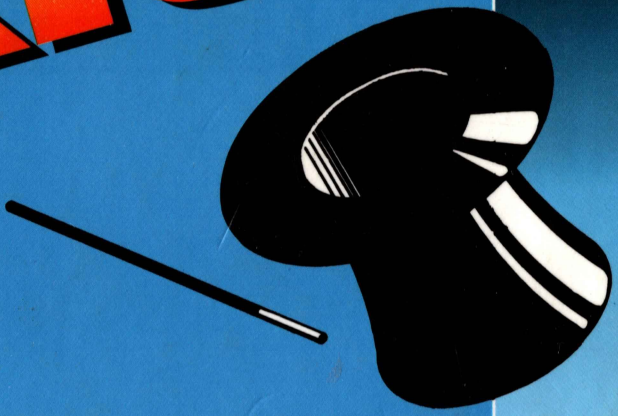
BLEEK · MAELGER · WELTNER

AMIGA

TIPS & TRICKS



DATA BECKER

Bleek
Maelger
Weltner

Amiga
Tips & Tricks

DATA BECKER

5. überarbeitete und erweiterte Auflage 1989

ISBN 3-89011-211-0

DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Umschlaggestaltung: Werner Leinhos
Textverarbeitung und Gestaltung: Sabine Wingerath

Text verarbeitet mit Word 4.0, Microsoft
Ausgedruckt mit Hewlett Packard LaserJet II
Druck und Verarbeitung: Mohndruck, Gütersloh

Vorwort zur 5. Auflage

30.000 Exemplare sind verkauft und wieder einmal haben wir uns zusammengesetzt, um neue, bessere und noch trickreichere Kniffe zu finden, mit denen wir neuen und alten Lesern brandheiße Informationen bieten können. Diese vollkommen neu überarbeitete Auflage enthält Tips & Tricks aus noch mehr Themengebieten als es bisher der Fall war. Neben den jetzt vollkommen aktualisierten Informationen zum Betriebssystem 1.3 haben wir nun auch das brandheiße GFA-BASIC mit in den Kreis der Programmiertricks aufgenommen.

Für die Erstellung der Hardware-Schaltungen danken wir ganz herzlich Frank Dzubilla, der in seiner zuverlässigen Art dafür gesorgt hat, daß wir bei der Arbeit mit dem Amiga mehr Komfort haben und viele Leser diesen Vorteil jetzt auch nutzen können.

Hinweis: Die für diese Auflage verwendeten Kickstart- und Workbench-Versionen sind die allerneuesten auf dem Markt. Wir verwendeten als Kickstart die Version 34.5, die unter der Bezeichnung 1.3 ab jetzt bei allen Commodore-Vertragshändlern als Kickstart-Diskette oder als ROM-Version zu beziehen sind, und als Workbench die Version 34.20, die auch unter der gleichen Bezeichnung 1.3 überall zusammen mit einer Dokumentation käuflich zu erwerben ist.

Dazu gehört eine Extras-Diskette mit vielen neuen Utilities, die wir auch in diesem Buch erklären und Anwendungen dafür vorstellen werden. Für die Programmierer unter Ihnen gibt es auch noch neue Include-Files. Das ist besonders wichtig, da einige Strukturen ergänzt und andere optimiert wurden.

Wer seinen Amiga wirklich ausnutzen und aus der großen Fülle des Möglichen schöpfen möchte, der liegt mit diesem Buch genau richtig! Das Bestreben dieses Buches ist es, sowohl dem

Hobby- als auch dem Profiprogrammierer viele Tips und Tricks über seinen Rechner, den Amiga, zu verraten und zu erklären, wie diese Dinge ermöglicht werden.

Oftmals sind nur wenige Systemkenntnisse nötig, um selbstgeschriebene Programme zu verkürzen, ihnen Profil zu geben, sie zu beschleunigen oder durch ganz neuartige Möglichkeiten aufzuwerten. Hier setzt dieses Buch ein: Auf den vor Ihnen liegenden Seiten verraten wir Ihnen nicht nur viele Detailkenntnisse über das Betriebssystem, den AmigaBASIC-Interpreter und das Disk Operating System, Sie finden auch eine wahre Fülle kleiner und deshalb übersichtlicher Programme, die diese neuen Erkenntnisse verdeutlichen. Die meisten dieser Programme lassen sich problemlos in eigene, selbstgeschriebene Programme übernehmen und somit sofort anwenden.

Diese Konzeption, die dem Profi das Hintergrundwissen und dem Anfänger fix und fertige Problemlöser liefert, hat sich bestens bewährt: Bereits die erste Auflage von "Amiga Tips & Tricks" bot aufregende Programme: Im wohl ersten deutschsprachigen Buch wurde der BASIC-Programmierer intensiv mit der Benutzung der Systemroutinen vertraut gemacht - Dinge wurden möglich, an die man gar nicht gedacht hatte. Heute nun wünschen wir Ihnen viel Spaß mit der komplett neubearbeiteten und um vieles reichhaltiger gewordenen Neuauflage!

Großhansdorf, im Mai 1989

Wolf-Gideon Bleek
Stefan Maelger
Tobias Weltner

Inhaltsverzeichnis

1. Einleitung

Das Wort Amiga hat es in sich! Es steht für beeindruckende Grafik, klaren Sound, gute Programme und ein Betriebssystem der unerschöpflichen Möglichkeiten. Alles dies liegt nun vor einem, doch man kann es nicht nutzen. Wo soll man anfangen? So lautet eine oft formulierte Frage. Dabei ist es klar! Bei den Tips & Tricks fangen wir hier an.

Wir möchten Ihnen aus möglichst vielen Themengebieten versteckte und bisher unbekannte Informationen bieten. Wir wollen kurze, aber wirkungsvolle Programme vorstellen, die nicht das tägliche Einerlei darstellen. Wir haben lange an dieser Supermaschine gearbeitet und alle unsere Erfahrungen zusammengetragen, um sie Ihnen zu präsentieren. In dieser Einleitung wollen wir Ihnen einen kurzen Wegweiser durch alle Kapitel geben, damit Sie schon jetzt genau wissen, welche Informationen Sie wo bekommen. In Kapitel 2, dem CLI-Kapitel, haben wir unsere besten Erfahrungen zum Command Line Interpreter zusammengetragen. Welche Probleme treten bei der Arbeit mit dem CLI auf, welche neuen Befehle gibt es, wie arbeite ich mit Batchdateien und was bringt z.B. der Mount-Befehl im Detail?

Dann steigen wir in die Welt des AmigaBASIC, der kostenlos mitgelieferten Programmiersprache von Microsoft. Sie lernen, auch die internen Funktionen des Betriebssystems zu nutzen und erfahren vieles über Grafik und Programmierung. Hier haben wir drei unser Bestes gegeben.

Unter der Voraussetzung, daß wir mit Ihnen einen aktiven Programmierer vor uns haben, können Sie im vierten Kapitel alles Wissenswerte zum Thema anwenderfreundliche und besonders professionelle Gestaltung der eigenen Programme erfahren. Wolf-Gideon Bleek und Sie werden mit Hilfe der Intuition.Library aus dem Vollen schöpfen und Programme schreiben, die den Preferences in der Bedienerfreundlichkeit in nichts nachstehen.

Wissen Sie eigentlich, wie die Arbeit mit der Diskettenstation funktioniert? Das ist eine wichtige Frage, denn jede Datensicherung im Programm und außerhalb läuft darüber ab! Doch keine Angst, wenn Sie nicht alles zu diesem Gebiet wissen. Die nötigen Grundlagen wird Ihnen leichtverständlich Tobias Weltner erklären, der sich im DOS so gut ausgekennt, wie keiner von uns.

Ohne vor noch so komplizierten Aufgaben zurückzuscheuen, hat sich Stefan Maelger auf den Dschungelpfad durch das AmigaBASIC gemacht und erstaunliche Dinge herausgefunden. Wissen Sie, wie AmigaBASIC seine Programme oder Variablen organisiert? Können Sie sich vorstellen, was man mit Eingriffen alles erreichen kann? Lesen und staunen Sie!

Jeder hat tagtäglich mit der Workbench zu tun. Ob es nur die kleinen Aufgaben der File-Pflege sind oder auch die großen und aufwendigen Arbeiten. Wolf-Gideon Bleek hat sich lange damit beschäftigt und sogar einen Weg herausgefunden, wie man die englische Benutzeroberfläche zu einer deutschen und ganz individuellen macht. Wer mehr wissen will und sich nicht mit dem Einfachen zufrieden gibt, ist hier genau richtig.

Die Icons, ein weiteres Spezialgebiet von Wolf-Gideon Bleek, haben im achten Kapitel besondere Aufmerksamkeit gefunden. Welche Daten im .info-File stehen und wie man sie nutzt oder verändert, ist ein großes Themengebiet in diesem Kapitel. Sie lernen die Bedienung des IconMerge von der Extras-Diskette kennen und auch, wie man mit eigenen Programmen die Icons modifiziert oder vollkommen umkrempelt. Fehler machen wir alle, doch wie kann man sie verhindern? Diese wirklich wichtige Frage wird durch die "Kontrollierte Fehlerbehandlung" erörtert und auch gelöst. Somit sind Ihre Programme vor Bedien- und Systemfehlern gefeit. Endlich!

Nachdem Sie nun sowohl fehlerfrei als auch geschickt unter Ausnutzung aller Libraries programmieren, steht der nächste wichtige Punkt auf dem Papier. Ein weiteres Ziel der professionellen Programmierung ist das Verkürzen von Routinen und das

damit verbundene Beschleunigen von Programmen. Hier hat sich Wolf-Gideon Bleek wie auch im vorhergehenden Kapitel Gedanken gemacht, und herausgekommen ist die effektive Programmierung, mit deren Hilfe Sie Ihre Programme um bis zu 20% in der Geschwindigkeit steigern können. Wer hätte das gedacht?

Nachdem nun so viel über AmigaBASIC und seine Programmierung gesagt wurde, meldet sich wieder Stefan Maelger zu Wort, um mit interessanten und ausgeklügelten Beispielen die Programmierung in Maschinensprache über BASIC darzubieten. Das Problem des Ladens und Zusammenfügens von Maschinenprogrammen wird genauso besprochen wie auch viele kleine und größere Beispiele, die Funktionen ermöglichen, die aus BASIC einfach unmöglich sind!

Eine wichtige Schnittstelle zur Außenwelt stellt die I/O-Kommunikation dar. Sie ermöglicht die Ausgabe auf dem Drucker, das Ansprechen des Trackdisk.Devices und die Besorgung von Speicher. Alles dies sind elementare Dinge, die man zur besseren Programmierung einfach braucht. Tobias Weltner weiß, wie man sie anwendet und hat sie ausführlich beschrieben.

Nun wird es ernst. Es geht an das Innere des Amiga. Wir beschäftigen uns mit der Hardware. Wußten Sie, daß der laute Lüfter am 2000er leiser gemacht werden kann? Haben Sie ein externes Laufwerk, das unbedingt abgeschaltet werden muß? Oder haben Sie störenden Speicher, ohne den so manches Programm laufen würde? Zu all diesen Problemen und noch einigen Raffinessen mehr haben Wolf-Gideon Bleek und Stefan Maelger Informationen zusammengetragen, die es Ihnen heiß und den Amiga kalt werden lassen!

Kennen Sie die DATA BECKER Software? Wissen Sie auch, wie man damit umgeht? Sie haben zwar das Handbuch gelesen, doch dies erklärt nur alle Funktionen, nicht die komplexe Anwendung am Arbeitstisch. Mit den Tips eines erfahrenen Autors wird es Ihnen leichtfallen, mit BECKERtext oder TEXTOMAT zu arbeiten!

Lange Arbeit, viele Erfahrungen. Wo bringt man dies unter? Das haben wir uns alle gefragt und daraus ist das 13. Kapitel entstanden: "Sammelsurium der Tips & Tricks". Wolf-Gideon Bleek hat sich die Aufgabe gestellt, alle als "Kleinigkeiten" abgetanen Kniffe, Tips, Hilfen und natürlich Tricks aufzuschreiben und in dieses Buch aufzunehmen. Jetzt können auch Sie auf jahrelange Arbeit zurückgreifen. Oft helfen Kleinigkeiten mehr als große Programme!

Hurra, es ist soweit! Tobias Weltner und Stefan Maelger mit ihrem Amiga 1000 haben das neue Betriebssystem Version 1.3 getestet. Herausgekommen ist ein ganzes Kapitel voller speziell auf diese Version abgestimmter Informationen. Was hat sich geändert? Wie kompatibel ist das neue Kickstart wirklich? Dies und vieles mehr finden Sie hier.

Als Abschluß und Abrundung des neuesten Tips & Tricks zum Amiga werden Sie im letzten Kapitel mit der Bedienung und Anwendung der mitgelieferten Utility-Programme vertraut gemacht. Es gibt haufenweise neue Parameter! Viele kleine Hilfen sind dazugekommen! Wolf-Gideon Bleek hat aufmerksam und genau hingeschaut und alles Wichtige für Sie ausprobiert. Hier ist es!

Nun haben Sie einen runden Überblick über das neueste Tips & Tricks. Wir alle hoffen, daß Sie Freude haben und viele Entdeckungen mit uns zusammen machen werden. Sie halten eine Fundgrube an Informationen zu allen möglichen Themengebieten in Ihren Händen. Wie Sie alles nutzen können, steht hier. Sie brauchen nur noch zu lesen![1]

1 Aus drucktechnischen Gründen ist es uns leider nur möglich, maximal 70 Zeichen pro Zeile zu drucken. Aus diesem Grund kann es bei Programmen zu erzwungenen Zeilenumbrüchen kommen. Wir bitten um Verständnis.

2. CLI - Command Line Interpreter

CLI steht für *Command Line Interpreter*. Es ist eine Benutzeroberfläche, die sich ohne die vielen bunten Symbole (Icons) und ohne Maus ganz auf die Tastatur verläßt. Das CLI kennt im Moment ca. 50 verschiedene Befehle (Workbench Version 1.2), es werden aber in Zukunft noch viele weitere dazu kommen.

Das CLI arbeitet eng mit AmigaDOS, dem Disk Operating System, zusammen. Viele Spezialbefehle erleichtern den Umgang mit Disketten; einige Funktionen lassen sich sogar von der Workbench aus überhaupt nicht lösen. Das CLI wird normalerweise vom Benutzer via Intuition aufgerufen. Es ist aber auch möglich, beliebige CLI-Kommandos von BASIC-Programmen aus zu benutzen (dasselbe gilt für "C").

2.1 Fragen zum CLI

Nach einigen Gesprächen mit frischgebackenen Amiga-Besitzern stellten wir fest, daß einige Fragen immer wieder auftauchen. Da wir davon ausgehen, daß auch viele von Ihnen sich diese Fragen stellen werden, geben wir Ihnen hier Antwort.

Frage 1
Wie gelange ich zum CLI?

Der Command Line Interpreter befindet sich serienmäßig auf jeder Workbench-Diskette. Es gibt eine Reihe von Wegen, um ans CLI heranzukommen:

a) Mit Intuition. Dies ist der Normalfall. Sie haben Ihr System gebootet, die Workbench-Diskette befindet sich im Laufwerk. Vor Ihnen liegt nun die tiefblaue "Workbench-Screen". Folgen Sie diesen Schritten:

- *Klicken Sie das Disk-Icon der Workbench-Disk an.*

  Es wird sich ein Fenster namens "Workbench..." mit allerlei Dingen darin öffnen.

- *Klicken Sie die Schublade "System" an.*

  Es wird sich ein Fenster namens "System" öffnen, wiederum randvoll gefüllt mit den Errungenschaften unserer heutigen Leistungsgesellschaft, vor allem aber mit einem Icon namens "CLI", das entweder "1>" darstellt (Version 1.1) oder ein nettes kleines Fenster mit 1> darin (Version 1.2 und darüber).

  Sollten Sie kein CLI-Icon vorfinden, so liegt das daran, daß bei Ihrer Workbench-Diskette noch die Kindersicherung eingeschaltet war. Klicken Sie in diesem Fall das Icon "Preferences" im "Workbench..."-Fenster an, und schalten Sie im Feld CLI von OFF auf ON. Speichern Sie am besten das Ergebnis mit SAVE gleich wieder ab. Nun müssen Sie das "System"-Fenster schließen und erneut öffnen. Und siehe da: Ein CLI-Icon!

- *Klicken Sie das CLI-Icon an.*

  Es wird unverzüglich ein Fenster namens "New CLI" erscheinen, das Sie in alle Richtungen vergrößern und verkleinern können, das jedoch keinen Ausknips-Knopf in der linken oberen Ecke besitzt. Sie haben es geschafft! Sie haben Ihr eigenes CLI!

b) Mit AmigaDOS. AmigaDOS verfügt über den Befehl "Execute", mit dem beliebige CLI-Befehle ausgeführt werden können. AmigaDOS wiederum kann über die systeminternen Libraries angesprochen werden. Auf diese Art und Weise kommen selbst AmigaBASIC und "C" ans CLI (siehe Beispielprogramme in diesem Buch).

c) Noch raffinierter! Ein ganz einfacher Weg ist der folgende: Sie haben gerade Ihr System eingeschaltet. Die Kickstart-Disk ist bereits erfolgreich geladen worden, nun flimmert eine gigantische Hand mit einer Workbench-Disk darin auf dem Bildschirm. Bis jetzt ist alles Routine. Aber jetzt! Legen Sie die Workbench-Disk ins Laufwerk. Die Hand ver-

schwindet, das System bootet. Geben Sie nun rasch "Tobi ist lieb!" über die Tastatur ein! Nun brauchen Sie nur noch zu warten, bis das Laufwerk zum Stillstand gekommen ist. Ist die rote Lampe erloschen, dann nehmen Sie die Workbench-Disk rasch aus dem Laufwerk. Löschen Sie nun "Tobi ist lieb!" wieder mit der BACKSPACE-Taste. Sobald der letzte Buchstabe getilgt ist, treten eine Reihe Fehlerrequester auf. Beachten Sie sie einfach nicht, sondern klicken Sie resolut immer wieder auf CANCEL. Endlich erscheint

```
1>
```

auf dem Bildschirm - das CLI! Geben Sie schnell noch

```
1> loadwb
```

ein, und das CLI steht zu Ihrer vollen Verfügung!

Frage 2
Wie kriege ich das CLI wieder weg?

Das CLI-Fenster hat keinen Ausschalter. Es löst sich mit folgender Eingabe von selbst in Wohlgefallen auf:

```
1> endcli
```

Haben Sie von einem CLI aus Programme gestartet, dann bleibt das CLI-Fenster allerdings solange erhalten, wie die Programme laufen.

Frage 3
Ich habe keine Schreibmaschine, aber einen an meinen Amiga angeschlossenen Drucker. Kann man da nichts machen?

Na klar! Das CLI ist der geborene Problemlöser. Geben Sie folgendes CLI-Kommando ein:

```
1> copy * to prt:
```

wobei * das Symbol des aktiven CLI-Fensters ist. Nach dieser Eingabe verschwindet das gewohnte CLI-Prompt "1>", lediglich der Cursor hält Ihnen die Treue. Jede Tastatureingabe wird nun nach Druck der RETURN-Taste auf den Drucker ausgegeben - quasi eine Schreibmaschine mit einzeiligem Korrekturspeicher!

Aus dem Schreibmaschinen-Modus kommen Sie wieder heraus, wenn Sie gleichzeitig die CTRL- und \-Taste drücken. Je nach Lust und Laune können Sie übrigens auch Text in ein anderes Fenster kopieren...

```
1> copy * to CON:10/10/300/100/Kopie-Text
```

...oder sich selbst wiederholen...

```
1> copy * to *
```

Frage 4

Ich besitze nur ein Diskettenlaufwerk. Jedesmal, wenn ich einen CLI-Befehl verwende, muß ich kurz die Workbench-Diskette einlegen. Kann man das nicht verhindern?

