips & Tricks zum Amiga, das hat's noch nie gegeben.

Entstanden ist die Idee zu diesem Sonderheft aufgrund der regen Beteiligung unserer Leser an der monatlichen Rubrik Tips & Tricks im AMIGA-Magazin; Sie finden in diesem Sonderheft daher größtenteils unveröffentlichte Beiträge – einige sind allerdings bereits im AMIGA-Magazin erschienen, von denen wir glauben, daß Sie im Sonderheft nicht fehlen dürfen.

Viel Spaß beim Tricksen wünscht Ihr

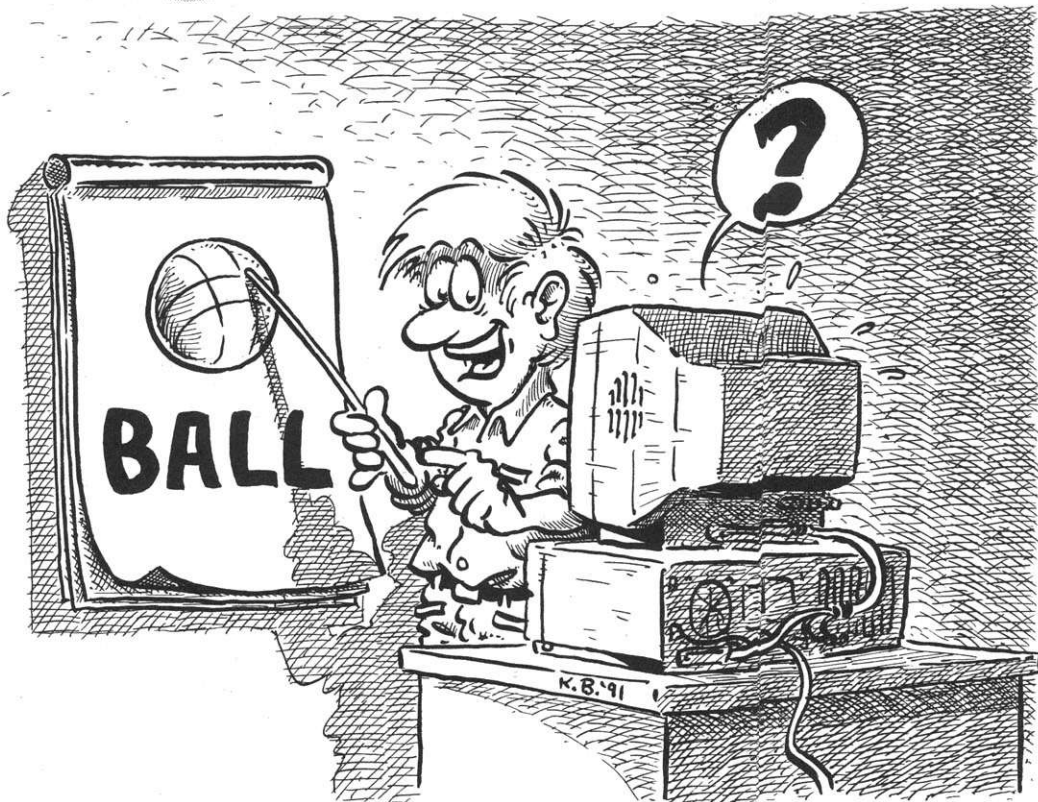
Hermann der User



Computersprachen

AmigaBASIC

Wenn man seinen Computer so richtig beherrschen möchte, muß man ihn auch programmieren. Hierzu bieten sich unterschiedliche Sprachen an: BASIC, C, Assembler usw. Klar, daß man gerade beim Programmieren Tausende von Kniffen anwenden kann, um noch mehr aus dem Amiga herauszuholen, sei es, um Programme zu beschleunigen, Speicherplatz zu sparen oder um Programme komfortabler zu machen.



Beginnen wir den Reigen der Programmiertips mit BASIC, genauer AmigaBASIC, eine Sprache, die gerade für Einsteiger optimal geeignet ist. Es folgt Gfa-BASIC, ein BASIC-Dialekt, der sich wachsender Beliebtheit unter Programmierern erfreut. Vor allem, wenn man den Gfa-Compiler einsetzt, der BASIC-Listings in schnelle, eigenständige Programme übersetzt.

Mit C und Assembler folgen dann Tips zu Sprachen, die unter fortgeschrittenen Amiga-Programmierern am gefragtesten sind. Wer die Sprachen kennenlernen möchte, sollte sich einen der C-Compiler auf Public Domain besorgen, gleiches gilt für Assembler, z.B.:

- Sozobon-C, C-Compiler, Fish-Disk 508;
- DICE, C-Compiler, Fish-Disk 491;
- A68k, Assembler, Fish-Disk 508;
- OMA, Assembler, Programmservice-Disketten zum AMIGA-Magazin 11/91 und 12/91.

Und schließlich bringen wir noch Tips zu Modula-2 und einige allgemeine Programmiertips, die für alle Sprachen gültig sind.

Der Screen ist weg

Wenn dem Amiga was nicht paßt, pfeift er seinen Unmut durch ein Kommunikationsfenster (Requester) links oben auf dem Bildschirm auszudrücken. Der Computer bringt hierzu die Workbench oder den Standard-Bildschirm (Screen) nach vorne, in dem sich das Ausgabe- und List-Fenster befinden. Klicken Sie »Cancel« an, verschwindet der Requester, aber der Screen bleibt im Vordergrund. Um auf Ihren Screen zurückzukommen, ziehen Sie die BASIC-Fen-

ster herunter, bis die Titelleiste des Workbench-Screens sichtbar wird. Danach klicken Sie mit der linken Maus-Taste das Tiefen-Gadget rechts oben an. Jetzt sollte Ihr Screen wieder sichtbar sein.

Dasselbe funktioniert über Tastenkombinationen:

- <Amiga_links m> bringt die Workbench in den Hintergrund und
- <Amiga_links n> holt sie wieder nach vorne.

ub

Wie tief ist der Screen?

AmigaBASIC besitzt keine Anweisung, um die Zahl der Bitplanes für die Farbdarstellung bzw. die Tiefe eines Screens festzustellen. Doch gibt es hierzu zwei Verfahren:

- Beim ersten brauchen wir die Zahl der Farben: WINDOW(6) liefert den um 1 verringerten Wert. Dessen Zweierpotenz ist die gesuchte Tiefe. Wir ermitteln Sie durch:

$\text{bitplanes\%} = \text{LOG}(\text{WINDOW}(6)+1)/\text{LOG}(2)$

- Der Befehl

`PRINT WINDOW(8)`

liefert die Adresse einer Datenstruktur, die wiederum die Adresse einer weiteren Struktur enthält. Deren Adresse holen wir uns mit:

`PEEKL(WINDOW(8)+4)`

Im fünften Byte dieser Struktur steht der gewünschte Wert. Man kann ihn ganz einfach mit der Zuweisung

$\text{bitplanes\%} = \text{PEEK}(\text{PEEKL}(\text{WINDOW}(8)+4)+5)$

auf einen Schlag aus dem Speicher lesen.

irw

Basic mag keine Umlaute

AmigaBASIC verträgt keine Umlaute im Variablennamen. Wenn Sie es trotzdem versuchen, gibt das einen »Syntax error«. Außerdem löscht der Interpreter kleine »ü« in Kommentarzeilen. Aus »REM ÄäÖöÜüß« wird »REM ÄäÖöÜß«. Umlaute in Texten, also zwischen Anführungszeichen, machen keine Probleme. Am Bildschirm sind jedoch das große und kleine »ü« nur schwer zu unterscheiden, weil sie fast gleich aussehen.

Die Konsequenz kann da nur sein, die Umlaute »Ä, Ö, Ü, ä, ö, ü« immer mit »Ae, Oe, Ue, ae, oe, ue« zu ersetzen.

ub

INPUT ohne Fragezeichen

INPUT gibt ab der aktuellen Cursor-Position den hinter dem Befehl angegebenen Text sowie ein Fragezeichen aus und erwartet dann die Tastatureingabe. Beispiel:

```
INPUT "Bitte Zahl eingeben";z
```

Wenn Sie das Fragezeichen stört, können Sie es unterdrücken, indem Sie den Strichpunkt durch ein Komma ersetzen:

```
INPUT "Bitte Zahl eingeben",z
```

Das funktioniert aber nur, wenn Sie hinter INPUT eine Zeichenkette angeben.

```
INPUT "",z
```

Wie im Beispiel gezeigt, darf es auch ein Leerstring sein.

irw

UNDO holt die Zeile wieder

Viele Programme haben eine Undo-Funktion, die mißlungene Operationen rückgängig macht. AmigaBASIC auch. Wenn Sie Änderungen in einer Zeile nicht übernehmen wollen, drücken Sie <Amiga_rechts >, ohne die Zeile vorher zu verlassen. Sofort erscheint die vorherige Zeile wieder am Bildschirm.

irw

Müllabfuhr für BASIC

Wenn Sie ein Programm mit AmigaBASIC editieren, zwischendurch testen und dann auf Diskette speichern, wird es wesentlich länger sein als Vorversionen mit denselben Anweisungen. Durch das Editieren und das Testen ist immer mehr Variablen- und Datenmüll entstanden. Beim Speichern eines Programms werden nicht nur die gültigen Variablen, sondern auch der Müll gespeichert. Er kann beseitigt werden. Geben Sie im Ausgabefenster des Interpreters

```
load "Programmname"
```

ein, drücken Sie <Return> und speichern Sie das Programm mit

```
save "Programmname.asc",a
```

Nach dem Tipp auf <Return> speichert BASIC das Programm als Textdatei ohne den Variablenmüll. An den Programmnamen haben wir ».asc« gehängt, damit das Original nicht überschrieben wird.

Zur Erfolgskontrolle können Sie ein SHELL-Fenster öffnen und die Länge beider Versionen mit dem LIST-Befehl vergleichen. Sie können das Ganze sogar noch weiter optimieren:

```
save "Programmname.bin",b
```

speichert das Programm in binärer Form. Sie sollten nun die Länge der drei Versionen prüfen und die kürzeste behalten.

ub

Grad in das Bogenmaß

Die trigonometrischen Funktionen von AmigaBASIC benötigen die Winkelangaben im Bogenmaß (Einheit: rad). Die Umrechnung ist einfach:

```
rad = (pi * grad)/180
```

Und wieder zurück:

```
grad = (rad * 180)/pi
```

Beachten Sie, daß Winkel gegen den Uhrzeigersinn gemessen werden. Wenn Sie sich die Definition von »Pi« erleichtern wollen, lesen Sie den Tip »Pi und Verwandte« (Seite 8).

ub

Out of Heap Space

AmigaBASIC liefert diese Fehlermeldung bei einem Überlauf des Editor-Zwischenspeichers. Er tritt ein, wenn man versucht, sehr große Textblöcke mit Cut zu löschen. »Out of Heap Space« signalisiert auch, daß der für BASIC reservierte Stapelspeicher (Stack) nicht ausreicht; z.B. wegen zu vieler geschachtelter Schleifen oder GOSUB-Aufrufe. In so einem Fall sollten Sie mit der Anweisung CLEAR mehr Stapelspeicher zuweisen.

CLEAR dient dazu, Variablen zu löschen und offene Dateien zu schließen. Mit CLEAR ändern Sie aber auch die Speicherreservierung und -einteilung von BASIC. Die Mindestgröße des Arbeitsspeichers ist 1 KByte. Nach oben ist die Grenze der verfügbare Systempeicher. Die Voreinstellung für den Speicher schluckt 25000 Byte.

Der Interpreter benötigt auch einen Stack. Diesen Speicherbereich verwendet er für interne Zwischenspeicherungen und die Rückkehradressen der Unterprogrammaufrufe. Der Stack ist auf 4789 Byte voreingestellt. Die aktuellen Speichergrößen ermitteln Sie mit der Funktion FRE() (siehe BASIC-Handbuch).

irw

TIPS ZU AMIGABASIC AUF EINEN BLICK

Tip/Thema	Seite
Der Screen ist weg / Screens per Tastendruck umstellen	4
Wie tief ist der Screen / Farbanzahl bestimmen	4
BASIC mag keine Umlaute / Keine »ü's« in BASIC-Programmen	5
INPUT ohne Fragezeichen / INPUT-Befehl	5
Undo holt die Zeile wieder / Bedienung	5
Müllabfuhr in BASIC / Programme entschlacken	5
Grad in Bogenmaß / Umrechnung	5
Out of Heap Space / CLEAR-Befehl	5
Maus oder Tastatur / Menüprogrammierung	6
LED an / Laufwerkslampe anschalten	6
Genauere Wurzel-Behandlung / Quadratwurzel berechnen	6
Saubere Sache / Laufwerksteuerung	7
Existenzfragen / Dateien prüfen	7
Wo ist der Apostroph / Sonderzeichen (REM)	7
»Pi« und Verwandte / Definitionen	8
Farbige Titel / Titelleiste einfärben	8
Verschieden Schriftarten / Schreibstil ändern mit POKE	8
Unsichtbarer Syntax Error / Störende Leerzeichen in Programmen	8
Trigonometrie auf dem Kopf / SIN, COS usw.	8
Fehler im BASIC-Handbuch / STRIG-Funktion	8
Object in use / Druckersteuerung	8
Texte, die den Punkt treffen / Cursorplatzierung	9
BASIC lückenlos / Leerzeichen entfernen	9
BASIC sortiert / Texte ändern per Programm	10
Line buffer overflow / Listeingabe	10
Funktionstasten nutzen	10
Error while opening / Default-Tool	11
Zu wenig Speicher? / Speicher sparen	11
Zeichnen mit System / Grafik-Bibliothek nutzen	11
So liegen Sie richtig / Ausgabefenster selektieren	11
Patch: BASIC ohne Bilder / Interpreter verändern	12
GROSSES List-Fenster / List-Fenster in PAL (256 Zeilen)	12
ASCII-Codes / Programm gibt ASCII-Tabelle aus	12
CUT & COPY einfacher / Bedienungshilfe	12
Find & Replace / Editor statt BASIC-List-Fenster	13
So liegen Sie richtig / Requester umlenken	13
Maus in Motion / Sprites steuern	13
Jetzt kommt Bewegung ins Spiel / Overscan	14
Schriftzug fährt ab / Textscrolling	14
BASIC schlägt Alarm / Alerts	14
Teile von BASIC speichern / Cut-Funktion	14
Bildausschnitt / Bilder ausschneiden	14
Anderer Zeichensatz / Font nachladen	15
Drucker testen / Anschluß überprüfen per BASIC-Befehl	16
Wieviel Speicher / AVAIL	16
Henon-Attraktor / Funktion plotten	16
Workbench auf und zu / System-Befehl nutzen	17

Menüs: Maus oder Tastatur

Wenn es darum geht, einen Menüpunkt zu selektieren, nehmen viele lieber die Maus, während andere die Tastatur bevorzugen, um ihren Amiga zu bedienen. Damit Ihre BASIC-Programme benutzerfreundlich sind, sollten Sie beide Varianten zulassen.

Das ist ganz einfach zu bewerkstelligen: Im Menü-Zeilentext, der das Aussehen der Menüs im Programm festlegt, geben Sie einfach am Ende noch einen »Ctrl«-Code (^) mit ein, z.B.: »^D«, »^E«,... Menü-Zeilen sehen dann etwa so aus:

```
MENU 1,1,1,"Drucken ^D"
MENU 1,2,1,"Zeigen ^Z"
```

Nun müssen Sie im Programm parallel zur Menüabfrage, die per ON MENU-Befehl läuft, mit INKEY\$ die Tastatur überwachen und testen, ob der Anwender eine der in den Menüs erwähnten Kombinationen drückt. Die CHR\$-Codes für die Ctrl-Kombinationen sind:

```
^A = CHR$ ( 1 )
^B = CHR$ ( 2 )
^C = CHR$ ( 3 )
...
^Y = CHR$ ( 25 )
^Z = CHR$ ( 26 )
```

Viel Spaß beim benutzerfreundlichen Programmieren.

Daniel Flück/ub

LED an

Mit dem Programm »LED.bas« schalten Sie die Laufwerkslampe (Leuchtdiode, LED) an und aus. Zum Anschalten müssen Sie nur in der letzten Zeile des Programms die Zahl 0 mit 255 vertauschen.

ub

```
LED = 12570880&
POKE LED,127
POKE LED,119
POKE LED+512,0
```

LED.bas

Zum Ein- und Ausschalten der Laufwerks-LED

Genaue Wurzel-Behandlung

»Quadrat.bas« berechnet die Quadratwurzel von ganzen Zahlen im Bereich von 1 bis 32 761 auf theoretisch bis zu über 16 000 Nachkommastellen genau. Für das Näherungsverfahren verwendet das Programm einen Algorithmus, der besonders schnell ist, vielleicht sogar der schnellste. Oder kennen Sie einen schnelleren?

```
start:
CLEAR ,50000&
PRINT "Universelle Quadratwurzelberechnung"
INPUT "Quadratwurzel welcher ganzen Zahl "; zahl
IF zahl < 1 OR zahl > 32761 OR zahl > INT(zahl) THEN
PRINT "Zahl muß zwischen 1 und 32761 liegen!" : PRINT
GOTO start
END IF
zahl% = zahl
IF INT(SQR(zahl%)) = SQR(zahl%) THEN
PRINT "Quadratwurzel von 'zahl%' ist exakt 'INT(SQR(zahl%))' !"
pr% = 0 : GOTO ende
END IF
INPUT "Wieviele Nachkommastellen sollen berechnet werden "; bis%
INPUT "1 = Bildschirmausgabe, 2 = Druckerausgabe, 3 = beides "; pr%
IF bis% < 2 OR pr% < 1 OR pr% > 3 THEN start
DIM w%(bis%+1), e%(2*(bis%+1))
w%(1) = INT(SQR(zahl%)) : e%(1) = w%(1)*w%(1)
IF pr% = 2 OR pr% = 3 THEN
OPEN "lpt1:" FOR OUTPUT AS #2
PRINT#2, "Die Quadratwurzel von 'zahl%' ist :"
```

```
END IF
IF pr% = 1 OR pr% = 3 THEN CLS :PRINT "Quadratwurzel von 'zahl%' : "
a$ = RIGHT$(STR$(w%(1)), LEN(STR$(w%(1)))-1)+", "
y% = 2 : l% = 2 : overall% = TIMER : GOTO raise
intervall:
a$ = a$+RIGHT$(STR$(w%(l%)),1)
IF pr% = 1 OR pr% = 3 THEN LOCATE y%,1 : PRINT a$
IF (pr% = 1 OR pr% = 3) AND LEN(a$) = 70 THEN
y% = y%+1
IF y% > 21 THEN INPUT "Return",x$ : CLS : y% = 1
END IF
IF (pr% = 2 OR pr% = 3) AND LEN(a$) = 70 THEN PRINT#2,a$
IF LEN(a$) = 70 THEN a$ = ""
l% = l%+1 : IF l% = bis%+2 THEN ende
raise:
w%(l%) = w%(l%)+1 : last% = l%-1
FOR a% = 1 TO l%-1
h% = l%+a%-1 : e%(h%) = e%(h%)+w%(a%)+w%(a%)
NEXT
e%(last%) = e%(last%)+2*w%(l%)-1
FOR u% = last% TO 2 STEP -1
IF e%(u%)>9 THEN
e%(u%-1) = e%(u%-1)+INT(e%(u%)/10)
e%(u%) = e%(u%) MOD 10
END IF
NEXT
IF e%(1) >= zahl% THEN found
IF w%(l%) = 9 THEN intervall
GOTO raise
found:
w%(l%) = w%(l%)-1
e%(last%) = e%(last%)-2*w%(l%)-1
FOR a% = 1 TO l%-1
h% = l%+a%-1 : e%(h%) = e%(h%)-w%(a%)-w%(a%)
NEXT
FOR u% = last% TO 2 STEP -1
IF e%(u%) < 0 THEN
h% = -INT(e%(u%)/10) : e%(u%) = e%(u%)+10*h%
e%(u%-1) = e%(u%-1)-h%
END IF
NEXT
GOTO intervall
ende:
IF pr% = 1 OR pr% = 3 THEN LOCATE y%,1 : PRINT a$
IF pr% = 2 OR pr% = 3 THEN PRINT#2,a$
IF pr% = 1 OR pr% = 3 THEN
PRINT FIX(TIMER-overall%)" Sekunde(n) Rechenzeit"
END IF
IF pr% = 2 OR pr% = 3 THEN
PRINT#2, FIX(TIMER-overall%)" Sekunde(n) Rechenzeit" : PRINT#2, ""
END IF
IF pr% = 2 OR pr% = 3 THEN CLOSE #2
INPUT "Noch eine weitere Berechnung (j/n) "; n$
IF n$="j" THEN PRINT : GOTO start
END
```

Wurzel.bas

Auf wieviel Stellen genau hätten's denn gern?

Das Programm läuft auf allen Amigas; es ist keine besondere Ausrüstung erforderlich. Zur Druckerausgabe brauchen Sie natürlich einen Drucker, Sie können die errechneten Stellen jedoch auch auf dem Bildschirm ausgeben lassen, oder beides parallel.

Einige Hinweise zum Prinzip des Programms: Zuerst legt »Quadrat.bas« ein Feld für die angenäherte Wurzel und eins für das angenäherte Quadrat an, in dem jeder Stelle der Zahl ein Element des Felds entspricht. Zu Beginn stehen alle Stellen auf Null.

Wenn man die irrationale Wurzel einer Zahl berechnen will, benutzt man eine Intervallschachtelung. Man geht im Prinzip so vor:

Man beginnt ganz links, mit der ersten Stelle. Dann erhöht man die Stelle um eins, bildet das Quadrat, der sich so ergebenden Dezimalzahl, und prüft, ob es größer als der vorgegebene Wurzelradikand ist; in diesem Fall, erniedrigt man die aktuelle Stelle wieder um eins und geht eine Stelle weiter nach rechts (und weiter wie gehabt). Im anderen Fall, und wenn die Stelle schon neun ist, kann die aktuelle Stelle nur den Wert neun besitzen, also geht man eine Stelle weiter nach rechts (weiter wie gehabt).

Ist das Quadrat noch zu klein, die aktuelle Stelle aber nicht neun, erhöht man die Stelle wieder um eins und macht weiter wie vorher.

So kann man recht einfach die Wurzel einer Zahl berechnen, jedoch ist die Methode langsam, weil man zu oft das Quadrat einer Zahl berechnen muß, die zu allem Übel ständig länger wird. Folglich liegt es nahe, an diesem Punkt eine Verbesserung anzusetzen:

Wenn man die Multiplikation einer Zahl mit sich selbst untersucht, stellt man fest, daß man in unserem Fall, in dem sich immer nur die letzte Stelle um eins erhöht, mit dem Ergebnis der letzten Quadrierung weiterarbeiten kann und nicht jede Stelle mit jeder Stelle multiplizieren muß.

Man muß bei jedem Durchlauf, in dem sich die letzte Stelle um eins erhöht, zum letzten Quadrat immer nur jede Stelle außer der letzten zweimal an die Position (Länge der Zahl + Stelle Nr. - 1) addieren, sowie daß Quadrat der letzten Stelle an die Stelle (2 * Länge der Zahl - 1) setzen und dabei alle Überträge berücksichtigen.

Ein vom Zeitaufwand optimales Verfahren. Bei der Subtraktion von eins an der aktuellen Stelle, wenn man die richtige Stelle gefunden hat, geht man analog vor, indem man die genannten Werte nicht addiert, sondern subtrahiert. Lediglich die Routine zur Übertragsberücksichtigung muß verändert werden.

Andrea Bomke/ub

Saubere Sache

Zum Reinigen von Diskettenlaufwerken sind die im Handel angebotenen Reinigungsdisketten ideal. Das Problem dabei ist, daß diese Disketten vom Betriebssystem nicht anerkannt werden und beim Einlegen nur die Warnung erscheint: »Not a Dos-Disk...«.

Um eine Reinigungsdiskette trotzdem zu nutzen, muß man das Laufwerk per Software anschalten. Das folgende BASIC-Programm erfüllt genau diesen Zweck.

```
SUB diskclean(z,l) STATIC
  IF l < 0 OR l > 3 THEN
    BEEP : PRINT "Laufwerksnummer falsch"
    EXIT SUB
  END IF
  PRINT "Bitte linke Maustaste drücken!"
  SLEEP
  WHILE MOUSE(0) = 0 : WEND
  IF l = 3 THEN l = 7
  IF l = 2 THEN l = 3
  df = 119-8*l
  prb = 12570880&
  ddrb = prb+512
  POKE prb,127
  POKE prb,df, POKE ddrb,0
  FOR sek = 1 TO z
    LOCATE 2,1 : PRINT "Sekunden:"; sek
    t = INT(TIMER)
    loop:
      IF INT(TIMER) = t THEN loop
    NEXT sek
    POKE ddrb,255
  END SUB
```

Diskclean.bas

Steuert den Motor des Laufwerks, wenn Sie z.B. eine Reinigungsdiskette einlegen

Es handelt sich um eine erweiterte Version des Programms in den Tips & Tricks des AMIGA-Magazins Ausgabe 11/90. Die Routine benötigt folgende Parameter:

- »zeit« – Zeit in Sekunden, nach der das Laufwerk wieder ausgeschaltet wird
- »laufwerk« – Laufwerksnummer (0 = internes Laufwerk usw.)

Ein gültiger Aufruf sähe z.B. so aus:

```
CALL diskclean(20,0,0)
```

Damit würde sich das interne Laufwerk für 20 Sekunden drehen.

Alexander Scholz/ub

Existenzfragen

Das folgende Unterprogramm »Exists.bas« ist für alle AmigaBASIC-Programmierer gedacht, die in ihrem Programm überprüfen wollen, ob eine bestimmte Datei existiert, ohne dabei die Datei auf Diskette einzurichten, falls es sie noch nicht gibt.

```
REM Programm kontrolliert, ob Befehl TYPE in C: existiert
```

```
LIBRARY "dos.library"
DECLARE FUNCTION Lock& LIBRARY

nam$ = "c:type"
nam$ = nam$ + CHR$(0)
Modus$ = "-2"
Exists& = Lock&(SADD(nam$), Modus&)
UnLock& Exists&
IF Exists& = 0 THEN
  PRINT "Datei nicht vorhanden"
END IF
LIBRARY CLOSE
END
```

Exists.bas

Sein oder nicht sein? Antwort in AmigaBASIC

Die Routine ist verglichen mit anderen, die dieselbe Aufgabe erfüllen, besonders schnell, da sie nur Funktionen aus der DOS-Library verwendet. Deshalb ist das Öffnen der DOS-Library und Deklarieren der Funktion »Lock&« Voraussetzung. Wichtig: Ihr Amiga benötigt die Datei »dos.bmap« im Verzeichnis »libs:« oder im Verzeichnis, in dem »Exists.bas« steht, um das Programm auszuführen.

Das Beispielprogramm überprüft die Existenz des Befehls TYPE im Verzeichnis »C:«. Die Routine funktioniert wie folgt:

Zuerst hängt der Amiga dem Namen der zu überprüfenden Datei ein Nullbyte an, danach kommt das Herzstück der Routine: die Funktion »Lock&«. Ihr übergeben wir als ersten Parameter die Adresse des Namens des Files oder des Directories, dessen Existenz wir überprüfen möchten (mit : SADD (Name\$)). Als zweiter Parameter ist der Modus gefragt, sprich »-1«, wenn in die Datei geschrieben werden soll, oder »-2«, wenn aus ihr gelesen werden soll.

Als Ergebnis bekommen wir von »Lock&« einen Zeiger auf eine FileLock-Struktur, die von einigen weiteren Library-Funktionen aus der DOS-Library benötigt wird, um Informationen über eine Datei zu gewinnen. Wenn das Ergebnis des Aufrufs von »Lock&« Null lautet, existiert die Datei oder das Directory nicht.

Wir geben als Modus »-2« an, da einige Dateien nur gelesen werden können (Schreibschutz oder protectet). Falls wir als Modus »-1« angeben und die Existenz einer solchen Datei überprüfen, erhalten wir ein Resultat, als ob die Datei nicht vorhanden sei.

Abschließend müssen wir mit der Funktion »UnLock&« den Speicher für die FileLock-Struktur wieder freigeben. Hierzu übergeben Sie der Funktion »UnLock&« den Zeiger auf FileLock&, den Sie zuvor erhalten haben. Ein Aufruf »UnLock& 0« bleibt wirkungslos.

Falls Sie den »UnLock&«-Befehl vergessen, können Sie auf die überprüfte Datei oder das Directory nicht mehr z.B. vom CLI aus zugreifen; Sie erhielten die Fehlermeldung: »Object in Use!«

Mit »Exists.bas« können Sie auch die Existenz eines Directories überprüfen, indem Sie als Namen z.B. »Name\$="df0:libs"« angeben.

Dirk von Diringshofen/ub

Wo ist der Apostroph?

Im AmigaBASIC-Handbuch steht, daß man für Kommentare statt REM auch ein Apostroph (»'«) verwenden kann. Auf der deutschen Amiga-Tastatur sind gleich drei davon. Welcher ist der richtige? Keiner, denn der Apostroph, den Sie für REM verwenden, erreichen Sie nur mit der Tastenkombination <Alt >.

ub

»Pi« und Verwandte

- Die Entwickler von AmigaBASIC haben keine Konstanten für »Pi« und »e« implementiert. So müssen Sie für »Pi« die Zahl »3.141593« oder bei doppelter Genauigkeit »3.141592653589794« selbst eintippen. Die Arbeit können Sie sich sparen – geben Sie stattdessen ein:

```
Pi = 4*ATN(1)
```

bzw.

```
Pi# = 4#*ATN(1#)
```

für 7- und 15-stellige Genauigkeit ein.

- Die LOG-Funktion berechnet den natürlichen Logarithmus einer Zahl und der ist nur für positive Zahlen definiert. Der dekadische Logarithmus der Zahl n kann mit der Formel

```
lg10 = LOG(n)/LOG(10)
```

berechnet werden.

Der natürliche Logarithmus von »e« ist 1. Für »e« können Sie daher die Eingabe von 2.718282 sparen, denn EXP(1) und EXP(1#) ergeben denselben Wert einfacher und doppelter Genauigkeit.

ub

Farbige Titel in BASIC

Wußten Sie, daß man die Titelleiste eines Amiga-BASIC Fensters einfärben kann (Das Menü wird auch farbig)? Das nächste Unterprogramm ändert die Vorder- und Hintergrundfarbe der Leiste:

```
SUB Farbe(x%,y%) STATIC
```

```
POKE WINDOW(7)+98,x%
```

```
POKE WINDOW(7)+99,y%
```

```
END SUB
```

Der Aufruf erfolgt folgendermaßen:

```
Farbe(Vordergrundfarbe%,Hintergrundfarbe%)
```

Probieren Sie es doch einfach einmal in einem Ihrer Programme aus. Beachten Sie, daß die Einfärbung erst dann erfolgt,

- wenn Sie die rechte Maustaste drücken, oder
- das Fenster deaktivieren und wieder aktivieren, oder
- die Fenstergröße mit Hilfe des Größengadgets verändern.

Norbert Tausch/ub

Verschiedene Schriftarten

In BASIC besteht die Möglichkeit, einen Text in unterschiedlichen Stilarten darzustellen:

- normal
- fett
- kursiv
- unterstrichen
- invertiert

Zur Wahl einer Schriftart muß man ins Byte an der Adresse

```
Rastport des BASIC-Windows +56
```

eine Kennziffer schreiben. Die Kennziffer kann sein:

- 0 für Normalschrift
- 1 für unterstrichen
- 2 für fett
- 4 für kursiv
- 8 für Breitschrift

Es sind auch Kombinationen möglich, z.B. kann man einen Text kursiv und unterstrichen darstellen; die Auswahl erfolgt über die logische UND-Verknüpfung der jeweiligen Kennziffern. Mit einem SUB-Programm, das alle Arbeiten bequem erledigt, macht man sich die Sache einfach:

```
SUB SchreibStil(Stil%) STATIC
```

```
POKE WINDOW(8)+56,Stil%
```

```
END SUB
```

WINDOW(8) liefert den Zeiger auf den Rastport des aktuellen Fensters. Der Rastport ist eine Datenstruktur in der wichtige Daten zu einem Fenster für die Zeichenfunktionen der Grafik-Bibliothek festgehalten sind, z.B. die Position der Schreibmarke (Cursor), die Farbe zum Zeichnen, ein Schriftstil usw. Unser Unterprogramm

schreibt die übergebene Kennziffer (sie muß mit »&« enden) an die gewünschte Adresse. Der Aufruf erfolgt mit:

```
SchreibStil(Kennziffer&)
```

Setzen Sie das Unterprogramm ein, um Textausgaben in Ihren BASIC-Programmen noch attraktiver zu machen.

Norbert Tausch/ub

Unsichtbarer Syntax error

Kein Fehler im Listing zu sehen und trotzdem bricht das Programm mit »Syntax error« ab. Tja, BASIC verträgt keine Leerzeichen hinter einem END IF. Die Abhilfe ist einfach: Cursor an's Ende der Anweisung END IF positionieren und <Return> drücken.

irw

Trigonometrie auf dem Kopf

Oft benötigt man die Umkehrfunktionen der trigonometrischen Funktionen SIN oder COS. Man übergibt ihnen z.B. den Sinus eines Winkels und erhält den Winkel. Die Umkehrfunktionen kennzeichnen Mathematiker mit dem Präfix »Arcus«. Für den Sinus heißt sie also Arcussinus. Jeder bessere Taschenrechner beherrscht das. In AmigaBASIC gibt es leider nur die Arcustangens-Funktion. Die übrigen Arcus-Funktionen leiten wir aus der ATN-Funktion ab:

```
Arcussinus = ATN(w/SQR(1-x^2))
```

```
Arcuscosinus = Pi/2 - Arcussinus
```

Beachten Sie, daß der Winkel im Bogenmaß angegeben werden muß (siehe Tip »Grad in Bogenmaß«, Seite 5, oben).

ub

Fehler im BASIC-Handbuch

Im BASIC-Handbuch sind Rückgabewerte der STRIG-Funktion vertauscht:

STRIG(n)

1

-1

Richtung

nach unten oder rechts

nach oben oder links

Außerdem fehlt die Angabe, daß die STRIG-Funktion bei gedrücktem Feuerknopf den Rückgabewert -1 liefert.

irw

Object in use – File already open

Der BASIC-Befehl LPRINT gibt Daten an den Drucker aus. Der Interpreter öffnet dafür einen Übertragungskanal zum Drucker, und solange er offen ist, kann kein anderes Programm – kein anderer Task – auf den Drucker zugreifen. Noch schlimmer: Wenn Sie nach LPRINT AmigaBASIC beenden, kann niemand mehr den Kanal schließen. Selbst von der Shell ist der Zugriff auf den Drucker unmöglich; nur die Notbremse <Ctrl Amiga_links Amiga_rechts> hilft.

Der gute Rat: Öffnen und schließen Sie den Übertragungskanal selbst (»PRT:« ist der Gerätenamen des Druckers):

```
OPEN "PRT:" FOR OUTPUT AS #1
```

```
PRINT #1,"Test"
```

```
CLOSE #1
```

Die Ausgabedatei wird korrekt geschlossen, während das folgende Kommando den Kanal zum Drucker offenläßt.

```
LPRINT "Test"
```

Sobald Sie versuchen, während desselben Programmablaufs nach LPRINT mit »PAR:« oder »PRT:« einen Übertragungskanal zu öffnen, unterbricht BASIC das Programm mit der Fehlermeldung:

```
File already open
```

CLEAR und END schließen den mit LPRINT geöffneten Kanal:

- Der Befehl CLEAR löscht allerdings gleichzeitig alle Variablen und schließt die offenen Dateien; es ist deshalb innerhalb eines Programms meist unbrauchbar.

- Mit END wird ein Programm beendet. Der Befehl ist also nur am Programmende sinnvoll. Hier hilft nur ein Trick:

```
LPRINT "TEST1"
LPRINT "TEST2"
REM und so weiter...
OPEN "PRT:" FOR OUTPUT AS #1
CLOSE #1
```

Gewöhnen Sie sich an, jedes BASIC-Programm mit END zu beenden, und Sie werden mit LPRINT keine Probleme mehr haben.

Hans Kasulske/ub

Texte, die den Punkt treffen

Jeder BASIC-Programmierer, der ein bißchen Grafik in seine Programme gebracht hat, wird sich sicherlich schon geärgert haben, daß man Text nicht genau im Fenster positionieren kann, der Befehl LOCATE arbeitet nämlich nur zeilen- und spaltenweise.

Um einen Text an eine genaue Position zu bringen, gibt es in der »graphics.library« den Befehl:

```
Move& Rastport&,x&,y&
```

Er plaziert im aktuellen Fenster, dessen Rastport-Struktur wir mir WINDOW(8) ansprechen, den Text- und Grafikkursor an der Position mit den Koordinaten x&,y&.

Da es sich bei »Move« um eine Library-Funktion handelt, benötigt Ihr Amiga die »bmap.«-Datei der »graphics.library«, die Sie am Anfang Ihres Programms öffnen müssen. Das Ganze geht natürlich einfacher durch entsprechende POKE-Befehle, wie im folgenden Unterprogramm gezeigt:

```
SUB TextXY(X&,Y&) STATIC
POKEW WINDOW(8)+36,X&
POKEW WINDOW(8)+38,Y&
END SUB
```

Der Aufruf erfolgt durch:

```
TextXY(X&,y&)
```

Man kann also einen Text auch ohne Library genau positionieren.

Jens Freisen/ub

BASIC lückenlos

Das Programm »Blank-Test.bas« ist ein schönes Beispiel, wie man eine ASCII-Datei per BASIC-Programm ändert. »Blank-Test.bas« löscht Leerzeichen (Blanks) am Ende von Textzeilen. Solche Blanks stören besonders in BASIC-Programmen, da sie den AmigaBASIC-Interpreter zum Absturz bringen können.

```
REM Blank-Test.bas
REM Da AmigaBASIC bei Leerzeichen hinter Programmzeilen
REM oft aussteigt, kann man sie mit Blank-Test beseitigen.
ON BREAK GOSUB Break
BREAK ON
WINDOW 2,"BLANK-TESTER von Christian Planckh"
COLOR 3,2 : LOCATE 3,10
PRINT "Source-File (UNBEDINGT IN ASCII-Format)"
COLOR 1,0 : LOCATE 3,52
INPUT File$
COLOR 3,2 : LOCATE 5,10
PRINT "Destination-File"
COLOR 1,0 : LOCATE 5,52
INPUT Dest$
COLOR 3,2 : LOCATE 7,10
PRINT "Soll ohne Unterbrechung geprüfert werden"
COLOR 1,0 : LOCATE 7,52
INPUT Ent$
IF Ent$ = "n" OR Ent$ = "N" THEN a = 1
LOCATE 10,1
REM Datei laden
OPEN File$ FOR INPUT AS #1
OPEN Dest$ FOR OUTPUT AS #2
ON ERROR GOTO Fehler
REM Hier gehts richtig los

Main:
```

```
REM ein Zeile der Datei lesen (bis CHR(13) = Return-Code)
LINE INPUT #1,a$
MainGoOn:
REM Test auf letztes Zeichen = Leerzeichen (Space)
IF RIGHT$(a$,1) = CHR$(32) THEN
COLOR 3
IF Lauf = 0 THEN
PRINT : PRINT a$+"###BAD LINE###"
END IF
a$ = LEFT$(a$,LEN(a$)-1)
IF RIGHT$(a$,1) = CHR$(32) THEN
Lauf = 1
GOTO MainGoOn
ELSE
PRINT a$+"###CORRECTED###" : PRINT
PRINT #2,a$ : COLOR 1,0
Lauf = 0
END IF
IF a = 1 THEN
SLEEP : SLEEP : SLEEP
ELSE
END IF
ELSE
PRINT a$+"###LINE END###"
PRINT #2,a$ : COLOR 1,0
IF a = 1 THEN
SLEEP : SLEEP : SLEEP
ELSE
END IF
END IF
GOTO Main
Fehler:
IF ERR = 62 THEN END : CLOSE 1,2,3 : CLEAR
Break:
CLS
PRINT "USER BREAK"
CLOSE 1,2,3
CLEAR
WINDOW CLOSE 2
END
```

BlankTest.bas

entfernt Leerzeichen am Ende jeder Zeile

Nach dem Start fragt das Programm zunächst nach der Quell-Datei (Source-File). Geben Sie hier den Namen und kompletten Pfad der zu überprüfenden Datei an. Danach <Return> drücken.

Die Datei muß im ASCII-Format vorliegen. BASIC-Programme sind also zu speichern mit:

```
SAVE <name>., a
```

Bei der folgenden Frage nach der Ziel-Datei (Destination-File) tippen Sie bitte Name und Pfad der Datei ein, in die das korrigierte Listing geschrieben wird. Ist die Datei noch nicht vorhanden, erzeugt Blank-Test eine dementsprechende neue ASCII-Datei. Der gewählte Name darf nicht mit dem der Ursprungsdatei identisch sein.

»Blank-Test« überprüft Textdateien auf zwei Arten: entweder mit oder ohne Unterbrechung. Antworten Sie auf die entsprechende Frage im Programm mit <J> (»Ja«) bzw. <Y> (»Yes«), testet »Blank-Test« jede Zeile, gibt sie mit einigen Zusatzinformationen aus und testet die nächste. Geben Sie <N> (»Nein«) ein, unterbricht »Blank-Test« nach jeder überprüften und ausgegebenen Zeile und wartet auf einen Druck auf die Leertaste.

Hinter jede getestete und ausgegebene Zeile schreibt das Programm eine kurze Ergebnisprotokoll, das nicht mitgespeichert wird: – Findet »Blank-Test« keine Leerzeichen hinter dem Text der Zeile, schreibt es dazu:

```
###LINE END###
```

Am Bildschirm erscheint z.B.: PRINT "Fischer's Fritz"###LINE END###

– Findet der Prüfalgorithmus Blanks hinter den Befehlen einer Zeile, lautet der Zusatztext: ###BAD LINE###

Anschließend gibt das Programm die korrigierte Zeile und die Meldung ###CORRECTED### aus; die Zeile erscheint farbig hervorgehoben. Ein Beispiel:

```
PRINT "Kasimodo" ###BAD LINE###
PRINT "Kasimodo"###CORRECTED##
```


Blanks zu entfernen, ist sicher nur eine Art, ASCII-Texte zu bearbeiten. Sie können auf ähnliche Weise z.B. in einem Programm bestimmte Zeichen durch andere ersetzen und das Ergebnis in einer neuen Datei speichern. Probieren Sie's aus: »Blank-Test.bas« bietet Ihnen eine gute Ausgangsbasis für eigene Programme.

Christian Planckh/ub

BASIC sortiert

»SortSUB.bas« dient dazu, Unterprogramme in einem Programm zu sortieren und ans Ende des Listings zu stellen, was z.B. wichtig ist, wenn man mit dem AC-BASIC-Compiler arbeitet.

Christian Planckh/ub

```
REM SortSub.bas

REM Wichtiges Tool u.a. für Besitzer des AC-BASIC-Compilers:
REM Da SUB-Routinen, die nicht am Ende eines Programms stehen,
REM nicht korrekt übersetzt werden, sollte man sie mit
REM SortSub.bas ans Ende verlegen.

ON BREAK GOSUB Breek
BREAK ON

Variablen:
True      = -1 : False    = 0
Zahlen$   = "0123456789"
Alphabet$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

Einlesen:
WINDOW 2,"BLANK-TESTER von Christian Planckh"
COLOR 3,2 : LOCATE 3,10
PRINT "Source-File (UNBEDINGT IN ASCII-Format)"
COLOR 1,0 : LOCATE 3,52
INPUT File$
COLOR 3,2 : LOCATE 5,10
PRINT "Destination-File"
COLOR 1,0 : LOCATE 5,52
INPUT Dest$
COLOR 3,2 : LOCATE 7,10
PRINT "Soll ohne Unterbrechung geprüfert werden "
LOCATE 8,10
INPUT Ent$
IF Ent$ = "n" OR Ent$ = "N" THEN a = 1
LOCATE 10,1
OPEN Dest$ FOR OUTPUT AS #1
FOR Pass = 1 TO 3
  OPEN File$ FOR INPUT AS #2
  InSub = False : InMain = True
  WHILE NOT EOF(2)
    LINE INPUT #2, InLine$
    Work$ = InLine$ : Work$ = UCASE$(Work$)
    CALL Bypass(Work$)
    CALL StripNum(Work$)
    CALL Bypass(Work$)
    CALL StripLab(Work$)
    CALL Bypass(Work$)
    CALL GetId(Work$, id1$)
    CALL Bypass(Work$)
    CALL GetId(Work$, id2$)
    IF id1$ = "SUB" THEN InSub = True : InMain = False
    IF NOT((id1$ = "DIM") AND (id2$ = "SHARED")) THEN GOTO NoDimShared
    IF (Pass = 1) THEN PRINT #1, InLine$
    GOTO DoneDimShared
  NoDimShared:
    IF InSub AND (Pass = 3) THEN PRINT #1, InLine$
    IF InMain AND (Pass = 2) THEN PRINT #1, InLine$
    IF (id1$ = "END") AND (id2$ = "SUB") THEN InSub = False : InMain = True
  DoneDimShared:
    CLOSE #2
  WEND
NEXT Pass
CLOSE #1
WINDOW CLOSE 2
END
Fehler:
IF ERR = 62 THEN END : CLOSE 1,2 : CLEAR
Breek:
CLS
```

```
PRINT "USER BREAK"
CLOSE 1,2
CLEAR
WINDOW CLOSE 2
END

SUB Bypass(St$) STATIC
WHILE (LEFT$(St$,1) = " ") OR (LEFT$(St$,1) = CHR$(9))
  St$ = MID$(St$,2)
WEND
END SUB

SUB StripNum(St$) STATIC
SHARED Alphabet$,Zahlen$
WHILE (St$ <> "") AND (INSTR(Zahlen$,LEFT$(St$,1)) <> 0)
  St$ = MID$(St$,2)
WEND
END SUB

SUB StripLab(St$) STATIC
SHARED Alphabet$,Zahlen$
Work$ = St$
IF INSTR(Alphabet$,LEFT$(Work$,1)) = 0 THEN EXIT SUB
WHILE (Work$ <> "") AND (INSTR(Alphabet$+Zahlen$,LEFT$(Work$,1)) <> 0)
  Work$ = MID$(Work$,2)
WEND
IF LEFT$(Work$,1) = ":" THEN St$ = MID$(Work$,2)
END SUB

SUB GetId(St$,Id$) STATIC
SHARED Alphabet$,Zahlen$
Id$ = ""
IF INSTR(Alphabet$,LEFT$(St$,1)) = 0 THEN EXIT SUB
WHILE (St$ <> "") AND (INSTR(Alphabet$+Zahlen$,LEFT$(St$,1)) <> 0)
  Id$ = Id$+LEFT$(St$,1)
  St$ = MID$(St$,2)
WEND
END SUB
```

SortSUB.bas

sortiert BASIC-Programme

Line buffer overflow

Beim Editieren von BASIC-Listings können Sie eine Zeile mit <Backspace> an die vorherige anhängen. Manchmal quittiert BASIC den Versuch jedoch mit: »Line buffer overflow«. Schneiden Sie in diesem Fall ein Leerzeichen mit CUT aus der vorherigen Zeile aus und schon funktioniert's.

irw

Funktionstasten nutzen

Professionelle Programme werden oft über die Funktionstasten gesteuert. Das funktioniert auch mit AmigaBASIC, weil der Interpreter bei INKEY neben den normalen Zeichen auch Steuerzeichen übergibt. Den ASCII-Code der Taste ermitteln Sie mit ASC(). Beispiel für eine Abfrage:

```
taste$ = INKEY$
WHILE taste$ = ""
  taste$ = INKEY$
WEND
Code = ASC(taste$)
IF Code = 129 THEN PRINT "F1 gedrückt!"
IF Code = 130 THEN PRINT "F2 gedrückt!"
```

Da die ASCII-Codes der Funktionstasten nicht im Handbuch stehen, liefern wir sie mit der folgenden Tabelle nach.

irw

F-Taste	Code	F-Taste	Code
F1	129	F6	134
F2	130	F7	135
F3	131	F8	136
F4	132	F9	137
F5	133	F10	138

Error while opening

So eine Meldung erscheint evtl. nach dem Anklicken eines Programmpiktogramms. Ihre Ursache: Die Workbench hat AmigaBASIC nicht gefunden.

Beim Speichern eines Programms hinterlegt der Interpreter in der »info«-Datei die Information, in welchem Verzeichnis er selbst (also AmigaBASIC) zu finden ist (Eintrag »default tool«). Beim Anklicken des Piktogramms sucht die Workbench dort die Datei AmigaBASIC, um sie zu laden. Findet die Workbench den Interpreter nicht, erscheint die obige Fehlermeldung.

Leider stimmt der Eintrag in der »info«-Datei in den meisten Fällen nicht. Klicken Sie dann das Piktogramm einmal an, rufen Sie danach Info im Workbench-Menü auf und tragen Sie in der Zeile »Default Tool« den korrekten Pfad ein (z.B. »ExtrasD:AmigaBASIC«).

irw

Zu wenig Speicher?

Jedes geöffnete Fenster braucht Speicherplatz. Wenn Ihr Computer nur 512 KByte hat, können geöffnete Fenster soviel Speicher belegen, daß der verbleibende Anteil für das gewünschte Programm nicht reicht. Oft hilft ein Trick:

Schieben Sie das Piktogramm des gewünschten Programms aus dem Fenster auf den Workbench-Screen. Schließen Sie danach alle Fenster. Auf der Workbench sollten sich jetzt nur noch das Piktogramm der RAM-Disk, das der Diskette und das Ihres Programms befinden. Wenn Sie Glück haben, hat BASIC nun genug Speicher zur Verfügung. Wenn nicht, können Sie aus der »Startup-Sequence« die Befehle LOADWB und ENDCLI löschen (nur beide zusammen), denn die Workbench braucht eine Menge Speicher. Programm und AmigaBASIC können jetzt nur noch vom CLI oder der Shell gestartet werden. In diesem Fall ist es sogar besser, gleich den Programmstart in die »Startup-Sequence« einzubauen.

irw

Zeichnen mit System

Wer in BASIC programmiert, erstellt i.a. Grafiken in einem Fenster mit den Zeichenfunktionen des AmigaBASIC. Vielen C-Programmierern ist allerdings bekannt, daß das Betriebssystem des Amiga spezielle Grafikroutinen bereitstellt. Sie sind in der »graphics.library« zusammengefaßt. Diese Funktionen sind in mancher Hinsicht vielseitiger als die BASIC-Befehle. Sie erlauben es z.B. direkt auf einem Screen zu zeichnen. Das erspart das speicherraubende Öffnen eines Fensters.

Was allerdings alle Zeichenroutinen benötigen, ist die Übergabe eines speziellen Zeigers; ein Zeiger auf die Rastport-Struktur. In diesem Datensatz sind alle wichtigen Daten zusammengefaßt, die das System zum Zeichnen auf einer bestimmten Fläche benötigt. Dabei kann es sich um ein Fenster (Window) oder auch um einen Bildschirm (Screen) handeln. Wenn beispielsweise eine Linie in einem Window gezeichnet werden soll, sieht der Befehl in C wie folgt aus:

```
Draw(Window->RPort, 10, 100)
```

Ein anderer Weg ist, die Rastport-Struktur eines normalen Intuition-Screens zu verwenden:

```
Draw (&(Screen->RastPort) 10, 100)
```

Die Möglichkeit, die Grafikfunktionen des Systems zu nutzen, bietet sich auch den BASIC-Anhängern. Doch muß der BASIC-Programmierer dann auf die normalen Grafikbefehle verzichten. Dafür erhält er andere effiziente Funktionen in die Hand. Um an die begehrten Routinen zu kommen, benötigt man die Datei »graphics.bmap«. Sie befindet sich auf der ExtrasD-Diskette im Ordner »Demos«. Die Bibliothek wird geöffnet durch den Befehl:

```
LIBRARY "graphics.library"
```

Am Ende eines Programms muß man dann immer daran denken, die Bibliothek wieder zu schließen. Der Befehl hierzu:

```
LIBRARY CLOSE
```

Ist die Bibliothek offen, bleibt nur eine Frage: Wie kommt ein Programmierer an die Adressen der Fenster-, Screen- und Rastport-Strukturen? Der Zeiger auf den Datensatz eines Fensters ist schnell gefunden. Der Aufruf

```
window = WINDOW(7)
```

liefert die gewünschte Adresse. Den Zeiger auf den Rastport eines Fensters findet man in der Fensterstruktur an 50.Stelle.

```
rastw = PEEKL(window+52)
```

Beim Screen sieht die Sache komplizierter aus. Zuerst öffnet man wie gewohnt einen Screen. Jetzt öffnet man ein kleines Fenster auf dem Screen, das kaum Speicherplatz kostet:

```
WINDOW 2, "", (0,0)-(0,0),0,1
```

Über den Umweg des Fensters erhält man den Zeiger auf den Datensatz des Screens:

```
screen = PEEKL(WINDOW(7)+46)
```

Der neue Zeiger liefert die Adresse der Screenstruktur. Das Fenster kann nun wieder geschlossen werden. Um an die Struktur des Rastports diese Screens zu gelangen, genügt die Anweisung:

```
&rast = screen-6+84
```

```
view = screen+44
```

Die zweite Zeile ermittelt einen Zeiger auf eine weitere wichtige Struktur; die ViewPort-Struktur. »view« wird benötigt, wenn man z.B. die RGB-Werte einer Farbe ändern möchte. Anstelle von PALETTE schreibt man:

```
CALL SetRGB4(view, 1, 15,15,15)
```

Der Aufruf setzt die Farbe 1 auf weiß. Um eine andere Zeichenfarbe zu wählen, ruft der Programmierer nun nicht mehr COLOR sondern die Funktion:

```
CALL SetAPen(rast, Farbrnummer)
```

Ein wichtiger Befehl ist MOVE. Er erlaubt es, den Grafikkursor zu positionieren. Viele Grafikbefehle aus der Grafikbibliothek starten ihre Aktion von diesem Cursor aus.

Den kompletten Bildschirm löscht der Befehl:

```
CALL Move(rast,0,0)
```

```
CALL ClearScreen(rast)
```

Eine Linie zieht man z.B. von (10,10) bis (100, 100) mit:

```
CALL Move(rast, 10, 10)
```

```
CALL Draw(rast, 100, 100)
```

Bei der Ausgabe von Text muß man jetzt auf den Bildpunkt genau bestimmen, wo der Text stehen soll. Dies geschieht wiederum mit MOVE und dann durch den Aufruf:

```
CALL Text (rast, SADD("Text"), textlänge)
```

Die SADD-Funktion in diesem Aufruf ist notwendig, da zur Textausgabe die Anfangsadresse des Textes im Speicher erforderlich ist. Damit soll es erst einmal genug sein. Probieren Sie die neuen Optionen, die Ihnen das Betriebssystem auch in BASIC eröffnet einmal aus. Es gibt viel zu entdecken.

Jürgen Brendel/ub

So liegen Sie richtig

In BASIC ist immer das zuletzt definierte Fenster aktiviert. Möchte man nun in einem anderen Fenster Daten von der Tastatur einlesen, leitet man mit

```
WINDOW n
```

die Eingabe entsprechend um. Bevor jedoch eine Eingabe erfolgen kann, muß das Window mit der Maus aktiviert werden. Der Aufruf der Intuition-Anweisung

```
ActivateWindow(&WindowHdl)
```

vor dem INPUT-Befehl macht diesen Handgriff überflüssig. Das Ganze in einem Programm sieht so aus:

```
LIBRARY "intuition.library"
```

```
WINDOW 2, "Ein Test-Fenster", (10,10)-(100,100),15
```

```
CALL ActivateWindow(WINDOW(7))
```

```
INPUT a$
```

Das kurze Beispiel zeigt, wie Sie ein Fenster auch ohne Maus aktivieren. In der ersten Zeile steht ein Statement zum Öffnen der Intuition-Library. Eingefleischte BASIC-Programmierer wissen selbstver-

ständig, daß dieser Befehl nur funktioniert, wenn sich die Datei »Intuition.bmap« im selben Verzeichnis wie das BASIC-Programm oder im Ordner »libs« der Startdiskette befindet.

Bruno Goldmann/ub

Patch: BASIC ohne Bilder

Viele BASIC-Programmierer haben eine »persönliche« BASIC-Diskette angelegt – mit Autostart, AmigaBASIC und allen wichtigen »bmap«-Dateien. Auf der Diskette kann man Platz sparen, wenn man auf die Workbench verzichtet; BASIC-Programme lassen sich auch vom CLI starten, dann braucht man die Icons nicht mehr; man kann die »info«-Dateien löschen. Allerdings erzeugt AmigaBASIC immer neue Piktogramme, wenn man ein Programm sichert. Die Lösung hierfür ist ein Patch:

```
CLEAR, 50000g
NeueStelle = ""
For I = 1 TO 12
  NeueStelle$ = NeueStelle$ + CHR$(0)
OPEN ":AmigaBASIC" FOR INPUT AS #1 LEN=4096
OPEN ":NeuesBASIC" FOR OUTPUT AS #2 LEN =4096
PRINT #2, INPUT$(H57a4,#1);
PRINT #2, INPUT$(H57a4,#1);
PRINT #2, INPUT$(H57a4,#1);
PRINT #2, INPUT$(H57a6,#1);
PRINT #2, NeueStelle$;
AS=INPUT$(12,#1)
PRINT a$
WHILE NOT EOF(1)
  PRINT #2,INPUT$(1,#1)
WEND
CLOSE #1,#2
```

Patch1.bas

Brauchen Sie Icons zu Ihren BASIC-Programmen?

Dieser Patch ist für die BASIC-Version mit einer Länge von genau 103 500 Byte geeignet. Ein kleiner Nebeneffekt: Programme werden vom BASIC-Interpreter schneller gespeichert.

Carsten Mallek/ub

GROSSES List-Fenster

Der im AMIGA-Magazin 8/88, Seite 87 abgedruckte Tip verändert das Ausgabe- und List-Fenster des AmigaBASIC mit einem Disketten-Monitor. Die Änderung läßt sich auch per Programm realisieren:

```
OPEN "R",1,"AmigaBASIC",1
FIELD #1,1 AS A$
LSET A$ = CHR$(1)
PUT #1,22509
LSET A$ = CHR$(0)
PUT #1,22510
PUT #1,22540
PUT #1,22541
PUT #1,22542
LSET A$ = CHR$(249)
PUT #1,22544
LSET A$ = CHR$(104)
PUT #1,22546
CLOSE #1
END
```

Patch2.bas

Spendieren Sie AmigaBASIC ein großes Fenster

Wenn Sie das Programm eingegeben haben, testen Sie es am besten mit einer Kopie von AmigaBASIC – sicher ist sicher.

Matthias Brehl/ub

ASCII-Codes

Welches Zeichen hat welchen ASCII-Code? Hier ein BASIC-Programm, das eine Liste aller verfügbaren Zeichen ausdruckt. Bitte beachten Sie, daß dieses Programm Endlospapier voraussetzt.

Andree Popp/ub

```
' Programm zum Drucken einer ASCII-Tabelle (Endlospapier!)
' In der DATA-Anweisung steht zunächst die
' ASCII-Nummer eines nichtdruckbaren
' Steuerzeichens, dann ggf. eine Bezeichnung
DIM Code$(255)
Do$ = STRING$(3,205)
Ein$ = STRING$(3,196)
Sonder$ = CHR$(222)+CHR$(219)+CHR$(221)
OPEN "par:" FOR OUTPUT AS #1
Anfang:
PRINT #1,CHR$(27);"a";CHR$(1);CHR$(27);"r";CHR$(0);
PRINT #1,CHR$(201);
FOR i = 0 TO 6
  PRINT #1,Do$;CHR$(209);Do$;CHR$(203);
NEXT
PRINT #1,Do$;CHR$(209);Do$;CHR$(187)
FOR i = 0 TO 7
  PRINT #1,CHR$(186);"DEZ";CHR$(179);"ASC";
NEXT
PRINT #1,CHR$(186)
Abtrennung$ = CHR$(199)
FOR i = 0 TO 6
  Abtrennung$ = Abtrennung$+Ein$+CHR$(197)+Ein$+CHR$(215)
NEXT
Abtrennung$ = Abtrennung$+Ein$+CHR$(197)+Ein$+CHR$(182)
Einlesen:
FOR i = 0 TO 255
  Code$(i) = " "+CHR$(i)+" "
NEXT
RESTORE Steuerzeichen
Test:
READ a,a$
IF a = -1 AND a$ = "-1" THEN GOTO Tabelle
Code$(a) = LEFT$(a$+" ",3)
GOTO Test
Tabelle:
FOR b = 0 TO 31
  PRINT #1,Abtrennung$
  FOR j = b TO b+224 STEP 32
    PRINT #1,CHR$(186);RIGHT$( " "+STR$(j),3);CHR$(179);Code$(j);
  NEXT
  PRINT #1,CHR$(186)
NEXT
Schluss:
PRINT #1,CHR$(200);
FOR i = 0 TO 6
  PRINT #1,Do$;CHR$(207);Do$;CHR$(202);
NEXT
PRINT #1,Do$;CHR$(207);Do$;CHR$(188)
PRINT #1,CHR$(27);"a";CHR$(0);CHR$(28);"@";
CLOSE #1
END
Steuerzeichen:
DATA 0,NUL,7,BEL,8,BS,9,HT,10,LF,11,VT,12,FF,13,CR,14,SO,15,SI,17
DATA DC1,18,DC2,19,DC3,20,DC4,24,CAN,25,EM,27
DATA ESC,28,FS,32,SP,127,DEL,255,SP
DATA 1,,2,,3,,4,,5,,6,,16,,22,,23,,26,,29,,30,,31,,-1,-1
```

ASCII.bas

Das Programm gibt die ASCII-Codes aller Zeichen auf dem Drucker aus

CUT & COPY einfacher

Wenn Sie in einem BASIC-Listing ein Wort mit »Cut« ausschneiden oder mit »Copy« kopieren, brauchen Sie es nicht umständlich mit einer Mausbewegung markieren. Mit der Maus zweimal anklicken genügt.

irw

Find & Replace

Manchmal möchte man wissen, ob in einem Listing ein Variablenname schon vergeben wurde. Oder man möchte eine Zeichenfolge durch eine andere ersetzen. Dafür gibt es in AmigaBASIC keine Anweisungen. Der Editor ED hat sie; er befindet sich normalerweise im Verzeichnis »c« der Workbench-Diskette. Sie können natürlich auch jeden anderen Editor mit dieser Funktion verwenden.

Kopieren Sie als erstes Ihr BASIC-Programm als Textdatei in die RAM-Disk mit:

```
save "ram:Programmname",a
```

Dann verkleinern Sie das BASIC-Fenster so weit wie möglich. Damit haben Sie Zugriff auf die Workbench und können ein CLI- oder Shell-Fenster öffnen. Dort rufen Sie den ED auf und geben den Programmnamen an:

```
ed "ram:Programmname"
```

Damit haben Sie das Programm im ED und können alle seine Funktionen nutzen. Wenn Sie nur einen Begriff suchen, drücken Sie <Esc> und geben ein:

```
f"Suchtext"
```

oder

```
f/Suchtext
```

Nach <Return> sucht der Amiga ab der aktuellen Cursorposition bis zum Textende nach der Zeichenfolge »SuchText«. Hat er den Text gefunden, können Sie mit der Tastenkombination <Ctrl g> weiter suchen.

Wollen Sie eine Zeichenfolge durch eine andere ersetzen, geben Sie nach <Esc> ein:

```
e"Suchtext" "Ersatztext"
```

oder:

```
e/Suchtext/Ersatztext
```

Nach <Return> sucht ED ab der aktuellen Cursorposition nach »Suchtext« und ersetzt ihn durch »Ersatztext«. Mit <Ctrl g> suchen und ersetzen Sie weiter.

Das Original des BASIC-Programms bleibt bei alldem unverändert, da wir mit dem ED nur die Version in der RAM-Disk bearbeiten. Speichern Sie das Programm unter dem alten, besser einem neuen Namen, und beenden Sie ED mit <Esc> <q>. Bringen Sie danach das BASIC-Fenster wieder auf normale Größe und laden Sie das geänderte Programm.

irw

So liegen Sie richtig

Mit <Amiga_links V> und <Amiga_links B> kann man DOS-Requester zwar beantworten, aber es wäre auch nützlich, die Requester auf den eigenen Screen umlenken oder verschwinden lassen zu können. Dafür ist das folgende Programm gedacht, dem man – einen Zeiger auf das Fenster übergibt, in dem der Amiga die Requester anzeigen soll;
– 0 übergibt, wenn man Requester auf den Workbench-Screen umlenken möchte; oder
– -1 übergibt, falls man Requester unterdrücken möchte

```
DECLARE FUNCTION FindTask& LIBRARY
LIBRARY "basicdemos/exec.library"
' ggf. Ihren bnaps-Pfad hier eintragen.
MyTask& = FindTask& (0) ' Task suchen

POKE (MyTask&+184),WINDOW(7) ' eigenes Window
' oder -1 fuer kein request, 0 = Workbench

LIBRARY CLOSE
END
```

Requester.bas

Requester umlenken – erstmal in BASIC

Das Ganze ist natürlich auch in Assembler machbar, wie das nächste Listing zeigt.

ub

```
Execbase = 4
FindTask = -294
WindowPtr = -1 ; bei -1 weg, 0 da ,ansonsten
; eigenen WindowPointer eintragen

Demo:
move.l Execbase,a6
sub.l a1,a1 ; eigenen Task
jsr FindTask(a6) ; suchen
move.l d0,a0
move.l Ptr,d0
jsr DosRequest ; Zeiger veraendern
rts ; Ende
; in D0 Zeiger auf Window, in A0 auf den Task
DosRequest:
move.l d0,184(a0) ; in ReqWindowPtr eintragen
rts
```

Requester.asm

Requester umlenken – Lösung in Assembler

Maus in Motion

Im AMIGA-Magazin 2/91 stand ein C-Programm, das den Sprite bewegt, der für den Mauszeiger verantwortlich ist. Das geht natürlich auch in BASIC. »MouseMove.bas« ändert die »Simple-Sprite«-Struktur, was in BASIC einfach zu realisieren ist.

```
DECLARE FUNCTION FindName& LIBRARY
LIBRARY "basicdemos/exec.library"
LIBRARY "basicdemos/graphics.library"
'In Verzeichnis Pfad eintragen, wo die EMAPS sind.
FindBase("graphics.library") 'graphics.library suchen
Grafbase& = Finded&
SimpleSprite& = PEEKL (208+Grafbase&) 'Sprite-
MouseSprite& = PEEKL (SimpleSprite&) 'Struktur
View& = PEEKL (34+Grafbase&) 'aktueller
ViewPort& = PEEKL (View&) 'Viewport
FOR y& = 0 TO 256
MoveSprite ViewPort&,MouseSprite&,10,y&
NEXT y&
LIBRARY CLOSE
END
SUB FindBase(grafname$) STATIC
SHARED Finded&
grafname$ = grafname$+CHR$(0)
Execbase& = PEEKL (4) 'ist absolut
Finded& = FindName& (Execbase&,SADD(grafname$))
END SUB
```

MausMove.bas

Die Maus läßt sich auch per Programm steuern

Sie finden im Beispiel eine Suchroutine, um die Basisadressen von Libraries zu finden; wobei über »ExecBase« die »LibList« nach den Namen durchsucht wird.

Andree Popp/ub

Kleiner zum Scrollen

Viele BASIC-Programmierer vergrößern das LIST-Fenster beim Programmieren wegen der besseren Übersichtlichkeit auf das Maximum. Ein Nachteil: Man verlangsamt dadurch die Geschwindigkeit beim Scrollen enorm. Um das zu verhindern, verkleinern Sie einfach das Fenster wieder soweit wie möglich, bevor Sie durch ein Listing scrollen.

Sean Durkin/ub

Jetzt kommt Bewegung in's Spiel

Der Amiga ist in der Lage, mehr als 640x512 Punkte auf dem Monitor darzustellen. Wie? Im sog. Overscan-Modus (bis 704x564). Um in BASIC ein größeres Bild verwenden zu können, muß der Bildschirm nach links oben verschoben werden, damit alle Punkte sichtbar sind. Diese Einstellung erfolgt mit den Preferences. Sie kann aber auch per Programm vorgenommen werden.

```
LIBRARY "intuition.library"
LIBRARY "graphics.library"
DECLARE FUNCTION ViewAddress& LIBRARY
ViewAdd& = ViewAddress&
POKEW ViewAdd&+14,112
POKEW ViewAdd&+12,30
CALL RemakeDisplay
SCREEN 1,704,560,tiefe,4
WINDOW 2,,,16,1
...
REM Grafikbefehle zum Zeichnen
...
POKEW ViewAdd&+14,129
POKEW ViewAdd&+12,44
RemakeDisplay
END
```

ScreenShift.bas

Im Overscan-Modus vergrößert sich die höchste darstellbare Auflösung um rund 10 Prozent

Mit »MoveScreen« läßt sich ein Screen lediglich in der Vertikalen verschieben. Eine Möglichkeit, ein Bild auch in der Horizontalen zu bewegen und zu zentrieren, besteht im Ändern der View-Daten. Die letzten vier Zeilen des Beispiels stellen vor Verlassen des Programms die alte Bildposition wieder her. Es kann u.U. nötig sein, im Overscan das Bild mit den Reglern am Monitor zu verkleinern, oder je nach Gerät andere Startkoordinaten in der View-Struktur einzusetzen. Hier sollten Sie ein wenig experimentieren.

Bruno Goldmann/ub

Schriftzug fährt ab

Folgendes Programm scrollt den Schriftzug »AMIGA-Magazin« von links oben schräg nach unten und vergrößert in dann.

ub

```
LOCATE 1,1 : PRINT "AMIGA-Magazin"
PALETTE 0,0,0,0
PALETTE 1,1,1,1
FOR i=0 TO 100
  SCROLL (i,i)-(i+104,i+9),1,1
NEXT i
FOR i=100 TO 308 STEP 2
  SCROLL (i-6,i)-(i+104,i+9),1,1
NEXT i
```

Scroll.bas

Ein Schriftzug kommt ins Rollen

BASIC schlägt Alarm

Auch in BASIC kann man Alarm-Meldungen (Alerts) ausgeben. Das folgende Listing zeigt, wie man vorgehen sollte.

Das Programm nutzt eine Subroutine zur Fehlerabfrage mit einem Recovery-Alert in BASIC. Aufgerufen wird der Alert mit:

```
DisplayAlert (Typ,Text,Höhe)
```

Die Variable Typ gibt an, welcher Art der Alert ist (Recovery-Alert = 0). Text ist eine Zeiger auf den darzustellenden Text und Höhe be-

stimmt die Höhe des Alerts in Punkten auf dem Bildschirm. Mit veränderter Fehlerabfrage und anderen Texten, kann man das Programm seinen Wünschen anpassen.

Wolfgang Stellwagen/ub

```
DECLARE FUNCTION DisplayAlert& LIBRARY
LIBRARY "intuition.library"
ON ERROR GOTO fehler

REM absichtlich Fehlermeldung erzeugen
OPEN "name" FOR INPUT AS#1

ende:
CLOSE 1
LIBRARY CLOSE
END

REM Fehlerabfrage und Aufruf Alert
fehler:
IF ERR <> 0 THEN CALL Alert
RESUME ende

SUB Alert STATIC
type& = 0 : hoehe& = 60
fehler = ERR : fehler$ = STR$(fehler)
text$ = CHR$(0)+CHR$(150)+CHR$(15)
text$ = text$+"Programmabbruch wegen Fehlernummer: "
text$ = text$+fehler$+CHR$(0)+CHR$(1)
text$ = text$+CHR$(0)+CHR$(150)+CHR$(28)+"Dr"+CHR$(252)
text$ = text$+"cken Sie BITTE die linke Maustaste"
text$ = text$+CHR$(0)+CHR$(1)+CHR$(0)+CHR$(160)+CHR$(45)
text$ = text$+"GESCHRIEBEN VON WOLFGANG STELLWAGEN"+CHR$(0)

warten:
taste& = DisplayAlert&(type&,SADD(text$),hoehe&)
IF taste& <> 1 THEN warten
END SUB
```

Alert.bas

System-Alert auch in BASIC machbar

Teile von BASIC speichern

Mit einem Trick ist es möglich, Teile eines BASIC-Programms zu speichern. Sie müssen lediglich im List-Fenster die erste Zeile des ausgewählten Blocks mit der Maus anklicken, die linke Maustaste drücken und die Maus bis zum Ende des gewünschten Blocks nach unten bewegen. Der ausgewählte Bereich erscheint farbig hervorgehoben. Nun klicken Sie das »Edit«-Menü an und rufen »Copy« auf.

AmigaBASIC speichert nun den Programmteil – im ASCII-Format – im Verzeichnis »RAM:« unter dem Dateinamen »BASICClip«. Sie können die Datei über die Shell, das CLI oder mit einem CLI-Tool wie Diskmaster unbenennen und auf Diskette kopieren.

Wenn Sie den herausgeschnittenen Teil später in einem BASIC-Programm verwenden möchten, laden Sie die gespeicherte Datei über die Shell mit folgendem Befehl ins Verzeichnis »RAM:«:

```
copy DF0: <Name> to RAM:BasicClip
```

Sobald Sie nun im »Edit«-Menü »Paste« anklicken, holt der Amiga den Programmteil aus der RAM-Disk und setzt in an der Stelle im Programm ein, an dem sich die Schreibmarke gerade befindet. Als Alternative bietet es sich an, den gesicherten Programmteil mit MERGE an ein neues Programm anzuhängen.

Martin Grewe/ik

Bildausschnitt

Wer schon einmal mit dem Icon-Editor auf der Extras-Diskette gearbeitet hat, kennt sicher die Funktion »Frame And Save«, um einen rechteckigen Ausschnitt aus einer Grafik als Bilddatei auf Diskette zu sichern. Hierzu klickt man (Taste sofort wieder loslassen) mit der Maus einen ausgewählten Punkt an, an dem eine Ecke des Recht-

ecks liegen soll. Es erscheint ein roter Rahmen, der mit einem Ende am Eckpunkt und mit dem anderen am Mausfeil hängt. Sie können den Rahmen mit der Maus so bewegen, daß er den gewünschten Ausschnitt umschließt; klicken Sie den entsprechenden zweiten Eckpunkt des Rechtecks, über dem der Mauszeiger am Schluß steht, ebenfalls an. Der Amiga speichert den angewählten Teil des Bilds anschließend in einer Datei, deren Name Sie im Auswahlfenster der SAVE-Funktion vorgeben.

So eine Schere für Bildausschnitte ist natürlich auch eine feine Sache, wenn man andere Grafiken bearbeiten möchte. »MakeBox.sub.bas« hält eine entsprechende Funktion für AmigaBASIC bereit. Das Programm, in dem Sie »MakeBox.sub.bas« aufrufen, benötigt die geöffnete »graphics.library« (siehe Beispiel.

```
SUB Makebox(x,y,i,j) STATIC
```

```
REM bei Programmaufruf stehen in x und y die Anfangskoordinaten des
REM Rechtecks, bei Rückkehr ins aufrufende Programm enthalten i und
REM j die x- bzw. y-Koordinaten des zweiten Eckpunkts.
REM Das Unterprogramm benötigt die Unterroutinen Invers und Normal
```

```
WHILE MOUSE(0) <> 0 : WEND 'Warten, bis Maustaste losgelassen wurde
CALL Invers
```

```
REM Auslesen der Mauskoordinaten
```

```
a = MOUSE(0)
i = MOUSE(1)
j = MOUSE(2)
```

```
REM Werte zwischenspeichern
```

```
ii = i
jj = j
```

```
box1:
```

```
REM Zeichnen des Rechtecks
```

```
LINE (x,y)-(ii,jj),,b
```

```
box2:
```

```
a = MOUSE(0)
i = MOUSE(1)
j = MOUSE(2)
IF a = 0 AND ii = i AND jj = j THEN GOTO box2
```

```
REM erneutes Zeichnen stellt ursprüngliches Zustand wieder her,
REM da Grafik im XOR-Modus
```

```
LINE (x,y)-(ii,jj),,b
ii = i
jj = j
```

```
REM Maustaste nicht gedrückt
IF a<>-1 THEN GOTO box1
```

```
REM ist die Maus überhaupt bewegt worden?
IF x = i OR y = j THEN GOTO box1
```

```
REM normalen Graphikmodus wieder einschalten
CALL Normal
END SUB
```

```
SUB Invers STATIC
REM XOR-Graphik einschalten, der Aufruf benötigt die Adresse der
REM Rastport-Struktur
CALL SetDrMd& (WINDOW(8),3)
END SUB
```

```
SUB Normal STATIC
REM normalen Grafikmodus (JAM1 anschalten)
CALL SetDrMd& (WINDOW(8),1)
END SUB
```

MakeBox.sub.bas

Schere für Grafiken

Das folgende Programm zeigt ein Beispiel für den Aufruf des Unterprogramms »MakeBox.sub.bas«.