Jedes CLI-Kommando ist auf der Workbench-Disk als kleines Programm im Directory "c:" abgespeichert. Wenn Sie nun einen CLI-Befehl verwenden, lädt der Amiga diesen normalerweise jedesmal von der Workbench-Disk nach. Dadurch spart man natürlich eine Menge Speicherplatz, denn die CLI-Kommandos belegen so keinen Systemspeicher. Auf der anderen Seite muß man laufend Disketten wechseln, wenn man nur über ein Laufwerk verfügt. Wenn Sie über genügend Speicher verfügen, können Sie aber alle (oder selektierte) CLI-Befehle ins RAM kopieren. Das geht so:

```
1> makedir ram:c
1> copy sys:c to ram:c
1> assign c: ram:c
```

Zunächst wird in der RAM-Disk ein Unter-Directory namens "c" angelegt. Anschließend werden alle CLI-Befehle in dieses Directory kopiert. Schließlich wird dem Amiga mitgeteilt, daß

das Kommando-Directory c: nun auf der RAM-Disk liegt. Ist Ihnen der Speicher Ihres Rechners doch noch zu schade, dann können Sie sich auf die meistbenutzten CLI-Kommandos beschränken. Zum Beispiel so:

```
1> makedir ram:c
1> copy sys:c/copy to ram:c
1> copy syss12:c/dir to ram:c
1> copy sys:c/list to ram:c
(...)
1> assign c: ram:c
```

Wollen Sie wieder zurück zum Workbench-CLI, dann funktioniert das in jedem Fall (sofern Sie die Workbench-Disk in das eingebaute Laufwerk 0 legen):

```
1> df0:c/assign c: df0:c
```

Nach Beendigung Ihrer Vorhaben empfiehlt es sich, das RAM-CLI wieder zu löschen, um Speicherplatzreserven zurückzubekommen:

```
1> delete ram:c#?
1> delete ram:c
```

Frage 5
Wie lassen sich CLI-Kommandos unterbrechen?

CTRL-C unterbricht einen Befehl. CTRL-D veranlaßt einen Execute-Befehl, so schnell wie möglich den Programmablauf zu unterbrechen. CTRL-E und CTRL-F werden nur in ganz besonderen Fällen gebraucht.

Frage 6
Gibt es einen Joker, vergleichbar dem * bei alten Commodore-Rechnern?

Ja, es handelt sich um die Symbolkombination #?. Das Zeichen * repräsentiert ja bereits das aktuelle CLI-Fenster.

```
1> delete ram:#?
```

löscht die gesamte RAM-Disk.

```
1> run amig#?
```

funktioniert jedoch beispielsweise nicht, denn der Amiga weiß nicht, welches Programm ausgeführt werden soll. Es könnten ja mehrere Programme existieren, die mit der Buchstabenkombination "amig" beginnen.

Frage 7
Gibt es eine Möglichkeit, die Befehlssyntax eines bestimmten CLI-Befehls herauszufinden?

Fast alle CLI-Befehle verfügen über eine Eingabehilfe. Falls Sie also nicht mehr genau wissen, wie ein spezieller Befehl aufgerufen wird, dann geben Sie einfach den Befehlsnamen gefolgt von einem Leer- und einem Fragezeichen ein. Zum Beispiel:

```
1> list ?
```

Das Ergebnis ist:

```
DIR,P=PATH/K,KEYS/S,DATES/S,NODATES/S,TO/K,S/K,SINCE/K,UPTO/K,
QUICK/S:
```

Na, alles klar? *DIR* steht für ein Directory, kann aber auch weggelassen werden (dann wird das augenblickliche Directory ausgegeben). Alle weiteren Angaben enthalten neben dem Optionswort eine Bedingung:

- */A:* Dieses Argument muß angegeben werden.
- */K:* Dieses Argument muß in Verbindung mit einem Parameter gegeben werden.
- */S:* Dieses Argument steht für sich allein.

So sind die folgenden Kommandos möglich:

```
1> list df0: keys nodates
```

Gibt das Inhaltsverzeichnis "df0:" mit den jeweiligen Anfangsblöcken, jedoch ohne Datumsangabe aus.

```
1> list df0: since 04-Oct-86 upto today
```

Gibt die Programme des Inhaltsverzeichnisses "df0:" aus, die zwischen dem 4. Oktober 1986 und heute geschrieben worden sind.

Frage 8
Ich verfüge über ein Laufwerk und möchte ein Programm kopieren. Wie funktioniert das?

Erste Möglichkeit: Es handelt sich um ein kleines Programm. Laden Sie es zunächst in die RAM-Disk:

```
1> copy programm to ram:
1> copy c/copy to ram:
```

Der Copy-Befehl wurde in der zweiten Anweisung ebenfalls kopiert, um zu verhindern, daß die Workbench nachgelegt werden muß. Legen Sie nun die Zieldiskette ins Laufwerk. Anschließend wird das Programm zurückkopiert:

```
1> ram:copy ram:programm to df0:
```

Legen Sie nun bitte wieder die Workbench-Disk ein. Die RAM-Disk muß nur noch gelöscht werden, und schon sind wir fertig:

```
1> delete ram:#?
```

Eine andere Methode bedient sich der Intuition-Icons. Sie müssen dazu zunächst die Originaldiskette einlegen und das Disk-Icon anklicken. Sobald das Icon des gewünschten Programms erscheint, legen Sie die Zieldiskette ein. Öffnen Sie auch diese durch Anklicken des Disk-Icons. Nun können Sie das Icon des Originalprogramms mit Hilfe der Maus ins Fenster der Zieldiskette bewegen. Der Rest geschieht automatisch per Requester: Die Disketten müssen ein paarmal gewechselt werden.

Achtung: Es gibt Programme, die gar kein Icon besitzen. Sie erscheinen also auch nicht als Symbol in einem Intuition-Fenster. Sie können aber solch einem Programm ein Icon beschaffen! Legen Sie dazu die Workbench-Disk ein. Die folgenden Zeilen sind nötig:

```
1> copy df0:clock.info to ram:
1> rename ram:clock.info as ram:programm.info
1> copy c/copy to ram:
```

Jetzt legen Sie die Diskette ein, auf der sich Ihr Originalprogramm befindet. Geben Sie nun ein:

```
1> ram:copy ram:programm.info to df0:
```

Nun hat Ihr Programm (hier namens programm) ein Icon. Legen Sie wieder die Workbench ein, und löschen Sie die RAM-Disk:

```
1> delete ram:#?
```

Frage 9
Ich möchte gern eine Liste aller CLI-Befehlsworte auf den Drucker ausgeben. Geht das?

Das funktioniert mit einer einfachen Befehlskombination:

```
1> list quick sys:c to prt:
```

Die Option "quick" bewirkt, daß nur die Befehlsnamen ausgegeben werden. Erschaffungsdaten, Uhrzeit, Schutzstatus sowie Filegröße werden nicht ausgedruckt. Die CLI-Befehle selbst stehen im Unter-Directory c: des System-Directories sys:. Schneller geht es, wenn Sie die Multitasking-Fähigkeiten Ihres Amigas ausnutzen:

```
1> run list quick sys:c to prt:
```

Hier wird ein weiterer Task geöffnet, der die Druckerausgabe bewerkstelligt. Sie können also gleich mit anderen Sachen weiterarbeiten, während Ihr Amiga quasi im Hintergrund die Befehlsworte ausgibt.

Frage 10
Ich habe zwei Laufwerke und möchte ein Programm kopieren. Ein leichtes Unterfangen?

Sicher. Es genügt eine CLI-Zeile:

```
1> copy df0:originalprogramm to df1:
```

Hierbei muß sich das Originalprogramm in Laufwerk 0 im Directory df0: befinden. Falls Ihr Programm ein Icon besitzt, muß auch dieses kopiert werden:

```
1> copy df0:originalprogramm.info to df1:
```

Natürlich können Sie auch per Intuition das Programm-Icon direkt von einer Diskette zur anderen schieben (siehe Frage 10).

Frage 11
Ich möchte eine ganze Diskette kopieren. Wie geht das?

Mit dem Befehl "diskcopy". Es spielt dabei keine Rolle, ob Sie über einen oder mehrere Drives verfügen. Achtung: Um ein ungewolltes Löschen der Original-Disk auf jeden Fall zu verhindern, sollten Sie den Schutzpin an der Seite der Originaldiskette zumindest für die Dauer des Kopierens nach oben schieben, falls dies noch nicht getan wurde.

Sie besitzen ein Laufwerk:

1. Legen Sie die Workbench-Diskette ein.
2. Tippen Sie den CLI-Befehl ein:

```
1> diskcopy from df0: to df0: name "kopie"
```

Nun erscheint die Aufforderung, die Quell-Diskette (SOURCE) einzulegen. Kommen Sie dem nach. Nach einer Weile muß die Ziel-Diskette (DESTINATION) eingelegt werden, und nach ein paar weiteren Wechseln haben Sie es geschafft.

Sie besitzen zwei Laufwerke:

1. Legen Sie die Workbench-Diskette ein.
2. Tippen Sie den CLI-Befehl ein:

```
1> diskcopy from df0: to df1: name "kopie"
```

Stecken Sie nun gemäß der Aufforderung die Quell-Disk in Laufwerk 0, die Ziel-Disk in Laufwerk 1. Die Disketten brauchen natürlich nicht mehr gewechselt zu werden.

Frage 12
Was ist eine Startup-Sequence und was kann man mit ihr machen?

Die Startup-Sequence ist eine Liste von CLI-Befehlen, die ganz zu Anfang beim Booten des Systems ausgeführt wird. Wie das aussieht, können Sie sich leicht veranschaulichen:

```
1> execute s/startup-sequence
```

Sie können sich diese CLI-Befehle auch einmal anschauen:

```
1> type s/startup-sequence
```

Wenn Sie Lust haben, können Sie sich auch eine eigene Startup-Sequenz schreiben. Sie sollten jedoch beachten, daß das Kommando loadwb unbedingt übernommen wird, da sonst das Intuition-Icon-System nicht aktiviert wird. Sollten Sie dann einmal den CLI durch endcli verlassen, stehen Sie quasi vor zugeschlagener Haustür, denn es wären keine Icons da, nur ein leerer Bildschirm. Eine eigene Startup-Sequenz kann man über das Kommando "ed" erstellen:

```
1> ed s/startup-sequence
```

Bei Version 1.2 der Workbench sehen Sie nun:

```
echo "Workbench Diskette (Version 1.2/33.43)"
echo " "
echo "(Datum und Uhrzeit mit 'Preferences' einstellbar)"
if EXISTS sys:system
     path sys:system add
endif
BindDrivers
setmap d
LoadWb
endcli > nil:
```

Mit den Cursortasten können Sie den Cursor bewegen. Auf Druck der ESC-Taste landen Sie in der Kontrollzeile. Ein "d" löscht die Zeile, in der sich der Cursor zuletzt befand. Löschen Sie beispielsweise die folgenden Anweisungen:

```
setmap d
endcli > nil:
```

Nun speichern Sie die Sequenz wieder durch Druck auf die ESC-Taste und anschließend "x". Probieren Sie die neue Sequenz doch gleich mal aus!

```
1> execute s/startup-sequence
```

Wie Sie sehen, haben Sie wieder eine amerikanische Tastaturbelegung, und das CLI-Fenster ist auch nicht verschwunden.

Frage 13
Kann der Amiga im CLI sprechen?

Sicherlich, wenngleich man auch keine Parameter verändern kann. Das Kommando heißt "say" und wirkt wie ein Print-Kommando. Leider ist es unmöglich, Programm-Files aussprechen zu lassen. Lediglich unmittelbar nachfolgender Text wird verlesen:

```
1> say tobi is a real nice guy!
```

Interessant ist dieser Befehl auch in Zusammenhang mit der Startup-Sequenz (Frage 13)! Stellen Sie sich vor, Ihr Amiga begrüßt Sie jedesmal nach dem Einschalten mit einem netten "Guten Tag!".

Frage 14
Wie kann ich ein C-Listing auf den Drucker ausgeben?

Am besten geschieht dies mit dem CLI-Befehl type. Ein Beispiel: Sie haben ein C-Listing, erstellt wahrscheinlich mit ed unter dem Namen test.c auf df1:. Geben Sie nun ein:

```
run type df1:test.c to prt: opt n
```

Mit "run" nutzen Sie die Multitasking-Fähigkeit des Amiga - während der Drucker lustig rattert, können Sie schon wieder etwas anderes fabrizieren. Der Zusatz "opt n" sorgt dafür, daß der Ausdruck des C-Listings mit Zeilennummern versehen wird. Dies ist recht hilfreich bei der Fehlersuche.

Frage 15
Wie nutze ich die Multitasking-Fähigkeiten des Amigas bei der täglichen Arbeit mit dem CLI?

Normalerweise verarbeitet das CLI immer einen Befehl nach dem anderen; von Multitasking kann also keine Rede sein. Man muß hier ganz klar sagen, daß das CLI selbst nicht mehrere Aufgaben gleichzeitig lösen kann. Durch das Multitasking-Betriebssystem des Amiga ist es aber möglich, mehrere Single-Task-CLIs gleichzeitig ablaufen zu lassen.

Wollen Sie also beispielsweise das Inhaltsverzeichnis der System-Diskette ausdrucken, anschließend einen Text editieren und schließlich einen Satz vom Amiga aussprechen lassen, so geschieht dies normalerweise so:

```
1> list sys: to prt:
1> ed text
1> say hello user
```

Schneller geht es, bedient man sich mehrerer CLIs:

```
1> run list sys: to prt:
1> run ed text
1> say hello user
```

Der vorgestellte Befehl "run" bewirkt, daß die nachfolgende Befehlskombination einem neuen CLI übertragen wird. Das ursprüngliche CLI hat dann damit nichts mehr zu schaffen und kann schon an die nächste Aufgabe gehen, ohne auf die Erledigung der ersten Aufgabe zu warten.

Einschränkend muß an dieser Stelle jedoch gesagt werden, daß auf jeden Fall vermieden werden sollte, daß zwei CLIs gleichzeitig auf ein und dasselbe Laufwerk (oder den Drucker) zurückgreifen. Im Falle des Disk Drives teilen sich dann nämlich beide CLIs Rechenzeit. Das hat zur Folge, daß die gesamte Operation länger dauert, als wenn sie nacheinander erfolgt wäre.

Eine weitere Möglichkeit, mehrere Aufgaben gleichzeitig lösen zu lassen, ist das Öffnen mehrerer CLIs durch den Befehl "NEWCLI". Dadurch bekommt der Anwender eine komplette weitere Eingabeschnittstelle. Diese Methode eignet sich besonders, wenn man nicht kurzfristig CLI-Befehle ausführen lassen möchte, sondern über einen längeren Zeitraum mit mehreren Funktionen des CLI beschäftigt ist. Das folgende Beispiel macht dies deutlich:

```
1> newcli
1> list df0: quick
2> type files opt h
```

Hier wurde ein neues CLI geöffnet, und alle Filenamen des Inhaltsverzeichnisses df0: wurden ausgegeben. Anschließend wurden von dem zweiten, neuen CLI aus die Datei-Inhalte ausgegeben. Dabei können Sie nun Dateinamen für Dateinamen aus dem ersten CLI-Fenster lesen und im zweiten Fenster ungestört arbeiten, ohne daß die Liste der Namen zerstört würde.

Eine weitere Möglichkeit des NEWCLI-Befehls sind seine Optionen. Man kann nämlich die Dimensionen des neuen CLI-Fensters selbst festlegen. Dies geschieht so:

```
1> newcli con:0/10/639/100/neuesCli
```

Die ersten beiden Zahlen geben x- und y-Koordinate der linken oberen Fensterecke an, die beiden folgenden Zahlen legen Breite und Höhe fest. Dadurch lassen sich neue CLI-Fenster geschickt legen, ohne andere zu verdecken. Arbeitet man mit mehreren CLIs, ist es lediglich nötig, jeweils die rechte obere Ecke eines jeden Fensters unbedeckt zu lassen. So kann man ein jedes Fenster bei Bedarf in den Vordergrund holen, um damit zu arbeiten.

Frage 16
Welche Möglichkeiten bietet der Amiga, Texte ausgeben auszugeben?

Die simpelste Möglichkeit ist das folgende Statement:

```
1> copy * to prt:
```

Sie finden nähere Informationen hierzu bei Frage 4. Etwas komfortabler ist der eingebaute CLI-Editor "ed". Wenn Sie also einen kurzen Brief schreiben wollen, bietet sich diese Möglichkeit an:

```
1> run ed ram:brief
```

Sofort erscheint das Ed-Fenster, und Sie können mit der Ausarbeitung Ihres Briefes beginnen. Sollten Sie dabei die deutsche Tastaturbelegung benutzen wollen, genügt der veränderte Aufruf:

```
1> setmap d
1> run ed brief
1> setmap usa
```

Unabhängig von Ihrem Grund-CLI läuft nun "Ed". Sie können jetzt ganz nach Herzenslust Texte eingeben. Wenn der Brief

schließlich fertiggestellt ist, genügt die Tastenkombination ESC+X, um ihn unter dem Namen "brief" auf die Diskette zu speichern. Anschließend kann er mittels

```
1> type brief to prt:
```

ausgedruckt werden. Als weiteren Vorteil gegenüber der einfachen Ausgabe von Frage 4 kann man die Tatsache ansehen, daß der geschriebene Text auf der Diskette gespeichert ist. Er geht also nach dem Ausschalten des Rechners nicht verloren und kann auch noch nach Wochen gedruckt oder durch

```
1> run ed brief
```

erneut verändert werden. Wollen Sie den Text jedoch nicht länger speichern, genügt:

```
1> delete brief
```

Eine dritte Möglichkeit, Texte zu erstellen, bietet das Notepad. Sie rufen es wie folgt auf:

```
1> setmap d
1> run utilities/notepad
1> setmap usa
```

Hierbei handelt es sich um einen etwas erweiterten Notizblock, mit dem Sie die verschiedenen Zeichensätze (Fonts) benutzen können. Das ist aber auch sein größter Vorteil, ansonsten ist "Ed" vorzuziehen.

Frage 17

Ich möchte die Dateien auf meiner Workbench-Diskette sichtbar machen. Wie geht das?

Wann immer Sie mit der Maus ein Disk-Icon anklicken, erscheinen auf dem Bildschirm nur die Dateien, die über ein gleichnamiges Info-File verfügen. Dieses Info-File enthält das Aussehen des Symbols, mit dem das entsprechende File repräsentiert werden soll.

Nun gibt es auf der Workbench-Diskette oftmals Dateien ohne ein solches .info-File. Möchte man auch diese Dateien sichtbar machen, bedient man sich am besten der nachfolgenden Kommandosequenz. Geben Sie zu diesem Zweck bitte zunächst die folgende Anweisung ein:

```
1> ed S:show
```

Kurz darauf wird sich der Editor "Ed" melden. Nun schreiben Sie einfach den folgenden Text ab:

```
.key datei/a
.bra (
.ket )
if exists sys:cli.info
    echo "erstelle .info-File"
    if exists (datei)
        copy sys:cli.info to (datei).info
    else
        echo "es gibt gar kein solches quell-file!"
    endif
else
    echo "kein .info-original gefunden"
endif
quit
```

Nun drücken Sie zunächst die ESC-, dann die X-Taste. Daraufhin wird dieser Text unter dem Namen "Show" als Kommandosequenz in das Verzeichnis S: geschrieben. Sie können nun jede Datei, die über kein eigenes .info-File verfügt und deshalb bisher unsichtbar blieb, nachträglich mit einem .info-File versehen. Dazu geben Sie lediglich ein:

```
1> execute show NameDesFiles
```

Durch den Execute-Befehl wird die Kommandosequenz "show" aktiviert. Der Befehl .key sorgt dafür, daß das Argument, der Name des gewünschten Programms, an Stelle des Platzhalters "datei" eingesetzt wird. Der Zusatz "/A" bewirkt, daß dieses Argument zwingend eingegeben werden muß. Die Befehle .bra und .ket definieren die Zeichen, die als Anfang und Ende den Platzhalter markieren.

Nun wird geprüft, ob es das .info-File "cli.info" gibt, denn das soll als Original dienen. Sollte sich dieses File nicht in Ihrem Directory befinden, dann sollten Sie es vor Aufruf des Executes mittels

```
1> copy df0:system/cli.info to df0:
```

ins Haupt-Directory kopieren. Unter Umständen liegen die neuen Datei-Symbole genau übereinander, denn es sind ja eineiige Zwillinge. Sie brauchen dann die Icons bloß auseinanderzuziehen und mittels des Workbench-Menüpunktes "Snapshot" einzufrieren.

Frage 18
Wie kann ich verschiedene Texte zusammenfassen?

Sehr häufig kommt es vor, daß man über verschiedene separate Textstücke verfügt. Das könnten zum Beispiel Teile eines "C"-Listings sein oder Briefkopf, Text sowie ein Anhang. "Ed" kennt leider keine Möglichkeit, zusätzliche Texte nachzuladen. Statt dessen gibt es den JOIN-Befehl. Nehmen wir den Fall an, Sie verfügen über die drei Text-Dateien "kopf", "text" und "schluss". Sie wollen aus diesen Teilstücken einen zusammenhängenden Text erstellen. Dies erledigt JOIN:

```
1> JOIN kopf text schluss AS brief
```

Sie erhalten unter dem Dateinamen "brief" eine Zusammenfassung der drei Einzelkomponenten.

Frage 19
Wie kann ich bestimmte Textpassagen meiner Dateien suchen lassen?

Mit Hilfe des SEARCH-Befehls können Sie beliebige Dateien nach einem markanten Wort oder Satz durchsuchen. Sie rufen ihn mit diesen Parametern auf:

```
1> search (name) SEARCH (suchtext) ALL
```

name Name der zu durchsuchenden Datei, sonst Disketten-Directory.
suchtext Text, nach dem gefahndet werden soll.
ALL Alle untergeordneten Directories werden ebenfalls durchsucht.

Beispiele

```
1> search df0: SEARCH "tobi" ALL
```

grast alle Dateien der Diskette im Laufwerk df0: nach dem Wort "tobi" ab.

```
1> search brief SEARCH "Meier"
```

kontrolliert, ob sich der Name "Meier" im File "brief" befindet.

```
1> search dokum#? SEARCH "Rinteln, den"
```

sucht alle Dateien im aktuellen Verzeichnis, die mit den Buchstaben "dokum" beginnen, nach dem Satzanfang "Rinteln, den" ab. C-Programmierer können ebenfalls mit diesem Befehl auf die Suche gehen, und zwar nach den Namen ihrer Prozeduren und Variablen innerhalb ihrer Quell-Listings.

Frage 20
Läßt sich der Inhalt eines Textfiles eigentlich sortieren?

Ja, das geht. Dazu dient der SORT-Befehl. Mit ihm lassen sich Text-Dateien mit bis zu 200 Zeilen alphabetisch sortieren. Dies ist besonders nützlich für Adreßdateien. Befinden sich beispielsweise in der Datei "adressen" die ungeordneten Adressen Ihres Freundeskreises, so genügt dieser Aufruf:

```
1> sort adressen to geordnet
```

um eine alphabetisch geordnete Liste unter dem Namen "geordnet" wiederzufinden. Sollen mehr als 200 Zeilen Text sortiert werden, dann wird es nötig, den Stack zu erhöhen. Das kann man mit dem STACK-Befehl machen.

2.2 Neue CLI-Befehle

Mit dem Ende des Jahres 1988 ist auch endlich die neue gültige Version der Workbench herausgekommen! Mit dem Kickstart 1.3 und der Workbench 34.20 liegt vor uns eine vollkommen überarbeitete Version des Betriebssystems. Neu hinzugekommen sind CLI-Befehle, die dem Benutzer das Leben noch leichter machen sollen. Es handelt sich dabei um "Ask", "Avail", "Break", "Eval", "FF", "GetEnv", "Lock", "NewShell", "RemRAD", "Resident", "SetPatch", "SetEnv", "Which", "EndSkip" und "XIcon". Alle diese Befehle sind natürlich nicht in den älteren Handbüchern erwähnt und sollen deshalb hier erklärt werden. Um keinen zu benachteiligen, werden sie jetzt in alphabetischer Reihenfolge beschrieben:

Ask (Fragen an den Benutzer)

Bisher war es in Batchdateien nur möglich, einen vorgegebenen Befehlsablauf abzuarbeiten. Dabei war die einzige Möglichkeit, diesen Ablauf zu beeinflussen, über das IF-Kommando erlaubt. Allerdings wünschen sich viele CLI-Anwender auch einmal den Weg über den Dialog zum Benutzer. So könnte man in der Startup-Sequence fragen, ob nicht z.B., anstatt das CLI zu schließen, dieses lieber offen bleiben soll, damit man nicht wieder erst das Icon anklicken muß. Und genau für dieses Problem wurde Ask geschaffen. Es ist jetzt endlich implementiert worden, daß Sie auch Fragen an den Benutzer richten können.

Dafür schreiben Sie einen Text hinter Ask, genau wie Sie es auch bei Echo machen. Dann wird beim Abarbeiten daraus eine Frage, die der Benutzer mit Yes oder No beantworten kann. Wenn die Frage nun mit Y oder N beantwortet wird, so erhält das Programm einen entsprechenden Fehlercode. Das heißt: Bei Y oder Yes läuft das Programm ohne irgendwelche Änderungen ab. Bei N oder No wird ein Fehlercode von 5 erzeugt, der dann z.B. über If Warn abgefragt werden kann. Sehen Sie hier ein Beispiel für eine Batch-Datei mit dem ASK-Kommando:

Avail (Wieviel? Wovon überhaupt?)

Vielleicht hatten Sie auch schon einmal folgendes Problem: Sie arbeiten gerade im CLI z.B. mit einem Compiler und brauchen unbedingt die Speicherbelegung, damit Sie beurteilen können, ob der neue Programm-Code noch in den Speicher paßt. Bisher war die Lösung des Problems nur damit zu bewältigen, daß man die Workbench über LoadWB lud und dann mit einem Klick in die Workbench-Screen in der Titelleiste den freien Speicher angezeigt bekam.

Natürlich war auch diese Angabe ziemlich ungenau, aber es reichte. Viele Softwarehersteller boten deshalb kleine Info-Programme an, die laufend die aktuelle Speicherverteilung in einem kleinen Window ausgaben. Aber der Nachteil war ein neuer Task, der vielleicht wertvolle Prozessorzeit stiehlt, und auch der Platz auf dem Bildschirm, der verdeckt wurde.

Im neuen CLI ist dies anders! Es gibt den Befehl Avail, mit dem die aktuelle Speicherverteilung ausgegeben wird. Aber nicht nur dies! Weil man das ja schon alles kennt, haben sich die Programmierer von Commodore noch einige Features ausgedacht: So wird zusätzlich zum freien FAST- und CHIP-RAM noch das belegte in einer Tabelle vermerkt. Weiterhin führt die Tabelle eine Spalte, in der nach momentaner Konfiguration die maximalen Speichergrößen angegeben sind.

Und als letztes finden wir noch den jeweils größten freien Speicherblock der RAM-Sorten. Somit kann man auch beurteilen, warum z.B. AllocMem() einen Fehler zurückliefert, obwohl bei anderen Programmen angezeigt wurde, daß genügend freier Speicher vorhanden ist. Dann war nämlich der Speicher nur in kleinen Bruchstücken erreichbar und nicht als Ganzes! Zum Abschluß dieser Befehlserklärung hier ein Beispiel der Speichertabelle. Ich habe sie ausgegeben, während ich diesen Text schrieb:

```
1> avail
```

Type	Available	In-Use	Maximum	Largest
chip	197984	325248	523232	175376
fast	48104	468784	516888	44192
total	246088	794032	1040120	175376

Break (Schluß damit!)

Bei der Arbeit mit dem CLI ist es mir häufig passiert, daß ich, ohne daran zu denken, einen Befehl mit RUN losschickte, damit er nebenbei abgearbeitet wird. Hier vergißt man aber häufig, daß dieser nicht mehr abgebrochen werden kann, weil keine Tastenkombination (weder CTRL-C, -D, -E noch -F) den Task erreicht. Er liegt praktisch in einer unerreichbaren Zone des CLI.

Dieses Manko kann aber verheerende Folgen haben! Und deshalb wurde auch dafür ein neues CLI-Kommando geschrieben: Mit Break haben Sie die Möglichkeit, jeden laufenden Task, den Sie vom CLI aus gestartet haben und den Sie sonst mit einer CTRL-Kombination abbrechen könnten, auch wieder abzubrechen! Das allgemeine Format des Kommandos lautet:

```
Break PROCESS/A, ALL/S, C/S, D/S, E/S, F/S
```

Nach Break geben Sie also als erstes die Nummer des Tasks an, der unterbrochen werden soll. Wenn Sie die Nummer gerade nicht wissen, können Sie eine Liste über Status anfordern. Als nächstes folgt die Art der Unterbrechung. Hierfür stehen die bekannten Buchstaben, die Sie auch über die Tastatur betätigen würden. Mit dem besonderen Schlüsselwort ALL können Sie das System anweisen, es mit allen Kombinationen zu versuchen, damit möglichst schnell und sicher abgebrochen wird.

Zum Austesten des Befehls sollten Sie folgendes einmal versuchen: Kopieren Sie ein möglichst langes Programm von der Diskette ins RAM. Z.B. Run Copy SYS:Clock to RAM:; und geben Sie dann Break 2 ALL ein. (Es wird dabei vorausgesetzt, daß der Befehl in den zweiten Process gelegt wurde!) Nach kurzer Zeit meldet das CLI, daß das Kommando unterbrochen und das File wieder gelöscht wurde.

FF (Schneller, schneller, schneller!)

Wenn Sie sich bei einer Erkundung der Workbench die neue Startup-Sequence angesehen haben, wird Ihnen der Aufruf dieses neuen Befehls aufgefallen sein. Dort steht nämlich:

```
FF >NIL: -0
```

Damit wird ein Programm aufgerufen, das die Ausgabe um ca. 50% beschleunigt! Es heißt FastFonts und verspricht wirklich nicht zu viel. Diese Neuerung können Sie, wie übrigens alle neuen CLI-Kommandos, auch mit dem Kickstart V1.2 (33.180) nutzen. Selbst hier wird die Textausgabe so schnell, daß das Arbeiten mit dem BASIC-Editor endlich einmal zur Freude wird!

Wenn Sie, was zwar unvorstellbar erscheint, aber immer einmal wider Erwarten vorkommt, die Textausgabe in die alte Geschwindigkeit schalten müssen, geben Sie dazu als Parameter einfach -n an (das n steht für irgendeine Zahl größer null).

GetEnv (Hole einen definierten Wert)

Es wird der Inhalt einer Variablen gelesen. Da es im CLI aber eigentlich keine Variablen gibt, legt dieser Befehl eine Datei in der RAM-Disk an. Sie liegt im Verzeichnis "RAM:Env" und besitzt den gleichen Namen wie die Variable, der etwas zugewiesen werden soll. Der Inhalt selbst wird in der Datei abgespeichert. Hier ist erst einmal ein Beispiel:

```
1> getenv test
```

Sie bekommen als Antwort die Meldung

```
Can't get test
```

womit Ihnen der Amiga ganz freundlich mitteilt, daß unter diesem Namen noch nichts abgespeichert worden ist. Ein zweiter Befehl muß hinzugezogen werden, mit dem man eine Zuweisung durchführen kann.

SetEnv (Speichere einen neuen Wert)

Hiermit können Sie einer Variablen ein String zuweisen. Name Name ist die Bezeichnung der Variablen. String String enthält eine Zeichenkette, die der Variablen zugewiesen wird. Schreiben Sie zum Beispiel:

```
1> setenv test "der böse wolf"
```

So erzeugen Sie damit einen neuen Wert im oben angesprochenen Verzeichnis. Im File "RAM:Env/Test" steht der Text "der böse wolf". Schauen Sie ruhig einmal nach! Diesen Aufruf kann man so oft und so verschieden wiederholen wie man will. Immer wieder wird der neue Wert in das File gleichen Namens geschrieben. Eine sinnvolle Anwendung, mit der sich viele gute Kniffe erstellen lassen, sehen Sie im Verlauf des Kapitels.

Lock (Der Schlüssel zum ...)

Blockiert oder gibt den Zugriff auf eine Festplattenpartition frei. Sie geben dafür als Parameter die Bezeichnung der Festplattenpartition an (z.B. HD0), zusätzlich einen Schalter, mit dem Sie den Schutz ein- oder ausschalten und ergänzen diese Angaben mit einem Passwort, das maximal vier Zeichen lang sein kann.

```
1> lock hd0: ON DATA
```

Nur mit der Eingabe von

```
1> lock hd0: OFF DATA
```

kann dieser Schutz wieder aufgehoben werden.

RemRAD

Wie Sie vielleicht wissen, bietet das neue System 1.3 eine resetfeste RAM-Disk. Doch die Sache hat auch einen Haken! Da diese immer den konstanten Speicher belegt, also nicht wie die alte dynamisch wächst oder schrumpft, stellt sie ein "speicherfressendes" Problem dar. Und wie soll man sie löschen, schließ-

lich ist sie ja resetfest. Aus diesem Grunde gibt es einen Befehl, der diese Arbeit übernimmt. Er löscht die resetfest RAM-Disk aus dem System. Das Icon bleibt zwar bestehen, doch hat es keine Bedeutung. Bei dem nächsten Reset ist sie dann ganz verschwunden!

Resident

Beim Arbeiten im CLI stellt man fest, daß beim Zugriff auf eine zweite Diskette jedesmal der CLI-Befehl von der Workbench-Diskette nachgeladen werden muß. Dies war in Version 1.0, 1.1 und 1.2 so. Gott sei dank ist dies in Version 1.3 nicht mehr nötig! Hier können Sie jedes Kommando fest in den Speicher integrieren. Dazu verfährt man folgendermaßen:

Zuerst rufen Sie das Kommando Resident gefolgt von dem Namen des Befehls, der resident in den Speicher gelegt werden soll, auf. Ab jetzt ist ein Speicherbereich reserviert, in dem das Kommando abgelegt ist. Bei einem Aufruf wird nun nicht von der Diskette, sondern aus dem Speicher geladen. Schließt man das letzte CLI-Fenster, so wird der Speicher für die Befehle ja nicht mehr gebraucht und alle Blöcke werden wieder freigegeben. Allerdings behält die neue Shell eine Liste der als resident gekennzeichneten Befehle, und sobald man ein neues CLI-Fenster wieder öffnet, werden nach dieser Liste alle Befehle wieder resident gemacht. Sehen wir uns jetzt das allgemeine Format an, um noch weitere Möglichkeiten zu erkunden:

```
Resident Name, File, Delete/S, Add/S, Replace/S, Pure/S, SYSTEM/S
```

Wir können also als erstes den Namen angeben, gefolgt von dem File (den Files), die dementsprechend vorhanden sein sollen. Außerdem gibt es natürlich verschiedene Modi, um neue Befehle resident zu machen, alte zu entfernen oder durch neue Files zu ersetzen. Alles dies wird durch die dementsprechenden Worte angezeigt. Verwendet man Resident ohne jeden Zusatz, wird eine Liste aller bisher resident gemachten Files ausgegeben. Dies ist sehr nützlich, wenn man z.B. eingeschränkt ohne Workbench-Diskette arbeitet und wissen möchte, auf welche Befehle man zurückgreifen kann.

Für die tägliche Arbeit mit dem CLI empfehle ich zwei Batchdateien. Die erste sollte die folgenden Kommandos für resident erklären: List, Dir, Info, Delete, Run, Cd und Copy. Sie können die Liste natürlich nach Belieben ändern. In dem zweiten Execute-File werden alle diese Befehle wieder aus der Liste entfernt. Auch das ist zu empfehlen, wenn man wieder die Workbench-Diskette im Laufwerk hat, denn dann kann man den Speicher wesentlich besser für andere Anwendungen gebrauchen. Die Resident-Tabelle nach dem Start über die Startup-Sequence:

Name	UseCount
Execute	1
CLI	SYSTEM
FileHandler	SYSTEM
Restart	SYSTEM
CLI	SYSTEM

SetPatch

Nicht selten treten gerade während des Programmierens fatale Fehler auf, die das System mit einer Guru-Meditation quittiert. Doch in vielen Fällen wäre dies nicht nötig. Es wäre vollkommen ausreichend, wenn man über den Sachverhalt durch einen Guru informiert würde, der Task entfernt würde und das Leben weiter seinen Lauf ginge. Aber bisher wurden gleich alle laufenden Tasks unterbrochen und gelöscht, um das gesamte System neu zu booten.

Damit soll jetzt Schluß sein! Durch das im Hintergrund laufende Programm SetPatch wird jeder kleine Guru abgefangen und in einen gutartigen umgewandelt. Somit werden manche Gurus vollständig in ihrer Wirkung unterbunden, und die Arbeit braucht nicht unterbrochen zu werden. In der täglichen Anwendung werden Sie aber trotzdem weniger mit dem neuen Kommando in Berührung kommen, denn genauso wie z.B. auch FastFonts wird SetPatch nur einmal als neuer Task in der Startup-Sequence aufgerufen und ist danach still im Hintergrund. Dies sieht übrigens so aus:

```
Run >NIL: C:SetPatch
```

Die Ausgabe wird dabei an das nicht existierende Gerät NIL: geschickt, damit das DOS-Window nicht für eine eventuelle Ausgabe aufgehalten wird, wie es sonst geschehen würde.

Which

Which erleichtert die Arbeit besonders bei komplex aufgebauten Amiga-Benutzersystemen. Hier soll uns als Beispiel ein Anwender mit einer Festplatte dienen. Er hat sicherlich alle Befehle, die sich aus dem CLI ansprechen lassen, gut geordnet in mehreren Verzeichnissen. Die bekannten Kommandos im C:-Verzeichnis, dann noch das System-Verzeichnis usw.

Möchte man eines dieser Kommandos kopieren, tritt ein Problem auf: Bei der bisherigen Arbeit brauchte man überhaupt nicht zu wissen, wo dieser Befehl untergebracht war, denn mit Hilfe des Path-Kommandos suchte das CLI danach. Jetzt ist man aber auf sich gestellt und muß alleine suchen. Doch nun tritt Which in Aktion! Which durchsucht alle angegebenen Verzeichnisse und gibt dann den Pfad aus, unter dem das Kommando wirklich zu erreichen ist.

```
1> Which Format
Workbench 1.3:System/Format
```

EndSkip

EndSkip ist ein Kommando nur für die Batch-Dateien. Es läßt einen Sprung, der bis hierher noch nicht sein LAB gefunden hat, hier aufhören und damit den nächsten Befehl hinter diesem Kommando ausführen.

XIcon (Execute auch auf der Workbench)

Dieser Befehl ermöglicht es, CLI-Befehle über Symbole der Workbench zu aktivieren und nicht erst in das CLI gehen zu müssen. Es funktioniert! Dazu gehen Sie wie folgt vor:

Schreiben Sie zuerst den oder die gewünschten Befehle in eine Batchdatei. Dabei sind alle Features des CLI erlaubt. Nun besor-

gen Sie sich ein Project-Icon. Das können Sie von den BASIC-Programmen oder den NotePad-Texten "klauen". Für genauere Informationen lesen Sie bitte im Kapitel "Icons" nach. Dieses Icon versehen Sie mit dem Namen der Batchdatei. Beide müssen in einem Verzeichnis liegen, das auch über Icons, also z.B. die Diskette erreichbar ist.

Dann trägt man zu guter Letzt noch über die Info-Funktion der Workbench bei DEFAULT-TOOL das Programm C:XIcon ein, und wir sind fertig! Jetzt reicht ein Doppelklick auf das neue Icon, und die CLI-Befehle werden ausgeführt. Für den Fall, daß Texte ausgegeben werden müssen, öffnet das Programm noch ein CON:-Fenster. Dieses wird nach dem Abarbeiten aber wieder geschlossen.

Alte Befehle - neue Parameter

Nicht nur neue Befehle sind seit der Version 1.3 in das CLI implementiert worden. Auch die alten schon vorhandenen Kommandos haben eine gründliche Überarbeitung durchgemacht. So sind z.B. alle CLI-Kommandos vollkommen neu programmiert worden. Dadurch sind alle schneller und in vielen Fällen auch wesentlich kürzer.