Thomas Blenkers/ik

```
LIBRARY "Graphics.Library"
Schleife:
WHILE click = 0 AND a$<>CHR$(27)
SLEEP
a$ = INKEY$
click = MOUSE(0)
x = MOUSE(1)
y = MOUSE(2)
WEND
CALL MakeBox (x,y,a,b)
PRINT "Sie haben ein Rechteck von"; x; ", "; y;" bis"
PRINT a; ", "; b; " angewählt"
IF a$<>CHR$(27) THEN GOTO Schleife
LIBRARY CLOSE
END
```

Demo.bas

Beispiel für den Einsatz von »MakeBox.sub.bas«

Anderer Zeichensatz

Warum soll man immer denselben Zeichensatz (Font) verwenden, wenn man mit dem Amiga arbeitet? Benutzen Sie doch einmal einen anderen Font – im Verzeichnis »FONTS« der Workbench-Diskette bzw. der Extras-Diskette stehen so viele verschiedene.

Das Programm »Font.bas« versucht zunächst, den gewünschten Font aus dem Kickstart-ROM (»OpenFont()«) und dann von Diskette (»OpenDiskFont()«) zu laden. Falls es den Zeichensatz findet, lädt es ihn, und die Routine »SetFont()« setzt die Schrift; sie wird am Ende des Programmes mit »CloseFont()« wieder gelöscht.

Bitte achten Sie darauf, daß im Verzeichnis, in dem Sie »Font.bas« ablegen auch die Dateien »diskfont.bmap« und »graphics.bmap« stehen müssen, damit das Programm läuft.

Karsten Schmitt/ik

```
' Programm zum Laden und Setzen einer Schrift
' aus einem Diskverzeichnis oder aus dem
' ROM (Topaz 8 oder Topaz 9)
' hier wird Times von der Extras1.3D:fonts gesetzt
```

```
LIBRARY "graphics.library"
LIBRARY "diskfont.library"
DECLARE FUNCTION OpenDiskFont& LIBRARY
DECLARE FUNCTION OpenFont& LIBRARY
```

```
SCREEN 2,640,256,2,2
WINDOW 2,"",0,2
OpenFont "Extras1.3D:fonts/Times",24
LOCATE 5,5: PRINT "Dies ist die Schrift »Times«"
WHILE INKEY$ = "" : WEND
GOTO Ende
```

```
SUB OpenFont (FontName$,Height%) STATIC
SHARED Font&
FontName$ = FontName$+".font"+CHR$(0)
TextAttr(0) = SADD(FontName$)
TextAttr(1) = Height%*2^16
Font& = OpenFont& (VARPTR(TextAttr(0)))
IF Font& = 0 THEN Font& = OpenDiskFont& (VARPTR(TextAttr(0)))
IF Font& THEN SetFont WINDOW(8),Font&
END SUB
```

```
Ende:
GOSUB CloseFont
LIBRARY CLOSE
WINDOW CLOSE 2:SCREEN CLOSE 2
END
```

```
CloseFont:
IF Font& THEN CloseFont Font&
RETURN
```

Font.bas

Neue Zeichensätze braucht das Land

Drucker testen

Das folgende Programm liest ein Byte aus der Speicherzelle aus, in der sich der aktuelle Druckerstatus befindet und gibt je nach Wert einen bestimmten Text aus.

Karsten Schmitt/ik

```
' Drucker-Test liefert den aktuellen
' Status des Druckers
WHILE INKEY$=""
    erg = PEEK(12570624&) 'SBFD00 - CIA-Register (Parallelport)
    LOCATE 1,1
    IF erg = 255 THEN PRINT "Drucker nicht eingeschaltet "
    IF erg = 252 THEN PRINT "Drucker ist ONLINE "
    IF erg = 251 THEN PRINT "Drucker hat kein Papier mehr"
    IF erg = 249 THEN PRINT "Drucker ist OFFLINE "
WEND
LOCATE 1,2
```

»Druckstatus.bas«

überprüft den angeschlossenen Drucker

Wieviel Speicher

Das Programm »Avail.bas« zeigt den momentan verfügbaren Speicher an.

```
' Programm zur Ermittlung des freien
' Chip- und Fast-RAMs, sowie des gesamten
' freien Speichers.
LIBRARY "exec.library"
DECLARE FUNCTION AvailMem&() LIBRARY
WHILE INKEY$ = ""
    LOCATE 1,1
    PRINT "Chip Memory : "; AvailMem&(2); "Byte"
    PRINT "Fast Memory : "; AvailMem&(4); "Byte"
    PRINT "Total Memory: "; AvailMem&(0); "Byte"
    PRINT
    PRINT "Largest Chip: "; AvailMem&(2+131072&); "Byte"
    PRINT "Largest Fast: "; AvailMem&(4+131072&); "Byte"
WEND
LIBRARY CLOSE
END
```

»Avail.bas«

Tankuhr für den Speicher

Das BASIC-Programm nutzt die Funktion »AvailMem(Typ)« der »exec.library«. Mögliche Speichertypen sind:

- 0 oder 1 : gesamter Speicher
- 2 oder 3 : nur Chip-Mem
- 4 oder 5 : nur Fast-Mem

Wird 131 072 zu einem Speichertyp addiert, erhalten Sie den größten, freien Speicherblock.

Karsten Schmitt/ik

Darstellung des Henon-Attraktors

Das BASIC-Programm »Henon.bas« zeichnet den aus der Chaos-Forschung stammenden und von Michel Henon entwickelten Attraktor auf einem Amiga-Screen. Dazu wird jeder Punkt der Zeichenebene (bestehend aus x- und y-Koordinate) nach dem Gleichungssystem berechnet

$$x = y + 1 - 1.4 * x^2$$

$$y = 0.3 * x$$

Man muß nur ein Anfangspunkt wählen. Die errechneten Werte werden in einer Schleife immer wieder neu in die Gleichungen eingesetzt. Zur Darstellung werden die ermittelten x- und y-Werte dann noch auf die Bildschirmkoordinaten umgerechnet. Wird die Schleife

oft genug durchlaufen, ergibt sich ein recht genaues Abbild dieser komplexen Zahlenmenge.

```
REM *** Attraktor von Henon ***
REM *** von A.Luedemann ***
REM *** geschrieben am 1.2.91 ***
SCREEN 1,640,240,2,2
WINDOW 1,Henon,,15,1
PALETTE 0,0,0,0 : REM Hintergrund schwarz
PALETTE 1,0,0,0 : REM Window "unsichtbar" machen
PALETTE 2,0,0,0
COLOR 3,0
LOCATE 13,7
PRINT "Der Attraktor von Henon"
x1 = -1.3 : x2 = 1.3 : REM Mengenbereich
y1 = -.4 : y2 = .4
xd = x2-x1
yd = y2-y1
xg = 600 : REM max Ausgabegrösse
yg = 220
ax = 1 : REM Anfangswerte
ay = 1
berechnung:
    nx = ay+1-1.4*ax^2 : REM Gleichungssystem nach Henon
    ny = .3*ax
    ax = nx
    ay = ny
    x = (nx-x1)*xg/xd : REM Umrechnung auf Bildschirmkoordinaten
    y = (ny-y1)*yg/yd
    PSET (INT(x),INT(y)),3
    IF INKEY$ = "s" THEN END
GOTO berechnung
```

Attraktor.bas

Hohe Mathematik mit Amiga

Wenn Sie y1 und y2 verändern, können Sie Ausschnitte der Grafik betrachten.

Alexander Lüdemann/ik

Workbench auf und zu

Folgendes Programm demonstriert, wie Sie von BASIC aus den Workbench-Screen schließen bzw. öffnen:

```
' Programm zum Öffnen und Schließen der Workbench
' List- und Ausgabefenster der WB müssen geschlossen sein

Init:
    LIBRARY "intuition.library"
    DECLARE FUNCTION OpenWorkBench& LIBRARY
    DECLARE FUNCTION CloseWorkBench& LIBRARY

SCREEN 2,640,256,2,2
WINDOW 2,,,2
WINDOW CLOSE 1

CloseWB:
    Result& = CloseWorkBench&
    IF Result& THEN
        PRINT "Workbench geschlossen."
    ELSE
        PRINT "Kann Workbench nicht schließen."
        WHILE INKEY$ = "" : WEND
        GOTO Ende
    END IF

    WHILE INKEY$ = "" : WEND

OpenWB:
    Result& = OpenWorkBench&
    WBenchToBack
    IF Result& THEN
        PRINT "Workbench geöffnet."
        FOR i = 0 TO 2500 : NEXT
    ELSE
        PRINT "Kann Workbench nicht öffnen."
    END
    END IF
```

```
Ende:
WINDOW 1
WINDOW CLOSE 2
SCREEN CLOSE 2
LIBRARY CLOSE
END
```

workbench.bas

Per Programm die Workbench schließen

Die Routinen »OpenWorkBench()« und »CloseWorkBench()« werden zu Beginn als Funktionen definiert, da sie einen Rückgabewert besitzen. Nach erfolgreichem Öffnen der Workbench wird der alte Screen mit »WBenchToBack()« in den Vordergrund gebracht.

Um das Programm erfolgreich zu starten, müssen das List- und Ausgabefenster (Window 1) auf der Workbench geschlossen sein. Außerdem muß sich die Datei »intuition.bmap« im Ordner »libs« oder im selben Verzeichnis wie das Demo-Programm befinden.

ik

Runden mit Basic

Wer in Basic eine Zahl runden will, kann dies leicht mit den Funktionen CINT() und CLNG() machen. Beide haben jedoch den gleichen Fehler: Wenn die Einerstelle gerade ist und die Nachkommastelle genau 5, runden die Funktionen ab, statt auf. Das nächste Programm beseitigt den Fehler mit einer Subroutine, die gleichzeitig auf n Dezimalstellen rundet.

Karsten Schmitt/ik

```
* Programm (Subroutine)
* zum Runden einer Zahl auf n Nachkommastellen
DEFDBL x
RANDOMIZE TIMER
FOR i = 0 TO 20
  x = RND*500
  PRINT x,
  Round x,3
  PRINT x
NEXT i
END
SUB Round (x,n%) STATIC
  x = x*10^n%
  IF (x)-FIX(x) = .5 AND FIX(x)/2=FIX(x/2) THEN x = x+.1
  x = CLNG(x)/10^n%
END SUB
```

Runden.bas

Besser als die Rundungsfunktionen von BASIC

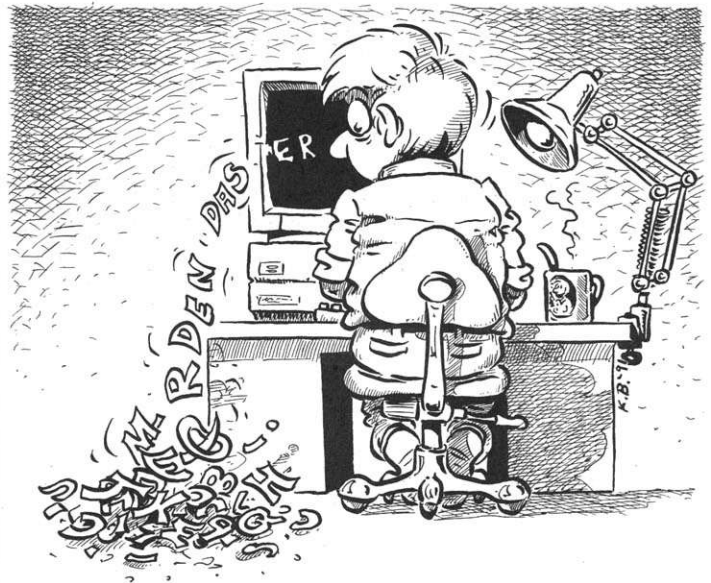
Eigene Icons für AmigaBASIC

Wer hat sich noch nicht darüber geärgert, daß die Icons (Bildsymbole) der AmigaBASIC-Dateien alle gleich aussehen? Ändern der Icons mit dem »IconEd« lohnt sich nicht, denn kaum hat AmigaBASIC 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 AmigaBASIC-Handbuch ausführlich beschrieben.

Anschließend können Sie das gewünschte Icon mit dem auf der Workbench-Diskette befindlichen Symbol-Editor (Icon-Ed) zeichnen.

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/GetDiskObjekt automatisch generiert. Die »bmap«-Datei wurde ins 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 Einheitsicon auftaucht.

Hans Keutgen/pa

```
LIBRARY ":basiclibs/icon.library"
DECLARE FUNCTION PutDiskObjekt& LIBRARY
DECLARE FUNCTION GetDiskObjekt& LIBRARY
filnam$="DBObj/DBIcon"+chr$(0)
diskobj$=GetDiskObjekt&(SADD(filnam$))
IF diskobj$=0 THEN ERROR 150
lslf:
  INPUT "Filename: ";file$
  IF file$="ende" THEN SYSTEM
  file$=file$+chr$(0)
  zw$=PutDiskObjekt&(SADD(file$),diskobj$)
  IF zw$=0 THEN ERROR 151
  GOTO lslf
```

Icon.bas

Gegen Monotonie – sorgt für neue BASIC-Icons

Wohin mit den .bmaps?

Um zu gewährleisten, daß AmigaBASIC 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« quitiert zu bekommen. Außerdem brauchen Sie die »bmap«-Dateien nun nur noch auf einer Diskette, egal wieviele BASIC-Disketten Sie haben.

Angela Schmidt/ub

Copy und Paste

Beim Entwickeln verschiedener Programme mit ähnlichen Strukturen kann es vorkommen, daß sich viele Programmteile untereinander gleichen. Eine einfache Methode, Programmteile von einem Programm ins 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 Speicher übernehmen.

Uwe Bäumlner/pa

Es darf gepatcht werden

Mit einem gepatchten CLI können Sie die tollsten Sachen machen, z.B. die Größe des CLI-Fensters und dessen Titel festlegen. Interessiert? Sie können außerdem einen vier Buchstaben langen CLI-Befehl angeben, der nach dem Start des CLI ausgeführt wird.

Schauen wir uns erst das zum Patchen benötigte BASIC-Programm an. Als kleine Vorbereitung geben Sie im CLI folgendes ein:

```
COPY system:CLI.info RAM:
PROTECT RAM:CLI.info WER
ENDCLI
```

So wird das Experimentieren erleichtert, da das CLI-Icon vor einem Überschreiben geschützt ist. Laden Sie nun AmigaBASIC und geben folgendes Programm ein:

```
OPEN »SYS:system/CLI« FOR INPUT AS #1
OPEN »RAM:CLI« FOR OUTPUT AS #2

PRINT #2, INPUT(6*76C, #1);
Wind$ = "0/0/640/256/Amiga-CLI"+STRING$(6,0)
Befehl$ = "list"
x = Wind$ + CHR$(0) + Befehl$
a$ = INPUT$(LEN(x$), #1)
PRINT #2, x$;
PRINT #2, INPUT(LOF(1)-6*76C-LEN(x$), #1);
CLOSE
```

CLI_Patcher.bas

Damit bekommen Sie ein CLI mit dem gewissen Extra

Die Bedeutung der Variablen:

- Wind\$: Der erste Teil dieses Strings entspricht auch der Fenstereingabe des Befehls NEWCLI. Damit dürfte die Syntax bekannt sein. Wichtig ist, daß die gesamte Zeichenkette 27 Byte lang sein muß. Falls Ihr String kürzer sein sollte, füllen Sie ihn entsprechend mit Null-Bytes auf. (Im obigen Beispiel werden 6 Null-Bytes angehängt).
- Befehl\$: Der auszuführende Befehl mit maximal vier Zeichen.

Das Programm übernimmt den ersten Teil des Original-CLI und kopiert die Daten in ein neues CLI-Programm. Dann flickt der »Patcher« die neuen Parameter des CLI-Windows ein. Am Schluß wird der Rest des Programms CLI kopiert. Sie haben nun ein neues Programm CLI in der RAM-Disk. Probieren Sie es gleich aus: Öffnen Sie das Fenster der RAM-Disk und klicken das CLI-Icon an. Wenn alles stimmt, öffnet sich ein riesiges CLI-Fenster, in dem sofort die Anweisung LIST ausgeführt wird. Im Prinzip können Sie auch alle anderen CLI-Befehle von der Workbench zugänglich machen. Doch damit nicht genug. Wenn Sie mehr wollen, als nur LIST oder einen anderen Befehl auszuführen, verwenden Sie EXECUTE.

»EXECUTE ist aber länger als vier Buchstaben«, meinen einige Zweifler. Richtig, aber nennen Sie EXECUTE einfach in »X« um. Oder geben Sie EXECUTE mittels ASSIGN einen anderen Namen:

```
ASSIGN X: C:EXECUTE
```

Experten sehen schon, was dabei herauskommt. Sie können beim Start des CLI von der Workbench den Befehl EXECUTE ausführen lassen und damit eine beliebige Befehlsdatei abarbeiten lassen. In dieser Batch-Datei können beliebige viele und beliebige lange CLI-Befehle stehen. Allerdings darf der Name des Batch-Files nicht länger sein als zwei Zeichen, aber das läßt sich verkraften, oder?

Vielleicht noch ein paar Anregungen zu diesem mächtigen Tip:

Wenn Sie z.B. ED von der Workbench starten möchten. Wählen Sie ein kleines CLI-Fenster, z.B.:

```
*0/0/300/40/CLI-ED"+STRING$(10,0)
```

Als Befehl geben Sie »ED ?« ein. Wenn Sie nun das CLI-Icon in der RAM-Disk anklicken, öffnet sich ein CLI-Fenster, und der Anwender braucht nur noch den Namen der zu bearbeitenden Datei eingeben. Natürlich sollte man dafür nicht das wertvolle CLI-Icon opfern. Weitaus besser ist es, wenn Sie ein neues CLI erstellen und beispielsweise CLI_ED nennen. Hierzu muß man nur das gepatchte RAM-CLI markieren und aus dem Menü der Workbench »Duplicate« wählen. Die Kopie wird mit »Rename« umbenannt und auf die Workbench kopiert. Und ein letzter Hinweis. Erstellen Sie für alle CLI-Befehle, die Sie von der Workbench aufrufen möchten, eigene Icons.

Angela Schmidt/ub

Gut geschaltet in BASIC

Das Programm »Gadget.bas« ermöglicht es AmigaBASIC-Programmieren, Schalter (Gadgets) einfach und schnell abzufragen:

Die Routine besteht im wesentlichen aus zwei SUB-Befehlen:

• SET.GADGET

```
SET.GADGET Nummer%, x%, y%, Text$
```

Dabei ist:

Nummer% : die Nummer des Gadgets

x% : X-Koordinate

y% : Y-Koordinate

Text\$: Text

Zwei Beispielaufufe zum Ausprobieren:

```
SET.GADGET 1,120,80,"Datei laden"
```

```
SET.GADGET 2,120,60,"Datei speichern"
```

• CHECK.GADGET

```
CHECK.GADGET Anzahl%, Wahl%
```

Hier ist:

Anzahl% : Höchste Nummer der abzufragenden Gadgets

Wahl% : Variable

Auch hier ein Aufrufbeispiel:

```
CHECK.GADGET 2,Wahl%
```

Diese Anweisung fragt die beiden oben gesetzten Gadgets ab. Das Ergebnis steht nun in der Variable Wahl% und kann im Programm eingesetzt werden, z.B. so:

```
IF Wahl%=1 THEN GOTO LadeDatei
```

```
IF Wahl%=2 THEN GOTO SpeichereDatei
```

Mit den Gadgetroutinen sind Sie nun in der Lage, all Ihre BASIC-Programme voll über die Maus steuern zu lassen — ein Komfort, der den Aufwand wert ist.

F.Bachmann/ub

```
REM Höchstmögliche Anzahl der Gadgets festlegen
DIM SHARED gadget.x%(10), gadget.y%(10), gadget.xl%(10), gadget.yl%(10)
SUB SET.GADGET (nr%, x%, y%, text$) STATIC
    gadget.x% (nr%) = x% : gadget.y% (nr%) = y%
    gadget.xl%(nr%) = x% + (LEN(text$)*8)+4 : gadget.yl%(nr%) = y%+12
    LINE (x%, y%)-(gadget.xl%(nr%), gadget.yl%(nr%)), 1, b
    POKEW WINDOW(8)+36, x%+2 : POKEW WINDOW(8)+38, y%+9
    PRINT text$
END SUB
SUB CHECK.GADGET (max%, Back%) STATIC
    MouseWait:
    WHILE SGN(MOUSE(0))
        xx = MOUSE(1) : yy = MOUSE(2)
        FOR i = 1 TO max%
            IF xx>gadget.x%(i) AND xx<gadget.xl%(i) AND yy>gadget.y%(i)
                AND yy<gadget.yl%(i) THEN Back%=i : EXIT SUB
        NEXT i
    WEND
    GOTO MouseWait
END SUB
```

Gadgets.bas

in Basic – am besten mit Unterprogrammen

Programmieren

Gfa-BASIC

Hier nun ein paar Tricks für Gfa-BASIC-Programmierer – sowohl für Gfa-Anfänger als auch echte Profis. Viele Kniffe lassen sich auch in andere Sprachen umsetzen.

Gfa-BASIC ist neben AmigaBASIC einer der beliebtesten BASIC-Dialekte für den Amiga. Deshalb haben wir für Gfa-BASIC-Programmierer natürlich auch eine Menge Tips gesammelt. Alle Programme lassen sich vom GFA-Interpreter mit MERGE laden, wurden mit der Gfa-Version 3.5 geschrieben und auf einem Amiga 2000 mit 1 MByte Chip- und 2 MByte Fast-RAM getestet; laufen aber auch mit der Grundversion eines Amiga 500 mit 512 KByte.

Gfa-Programme immer größer

Wenn man in Gfa-BASIC ein Programm schreibt und es oft startet, ändert und erweitert, stellt man fest, daß das Programm auf der Diskette immer mehr Platz benötigt und der Speichervorgang immer länger dauert. Dem Programm wird mehr und mehr Speichermüll angehängt, den es gar nicht braucht. Dasselbe beobachtet man auch beim normalen AmigaBASIC.

Die Programmgröße läßt sich zum Glück wieder reduzieren, indem Sie das Programm zwischendurch mit »SAVE, A« aus der Menüleiste speichern. Anschließend löschen Sie den Speicher mit NEW und lesen das vorher gespeicherte Programm mit MERGE wieder ein.

Nach dieser Prozedur kann man das Programm wieder mit dem normalen SAVE speichern. Es ist jetzt ein wenig kürzer. Der Effekt macht sich besonders dann bemerkbar, wenn ein Programm viele Variablennamen verwendet. Die Aktion lohnt sich auf jeden Fall, wenn Sie ein Programm fertiggestellt haben und es unkompiert benutzen möchten.

F.Reinert/ub

Gfa-Absturz

Einige Gfa-BASIC-Kommandos werden vom Gfa-Compiler V.3.5 nicht ordnungsgemäß verarbeitet:

- Als erstes ist da der Befehl CHDIR zu nennen. Normalerweise wechselt er das aktuelle Verzeichnis. Ein kompiliertes Programm stürzt allerdings früher oder später ab; spätestens, wenn man es beendet. Ersatzweise sollte ein Pfad, der oft benötigt wird, in einer String-Variablen gespeichert und bei Bedarf dem Filename vorangestellt werden. Das erfordert zwar mehr Programmaufwand, dafür läuft das Programm einwandfrei.
- Als zweiter Befehl ist CHAR zu erwähnen. Man verwendet ihn, um beliebige ASCII-Texte in einen bestimmten Speicherbereich zu schreiben. Bei dem Speicherbereich dürfte es sich vornehmlich um Strukturen handeln.

Ein mit CHAR geschriebener Text wird automatisch mit »CHR\$(0)« abgeschlossen. Der Gfa-Interpreter erledigt diese Aufgabe einwandfrei. Ein kompiliertes Gfa-Programm, das den CHAR-Befehl benutzt, stürzt ab. Um dem aus dem Weg zu gehen, muß man den Text »zu Fuß«, Buchstabe für Buchstabe in den Speicher

schreiben bzw. aus dem Speicher lesen. Hierzu dienen die beiden Unterprogramme »readchar\$« und »writechar\$«, die Sie in Ihre Programme einbauen können:

```
' "CHAR"-Ersatz für GFA-BASIC 3.5
' Adresse = Zieladresse des Textes
' Text$ = Text, der geschrieben werden soll
' Anwendung:
' "C"-Text lesen.... text$=FN readchar$(adresse)
' "C"-Text schreiben... writechar(adresse,text$)
```

```
PROCEDURE writechar(adr$,text$)
LOCAL i
IF text$ <> ""
FOR i = 0 TO LEN(text$)-1
POKE adr$+i,ASC(MID$(text$,i+1,1))
NEXT i
POKE adr$+i,0
ELSE
POKE adr$,0
ENDIF
ENDIF
RETURN
FUNCTION readchar$(adr$)
LOCAL text$,i
text$ = ""
i = 0
DO
EXIT IF PEEK(adr$+i) = 0
text$ = text$+CHR$(PEEK(adr$+i))
INC i
LOOP
RETURN text$
ENDFUNC
```

»readchar« und »writechar«

Gfa-BASIC-Funktionen zum – absturzfrenen ! – Lesen und Schreiben von Daten in Strukturen

»readchar« ist als Funktion ausgelegt, sodaß ein Text mit:

```
text$ = FN readchar$(adresse)
```

in die Variable text\$ übertragen wird. Mit:

```
writechar(adresse,text$)
```

wird der Inhalt von text\$ ab Adresse »adresse« in den Speicher geschrieben und mit CHR\$(0) abgeschlossen. Wenn Sie beide Unterprogramme als ASCII-File mit der Endung ».LST« auf Diskette speichern, lassen sie sich in jedes Programm einbinden (MERGE).

ub

TIPS ZU GFA-BASIC AUF EINEN BLICK

Tip	Thema	Seite
Immer_größer	Tip zum Speichern von Gfa-Programmen	19
Gfa-Absturz	Fehler des Gfa-Compilers beheben	19
Color-Fade	Beliebige Farbreister ein- und ausblenden.	20
IFF-Lib_mit_GFA	IFF-Bilder mit der 'iff.library' von der PD-Disk Fish 301 einlesen und darstellen.	21
Farbreister	Requester zum Ändern der Farbreister	22
Rahmenlos	»req.library« (Requester-Bibliothek) im Einsatz	22
	Fenster ohne Rahmen und Titelleiste	23

Color-Fade mit Gfa-BASIC

Es ist immer eindrucksvoll, wenn Texte nicht einfach auf dem Computermonitor erscheinen, sondern langsam eingeblendet werden. Wie erreicht man so etwas in eigenen Programmen?

Ganz so leicht ist es nicht. Sicher, den ganzen Bildinhalt ein- und auszublenden, ist einfach: Die Farbpalette wird einfach von 0 bis 100 Prozent aufgebaut und schon ist es passiert. Doch wenn Ihre Ansprüche höher sind, Sie z.B. einen Schriftzug vor einem Hintergrundbild einblenden möchten, müssen Sie beliebige Farbbereiche einblenden können.

Dabei tauchen zwei Schwierigkeiten auf:

- Wenn der Hintergrund (Farbe 0) nicht mit ausgeblendet wird, müssen die anderen Farben an den Hintergrund angeglichen werden.
- Wenn der Hintergrund hell ist, müssen einige Farben (je nach ihrer Helligkeit) heller, andere dunkler werden.

Die folgenden Color-Fade-Routinen bestehen aus fünf Unterprogrammen (Listing »color-fade.lst«), mit denen es möglich ist, jedes Farbregister einzeln oder beliebig viele auf einmal auszublenden.

```

Color-Fade mit Gfa-BASIC

FUNCTION read_color(screen,register)
  col% = -1
  IF SCREEN(screen)
    colomap% = LONG(SCREEN(screen)+48)
    col% = GetRGB4(colomap%,register)
  ENDIF
  RETURN col%
ENDFUNC

FUNCTION get_rgb(screen)
  LOCAL i,col%
  IF SCREEN(screen)
    anz_farben = 2^(BYTE(LONG(SCREEN(screen)+88)+5))
    ERASE r(),g(),b()
    DIM r(31),g(31),b(31)
    colomap% = LONG(SCREEN(screen)+48)
    FOR i = 0 TO anz_farben-1
      col% = GetRGB4(colomap%,i)
      r(i) = SHR(col%,8)
      g(i) = SHR((col% AND &HF0),4)
      b(i) = col% AND 15
    NEXT i
    RETURN anz_farben
  ELSE
    RETURN TRUE
  ENDIF
ENDFUNC

PROCEDURE fade_in(screen,start_col,end_col,v)
  LOCAL k,i,r,g,b,r0,g0,b0
  f_start = v-1
  f_end = 1
  f_step = -1
  @fade(screen)
  FOR i = start_col TO end_col
    SETCOLOR i,r(i),g(i),b(i)
  NEXT i
  RETURN

PROCEDURE fade_out(screen,start_col,end_col,v)
  LOCAL k,i,r,g,b,r0,g0,b0
  f_start = 1
  f_end = v-1
  f_step = 1
  @fade(screen)
  FOR i = start_col TO end_col
    SETCOLOR i,r0,g0,b0
  NEXT i
  RETURN

PROCEDURE fade(screen)
  ERASE f()
  DIM f(2,31)
  col% = FN read_color(screen,0)
  r0 = SHR(col%,8)
  g0 = SHR((col% AND &HF0),4)

```

```

b0 = col% AND 15
IF start_col = 0
  r0 = 0
  g0 = 0
  b0 = 0
ENDIF
FOR k = start_col TO end_col
  f(0,k) = (r(k)-r0)/v
  f(1,k) = (g(k)-g0)/v
  f(2,k) = (b(k)-b0)/v
NEXT k
FOR k = f_start TO f_end STEP f_step
  FOR i = start_col TO end_col
    r = INT(r(i)-k*f(0,i))
    g = INT(g(i)-k*f(1,i))
    b = INT(b(i)-k*f(2,i))
    SETCOLOR i,r,g,b
  NEXT i
NEXT k
RETURN

```

Colorfade.lst

Fünf Gfa-Unterprogramme zum sauberen Einblenden, die Sie in eigene Programme einbauen können

Zu den einzelnen Funktionen:

• 1.Funktion:

color = FN read_color(screen,register)

Das Unterprogramm ermittelt die RGB-Zusammensetzung des Farbregisters »register«. Sie setzt sich wie folgt zusammen:

color = rot*256 + grün*16 + blau

Übergeben Sie in »screen« die Nummer des Screens, den Sie verwenden. »color« liefert die Farbeinstellung.

• 2.Funktion:

num_col = FN get_rgb(screen)

Liest alle verwendeten Farbregister und speichert die Werte nach Grundfarben getrennt in den Feldern »r()«, »g()« und »b()«.

»num_col« gibt entweder die Anzahl der möglichen Farben oder im Fehlerfall »-1« an. Sie sollten die Funktion immer aufrufen, nachdem Sie Ihren Screen geöffnet, oder die Farbpalette gewechselt haben. Die von der Funktion gelesenen Farbwerte dienen als maximale Helligkeit beim Ein- und Ausblenden.

• 3. Funktion:

@fade_in(screen,start_col,end_col,v)

Einblenden des Farbbereichs von »start_col« bis »end_col«. Sie müssen die Nummern der Farbregister angeben (0 = Hintergrund, 1 = Schriftfarbe 1, usw.). Mit dem Parameter »v« bestimmen Sie die Geschwindigkeit des Einblendens. Je größer der Wert ist, desto niedriger das Tempo. Brauchbare Werte (unkompiliert) sind 15 bis 50. Bei »v = 50« erkennt man aber deutlich die Farbabstufung.

Übrigens: In unserem Programm ist »screen« mit anzugeben, obwohl sich die Helligkeitsänderungen immer auf dem aktuellen Bildschirm (Screen) bemerkbar machen. Wir umgehen damit den (Tipp-) Aufwand, den aktuellen Screen im Programm festzustellen).

• 4. Funktion:

@fade_out(screen,start_col,end_col,v)

Mit diesem Unterprogramm werden Farbbereiche ausgeblendet. Die Parameter sind die gleichen wie bei fade_in.

• 5. Funktion:

@fade(screen)

Die eigentliche »fade«-Routine, die auch die Unterprogramme »fade_in« und »fade_out« benötigen. Hier werden, je nach gewünschter Geschwindigkeit, die Helligkeitsstufen berechnet und die Farbregister gesetzt.

Sie sollten die fünf Unterprogramme zweckmäßigerweise als ASCII-Datei speichern. Klicken Sie hierzu »SAVE, A« in der Menüleiste des Gfa-Interpreters an. Die Fade-Routinen stehen somit immer zum Einbinden in Programme bereit (MERGE). Das nächste Programm zeigt zur Demonstration der Routinen noch ein Beispiel. Im Listing sind die Unterprogramme zu ergänzen.

' Demo zu Color-Fade in Gfa-BASIC von J.Reinert

```
OPENS 2,0,0,640,256,3,&H8000
OPENW #2,0,0,640,256,0,4111,2
SETCOLOR 0,6,6,6
VOID FN get_rgb(2)
CLS

@fade_out(2,2,7,40)
PCOLOR 2,0
PRINT AT(20,5);"Eine kleine Demonstration für"
PCOLOR 3,0
PRINT AT(20,7);"    Einblenden und"
PCOLOR 4,0
PRINT AT(20,9);"    Ausblenden von Farben"
PCOLOR 5,0
PRINT
PRINT "    Das Aus- und Einblenden von Farben klappt Spitze,"
PCOLOR 6,0
PRINT
PRINT "    wenn man einiges beachtet. So sollte man nie Farben"
PCOLOR 7,0
PRINT
PRINT "    einblenden, die schon eingeblendet sind."
FOR i=2 TO 7
  @fade_in(2,i,i,35)
  DELAY 2
NEXT i
FOR i=1 TO 3
  @fade_in(2,6,7,30)
  DELAY 1
NEXT i
DELAY 6
@fade_out(2,0,7,25)
CLS
FOR i=0 TO 7
  PCOLOR i,0
  PRINT "Das ist die Farbe aus Register ";i;".    Farbwert= $";
  PRINT HEX$(i*256+g(i)*16+b(i))
NEXT i
PCOLOR 7,0
@fade_in(2,0,7,25)
DATA 0,6,2,4,2,6,2,5,3,7
FOR i=1 TO 5
  READ v,b
  LOCATE 2,20
  PRINT "    Ein- und Ausblenden der Farben ";v; bis ";b;""
  FOR n=1 TO 3
    DELAY 0.5
    @fade_out(2,v,b,20)
    DELAY 0.5
    @fade_in(2,v,b,20)
  NEXT n
NEXT i
DELAY 3

@fade_out(2,0,7,28)
CLS
LOCATE 26,13
PCOLOR 7,0
PRINT "Ende der Demonstration"
@fade_in(2,5,7,30)
DELAY 3

@fade_out(2,5,7,30)
DELAY 1
CLOSEW #2
CLOSES 2
END
```

Demo_Color-Fade.gfa

Bitte einblenden -- ein Beispiel wie man's macht

Geben Sie das Demoprogramm »Demo_Color-Fade.gfa« ein und hängen die Unterprogramme »Color-Fade« mit MERGE an. Danach können Sie starten. Unser Beispiel zeigt einige Möglichkeiten des Ein- und Ausblendens. Falsche Parameter, z.B. eine falsche Screen-Nummer, zu hohe Farbregister usw., werden im Programm nicht abgefangen. Treffen Sie hierfür bitte selbst Vorsorge.

F.Reinert/ub

iff.library mit Gfa 3.5

Bilder im ungepackten ACBM-Format zu laden, ist mit den zu Amiga- oder Gfa-BASIC gelieferten Programmen relativ einfach. Bei AmigaBASIC heißt das Programm hierzu »LoadACBM« (Extras-Diskette) und bei Gfa-BASIC 3.5 »LOAD_ACBM«.

Und dann gibt es da noch »LoadIFF«, um eine IFF-Datei zu laden; das Programm ist aber – typisch BASIC – sehr langsam. Schneller geht's mit der »iff.library«. Sie ist u.a. auf der Fish-Disk 301 zu finden. Wie die Library unter Gfa-BASIC 3.5 anzuwenden ist, zeigt das Programm »ShowIFF.bas«.

```
' | ShowIFF mit Gfa-BASIC 3.5
' Die folgenden vier Zeilen müssen immer am Programmanfang stehen
DIM m68%(16) ! Prozessor-Register für RCALL
lib$ = "iff.library" ! Name der Library
LET iffbase% = OldOpenLibrary(VARPTR(lib$)) ! Library öffnen
IF iffbase%<>0 ! ist sie auf?
file$ = DIR$(0) Start-Pfad ermitteln
DO
FILESELECT "Bild laden","Laden",file$,file$ ! Bild auswählen
EXIT IF file$="" ! Ende wenn "Abbruch"
fehler = FN loadiff(file$,0) ! Bild laden
IF NOT fehler
  WHILE MOUSEK = 0 ! Warten, wenn alles klar
  WEND
ENDIF
CLOSES 2 ! Screen schliessen
LOOP
' Und so sollte das Programm beendet werden
VOID CloseLibrary(iffbase%) ! Library schliessen
ELSE
PRINT "Library geht nicht auf"
DELAY 4
ENDIF
END

FUNCTION loadiff(bild$,screennum)
LOCAL fehler,bmhd,file2$,ifffile% ! Lokale Variablen
fehler = TRUE
IF bild$ <> ""
file2$ = bild$+CHR$(0) ! Bildname + chr$(0)
LET ifffile% = FN openiff(VARPTR(file2$)) ! IFF-Datei öffnen
IF ifffile% > 0 ! Bild nicht vorhanden
IF screennum = 0
bmhd = FN getbmhd(ifffile%) ! Bild-Parameter ermitteln
breite = WORD(bmhd)
hoehe = WORD(bmhd+2)
tiefe = BYTE(bmhd+8)
viewmode = FN getviewmodes(ifffile%)
OPENS 2,0,0,breite,hoehe,tiefe,viewmode ! Screen 2 öffnen
screennum = 2
ENDIF
LET screen% = SCREEN(screennum)
LET viewport% = screen%+44
LET rastport% = screen%+84
LET colormap% = LPEEK(viewport%+4)
colotab% = LPEEK(colormap%+4)
bitmap% = screen%+184
farben = FN getcolortab(ifffile%,colotab%) ! Farben holen
VOID LoadRGB4(viewport%,colotab%,farben) ! Farben aktivieren
fehler = FN decodepic(ifffile%,bitmap%) ! Bild zeigen
VOID FN closeiff(ifffile%) ! Datei schliessen
IF fehler = 0
fehler = TRUE
ELSE
fehler = FALSE
ENDIF
ENDIF
RETURN fehler
ENDFUNC
```

' Funktionen der iff.library

```
FUNCTION openiff(filename%)
m68%(8) = filename%
m68%(14) = iffbase%
RCALL iffbase%-30,m68%()
RETURN m68%(0)
ENDFUNC
```



```

FUNCTION closeiff(ifffile%)
m68%(9) = iffbase%
m68%(14) = iffbase%
RCALL iffbase%-36,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION findchunk(ifffile%,chunkname%)
m68%(9) = iffbase%
m68%(0) = chunkname%
m68%(14) = iffbase%
RCALL iffbase%-42,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION getbmhd(ifffile%)
m68%(9) = iffbase%
m68%(14) = iffbase%
RCALL iffbase%-48,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION getcolortab(ifffile%,colortable%)
m68%(9) = iffbase%
m68%(8) = colortable%
m68%(14) = iffbase%
RCALL iffbase%-54,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION decodepic(ifffile%,bitmap%)
m68%(9) = iffbase%
m68%(8) = bitmap%
m68%(14) = iffbase%
RCALL iffbase%-60,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION savebitmap(name%,bmap%,ctab%,crmd%)
m68%(8) = name%
m68%(9) = bmap%
m68%(10) = ctab%
m68%(0) = crmd%
m68%(14) = iffbase%
RCALL iffbase%-66,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION saveclip(name%,bmap%,ctab%,crmd%,x%,y%,w%,h%)
m68%(8) = name%
m68%(9) = bmap%
m68%(10) = ctab%
m68%(0) = crmd%
m68%(1) = x%
m68%(2) = y%
m68%(3) = w%
m68%(4) = h%
m68%(14) = iffbase%
RCALL iffbase%-72,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION ifferror
m68%(14) = iffbase%
RCALL iffbase%-78,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION getviewmodes(ifffile%)
m68%(9) = iffbase%
m68%(14) = iffbase%
RCALL iffbase%-84,m68%()
RETURN m68%(0)
ENDFUNC

FUNCTION newopeniff(filename%,memtype%)
m68%(8) = filename%
m68%(0) = memtype%
m68%(14) = iffbase%
RCALL iffbase%-90,m68%()
RETURN m68%(0)
ENDFUNC

```

IFF-ShowIFF.bas

Nutzen Sie die Funktionen der IFF-Library in Gfa-BASIC, z.B. um Bilder zu laden

Das Programm verarbeitet alle gängigen Bildformate (Lores, Hires, interlaced, noninterlaced, HAM).

Als erstes ist es unbedingt erforderlich, die »iff.library« von der Public-Domain-Diskette ins »libs«-Verzeichnis der verwendeten Workbench-Diskette zu kopieren. Da es sich bei der »iff.library« um eine externe Bibliothek handelt, müssen die ersten vier Zeilen (ab »DIM m68%(16)«) immer am Programmanfang stehen. Die Zeilen dimensionieren ein Integer-Feld, das zur Übergabe der Prozessor-Register verwendet wird, und öffnen die »iff.library«.

Sollte die Bibliothek nicht zu öffnen sein, endet das Programm, da ansonsten ein System-Absturz erfolgt.

Am Ende des Listings befinden sich Funktionen zur Kommunikation mit der Library. Um sich ein wenig Tipparbeit zu sparen, können Sie diese Unterprogramme mit »FDCONV.GFA« (von der Gfa-BASIC-Diskette) aus dem »FD«-File (Endung ».fd«) zur Library selbst erzeugen. Unglücklicherweise werden von »FDCONV.GFA« Prozeduren ohne Rückgabewert erzeugt, sodaß der erzeugte Quellcode noch angepaßt werden muß. Eingefleischte Programmierer sollten das Konvertierungsprogramm so ändern, daß automatisch Funktionen erzeugt werden.

Um Speicherplatz zu sparen, brauchen Sie nur die Funktionen zu übernehmen, die Sie tatsächlich verwenden. Der Vollständigkeit halber binden wir im Beispiel alle ein.

Der wichtigste Programmteil ist das Unterprogramm »loadiff()«. Es öffnet entweder automatisch einen Screen mit den richtigen Parametern oder Sie geben den Screen mit an, in dem die Grafik erscheinen soll. Für die richtige Screen-Einstellung müssen Sie selbst sorgen.

Weitere Erklärung entnehmen Sie bitte den Kommentaren im Listing. Funktionsbeschreibungen und Offsets der Library befinden sich auf der PD-Disk.

Das Unterprogramm »loadiff()« öffnet, wenn kein Screen angegeben ist (übergebene Screen-Nummer = 0), immer einen Screen mit der Nummer 2. Die Variable »iffbase%« darf nicht verändert werden (auch nicht ihr Inhalt). Alle anderen Variablen wie »breite«, »tiefe« etc. können Sie beliebig weiterverwenden.

Beachten Sie, daß der Amiga immer ein ganzes Bild lädt. Hat das Bild eine Breite von 320 und eine Höhe von 256 Punkten, kann man es trotzdem in einen Screen der Ausmaße 640 x 512 laden. Es erscheint dann im oberen linken Viertel.

J.Reinert/ub

Farbrequester-Bibliothek

Aus vielen Programmen kennt man sie: die Color-Requester. Sie dienen dazu, die verwendete Farbpalette zu verändern. Wer so etwas schon einmal programmiert hat, weiß, wie aufwendig es ist. Die »req.library«, zu finden u.a. auf der Fish-Disk 419, nimmt einem viel Arbeit ab.

Sie beinhaltet u.a. einen einfachen Color-Requester, mit dem es möglich ist, Farben mit R-G-B-Reglern einzustellen, Farbtreppe zu erzeugen, Farben zu kopieren und zu tauschen. Außerdem werden die aktuellen RGB-Werte angezeigt.

Zur Verwendung: Als erstes muß man die »req.library« in den »libs«-Ordner der Workbench-Diskette kopieren. Testen Sie dann die Bibliothek mit dem Programm »REQ.LIB-DEMO« (Beispiel in Gfa-BASIC).

Das folgende Demo-Listing ist relativ lang, was z.B. an der Fehler-Auswertung liegt, es zeigt aber auch eine Möglichkeit, System-Rückfragen (Requester wie »please insert...«) auf den eigenen Screen umzuladen; der Workbench-Screen bleibt also im Hintergrund.

Um die Workbenchfarben einzustellen, müssen Sie alle Zeilen mit den Variablen »mytask%« und »oldwindowptr%« löschen; ebenso die Zeile

```
IF mytask% <> 0 ...
```

und den zugehörigen Befehl ENDIF.

Zusätzlich finden Sie im Beispiel eine Funktion zum Ermitteln der Farbwerte: »read_color« (Farbe lesen). Sie wird aufgerufen mit:

```
farbwert = FN read_color(screen,register)
```

»screen« ist die Nummer des verwendeten Screens, »register« enthält die Nummer des Farbregisters, das ausgelesen werden soll. Je nach Tiefe (Bitplanes) des Screens nimmt dieser Parameter Werte von 0 bis 31 an. In »farbwert« erhält man dann die RGB-Anteile zurück. (256*R + 16*G + B). Den Farbwert kann man direkt mit SETCOLOR verwenden. Mit einer geeigneten Programmschleife ist es möglich, die gesamte Farbpalette auf Diskette oder Festplatte zu speichern und später wieder einzustellen.

F.Reinert/ub

```
' ColorRequester mit der "req.library" in Gfa-BASIC

mytask% = FindTask(0) ! eigenen Task suchen
DIM m68%(16) ! 68000-Register
libname$ = "req.library" ! Name der Bibliothek
regbase% = OldOpenLibrary(VARPTR(libname$)) ! Library öffnen
IF regbase% <> 0 ! Library geöffnet?
  OPENS 2,0,0,640,256,3,&H8000 ! Screen öffnen
  OPENW #2,0,0,640,256,0,4111,2 ! Fenster volle Größe
  IF mytask% <> 0
    oldwindowptr% = LONG(mytask%+184) ! Windowpointer in Task-Struktur
    LONG(mytask%+184) = WINDOW(2) ! Windowpointer setzen
    ! Alle Requester gehen nun in neuen Screen
    ' --- Ab hier steht das Hauptprogramm ---
    colour = 0
    newcolor = FN colorrequester(colour) ! Aufruf Color-Requester
    IF newcolor = 65535
      PRINT "Color-Requester wurde Abgebrochen"
    ELSE
      PRINT "Zuletzt gewähltes Farbregister war : ";newcolor
    ENDIF
    PRINT
    PRINT "Ende mit einer Taste"
    WHILE INKEY$ = "" ! Warten auf Taste
      WEND
    ' --- Bis hier steht das Hauptprogramm ---
    VOID CloseLibrary(regbase%)
    LONG(mytask%+184) = oldwindowptr% ! Windowpointer restaurieren
  ENDIF
ELSE
  ALERT 0,"Library konnte nicht geöffnet werden",1," OK ",back
ENDIF

CLOSEW #2 ! Window und Screen schließen
CLOSES 2
END

FUNCTION colorrequester(colour%) ! Requester aufrufen
  m68%(0) = colour%
  m68%(14) = regbase%
  RCALL regbase%-90,m68%()
  RETURN m68%(0)
ENDFUNC
```

Farben einstellen im Programm

Am einfachsten geht's mit dem Farb-Requester aus der »req.library« von der Fish-Disk 419

Rahmenlose Fenster in Gfa 3.5

Wer versucht, rahmenlose Fenster mit Gfa-BASIC 3.5 zu öffnen, wird irgendwann feststellen, daß u.U. die Fenster-Titelleiste am oberen Bildrand auftaucht. Besonders unangenehm ist das, wenn man zuvor eine Titelgrafik lädt.

Das Ganze liegt daran, daß Gfa-BASIC automatisch einen Fenster-Titel vergibt. Abhilfe schafft ein kleiner Trick, der allerdings nicht ganz sauber ist: Nach Öffnen des Fensters wird der Pointer, der auf den Fenstertext zeigt, auf Null gesetzt. Der Haken an der Sache steckt wie immer im Detail:

Wenn man ein BASIC-Programm nun einfach beendet, oder das Fenster schließt, stürzt der Interpreter ab. Der alte Zeiger muß deshalb in einer Variablen gespeichert werden und vor dem Schliessen des Fensters wieder an die richtige Stelle gesetzt werden. Pro-

grammfehler müssen also unbedingt abgefangen werden. Das Programm »Borderless.gfa« zeigt, wie's geht.

```
' | Borderless-Windows mit Gfa-BASIC 3.5 |

window_flags = &H800 ! Borderless-Flag

ON ERROR GOSUB fehler
ON BREAK CONT.

OPENS 2,0,0,640,256,2,&H8000 ! Screen öffnen
OPENW #2,0,0,640,256,0,window_flags,2 ! Fenster öffnen
mywindow = WINDOW(2) ! Adresse Fenster-Struktur
oldtitel = LONG(mywindow+32) ! alten Titelpointer merken
LONG(mywindow+32) = 0 ! Titelpointer auf null
CLEARW #2 ! Fenster löschen
OPENW #3,20,100,450,80,0,4103,2 ! 2.Fenster öffnen
OPENW #2

PRINT
PRINT "Hier ein Fenster ohne Rahmen und ohne Titelleiste."
PRINT "Aktivierungs- und Menüfunktionen arbeiten wie gewohnt."
PRINT "Zum Beenden bitte beide Maustasten drücken"

OPENW #3
PRINT
PRINT "Das ist ein normales Fenster."

WHILE MOUSEK<>3
  WEND

prgende:
  LONG(mywindow+32) = oldtitel ! Titelpointer schreiben
  CLOSEW #2 ! Fenster und Screen
  CLOSEW #3 ! schliessen. Ende.
  CLOSES 2
  END

PROCEDURE fehler
  ALERT 0,"Programmfehler",1," Okay ",back
  RESUME prgende
RETURN
```

borderless.gfa

Mit einem Trick verschwindte auch in Gfa-BASIC die Titelleiste; aber man muß aufpassen

Wenn Sie andere Fensternummern als 2 für das Borderless-Window vergeben, müssen Sie die Argumente in den Zeilen, in denen die Fensternummer auftaucht, entsprechend verändern:

mywindow = window(n)

Das gilt natürlich auch für die Screen-Nummern. Ein Druck auf eine der Maustasten beendet das Programm.

F.Reinert/ub

MACHEN SIE MIT

Wollen Sie sich an der Rubrik »Tips & Tricks« im AMIGA-Magazin beteiligen? Schicken Sie uns Ihre Tips. Am besten, Sie senden uns Ihre Programme und Texte auf Diskette (ASCII-Format). Für Bastelanleitungen fügen Sie bitte einen Bauplan bei. Die Adresse:

AMIGA-Redaktion

Kennwort: Tips & Tricks

Markt & Technik Verlag AG

Hans-Pinsel-Str.2 8013

Haar bei München

Alle Einsendungen werden von uns geprüft. Wenn Ihr Beitrag in die engere Wahl für eine Veröffentlichung kommt, werden Sie benachrichtigt. Haben Sie auf jeden Fall ein wenig Geduld: Vom Zeitpunkt, an dem Ihr Beitrag die Redaktion erreicht, können bis zum Abdruck im AMIGA-Magazin oder Sonderheft zwei bis drei Monate vergehen.

Programmieren

Zaubersprache

Wenn Sie den Amiga in Assembler programmieren, steht Ihnen das ganze System offen. Assembler ist zwar »gefährlicher«, da fehleranfälliger, als andere Sprachen und auch aufwendiger, aber mit den richtigen Tricks spart man Programmierarbeit und man bekommt vor allem schnelle und leistungsfähige Programme.

Es grenzt an Zauberei – die Assembler-Tricks zeigen, wie Sie den Amiga wirklich bis ins letzte Bit ausreizen. Zum Einsatz kommen Seka, Devpac, Profimat und der OMA-Assembler. Ganz gleich, welchen Assembler Sie verwenden, alle Listings lassen sich selbstverständlich an jeden anpassen.

Leuchtdiode

Schon oft wurden in Zeitschriften Programme zum Ausschalten der Power-LED vorgestellt. Wie schaltet man aber die LED des Diskettenlaufwerks aus bzw. an?

Hier die Lösung für Assembler-Programmierer zum Anschalten:

```
LED EQU $BFD100
move.b #127, LED
move.b #119, LED
move.b #0, LED+512
```

Zum Ausschalten ersetzt man die letzte Zeile durch:

```
move.b #255, LED+512
```

Sean Durkin/ub

Assembler optimal

Hier ein paar Tips, wie Sie Assembler-Programme optimieren:

- Mit folgendem Befehl läßt sich die Addition einer vorzeichenbehafteten 16-Bit-Zahl zu einem Adreßregister (32 Bit; »add.l Dx.w,Ax«) einfacher realisieren, ohne das Datenregister auf 32 Bit zu erweitern:

```
lea (Ax,Dx.w),Ax
```

- Mit der Anweisung

```
addq.l #xx,Ax
```

kann man nur Zahlen bis 8 addieren. Mit einem anderen Befehl ist es jedoch möglich, bei gleicher Geschwindigkeit wie mit »addq« (8 Taktzyklen) vorzeichenbehaftete 16-Bit-Zahlen unmittelbar zu einem Adreßregister zu addieren:

```
lea d(Ax),Ax
```

- Um eine Zahl zu verdoppeln, gibt es mehrere Wege. Die langsamste Methode ist:

```
mulu #2,Dx
```

Schneller (8 Taktzyklen) geht's schon mit:

```
lsl #1,Dx
```

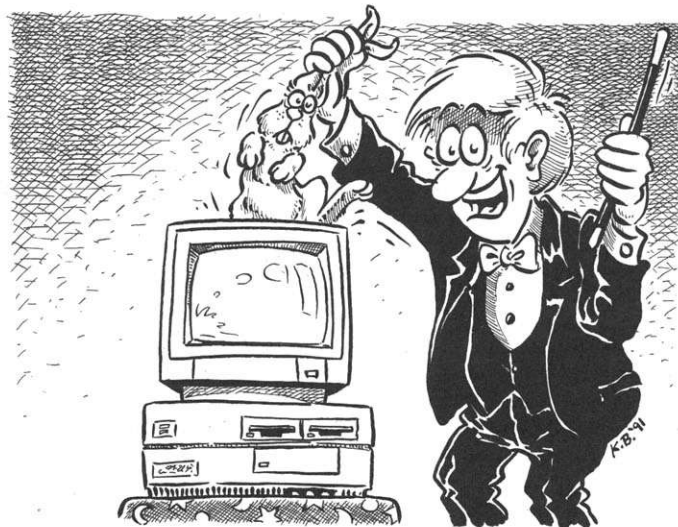
Die schnellste Methode ist jedoch (4 Taktzyklen):

```
add Dx,Dx
```

Dieser Befehl lohnt sich auch beim Multiplizieren mit 4, da die Dauer des »lsl«-Befehls von der Anzahl der Verschiebungen abhängt. Bei

```
lsl #2,Dx
```

wären es 10, die zweimalige Addition benötigt jedoch nur 8 Zyklen.



- In einer Schleife lohnt es sich in Bezug auf Geschwindigkeitsvorteile, wenn man keine unmittelbaren Befehle verwendet, sondern alle in der Schleife mehrmals benötigte Werte und die Zählvariable selbst vor der Schleife in Daten- oder Adreßregistern zwischenspeichert und dann die Register verwendet, da der Prozessor dann nicht mehr den Wert aus dem Speicher holen muß.

- Oft findet man in Programmen, daß bei absoluten Adressen mit »btst« Bits über 7 geprüft werden, z.B. das »Blitter-finished-Bit«:

```
btst #14,$dff002
```

Das ist eigentlich falsch, da bei den Bit-Befehlen bei Anwendung auf eine absolute Adresse nur Zugriff auf Bit 0 bis 7 erlaubt ist. Im obigen Beispiel müßte der Befehl also lauten:

```
btst #6,$dff001
```

Der Befehl funktioniert trotzdem, weil die meisten Assembler

```
btst #14,Adresse
```

automatisch umformen in:

```
btst #6,(Adresse-1)
```

Ein Fehler wäre es jedoch, wenn man mit

```
btst #5,$dff001
```

prüfen wollte, ob Sprite-DMA erlaubt ist. Dieser Befehl würde nämlich das Blitter-Zero-Bit testen. Der korrekte Befehl würde lauten:

```
btst #5,$dff002
```

Christian Seiler/ub

Ohne Preferences

Die Preferences dienen u.a.:

- zum Einstellen des Druckertreibers,
- der seriellen Schnittstelle,
- des Grafikdrucks,
- der Workbenchfarben
- oder der Zahl der Zeichen pro Zeile (80 oder 60)

Die Preferences belegen allerdings 56 628 Byte Speicher. Das kann knapp werden, wenn man keine Speichererweiterung im Amiga 500 hat und z.B. der BASIC-Interpreter sowie ein längeres Programm geladen ist. Und außerdem muß man die Preferences immer laden, was Zeit kostet.

Es geht zum Glück auch ohne Preferences, wenn man etwas an den Systemeinstellungen verändern möchte, wie das nächste Beispiel zeigt: Das Programm »Z_Z.asm« ist nur 332 Byte lang – und außerdem schneller als die Preferences. »Z_Z.asm« dient dazu, eine bestimmte Einstellung in den Preferences zu ändern, in unserem Fall die Einstellung der Zahl der Zeichen/Zeile (60/80).

Das erste Byte der Preferences-Struktur legt die Zahl der Zeichen pro Zeile fest, in der Texte auf dem Bildschirm erscheinen. Mit der Routine »GetPrefs()« (Exec-Library) prüft unser Programm den aktuellen Modus und speichert den gegenteiligen per »SetPrefs()«.

Die »system-configuration« im Verzeichnis »Devs:« wird durch die Umstellung nicht verändert. Nach Reset ist die alte Fassung aktiv. Es handelt sich bei »Z_Z.asm« um eine Devpac-Assembler-Routine. Speichern Sie das ausführbare Programm im C-Verzeichnis. Wenn Sie es per ICONX mit einem Icon in die Workbench einführen, ändern Sie durch Anklicken des Symbols den Zeichenmodus.

In einer Batch-Datei oder vom CLI bzw. der Shell aus rufen Sie einfach das Programm mit seinem Namen auf:

Z_Z

Ein Benutzen im CLI hat allerdings wenig Sinn, da sich der Modus eines laufenden CLI-Windos so nicht ändern läßt, es sei denn, man eröffnet einen neuen Task.

Wolfgang Wallat/ub

```

Z_Z

opt c+,d+

movea.l $4,a6
movea.l #int,a1
moveq   #$0,d0

jsr     -$0228(a6) ' EXEC open Library
tst.l   d0
beq.s   no_int

move.l   d0,IntBase
movea.l   d0,a6 ' IntBase
movea.l   #Puffer,a0 ' Speicher bereitstellen
moveq     #$1,d0 ' 1 Byte lang

jsr     -$84(a6) ' INT GetPrefs
move.b   Puffer,d2 ' Inhalt Puffer
cmp.b    #$8,d2 ' mit 8 vergleichen
beq.s    breit

move.b   #$8,Puffer ' <>8 dann 8 speichern
bra.s    set

breit
move.b   #$9,Puffer ' 8 dann 9 speichern

set
movea.l   #Puffer,a0
moveq     #$1,d0
move.l    #$-1,d1
jsr     -$144(a6) ' INT SetPrefs

ausstieg
movea.l   $4,a6
movea.l   IntBase,a1
jsr     -$019e(a6) ' EXEC close Library

no_int
moveq     #0,d0
rts

* Variablen
int      dc.b 'intuition.library',0
IntBase  dc.l 0
Puffer   dc.b 0
    
```

Z_Z.asm

Assembler-Programm (Dvpac) um die Zahl der Zeichen (60/80) über die die Preferences zu wechseln

Stringvergleich in Assembler

Mit der Funktion »strcmp()« besteht in C die Möglichkeit, zwei Strings (Zeichenketten) miteinander zu vergleichen. In Assembler heißt es, so etwas selbst zu programmieren.

»Strcmp.s« ist eine kurze und schnelle Assembler-Routine zum Vergleich von Strings. Dabei werden die Strings jedoch nur auf Gleichheit getestet. Eine Unterscheidung in »größer« und »kleiner« wird nicht getroffen, kann aber von jedem Assembler-Programmierer eingebaut werden. Die Adressen der Strings werden der Funktion in den Adreßregistern A0 und A1 übergeben. Der Aufruf erfolgt mit:

jsr strcmp

Das Ergebnis wird über das Datenregister D0 in Form eines Boole-leanwerts zurückgegeben (TRUE oder FALSE, entspricht 1 bzw. 0) und kann mit

```

tst.b d0
bne Strings_gleich
Strings_verschieden:
    
```

```

    .....
    .....
Strings_gleich:
    .....
    .....
    
```

abgefragt werden. »Strings_gleich« steht dabei für den Programmteil, der abgearbeitet werden soll, wenn die Strings gleich sind. Ansonsten wird bei »Strings_verschieden« fortgefahren.

Christof Brühann/ik

```

; strcmp - testet zwei Strings auf Gleichheit

; Eingabe: a0 - Zeiger auf string1
;          a1 - Zeiger auf string2

; Ausgabe: d0 - TRUE (1) für Gleichheit, sonst FALSE (0)

strcmp:
move.b (a0),d0
cmp.b (a1)+,d0 ; vergleiche zwei Byte der Strings
bne not_equal  ; verschieden -> mit 0 in d0 zurück

tst.b (a0)+    ; waren die Bytes die letzten?
bne strcmp     ; nein -> noch ein Durchgang

move.l #1,d0   ; Strings gleich - mit 1 in d0 zurück
rts

not_equal:
sub.l d0,d0

rts
    
```

»strcmp.asm«

Hiermit testet man zwei Strings auf Gleichheit

TIPS ZU ASSEMBLER AUF EINEN BLICK

Tip	Thema	Seite
Leuchtdiode	Hardwareprogrammierung	24
Assembler optimal	Programmierungsmethoden	24
Ohne Preferences	Systemeinstellungen	24
Stringvergleich	Programmiertechnik	25
Install RAM-Disk	RAM-Disk per Programm füllen	26
Für alle Fälle gewappnet	»system-configuration«	27
Vier für Strings	String-Routinen	28
Pseudo-Laufzeitsystem	Abbruch mit <Ctrl C>	29
Alle Offsets in Assembler	Programmierhilfe	30
Schnelles Pixelsetzen	Grafikfunktion	31
Bildschirmsplitting	Copperprogramm	32

Install RAM-Disk

Jeder kennt das: Man arbeitet im CLI und braucht öfters Befehle wie DIR, COPY etc. Das Alles kostet natürlich viel Zeit, weil der Amiga die verwendeten Befehle immer wieder von Diskette lädt. Weil's schneller geht, kopiert man die Befehle in die RAM-Disk und weist den Amiga mit ASSIGN oder PATH an, die CLI-Befehle in der RAM-Disk zu suchen, z.B. mit folgender Batch-Datei:

```
makedir ram:c
path ram:c add
SYS:c/copy SYS:c/copy ram:c
copy SYS:c/dir ram:c
copy SYS:c/list ram:c
copy SYS:c/info ram:c
...
```

RAM-Disk.bat

Sind die CLI-Befehle in der RAM-Disk, geht's um einiges schneller

Startet man seinen Amiga jedoch öfters neu, kostet auch die häufige Ausführung des COPY-Befehls

```
copy SYS:c/dir ram:c...
```

eine Menge Zeit.

Nutzen Sie hierfür »Install RAM-Disk.asm«. Es ist ein Assembler-Programm, das automatisch, falls noch nicht vorhanden, die RAM-Disk installiert, um dann die gewünschten Files hineinzuschreiben.

Es sind nur das asemblierte Programm und der »RAM-Handler« im »«-Verzeichnis der System-Diskette nötig. Falls letztere Datei fehlt, erscheint eine Fehlermeldung.

Diese Methode, die RAM-Disk mit Befehlen zu füllen, ist bedeutend schneller und komfortabler als die zuerst beschriebene. Das Programm wurde mit ASMone V1.0 erstellt. Passen Sie das Listing gegebenenfalls an an Ihren Assembler an.

David Szpak/ub

```
openlibrary = -408 ; Offsets Exec-Funktionen
closelibrary = -414
open = -30 ; Offsets DOS-Funktionen
close = -36
read = -42
write = -48
output = -60
delete = -72
prglength1 = 7856 ; Hier File-Laengen angeben
prglength2 = 2112
prglength3 = 7840
```

```
Prog_start:
lea dosname(pc),a1 ; dos.library oeffnen
move.l $4.w,a6
jsr openlibrary(a6)
move.l d0,dosbase
move.l d0,a6
beg.w return ; Fehler beim Oeffnen
move.l #head,d2
move.l #headsize,d3
bsr.w write_text
move.l #handlername,d1
move.l #1005,d2 ; Modus = lesen
jsr open(a6)
tst.l d0
beg.w ram_error ; Fehler aufgetreten
move.l #ram_disk,d1
jsr delete(a6)
move.l #filename1,d1 ; für eigene Zwecke diesen Teil
move.l #prglength1,d3 ; entsprechend erweitern. Hier
move.l #data1,datapointer ; für drei Dateien als Beispiel:
bsr.s Copy_file
move.l #text1,d2
move.l #textsize1,d3
bsr.w write_text
```

```
move.l #filename2,d1
move.l #prglength2,d3
move.l #data2,datapointer
bsr.s Copy_file
move.l #text2,d2
move.l #textsize2,d3
bsr.s write_text
move.l #filename3,d1
move.l #prglength3,d3
move.l #data3,datapointer
bsr.s Copy_file
move.l #text3,d2
move.l #textsize3,d3
bsr.s write_text
Return:
move.l $4.w,a6
move.l dosbase(pc),a1
jsr closelibrary(a6)
rts
Ram_error:
move.l #ram_errortext,d2
move.l #errortextsize,d3
bsr.s write_text
bra.s return
Copy_file:
move.l #1006,d2 ; Modus = schreiben
jsr open(a6)
move.l d0,d4 ; Zeiger auf Filehandle-Struktur retten
move.l d0,d1
move.l datapointer(pc),d2
jsr write(a6)
move.l d4,d1 ; und zurueckschreiben
jsr close(a6)
rts
Write_text:
jsr output(a6) ; CLI-Ausgabekanal ermitteln
move.l d0,d1
jsr write(a6) ; und Text ausgeben
rts
ram_errortext:
dc.b "***Fehler** beim Installieren der RAM-Disk"
dc.b " Kopieren Sie '", $9b, "0:33mRAM-Handler", $9b, "031m" ins
dc.b " '", $9b, "0:33m'", $9b, "031m" directory!", 10, 10
head:
dc.b 10, $9b, "1:31:42mInstall RAM-Disk"
dc.b $9b, "0:31m", 10, 10
text1:
dc.b "File '", $9b, "0:33mdir", $9b, "0:31m" has been installed"
dc.b " successfully!", 10
text2:
dc.b "File '", $9b, "0:33mtype", $9b, "0:31m" has been installed"
dc.b " successfully!", 10
text3:
dc.b "File '", $9b, "0:33mcopy", $9b, "0:31m" has been installed"
dc.b " successfully!", 10, 10
text4:
handlername:
dc.b "sys:l/Ram-Handler", 0
even
dosbase: dc.l 0
datapointer: dc.l 0
dosname: dc.b "dos.library", 0
ram_disk: dc.b "ram:", 0
filename1: dc.b "ram:dir", 0 ; destination filenames
filename2: dc.b "ram:type", 0
filename3: dc.b "ram:copy", 0
data1:
incbin "df0:c/dir" ; source-file-names
data2:
incbin "df0:c/type" ; mit seka einfach die files mit
data3: ; ri an 'data1', 'data2' etc. laden.
incbin "df0:c/copy" ; im source 'blk.b prglength1,0'
; 'blk.b prglength2,0' etc. schreiben.
errortextsize = head-ram_errortext
headsize = text1-head ; hier Labels angeben um
textsize1 = text2-text1 ; Textlaenge zu berechnen...
textsize2 = text3-text2
textsize3 = text4-text3
```

RAM-Disk.asm

Sind die CLI-Befehle in der RAM-Disk, geht's um einiges schneller

Für alle Fälle gewappnet

Wenn man häufig mit unterschiedlichen Konfigurationen, d.h. mit unterschiedlichen Druckereinstellungen etc. arbeitet, muß man bei jeder Änderung die Preferences laden. Das entfällt mit Hilfe des folgenden Programms: »LoadSys.asm« lädt und aktiviert »system-configuration«-kompatible Dateien. Es stellt eine Optimierung zum Trick »Ohne Preferences« auf Seite 24 dar.

```
; Eingabe:
; Loadsys      ... lädt devs:system-configuration
; Loadsys FILENAME ... lädt system-configuration File
; Loadsys      ... gibt (c)opyright Meldung aus
; Definition für Libraries
Exechbase = 4
; Exec Offsets
OldOpenLibrary = -408
CloseLibrary = -414
AllocMem = -198
FreeMem = -210
; DOS-Offsets
Open = -30
Close = -36
Read = -42
Write = -48
Input = -54
Output = -60
Lock = -84
Unlock = -90
Examine = -102
Exit = -144
; Intuition-Offsets
SetPrefs = -324
GO:
    bsr.w START
    bsr.w OPENALL
    bsr.w HoleInfo
    bsr.w OpenFile
    bsr.w SetP
    bsr.w Ausgang
START
    cmp.w #1,d0          ; Eingabe länger als
    bhi.s LAENGER2       ; ein Zeichen?
    lea default,a5       ; Wenn ja,
    bra.s retsys         ; Default-Wert verwenden
LAENGER2
    cmp.w #2,d0          ; Eingabe länger als
    bhi.s SPACES         ; zwei Zeichen
    cmp.b #"?",(a0)       ; Eingabe = ?
    bne.s SPACES
    lea EM8,A2
    moveq #1,d7
    bra.s retsys
SPACES
    move.l a0,a5
    lea -$01(a0,d0.w),a0
MARK
    cmp.b #$20,-(a0)      ; $20 = SPACE
    dbne d0,MARK
    addq.w #1,a0
    clr.b (a0)
    retsys
    rts
OPENALL
    move.l Exechbase,a6
    lea Dosname,a1
    moveq #0,d0
    jsr OldOpenLibrary(a6)
    move.l d0,Dosbase
    exg d0,a6
    jsr Output(a6)        ; Output()
    move.l d0,OUT
    BEQ.L Ausgabe
    cmp.l #1,d7
    BEQ.L Ausgabe
    move.l Exechbase,a6
    lea EM1,a2
    lea Intrname,a1
    moveq #0,d0
    jsr OldOpenLibrary(a6)
    move.l d0,Intbase
```

```
BEQ.L Ausgabe
rts
HoleInfo
    move.l Exechbase,a6
    move.l #260,d0
    move.l d0,d6
    moveq #0,d1
    lea EM3,a2
    suba.l a4,a4
    jsr Allocmem(a6) ; AllocMem(byteSize,requirements) (D0/D1)
    move.l d0,a4
    tst.l d0
    BEQ.L Ausgabe
    move.l a5,d1
    move.l #-2,d2          ; Type = Lesen
    move.l Dosbase,a6
    lea EM4,a2
    jsr Lock(a6)          ; Lock(name,type) (D1/D2)
    move.l d0,d4
    BEQ.L Ausgabe
    move.l d4,d1
    move.l a4,d2
    jsr Examine(a6)       ; Examine(lock,fileInfoBlock) (D1/D2)
    tst.l d0
    BEQ.s Ausgabe
    lea EM7,a2
    cmpi.l #232,124(a4)   ; File 232 Byte lang?
    BNE.s Ausgabe
    move.l d4,d1
    jsr Unlock(a6)        ; Unlock(lock) (D1)
    rts
OpenFile
    move.l a5,d1
    move.l #1005,d2       ; Mode_old
    move.l Dosbase,a6
    lea EM4,a2
    jsr Open(a6)          ; Open(name,accessMode) (D1/D2)
    move.l d0,d4
    BEQ.s Ausgabe
    move.l d4,d1
    move.l a4,d2
    move.w #232,d3
    lea EM5,a2
    jsr Read(a6)          ; Read(file,buffer,length) (D1/D2/D3)
    move.l d0,d5
    move.l d4,d1
    move.l Dosbase,a6
    jsr Close(a6)         ; Close(file) (D1)
    tst.l d5
    beq.s Ausgabe
    rts
SetP
    move.l Intbase,a6
    move.l a4,a0
    move.w #232,d0
    moveq #1,d1
    jsr Setprefs(a6) ; SetPrefs(preferences,size,flag) (A0,D0/D1)
    rts
Ausgabe
    bsr cpy
    move.l Dosbase,a6
    move.l OUT,d1
    move.l #EMDAT,d2
    move.l #31,d3
    jsr Write(a6)         ; Write(file,buffer,length) (D1/D2/D3)
    rts
Ausgang
    move.l Exechbase,a6
    cmp.l #1,d7
    beq.s QUIT
    cmp.l #0,a4           ; Wurde Speicher reserviert?
    beq.s KeinsPEICHER    exg a4,a1
    move.l d6,d0
    jsr FreeMem(a6)       ; FreeMem(memoryBlock,byteSize) (A1,D0)
    rts
KeinsPEICHER
    move.l Intbase,a1
    jsr CloseLibrary(a6)
    move.l Dosbase,a6
    moveq #0,d1
    jsr Exit(a6)          ; Exit(returnCode) (D1)
    rts
CPY:
    moveq #21,d0          ; 22 Byte kopieren
    lea EMDAT,a3
    CPYM:
    move.b (a2)+,(a3)+
```



```

dbra d0,CPYM
move.b #5a,(a3)+ ; mit 5a abschließen
rts
default dc.b "devs:system-configuration",0
; Fehler & (c)opyright Meldungen
EMDAT blk.b 32
EM1 dc.b "Get file handle failed"
EM2 dc.b "Open Library failed "
EM3 dc.b "AllocMem failed "
EM4 dc.b "File not found "
EM5 dc.b "Open file failed "
EM6 dc.b "Read file failed "
EM7 dc.b "Not a Preferences file"
EM8 dc.b "LoadSys by Herwig H. "
; Definitionen für Libs
Dosname dc.b "dos.library",0
Dosbase dc.l 0
Intname dc.b "intuition.library",0
Intbase dc.l 0
OUT dc.l 0

```

LoadSys.asm

»system-configuration« ändern mit einem Aufruf

Gehen Sie so vor:

Laden Sie Preferences, stellen Sie die gewünschte Konfiguration ein und speichern Sie sie (Save). Danach erstellen Sie eine Kopie der Datei »system-configuration« im Verzeichnis »Devs«, z.B. mit dem Namen »conv1«. In der »system-configuration« stehen alle Parameter, die Sie mit Preferences einstellen:

```
copy devs:system-configuration to devs:conv1
```

Auf dieselbe Art können Sie mehrere Konfigurationen erstellen, wobei Sie jeder Datei einen neuen Namen geben (»conv2«, »conv3«, usw.).

Das Programm »loadsys« benutzt man dann so, um eine bestimmte Konfiguration zu aktivieren:

```
Loadsys devs:conv1
```

Damit laden und initialisieren Sie die in »conv1« gespeicherte Konfiguration.

Das Programm wurde mit MasterSeka V1.70 geschrieben.

Herwig Höller/ub

Vier für Strings

Im folgenden Artikel finden Sie vier nützliche Assembler-Routinen, die Ihnen die Handhabung von Zeichenketten (Strings) in Assembler erleichtern.

Die vier Assembler-Routinen, die wir Ihnen anbieten sind:

- Ucase: Wandelt Klein- in Großbuchstaben
 - String_Length: ermittelt Länge der Zeichenkette
 - Instr: sucht String2 in String1
 - Replace: ersetzt (Teil von) String1 durch (einen Teil von) String2
- Die Routinen wurden mit dem Assembler Devpac 2.0 getestet. Dieser Assembler unterstützt lokale Labels, von denen hier reger Gebrauch gemacht wird, da so das doppelte Definieren eines Labels durch diese Unterprogramme ausgeschlossen bleibt.

Sollte es bei bestimmten Assemblern Probleme geben, müssen Sie eigene Labels im Listing einsetzen.

Bei allen Routinen ist Voraussetzung, daß die zu übergebenden Strings mit einem Null-Byte enden.

C.Straka/ub

```

; ** Ucase - wandelt alle Kleinbuchstaben innerhalb **
; ** eines Strings in Großbuchstaben **
; Hinweis: Routine arbeitet analog zur BASIC-Anweisung UCASE$
; Stringadresse -> A0
Ucase:
move.l a0,-(sp)
.loop
cmp.b #0,(a0) ; Null-Byte des Strings?

```

```

beq .ende ; wenn ja, fertig
cmp.b #97,(a0) ; Zeichen >= 'a' ...
bmi .weiter
cmp.b #122,(a0) ; ... und <= 'z'
bpl .weiter
bclr #5,(a0) ; dann wandeln
.weiter
move.b (a0),(a0)+ ; nächstes Zeichen
bra .loop
.ende move.l (sp)+,a0
rts

```

Ucase.asm

wandelt alle Kleinbuchstaben in Großbuchstaben

```

; ** String_Length - ermittelt Länge einer Zeichenkette **
; Hinweis: Routine arbeitet analog zur BASIC-Anweisung LEN
; Stringadresse -> A0
; Stringlänge -> D0.L
String_Length:
move.l a0,-(sp)
moveq #0,d0 ; D0 löschen
.loop
addq.l #1,d0 ; D0 um eins erhöhen
cmp.b #0,(a0)+ ; Null-Byte des Strings?
bne .loop ; wenn nicht, nächstes Zeichen
subq.l #1,d0 ; wenn ja, dann D0-1
move.l (sp)+,a0
rts

```

String_Length.asm

ermittelt Länge einer Zeichenkette

```

; ** Replace - ersetzt [einen Teil von] String1 durch **
; ** [einen Teil von] String2 **
; Hinweis: Routine arbeitet analog zur BASIC-Anweisung MID$
; Stringadresse1; Stringadresse2; Position, ab der ersetzt
; werden soll; Anzahl der zu ersetzenden Zeichen; Stringlänge
; von String1 -> A0,A1,D0.L,D1.L,D2.L
Replace:
movem d1-d2/a0-a1,-(sp)
sub.l d0,d2 ; Position > Stringlänge?
bmi .ende ; ja, fertig, falsche Eingabe
subq.l #1,d0
subq.l #1,d1
add.l d0,a0 ; Adresse der Startposition in A0
.loop
cmp.b #0,(a1) ; Null-Byte von String2?
beq .ende ; ja, dann fertig, ersetzt
cmp.b #0,(a0) ; Null-Byte von String1?
beq .ende ; ja, dann fertig, ersetzt
move.b (a1)+,(a0)+ ; ersetzen
dbra d1,.loop ; noch weiter ersetzen?!
.ende
movem (sp)+,d1-d2/a0-a1
rts

```

Replace.asm

ersetzt [Teil von] String1 durch [Teil von] String2

```

; ** Instr - ermittelt Position des ersten Auftretens **
; ** von String2 innerhalb von String1 **
; Hinweis: Routine arbeitet analog zur BASIC-Anweisung INSTR
; Stringadresse1 --> A0
; Stringadresse2 --> A1
; Stringlänge von String1 --> D0.L
; Startpos. des zu durchsuchenden Strings ($1-$FFFFFFF) --> D1.L
; Position von String2 innerhalb von String1 --> D0.L

```

```
Instr:
movem.l d1/a0-a3,-(sp)
sub.l d1,d0 ; ist Startposition > Stringlänge
bmi .ende ; dann fertig, da falsche Eingabe
move.l d1,d0 ; Startposition in D0
moveq #0,d1 ; D1 löschen
move.l a0,a2 ; A0 in A2 merken
move.l a1,a3 ; A1 in A3 merken
subq.l #1,d0
.begin
add.l d0,a0 ; Adresse der Startposition in A0
.loop
addq.l #1,d0 ; D0 um eins erhöhen
cmp.b #0,(a0) ; Null-Byte des Strings?
beq .ende ; wenn ja, fertig, nicht gefunden
move.b (a0)+,d1
cmp.b (a1),d1 ; ist (a1) = (a0)
bne .loop ; nein, dann weiter suchen
addq.l #1,a1 ; ja, nächstes Zeich. v. String2 finden
.loop2
cmp.b #0,(a1) ; ist es das Null-Byte?
beq .ende2 ; dann fertig, gefunden
cmp.b #0,(a0) ; ist schon das Null-Byte v. String2 da?
beq .ende ; dann fertig, nicht gefunden
move.b (a0)+,d1
cmp.b (a1)+,d1 ; vergl. Stringbyte1 mit Stringbyte2
beq .loop2 ; wenn gleich, dann überprüfe weiter
move.l a2,a0 ; wenn ungleich, setze A0 und A1 zurück
move.l a3,a1
bra .begin ; und versuche Glück im noch nicht
; untersuchten Teil von String1
.ende
moveq #0,d0 ; nicht gefunden: D0 = 0
.ende2
movem.l (sp)+,d1/a0-a3 ; gefunden: Position in D0
rts
```

Instr.asm

sucht String2 in String 1

```
Execbase = 4
FindTask = -294
Open:
move.l Execbase,a6
sub.l a1,a1 ; eigenen
jsr FindTask(a6) ; Task suchen
move.l d0,a0
move.l #$8000,d0 ; <Ctrl f> als Break
bsr break
; fuer <Ctrl d> = $1000, e = $2000, c = $4000, f = $8000
; Start nur als »Process«
; ab hier Ihr Programm
Main:
move.b #2,$bfe001 ; LED an
bchg #1,$bfe001 ; LED aus
bra Main
; Exception-Signale setzen, a0 Task, d0 Signal
break:
move.l d0,30(a0) ; Break-Signal
move.l #Code,42(a0) ; &-Code setzen
rts
; auf dem Stack befinden sich noch: Exception-Adresse, PC der
; unterbrochenen Zeile, SR, D0-D7, A0-A6 (ST um 74 Byte grosser)
Code:
move.l Execbase,a6
sub.l a1,a1 ; eigenen
jsr FindTask(a6) ; Task suchen
move.l d0,a0
move.l 176(a0),a7 ; Return-Adresse holen und
suba #4,a7 ; Stack auf Ruecksprung setzen
rts ; Ende
; wenn Programm weiterlaufen soll, nur rts, d.h.D0 nicht aendern,
; da die Signale drinstehen
```

Break.asm

Abbruch von Assembler-Programmen mit <Ctrl c>

Pseudo-Laufzeitsystem

Fast jede Hochsprache hat ein Laufzeitsystem, um z.B.:

- Fehler während des Programmablaufs abzufangen,
- die Tastatur auf Unterbrechungen (<Ctrl C>) zu überprüfen, oder
- Bibliotheken automatisch zu öffnen und am Ende des Programms wieder zu schließen.

Bei Assemblern fehlt ein solches System, doch gibt es eine einfache und universell einsetzbare Methode, um auch Assemblerprogramme mit einem Quasi-Laufzeitsystem auszustatten. Ein Beispielprogramm zeigt, was man machen muß, damit man Programme mit <Ctrl C> abbrechen kann:

Erklärung: In der »Process«-Struktur eines Tasks gibt es eine »Except«-Struktur, die den Task bei den bestimmten Signalen unterbricht. Das macht sich unser Pseudo-Laufzeitsystem zunutze: Es setzt das Signal zur Unterbrechung im »tc_SigExcept«-Feld fest und setzt im »tc_ExceptCode«-Feld des Tasks einen Zeiger auf den Programmcode, der nach einer Exception ausgeführt werden soll.

Drückt man nun z.B. <Ctrl F>, hält der Amiga das laufende Programm an und legt folgendes auf dem Stack ab:

- die ExecHandler-Adresse,
- den Programm-Counter,
- die letzte Adresse im Programm,
- das Status-Register (SR),
- D0 bis D7 und
- A0 bis A6.

Anschließend setzt der Amiga seine Arbeit ab der Adresse »Code« fort, wobei das unterbrochene Programm beendet wird.

Sie können in der Unterbrechungsroutine natürlich eigene Routinen einbauen, z.B. den Stack auswerten und mit »rts« an die Stelle zurückkehren, in der Ihr Programm unterbrochen wurde.

ub

SIZEWINDOW & SizesMaliWindow

»BigWindow.asm« dient zum Vergrößern eines CLI/Shell-Fensters auf seine maximale Größe. Programme dieser Art gibt es schon einige, unsere Variante ist ein besonderer Vertreter seiner Spezies:

- a.) Das ausführbare Programm belegt wenig Speicher: 312 Byte. Viel kleinere Versionen dürfte es bei dieser Art Programme kaum geben. Man könnte es zwar ca. noch um 30 Byte reduzieren, darunter würde aber die Verständlichkeit des Quelltextes leiden.
- b.) »BigWindow« verwendet sowohl die (»Intuition«-) Systemfunktion »MoveWindow()« als auch »SizeWindow«. Wenn sich ein Fenster nicht in der richtigen Position befindet, bevor man es vergrößert, korrigiert »BigWindow« die Lage automatisch; man muß also nicht nach Indien fahren, um dort zu meditieren.

```
; BigWindow.s bzw. SmallWindow.s
; Ordentliche (!) Kurzversion des Window-Vergrößerungsbefehls
; [312 Byte]
; verschiebt Fenster nach (0,0) [MoveWindow] und
; vergrößert es auf (640,256) [SizeWindow] bzw.
; *** verkleinert es auf (100,30) ***
; benötigte System-Funktionen:

; exec.library
Execbase = 4
Openlibrary = -408
Closelibrary = -414

; intuition.library
MoveWindow = -168 ; MoveWindow(window,dx,dy) (A0,D0/D1)
SizeWindow = -288 ; SizeWindow(window,dx,dy) (A0,D0/D1)
Beginn:
movem.l d0-d7/a0-a6,-(SP) ; Register auf Stapel
move.l Execbase,a6
lea INtname,a1
jsr Openlibrary(a6) ; Öffnen von INTBASE
move.l d0,IntBase
move.l vd0,a6 ; INTBASE nach a6
```

```

move.l $34(a6),A1
MOVE.L A1,WINDOW
move.l window,a1
move.w $06(a1),a2 ; Y-Koordinate der linken oberen Ecke
move.w a2,OBEIN
move.w $04(a1),a2 ; X-Koordinate der linken oberen Ecke
move.w a2,LINKS
move.w $08(a1),a2 ; Breite des aktuellen Windows
move.w a2,BREITE
move.w $0a(a1),a2 ; Höhe des aktuellen Windows
move.w a2,HOEHE

CODE:
moveq #0,d1 ; Soll X-Koordinate der linken oberen Ecke
moveq #0,d0 ; Soll Y-Koordinate der linken oberen Ecke
move.l window,a0
sub.w LINKS,d0
sub.w OBEIN,d1
jsr Movewindow(a6) ; Window wird bewegt
move.l Window,a0
move.l #640,d0 ; Breite des aktuellen Windows
; ** move.l #100,d0 ; Breite des aktuellen Windows (small) **
sub.w BREITE,d0
move.w #256,d1 ; Höhe des aktuellen Windows
; ** move.w #30,d1 ; Höhe des aktuellen Windows (small) **
sub.w HOEHE,d1
jsr sizewindow(a6) ; vergrößere (-kleinere) Fenster
move.l Execbase,a6
move.l INTBASE,a1
jsr Closeslibrary(a6) ; Schließen von Intuition
movem.l (sp)+,d0-d7/a0-a6 ; Register vom Stapel
rts ; Return To System

INTNAME: dc.b "intuition.library",0
INTBASE: dc.l 0
WINDOW: dc.l 0
OBEIN: dc.w 0 ; Y-Koordinate der linken oberen Ecke
LINKS: dc.w 0 ; X-Koordinate der linken oberen Ecke
BREITE: dc.w 0 ; Breite des aktuellen Windows
HOEHE: dc.w 0 ; Höhe des aktuellen Windows

```

BigWindow.s

Vergrößern Sie ein CLI-Fenster mit einem Befehl

Rufen Sie das Programm übers CLI oder die Shell mit seinem Namen auf, vergrößert es das aktuelle Fenster. Das Programm (Seka-Assembler) sollte im Chip-Memory abgelegt sein.

In manchen Zeilen sind im Listing Alternativen angegeben. Wenn Sie diese einsetzen, erhalten Sie ein Programm, um das aktuelle Fenster zu verkleinern. Eine Anwendung ist z.B. wenn Sie mit der Workbench arbeiten möchten, aber das CLI-Fenster beim Start offenlassen wollen. Damit das CLI-Fenster nicht auf dem Bildschirm stört, verkleinern Sie es einfach auf seine minimale Größe. *ub*

Alle Offsets für Assembler

Für Assembler-Programmierer, die häufig Systemroutinen benutzen, ist es oft mühsam, die Offsets für die Einsprünge in die Libraries zu berechnen beziehungsweise abzutippen; und nicht jeder Assembler stellt die Offsets in Form von Include-Dateien zur Verfügung. Es gibt einen relativ einfachen Weg, sich solche Include-Dateien selber anzufertigen: Man kann die FD-Dateien auf der Extras-Diskette

```

INPUT "FD-Datei (mit Pfad)",Source$
INPUT "Ziel-Datei (ebenfalls mit Pfad)",Dest$

OPEN "I",1,Source$
OPEN "O",2,Dest$

WHILE NOT(EOF(1))
Zeile$ = "" : z$ = ""
WHILE z$ <> CHR$(10) AND NOT (EOF(1))
z$ = INPUT$(1,1)
Zeile$ = Zeile$+z$
WEND
z$ = LEFT$(Zeile$,1)
IF z$ <> "*" AND z$ <> "#" AND LEN(Zeile$)>1 THEN

```

```

PRINT#2,LEFT$(Zeile$,INSTR(Zeile$,"(")-1);"EQU ";counter
counter = counter-6
ELSE
IF z$ = "*" THEN PRINT#2,";";Zeile$
IF z$ = "#" AND MID$(Zeile$,3,4) = "bias" THEN
counter = -VAL(MID$(Zeile$,8,2))
END IF
WEND
CLOSE 1
CLOSE 2

```

Offset.bas

BASIC ist auch für Assembler-Programmierer nützlich

kette in ein assemblerlesliches File verwandeln. Das BASIC-Programm »Offset.bas« zeigt, wie man eine solche Konvertierung vornimmt. Die Einbindung der Offsets in eigene Programme sollte dann jeder Programmierer selbst vornehmen können. *ub*

Schnelles Pixelsetzen

Manchem Assemblerprogrammierer ist der Aufruf der Prozedur »WritePixel()« aus der »graphics.library« zu langsam. Schneller ist es, direkt in die Bitmap zu zeichnen. Zum Setzen eines Punkts in einer Bitplane muß das Byte ermittelt werden, in dem das Bit, und damit der Punkt, gesetzt werden soll. Dazu dividieren Sie die x-Koordinate durch acht; die Breite der Bitplane (in Byte) multiplizieren Sie mit der y-Koordinate; das Produkt addieren Sie zur x-Koordinate/8. Danach sind noch die übrigen 7 Bit (Bildpunkte) des Byte auszumas-kieren, damit Sie nicht mehrere Bits auf einmal setzen. Schließlich müssen Sie das verbleibende Bit setzen. Eine Routine, die Ihnen einen Teil der Aufgabe abnimmt zeigt das Listing »SetPixel.asm«.

```

; A0: Zeiger auf die Bitplane
; D0: x-Koordinate (Langwort)
; D1: y-Koordinate (Langwort)
; D2: Breite der Bitplane in Byte
SetPixel divu.w #8,d0 ; x-Koord./8 -> Byte des Pixels
mulu d2,d1 ; Zeile mit Anzahl Byte/Zeile multiplizieren
add.w d0,d1 ; = zu addierender Offset
swap d0 ; obere Hälfte ausmaskieren
moveq #7,d3 ; 8 Bits/Byte
sub.w d0,d3 ; unpassende Bits ausmaskieren
bset d3,(a0,d1.w) ; Pixel (Bit) setzen
rts ; und zurück!

```

SetPixel.asm

Schneller als die »graphics.library«

»Setpixel.asm« erwartet folgendes in den entsprechenden Registern:

- A0: Zeiger auf die Bitplane. Die Zeiger für die einzelnen Bitplanes befinden sich im Viewport ab Adresse (Offset) \$C0 (siehe Tabelle).
- D0: x-Koordinate
- D1: y-Koordinate
- D2: Breite der Bitplane (in Byte).

Offsets: Bitplanes im ViewPort

Offset	Bitplane
\$C0	Bitplane 1
\$C4	Bitplane 2
\$C8	Bitplane 3
\$CC	Bitplane 4
\$D0	Bitplane 5


```

indir ":include/"
include "exec/exec_lib.i"
include "intuition/intuition_lib.i"
include "graphics/graphics_lib.i"

    lea intname,a1
    moveq.l #0,d0
    CALLExec OpenLibrary ; Intuition öffnen
    move.l d0,_IntuitionBase
    lea gfxname,a1
    moveq.l #0,d0
    CALLExec OpenLibrary ; Graphics öffnen
    move.l d0,_GfxBase
    lea screen,a0
    CALLINT OpenScreen ; Screen öffnen
    move.l d0,screenpointer
    move.l d0,a0
    move.l $c4(a0),bitplane ; Bitplane 2
    lea $54(a0),a0
    move.l a0,RP ; Rastport
    move.l a0,a1 ; Farberegister 2 als
    move #2,d0 ; Vordergrundfarbe
    CALLGRAF SetAPen
    move.l _GfxBase,a6
    move #640,d2
    moveq #10,d3
loop1 move d2,d0
    move d3,d1
    move.l RP,a1
    jsr _LVOWritePixel(a6)
    subq #1,d2
    bne.s loop1
    move #640,d2
    addq #1,d3
    cmpi #255,d3
    bne.s loop1
    move.l RP,a1
    moveq #0,d0
    CALLGRAF SetAPen
    move #640,d2
loop2 move.l RP,a1
    move.l d2,d0
    move.l d3,d1
    jsr _LVOWritePixel(a6)
    subq #1,d2
    bne.s loop2
    move #640,d2
    subq #1,d3
    cmp #9,d3
    bne.s loop2
    move.l screenpointer,a0
    move.l #title2,$16(a0)
    moveq.l #1,d0
    CALLINT ShowTitle
    move.l #640,d2
    moveq.l #10,d3
    moveq #80,d4
    move.l bitplane,a0
loop3 move.l d2,d0
    move.l d3,d1
    bsr SetPixel ; Punkt setzen
    subq #1,d2
    bne.s loop3
    move #640,d2
    addq #1,d3
    cmp #255,d3
    bne.s loop3
loop4 move.l d2,d0
    move.l d3,d1
    bsr ClrPixel ; Punkt löschen
    subq #1,d2
    bne.s loop4
    move #640,d2
    subq #1,d3
    cmp #9,d3
    bne.s loop4

ende move.l screenpointer,a0
    move.l #title3,$16(a0)
    moveq.l #1,d0
    CALLINT ShowTitle
wait btst #6,$bfe001
    bne.s wait
    move.l screenpointer,a0

```

```

CALLINT CloseScreen
move.l _GfxBase,a1
CALLExec CloseLibrary
move.l _IntuitionBase,a1
CALLExec CloseLibrary
rts

SetPixel divu.w #8,d0
    mulu d4,d1
    add.w d0,d1
    swap.w d0
    moveq #7,d5
    sub.w d0,d5
    bset d5,(a0,d1.w)
    rts

ClrPixel divu.w #8,d0 ; der selbe Vorgang
    mulu d4,d1
    add.w d0,d1
    swap.w d0
    moveq #7,d5
    sub.w d0,d5
    bclr d5,(a0,d1.w) ; Pixel (Bit) löschen
    rts

screen dc.w 0,0,640,256,2
    dc.b 0,1
    dc.w $8000,$10
    dc.l 0,title1,0,0
screenpointer dc.l 0

intname dc.b "intuition.library",0
gfxname dc.b "graphics.library",0
even
title1 dc.b "Zuerst die langsame WritePixel-Routine",0
even
title2 dc.b "Aber hier die wesentlich schnellere SetPixel-Routine",0
even
title3 dc.b "» Maustaste drücken «",0
even

_IntuitionBase dc.l 0
_GfxBase dc.l 0
RP dc.l 0
bitplane dc.l 0

```

SetPixelDemo.asm

zeigt den Einsatz unserer Pixel-Routine

Wollen Sie einen Punkt wieder löschen, verwenden Sie statt BSET den Befehl BCLR. Eine Demonstration der Routine und den Geschwindigkeitsunterschied zu »WritePixel« zeigt »SetPixelDemo.asm«. Als Assembler haben wir Devpac V2.14D verwendet.

»SetPixel.asm« prüft nicht, ob die angegebenen Koordinaten innerhalb der Bitplane liegen, hierfür müssen Sie selbst sorgen.

Jürgen Schlierf/ik

Falscher Fehler

Ruft man eine selbstgeschriebene Assembler-Routine aus der »Startup-Sequence« auf, kann es beim Rücksprung (»rts«) zu einer Fehlermeldung kommen, obwohl das Programm einwandfrei arbeitet. Die Ursache ist im Prozessor-Register d0 zu suchen. Enthält es nach Ablauf eines Programms einen Wert ungleich null, nimmt das CLI (Command Line Interpreter) an, daß ein Fehler aufgetreten ist. Ihr Programm sollte daher vor dem Rücksprung den Inhalt des Registers d0 löschen:

```

ende:
    clr.l d0 ; Register löschen
    rts ; und zurück

```

Die durch einen Rückgabewert ungleich Null erzeugten Fehlermeldungen, »Returncode« genannt, spielen beim Ablauf von Stapeldaten (Batch-Dateien) wie der »Startup-Sequence« eine Rolle: Mit dem CLI-Befehl FAILAT legt man fest, ab welchem Rückgabewert die Bearbeitung einer Stapeldatei abgebrochen wird.

Immo Müller de Vries/C. Kögler

PRINT.asm

Befehle wie PRINT oder »printf()« erleichtern in Hochsprachen die Ausgabe eines Textes auf dem Bildschirm. Wohlgeemerkt, in Hochsprachen – Assembler-Programmierer sind gezwungen, den Text über die Funktion »Write« aus der »dos.library« auszugeben. Das sieht in den meisten Programmen etwa so aus (Bsp. für Devpac):

```
....
CALLDOS Output
move.l d0,d1
move.l #textanfng,d2
move.l #textende-textanfng,d3
CALLDOS Write
....
textanfng:
    dc.b "alloah"
textende:
```

Die Übersichtlichkeit eines Listings geht bei dieser Methode verloren, da die Ausgaberroutine und der zugehörige Text an unterschiedlichen Stellen im Programm stehen. Das Unterprogramm »Print.asm« schafft Abhilfe. Wir zeigen seine Anwendung an einem Beispiel für den Devpac-Assembler:

Folgender Trick kommt zum Einsatz: Der MC 68000 legt beim Aufruf eines Unterprogramms eine Rücksprungadresse als Langwort auf den Stapel (Stack). Die Adresse zeigt auf den Maschinenbefehl hinter dem Sprungbefehl. In unserem Fall befindet sich an dieser Stelle der auszugebende Text. Die Unterroutine »Print« holt sich diese Adresse vom Stapel, ermittelt die Länge des Textes und gibt

```
incdir "include/"
include exec/exec_lib.i
include libraries/dos_lib.i
include libraries/dos.i

* dos.library öffnen
start:
    move.l #dosname,a1
    moveq #0,d0           alle Versionen
    CALLEXEC _OpenLibrary
    tst.l d0
    beq raushier          keine dos.library
    move.l d0,_DOSBase     Zeiger retten

* Achtung, Trick 17
    bsr Print
    dc.b "Hallo Welt",10,0
    cnop 0,2

* dos.library schließen
    move.l _DOSBase,a1
    CALLEXEC _CloseLibrary

raushier:
    rts                  auf Wiedersehen!
    _DOSBase dc.l 0       Zeiger
    dosname  DOSNAME
    * hier beginnt die Unterroutine
    Print:
        move.l (sp),a0
        move.l a0,d2
    Loop:
        tst.b (a0)+
        bne.s Loop
        move.l a0,d0
        addq.l #1,d0
        bclr #0,d0
        move.l d0,(sp)
        move.l a0,d3
        sub.l d2,d3
    * Aufruf der DOS-Funktion Write
        move.l _DOSBase,a6
        CALLDOS Output
        move.l d0,d1
        CALLDOS Write
        rts
```

Print.asm

Programmieren in Assembler wie in einer richtigen Hochsprache - mit Makros geht's

ihn aus. Nun muß die Rücksprungadresse durch Addition der Textlänge korrigiert werden. Sie wird zusätzlich auf Wortlänge justiert, um einen Fehler durch Sprung auf eine ungerade Adresse zu verhindern. Wir fügen beim Devpac-Assembler hierzu die Anweisung »cnop 0,2« hinter der Definition des Strings ein. Wichtig: ein Text muß mit einer »0« enden. Der vollständige Aufruf von »Print« lautet:

```
bsr Print
dc.b "Text",0
cnop 0,2
```

In unserem Beispiel verwenden wir zusätzlich:

```
dc.b "Text",10,0
```

Der Amiga gibt in diesem Fall hinter dem Text einen Zeilenvor-schub (»10«) aus. Die Idee, die Rücksprungadresse auf dem Stack um die Länge eines Textes zu erhöhen, hatte der Amerikaner Glen Bredon. Er machte sie in 6502-Assembler populär.

Jörg Schmidt/ub

Bildschirmsplitting

Diese kleine Farbdemonstration splittet den Bildschirm in zwei Hälften. Das Programm demonstriert die Abfrage der Position des Rasterstrahls mit Hilfe der Hardwareregister.

Das Programm ist mit dem Seka-Assembler geschrieben. Nach dem Start wird zunächst jeder weitere Task gesperrt. Danach verändert das Programm ab einer bestimmten Bildschirmzeile die Hintergrundfarbe. Zur besseren Demonstration wird diese Farbe laufend geändert. Nach Durchlaufen der Farbskala erhöht das Programm die Zeile in der der Bildschirm gesplittet werden soll. Sie können den Programmablauf durch einen Druck der linken Maustaste stoppen. Noch besser wäre es, die Farbe des Hintergrunds per Copper ab einer Position zu verändern, aber das ist eine andere Geschichte.

Klaus Kuphal/ub

```
execbase = 4
forbid = -132
permid = -138

start:
    movem.l d0-d2/a1/a2/a6,-(SP)
    lea $dff004,a1      ; u.a. Y-POS des Rasterstrahls
    lea $dff180,a2      ; Farbregister
    move.l execbase,a6
    jsr forbid(a6)      ; Multitasking aus
    zeilenloop:
        move.l $2b00,d0 ; »2b« = Zeile für Split
                        ; »1b« = Minimum

    Farbloop:
        move.w $0000,d1 ;erste Farbe
    wait:
        move.l (a1),d2
        and.l #$0001ff00,d2 ; y-Pos. in d2
        cmp.l d0,d2      ; Zeile erreicht ?
        bne wait
        move.w d1,(a2)    ; neue Farbe ins Register
    waitl:
        move.l (a1),d2
        and.l #$0001ff00,d2
        cmp.l d0,d2
        beq waitl        ; nächste Zeile abwarten
        cmp.b #$bc,$bfe001
        beq done         ; linke Maustaste gedrückt ?
        add.w #$100,d1    ; neue Farbe
        cmp.w #$1000,d1   ; Farbe wechselt laufend
        blt wait
        add.l #$0100,d0    ; Splitzeile +1
        cmp.l #$13900,d0  ; größer Maximum?
        blt farbloop
        bge zeilenloop
    done:
        move.l execbase , a6
        jsr permid(a6)
        movem.l (sp)+,d0-d2/a1/a2/a6
        rts
```

Split.s

Das Seka-Programm spaltet den Bildschirm

Programmieren

C wie Cäsar

C gehört neben BASIC zur beliebtesten Sprache auf dem Amiga. Im Mittelpunkt steht vor allem die Systemprogrammierung: Betriebssysteme wie Unix und AmigaOS wurden in C geschrieben; das zeigt wie leistungsfähig diese Hochsprache ist. Aber es muß nicht gleich ein Betriebssystem sein, fangen wir langsam an – maChbar ist alles...

Mit ein Grund, warum C auf dem Amiga so erfolgreich ist, sind die beiden C-Compiler von SAS (ehem. Lattice) und Manx (Aztec-C). Mittlerweile haben Sie sich zu extrem leistungsfähigen Entwicklungswerkzeugen gemausert, mit denen man schnelle und speicherplatzsparende Programme erzeugen kann. Wer natürlich noch nicht das nötige Kleingeld für die Profi-C-Compiler hat, der kann auch mit den C-Compilern aus der Public-Domain-Szene sein Glück versuchen:

– Dice: Fish-Disk 491

– Sozobon C: Fish-Disk 508

Beide Systeme sind hervorragend geeignet, um sich in C einzuarbeiten, und man ist mit ihnen durchaus in der Lage, kleinere Projekte zu verwirklichen – versuchen Sie z.B., die Programme aus den folgenden Tips an die PD-C-Compiler anzupassen; es ist im Grunde gar nicht so kompliziert.

Endlose Kommentare

Der SAS/Lattice Compiler V5.10 läßt als Kommentarzeichen einen »Doppel-Slash« `»//«` zu. `»//«` gilt für eine Zeile und muß nicht geschlossen werden. Es gilt z.B.:

```
// Das ist ein Kommentar
a = b + c ; // links von diesem Kommentar ist C-Quellcode
```

Überlegt angewandt spart das einiges an Tipparbeit ohne, daß die Übersichtlichkeit leidet.

Richard Buschold/ub

Für Zeitlose

»SDatum.c« ist ein einfaches Programm für Anwender, deren Amiga keine akkugepufferte Uhr besitzt.

Immer wenn man den Amiga anschaltet, sollte man als erstes mit Hilfe des Befehls DATE das Datum und die Zeit setzen. Das kann, wenn man öfter den Amiga ein- und ausschaltet, mit der Zeit lästig werden. Wenn man aber »SDatum.c« kompiliert, linkt und in die »Startup-Sequence« einbindet, wird man beim Start des Amiga automatisch zur Eingabe des Datums und der Zeit aufgefordert. Dann führt das C-Programm mittels der Exec-Funktion »Execute()« den Befehl DATE mit den eingegebenen Daten aus. Die Befehle DATE und RUN müssen sich im Verzeichnis »C:« befinden, damit alles funktioniert.

Das Format des Datums ist: »TT-MMM-JJ«, z.B.: »18-Jul-75« oder »03-Dez-91«. Das Format für die Zeit lautet: »SS:MM:SS«. Die Angabe der Sekunden kann entfallen. Gültige Eingaben wären: »17:45:30« oder »18:24«.

Das Programm (getestet mit Aztec C 3.4a) wird übersetzt mit:

```
cc prg.c +l
ln prg.o -lc32
```

Sören Knebel/ub

```
#define bold printf("\033[1m\033[1m\033[11m")
#define norm printf("\033[0m")
void main()
{char datum[11],zeit[10],befehl[25];
 bold;
 printf("SDatum\033[0m by Sören Knebel\n");
 printf("Bitte Datum (dd-mm-yy): ");
 scanf("%s",&datum);
 printf("Bitte Zeit (hh:mm:ss) : ");
 scanf("%s",&zeit);
 printf("\nMoment...\n");
 sprintf(befehl,"date %s %s",datum,zeit);
 Execute(befehl,0L,0L); }
```

Datum.c

Damit Sie immer wissen, was die Uhr geschlagen hat

Optimales C die erste

Wenn man C-Programme vollends optimiert, sind die Assembler-Ergebnisse des C-Compilers kaum noch von sauber entwickelten Assembler-Programmen zu unterscheiden. Daß es so weit kommt, liegt neben intelligenten Optimierungsmethoden moderner C-Compi-

TIPS ZU C AUF EINEN BLICK

Tip	Thema	Seite
Endlose Kommentare	Programmierungsmethoden	33
Für Zeitlose	Zeit einstellen	33
Optimales C die erste	Programmierungsmethoden	33
Wie arbeitet der DIR-Befehl?	DOS-Library	34
Maus aus	Hardwareregister	34
Optimales C die zweite	Programmierungsmethoden	34
Kleine Gemeinsamkeit	Kleinstes gemeinsames Vielfaches	35
Commander	Programmierungsmethoden	35
Rot, grün und blau	SetRGB.c	37
PowerColor	Verbesserung zu SetRGB.c	38
Move	Neuer CLI-Befehl	39
itoa – Lattice hat's	Integer to ASCII	39
Link-Library mit Lattice	Linken	39
C-opperdemo	Copperprogrammierung	39
Nicht zu dynamisch	Speicherverwaltung	41
MutualExclude-Gadget	Gadgetprogrammierung	42
Warnungen	Beep-Befehl	43
Herein, Heraus	Druckerausgabe	43
Highspeed-Amiga	Bildschirm ausschalten	43
Ist irgendwo ein Fenster auf?	Windows abfragen	44
Anmeldliste für Geräte	Devices	44
Vollbremsung	Warteschleifen sind tot	45
Sprechen Sie C?	Sprachausgabe	45
Kalender	Monatskalender selbstgemacht	45
Icons automatisch öffnen	Workbench	46
Vom String zur Zahl	ASCII konvertieren	47
Kürzerer C-Code	Programmierungsmethoden	47
beep, beep	Noch ein Beep-Befehl	47

ler hauptsächlich am Geschick des Programmierers. Doch zu viel des Guten geht häufig auch auf Kosten der Lesbarkeit und der Übersichtlichkeit von C-Programmen, d.h. die strukturellen Vorteile der Hochsprache gehen verloren. Es lohnt sich eigentlich nur, solche Programmteile zu optimieren, die auch mehrfach ausgeführt werden, also hauptsächlich Schleifenblöcke. Allzu kryptische und dadurch unleserliche Optimierungen sollten auch ausreichend kommentiert werden, um ein Programm lesbar zu halten. Doch nun zu den Optimierungs-Möglichkeiten:

- Divisionen und Multiplikationen ganzer Zahlen mit Zweierpotenzen lassen sich effektiv mit Shift-Befehlen realisieren, doch lassen sich auch bei anderen Zahlen durch einen kleinen Trick die Berechnungszeiten enorm verkürzen:

```
i = (j<<4)-j-j-j; /* entspricht i = j*13 */
i = (j<<3)+j+j; /* entspricht i = j*10 */
```

- Library-Funktionen aus dem umfangreichen Funktions-Kontingent heutiger C-Compiler sind mit Vorsicht zu genießen. Besonders, wenn sie häufiger aufgerufen werden, sollten sie auf ihre Effizienz hin überprüft werden.

Oftmals sind maßgeschneiderte Funktionen um ein Vielfaches kürzer und schneller als die Standardfunktionen, die mit jeder möglichen – und unmöglichen – Eingabe fertig werden müssen. Als Musterbeispiel sei da die Funktion »pow()« genannt. Sie rechnet intern mit Logarithmus-Funktionen (um auch Kommazahlen zu verarbeiten) und ist somit relativ langsam. Oft besteht die Eingabe allerdings aus Ganzzahlen. Eine eigene Routine zur Potenzierung bewirkt hier einen deutlichen Geschwindigkeitszuwachs.

- Die Benutzung von Feldern ist bei zeitkritischen Programmteilen nicht gerade vorteilhaft. Bei jedem Feldzugriff wird nämlich intern eine Multiplikation ausgeführt:

```
int n,m[100];
n = m[40];
```

Hierbei wird der Index (hier: 40) mit der Länge des Basiselementes (hier: 4) multipliziert, das Ergebnis (Bsp.: 160) addiert auf die Adresse von »m«, ergibt schließlich die Adresse, deren Inhalt nach »n« transferiert wird. So eine nicht offensichtliche Multiplikation wird dann noch besonders schwerwiegend, wenn es sich bei der Länge von Feldbasiselementen nicht um Potenzen von »2« handelt, wie das z.B. bei Strukturen der Fall sein kann. Dann nämlich wird eine »echte« Multiplikation unvermeidbar und somit viel Rechenzeit verschwendet. Glücklicherweise läßt sich das in vielen Fällen durch die Benutzung von Zeigern vermeiden, was das folgende Beispiel veranschaulicht:

```
short *p,i,m[50];
for (i=0;i<50;i++) m[i]=0; /* erst (langsam) mit Feldern */
for (p=m;p<=&m[49];p++) *p = 0; /* dann schnell mit Zeigern */
```

Natürlich kann das obige Beispiel durch die explizite Verwendung von »register«-Variablen noch beschleunigt werden, falls das der Compiler nicht bereits erledigt; in Zweifelsfällen sollte man den erzeugten (Assembler-)Code daraufhin untersuchen.

Thomas Becker/ik

Wie arbeitet der DIR-Befehl?

Der Befehl DIR aus dem »C«-Verzeichnis dient dazu, den Inhalt eines Verzeichnisses (Directory) auf dem Bildschirm anzuzeigen. Doch wie arbeiten solche Befehle? Wie erreichen Programmierer die einzelnen Namen eines Verzeichnisses?

Das Programm »ShowDir.c« gibt Antwort auf diese Fragen. Es gibt das Verzeichnis aus, dessen Name man als Parameter übergibt. Der erste Schritt ist es, einen »Lock« auf das Verzeichnis zu besorgen, mit dem dann im Verlauf gearbeitet wird. (Anm.: Es heißt: »der Lock«, nicht: »die Lock«; aus dem Englischen: lock = Riegel).

Mit dem Lock und der Betriebssystemfunktion »Examine()« wird nun ein »FileInfoBlock« ausgefüllt. In diesem Datenblock befinden sich nach dem Aufruf Informationen über das gewünschte Verzeichnis. Aus dem Eintrag »fib_DirEntryType« geht z.B. hervor, ob es sich überhaupt um ein Verzeichnis handelt oder eine Datei. Nur bei

einem Verzeichnis (fib_DirEntryType > 0) ist der nächste Schritt sinnvoll: Mit Hilfe von »ExNext()« gelangen wir an die Namen der Directory-Einträge, denn immer wenn der Aufruf dieser Funktion erfolgreich war (TRUE), wurde ein neuer »FileInfoBlock« ausgefüllt, in dem auch im Eintrag »fib_FileName« der Name des Files (oder Directories) steht. Diese brauchen dann nur nach nacheinander ausgedruckt werden, bis »ExNext()« FALSE zurückgibt. Handelt es sich bei einem Eintrag wieder um ein Directory, erfolgt beim Ausdrucken hinter dem Namen die Kennzeichnung »DIR«.

Christof Brühann/ik

```
/* »ShowDir.c« von Christof Brühann
- demonstriert die Arbeitsweise von DIR -
Aufruf mit Aztec 3.6: cc ShowDir.c +l
ln ShowDir.o -lc32
ShowDir <dirname> */

#include functions.h
#include libraries/dosextens.h
struct FileInfoBlock info;
struct FileLock *lock;
void main(argc,argv)
SHORT argc;
UBYTE **argv;
{
    if (lock=(struct FileLock *)Lock(argv[1],ACCESS_READ))
    {
        if (Examine(lock,&info))
        if (info.fib_DirEntryType>0)
        while (ExNext(lock,&info))
        {
            printf (" %s",&info.fib_FileName);
            if (info.fib_DirEntryType>0) printf (" (DIR)");
            printf ("\n");
        }
        UnLock(lock);
    }
}
```

»ShowDir.c«

Das Programm zeigt, wie ein Verzeichnis gelistet wird

Maus Aus

Das folgende C-Programm setzt ein Hardware-Register so, daß der Mauszeiger verschwindet.

Andreas Schulz/ub

```
main()
{
    char *adr = 0xdff096;
    *adr = 32;
}
```

»Maus_aus.c«

Löschen der Maus über ein -Hardware-Register

Optimales C die zweite

Es liegt hauptsächlich in den Händen des Programmierers, aus dem Universalgenie C das zu machen, was es ist: eine optimale Programmiersprache! Damit es auch so weit kommt, folgen nun weitere Optimierungsregeln, die jeder C-Programmierer kennen und auch effektiv anwenden sollte:

- Sehr nützlich und ebenso übersichtlich ist die mehrfache Zuweisung, wie:

```
a = b = c = 4;
```

Diese Konstruktion bringt den Vorteil, daß die Zwischenergebnisse immer in den schnellen CPU-Registern zwischengespeichert werden und der Amiga somit ohne Umwege den nächste Variablenwert be-

legen kann. Noch mehr Gewinn bringt diese Regel bei geschachtelten Konstruktionen wie:

```
if ((c=getint()+100) != 200) {...
statt:
c = getint();
c = c+100
if (c != 200) {...
```

Bei der zweiten Variante kann man davon ausgehen, daß vier Hauptspeicherezugriffe auf die Variable »c« erfolgen, während die schnellere, geschachtelte Version nur einen langsamen Hauptspeicherezugriff benötigt. Die kompliziert aussehende verschachtelte Programmierzeile erzeugt man, indem man immer den Ausdruck, der »c« gleichgesetzt wurde, in der nächsten Zeile für »c« einsetzt. Ebenso kann man durch die umgekehrte Vorgehensweise eine verschachtelte Zeile entschachteln.

• Auch Fallunterscheidungen können Sie durch die Verwendung des Fragezeichen-Doppelpunkt-Operators schachteln:

```
a = (x != 3) ? a+x : x+1
statt:
if (x != 3) a += x;
else a = x+1;
```

Hierbei ist der Wert von »x« nach dem Vergleich höchstwahrscheinlich noch in einem Register und kann dann zur Belegung von »a« weiterverwendet werden.

• Bei komplexen Fallunterscheidungen, die durch umfangreiche logische Verknüpfungen verursacht werden, spart man oftmals Zeit:

```
if (func1() && func2() && func3()) {...
```

Gibt die erste Funktion eine »0« zurück, gleichbedeutend mit FALSE, kann der gesamte Ausdruck nicht mehr wahr werden, womit »func2()« und »func3()« gar nicht mehr ausgeführt werden brauchen (»Lazy And«, oder »short evaluation«), was die meisten C-Compiler auch so in die Tat umsetzen. Durch geschicktes Sortieren der Funktionen, wenn man vorher schon abschätzen kann, wie häufig welche Funktion wahr oder falsch wird, lassen sich Programmteile durch Logik und vorausschauendes Abschätzen beschleunigen.

• Durch übertriebene Verwendung von Mini-Funktionen wird nicht nur die gesamte Strukturierung schlechter, sondern auch der interne Aufwand während der Programmausführung geht stark auf Kosten der Geschwindigkeit. Durch die Benutzung des Präprozessors kann ein Programmierer Mini-Funktionen und -Routinen als Makros definieren, die der Compiler beim Übersetzen im Quelltext einsetzt:

```
#define isascii(x) (((x)&0x80)==0)
```

Dieses Beispiel aus dem Header-File »ctype.h« testet, ob der einge-setzte Funktionsparameter ein gültiges ASCII-Zeichen ist und kehrt dementsprechend zurück (wahr oder falsch). Die Variable »x« dient hierbei als Übergabeparameter. Bei diesen Makros entfällt der große Aufwand, der sonst durch die Verwendung von Funktionen angefallen wäre. Einzig und allein der Quelltext wird etwas mächtig er, falls größere Makros häufig im Quelltext vorkommen.

Thomas Becker/ik

Kleine Gemeinsamkeit

Bei vielen Programmierprojekten wird ein kleinstes gemeinsames Vielfaches (kurz kgV) von zwei ganzen Zahlen benötigt. Beispielsweise das kgV von »8« und »10« ist »40«. Zur Errechnung des kgV existiert glücklicherweise ein simpler und schneller Algorithmus, den wir hier in C als Funktion »kgV()« in einem Demoprogramm umgesetzt zeigen.

Der Funktion »kgV()« werden zwei positive, ganze Zahlen übergeben und sie gibt dann das kleinste gemeinsame Vielfache zurück. Im Demo-Programm wird das kgV aus mit der Zufallsfunktion der »c.lib« gewonnenen Zufalls-Integer-Zahlen errechnet.

Am praktischsten für den späteren Gebrauch von »kgV()« ist es, die Funktion im Demo-Programm (ohne den main-Teil) normal zu kompilieren (Compiler-Optionen für Aztec-C siehe Listing) und bei Bedarf beim Linken ans Hauptprogramm anzuhängen, mit:

```
ln haupt.o kgv.o -lc
```

Dabei muß man dann die Funktion »kgV()« mit extern LONG kgV()« im Hauptprogramm anmelden.

Thomas Becker/ik

```
/* kgV-Funktion
 * Aufruf: d = kgV(a,b)
 * d,a,b sind dabei > 0!
 * Compiler Options:
 * cc ggt.c
 * ln +q ggt.o -lc
 */
#include <exec/types.h>
#define ANZAHL 25
main () /* dient zur Demonstration von kgv() */
{
    ULONG a,b,kgv();
    int rand(); /* Zufallsfunktion der c.lib */
    VOID srand();
    SHORT i=ANZAHL;
    srand(13); /* irgendein Start-Seed */
    while (i--) /* Errechne Zufallszahlen und gebe ihren kgV aus */
    {
        a = rand(); b = rand();
        printf ("kgV (%5lu,%5lu) \t= %5lu\n",a,b,kgv(a,b));
    }
} /* end of main() */
/* Bestimmt den kgV von a,b
Achtung: a,b>0 !!! */

ULONG kgv (a,b)
ULONG a,b;
{
    ULONG x = a, y = b;
    while (x!=y)
        if (x<y) x+=a;
        else y+=b;
    return (x);
} /* end of kgv() */
```

»kgV.c«

Kleinstes gemeinsames Vielfaches zweier Zahlen in C

Commander

»Commander.c« ist besonders für Aztec-C-Programmierer geeignet. Mit diesem Werkzeug (Tool) kann man auf Mausclick einen Quelltext editieren, übersetzen lassen oder das fertige Programm ausführen. Vorher muß man allerdings die Aufrufe für die einzelnen Aufträge in die folgenden Gadgets (Schalter) eintragen:

• CC-USAGE: Hier trägt man den Aufruf des Compilers ein, z.B.:

```
cc bsp_programm.c +l -s -a
```

• AS-USAGE: Für Eintrag zum (wenn der Assembler nicht vom Compiler aufgerufen wird) Aufruf des Assemblers, z.B.:

```
as bsp_programm.asm
```

• LN-USAGE: Aufruf des Linkers, z.B.:

```
ln bsp_programm -lc32
```

• ED-USAGE: tragen Sie hier Ihren Editor ein. Beispiel:

```
ed bsp_programm
```

• PRG-USAGE: ruft Programm auf. Beispiel:

```
bsp_programm
```

Die Gadgets und Ihre Funktion zeigt die Tabelle.

Gadget	Funktion
RUN-PROGRAM	ruft fertiges Programm auf
CC+AS+LN PROG.	kompiliert, assembliert und linkt Quelltext
CC+LN PROGRAM	kompilieren und linken
EDIT PROGRAM	startet Editor

Sobald Sie die Gadgets »RUN-PROG.«, »CC+AS+LN-PROG.«, »CC+LN PROG.« und »EDIT PROGRAM« auswählen, wird der Commander automatisch in den Hintergrund gesetzt. Oben wird

noch das Datum mit dem Wochentag und der aktuellen Uhrzeit angezeigt. Darunter steht der noch verfügbare Speicherplatz in KByte.

Das Programm wurde mit Aztec C 3.4a geschrieben und folgendermaßen übersetzt:

```
cc prg.c +l -s
ln prg.o -lc32
```

Sören Knebel/ub

```
/* COMMANDER done by SÖREN KNEBEL */

#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <libraries/dos.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <stdio.h>
#include <time.h>
#define s 80

struct IntuitionBase *IntuitionBase = 0L;
struct GfxBase *GfxBase = 0L;
static struct RastPort *my_rp;
static struct ViewPort *my_vp;
struct Window *MyWin;
struct IntuiMessage *message;

void *OpenLibrary(), *OpenWindow();
void *CloseLibrary(), *CloseWindow();

struct TextAttr Font = {"topaz.font",
    TOPAZ_SIXTY,FS_NORMAL,FPF_ROMFONT,};
unsigned char CCStringBuffer[s] = "", CCUndoBuffer[s];
unsigned char ASStringBuffer[s] = "", ASUndoBuffer[s];
unsigned char LNStringBuffer[s] = "", LNUndoBuffer[s];
unsigned char EDStringBuffer[s] = "", EDUndoBuffer[s];
unsigned char PGStringBuffer[s] = "", PGUndoBuffer[s];

short PairsA[] = { 0, 0, 252, 0, 252, 12, 0, 12, 0, 0, };
short PairsB[] = { 0, 0, 171, 0, 171, 11, 0, 11, 0, 0, };

struct Border CCBorder = {-2, -2, 1, 0, JAM1, 5, PairsA, NULL,};
struct Border ASBorder = {-2, -2, 1, 0, JAM1, 5, PairsA, NULL,};
struct Border LNBorder = {-2, -2, 1, 0, JAM1, 5, PairsA, NULL,};
struct Border EDBorder = {-2, -2, 1, 0, JAM1, 5, PairsA, NULL,};
struct Border PGBorder = {-2, -2, 1, 0, JAM1, 5, PairsA, NULL,};
struct Border RunPrgBorder = {-1, -1, 1, 0, JAM1, 5, PairsB, NULL,};
struct Border CALPrgBorder = {-1, -1, 1, 0, JAM1, 5, PairsB, NULL,};
struct Border CLPrgBorder = {-1, -1, 1, 0, JAM1, 5, PairsB, NULL,};
struct Border EDPrgBorder = {-1, -1, 1, 0, JAM1, 5, PairsB, NULL,};
struct Border AboutBorder = {-1, -1, 1, 0, JAM1, 5, PairsB, NULL,};

struct IntuiText CCText = { 1, 0, JAM2, -115, 1, &Font,
    (UBYTE *) "CC-USAGE:", NULL, };
struct IntuiText ASText = { 1, 0, JAM2, -115, 1, &Font,
    (UBYTE *) "AS-USAGE:", NULL, };
struct IntuiText LNText = { 1, 0, JAM2, -115, 1, &Font,
    (UBYTE *) "LN-USAGE:", NULL, };
struct IntuiText EDText = { 1, 0, JAM2, -115, 1, &Font,
    (UBYTE *) "ED-USAGE:", NULL, };
struct IntuiText PGText = { 1, 0, JAM2, -115, 1, &Font,
    (UBYTE *) "PRG-USAGE:", NULL, };
struct IntuiText RunPrgText = { 1, 0, JAM2, 1, 1, &Font,
    (UBYTE *) "RUN PROGRAM", NULL, };
struct IntuiText CALPrgText = { 1, 0, JAM2, 1, 1, &Font,
    (UBYTE *) "CC+AS+LN PROGRAM", NULL, };
struct IntuiText CLPrgText = { 1, 0, JAM2, 1, 1, &Font,
    (UBYTE *) "CC+LN PROGRAM", NULL, };
struct IntuiText EDPrgText = { 1, 0, JAM2, 1, 1, &Font,
    (UBYTE *) "EDIT PROGRAM", NULL, };
struct IntuiText AboutText = { 1, 0, JAM2, 1, 1, &Font,
    (UBYTE *) "ABOUT...", NULL, };
char Date[80], Mem[80];
struct IntuiText DateText = { 1, 0, JAM2, 0, 0, NULL,
    (UBYTE *) "Date", NULL, };
struct IntuiText MemText = { 1, 0, JAM2, 0, 0, NULL,
    (UBYTE *) "Mem", NULL, };
struct StringInfo CCStringInfo = { &CCStringBuffer[0],
    &CCUndoBuffer[0], 0, s, 0, 0, 0, 0, 0, NULL, 0, NULL, };
struct StringInfo ASStringInfo = { &ASStringBuffer[0],
    &ASUndoBuffer[0], 0, s, 0, 0, 0, 0, 0, };
struct StringInfo LNStringInfo = { &LNStringBuffer[0], &LNUndoBuffer[0], 0,
    s, 0, 0, 0, 0, 0, 0, NULL, 0, NULL, };

struct StringInfo EDStringInfo = { &EDStringBuffer[0],
    &EDUndoBuffer[0], 0, s, 0, 0, 0, 0, 0, NULL, 0, NULL, };
struct StringInfo PGStringInfo = { &PGStringBuffer[0],
    &PGUndoBuffer[0], 0, s, 0, 0, 0, 0, 0, NULL, 0, NULL, };

struct Gadget CCStringGadget = (NULL, 150, 50, 250, 10,
    GADGHCMP, RELVERIFY, STRGADGET, (APTR) &CCBorder, NULL, &CCText,
    NULL, (APTR) &CCStringInfo, 1, NULL, );
struct Gadget ASStringGadget = (&CCStringGadget, 150, 70, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &ASBorder,
    NULL, &ASText, NULL, (APTR) &ASStringInfo, 2, NULL, );
struct Gadget LNStringGadget = (&ASStringGadget, 150, 90, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &LNBorder,
    NULL, &LNText, NULL, (APTR) &LNStringInfo, 3, NULL, );
struct Gadget EDStringGadget = (&LNStringGadget, 150, 110, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &EDBorder,
    NULL, &EDText, NULL, (APTR) &EDStringInfo, 4, NULL, );
struct Gadget PGStringGadget = (&EDStringGadget, 150, 130, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &PGBorder, NULL,
    &PGText, NULL, (APTR) &PGStringInfo, 5, NULL, );
struct Gadget RunPrgGadget = (&PGStringGadget, 430, 49, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &RunPrgBorder,
    NULL, &RunPrgText, NULL, NULL, 6, NULL, );
struct Gadget CALPrgGadget = (&RunPrgGadget, 430, 69, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &CALPrgBorder,
    NULL, &CALPrgText, NULL, NULL, 7, NULL, );
struct Gadget CLPrgGadget = (&CALPrgGadget, 430, 89, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &CLPrgBorder,
    NULL, &CLPrgText, NULL, NULL, 8, NULL, );
struct Gadget EDPrgGadget = (&CLPrgGadget, 430, 109, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &EDPrgBorder,
    NULL, &EDPrgText, NULL, NULL, 9, NULL, );

struct NewWindow MyNewWin = { 0, 0, 640, 256, 0, 1,
    CLOSEWINDOW | GADGETUP | GADGETDOWN, WINDOWDEPTH | ACTIVATE |
    WINDOWCLOSE | SMART_REFRESH, &EDPrgGadget, NULL,
    (UBYTE *) "Commander by Sören Knebel", NULL, NULL,
    100, 100, 640, 256, WBENCHSCREEN, };

void an() { ActivateWindow(MyWin); }

void auf() {
    if(!(IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 0L)))
        exit(FALSE);
    if(!(GfxBase = (struct GfxBase *)
        OpenLibrary("graphics.library", 0L)))
        exit(FALSE);
    if(!(MyWin = (struct Window *)
        OpenWindow(&MyNewWin)))
    {
        printf("Can't continue! Where's Window?\n"); exit(FALSE);
    }
}

void zu() {
    if(MyWin != 0L) CloseWindow(MyWin);
    if(GfxBase != 0L) CloseLibrary(GfxBase);
    if(IntuitionBase != 0L) CloseLibrary(IntuitionBase);
}

void init() {
    my_rp = (struct RastPort *) MyWin->RPort;
    my_vp = (struct ViewPort *) ViewPortAddress(MyWin);

    void drawbox() {
        SetAPen(my_rp, 1);
        Move(my_rp, 2, 46);
        Draw(my_rp, 637, 46);
        Move(my_rp, 2, 142);
        Draw(my_rp, 637, 142);
    }

    char *wtag[] = { "Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday" };
    char *mon[] = { "January", "February", "March", "April", "May",
        "June", "July", "August", "September", "October", "November",
        "December" };

    void main()
    {
        ULONG MessageClass;
        USHORT code, GadgetNr;

        time_t just;
        struct tm *zt;
        struct Message *GetMsg();
    }
}
```

```
struct StringInfo EDStringInfo = { &EDStringBuffer[0],
    &EDUndoBuffer[0], 0, s, 0, 0, 0, 0, 0, NULL, 0, NULL, };
struct StringInfo PGStringInfo = { &PGStringBuffer[0],
    &PGUndoBuffer[0], 0, s, 0, 0, 0, 0, 0, NULL, 0, NULL, };

struct Gadget CCStringGadget = (NULL, 150, 50, 250, 10,
    GADGHCMP, RELVERIFY, STRGADGET, (APTR) &CCBorder, NULL, &CCText,
    NULL, (APTR) &CCStringInfo, 1, NULL, );
struct Gadget ASStringGadget = (&CCStringGadget, 150, 70, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &ASBorder,
    NULL, &ASText, NULL, (APTR) &ASStringInfo, 2, NULL, );
struct Gadget LNStringGadget = (&ASStringGadget, 150, 90, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &LNBorder,
    NULL, &LNText, NULL, (APTR) &LNStringInfo, 3, NULL, );
struct Gadget EDStringGadget = (&LNStringGadget, 150, 110, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &EDBorder,
    NULL, &EDText, NULL, (APTR) &EDStringInfo, 4, NULL, );
struct Gadget PGStringGadget = (&EDStringGadget, 150, 130, 250,
    10, GADGHCMP, RELVERIFY, STRGADGET, (APTR) &PGBorder, NULL,
    &PGText, NULL, (APTR) &PGStringInfo, 5, NULL, );
struct Gadget RunPrgGadget = (&PGStringGadget, 430, 49, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &RunPrgBorder,
    NULL, &RunPrgText, NULL, NULL, 6, NULL, );
struct Gadget CALPrgGadget = (&RunPrgGadget, 430, 69, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &CALPrgBorder,
    NULL, &CALPrgText, NULL, NULL, 7, NULL, );
struct Gadget CLPrgGadget = (&CALPrgGadget, 430, 89, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &CLPrgBorder,
    NULL, &CLPrgText, NULL, NULL, 8, NULL, );
struct Gadget EDPrgGadget = (&CLPrgGadget, 430, 109, 170,
    10, GADGHCMP, RELVERIFY, BOOLGADGET, (APTR) &EDPrgBorder,
    NULL, &EDPrgText, NULL, NULL, 9, NULL, );

struct NewWindow MyNewWin = { 0, 0, 640, 256, 0, 1,
    CLOSEWINDOW | GADGETUP | GADGETDOWN, WINDOWDEPTH | ACTIVATE |
    WINDOWCLOSE | SMART_REFRESH, &EDPrgGadget, NULL,
    (UBYTE *) "Commander by Sören Knebel", NULL, NULL,
    100, 100, 640, 256, WBENCHSCREEN, };

void an() { ActivateWindow(MyWin); }

void auf() {
    if(!(IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 0L)))
        exit(FALSE);
    if(!(GfxBase = (struct GfxBase *)
        OpenLibrary("graphics.library", 0L)))
        exit(FALSE);
    if(!(MyWin = (struct Window *)
        OpenWindow(&MyNewWin)))
    {
        printf("Can't continue! Where's Window?\n"); exit(FALSE);
    }
}

void zu() {
    if(MyWin != 0L) CloseWindow(MyWin);
    if(GfxBase != 0L) CloseLibrary(GfxBase);
    if(IntuitionBase != 0L) CloseLibrary(IntuitionBase);
}

void init() {
    my_rp = (struct RastPort *) MyWin->RPort;
    my_vp = (struct ViewPort *) ViewPortAddress(MyWin);

    void drawbox() {
        SetAPen(my_rp, 1);
        Move(my_rp, 2, 46);
        Draw(my_rp, 637, 46);
        Move(my_rp, 2, 142);
        Draw(my_rp, 637, 142);
    }

    char *wtag[] = { "Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday" };
    char *mon[] = { "January", "February", "March", "April", "May",
        "June", "July", "August", "September", "October", "November",
        "December" };

    void main()
    {
        ULONG MessageClass;
        USHORT code, GadgetNr;

        time_t just;
        struct tm *zt;
        struct Message *GetMsg();
    }
}
```



```

struct Gadget *GadgetPtr;
int tnr, tag, mnr, jahr, std, min, sek;
register short chip, fast, all;

auf();
init();
RefreshGadgets(&CCStringGadget, MyWin, NULL);
drawbox();
FOREVER

{
    Delay(10L);
    time(&just);
    zt = localtime(&just);
    tnr = zt->tm_wday;
    tag = zt->tm_mday;
    mnr = zt->tm_mon;
    jahr = zt->tm_year;
    std = zt->tm_hour;
    min = zt->tm_min;
    sek = zt->tm_sec;
    chip = AvailMem(MEMF_CHIP) >> 10;
    fast = AvailMem(MEMF_FAST) >> 10;
    all = AvailMem(MEMF_CHIP) + AvailMem(MEMF_FAST) >> 10;
    sprintf(Date, " %s %d-%d %s %2d:%02d:%02d",
    mon[mnr], tag, jahr, wtag[tnr], std, min, sek);
    sprintf(Mem, "      Free Memory: %dK      ",
    all);
    PrintIText(my_rp, &DateText, 145L, 15L);
    PrintIText(my_rp, &MemText, 145L, 26L);
    if(message = (struct IntuiMessage *)
    GetMsg(MyWin->UserPort))
    {
        MessageClass = message->Class;
        code = message->Code;
        GadgetPtr = (struct Gadget *)message->IAddress;
        GadgetNr = GadgetPtr->GadgetID;
        ReplyMsg(message);
        switch(MessageClass)
        {
            case CLOSEWINDOW: Delay(30L);
                zu();
                exit(TRUE);
                break;
            case GADGETUP:
            case GADGETDOWN:
                switch(GadgetNr)
                {
                    case 6: WindowToBack(MyWin);
                        Execute(&PGStringBuffer[0], 0L, 0L);
                        WindowToFront(MyWin);
                        an();
                        continue;

                    case 7: WindowToBack(MyWin);
                        Execute(&CCStringBuffer[0], 0L, 0L);
                        Execute(&ASStringBuffer[0], 0L, 0L);
                        Execute(&LNStringBuffer[0], 0L, 0L);
                        WindowToFront(MyWin);
                        an();
                        continue;

                    case 8: WindowToBack(MyWin);
                        Execute(&CCStringBuffer[0], 0L, 0L);
                        Execute(&LNStringBuffer[0], 0L, 0L);
                        WindowToFront(MyWin);
                        an();
                        continue;

                    case 9: WindowToBack(MyWin);
                        Execute(&EDStringBuffer[0], 0L, 0L);
                        WindowToFront(MyWin);
                        an();
                        continue;
                }
            }
        }
    }
}

```

Commander.c

Zum Kompilieren und Linken per Mausklick

Rot, grün und blau

Das Programm »SetRGB.c« ändert die Farbe des angegebenen Farbregisters. Die einzelnen Farbkomponenten (Rot, Grün, Blau) können Werte zwischen 0 und 15 annehmen. Daraus ergeben sich $16^3 = 4096$ verschiedene Farben.

Sie übersetzen »SetRGB.c« folgendermaßen (Aztec-C V.3.4a):

```
cc setrgb.c +l -s
```

```
ln setrgb.o -lc32
```

```

/* SetRGB.c */

#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>

#define in "intuition.library"
#define gfx "graphics.library"

struct IntuitionBase *IntuitionBase = 0L;
struct GfxBase *GfxBase = 0L;
struct Window *Win;
struct ViewPort *VP;

open_all(){
    if(!(IntuitionBase = (struct IntuitionBase *)
        OpenLibrary(in,0L)))
        exit(FALSE);
    if(!(GfxBase = (struct GfxBase *)OpenLibrary(gfx,0L)))
        exit(FALSE);
    close_all(){
        if(GfxBase != 0L) CloseLibrary(GfxBase);
        if(IntuitionBase != 0L) CloseLibrary(IntuitionBase);
    }

    main(ac,av)
    int ac; char **av;
    {
        long reg,r,g,b;
        open_all();
        Win = IntuitionBase->ActiveWindow;
        VP = (struct ViewPort *)ViewPortAddress(Win);
        if(ac != 5){
            printf("Usage: %s [Farbregister] [R] [G] [B] \n",av[0]);
            exit(FALSE);
        }
        else{
            reg = atol(av[1]);
            r = atol(av[2]);
            g = atol(av[3]);
            b = atol(av[4]);
            SetRGB4(VP,reg,r,g,b);
            close_all();
            exit(TRUE);
        }
    }
}

```

SetRGB.c

zum Einstellen einer bestimmten Farbe

Beim Aufruf von SetRGB erscheint folgende Informationszeile:

```
Usage: SetRGB [Farbregister] [R] [G] [B]
```

»Farbregister« ist hierbei die Nummer des Farbregisters, das Sie verändern möchten:

- »0« ist das Register der Hintergrund-,
 - »1« das der Vordergrundfarbe.
 - Mit »2« adressieren Sie das Register, das die Farbe der Depth-Gadgets eines Windows enthält (Standard-Workbench- bzw. Shell-Screen) und
 - mit »3« das Register der Cursorfarbe.
- »R«, »G« und »B« geben die Rot-, Grün- und Blaukomponente der

neuen Farbe an (Werte von 0 bis 15). Beispielaufufe:

```
SetRGB 0 0 0 15 ; ändert die Hintergrundfarbe in Blau
SetRGB 1 15 15 15 ; Vordergrundfarbe in Weiß
SetRGB 2 9 9 9 ; Depth-Gadgets eines Windows in Grau
SetRGB 3 15 15 0 ; Cursorfarbe Gelb
```

»SetRGB« kann nur vom CLI aus aufgerufen werden. Binden Sie es z.B. in der »Startup-Sequence« ein, um Farben beim Start zu ändern, wenn kein Platz mehr für die Preferences auf der Diskette ist.

Sören Knebel/ub

PowerColor

»PowerColor.c« macht dasselbe wie »SetRGB«, jedoch ist es durch seine Gadgets einfacher zu handhaben und kann auch von der Workbench gestartet werden.

Erläuterung der einzelnen Gadgets:

Gadget	Funktion
R+	ändert Rotkomponente um +1
R-	ändert Rotkomponente um -1
G+	ändert Grünkomponente um +1
G-	ändert Grünkomponente um -1
B+	ändert Blaukomponente um +1
B-	ändert Blaukomponente um -1

Der Aufruf von PowerColor lautet:

PowerColor [Farbregister]

Das Farbregister ist, wie bei »SetRGB«, eine Zahl zwischen »0« und »3«. »0« ist die Hinter-, »1« die Vordergrundfarbe, usw. Das Programm ist mit Aztec C 3.4a getestet worden. Übersetzung mit:

```
cc prg.c +l -s
ln prg.o -lc32
```

Sören Knebel/ub

```
/* PowerColor */
#include <exec/types.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#define CR printf("\n") /* carriage return */
struct IntuitionBase *IntuitionBase = 0L;
struct GfxBase *GfxBase = 0L;
struct Window *MyWindow, *OpenWindow();
static struct RastPort *my_rp;
struct ViewPort *my_vp;
struct IntuiMessage *message;
void *OpenLibrary();
struct IntuiText RedPlusText = {2,0,JAM2,4,7,NULL,
(UBYTE *)"R+",NULL};
struct IntuiText RedMinusText = {2,0,JAM2,4,7,NULL,
(UBYTE *)"R-",NULL};
struct IntuiText GreenPlusText = {2,0,JAM2,4,7,NULL,
(UBYTE *)"G+",NULL};
struct IntuiText GreenMinusText = {2,0,JAM2,4,7,NULL,
```

MACHEN SIE MIT

Wollen Sie sich an der Rubrik »Tips & Tricks« im AMIGA-Magazin beteiligen? Schicken Sie uns Ihre Tips. Am besten, Sie senden uns Ihre Programme und Texte auf Diskette (ASCII-Format). Für Bastelanleitungen fügen Sie bitte einen Bauplan bei. Die Adresse:

AMIGA-Redaktion, Kennwort: Tips & Tricks

Markt & Technik Verlag AG

Hans-Pinsel-Str.2 8013

Haar bei München

Alle Einsendungen werden von uns geprüft. Wenn Ihr Beitrag in die engere Wahl für eine Veröffentlichung kommt, werden Sie benachrichtigt. Haben Sie auf jeden Fall ein wenig Geduld: Vom Zeitpunkt, an dem Ihr Beitrag die Redaktion erreicht, können bis zum Abdruck im AMIGA-Magazin oder Sonderheft zwei bis drei Monate vergehen.

```
(UBYTE *)"G-",NULL);
struct IntuiText BluePlusText = {2,0,JAM2,4,7,NULL,
(UBYTE *)"B+",NULL};
struct IntuiText BlueMinusText = {2,0,JAM2,4,7,NULL,
(UBYTE *)"B-",NULL};
short Pairs[] = {0,0,21,0,21,21,0,21,0,0};

struct Border myBorder = {-1,-1,1,0,JAM1,5,Pairs,NULL};
struct Gadget BlueMinusGadget = {NULL,120,50,20,20,
GADGBOX,RELVERIFY,BOOLGADGET,(APTR)&myBorder,NULL,
&BlueMinusText,NULL,NULL,6,NULL};
struct Gadget BluePlusGadget = {&BlueMinusGadget,120,20,20,20,
GADGBOX,RELVERIFY,BOOLGADGET,(APTR)&myBorder,NULL,
&BluePlusText,NULL,NULL,5,NULL};
struct Gadget GreenMinusGadget = {&BluePlusGadget,85,50,20,20,
GADGBOX,RELVERIFY,BOOLGADGET,(APTR)&myBorder,NULL,
&GreenMinusText,NULL,NULL,4,NULL};
struct Gadget GreenPlusGadget = {&GreenMinusGadget,85,20,20,20,
GADGBOX,RELVERIFY,BOOLGADGET,(APTR)&myBorder,NULL,
&GreenPlusText,NULL,NULL,3,NULL};
struct Gadget RedMinusGadget = {&GreenPlusGadget,50,50,20,20,
GADGBOX,RELVERIFY,BOOLGADGET,(APTR)&myBorder,NULL,
&RedMinusText,NULL,NULL,2,NULL};
struct Gadget RedPlusGadget = {&RedMinusGadget,50,20,20,20,
GADGBOX,RELVERIFY,BOOLGADGET,(APTR)&myBorder,NULL,
&RedPlusText,NULL,NULL,1,NULL};
struct NewWindow MyNewWindow = {0,0,331,80,0,1,CLOSEWINDOW |
GADGETUP|GADGETDOWN, WINDOWDEPTH|WINDOWDRAG|WINDOWCLOSE |
ACTIVATE|WINDOWZIPPING|SMART_REFRESH, &RedPlusGadget,NULL,
(UBYTE *)"PowerColor",NULL,NULL,20,20,640,256,WBENCHSCREEN);
```

```
void Close_All_The_Stuff(){
if(MyWindow!= 0L) CloseWindow(MyWindow);
if(GfxBase!= 0L) CloseLibrary(GfxBase);
if(IntuitionBase!= 0L) CloseLibrary(IntuitionBase);
void Open_All_The_Stuff(){
if(!IntuitionBase = (struct IntuitionBase *)
OpenLibrary("intuition.library", 0L)){Close_All_The_Stuff();
exit(FALSE);}
if(!GfxBase = (struct GfxBase *)
OpenLibrary("graphics.library", 0L)){Close_All_The_Stuff();
exit(FALSE);}
if(!MyWindow = (struct Window *)
OpenWindow(&MyNewWindow)){
printf("Whoops! ...Something wrong with window!?\n");
Close_All_The_Stuff();exit(FALSE);}
void Argument_Error(){
printf("PowerColor\n");
printf("Usage: PowerColor [Farbregister]\n");
void Print(s,x,y)
char *s; int x,y;{
Move(my_rp,x,y); Text(my_rp,s,strlen(s));
void main(ac,av)
int ac; char **av;{
ULONG MessageClass; USHORT code, GadgetNr;
struct Message *GetMsg();
struct Gadget *GadgetPtr;
long RegNum,r = 0,g = 0,b = 0;
if(ac!= 2){Argument_Error(); exit(FALSE);}
else
RegNum = atol(av[1]);
Open_All_The_Stuff();
my_rp = (struct RastPort *)MyWindow->RPort;
my_vp = (struct ViewPort *)ViewPortAddress(MyWindow);
RefreshGadgets(&RedPlusGadget, MyWindow, NULL);
Print("Click at the gadgets", 145L, 25L);
Print("to change the color!", 145L, 35L);
Print("Use 'R' for red,", 145L, 50L);
Print(" 'G' for green", 145L, 60L);
Print("and 'B' for blue.", 145L, 70L);
FOREVER{
Delay(1L);
if(message = (struct IntuiMessage *)
GetMsg(MyWindow->UserPort)){
MessageClass = message->Class;
code = message->Code;
GadgetPtr = (struct Gadget *)message->IAddress;
GadgetNr = GadgetPtr->GadgetID;
ReplyMsg(message);
switch(MessageClass){
case CLOSEWINDOW:
Close_All_The_Stuff();
exit(TRUE);
break;
```

```

case GADGETUP:
case GADGETDOWN:
switch(GadgetNr){
case 1: r++; if(r>15) r = 0;
SetRGB4(my_vp, RegNum, r, g, b);
continue;
case 2: r--; if(r<0) r = 15;
SetRGB4(my_vp, RegNum, r, g, b);
continue;
case 3: g++; if(g>15) g = 0;
SetRGB4(my_vp, RegNum, r, g, b);
continue;
case 4: g--; if(g<0) g = 15;
SetRGB4(my_vp, RegNum, r, g, b);
continue;
case 5: b++; if(b>15) b = 0;
SetRGB4(my_vp, RegNum, r, g, b);
continue;
case 6: b--; if(b<0) b = 15;
SetRGB4(my_vp, RegNum, r, g, b);
continue;
}}}}

```

PowerColor.c

Wie SetRGB.c – aber mit wesentlich mehr Komfort

Move

Das folgende als CLI-Befehl einsetzbare Programm »move« kopiert Dateien in ein anderes Verzeichnis. Das Besondere dabei ist, daß das alte File gelöscht wird und somit nur noch ein Exemplar existiert. Der Aufruf vom CLI:

```
Move [oldfile] [newfile]
```

Sie müssen immer die komplette Pfade angeben, also z.B.:

```
Move dh0:grafik/dpaint/dpaint dh0:leer/dpaint
```

Ein Start von der Workbench ist nicht vorgesehen. Das »Moven« von ganzen Verzeichnissen ist allerdings nicht möglich.