Das Interessante für die Anwender sind aber die neuen Parameter, die vielfältige und neue Einsatzgebiete erschließen. In diesem Kapitel finden Sie nun eine Auswahl der Befehle, bei denen Parameter ergänzt wurden. Jedoch wird die Workbench immer wieder neu bearbeitet. Deshalb tut man gut daran, sich von Zeit zu Zeit die neueste Version zu besorgen und dort bei den Kommandos mit dem Fragezeichen die neuesten Angaben abzurufen. So bekommt man Informationen, die in keinem Handbuch stehen!

Assign

Assign hat in seiner neuesten Version einiges hinzugelernt. Jetzt kann man nicht nur einem Namen ein Verzeichnis zuordnen. Wenn diese Zuordnung nicht mehr gebraucht wird, kann sie auch wieder mit dem Wort DISMOUNT oder UNMOUNT ge-

löscht werden. Das ist insofern interessant, als eine Diskette solange als Symbol auf der Workbench gelassen wird, wie sie irgendwie in Gebrauch ist. Das ist auch der Fall, wenn wir ein Verzeichnis von ihr mit Assign gebrauchen. Ebenfalls neu ist die Prüfmöglichkeit von Assign. Verwendet man das Schlüsselwort EXISTS, dann gibt Assign einen Wert von 5 zurück, wenn der Name nicht existierte.

Copy

Copy wurde noch mehr auf das Kopieren und Herstellen von Originalen getrimmt. So bietet die jetzige Version vier Flags, mit deren Hilfe man alle Parameter einstellen kann. Mit DATE und COM kann man Copy anweisen, auch das Datum und den Kommentar des Originals mit zu übernehmen. Weiterhin erlaubt NOPRO eine Veränderung der Protection-Bits. Allgemein werden diese ja bei der Kopie übernommen. Verwendet man CLONE, dann werden alle drei Einstellungen genau aus dem Original übernommen.

Format

Auch Format hat einige Verbesserungen erfahren. Mit QUICK beschleunigt sich das Formatieren um einiges. Dabei werden jetzt nicht mehr alle Tracks gelöscht, sondern nur noch der Root-Block, der Boot-Block und die BitMap der Diskette. FFS weist Format an, das neue FastFileSystem als Format zu nehmen, und NOFFS unterbindet diese Einstellung.

Install

Die Programmierer des Amiga kennen das Viren-Problem. Deshalb wurde Install um einige Optionen ergänzt. Install kann jetzt auch den Boot-Block daraufhin überprüfen, ob sich dort vielleicht unbekannte Daten (Daten eines Virus) befinden. Dann wird eine Warnung ausgegeben und der Fail-Code auf 5 gesetzt. Mit NOBOOT kann eine Diskette von einem Virus gereinigt werden, und trotzdem ist sie dann nicht bootfähig.

List

List hat eine äußerst interessante Variante erfahren. Zusätzlich zu all den Parametern, die man bisher angeben konnte, gibt es jetzt auch noch LFORMAT. Hiermit definiert man einen String, über den die Ausgabe erstellt wird. In diesem String kann man feststehenden Text und das Formatzeichen %S gebrauchen. Der Text wird bei der Ausgabe einfach wiedergegeben, doch für das %S setzt List den Filenamen ein. So können Sie Listen kreieren, die z.B. in eine Batchdatei umgeleitet werden, um diese nachher ausführen zu lassen. Hier ein Beispiel:

```
LIST >RAM:Test #? LFORMAT="Copy %S to RAM:C"
```

Dieses Komando erstellt ein Batchfile, in dem alle Copy-Kommandos stehen, die das aktuelle Directory in das RAM-Verzeichnis C kopieren. Wenn Sie %S zweimal im String verwenden, werden beide Male die Filenamen eingesetzt. Somit kann man leicht Renames konstruieren:

```
LIST >RAM:Test #? LFORMAT="Rename %s /Backup/%S"
```

Wenn Sie %S im String dreimal verwenden, wird der erste durch den Filenamen mit Path ersetzt und der zweite nur durch den Filenamen. Braucht man den zweiten nicht, kann man ihn mit dem Kommatarzeichen ; unterdrücken:

```
LIST >RAM:Test #? LFORMAT="Join %S%S to RAM:Oberfile ;%S"
```

Protect

Seit der Version 1.3 gibt es vier weitere Protection-Bits, die auch von Protect unterstützt werden. Es sind dies Hidden (H), Script (S), Pure (P) und Archive (A). Wenn Hidden gesetzt ist und man über List den Inhalt eines Verzeichnisses ausgeben läßt, werden diese Files nicht ausgegeben: Sie sind versteckt (Betriebssystem 1.3 unterstützt dies nicht).

Setzt man bei einem File das Script-Flag, kann dieses auch ohne Execute als Batch-Datei ausgeführt werden. Das Pure-Flag muß bei allen Programmen gesetzt werden, die resident gemacht wer-

den sollen, Archive wird immer dann gesetzt, wenn ein Programm gerade kopiert wurde. Somit können Kopierprogramme die Files auslassen, die schon kopiert sind.

Version

Als letzter der erneuerten Befehle ist Version zu nennen. Mit ihm können wir jetzt auch die Versionsnummern der Libraries erfahren. Dazu gibt man hinter Version noch die Namen der Library an:

```
Version intuition.library
intuition.library version 33.702
```

2.3 Neue Startup-Sequences

Wer nicht eine Akku-gepufferte Uhr sein Eigen nennen kann, wird sich bereits gefragt haben, ob die Systemzeit auch in der Startup-Sequence eingestellt werden kann. Dem ist in der Tat so. Der Schlüssel zu diesem ist der DATE-Befehl. Gibt man anstelle von Zeitangaben ein Fragezeichen ein, erklärt sich der Befehl selbst und wartet auf die Eingabe der aktuellen Zeit. Erweitern Sie Ihre Startup-Sequence doch einmal um folgende Zeilen:

```
Resident Echo ;Nur unter Kickstart V1.3
Resident DATE ;Nur unter Kickstart V1.3
DATE      ;Ausgabe der aktuellen Zeit
Echo
Echo "Bitte geben Sie das aktuelle Datum in obigem Format ein."
Echo
DATE ?    ;Eingabe der aktuellen Zeit
Echo
Echo "Das aktuelle Datum ist nun:"
Echo
DATE      ;Ausgabe des geänderten Datums
Echo
```

Ähnliche Abfragen können Sie mit allen Befehlen erstellen, die anstelle von Parametern ein Fragezeichen akzeptieren. Sollten Sie Ihren Amiga jeden Tag einmal anschalten, besteht eine weitere Möglichkeit, das aktuelle Datum zu setzen:

```
                ;noch ist das Datum von Gestern eingestellt
DATE tomorrow ;Datum um einen Tag erhöhen
Echo "Heute haben wir den"
Date
Echo
Echo "System:"
Info            ;Information über Disks ausgeben
```

Die Betriebssystem-Version 1.3 bietet bekanntlich endlich die Möglichkeit, von einem anderen Gerät zu booten als dem internen Laufwerk DF0. Einige Festplatten beziehungsweise deren Controller fahren unter V1.3 das System hoch, ohne daß eine Bootdisk in einem Diskettenlaufwerk liegt. Nicht jeder Controller gestattet den Autoboot, so daß bei den meisten Platten grundsätzlich von Disk gebootet werden muß.

Um die Geschwindigkeit einer Harddisk sowohl unter V1.2 wie auch unter V1.3 beim Boot-Vorgang nutzen zu können, muß der größte Teil der Startup-Sequence auf der Festplatte ausgeführt werden. Ebenso sollten alle weiteren Diskettenzugriffe des Betriebssystems auf die Festplatte umgelenkt werden. Gehen Sie folgendermaßen vor:

1. Geben Sie im CLI ein:

   ```
   Copy SYS: DH0: all quiet
   ```

   Damit wird die gesamte Workbench-Disk auf die Festplatte kopiert.

2. Editieren Sie die Startup-Sequence, die sich nun auf Ihrer Harddisk befindet:

   ```
   Ed DH0:S/Startup-Sequence
   In der obersten Zeile fügen Sie ein:
   DH0:C/Assign C: DH0:C
   Assign SYS: DH0:
   Assign DEVS: DH0:DEVS
   Assign SYSTEM: DH0:SYSTEM
   Assign L: DH0:L
   Assign S: DH0:S
   Assign LIBS: DH0:LIBS
   Assign FONTS: DH0:FONTS
   Assign UTILITIES: DH0:UTILITIES
   CD DH0:
   ```

Löschen Sie mit CTRL-B die Zeilen:

```
BindDrivers
Mount DH0:
```

3. Ändern Sie die Startup-Sequence der Workbench durch

```
ED s/Startup-Sequence
```

Löschen Sie alle Befehle und ersetzen Sie sie durch

```
BINDDRIVERS ;nicht vergessen!
MOUNT DH0:  ;sofern nicht bereits durch binddrivers

vorhanden (Automount-Controller).
;Bei einigen Controllern noch 'CD DH0:' hinzufügen.
EXECUTE DH0:S/Startup-Sequence
```

4. Führen Sie einen Reset durch und prüfen Sie, ob es funktioniert.

5. Bei einwandfreier Funktion können folgende Dateien auf der Harddisk gelöscht werden:

   Die Expansion-Schublade (Treiber wurden durch binddrivers bereits geladen),

   ```
   DEVS/system-configuration
   ```

   Von der Workbench können alle nicht benötigten Dateien gelöscht werden. Da das vom Controller abhängt, sollten Sie sich vorher eine Sicherheitskopie dieser Disk anfertigen.

Bedenkenlos können im allgemeinen die Verzeichnisse SYSTEM, FONTS, UTILITIES, T, DEVS/PRINTERS, DEVS/KEYMAPS, LIBS/Mmagic.library und DEVS/clipboards gelöscht werden.

Wer im Besitz des Kickstart V1.3 ist, keinen Autoboot-Controller hat, sich aber eines genügend großen Speichers erfreut, kann natürlich auch noch einen anderen Weg gehen. Die von Haus aus bootfähige resetfeste RAM-Disk wird zunächst durch die Startup-Sequence der Workbench eingerichtet (mehr dazu an anderer Stelle dieses Buches). Die beim Systemstart benötigten Dateien und Verzeichnisse werden auf die resetfeste RAM-Disk kopiert (inklusive einer Startup-Sequence, die der der Workbench entspricht, jedoch nicht die Initialisierung der RAM-Disk beinhal-

tet). Danach dürfte unsere bootfähige RAM-Disk genauso bestückt sein wie die abgespeckte Workbench bei der Bootumleitung von Disk auf HardDisk. Bei einem Reset wird die Startup-Sequence im RAM abgearbeitet, von dort auf DH0 verzweigt.

2.3.1 RAM-Disk und Datenverarbeitung

Wer war nicht schon einmal dem Herzinfarkt nahe, als der Amiga ausgerechnet bei der wichtigsten Datendisk einen READ/WRITE-ERROR meldete; wer ist noch nicht mit einem schmerzenden Abdruck der Space-Taste unter seinem linken Auge aufgewacht, nach dem er Stunden zuvor SUPERBASE zum reformatieren seiner Dateien aufgefordert hat?!?

Kennen Sie derartige Probleme? Ja? Dann sind Sie hier genau richtig. Derartige Probleme hatte ich nämlich auch häufig genug - bis, ja bis ich mir die Parameter der 1.3-CLI-Befehle einmal genauer angesehen habe.

Bringen wir einmal die Probleme auf einen Punkt. Das nervigste an Datenverarbeitungen ist das ständige Schrubben der Disketten (es müssen ja eine Unmenge Daten bearbeitet werden können, die eventuell nicht ins RAM passen könnten). Das ewige hin und her schreiben der Daten, diese recht zeitintensive Hauptbeschäftigung mancher Datenbanken ist es auch, die zu den Schreib-/Lese-Fehlern führt. Die Aufzeichnungsdichte des Amiga übersteigt nämlich bereits leicht die Aufzeichnungsdichte, für die die allgemein erhältlichen Disketten ausgelegt sind. Wenn dann auf handelsüblichen Disks (135 TPI) ständig die Schreib-/Lese-Köpfe die Magnetschicht abhobeln...

Wir müssen also die Disketten-Zugriffe auf ein Minimum beschränken. Sinnvollerweise sortieren wir Dateien daher im RAM. Da wir unsere Dateiverwaltung kaum davon überzeugen können, nicht auf einer Disk zu sortieren, nehmen wir die RAM-Disk (die resetfeste, wenn möglich). Zu diesem Zweck ist es angebracht, Ihre Dateien in Gruppen zu ordnen und jede Gruppe in ein eigenes Directory zu legen. Die benötigten Directories wer-

den dann je nach Bedarf ins RAM kopiert, eventuell schon durch die Startup-Sequence. Bevor wir uns näher damit beschäftigen, sollten wir uns noch ein anderes Problem ansehen, den Stromausfall. Dieser bereitete mir häufig Kopfzerbrechen darüber, wie ich am sinnvollsten meine RAM-Dateien vor dem Totalverlust retten kann. Klar, ich könnte regelmäßig die RAM-Dateien auf die Disk zurückschreiben. Die Sache hat nur einen Haken: Normalerweise verändere ich nicht alle im RAM stehenden Dateien, so daß ein arg zeitintensives Diskettenquälen angesagt wäre. Man müßte daher nur die geänderten Dateien auf die Diskette zurückschreiben können, und das geht so:

Bei jedem Schreibzugriff auf eine Datei wird deren Datum neu gesetzt. Hier bietet sich uns ein Ansatzpunkt. Die Dateien müssen so ins RAM kopiert werden, daß ihr ursprüngliches Datum unverändert bleibt. Zu diesem Zweck bietet der COPY-Befehl den Parameter DATE an:

```
COPY "Datei" RAM: DATE
```

Leider ist es nicht möglich, dem COPY-Befehl mitzuteilen, daß nur Dateien ab einem bestimmten Datum kopiert werden sollen. So muß für jede Datei, die vom RAM auf Disk zurückkopiert werden soll, angegeben werden:

```
COPY RAM:Datei Datendisk:
```

Glücklicherweise existiert aber noch der LIST-Befehl, der in der V1.3-Version den Parameter LFORMAT hat (siehe neue CLI-Befehle). Mit diesem Befehl kann man sich nun eine Batch-Datei erstellen, die aus lauter COPY-Befehlen besteht. Damit sieht ein Batchfile zum Kopieren aller geänderter Dateien aus dem RAM auf Disk so aus:

```
LIST >RAM:Batchfile RAM: SINCE TODAY LFORMAT "COPY RAM:%s
Datendisk:"
EXECUTE RAM:Batchfile
DELETE RAM:Batchfile
DELETE Datendisk:Batchfile
```

Mit einem Icon ausgestattet und XIcon als DefaultTool (siehe neue CLI-Befehle) können Sie dieses Batchfile jederzeit von der Workbench anklicken, eine feine Sache, wie ich meine.

Wer die Geschichte gern automatisieren möchte, kann dank des neuen SKIP-Befehls folgendermaßen vorgehen:

1. Batchfile ändern in:

```
LAB meinlabel     ;Label definieren
WAIT 600          ;10 Minuten warten
LIST >RAM:Batchfile RAM: SINCE TODAY LFORMAT "COPY RAM:%s
Datendisk:"
EXECUTE RAM:Batchfile
DELETE RAM:Batchfile
DELETE Datendisk:Batchfile
SKIP meinlabel    ;nur V1.3 Skip springt auch rückwärts
```

2. Eingeben:

```
RUN NamedesBatchFiles
```

3. Nach dem Beenden der Datenverarbeitung:

```
BREAK [Prozess] (siehe neue CLI-Befehle)
```

2.3.1.1 MACROs im CLI mit ALIAS

Der neue Befehl ALIAS (V1.3) gestattet es dem Benutzer, CLI-Befehle zu definieren. Dabei wird einem frei erfundenen Namen eine CLI-Kommando-Sequenz übergeben. Es besteht die Möglichkeit, Variablen, die später hinter dem Namen angegeben werden, zu übernehmen. Ein Beispiel ist xcopy:

```
ALIAS xcopy copy [] clone
```

Die eckigen Klammern dienen der Variablenübernahme. Um den neuen Befehl xcopy auszuprobieren, den wir hiermit definiert haben, schreiben wir folgendes:

```
xcopy sys:c/assign ram:
```

Der Assign-Befehl wird in die RAM-Disk kopiert. Das mag sicherlich noch nicht sehr sinnvoll sein, zeigt aber zumindest schon einmal die Variablen-Übergabe, denn intern wird durch obige Eingabe folgender Befehl ausgeführt:

```
copy sys:c/assign ram: clone
```

So richtig interessant wird die Sache mit Beispielen wie diesem:

```
ALIAS start ed s/startup-sequence
```

Geben Sie nun "start" ein, können Sie unmittelbar darauf die Startup-Sequence editieren! Nun kann man natürlich einmal die Namen oder Bedeutungen der selbsterfundenen Befehlsworte vergessen haben. Dann genügt:

```
ALIAS
```

Schon werden alle festgelegten Namen und ihre Definitionen ausgegeben.

2.3.1.2 CLI-Startup, die Startup-Sequence des CLI

Vielleicht haben Sie bereits die neue Datei im S-Directory der Workbench 1.3 entdeckt: "CLI-Startup". Sie ist nichts anderes als eine Startup-Sequence für das CLI. Alles, was Sie in dieser Datei finden, ist beim Öffnen des Shells aktiviert, beispielsweise sollten Sie immer benötigte ALIAS-Definitionen in diesem Batchfile ablegen. Klicken Sie von der Workbench aus das SHELL an, können Sie auch schon die in dieser Datei mit ALIAS definierten Befehlsworte verwenden.

Das EXECUTE [Batchdatei] entfällt also. Auch alle anderen Voreinstellungen (etwa mit RESIDENT im RAM abgelegte Befehle) werden beim Starten des Shell sofort ausgeführt. Im folgenden Kapitel finden Sie etliche Beispiele für ALIAS, die Sie mit ED in der CLI-Startup-Datei verewigen können.

2.3.1.3 ALIAS-Befehle

Anwendungen von ALIAS gibt es sicherlich unbegrenzt viele. Hier ein paar Beispiele:

```
ALIAS BORDERLESS echo "*e[0x*e[0y*e[0;0H*e[J"
```

Das Shell-Window hat keinen Rahmen mehr, die gesamte Fläche wird zur Ausgabe genutzt. Vorher sollte das Window auf die gewünschte Größe gebracht werden.

```
ALIAS CLS echo "*e[0;0H*e[J"
```

Löscht den Bildschirm.

```
ALIAS FETT echo "*e[1;31;40m"
```

Schaltet auf Fettdruck um.

```
ALIAS ULINE echo "*e[4;31;40m"
```

Schaltet Unterstreichen ein, was beim LIST-Befehl von Vorteil sein dürfte.

```
ALIAS NORM echo "*e[0;31;40m"
```

Schaltet zurück auf normale Darstellung.

```
ALIAS PRINT run >nil: type >prt: []
```

Gibt das als Parameter anzugebende ASCII-File im Multitasking-Betrieb auf dem Drucker aus.

```
ALIAS DELDISK sys:system/format >nil: <nil: drive [] name "Leer" quick
```

Löscht blitzschnell alle Dateien und Verzeichnisse auf dem angegebenen Drive.

```
ALIAS CtoRAM run >nil: copy sys:c ram: all quiet
```

Kopiert das C-Directory ins RAM.

```
ALIAS Tree dir [] opt a
```

Gibt alle Verzeichnisse und Dateien des angegebenen Directories aus.

```
ALIAS PrintTree run >nil: dir >prt: opt a
```

Gibt im Multitasking-Betrieb alle Dateien und Verzeichnisse des aktuellen Directories auf dem Drucker aus.

```
ALIAS Batch run >nil: execute "Filename"
```

Führt im Multitasking-Betrieb eine gewünschte Batch-Datei aus, wodurch das lästige Eingeben von Execute entfällt.

```
ALIAS myC copy >nil: sys:c(assign|type|dir|copy) ram:
```

Kopiert alle in der runden Klammer mit dem senkrechten Strich voneinander getrennten CLI-Befehle ins RAM.