Andreas Stolz/ub

```

/* Programm: Move.c
Kopiert ein File in ein neues Verzeichnis und löscht
die alte Datei, so daß kein Duplikat entsteht.
Aufruf:
Move [oldfile] [newfile]
Compilieraufbau mit Lattice 5.0:
lc -L Move */

#include <libraries/dos.h>
struct DOSLibrary *DOSBase;
struct FileHandle *newf, *oldf, *Open();
void main(), move(), delfile();

void main(argc, argv)
int argc;
char *argv[];
{ if (argc != 3)
{ printf("[oldfile] [newfile]\n");
exit();
}
move(argv[1],argv[2]);
delfile(argv[1]);
}

void move(f1,f2)
char *f1,*f2;
{ register char puffer[10000];
register long anzahl,stat;
oldf = Open(f1, MODE_OLDFILE);
if (oldf == NULL)
{ printf("can't open %s\n",f1);
exit();
}
newf = Open(f2, MODE_NEWFILE);
if (newf == NULL)
{ printf("can't find %s\n",f2);
Close (newf);
}
}

```

```

exit();
}
do
{ anzahl = Read(oldf, puffer,10000);
stat = Write(newf, puffer,anzahl);
}
while (anzahl != NULL);
Close(oldf);
Close(newf);
}

void delfile(f1)
char *f1;
{ int err;
err = DeleteFile(f1);
if (err == NULL)
{ printf("can't delete %s\n",f1);
}
else printf("delete %s\n",f1);
}
}

```

Move.c

Ein wichtiger Befehl zum Verschieben von Dateien

itoa – Lattice hat's

In Ausgabe 3/91 des AMIGA-Magazins wird mangels einer C-Funktion »itoa« eine kurze Assembler-Routine vorgestellt. Zumindestens der Lattice (SAS)-C-Compiler bietet aber eine solche Funktion, sie heißt nur anders. Hinter dem Namen »stci_d« etc. (Handbuch L196) versteckt sich genau diese Funktion. Außerdem gibt es dort die Funktion noch als Version, die eine Zahl als Hexadezimal-Wert ausgibt (»stci_h«). Die entsprechenden Umkehrfunktionen sind ebenfalls vorhanden (L190).

Als Typ wird für diese Funktionen »LATTICE« angegeben; sie ist bei anderen C-Compiler also nicht vorhanden. Die Benutzung ist ganz einfach:

```

Länge = stci_d(char *out,int i);
Länge = stcd_i(char *in,int *out);

```

ub

Link-Library mit Lattice

Beim Erzeugen einer Link-Library in C machen es sich die meisten Leute ein wenig zu einfach: Sämtliche Routinen werden in einer Datei zusammengefaßt, diese kompiliert und dann in »Library.lib« umbenannt (Auch die Benutzung von OML zu dieser Aufgabe, wie bei den Routinen für die indexsequentiellen Dateien im AMIGA-Magazin 12/90 nutzt nichts) Die Überraschung ist groß, wenn man feststellt, daß beim Benutzen nur einer der Funktionen gleich alle eingebunden werden, obwohl sie nicht gebraucht werden.

Die korrekte Lösung ist etwas aufwendiger: Jede Funktion ist in eine eigene Datei zu schreiben und einzeln zu kompilieren. Und dann muß man mit dem entsprechenden Hilfsprogramm (bei SAS/Lattice: OML) diese Dateien zu einer Library zusammenbinden:

```
OML Library.lib r file1.o file2.o file3.o ....
```

Etwas bequemer geht es beim SAS-C noch: Man kann direkt beim Kompilieren angeben, daß eine Objektdatei an eine Library angefügt werden soll:

```
lc [Optionen] -RLibrary.lib File.c
```

Danach werden nur noch die wirklich benötigten Files gelinkt.

ub

C-opperdemo

Was kann man nicht alles mit dem Copper machen? Dieser Coprozessor im Amiga ist ein wahrer Zauberchip. Vor allem für Farbspielereien auf dem Bildschirm setzen Programmierer ihn gerne ein, aber auch für anderes: Ein C-Programm zeigt, wie man den Copper programmiert, um Spiegeleffekte auf den Monitor zu zaubern.

»Copperdemo.c« besteht aus einem Haupt- und drei Unterprogrammen. Das Hauptprogramm (main()) öffnet die benötigten Libraries und einen Lores-Screen mit einem Window. Für die Copperroutine, die den Spiegeleffekt berechnet und in die Copperliste einträgt, wird zusätzlicher Speicher für die Copperliste im System angemeldet (siehe »AllocMem()« im Beispiellisting).

```
/* Copperdemo.c -> Spiegeleffekt mit dem Copper <- */
/* Compiler: Aztek Compiler V. 3.6. */
/* Übersetzungsanweisungen: */
/* cc Copperdemo.c -s +l */
/* ln Copperdemo.o -lc32 */

#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <hardware/custom.h>
#include <exec/memory.h>
#include <exec/types.h>
#include <functions.h>

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct RastPort *RastPort;
struct Screen *Screen;
struct Window *Window;
struct UCopList *UCopList;
struct ViewPort *ViewPort;

/* Screen- und Windowstrukturen */
struct NewScreen NewScreen={0,0,320,256,4,1,2,0,CUSTOMSCREEN,0,
"COPPERDEMO (C) 1991 by Norbert Balk",0,0};
struct NewWindow NewWindow={0,10,320,20,3,4,CLOSEWINDOW,
WINDOWCLOSE|SMART_REFRESH|ACTIVATE|SIZEBRIGHT|WINDOWDEPTH,
0,0,"-- Close für ENDE",0,0,320,20,320,20,CUSTOMSCREEN};
/* Requester-Strukturen, um einen Programmfehler anzuzeigen */
struct IntuiText anzeigen_text={0,0,JAM1,16,6,0,
"ACHTUNG PROGRAMMFEHLER",0,0};
struct IntuiText anzeigen_textII={0,0,JAM1,16,26,0,"TEXT",0,0};
struct IntuiText okay_text={0,0,JAM1,7,4,0,"OK A Y",0,0};

VOID main()
{
if((IntuitionBase = (struct IntuitionBase *)
OpenLibrary("intuition.library",0)) == NULL)
Ende("Intuitionlibrary");
if((GfxBase = (struct GfxBase *)
OpenLibrary("graphics.library",0)) == NULL)
Ende("Graphicslibrary");
if((Screen = OpenScreen(&NewScreen)) == NULL)
Ende("Bildschirm");
NewWindow.Screen = Screen;
if((Window = OpenWindow(&NewWindow)) == NULL) Ende("Window");
UCopList = (struct UCopList*)
AllocMem(sizeof(struct UCopList),MEMF_CHIP|MEMF_CLEAR);
/* Speicher für die UserCopperListe */
if(UCopList==NULL) Ende("Copperlisten");
RastPort=&Screen->RastPort;
ViewPort=&Screen->ViewPort;
Rand(0,50,320,100,2,14); /* Aufbau einer kleinen Grafik */
Rand(0,150,320,200,2,14);
SetAPen(RastPort,1);
SetBPen(RastPort,15);
Move(RastPort,80,78);
Text(RastPort,"Ohne Copper-Spiegelung",22);
Move(RastPort,80,178);
Text(RastPort,"Ohne Copper-Spiegelung",22);
Copper(100,150,320); /* Eine Cop.liste für zwei Spiegelbereiche */
Copper(200,250,320); /* ( 100->150 und 200->250 ) aufbauen */
CEND(UCopList); /* Ende der Copperliste */
Delay(50); /* Kurze Pause */
ViewPort->UCopIns=UCopList; /* Cop.liste in Viewport einbinden */
RemakeDisplay(); /* Viewport neu berechnen. Start neue Cop.liste */
Move(RastPort,80,78);
Text(RastPort," Mit Copper-Spiegelung",22);
Move(RastPort,80,178);
Text(RastPort," Mit Copper-Spiegelung",22);
Wait(1<Window->UserPort->mp_SigBit);
/* Closegadget für Programmende */
Ende();
}
```

```

}
Ende(info) /* Aufruf: "Ende("Fehlerstring")" */
char *info;
{
if(*info!=0){
anzeigen_textII.IText=info;
anzeigen_text.NextText=&anzeigen_textII;
AutoRequest(0,&anzeigen_text,0,&okay_text,0,0,640,80);
}
if(Window) CloseWindow(Window);
if(Screen) CloseScreen(Screen);
if(GfxBase) CloseLibrary(GfxBase);
if(IntuitionBase) CloseLibrary(IntuitionBase);
exit(0);
}
Rand(h1,h2,h3,h4,h5,h6)
/* Aufruf: "Rand(x1,y1,x2,y2,Farbe,Anzahl)" */
SHORT h1,h2,h3,h4,h5,h6;
/* x1,y1,x2,y2 = Angaben in Pixel */
{ SHORT h7;
for(h7=1;h7<=h6;h7=h7+1){
SetAPen(RastPort,h5);
RectFill(RastPort,h1,h2,h3,h4);
h1=h1+1; h2=h2+1; h3=h3+1; h4=h4+1; h5=h5+1;
}
}
Copper(h1,h2,h3) /* Aufruf: "Copper(y1,y2,Bildschirmbreite)" */
SHORT h1,h2,h3; /* Angaben in Pixel. y1,y2 vertikale Positionen */
{
SHORT h4,h5,h6;
h4=(h3/8)*2;
h5=(h2-h1)*h4;
h6=h2+1;
CWAIT(UCopList,h1,0); /* Start des Spiegelbereiches */
CMOVE(UCopList,custom.bpl1mod,(-h4));
/* Eine Bildschirmzeile zurück */
CMOVE(UCopList,custom.bpl2mod,(-h4)); /* Bildschirmbreite/8*2 */
CWAIT(UCopList,h2,0); /* Ende des Spiegelbereiches */
CMOVE(UCopList,custom.bpl1mod,h5); /* Spiegelbereich überspringen */
CMOVE(UCopList,custom.bpl2mod,h5); /* (y2-y1)*Bildschirmbreite/8*2 */
CWAIT(UCopList,h6,0);
CMOVE(UCopList,custom.bpl1mod,0); /* 0 = normales Lesen der */
CMOVE(UCopList,custom.bpl2mod,0); /* Bitplanes bzw. Rasterzeilen */
}
}
```

Copperdemo.c

Spiegeleffekte programmiert man am besten mit Hilfe des Coppers

Damit der Spiegeleffekt deutlicher wird, zeichnet die Routine »RAND« eine kleine Grafik. Bei einem Fehler (z.B.: kein Speicher für den Bildschirm oder für das Fenster vorhanden), wird die Routine »ENDE« aufgerufen, die einen Fehlertext in einem »AutoRequester« anzeigt und geöffnete Bildschirme, Windows und Libraries schließt.

Routinenbeschreibung:

- RAND(x1,y1,x2,y2,Farbe,Anzahl): zeichnet Rechtecke mit (x1,y1) als oberer linken und (x2,y2) als unterer rechten Ecke. Der Parameter »Anzahl« gibt die Zahl der Rechtecke an, die gezeichnet werden sollen. »Farbe« bestimmt die Farbe des ersten Rechtecks; bei allen anderen Rechtecken, die um einen Pixel kleiner als das vorhergehende gezeichnet werden, erhöht sich die Farbnummer um eins.
- ENDE("Fehlertext"): beendet das Demo-Programm und gibt mittels eines Auto-Requesters einen Fehlertext aus. ENDE gibt auch alle benötigten Bildschirme und Libraries ans System zurück.
- COPPER(y1,y2,Bildschirmbreite): die wichtigste Routine des Programms. Sie erstellt die Copperliste für den Spiegeleffekt. Es sind:
 - y1 = Beginn des Bereichs, in dem rückwärts aus den Bitplanes gelesen wird (Änderung der Modulo-Werte);
 - y2 = Ende des Bereichs, in dem rückwärts aus den Bitplanes gelesen wird, und Rücksetzung der Modulo-Werte, um ein normales Lesen aus den Bitplanes zu ermöglichen;
 - Bildschirmbreite = Breite in Pixel des verwendeten Screens, z.B. 320 bei einem Lores-, 640 bei einem Hires-Screen.

Das Beispiellprogramm zeigt, daß man auch in mehreren Bereichen des Bildschirms den Spiegeleffekt einbauen kann, allerdings

dürfen sich die Bereiche nicht überschneiden und der Bereich mit dem kleinsten »y1«-Wert muß zuerst installiert werden. Zum Schluß muß das Programm die Copperliste mit »CEND(UCopList)« beenden und in den ViewPort eintragen (s. Kommentare im Listing).

Norbert Balk/ub

Primzahlen

Die Berechnung von Primzahlen mit dem Computer ist ein recht interessantes Gebiet und beliebtes Thema. Hier kann jeder zeigen, daß er programmieren kann; hier geht's darum, möglichst effektive und schnelle Routinen zu schreiben – und Speicherplatz schonende:

Meist wird zur Berechnung von Primzahlen das Sieb des Eratosthenes benutzt – aber meist auch in einer Bit-verschwendenden Art und Weise.

Das Programm »Prim.c« nutzt jedes Bit des reservierten Bereichs für die Ermittlung von Primzahlen. Auf einem 1-MByte-Amiga kann man einen 320 000 Byte großen zusammenhängenden Speicherblock erhalten. Damit kann man mit dieser Version immerhin die Zahlen bis 5 120 001 überprüfen. Das Programm gibt zusätzlich alle im Bereich enthaltenen Primzahlen-Zwillinge aus (2 und 3 werden wohl nicht als Zwilling gerechnet).

Rüdiger Dreier/ub

```
/* Programm zum Berechnen von Primzahlen */
/* © 1991 by Rüdiger Dreier */
#include <exec/types.h>
#include <proto/exec.h>
#include <proto/dos.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define SIZE 320000
/* Jedes char gibt einem 8 Bits */
/* Ergibt Zahlenbereich bis MAXNUMBER*2+3 */
#define MAXNUMBER 8*SIZE-1
/* Enthält die Flags, ob Primzahl oder nicht */
/* 0: Primzahl */
/* 1: keine Primzahl */
char __far PrimFlags[SIZE]; /* Prototypes */
VOID Ausgabe(VOID);
VOID AusgabeZwillinge(VOID);
VOID SetFlag(ULONG Nummer);
SHORT GetFlag(ULONG Nummer);

main()
{
    ULONG i, /* Laufvariable */
    k, /* Laufvariable */
    PZahl, /* Primzahl */
    count; /* Zähler */
    /* Um den Fehler beim Löschen des globalen Speichers zu umgehen */
    for(i = 0; i < SIZE; i++)
    {
        PrimFlags[i] = 0;
    }
    count = 1; /* Die 2 ist schon als Primzahl bekannt */
    for(i = 0; i < MAXNUMBER; i++)
    {
        if(!GetFlag(i))
        {
            /* PZahl ist die nächste Primzahl */
            PZahl = i+3;
            /* Die Vielfachen einer Primzahl sind keine Primzahl */
            for(k = i+PZahl; k < MAXNUMBER; k += PZahl)
            {
                SetFlag(k);
            }
            count++;
        }
    }
    printf("Habe %ld Primzahlen gefunden.\n", count);
    Ausgabe();
    AusgabeZwillinge();
    /* Diese Routine gibt die gefundenen Primzahlen aus */
    /* ACHTUNG : Kann sehr lange dauern..... */
    VOID Ausgabe(VOID);
}
```

```
ULONG i;
SHORT count = 1;
printf("%lld", 2); /* Die einzige Ausnahme */
for (i = 0; i < MAXNUMBER; i++)
{
    if (!GetFlag(i))
    {
        printf("%lld", 2*i+3);
        count++;
        if (count == 7)
        {
            count = 0;
            printf("\n");
        }
    }
}
printf("\n");
/* Gibt Primzahlzwillinge aus */
/* ACHTUNG : Kann sehr lange dauern..... */
VOID AusgabeZwillinge(VOID)
{
    ULONG i;
    ULONG count = 0;
    BOOL this, next;
    next = GetFlag(0);
    for (i = 0; i < MAXNUMBER; i++)
    {
        this = next;
        next = GetFlag(i+1);
        if (! (this || next) )
        {
            count++;
            printf("Nr: %lld : %lld %lld\n", count, 2*i+3, 2*i+5);
        }
    }
    printf("\nHabe %ld gefunden\n", count);
} /* Setzt Bit Nummer auf eins */
VOID SetFlag(ULONG Nummer)
{
    ULONG Byte;
    SHORT Bit;
    char add;
    Byte = Nummer/8;
    if(Byte > SIZE-1) return; /* Zu großer Wert... */
    Bit = Nummer-Byte*8;
    add = 1<Bit; /* Das passende Bit setzen */
    PrimFlags[Byte] = PrimFlags[Byte] | add; /* Und speichern */
}
/* Prüft den Wert von Bit Nummer */
SHORT GetFlag(ULONG Nummer)
{
    ULONG Byte;
    SHORT Bit;
    char get;
    Byte = Nummer/8;
    if(Byte > SIZE-1) return(1); /* Zu großer Wert, keine Primzahl */
    Bit = Nummer-Byte*8;
    get = PrimFlags[Byte] & (1<Bit);
    Bit = (get != 0); /* Wenn das Bit gesetzt war, ist Bit 1 */
    return(Bit);
}
```

Prim.c

Primzahlen mit dem Computer zu berechnen, ist eine interessante Programmieraufgabe

Nicht zu dynamisch

Bei Programmen, die intensiv dynamische Speicherverwaltung einsetzen, führen bereits kleine Programmierfehler zu zu nicht reproduzierbaren Speicherverluste. Besonders bei dynamischer Stringverwaltung mit berechneten Speichergrößen kann ein kleiner Tippfehler oder eine Unachtsamkeit katastrophale Folgen haben. Der Grund liegt in der Speicherverwaltung des Amiga. Jede angeforderte Menge Speicher wird auf ein Vielfaches von 8 aufgerundet. Desgleichen gilt bei der Freigabe von Speicher. Wird nun ein z.B. 9 Byte langer Bereich reserviert, aber (warum auch immer) nur ein 8 Byte

langer Bereich freigegeben, fehlen einem am Ende gleich 7 Byte. Wenn dann die Länge der angeforderten Speicherbereiche auch noch von Benutzereingaben abhängt, kann man sehr lange nach einem solchen Fehler suchen.

Abhilfe schaffen Programme wie MemTrace (Fish 163, Author: Jojo Wesener). Es protokolliert jeden Aufruf von »AllocMem()« und »FreeMem()«, und lenkt sie in eine eigene Routine um. So können Programmierer feststellen, ob der gesamte Speicher freigegeben wurde und ob Speicher freigegeben wurde, der gar nicht (mehr) belegt war.

Benutzung: In JEDEM Teil des Programms

```
#include "mem.h"
```

einfügen. Außerdem immer MEMTRACE definieren, z.B. Lattice kompilieren mit:

```
-dMEMTRACE
```

Außerdem muß man vor dem Ende des Programms »freeall()« aufrufen und schließlich noch mit »mem.o« linken. Dann werden einem alle Fehler gemeldet – fast. Der oben beschriebene Fehler war dem Autor wohl zu unwahrscheinlich. Aber es ist nicht schwer, eine solche Abfrage mit einzubauen, alles nötige ist bereits vorbereitet:

Man braucht nur ein paar Zeilen in mem.c einfügen:

```
VOID freetrack(char *addr, LONG amt, char *file, char *line)
{
    membkt_t *bkt;
    if( (bkt = find_bkt( addr )) == NULL )
    {
        printf("freetrack: error >%s<, %d, 0x%x -- 0x%x\n", file, line,
            addr, amt);
        return;
    }
    /* Die folgenden Zeilen einfügen */
    if(bkt->amt != amt)
    {
        printf("freetrack: error >%s<, %d, 0x%x -- 0x%x <->
0x%x\n", file, line,
            addr, amt, bkt->amt);
    }
    /* Einfügen ENDE */
    /* Die folgende Zeile ändern */
    (void)FreeMem(addr, bkt->amt);
    /* Ändern ENDE */
    put_bkt(bkt);
}
```

mem.c

Mit ein paar zusätzlichen Zeichen noch sicherer

So werden einem auch diese Fehler noch berichtet. Bei der Benutzung von MemTrace treten keine FreeTwice-Errors mehr auf und es geht kein Speicher mehr verloren. Die Programmentwicklung macht so viel mehr Spaß.

Rüdiger Dreier/ub

MutualExclude-Gadget

Die Gadget-Struktur enthält neben bekannten und vielbenutzten Einträgen auch eine Long-Variable namens »MutualExclude«, die eine ähnliche Funktion wie eine Variable gleichen Namens in der »MenuItem«-Struktur besitzt.

Das könnte man bei TOGGLESELECT-Gadgets dazu verwenden, bei Aktivierung eines Gadgets ein oder mehrere andere (maximal 32) Gadgets gleichzeitig zu deaktivieren, wäre da nicht ein kleiner Haken: Die Variable wird von den derzeitigen Betriebssystemversionen 1.2 und 1.3 noch nicht unterstützt. Folglich bleibt dem von allen Göttern verlassenem User nur übrig, sich selbst eine solche Routine zu schreiben, die diese Variable verwendet und den oben beschriebenen Zweck erfüllt.

Die Routine ist dabei auf die Nummerierung der Gadgets durch den Programmierer in der eh schon vorhandenen USHORT-Varia-

blen »Gadget.GadgetID« und auf eine geöffnete »intuition.library« angewiesen und kann bei der Gadgetabfrage gleich nach dem Befehl »ReplyMsg()« aufgerufen werden. Die Nummern der Gadgets, die deaktiviert werden sollen, stellt man nun wie in der »MenuItem«-Struktur ein: Soll also das Gadget mit der GadgetID 1 beim Anklicken deaktiviert werden, wird das erste Bit der »MutualExclude«-Variable gesetzt, d.h. die Variable erhält den Wert 1. Will man das zweite Gadget ansprechen, wird das zweite Bit gesetzt, usw. Beispiele zeigt die Tabelle:

zu deaktivierende Gadgets	MutualExclude (binär)	MutualExclude (dezimal)
1	1	1
2	10	2
1 und 2	11	3
3	100	4
3 und 1	101	5

Die Funktion in der Grundform ermöglicht nicht, daß alle Gadgets gleichzeitig deaktiviert sein können, was im Anwendungsfall seltener vorkommen dürfte. Wollen Sie diese Möglichkeit offenhalten, entfernen Sie einfach die entsprechende Abfrage.

Das Listing »GadgetDemo.c« zeigt ein Beispiel für den Einsatz unserer Gadget-Funktion. Verwendet wurde der Aztec-C-Compiler, Compiler- und Linkeraufruf lauten:

```
CC Demo
```

```
LN Demo -lc
```

Steffen Nebel/ub

```
void mexclude(struct Gadget *actgadget, struct Gadget *selected)
/* Parameter sind ein Pointer, der auf eine Gadget-Struktur zeigt,
von der ab sich die Funktion durch alle folgenden durchhangelt
und ein Pointer auf das gerade aufgerufene Gadget */

{
    unsigned short gadgetid;
    /* Variable, in der die aktuelle Gadgetnummer abgelegt wird */

    if((selected->Flags & SELECTED) == NULL)
    /* Falls das angeklickte Gadget schon aktiviert war, */
    /* wird es wieder in seinen ursprünglichen Zustand versetzt */
    /* und neu gezeichnet, dadurch ist immer ein Gadget aktiviert */

    { selected->Flags ^= SELECTED;
      RefreshGList(selected, Window, NULL, 1);
    }
    else /* andernfalls, */
    {
        do /* Schleifenbeginn.... */
        {
            if(actgadget->Activation & TOGGLESELECT)
            /* Falls aktuelles Gadget vom TOGGLESELECT-Typ ist, */

            if((gadgetid = actgadget->GadgetID) < 32)
            /* falls Gadgetnummer < 32, Nummern > 32 können durch
            MutualExclude nicht mehr angesprochen werden, */

            if(selected->MutualExclude & (1L << (gadgetid - 1L)))
            /* und falls schließlich das Bit gemäß der GadgetID
            gesetzt ist, das Gadget also deaktiviert werden muß */

            if(actgadget->Flags & SELECTED)
            /* falls Gadget aktiviert ist, */

            {
                actgadget->Flags ^= SELECTED; /* Gadget deaktivieren */
                RefreshGList(actgadget, Window, NULL, 1);
                /* Gadget neu zeichnen */
            }

            while((actgadget = actgadget->NextGadget) != NULL);
        } /* Schleife durchlaufen, bis Ende der Gadget-Liste erreicht */
    }
}
```

MutualExclude.c

Ein MutualExclude-Bit für Gadgets wird von OS1.3 nicht unterstützt – aber selbst ist der Programmierer

Warnungen

»Beep.c« läßt den Bildschirm ein- bis neunmal aufblitzen, was man sehr sinnvoll einsetzen kann, z.B. so:

Stellen Sie sich vor, Sie schreiben gerade an einem Text, und lassen im Hintergrund eine Batch-Datei ablaufen. Wie stellen Sie fest, ob alles normal läuft oder ob es irgendwelche Fehler gibt, die vielleicht gar zum Abbruch führen?

Sie können zwar über IF..WARN..ERROR Fehler abfragen, und über den ECHO-Befehl Fehlermeldungen ausgeben, aber wenn Sie im aktuellen Fenster gerade die Ausgabe ins >NIL: umleiten, ist alles umsonst. Auch ein Bildschirmblinken, das mit dem Befehl ECHO »<Ctrl G>« ausgegeben werden kann, bietet keine optimale Lösung.

Mit dem folgenden Utility erfahren Sie sofort, wann ein Fehler auftritt und wenn, wie schwer er ist. Ein leichter Fehler (Errorcode WARN) läßt z.B. zweimal blinken, ein schwerer Fehler (Errorcode FAIL) viermal.

ub

```
/* BEEP.c
Flash workbench screen <n> times
Usage: BEEP <n> [0..9]
NUR VOM CLI AUS AUFRUFEN !
Compilierungsanweisungen für SAS/C V5.1 bzw. Lattice:
lc -rr -v -mt -O -cismug beep
blink beep.o      (Kein Startup-Modul linken)
*/
int main(char*);
#include <proto/dos.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <string.h>
#include <exec/memory.h>
#define printf(s) Write(output,s,strlen(s))
int main(line)
char *line;
{
    struct DosBase *DOSBase;
    struct IntuitionBase *IntuitionBase;
    struct Screen *scr;
    BPTR output;
    UBYTE n,i;
    if (!(DOSBase = (struct DosBase *)
        OldOpenLibrary("dos.library")))
        return(20);
    if !(IntuitionBase = (struct IntuitionBase *)
        OldOpenLibrary("intuition.library"))
        return(20);
    output = Output();
    n = *line-'0';
    if (n>9){
        printf("\x1b[1mUsage: \x1b[0m\x1b[33mBEEP <n> [0..9]\n"
            "\x1b[31m\nFlash Workbench Screen <n> times.\n");
        return(5);
    }
    scr = (struct Screen*)OpenWorkBench();
    for(i = 0; i < n; i++) {
        DisplayBeep(scr);
        Delay(TICKS_PER_SECOND / 3);
    }
    /* Warte 1/3 Sekunde zwischen zwei Display Beeps */
    return(0);
}
```

Beep.c

Wenn Sie wirklich sehen wollen, ob irgendwo etwas nicht in Ordnung ist

SAS/C Fehler behoben

Mit diesem Tip können Sie ein Programm aus dem SAS/C (V 5.1) Compilerpaket korrigieren, indem Sie es patchen.

Bekanntlicherweise hat der Compiler jetzt ein praktisches Workbench-Interface, von dem aus man die Compileroptionen komfortabel anwählt. Das Programm, mit dem man die Einstellungen vor-

nimmt, heißt »SASCOptions«. Es befindet sich normalerweise im Verzeichnis »SYS:Prefs«.

In diesem neuen Feature steckt natürlich, wie könnte es anders sein, ein Fehler oder, genaugenommen, eine Verwechslung. Wenn man nämlich unter »Advanced Options« die Einstellung »Nested Comments« (verschachtelte Kommentare) einstellt, meldet der Compiler beim nächsten Durchlauf einen »unrecognized Option«-Fehler.

Ein Blick in die Umgebungsvariable »SASCOPTS« klärt das Rätsel auf: Dort wurde die Compileroption »-cn« selektiert, die aber gar nicht existiert – hier hat der Programmierer wohl an »n« für »nested« gedacht und nicht an »C« wie »C«omments, denn die richtige Option wäre »-cc«.

Ein wirklich ärgerlicher Fehler, aber die Lösung ist erfreulich einfach. Man nehme einfach einen Filemonitor (z.B. »NewZap«) oder einen guten Editor (z.B. »CygusEd«) und suche nach der Zeichenfolge »-cn«. Sie wird nur einmal gefunden (etwa bei Byte 17400) und kann nun gefahrlos gegen »-cc« vertauscht werden. Genauso verfahren Sie mit der Zeichenkette »suwndei« (bei Byte 18200), die durch »suwcdci« ersetzt werden muß.

Wer es noch einfacher haben möchte, benutzt »RepString« (ein Utility von Fish 180). Die Aufrufe lauten:

```
RepString SASCOptions dummy -cn -cc
RepString dummy NewSASCOptions suwndei suwcdci
delete dummy
```

Nachdem das Programm zweimal gemeldet hat:

```
Changes 000001 *** Finished ***
```

haben Sie eine verbesserte Version der »SASCOptions« mit dem Namen »NewSASCOptions«. Sie sollten das alte Programm löschen oder umbenennen und dem neuen Programm den Namen »SASCOptions« geben.

ub

Herein, Heraus

Sicherlich haben Sie schon folgende oder ähnliche Befehlszeilen direkt in der Shell oder in Batch-Dateien eingegeben, um etwas auf dem Drucker auszugeben:

```
type >prt: $filename
```

Hierbei wird bekannterweise die (Standard-) Ausgabe umgelenkt, in diesem Fall zum Drucker. Allerdings kann es schon mal passieren, daß ein Befehl aus irgendeinem Grund nicht korrekt abläuft und stattdessen eine Fehlermeldung ausgibt. Und klar, was schiefgehen kann, wird auch schiefgehen – laut Murphys Law –, und so landet die Fehlermeldung auf dem Drucker, wodurch mal wieder ein fast sauberes weißes DIN-A-4-Blatt in den Papierkorb wandern darf. Aber es könnte noch schlimmer kommen: Wird eine Fehlermeldung nach »nil:« umgeleitet, bekommt man sie wohl nie zu Gesicht und das artet zu einer zeitraubenden Fehlersuche aus.

Um die Ausgabe von Fehlermeldungen mit absoluter Sicherheit auf den Bildschirm zu leiten, gibt es in der Programmiersprache C den durch den »Startup-Code« vorinitialisierten Filezeiger »stderr«, der immer auf das aktuelle Fenster des aufgerufenen Programms verweist. »stderr« wird folgendermaßen verwendet:

```
if (Error_aufgetreten)
{
    fprintf(stderr,"Es ist ein Fehler aufgetreten !\n");
    exit (RETURN_ERROR);
}
else
{
    fprintf(stdout,"Daten rollen nun auf Std.-Ausg.-Gerät, juchee !\n");
}
```

Mit der »fprintf«-Funktion geben wir hier an, wohin die Daten geschickt werden sollen:

- stdin: steht für die Standard-Eingabe;
- stdout: für die Standard-Ausgabe;
- stderr: für das aktuelle Fenster als Ausgabe

Dabei fungiert »printf()« als Spezialfall von »fprintf()« für:

```
fprintf(stdout,...);
```

Auch in anderen Programmiersprachen stehen solche oder ähnliche Möglichkeiten offen, die Sie man auch nutzen sollten, um bedienungsfreundlichere Software zu entwickeln.

Thomas Becker/ik

Highspeed-Amiga: Bildschirm aus

Bei einigen Anwendungen erweist es sich als sinnvoll, die Rechengeschwindigkeit des Amiga heraufzusetzen. Das ist ohne weiteres realisierbar, kennt man erst einmal die Bedeutung der einzelnen Bits des DMACON-Registers. Zuerst einmal eine Lösung für C-Programmierer. In der Include-Datei »graphics/gfxmacros.h« findet man die Macros »OFF_DISPLAY« und »ON_DISPLAY«:

• OFF_DISPLAY sperrt die DMA-Kanäle (Direkt Memory Access = direkter Zugriff auf den Speicher), die für die Bildschirmdarstellung erforderlich sind. Damit erhält der Prozessor mehr Buszyklen. Seine Operationen werden beschleunigt.

Eine Lösung in Assembler zeigt das folgende Beispiel:

```
start:
    move.w #$01C0, $DFF096 ;löschen der Bits 6,7 und 8
loop:
    btst #6,$BFE001
    bne loop ; auf Maustaste warten
    move.w #$83C0, $DFF096 ; setzen der Bits 6,7,8 und 9
rts
```

Das Programm sperrt alle für die Darstellung eines Bildes erforderlichen DMA-Kanäle. Man kann das Register nicht einfach beschreiben, sondern muß den Aufbau des Registers berücksichtigen.

Anhand der Liste fällt es leicht, die verschiedenen DMA-Kanäle zu sperren und wieder freizugeben. Die entsprechende Lösung in BASIC ist denkbar einfach, zum Beispiel für den Copper:

```
Copper_sperren: POKEN 14676118,448
Copper_freie: POKEN 14676118,33728
```

Durch diese Maßnahme wird die Rechenngeschwindigkeit des Amiga um etwa 10% erhöht. Das lohnt sich auf jeden Fall bei rechenintensiven Programmen.

Thomas Gehling, Stefan Klumpers/ub

DMACON : Adresse \$DFF096

Bit-Nummer	Name	Funktion
0-3	AUDxEN	Audio-DMA für die Tonkanäle einschalten
4	DSKLEN	Disk-DMA einschalten
5	SPREN	Sprites-DMA zulassen
6	BLTEN	Blitter-DMA
7	COPEN	Copper-DMA
8	BPLEN	Bitplane-DMA
9	DMAEN	Kontroll-Bit für alle Kanäle
		Bit = 0 : DMA gesperrt
		Bit = 1 : DMA erlaubt
10-14		unwichtig für diese Anwendung
15	Set/Clr	Bits setzen/löschen
		Bit =0 : adressierte Bits löschen
		Bit =1: adressierte Bits setzen

Macro in voller Aktion

Schauen wir uns die Verwendung des oben beschriebenen Macros in einem Beispielprogramm in C an:

Das Programm »Display_aus.c« schaltet den Bildschirm für kurze Zeit aus. Verwenden Sie die Makros z.B., wenn Sie eine Grafik auf dem Bildschirm aufbauen. Erstens läuft ein Programm bei ausgeschaltetem BitPlane_DMA schneller, zweitens wird das Bild erst gezeigt, wenn es komplett ist.

Lars Schrix/ub

```
#include <exec/types.h>
#include <graphics/gfxmacros.h>

main()
{
    int i ;
    OFF_SCREEN;

    for (i=0;i<10000;i++);

    ON_SCREEN;
}
```

Display_aus.c

DISPLAY-Makros in voller Aktion

Ist irgendwo ein Fenster auf?

Sie wollen einen Überblick der zur Zeit geöffneten Screens und Fenster? Ganz einfach, nehmen Sie das Programm »Screen.c«:

```
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>

struct IntuitionBase *IntuitionBase;
struct Screen *sc;
struct Window *wi;

void main()
{
    IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library",0);
    sc = IntuitionBase->FirstScreen;

    while (sc != 0)
    {
        printf("SCREEN : (structure at %x) %s\n",sc,sc->Title);
        wi = sc->FirstWindow;

        while (wi != 0)
        {
            printf("WINDOW : (structure at %x) %s\n",wi,wi->Title);
            wi = wi->NextWindow;
        }

        sc = sc->NextScreen;
    }
    CloseLibrary(IntuitionBase);
}
```

Screen.c

Zeigt alle Fenster und Screens

Das C-Programm listet alle vorhandenen Screens und Windows auf, indem es deren Namen und die Adressen der Kontrollstrukturen ausgibt.

Peter Worofka/ub

Anmeldeliste für Geräte

Ein nützliches Tool für alle Programmierer ist das C-Programm »Device.c«. Es gibt die Namen und Adressen der Message-Ports aller dem Amiga angemeldeten Geräte aus.

Die Aufrufe zum Übersetzen und Linken mit dem Aztec-C-Compiler lauten:

```
cc device.c +L -S
ln device.o -LC32
```

Wenn Sie das C-Programm übersetzt haben, kopieren Sie den neuen Befehl »device« am besten in den C-Ordner Ihrer Workbench.

Peter Worofka/ub

```
#include <libraries/dosextern.h>

extern struct DosLibrary *DOSBase;
struct DeviceList *dev;

void main()
{
    dev = (struct DeviceList *) BADDR(((struct DosInfo *) BADDR(((struct
    RootNode *) (DOSBase->dl_Root))>rn_Info)) ->di_DevInfo);
    while (dev != 0)
    {
        switch ((int) (dev->dl_Type))
        {
            case DLT_DEVICE : printf("DEVICE : ");break;
            case DLT_DIRECTORY : printf("Directory : ");break;
            case DLT_VOLUME : printf("Volume : ");break;
            default : printf("Unknown : ");
        }
        printf("(Handler MsgPort at %6x) ",dev->dl_Task);
        Write(Output(),
        ,BADDR(dev->dl_Name)+1
        ,(int) *((char *)BADDR(dev->dl_Name)));
        printf(":\n");
        dev = (struct DeviceList *) BADDR(dev->dl_Next);
    }
}
```

Device.c

Ein praktisches Tool: Es zeigt auch Message-Ports an

Vollbremsung

Es war einmal.... In Zeiten des C 64 brauchte man sich über Warteschleifen keine Gedanken zu machen.

In BASIC ging das so:

```
FOR I = 1 TO 1000 : NEXT
```

Wie realisiert man Warteschleifen auf dem Amiga? Die korrekte Antwort lautet: »gar nicht«. Warteschleifen sind verpönt; sie vergeuden nur Rechenzeit. Ein Multitasking-Computer sollte nicht warten.

Das Mittel der Stunde ist »Delay()« aus der »DOS.library«. Der Aufruf in C:

```
Delay( Time );
```

Der Effekt: Der Aufruf hält das ablaufende Programm um die in »Time« angegebene Anzahl von 1/50 Sekunden an. Die gesamte Rechenzeit steht anderen Tasks zur Verfügung.

Claudio Pitrella/ub

Sprechen Sie C?

Das folgende C-Programm »Speak.c« zeigt, wie Sie in C Sprache ausgeben:

```
/* Speak.c
Übersetzung CC Speak.c +L */

#include <exec/types.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <libraries/translator.h>
#include <devices/narrator.h>

ULONG TranslatorBase;

struct MsgPort *narratorport;
struct MsgPort *CreatePort();
struct narrator_rb request;

UBYTE original[]={"Hello, i am the amiga*"};
UBYTE ready[500];
BYTE Kanal[4]={ 3,5,10,12 };
BOOL flag=FALSE;
```

```
main()
{ int fehler;
  TranslatorBase = (struct Library *)
    OpenLibrary("translator.library",0L);
  if(TranslatorBase == NULL) Close_All("No translator.library!");
  narratorport=CreatePort("narrator.port",0);
  if(narratorport == NULL) Close_All("No narrator.port!");
  if( OpenDevice("narrator.device",0,&request,0) )
    Close_All("No narrator.device!");
  flag = TRUE;
  fehler = Translate(original,sizeof(original),ready,sizeof(ready));

  request.message.io_Message.mn_ReplyPort = narratorport;
  request.message.io_Command = CMD_WRITE;
  request.message.io_Data = (APTR) ready;
  request.message.io_Length = sizeof(ready);
  request.rate = 170L;
  request.volume = 64L;
  request.sex = FEMALE;
  request.mode = MALE;
  request.pitch = 110L;
  request.sampfreq = 20000L;
  request.ch_masks = Kanal;
  request.nm_masks = sizeof(Kanal);

  DoIO(&request);
  Close_All("END!");
}

Close_All(error)
char error[];

{
  puts(error);
  if(TranslatorBase) CloseLibrary(TranslatorBase);
  if(flag) CloseDevice(&request);
  if(narratorport) DeletePort(narratorport);
  exit(0);
}

else
  printf("linke Taste gedrückt\n");
  if(IntuitionBase) CloseLibrary(IntuitionBase);
}
```

Speak.c

Sprachausgabe in C programmiert

Sie müssen als erstes die »translator.library« zum Übersetzen und das »narrator.device« zur Ausgabe öffnen. Die Struktur »narrator_rb« ist mit den nötigen Werten zu füllen. »Translate()« übersetzt einen String in Phonem-Code, der ans »narrator.device« übergeben wird.

Über »DoIO()« startet das Programm die Sprachausgabe. Sie können mit der gezeigten Methode Ihre Programme leicht mit Sprachausgaben ausstatten, doch beachten Sie, daß die »translator.library« (LIBS:) und das »narrator.device« (DEVS:) in den angegebenen Verzeichnissen stehen müssen.

Sascha Wilkens/ik

»C«alender

»Cal.c« erstellt einen Monatskalender, wie Sie ihn von Wandkalendern her kennen. »Cal« demonstriert anschaulich die Mächtigkeit der Zeit- und Datumfunktionen unter ANSI-C.

Bitte beachten Sie jedoch den Gültigkeitsbereich: Sie können sich jeden Monat im Bereich von 1970 bis 2037 anzeigen lassen.

»Cal.c« wird im CLI nach folgendem Schema aufgerufen:

```
cal [monat [jahr]]
```

Dabei kann »monat« die Werte eins bis 12 annehmen. »jahr« gibt das Jahr (1970 - 2037) an, für das der Amiga den Monatskalender zeigen soll. Geben Sie »jahr« nicht an, setzt »Cal« das laufende Jahr ein. Beispiel:

```
cal 10 1991
```


Ohne Parameter gibt das Programm den laufenden Monat aus. Um eine korrekte Anzeige zu erhalten, muß die Systemuhr natürlich richtig eingestellt sein.

Gerd Wienand/ik

```
/* cal.c - version 1.0, 04.Mar.91 */
/* Print calendar */
/* Compile: cc cal.c -safnprs; */
/* Link: ln cal.o -lc */

#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <time.h>
#include <functions.h>
char *monthname[] =
{ "January", "February", "March", "April", "May", "June", "July",
  "August", "September", "October", "November", "December" };
char monthlen[12] =
{ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
void main(long argc, char **argv)
{ struct tm *tm;
  unsigned long month, year;
  long day;
  short col = 0L;
  time_t t;
  if(argc == 2L && *argv[1] == '-')
    /* Gebrauchsanweisung und exit() */
    printf("Print calendar\nV1.0 \"__DATE__\" by --gw--\n\n"
      "Usage: %s [month 1-12] [year 1970-2037]\n", argv[0]);
  else
  { t = time(NULL);
    tm = localtime(&t); /* aktuelle tm-struct */
    month = (argc > 1L) ? atol(argv[1]) - 1L : tm->tm_mon;
    year = (argc > 2L) ? atol(argv[2]) - 1900L : tm->tm_year;
    if(month > 11L || (year < 70L || year > 137L))
      printf("%s: Operand außerhalb Gültigkeit\n", argv[0]);
    else
    { tm->tm_mday = 0L; /* erster Tag im Monat */
      tm->tm_mon = month;
      tm->tm_year = year;
      t = mktime(tm);
      tm = localtime(&t);
      day = (long)(-1L - tm->tm_wday) % 7L;
      monthlen[1] += !(year % 4L);
      printf(" %s %ld\nSu Mo Tu We Th Fr Sa",
        Monthname[month], Jahr+1900L);
      while(day++ < (long)monthlen[month])
      { if(!(col++ % 7L)) /* line feed? */
        printf("\n");
        if(day <= 0L)
          printf(" ");
        else
          printf("%2d ", day);
      }
      printf("\n");
    }
  }
}
```

Cal.c

Ein sog. »C«alender-Programm

Icons automatisch öffnen

Sie wollen direkt nach dem Start der Workbench einige Schubladen Ihrer Startdiskette geöffnet auf dem Bildschirm vorfinden?

»IconOpen.c« erledigt genau diese Aufgabe. Es simuliert einen doppelten Mausklick über dem gewünschten Icon.

Um seine Arbeit ordentlich zu verrichten, benötigt »Icon Open.c« allerdings genaue Angaben darüber, welches Icon es auswählen soll. Der Aufruf (aus der »Startup-Sequence«):

IconOpen Fenstername IconPfad

Dabei sind:

- **Fenstername:** der Name, des Fensters, in dem das Icon enthalten ist. Wenn Sie den Fensteramen nicht wissen, sehen Sie einfach auf der Workbench in der Titelleiste des entsprechenden Fensters

nach. Der Fenstername entspricht immer dem Namen des Verzeichnisses, in dem sich das Icon befindet (ohne Pfadangabe). Achten Sie auf Groß-/Kleinschreibung. Für Disk-Icons heißt der Fenstername »Workbench«.

- **IconPfad:** Geben Sie hier den vollständigen Pfadnamen des Icons an (allerdings ohne die Erweiterung ».icon«).

Einige Beispiele:

IconOpen Workbench df0:Disk

simuliert einen Doppelklick auf das Icon des Laufwerks »df0:«.

IconOpen "AMIGA-HD" dh0:Beckertext

Hier wird ein Doppelklick auf das Icon »dh0:Beckertext« vorgetauscht; der Fenstername lautet »AMIGA-HD«.

»IconOpen« eignet sich, bereits nach dem Start des Amiga die gewohnte Arbeitsumgebung einzurichten. Dazu bietet es sich an, das Programm in der »Startup-Sequence« aufzurufen. Selbstverständlich funktioniert »IconOpen.c« nur, wenn die Workbench mit LOADWB bereits aktiviert ist.

Gerd Wienand/ik

```
/* IconOpen - version 1.0, 04.Mar.91 */
/* open workbench icon by doubleclick-simulation */
/* compile: cc iconopen.c -safnprs; */
/* link: ln iconopen.o -lc */

#include <devices/input.h>
#include <devices/inputevent.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <workbench/icon.h>
#include <workbench/workbench.h>
#include <string.h>
#include <stdio.h>
#include <functions.h>
struct IntuitionBase *IntuitionBase;
struct IconBase *IconBase;
struct MsgPort *mport;
struct IOStdReq *req;
struct Window *win;
struct DiskObject *dob;
short i;
struct InputEvent ie;
void main(long argc, char **argv)
{ if (argc != 3L) /* wrong number of params */
  puts("Open WorkBench Icon\nV1.0 \"__DATE__\" by --gw--"
    "Usage: IconOpen <Fenstername> <icon-Pfad ohne"
    "\".info\"-suffix>");
  else if(IntuitionBase = OpenLibrary("intuition.library", 0L))
  { if (IconBase = OpenLibrary("icon.library", 0L))
    {
      if(mport = (struct MsgPort *)CreatePort(NULL, 0L))
      { if (req = CreateStdIO(mport))
        { if (!OpenDevice("input.device", 0L,
          (struct IORequest *)req, 0L))
          { req->io_Command = IND_WRITEEVENT;
            req->io_Flags = 0;
            req->io_Length = sizeof(struct InputEvent);
            req->io_Data = (APTR)&ie;
            win = IntuitionBase->ActiveScreen->FirstWindow;
            while(win) /* look for the window */
            { if(!(strcmp(argv[1], (char *)win->Title)))
              /* case sensitive */
              { WindowToFront(win);
                if(dob = GetDiskObject(argv[2]))
                /* get icon from disk */
                { /* place mousepointer over that icon */
                  ie.ie_Class = IECLASS_POINTERPOS;
                  ie.ie_Code = IECODE_NOBUTTON;
                  ie.ie_X = dob->do_CurrentX + win->LeftEdge + 24;
                  ie.ie_Y = (dob->do_CurrentY + win->TopEdge + 12) << 1;
                  DoIO((struct IORequest *)req);
                  for(i = 2; i-->0)
                    /* make two clicks without releasing */
                    { /* left mousebutton (oops!) */
                      ie.ie_Class = IECLASS_RAWMOUSE;
                      ie.ie_Code = IECODE_LBUTTON;
                      ie.ie_Qualifier = IEQUALIFIER_RELATIVEMOUSE;
                      ie.ie_X = 0;
                      ie.ie_Y = 0;
                      DoIO((struct IORequest *)req);
                    }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```

        FreeDiskObject(dob); /* cleanup */
    }
    break;
}
win = win->NextWindow;
}
CloseDevice((struct IORequest *)req);
}
DeleteStdIO(req);
}
DeletePort(mport);
}
CloseLibrary(IconBase);
}
CloseLibrary(IntuitionBase);
})
else
printf("linke Taste gedrückt\n");
if(IntuitionBase) CloseLibrary(IntuitionBase);
}
}

```

IconOpen.c

Direkt nach dem Start: Icons auf der Workbench

Vom String zur Zahl

C-Programmierer haben manchmal Schwierigkeiten, einen (eingebenen) String in eine Hex-Zahl zu verwandeln. Die C-Library einiger Compiler stellt keinen Befehl dafür zur Verfügung. Das Listing »Hex.c« enthält eine Funktion namens »atoh()«, die einen ASCII-String in eine hexadezimale Zahl umwandelt.

```

/* Hex.c */
unsigned char *atoh (String)
char *String;
{
    int i;
    unsigned char *hex;
    unsigned long faktor;
    hex = (unsigned char *)0L;
    faktor = 1L;
    if (strlen (String) <= 8)
        for (i=(strlen (String)-1);i>=0;i--)
            if ((String[i] >= '0') && (String[i] <= '9'))
                {
                    hex += (String[i] - '0')*faktor;
                    faktor *= 0x10;
                }
            if (((String[i] & 223L) >= 'A') &&
                ((String[i] & 223L) <= 'F'))
                {
                    /* Hex-Ziffern */
                    hex += ((String[i] & 223L) - 'A' + 0xa)*faktor;
                    faktor *= 0x10;
                }
        }
    return (hex);
}

```

Hex.c

zeigt die Umwandlung einer Zeichenkette in eine Zahl

Übergeben Sie der Funktion einfach (einen Zeiger auf) einen String und die Funktion liefert als Rückgabewert eine im String enthaltene Hex-Zahl zurück.

Sascha Wilkens/ik

Kürzerer C-Code

Obwohl der Startup-Code des Manx- und Lattice-C-Compilers schon sehr kompakt ist, stört er doch manchmal. Besonders, wenn ein Programm nicht all seine Funktionen benutzt. Durch das Überschreiben einiger C-Library-Funktionen kann der Code erheblich verkürzt werden. Hierzu wird die Funktion mit »{ }« definiert und überschreibt dabei die lange Library-Funktion. Folgende Funktionen eignen sich zu diesem Vorgehen:

- `_main`: Es werden keine Standardausgabekanäle mehr geöffnet; `printf()` und `puts()` dürfen nicht angewendet werden (sofern sie die Kanäle nicht selbst öffnen).
- `_cli_parse`: Die Parameter, die beim Programmaufruf übergeben wurden, werden nicht aufbereitet.
- `_wb_parse`: Die Funktion kann überschrieben werden, wenn ein Programm nicht von der Workbench gestartet wird. Beispiel:

```
_wb_parse() {} ; _cli_parse {}
```

Untersuchen Sie den Startup-Code darauf, ob Funktionen nicht benötigt werden, indem sie ein Programm kompilieren, mit der Option »-W« linken und dann mit einem (symbolischen) Debugger den Code »durchforsten«.

Sascha Wilkens/ik

beep, beep

Manchmal ist es notwendig, den Computerbenutzer mit einem akustischen Signal auf etwas aufmerksam zu machen, z.B. auf einen Fehler oder das Ende einer langen Berechnung.

Der hier vorgestellte Beep-Befehl ist ein Beispiel dafür, wie man den Amiga zur Ausgabe eines Tons variabler Höhe und Dauer überredet. Die Funktion läßt sich leicht in eigene Programme einbauen, wenn Sie darauf geachten, daß das Feld `SoundData[]` im Chip-Memory zu liegen kommt (Linker-Optionen beachten).

»Beep« (als CLI-Befehl) erwartet zwei Parameter:

- die Tonhöhe (period), die im Bereich von 120 bis 3000 liegen sollte
- die Tondauer (cycles)

Der Aufruf vom CLI sieht so aus:

```
Beep period cycles
```

Das folgende Listing wurde mit dem Aztec-C-Compiler V. verfaßt. Compiler- und Linker-Aufrufe stehen im Kopf des Programms.

Gerd Weinand/ik

```

/* beep in a given period and length (cycles)
 * Compile: cc beep.c -sainprs
 * Link: ln beep.o +cd -lc */
#include <exec/types.h>
#include <devices/audio.h>
#include <stdio.h>
#include <stdlib.h>
#include <functions.h>
void _wb_parse() {} /* no wb_args welcome */
struct IOAudio *req;
struct MsgPort *mport;
unsigned char SoundData[] =
{ 100, 120, 110, 80, 100, 100, 127, 90, 50, 0,
  -100, -120, -110, -80, -100, -100, -127, -90, -50, 0 };
unsigned char unit = 0x0f;
void main(long argc, char **argv)
{
    if (argc != 3L) /* wrong number of params */
        {
            puts("Beep\nV1.0 \"_DATE_\" by Gerd --gw-- Weinand\n\n"
                "Usage: Beep <period> <cycles>");
        }
    else if (mport = (struct MsgPort *)CreatePort((char *)NULL, 0L))
        if (req = (struct IOAudio *)CreateExtIO(mport,
            sizeof(struct IOAudio)))
            {
                req->ioa_Data = &unit;
                req->ioa_Length = 1L;
                req->ioa_Request.io_Flags = ADIOF_NOWAIT;
                if (!OpenDevice(AUDIONAME, 0L, (struct IORequest *)req, 0L))
                    {
                        req->ioa_Request.io_Command = CMD_WRITE;
                        req->ioa_Request.io_Flags = ADIOF_PERVOL | IOF_QUICK;
                        req->ioa_Data = (UBYTE *)&SoundData;
                        req->ioa_Length = 20L;
                        req->ioa_Volume = 64L;
                        req->ioa_Period = (unsigned short)atol(argv[1]);
                        req->ioa_Cycles = (unsigned short)atol(argv[2]);
                        BeginIO((struct IORequest *)req);
                        WaitIO((struct IORequest *)req);
                        CloseDevice((struct IORequest *)req);
                        DeleteExtIO((struct IORequest *)req);
                        DeletePort(mport);
                    }
            }
}

```

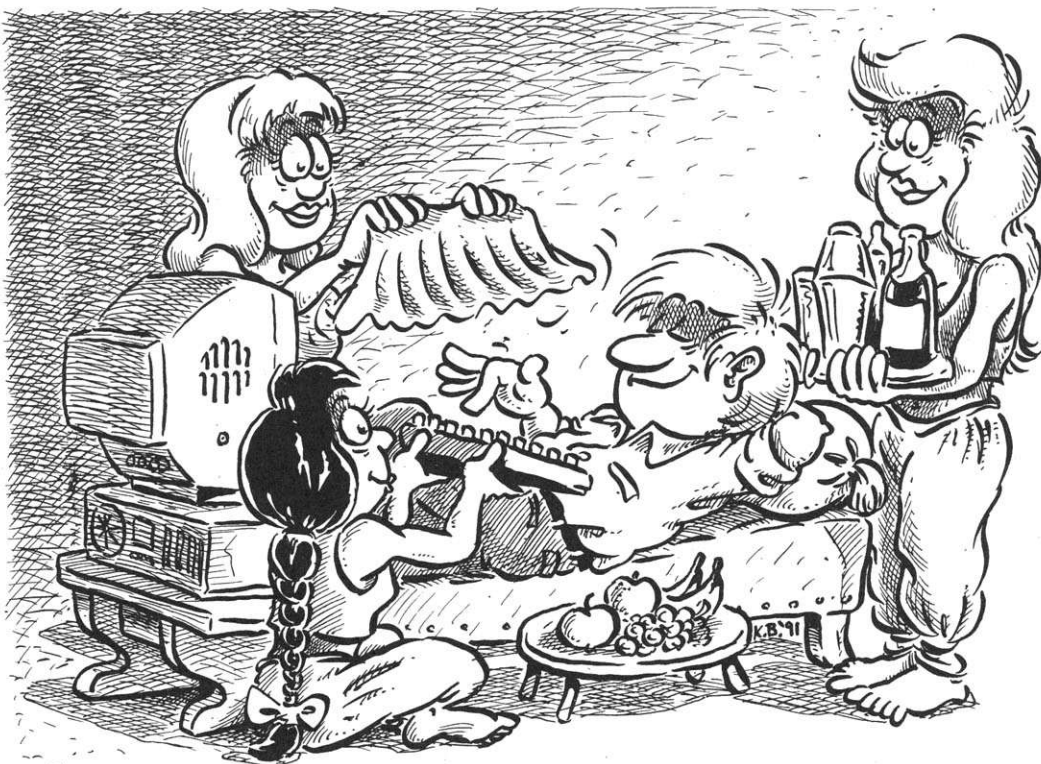
beep_beep.c

Der erste Schritt zum Synthesizer

Amiga im Griff

Workbench, CLI und Shell

Es gibt zwei verschiedenen Arten, den Amiga zu bedienen, d.h. Programme zu starten, Dateien zu kopieren etc.: Der komfortable Weg geht über die Workbench, vorwiegend durch Maus und Menüs gesteuert; oder aber der etwas technischere Weg über die Shell bzw. das CLI, in dem man dem Amiga die Kommandos über die Tastatur gibt. Beide Verfahren haben Vor- und Nachteile; und für beide gibt's eine Menge Ratschläge.



Alles Geschmackssache – der eine bevorzugt die Workbench, der andere startet Programme lieber über's CLI; der eine findet Bildsymbole (Icons) übersichtlicher, der andere möchte Verzeichnisse lieber in Listenform betrachten. Am besten, man schaut sich beide Bedieneroberflächen des Amiga an und nutzt immer von Fall zu Fall die günstigere.

Shell-Speicher

Die Shell legt alle eingegebenen Kommandos in einem Puffer ab. Das ist praktisch, wenn man einen Befehl mehrmals ausführen lassen möchte, da man ihn nicht erneut eintippen muß. Man kann mit den Cursor-Tasten <Cursor_hoch> und <Cursor_runter> immer die letzte oder nächste Befehlszeile anzeigen lassen, diese ggf. editieren und mit <Return> auch ausführen lassen. Sie löschen den Puffer mit <Ctrl S>.

Andreas Schulz/ub

Mäuse sichern

Wenn man einen selbsterstellten Mauszeiger auf einer anderen Diskette haben möchte, gibt es zwei Wege: Die kompliziertere ist die,

sein Kunstwerk noch einmal mit Preferences zu malen und schließlich zu speichern. Allerdings ist das zeitraubend. Schneller dagegen ist es, nur die Konfigurationsdatei zu kopieren. Sie befindet sich im Verzeichnis »devs« jeder Start-Diskette bzw. der Workbench-Diskette und heißt »system-configuration«.

Kopieren Sie einfach diese Datei von Ihrer Diskette, auf der die fertige Maus steht, ins »devs«-Verzeichnis der Diskette, mit der Sie den Mauszeiger einsetzen möchten. Wenn Sie die präparierte Diskette booten, wird Sie Ihr kleines Kunstwerk begrüßen. Mit der »system-configuration« übernehmen Sie gleichzeitig auch die Farben, die Auflösung, den Druckertreiber und die Mausgeschwindigkeit.

ub

UNIX & AMIGA (feelings...)

Einige Amiga-User haben sicherlich häufig mit dem Unix-Betriebssystem zu tun, sei es im Beruf, auf der Universität oder sonstwo. Trotz vieler Gemeinsamkeiten von Unix und AmigaDOS gibt es kleine, aber oft sehr ärgerliche Unterschiede – beispielsweise bei den Shell-Umgebungen.

Da kann man leicht »list« mit »ls« verwechseln, wodurch auf dem Amiga eine unschöne »Unknown command«-Meldung auftaucht.

Um solche Peinlichkeiten zu vermeiden, sollte man die gebräuchlichsten Unix-Kommandos auf dem Amiga simulieren, was sich sehr gut mit ALIAS bewerkstelligen läßt. Darüberhinaus ermöglicht die Benutzung der ARP-Kommandos mit der dazugehörigen Shell ASH, besser noch die WShell, volle Musterunterstützung, Piping, Variablen etc., also Shell-Features, wie man sie von Unix her gewohnt ist. Aus diesem Grunde ist hier eine Batch-Datei angefügt, welche die gebräuchlichsten Unix-Kommandos simuliert. Die Batch-Datei sollte idealerweise während des Bootvorgangs abgearbeitet werden, damit die »neuen« Kommandos sofort zur Verfügung stehen. Packen Sie die ALIAS-Definitionen am besten in die Datei »shell-startup« im Verzeichnis »s«. Unsere ALIAS-Befehle können ein echtes Unix auf dem Amiga natürlich nicht ersetzen.

Thomas Becker/ik

```
; Batch-Datei: Unix-Simulation
; klappt natürlich zusammen mit ARP (Joker,Pipes...) am besten !!!
alias pwd cd []
alias ls list []
alias ps status [] full
alias kill break [] all
alias nice arun [] pri -10
alias cp copy []
alias rm delete []
alias mdir mkdir []
alias mv move []
alias man more dh0:doc/[] .doc ; alle doc-Files sollten sich
                                ; im doc-Verzeichnis befinden
alias install chmod []
```

Unix.bat

Unix-feeling auf dem Amiga

Beschleunigte Batch-Dateien

Batch-Dateien sind ein wichtiges und hilfreiches Instrument im täglichen Umgang mit dem Amiga. Ein Nachteil von Batch-Dateien besteht allerdings in ihrer relativ langsamen Abarbeitung, da jeder einzelne Befehl erst zu seiner Ausführung gesucht, gefunden und dann geladen werden muß. Selbst wenn die Batch-Befehle von Festplatte oder aus der RAM-Disk geladen werden, wird der Zeitaufwand – besonders bei längeren Batch-Dateien – spürbar.

Muß ein Befehl einer Batch-Datei geladen werden, durchsucht die Shell zuerst die ALIAS-Definitionsliste, dann die Shell-internen Befehle, daraufhin die Liste der residenten Programme, das aktuelle Verzeichnis, das logische Verzeichnis »c:« und, wenn das auch nicht zu einem positiven Ergebnis führt, noch die Pfadliste, die durch den PATH-Befehl bestimmt wird, schließlich wird die Fehlermeldung »Unknown Command« ausgegeben, wenn alle Suchvorgänge erfolglos waren.

Da auch die Pfadliste des PATH-Befehls mitunter viele Einträge umfaßt, wird schnell deutlich, daß während der Abarbeitung einer Batch-Datei viel Zeit zur Suche des auszuführenden Befehls verschwendet wird. Man vermeidet das, indem man den Ort der Befehle (den vollständigen Pfad- und Dateinamen) angibt, z.B.:

```
c:mount newcon: pip: speak: null:
c:stack 10240
c:run sys:system/noclick >nil:
```

Ebenso beschleunigend wirkt die Maßnahme bei den beliebten ALIAS-Definitionen in Verbindung mit der Shell:

```
alias show c:dir [] hilite size
alias talk c:run c:copy [] speak:opt/r/sl60
alias clear c:echo **E[0;0H*E[J*
```

Aber Achtung: So eine Vorgehensweise lohnt sich natürlich nicht bei residenten Dateien oder bei Dateien im aktuellen Verzeichnis, denn sie werden ja ohnehin schnell gefunden. Deshalb sollte man sich schon im klaren sein, welche Kommandos erst vom langsamen, externen Datenträger geladen werden müssen und welche nicht! Durch diesen kleinen, aber wirkungsvollen Trick wird ein Großteil der

Suchvorgänge durch die Shell bei Batch-Dateien vermieden. Messungen ergeben eine Performance-Steigerung von bis zu 30 Prozent, was den größeren Tippaufwand bei Batch-Dateien rechtfertigt.

Thomas Becker/ik

Unabhängig

Mit dem CLI kann man viel machen, die von der Workbench nicht unterstützt werden. Beispielsweise besteht nur im CLI Zugriff auf alle Dateien und Verzeichnisse einer Diskette. Jeder CLI-Befehl ist ein eigenständiges Programm und muß vor der Ausführung geladen werden. Der Amiga erwartet, daß sich die Befehle im C-Ordner der Startdiskette befinden. Gibt man einen Befehl ein, nachdem die Diskette gewechselt wurde, »schreit« der Amiga sofort nach der Startdiskette. Der Trick, alle benötigten Befehle in die RAM-Disk zu kopieren, hat einen Nachteil: Die Befehle belegen Speicherplatz.

Wenn Sie mit mehreren Disketten arbeiten, besitzen diese meist alle einen eigenen C-Ordner. Der Amiga nimmt davon jedoch keine Notiz und fordert weiterhin die Startdiskette. Er läßt sich überlisten,

TIPS ZU CLI, SHELL UND WORKBENCH

Tip	Thema	Seite
Shell-Speicher	Shell-Bedienung	48
Mäuse sichern	»system-configuration«	48
Unix & Amiga	Unix-Simulation (ALIAS-Befehl)	48
Beschleunigte Batch-Dateien	Befehlsabarbeitung	49
Unabhängig	Nochmal ASSIGN-Befehl	49
Go Directory	MAKEDIR und CD kombiniert	50
Device-Abfrage	Shell-Skript zum Test von Devices	50
Lamer Exterminator	Viren killen	50
Unterschiedliche ECHOs	Schriftstile und ECHO	50
Die Maus stoppt alles	Bedienungshinweis	51
Entscheidender Schreibschutz	Shell-Skript testet Schreibschutz	51
RAM Disk	RELABEL-Befehl	51
Tastatur statt Maus	Requesterantwort mit der Tastatur	51
Kommandopfade	PATH-Befehl	51
Alles halt	<Ctrl C> im Einsatz	52
Weg mit dem Laufwerk	ASSIGN REMOVE	52
CLI und nur ein Laufwerk	Namen von Disketten	52
RUN beim Start	Multitasking	52
Rekursion mit AmigaDOS	»Programmierung« im CLI	52
Shell schlägt CLI	Shell öffnen	53
Schnellere Disketten	Ordnung auf Disketten	53
Disketten einfach formatieren	Batch-Datei	53
LIST ohne Tücken	Optionen von LIST	53
Laden von der Werkbank	Default-Tool	54
Alternative zu RUNBACK	RUN >NIL: Programm	54
LoadWB mit Optionen	Zusatzmenü	54
Verzeichnisse von PD-Disks	Trick 17 zum Multitasking	54
Schnellstart von Workbench	schnelle »Startup-Sequence«	54
Zu wenig Speicher?	Speicherplatz sparen	54
Unerklärlicher Absturz	STACK	54
Alles auf einen Schlag	Mehrfachauswahl mit der Maus	55
Übersichtliche Kommentare	FILENOTE-Befehl	55
Ohne Uhr die richtige Zeit	DATE	55
CLI-Fenster ohne Rahmen	Escape-Sequenzen	55
Editieren	RUN	56
Read Error, halb so wild	Disk-Validator	56
CLI unter Kontrolle	<Ctrl>-Kombinationen	56
Schnelle Verzeichniswechsel	CD-Befehl	56
Schaukel für die Shell	Joker-Kombination »#?«	56
Arbeiten mit LIST	Optionen von LIST	56
Workbench oder CLI?	ASK-Befehl	56
Kürzere Bootzeiten mit TYPE	TYPE statt ECHO	57
Tolles Tool mit Anleitung	MicroEmacs	57
Es geht auch ohne EXECUTE	Skript-Bit	57
Spiel mit der Maus kostet Zeit		57

BEDIENUNG

wenn man zusätzlich zum Namen den Suchpfad des Befehls angibt. Statt DIR schreibt man:

```
df0:c/DIR
```

Auf die Dauer ist es jedoch umständlich, immer den Suchpfad anzugeben. Abhilfe schafft der Befehl ASSIGN. Er dient dazu, einem logischen Gerät einen Ordner einer bestimmten Diskette zuzuweisen. Dieser Ordner läßt sich sowohl über seinen eigenen Namen, als auch über den des logischen Gerätes ansprechen. Einige logische Geräte werden bereits beim Start zugewiesen. So zeigt c: auf den C-Ordner der Startdiskette. Mit ASSIGN lenkt man man die Zuweisung um, beispielsweise auf den C-Ordner der Diskette, die sich augenblicklich im internen Laufwerk befindet.

```
df0:c/ASSIGN c: df0:c
```

Nun sucht das CLI seine Befehle auf der aktuellen Diskette. Einziger Nachteil der Methode: Nach einem Diskettenwechsel muß der ASSIGN-Befehl erneut eingegeben werden. Wenn Sie wissen wollen, welche logischen Geräte dem Amiga bekannt sind, verwenden Sie den ASSIGN-Befehl ohne Parameter.

Christoph Kögler/ub

Go Directory

Wenn man ein neues Verzeichnis anlegt, ernennt man es anschließend oft mit CD direkt zum aktuellen Directory, um dort zu arbeiten. MAKEDIR und CD sind – zwei – Befehle, die diese Aufgabe erledigen... doch reicht nicht einer? Hier das passende Shell-Skript mit Namen »GD.bat« (Go Directory), das Sie wie CD einsetzen.

```
.key dir
.bra {
.ket }
if "{dir}" EQ ""
cd
else
if not exists {dir}
mkdir {dir}
endif
cd {dir}
endif
```

GD. bat

CD und DIR in einem Abwasch

Das Verzeichnis, in das man wechseln möchte, wird natürlich nur dann mit MAKEDIR angelegt, wenn es vor dem Aufruf von »GD.bat« noch nicht existiert. Am besten kopieren Sie GD ins Verzeichnis »s:« und setzen das Skript-Flag mit:

```
protect s:gd -s add
```

Sie können dann zum Aufruf von GD (ab Kickstart 1.3) das lästige EXECUTE weglassen und einfach nur tippen:

```
GD verzeichnis
```

Der Amiga erkennt automatisch, daß es sich um ein Script handelt und führt die Befehle aus.

ub

Device-Abfrage mit Shell-Befehlen

Oft möchte man abfragen, z.B. in Batch-Dateien, ob ein Gerät existiert, also auch angemeldet (geMOUNTed) ist. Man würde z.B. eingeben, um zu prüfen, ob Laufwerk df1: existiert:

```
if exists df1:
```

Fehlt das Laufwerk, bekommt man allerdings die recht störende Aufforderung zu sehen: »PLEASE INSERT VOLUME DF1:«

Wie man besser nach einem »Device« fragt, zeigt das folgende Beispiel anhand der resetfesten RAM-Disk. Damit dürfte der Requester »PLEASE INSERT VOLUME xxx:« in einer Batch-Datei für immer verbannt sein. Sie können die Routine in eigene Programme einbauen und so anwenderfreundlicher gestalten.

Felix Farago/ub

```
echo "Schau nach, ob RAD: existiert..." noline
assign >NIL: RAD: exists ; so wird nach einem Device gefragt
                        ; als Gerät können Sie einsetzen
                        ; was Sie möchten
if warn ; wenn Gerät nicht existiert
echo "RAD: konnte nicht gefunden werden."
Skip Ende
else
echo "Gefunden."
lab Ende
```

Devicetest.bat

Device-Abfrage mit CLI-Befehlen ohne lästige Systemmeldungen (Requester)

Lamer Exterminator

Daß man im CLI bzw. in der Shell mit

```
install df0: +
```

unliebsame Bootblockviren entfernt, ist wahrscheinlich vielen Amiga-Benutzern bekannt. Aber nicht nur Bootblockviren kann man mit dem CLI vernichten, sondern auch den Lamer-Virus! Man muß nur im CLI den Befehl

```
type DF0:s/startup-sequence opt h +
```

eingeben und <Return> drücken. Ist das Laufwerk zur Ruhe gekommen, nimmt man die Startdiskette aus dem internen Laufwerk und legt die Diskette ein, die man testen will. Nun drückt man nochmals <Return> und schon wird die »Startup-Sequence« im Hex-Code ausgegeben. Wenn in der ersten Zeile nach »0000« steht:

```
: A0A0A0A0 A0
```

ist der Lamer-Virus am Werk.

Nun muß man das infizierte Programm löschen. Hierzu legen Sie die Startdiskette ins interne Laufwerk und tippen den Befehl ein:

```
delete "DF0:X" + e
```

»X« steht hier für das fünfmalige Drücken der Tastenkombination <Alt Shift Space> (<Alt> und <Shift> gedrückt halten und fünfmal <Space> drücken). Wieder läßt man das Laufwerk zur Ruhe kommen, legt die infizierte Diskette ein und drückt nochmals <Return>. Nun muß man nur noch mit einem beliebigen Editor (z.B. ED oder MEmacs) die erste Zeile der »Startup-Sequence« löschen.

ub

Unterschiedliche ECHOs

Mit der ECHO-Anweisung im CLI bzw. in der Shell kann man Texte in verschiedenen Schriftstilen und Farben anzeigen. Das geschieht nach folgendem Muster:

```
echo "e[XXmBeliebiger Text"
```

Für »XX« muß man nun verschiedene Zahlen einsetzen, die folgender Tabelle zu entnehmen sind:

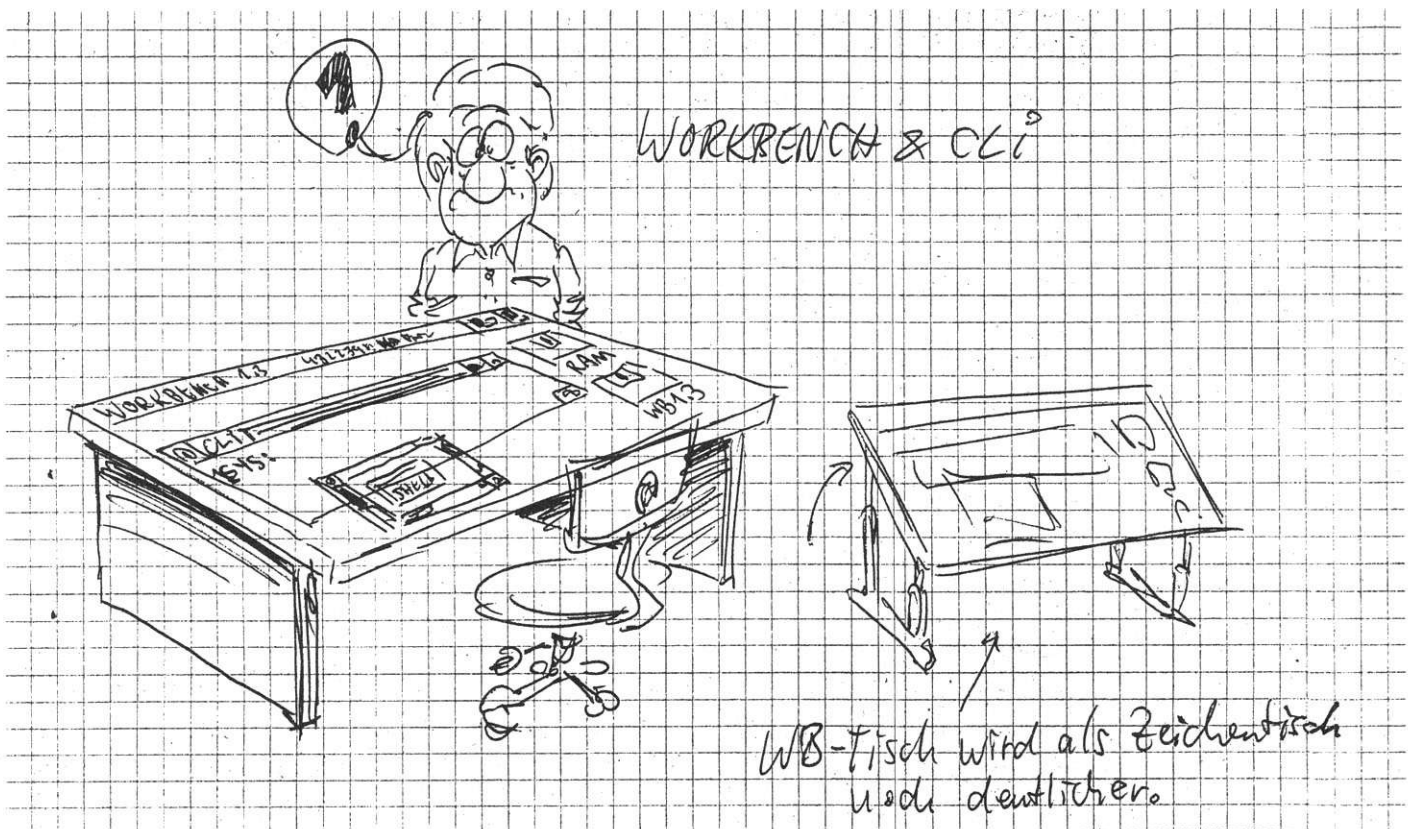
»XX«	Schriftart/Farbe
0	normal
3	kursiv an
23	kursiv aus
4	unterstreichen an
24	unterstreichen aus
1	fett an
22	fett aus
30-33	Vordergrundfarbe
40-49	Hintergrundfarbe

Man kann natürlich auch verschiedene Einstellungen auf einmal wählen. Hierzu muß man nur hinter den Zahlen »;« eingeben. Für kursiv und unterstrichen also:

```
echo "e[3;4mBeliebiger Text".
```

Der Buchstabe »m« beendet die Schriftstileinstellung.

ub



Die Maus stoppt alles

Wenn man die Ausgabe im CLI anhalten möchte, reicht es, wenn man die rechte Maustaste gedrückt hält. Sobald man sie losläßt, fährt der Amiga mit der Ausgabe fort.