2.3.1.4 Batch-Programmierung

Zur richtigen Batch-Programmierung gehören Abfragen und Verzweigungen, denn für jede Kleinigkeit eine neue Batch-Datei zu erstellen, dürfte wohl etwas aufwendig sein. Kommen wir gleich zu einigen neuen Features der Workbench 1.3. Neu ist hier beispielsweise der ASK-Befehl, der auf eine Eingabe wartet. Ist diese Eingabe 'Y' oder 'YES', gibt dieser Befehl einen Fehlercode zurück, den man als Warnung einstufen kann. Diese Warnung kann durch den IF-Befehl abgefragt werden. Hier ein Beispiel, das in die Startup-Sequence eingebaut das C-Directory bei Bedarf ins RAM kopiert:

```
ASK "Soll ich das C-Directory ins RAM kopieren (Y/N)?"
IF WARN
  COPY SYS:C RAM: ALL QUIET
  ASSIGN C: RAM:
ENDIF
```

Sollten Sie bei obigem Beispiel nicht mit Y(ES) antworten, wird die Befehlsfolge übersprungen und mit dem auf ENDIF folgenden Programm weitergemacht. Auch Verschachtelungen sind möglich:

```
ASK "Wollen Sie ein gutes Programm starten (Y/N)?"
IF WARN
  ASK "Eine Textverarbeitung (Y/N)?"
  IF WARN
    ASK "BECKERtext (Y/N)?"
    IF WARN
      ASSIGN FONTS: BECKERtext:FONTS
      ASSIGN PRT: BECKERtext:PRT
      CD BECKERtext:
      BECKERtext
      SKIP Ende
    ENDIF
    ASK "TEXTOMAT (Y/N)?"
    IF WARN
      ASSIGN FONTS: TEXTOMAT:FONTS
      TEXTOMAT:TEXTOMAT
      SKIP Ende
    ENDIF
  ENDIF
  ASK "Oder vielleicht PROFIMAT?"
  IF WARN
    ASSIGN TABELLEN: PROFIMAT:TABELLEN
    CD PROFIMAT:
    PROFIMAT
  ENDIF
ENDIF
LAB Ende
```

Wie Sie hier sehen können, liegt eine relativ einfache Verschachtelung von IF-ENDIF-Bereichen vor. Eine solcherart programmierte Batch-Datei paßt sich dem jeweiligen Wunsch des Users an. Bei allen, die nicht ständig von der Workbench aus arbeiten, sich vielmehr häufig mit bestimmten Programmen beschäftigen, bietet sich der Einbau in die Startup-Sequence förmlich an.

Gerade die Startup-Sequence bot unter Workbench 1.2 meist ein recht trauriges Bild. Zunächst eventuell eine Abfrage, gefolgt von einem oder mehreren Programmen und danach war Feierabend. Der neue SKIP-Befehl (1.3) erlaubt jetzt erstmalig das Zurückspringen in einer Batch-Datei. So lassen sich ganz wun-

derbare Schleifen programmieren. Ein Beispiel für alle, die die Finger nicht vom C-Compiler lassen können:

```
...                          ;<< Initialisierung
LAB loop                    ;Label für Schleife definieren
  ED RAM:Source.C            ;Aufruf eines C-Source-Editors
  CC RAM:Source.C -oTMP.O ;Compilieren (Optionen nach
  LN RAM:TMP.O +lc           ;Linken        Wunsch)
ASK "Editieren (Y/N)"      ;Abfrage
IF WARN SKIP loop          ;Rücksprung
...
```

2.4 Richtiges Ausnutzen des Mount-Befehls

Mount

Ja, der Mount-Befehl wird wahrlich sehr selten genutzt. Aber warum? Schließlich bietet er Möglichkeiten, von denen man bisher nur geträumt hat! Um den Befehl genauer zu erkunden, müssen wir erst einmal verstehen, was er überhaupt macht. Das Kommando Mount bindet in das Betriebssystem des Amiga ein neues Device ein. Sehen wir uns dazu zuerst an, welche Devices es schon gibt. Dies kann man ganz leicht über Assign erfahren. Sie könnten dann etwa folgenden Ausdruck bekommen:

```
Volumes:
Tips & Tricks [Mounted]
RAM DISK [Mounted]
BECKERtext
Workbench 1.3 Wgb [Mounted]
Directories:
FONTS          Volume: BECKERtext
ENV            RAM:Env
T              RAM:T
S              Workbench 1.3 Wgb:S
L              Workbench 1.3 Wgb:L
C              Workbench 1.3 Wgb:C
DEVS           Workbench 1.3 Wgb:devs
LIBS           Workbench 1.3 Wgb:libs
SYS            Workbench 1.3 Wgb:
Devices:
NEWCON DF1 DF0 PRT
PAR SER RAW CON RAM
```

Für uns ist dabei der letzte Abschnitt besonders interessant. Nachdem alle angeschlossenen Laufwerke und die besonderen Directories aufgeführt sind, kommt eine Liste der ansprechbaren Devices.

DF0: und DF1: sind Ihnen hinreichend bekannt. Mit PRT: kann man den Drucker erreichen, und PAR: bzw. SER: stellen die beiden Schnittstellen dar. Über RAW: und CON: kann man die Ausgabe erledigen, ohne den Weg über Intuition zu gehen. Das letzte Device RAM: kennzeichnet die RAM-Disk, mit der wir fast täglich arbeiten.

Nun sind alle hier aufgeführten Devices schon von Anfang an in das System eingebunden und können jederzeit erreicht werden. Wollen wir aber ein neues Device ansprechen, ist dies erst über Mount dem System bekannt zu machen. Diesen Weg müssen wir z.B. gehen, um für das neue CLI, die Shell, einen Editor zu erstellen, der intelligenter ist (NEWCON). Lesen Sie mehr zu diesem Thema im Kapitel über das neue Betriebssystem.

Für dieses Vorhaben brauchen wir einen Eintrag in der Mount-List, die sich im Verzeichnis DEVS: befindet. Hier ist zunächst ein Beispiel für ein externes Laufwerk, das auch als DF1: angesprochen werden kann. Sie finden dieses Beispiel übrigens schon in Ihrer Mountlist:

```
DF1:        Device = trackdisk.device
            Unit   = 1
            Flags  = 1
            Surfaces  = 2
            BlocksPerTrack = 11
            Reserved = 2
            PreAlloc = 11
            Interleave = 0
            LowCyl = 0  ;  HighCyl = 79
            Buffers = 20
            BufMemType = 3
#
```

Die Definition besteht im wesentlichen aus dem Namen des neuen Devices, also DF1:, und der Endkennung #. Alles, was

dazwischen steht, hängt von dem jeweiligen Device ab. Es gibt allerdings Parameter, die häufig benutzt werden:

Device

Hier findet Mount den Namen des Device-Treibers. Alle Devices werden natürlich im Verzeichnis DEVS: abgelegt, wo Sie z.B. das ramdrive.device oder das narrator.device finden. Setzen Sie den Namen entsprechend ein.

Unit

Hiermit bezeichnet man die Nummer des jeweiligen Gerätes. So hat das erste physikalische Diskettenlaufwerk die Nummer 0, das zweite - ob intern oder extern, ist egal - die Nummer 1 usw. Ein 5¼"-Laufwerk hat in jedem Fall die Nummer 2. Man zählt hier also die Nummer der bisher angeschlossenen Laufwerke eines Typs. Dies gilt nicht nur für Diskettenstationen!

Flags

Enthält einen Wert, der je nach Device unterschiedlich ist.

Surfaces

Gibt an, um wie viele Schreiboberflächen es sich handelt. Ein Diskette hat im Normalfall 2 Schreibseiten, wohingegen die Harddisk schon ab 4 hat.

Reserved

Anzahl der Datenblöcke des Boot-Blocks.

BufMemType

Speichertyp für den Datenpuffer:

```
0,1 = egal; 2,3 = CHIP-RAM; 4,5 = FAST-RAM
```

BootPri

Setzt die Priorität des Laufwerks für einen Boot. Je höher die Nummer, desto eher wird von diesem Device gebootet. Diese Option wird besonders ab Version 1.3 des Betriebssystems wichtig. Dann kann auch das Booten von der RAM-Disk oder der Festplatte eingestellt werden. Und so geht das für jeden Eintrag weiter. Ich möchte hier nicht detailliert auf jeden eingehen, weil die Daten doch sehr stark davon abhängen, welches Device eingebunden werden soll. Sehen Sie sich lieber die Tips & Tricks für Spezialfälle an:

2.4.1 Neue Namen für alte Hüte

Wenn man es gewohnt ist, die Gerätenamen der IBM-Kompatiblen zu benutzen, wird man einige Schwierigkeiten haben, sich an die Bezeichnungen des Amiga zu gewöhnen. Diese Erfahrungen macht man besonders stark, wenn man oft zwischen beiden System wechselt, sei es im Betrieb und zu Hause oder wenn man stolzer Besitzer einer PC-Karte ist. Für diesen Fall wäre es am geeignetsten, die Namen DF0, DF1 und wie sie alle heißen nach denen der IBMs zu benennen.

Ein findiger Leser wird sogleich bemerken, daß doch über Assign jedem Verzeichnis ein neuer Name zugeordnet werden kann. Also probieren wir es aus:

```
Assign B: DF1:
```

Wenn Sie es wirklich einmal ausprobieren, werden Sie sicherlich begeistert sein. Anstatt immer DF1: tippen zu müssen, reicht jetzt B: aus. Doch wechseln Sie einmal die Diskette. Schon stellt sich der Amiga stur und fordert die andere Diskette. Assign bezieht sich also immer nur auf das existierende Verzeichnis und nicht auf eine Diskettenstation. Dafür gehen wir einen anderen Weg.

Zunächst wird in der MountList einfach die Definition für DF1: kopiert, denn sie enthält alle nötigen Daten für ein Disketten-

laufwerk. Dann ändern wir den Namen DF1: in B: um. Das war's schon, denn jetzt reicht nach dem Abspeichern ein Mount B: aus, und sofort kann auch das Laufwerk DF1: als B: angesprochen werden. Das gleiche ist natürlich auch mit DF0: möglich. Dazu muß in der MountList noch einmal eine Kopie des alten Eintrags gemacht werden. Jetzt verändern Sie aber nicht nur den Namen in A:, sondern auch die Unit von 1 auf 0, damit wirklich das 0. Laufwerk angesprochen wird.

Nach diesen Definitionen können Sie DF1: sowohl unter seinem alten Namen als auch unter dem neuen Namen B: ansprechen. Gleiches gilt auch für DF0:! Die Tabelle der Devices wurde um die beiden neuen Geräte ergänzt, wie man leicht über Assign nachprüfen kann.

```
Devices:
A B NEWCON DF1 DF0
PRT PAR SER RAW CON
RAM
```

2.4.2 Weniger ist mehr!

Nicht nur die oben genannte Spielerei kann über Mount realisiert werden. Es gibt sogar sehr ernsthafte Anwendungen, mit denen man Geld sparen und bei vielen Freunden Verwirrung stiften kann! Für dieses Vorhaben werden wir nicht - wie eben beschrieben - einfach den Namen unseres Devices ändern. Wie gehen tiefer und wagen uns in die Welt der Parameter vor. Hierunter finden wir drei Stück, deren Bedeutung leicht erkennbar ist und deren Veränderung mehr erreicht, als man denken könnte.

Stellen Sie sich folgende Situation vor: Sie haben bei RAMSCH-KAUF eine 10er Packung ABCD-Disketten gekauft, die im Super-Sonder-Angebot nur 45,80 DM gekostet haben. Nur stand es leider nicht auf der Verpackung, daß sie minderwertig waren. Erst beim Formatieren als Datendisketten merkten Sie, daß fast alle Disks auf der Seite 1 einen Hardwarefehler meldeten und das Formatieren abgebrochen wurde.

Nun kommt der Trick! Wir gehen wieder in die MountList und kopieren die Einstellungen für DF1:, verändern den Namen in "WGB:" und - das ist der Clou - verändern weiterhin unter Surfaces den Eintrag von 2 in 1. Damit haben wir eine neue Diskettenstation geschaffen, die unsere Disketten nur auf einer Seite formatiert!

Wenn Sie jetzt die Formatierungsroutine im CLI aufrufen mit: Format Drive WGB: Name "1 Surface Test", werden Sie feststellen, daß das Formatieren schneller geht, weil nur noch die Hälfte beschrieben wird, und Sie können auch noch diese Disketten benutzen. Somit ist nur noch die Hälfte verloren! Es ist dabei darauf hinzuweisen, daß diese Diskette nur über das neu eingerichtete Device gelesen werden kann. Sie haben also nur über eine persönliche Schnittstelle Zugriff auf die darauf abgespeicherten Daten. Es ist somit gleichzeitig ein gewisser Kopierschutz! Kleiner Tip am Rande: Kaufen Sie demnächst lieber Qualitätsdisketten, dann brauchen Sie unseren Tip nicht!

Unter dem gleiche Stichwort gibt es noch zwei andere Methoden. Die erste behandelt den Fall, daß nicht immer nur eine Seite der Diskette beschädigt oder defekt sein kann. Gut ist es auch möglich, daß z.B. die ersten 5 Tracks ständig Read-Errors produzieren. Dann kann man die Anzahl der Schreibseiten lassen, wie sie ist. Hier empfiehlt es sich, eher die Variable LowCyl auf 5 zu stellen. Dann beginnt der Formatierungsvorgang erst ab diesem Track. Alles weitere läuft genauso ab wie oben beschrieben. Der umgekehrte Fall beinhaltet Read-Errors am anderen Ende der Diskette. Dort sind vielleicht Track 71-79 nicht mehr lesbar. Hier ändern Sie einfach HighCyl und gehen wieder den schon beschriebenen Weg.

Ein letzter Hinweis zu der hier erwähnten Methode sei noch genannt: Bei der Verwendung neuer Formate gibt es einige Komplikationen mit der Workbench und den Diskettenstationen. Das erste Problem liegt an der Workbench. Wenn ein neues Laufwerk eingerichtet wurde und Sie dort Ihre Diskette formatiert haben, werden Sie erstens nicht mehr das Icon DF1:NDOS los. Das mag zwar nicht so schlimm sein, aber daraus resultiert das zweite

Problem. Das Laufwerk DF1: ist jetzt nicht mehr ansprechbar, ob man eine Diskette im normalen Amiga-Format einlegt oder nicht. Er meldet sich immer mit "No disk present in unit 1", womit unmißverständlich zu verstehen gegeben wird, daß nur noch das neue Format akzeptiert wird. Allerdings können Sie das neue Format auch von der Workbench ansprechen!

Und darin liegen einige interessante Anwendungen. Wenn man z.B. alle Datendisketten mit diesem Format verwendet, stellt das eigene Lesen kein Problem dar, doch alle anderen, die nicht unbedingt die Daten lesen sollen, haben keinen Zugriff ohne die richtige MountList.

2.4.3 Drucker-Spooler

Wenn wir im Zusammenhang mit dem Multitasking von einem Drucker-Spooler sprechen, kann dies eigentlich nur eines bedeuten: Wir binden einen neuen Task in das System ein, der eine Datei auf dem Drucker ausgibt.

Für einen neuen Task haben wir im CLI ja bekanntlich den Befehl RUN, als Spooler-Programm wählen wir eine Batch-Datei. Das Ganze sieht dann so aus:

Nehmen Sie sich das CLI und geben Sie ein:

```
ED c:PRINT
```

Nun das folgende Programm:

```
.key filename/a,typ/s        ;Übernehmen der Parameter

;--------------------------
;     Drucker-Spooler
;--------------------------
;(c) 1987 by Stefan Maelger
;--------------------------

if not exists <filename>     ;Prüfen ob File vorhanden

 echo "File not found"       ;nein?
```

```
 quit                          ;-dann raus hier
else                           ;ansonsten:
 copy <filename> to ram:<filename>
                               ;File in die RAM-Disk kopieren
 if <typ> eq "DUMP"          ;Hex-Dump ausgeben?
  run >nil: type ram:<filename> to prt: opt h
                               ;-HexDump-Spooling
 else                          ;sonst:
  run >nil: type ram:<filename> to prt: opt n
                               ;-normales Spooling
 endif
 delete ram:<filename>         ;Speicher wieder freigeben
endif
echo "printing"                ;Ausgabe bestätigen
quit
```

Speichern Sie das File nun mit ESCAPE-X ab. Die Routine können Sie ab jetzt mit der Eingabe von:

```
EXECUTE PRINT filename (DUMP)
```

aufrufen. Der Parameter DUMP ist optional und kann daher weggelassen werden. Da der Befehl EXECUTE sehr lang ist - man ist ja schließlich schreibfaul - empfehle ich, folgendes einzugeben:

```
run >nil: copy sys:c/EXECUTE to sys:c/DO quiet
```

Damit steht Ihnen neben dem alten Befehl EXECUTE jetzt auch noch der Befehl DO zur Verfügung, der natürlich das gleiche macht, was EXECUTE auch tut. Also beispielsweise:

```
DO PRINT filename
```

Von einem einfachen "RENAME sys:c/EXECUTE TO sys:c/DO" muß ich hier abraten, da es zwar ganz wunderbar die Arbeit erleichtert, wenn alle CLI-Befehle nur noch ein oder zwei Buchstaben lang sind, Ihnen aber mit ziemlicher Sicherheit nach einiger Zeit die Übersicht fehlt, so daß Sie schließlich nicht mehr wissen, welcher Befehl nun was bewirkt.

3. Das AmigaBASIC

Das mit dem Amiga ausgelieferte BASIC stammt aus dem Hause Microsoft. Kenner werden die Ohren spitzen: Sollte es möglich sein, daß diese Firma einen BASIC-Interpreter entwickelt hat, der auf die unglaublichen Möglichkeiten des Amiga abgestimmt ist?

Leider nutzt das AmigaBASIC nicht alle Möglichkeiten Ihres Gerätes aus. Zwar werden Fenster, Screens und Maus voll unterstützt, aber eine Reihe anderer Dinge lassen sich mit den herkömmlichen Befehlen des Interpreters nicht programmieren, wie zum Beispiel nachladbare Zeichensätze, ausführliche Diskettenprogrammierung und Zugriff auf das Disk Operating System (DOS).

Einen Befehl gibt es allerdings, dem der fortgeschrittene BASIC-Programmierer besondere Beachtung schenken sollte. Es ist der LIBRARY-Befehl. Mit seiner Hilfe werden sich all die fehlenden Befehle nachträglich implementieren lassen. Wie das funktioniert, erfahren Sie auf den nächsten Seiten.

3.1 Implementierung der Amiga-Kernel-Befehle

Das AmigaBASIC läßt sehr flexible Programmierungen zu. Neben den ureigenen Befehlen des AmigaBASIC (wie zum Beispiel "PRINT", "IF...THEN...ELSE") kann der Interpreter auch fremde Befehle verarbeiten, wenn sie in Form kleiner Maschinensprache-Routinen organisiert sind. So lassen sich eigene Befehle leicht in den bestehenden Befehlsvorrat des BASIC integrieren. Viel einfacher als das mühevolle Programmieren eigener neuer Routinen ist der Zugriff auf bereits bestehende Maschinenprogramme. Das Betriebssystem des Amiga verfügt bereits über solche Maschinenprogramme, die normalerweise die immer anfallende Arbeit des Systems verrichten. Diese Routinen werden

zusammengefaßt als "Kernel" bezeichnet, was englisch ist und für "Kern" steht: der eigentliche Kern des Systems.

Das Betriebssystem unterteilt diese zahllosen Maschinenprogramme in insgesamt dreizehn "Libraries" (engl. Büchereien), streng geordnet nach Thematik. Für unsere weiteren Routinen werden jedoch nur fünf dieser Libraries benötigt, und zwar:

- **exec.library**
  Zuständig für Tasks, Listen, I/O, allgemeine Systembelange, Speicherverwaltung

- **graphics.library**
  Zuständig für Text und elementare Grafik.

- **intuition.library**
  Zuständig für Fenster, Screens, Requester, Alarmmeldungen.

- **dos.library**
  Zuständig für das Disk Operating System (DOS).

- **diskfont.library**
  Zuständig für die Verwaltung nachladbarer Zeichensätze.

Jede dieser Libraries ist gefüllt mit Maschinenroutinen für die entsprechenden Aufgabenbereiche. Damit AmigaBASIC diese Routinen verwenden kann, benötigt es dreierlei Informationen: Vor allem muß der Interpreter den Namen einer jeden Routine wissen. Dieser Name unterliegt keinerlei Beschränkungen. Sie können prinzipiell jeder Maschinenroutine einen selbstkreierten Namen zuordnen.