Stefan Kessner/ub

Entscheidender Schreibschutz

Wollen Sie den Schreibschutz einer Diskette beim Starten einer Diskette prüfen. Nichts einfacher als das. Hierzu müssen Sie mit dem Editor ED die »Startup-Sequence« verändern:

```
ed s/startup-sequence
```

Ergänzen Sie die Startsequenz um folgende Befehle:

```
failat 25
protect DF0:Hilfsdatei
if warn ECHO "schreibgeschützt"
skip Marke
endif
echo "kein Schutz"
lab Marke
echo "Das wars..."
```

So können Sie feststellen, ob Ihre Disk schreibgeschützt ist oder nicht. Eine Voraussetzung ist, daß die Datei »Hilfsdatei« auf der Disk steht. Sie können Sie ebenfalls mit dem ED leicht erzeugen.

```
ED Hilfsdatei
```

Tippen Sie in die Datei einfach ein paar Zeichen und speichern den Text mit <Esc X>. Mit der Schreibschutzabfrage lassen sich auch zwei verschiedene Programme ausführen. Welches Programm aufgerufen wird, bestimmt letztendlich der Schreibschutz.

Stephan Wolff/ub

RAM

Disk

Die RAM-Disk heißt seit der Workbench 1.3 »RAM DISK«. Dieser neue Volume-Name ist problematisch, da einige Programme einen

Leerschritt im Volume-Namen nicht verkraften und sich dann fehlerhaft verhalten. Wer mit solchen Problemen zu kämpfen hat, dem soll nun geholfen sein:

Mit dem Kommando RELABEL ist es möglich, jeden Volume-Namen zu ändern. Zu achten ist dabei auch auf die Eindeutigkeit des gewählten Namens, beispielweise sollte das Drive »ram:« nicht den Namen »sys:« erhalten, da »sys:« bereits den Namen des Boot-Geräts symbolisiert.

Es empfiehlt sich, folgende Lösung am besten in der »Startup-Sequence« unterzubringen:

```
relabel drive ram: name RAMDISK
```

Thomas Becker/ik

Tastatur statt Maus

System-Requester wie »"Please insert volume XXXX"« beantwortet man auf zwei Arten:

- Entweder man klickt eines der Gadgets an,
- oder man benutzt die Tastatur

Um das linke Gadget anzuwählen (»RETRY«), drückt man <Amiga V>. Zum Anwählen des rechten Gadgets (»CANCEL«) drückt man <Amiga B>. Das funktioniert übrigens auch mit allen anderen System-Requestern, die zwei Gadgets haben.

ub

Kommandopfade

Wer kennt es nicht? Man hat wieder einen neuen Befehl für's CLI bzw. die Shell ausfindig gemacht, den man gut gebrauchen kann. Also schnell ins »c«-Verzeichnis der Workbench-Diskette damit. Allerdings hat das den Nachteil, daß man nach einer Zeitlang nicht mehr weiß, was zur Original-Workbench Diskette gehört und was nicht. Außerdem schwillt das Verzeichnis immer mehr an und die Anzeige des Verzeichnisses dauert recht lange. Abhilfe schafft hier ein eigenes Verzeichnis, z.B. »C2«. Man erstellt es mit:

```
mkdir c2
```


Anschließend können Sie all Ihre Sonderbefehle in dieses Verzeichnis packen.

Allerdings findet der Amiga die Befehle in diesem Verzeichnis nicht auf Anhieb. Hier hilft der Befehl PATH:

```
path df0:c2 add
```

Von nun an behandelt der Amiga das C2-Verzeichnis so, als wäre es das C-Verzeichnis, und die Befehle im C-Verzeichnis sind weiterhin verfügbar.

Am besten Sie fügen den Befehl mit einem Editor in die »Startup-Sequence« ein, dann sucht der Amiga immer automatisch auch in »c2« nach Kommandos.

ub

Alles halt!

Man kann die Textausgabe des CLI- bzw. Shell-Befehls TYPE durch <Ctrl C> stoppen. Das gilt aber nicht nur für TYPE. Auch viele andere Programme, die etwas im CLI ausgeben, lassen sich mit <Ctrl C> beenden, z.B. Disksalve, A68k etc.

Sean Durkin/ub

Weg mit dem Laufwerk

Jedem Festplattenbesitzer ist es ein Dorn im Auge, daß eine erhebliche Anzahl von Programmen (hauptsächlich Spiele) nur deswegen nicht auf der Platte läuft, weil ein Programm immer stur auf die Floppy zugreift. Mit einem kleinen Kniff läßt sich dieser Mißstand bei vielen Programmen beheben. Geben Sie im CLI oder der Shell ein:

```
assign remove ; entfernt logisches Gerät DF0:
```

```
assign df0: <verzeichnis> ; damit werden alle folgenden Zugriffe
```

```
; auf DF0: auf das gewünschte Verzeichnis umgelenkt
```

Der Trick funktioniert natürlich nur dann, wenn das jeweilige Programm nicht direkt unter Umgehung des Betriebssystems auf ein Disketten-Laufwerk zugreift.

Helmut Kirchweger/ub

CLI und nur ein Laufwerk

Jeder, der nur ein Laufwerk besitzt, kennt das Problem: Es ist nicht möglich, ein Verzeichnis (Directory) einer anderen Diskette als der Workbench-Diskette aufzulisten. Zumindest nicht, wenn Sie es mit folgendem Befehl versuchen:

```
dir df0:
```

Der Computer greift nach Lesen des DIR-Befehls auf der Workbench-Diskette gleich auf deren Inhaltverzeichnis zu. Es ist also nicht machbar, die Diskette zu wechseln. Für diesen Fall gibt es eine einfache Lösung: Setzen Sie statt »df0:« den Namen der Namen der Diskette ein, z.B.:

```
dir Basic:
```

Nun fordert der Amiga Sie nach Lesen des DIR-Befehls auf, die Diskette »Basic« einzulegen und listet deren Verzeichnis auf.

Die Namen Ihrer Disketten erfahren Sie übers CLI oder die Shell mit INFO. Der Befehl RENAME dient zum Ändern des Namens. Auf der Workbench zeigt der Amiga die Namen unter den Diskettensymbolen. Auch hier gibt es ein Kommando, um den Titel zu ändern, den Menüpunkt »Rename«.

Jens Thorwarth/ik

RUN in der »Startup-Sequence«

Nehmen wir an, Sie starten den Amiga (ein Laufwerk) und wollen in der »Startup-Sequence« ein Programm im Hintergrund ablaufen lassen, das ein wenig Zeit braucht. Damit am Ende das CLI-Fenster schließt, starten Sie das betreffende Programm mit RUN oder RUNBACK. Die »Startup-Sequence« könnte also so aussehen:

```
echo "start"
```

```
; andere Befehle zur freien Auswahl
```

```
;
```

```
; "freundliche" Erinnerung, den Drucker einzuschalten
```

```
runback sys:System/InitPrinter
```

```
echo "achten Sie auf Ihr Laufwerk!!!!!"
```

```
;
```

```
; andere Befehle zur freien Auswahl
```

```
;
```

```
loadwb
```

```
endcli
```

Was passiert mit dieser Version der »Startup-Sequence«?

Kaum führt der Amiga RUN bzw. RUNBACK aus, geht das große Rattern auf der Startdiskette los. Der RUN-Befehl will das Programm »InitPrinter« laden und gleichzeitig versucht der Amiga, die nächsten Befehle der »Startup-Sequence« zu laden und auszuführen. Das Betriebssystem teilt die Zugriffsberechtigung für das Laufwerk nach folgendem Prinzip zu: Einmal liest RUN (ein eigener Task); einmal liest der Task, der für die Ausführung der Startsequenz verantwortlich ist. Dabei kommt es häufig zu großen Zeitverzögerungen, da die Laufwerksköpfe ständig umpositioniert werden: dadurch rattert's.

Abhilfe schafft hier, das zu startende Programm in die RAM-Disk zu kopieren und es von dort zu initialisieren. Wird der RUN/RUNBACK-Befehl mehrmals benutzt, sollte man auch ihn in die RAM-Disk packen:

```
echo "start"
```

```
; andere Befehle zur freien Auswahl
```

```
;
```

```
copy run to ram:
```

```
copy sys:System/InitPrinter to ram:
```

```
ram:run >nil: ram:InitPrinter
```

```
;
```

```
echo "was macht das Laufwerk jetzt?"
```

```
;
```

```
; andere Befehle zur freien Auswahl
```

```
;
```

```
; Speicherplatz evtl. wieder freigeben
```

```
; delete ram:run
```

```
; delete ram:InitPrinter
```

```
;
```

```
loadwb
```

```
endcli;
```

Sie können natürlich auch das Kopieren des C-Verzeichnisses vor den RUN-Befehl setzen, wenn dies sowieso beim Start im RAM abgelegt werden soll — und gerade, wenn man nur ein Laufwerk besitzt, ist das ja sinnvoll:

```
..
```

```
..
```

```
copy C: to ram:c
```

```
path ram:c
```

```
runback sys:System/InitPrinter
```

```
..
```

```
..
```

Der durch Umsicht bei der Wahl der Befehle in der »Startup-Sequence« zu erzielende Geschwindigkeitsgewinn ist beträchtlich; prüfen Sie, ob Ihr Amiga optimal startet.

Thomas Blenkins/ik

Rekursion mit AmigaDOS

Wer nur gelegentlich mit dem Programm »CLIMate« arbeitet, wird sich schon oft über die »fastdir«-Dateien geärgert haben, die das CLI-Werkzeug in allen Verzeichnissen und Unterverzeichnissen hinterläßt, die es jemals besucht hat. Die »fastdir«-Einträge helfen zwar CLIMate, ein Directory schneller einzulesen, für andere Programme, insbesondere AmigaDOS, sind sie aber wertlos.

Es liegt also nahe, alle »fastdir«-Dateien wieder zu löschen. Doch das ist aufwendig, zumal man dazu natürlich CLIMate nicht verwenden kann. Hier hilft ein kleines Batch-Programm (»ExFast«), das, angesetzt auf ein Directory, die »fastdirs« in allen Unterverzeichnissen löscht. ExFast sollte am besten im »S«-Verzeichnis ste-

hen. Der Aufruf der Befehls-Datei vom CLI aus, erfolgt dann mit dem Kommando:

```
execute ExFast Directory
```

Das kurze Programm liefert ein Beispiel für eine recht nützliche Fähigkeit des LIST-Befehls, die es gestattet, auch Batch-Dateien quasi rekursiv zu schreiben. LIST erzeugt wieder ein Batch-File namens »ram:x«, das die Aufrufe für alle Subdirectories umfaßt. Der EXECUTE-Befehl schließlich führt diese aus, wobei ggf. wieder neue Verzweigungen entstehen, bis keine untergeordneten Directories mehr vorhanden sind. Diese Technik ist natürlich auch für andere Aufgaben einsetzbar. Wie wäre es beispielsweise, alle »info«-Dateien in allen Unterverzeichnissen zu löschen? Dazu müssen Sie nur den DELETE-Befehl etwas ändern:

```
delete >NIL: #?.info
```

Statt eines simplen DELETE-Aufrufs könnte der Amiga auch komplexere Aufgaben ausführen, z.B. nach Linkviren suchen, etc. Die Rekursion spart in fast allen Fällen eine Menge Arbeit.

Rolf Beck/ik

```
.key Dir
.default Dir ""

; ExFast
; löscht alle .fastdir-Files im angegebenen Verzeichnis
; und allen Unterverzeichnissen
; Aufruf: execute ExFast <Dir>
failat 21 ; LIST soll nicht abbrechen, wenn keine
          ; Verzeichnisse mehr vorhanden sind
cd "<Dir>" ; aktuelles Verzeichnis anwählen
delete >NIL: .fastdir ; und dort die Aufgabe erfüllen
list >ram:x dirs lformat="execute ExFast %s" ; Selbstaufruf
if not fail
    execute ram:x
endif
cd /
```

ExFast.bat

Rekursive Programmierung per Batch-Datei

Shell schlägt CLI

Die Shell der Workbench 1.3. ist schon eine praktische Sache, die dem CLI bei weitem überlegen ist. Sie gelangen in diese Benutzeroberfläche, indem Sie das Shell-Symbol auf der Workbench anklicken.

Doch was ist zu tun, wenn man mit der Shell arbeiten möchte, ohne zuvor die Workbench zu starten? Ganz einfach, verwenden Sie folgende Befehle in der »Startup-Sequence« (Verzeichnis »s«):

```
mount newcon:
```

```
newcli "NEWCON:x/y/Breite/Höhe/Titel"
```

x und y sind die Koordinaten der linken oberen Ecke des Shell-Fensters; Breite und Höhe legen die Größe des Windows fest, und Titel bestimmt den Text, der in der Titelleiste erscheint. Ersetzt man das Zauberwort »NEWCON« durch »CON«, erhält man ein normales CLI-Fenster.

Der Befehl:

```
newcli "NEWCON:0/0/640/256/Amiga-Shell"
```

öffnet also ein Shell-Window, das den gesamten Bildschirm einnimmt. Achten Sie darauf, daß sich im Verzeichnis »l« der Newcon-Handler und in der »Mountlist« (Verzeichnis »devs«) ein Eintrag für »Newcon:« befinden. Soll beim Start nur die Shell geöffnet werden, löschen Sie den Befehl LOADWB aus der »Startup-Sequence«.

Martin Regensburg/ub

Schnellere Disketten

Das Amiga-Betriebssystem hat die Eigenart, Dateien oder Programme nach dem Motto auf Disketten zu schreiben: Wo Platz ist, wird was reingequetscht, und wenn es auch nur ein Bruchteil einer

Datei ist; der Rest kommt auf eine andere Spur. Das bedeutet im Klartext: Wenn man häufig Dateien auf Disketten löscht und wieder darauf speichert, treten beim Arbeiten mit der Diskette extreme Geschwindigkeitsverluste auf, da einzelne Programme auf der Diskette verteilt sind und der Lesekopf beim Lesen oft auf andere Spuren wechseln muß. Falls man ein zweites Laufwerk besitzt, besteht nun die Möglichkeit, im CLI die Files komplett auf eine leere, formatierte Diskette zu kopieren:

```
copy df0: to df1: all
```

Dadurch werden die Dateien geordnet die leere Diskette kopiert und die Diskette wird schneller.

Wenn man kein zweites Laufwerk besitzt, aber mindestens 1 MByte Speicher, kann man dieselbe Prozedur ausführen, indem man die RAM-Disk als Zwischenspeicher einsetzt. Sollten Ihnen lediglich 512 KByte bei einem Laufwerk zu Verfügung stehen, müssen Sie die Dateien portionsweise über die RAM-Disk auf die neue Diskette kopieren.

Wenn Sie nun auf der aufgeräumten Diskette abermals häufig Dateien löschen und wieder andere draufschreiben, entsteht mit der Zeit natürlich wieder das alte Durcheinander. Wie im Haushalt: Unordnung entsteht immer.

Ingo Gotsch/ub

Disketten einfach formatieren

Dieser Tip eignet sich für alle, die öfter Disketten formatieren bzw. löschen. Eine einfache Batch-Datei (Listing) erleichtert diesen Vorgang. Nach Aufruf der Batch-Datei (über EXECUTE, oder die Datei steht im Verzeichnis »S:« und das S-Bit ist gesetzt, dann reicht die Eingabe des Namens) führt der Amiga den dort angegebenen FORMAT-Befehl so oft aus, bis Sie mit <Ctrl c> oder <Ctrl d> stoppen.

Rodja Adolph/ik

```
; Batch : Disketten formatieren
Failat 31 ; diverse Fehlermeldungen verharmlosen
          ; (Bad Block, No Sector Header...)

LAB Start ; Anfang der Schleife
Format DRIVE DF1: NAME Blank
; DF1: kann geändert werden.
; je nach Wunsch noch ein QUICK und/oder NOICONS
; dranhängen (bei schon beschriebenen Disks)
SKIP Start BACK ; Ende der Schleife (Abbruch mit <Ctrl C>)
```

Batch-Datei

zum einfachen Formatieren mehrerer Disketten

LIST ohne Tücken

Der LIST-Befehl des CLI ist ein leistungsstarker Befehl. Er dient zum Auflisten ganzer Directories oder einzelner Dateien. Dabei zeigt LIST u.a. die Größe jeder Datei und wann sie auf der Diskette gespeichert wurde. LIST besitzt eine Menge zusätzlicher Funktionen und Optionen. Diese Optionen benutzen jedoch Buchstaben als Schlüsselwörter. Nun gibt es Konflikte, wenn man sich Verzeichnisse anschauen möchte, deren Name aus einem dieser Buchstaben besteht. Ein Beispiel ist das Direktory »s« auf der Workbench-Diskette. Wer »LIST s« eingibt, bekommt eine Fehlermeldung. Für den Buchstaben »S« ist die Funktion »Subname« reserviert. Es wird also nichts mit dem Auflisten des Verzeichnisses S, oder?

Umgehen sie die Einschränkung einfach, indem Sie den Befehl so eingeben:

```
LIST "s"
```

Durch die Anführungszeichen um den Buchstaben verwechselt der Amiga »s« nicht mehr mit einer Option.

Arne Zimmermann/ub

Laden von der Werkbank

Es dauert manchmal recht lange, bis der Amiga auf der Workbench ein Inhaltsverzeichnis geladen hat. Und erst wenn ein komplettes Directory geladen wurde, kann man ein Programm aus einem Fenster mit Doppelklick aktivieren. Hat man nun aber auf der jeweiligen Diskette nur ein Programm, das man immer braucht, geht man folgendermaßen vor:

1. Durch einmaliges Anklicken die betreffende Diskette aktivieren.
2. Aus dem Workbench-Menü INFO wählen.
3. Das zu startende Programm z.B. das Notepad mit vollem Pfad ins Feld »DefaultTool« eintragen.
4. Das INFO-Fenster mit SAVE verlassen.

Jetzt kann man das eingetragene Programm einfach starten, indem man das Icon der Diskette auf ein anderes legt oder im Workbench-Menü »DUPLICATE« aufruft. Im Feld DefaultTool steht normalerweise »DiskCopy«. Es dient zum Kopieren von Disketten. Hier können Sie beliebige andere Programme eintragen. Der Fantasie sind keine Grenzen gesetzt. Sie können ein anderes besseres Kopierprogramm, einen Editor oder eines Ihrer Lieblingsspiele eintragen. Es steht Ihnen von nun an immer zur Verfügung, ohne daß Sie lange suchen und warten müssen.

Ralf Kaiser/ub

Alternative zu RUNBACK

Um das CLI-Fenster nach einem Programmstart wieder zu schließen, ohne das gestartete Programm zu beenden, kann man (statt RUNBACK) auch RUN verwenden. Sie brauchen nur direkt hinter RUN die Zeichen »>NIL:« einzufügen. Also beispielsweise:

```
run >NIL: programmname
```

Durch das »>NIL:« werden alle Ausgaben des gestarteten Programms in eine Art »schwarzes Loch« (logisches Gerät NIL:) geschickt, d.h. sie erscheinen nicht im CLI-Fenster. Das ermöglicht auch das Schließen des CLI-Fensters nach einem Programmstart.

Sven Kreutz/ik

LoadWB mit Optionen

Vielen Amiga-Anwendern wird der Parameter »-delay« für den CLI-Befehl LOADWB bekannt sein. Weit weniger bekannt ist die Option »-debug«. Wird »loadwb« mit diesem Parameter aufgerufen, erhält das Workbench-Fenster ein neues Menü, das zwei Funktionen besitzt:

- »flushlibs« und
- »debug«

»Flushlibs« löscht alle nicht mehr benötigten Libraries aus dem Speicher, um Platz zu sparen. »Debug« sollten Sie nicht aufrufen, da der Amiga dann einen internen Debugger startet, der nur von einem zweiten Terminal aus bedient werden kann, das mit dem Amiga über die serielle Schnittstelle verbunden sein muß. Das Menü hat keinen Titel und ist daher unsichtbar. Es befindet sich neben dem »Special«-Menü.

Sean Durkin/ik

Verzeichnisse von PD-Disketten

Wenn Sie öfter PD-Disketten, die keine Icons besitzen, mit dem CLI erforschen, kennen Sie vielleicht dieses Problem: Man läßt sich das Directory der PD-Disk anzeigen und probiert einige Programme aus. Da inzwischen der angezeigte Directory-Inhalt nach oben verschwunden ist, muß man das Directory nochmals einlesen, drei Programme später noch einmal etc. Das kann mitunter lange dauern. Folgende Tricks helfen:

- Sie öffnen ein zweites CLI (mit NEWCLI) und lassen dort das Verzeichnis ausgeben. Danach arbeiten Sie mit dem ersten CLI weiter. Sie brauchen sich dann nur das zweite CLI ansehen, um zu wissen, was sich auf der PD-Disk befindet.

- Sie lenken die Ausgabe des Verzeichnisses auf den Drucker um. Das erreichen Sie mit dem Befehl

```
dir >PRT: Diskette:
```

- Sie lenken die Ausgabe von DIR mit dem Befehl

```
dir >ram:file disk:
```

in die Datei »ram:file« um. Zum Ansehen des Verzeichnisses geben Sie dann ein:

```
type "ram:file"
```

Sean Durkin/ik

Schnellstart der Workbench

Werden die Befehle der (Standard-) »Startup-Sequence« etwas umgestellt, kann bereits vor der vollständigen Abarbeitung dieser Datei mit der Workbench gearbeitet werden.

Ihre »Startup-sequence« (Ausschnitt) sollte nach der Umstellung etwa folgendermaßen aussehen:

```
Addbuffers df0: 25
c:SetPatch >NIL: ;patch system functions
cd c:
echo "A500/A2000 CH Workbench disk. Release 1.3 version 34.20*N"
Sys:System/FastMemFirst ; move C00000 memory to last in list
BindDrivers
; Hier folgen die verschobenen Befehle :
path ram: c: sys:utilities sys:system s: sys:prefs add;->einfügen 1)
loadwb delay ; -> einfügen 1)
echo "WB gestartet.(CLI Window verkleinern für Sofortzugriff)"; 2)
; Ab hier ganz normal weiter
SetClock load ;load system time from real time clock (A1000
; owners should replace the SetClock load with Date
FF >NIL: -0 ;speed up Text
resident CLI L:Shell-Seg SYSTEM pure add; activate Shell
resident c:Execute pure
```

Die Befehle PATH und LOADWB werden also sehr früh aufgerufen. Vergessen Sie nicht, die beiden Befehle an den ursprünglichen Stellen zu entfernen. Besonders eignet sich dieser Tip für Besitzer eines zweiten Diskettenlaufwerks. Sie können darauf über die Workbench bereits während des Bootens zugreifen. Ein Zugriff auf die Startdiskette vor vollständiger Abarbeitung der »Startup-Sequence« empfiehlt sich aufgrund der Geschwindigkeitsminderung nicht.

Andi Egloff/ik

Zu wenig Speicher?

Jedes geöffnete Fenster braucht Speicherplatz. Wenn Ihr Computer nur 512 KByte hat, können geöffnete Fenster soviel Speicher belegen, daß der verbleibende Anteil für das gewünschte Programm nicht reicht. Oft hilft ein Trick:

Schieben Sie das Piktogramm des gewünschten Programms aus dem Fenster auf den Workbench-Screen. Schließen Sie danach alle Fenster. Auf der Workbench sollten sich jetzt nur das Piktogramm der RAM-Disk, das der Diskette und das Ihres Programms befinden. Wenn Sie Glück haben, hat der Amiga nun genug Speicher zur Verfügung. Wenn nicht, können Sie aus der »Startup-Sequence« die Befehle LOADWB und ENDCLI löschen (nur beide zusammen), denn die Workbench braucht eine Menge Speicher. Programme können jetzt allerdings nur noch vom CLI oder der Shell gestartet werden.

ub

Unerklärlicher Programmabsturz

Von der Workbench aus aufgerufen funktioniert das Programm einwandfrei, bei Aufruf vom CLI stürzt es ab. Woran liegt's? In so einem Fall klicken Sie das Programm-Icon an und danach INFO aus dem Workbench-Menü. Wenn im Feld »STACK« des Dialogfensters mehr als 4000 Byte eingetragen sind, könnte das die Ursache des Absturzes sein.

Der Stack ist ein besonderer Speicherbereich. Dort hinterlegt der Mikroprozessor u.a. die Rückkehradressen der Unterprogrammaufrufe. Öffnet man ein CLI- oder Shell-Fenster, reserviert AmigaDOS für davon gestartete Programme einen Stack mit 4000 Byte. Braucht ein Programm einen größeren Stack, muß man ihn vor dem Start erhöhen. Das erfolgt mit dem gleichnamigen Systembefehl STACK (siehe AmigaDOS-Handbuch).

ub

Alles auf einen Schlag

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 auf der Workbench zu verschieben bzw. 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 den Namen eines Programms, das beim Aktivieren des Programms automatisch geladen werden soll.

Ein Beispiel: Jedes AmigaBASIC-Programm erhält als »Default-Tool« den Namen »AmigaBASIC«. Häufig steht in diesem Namen aber gleichzeitig der Name eines bestimmten Verzeichnisses. Was nun, wenn AmigaBASIC sich in einem anderen Verzeichnis befindet. Dann erscheint nach dem Aktivieren des Programms die Meldung:

»Error while opening Amiga-BASIC«.

- Der Weg das zu umgehen, wäre den Namen in der Info-Datei zu ändern. Das geht über das Menü »Info« der Workbench.
- Eine andere Möglichkeit ist, erst AmigaBASIC 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 des AmigaBASIC-Interpreters an. Dabei spielt es keine Rolle, ob es sich im selben Ordner oder irgendeinem anderen Fenster befindet.

Dieter Bendrich/ub

**Life starts at '030
fun at '040
impotence at '86!**

Übersichtliche Kommentare

Mit dem CLI-Befehl FILENOTE kann jeder Anwender eine Notiz zu einer Datei schreiben, die in einer Kommentarzeile steht. Diese Kommentarzeile erscheint nur beim LIST-Befehl im Anschluß an die übrigen Daten. Allerdings erscheint der Kommentar zu einer Datei in einer gesonderten Zeile und verschiebt die gesamte Dateiauflistung:

```
test      DIR rwed 30-MAY-87 02:41:29
old       DIR rwed 30-MAY-87 02:43:30
dieser Kommentar zu File old stört
nächste   DIR rwed 04-JUN-87 12:12:12
```

Mit Hilfe von Steuerzeichen im Text können Sie Kommentare auch direkt hinter der Uhrzeit erscheinen lassen. Geben Sie, um eine Bemerkung zu einer Datei zu schreiben, den Befehl in dieser Form ein:

```
FILENOTE test "(<CTRL,k>,7*(<CTRL,i>,3*(<SPC>))Kommentar"
```

Dieser Trick erlaubt es Ihnen, zumindest kurze Kommentare direkt hinter dem Datum einzufügen. Bei Eingabe von weniger als sieben <CTRL,i> (=TAB) können Sie auch längere Bemerkungen einfügen, ohne die Zeilenlänge zu überschreiten. Sie überschreiben dann allerdings das Datum bzw. die anderen Anzeigen.

Bernd Zimmermann/ub

Ohne Uhr immer die richtige Zeit

Ein Tip für jeden, der keine eingebaute Uhr besitzt und dennoch immer das genaue Datum benötigt. Dieser Tip erspart Ihnen nicht, das Datum beim Starten des Amiga einzugeben, aber Sie brauchen die Arbeit nicht nach jedem Reset zu wiederholen.

Ergänzen Sie die »Startup-Sequence« um die folgenden Befehle:

```
FAILAT 30
ECHO "derzeitiges Datum"
DATE
DATE ?
DATE
DATE to datefile
```

Zunächst gibt der Amiga das gespeicherte Datum aus. Das zweite DATE zeigt die Optionen des DATE-Befehls und wartet auf die Eingabe eines Datums. Sie können nun das aktuelle Datum eingeben. Stimmt das angegebene Datum bereits, drücken Sie <Return>. Ihre Eingabe wird vom dritten DATE gelesen und mit dem letzten Befehl auf der Diskette gespeichert. Damit ist »datefile« die zu letzt gespeicherte Datei auf der Diskette. Bei jedem Start wird deren Datum automatisch als aktuelles Datum verwendet. So brauchen Sie immer nur einmal pro Tag das Datum eintippen, auch wenn Sie den Amiga zwischenteilich ausschalten.

Ist das Datum aktualisiert, können Sie am folgenden Tag die Korrektur des Datums mit der Eingabe »tomorrow« abkürzen.

Rainer Klier/ub

CLI-Fenster ohne Rahmen

Wenn ein CLI-Fenster keinen Rahmen hätte, könnte man mit dem normalen Zeichensatz 80 statt 77 Zeichen pro Zeile nutzen. Hierfür gibt es eine einfache Lösung.

Der Rahmen läßt sich mit Escape-Befehlen (<Esc>) ausschalten. <Esc>-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. Das ist auch über die Tastatur möglich. Drücken Sie nacheinander die Tasten <Esc>, <[>, <4> und <m>. Jetzt ist Unterstreichen eingeschaltet. Es gibt aber nicht nur Befehle für die Einstellung der verschiedenen Darstellungsarten, sondern auch für die Formatsteuerung:

Offsets: Bitplanes im ViewPort

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 der Inhalt mit <Ctrl L> gelöscht werden. Sofort verschwindet auch der Rand des Fensters. Schließlich sollten Sie noch einmal auf <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> und <Return>. Jeder, der seinen Bildschirm häufig auf diese Art vergrößern will, kann die benötigten <Esc>-Sequenzen auch in zwei Textdateien »Rahmen-Aus« beziehungsweise »Rahmen-Ein« zusammenfassen. Dann genügt der einfache Aufruf

type Rahmen-Aus

Winfried Stappert/ub

Editieren

Wenn Sie mit dem CLI arbeiten und den Editor mit »ed« aufrufen, können Sie zwar editieren, aber sonst keine weitere Arbeit mit dem CLI ausführen. Wer beispielsweise während des Editierens einmal den Namen einer Datei benötigt, steht vor dem Problem, daß das CLI auf keine Kommandos reagiert. Der folgende Befehl hilft:

RUN ED name

Nun arbeitet der Editor als eigenständiger Task. Im CLI-Fenster kann normal weitergearbeitet werden. Durch diesen Trick sparen Sie sich das Speichern und erneute Laden der zu editierenden Datei, falls Sie kurzfristig mit dem CLI arbeiten wollen.

Norbert Cohen/ub

Read Error, halb so wild

Kennen Sie die Systemmeldung:

Read Error Can't Validate Disk?

Sie kommen dann nicht mehr an die Daten auf der Diskette. Aber nicht verzagen, es gibt eventuell eine Rettung: Kopieren Sie das gesamte Dateiverzeichnis »|« von einer Workbench-Diskette in die RAM-Disk.

```
makedir ram:l  
copy df0:l ram:l all  
assign L: ram:l
```

Mit Hilfe der Zuweisung durch den ASSIGN-Befehl sucht der Amiga den Disk-Validator nicht mehr auf der eingelegten Diskette, sondern in der RAM-Disk. Legen Sie nun Ihre defekte Diskette erneut ein. Sollte der Amiga diesmal die Diskette ohne Fehlermeldung annehmen, kopieren Sie schnell alle benötigten Dateien auf eine andere Diskette.

Henning Müller/ub

CLI unter Kontrolle

Im CLI lassen sich viele Funktionen mit <Ctrl>-Kombinationen auslösen, d.h. durch Drücken von <Ctrl> und einer anderen Taste gleichzeitig. Die folgende Tabelle zeigt eine Übersicht aller Kombinationen. Ein kleiner Hinweis: Sie können die Tabelle kopieren oder ausschneiden und an Ihrem Arbeitsplatz aufhängen. So haben Sie immer Gelegenheit, sich schnell über die Tastaturbefehle zu informieren, meist ist das Handbuch ja gerade dann verschwunden, wenn man es am meisten braucht.

C. Krieg/ub

<Ctrl>-Kombinationen im CLI

Tastenkombination	Funktion
Ctrl l	löscht Bildschirm
Ctrl d	bricht Ausführung einer Befehlsdatei ab
Ctrl c	bricht laufenden Befehl ab (z.B. DIR)
Ctrl n	schaltet zweiten Zeichensatz ein
Ctrl o	schaltet amerikanischen Zeichensatz ein
Ctrl x	löscht aktuelle Zeile
Ctrl i	= TAB
Ctrl g	läßt den Bildschirm aufblitzen
Ctrl h	= Backspace
Ctrl j	Zeile nach unten; Eingabe mehrerer CLI-Befehle auf einmal möglich
Ctrl k	Zeile nach oben

Schnelle Verzeichniswechsel

Wenn man mit Hilfe des Befehls CD im Dateibaum sehr weit aufgestiegen ist und ins Hauptverzeichnis zurück möchte, braucht man nur eintippen:

CD :

Hinter dem Doppelpunkt kann auch jeder beliebige Pfad angegeben werden. Beispielsweise gelangt man mit

CD :s

ins Unterverzeichnis »s«, unabhängig vom Verzeichnis, in dem man vorher gearbeitet hat.

Maik Jablonski/ub

Schaufel für die Shell

Arbeiten Sie gerne mit der Shell? Dann ist die Kombination »#?« sicherlich interessant für Sie. Wenn Sie mit dieser Zeichenfolge arbeiten, können Sie viel schneller Dateien kopieren, löschen und durchsehen. Hier einige Beispiele, wie man »#?« einsetzt:

- Falls Sie alle Dateien mit dem Ende ».info« (aus dem aktuellen Verzeichnis) löschen wollen, geben Sie nur ein:

delete #?.info

- Tippen Sie ein:

copy Programm#? to df1:

Der Befehl kopiert alle Programme, die mit der Zeichenfolge »Programm« beginnen, nach »df1:«.

Sie können »#?« natürlich auch mit DIR bzw. LIST anwenden.:

- Das Kommando

dir #?TXT

gibt alle Dateien und Verzeichnisse (komplett) aus, deren Name mit »TXT« enden.

Genauso funktioniert LIST:

list Datei#?

zeigt alle Dateien und Verzeichnisse, die mit der Zeichenfolge »Datei« beginnen.

Alexander Koch/ub

Arbeiten mit LIST

Wie würden Sie vorgehen, wenn Sie bei vielen Dateien in einem Verzeichnis das Datum ändern möchten? Wählen Sie den mühsamen Weg, alle Dateien mit SETDATE zu bearbeiten? Einfacher und eleganter läßt sich das mit dem LIST-Befehl bewerkstelligen (gültig für Workbench 1.3):

```
list >ram:exedatei #? lformat="setdate %S today"
```

Die Befehlszeile erzeugt in der RAM-Disk die Datei »EXEDATEI«. In ihr sind alle Einträge des aktuellen Verzeichnisses aufgelistet – versehen mit dem Befehl SETDATE und dem Schalter TODAY, so daß die Datei folgendermaßen aussieht:

```
setdate <Datei-A> today  
setdate <Datei-B> today  
setdate <Datei-C> today usw.
```

Die Datei läßt sich mit EXECUTE abarbeiten. Anstelle von SETDATE können Sie z.B. auch PROTECT oder DELETE verwenden.

Christian Benner/ub

Workbench oder CLI?

Ins CLI gelangt man nach dem Laden der Workbench durch Anklicken des CLI-Symbols in der Schublade »Systems«. Wer häufig im CLI arbeitet, kann auch die »Startup-Sequence« im Verzeichnis »s« der Workbench-Diskette ändern und dort den Aufruf LOADWB und ENDCLI löschen, um nach dem Start des Amiga gleich im CLI zu bleiben. Wer mal mit dem CLI und mal mit der Workbench arbeiten möchte, sollte eine »Startup-Sequence« ähnlich der im folgenden Listing gezeigten verwenden. Die Sequenz bedient sich des Befehls ASK, um die Abfrage zu realisieren.

Marek Duda/ub

```
ADDBUFFERS df0: 20
system/SETMAP d
ECHO ""
ECHO ""
ECHO " Wollen sie die Workbench oder das CLI"
ASK " Taste y = Workbench - n = CLI"
IF NOT WARN
SKIP abc
ENDIF
LOADWB
ENDCLI >nil:
```

Auswahl.bat

Mit einer solchen »Startup-Sequence« haben Sie beim Booten die Wahl: Workbench oder CLI

Kürzere Bootzeiten mit TYPE

Viele Programmierer verändern die »Startup-Sequence« auf der Startdiskette, um beim Booten des Amiga mit ECHO Texte auszugeben. Bei längeren Texten hat das den Nachteil, daß für jede Zeile der ECHO-Befehl erneut von Diskette geladen werden muß, wodurch die Zeit beim Starten unnötig verlängert wird. Schneller geht es, wenn man den gewünschten Text in einer Datei speichert und ihn in der »Startup-Sequence« mit TYPE ausgibt.

Lutz Beyert/ub

**It's impossible to
make things
foolproof because
fools are so
ingenious**

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 ans 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« an-

zuklicken. 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/ub

Es geht auch ohne Execute

Wir nehmen an, Sie wissen, wie nervig es ist, wenn man zum Ausführen von Batch-Dateien jedesmal schreiben muß:

```
exexecute <Dateiname> (Parameter)
```

Doch den EXECUTE-Befehl vor dem Dateinamen kann man mit einer Workbench ab Version 1.3 auch weglassen. Erstellen Sie eine Textdatei mit z.B. folgendem Inhalt:

```
echo "Hallo, ich bin ein Batch!"
```

```
echo "und schon ist ende"
```

Speichern Sie die Folge unter dem Namen »Test« und geben Sie anschließend ein:

```
protect Test s add
```

Nun können Sie einfach den Dateinamen eingeben und die Datei wird ausgeführt. Wenn das s-Bit gesetzt ist, kann man sich also EXECUTE sparen, denn bevor AmigaDOS die Datei ausführt, überprüft der Amiga, ob dieses Bit gesetzt ist. Wenn ja, erkennt das System, daß es sich um eine Befehlsdatei handelt und führt sie aus. Damit ist auch klar, daß das s-Bit wirklich nur bei Batch-Dateien gesetzt werden darf. Andernfalls ist eine Fehlermeldung die Folge.

Jürgen Reis/ub

Spiel mit der Maus kostet Zeit

Vor allem bei zeitaufwendigen Programmen kommt es oft vor, daß man aus Langeweile die Maus bewegt. Geht es Ihnen auch oft so? Das führt jedoch zu einer Verlangsamung des Programms. Als Beispiel dient eine FOR-NEXT-Schleife in BASIC:

```
FOR i = 1 TO 1000 : NEXT i
```

Im Normalfall benötigt die Schleife etwa 4 Sekunden. Wenn Sie nun ungeduldig mit der Maus spielen, kann sich die Ablaufzeit um bis zu 0,6 Sekunden erhöhen.

Torsten Kersch/ub

MACHEN SIE MIT

Wollen Sie sich an der Rubrik »Tips & Tricks« im AMIGA-Magazin beteiligen? Schicken Sie uns Ihre Tips. Am besten, Sie senden uns Ihre Programme und Texte auf Diskette (ASCII-Format). Für Bastelanleitungen fügen Sie bitte einen Bauplan bei. Die Adresse:

AMIGA-Redaktion

Kennwort: Tips & Tricks

Markt & Technik Verlag AG

Hans-Pinsel-Str.2 8013

Haar bei München

Alle Einsendungen werden von uns geprüft. Wenn Ihr Beitrag in die engere Wahl für eine Veröffentlichung kommt, werden Sie benachrichtigt. Haben Sie auf jeden Fall ein wenig Geduld: Vom Zeitpunkt, an dem Ihr Beitrag die Redaktion erreicht, können bis zum Abdruck im AMIGA-Magazin oder Sonderheft zwei bis drei Monate vergehen.

Rund um Amiga

Die Schatzkiste

Und jetzt was ganz Besonderes: Eine bunte Mischung von Tricks rund um den Amiga, bei der wirklich jeder auf seine Kosten kommen sollte: von Tips zu Festplatten, Public Domain etc., bis zu MS-DOS oder zur ARP-Library. Alles was für Ihre tägliche Arbeit mit dem Amiga nützlich ist. Schauen Sie in Ruhe durch, was wir hier alles aus dem Amiga herauszaubern – es sollte für jeden etwas dabei sein.

Wo tummeln sich die meisten Kunden im Winterschlußverkauf? An den Wühltischen, denn hier findet jeder etwas Brauchbares. Wühlen Sie auch mal auf den nächsten Seiten, hier gibt es jede Menge guter Ideen zum Amiga. Der Unterschied zum Winterschlußverkauf? Die Schatzkiste ist voller Tips, die Sie ganz bestimmt auch im Sommer ganz gut brauchen können. Und Sie brauchen auch nicht an der Kasse Schlange zu stehen – hier ist alles umsonst.

Whats the best way to catch a rabbit???
Hide in a bush and mimic the voice of a carrot

DIE AMIGA-SCHATZKISTE

Tip/Thema	Seite
Platte mit PC-Karte zu langsam	58
Alles anders unter ARP	58
Amiga-Blacklash und MS-DOS	59
Gucklöcher / Fenster ganz klein	59
Langsam, schnell, schneller / Textscrolling	60
Safer Hard-Disk / Festplattenschutz	60
Pipes / ARP-Shell	60
Das sichere RESIDENT / noch mal ARP	61
Ruckzuck entpackt / Batch-Dateien zum Entpacken	61
Disketten unter Amiga-DOS / Diskettenformate	62
Rohrleger im CLI – Pipes	65
Gut gepatcht ist halb gewonnen	65
Laufwerke im Griff / Requester anpassen	66
Externe Programme starten / Execute()	66
BASIC mit nur einem Laufwerk / BASIC-Diskette	66
Ceitansage / Uhr stellen in C	67
Schnelle Platte / Puffer für Festplattenzugriffe	67
Speicheraufteilung / Speicher unter BASIC	67

Platte mit PC-Karte zu langsam

Dank der Janus-Software, die mit der PC-Karte von Commodore geliefert wird, besteht u.a. die Möglichkeit, Festplatten von der PC-Seite aus zu nutzen. Beispielsweise läßt sich eine Autoboot-Partition mit dem Befehl »MakeAB« erstellen. Je größer jedoch die Partition wird, desto langsamer wird, von der PC-Seite aus, der Festplattenzugriff auf sie.

Die Lösung des Problems ist ein größerer Puffer (»Buffer«), in dem die Daten im RAM zwischengespeichert werden. Der Buffer für eine Partition läßt sich mit dem CLI-Befehl ADDBUFFERS erhöhen. Der Aufruf lautet:

```
addbuffers <Partition> <Puffergröße>
```

Ein Beispiel:

```
addbuffers fd0: 200
```

Dabei ist »Partition« der Name der Festplattenpartition, auf der sich das PC-File befindet. »Größe« stellt die Anzahl der Blocks dar, um welche die Größe des Puffers erhöht werden soll. Der optimale Wert hängt von der Größe der PC-Partition ab. Bei 10 MByte hat sich eine Größe von 200 Blocks bewährt. Allerdings sollten Sie ruhig ein wenig experimentieren, um eine optimale Einstellung für Ihr System zu finden.

Christof Brühann/ik

Alles anders unter ARP

Wer sich mit ARP beschäftigt, wird schnell auf Probleme mit dem »*«-Zeichen stoßen. Das »*«-Zeichen wird unter ARP als Joker verwendet, wie unter MS-DOS und Unix üblich. Doch was passiert mit bei eingefleischten Amiga-Usern so beliebten Konstruktionen wie der folgenden?

```
copy * ram:text
```

Die Befehlszeile dient normalerweise dazu, einen kurzen Text zu verfassen, ohne gleich einen Editor zu starten; »*« steht hier für das aktuelle CLI-Fenster, dessen Ausgaben umgelenkt werden. Aber Achtung: Unter ARP hat die Befehlssequenz eine mächtige Wirkung: Ab sofort versucht der Amiga nämlich, alle Dateien des aktuellen Verzeichnisses ins Unterverzeichnis »text« der RAM-Disk zu kopieren, falls es existiert. Durch solche Mißverständnisse kann man recht schnell großen Schaden anrichten. Doch was kann man dagegen tun?

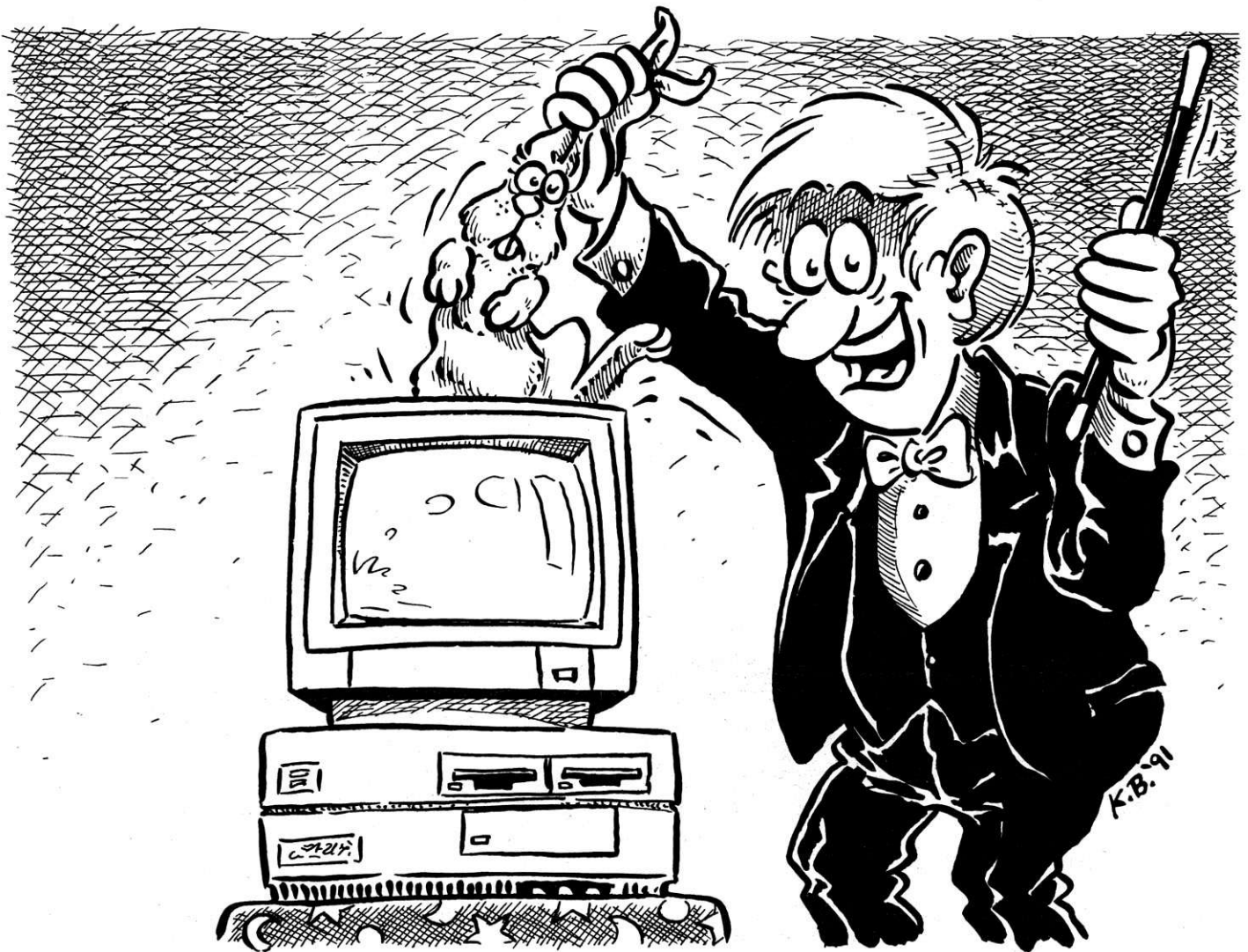
Einmal kann man mit

```
set BCPL True
```

erreichen, daß der Stern (»*«) wieder seine herkömmliche Bestimmung als Identifikation des aktuellen Fensters erlangt. Übrigens können mit dem Kommando

```
set Listall
```

wichtige Systemeinstellungen nachgesehen werden. Im Zweifelsfalle also gleich selber ausprobieren.



Doch diese Lösung ist nüchtern betrachtet eher ein Verlust als ein Gewinn, da man so auf die Wirkung des beliebten Sterns als Joker verzichtet muß. Doch glücklicherweise gibt es auch eine bessere Lösung, da die Arp-Befehle eine direkte Eingabe von Tastatur aus erlauben. Man kann beispielsweise mit

```
type
```

ohne weitere Parameter, Text von Tastatur aus eingeben, der dann sogleich wieder auf dem Standardausgabegerät (hier unser aktuelles Fenster) ausgegeben wird. Somit löst sich auch unser ursprüngliches Problem: Die Anweisung

```
type to ram:text
```

speichert unsere Eingaben von der Tastatur in der Datei »text« der RAM-Disk. Wichtig ist dabei, das Schlüsselwort »to« nicht wegzulassen, sonst sucht der TYPE-Befehl die Datei »text«, die dann als Eingabe fungiert und nicht wie vorgesehen als Ausgabe.

Passen Sie auch darauf auf, daß ohne Arp das »*-Symbol als Escape-Zeichen innerhalb von Namen interpretiert wird. Deswegen sollte man, vorausgesetzt ARP wird benutzt, mit

```
set escape \
```

das »\«-Zeichen zum Escape-Zeichen umbenennen, damit keine Komplikationen auftreten. Zu guter letzt noch ein kleines Beispiel einer ALIAS-Definition, die als Prompt immer die zwei aktuellen CLI/Shell-Variablen ausgibt:

```
alias prom prompt "RC = \\$? R2 = \\$?2 :"
```

Nach der Eingabe von »prom« werden sofort das neue Prompt angezeigt und zugleich die Variablenwerte ständig aktualisiert.

Thomas Becker/ik

Amiga-Backslash unter MS-DOS

Viele Amiga-Anwender, die mit einer XT/AT-Emulation arbeiten, kennen das Problem: Der Backslash ist auf der Amiga-Tastatur nicht an seiner gewohnten Stelle. Es ist zwar recht einfach, durch Eingabe des ASCII-Codes (hier <Alt> drücken und »92« eingeben) das gewünschte Zeichen auf den Bildschirm zu bringen, doch wer durch Eingabe des Verzeichnisnamens in die einzelnen Verzeichnisse der Diskette oder Festplatte wechseln möchte, dem nutzt die Eingabe des ASCII-Codes wenig. Hier hilft der DOS-Befehl PROMT, den man u.a. verwendet, um die Tastatur umzuprogrammieren.

Mittels des Befehls läßt sich der Backslash auf eine nicht benutzte Funktionstaste legen. Hier bietet sich die Taste <F 10> an, da sie erfahrungsgemäß von wenigen Programmen benutzt wird. Folgender PROMT-Befehl belegt die <F 10> Taste mit dem Backslash.

```
prompt $e[0;68;"\p
```

Noch einfacher wird es, wenn man den Befehl in die Datei »autoexec.bat« einbindet, so etwas wie die »Startup-Sequence« für MS-DOS. So wird bei jedem Start des Betriebssystems automatisch die Taste <F 10> mit dem Backslash belegt.

Jürgen Porcher/ub

Gucklöcher

Fenster (z.B. CLI-Windows o.ä.) kann man normalerweise nur bis zu einer bestimmten Größe verkleinern. Ist einem ein Fenster noch

zu groß, läßt es sich durch gleichzeitiges Drücken von <Amiga_links N> und Verkleinern mit der Maus noch weiter schrumpfen. Damit sollte man aber vorsichtig sein, da man das Vergrößerungssymbol bei einem zu kleinen Fenster nicht mehr findet.

Sean Durkin/ub

Langsam, schnell, schneller

Immer, wenn ein Fenster teilweise vor einem anderen liegt, wird das Textscrolling im verdeckten Fenster langsamer und unregelmäßiger, denn der Aufwand, den der Computer beim Textscrolling betreibt, ist umso größer, je mehr Objekte dem Text im Weg sind.

Um ein langsames Scrolling auf Normalmaß zu beschleunigen, muß man alle störenden Objekte zur Seite schieben bzw. hinter dem Fenster plazieren, in dem gescrollt werden soll. Das kann einmal durch Maus-Klick auf das »window-to-front«-Gadget an der rechten oberen Window-Begrenzung geschehen. Aber dieser Schalter ist – natürlich – häufig auch verdeckt.

Praktischer ist es, Allround-Utilities wie »MachII«, »DMouse« oder »QMouse« einzusetzen und den Umweg über die Window-Gadgets zu vermeiden: Neben vielen anderen Vorzügen, die diese Utilities bieten, kann man die »window-to-back«- und »window-to-front«-Operationen auf Maustasten-Kombinationen legen. Beispielsweise wird die »window-to-front«-Operation durch einen doppelten Mausklick auf die linke Taste und »window-to-back« durch Drücken auf die rechte Maustaste, während die linke gehalten wird, erzeugt.

Thomas Becker/ik

I am fully operational and all of my circuits are functioning perfectly. . .

Safer Hard-Disk

Schnell ist es passiert, daß man – oder schlimmer noch, jemand anderes – mit Hilfe des DELETE-Befehls samt Joker-Gewalt wichtige Datenbestände gelöscht hat. Besonders als Festplattenbesitzer ist man durch unbeabsichtigte Löschanöver gefährdet, weil hier der Löschvorgang umso rasanter vor sich geht; und nach dem ersten Schock hat DELETE meist schon eine Menge Files vertilgt.

Es besteht zwar die Möglichkeit, jede Datei durch Löschen des d-Flags mit PROTECT vor Überschreiben und Löschen zu schützen, allerdings ist die Methode aufwendig, um eine ganze Platte zu schützen. Als Alternative existiert der Befehl LOCK, der einen Schreibschutz für die gesamte Platte oder eine Partition, die unter der Verwendung des FastFileSystems formatiert wurde, aktiviert oder deaktiviert. Das kann in der »Startup-Sequence« wie folgt geschehen:

```
lock dh0: on
```

Auch während der Arbeit am Amiga kann man so jederzeit seine Hard-Disk schützen. Deaktiviert wird dieser Schutz mit:

```
lock dh0: off
```

Da der Schutz leicht von jedermann aufzuheben ist, kann man LOCK mit einem Paßwortschutz verwenden. Hierzu dienen die Batch-Dateien »schutz_on.sh« und »schutz_off.sh« (für Anwender des ARP), die am besten im s:-Verzeichnis aufgehoben sind.

```
.key drive
.dol #
failat 21
echo "Enter Password: " noline
read password
lock >nil: <drive#dh0:> on $password
if val $? not eq 0
    echo "Locking-on-Error !"
    quit >nil: 20
else
    echo "**N*e[33m<drive#dh0:> ist schreibgeschützt !! *N*e[0m"
endif
```

»SchutzOn.bat«

Schreibschutz für Ihre Hard-Disk mit Paßwortabfrage

```
.key drive
.dol #
failat 21
echo "Enter Password: " noline
read password
lock >nil: <drive#dh0:> off $password
if val $? not eq 0
    echo "Locking-off-Error !"
    quit >nil: 20
else
    echo "**N*e[33m<drive#dh0:> beschreibbar !! *N*e[0m"
endif
```

»SchutzOff.bat«

Schreibschutz wieder ausschalten

Ruft man die Batch-Dateien ohne Parameter auf, nimmt der Amiga als Festplattenbezeichner »dh0:«. Ansonsten gibt man den Namen der zu schützenden Festplatte mit an.

Generell ist bei der Verwendung von Batch-Dateien (so auch bei »Schutz_off.bat«) darauf zu achten, daß das logische Verzeichnis t: bei aktiviertem Schreibschutz nicht auf das geschützte Gerät verweist, da dort immer die temporären Dateien bei Ausführung einer Batch-Datei abgelegt werden. Ist das nicht mehr möglich, stoppt auch die Ausführung der Batch-Datei. Es empfiehlt sich, mit:

```
makedir ram:t
assign t: ram:t
```

das logische Verzeichnis »t:« in der RAM-Disk anzulegen.

Bei aktiviertem Schreibschutz der Platte ist es möglich, Datenbestände mit ruhigem Gewissen zu nutzen oder nutzen zu lassen. Darüberhinaus hilft der Schreibschutz beim Kampf gegen Viren, sowie gegen die Verwandtschaft, die sich vor dem Rechner »austobend« – hoffentlich nicht tobend, weil sie nicht mehr an die Platte kommen – nun keinen größeren (Daten)schaden mehr anrichten kann.

Thomas Becker/ik

Pipes

Die ArpShell kann mit ihren Pipes den Umgang mit Environment-Variablen (ENV:) vereinfachen. Shellscripts können so benutzerunabhängig geschrieben werden. Ein Beispiel:

Ein Amiga wird von mehreren Personen abwechselnd benutzt. Jeder hat seine Lieblingsumgebung mit anderen Farben, Textanzeigen und Editoren. Man verwendet nun folgende Variablen:

- USER,
- ANZEIGER und
- EDITOR.

Person A setzt sie folgendermaßen:

```
setenv USER Fridolin
setenv ANZEIGER Muchmore
setenv EDITOR dme
```

Die »Startup-Sequence« könnte so aussehen:

```
echo: "Hallo $USER, willkommen am Amiga"
```



```
if $USER eq "Fridolin"
echo "Ich starte DPaint"
run DPaint
endif
if $USER eq "Helga"
echo "Ich lade die Textverarbeitung"
[...]
endif
$ANZEIGER Witz_des_Tages
```

Hier wird die Verwendung der Variablen deutlich:

```
<Variablenname>
```

Interessant ist auch, daß man den Variablennamen wie einen Befehl verwenden kann. Man könnte nun den »Befehl« \$EDITOR auf eine Funktionstaste legen. Je nach persönlicher Definition der Variablen wird ein anderer Editor gestartet. Eine besondere Form der Variablen sind die CLI-Befehle selbst. Ein typisches Beispiel wäre:

```
echo "Heute ist der $(date). Guten Tag"
```

Hier wird das Ergebnis des Befehls DATE in die Ausgabe des ECHO-Befehls eingebaut. Sie sehen, es öffnen sich viele Variationsmöglichkeiten.

Thorsten Gau/ub

Das sichere RESIDENT

Viele Anwender benutzen den RESIDENT-Befehl. Er lädt Programme ein einziges Mal in den Speicher und sorgt so bei erneutem Aufruf des entsprechenden Programms für großen Geschwindigkeitsgewinn, da der Amiga das Programm kein zweites Mal laden muß, sondern direkt ausführt. Wer über genügend Speicher verfügt, kann das Verfahren mit möglichst vielen Programmen nutzen.

Bei CLI-Befehlen ist das alles einfach, doch bei großen Anwendungen gibt es Probleme – die Reise nach Indien ist oft schnell gebucht. Der Grund ist, daß Programme, die resident gemacht werden sollen, ganz spezielle Anforderungen erfüllen müssen. Sie müssen »reentrant« programmiert sein [siehe hierzu entsprechende Fachliteratur]. Doch wer weiß schon, ob das der Fall ist? Manchmal macht sich der Fehler erst nach mehreren Aufrufen bemerkbar. Und genau wenn man denkt, es klappt, kann man den Guru am allerwenigsten brauchen.

Der sicherere Weg ist die Verwendung des ARP-Befehls »ARes« statt Commodores RESIDENT-Routine. Die ARP-Version bildet bei jedem Aufruf des Programms eine Checksumme und informiert den Anwender, falls das Programm fehlerhaft ist. So erfährt man, wenn ein Programm ungeeignet ist resident gemacht zu werden.

Thorsten Gau/ub

Ruckzuck entpackt

Mailboxen bieten oft die Möglichkeit, Programme abzurufen. Meistens sind die Dateien mit einem (oder sogar mehreren) der vielen Archiv- oder Packerprogrammen komprimiert, um die Übertragungszeit zu verkürzen und damit Geld zu sparen. Weit verbreitet sind hier z.B. die Archivprogramme:

```
-- »ARC« (Fish-Disk 70),
-- »ZOO« (Fish-Disk 164),
-- »LHARC« (Fish-Disk 404),
-- das dazu kompatible »LZ« (Fish-Disk 465)
-- oder das etwas ältere »TAR« (Fish-Disk 445).
```

Ein reiner Packer dagegen ist »COMPRESS« (Fish-Disk 6 oder 51). Zur Steigerung der Übertragungssicherheit werden die Dateien oft mit »UUENCODE« (Fish-Disk 92) kodiert.

Weil es recht unpraktisch ist, sich ständig mit diesen verschiedenen Programmen herumzuschlagen, die noch dazu unterschiedliche Syntax haben, hier zwei Shell-Skripts, die das Leben mit Packern erleichtern. Packen Sie die Dateien am besten in den Ordner »s:« Ihres Systems. Das Skript »ALIST.bat« zeigt automatisch mit Hilfe des richtigen Programms die im Archiv enthaltenen Dateien an (»arc«, »zoo«, »lzh«, »tar«-Dateien).

```
.key name/A
.bra {
.ket }
if exists {name}.arc
arc v {name}.arc
SKIP Ende
endif
if exists {name}.zoo
zoo l {name}.zoo
echo ""
skip Ende
endif
if exists {name}.lzh
lharc -x -a v {name}.lzh
skip Ende
endif
if exists {name}.tar
echo "nListing files from archive: '{name}.tar'"
tar -f {name}.tar -tv
echo ""
skip Ende
endif
echo "nKein Archiv {(name)} gefunden!"
quit 5
lab Ende
```

AList.bat

Die Script-Datei zeigt alle in einem gepackten Archiv enthaltenen Dateien

```
.key name/A
.bra {
.ket }
if exists {name}.uue
uudecode {name}.uue
endif
if exists {name}.Z
compress -v -d {name}
endif
if exists {name}.arc
arc x {name}.arc
skip Ende
endif
if if exists {name}.zoo
echo "nExtracting files from archive: '{name}.zoo'"
zoo x// {name}.zoo
echo ""
skip Ende
endif
if exists {name}.lzh
lharc -x -a -m x {name}.lzh
skip Ende
endif
if exists {name}.tar
echo "nExtracting files from archive: '{name}.tar'"
tar -f {name}.tar -xv
echo ""
skip Ende
endif
Echo "nKein Archiv {(name)} gefunden!"
quit 5
lab Ende
```

AX.bat

entpackt ein Archiv direkt ins aktuelle Verzeichnis

Die Skript-Datei »AX.bat« geht sogar noch einen Schritt weiter: Sie entpackt ein Archiv direkt ins aktuelle Verzeichnis. Haben Sie z.B. eine Datei mit Namen »supergame.uue« aus einer Mailbox bekommen, genügt der Befehl

```
AX.bat supergame
```

um die ausführbaren Dateien zu erhalten. Es könnte etwa folgendes geschehen:

»supergame.uue« wird mit »UUENCODE« rückkodiert zu »supergame.lzh«. Diese Datei wurde also mit »LHARC« gepackt. Der

Amiga entpackt nun das Archiv, wobei die einzelnen Dateien im aktuellen Verzeichnis landen, z.B. als »supergame«, »supergame.info« und »supergame.highscores«. Und schon kann man losspielen.

Andreas Mittermayr/Martin Ramsch/ub

Disketten unter AmigaDOS

Niemals waren Festplatten so qualitativ und preisgünstig wie heute. Dennoch ist und bleibt die normale 3-1/2-Zoll-Diskette immer noch der am meisten verbreitete Datenträger auf dem Amiga. Einige Besonderheiten im programmier-technischen Umgang mit den kleinen schwarzen Scheiben muß man allerdings kennen.

Der Amiga gehört zu den wenigen Computern, die standardmäßig zwei verschiedene File-Systeme für die Abwicklung der Disk- und Festplatten-Ein- und Ausgaben liefern. Von beispielsweise dem C 64 kennt man noch die lästige Praxis, sich sog. Floppy-Beschleuniger o.ä. erst nachträglich organisieren zu müssen.

```
; FileLen Autor Georg Sonnenberg
; Fälscht die Längen-Angabe eines
; beliebigen Disketten-Files

execbase = 4
allocmem = -198
freemem = -210
findtask = -294
addport = -354
remport = -360
oldopenlib = -408
closelib = -414
opendevice = -444
closedevice = -450
doio = -456
write = -48
output = -60
lock = -84
unlock = -90
info = -114

main:
move.l a7,stack ; Stackpointer merken
move.l a0,a4 ; Parameter-Text
move.l d0,d4 ; Länge Parameter-Text
sub.l a2,a2
bsr open_dos ; dos.lib aufmachen
bsr get_param ; Parameter holen
bsr alloc ; Speicher saugen
bsr get_block ; File lokalisieren
bsr open_trackdisk ; trackdisk öffnen
bsr modify ; File modifizieren
exg a5,a6
bra quit ; Abschluß ->

open_dos:
move.l execbase.w,a6
lea dosname(pc),a1
jsr oldopenlib(a6) ; Library öffnen
tst.l d0
move.l d0,a5 ; dosbase merken
beq exit ; Fehler? ja: ->
rts

get_param:
clr.b -1(a4,d4.w) ; Abschluß Parameter-Text
moveq #0,d1 ; diverse Arbeitsregister
moveq #10,d2
moveq #32,d3
moveq #48,d4
param_loop_10:
move.b (a4)+,d1 ; Character aus Parameter
beq param_error
cmp.b d3,d1 ; SPACE?
beq.s param_loop_10 ; ja: weitersuchen ->
lea -1(a4),a3 ; Zeiger auf Filename
param_loop_20:
move.b (a4)+,d1 ; Character aus Parameter
beq param_error
cmp.b d3,d1 ; SPACE?
bne.s param_loop_20 ; nein: weitersuchen ->
clr.b -1(a4) ; Abschluß Filename
param_loop_30:
move.b (a4)+,d1 ; Character aus Parameter
beq param_error
```

Das normale, ursprüngliche DOS-File-System unterscheidet sich in einem wesentlichen Punkt von üblichen DOS-Systemen anderer Computern:

Die Inhaltsverzeichnisse auf Disketten liegen nicht standardmäßig an einer bestimmten Stelle auf der Disk, etwa auf einem bestimmten Track, sondern sind im Extremfall über die ganze Diskette verteilt. Außerdem enthält jede Datei auf der Diskette neben dem Header-Block (mit grundsätzlichen Informationen wie Name der Datei, Länge, Erstellungsdatum) und eventuellen Extension-Blöcken (die bei großen Dateien die einzelnen Datenblöcke einander zuweisen) in ihren eigentlichen Daten-Blöcken weitere, sich wiederholende Informationen. Durch diese Redundanzen kann immer von einem beliebigen Datenblock auf den Datei-Header und auf die anderen Datenblöcke geschlossen werden und umgekehrt, vom Datei-Header auf die evtl. übergeordneten Verzeichnisse und umgekehrt etc.

(Um ebenfalls Redundanzen, hier aber mit anderen Computerbüchern zu vermeiden, werden die Einzelheiten der Disk-Blöcke und

```
cmp.b d3,d1 ; SPACE?
beq.s param_loop_30 ; ja: weitersuchen ->
moveq #0,d5 ; Summe löschen
param_loop_40:
sub.b d4,d1 ; ASCII in Zahlenwert
bmi param_error ; umwandeln
cmp.b d2,d1 ; keine Zahl?
bge param_error ; ja: Fehler ->
mulu d2,d5 ; Summe errechnen
add.l d1,d5
move.b (a4)+,d1 ; Parameter zuende?
beq.s param_ok ; ja: ->
cmp.b d3,d1 ; SPACE?
bne.s param_loop_40 ; nein: weiteraddieren ->

param_ok:
rts
alloc:
move.l #32+80+36+512,d0
move.l #$10003,d1 ; Chipmem löschen
jsr allocmem(a6) ; Speicher anfordern
tst.l d0 ; hat geklappt?
beq mem_error ; nein: ->
move.l d0,a2 ; sonst Speicher merken
rts

get_block:
exg a5,a6 ; dosbase nach a6
move.l a3,d1 ; Zeiger auf Filenamen
moveq #-2,d2 ; Zugriff Lesen
jsr lock(a6) ; Lock = File suchen
move.l d0,d1 ; gefunden?
beq filename_error ; nein: ->
add.l d0,d0 ; BCPL-Pointer umwandeln
add.l d0,d0
move.l d0,a3 ; Zeiger auf Lock
move.l 4(a3),d6 ; Block-Nr. des Files
move.l d1,d7
lea 32+80(a2),a3
move.l a3,d2
jsr info(a6) ; Infos über Disk holen
move.l d0,d4
move.l d7,d1
jsr unlock(a6) ; Lock wieder freigeben
tst.l d4 ; Infos über Disk ok?
beq filename_error ; nein: ->
move.l 32+80+4(a2),d7 ; Disk-Unit des Files
rts

open_trackdisk:
exg a5,a6 ; execbase nach a6
sub.l a1,a1 ; eigener Task
jsr findtask(a6) ; Task-Adresse holen
move.l a2,a1 ; reservierter Speicher
move.l d0,16(a1) ; Task eintragen
jsr addport(a6) ; Message-Port anmelden
lea 32(a2),a1 ; reservierter Speicher
move.l d7,d0 ; Disk-Unit des Files
moveq #0,d1 ; Flags löschen
lea trdname(pc),a0
jsr opendevice(a6) ; trackdisk.dev öffnen
tst.l d0 ; hat geklappt?
bne device_error ; nein: ->
rts
```

-Strukturen an dieser Stelle nicht erläutert. Wenn Sie sich dafür interessieren, werfen Sie einen Blick z.B. in die Bücher »Amiga Intern« oder »Das große Floppybuch«, beide von Data Becker.)

Die Entwickler der Daten-Organisation schufen ein System mit scheinbar hohen Sicherheitsreserven: Wenn ein Block oder gar ein Track auf der Diskette z.B. durch Beschädigung der Diskette, durch Programm-Fehler o.ä. zerstört wird, könnte der Inhalt der Diskette mit relativ geringem Aufwand durch die zahlreichen Verkettungen der einzelnen Blöcke wieder rekonstruiert werden.

Diese Sicherheitsreserve ist aber nur eine scheinbare, denn wenn die zerstörten Blöcke oder Tracks Programm- oder andere wichtige Daten enthielten, nützt es auch nichts, wenn man zwar weiß, wo auf der Diskette welche Blöcke zu welchen Files gehören, die Blöcke selbst aber nicht wiederhergestellt werden können und die Files z.T. zerstört bleiben.

Vielmehr bringt dieses halbdurchdachte System lästige Nachteile mit sich, z.B. längere Wartezeiten zum Anzeigen der Directories,

längere Ladezeiten von Programmen und anderen Dateien, da die Verkettungen untereinander auch immer vom Betriebssystem überprüft, Checksums errechnet werden müssen etc. Außerdem muß ein Datenblock immer von seinen Verwaltungsbytes befreit werden, d.h. seine reinen Daten müssen im Speicher noch umkopiert werden. Ferner fordern die Verwaltungsdaten auch ihren Platz, der den eigentlichen Programm- oder anderen Daten dann natürlich fehlt.

Letzteres ist auch der Grund dafür, warum man beispielsweise ein File der Länge 110 000 Byte nicht auf eine Diskette kopieren kann, die mit vielleicht noch 111 000 freien Byte scheinbar genug Platz zu Verfügung stellt: Die Verwaltungsdaten des Files müssen auch noch untergebracht werden, und dafür reicht der Platz dann nicht mehr.

Aus diesen Gründen entwickelte Commodore bald das sog. Fast-File-System. Dieser neue Handler ist nicht nur in einer schnelleren und kürzeren Assembler-Version programmiert, sondern erspart sich auch fast alle o.g. arbeitsintensiven Verwaltungsaufgaben der Dateien. Ein Datenblock einer Datei enthält hier tatsächlich 512 Byte

```
modify:
lea 32(a2),a1 ; reservierter Speicher
move.l a2,14(a1) ; Message-Port eintragen
move.w #2,28(a1) ; Command: Lesen
move.l #512,36(a1) ; Länge 1 Block
lea 32+80+36(a2),a3 ; reservierter Speicher
move.l a3,40(a1) ; als Buffer eintragen
moveq #9,d0
lsl.l d0,d6 ; Block-Nr. * 512
move.l d6,44(a1) ; als Offset eintragen
jsr doio(a6) ; Block einlesen
tst.b 31(a1) ; Fehler aufgetreten?
bne dos_error ; ja: ->
lea 32+80+36(a2),a3 ; Buffer
move.l a3,a4
move.l d5,324(a3) ; neue Länge eintragen
moveq #0,d1
move.l d1,20(a3) ; alte Checksum löschen
moveq #512/4-1,d2 ; Zähler 127 Langworte
checksum:
sub.l (a4)+,d1 ; neue Checksum errechnen
dbra d2,checksum
move.l d1,20(a3) ; neue Checksum eintragen
move.w #3,28(a1) ; Command: Schreiben
jsr doio(a6) ; Block zurückschreiben
tst.b 31(a1) ; Fehler aufgetreten?
bne dos_error ; ja: ->
move.w #4,28(a1) ; Command: Update Track
jsr doio(a6) ; Block sofort schreiben
tst.b 31(a1) ; Fehler aufgetreten?
bne.s dos_error ; ja: ->
move.w #9,28(a1) ; Command: Motor
clr.l 36(a1) ; Flag: Motor aus
jmp doio(a6) ; Motor ausstellen
close_trackdisk:
move.l a2,d2 ; reservierter Speicher
beq.s do_not_rem ; gar keiner da: ->
lea 32(a2),a1
tst.l 14(a1) ; trackdisk.device offen?
beq.s do_not_close ; nein: ->
jsr closedevice(a6) ; sonst schließen
do_not_close:
move.l a2,a1
tst.l 16(a1) ; gibt's Message-Port?
beq.s do_not_rem ; nein: ->
jmp remport(a6) ; sonst abmelden
do_not_rem:
rts
free:
move.l a2,d2 ; reservierter Speicher?
beq.s do_not_free ; gar keiner da: ->
move.l a2,a1
move.l #32+80+36+512,d0
jmp freemem(a6) ; sonst wieder freigeben
do_not_free:
rts
param_error:
move.l #param_txt,d2 ; Zeiger auf Fehlermeldung
move.l #param_txt_len,d3
bra.s error
mem_error:
```

```
move.l #mem_txt,d2
move.l #mem_txt_len,d3
bra.s error
filename_error:
move.l #file_txt,d2
move.l #file_txt_len,d3
exg a5,a6
bra.s error
device_error:
move.l #device_txt,d2
move.l #device_txt_len,d3
bra.s error
dos_error:
move.l #dos_txt,d2
move.l #dos_txt_len,d3
error:
exg a5,a6 ; dosbase nach a6
jsr output(a6) ; Output-Handle holen
move.l d0,d1 ; Fehler?
beq.s quit ; ja: ->
jsr write(a6) ; sonst Message ausgeben
quit:
move.l a6,a1
move.l a5,a6
jsr closelib(a6) ; dos.lib schließen
bsr close_trackdisk ; trd.device schließen
bsr free ; Speicher freigeben
exit:
moveq #0,d0 ; Fehlerflag löschen
move.l stack(pc),a7 ; Stack ggf. korrigieren
rts ; das war's
stack: dc.l 0
param_txt:
dc.b "Parameter sind nicht okay.",10,13
dc.b "Usage is: FileLen <Filename> <NewLen>"
dc.b 10,13,0
param_txt_len = *-param_txt
file_txt:
dc.b "Kann das File nicht finden.",10,13,0
file_txt_len = *-file_txt
mem_txt:
dc.b "Nicht genügend Speicher.",10,13,0
mem_txt_len = *-mem_txt
device_txt:
dc.b "Kann trackdisk.device nicht öffnen."
dc.b 10,13,0
device_txt_len = *-device_txt
dos_txt:
dc.b "DOS-Fehler, kann Block nicht schreiben."
dc.b 10,13,0
dos_txt_len = *-dos_txt
dosname:
dc.b "dos.library",0
trdname:
dc.b "trackdisk.device",0
```

FileLenght.asm

Der Angabe der Länge einer Datei sollten Sie nicht trauen, mit diesem PProgram kann man sie fälschen

Daten – alle Verwaltungsarbeiten wie Checksums errechnen, Daten umkopieren etc. entfallen. Diese Features, oder besser: der Verzicht auf die bisherigen Features machen das FastFileSystem natürlich schneller. Außerdem können evtl. verlorene Dateien auf Diskette immer noch nachträglich wieder gefunden werden, wenn zwischenzeitlich keine essentiellen Blöcke gelöscht wurden, etwa durch nachträgliche Schreibzugriffe.