In der Praxis gibt es jedoch für jede Maschinenroutine einen standardisierten Namen. Zweitens muß dem Interpreter mitgeteilt werden, an welcher Stelle innerhalb der Library die entsprechende Routine gefunden werden kann. Dazu besitzt jede Library eine Offset-Tabelle.

muß das AmigaBASIC wissen, welche Übergaberegister für die Routine benötigt werden. Insgesamt gibt es acht Daten- und fünf Adreßregister, die AmigaBASIC benutzen kann:

```
 1 =  Datenregister d0
 2 =  Datenregister d1
 3 =  Datenregister d2
 4 =  Datenregister d3
 5 =  Datenregister d4
 6 =  Datenregister d5
 7 =  Datenregister d6
 8 =  Datenregister d7
 9 =  Adreßregister a0
10 =  Adreßregister a1
11 =  Adreßregister a2
12 =  Adreßregister a3
13 =  Adreßregister a4
```

Für jede Library gibt es deshalb ein sogenanntes .bmap-File. In dieser Datei sind die nötigen Informationen für alle Befehle enthalten, die in der Library organisiert sind. Mittels des vom Amiga auf der "Extras"-Diskette mitgelieferten "ConvertFd"-Programms können Sie die nötigen .bmap-Files leicht erstellen. Bevor Sie nun fortfahren, sollten Sie diese Dateien anlegen:

```
graphics.bmap
intuition.bmap
exec.bmap
dos.bmap
diskfont.bmap
```

Kopieren Sie diese Dateien anschließend auf die Workbench-Diskette in das Unter-Directory "LIBS:", oder sorgen Sie dafür, daß sich die .bmap-Dateien immer zusammen mit Ihren Programmen in ein- und demselben Inhaltsverzeichnis befinden. Vom CLI aus könnte das Kopieren so geschehen:

```
1> copy graphics.bmap to libs:
1> copy intuition.bmap to libs:
```

3.1.1 Nutzung der Systembibliotheken in GFA

Auch der GFA-Programmierer verfügt über die Möglichkeit, die zahlreichen System-Bibliotheken und die in ihnen enthaltenen Funktionen für seine Zwecke zu verwenden. Im Vergleich zu AmigaBASIC hat er es sogar ein wenig einfacher.

GFA unterscheidet zwischen zwei Arten von Bibliotheken. Da sind zunächst die gebräuchlichsten unter ihnen wie beispielsweise Intuition, Exec und Graphics. Diese Bibliotheken hat GFA in weiser Voraussicht bereits für Sie geöffnet. Doch damit nicht genug: GFA kennt bereits alle in diesen Bibliotheken abgelegten Routinen beim Namen und weiß, welche Parameter sie verlangen. Der Umgang mit diesen Bibliotheken und Routinen ist also denkbar einfach, denn Sie brauchen diese Bibliotheken weder zu öffnen noch die in ihnen lagernden Routinen zu deklarieren noch "fd."- oder ".bmap"-Dateien zu erzeugen. Allerdings kennt GFA nicht alle Bibliotheken, denn das wäre zu speicheraufwendig. So besteht die zweite Kategorie aus den restlichen Bibliotheken, deren Funktionen nicht ohne weiteres verwendet werden können.

Ob nun eine von Ihnen gewählte Betriebssystem-Funktion zur ersten und benutzerfreundlichen oder zur zweiten, etwas aufwendigeren Kathegorie zählt, läßt sich leicht herausfinden. Sie brauchen nämlich lediglich den korrekten Funktionsaufruf in ihr Programmlisting einzufügen. So beispielsweise:

```
mem=allocmem(100,2^16)
```

Da GFA die Funktion AllocMem aus der Exec-Bibliothek kennt, wandelt der Interpreter obige Zeile automatisch um in:

```
mem=AllocMem(100,2^16)
```

Damit ist dieser Funktionsaufruf akzeptiert. Handelt es sich aber um eine unbekannte Funktion, so interpretiert GFA diese als Variable und beläßt die Kleinschreibung. Solche Aufrufe sind nicht erlaubt und müssen von Ihnen wieder gelöscht werden.

Wollen Sie jedoch partout eine der weniger gebräuchlichen und für GFA unbekannten Routinen verwenden, so hilft Ihnen das folgende Programm weiter. Es benötigt allerdings die von Commodore mitgelieferte Extras-Diskette, bzw. die auf ihr enthaltenen fd.-Dateien. Aus diesen Informationen entwickelt es ein GFA-Makro der gewünschten Funktion. Zunächst das Listing:

```
' Betriebssystem-Bibliotheksfunktionen untersuchen
' "Extras"-Diskette (bzw. fd.-Dateien) erforderlich
'
OPENW 0
bibliotheken_holen
DO
  SLEEP
LOOP
'
'
'
PROCEDURE bibliotheken_holen
  count=2
  volume$="Extras 1.3:"
  inhalt$="fd1.3"
  work$=volume$+inhalt$
  CHDIR work$
  DIR work$ TO "RAM:bibliotheken.user"
  DIM biblio$(200)
  DIM store$(200)
  DIM func$(30)
  OPEN "I",#1,"RAM:bibliotheken.user"
  WHILE NOT EOF(#1)
    LINE INPUT #1,in$
    in$=TRIM$(in$)
    IF RIGHT$(in$,7)="_lib.fd"
      INC count
      biblio$(count)=LEFT$(in$,LEN(in$)-7)
    ENDIF
  WEND
  CLOSE #1
  biblio$(0)="Bibliotheken"
  biblio$(1)="QUIT"
  biblio$(2)="MAKE MAKRO"
  biblio$(count+1)=""
  biblio$(count+2)=""
  count=count+1
  MENU biblio$()
  ON MENU GOSUB auswertung
  KILL "RAM:bibliotheken.user"
  PRINT "Programm initialisiert. Das Menue steht Ihnen";
  PRINT " jetzt zur Verfuegung."
RETURN
```

```
PROCEDURE auswertung
  eintrag=MENU(0)
  IF eintrag<19
    IF eintrag=1
      END
    ELSE IF eintrag=2
      make_makro
    ELSE
     bibliothek1$=biblio$(eintrag)
     CLS
     PRINT "Lade die Funktionsdefinitionen der Bibliothek ";bibliothek1$
     PRINT
     bibliothek$=bibliothek1$+"_lib.fd"
     OPEN "I",#1,bibliothek$
     fix=25
     fax=fix+1
     offset=fix
     position=0
     faktor=-1
     WHILE NOT EOF(#1)
        LINE INPUT #1,in$
        PRINT in$
        IF LEFT$(in$,1)<>"*" AND LEFT$(in$,1)<>"#"
          INC offset
          INC position
          IF offset>fix
            INC faktor
            offset=2
            biblio$(count+faktor*fax+0)=""
            biblio$(count+faktor*fax+1)="Funktionen "
          ENDIF
          finder=INSTR(in$,"(")
          biblio$(count+offset+fax*faktor)=LEFT$(in$,finder-1)
          store$(position)=in$
        ELSE
          temp=INSTR(in$,"base")
          IF temp>0
            basis$=MID$(in$,temp+5,4)+"Base"
          ENDIF
        ENDIF
      WEND
      CLOSE #1
      PRINT "Ladevorgang beendet."
      biblio$(count+offset+fax*faktor+1)=""
      biblio$(count+offset+fax*faktor+2)=""
      MENU biblio$()
    ENDIF
  ELSE
    feld$=biblio$(eintrag)
    found$=""
    FOR loop=1 TO position
      IF INSTR(store$(loop),feld$)=1
        found$=store$(loop)
```

```
        lib_offset=24+6*loop
        loop=position
      ENDIF
    NEXT loop
    CLS
    PRINT "Name der Routine: ";feld$
    PRINT
    PRINT "Basis = ";basis$
    PRINT "Offset = -";lib_offset
    PRINT "Template: ";
    check(found$)
    func=VAL(func$(1))
    PRINT func$(0);" (";
    IF func>0
      FOR loop=1 TO func
        PRINT func$(1+loop);
        IF loop<func
          PRINT ",";
        ELSE
          PRINT ")"
        ENDIF
      NEXT loop
      PRINT
      PRINT
      PRINT "Registerzuordnung:"
      PRINT
      FOR loop=1 TO func
        PRINT func$(1+func+loop);" = ";func$(1+loop)
      NEXT loop
    ELSE
      PRINT ")"
    ENDIF
  ENDIF
RETURN
PROCEDURE make_makro
  IF feld$<>""
    datei$="RAM:"+UPPER$(feld$)+".LST"
    OPEN "O",#1,datei$
    PRINT #1,"DIM reg%(16)"
    IF basis$<>"_SysBase" AND basis$<>"_IntBase" AND basis$<>"_GfxBase"
      basis$=RIGHT$(basis$,LEN(basis$)-1)
      PRINT #1,"bibliothek$="+CHR$(34);bibliothek1$;".library";
      PRINT #1,CHR$(34);"+chr$(0)"
      PRINT #1,basis$;"=OpenLibrary(VARPTR(bibliothek$),0)"
      PRINT #1,"IF ";basis$;"=0"
      PRINT #1,"  PRINT "+CHR$(34);bibliothek1$;
      PRINT #1," konnte nicht geoeffnet werden."
      PRINT #1,"  PRINT "+CHR$(34);"Ist ";bibliothek1$;
      PRINT #1," vielleicht ein Device?"
      PRINT #1,"ENDIF"
    ENDIF
    PRINT #1,"FUNCTION ";feld$;
    IF func>0
```

```
    IF func>0
      PRINT #1,"(";
      FOR loop=1 TO func
        PRINT #1,func$(1+loop);"%";
        IF loop<func
          PRINT #1,",";
        ELSE
          PRINT #1,")"
        ENDIF
      NEXT loop
      FOR loop=1 TO func
        IF UPPER$(LEFT$(func$(1+func+loop),1))="A"
          adder=8
        ELSE
          adder=0
        ENDIF
        reg=VAL(RIGHT$(func$(1+func+loop),1))
        reg=reg+adder
        PRINT #1,"  reg%(";reg;") = ";func$(1+loop);"%"
      NEXT loop
    ELSE
      PRINT #1," "
    ENDIF
    IF basis$="_SysBase"
      basis$="LPEEK(4)"
    ENDIF
    PRINT #1,"  reg%(14)=";basis$
    PRINT #1,"  RCALL ";basis$;"-";lib_offset;",reg%()"
    PRINT #1,"  RETURN reg%(0)"
    PRINT #1,"ENDFUNC"
    CLOSE #1
    PRINT "MACRO DONE."
  ELSE
    PRINT "MACRO FAILED - CHOOSE FUNCTION FIRST."
  ENDIF
RETURN
> PROCEDURE check(k$)
  flag=0
  index=0
  ERASE func$()
  DIM func$(30)
  FOR loop=1 TO LEN(k$)
    c$=MID$(k$,loop,1)
    IF flag=2
      IF c$<>"(" AND c$<>")"
        IF c$="/" OR c$=","
          INC index
        ELSE
          func$(index)=func$(index)+c$
        ENDIF
      ENDIF
    ELSE IF flag=1
      IF c$<>")"
```

```
        IF c$=","
          INC index
        ELSE
          func$(index)=func$(index)+c$
        ENDIF
      ELSE
        flag=2
        func$(1)=STR$(index-1)
        INC index
      ENDIF
    ELSE IF flag=0
      IF c$<>"("
        func$(index)=func$(index)+c$
      ELSE
        flag=1
        INC index
        INC index
      ENDIF
    ENDIF
  NEXT loop
  IF func$(2)=""
    func$(1)="0"
  ENDIF
RETURN
```

Nun zur Anwendung des "Makro-Makers". Tippen Sie das Programm ab. Nachdem Sie das Listing sicherheitshalber auf Diskette abgespeichert haben, können Sie es starten. Falls Sie noch nicht die Extras-Diskette eingelegt haben, werden Sie hierzu aufgefordert. Wie aus dem Listing unschwer zu entnehmen ist (zweite Zeile der Prozedur "bibliotheken_holen"), verlangt das Programm die Extras-Diskette "Extras 1.3". Sollten Sie Ihre Extras-Diskette umbenannt haben oder es sich um die Version 1.2 handeln, ist es ein leichtes Unterfangen, diese Zeile Ihrem Diskettennamen anzupassen.

Anschließend lädt das Programm alle auf dieser Diskette im Directory "fd1.3" gespeicherten Bibliotheksnamen ein. Auch hier müssen Sie das "fd1.3" in "fd" oder "fd1.2" ändern, wenn Sie eine ältere Extras-Diskette besitzen, da das Programm das gesuchte Verzeichnis sonst natürlich nicht finden kann.

Sobald der Ladevorgang abgeschlossen ist, steht Ihnen ein Menü zur Verfügung. Es besitzt den Punkt "Bibliotheken". Wählen Sie ihn einfach einmal an. Neben den Funktionen "QUIT" und

"MACRO" finden Sie dort alle zur Verfügung stehenden Bibliotheken. Sie können nun eine beliebige davon anklicken.

Auf dem Bildschirm werden die jetzt eingelesenen Daten sichtbar dargestellt. Ist der Lesevorgang abgeschlossen, finden Sie die erweiterten Menü-Punkte "Funktionen". Sie enthalten alle Routinen der zuvor angeklickten Bibliothek. Nun können Sie sich eine beliebige Routine herauswählen, indem Sie sie anklicken. Prompt wird der korrekte Routinenname, der Aufruf, die Bibliothek und das Template sowie die verwendeten Eingaberegister auf dem Bildschirm ausgegeben. Dies soll nur Ihrer Information dienen. Sie können nun jedoch auch den unter "Bibliotheken" zu findenden Menüpunkt "MACRO" anwählen. In Sekundenschnelle erstellt das Programm ein GFA-Makro, das den Aufruf dieser Routine ermöglicht. Auf diese Weise können Sie beliebig viele Routinen aus beliebigen Bibliotheken in Makros umwandeln.

Sobald Sie genügend Routinen in Makros verwandelt haben, können Sie das Programm über den Menüpunkt "QUIT" verlassen. Geben Sie jetzt "NEW" ein, um den GFA-Programmspeicher zu löschen. Die Makros befinden sich nun in der RAM-Disk als ASCII-Dateien. Wählen Sie den GFA-Menüpunkt "MERGE" an, anschließend das Laufwerk "RAM:". Jetzt können Sie alle gewünschten Makros miteinander kombinieren, indem Sie sie nacheinander mittels MERGE einladen.

Da die Makros als Funktionen deklariert sind, erfolgt ihr Aufruf so:

```
rückwert=@Funktionsname(parameter1,...,parameterx)
```

bzw. am Beispiel der AllocMem-Routine:

```
mem=@allocmem(100,2^16)
```

Ist Ihnen der Rückwert egal, so besteht alternativ die Möglichkeit:

```
VOID Funktion(parameter1,...,parameterx)
bzw.
~ Funktion(s.o.)
```

Anmerkung: Wenn Ihr Makro keine Bibliothek öffnet (einige Zeilen vor dem Makro), so steht Ihnen dieselbe Funktion auch ohne Makro zur Verfügung.

3.1.2 Umwandlung GFA-AmigaBASIC

Beim Aufruf von internen Routinen des Betriebssystems werden oft Standardparameter benötigt, die je nach BASIC-Dialekt auf unterschiedliche Weise ermittelt werden können.

Die Grafikbibliothek verwendet bei fast jedem Aufruf den Parameter "Rastport". AmigaBASIC liefert den Rastport des aktuellen Fensters immer in der Variablen WINDOW(8). GFA-BASIC liefert diesen Wert nur indirekt auf folgende Weise:

```
OPENW 0
rastport=LPEEK(WINDOW(0)+50)
```

Wurde ein anderes Fenster als Nr. 0 geöffnet, ändert sich die Zahl in der zweiten Zeile entsprechend.

Ein in der Intuition-Bibliothek häufig gebrauchter Wert ist der Zeiger auf die aktuelle Fensterstruktur. Dieser Zeiger liegt bei AmigaBASIC in der Variablen WINDOW(7). GFA liefert denselben Wert in der Variablen WINDOW(x), wobei x die Nummer des gewünschten und zuvor mittels OPENW x geöffneten Fensters ist.

Oftmals ist es bei einem Systemaufruf nötig, Textstrings zu übergeben. Hierzu übergibt man lediglich die Anfangsadresse der Speicherstelle, ab der der Text hinterlegt ist. Als Textende-Kennzeichen dient hierbei ein CHR$(0) am Ende des Textes.

AmigaBASIC liefert die Anfangsadresse eines einfachen Strings mit Hilfe der Funktion SADD:

```
Anfangsadresse=SADD(a$)
```

Anders bei GFA. Hier leistet die Funktion VARPTR dieselbe Aufgabe:

```
Anfangsadresse=VARPTR(a$)
```

oder:

```
Anfangsadresse=LPEEK(*a$)
```

Der Ausgleich dieser wenigen Unterschiede reicht in den meisten Fällen aus, unter AmigaBASIC verwendete Systemaufrufe in GFA zu verwenden (und umgekehrt).

3.2 AmigaBASIC-Grafik

Sicherlich werden Sie an dieser Stelle keine ausführliche Einführung in das Grafiksystem des Amiga finden, denn dazu ist dieser Spezialbereich viel zu komplex und umfangreich (schauen Sie doch einmal in das Amiga Supergrafikbuch von DATA BECKER, wenn Sie dieses Thema ganz besonders reizt!). Vielmehr möchten wir Ihnen kleine Tips und Tricks vorstellen, mit denen Sie eigene Programme schneller, professioneller und oftmals wesentlich vielseitiger gestalten können. Sehen Sie selbst:

3.2.1 Zeichenmodi verändern

Ob Sie es bemerkt haben oder nicht: Ihr Amiga verfügt über insgesamt vier verschiedene Zeichenmodi. Jeder Zeichenmodus stellt Grafiken auf seine ganz spezielle Art auf dem Bildschirm dar. Dies sind die Darstellungsarten:

JAM 1: Nur die eigentliche Zeichenfarbe wird auf den Bildschirm gebracht. Der Hintergrund bleibt unbeschädigt.

JAM 2: Wie JAM 1, jedoch werden alle ungesetzten Punkte im Zielgebiet mit der Hintergrundfarbe gezeichnet. Der Hintergrund wird also gelöscht.

INVERSEVID: Wie JAM 2, jedoch werden Hinter- und Vordergrund vertauscht. Die Zeichnung erscheint invers.

COMPLEMENT: Wie JAM 1, jedoch wird die neue Zeichnung mit dem Hintergrund Punkt für Punkt verknüpft. Ein gesetzter Punkt wird gelöscht, ein gelöschter gesetzt.

Diese vier Modi lassen sich außerdem untereinander mischen, wobei neun Kombinationsmöglichkeiten entstehen. Leider kennt AmigaBASIC keinen Befehl, um diese Zeichenmodi willkürlich zu verändern. Deshalb muß der entsprechende Befehl aus der internen Grafik-Bibliothek aktiviert werden. Er hat das Format:

```
SetDrMd (rastport, modus)
```

Die Adresse des Rastports ist für das aktuelle Ausgabefenster immer in der Variablen WINDOW(8) gespeichert. Für BASIC sieht das Format also so aus:

```
SetDrMd (WINDOW(8), modus)
```

Hier nun eine Sammlung kleiner Routinen, die die Verwendung des SetDrMd()-Befehls veranschaulichen:

```
'##############################
'#                            #
'# Programm: Zeichenmodi      #
'# Autor:    tob              #
'# Datum:    3.8.87           #
'# Version:  1.0              #
'#                            #
'##############################
LIBRARY "graphics.library"
  Schatten "Guten Abend, liebe Leute!", 11
  LOCATE 4,8
  Outline  "OUTLINE: Zum Hervorheben gut geeignet.", 10
```

```
  LIBRARY CLOSE
  END
SUB Schatten (text$, space%) STATIC
  cX% = POS(0)*8
  cY% = (CSRLIN - 1)*8
  IF cY% < 8 THEN cY% = 8
  CALL SetDrMd(WINDOW(8),0) 'JAM1
  FOR loop% = 1 TO LEN(text$)
    in$ = MID$(text$, loop%, 1)
    CALL Move(WINDOW(8), cX%+1, cY%+1)
    COLOR 2, 0
    PRINT in$;
    CALL Move(WINDOW(8), cX%, cY%)
    COLOR 1, 0
    PRINT in$;
    cX% = cX% + space%
  NEXT loop%
  CALL SetDrMd(WINDOW(8),1) 'JAM2
  PRINT
END SUB
SUB Outline (text$, space%) STATIC
  cX% = POS(0)*8
  cY% = (CSRLIN - 1)*8
  IF cY% < 8 THEN cY% = 8
  FOR loop% = 1 TO LEN(text$)
    in$ = MID$(text$, loop%, 1)
    CALL SetDrMd(WINDOW(8),0) 'JAM1
    FOR loop1% = -1 TO 1
      FOR loop2% = -1 TO 1
        CALL Move(WINDOW(8),cX% + loop2%, cY% + loop1%)
        PRINT in$;
      NEXT loop2%
    NEXT loop1%
    CALL SetDrMd(WINDOW(8),2) 'COMPLEMENT
    CALL Move(WINDOW(8), cX%, cY%)
    PRINT in$;
    cX% = cX% + space%
  NEXT loop%
  CALL SetDrMd(WINDOW(8),1) 'JAM2
  PRINT
END SUB
```

Eine weitere Anwendung demonstriert noch einmal den COMPLEMENT-Modus: Das Rubberbanding. Was, Sie wissen nicht, was das ist? Dabei arbeiten Sie damit sicherlich täglich: Jedesmal, wenn Sie die Größe eines Windows verändern, erscheint dieses erfrischend orange Gummiband, mit dessen Hilfe Sie eine passende Größe finden können.

Dieses Gummiband wird normalerweise von Intuition verwaltet. Die Technik ist recht einfach: Um zu verhindern, daß durch das Gummiband der Bildhintergrund verändert wird, friert Intuition zunächst alle Bildaktivitäten ein (Dies ist der Grund, warum ein Malprogramm zum Beispiel die Arbeit unterbricht, wenn Sie ein Window vergrößern oder verkleinern). Das Gummiband wird anschließend im Zeichen-Modus COMPLEMENT auf den Bildschirm gezeichnet. So kann es durch einfaches Überschreiben problemlos wieder gelöscht werden, ohne den Bildhintergrund zu verändern.

Dieser Effekt läßt sich auch von BASIC aus recht einfach programmieren. Das folgende Programm ermöglicht einen Einblick und benutzt auch gleich noch ein paar andere interessante AmigaBASIC-Befehle. Professionelles "Rubberbanding" ist kein Problem mehr:

```
'##############################
'#                            #
'# Programm: Rubberbanding    #
'# Autor:    tob              #
'# Datum:    3.8.87           #
'# Version:  2.0              #
'#                            #
'##############################
LIBRARY "graphics.library"
main:     '* Demonstriert Rubberbanding
          CLS
          '* Rechteck
          PRINT "a) Zeichnen Sie ein Rechteck!"
          Rubberband
          LINE (m.x, m.y) - (m.s, m.t),,b
          '* Linie
          LOCATE 1,1
          PRINT "b) ...und nun eine Linie!       "
          Rubberband
          LINE (m.x, m.y) - (m.s, m.t)
          '* Flaeche bestimmen
          LOCATE 1,1
          PRINT "c) Nun wird eine Flaeche bestimmt..."
          Rubberband
          x = ABS(m.x - m.s)
          y = ABS(m.y - m.t)
          PRINT "Breite (x) = ";x
          PRINT "Hoehe  (y) = ";y
          PRINT "Inhalt     = ";x*y; " Punkte."
          LIBRARY CLOSE
```

```
        END
SUB Rubberband STATIC
    SHARED m.x, m.y, m.s, m.t
    CALL SetDrMd(WINDOW(8),2) 'COMPLEMENT
    WHILE MOUSE(0) = 0
      maus = MOUSE(0)
    WEND
    m.x = MOUSE(1)
    m.y = MOUSE(2)
    m.s = m.x
    m.t = m.y
    WHILE maus < 1
      m.a = m.s
      m.b = m.t
      m.s = MOUSE(1)
      m.t = MOUSE(2)
      IF m.a <> m.s OR m.b <> m.t THEN
        LINE (m.x, m.y) - (m.a, m.b),,b
        LINE (m.x, m.y) - (m.s, m.t),,b
      END IF
      maus = MOUSE(0)
    WEND
    LINE (m.x, m.y) - (m.s, m.t),,b
    PSET (m.x, m.y)
    CALL SetDrMd(WINDOW(8), 1)
END SUB
```

3.2.2 Veränderter Text-Stil

Der Amiga ist in der Lage, Schrift durch Rechenoperationen zu verändern. Einfache Algorithmen (Rechenvorschriften) können so die Schriftarten "Unterstreichen", "Fettdruck" und "Kursivdruck" erzeugen. Vor allem in Textverarbeitungen, aber auch zur besseren Aufgliederung jeglichen Textes sind diese Modi außerordentlich nützlich. Leider unterstützt BASIC die Schriftverformung von Hause aus nicht. Wieder ist eine Systemfunktion aus der Grafik-Bibliothek der Retter in der Not:

```
SetSoftStyle (WINDOW(8), stil, enable)
```

stil:

0 normal (reset)
1 unterstrichen
2 fett
3 unterstrichen + fett

4 kursiv
5 unterstrichen + kursiv
6 fett + kursiv
7 unterstrichen, fett + kursiv

Das folgende Programm demonstriert die Möglichkeiten:

```
'#################################
'#                               #
'# Programm: Text-Stil           #
'# Autor:    tob                 #
'# Datum:    12.8.87             #
'# Version:  1.0                 #
'#                               #
'#################################
DECLARE FUNCTION AskSoftStyle% LIBRARY
DECLARE FUNCTION SetSoftStyle% LIBRARY
LIBRARY "graphics.library"
var:  '* Die Modi (die Zahlenwerte koennen auch
      '* direkt verwendet werden)
       normal%   = 0
       unterstr% = 1
       fett%     = 2
       kursiv%   = 4
demo: '* Beispiel
       CLS
       Stil unterstr% + kursiv%
       PRINT TAB(20); "Die verschiedenen Schriftarten"
       LOCATE 5,1
       Stil normal%
       PRINT "Ihr Amiga ist in der Lage, vier verschiedene Schriftarten"
       PRINT "darzustellen. Das sind:"
       PRINT
       PRINT "a) NORMALnormal"
       Stil unterstr%
       PRINT "b) UNTERSTRICHENunterstrichen"
       Stil fett%
       PRINT "c) FETTfett"
       Stil kursiv%
       PRINT "d) KURSIVkursiv"
       PRINT
       Stil normal%
       PRINT "Hier alle Formen, inkl. Mischformen:
       FOR loop% = 0 TO 7
         Stil loop%
         PRINT "Mischform Nr. ";loop%
       NEXT loop%
       * ...und mit "Normal" beenden...
       Stil normal%
       LIBRARY CLOSE
```

```
      END
SUB Stil (nr%) STATIC
    bits% = AskSoftStyle%(WINDOW(8))
    news% = SetSoftStyle%(WINDOW(8), nr%, bits%)
    IF (nr% AND 4) = 4 THEN
      CALL SetDrMd(WINDOW(8),0)
    ELSE
      CALL SetDrMd(WINDOW(8),1)
    END IF
END SUB
```

Variablen

bits%	diejenigen Stil-Bits, die mit diesem Zeichensatz möglich sind
news%	neu gesetzte Stil-Bits
nr%	eingegebene Stil-Bits

Programmbeschreibung

Denkbar einfach: Sie übergeben an das SUB "Stil" die Bitkombinationen des Stils (siehe oben). Intern wird die AskSoftStyle()-Routine aufgerufen, die die möglichen Bits für den augenblicklichen Zeichensatz zurückgibt. Diese Bits werden zusammen mit den neu zu setzenden Bits an SetSoftStyle() übergeben. Diese Funktion setzt den neuen Stil, wenn die entsprechenden Maskenbits in bits% vorhanden sind. Ansonsten werden die Bits nicht gesetzt.

Ist in irgendeiner Kombination der Stil "Kursiv" (nr% and 4 = 4) gewählt worden, dann wird der Zeichenmodus "JAM1" eingeschaltet (siehe Kapitel 3.2.1). Nur so wird kursiver Text einwandfrei ausgegeben, weil im normalen Modus JAM2 jeweils der rechte, in das Feld des nächsten Buchstabens hineinreichende Teil des Zeichens beschädigt (abgehackt) wird. Wurde der Punkt "Kursiv" nicht benutzt, setzt SetDrMd() den Normalmodus "JAM2".

3.2.3 Move - Kontrolle über den AmigaBASIC-Cursor

In einigen der vorangegangenen Befehle haben wir ihn schon benutzt, den "graphics.library"-Befehl Move. Kennt AmigaBASIC nur die Möglichkeit, die Cursorposition zeichenweise (LOCATE-Befehl) oder pixelweise in x-Richtung (durch PTAB) zu verändern, so läßt sich die Cursorposition mit Hilfe des Move-Befehls problemlos pixelweise sowohl in x- als auch y-Richtung verschieben. Der Aufruf des Befehls in BASIC muß folgendermaßen erfolgen:

```
Move&(WINDOW(8),x%,y%)
```

Um die Sache zu vereinfachen, haben wir wieder einen kleinen Befehl geschrieben, der in den verschiedensten Situationen äußerst praktisch sein kann:

```
xyPTAB x%,y%
```

Programm-Größe: 455 Bytes
Bemerkung: "graphics.bmap" muß sich auf Disk befinden.

```
DECLARE FUNCTION Move& LIBRARY
LIBRARY "graphics.library"
var:
text$="Es geht bergab..."
text$=" "+text$+" "
empty$=SPACE$(LEN(text$))
fontheight%=8
main:
FOR y%=6 TO 100
  xyPTAB x%,y%
  PRINT text$
  xyPTAB x%,y%-fontheight%
  PRINT empty$
  x%=x%+1
NEXT y%
LIBRARY CLOSE
END
'---------------------------------
SUB xyPTAB(x%,y%) STATIC
  e&=Move&(WINDOW(8),x%,y%)
END SUB
```

Fehlerquellen: Kaum möglich

Variablen

text$	Demo-Text
empty$	Leerstring, der für ein restloses Verschwinden beim y-Verschieben sorgt
fontheight%	Höhe des Font
x%, y%	Bildschirm-Koordinaten
e&	Error-Meldung des Move&-Befehls

Programmbeschreibung

Der Move&-Befehl wird als Funktion deklariert, und die einschlägige Library wird geöffnet. Der Demo-Text huscht im SoftScroll-Mode über den Bildschirm, die Library wird wieder dichtgemacht, und das Programm endet. Die eigentliche SUB-Routine ist denkbar einfach, denn im Prinzip werden die nötigen Koordinaten nur dem Move-Befehl übergeben.

So simpel diese Routine aussieht, so leistungsfähig ist sie. Mit ihrer Hilfe kann Text, wie im Beispiel, pixelweise in sämtliche Himmelsrichtungen verschoben werden. Entweder im Smear-Effekt (SetDrMd mode%=JAM1) oder als SoftScrolling (SetDrMd mode%=JAM2).

3.2.4 Schnelleres Grafikformat <-> IFF

Das Schöne am IFF-ILBM-Dateiformat ist zweifellos die Tatsache, daß es sich schnell als Standard durchgesetzt hat. Die Konzeption von IFF ist geradezu einzigartig. Sie läßt neben den wenigen allgemeinverständlichen Dateiteilen auch jede Erweiterung von Programmen in Form weiterer Dateiteile zu. Die einzelnen Dateiblöcke nennt man Chunks.

Sicherlich haben Sie bereits viele Ladeprogramme für ILBM-Bilder in Zeitschriften gesehen, oder gar den Schnellader für IFF-Grafiken im DATA-BECKER-Buch "Supergrafik Amiga", der von Tobias Weltner stammt. Wenn Sie sich aber - bei den Grafikfähigkeiten des Amiga - einmal damit befaßt haben, für

ein eigenes Programm ein Titelbild zu laden, kann ich mir gut vorstellen, daß Ihnen die User schon während der langen Ladezeit davongelaufen sind, richtig?

Wir sollten uns daher ernsthafte Gedanken darüber machen, warum das so sein muß. IFF-ILBM-Grafiken haben entscheidende Nachteile, wenn es darum geht, sie schnell zu laden. Der erste Punkt ist der, daß sehr viel Zeit damit verloren geht, die verschiedenen Chunks zu identifizieren und unwichtige zu überlesen. Der zweite Punkt sind die verschiedenen Methoden, wie ein Bild schließlich gespeichert wurde, also das ILBM-Format. Hierbei wird beispielsweise bei einem Bild mit fünf Bitplanes zunächst Zeile 1 der ersten Bitplane gespeichert, dann folgt Zeile 1 der zweiten Bitplane und so weiter, bis schließlich Zeile 1 der fünften Bitplane an die Reihe kommt. Wenn Sie dazu noch wissen, daß eine Bitplane immer in einem Stück im Speicher liegt, können Sie sich wohl nur wenige Anwendungen vorstellen, bei denen eine derartige Umständlichkeit von Nutzen sein kann. Der dritte Punkt kommt beispielsweise bei Deluxe-Paint II dazu. Hier wird zusätzlich jede Zeile einer Bitplane gepackt und muß daher beim Laden wieder decodiert werden.

Aus diesen Gründen werden Sie bei keinem Profi-Programm, welches nicht IFF-Files laden können soll, ein derartiges Dateiformat vorfinden. Ein Profi *will* ja zum einen nicht, daß seine Bilder (z.B.: Defender of the Crown) zu anderen Programmen kompatibel sind, zum anderen legt er äußersten Wert darauf, seine Bilder *schnell* in den Speicher zu bekommen. Selbst ein Malprogramm muß seine Bilder nicht im langsamen Standardformat speichern und laden, obwohl hier die Möglichkeit dazu vorhanden sein sollte.

Da man als Programmierer seinen Programmen, auch wenn sie in AmigaBASIC geschrieben wurden, einen professionellen Touch mitgeben möchte, schrieb ich folgendes Programm. Dieses Programm liest zunächst eine IFF-ILBM-Grafik ein (Wer will schließlich auf DPaint als Malprogramm verzichten?) und speichert sie danach in folgendem Format ab:

```
        Bitplane 1 (in einem Stück)
        Bitplane 2 ...
    ... letzte Bitplane
        Hardware-Farbregister-Inhalt
```

Dazu wird ein AmigaBASIC-Ladeprogramm generiert, welches dieses Bild lädt und anzeigt (bis zum Maus-Klick). Das Amiga-BASIC-Programm ist natürlich ein ASCII-File, das sich sowohl mit MERGE oder CHAIN mit anderen Programmen verknüpfen läßt, als auch durch ein Anklicken des Icons von der Workbench aus startbar ist.

In dem folgenden Listing ist ein Schnellader für IFF-ILBM-Grafiken eingebaut. Vergleichen Sie einmal! Bei eigenen Tests des neuen Formates ergab sich unter Verwendung eines Bildes im Format 320 mal 200 mal 5 eine reine Ladegeschwindigkeit von MEHR ALS 41000 BYTE PRO SEKUNDE !