Ist dieser Sachverhalt der zwei File-Systeme für uns nun von Vorteil? Ein definitives »Jein«! Das FFS wird nahezu ausschließlich auf Festplatten angewendet; für Disketten ist es deshalb wenig geeignet, weil die meisten Disketten im normalen DOS-Format vorliegen und es ständig lästige Kompatibilitätsprobleme zwischen beiden Formaten gäbe.

Ist denn das alte DOS-System deshalb schlechter? Ein provokantes »Nein«! Denn nicht alle seine Redundanzen werden ausgiebig überprüft, und diese Unachtsamkeiten im Betriebssystem kann man sich für verschiedene Aufgaben zunutze machen. (Einiges davon funktioniert auch im FFS.)

Da ist z.B. die Sache mit der Datei-Länge. Gibt man den folgenden DOS-Befehl ein:

```
list c:dir
```

erhält man eine Antwort nach folgendem Muster:

```
dir 2200 --p-rwed 02-Mai-1990 16:58:10
```

Die angegebene Länge 2200 Byte ermittelt der LIST-Befehl aber nicht, indem er die Datei tatsächlich nachmißt, sondern indem er auf den entsprechenden Eintrag im Header-Block (Offset 324) zurückgreift. Das AmigaDOS errechnet diesen Wert immer nur dann, wenn eine Datei neu erstellt wird.

Ein Linkvirus nun, der sich in ein bestehendes Programm einfügt, kann diesen Wert so fälschen, daß das infizierte Programm immer noch die gleiche Länge wie vorher angibt, obwohl es in Wirklichkeit

um den Virus erweitert wurde. Insofern ist der Ratschlag, mit dem LIST-Befehl die Längen diverser Files zu überprüfen und so auf evtl. Linkviren zu schließen, nicht sehr sinnvoll.

Wie einfach es geht, die Länge eines beliebigen Files auf Diskette zu manipulieren, zeigt das auf den letzten beiden Seiten abgedruckte Assembler-Programm »FileLen.s« (Seka-Listing)

Assemblieren Sie das Programm und speichern Sie es als Object-Datei unter dem Namen »FileLen«. Wenn Sie keinen Assembler haben oder mögen, können Sie sich auch des nachfolgenden BASIC-Loaders bedienen: Starten Sie ihn mit dem BASIC-Befehl RUN, danach befindet sich das Programm FileLen in der RAM-Disk.

Wenn Sie »FileLen« ausprobieren möchten, geben Sie erst den Filenamen der Datei ein, dessen Länge Sie manipulieren wollen, und danach die gewünschte, neue Länge. Überzeugen Sie sich mit LIST vom Ergebnis.

• Vorher:

```
list c:dir
```

gibt die Ausgabe:

```
dir 2200 --p-rwed 02-Mai-1990 16:58:10
```

• Nach dem Befehl:

```
FileLen c:dir 10
```

gibt LIST aus:

```
dir 10 --p-rwed 02-Mai-1990 16:58:10
```

Achtung: Sie müssen nach dem Aufruf von »FileLen« die Diskette aus dem Laufwerk holen und nochmal einlegen, oder DISKCHANGE benutzen, damit der LIST-Befehl die neuen Längenangaben auch erkennt. Sie sind aber auf jeden Fall auf Disk geschrieben.

Wie Sie sehen, kann man jedem beliebigen File fast jede beliebige Länge zuweisen (sie muß nur in einem Langwort untergebracht werden können), ohne daß ein evtl. Hinweis des Datums darüber noch Zeugnis ablegt. Das modifizierte Programm bleibt auch unabhängig

```
REM FileLen Autor Georg Sonnenberg
```

```
REM Dieser BASIC-Loader kann z.B. von AmigaBasic
REM aus aufgerufen werden. Er generiert dann das
REM lauffähige Programm 'FileLen' in der Ram-Disk.
```

```
OPEN "0",1,"RAM:FileLen"
```

```
FOR i = 1 TO 888
  READ x
  PRINT#1,CHR$(x);
NEXT
```

```
CLOSE 1:KILL "RAM:FileLen.info"
```

```
DATA 0,0,3,243,0,0,0,0,0,0,0,2,0,0,0,0,0,0
DATA 0,1,0,0,0,199,0,0,0,0,1,0,0,3,233,0,0,0
DATA 199,35,207,0,0,2,40,40,72,40,0,149,202,97
DATA 0,0,28,97,0,0,46,97,0,0,120,97,0,0,142
DATA 97,0,0,196,97,0,0,234,205,77,96,0,1,232
DATA 44,120,0,4,67,250,2,201,78,174,254,104,74
DATA 128,42,64,103,0,1,228,78,117,66,52,64,255
DATA 114,0,116,10,118,32,120,48,18,28,103,0,1
DATA 108,178,3,103,246,71,236,255,255,18,28,103
DATA 0,1,94,178,3,102,246,66,44,255,255,18,28
DATA 103,0,1,80,178,3,103,246,122,0,146,4,107
DATA 0,1,68,178,2,108,0,1,62,202,194,218,129
DATA 18,28,103,4,178,3,102,232,78,117,32,60,0
DATA 0,2,148,34,60,0,1,0,3,78,174,255,58,74
DATA 128,103,0,1,40,36,64,78,117,205,77,34,11
DATA 116,254,78,174,255,172,34,0,103,0,1,34,208
DATA 128,208,128,38,64,44,43,0,4,46,1,71,234,0
DATA 112,36,11,78,174,255,142,40,0,34,7,78,174
DATA 255,166,74,132,103,0,0,254,46,42,0,116,78
DATA 117,205,77,147,201,78,174,254,218,34,74,35
DATA 64,0,16,78,174,254,158,67,234,0,32,32,7
DATA 114,0,65,250,2,7,78,174,254,68,74,128,102
DATA 0,0,224,78,117,67,234,0,32,35,74,0,14,51
DATA 124,0,2,0,28,35,124,0,0,2,0,0,36,71,234
DATA 0,148,35,75,0,40,112,9,225,174,35,70,0,44
DATA 78,174,254,56,74,41,0,31,102,0,0,186,71
DATA 234,0,148,40,75,39,69,1,68,114,0,39,65,0
DATA 20,116,127,146,156,81,202,255,252,39,65,0
DATA 20,51,124,0,3,0,28,78,174,254,56,74,41,0
```

```
DATA 31,102,0,0,140,51,124,0,4,0,28,78,174,254
DATA 56,74,41,0,31,102,122,51,124,0,9,0,28,66
DATA 169,0,36,78,238,254,56,36,10,103,26,67,234
DATA 0,32,74,169,0,14,103,4,78,174,254,62,34
DATA 74,74,169,0,16,103,4,78,238,254,152,78,117
DATA 36,10,103,12,34,74,32,60,0,0,2,148,78,238
DATA 255,46,78,117,36,60,0,0,2,44,38,60,0,0,0
DATA 68,96,56,36,60,0,0,2,142,38,60,0,0,0,27
DATA 96,42,36,60,0,0,2,112,38,60,0,0,0,30,205
DATA 77,96,26,36,60,0,0,2,169,38,60,0,0,0,38
DATA 96,12,36,60,0,0,2,207,38,60,0,0,0,42,205
DATA 77,78,174,255,196,34,0,103,4,78,174,255
DATA 208,34,78,44,77,78,174,254,98,97,0,255,112
DATA 97,0,255,140,112,0,46,122,0,4,78,117,0,0
DATA 0,0,80,97,114,97,109,101,116,101,114,32
DATA 115,105,110,100,32,110,105,99,104,116,32
DATA 111,107,97,121,46,10,13,85,115,97,103,101
DATA 32,105,115,58,32,70,105,108,101,76,101,110
DATA 32,60,70,105,108,101,110,97,109,101,62,32
DATA 60,78,101,119,76,101,110,62,10,13,0,75,97
DATA 110,110,32,100,97,115,32,70,105,108,101,32
DATA 110,105,99,104,116,32,102,105,110,100,101
DATA 110,46,10,13,0,78,105,99,104,116,32,103
DATA 101,110,252,103,101,110,100,32,83,112,101
DATA 105,99,104,101,114,46,10,13,0,75,97,110
DATA 110,32,116,114,97,99,107,100,105,115,107,46
DATA 100,101,118,105,99,101,32,110,105,99,104
DATA 116,32,246,102,102,110,101,110,46,10,13,0
DATA 68,79,83,45,70,101,104,108,101,114,44,32
DATA 107,97,110,110,32,66,108,111,99,107,32,110
DATA 105,99,104,116,32,115,99,104,114,101,105,98
DATA 101,110,46,10,13,0,100,111,115,46,108,105
DATA 98,114,97,114,121,0,116,114,97,99,107,100
DATA 105,115,107,46,100,101,118,105,99,101,0,0
DATA 0,2,255,0,0,0,3,236,0,0,0,6,0,0,0,0
DATA 0,0,0,2,0,0,1,190,0,0,1,204,0,0,1,218
DATA 0,0,1,234,0,0,1,248,0,0,0,0,0,3,242
DATA 0,0,3,235,0,0,0,1,0,0,3,242
```

BASIC-Lader.bas

Mit diesem Lader erzeugen Sie das Programm »FileLen« auch ohne Assembler

von seinem Längen-Wert lauffähig; der DIR-Befehl in unserem Beispiel funktioniert immer noch tadellos.

Wer sich also vor Linkviren schützen möchte, sollte sich niemals auf die LIST-Angaben seiner Files verlassen.

Kann man denn etwas gegen die gefälschten Werte tun? Ein selbstsicheres »Klar«! Erstens könnte man ein Programm schreiben, welches das zu untersuchende File mißt, d.h. über seine Datenblocks hangelt und die Bytes tatsächlich nachzählt.

Eine zweite und einfachere Möglichkeit wäre, AmigaDOS zum Nachzählen zu veranlassen. Das geschieht einfach, indem man das zu untersuchende File irgendwohin kopiert, z.B. in die RAM-Disk, denn wenn ein File neu erstellt wird (in diesem Fall die Kopie), wird der Längen-Wert wie oben bereits gesagt neu errechnet. Ein kleines Beispiel dazu:

```
copy c:dir ram:
list c:dir
gibt die Ausgabe
dir          1000000 --p-rwed 02-Mai-1990 16:58:10
Und der folgende Befehl:
list ram:dir
gibt die tatsächliche Länge:
dir          2200 --p-rwed 23-Jul-1990 20:59:00
```

Reentrant code – the only way to fly.

Gibt der LIST-Befehl zwei unterschiedliche Werte aus, ist an der Länge des Files herumgespielt worden.

Lassen Sie sich aber nicht durch das unterschiedliche Datum verwirren; da die Kopie später entstand als das Original, besitzt es zwangsläufig ein anderes Datum.

Genauso wie die Längen-Angaben der Files verändert werden können, kann man auch die Datums-Angaben fälschen. Wenn Amiga-DOS im Normalfall ein File erstellt, wird es ihm immer das aktuelle Datum mit auf den Weg geben. Ein Virus oder ein anderes garstiges Programm kann diesen Wert danach aber sofort wieder auf den vorherigen Wert zurücksetzen und so den eben ausgeführten Schreibzugriff verschleiern.

Mit derartigen Taschenspieler-Tricks lassen sich aber nicht nur Linkviren füttern, sondern auch der seriöse Anwender kann Kapital herauschlagen.

Wie bereits erwähnt, besteht ein Datenblock im normalen DOS-Format nicht nur aus den reinen Daten, sondern auch aus Verwaltungsangaben. Im Langwort bei Offset 12 ist angegeben, wieviele reine Datenbytes der betreffende Datenblock enthält. In der Regel ist das der Wert \$1E8 = 488 Byte. Da das File aber in den seltensten Fällen ein ganzzahliges Vielfaches von 488 Byte lang ist, enthält der letzte Datenblock einen entsprechend geringeren Wert. Wenn also von diesem letzten Datenblock nicht alle 488 möglichen Bytes genutzt werden, weil beispielsweise nur die ersten 240 Byte noch zum File gehören, ein Datenblock aus organisatorischen Gründen aber immer 488 Byte zur Verfügung stellt, ergeben sich im letzten Datenblock immer einige ungenutzte, nicht definierte Daten. Hier ist Platz; in diesem Bereich kann man nun beispielsweise eine Seriennummer, ein Paßwort oder andere Informationen ablegen – was man gerade möchte.

Obwohl diese Daten im strengen Sinne zu keinem Programm gehören, ist ihre Position dennoch gegen evtl. Überschreiben geschützt. Sie sind also sicher.

Wird das File allerdings z.B. auf eine andere Disk kopiert, werden diese unbenutzten Daten nicht mitübertragen. Will man nun feststel-

len, ob man eine Originaldiskette besitzt, kann man entweder mit einem Disk-Monitor am Ende des letzten Datenblocks nachsehen, ob z.B. das geheime Paßwort noch im Block enthalten ist, oder vom betreffenden Programm aus selbst den entsprechenden Block 'heimlich' einlesen und überprüfen. Ist die gesuchte Information nicht mehr da, so wurde dieses File mindestens einmal über das AmigaDOS kopiert. Uns steht hier also ein kleiner Kopierschutz vom Betriebssystem zu Verfügung.

Bestimmt ist das nicht der ultimative Kopierschutz aller Zeiten: Wenn man gleich die ganze Diskette kopiert, werden die scheinbar ungenutzten Daten sehr wohl übertragen – die Disk wird ja vollständig dupliziert. Wohl aber kann dieser Schutz das Kopieren von und zu Festplatten oder das Verbreiten über Mailboxen, Netzwerke etc. erschweren.

Übrigens hat man auch die Möglichkeit, die Data-Größen der einzelnen Datenblöcke mitten in der Datei zu manipulieren und die geheimen Daten damit noch viel besser zu verbergen. Besteht ein File beispielsweise aus zehn Datenblöcken, kann man durchaus den fünften Block von den standardmäßigen 488 Byte auf beispielsweise 200 Byte drosseln und in die freigewordene Stelle einen kleinen geheimen Text einfügen. Wird das File dann vom DOS geladen oder kopiert, bleibt der gewünschte Effekt der gleiche:

Die scheinbar ungenutzten Daten werden ignoriert und das File neu strukturiert in voll mit 488 Bytes ausgenutzten Größen. (Dadurch paßt es dann u.U. in eine Länge von nur noch neun Datenblöcken.)

Ein Problem bleibt jedoch die Erstellung eines solchen Files, das nach dieser Herzeingriff-Operation immer noch lauffähig bleiben soll. Am ehesten geeignet scheinen für diese Methode Textfiles zu sein, deren Inhalt man mit einem Disketten-Monitor leicht von Hand editieren kann.

Weitere Manipulationen mit den strukturellen Informationen auf der Diskette führen zu kaum brauchbaren Ergebnissen. Ein witziger Gag aus der Anfangszeit des Amiga bestand darin, den Next-Hash-Eintrag eines Directory-Blocks (Offset 496) auf sich selbst zu setzen. Wenn man sich dieses Directory dann mit dem DIR-Befehl ansehen möchte, gibt es sich immer wieder selbst aus. Zum Glück sind diese Scherze vorbei.....

Georg Sonnenberg/ub

Rohrleger im CLI

Zum Amiga Replacement Project (ARP) gehört u.a. eine eigene Shell (ASH). Sie unterscheidet sich von Commodores Shell auf der Workbench-Diskette (ab V. 1.3; NEWSHELL newcon:) nur darin, daß sie zusätzlich eine tolle Pipeunterstützung bietet.

»Was soll das denn?«, werden viele jetzt fragen, »ich habe doch schon PIPE: und die benutze ich sowieso nie.«

Doch die Pipes unter ARP können mehr. Die grundsätzliche Syntax lautet:

```
Befehl1 | Befehl2 (| Befehl3 | ... | ...)
```

Das bedeutet, daß die Ausgabe von Befehl1 als Eingabe für Befehl2 verwendet wird. Ein paar praktische Beispiele:

```
dir | more ; gibt aktuelles Verzeichnis seitenweise aus
```

```
list | sort ; gibt alphabetisch sortiertes Directorylisting aus
```

Eine gute Idee ist es, solche Konstruktionen als ALIAS in die »Shell-Startup« zu schreiben, um sie immer parat zu haben.

Thorsten Gau/ub

Gut gepatcht ist halb gewonnen

Es darf gepatcht werden. Der erste Patch vergrößert beim Seka-Assembler die Zahl der sichtbaren Zeilen im EDIT-Modus von 14 auf 24.

```
; Patch für den Seka-Assembler
$006c6 4e71 4e71
$006d7 0b
$008c9 30
$0090d 17
```

```
$00951 18
$0098c 39
$009b2 39
$00a42 39
$00a4d 18
$050e0 36
$050f1 32
$0517f "256"
```

Alles was Sie zum Patchen brauchen, ist das Programm »ProPatch« aus der AMIGA 1/89, Seite 64. Mit diesem Werkzeug und der Patch-Befehlsdatei können Sie den Seka-Assembler entsprechend modifizieren. Falls Sie nicht über dieses Werkzeug verfügen, können Sie auch das Programm FileZap aus der Public Domain nehmen. In der Patch-Liste steht immer die Adresse, die im Programm verändert werden muß und der Wert bzw. die Werte, die an/ab dieser Adresse einzusetzen sind.

Auch den zweiten Patch veröffentlichen wir in Form einer Patch-Befehlsdatei für Propatch. Der Patch paßt den Editor »MicroEmacs« von der Extras-Diskette 1.2 an die europäische Bildschirmauflösung von 256 Zeilen in der Vertikalen an:

```
; PAL-Patch für MicroEmacs im Ordner
; Tools der Extras-Diskette 1.2
$075c8 0100
$075d0 0200
$075e2 0100
$092e8 0200
$092ff 3d
$09306 0100
$09311 1d
$09317 1d
$09328 0100
$09958 0100
$09a79 1d
$09a9b 1d
$09dea 0100
$09e0e ".PAL"
$09e1d f2
$09e3e 0100
$09e42 0100
```

Sollten Sie eines Ihrer Programme patchen wollen, denken Sie daran, vorher eine Sicherheitskopie zu erstellen. So sind Sie vor Fehlern sicher.

Peter Rohde/ub

Calm down. It's just ones and zeroes.

Laufwerke im Griff (Requester)

Sicherlich kennen Sie alle die kleinen Boxen (Dateiauswahl-Fenster, Requester) mit Gerätebezeichnungen (DF1,DF2,DH0) in den Diskmenüs von DeluxePaint, Zing, usw.....

Haben Sie sich auch schon darüber geärgert, daß dort »DH0« steht, obwohl man vielleicht über eine Filecard im PC-Slot (Sidecar) verfügt, also im Janussteckplatz. Warum kann dort also nicht »JH0« stehen? Nun damit ist jetzt Schluß:

Als erstes braucht man einen Filemonitor, z.B. »Filezap«. Als Programm wählen wir zum DeluxePaint. Das Programm wird über den Filemonitor geladen und kann begutachtet werden (Sicherheitskopie verwenden). Die Änderung ist recht einfach. Man sucht im Program-

mcode die Bezeichnungen »DH0« und ersetzt sie durch »JH0«, »VD0« oder ähnliches – Zurückschreiben nicht vergessen. Von nun an können Sie auch ihre Festplatte, eine resetfeste RAM-Disk oder ein 5.25-Zoll-Laufwerk direkt anklicken. Genauso können Sie im übrigen PAL-Versionen erzeugen. Suchen Sie im Programmcode z.B. von ED nach »RAW« oder »CON«. Hinter diesen Kürzeln stehen die Koordinaten des Fensters. Passen Sie diese Ihren eigenen Bedürfnissen an.

Robert Albrecht/ub

Externe Programme starten

Wer die Befehle des CLI oder der Shell in Programmen nutzen möchte, kann das über den Befehl »Execute()« der »dos.library«. Der Funktion sind folgende Parameter zu übergeben:

- D1: Zeiger auf einen mit Null abgeschlossenen Text. Der Text gibt das auszuführende Kommando an; er entspricht also genau der Anweisung, die Sie im CLI eintippen würden.
- D2: File-Handle einer Text-Datei oder eines CLI-Fensters. Hier wird normalerweise eine Null eingetragen.
- D3: Angabe über Ausgabe-Datei. Übergibt man eine Null, erscheinen die Ausgaben im aktuellen CLI-Fenster.

Hier ein Beispiel:

```
execute:
move.l #4,a6
lea dosname,a1 ; Adresse des Bibliotheks-Namen
move.l #0,d0
jsr -408(a6) ; »OpenLib()« aufrufen
move.l d0,a6
move.l #befehl,d1 ; Zeiger auf Befehlszeile
move.l #0,d2 ; Null eintragen
move.l #0,d3 ; Ausgabe-Datei -> CLI-Fenster
jsr -222(a6) ; Execute() aufrufen
rts befehl: dc.b "dir",0 ; CLI-Befehl DIR
dosname: dc.b "dos.library",0
```

Wichtig: Damit der Trick funktioniert, muß sich der Befehl RUN im Verzeichnis »C:« befinden.

Claudio Pitrella/ub

BASIC mit nur einem Laufwerk

Wer mit nur einem Diskettenlaufwerk in BASIC programmieren möchte, wird sich bald darüber ärgern, daß er immer wieder die Workbench-Diskette einlegen muß. So muß z.B. für den Befehl SAY das »narrowrot.device« von der Startdiskette nachgeladen werden. Diese Diskettenwechsel kann man sich ersparen, wenn man Amiga-BASIC direkt auf die Workbench-Diskette kopiert. Folgende Schritte sind hierzu nötig:

- Sie sollten zunächst mit DISKCOPY eine Kopie der Workbench erstellen und Ihren Amiga mit dieser Kopie neu starten.
- Um auf der Workbench-Kopie Platz zu schaffen, löschen Sie mit DELETE alle Files, die Sie mit Sicherheit nicht brauchen. Dies wären beispielsweise:
 - Unnötige Druckertreiber (im Ordner: devs/printers)
 - Einige Tastaturlayouts (im Ordner: devs/keymaps)
 - Eventuell sogar einige Befehle aus dem c-Directory
 - Utilities, wie das Notepad, die Uhr (Clock) oder den Rechner (Calculator)

Kopieren Sie anschließend AmigaBASIC und das zugehörige »info«-File von der ExtrasD-Diskette mit folgenden Befehlen auf die neue Diskette:

```
copy ExtrasD:AmigaBASIC to SYS:
copy ExtrasD:AmigaBASIC.info to SYS:
```

Ändern Sie die »Startup-Sequence«, damit AmigaBASIC direkt nach dem Start aufgerufen wird. Eine Änderung könnte wie folgt aussehen:

```
System/SETMAP d
loadwb
```


AmigaBASIC
endcli

Speichern Sie die neue Startdatei mit <ESC x>. Die beiden letzten Anweisung müssen Sie nur durchführen, wenn das Amiga-BASIC von der Workbench-Diskette auch automatisch gebootet werden soll.

Thomas Freitag/ub

Do you work to live or live to work?

Ceitansage

»SDatum.c« ist ein Programm für Besitzer eines Amiga ohne akkugesperrte Echtzeituhr.

Bei jedem Neustart des Amiga sollten Sie das Datum und die Zeit mit DATE im CLI setzen. Das kann, wenn Sie den Amiga häufig per Reset oder durch Einschalten starten, mit der Zeit sehr lästig werden – und es kostet Zeit.

»SDatum.c« erleichtert Ihnen die Eingabe und fragt Sie während der Amiga weiter an der »Startup-Sequence« arbeitet, nach dem aktuellen Datum und der Uhrzeit.

```
/* SDatum */

#define bold printf("\033[1m\033[1m\033[11m")
#define norm printf("\033[0m")

void main()
{
    char datum[11],zeit[10],befehl[25];
    bold;
    printf("SDatum\033[0m by SK\n");
    printf("Bitte Datum (dd-mm-yy): ");
    scanf("%s",&datum);
    printf("Bitte Zeit (hh:mm:ss) : ");
    scanf("%s",&zeit);
    printf("\nMoment...n");
    sprintf(befehl,"date %s %s",datum,zeit);
    Execute(befehl,0L,0L);
}
```

SDATUM.C«

Ein Beispiel, wie man mit »Execute()« aus einem C-Programm heraus CLI-Befehle aufruft

Bauen Sie den Aufruf von »SDatum.c« lediglich in die »Startup-Sequence« ein. Das C-Programm führt dann mittels »Execute()« aus der »exec.library« den Befehl DATE mit den eingegebenen Daten aus. DATE muß sich selbstverständlich auf der Startdiskette befinden. Das Format des Datums ist TT-MMM-JJ, z.B. 18-Jul-75 oder 03-Dez-91. Das Format für die Zeit ist SS:MM:SS. Dabei kann die Angabe der Sekunden weggelassen werden, z.B.: 17:45:30 oder 18:24. Das Programm wurde mit Aztec C 3.4a geschrieben und folgendermaßen übersetzt:

```
cc prg.c +l
ln prg.o -lc32
```

Sören Knebel/ik



Schnelle Platte

Eine Festplatte ist im Vergleich zu einem Diskettenlaufwerk schnell. Es ist sogar möglich, die Festplatte noch schneller zu machen. Das Geheimnis liegt in der »Mountlist«. Man findet Sie im Verzeichnis »devs« einer Start-Diskette. Ändern Sie die Datei mit einem Texteditor (MEmacs von der Extras- oder ED von der Workbench-Diskette). Fügen Sie die Zeile ein:

```
Buffers = 200
```

Für die Daten der Festplatte wird durch diesen Befehl eine Art RAM-Disk angelegt. Bei häufigen Zugriffen auf die Hard-Disk stehen dann viele Daten noch im Puffer und müssen nicht von der Platte geladen werden. Ein Nachteil: Das Verfahren kostet Speicher.

Norbert Cohen/ub

Speicheraufteilung

Wenn man in AmigaBASIC programmiert und mehr Speicher braucht als die beim Start des BASIC-Interpreters zur Verfügung gestellten 25000 Byte, benutzt man normalerweise den Befehl CLEAR. Doch oft funktioniert der Befehl nicht, auch wenn FRE(0) anzeigt, daß noch reichlich Speicherplatz vorhanden sei. Wissen Sie woran das liegen könnte?

Der Fehler liegt an CLEAR selbst: Der AMIGA versucht, erst neuen Speicher zu reservieren, bevor er den alten Bereich freigibt. Um doch ans Ziel zu kommen, gibt es eine einfache Methode: Man setzt zuerst den Bereich auf seine Mindestgröße herab, die bei 1024 Byte liegt.

```
CLEAR ,1024
```

Danach gibt man die gewünschte Zahl an:

```
CLEAR , 500000
```

Die Methode sollte man aber nur im Direktmodus anwenden. Im Programmablauf sollte man mit

```
CLEAR, 25000-FRE(0) ' benötigter Programmspeicher
```

```
CLEAR, FRE(-1)-50000 ' gesamter freier Speicher
```

arbeiten. Die Methode reserviert immer den gesamten zur Verfügung stehenden Speicher abzüglich einer Extraportion fürs Betriebssystem (im Bsp. 50 000 Byte).

ub

Public Domain

Öffentliche Hilfe

Public Domain ist eine feine Sache. Für fünf Mark bekommt man eine ganze Diskette voller Programme, Tools, Utilities, Spiele oder ähnliches. Die Auswahl ist riesig; allein die Fish-Serie enthält mittlerweile über 550 Disketten. Da sollte man schon mal reinschauen, genauso wie in unsere Tips zu Public-Domain-Programmen.

Sie brauchen sich nicht zu verstecken, die Public-Domain-Programme. Teilweise findet man auf PD-Disketten hochwertige Programme, wie Textverarbeitungen, Assembler, Compiler, die sich mit kommerziell vertriebenen Produkten messen können. Nicht selten ist ein PD-Programm sogar allen kommerziellen Vertretern überlegen.

Nur den Überblick über das gewaltige Angebot an PD-Disketten zu wahren, ist eine Kunst, deshalb hier ein paar Ratschläge zu besonders praktischen Programmen.

Laute Laufwerke

Haben Sie sich schon über lautes »Röhren« eines Laufwerks beim Einlesen eines Directories geärgert? Abhilfe schafft »SlowMe« im Verzeichnis »A68kex« der Fish-Disk 431.

Das Programm beschleunigt auch die Diskettenzugriffe allgemein und ist wegen seiner geringen Größe (260 Byte) wie geschaffen für den C-Ordner Ihrer Workbench-Diskette.

Sean Durkin/ub

Funktionstasten mit Funktion

Wollten Sie sich auch schon einmal einen eigenen Tastaturreiber, z.B. für komfortablere Nutzung des CLI, erstellen? Genau diese Aufgabe übernimmt »KeyMapEd« von der Fish-Disk 388. Mit dem Programm ist es z.B. möglich, die Funktions-Tasten mit CLI-Befehlen zu belegen. Der neue Tastaturreiber kann dann nach Eingabe von

```
setmap Treibername  
benutzt werden.
```

Sean Durkin/ub

Der Amiga tippt selber

Kennen Sie das? Sie lassen gerade eine Anleitung auf dem Monitor ausgeben, in der beispielsweise folgendes steht:

```
> Geben Sie bitte ein:  
SuperProgramm opt ik-u+p NAME myfile  
OBJFILE myfile.o ..... <  
..uSW.....
```

Schon geht das mühsame Abtippen los...»S.u.p.e. ...«.
Einfacher geht's mit einem PD-Snap-Utility (z.B.: »OSnap«): Sie halten im Fenster, in dem die Anleitung steht, einfach die Taste <Amiga_links> fest und umrahmen mit der Maus den gewünschten Text. Dann klicken Sie ins CLI-Window und drücken die Kombination

<Amiga_links >. Schon erscheint der zuvor ausgeschnittene Text wie von Geisterhand getippt in der Kommandozeile.

Snapper lassen sich auch gut anwenden, wenn man sich in einem Text auf andere Texte beziehen möchte: Einfach den Ursprungstext ausscheiden und im eigenen Text einfügen.

Viele Snapper bieten darüberhinaus die Möglichkeit, Grafik auf dieselbe Weise irgendwo auszuscheiden, um sie zu speichern oder weiterzuverarbeiten.

Thorsten Gau/ub

Fish ist gesund

Virusschutz ist wichtig. Wenn man aber jedesmal, wenn man feststellen möchte, ob der Speicher virusfrei ist, einen Virenkiller hervorkramen, neu booten und sich durch die manchmal umständliche Benutzeroberfläche bis zum »Check memory«-Befehl quälen muß, wünscht man sich schnell eine bessere Medizin gegen die bedrohliche Virenplage.

Es gibt eine: »VMK« von der Fish-Disk 328. Das Programm testet den Speicher auf 27 (!) bekannte Viren und andere resetfeste Programme, und warnt ggf. den User. Es besitzt keine Benutzeroberfläche, da es als CLI-Befehl benutzt wird, und benötigt nur wenig Platz auf einer Diskette. Allerdings lassen sich die Viren nur aus dem Speicher, nicht aber von Disketten entfernen.

Sean Durkin/ub

The fastest routine is the one you don't call.

Entpacken inbegriffen

Wer viel mit Public-Domain-Software arbeitet, kennt sicher folgendes Problem: Zu den meisten Programmen gibt es ausführliche Anleitungen, die man einerseits ab und zu braucht, welche aber andererseits zu lang sind, um sie auf derselben Diskette zu speichern, auf der sich das Programm befindet.

Viele Anwender legen sich deshalb Anleitungsdisketten an und benutzen Programme wie LHARC oder ZOO, um die Dateien zu komprimieren. Das hat einen entscheidenden Nachteil: Möchte man eine Anleitung lesen, muß man erst das Archiv mit dem Packer entpacken, um die Datei dann normal zu lesen.

Einfacher geht's mit den PD-Programmen »PowerPacker« und »MuchMorePopa«. Mit »PowerPacker« packt man einzelne Dateien. Um sie später direkt zu lesen, verwendet man dann »MuchMorePopa« (Fish 378). Es liest die Datei und zeigt sie ungepackt an. Die Methode hat zwei Vorteile: Erstens hat man die Möglichkeit, Textdateien platzsparend zu archivieren, und zweitens gehört »MuchMore« mit seinem butterweichen Softscrolling und vielen nützlichen Funk-

tionen zu den besten Textanzeigern auf dem Amiga, den man sich unbedint anschauen sollte.

Thorsten Gau/ub

Fish als Fensterputzer

Ist es Ihnen schon passiert, daß Sie ein Window aus irgendeinem Grund nicht mehr schließen konnten, oder daß Sie etwas an einem Fenster verändern wollten, aber nichts funktionierte?

Das Programm »Fenster« von der Fish-Disk 305 hilft. Es besitzt Funktionen, um alle sichtbaren Windows beliebig zu manipulieren. Man kann alle IDCMP-Flags und vieles mehr ändern. Es besteht zusätzlich die Möglichkeit, Windows als IFF-Datei zu speichern oder Fensterinhalte direkt auszudrucken.

Sean Durkin/ub

CustReq statt ASK

Wenn man nur auf Wunsch ein bestimmtes Programm beim Start des Amiga initialisieren oder eine Befehlssequenz ausführen lassen möchte, benutzt man den Befehl ASK in der »Startup-Sequence«. Allerdings ist er sehr umständlich. Er erwartet vom User eine Tasteingabe, was bei einem mausorientiertem Computer fast ein Verbrechen ist.

Abhilfe schafft hier »CustReq«; der Befehl ersetzt ASK und ist zu finden auf der Fish-Disk 242 (Public Domain; Version 2.0).

»CustReq« öffnet ein Fenster, und man wählt mit der Maus zwischen zwei Gadgets (z.B. »JA« für y(es) und »NEIN« für n(o)). Man hat auch die Möglichkeit, eine Zeitbegrenzung zu wählen, nach der der Requester verschwindet. Falls man in dieser Zeit keines der Gadgets drückt, wird der Default-Wert angenommen (entweder »JA« oder »NEIN«).

ub

Plotter

Einen Preferences-kompatiblen (meist Matrix-) Drucker hat fast jeder. Dann noch einen Plotter kaufen? Brauchen könnte man ihn ab und zu ja schon...

...aber auch nur ab und zu; ein paar schöne Programme, die einen HP-Plotter ansprechen, gibt es ja (mCad, Multiplot), aber lohnt sich dafür ein Plotter?

Die Antwort wird für die meisten »nein« lauten, es geht ja auch viel billiger: Auf Fish-Disk 373 (V1.3b) bzw. 333 (V1.3) befindet sich das Programm »PLT:«, das über einen kleinen Eintrag in der Datei »mountlist« (im Verzeichnis »devs:«) angemeldet wird:

```
PLT:      Handler   = L:plt-handler
          StartUp   = PRA/000-1
          StackSize = 5000
          Priority   = 5
          GlobVec    = 1
```

Plotter.mnt

»mountlist«-Eintrag, um das Plotterprogramm von der Fish-Disk 373 anzumelden

Außerdem muß »PLTDATA:« auf ein Verzeichnis zeigen, in dem die Fonts für das Programm stehen. Am besten kopiert man es nach »Fonts:«.

Mit dem Kommando:

```
mount PLT:
```

im CLI wird das Programm aktiviert. Daten, die an einen Plotter geschickt werden sollen, werden dann einfach nach PLT: geschickt.

Das Ganze sähe z.B. so aus:

```
copy Plot.plt to PLT:
```

Mit den zusätzlichen Parametern sollten Sie ein bißchen herumspielen. So werden durch

```
copy Plot.plt to PLT:S/000-3
```

die mit Stift 0 gezeichneten Linien 3 Punkte dick gezeichnet (S gibt die Zeichnung auf einem Bildschirm aus). So wird aus einem normalen Drucker auch noch ein guter Plotter. Mit Farbdruckern ist sogar ein Farbausdruck möglich. Und billiger als ein richtiger Plotter ist diese Lösung allemal.

ub

AmigaDOS bei der Arbeit zusehen

Woran liegt's? Ein PD-Programm läuft nicht, obwohl es bei Ihren Bekannten funktioniert. Wahrscheinlich fehlen dem Programm wichtige Libraries oder es benötigt Dateien in bestimmten Verzeichnissen. Was tun?

Auf der Fish-Disk 451 befindet sich das Programm »SnoopDOS«. Einmal gestartet zeigt es im CLI an, was gerade mit Ihren Dateien passiert. Hierzu gehören Schreib/Lese-Zugriffe, Löschen von Dateien, Wechsel des aktuellen Directories oder das Ausführen von Programmen. Hinter jeder DOS-Aktion wird angezeigt, ob sie erfolgreich war oder nicht. So läßt sich sofort sehen, auf welche Dateien ein Programm zugreifen möchte.

Das ist auch dann praktisch, wenn man sich eine private Workbench einrichten möchte und sich nicht sicher ist, ob eine bestimmte Datei gebraucht wird oder gelöscht werden kann.

Thorsten Gau/ub

Schreib/Lese-Fehler ade

Schreib- und Lese-Fehler auf Disketten kann man nie ausschließen. Sie entstehen z.B., wenn Staub das korrekte Schreiben bzw. Lesen der Daten verhindert. Fehler liegen beispielsweise vor, falls eine Diskette falsche Track-Header besitzt (dienen zum Strukturieren einer Diskette) oder ein Sektor eine falsche Checksumme erhält.

Ein Nachteil: Der Amiga kann beim Auftreten von Fehlern einen ganzen Track (11 Sektoren) nicht mehr richtig lesen, auch wenn nur ein oder zwei Sektoren defekt sind. Möglicherweise sind die zerstörten Sektoren sogar leer, so daß es zu keinerlei Datenverlust käme, wenn man die intakten Sektoren wieder lesen könnte. Doch wie kann man Spuren restaurieren? Genau das ermöglicht das Programm »TrackSalve()« von der Fish-Disk 355 (Public Domain). Der CLI-Aufruf zum Retten der Daten lautet:

```
TrackSalve as
```

Es ist sicher auch von Vorteil, Datenverluste schon beim Schreiben auf Diskette abzufangen, d.h. beim Schreiben auf Diskette zu testen, ob eine Datei korrekt übertragen wurde (Verify). Auch hierbei hilft TrackSalve. Es bietet die Möglichkeit, geschriebene Daten zu

PUBLIC-DOMAIN-RATGEBER

Tip/Thema	Seite
Laute Laufwerke / Diskettenzugriffe	68
Funktionstasten mit Funktion / KeyMap-Editor	68
Der Amiga tippt selber / Eingabe direkt vom Screen übernehmen	68
Fish ist gesund / Virenkiller	68
Entpacken inbegriffen / Packer selbst für Libraries	68
Fish als Fensterputzer / Fensterdaten ändern	69
CustReq anstatt ASK / Bedienungshilfe	69
Plotter / Plotten statt drucken	69
AmigaDOS bei der Arbeit zusehen / Diskettenzugriffe überwachen	69
Schreib/Lese-Fehler ade / Diskettenschutz	69
Konvertierungen / AmigaDOS ruft MS-DOS	70
Besser im Bild / Bilder auf den Screen bannen	70
Noch ein Packer / Dateien komprimieren	70

überprüfen. Tritt hierbei ein Fehler auf, kann man die Daten erneut auf Diskette schreiben. Der Aufruf hierfür lautet:

TrackSalve av

Der Einsatz von TrackSalve verlangsamt den Schreibzugriff etwas, allerdings erhöht sich die Datensicherheit. Weiterhin bietet TrackSalve die Funktion, das Klicken der leeren Laufwerke zu unterbinden, und TrackSalve schaltet auch den Laufwerksmotor ab, falls ein Programm dies vergessen sollte.

Die Aufrufe:

- für das Nicht-Klicken

TrackSalve an

- für die Motor-Kontrolle

TrackSalve au

Da TrackSalve kaum Speicher oder Rechenzeit benötigt, sollte man es stets in der »Startup-Sequence« aufrufen. Es bietet sich z.B. folgender Aufruf an:

TrackSalve avu ln

Mit »ln« wird das Klicken nur für »df1:« unterdrückt, was sinnvoll ist, wenn in »df0:« die Startdiskette steckt.

ub

Computers don't make errors, just slight mistakes

Konvertierung

Auf Anfragen, wie man mit dem Amiga MS-DOS-Disketten schreiben bzw. lesen kann, wird oft auf »Cross2DOS« verwiesen. Es geht aber auch billiger: Auf Fish 382 befindet sich das »MSH:«-System. Hiermit wird ein Device geschaffen, das man wie eine Diskette ansprechen kann: statt DF0: nimmt man MSH:.

Die Bedienung ist dadurch denkbar einfach. Zum Installieren muß nur die »mountlist« (zu finden im Verzeichnis »devs:«) um einen Eintrag erweitern:

```
MSH: FileSystem = L:MessyFileSystem
Device = messydisk.device
Unit = 0
Flags = 0
LowCyl = 0 ; HighCyl = 79
Reserved = 0
Surfaces = 2
BlocksPerTrack = 9
Buffers = 5
DosType = 1
BufMemType = 1
BootPri = 0
Stacksize = 4096
Priority = 5
GlobVec = -1
Mount = 1
```

msh_mountlist

**Wichtig zur Installatioin: der Eintrag für die Datei
»mountlist« im Verzeichnis »devs:«**

Außerdem muß man die notwendigen Files nach »DEVs:« (»messydisk.device«) und »L:« (»MessyFileSystem«) kopieren.

Wenn das neue Device mit

MOUNT MSH:

angemeldet ist, erkennt das Programm beim Einlegen einer Diskette ins entsprechende Laufwerk automatisch, ob es eine AmigaDOS- oder eine MS-DOS-Diskette ist. Der einzige Nachteil ist, daß man ständig eine »unbekannte« Diskette im Laufwerk hat, solange sich eine MS-DOS-Diskette in diesem befindet, aber damit sollte man leben können.

ub

Besser im Bild

Meistens setzt man am Amiga Programme wie »ShowIFF« oder »PicView« ein, um auf Diskette gespeicherte IFF-Bilder auf den Bildschirm zu bannen.

- Eine einfache Alternative, Bilder auf die Mattscheibe zu bringen, ist es, die Bilder in ausführbare Programme umzuwandeln. Dies läßt sich einfach mit dem auf der CACTUS 34 (PD) enthaltenen Befehl »IFF-Ex« erledigen. Die Syntax lautet:

IFF-Ex Bildname Programmname

Das Verfahren hat den Vorteil, daß sich die Bilder bequem mit einem Doppelklick aufrufen lassen, vorausgesetzt sie besitzen ein Pictogramm (Icon). Natürlich können Sie die Bilder weiterhin vom CLI aus aufrufen, einfach durch Eingabe des Namens, den Sie für den Parameter »Programmname« wählen. Des weiteren können Sie die »Bildprogramme« packen (crunchen), z.B. mit dem »Power-Packer« oder »PackIt«, so daß eine Menge Bilder auf einer Diskette Platz haben. Es besteht auch die Möglichkeit, ein Intro-Bild zu schaffen: Dazu setzt man den Programmnamen an die erste Stelle der »Startup-Sequence«. Nach Einlegen der Diskette erscheint das CLI-Fenster (wie immer), kurz darauf erscheint das Intro-Bild. Es wird solange angezeigt, bis Sie die linke Maustaste drücken. Danach führt der Amiga die »Startup-Sequence« weiter aus.

- Der Befehl »IFF-Mod« befindet sich ebenfalls auf der CACTUS 34. Damit lassen sich Bilder- und Musik-Dateien in Form eines aufrufbaren Programms kombinieren, das bei seinem Aufruf das Bild anzeigt und dazu Musik spielt. Der Aufruf lautet:

IFF-Mod Bildname Modulname Programmname

Allerdings ist es nur möglich, Soundtracker-Module zu verwenden. Die fertigen Programme lassen sich, wie die Bilder durch einen Doppelklick oder vom CLI aus durch Eingabe des Namens starten. Voraussetzung für ein einwandfreies Funktionieren ist: Die Befehle »IFF-Ex« und »IFF-Mod« müssen sich im C-Verzeichnis Ihrer Diskette befinden, sowie Bilder und Sound-Module im Stammverzeichnis (Root-Directory).

Detlev Kraft/ub

Noch ein Packer

Kein Platz mehr auf der Workbench-Diskette, obwohl noch ein weiteres Utility gewünscht wird – was tun? Abhilfe dagegen schafft das Shareware-Programm »Imploder« von der »Kickstart PD-370«. Dieser Cruncher (Packer) erlaubt es, Dateien auf bis zu 50 Prozent der ursprünglichen Länge zu reduzieren und trotzdem lauffähig zu erhalten.

Das Besondere am »Imploder« ist, daß man neben CLI-Befehlen und Programmen auch Libraries, Fonts und Batch-Dateien komprimieren kann; lediglich Handler (im »l:«-Verzeichnis) und CLI-Befehle, die resident gemacht werden (z.B. DIR), sollte man nicht crunchen, da es sonst eine Reise nach Indien gratis gibt (GURU!). Der »Imploder« benutzt eine Library (»explode.library«), um die Dateien zu decrunchen, so ist es auch möglich, Batch-Dateien zu packen und trotzdem auszuführen. Die Library muß – ungepackt – auf Ihrer Arbeits-Diskette im libs-Verzeichnis stehen. Eine mit dem »Imploder« behandelte Workbench-Diskette 1.3 ist je nach Crunchmode (siehe Imploder-Anleitung) nur noch zu 30 bis 40 Prozent des vorher gebrauchten Platzes belegt, so das wieder etliche KByte frei sind.

ub

Programmieren Hardware

Bis jetzt ging's meist um Programmierung und die Bedienung des Amiga – doch es existiert noch eine andere Seite: die Hardware. Und klar, daß es auch hierzu ein paar Tips gibt, wie sich mehr aus dem Amiga herauskitzeln läßt.

Sie brauchen jetzt nicht gleich den Lötcolben herauszukramen, es geht auch größtenteils ohne. In den folgenden Beiträgen wollen wir uns vor allem mit Dingen beschäftigen, die ohne großen Aufwand umzusetzen sind, u.a. Tips zu Monitoren, Druckern und Timingfragen.

Einstellungssache

Ein Trick für alle, die mit einem 1081/84-Monitor an ihrem Amiga arbeiten: Mit zwei Drehknöpfen an der Hinterseite (»Horizontal Width« <-> und »vertikal Hight« ±) verändern Sie das Bild in der Breite und Höhe. Machen Sie das Bild einmal mit beiden Reglern so klein wie möglich (Regler im Uhrzeigersinn bis zum Anschlag drehen). Anschließend steht das Bild allerdings nicht mehr in der Mitte; korrigieren Sie die Position mit dem Einsteller in den »Preferences«.

Die neue Einstellung hat mehrere Vorteile:

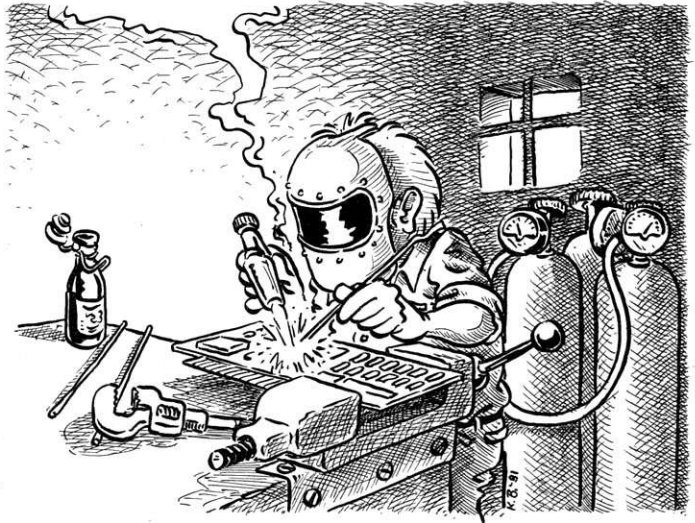
- Man kann den Overscan-Modus ohne (oder mit weniger) lästigen Verzerrungen am Rand benutzen. Flexible Programme wie Cygnus Ed Professional (CED), Deluxe Paint II und III unterstützen den Overscan-Modus. Man kann so mit CED einen Screen in der Größe von 704*276 Punkten benutzen. Damit lassen sich 84*33 Zeichen darstellen, 481 Zeichen mehr als mit der normalen Auflösung.
- Der Treppen-Effekt wird ein wenig eingedämmt, weil der Pixelabstand kleiner ist.

Herwig Höller/ub

**It's not possible to
both understand
and appreciate
Intel CPUs**

Widerstand an Power-LED

Beim Amiga bietet sich die Möglichkeit, den Lowpass-Filter mit Hilfe der Steuerung der Power-LED ein- bzw. auszuschalten, so daß am Zustand der Power-LED auch der des Lowpass-Filters abzulesen ist. Doch leider ist das bei den neuen Amigamodellen nicht mehr



so leicht: Die Leuchtdiode geht hier beim Ausschalten nicht mehr ganz aus, sondern wird nur ein wenig dunkler. Oft ist dann nicht genau festzustellen, in welchem Zustand sich die LED eigentlich befindet. Abhilfe schafft hier ein Widerstand von 68 Ω, der parallel zur LED geschaltet wird. Die LED wird dadurch im eingeschalteten Zustand kaum dunkler, erlischt aber beim Ausschalten ganz. Soll nur der Kontrast erhöht, ein völliges Verdunkeln der LED jedoch verhindert werden, empfiehlt es sich, etwas größere Widerstandswerte zu probieren.

Christof Brühann/ub

Stereo bei Mono-Monitoren

Man hat nur einen Mono-Monitor, möchte aber die Stereo-Effekte des Amiga ausnutzen. Dazu benötigt man nur einen Stereoverstärker am besten mit CD-Eingang und ein Cinch-Kabel; es ist bei CD-Playern meistens im Lieferumfang enthalten und im Fachhandel auch einzeln zu kaufen (ca. 15 Mark). Nun verbindet man den Ton-

HARDWARE-TRICKS

Tip/Thema	Seite
Einstellungssache / Bild zentrieren mit Preferences	71
Widerstand an Power-LED / Lowpass-Filter	71
Stereo bei Mono-Monitoren / Amiga an Stereoanlage	71
Hardware-Timing / Rasterstrahl zur Synchronisation	72
Drucker steuern / Prt: vs. Par:	72
Coprozessor 68881 und 68882 / Matheprozessor nutzen	72
SonixPatch / Sonix-Installation auf Festplatte	72
Sicherung / Portstecker A 2000	73
Umstieg leichtgemacht / c 64 und Amiga an einem Monitor	73
Aktion »Saubere Maus« / Maus säubern	73

Ausgang des Amiga und den CD-Eingang mit dem Cinch-Kabel (auf richtige Farben achten) und hat endlich den langersehnten Stereo-Effekt. Man sollte den Computer am CD-Eingang anschließen, da andernfalls die Qualitätseinbußen zu hoch sind.

ub

Hardware-Timing

Wer ein aufwendiges Spiel programmieren möchte, kommt nicht umhin, die Amiga-Bremse namens Betriebssystem abzuschalten. Ist das Betriebssystem aber ausgeschaltet, kann der Programmierer auch nicht mehr die bequemen System-Funktionen nutzen, sondern muß sich um alles selber kümmern. Und dabei entstehen manchmal Fehler, die später dafür verantwortlich sind, daß das eine oder andere Programm nicht mehr auf einem Amiga mit einem schnelleren (oder auch langsameren) Prozessor läuft.

Ein wichtiges Thema ist in diesem Zusammenhang die Programmierung von Warteschleifen:

– Obwohl bereits seit Jahren davon abgeraten wird, zu diesem Zweck einfach eine simple Zählschleife mit dem Prozessor abzuarbeiten, weil beispielsweise ein 68020 viel schneller fertig wäre als ein 68000, wird die Methode ihrer Einfachheit halber immer wieder verwendet.

– Natürlich kann man Programme mit Hilfe des »VerticalBlank«-Interrupts synchronisieren, der alle 1/50 Sekunde ausgelöst wird und den Bildschirmaufbau steuert. Die Auflösung von nur einer 50stel Sekunde reicht aber für viele Anwendungen nicht aus.

– Und die Programmierung der CIA-Timer ist nicht jedermanns Sache, da sie relativ kompliziert ist.

Deshalb im folgenden nun eine Alternative:

Synchronisieren Sie ein Programm mit dem Rasterstrahl. In der PAL-Norm dauert eine Rasterzeile etwa 63 Mikrosekunden, diese Auflösung sollte den allermeisten Erfordernissen genügen. Wenn Sie beispielsweise eine Zeit von sechs Millisekunden warten wollen (das ist die offizielle Wartezeit zwischen zwei Step-Impulsen für die Bewegung der Leseköpfe in den Disketten-Laufwerken), entspricht das etwa 95 Rasterzeilen, die Sie warten müssen. Die folgende Routine erledigt das für Sie:

```
wait:                ; Übergabe: d7 = Anzahl der Rasterzeilen

    lea    $dff006,a6
    move.b (a6),d6    ; aktuelle Zeile

loop:
    cmp.b  (a6),d6    ; auf nächste Zeile
    beq.s  loop       ; warten
    move.b (a6),d6
    dbra   d7,loop    ; das x-mal machen
    rts
```

wait.asm

Nutzen Sie den Rasterstrahl für Ihr Programm-Timing

Die Routine wartet immer, bis sich der Wert der aktuellen Rasterzeile ändert, und das so oft, wie Sie in d7 angegeben haben.

Dieses Timing ist nicht ganz präzise, da strenggenommen auch noch die aktuelle horizontale Position berücksichtigt werden müßte. Im Extremfall würde die Routine also fast eine Zeile zu wenig warten. Da der Zähler in d7 aber durch den »dbra«-Befehl nicht bis Null, sondern bis -1 runterzählt, wird das wieder ausgeglichen. Die Routine wartet also ggf. etwas zu lange, aber niemals zu wenig. In der Praxis kann diese geringe Abweichung im Mikrosekunden-Bereich in den allermeisten Fällen ohnehin vernachlässigt werden.

Insgesamt ist die Routine ähnlich kurz und unkompliziert wie eine einfache Zählschleife, arbeitet aber unabhängig vom eingebauten Prozessor auf allen Rechnern gleich zuverlässig.

Georg Sonnenberg/ub

Drucker steuern

Wer versucht, den Drucker als Ausgabegerät mit PRT: anzusprechen, erlebt häufig eine Überraschung, wenn er Steuerzeichen übertragen möchte: Der Drucker reagiert nicht auf alle Codes. Der Amiga filtert Ausgaben über PRT: und siebt einige Steuerzeichen heraus. Möchte man dennoch Steuerzeichen an den Drucker schicken, muß man sich an den ANSI-Standard halten. Er ist im Benutzerhandbuch des Amiga im Anhang C beschrieben. Ein Beispiel: Um bei einem Epson-Drucker Schmalschrift zu aktivieren, müßte man laut Druckerhandbuch das Steuerzeichen CHR\$(15) verwenden. Es wird aber von PRT: ausgesiebt. Man muß das Standardäquivalent

CHR\$(27); CHR\$(91); CHR\$(52); CHR\$(119)

bemühen. Die Kurzschreibweise lautet:

ESC[4w

Der Standard hat den Vorteil, daß die möglichen Steuerzeichen für alle Drucker gleich sind. Die eigentliche Übersetzung der Standardzeichen des Amigas in Steuerzeichen für den Drucker übernimmt der Druckertreiber. Bei einem Druckerwechsel muß man an einem Programm nichts ändern, sondern nur einen anderen Treiber einstellen.

Doch was macht man, wenn man Zeichen an den Drucker schicken möchte, die vom Amiga nicht übersetzt sondern nur herausgesiebt werden? Ganz einfach, man schickt die Daten pur über das Parallel-Device »PAR:« bzw. über »SER:«, wenn der Drucker an der seriellen Schnittstelle angeschlossen ist. Mit dieser Methode kann man alle Steuerzeichen so an den Drucker schicken, wie Sie auch im Handbuch des Druckers stehen.

Dieter Braun/ub

Coprozessoren 68881 und 68882

Wer eine Turbokarte mit Matheprozessor besitzt (68881 oder 68882), wird schnell bemerken, daß viele Programme die schnellen Coprozessoren gar nicht nutzen.

Auf der »AmigaLibDisk« 319 (Fish-Disk) befindet sich ein Ersatz für die »mathtrans.library«. Wenn man die neue »mathrans.library« installiert, laufen auch die Programme schneller, die mathematische Funktionen einsetzen, aber keine spezielle Version für die schnelleren Coprozessoren besitzen.

ub

SonixPatch V1.4 Version

Die Aufgabe: Sonix V1.4 soll auf Festplatte.

Das erste Hindernis: Sonix benutzt einen Loader, der das eigentliche Programm Sonix.code im »Miscellaneous«-Ordner in den Speicher schreibt, der Loader sucht also immer auf Diskette die Datei »Sonix.code«.

Das zweite Hindernis: Das Programm »Sonix.code« sucht seine Instruments und Scores entweder auf DF0: oder auf DF1:. Das sieht man bereits am File-Requester.

Die Lösung:

1) Gehen wir davon aus, daß das Musikprogramm in folgendem Pfad installiert wird:

DH0:Sound/Sonix

Am schnellsten geht's, wenn Sie auf der Workbench eine leere Schublade (Empty-Ordner) aktivieren und DUPLICATE auswählen (Workbench-Menü). Taufen Sie die Schublade mit RENAME in »Sound« um. Dasselbe muß dann mit der Sonixschublade geschehen, nur daß wir diese dann ins bestehende Sounddirectory packen.

2) Jetzt müssen wir die Sonix-Umgebung schaffen. Dazu kopieren wir von der Sonix-Diskette folgende Subdirectories und Files in den geschaffenen Sonix-Ordner:

- Miscellaneous,
- Instruments,
- Scores und
- Sonix.

Das Ganze geht ebenfalls am besten über die Workbench, weil so die »info«-Dateien und damit die Symbole (Icons) der Schubladen mitkopiert werden.

Als nächstes müssen Sie den Sonix-Loader im Sonix-Ordner übers CLI (!) umbenennen:

```
rename dh0:Sound/Sonix/Sonix DH0:Sound/Sonix/SonixLoader
```

Schreiben Sie anschließend die folgende Batch-Datei und speichern Sie unter dem Namen »DH0:Sound/Sonix/Sonix«:

```
assign remove df1:
assign df1: :sound/sonix
:sound/sonix/sonixloader
assign remove df1:
mount df1:
```

Im C-Directory Ihrer Festplatte muß sich das Hilfsprogramm IconX befinden, das ab Betriebssystem 1.3 mit ausgeliefert wird. Wenn Sie alle Schritte erledigt haben, aktivieren Sie das Sonix-Icon und wählen Sie aus dem Workbench-Menü INFO aus: Die Info-Eingabemaske erscheint. Dort geben Sie bei »Default Tool«

```
c:IconX
```

ein und speichern die Einstellung mit »Save«.

Schauen Sie mit

```
type dh0:devs/mountlist
```

nach, ob dort der Eintrag DF1: vorhanden ist, wir müssen nämlich bei der Benutzung von Sonix vorübergehend DF1: suspendieren und umbenennen, um dem Amiga vorzugaukeln, daß jetzt der Sonix-Ordner DF1: ist. Wenn Sie Sonix beenden, muß das Laufwerk natürlich wieder arbeiten; dazu benutzen wir in unserer BatchDatei am Ende:

```
mount df1:
```

Der Befehl bindet das Laufwerk wieder ordnungsgemäß. Hier der Eintrag, der in der »mountlist« stehen sollte:

```
DF1:      Device   = trackdisk.device
          Unit     = 1
          Flags    = 1
          Surfaces = 2
          BlocksPerTrack = 11
          Reserved = 2
          Interleave = 0
          LowCyl = 0 ; HighCyl = 79
          Buffers  = 20
          BufMemType = 3
#
```

Sonix auf Platte

Diesen Eintrag in der »mountlist« brauchen Sie, um das Laufwerk DF1: wieder einzubinden

3) Der Großteil der Arbeiten ist erledigt, jetzt kommt der Patch, der am File »SonixLoader« vorgenommen wird. Dazu benutzt man einen Filezapper, z.B. »Newzap« oder die neue Version »Filemaster1.2«. Man kann auch das File wieder auf eine Diskette packen und dort mit einem Diskmonitor bearbeiten.

Daß die Datei nur 1572 Byte groß ist, erleichtert uns die Arbeit. Wir suchen in dem Binärfile folgenden Eintrag:

```
Sonix.task.Miscellaneous/Sonix.code
```

und ersetzen ihn ab »Miscellaneous« durch:

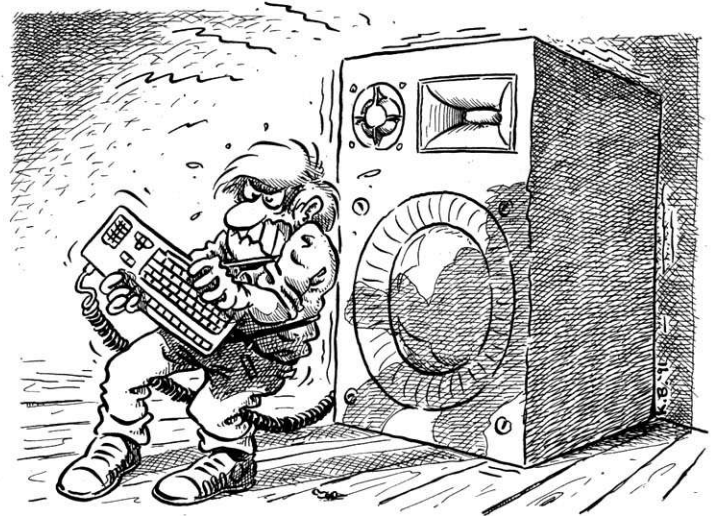
```
DH0:Sound/Sonix/Miscellaneous/Sonix.code
```

Dabei ist darauf zu achten, daß der Punkt zwischen »Sonix.Code« auf der Hexadezimal-Seite unbedingt als »2E« erscheint. Schreiben Sie die gepatchte File zurück und begeben Sie sich zur Workbench. Starten Sie Sonix durch Anklicken. Wenn Sonix erfolgreich geladen wurde, wählen Sie »Instruments« aus dem Menü und versuchen, eins der Instrumente zu laden, indem Sie »DF1:« anklicken. Sonix sollte sich dabei nicht im Laufwerk befinden. Wenn Sie fertig sind, beenden Sie das Programm und geben im CLI ein:

```
dir df1:
```

Falls jetzt nicht der Inhalt der Sonix-Schublade auffährt, ist die Installation gelungen.

Andreas Schulz/ub



Sicherung

Die Spannungsversorgung externer Geräte kann beim Amiga 2000 auch über die Portstecker erfolgen. Bei neueren Boards sind diese Ausgänge mit US-Picofuse-Sicherungen 4 A unter der Kennziffer »F2« bis »F5« abgesichert. (Im Schaltplan des Benutzerhandbuchs nicht eingezeichnet). Wer sich wundert, daß seine externe Floppy nicht läuft oder ein RGB-Splitter über den RGB-Port nicht mit Spannung versorgt werden kann, hat wahrscheinlich die Sicherung irgendwann »gekillt«; erfahrungsgemäß brennen die Sicherungen teilweise bei Spitzenwerten (Einschaltstrom) weit unter dem Nennwert durch.

Original-Sicherungen sind weder über Commodore noch über den einschlägigen Fachhandel erhältlich. Ersatz ist dennoch möglich: mit lötbaren Wickmann-Kleinstsicherungen Typ »TR5-M«. Es ist klar, daß diese Arbeiten nur von Fachwerkstätten oder versierten Usern mit Lötbegabung ausgeführt werden sollten.

Gerhard Schulze/ub

Umstieg leichtgemacht

Viele Freaks sind vom C64 auf den Amiga umgestiegen. Die meisten wollen ihren C64 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/ub

Aktion: »Saubere Maus«

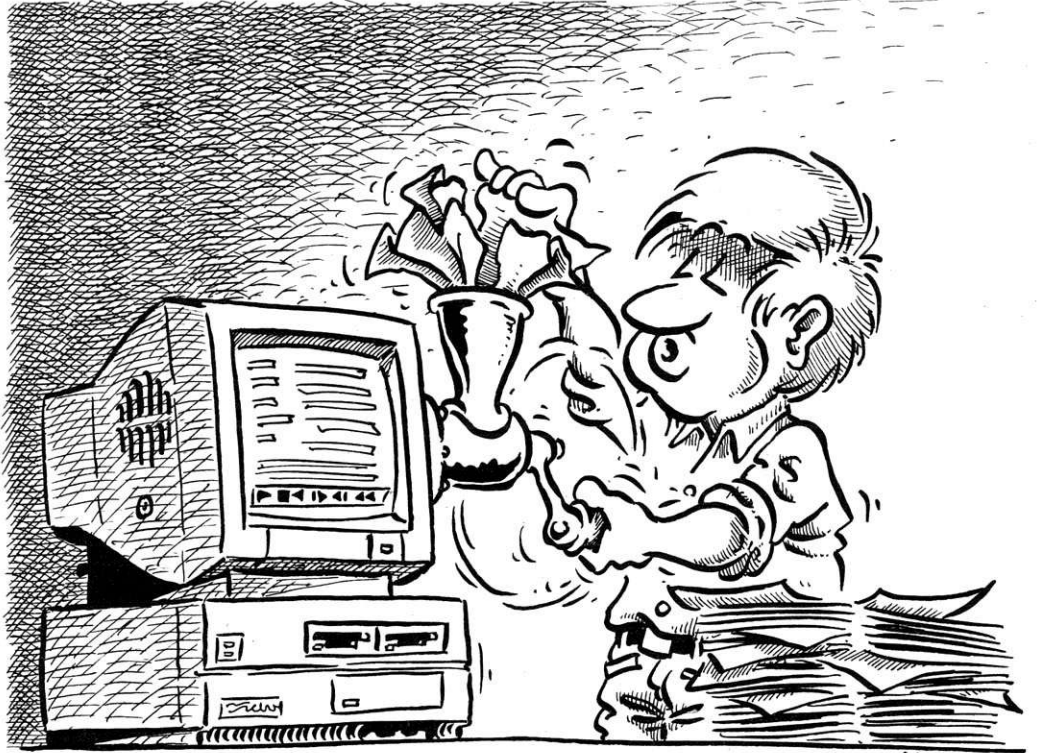
Irgendwann stellt jeder Amiga-Besitzer fest, daß die Maus ihren Dienst nicht mehr richtig versieht. Es treten Schwierigkeiten beim Positionieren auf. Der Grund liegt in der Verschmutzung der »Mäuse-Innereien«. Öffnen Sie den Schiebeverschluss an der Unterseite der Maus öffnen und die Kugel herausnehmen. Wenn Sie in die Öffnung schauen, sehen Sie bereits die Rollen, die zur Wegerfassung dienen. Der Dreck an diesen Rollen behindert die Funktion der Maus. Sie können die Rollenflächen durch vorsichtiges Abwischen mit einem Wattestäbchen reinigen. Danach legen Sie die Kugel zurück, schließen die Maus und schon ist die Maus wie neu.

Ralf Schmid/ub

Softwarehilfe

In der Praxis

Klar, man muß kein Programmierer oder Computerspezialist sein, um seinen Computer richtig einsetzen zu können; es reicht, wenn man die richtigen Anwendungsprogramme hat: Das geht von Textverarbeitungen, über Mal- und Animationsprogramme bis zu Dateiverwaltungen. Und für alle gibt's ein paar gute Ideen, wie man sie noch besser nutzen kann, z.B. wie man Speicherplatz optimal nutzt oder wie man Tipparbeit spart.



K.B.'91

Was machen Sie mit Ihrem Computer? Die meistgenannte Antwort auf diese Frage lautet immer noch: »Textverarbeitung«. Auf dem Amiga hört man aber immer mehr »Malen« oder »Animation«. Der Amiga ist halt ein Computer für Kreative. Damit Sie Ihre Kreativität voll ausschöpfen können, hier ein paar richtige Anwendungstipps.

TeX

TeX ist ein hervorragendes Satzprogramm, um auf dem Amiga in anständiger Qualität Texte zu erstellen, und das beste Werkzeug, um mathematische Formeln aufs Papier zu bannen. Aber gerade dem Anfänger fällt es schwer, durch den Wust an Befehlen und Funktionen zu dringen.

So ist man zunächst stolz, das erste Summenzeichen auf's Papier gezaubert zu haben, stellt dann aber meistens fest, daß irgendetwas nicht so recht stimmt. Mit

```
\sum_{i = 0}^{\infty}
```

erhält man nämlich ein Summenzeichen, bei dem die Grenzen neben dem Summenzeichen stehen.

Nun hätte man aber wahrscheinlich gerne die Grenze über und unter dem Summenzeichen plaziert und nicht rechts daneben. Mit

```
\sum\limits_{i = 0}^{\infty}
```

setzt TeX wie gewünscht die Grenzen über und unter das Summenzeichen.

Aber das ist doch recht viel Tipparbeit, da müßte man doch etwas vereinfachen können?

Hier hilft einem die Möglichkeit, eigene Befehle zu formulieren. Mit

```
\def\Sum {\sum\limits}
```

definiert man den Befehl »\Sum«, der wie folgt benutzt wird:

```
\Sum_{i = 0}^{\infty}
```

Auf die gleiche Art kann man den Befehl »\Lim« einführen, der das Limes-Symbol passend auf's Papier bringt. Allgemein eignet sich der »\limits«-Befehl zum direkten Unter- oder Übersetzen von Texten. So erhält man dann auch z.B.:

```
\sup\limits_n \ln N
```

Auch andere lange und umständliche Befehlssequenzen kann man auf die beschriebene Art handlich gestalten. Sollte dann der verwendete Editor noch Floskeltasten kennen, hat man eine einfache Möglichkeit, qualitativ hochwertige Text zu setzen, die auch Sonderzeichen enthalten.

ub

Animation mit DPaint

Um eine Animation mit Deluxe Paint III auf einem Amiga mit – nur – 1 MByte Speicher in Hires zu genießen, z.B. den Globus aus Ausgabe 7/91 des AMIGA-Magazins, muß man folgendes machen:

Nehmen Sie zunächst einen Animations-Brush auf: <Amiga_links> gedrückt halten und mit der Taste und der Maus den Pinsel aufnehmen. Anschließend wechseln Sie in den Hires-Modus und können dort mit den Tasten <7> und <8> die Animation in Fernsehqualität betrachten.

Harald Mohring/ub

Banner-Druck mit Notepad

Banner-Bänder sehen schön aus und man kann sie besonders zur Dekoration bei festlichen Anlässen benutzen. Und mit dem Amiga Banner zu drucken, ist relativ einfach:

Als erstes braucht man das »Notepad«, um Texte zu schreiben und um sie mit verschiedenen Zeichensätzen darzustellen.

Das zweite wären die »Preferences« (Version 1.3), um den Ausdruck von Grafiken zu manipulieren.

Der Weg vom Notepad-Text zum ausgedruckten Banner sieht dann so aus:

Man ändere in den Preferences die Einstellungen so:

PREFERECE-EINSTELLUNG

Paper Size:	U.S. Legal
Graphic1:	
Aspect:	Vertical
Graphic2:	
Scaling:	Integer
Limits:	Multiply

Als Font können Sie jeden beliebigen Font wählen. Große Fonts eignen sich am besten, z.B. Basel 45 auf der AMOS-Demo-Diskette.

Die Einstellungen bei Limits hängen vom verwendeten Font ab. Die folgende Tabelle zeigt einige Einstellungen. Sie können sie beliebig erweitern, indem Sie mit den Limit-Parametern und anderen Zeichensätzen experimentieren.

ZEICHENSÄTZE

Font	Width Limit	Height Limit
Basel 45	7	26
diamond 20	12	132
sapphire 19	12	132
rubby 15	12	132
topaz 11	12 (24)	132

Wie man in der Tabelle sieht, erweist sich die Einstellung »12..132« als besonders praktikabel. Probieren Sie Sie ruhig bei anderen Zeichensätzen aus und ändern die Werte so, wie sie es für nötig halten. Für die Druckdichte kann man Density 1 wählen. Möchten Sie aber, daß der Druck intensiver wirkt, erhöhen Sie Density.

Sie werden feststellen, daß die Druckhöhe teilweise von der Höhe des Notepad-Windows abhängt. Da der Amiga den Text nämlich um 90 Grad dreht, werden die Leer-Zeilen unter dem Notepad-Text mitvergrößert. Das hat zur Folge, daß der gedruckte Text viel kleiner wird. Sie können das Problem abschaffen, indem Sie die untere Fensterleiste so nah wie möglich an den Text bringen; dann dürfte Ihrem Banner-Projekt nichts mehr im Wege stehen.

Versuchen Sie auch, mit den Drucker-Parametern in den Preferences zu spielen – sicher finden Sie so manche Einstellung, die Ihnen ein sehenswertes Ergebnis liefert.

Felix Farago/ub

Feinschliff für Bilder

Wer mit Sculpt/Animate-3D oder -4D arbeitet, wird sich über die häufige relativ grobkörnige Bilddarstellung geärgert haben. Es gibt jedoch eine Methode, die Bildqualität zu verbessern. Voraussetzung hierfür sind mindestens 2 MByte (besser 4 MByte) RAM, eine Festplatte und das Programm »The Art Department« (TAD) von ASDG.

Vor der Berechnung mit Sculpt stellen Sie die Auflösung im Observer/Mode-Requester auf Hires-Interlace (640x512) ein. Auch Overscan ist möglich. Die Farbtiefe wird auf 24 Bit gesetzt.

Der Amiga berechnet jetzt das Bild mit 16 Millionen Farben und speichert es automatisch. Man kann es anschließend mit »The Art Department« laden und in HAM, AHAM oder sogar in Dynamic Hires umrechnen lassen, wobei die Ergebnisse in HAM und Hires nicht immer zufriedenstellend sind. Da das AHAM-Format nicht von allen Programmen verarbeitet wird, ist es am besten, das Bild in diesem Format mit Digi-View zu laden und in HAM+ zu speichern.

Volker Umland/ub

There is no problem too big that can't be solved with high explosives.

Deluxe Video III: Falsche Fische

Beim Start einer Deluxe-Video-III-Animation sucht und lädt der Amiga alle zugehörigen Teile wie Brushes etc. in den Speicher. Findet er eines der Teile nicht, erscheint eine Fehlermeldung. Auf der Beispieldiskette »DV Beispiele2« befindet sich die Animation »Relative Bewegung«. Genau bei ihr tritt die Fehlermeldung auf.

Folgende Brushes können nicht geladen werden:

- Fish.Brush
- FishSchool.Brush
- LostFish.Brush
- Schauen Sie im Verzeichnis »Brushes« nach – dort finden Sie:
- Fisch.Brush
- FischSchool.Brush
- VerirrterFisch.Brush

Aha! Der Hersteller hat es gut gemeint und die Namen eingedeutscht. Ändern Sie die Schreibweise der Namen in genau die Form, wie sie das Programm in der Fehlermeldung ausgibt. Da die Brushes alle ein Icon haben, brauchen Sie nur in der »Brushes«-Schublade nachzusehen und die Änderung mit dem »Rename«-Befehl über die Workbench vorzunehmen.

ub

MACHEN SIE MIT

Wollen Sie sich an der Rubrik »Tips & Tricks« im AMIGA-Magazin beteiligen? Schicken Sie uns Ihre Tips. Am besten, Sie senden uns Ihre Programme und Texte auf Diskette (ASCII-Format). Für Bastelanleitungen fügen Sie bitte einen Bauplan bei. Die Adresse:

AMIGA-Redaktion

Kennwort: Tips & Tricks

Markt & Technik Verlag AG

Hans-Pinsel-Str.2 8013

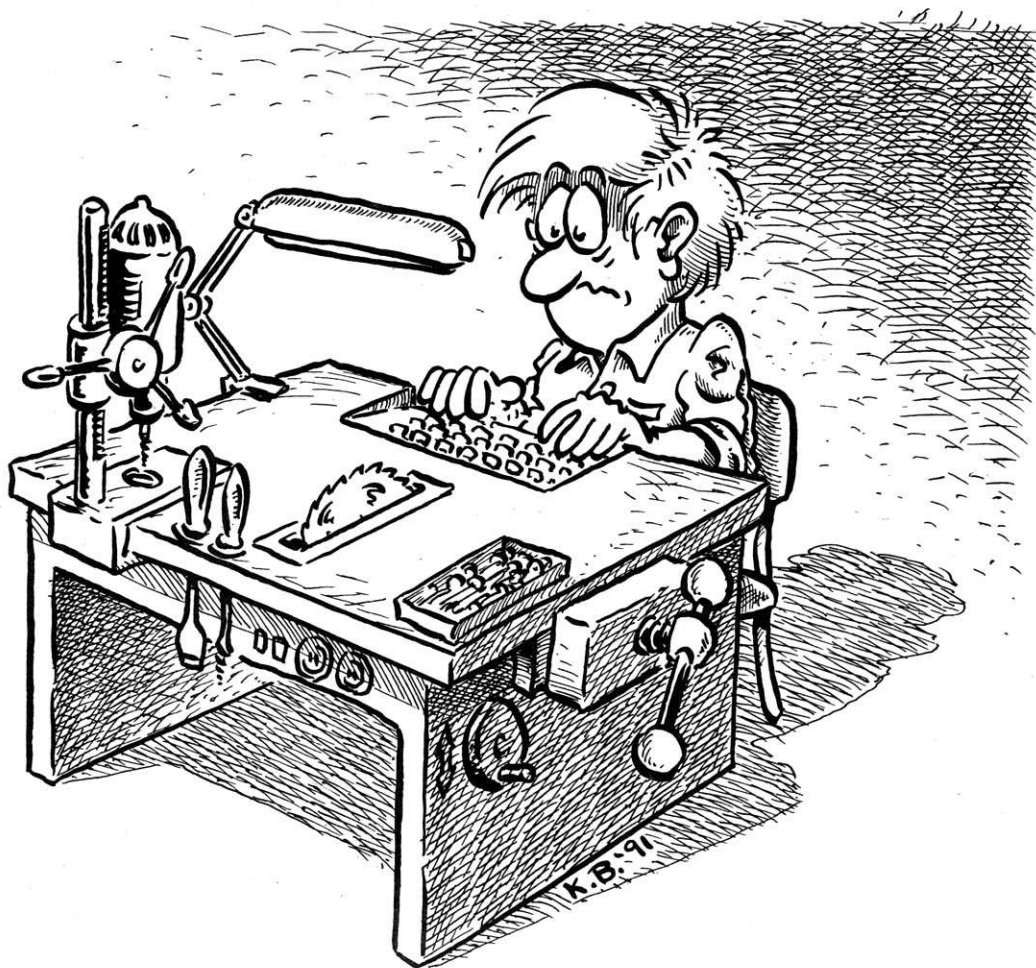
Haar bei München

Alle Einsendungen werden von uns geprüft. Wenn Ihr Beitrag in die engere Wahl für eine Veröffentlichung kommt, werden Sie benachrichtigt. Haben Sie auf jeden Fall ein wenig Geduld: Vom Zeitpunkt, an dem Ihr Beitrag die Redaktion erreicht, können bis zum Abdruck im AMIGA-Magazin oder Sonderheft zwei bis drei Monate vergehen.

Neues Betriebssystem

3000er und Workbench 2.0

Seit Anfang 1991 kann man den Amiga 3000 erwerben. Und seit kurzem sind auch endlich der Amiga 500 Plus und das neue Betriebssystem 2.0 fertig und beim Commodore-Fachhändler erhältlich. Sogar ein sog. Upgrade-Kit zum Aufrüsten des Amiga 500 bzw. des Amiga 2000 gibt's, d.h. OS 2.0 dürfte sich rasend schnell verbreiten, denn seine Vorteile gegenüber dem alten System sind enorm. Und natürlich haben wir schon ein paar Tricks für Sie auf Lager.



So wie unserem Freund Hermann soll es Ihnen mit dem neuen Betriebssystem, d.h. vor allem der neuen Workbench natürlich nicht gehen. Deshalb hier die ersten Tips zum Amiga 3000, zur Workbench 2.0 und vor allem zur Frage, welche Programme laufen auch mit den neuen Amigas – bzw. wie kriegt man Verweigerer doch noch zum Laufen?

Productivity vorgaukeln

Der Productivity-Modus ist eine neue Betriebsart des Amiga. Er bietet eine flimmerfreie (ohne Interlace-Flimmern) Auflösung von 1280 Punkten in der Horizontalen und 512 Punkten in der Vertikalen.

Das Alles allerdings nur mit vier Farben. Wichtig: Die Ausgabe auf den Monitor erfolgt in diesem Modus mit einer höheren Frequenz, was ein ruhigeres und schärferes Bild zur Folge hat; man benötigt allerdings einen Multiscan-Monitor, der etwas kostspieliger ist als herkömmliche Modelle.

Nur wenige Programme unterstützen derzeit den neuen Modus, doch kann man sich seine Vorteile mit Programmen zunutze machen, die ansonsten im Interlace-Modus arbeiten. Dazu sollten Sie den Workbench-Screen im Productivity-Modus laufen lassen. Nach dieser Einstellung (natürlich nur mit Multiscan-Monitor) müssen Sie zusätzlich Overscan einstellen. Das erhöht die Zeilenzahl von 640 *

480 auf 640 * 495, was wichtig ist, um den Zeilenunterschied zu 512 (Interlace) etwas zu verringern.

Nun starten Sie ein Programm, das einen Interlace-Bildschirm öffnet (z.B. PageStream, Beckertext 2.0, DynaCAD, IntroCAD, etc.). Wenn Sie diesen Screen nach unten ziehen, wird der obere Rand des Workbench-Screens ebenfalls flimmern. (Fehler im Flickerfixer des A3000). Der Productivity- und der Interlace-Modus können nicht gleichzeitig dargestellt werden, wie es mit den Standard-Modi möglich ist. Das nutzen wir aus:

Nehmen Sie den WB-Screen nach vorne und schieben ihn nach unten, bis er fast unsichtbar ist. Jetzt können Sie mit Ihrem Programm ganz normal weiterarbeiten – allerdings im Productivity-Modus. Nachteil: Der untere Rand ist nicht sichtbar (17 Zeilen fehlen). Trotzdem ist die erhöhte Frequenz angenehm spürbar.

ub

AMIGA 3000 und die Software

Nicht jede Software, die für den Amiga 500/2000 geschrieben wurde, läuft auf dem Amiga 3000 einwandfrei, u.a. AmigaBASIC und GFA-BASIC 3.5 (mittlerweile gibt es eine Spezialversion für OS 2.0 und den Amiga 3000: V 3.52.)

Folgende Programme arbeiten zuverlässig:

- MultiTerm (Pro 2.14 BTX Programm),
- KindWords (Textverarbeitung),
- XCopy professional (Kopierprogramm).

Falls Sie selbst Erfahrungen mit Software für den Amiga 3000 gemacht haben, schreiben Sie uns, welche Software läuft und wo es evtl. hakt. Wir werden im AMIGA-Magazin und weiteren Tips & Tricks-Sonderheften ständig über diesen Thema informieren und auch ggf. Lösungen anbieten, um ein Programm am Amiga 3000 lauffähig zu machen.

ub

MultiTerm und der 3000er

Es gibt viele, die Btx betreiben und nun auf einem AMIGA 3000 umgestiegen sind. Und häufig gibt es Probleme bei der Benutzung von MultiTerm. Die Schwierigkeiten liegen daran, daß man die Funktionstastenbelegungen nicht mehr benutzen kann.

Um das zu ändern, muß man nur »ihelp« abschalten. Dazu startet man das Programm »Exchange« und klickt »ihelp« an. Jetzt muß man nur noch auf »disable« drücken und schon klappt alles wieder wie gewohnt. Nach Beendigung von MultiTerm, sollte man in »Exchange« »ihelp« wieder mit »enable« aktivieren.

Christian Benner/ub

Startup-Sequence auf dem 3000er

Es ist beim 3000er nicht nötig, ständig die »Startup-Sequence« zu ändern, wenn man nur ein zusätzliches Programm darin aufrufen möchte. Man kopiert einfach auf der Workbench das Symbol (Icon) des Programms in die Schublade »WBSStartup«. Beim nächsten Systemstart startet der Amiga 3000 das Programm automatisch.

ub

3000er und AmigaBASIC 1.2

Wenn man beim Amiga 3000 das Fast-Memory mit dem Programm »NoFastMem« in der System-Schublade ausschaltet läuft AmigaBASIC problemlos.

Man sollte jedoch kein laufendes BASIC-Programm, welches Systemroutinen benutzt, abbrechen und editieren, da sich der Amiga sonst mit einem System-Request meldet.

Man kann zwar mit einem Mausklick auf »Suspend« einen Guru verhindern, aber das Programm bleibt im Hintergrund stehen und belegt weiterhin wertvollen Speicher, d.h. der vom Programm reservierte Speicher kann nicht genutzt werden.

Erfahrene Benutzer können das Fenster mit dem Programm »XOper« von der Fish-Disk 318 schließen.

Michael Christoph/ub

Neuerungen von OS 2.0

Möchten Sie Screens horizontal mit der Maus verschieben? Was vorher nur von oben nach unten (und umgekehrt) möglich war, funktioniert nun auch von links nach rechts bzw. rechts nach links. Um das zu sehen, müssen Sie nur »AmigaVision« laden (das ja jetzt mit dem A3000 ausgeliefert wird) und Overscan als Default-Auflösung einstellen. Wenn Sie die Ziehleiste des Screens mit der Maus fassen, können Sie ihn wahrlich in alle Richtungen bewegen.

Übrigens ist das automatische Scrollen des Workbench-Screens nicht nur mit übergroßem Screens möglich: Sobald Sie einen Screen mit der Maus nach unten ziehen, die Maustaste loslassen und die Maus weiter nach unten ziehen (über den Bildschirmrand hinaus) scollt der Bildschirm wie von Geisterhand sanft nach oben.

ub

One magician is two too many

Terminalprogrammersatz

Sie wollen Daten von einem Computer an den Amiga 3000 übertragen? Normalerweise schickt man die Daten per Terminalprogramm von einem System zum anderen.

AmigaDOS läßt sich aber auch als Terminalprogramm mißbrauchen. Das Programm auf dem Zweitcomputer (bei Laptops ist häufig im Lieferumfang) ist natürlich nach wie vor erforderlich.

Geben Sie

TYPE <SER: TO RAM:Dateiname

ein, wird die Dateneingabe von Tastatur auf die serielle Schnittstelle umgeleitet; und alles, was über die serielle Schnittstelle hereinkommt, wird in »RAM:Dateiname« geschrieben.

ALLES AMIGA 3000

Tip	Thema	Seite
Productivity vorgaukeln / Neuer Screen-Modus		76
Amiga 3000 und die Software/ Kompatibilität		77
MultiTerm und der 3000 er / So läuft's		77
»Startup-Sequence« / WorkbenchStartup leichtgemacht		77
AmigaBASIC 1.2 / Läuft nur mit Trick 17		77
Neuerungen / Screens horizontal verschieben		77
Terminalprogrammersatz / Datenübertragung		77
RENAME unter OS 2.0 / Befehlerweiterung		78
? ALIAS / Nützliche ALIAS-Befehle		78
Resettfestes / Reset mit dem neuen Betriebssystem		78
Alles unter IControl / <Ctrl>-Funktionen		78
»PCD« verbessert für 2.0 / Verbesserung des 1.3-Skripts		78
Kompatibilität mit OS 2.0 / Was läuft?		79

Nun können Sie Ihre Daten losschicken. Wenn die Übertragung beendet ist, müssen Sie mit einem Stopcode die Verbindung von DOS mit der seriellen Schnittstelle abbrechen, im Terminal heißt diese Funktion »Verbindung abbrechen« oder ähnlich. Auch ist es möglich, in der Terminalemulation <Ctrl c> zu drücken. Danach gibt Amiga-DOS eine Fehlermeldung, die Sie nicht zu berücksichtigen brauchen. Danach können Sie die Daten mit einem Textverarbeitungs- oder DTP-Programm laden und gestalten.

/ub

RENAME unter OS 2.0

Endlich versteht AmigaDOS auch mit RENAME Pattern-Matching: Mehrere Dateien werden entsprechend den Angaben umbenannt bzw. verschoben. Ein kleines Beispiel:

Sie befinden sich im Textverzeichnis Ihrer Textverarbeitungsschublade, z.B. »Work:BTII/Text«. Gerade haben Sie ein neues Verzeichnis »Work:BTII/ASC« angelegt. Um nun alle ASCII-Dateien mit Endung ».ASC« ins neue Verzeichnis zu kopieren, genügt mit AmigaDOS 2.0:

```
rename #?.ASC AS /ASC
```

Der Befehl RENAME wird hier also als MOVE-Befehl für mehrere Dateien eingesetzt.

ub

? Alias ?

Ja, auch »?« kann man unter der ARP-Shell oder der Shell von Workbench 2.0 als ALIAS verwenden, aber man muß bei der Definition Anführungszeichen verwenden. Beispiel:

```
alias "..." cd / ;(eine Verzeichnisebene zurückgehen)
```

Weitere nützliche ALIAS-Befehle sind z.B.:

```
alias ":" cd : ; zur Verzeichnis-Wurzel
```

```
alias "." dir ; aktuelles Verzeichnis zeigen,  
; bzw. Abkürzung für DIR
```

```
alias "?" echo "Sein oder nicht sein, das ist hier die Frage"
```

Das letzte Beispiel nach dem Motto: »Literatur hilft immer!«.

David Spaak/ub

Resetfestes

Gar manches macht sich im Amiga fest; so fest, daß es selbst nach einem Reset noch fest ist - eben resetfest, z.B.:

- die Testversionen der Kickstart 2.0,
- die residente RAM-Disk,
- diverse Handler und
- Viren.

Manchmal stört diese Hartnäckigkeit. Denn gerade um letztere »Tierchen« wieder aus dem Amiga herauszubugsieren, scheinen mitunter nur noch recht unorthodoxe Methoden angebracht (Rechner ausschalten).

Dabei kann man auch das Betriebssystem überreden, die resetfesten Module aus dem Speicher zu fegen. Wie das geht, zeigen folgende Beispiele:

Wer sich erst nach dem nächsten Reset aller resetfesten Strukturen entledigen und auch die Vektoren Cold-, Cool- und Warm-Capture löschen möchte, braucht nur ein kurzes Programm zu starten:

```
move.l 4.w,a6 ; Adresse Exeabase  
move.l a6,$26(a6) ; ChkBase zerstören  
moveq #0,d0  
rts ; zurück ins CLI
```

Das System bleibt nach Aufruf dieses Programms intakt. Erst beim nächsten Reset wird alles wieder in den Nullzustand gebracht.

Wer nicht bis zum nächsten Reset warten möchte, für den kann auch das Programm selbst den Reset auslösen. Das sähe dann z.B. so aus:

```
move.l 4.w,a6 ; Adresse Exeabase  
move.l a6,$26(a6) ; ChkBase zerstören
```

```
jsr -150(a6) ; in Supervisor springen  
jmp $fc0002 ; Reset-Routine starten
```

Das Programm verursacht also einen sofortigen Reset (nicht zu verwechseln mit unkontrolliertem Absturz).

Auf einem Amiga 3000 oder überhaupt unter Kickstart 2.0 sollte man für den Reset besser gleich die Betriebssystem-Routine »ColdReboot« verwenden. Das geht dann folgendermaßen:

```
move.l 4.w,a6 ; Adresse Exeabase  
move.l a6,$26(a6) ; ChkBase zerstören  
jmp -726(a6) ; ColdReboot starten
```

Noch kürzer geht's mit einem einzigen Befehl, der einfach die Adresse der Exeabase ungerade macht, z.B.:

```
st 7.w ; entspricht move.b #$ff,7
```

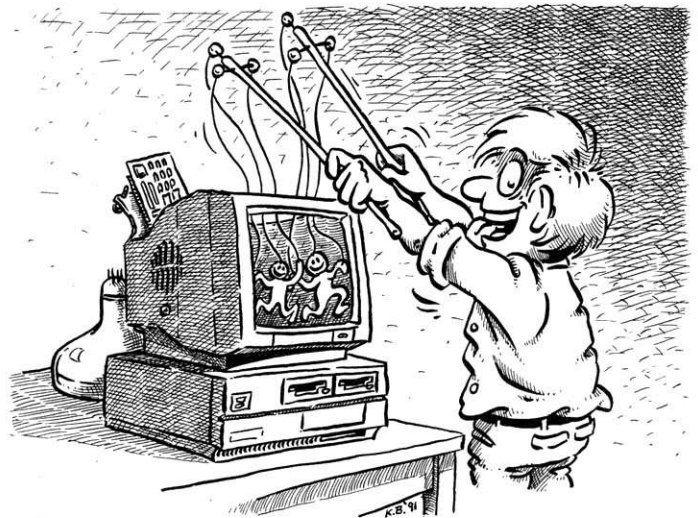
oder etwa in BASIC:

```
poke 7,1
```

Führt man diesen Befehl im Multitasking-Betrieb aus, stürzt der Rechner innerhalb der nächsten 50stel Sekunde mit einem Adreß-Fehler ab. Deshalb bekommt man leider auch einen Guru. Danach ist das System aber ebenso sauber wie bei den obigen Routinen.

All diese Routinen funktionieren mit den normalen Kickstart-Versionen. Wer mit gepatchten Exoten, z.B. der UFO-Kickstart, arbeitet, wird vielleicht nicht immer zum gewünschten Ergebnis kommen.

Georg Sonnenberg



Alles unter IControl

Deluxe Paint III bietet mit dem »Anim«-Painting eine einfache, aber leistungsfähige Funktion zum »Belebten Malen«. Sie wird mit der Taste <Amiga_links> aktiviert. Bei Verwendung der Workbench 2.0 stößt dies jedoch auf Probleme, da die Taste standardmäßig für die unter Workbench 2.0 neue Funktion zum Herunterziehen eines Screens mit der Maus vorgesehen ist.

Abhilfe schafft hier eine Änderung im Preferences-Programm »IControl«: Man wählt einfach eine andere Taste unter der Überschrift »Mouse Screen Drag«. Man kann auch mehrere Tasten auswählen, wobei sich eine Kombination aus der standardmäßig eingestellten Taste <Amiga_links> und <Alt> anbietet, da beide auf der Tastatur nebeneinander liegen und damit leicht zu merken sind.

Sven Kreutz/ub

»PCD« verbessert für 2.0

Auf der Workbench-Diskette Version 1.3 findet man im »s«-Verzeichnis das Skript »PCD«, das wie CD verwendet wird. Das Besondere ist, daß sich »PCD« das jeweils letzte Verzeichnis merkt. Um dorthin zurückzukehren, genügt es, »PCD« ohne eine Verzeichnisangabe aufzurufen.

Die Version von Commodore hat den Nachteil, daß sie mit dem AS-SIGN-Befehl arbeitet. Das hat zur Folge, daß nach einem »PCD«-Aufruf auf eine Diskette das Diskettensymbol (Icon) auf der Workbench stehenbleibt, selbst wenn die Diskette schon längst nicht mehr im Laufwerk steckt, weil eben noch der ASSIGN-Befehl auf die Diskette wirksam ist.

Hier nun eine Version von »PCD.bat«, die mit den Shell-Variablen der Workbench 2.0 arbeitet. Wir nutzen hier eine Besonderheit der Shell: Steht ein Befehl in Rückwärtsapostrophen »`, wird er zuerst ausgeführt und die Ausgabe, die der Befehl produziert, an seiner Stelle in der Kommandozeile eingesetzt. Also bewirkt z.B. der Befehl

```
echo "Today we have `date`!"
```

daß letztendlich etwa die Befehlszeile

```
echo "Today we have Sunday 26-May-91 02:36:29!"
```

ausgeführt wird, mit dem Ergebnis:

```
Today we have Sunday 26-May-91 02:36:29!
```

Das nutzt »PCD2.0.bat«, um das aktuelle Verzeichnis zu ermitteln: {\$\$} wird von der Shell ersetzt durch die aktuelle CLI-Nummer.

David Spaack/ub

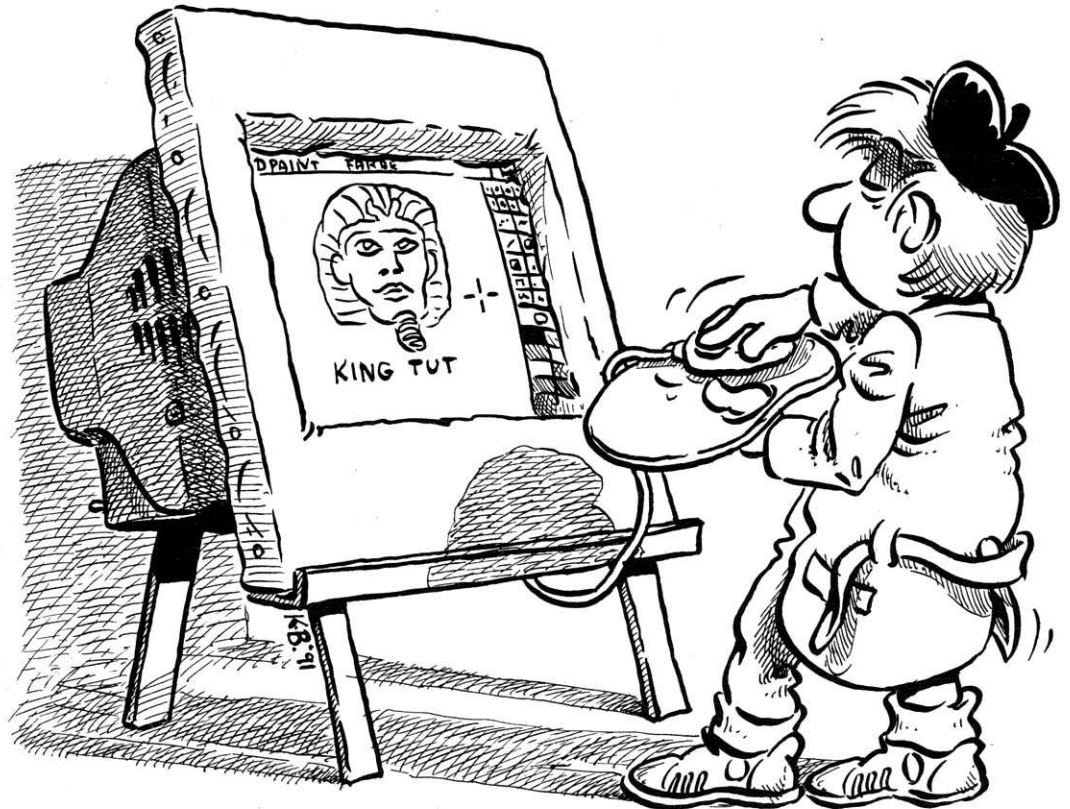
```
.key dir
.bra {
.ket }
if "{dir}" EQ ""

set ofrom($$) = "`cd`"
cd $from($$)
set from($$) = $ofrom($$)
unset ofrom($$)

else
set from($$) = "`cd`"
cd {dir}
endif
```

PCD2.0.bat

Verbesserte Version der Batch-Datei von der Extras-Diskette 1.3 für die Workbench 2.0



Kompatibilität mit OS 2.0

Welche Programme laufen unter OS 2.0 auf einem Amiga 3000? Hier noch eine Übersicht bekannter Titel:

KOMPATIBILITÄT

Programm

OS2.0-kompatibel

IntroCAD	ja
PageStream	ja
Broadcast Titler 2	nein
Deluxe Paint III	ja
PhotonPaint 2	ja
DigiPaint	ja
Sonix	nein
DeluxeMusic	ja
Audiomaster III	ja
Noisetracker 2	nein
Startrekker 1.3 (PD, exelenter Soundeditor)	ja
Devpac	ja
Defender of the Crown	ja!
Jinxter	nein
Out Run	ja
	mit Problemen, aber schneller!
Buggy Boy	ja!
Rock'n'Roll	nein
Great Giana Sisters	ja
Starglider II	ja
AmigaBASIC	nein!!!
	findet Programme nicht mehr nur mit Tricks nutzbar
Amiga-Oberon	ja

Amiga-Oberon ist ein besonders schönes Beispiel für Kompatibilität zum neuen Betriebssystem: Der Editor unterstützt voll die neuen Features der Workbench; in den Menüs tauchen die neuen Fonts auf, dasselbe gilt für die Auflösungen, sogar der neuen File-Requester wird benutzt, nachdem Oberon den der »arp.library« nicht erreichen konnte.

ub

**2.0 is not the
answer
2.0 is the question
" Yes "
is the answer**

(Andy Finkel)

Beckertext & Co**Schreiberlinge**

Die gute, alte Schreibmaschine hat schon lange ausgedient: Den Todesstoß versetzt hat ihr die Textverarbeitung mit dem Computer, die verbreitetste Computeranwendung überhaupt. Enorm, was man da alles machen kann. . . Nutzen Sie Ihr Programm überhaupt voll aus?

Selbst die gute, alte Textverarbeitung muß aufpassen; Desktop Publishing wird immer populärer; die Programme immer besser und preiswerter. Eine Folge ist, daß Textverarbeitungsprogramme immer mehr Funktionen bekommen, z.B. auch Grafik einbinden können, allerdings auch immer komplizierter – wöbder Übergang vom reinen Textprogramm zum DTP-Programm zerfließt: Eine Folge der Komplexität: Nur wer sein Programm kennt, erhält zufriedenstellende Ergebnisse.

Beckertext-2.0-Accelerator

Falls Sie bei Beckertext die größte Verarbeitungsgeschwindigkeit wünschen, sollten Sie die Bildschirmauflösung 640 x 256 einstellen, eventuell mit nur einer Bitplane (nur zwei Farben). Die Geschwindigkeit ist dann ungemein höher als etwa auf dem Workbench-Screen im Productivity-Modus.

ub

Texte formatieren

Texte, die von einem anderen Computer an den Amiga übertragen werden, liegen meist in Rohfassung vor, als reine ASCII-Texte: ohne Formatierungen, Schriftänderungen etc. Hier eine Methode, die anschließende Formatierung auf dem Amiga zu erleichtern:

Beim Eingeben der Texte auf dem anderen System schreiben Sie vor jedem Wort, das später z.B. unterstrichen werden soll, das Symbol »_«. Für andere Schriftanweisungen wählen Sie entsprechende Symbole oder Zeichenfolgen, beispielsweise für kursiv »\«.

Die Pseudo-Steuerzeichen können nun problemlos zum Amiga übertragen werden. Sie müssen anschließend nur noch alle »_« suchen und die Wörter unterstreichen. Dazu bietet sich bei Beckertext ein Makro an. Er würde folgendermaßen lauten:

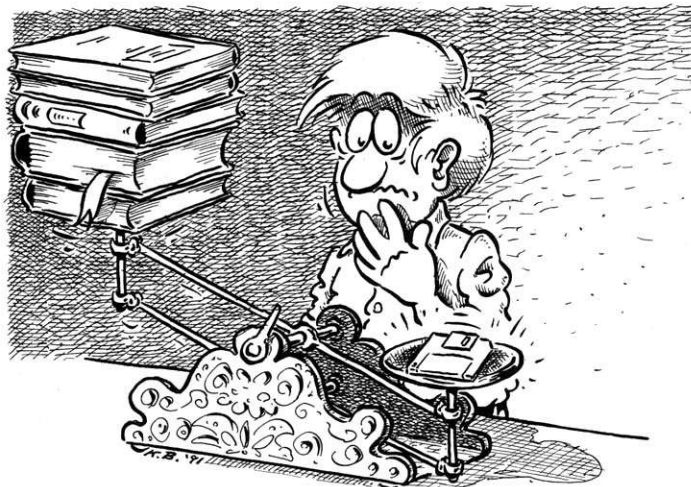
```
\enächste\r\lösche\r\emarkiere wort\r\eunterstr\r
```

Legen Sie dieses Makro auf eine Funktionstaste, z.B. <F1>. Geben Sie (bitte vor jedem Einsatz des Makros) den Suchbegriff (also »_«) im Suchrequester an. Jedesmal, wenn Sie nun <F1> drücken, sucht Beckertext das nächste zu unterstreichende Wort, löscht das »_«, markiert das Wort und unterstreicht es.

ub

Schließen der Workbench

Wer Beckertext 2.0 auf einem eigenen Screen starten will, sollte das nicht mit einem Doppelklick auf's Icon machen. Wird Beckertext über die Workbench gestartet, versucht der Amiga, die Workbench zu schließen. Das gelingt zunächst auch, aber die Requester, die zum Diskettenwechseln auffordern, öffnen den Screen wieder (je-



doch ohne Fenster und Icons). Um das zu verhindern, muß man ein Fenster, das im Zusammenhang mit Beckertext steht, geöffnet halten: Man lädt das Programm einfach übers CLI mit:

```
BT-II-2:BT-II [-i] [-p = n]
```

wobei »-i« den Screen im Interlace-Modus startet und »n« die Anzahl der Bitplanes des Screens vorgibt. Danach muß man das CLI mit ENDCLI und die Workbench mit Beckertext über die Befehlszeile schließen. Der Befehl lautet: »WORKBENCH 0«. Zur Erinnerung: Sie aktivieren die Befehlszeile mit <Esc>.

ub

Diskettenverwaltung

Mit Beckertext 2.0 läßt sich viel anstellen: Sogar zu einer kleinen Datenverwaltung kann man die Textverarbeitung mißbrauchen.

Bilden Sie zuerst mit Tabulatoren mehrere Spalten (nicht zu verwechseln mit dem Spaltendruck im Formatmenü). Legen Sie die Spalten nach Gesichtspunkten der späteren Sortierung an, z.B. für ein Programmverwaltung:

- Nr. (ganz links = Spalte 0),
- Programmart (Spalte 1),
- Name,
- Versionsnummer,
- Kickstart

Jetzt können Sie Ihre Daten Zeile für Zeile eingeben, also z.B.:

```
20, DTP, Pagestream, 2.0, 1.3/2.0.
```

Mit der Sortierfunktion ordnen Sie Ihre Diskettensammlung nun nach Programmart, Nr., Name, Version oder nach der Kickstartversion.

ub

Beckertext und der Speicher

Dieser Trick sollte eigentlich unnötig sein, wenn Beckertext 2.0 optimal programmiert wäre. Die Speicherbenutzung sollte noch mal geprüft werden. Und der Dateiaufbau! Alle Einstellungen, wie Druckertreiber, werden in jedes Dokument geschrieben; große Teile einer Datei bestehen so aus leerem Code, was einen zehneitigen Text mit verschiedenen Formatierungen auf über 120 KByte aufbläht.

Aber nun zum »Trick«: Speichern Sie große Dokumente öfter und brechen anschließend Beckertext II ab; am besten mit Reset, da Beckertext nicht den gesamten Speicher freigibt. Laden Sie den Text anschließend neu. Bis zu 200 KByte Speicher kann man durch diese »Optimierung« zurückgewinnen.

ub

Labelprint mit Pagestream 2

Wollen Sie mit Pagestream 2 Diskettenetiketten bedrucken? Stellen Sie hierzu das Dokumentformat auf die Größe der Etiketten ein. Vergessen Sie nicht, den Abstand zwischen zwei Etiketten mit einzuberechnen. Stellen Sie danach auf die Masterpage um und gestalten Sie den Hintergrund des Labels. Setzen Sie hier auch nach Ihren Wünschen an irgendeiner Stelle einen Platzhalter für die Seitennummern. Zusatztip: Teilen Sie auf, zwischen den Nummern 1 bis 9, 10 bis 99, etc., falls es auf die exakte Ausrichtung ankommt.

Wenn Sie zurück zur ersten Seite springen, sehen Sie Ihr Diskettenlabel für die Diskette Nr.1. Sobald Sie nun beim Drucken einstellen, von welcher bis zu welcher Seite der Amiga drucken soll, gibt Pagestream genau diese Etiketten aus und setzt die Seitennummern (Diskettennummer) richtig ein. Beim Ausdrucken sollten Sie eventuelle Fehler beim Papiertransport durch Veränderung der Papiergröße korrigieren.

ub

Diskettenwechsel...ade

Alle Beckertext-2.0-Besitzer, die nur ein Laufwerk und 1 MByte RAM ihr eigen nennen, müssen sich bei jedem Laden als Diskjockey betätigen. Jede Diskette muß fünfmal (!) ins Laufwerk – das fördert nicht gerade den Gebrauch einer Textverarbeitung.

Das läßt sich jedoch zunächst auf viermal reduzieren, wenn man sich per CLI oder Shell vom Ladebild trennt (Leerzeichen ist wichtig). Erstellen Sie zunächst eine Kopie der Beckertext-Original-Diskette. Alle Änderungen sollten selbstverständlich nur auf den Sicherheitskopien durchgeführt werden. Geben Sie im CLI ein:

```
RENAME "BT-II-2:BT-II.info" BT-II-2:BT-II.info"
```

```
DELETE "BT-II-2:BT-II"
```

Eine weitere Änderung erspart nochmals zwei Diskettenwechsel; man verliert jedoch gleichzeitig ca. 31 KByte RAM: Fügen Sie zwei Zeilen am Anfang der »Startup-Sequence« (Verzeichnis »s«) ein (ggf. hinter dem ADBUFFERS-Befehl):

```
copy libs: RAM:libs quiet
```

```
assign libs: RAM:libs
```

Um den Speicherverlust auszugleichen, ist es günstig, wenn man möglichst nur Formatschablonen verwendet, da sie gegenüber der Direktformatierung weniger Speicher verbrauchen.

ub

Numerische Tastatur

Jeder Texteditor hat eine andere Steuerung. Wenn man mehrere Texteditoren benutzt, verliert man leicht die Übersicht, welche Tastenkombination welche Funktion erfüllt. Wie einfach wäre es, wenn die Tasten der Numerik-Tastatur die Bedeutung hätten, die an den Seiten steht:

- PageUp,
- PageDown,
- Home,
- End, usw

Die Lösung ist einfach: Man sucht sich ein Programm, das die Änderung der Tastaturbelegung (KeyMap) durchführt. Im Handbuch zum jeweiligen Texteditor findet man die entsprechenden Kombinationen, die man dann an Stelle der Numerik-Tasten einsetzt. Die modifizierte Tastaturbelegung speichert man unter einem anderen Namen, z.B. »d_ed«. Vor der Benutzung gibt man in CLI den Befehl

```
setMap d_ed
```

und schon ist die modifizierte Tastaturbelegung aktiviert. Man kann es noch besser machen, indem man eine Script-Datei schreibt und ins »s«-Verzeichnis kopiert. Dort sollte stehen:

```
setmap d_ed
```

```
ed
```

```
setmap d
```

Die Script-Datei schaltet die normale Belegung nach Beendigung des Editors wieder ein. Bei mehreren Editoren muß man entsprechend mehrere Script-Dateien schreiben.

Martin Busik/ub

Bug? Data Becker & GFA only produce features

Fenster leeren

Bei Beckertext 1 gab es die Funktion, einen Text nach dem Speichern mit dem Menübefehl »Text löschen« komplett zu löschen und direkt mit einer neuen Arbeit zu beginnen. Der Menüpunkt fehlt bei Beckertext 2.0. Hier gibt es die Möglichkeit, den gesamten Text unter Menüpunkt »Alles markieren« zu kennzeichnen und dann mit »Löschen« verschwinden zu lassen.

Viel einfacher geht es aber, wenn Sie – nach Speichern eines Textes – den Menüpunkt »Schließen« aufrufen oder das Schließgadget benutzen. Dann erscheint ein Requester, der darauf hinweist, daß Sie das letzte Fenster schließen möchten, und der fragt, ob Sie Beckertext 2.0 beenden wollen? Außerdem sind zwei Schalter zu sehen: »Beenden« und »Nicht beenden«. Klicken Sie auf den zweiten, wird der gesamte Inhalt des Fensters gelöscht. So haben

TIPS ZU TEXTVERARBEITUNGEN

Tip	Thema	Seite
Beckertext-2.0-Accelerator / Beschleunigung mit einer Bitplane		80
Texte formatieren / Datenübertragung von anderen Systemen		80
Schließen der Workbench / Beckertext auf eigenem Screen		80
Diskettenverwaltung / Beckertext 2.0 in Spezialanwendung		80
Beckertext und der Speicher		80
Labelprint mit Pagestream 2 / Diskettenetiketten		81
Diskettenwechsel...ade / Besserer Start mit Beckertext 2.0		81
Numerische Tastatur / KeyMap-Editor		81
Fenster leeren / Text löschen (Beckertext 2.0)		81
Speichern: Sichere Sache / Automatisches Speichern (BT 2.0)		82
Druckertreiber für Beckertext 2.0 / Richtige Druckerwahl		82
Bildverbesserung per Software / Farbwahl zur Textverarbeitung		83
Outline und Shadow / Druckeroptionen		83
Und ED druckt doch / Drucken statt speichern		83

Sie relativ schnell wieder ein leeres Fenster und können einen neuen Text beginnen.

Selbstverständlich funktioniert das Ganze nur, wenn nur ein Fenster geöffnet ist. Bei geöffnetem zweiten Fenster wird es mit dem Kommando »Schließen« (falls bereits gesichert wurde) sofort, ohne weitere Abfrage, geschlossen.

ub

Speichern: Sichere Sache

Achtung, bei der Einstellung »Auto Speichern« von Beckertext 2.0:

Ihr Text wird hierbei automatisch im aktuellen Verzeichnis unter dem vorgegebenen Dateinamen gespeichert und erhält als Suffix eine sechsstellige Endung in hexadezimaler Schreibweise. Im Lade-Requester wird diese Datei nur angezeigt, wenn Sie als Suffix »alles« anklicken. Auch wenn Sie die Funktion »Auto Speichern« eingestellt haben, müssen Sie Ihre Arbeit vor Schließen des Fensters unter dem Dateinamen mit dem gewünschten Suffix speichern. Beckertext 2.0 fordert Sie auch mit einem entsprechenden Requester dazu auf.

Schließen Sie das Fenster, ohne vorher nochmals zu speichern, werden Sie feststellen, daß die automatisch gespeicherte Datei spurlos verschwunden ist. Stürzt Ihr Rechner aber ab, bleibt die zuletzt automatisch gesicherte Datei erhalten.

Auch durch Speichern vor Schließen des Fensters, wird eine automatisch gesicherte Datei gelöscht. Wobei das natürlich seine Richtigkeit hat, ansonsten wäre die Datei immer zweimal vorhanden.

Sollten Sie bei einem neuen Projekt noch keinen Dateinamen vergeben haben, speichert der Amiga automatisch mit sechsstelligem hexadezimalen Suffix. Er legt die Datei in diesem Fall im Hauptverzeichnis von Beckertext ab. Sie finden die Datei, wenn Sie im Lade-Requester als Suffix »alles« anklicken.

Speichern Sie einen Text als ASCII-Text, wird die automatisch angelegte Sicherungskopie dadurch nicht gelöscht. Die automatische Sicherung erfolgt nämlich immer im Beckertext-Format. Dabei werden alle Formatierungen und Einstellungen mit dem Dokument gerettet. Man erkennt das daran, daß die automatisch erstellte Sicherungskopie um einiges größer ist als die ASCII-Datei. In diesen Fällen bleibt Ihnen nur das Löschen der Sicherungskopie von Hand über den Menüpunkt »Datei löschen«. Achten Sie darauf, daß Sie die richtige Datei löschen: Es ist die Datei mit dem hexadezimalen Suffix. Solange Sie sich im Programm befinden, läßt Beckertext das Löschen der Datei aber nicht zu. Die Datei gilt als noch benutzt. Erst nach einem Neustart von Beckertext kann man sie löschen.

ub

Druckertreiber für Beckertext II

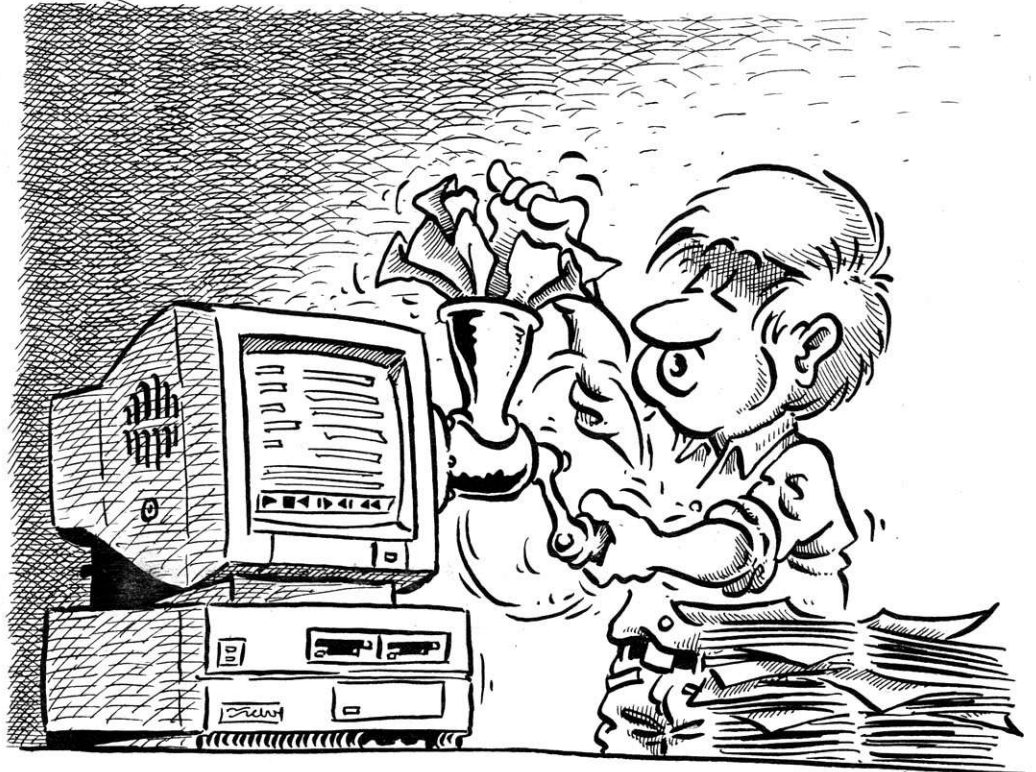
Beckertext II verwendet eigene Druckertreiber. Dennoch versucht sich das Textverarbeitungsprogramm beim Start, automatisch auf den Drucker einzustellen, der in den Preferences der Workbench eingestellt ist.

Beckertext ermittelt hierzu den Namen des eingestellten Druckers, hängt die Endung ».prt« an und sucht im Verzeichnis »BT-II/prt«

einen Drucker mit dem neuen Namen. Wird Beckertext fündig, lädt es den entsprechenden Treiber und arbeitet so vom Start an immer mit dem richtigen Drucker.

Nun besitzt Beckertext II einige Treiber für Drucker, für die es gar keinen speziellen Workbenchtreiber (V 1.3) gibt, z.B. den Star LC-24. Besitzer dieses Druckers verwenden in der Regel den Treiber mit Namen »EpsonQ«.

Wenn EpsonQ eingestellt ist, lädt Beckertext allerdings den entsprechenden Epson-Treiber. Das ist natürlich eine recht gute Wahl, besser wäre es jedoch den originalen Star-Treiber einzusetzen. Was kann man machen?



K.B.'91

— Sie erstellen eine Kopie des Treibers »EpsonQ« auf Ihrer Workbench-Diskette und benennen ihn in »StarLC24-10« um. Den neuen Namen selektieren Sie dann in den Preferences. Das folgende CLI-Kommando eignet sich, um den neuen Treiber zu erzeugen.

copy SYS:devs/printers/epson SYS:devs/printers/StarLC24-10

Bei Beckertext sind keine Veränderungen erforderlich.

— Der zweite Weg wäre, den Beckertext-Treiber »StarLC24-10.prt« in »EpsonQ.prt« umzubenennen. Hierzu muß man den originalen »EpsonQ.prt«-Treiber allerdings löschen oder umbenennen.

Haben Sie sich für einen Weg entschieden, prüfen Sie das Ergebnis: Sie starten Beckertext und drücken die Tastenkombination <Amiga_links F9>. Beckertext II zeigt im erscheinenden Dialogfenster den eingestellten Treiber.

Günter Kränze/ub

**Free speech is the
right to shout
»theatre« in a crowded
fire**

Bildverbesserung per Software

Wenn Sie keinen besonders hochwertigen Monitor besitzen, sind Ihnen vielleicht schon die Verzerrungen der Fenster-Titelleisten aufgefallen, die besonders in den Ecken des Bildschirms auftreten. Es gibt jedoch eine einfache Möglichkeit, diesen Effekt zu mildern. Man stellt sich die Bildschirmfarben mit Hilfe des Programms *Preferences* z.B. so ein:

```
Farbe 0: Türkis (066)
Farbe 1: Schwarz (000)
Farbe 2: Grau (777)
Farbe 3: Weiß (FFF)
```

Die Werte in Klammern geben die Positionen der RGB-Regler an (0 = links, F = rechts). Die Hauptsache bei der Farbwahl ist, daß Farbe 0 heller ist als Farbe 1. Dadurch steigt die Bildqualität.

Bei den gewählten Farben 0 und 1 ist der Kontrast relativ niedrig, was auch das Arbeiten im *Interlace*-Modus erleichtert, da das Flimmern verringert wird.

Übrigens verwendet die *Workbench 2.0* eine recht ähnliche Farbpalette, bei der aber Farbe 2 heller ist als Farbe 3. Und genau wie bei *Workbench 2.0* sehen Icons der *Workbench*-Versionen bis 1.3 jetzt recht merkwürdig aus. Abhilfe schafft »RemapIcon« von der *Fish-Disk 372*, das die Farben 0 und 1 der Icons vertauscht.

Gerson Kurz/ub

OUTLINE UND SHADOW

Haben Sie auch schon einmal damit zu kämpfen gehabt, daß Ihr Drucker so tolle Optionen wie »Shadow« oder »Outline« bietet, diese aber nicht von Ihrem Textverarbeitungsprogramm unterstützt werden?

Mit einem Editor (z.B. »DME« von der *Fish-Disk 284*) können Sie diese Optionen nutzen.

Als erstes brauchen wir den zu druckenden Text in Form einer *ASCII*-Datei – wohlgemerkt, den zu druckenden Text mit allen Steuerzeichen etc. wie das Textprogramm ihn an den Drucker überträgt. Manche Textprogramme bieten hierzu die Option, den zu druckenden Text in eine Datei umzulenken (»Dokumentum«). Bei anderen

Programmen muß man die Druckerausgabe mit dem Kommando *CMD* umleiten. *CMD* steht im Verzeichnis »utilities« auf der *Workbench-Diskette* (1.3).

Nehmen wir an, Ihr Drucker ist an der parallelen Schnittstelle angeschlossen, lautet der Befehl vom *CLI* aus:

```
run cmd parallel ram:testdatei
```

Nach diesem Befehl lenkt der *Amiga* die nächste Ausgabe über »par:« in die Datei »testdatei« in der *RAM-Disk*. Auch wenn Ihr Textprogramm Texte nicht direkt über die parallele Schnittstelle ausgibt, sondern »prt:« verwendet, funktioniert das Ganze: »prt:« ist das *Printer-Device*. Sobald der *Amiga* es einsetzt, werden die Zeichen, die an den Drucker geschickt werden, mit Hilfe des in den *Preferences* eingestellten Druckers übersetzt. Anschließend erfolgt die Druckausgabe jedoch wieder über »par:«, wenn in den *Preferences* die parallele Schnittstelle eingestellt ist.

Nun haben wir die Druck-Datei im *ASCII*-Format vorliegen: Laden Sie den zu druckenden Text mit dem Editor (z.B. *Cygnus-Ed* oder *DME*). Stellen Sie vor die Zeilen die in einer besonderen Art gedruckt werden sollen, die jeweils dafür vorgesehenen *Escape*-Steuerzeichen. Eine Liste aller unterstützten *Escape*-Codes finden Sie in Ihrem Druckerhandbuch. Bei *DME* erhalten Sie das *Escape*-Zeichen, wenn Sie gleichzeitig <Shift> und <Esc> drücken (Mit dem *Standardeditor* »ED« von der *Workbench* ist es dagegen nicht möglich, *Escape*-Sequenzen einzugeben). Den modifizierten Text können Sie dann ausdrucken, bzw. speichern, wobei Sie als Dateinamen »par:« wählen.

Uwe Kleine-Vogelpoth/ub

Und Ed druckt doch

Zwar hat der *Ed* keine Druckfunktion, doch läßt sich dies mit folgendem Trick leicht umgehen:

Man »speichert« den Text mit »sa /prt:« einfach an den Druckerport ab. Will man nur einen Teil des Textes ausdrucken, markiert man diesen mit den Blockfunktionen und druckt ihn dann mit »wb /prt:« aus.

ub

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber
Redaktionsdirektor: Dr. Manfred Gindler

Chefredakteur: Albert Absmeier, verantwortlich für den redaktionellen Teil
Stellv. Chefredakteur: Ulrich Brieden (Projektleitung)
Redaktion: Peter Aurich
Redaktionsassistent: Catharina Winter, Helga Weber (Fax: 089/4613-433)
Freie Mitarbeiter: Ingolf Krüger (ik), Ilse u. Rudolf Wolf (irw, Österreich)

Verlagsleitung: Wolfram Höfler

Desktop Publishing: Ulrich Brieden, Markus Kobs
Titel-Gestaltung: Friedemann Porscha
Desktop Publishing: Ulrich Brieden
Illustration: (Comic: Hermann der User: Karl Bielmeier)

Auslandsrepräsentation/Auslandsniederlassungen:

Schweiz: »Markt & Technik«, Vertriebs AG, Kollerstraße 37, CH 6300 Zug, Tel. 0041/42/440550, Fax: 0041/42/415770

Österreich: Markt & Technik Ges. mbH, Große Neugasse 28, A-1040 Wien, Tel. 0043/1/58713930, Fax: 0043/1/587139333

Gesamtvertriebsleitung: York von Heimbürg
Vertriebsmarketing: Helmut Player

Produktion: Klaus Buck (Ltg./180), Wolfgang Meyer (Stellv./887)

Druck: R. Oldenbourg, Graphische Betriebe GmbH, Hürderstraße 4, 8011 Kirchheim

Warenzeichen: Commodore ist Inhaber des Warenzeichens Amiga

Urheberrecht: Alle erschienenen Beiträge sind urheberrechtlich geschützt. Alle Rechte vorbehalten. Reproduktionen gleich welcher Art, ob Fotokopie, Mikrofilm oder Erfassung in Datenverarbeitungsanlagen, nur mit schriftlicher Genehmigung des Verlages.

Haftung: Für den Fall, daß im *Tips&Tricks*-Sonderheft unzutreffende Informationen enthalten sein sollten, kommt eine Haftung nur bei grober Fahrlässigkeit des Verlages oder seiner Mitarbeiter in Betracht.

© 1991 Markt & Technik Verlag Aktiengesellschaft

Vorstand: Otmar Weber (Vors.), Bernd Balzer, Dr. Rainer Doll, Lutz Glandt

Direktor Zeitschriften: Michael M. Pauly

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen: Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon 089/4613-0, Telex 522052, Telefax 089/4613-100

Die Redaktion bedankt sich bei allen, die an der Planung und Erstellung des Sonderhefts beteiligt waren; besonders bei der Produktion und Druckerei für die gezeigte Geduld.

Programmierkniffe

Hokus Pokus

Und jetzt – zum guten Abschluß – greifen wir für alle Programmierer auf dem Amiga noch mal ganz tief in die Trickkiste: mit Programmierkniffen, die für alle Sprachen anwendbar sind. Sie finden viele Beispiele in C, Modula-2, Assembler und auch BASIC. Und alle Tips sollten sich in die anderen Sprachen übertragen lassen.

Wer hat noch nicht, wer will noch mehr? Es gibt so viele Tricks, die man beim Programmieren einsetzen kann, damit ließe sich allein ein Sonderheft füllen, doch wir bringen auch eine ganze Menge: Die nächsten 14 Seiten gehören noch mal ganz den Programmierern.

Requester: Nicht so hastig

Wenn man einen Requester mit der Systemfunktion »Request« öffnet, muß man immer erst einen Augenblick warten, bis man den Requester abfragt. Das Öffnen des Requesters läuft asynchron als eigener Task: Das Programm, das den Requester anlegt, läuft also weiter, während der Requester aufgebaut wird. Fragt man den Requester zu früh ab, greift man evtl. in Leere.

Die erforderliche Pause erreicht man z.B. mit »Delay()« aus der »exec.library« (s.a. Seite 87). Es geht aber noch eleganter; in Modula-2 sähe das Ganze so aus:

```
Request(req, MyWindow);
WaitPort(MyWindow^.userPort);
msg := GetMsg(MyWindow^.userPort);
class := msg^.class;
ReplyMsg(msg);
IF reqSet IN class THEN ....
ELSE .... END;
```

Statt nach Request() die Systemfunktion Delay() aufzurufen, sollte man in der »NewWindow«-Struktur von »my_window« das IDCMP-Flag »REQSET« setzen. Dann kann man mit »WaitPort()« auf das vollständige Erscheinen des Requesters warten.

Lothar Armbrüster/ub

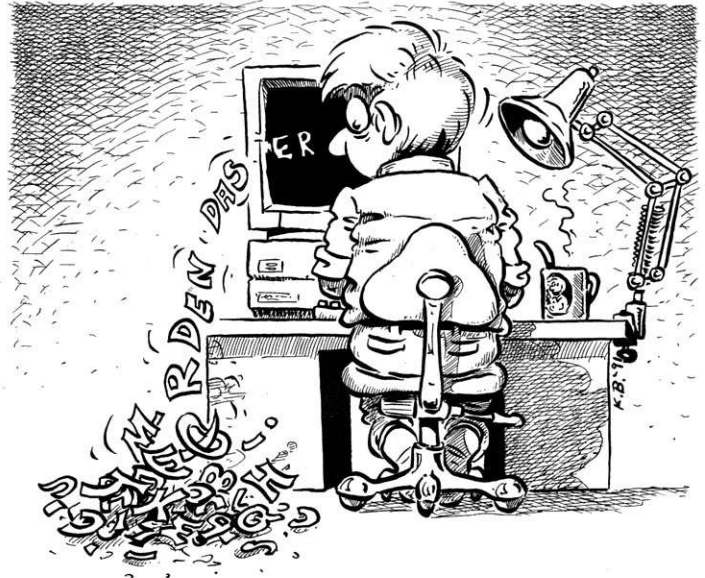
Rotieren mit Dynamik

In der im AMIGA-Magazin 1/91 S. 200 und 2/91 S. 65 abgedruckten Modula-2-Bibliothek findet sich u.a. die Prozedur »RotateString«. Allerdings muß man einige Beschränkungen hinnehmen, wenn man einen String umkehrt: So ist »String2« definiert als

```
ARRAY[0..12] OF CHAR
```

was bei Aufrufen von »RotateString« mit längeren Strings natürlich zu einigen Problemen führt. Und bei offenen Feldern ist die Beschränkung erst recht nicht tragbar. Besser wäre es, den folgenden Weg zu wählen:

Man übergibt der Prozedur nach wie vor ein offenes Feld, alloziert Speicher mit der Feldgröße, schreibt aus dem offenen Feld von hinten nach vorn byteweise die Elemente in den allozierten Speicher und kopiert wieder den Speicherbereich von der Anfangsadresse mit der Feldgröße an die Adresse der Stringvariablen. Zum Schluß muß



man noch den allozierten Speicher wieder freigegeben und die String-Ende-Kennung« anfügen.

Genau das macht die neue Prozedur »RotateString«:

```
PROCEDURE RotateString (VAR str : ARRAY OF CHAR) : BOOLEAN ;
VAR
  adr, start : ADDRESS ;
  i          : LONGINT ;

BEGIN
  adr := AllocMem (Length(str), MemReqSet{fast, memClear}) ;
  IF adr # NIL THEN
    start := adr ;
    FOR i := Length (str)-1 TO 0 BY -1 DO
      CopyMem (ADR(str[i]), adr, LONGINT(SIZE(CHAR))) ;
      INC (adr) ;
    END ; (* FOR *)
    CopyMem (start, ADR(str), Length(str)) ;
    str[Length(str)+1] := nul ;
    FreeMem (start, Length(str)) ;
    RETURN (TRUE) ;
  ELSE
    RETURN (FALSE) ;
  END ; (* IF *)
END RotateString ;
```

RotataString.mod

Lassen Sie Strings rotieren

Der Rückgabewert ist TRUE, wenn der Speicher alloziert werden konnte und damit die Prozedur ausgeführt werden kann, ansonsten FALSE. Jetzt brauchen wir ein Testprogramm.

Das folgende Modula-2-Programm »RotateTest.mod« verlangt eine Integerzahl zur Eingabe, wandelt sie mit »IntToString« in einen String und kehrt ihn schließlich um. Aus »-54« wird so »45-«. Übersetzungshinweise (M2Amiga):


```
m2c -y -d StringTool.def
m2c -y -d StringTool.mod
m2c -y -d RotateTest
m2l -x +r RotateTest
```

Das Modul ist somit re-entrant-fähig. Kompiliert und getestet haben wir das Programm mit M2Amiga V4.0 auf einem Amiga-2000B mit Kickstart 1.2 & 1.3.

```
MODULE RotateTest ;

FROM StringTool IMPORT IntToString, RotateString ;
FROM InOut IMPORT WriteString, WriteLn, ReadLongInt ;

VAR string : ARRAY[1..40] OF CHAR ;
    zahl : LONGINT ;
    ok : BOOLEAN ;

BEGIN
    WriteString("Bitte geben Sie eine Integer-Zahl ein: ") ;
    ReadLongInt(zahl) ;
    WriteLn ;

    IntToString(zahl, string) ;
    ok := RotateString(string) ;
    WriteString(string) ;
    WriteLn ;

END RotateTest.
```

RotataTest.mod

Ein Programm, um das obige Listing zu prüfen

Wer mit dem Modul aus dem AMIGA-Magazin arbeitet, muß die alte Prozedur nur durch die neue ersetzen, allerdings muß man die Import-Liste des Implementationsmoduls erweitern:

```
FROM SYSTEM IMPORT ADR
FROM ASCII IMPORT nul
FROM ExecL IMPORT CopyMem, FreeMem, AllocMem
FROM ExecD IMPORT MemReqs, MemReqSet
```

Im Definitionsmoduls muß nun statt:

```
PROCEDURE RotateString (VAR s : ARRAY OF CHAR) ;
```

folgendes stehen:

```
PROCEDURE RotateString (VAR str : ARRAY OF CHAR) : BOOLEAN ;
```

Die Prozedur »IntToString« ruft »RotateString« auf, weswegen diese ebenfalls geändert werden muß:

Eine lokale Variable vom Typ BOOLEAN ist nötig für die Funktionsprozedur »RotateString« und der Prozedur-Aufruf muß nun entsprechend anders lauten.

Die Ausführungen sind gültig für den Modula-2-Compiler M2Amiga Version 4.0, Besitzer älterer Versionen müssen beachten, daß die IMPORT-Listen anders lauten, da Definitionen und Prozeduren nicht getrennt sind, also:

```
FROM Exec IMPORT CopyMem, FreeMem, AllocMem, MemReqs, MemReqSet;
```

Statt

```
FROM String IMPORT Length ;
```

muß stehen:

```
FROM Strings IMPORT Length ;
```

Außerdem ist zu wichtig, daß das String-Modul nun »StringTool« heißt und nicht »MyTools«. Der Name muß gegebenenfalls im Testprogramm in der IMPORT-Liste geändert werden.

ub

Sesam öffne dich

Es gibt zwei verschiedene Exec-Funktionen, um eine Library zu öffnen:

- Eine heißt »OldOpenLibrary«. Der Offset zum Aufruf beträgt -408.
- Die Funktion »OpenLibrary« dagegen wird mit dem Offset -552 aufgerufen.

Welches ist der Unterschied zwischen den beiden Routinen? Kennen Sie ihn bereits?

Nur beim Aufruf von OpenLibrary muß der Programmierer eine Versionsnummer übergeben. OldOpenLibrary funktioniert ohne weitere Parameter.

Klaus Wenger/ub

Bessere Windows

Der folgende Tip hilft Ihnen, sauberere Amiga-Programme zu schreiben, die möglichst wenig Rechenzeit benötigen, und einen häufig gemachten Programmierfehler zu umgehen:

Angenommen, Sie möchten ein Window mit fester Größe, ohne »Sizing«-Gadget, öffnen und Sie haben das »SMART_REFRESH«-Bit in der »Flags«-Variable der »NewWindow«-Struktur gesetzt. Dann kümmert sich Intuition selbständig durch Speichern überlappender Windowbereiche darum, daß der Inhalt des Fensters bewahrt wird. Das würde heißen, Ihr Programm wird niemals eine »REFRESHWINDOW«-Meldung erhalten, die dazu auffordert, das Window neu zu zeichnen. Deshalb ist es doch auch überflüssig, das »NOCAREREFRESH«-Bit zu setzen, oder???

Leider falsch, denn das Bit steht nicht nur für den Fall, daß keine »REFRESHWINDOW«-Message geschickt wird, sondern es hat noch andere Auswirkungen:

Wenn das »NOCAREREFRESH«-Bit nicht gesetzt ist, werden die Systemgadgets jedesmal erneuert, wenn man etwas am Window verändert. Das bemerkt man daran, daß bei jedem Verschieben, Überlappen und Klicken in den Vordergrund die Gadgets in der Titelleiste des Windows flackern (ab OS 2.0 ist das Flackern noch ausgeprägter). Das kostet viel Rechenzeit und sieht auf Dauer sehr unprofessionell aus.

Doch Sie können leicht Abhilfe schaffen, indem Sie wirklich immer das »NOCAREREFRESH«-Bit setzen, auch wenn Sie nicht selbst das Neuzeichnen des Windows übernehmen und sowieso niemals eine »REFRESHWINDOW«-Message erhalten werden.

ub

TRICKKISTE FÜR PROGRAMMIERER

Tip	Thema	Seite
Requester: nicht so hastig / Request()		84
Rotieren mit Dynamic / RotateString.mod		84
Sesam öffne dich / OpenLibrary()		85
Bessere Windows / Window-Gadgets		85
Reservieren nicht vergessen / AllocMem() etc.		86
Höhere Auflösung / Screens und Overscan		86
Modula-2 geht den direkten Weg / Mausebfrage		86
Request nur mit Pause / Request() die zweite		87
Vorsicht Glatteis / Blitterfunktionen		87
Speichertests / AvailMem()		87
BASIC beim Start / RUN AmigaBASIC		88
Hintertür / Guru nicht sofort tödlich		88
Prozessortyp / Systemdaten lesen in Modula-2		88
Musik im CLI / Sound.c		90
Der Drucker verlangt viel		90
RAW-Keyes für Intuition / Tabelle		90
Eine Frage Mr. Joy / Joystick-Port abfragen		90
Fenster unter AmigaDOS 2.0 / Open()		91
ARexx-Taschen-Amiga / Rechenfunktionene unter ARexx		91
Auch König Oberon trickst / Tasks		92
Do you remember / Noch mal zur Erinnerung AllocMem() etc.		92
Zeichenketten verschlüsseln / Codierfunktion in BASIC		93
Text schwenkt marsch / Scrolling in Assembler		94
Das Ganze in C / Das Ganze in C (ist doch klar, oder?)		95
CIA kennt die Zeit / genaues Timing über Hardwaretimer		96
Tastatur läßt von sich hören / Tastatur-Piep		97
Neue Maus / Mauszeiger in BASIC ändern		98
Die Maus läßt fragen / ASK-Ersatz		98
Supervisor / 68000er-Betriebszustände		98

Reservieren nicht vergessen

Die Funktion »AllocMem()« dient zum Reservieren von Speicher. Wenn man viele Speicherbereiche benötigt, muß man die Funktion entsprechend oft aufrufen. Die Zeiger auf die reservierten Bereiche legt man in Variablen oder Feldern ab. Doch was passiert, wenn mitten im Programm der Speicher voll ist oder man ein Programm beenden möchte? Man muß alle bis dahin reservierten Bereiche wieder freigeben. Erst dann kann man das Programm verlassen. Doch halt, es geht auch anders. Intuition – Dein Freund und Helfer – hält zwei komfortable Routinen für Programmierer bereit, die die Speicherverwaltung vereinfachen (s.a. Seite 92).

```
struct Remember *remb;  
remb = NULL;  
mem1 = AllocRemember(&remb, 1024, MEMF_CHIP|MEMF_PUBLIC);  
mem2 = AllocRemember(&remb, 4711, MEMF_CLEAR|MEMF_PUBLIC);  
/*.....*/  
FreeRemember (&remb, TRUE);
```

Mit der Funktion »AllocRemember()« kann man Speicher reservieren und anschließend alle Bereiche auf einen Schlag wieder freigeben. Beachten Sie, daß Sie der Funktion einen Zeiger auf einen Zeiger übergeben müssen. Auch sollten Sie diesen Zeiger wie im Beispiel gezeigt vor dem ersten Aufruf auf Null setzen.

Daniel Swertz/ub

“Ich hab’s gleich”, sagte der Programmierer

Höhere Auflösung

Überall steht geschrieben, der Amiga hätte eine maximale Bildschirmauflösung von 640 x 512 Punkten. In Wahrheit steht Ihnen eine weit höhere Auflösung zur Verfügung. Ein Screen kann bis zu 704 * 564 Punkte groß sein (Overscan). Die 69376 zusätzlichen Punkte können Sie mit Hilfe der Grafikbefehle des Betriebssystems ohne weiteres nutzen. Besonders in C und Assembler ist das recht einfach:

Um einen Screen zu öffnen, müssen Sie eine »NewScreen«-Struktur erstellen. Der dritte Wert dieser Tabelle gibt die Screenbreite an, der vierte die Höhe. Üblicherweise stehen dort, beispielsweise für einen hochauflösenden Screen im Interlace, die Werte 640 und 512. Ersetzen Sie die Werte durch 704 bzw. 564; Sie können natürlich auch kleinere Werte wählen.

```
NewScreen.Width=704;  
NewScreen.Height=564;
```

Wenn Sie nun die Funktion »OpenScreen« anspringen erscheint der übergroße Screen. Er ist aber nur zum Teil sichtbar. Um dem abzuwehren, müssen Sie folgendes ändern:

Erstens sollten Sie das Bild des Monitors etwas verkleinern und neu zentrieren. Die Regler hierzu befinden sich im allgemeinen an der Rückseite des Monitors.

Zweitens müssen Sie das Bild mit Hilfe der »Preferences« nach oben links verschieben. Klicken Sie hierzu das große Rechteck in der Mitte des »Preference-Window« an und bewegen das gesamte Fenster mit der Maus in die gewünschte Ecke. Für die genaue Einstellung ist ein wenig Experimentieren notwendig. Sie dürfen das Bild nicht zu weit schieben, da sonst der obere Rand des Screens verschwindet. Wenn Sie den Monitor einmal eingestellt haben, können Sie in Ihren Programmen sofort die »Super-Hochauflösung« ausprobieren.

Der Effekt läßt sich im übrigen auch in allen anderen View-Modi ausnutzen, um mehr Punkte darzustellen. So stehen Ihnen für einen LowRes-Screen 352 Punkte in der Horizontalen zur Verfügung.

Warum nicht gleich die volle Auflösung?

Die Einschränkung auf 640 x 512 Punkte erfolgt, weil nicht alle Monitore diese extreme Auflösung darstellen können. Besonders an den Rändern kommt es zu Unregelmäßigkeiten. Außerdem sind in den Grenzbereichen nicht alle Sprites sichtbar.

Jürgen Brendel/ub

Modula-2 geht den direkten Weg

Da die Abfrage der Maustasten über das Betriebssystem des Amiga umständlich zu handhaben ist, fragen viele Programme die Hardware-Register direkt ab. Eine solche Abfrage ist auch in Modula-2 möglich. Die beiden folgenden Routinen wurden für den Modula-2-Compiler M2Amiga geschrieben. Sie arbeiten auch mit der PD-Version von der Fish-Disk 113. Die beiden Prozeduren sind als Modul ausgelegt und können durch:

```
FROM MouseButton IMPORT leftMouseButton, rightMouseButton;
```

in ein eigenes Programm importiert werden. Man kann die Prozeduren auch direkt ins Programmmodul schreiben, sie sind dann aber nicht mehr für andere Module verfügbar. Zunächst das Definitionsmodul:

```
DEFINITION MODULE MouseButton;  
  
(* Abfrage der rechten und linken Maustaste *)  
  
(* TRUE : rechte Maustaste gedrückt *)  
PROCEDURE rightMouseButton() : BOOLEAN;  
  
(* TRUE : linke Maustaste gedrückt *)  
PROCEDURE leftMouseButton() : BOOLEAN;  
  
END MouseButton.  
Anschließend folgt das Implementationsmodul:  
IMPLEMENTATION MODULE MouseButton  
  
FROM SYSTEM IMPORT BITSET;  
  
PROCEDURE rightMouseButton() : BOOLEAN;  
VAR  
  POTGOR[0DFF016H] : BITSET;  
BEGIN  
  IF (10 IN POTGOR) THEN RETURN FALSE ELSE RETURN TRUE END;  
END rightMouseButton;  
  
PROCEDURE leftMouseButton() : BOOLEAN;  
VAR  
  CIA[0BFE001H] : SET OF [0..7];  
  (* Adresse eines CIA-Registers *)  
BEGIN  
  (* wenn 7 Bit nicht gesetzt, ist die M-taste gedrückt *)  
  IF (6 IN CIA) THEN RETURN FALSE ELSE RETURN TRUE END;  
END leftMouseButton;  
END MouseButton.
```

MouseButton.mod

Mausabfrage über die Hardware

Das nächste Demoprogramm zeigt die Anwendung des neuen Moduls. Es besteht aus einer Endlosschleife. Sobald Sie die linke Maus-Taste drücken, gibt der Amiga eine Meldung auf dem Bildschirm aus. Drücken Sie die rechte Maus-Taste, wird die Schleife verlassen.

Zur Funktionsweise der Prozeduren. Durch Angabe der Speicherstelle in eckigen Klammern hinter der Variablen wird eine Mengenvariable auf ein Register des IO-Bausteins gesetzt. Die Mengeninhalte entsprechen den Bits des Registers. Somit kann der Zustand bestimmter Bits im Register leicht abgefragt werden.

Martin Murnleiter/ub

```
MODULE MouseButtonDemo;
(* Demonstration des Moduls MouseButtons *)

FROM MouseButtons IMPORT leftMouseButton, rightMouseButton;
FROM Terminal IMPORT WriteString, WriteLn;

BEGIN
LOOP
IF leftMouseButton() THEN
WriteString('linke Maustaste');WriteLn;
END;
IF rightMouseButton() THEN
WriteString('rechte Maustaste');WriteLn;
EXIT;
END;
END;
WriteString('Ende'); WriteLn;
END MouseButtonDemo.
```

MouseButtonDemo.mod

Programm, um das Modul »MouseButton« zu prüfen

Request — nur mit Pause

Wird ein Requester mit der Intuition-Funktion Request() aktiviert (er beinhaltet mehrere Image-Strukturen, String-Gadgets und ein Boolean-Gadget), dann meldet sich die Funktion Request() vorzeitig zurück d.h. das Programm läuft weiter, obwohl der Requester noch nicht fertig aufgebaut ist.

Unangenehm wird es, wenn anschließend z.B. eines der String-Gadgets aktiviert werden soll, was natürlich nicht geht.

Abhilfe schafft eine Zwangspause, zwischen Request() und ActivateGadget(), realisiert mit Delay(). Ein Beispiel:

```
...
Request(&RequesterStructure1, my_window);
Delay(5L);
ActivateGadget(&Gadget1, my_window, &RequesterStructure1);
...
```

Der Wert, den Sie der Funktion Delay() übergeben, muß ggf. auf die unterschiedlichen Rechnerkonfigurationen abgestimmt werden.

Harald Platz/ub

Vorsicht Glatteis

Manche Funktionen der »graphics.library« werden nicht sofort ausgeführt, sondern starten den Blitter und kehren dann ins Programm zurück; die eigentliche Arbeit erledigt dann der Coprozessor parallel zum Hauptprozessor. Hier muß der Programmierer darauf achten, daß er manche Dinge erst verändern darf, wenn die Blitter-Operationen beendet sind. Viele Grafik-Funktionen und Befehle, die Speicherbereiche kopieren und verschieben, nutzen den Blitter. Der Blitter erledigt beispielsweise folgende Aufgaben:

- Er löscht Speicherbereiche,
- scrollt einen rechteckigen Bildschirmbereich,
- zieht Linien oder zeichnet Polygone und
- kopiert Speicherbereiche.

So ist es gefährlich, den Speicherplatz einer Bitmap freizugeben, die zuvor mit »ClipBlit()« kopiert wurde. Es kann sein, daß der Blitter noch beschäftigt ist, während der Speicherplatz freigegeben und neu beschrieben wird. Da zieht man sich bzw. anderen Programmen den Boden unter den Füßen weg.

Eine ähnliche Situation liegt beim Einsatz von höheren Programmiersprachen vor, wenn man in einem Unterprogramm, das mit Areas arbeitet und z.B. mehrere Punkte eines Vielecks verbinden soll, wichtige Datenstrukturen lokal deklariert. Beim Rücksprung kann es dann geschehen, daß der Blitter noch Polygone zeichnet, während die zugehörigen »TmpRas«- oder »AreaInfo«-Strukturen wieder freigegeben werden. Also auch in C, Modula-2 oder BASIC

muß man aufpassen, wenn man Grafikfunktionen einsetzt, die der Blitter ausführt.

Als Lösung bietet es sich in diesem Beispiel an, Variablen am Anfang des Programms global zu deklarieren.

Ein weiterer Weg, Komplikationen zu vermeiden, ist die Verwendung von Warteschleifen hinter den Blitteraufrufen. Das verzögert jedoch die Ausführung Ihrer Programme und ist unsicher, da die nötigen Wartezeiten nicht vorauszusagen sind.

Die sicherste Methode, wenn Sie Daten unmittelbar nach einer Blitteroperation manipulieren möchten, ist es, die Funktion »WaitBlit()« einzusetzen; die Funktion wartet, bis der Blitter alle Aufgaben erledigt hat.

Bernhard Schloß/ub

Speichertests

Manchmal ist es nötig, innerhalb von Befehlsdateien in Abhängigkeit der Größe des verfügbaren RAM-Speichers bestimmte Aktionen zu starten oder zu unterlassen. Dafür gibt es mehrere Möglichkeiten:

- Unter Verwendung von CLI-Kommandos selbst:

In der »Startup-Sequence« müssen folgende Befehle stehen:

```
makedir ram:env
assign ENV: ram:env
```

```
avail >env:xx total ; oder »chip« bzw »fast«
setenv yy "100000" ; abzufragende Speichergröße
```

```
if $xx gt $yy val
echo "...ausreichend Speicher!"
else
echo "...Speicher reicht nicht!"
endif
```

```
delete >nil: ENV:xx\yy
```

avail.bat

Speichertest als Batch-Datei für's CLI

- Oder sie verwenden ein Assembler-Programm, das in diesem Fall die System-Funktion »AvailMem« einsetzt:

```
***** Testet auf vorhandenen Speicher
***** Rückgabewert: 0 wenn Speicher vorhanden ist
*****          5 wenn Speicher nicht vorh. ist
***** Syntax: memo <Zahl>
***** <Zahl> ist Speichergröße in KByte!

ExecBase = 4
AvailMem = -216
MEMF_CHIP = 1<<1
MEMF_FAST = 1<<2

cmp.l #5,d0
bhi Fehler ;zu viele Ziffernstellen
cmp.l #1,d0
bls Fehler ;Zahl vergessen anzugeben
subq.b #1,d0
add.l d0,a0
clr.l d1
clr.l d2
lea Multab,a1

Zeichen:
move.b -(a0),d1 ;Ziffer holen
cmp.b #53,a,d1 ;Zeichencode größer '9' ?
bcc Fehler
sub.b #530,d1 ;ASCII -> BCD
bcs Fehler ;Zeichencode kleiner '0' ?
mulu (a1),d1 ;multipl. mit Zehnerstelle
add.l d1,d2
addq.l #2,a1 ;nächste Zehnerstelle
subq.l #1,d0 ;alle Ziffern eingelesen ?
bne Zeichen
OK: mulu #1024,d2 ;multipl. mit 'k'
move.l ExecBase,a6
move.l #MEMF_CHIP,d1
jsr AvailMem(a6) ;wieviel CHIP-RAM noch frei ?
move.l d0,d3
```



```

move.l #MEMF_FAST,d1
jsr AvailMem(a6) ;wieviel FAST-RAM noch frei ?
add.l d3,d0 ;CHIP+FAST
cmp.l d2,d0 ;soviel wie angefordert ?
bcs warn ;nō -> warn
clr.l d0 ;ja -> O.K.
rts
warn: move.l #5,d0 ;Warncode 5
rts
Fehler: move.l #20,d0 ;Fehlercode 20
rts
Multab: dc.w 1
dc.w 10
dc.w 100
dc.w 1000
    
```

memo.asm

Noch'n Speichertest, diesmal als Assemblerprogramm

Das Assemblerprogramm belegt ca. 200 Byte. Die Befehlsdatei sieht nun so aus:

```

memo 977
; (man beachte den Unterschied zwischen 1000 Byte und 1 KByte)
if not warn
    echo "...ausreichend Speicher!"
else
    echo "...Speicher reicht nicht!"
endif
    
```

Im Vergleich zur ersten Version spart man die Ausführung der Befehle AVAIL, MAKEDIR, ASSIGN, SETENV, DELETE.

Ralf Gruner, Thomas Horak/ub

**I'd really like to
change the world -
But they won't give
me the source**

BASIC beim Start

Wer sich eine autostartende BASIC-Diskette angelegt hat, kann auch ein bestimmtes BASIC-Programm direkt aus der »Startup-Sequence« aufrufen. Man muß lediglich den Namen des aufzurufenden Programms in der »Startup-Sequence« mit angeben:

```
RUN AmigaBasic name
```

Eine Anwendung dieses Tricks: Der Amiga lädt direkt nach dem Booten eines Ihrer Programme und läßt es ablaufen. Sie sparen sich sowohl den Aufruf des Interpreters als auch den LOAD-Befehl für Ihr Programm. Auch der Umgang mit AmigaBASIC kann vereinfacht werden: Man kann ein ganz spezielles Startprogramm in BASIC schreiben, das der Amiga nach Laden des Interpreters ausführt. Das Programm könnte beispielsweise den Stack und Programmspeicher dimensionieren oder das BASIC-Verzeichnis einstellen, auf das der Amiga beim LOAD- und SAVE-Befehl zugreift. Ein Beispiel:

```

CHDIR "Programme"
FILES
CLEAR ,30000,8000
NEW
    
```

Das Programm ändert das Hauptverzeichnis auf den Ordner »Programme« und gibt dessen Inhalt aus. Anschließend wird der Programmspeicher mit CLEAR neu dimensioniert und das Startpro-

gramm löscht sich selbst. Nun kann der Programmierer mit der eigentlichen Arbeit beginnen. Der Amiga steht ihm wie gewohnt zur Verfügung.

Peter Czaja/ub

Hintertür

Ein Guru ist der Feind aller Programmierer. Erst erscheint auf dem Bildschirm ein Dialog-Fenster (Requester) mit dem Text »Software-Failure«. Sobald man in diesem Fenster das Feld »Cancel« anklickt, erscheint eine Systemmeldung (»Alert«) und alle Daten sind verloren.

Häufig gibt es allerdings eine Chance, wichtige Daten zu retten, auch wenn der berühmte Requester bereits erschienen ist. Man darf nur nicht sofort Cancel anklicken. Jede Fehlermeldung betrifft im allgemeinen nur einen Task. Hat man neben dem Programm, das zum Absturz führte, noch ein anderes Programm laufen, ist dieses vom Absturz meist nicht betroffen. Haben Sie ein zweites CLI-Fenster geöffnet, können Sie in diesem Fenster auch nach einem Guru weiterarbeiten. Ein Programmierer, dessen Programm abgestürzt ist, hat also die Möglichkeit wichtige Dateien aus dem Speicher des Amiga zu retten. Erst nach der Rettungsaktion betätigt man den Schalter Cancel.

Also nicht vergessen: lassen Sie bei kritischen Tests immer eine Hintertür offen. Wenn in einem Fenster etwas schief geht, haben Sie im zweiten einen Rettungsanker.

Heinz Berla/ub

PROZESSORTYP

Viele Firmen bieten Prozessorkarten für die Amiga-Modelle an, um deren Rechenleistung zu steigern. Hat man nun eine solche Karte im Einsatz, die es eventuell sogar noch erlaubt, zwischen den verschiedenen Prozessortypen umzuschalten, stellt sich dem Anwender die Frage, welcher Prozessor gerade aktiv ist? Das vorliegende Modula-2-Programm »WBenchInfo« beantwortet diese Frage, indem es in der Titelleiste des Workbenchscreens den momentan eingesetzten Prozessortyp, sowie die dargestellte Fernsehnorm anzeigt. Das Programm wurde für den TDI-Compiler (Version 3.0a) geschrieben:

Besonders in Hinblick auf die neuen Grafikchips, die es erlauben, softwaremäßig zwischen NTSC- und PAL-Modus umzuschalten, wurde die Anzeige der Fernsehnorm ins Programm aufgenommen.

Folgende Prozessortypen werden erkannt: Die Hauptprozessoren der Typen 68000, 68010 und 68020, sowie ein Mathematik-Coprocessor 68881.

Bei der Fernsehnorm unterscheidet das Programm zwischen dem amerikanischen NTSC- und dem europäischen PAL-Modus, sowie zusätzlich dem GENLOCK-Modus.

Im Quellcode des Programms sind auch die Routinen vorgesehen, die das Vorhandensein der Prozessoren 68030 und 68882 abfragen. Sie können diese bei Bedarf in den Quellcode übernehmen, was mit Einführung des Amiga 3000 recht sinnvoll wäre.

Nun zur Vorgehensweise von »WBenchInfo«: Das Programm wird mit dem Befehl

```
RUN > NIL: WBenchInfo
```

über's CLI bzw. die Shell (Workbench 1.3) aufgerufen. Die Umleitung des RUN-Befehls nach NIL: ist notwendig, um das CLI-Fenster nach dem Start von »WBenchInfo« schließen zu können.

Nun wird auf dem Workbenchscreen in der Titelleiste ein Layer erzeugt, in dem die Prozessorkonfiguration angezeigt wird.

Öffnet ein Programm allerdings einen eigenen Screen, ist es nicht möglich, »WBenchInfo« dort zu starten. Es wird, ohne eine Fehlermeldung zu erzeugen, sofort abgebrochen. Da jedoch die meisten Programme auf der Workbenchoberfläche eigene Windows öffnen, dürfte einem Einsatz des Programms nichts im Wege stehen. Der erzeugte Workbenchscreen muß jedoch mindestens 640 Punkte breit sein.

Zur Überprüfung auf die verschiedenen Prozessortypen müssen noch einige Libraries geöffnet werden, die den Zugriff auf die Daten des Betriebssystems erlauben. Dies sind die »exec.library«, die den verwendeten Prozessor verwaltet und die »graphics.library«, die angibt, in welcher Fernsehnorm das Monitorbild erzeugt wird. Nun wird mit Hilfe der Routinen der »layers-« und der »intuition.library« ein Layer der benötigten Größe erzeugt, in der Titelleiste des Workbenchscreens angezeigt, der Text eingetragen und die geöffneten Libraries wieder geschlossen.

Das Programm prüft von nun an in Abständen von jeweils einer Sekunde, ob sich die Konfiguration eventuell verändert hat und zeigt ggf. den neuen Text im Layer an.

Da die Zeitverzögerung mit dem »Delay()«-Befehl aus der »DOS.library« arbeitet, muß diese ebenfalls geöffnet werden. Der Einsatz von »Delay()« hat den Vorteil, daß das Programm nicht wertvolle Prozessorzeit beansprucht, sondern den Task in den Wait-Status versetzt. Die Abstände, in denen die Konfiguration getestet wird, kann durch Erhöhen des Multiplikationsfaktors im »Delay()«-Befehl erhöht werden.

Roland Wintgen/ub

```

MODULE WBenchInfo;
(* V2.0 (c) 12.10.89 by "Roland Wintgen"
  $A+, $Q+, $S-, $T- Compileranweisungen *)
FROM SYSTEM      IMPORT ADDRESS ,NULL;
FROM Intuition    IMPORT ScreenFlags;
FROM IntuitionBase IMPORT IntuitionBase;
FROM Libraries    IMPORT OpenLibrary ,CloseLibrary;
FROM GraphicsLibrary IMPORT GraphicsBase ,Jaml;
FROM GraphicsBase  IMPORT GfxBasePtr ,NTSC ,GENLOC ,PAL;
FROM LayersLibrary IMPORT LayersBase ,LayerPtr ,
  CreateUpfrontLayer,LayerSmart;
FROM Pens          IMPORT Move, SetDrMd, SetAPen;
FROM Text          IMPORT Text;
FROM Rasters       IMPORT SetRast;
FROM ExecBase      IMPORT ExecBasePtr ,AFB68010 ,AFB68020
  (*,AFB68030*),AFB68881 (*,AFB68882*);
FROM DOSProcessHandler IMPORT Delay ,Exit;
FROM DOSLibrary     IMPORT DOSBase ,TicksPerSecond;
VAR ExecBase[4]:ExecBasePtr;
    GfxBase:GfxBasePtr;
    IntuitionBasePtr:POINTER TO IntuitionBase;
    Layer:LayerPtr;
    Title:ARRAY [0..23] OF CHAR;
    flag:BOOLEAN;
BEGIN
  DOSBase:=OpenLibrary("dos.library",0);
  ExecBase:=ExecBasePtr(OpenLibrary("exec.library",0));
  GraphicsBase:=OpenLibrary("graphics.library",0);
  GfxBase:=GraphicsBase;
  IntuitionBasePtr:=ADDRESS(OpenLibrary("intuition.library",0));
  LayersBase:=OpenLibrary("layers.library",0);
  IF ~(ScreenType1 IN IntuitionBasePtr^.ActiveScreen^.Flags) & (WBenchScreen
  IN IntuitionBasePtr^.ActiveScreen^.Flags) & (IntuitionBasePtr^.ActiveScreen
  en^.Width > 639)
  THEN Title:="68000 | | PAL ";
  Layer:=CreateUpfrontLayer(IntuitionBasePtr^.ActiveScreen^.LInfo,
  IntuitionBasePtr^.ActiveScreen^.BMap,
  LONGCARD(IntuitionBasePtr^.ActiveScreen^.Width)-240,
  0,LONGCARD(IntuitionBasePtr^.ActiveScreen^.Width)-54,9,
  LayerSmart,NULL);
  LOOP
    flag:=FALSE;
    IF (AFB68010 IN ExecBase^.AttnFlags)
    THEN IF (Title[3] # "1")
      THEN Title[3]:="1";
      flag:=TRUE
    END
    ELSIF (AFB68020 IN ExecBase^.AttnFlags)
    THEN IF (Title[3] # "2")
      THEN Title[3]:="2";
      flag:=TRUE
    END
    (*ELSIF (AFB68030 IN ExecBase^.AttnFlags)
    THEN IF (Title[3] # "3")
      THEN Title[3]:="3";
      flag:=TRUE
    END*)
  END
END

```

```

ELSE IF (Title[3] # "0")
  THEN Title[3]:="0";
  flag:=TRUE
END
END;
IF (AFB68881 IN ExecBase^.AttnFlags)
THEN IF (Title[12] # "1") THEN Title[8]:="6";
  Title[9]:="8";
  Title[10]:="8";
  Title[11]:="8";
  Title[12]:="1";
  flag:=TRUE
END
(*ELSIF (AFB68882 IN ExecBase^.AttnFlags)
THEN IF (Title[12] # "2")
  THEN Title[8]:="6";
  Title[9]:="8";
  Title[10]:="8";
  Title[11]:="8";
  Title[12]:="2";
  flag:=TRUE
END*)
ELSE IF (Title[12] # " ")
  THEN Title[8]:=" ";
  Title[9]:=" ";
  Title[10]:=" ";
  Title[11]:=" ";
  Title[12]:=" ";
  flag:=TRUE
END
END;
CASE GfxBase^.DisplayFlags OF
  NTSC:IF (Title[18] # "T")
    THEN Title[16]:=" ";
    Title[17]:="N";
    Title[18]:="T";
    Title[19]:="S";
    Title[20]:="C";
    Title[21]:=" ";
    Title[22]:=" ";
    flag:=TRUE
  END
  | GENLOC:IF (Title[18] # "N")
    THEN Title[16]:="G";
    Title[17]:="E";
    Title[18]:="N";
    Title[19]:="L";
    Title[20]:="O";
    Title[21]:="C";
    Title[22]:="K";
    flag:=TRUE
  END
  | PAL:IF (Title[18] # "P")
    THEN Title[16]:=" ";
    Title[17]:=" ";
    Title[18]:="P";
    Title[19]:="A";
    Title[20]:="L";
    Title[21]:=" ";
    Title[22]:=" ";
    flag:=TRUE
  END
END
END;
IF flag
THEN SetRast(Layer^.rp,1);
  SetDrMd(Layer^.rp,Jaml);
  SetAPen(Layer^.rp,0);
  Move(Layer^.rp,1,7);
  Text(Layer^.rp,Title,23)
END;
Delay(1*TicksPerSecond)
END
END;
CloseLibrary(LayersBase);
CloseLibrary(ADDRESS(IntuitionBasePtr));
CloseLibrary(ADDRESS(GraphicsBase));
CloseLibrary(ADDRESS(ExecBase));
CloseLibrary(DOSBase);
END WBenchInfo.

```

WorkBenchInfo.mod

Programm zeigt den Prozessortyp an

Musik im CLI

Das C-Programm »Sound.c« läßt das CLI erklingen. Es kann einen Ton beliebiger Frequenz und Dauer erzeugen:

Das Listing (»Sound.c«) wurde für den Aztec-C-Compiler geschrieben. Sie übersetzen und linken das Programm mit den Befehlen:

```
cc sound.c -A +L
ln sound -LC32 -LM32
```

Aufgerufen wird »Sound« mit:

Sound Frequenz Dauer

Die Frequenz wird in Hertz angegeben, die Dauer in 1/50-Sekunden. (Achtung! Das Programm prüft nicht auf unzulässige Eingabe wie Frequenz = 0). Besonders gut eignet sich »Sound.c« für akustische Fehlermeldungen in Stapeldateien. Durch mehrfachen Aufruf lassen sich kleine Melodien erzeugen.

Malte Zöckler/ub

```
#include <exec/types.h>
#include <exec/memory.h>

main(argc,argv)
long argc;
char *argv[];
{
    BYTE *wave;
    UWORD *Volume = 0xdff0a8, *Laenge=0xdff0a4,
        *Period = 0xdff0a6, *DMACON=0xdff096;
    long *Memory = 0xdff0a0, i;
    if (argc != 3)
    { printf("USAGE: %s Frequenz(in Hz) Dauer(in 1/50 sec.)\n",
        argv[0]);
        exit(FALSE);
    }
    if ((wave = AllocMem(16, MEMF_CLEAR | MEMF_CHIP)) == NULL)
        exit(FALSE);
    for (i=8; i<8; i++)
        *(wave+8+i) = 10*i;
    *Memory = wave; /* Speicher fuer DMA */
    *Volume = 64; /* maximale Lautstaerke */
    *Laenge = 8; /* 16 Byte Laenge */

    *Period = 1000000000/(atoi(argv[1])*16*279.37);
    /* Periodendauer berechnen */
    *DMACON = 0x8001; /* Los geht's */
    Delay(atoi(argv[2])); /* Warten */
    *DMACON = 0x000f; /* Stop ! */
    FreeMem(wave,16);
}
```

Sound.c

Musik erklingt im CLI

Der Drucker verlangt viel

Nach dem Booten ist der Drucker (prt:) noch nicht ansprechbar. Daher muß sich beim ersten Versuch, Daten an den Drucker zu senden, die Start-Diskette im Laufwerk befinden. Durch folgende Zeile in der »Startup-Sequence« wird prt: bereits vorzeitig aktiviert:

COPY nil: to prt:

Für das Öffnen benötigt der Amiga einige Dateien in den genannten Verzeichnissen:

- l/Port-Handler
- devs/parallel.device beziehungsweise
- devs/serial.device
- devs/printer.device
- devs/printers/printername

Prt: wird übrigens auch aktiviert, wenn der Drucker noch nicht eingeschaltet ist.

Dieter Bruns/ub

Integer math: it's all so pointless

RAW-Keyes für Intuition

Wer sich mit der Programmierung der Benutzeroberfläche des Amiga, »Intuition« beschäftigt, braucht Angaben über die RAWKEY-Codes des numerischen Tastenblocks. Die Tabelle oben rechts zeigt alle Werte, die nötig sind.

Wichtig sind außerdem die Codes für die Leertaste (0x40) und die <Ctrl>- (0x63) und die <Caps Lock>-Taste (0x62).

Marc Jackisch/ub

RAW-KEYS

Taste	Code	Qualifier
[	0x5a	0x0000
]	0x5b	0x0000
/	0x5c	0x0000
*	0x5d	0x0000
-	0x4a	0x0100 (IEQUALIFIER_NUMERICPAD)
+	0x5e	0x0000
Enter	0x43	0x0100 (IEQUALIFIER_NUMERICPAD)
.	0x3c	0x0100 (IEQUALIFIER_NUMERICPAD)
0	0x0f	0x0100 (IEQUALIFIER_NUMERICPAD)
1	0x1d	0x0100 (IEQUALIFIER_NUMERICPAD)
2	0x1e	0x0100 (IEQUALIFIER_NUMERICPAD)
3	0x1f	0x0100 (IEQUALIFIER_NUMERICPAD)
4	0x2d	0x0100 (IEQUALIFIER_NUMERICPAD)
5	0x2e	0x0100 (IEQUALIFIER_NUMERICPAD)
6	0x2f	0x0100 (IEQUALIFIER_NUMERICPAD)
7	0x3d	0x0100 (IEQUALIFIER_NUMERICPAD)
8	0x3e	0x0100 (IEQUALIFIER_NUMERICPAD)
9	0x3f	0x0100 (IEQUALIFIER_NUMERICPAD)

Eine Frage Mr. Joy

Vielen Programmierern, gerade Assemblerneulingen, bereitet die Joystickabfrage Kopfschmerzen, da man nicht nur Bits testen, sondern diese teilweise auch noch miteinander verknüpfen muß. Doch es geht leichter, als es zu sein scheint, wie es das Programm »JoystickTest.asm« demonstriert.

Anstatt sich mit einzelnen Bits herumzuschlagen, liest man einfach das Register »JOYxDAT« und maskiert die für die Joystickstellung nicht relevanten Bits aus. Durch diese Prozedur ergibt sich für jede Joystickstellung ein anderer, charakteristischer Wert, den man nur noch zu testen braucht und dann in die entsprechende Routine verzweigt, die auf eine bestimmte Joystick-Aktion reagiert.

»JoystickTest.asm« testet für Port 1 alle acht(!) Richtungen und den Feuerknopf. Um Port 2 zu testen, müssen die Befehle

```
move $dfff00c,d0
```

```
und
```

```
bst #7,$bfe001
```

```
durch
```

```
move $dfff00a,d0"
```

```
bzw.
```

```
bst #6,$bfe001
```

ersetzt werden.

Gerson Kurz


```
JoyTest:
move    $dff00c,d0
and     #$0000001100000011,d0
cmp     #$0300,d0
beq     Links
cmp     #$0003,d0
beq     Rechts
cmp     #$0100,d0
beq     Oben
cmp     #$0001,d0
beq     Unten
cmp     #$0103,d0
beq     RechtsOben
cmp     #$0200,d0
beq     LinksOben
cmp     #$0301,d0
beq     LinksUnten
cmp     #$0002,d0
beq     RechtsUnten
cmp     #$7f00,d0
btst    #7,$bfe001
beq     Feuer
rts
```

```
Links:
...
rts
Rechts:
.
```

JoystickTest.asm

Joystickabfrage über die Hardwareregister

Fenster unter AmigaDOS 2.0

Wie jeder Programmierer weiß, lassen sich Fenster auf dem Amiga prinzipiell mit zwei Methoden öffnen, über Intuition oder über AmigaDOS. Letztere Methode funktioniert über den »Open()«-Befehl der »dos.library«, wobei statt einer Datei (File) ein Fenster mit folgender Definition geöffnet wird:

```
"CON:[x]/[y]/[breite]/[höhe]/[name]",0
```

Neu ist bei Kickstart 2.0 die Möglichkeit, hinter dem Namen Definitionen über die Art des Windows anzugeben, und zwar so:

```
"CON:[x]/[y]/[breite]/[höhe]/[name]/[flag]/[.....]",0
```

Dabei sind folgende Flags erlaubt:

- size / nosize - Gadget für Fenstergröße
- drag / nodrag - Gadget zum Bewegen des Fensters
- depth / nodepth - Gadget zum Ändern der Tiefe des Fensters
- close / noclose - Gadget zum Schließen des Fensters
- backdrop - Fenster liegt immer hinter allen anderen Fenstern
- noborder - Keine Fensterumrahmung

Wer z.B. bisher besondere Programme benutzte, um in seinem Haupt-CLI die Umrandung auszuschalten, hat's jetzt einfacher:

```
newcli window con:0/0/640/512//nosize/nodrag/nodepth/noborder/backdrop
```

öffnet ein Fenster in der maximal möglichen Auflösung (wird in den Preferences festgelegt) ohne irgendwelche störenden Gadgets, die immer auf der unteren Ebene liegen. (Damit wird auch die Workbench verdeckt, d.h. besser nicht bei Workbench-Usern)

Gerson Kurz/ub

ARexx-Taschen-Amiga

Wozu ist ein Computer gut, der nicht einmal rechnen kann? Im CLI des Amiga gibt's nur das Kommando EVAL, um kurze Berechnungen durchzuführen, doch das Programm arbeitet nur mit natürlichen Zahlen, kann nur zwei Operanden bearbeiten und hat viel zu wenig Funktionen.

Doch mittels der Sprache Arexx (Bestandteil von OS 2.0) und der ARP-Shell läßt sich ein sehr komfortables Kommando implementieren, um mathematische Ausdrücke zu berechnen. Bei Arexx wird

das Program »calc.rexx« mitgeliefert, das so lautet:

```
/* calculator program: interprets the argument string */
arg x
interpret 'say ' x
```

Man ruft das Programm (nach der Arexx-Installation) mit der Befehlszeile auf:

```
rx calc <expression>
```

Damit lassen sich alle mathematischen Befehl verwenden, über die Arexx verfügt. Um auch transzendente Funktionen (wie SIN,POWER,LOG) aufrufen zu können, müssen Sie die folgende neue Version des obigen Programms verwenden:

```
/* calculator program vl.2 :
now supports extended rexxmathlib function */

/* Zwei immer nützliche Variablen !!

pi = 3.14159265359
e = 2.71828182846

/* Library einbinden */

if ~show('l','rexxmathlib.library') then
if ~addlib('rexxmathlib.library',0,-30) then
do
say "Unable to open 'rexxmathlib.library'"
exit 0
end

/* Hilfstext bei Aufruf ohne Parameter */

if arg() ~= 1 then do
say "Usage : rx calc [expression]"
say "          where expression is any valid mathematical term"
exit 0
end

/* Ausrechnen */

arg x
interpret 'say ' x
exit 0
```

Taschenrechner.rexx

Bringen Sie Ihrem Amiga mit Arexx das Rechnen bei

Aber Achtung, um diese Version zu verwenden, müssen folgende Dateien im Verzeichnis »LIBS:« vorhanden sein:

- LIBS:rexxmath.library
- LIBS:rexxsyslib.library
- LIBS:mathieeedoubtrans.library
- LIBS:mathieeedoubbas.library

Wie Sie an der Verwendung der beiden letzten Librarys sehen, können Sie nun im doppelt genauen IEEE-Format rechnen. Für eine genaue Beschreibung der durch die »rexxmath.library« bereitgestellten Funktionen lesen Sie bitte die entsprechende Fachliteratur, aber probieren sie doch mal Ausdrücke aus wie:

```
rx calc sin(e**2-1/tan(pi))/log(3)
```

Das Ergebnis: »0.329543803«

Um den Umgang mit diesem Programm zu erleichtern, können Sie noch den ARP Befehl ALIAS verwenden. Geben Sie ein:

```
alias ? rx calc []
```

und von jetzt an können Sie Ausdrücke (ähnlich wie im Seka-Assembler) einfach durch

```
? [Expression]
```

berechnen lassen.

Im Gegensatz zum Seka muß sowohl beim expliziten Aufruf

```
rx calc []
```

als auch bei Verwendung des Befehls ALIAS ein Leerzeichen zwischen den Programmnamen und dem Ausdruck stehen.

Gerson Kurz/ub

Auch König Oberon trickst

Wer sich mit der Programmierung von Tasks, Interrupts, Input-Handlern und ähnlichem in Oberon beschäftigt, stellt fest – oder liest es im Handbuch, aber wer liest Handbücher? –, daß von eben diesen Tasks etc. nicht mehr auf die globalen Variablen des Moduls zugegriffen werden kann, in dem die betreffende Prozedur (PROCEDURE) deklariert ist.

Wenn man jedoch die betreffende globale Variable in einem Assemblerteil einrichtet, der später dazugelinkt wird, kann man auf die Variable doch zugeifen. Dazu muß man in der Variablendeklaration des Moduls (MODULE) auf das entsprechende Label im Assembler-teil verweisen. Ein Beispiel:

• Im Oberon-Listing steht:

```
MODULE Variablentest;
VAR TransPort["blub"]:E.MsgPortPtr;(* die globale Variable*)
```

```
PROCEDURE Task*;
VAR myMsg : E.MessagePtr;
BEGIN
    E.PutMsg(TransPort,myMsg);(* Zugriff auf glob. Var. *)
END Task;
```

END Variablentest.

Und im Assemblerteil verwendet man:

```
blub: dc.l 0
```

So kann man etwa eine Message von einem nachträglich eingerichteten Task an den Haupttask schicken.

Die Angaben beziehen sich auf den Oberon-Compiler von Fridtjof Siebert, erhältlich bei A+L.

Jörg von Frantzius/ub

Do you remember?

Erinnert sich Ihr Programm an alle »AllocMem()«-Aufrufe, um später die Speicherblöcke wieder freizugeben? Besonders bei einer Vielzahl von Speicheranforderungen kann man leicht den Überblick verlieren, und ein Programm entwickelt sich zum Speicherfresser.

Das Amiga-Betriebssystem bietet kein sog. Resource-Tracking an, das alle Anforderungen aufzeichnet und nach Beendigung des Programms alle noch ausstehenden Ressourcen wieder freigibt. Zwar übernehmen einige Laufzeitsysteme (z.B. von Kickpascal oder M2Amiga) Teile dieser Aufgaben, dennoch obliegt dem Programmierer das meiste dieser Pflicht. Dabei wird er von einem meist unbekannten Routinenpaar der »intuition.library« unterstützt: die Funktionen »AllocRemember()« und »FreeRemember()«.

»AllocRemember()« basiert auf der Exec-Routine »AllocMem()« und erlaubt es, Speicherplatz zu reservieren. Im Gegensatz zur Exec-Funktion richtet »AllocRemember()« zusätzlich einen Knoten ein, über den alle durch diese Routine belegten Speicherplätze verkettet werden.

Mit »FreeRemember()« können abschließend alle durch »AllocRemember()« belegten Memory-Bereiche auf einen Schlag wieder freigegeben werden. Die Routinen sind besonders interessant, wenn mehrere Daten gespeichert werden sollen, deren Anzahl aber vorher nicht gekannt ist, wie u.a. das Ablegen der Dateinamen eines Verzeichnisses.

• Für Assemblerprogrammierer:

```
MemBlock = AllocRemember (RememberKey, Size, Flags)
D0 = -396 (Offset) A0 D0 D1
FreeRemember (RememberKey, ReallyForget)
-408 A0 D0
```

• Für C-Programmierer:

```
APTR AllocRemember( struct Remember **, ULONG, ULONG)
```

```
VOID FreeRemember ( struct Remember **, BOOL)
```

Bei »AllocRemember()« sind die Parameter »Size« und »Flags« identisch mit denen der Exec-Funktion »AllocMem()«, sie bestimmen Größe und Typ (z.B. MEMF_CHIP) des gewünschten Speicherbereiches. »RememberKey« ist der Zeiger auf den Zeiger auf die erste »Remember«-Struktur. Er muß vor dem ersten Aufruf von »AllocRemember()« NULL (= 0) sein, damit die Funktion korrekt arbeitet. Bevor Sie nach »FreeRemember()« erneut Speicherbereiche belegen, muß er ebenfalls wieder auf NULL gesetzt werden.

Assemblerprogrammierer lassen den Pointer auf ein Longword mit dem Startwert 0 zeigen. Die Funktion der »Intuition.library« liefert den Zeiger auf den reservierten Speicherblock und ergänzt einen Knoten in Form einer (weiteren) »Remember«-Struktur, falls die Anforderung nicht erfüllt werden kann, wird NULL zurückgegeben. Auch in diesem Falle können Sie weiterhin Speicherbereiche über »AllocRemember()« reservieren.

Irren ist menschlich.....

»AllocRemember()« belegt bei jedem Aufruf zwei Speicherblöcke, einmal den gewünschten Speicherblock und zum anderen Platz für den Knoten: die sog. »Remember«-Struktur, definiert im Include-File »Intuition/intuition.h«:

```
struct Remember
{
    struct Remember *NextRemember;
    ULONG RememberSize;
    UBYTE *Memory;
};
```

»NextRemember()« bildet den Zeiger auf die nächste »Remember«-Struktur, während »RememberSize« die Größe des Speicherbereichs angibt, auf den »Memory« zeigt. Der Pointer auf den ersten Knoten wird im Feld »RememberKey«, der als Parameter beim Funktionsaufruf benötigt wird, abgelegt. Der Nachteil der Routine:

Jede Speicheranforderung bedeutet eine weitere für den Knoten, was Zeit kostet und zur Fragmentierung des RAMs führt.

Anhand der Liste können alle Speicherbereiche auf einen Schlag mit der Intuition-Routine »FreeRemember()« freigegeben werden. Der Boolean-Ausdruck »ReallyForget« definiert, ob nur die einzelnen Speicherblöcke (FALSE) oder auch die zugehörigen »Remember«-Knoten freigegeben werden sollen (TRUE). Zu beachten ist, daß beim Parameterwert FALSE Intuition keinerlei Informationen über die einzelnen Speicherbereiche mehr vorliegen und das Programm dann alle Blöcke mit Hilfe der Exec-Routine »FreeMem()« freigeben muß.

Das folgende Beispielprogramm, erstellt mit dem SAS-C-Compiler V5.10, belegt die angegebene Anzahl von Speicherblöcken. Der Anwender kann festlegen, ob »Remember«-Strukturen und Speicherblöcke gemeinsam oder getrennt freigegeben werden sollen (siehe Kommentare im Listing).

Roger Fischlin/ub

```
/* - Remember-Demo -
- Autor : Roger Fischlin -
- Sprache : SAS-C V5.10 -
*/
/* ----- Include-Files ----- */
#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <proto/all.h>
#include <stdio.h>
/* ----- Prototype ----- */
void ShowRememberKey (struct Remember* Key);
```

```

/* ----- Konstanten ----- */
#define MEMORY_SIZE 1024      /* Größe des Speicherblocks */
#define MEMORY_TYPE MEMF_PUBLIC /* dessen Typ */
#define MEMORY_MAXBLOCKS 10  /* max. Anzahl Speicherblöcke */
/* ----- Variablen ----- */
struct IntuitionBase* IntuitionBase;
struct Remember* RememberKey = NULL;
int Number = 0;
int Count = 0;
char Antwort [2];
BOOL Fehler = FALSE;
CPTR Memory [MEMORY_MAXBLOCKS];
/* ----- die verketteten Remember-Strukturen auflisten ----- */
void ShowRememberKey (struct Remember* Key)
{
    printf("\nDie verketteten Remember-Strukturen :\n");    while (Key)
    {
        printf("$%08x NextRemember : $%08x\n",Key,Key->NextRemember);
        printf("      RememberSize : %d\n",Key->RememberSize);
        printf("      Memory      : $%08x\n",Key->Memory);
        Key=Key->NextRemember;
        /* Zeiger auf nächste Remember-Struktur */
    }
}

/* ----- Hauptprogramm ----- */
void main (void)
{
    /* Intuition-Library öffnen */
    IntuitionBase = (struct IntuitionBase*)
        OpenLibrary("intuition.library",33L);
    /* Konnte Library geöffnet werden ? */
    if (IntuitionBase)
    {
        /* wieviele Blocks sollen belegt werden ? */
        printf("wieviele Blöcke (max. %d ) sollen belegt werden ? ",
            MEMORY_MAXBLOCKS);
        scanf("%d",&Number);
        if (Number>MEMORY_MAXBLOCKS) Number=MEMORY_MAXBLOCKS;
        RememberKey = NULL ;
        /* der Remember-Key muß NULL vor dem ersten Aufruf sein ! */
        while ((Count<Number) && (!Fehler) )
        {
            if (Memory[Count] =
                (CPTR)AllocRemember (&RememberKey, MEMORY_SIZE, MEMORY_TYPE))
                printf("Memory[%03d] = $%08x\n",Count+1,Memory[Count]);
            else
            {
                printf("Speicherblock #d konnte nicht belegt werden !\n",
                    Count+1);
                Fehler=TRUE;
            }
            Count++;
        }
        ShowRememberKey(RememberKey); /* Liste ausgeben */
        printf("Speicher getrennt freigeben (j/n) ? ");
        scanf("%s",&Antwort);
        if (Antwort[0] == 'n')
        {
            FreeRemember(&RememberKey,TRUE);
            /* Remember-Strukturen und Speicher freigeben */
            printf("Remember-Strukturen und Speicher freigegeben\n");
        }
        else {
            FreeRemember(&RememberKey,FALSE);
            /* nur Remember-Strukturen freigeben */
            printf("Remember-Strukturen freigeben\n");
        }
        for (Count=0 ; Count<Number ; Count++)
        {
            if (Memory[Count]) /* NULL-Zeiger abfangen */
            {
                FreeMem((void *)Memory[Count],MEMORY_SIZE);
                printf("Speicherblock bei $%08x freigegeben\n",
                    Memory[Count]);
            }
        }
        CloseLibrary((struct Library*) IntuitionBase);
    }
    else printf("konnte Intuition-Library V33 nicht öffnen !\n");
}

```

RememberDemo.c

Speicherverwaltung leichtgemacht

Zeichenketten verschlüsseln

Mit den beiden Unterroutrinen »Codiere« und »Decodiere« ist es möglich, eine Zeichenkette sicher zu verschlüsseln. Das wird dadurch erreicht, daß im Verschlüsselungsalgorithmus die periodische Sinusfunktion vorkommt. Entschlüsselung ist nicht möglich, da einem Funktionswert der Sinusfunktion unendlich viele Ausgangswerte zugeordnet sind, und so keine eindeutige Umkehrung der Funktion existiert. Aufgerufen werden die beiden Routinen fast gleich: In beiden Fällen muß die Variable »text\$« den zu codierenden Text enthalten, und nach dem Aufruf der Routine steht der verschlüsselte String in »codtext\$«. Der Unterschied liegt in der Funktion der Variablen »schlüssel«.

...und Computer sind auch nur Menschen

In der Routine »Codiere« kann kein bestimmter Schlüssel angegeben werden, da er durch das Programm zufällig ermittelt wird. Nach der Verschlüsselung steht in dieser Variable der Schlüssel, der unbedingt mit dem Paßwort gespeichert werden muß, um eine spätere Kontrolle zu ermöglichen. In der Routine »Decodiere« hingegen muß der Schlüssel zusammen mit dem zu verschlüsselnden Text angegeben werden. Die Routine ermöglicht eine Kontrolle eines eingegebenen Paßworts: stimmt der verschlüsselte String mit dem überein, der vorher durch »Codiere« ermittelt wurde, ist das Paßwort korrekt.

Damit eignet sich dieses Programm nicht zur Verschlüsselung von Daten, die später einmal wieder in Klartext zurückverwandelt werden müssen. Jedoch kann man mit ihm eine sehr sichere Paßwortkontrolle realisieren. Ein Beispiel: Um eine Zugangsberechtigung zu einem System zu erhalten, muß der User ein Paßwort angeben. Bei seiner Neueintragung wird zusammen mit dem angegebenen Paßwort auch der erzeugte Schlüssel gespeichert. Will sich der User nun wieder Zugang zum System verschaffen, gibt er sein Paßwort ein. Das Programm codiert dieses mit Hilfe von »Decodiere« und dem gespeicherten Schlüssel. Ist der String, den die Routine liefert, korrekt (entspricht dem original-verschlüsselten String), ist das Paßwort korrekt. Diese Arbeitsweise wird im Demoprogramm aufgezeigt.

Christian Meis

```

'Demoprogramm zu Verschlüsselungsroutinen
'written by Christian Meis
marke:
CLS : INPUT "Neues Paßwort oder Paßwortkontrolle (n/k)? ",a$
a$=UCASE$(a$)
IF a$="N" THEN GOTO neu
IF a$="K" THEN GOTO kontrolle
GOTO marke
neu:
CLS : LOCATE 10,1
PRINT "Neues Paßwort eingeben" : PRINT
PRINT "Bitte Usernamen eingeben:" : LINE INPUT user$
PRINT : PRINT "Bitte Paßwort eingeben:" : LINE INPUT pass$
IF pass$="" OR user$="" THEN GOTO neu
Codiere code,pass$,codedpass$
OPEN "df0:"+"user$ FOR OUTPUT AS #1
PRINT #1,code
PRINT #1,codedpass$
CLOSE #1
PRINT "Fertig." : END
kontrolle:
ON ERROR GOTO fehler
CLS : LOCATE 10,1
PRINT "Paßwort kontrollieren" : PRINT

```



```
PRINT "Bitte Usernamen eingeben." : LINE INPUT user$
OPEN "df0:"+user$ FOR INPUT AS #1
PRINT "Bitte Paßwort eingeben." : LINE INPUT pass$
INPUT #1,code : INPUT #1,codedpass$ : CLOSE #1 : ON ERROR GOTO 0
Decodiere code,pass$,kontroll$
IF kontroll$=codedpass$ THEN PRINT "Paßwort ist richtig." : END
PRINT "Paßwort ist falsch." : END
```

```
fehler:
IF ERR=53 THEN PRINT "User existiert nicht." : END
RESUME
```

```
SUB Codiere(schlüssel,text$,codetext$) STATIC
RANDOMIZE TIMER : schlüssel=RND(1)
FOR a=1 TO LEN(text$)
x=ASC(MID$(text$,a,1))
x=(INT(x*schlüssel)+1)*SIN((a*10+3)/12)
cloop1:
IF x>255 THEN x=x-255 : GOTO cloop1
cloop2:
IF x<0 THEN x=x+255 : GOTO cloop2
codetext$=codetext$+CHR$(x)
NEXT a
codetext$=CHR$(7)+codetext$
END SUB
```

```
SUB Decodiere(schlüssel,text$,codetext$) STATIC
FOR a=1 TO LEN(text$)
x=ASC(MID$(text$,a,1))
x=(INT(x*schlüssel)+1)*SIN((a*10+3)/12)
dloop1:
IF x>255 THEN x=x-255 : GOTO dloop1
dloop2:
IF x<0 THEN x=x+255 : GOTO dloop2
codetext$=codetext$+CHR$(x)
NEXT a
codetext$=CHR$(7)+codetext$
END SUB
```

Paßwort.bas

Sicher verschlüsselt mit Sinus-Funktion

Text schwenkt – Marsch

Viele Programmierer, gerade Assemblerfreaks, schlagen sich mit der Frage herum, wie man einen Scrolltext programmiert? Wir wollen hier eine recht einfache, aber elegante Methode vorstellen:

»Scroll.asm« öffnet ein Fenster mit einem Text darin, verschiebt diesen um ein Textzeichen nach links und gibt am freiwerdenden rechten Rand das nächste Textzeichen aus.

Unser Programm den Text mit der Routine »ScrollRaster()« aus der »graphics.library«. Und das war's auch schon! Den Text können

WAR'S DAS...

... oder wer will noch mehr? Noch mehr Tips & Tricks natürlich.

Aller Anfang ist schwer – das gilt auch für die Tips&Tricks -Sonderhefte. Wie hat Ihnen die Erstausgabe gefallen? Was können wir besser machen? Stimmt die Mischung? Welche Tips waren die besten?

Viele Fragen, auf die wir gerne eine Antwort von Ihnen hätten; damit das nächste Sonderheft noch besser wird? Schreiben Sie uns Ihre Meinung – geben Sie uns ein paar Tips & Tricks. Unsere Adresse

AMIGA-Redaktion, Kennwort: Tips & Tricks-Sonderheft
Markt & Technik Verlag AG
Hans-Pinsel-Str.2, 8013 Haar bei München

Sie natürlich beliebig verändern, und Sie müssen nicht einmal die Textlänge angeben – das macht der Assembler für Sie. Das folgende Programm ist mit dem Seka-Assembler geschrieben worden, aber es ist ohne größere Änderungen auf andere Assembler übertragbar. Es kann lediglich sein, daß die Anweisung

```
blk.b 70,$20
```

durch

```
dc.b 70,$20
```

o.ä. ersetzt werden muß.

ub

```
OpenWindow = -204
CloseWindow = -72
ScrollRaster = -396
Text = -60
Move = -240
ClearScreen = -48
WaitTOF = -270
OpenLibrary = -408
CloseLibrary = -414
```

```
move.l 4,a6 ; ExecBase -> a6
lea gfxname,a1
jsr OpenLibrary(a6) ; graphics.library öffnen
move.l d0,gfxbase
lea intname,a1
jsr OpenLibrary(a6) ; intuition.library öffnen
move.l d0,intbase
move.l d0,a6
lea window,a0
jsr OpenWindow(a6) ; Fenster öffnen
move.l d0>windowptr
move.l d0,a0
move.l $32(a0),rp ; Rastport merken
move.l gfxbase,a6
move.l rp,a1
jsr ClearScreen(a6) ; Fenster löschen -> Rahmen weg
```

```
marke:
move.l textlength,d6 ; wird unten berechnet!
lea textstart,a5 ; Beginn des Textes
```

```
loop:
move.w #630,d0
move.w #9,d1
move.l rp,a1
jsr Move(a6) ; Grafikcursor -> (630/9) = rechtes Fensterende
move.l rp,a1
move.l a5,a0
moveq #1,d0
jsr Text(a6) ; Zeichen ausgeben
add.l #1,a5 ; Zeiger auf nächstes Zeichen setzen
jsr WaitTOF(a6) ; auf Elektronenstrahl warten
move.l rp,a1
move.w #8,d0 ; Zahl an Pixeln, um die nach links gescrollt wird
move.w #0,d1
move.w #0,d2
move.w #0,d3
move.w #640,d4
move.w #12,d5
jsr ScrollRaster(a6) ; Text nach links scrollen
btst #6,$bfe001 ; linke Maustaste gedrückt?
beq.s ende ; ja -> Ende
dbra d6,loop ; nächstes Zeichen
bra marke ; Text beginnt von vorne
```

```
ende:
move.l>windowptr,a0
move.l intbase,a6
jsr CloseWindow(a6) ; Fenster schließen
move.l intbase,a1
move.l 4,a6
jsr CloseLibrary(a6) ; intuition.library schließen
move.l gfxbase,a1
jsr CloseLibrary(a6) ; graphics.library schließen
rts ; Programmende
```

```
gfxname: dc.b 'graphics.library',0
intname: dc.b 'intuition.library',0
even
intbase: dc.l 0
```

```
gfxbase: dc.l 0
windowptr: dc.l 0
rp: dc.l 0
window:
dc.w 0,210,640,12 ; Window: (0|210)-(640|222)
dc.b -1,-1 ; Default-Zeichenstifte verwenden
dc.l 0,0,0,0,0,0
dc.w 10,10,500,10,1 ; Min+Max (unwichtig), WBenchScreen
even
```

; Zwischen textstart und textende steht der zu scrollende Text mit
; "dc.b". Die Textlänge wird vom Assembler berechnet, und muß nicht
; angegeben werden.

```
textstart:
dc.b 'Wie man sieht, ist es ganz einfach, einen zwar primitiven '
dc.b 'aber funktionierenden Scrolltext zu programmieren. Zwar '
dc.b 'kann er nicht mit den Profiscrolltexten aus den Intros '
dc.b 'verglichen werden, aber für eine kurze Nachricht an einen '
dc.b 'Kumpel reicht er, und es geht ja nur ums Prinzip...'
dc.b 'Beendet wird das Programm mit der linken Maustaste.'
blk.b 70,$20 ; 70 Leerzeichen, bevor der Text
; wieder beginnt

textende:
even
textlength: dc.l [textende-textstart-1]
```

Scroll.asm

Laufschrift, d.h. Scrolling, mit System

Das Ganze in C

Scrolling in C ist natürlich ebenfalls eine einfache Sache: Gescrollt wird ebenfalls über »ScrollRaster()«, was den Vorteil hat, sehr einfach eine Laufschrift zu erzeugen. Um den Scrolltext zu wechseln, geben Sie einfach Ihren Text im Array »scrollt[]« ein.

Die Scrollgeschwindigkeit verändern Sie mit »s«. Nur Werte bis 8 sind sinnvoll. Probieren Sie mal

s = 3

Natürlich können Sie eine andere Copper-Liste oder einen anderen Font benutzen, oder wie wär's mit einem Sternenhintergrund? Die Möglichkeiten sind unbegrenzt!

ub

```
#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>

#define adress 0xbfe001

UBYTE scrollt[] = "HELLO AMIGA FREAKS I hope you like this little \
    routine    Press left mouse button to exit \
    Bye Bye.....";

/* Hier kann ihr Scrolltext stehen. Mit dem Zeichen \
    setzen Sie in der nächsten Zeile Ihren Text fort. */

UBYTE *pocol,*pocol2;

struct IntuitionBase *IntuitionBase;
struct Screen *FScreen;
struct Window *FWindow;
struct GfxBase *GfxBase;
struct UCPList *clist=NULL;
struct RastPort *r;
struct ViewPort *vp;

struct NewScreen FirstNewScreen =
{ 0, 0, 320, 256, 4, 0, 0, NULL, CUSTOMSCREEN, NULL, NULL, NULL, NULL, };

struct NewWindow nw = { 0, 0, 320, 256, 0, 0, NULL, BORDERLESS |
    BACKDROP | RMBTRAP, NULL, NULL, (UBYTE *)"", NULL, NULL,
    0, 0, 0, 0, CUSTOMSCREEN, };

MakeCList()
{ int i;
```

```
pocol = 0xdff180;
pocol2 = 0xdff182;
for(i=0;i<15;i++)
{ CWAIT(clist,120+i*4,0);
  CMOVE(clist,*pocol,0x000|i); }
for(i=0;i<15;i++)
{ CWAIT(clist,120+i*4+16,0);
  CMOVE(clist,*pocol,0x000|(18-i)); }
CEND(clist);
vp->UCopIns = clist;
RethinkDisplay();
}
```

```
VOID scrollen()
{ char *cia;
  int x,y,p,s,w,l,c;
  p = 0; c = 0; w = 300; s = 2; y = 167;
  /*s ist die Geschwindigkeit, Werte bis 8 sind sinnvoll*/
  l = sizeof(scrollt);
  l = l-1;
  cia = (char *)adress;
  SetAPen(r,1);
  do
  { WaitTOF();
    ScrollRaster(r,s,0,0,y,w-l,y+9);/*Hier wird gescrollt*/
    Move(r,(w-s-p),y+8);
    if(l<=c) c = 0;
    Text(r,&(scrollt[c]),1);
    p = p+s;
    if (p>(8-s))
    { p = 0; c++; }
  } while((*cia & 0x40) == 0x40);
}
```

main()

```
{ Open_All();
  r = &(FScreen->RastPort);
  ShowTitle(FScreen, FALSE);
  MakeCList();
  scrollen();
  Close_All();
}

/*Alles öffnen*/
Open_All()
{ void *OpenLibrary();
  struct Window *OpenWindow();
  struct Screen *OpenScreen();
  if (!(IntuitionBase = (struct IntuitionBase *)
    OpenLibrary("intuition.library", 0L)))
  { printf("Sorry, keine intuition.library gefunden!\n");
    Close_All();
    exit(FALSE); }
  if (!(GfxBase = (struct GfxBase *)
    OpenLibrary("graphics.library", 0L)))
  { printf("Sorry, kann Gfx.lib nicht aufmachen!\n");
    Close_All();
    exit(FALSE); }
  if (!(FScreen = (struct Screen *) OpenScreen(&FirstNewScreen)))
  { printf("Screen hat keine Zeit!\n");
    Close_All();
    exit(FALSE); }
  nw.Screen = FScreen;
  if (!(FWindow = (struct Window *) OpenWindow(&nw)))
  { printf("nOh Sorry, Window will nicht aufgehen\n");
    Close_All();
    exit(FALSE); }
  if (!(clist=(struct UCPList *)
    AllocMem(12,MEMF_PUBLIC|MEMF_CLEAR))) Close_All();
  vp = &(FScreen->ViewPort);
  SetRGB4(vp,0,0,0,0); }

/*Alles geöffnete schliessen*/
Close_All()
{ if (FWindow) CloseWindow(FWindow);
  if (FScreen) CloseScreen(FScreen);
  if (GfxBase) CloseLibrary(GfxBase);
  if (IntuitionBase) CloseLibrary(IntuitionBase);
```

Scroll.c

Text-Scrolling die zweite; diesmal in C

CIA kennt die Zeit

Die CIA-Timer sind für Timingprobleme geradezu prädestiniert, denn sie laufen auf allen heutigen AMIGAs mit fast gleicher Geschwindigkeit (und zwar unabhängig vom Systemzustand, nur auf NTSC-Rechnern sind sie um ca. 0,9 Prozent beschleunigt - im VBlank-Interrupt beläuft sich dieser Unterschied dagegen schon auf ganze 20 Prozent!!!).

Bei der Beschreibung der CIA-Programmierung wird jedoch oft vergessen, daß wir die beiden 16-Bit-Timer auch zu einem 32-Bit-Timer kombinieren können. Damit läßt sich der Zählbereich von knappen 92 Millisekunden auf weit mehr als eineinhalb Stunden erweitern, ohne daß nach der Initialisierung auch nur ein weiterer Programmschritt auf den Zählvorgang verschwendet werden müßte. Wie wir den CIA zu diesem Zweck programmieren, ist dem dokumentierten Beispielprogramm zu entnehmen.

```

; ***
; *** CIA-Example.s
; ***
; *** Beispiel für die Zusammenfassung der beiden 16-Bit-
; *** Timer im CIA 8520 zu einem 32-Bit-Timer
; ***
; *** V1.0 for Seka-Assembler, 12-Oct-1990
; ***
; *** written by Sven Golchert
; ***

; *** Exec-Offsets ***
ExecBasePtr = 4
Forbid      = -132
Permit      = -138

; *** CIA-B-Offsets ***
ciaTAL = $0400 ; LoByte des Timer A-Puffers
ciaTAH = $0500 ; HiByte des Timer A-Puffers
ciaTBL = $0600 ; LoByte des Timer B-Puffers
ciaTBH = $0700 ; HiByte des Timer B-Puffers
ciaICR = $0800 ; Interrupt-Kontrollregister
ciaCRA = $0e00 ; Timer A-Kontrollregister
ciaCRB = $0f00 ; Timer B-Kontrollregister

; *** Custom Chip Offsets ***
IntReqR= $001e ; Interrupt-Anforderungen READ ONLY
IntEna = $009a ; Interrupt-Zulassungen WRITE ONLY
IntReq = $009c ; Interrupt-Anforderungen WRITE ONLY
Col00 = $0180 ; Farbregister 0 WRITE ONLY

START:

move.l ExecBasePtr.w,a6 ; Multitasking verbieten
jsr Forbid(a6)
lea $dff000,a4 ; Custom Chips Basisadresse
lea $bfd000,a5 ; CIA-B Basisadresse

RESET_CIA_B:

move.w #$2000, IntEna(a4) ; verbiete das Auslösen des
; CIA-B-Interrupts
move.b #$7f, ciaICR(a5) ; sperre alle möglichen Interrupt-
; quellen des CIA-B
move.b #$00, ciaCRA(a5) ; stoppe Timer A
move.b #$00, ciaCRB(a5) ; dito Timer B
move.w #$2000, IntReq(a4) ; lösche mögliche Interrupt-
; Anforderung von CIA-B

SET_CIA_B_TIMERS:

; Zunächst die Timer mit der gewünschten Anzahl Warteimpulse
; initialisieren. Der Seka kann die gesuchte Zahl anzeigen:
; '? Wartezeit [ms] * E-Takt (kHz) / 1000 <RETURN>'
; bzw. '? Wartezeit [s] * E-Takt (kHz) <RETURN>'
; (der E-Takt beträgt bei PAL 7093,79 kHz, bei NTSC 7159,09).
; Das niederwertige Wort kommt in den Timer A, das höherwertige
; in Timer B. Das Beispiel gilt für $000ad303 (= 1 s)

move.b #$03, ciaTAL(a5)
move.b #$d3, ciaTAH(a5)
move.b #$0a, ciaTBL(a5)
move.b #$00, ciaTBH(a5)

```

```

START_CIA_B_TIMERS:

move.b #$82, ciaICR(a5) ; Wenn Zähler B unterläuft (Ende)
; soll sich der CIA mit
; Interrupt-Request melden

move.b#$49, ciaCRB(a5) ; Inmode 'Timer A', d.h., daß mit
; jedem Unterlauf von TA Timer B
; um eins vermindert wird

move.b #$01, ciaCRA(a5) ; Inmode 'Clock': E-CLK des 68000

ADJUST_TIMER_A_LATCH:

; Wir schreiben jetzt noch $fff in den Latch von Timer A.
; Tun wir das nicht, kommt es zu falschen Zählzeiten, denn
; nach jedem Unterlauf würde Timer A mit dem alten Latch (im
; Beispiel $d303) initialisiert, und statt nach 1 s ist
; die CIA im Beispiel bereits nach rund 0,84 Sek. fertig!

move.b #$ff, ciaTAL(a5)
move.b #$ff, ciaTAH(a5)

WAIT_FOR_INTERRUPT:

btst #13, IntReqR(a4) ; Warteschleife, bis CIA-B-
beq.s WAIT_FOR_INTERRUPT ; Request eintrifft
tst.b ciaICR(a5) ; CIA-Interruptstatus löschen
move.b #$00, ciaCRA(a5) ; Timer A anhalten (Timer B hält
; automatisch; Runmode 'Oneshot')
move.w #$2000, IntReq(a4) ; CIA-B-Request löschen

EXIT:

move.w #$a000, IntEna(a4) ; CIA-B-Interrupts zulassen
move.l ExecBasePtr.w,a6 ; Multitasking wieder zulassen
jsr Permit(a6)
rts

```

Timing.asm

Für exaktes Timing sollte man die CIAs einsetzen

Probleme könnten mit AMIGAs der nächsten Generation auftreten, denn eine Änderung in Kickstart 2.0 deutet darauf hin, daß die Zählrate der CIA-Timer evtl. nicht mehr direkt vom Haupttakt abhängig sein wird (siehe AMIGA 10/90, Seite 62, »Zukunftssicheres Programmieren, Teil 4«, Timing). Aber – warten wir's ab!

Sven Golchert/ub

Wer Amiga sagt, muß auch plus sagen

Copperlisten in Assembler!

Wenn man in Assembler eine Copperliste generieren will, steht man vor dem Problem, die dem Copper verständlichen Befehle zu erzeugen. Beispiel:

```
DC.L $7001ffff ; Warten, bis Zeile 112 erreicht
```

Wäre es nicht einfacher mit folgendem Makro?

```
CMWAIT 0,112
```

Folgendes Beispielprogramm demonstriert die leichte Handhabung der Makros, indem eine kleine Copperliste erzeugt wird, die die Hintergrundfarbe manipuliert. Nach Drücken der linken Maustaste, wird das Programm beendet.

Das Programm wurde auf dem Profimat getestet. Die Umsetzung auf andere Assembler sollte einfach sein, sofern diese Makrodefinitionen erlauben.

Rainer Haßmann/ub

Tastatur läßt von sich hören

Wer wenig Übung beim Schreiben mit der Tastatur hat, wird laufend auf dem Monitor überprüfen, ob er nicht evtl. einen Buchstaben doppelt geschrieben, oder durch zu zaghaftes Anschlagen einen Buchstaben ausgelassen hat.

Eine Tastatur, die sich akustisch zu Wort meldet, verbreitet hier ein gutes Gefühl der Sicherheit. Diesem Zweck dient das folgende, äußerst kurze Assembler-Programm. Es kann ganz normal aus dem CLI mit <RUN> gestartet werden. Ein Einbau in die »Startup-Sequence« ist natürlich auch möglich. Durch Betätigung der <Esc>-Taste beendet man das Programm.

Olaf Keitsch/ub

; Folgende Makros erlauben eine einfache Behandlung des Coppers

```
CMOVE: MACRO %1,%2 ;Format:cmove wert,ziel
    DC.w (%2&$ffff),%1
    ENDM
CWAIT: MACRO %1,%2 ;Format:cwait x,y
    DC.w ((%2<<8)+(%1<<1)+1),$ffff
    ENDM
CEND: MACRO ;Format:cend
    DC.w $ffff,$ffff
    ENDM
```

```
demo:
    move.l 4,a6
    move.l #Coppersize,d0
    move.l #2,d1
    jsr -198(a6)
    move.l d0,Chip
    move.l d0,a1
    beq MemError
    move.l #Copperlist,a0
    move.l #Coppersize-1,d0
Copperlist: ;Copperliste ins Chipram kopieren
    move.b (a0)+,(a1)+
    dbra d0,Copperlist
    move.l 4,a6
    lea GfxName,a1 ;GfxLibrary öffnen
    jsr -408(a6)
    move.l d0,GfxBase
    move.l d0,a3
    add.l #32,a3 ;alte Copperliste merken
    move.l 4,a6
    jsr -132(a6) ;Multitasking einschalten
    lea $dff000,a5
    move.w #0080,$96(a5)
    move.l (a3),d3
    move.l Chip,(a3)
    move.w #80a0,$96(a5)
waitbutton:
    btst #6,$bfe001 ;Auf linke Maustaste warten
    bne waitbutton
    move.w #0080,$96(a5)
    move.l d3,(a3) ;Alte Copperliste setzen
    move.w #8180,$96(a5)
    move.l 4,a6
    jsr -138(a6) ;Multitasking einschalten
    move.l GfxBase,a1 ;GfxLibrary schließen
    jsr -414(a6)
    move.l 4,a6
    move.l Chip,a1 ;Speicher fuer Copperliste freigeben
    move.l #Coppersize,d0
    jsr -210(a6)
    moveq #0,d0
    rts
MemError:
    move.l #-1,d0
    rts
Copperlist:
    CMOVE $000,$180 ;Farbe schwarz ins Hintergrundregister
    CWAIT 0,125 ;Auf X=0,Y=125 warten
    CMOVE $f00,$180 ;Farbe rot ins Hintergrundregister
    CWAIT 0,225 ;Auf X=0,Y=225 warten
    CMOVE $ff0,$180 ;Farbe gelb ins Hintergrundregister
    CEND ;Auf nicht erreichbare Zeile warten ...
Copperend:
Coppersize=Copperend-Copperlist

GfxBase: dc.l 0
Chip: dc.l 0
GfxName:dc.b 'graphics.library',0
even
end
```

Copperdemo.asm

Makros erleichtern die Copperprogrammierung

```
delay = -$00c6
openlibrary = -$0228
closelibrary = -$019e
execbase = 4

move.l execbase,a6 ; DOS.library öffnen
lea dosname,a1
move.l #0,d0
jsr openlibrary(a6)
move.l d0,dosbase
clr.l d2

prog: clr.l d0
    move.b $bfe01,d0 ; Register mit Keycode laden
    btst #0,d0 ; up/down-Flag testen
    bne nono ; Taste noch gedrückt springe
    move.l #19000,d3 ; sonst lade Zähler
nono: clr.l d0
    move.b $bfe01,d0 ; Keycode Zwischenabfrage
    cmp.b d0,d2 ; gleiche Taste?
    bne go ; wenn nicht Tonausgabe
    btst #0,d0 ; up/down-Flag testen
    beq go ; Taste nicht gedrückt -> weiter
    dbeg d3,nono ; Schleife fertig?
    move.l #1,d3

go: move.b d0,d2 ; Keycode merken
    jsr sound ; Tonausgabe
    clr.l d1
    moveq #74,d1 ; Keycode für Abbruch setzen
    cmp.b d0,d1 ; Abbruchtaste gedrückt?
    bne prog ; wenn ja -> weiter zum Ende
    move.l execbase,a6 ; DOS.library schließen
    move.l dosbase,a1
    jsr closelibrary(a6)
    rts ; Ende
```

```
sound: btst #0,d0 ; Taste losgelassen?
    beq routend ; wenn ja -> keine Tonausgabe
    cmp.b #7f,d0
    beq nix ; zur Tonausgabe
    btst #7,d0 ; Sondertaste?
    beq routend ; wenn ja, ist nicht!!!
nix: movem.l d0-d3,-(a7)
    lea.l wave(pc),a0 ; Zeiger auf Wave
    move.l a0,$dff0a0 ; in Register von Kanal 0
    move.l a0,$dff0b0 ; in Kanal 1
    move.w #78,$dff0a4 ; Länge der Wave in Words
    move.w #78,$dff0b4 ; und in Kanal 1
    move.w #2540,$dff0a6 ; Tonhöhe(Periode) Kanal 0
    move.w #2540,$dff0b6 ; und in Kanal 1
    move.w #64,$dff0a8 ; Lautstärke von Kanal 0
    move.w #64,$dff0b8 ; und Kanal 1
    move.w #1000000000000001,$dff096 ; Audio-DMA Start
    jsr warte
    move.w #1000000000000001,$dff096 ; Audio-DMA Stopp
    movem.l (a7)+,d0-d3
routend: rts ; Rückkehr
warte: move.l dosbase,a6 ; Library aufrufen
    move.l #4,d1
    jsr delay(a6)
    rts
even
wave:
dc.b 45,80,100,80,45,0,-45,-80,-100,-80,-45,0 ; Wavedaten
dc.b 41,72,90,72,41,0,-41,-72,-90,-72,-41,0
```

TRICKKISTE

```
dc.b 37,65,82,65,37,0,-37,-65,-82,-65,-37,0
dc.b 33,59,74,59,33,0,-33,-59,-74,-59,-33,0
dc.b 30,54,67,54,30,0,-30,-54,-67,-54,-30,0
dc.b 27,48,61,48,27,0,-27,-48,-61,-48,-27,0
dc.b 25,44,55,44,25,0,-25,-44,-55,-44,-25,0
dc.b 22,40,50,40,22,0,-22,-40,-50,-40,-22,0
dc.b 20,36,45,36,20,0,-20,-36,-45,-36,-20,0
dc.b 18,32,41,32,18,0,-18,-32,-41,-32,-18,0
dc.b 16,29,37,29,16,0,-16,-29,-37,-29,-16,0
dc.b 15,27,33,27,15,0,-15,-27,-33,-27,-15,0
dc.b 13,24,30,24,13,0,-13,-24,-30,-24,-13,0
even
```

```
dosbase:
dc.l 0
dosname:
dc.b 'dos.library',0
```

Tastatur_beep.asm

»beep, beep« – pro Tastendruck ein Ton

Neue Maus

Sie wollen den Mauszeiger verändern? Von BASIC aus? Das Programm »MouseSet.bas« zeigt, wie es gemacht wird:

Ein ähnliches Programm wurde im AMIGA-Magazin veröffentlicht – allerdings in C. Aber in BASIC geht so etwas natürlich auch.

Karsten Schmidt/ub

```
Init:
DIM maus%(37)
LIBRARY "intuition.library"
SCREEN 2,640,256,2,2
WINDOW 2,"Mouse Set",,,2

MausData:
DATA &H0000,&H0000,&H0000,&H0000,&H0140,&H01C0
DATA &H0140,&H01C0,&H0140,&H01C0,&H0140,&H01C0
DATA &H0140,&H01C0,&H0140,&H01C0,&H7F7F,&H7FFF
DATA &H0080,&H7F7F,&H7F7F,&H7FFF,&H0140,&H01C0
DATA &H0140,&H01C0,&H0140,&H01C0,&H0140,&H01C0
DATA &H0140,&H01C0,&H0140,&H01C0,&H0140,&H01C0
DATA &H0000,&H0000

FOR i=0 TO 37
READ maus%(i)
NEXT
SetPointer WINDOW(7),VARPTR(maus%(0)),17,16,-9,-8
' Mauszeiger setzen
WHILE INKEYS="" :WEND
ClearPointer WINDOW(7)
' Mauszeiger wieder löschen

Ende:
WINDOW CLOSE 2
SCREEN CLOSE 2
LIBRARY CLOSE
END
```

MouseSet.bas

C-Programm aus dem AMIGA-Magazin – jetzt in BASIC

Die Maus läßt fragen...

Das Assemblerprogramm »MausASK.asm« ist ein Ersatz für das CLI-Kommando ASK. Nach dem Aufruf wartet es darauf, daß eine Maustaste gedrückt wird. Dabei entspricht die linke Maustaste der Eingabe von »J« für »Ja«, die rechte Maustaste »N« für »Nein«; entsprechend wird der Returncode gesetzt, so daß einer Verwendung in Ihren Batch-Dateien nichts im Wege steht.

Das Programm wurde mit dem A68k-Assembler (PD) geschrieben, ist aber auf jeden Assembler übertragbar. Assemblieren Sie

den Quelltext und speichern Sie das ausführbare Programm am besten im Verzeichnis »c« Ihrer Startdiskette oder Ihrer Festplatte, dann können Sie es in jeder beliebigen Batch-Datei einsetzen.

ub

```
loop:
btst #10,$DFF016
beq right
btst #6,$BFE001
beq left
bra loop

right:
clr.l d0
rts

left: move.l #5,d0
rts
END
```

MausASK.asm

Wer lieber mit der Maus antwortet; Ersatz für ASK

Programming is like sex: One mistake and you have to support it for life

Supervisor

Der 68000-Prozessor kennt zwei Betriebszustände: den User- und den Supervisor-Modus. Letzterer erlaubt einige zusätzliche, privilegierte Befehle, z.B. das Verändern des Status-Registers.

Manchmal möchte man zwischen beiden Modi hin- und herschalten, aber das ist nicht so einfach, weil man im User-Modus nicht auf das Status-Register zugreifen darf. Wenn man es trotzdem versucht, erhält man einen sauberen Systemabsturz (Guru-Meditation!). Mit folgendem Assemblerprogramm schafft man es trotzdem, in den Supervisor-Modus zu schalten:

```
move.l 4,a6
jsr -150(a6) ; Exec-Funktion SuperState
move.l d0,spwert
... ; weiterer Programmablauf
spwert: dc.l 0
```

Wir bedienen uns hier der Systemfunktion »SuperState«. Im Datenregister d0 erhalten wir den Stand des Stack-Pointers (SP) zurück.

Natürlich kann man vom Supervisor-Modus jederzeit wieder in den User-Modus zurückschalten:

```
move.l 4,a6
move.l spwert,d0
jsr -156(a6)
...
```

Die Kurzlistings sollten mit jedem auf dem Markt befindlichen Assembler zu assemblieren sein.

ub



Amiga Magazin hat für große Computer-Probleme einfache Lösungen.

Amiga Magazin hat für Spielernaturen und solche die es werden wollen den großen Extrateil Amiga Play.

Amiga Play prüft Spiele auf Herz und Nieren - für besonders schwere Fälle gibt es die Tips&Tricks.

Amiga Magazin hat für Anfänger, Fortgeschrittene und Profis aktuelles Know how über Grafik und Animation.

Diese Vereinbarung können Sie innerhalb von acht Tagen bei Markt&Technik Verlag AG, Postfach 1308, 8013 Haar widerrufen.

test Heft

Das Amiga Magazin Abonnement können Sie auch telefonisch unter 089 - 4613 - 369 bestellen.



Lernen Sie jetzt das Abo inklusive Testheft und allen Vorteilen kennen:

Sie zahlen für 12 Ausgaben nur 79,-DM

die Lieferung ist kostenlos per Post frei Haus,

Sie können Amiga Magazin viertel-, halb- und ganzjährig abonnieren und jederzeit wieder kündigen,

die Testausgabe zum Kennenlernen ist kostenlos und gehört Ihnen, auch wenn Sie Amiga Magazin nicht abonnieren möchten.

BESTELLCOUPON

Senden Sie mir bitte eine kostenlose Ausgabe Amiga Magazin zum Kennenlernen. Erhalten Sie 10 Tage nach Erhalt der Testausgabe von mir keine schriftliche Absage, möchte ich Amiga Magazin weiterhin beziehen. Ich erhalte Amiga Magazin dann regelmäßig frei Haus zum günstigen Jahresabopreis von 79,-DM (Auslandspreis zzgl. Porto). Das Abonnement verlängert sich automatisch um ein Jahr zu den dann gültigen Bedingungen. Ich kann jederzeit ohne Frist kündigen. Den Betrag für noch nicht gelieferte Ausgaben erhalte ich zurück.

Name, Vorname

Straße, Hausnummer

PLZ, Wohnort

Ich bezahle das Abonnement: vierteljährlich (19,75DM) halbjährlich (39,50DM) jährlich (79,-DM)

Geldinstitut

Kontonummer, BLZ

Datum, 1. Unterschrift

Diese Vereinbarung kann ich innerhalb von acht Tagen bei Markt&Technik Verlag AG, Postfach 1304, 8013 Haar widerrufen. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs. Ich bestätige die Kenntnisnahme des Widerrufsrechts durch meine 2. Unterschrift.

Datum, 2. Unterschrift

Ich erlaube Ihnen hiermit mir interessante Zeitschriftenangebote auch telefonisch zu unterbreiten (ggf. streichen)

Telefon (Vorwahl)

A 30 C 1A
AMIGA

Kontroller mit 8 MB Speicher bieten einfach mehr Harmonie!

Daß die verschiedenen Peripherie-Geräte, wie z.B. Festplatte, DAT-Rekorder, Streamer etc. dirigiert werden müssen, weiß jeder – der Kontroller macht hier die Musik. Von ihm werden nicht nur die "Grundkenntnisse" gefordert (z.B. Autoboot unter FFS), sondern auch besondere Qualifikationen, wie dt. Intuition-Oberfläche im 2.0-er Look, beliebige Partitions (z.B. PC), A-MaxII Support, automatisches Einlesen der Unit-Parameter, Disconnect / Reconnect, externer SCSI-Port für weitere Units und vieles mehr. Zu alledem ist bei bsc aber noch etwas hinzugekommen. Ab jetzt muß ein guter Kontroller on board



Platz für 8MB FAST-Speicher bieten – abschaltbar und mit 4MBit-ZIP-Bausteinen (aufrüstbar auch auf 6MB). Diese Qualifikationen, die sich bei MemoryMaster bewährt haben, sorgen jetzt für Wohlklang bei allen neuen bsc-Kontrollern!

OKTAGON508

SCSI-2 Kontroller für A500 im externen Gehäuse mit Platz für 3.5" Festplatte.

OKTAGON2008

SCSI-2-Filecard-Kontroller f. A2000/3000

AT-Bus 508/2008

Preiswerter AT-Bus-Kontroller f. A500/2000/3000. Lieferbar ab Dezember '91.

bsc – When good ideas become reality!

MEMORYMASTER

Speichererweiterung für den A2000/3000, halbe Baulänge, 4MBit-Bausteine, Ausbaustufen: 2, 4, 6 oder 8MB. Testprogramm. Unverb.Preisempf. mit 2 MB **DM 378,-**

PICTUREMANAGER

Schnittplatzprogramm für Animationen. Arbeitet u.a. mit DPaintIII™, Video Scope3D™, TurboSilver™, Sculpt4D™, Imagine™. Unverb.Preisempf.: **DM 238,-**

MULTIFACECARD2

Extrem schnelle Schnittstellenerweiterung für den Amiga® 2000/3000. Mit je zwei parallelen und seriellen Schnittstellen. Unverb.Preisempf.: **DM 398,-**

TOPSCAN

Die neue Scanner-Software, speziell für die hohen Anforderungen des Amiga®-Grafik-Anwenders. Unverb.Preisempf.: **DM 598,-**

Upgrades

Fragen Sie uns nach den günstigsten bsc - Umtausch- und Aufrüstaktionen für alle alten Kontroller auf die neue Oktagon-Serie.

THI TOOLS

Software-Paket mit Hochleistungs-BackUp für den leichten und sicheren Umgang mit Daten, Dateien und dem AMIGA®-Betriebssystem. Unverb.Preisempf.: **DM 98,-**

bsc - Produkte und Beratung erhalten Sie im guten Fachhandel: 1000 Berlin, COM Computer, 030/2190930 • 1000 Berlin 65, HD - Computer, 030/4657028 • 1000 Berlin 15, P.C.C., 030/8837707 • 1000 Berlin 19, S & M Elektronik GbR mbH, 030/3218351 • 1000 Berlin 44, W & L - Computer GbR, 030/6227371 • 2000 Hamburg 76, Joystick GmbH, 040/25145 92 • 2060 Bad Oldesloe, Joël Datentechnik, 04531/1521 • 2400 Lübeck 1, Joystick, 0451/705165 • 2800 Bremen 1, ECS GmbH, 0421/611430 • 2802 Ottersberg 1, Dodenhof GmbH, 04297/3433 • 3000 Hannover 1, ComData, 0511/9904230 • 3000 Hannover, DART, 0511/858260 • 3000 Hannover 1, HD - Computer, 0511/8094484 • 3070 Nienburg, Text & Data, 05021/5416 • 3181 Röhren, ADC - Andrea Dohm, 05367/1235 • 3300 Braunschweig, BBM Datensystem GbR, 0531/72844 • 3578 Schwalmstadt, Treysa - Zebra Grafikservice, 06691/23437 • 4500 Dortmund, MAC-Soft, 0231/161817 • 4630 Bochum, multi-RAK, 0234/795278 • 5100 Aachen, Software Corner, 0241/533131 • 5300 Bonn 1, Ariza Elektronik, 0228/662135 • 5603 Wülfrath, Rainbowdata GbR, 02058/1366 • 6000 Frankfurt 1, GTI Software Boutique, 069/233561 • 6370 Oberursel, GTI Home Computer Centre, 06171/730 48 • 7039 Weil, Unger & Schumm, 07157/62481 • 8000 München 70, Computer Corner, 089/7144395 • 8000 München, HAL, 089/874 054 • 8000 München 80, Modl Plus Foto, 089/4801650 • 8031 Gilling, Miky Wennigatz, 08105/24540 • 8098 Pfaffing, ADS GmbH, 08935/45296 • 8480 Weiden/Obf., Hölst-electronic, 0961/35051 • 8541 Rohr-Regelsbach, Werbeverlag Esser, 09122/82563 • 8700 Würzburg, Amtek G.Schneider, 0931/887124 • 8700 Würzburg, Top3 Markt, 0931/93012 • 8858 Neuburg/Donau, Donausoft, 08431/49798 • 8870 Günzburg, Amiga-Shop-Günzburg, 08221/6227 • 8900 Augsburg, Ziegler & Föh, 0821/814453 • **Distributor Schweiz:** 2502 Biel, Swissoft AG, 032/231833 • • Weitere Händler und Informationen bekommen Sie direkt von uns. • • Händler Anfragen / Dealer inquiries: **bsc büroautomation AG • Pf.40 03 68 • 8000 München 40**



bsc büroautomation AG
München