Zu gut deutsch: Das Bild war nach einer LADEZEIT VON UNTER EINER SEKUNDE im Speicher. Im Gegensatz dazu brauchte das Laden des IFF-Files etwa 100mal solange! Hier nun das Listing:

```
' ####################################
' #   load pictures like a prof with   #
' #------------------------------------#
' #    F A S T - G F X    A m i g a    #
' #------------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
  DECLARE FUNCTION xOpen& LIBRARY
  DECLARE FUNCTION xRead& LIBRARY
  DECLARE FUNCTION xWrite& LIBRARY
  DECLARE FUNCTION Seek& LIBRARY
  DECLARE FUNCTION AllocMem& LIBRARY
  DECLARE FUNCTION AllocRaster& LIBRARY

REM **** LIBRARYS OEFFNEN **********************
  LIBRARY "dos.library"
  LIBRARY "exec.library"
  LIBRARY "graphics.library"

REM **** FEHLERVERZWEIGUNG EINRICHTEN ***********
  ON ERROR GOTO errorcheck
```

```
REM **** EINGABE DER DATEINAMEN *****************
eingabe:

REM **** SPEICHERPLATZ VOM BASIC-WINDOW   *******
REM **** FREIGEBEN UND MINISCREEN OEFFNEN *******
  WINDOW CLOSE WINDOW(0)
  SCREEN 1,320,31,1,1
  WINDOW 1,"FAST-GFX-CONVERTER",,0,1
  PALETTE 0,0,0,0
  PALETTE 1,1,0,0
  FOR i=1 TO 4
    MENU i,0,0,""
  NEXT
  PRINT "IFF-ILBM-Bild:"
  LINE INPUT filename$
  PRINT "Fast-GFX-Bild:"
  LINE INPUT target$
  PRINT "Name des Loaders:"
  LINE INPUT loader$
  CHDIR "df0:"

REM **** IFF-DATEI OEFFNEN **********************
  file$=filename$+CHR$(0)
  handle&=xOpen&(SADD(file$),1005)
  IF handle&=0 THEN ERROR 255

REM **** EINGABE-BUFFER SCHAFFEN ****************
  buffer&=AllocMem&(160,65537&)
  IF buffer&=0 THEN ERROR 254
  farbbuffer&=buffer&+96

REM **** FORM-CHUNK LESEN, BZW. PRUEFEN *********
  r&=xRead&(handle&,buffer&,12)
  IF PEEKL(buffer&)<>1179603533& THEN ERROR 253
  IF PEEKL(buffer&+8)<>1229734477& THEN ERROR 252
  bmhdflag%=0
  flag%=0

REM **** LESE CHUNKNAME + CHUNKLAENGE ***********
  WHILE flag%<>1
    r&=xRead&(handle&,buffer&,8)
    IF r&<8 THEN flag%=1:GOTO whileend

    lang&=PEEKL(buffer&+4)

REM **** BMHD-CHUNK? (CVL("BMHD")) **************
    IF PEEKL(buffer&)=1112361028& THEN

      r&=xRead&(handle&,buffer&,lang&)

      breite%=PEEKW(buffer&)      :REM * BILDBREITE
      hoehe%=PEEKW(buffer&+2)     :REM * BILDHOEHE
      tiefe%=PEEK(buffer&+8)      :REM * BILDTIEFE
```

```
      gepackt%=PEEK(buffer&+10)    :REM * PACK-STATUS
      sbreite%=PEEKW(buffer&+16)   :REM * SCREENBREITE
      shoehe%=PEEKW(buffer&+18)    :REM * SCREENHOEHE

      bytes%=(breite%-1)\8+1
      sbytes%=(sbreite%-1)\8+1
      colmax%=2^tiefe%
      IF colmax%>32 THEN colmax%=32
      IF breite%<321 THEN mode%=1 ELSE mode%=2
      IF hoehe%>256 THEN mode%=mode%+2
      IF tiefe%=6 THEN planedazu%=1 ELSE planedazu%=0

REM **** NEUEN SCREEN HOCHFAHREN ****************
      WINDOW CLOSE 1
      SCREEN CLOSE 1
      SCREEN 1,breite%,hoehe%,tiefe%-planedazu%,mode%
      WINDOW 1,,,0,1

REM **** SCREEN-DATEN ERMITTELN *****************
      picscreen&=PEEKL(WINDOW(7)+46)
      viewport&=picscreen&+44
      rastport&=picscreen&+84
      colormap&=PEEKL(viewport&+4)
      colors&=PEEKL(colormap&+4)
      bmap&=PEEKL(rastport&+4)

REM **** HALFBRIGHT ODER HOLD-AND-MODIFY ? ******
      IF planedazu%=1 THEN

REM **** DANN 6. BITPLANE ERSTELLEN LASSEN ******
        plane6&=AllocRaster&(sbreite%,shoehe%)
        IF plane6&=0 THEN ERROR 251

REM **** UND IN DIE DATENSTRUKTUR EINBINDEN *****
        POKE bmap&+5,6
        POKEL bmap&+28,plane6&

      END IF

      bmhdflag%=1

REM **** CMAP-CHUNK (FARBEN, JE FARBE: R,G,B) ***
    ELSEIF PEEKL(buffer&)=1129136464& THEN

      IF (lang& OR 1)=1 THEN lang&=lang&+1
      r&=xRead&(handle&,buffer&,lang&)

      FOR i%=0 TO colmax%-1

REM **** UMRECHNEN IN DIE FORM, IN DER SIE IN ***
REM **** DEN HARDWARE-REGISTERN LIEGEN        ***
        POKE farbbuffer&+i%*2,PEEK(buffer&+i%*3)/16
        gruenblau%=PEEK(buffer&+i%*3+1)
```

```
        gruenblau%=gruenblau%+PEEK(buffer&+i%*3+2)/16
        POKE farbbuffer&+i%*2+1,gruenblau%

      NEXT

REM **** CAMG-CHUNK = VIEWMODE (Z.B. HAM,LACE ***
    ELSEIF PEEKL(buffer&)=1128353095& THEN

      r&=xRead&(handle&,buffer&,lang&)

      viewmode&=PEEKL(buffer&)
REM **** BODY-CHUNK = BITMAPS, ZEILE FUER ZEILE *
    ELSEIF PEEKL(buffer&)=1112491097& THEN

REM **** EXISTIERT DIE SCREEN UEBERHAUPT ? ******
      IF bmhdflag%=0 THEN ERROR 250

REM **** SIND DIE EINZELNEN ZEILEN ETWA *********
REM **** AUCH NOCH CODIERT ?            *********
      IF gepackt%=1 THEN

REM **** DANN HILFT NUR NOCH POKEN !!! **********
        FOR y%=0 TO hoehe%-1
          FOR z%=0 TO tiefe%-1
            ad&=PEEKL(bmap&+8+4*z%)+y%*sbytes%
            count%=0
            WHILE count%<bytes%
              r&=xRead&(handle&,buffer&,1)
              code%=PEEK(buffer&)
              IF code%>128 THEN
                r&=xRead&(handle&,buffer&,1)
                wert%=PEEK(buffer&)
                endbyte%=count%+257-code%
                FOR x%=count% TO endbyte%
                  POKE ad&+x%,wert%
                NEXT
                count%=endbyte%
              ELSEIF code%<128 THEN
                r&=xRead&(handle&,ad&+count%,code%+1)
                count%=count%+code%+1
              END IF
            WEND
        NEXT z%,y%

REM **** ODER ETWA NICHT GEPACKT ? - WIE SCHOEN *
      ELSEIF gepackt%=0 THEN

REM **** DER DOS-BEFEHL READ FUELLT DIE BITMAPS *
        FOR y%=0 TO hoehe%-1
          FOR z%=0 TO tiefe%-1
            ad&=PEEKL(bmap&+8+4*z%)+y%*sbytes%
            r&=xRead&(handle&,ad&,bytes%)
        NEXT z%,y%
```

```
REM **** NANU? CODIERUNGS-METHODE UNBEKANNT? ****
      ELSE

        ERROR 249

      END IF

    ELSE

REM **** CHUNK KOENNEN WIR NICHT BRAUCHEN. ******
REM **** LESEZEIGER IN DATEI VERSCHIEBEN   ******
      IF (lang& OR 1)=1 THEN lang&=lang&+1
      now&=Seek&(handle&,lang&,0)

    END IF

REM **** ENDE DER LESEROUTINE ******************
whileend:

  WEND

REM **** FARBEN EINLADEN UND FILE SCHLIESSEN ****
  IF bmhdflag%=0 THEN ERROR 248
  CALL LoadRGB4(viewport&,farbbuffer&,colmax%)
  CALL xClose(handle&)

REM **** VIEWMODE GELESEN? DANN AUCH EINSTELLEN *
  IF viewmode&<>0 THEN
    POKEW viewport&+32,viewmode&
  END IF

REM **** ZIELDATEI OEFFNEN **********************
  file$=target$+CHR$(0)
  handle&=xOpen&(SADD(file$),1005)
  IF handle&=0 THEN
    handle&=xOpen&(SADD(file$),1006)
  END IF

REM ******************************************
REM **** SO KOENNEN SIE IN SEKUNDENSCHNELLE *****
REM **** GRAFIKEN ABSPEICHERN               *****

  bitmap&=sbytes%*hoehe%  :REM GROESSE EINER BITPLANE

  FOR i%=0 TO tiefe%-1
    ad&=PEEKL(PEEKL(WINDOW(8)+4)+8+4*i%)
    w&=xWrite&(handle&,ad&,bitmap&)
  NEXT

  w&=xWrite&(handle&,farbbuffer&,64)

REM **** DATEI SCHLIESSEN, BUFFER FREIGEBEN *****
  CALL xClose(handle&)
```

```
  CALL FreeMem(buffer&,160)

REM ********************************************
REM **** BASIC-PROGRAMM ERZEUGEN (ASCII-FORMAT) *
  OPEN loader$ FOR OUTPUT AS 1

  PRINT#1,"' ##################";CHR$(10);
  PRINT#1,"' # Fast-Gfx Loader #";CHR$(10);
  PRINT#1,"' #----------------#";CHR$(10);
  PRINT#1,"' # ";CHR$(169);"'87 S. Maelger #";CHR$(10);
  PRINT#1,"' ##################";CHR$(10);
  PRINT#1,CHR$(10);

REM **** BETRIEBSSYSTEM-ROUTINEN ZUM LADEN ******
  PRINT#1,"DECLARE FUNCTION xOpen& LIBRARY";CHR$(10);
  PRINT#1,"DECLARE FUNCTION xRead& LIBRARY";CHR$(10);
  PRINT#1,"DECLARE FUNCTION AllocMem& LIBRARY";CHR$(10);

REM **** IM FALLE VON H.A.M. ODER HALFBRIGHT ****
  IF tiefe%=6 THEN

    PRINT#1,"DECLARE FUNCTION AllocRaster& LIBRARY";
      PRINT#1,CHR$(10);

  END IF

REM **** BENOETIGTE LIBRARIES ******************
  PRINT#1,CHR$(10);
  PRINT#1,"LIBRARY ";CHR$(34);"dos.library";CHR$(34);
    PRINT#1,CHR$(10);
  PRINT#1,"LIBRARY ";CHR$(34);"exec.library";CHR$(34);
    PRINT#1,CHR$(10);
  PRINT#1,"LIBRARY ";CHR$(34);"graphics.library";CHR$(34);
    PRINT#1,CHR$(10);
  PRINT#1,CHR$(10);

REM **** SPEICHER FUER PALETTE RESERVIEREN ******
  PRINT#1,"b&=AllocMem&(64,65537&)";CHR$(10);
  PRINT#1,"IF b&=0 THEN ERROR 7";CHR$(10);

REM **** BILD-DATEI OEFFNEN *********************
  PRINT#1,"file$=";CHR$(34);target$;CHR$(34);"+CHR$(0)";
    PRINT#1,CHR$(10);
  PRINT#1,"h&=xOpen&(SADD(file$),1005)";CHR$(10);
REM **** SCREEN ERSTELLEN ***********************
  PRINT#1,"WINDOW CLOSE WINDOW(0)";CHR$(10);
  PRINT#1,"SCREEN 1,";MID$(STR$(sbreite%),2);",";
    PRINT#1,MID$(STR$(hoehe%),2);",";
    PRINT#1,MID$(STR$(tiefe%-planedazu%),2);",";
    PRINT#1,MID$(STR$(mode%),2);CHR$(10);
  PRINT#1,"WINDOW 1,,,0,1";CHR$(10);
  PRINT#1,"viewport&=PEEKL(WINDOW(7)+46)+44";CHR$(10);
REM **** ALLE FARBEN AUF NULL SETZEN ************
```

```
  lcm$="CALL LoadRGB4(viewport&,b&,"
  lcm$=lcm$+MID$(STR$(colmax%),2)+")"+CHR$(10)
  PRINT#1,lcm$;

REM **** BEI HAM ODER HALFBRIGHT 6.PLANE ********
  IF tiefe%=6 THEN

    PRINT#1,"n&=AllocRaster&(";
      PRINT#1,MID$(STR$(sbreite%),2);",";
      PRINT#1,MID$(STR$(hoehe%),2);")";CHR$(10);
    PRINT#1,"IF n&=0 THEN ERROR 7";CHR$(10);
    PRINT#1,"bmap&=PEEKL(PEEKL(WINDOW(7)+46)+88)";CHR$(10);
    PRINT#1,"POKE bmap&+5,6";CHR$(10);
    PRINT#1,"POKEL bmap&+28,n&";CHR$(10);
    PRINT#1,"POKEL viewport&+32,PEEKL(viewport&+32)OR 2^";

REM **** UND VIEWMODE EINSTELLEN ***************
    IF (viewmode& OR 2^7)=2^7 THEN

REM **** HALFBRIGHT-BIT SETZEN ******************
      PRINT#1,"7";

    ELSE

REM **** HOLD-AND-MODIFY - BIT SETZEN ***********
      PRINT#1,"11";

    END IF

      PRINT#1,CHR$(10);

  END IF

REM **** UND NUN DIE LADEROUTINE ****************
  PRINT#1,"FOR i%=0 TO";STR$(tiefe%-1);CHR$(10);
  PRINT#1,"  ad&=PEEKL(PEEKL(WINDOW(8)+4)+8+4*i%)";CHR$(10);
  PRINT#1,"  r&=xRead&(h&,ad&,";
    PRINT#1,MID$(STR$(bitmap&),2);"&)";CHR$(10);
  PRINT#1,"NEXT";CHR$(10);

REM **** PALETTE LESEN (GLEICH IN RICHTIGER FORM)
  PRINT#1,"r&=xRead&(h&,b&,64)";CHR$(10);

REM **** FILE WIEDER SCHLIESSEN *****************
  PRINT#1,"CALL xClose(h&)";CHR$(10);

REM **** FARBTABELLE SETZEN LASSEN **************
  PRINT#1,lcm$;

REM **** BUFFER FUER FARBEN WIEDER FREIGEBEN ****
  PRINT#1,"CALL FreeMem(b&,64)";CHR$(10);

REM **** LIBRARIES WIEDER SCHLIESSEN ************
```

```
  PRINT#1,"LIBRARY CLOSE";CHR$(10);

REM **** WARTEN AUF MOUSE-CLICK *****************
  PRINT#1,"WHILE MOUSE(0)<>0:WEND";CHR$(10);
  PRINT#1,"WHILE MOUSE(0)=0:WEND";CHR$(10);

REM **** SCREEN SCHLIESSEN UND BASIC-WINDOW *****
REM **** WIEDER AUF WORKBENCH-SCREEN HOLEN  *****
  PRINT#1,"WINDOW CLOSE 1";CHR$(10);
  PRINT#1,"SCREEN CLOSE 1";CHR$(10);
  PRINT#1,"WINDOW 1,";CHR$(34);"OK";CHR$(34);
    PRINT#1,",(0,11)-(310,185),0,-1";
    PRINT#1,CHR$(10);CHR$(10);

  CLOSE 1

REM **** ZURUECK ZUR WORKBENCH ******************
  WINDOW CLOSE 1
  SCREEN CLOSE 1
  WINDOW 1,,,0,-1
  PRINT "Creating Loader-Icon"

REM **** DATEN FUER SPEZIAL-ICON EINLESEN *******
  RESTORE icondatas

  file$=loader$+".info"+CHR$(0)

  a$=""
  FOR i%=1 TO 486
    READ b$
    a$=a$+CHR$(VAL("&H"+b$))
  NEXT

REM **** UND PER WRITE UEBER DAS DATEN-FILE- ****
REM **** ICON SCHREIBEN (MODE=OLDFILE)       ****
  h&=xOpen&(SADD(file$),1005)
  w&=xWrite&(h&,SADD(a$),498)

  CALL xClose(h&)

  LIBRARY CLOSE
  MENU RESET
  END

REM **** FEHLER-ABFANGEN ************************
errorcheck:

  n%=ERR

  IF n%=255 THEN
    PRINT "Bild nicht gefunden"
    GOTO goon
  ELSEIF n%=254 THEN
```

```
    PRINT "Nicht genug Speicher!"
    GOTO goon
  ELSEIF n%=253 OR n%=252 THEN
    PRINT "Kein IFF-ILBM-Bild!"
    GOTO goon
  ELSEIF n%=251 THEN
    PRINT "Kann keine 6.Plane hochziehen."
    GOTO goon
  ELSEIF n%=250 THEN
    PRINT "Kein BMHD-Chunk vorm BODY!"
    GOTO goon
  ELSEIF n%=249 THEN
    PRINT "Crunch-Algorythmus unbekannt"
    GOTO goon
  ELSEIF n%=248 THEN
    PRINT "Ich blicke nicht mehr durch"
    GOTO goon

  ELSE
    CLOSE
    CALL xClose(handle&)
    CALL FreeMem(buffer&,160)
    LIBRARY CLOSE
    MENU RESET
    ON ERROR GOTO 0
    ERROR n%
    STOP

  END IF

  STOP

goon:

  IF n%<>255 THEN
    CALL xClose(handle&)
    IF n%<>254 THEN CALL FreeMem(buffer&,160)
  END IF

  BEEP
  LIBRARY CLOSE
  RUN

icondatas:
DATA E3,10,0,1,0,0,0,0,0,0,0,0,0,2E,0,1F,0,5,0,3,0,1
DATA 0,1,BD,A0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,4,0,0,0,F2,98,0,0,0,0,80,0,0,0,80,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,10,0,0,0,0,0,0,2E,0,1F,0,2,0
DATA 2,B1,E0,3,0,0,0,0,0,0,0,0,0,0,3,FF,FF,FF,FF,0
DATA 3,0,0,0,3,0,2,0,0,0,1,0,2,0,0,0,1,0,2,7,80,0,1
DATA 0,2,1,F8,0,1,0,2,0,3F,C0,1,0,2,3,FC,0,1,0,2,0
DATA 1F,C0,1,0,2,0,1,FE,1,0,2,0,0,1F,F1,0,2,0,0,FF,1
DATA 0,3,0,1F,FE,3,0,3,FF,FF,FF,FF,0,0,0,6A,BF,F0,0
```

```
DATA 0,0,0,7,FE,0,0,0,0,0,FF,80,7F,EF,FF,FD,FF,F8,7F
DATA EF,FF,FD,E0,38,7F,EF,FF,FD,FF,F8,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,3E,7C,F9,B0,0,0,20,40
DATA 80,A0,0,0,3C,4C,F0,40,0,0,20,44,80,A0,0,0,20,7C
DATA 81,B0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,FF,FF,FF,FF,0
DATA 4,0,0,0,0,80,4,FF,FF,FF,FC,80,5,FF,FF,FF,FE,80
DATA 5,FF,FF,FF,FE,80,5,FF,FF,FF,FE,80,5,FF,FF,FF,FE
DATA 80,5,FF,FF,FF,FE,80,5,FF,FF,FF,FE,80,5,FF,FF,FF
DATA FE,80,5,FF,FF,FF,FE,80,5,FF,FF,FF,FE,80,5,FF,FF
DATA FF,FE,80,4,FF,FF,FF,FC,80,4,0,3,FF,80,80,7,FF
DATA 95,7F,FF,80,1,FF,FF,FF,FE,0,7F,FF,FF,FF,FF,F8
DATA 80,10,0,2,FF,84,80,10,0,2,7F,C4,B0,10,0,2,0,4
DATA 7F,FF,FF,FF,FF,FC,38,0,0,0,0,38,30,0,0,0,0,18,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,C,3A,41
DATA 6D,69,67,61,42,41,53,49,43,0
```

3.2.5 IFF-Brushes werden Objects

Sie besitzen ein Super-Malprogramm wie DeLuxe Paint? Was liegt denn dann näher, als es zum Erstellen von Sprites, VSprites, BOb's, AnimOb's oder was der Dinge mehr sind zu verwenden? Dazu müssen wir jedoch vorher eine Formatwandlung vornehmen. Dazu habe ich das folgende Programm geschrieben. Es wandelt jedes IFF-Bild in ein Object-File um. Voraussetzung hierfür ist jedoch, daß das Bild nicht zu groß für einen Object-String ist.

Danach wird dieses Object aktiviert und kurz bewegt. Da dabei keine speziellen Techniken der Speicherung des Hintergrundes verwendet werden, kann es bei zu vielen Bitplanes zu einem leichten Flimmereffekt kommen.

Nun aber zunächst das Programm:

```
' ####################################
' #  Use DPaint as Object-Editor with  #
' #----------------------------------#
' # B R U S H -  T R A N S F O R M E R #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
  CLEAR,30000&
  DIM r(31),g(31),b(31)
```

```
eingabe:
  PRINT "Brush-Name (+Pfad): ";
  LINE INPUT brush$
  PRINT
  PRINT "Object-Datei (+Pfad): ";
  LINE INPUT objectfile$
  PRINT
  PRINT "Farbdatei anlegen? (J/N) ";
warte:
  a$=LEFT$(UCASE$(INKEY$+CHR$(0)),1)
  IF a$="N" THEN
    PRINT "NEIN!"
  ELSEIF a$="J" THEN
    PRINT "Na klar."
    colorflag%=1
    PRINT
    PRINT "Farbdatei-Name (+Pfad): ";
    LINE INPUT colorfile$
  ELSE
    GOTO warte
  END IF
  PRINT

  OPEN brush$ FOR INPUT AS 1
  a$=INPUT$(4,1)
  IF a$<>"FORM" THEN CLOSE 1:RUN

  a$=INPUT$(4,1)
  a$=INPUT$(4,1)
  IF a$<>"ILBM" THEN CLOSE 1:RUN

lesechunk:
  a$=INPUT$(4,1)

  IF a$="BMHD" THEN
    PRINT "BMHD-Chunk gefunden."
    PRINT
    a$=INPUT$(4,1)
    breite%=ASC(INPUT$(1,1)+CHR$(0))*256
    breite%=breite%+ASC(INPUT$(1,1)+CHR$(0))
    PRINT "Bildbreite:";breite%;" Pixel"
    IF breite%>320 THEN
      PRINT "Ist mir zu breit."
      BEEP
      CLOSE 1
      RUN
    END IF
    hoehe%=ASC(INPUT$(1,1)+CHR$(0))*256
    hoehe%=hoehe%+ASC(INPUT$(1,1)+CHR$(0))
    PRINT "Bildhöhe  :";hoehe%;" Pixel"
    IF hoehe%>200 THEN
      PRINT "Ist mir zu hoch."
```

```
    BEEP
    CLOSE 1
    RUN
  END IF
  a$=INPUT$(4,1)
  planes%=ASC(INPUT$(1,1))
  PRINT "Bildtiefe :";planes%;" Planes"
  IF planes%>5 THEN
    PRINT "Zu viele Planes!"
    BEEP
    CLOSE 1
    RUN
  ELSEIF planes%*((breite%-1)\16+1)*2*hoehe%>32000 THEN
    PRINT "Hey! Zu viele Bytes fuer einen Object-String!"
    BEEP
    CLOSE 1
    RUN
  END IF
  a$=INPUT$(1,1)
  gepackt%=ASC(INPUT$(1,1)+CHR$(0))
  IF gepackt%=0 THEN
    PRINT "Packstatus: NICHT gepackt."
  ELSEIF gepackt%=1 THEN
    PRINT "Packstatus: ByteRun1-Algorythmus."
  ELSE
    PRINT "Packstatus: unbekannt"
    BEEP
    CLOSE 1
    RUN
  END IF
  a$=INPUT$(9,1)
  Status%=Status%+1
  PRINT
  PRINT

ELSEIF a$="CMAP" THEN
  PRINT "CMAP-Chunk gefunden."
  a$=INPUT$(3,1)
  l%=ASC(INPUT$(1,1))
  farben%=l%\3
  PRINT farben%;"Farben gefunden"
  FOR i%=0 TO farben%-1
    r(i%)=ASC(INPUT$(1,1)+CHR$(0))/255
    g(i%)=ASC(INPUT$(1,1)+CHR$(0))/255
    b(i%)=ASC(INPUT$(1,1)+CHR$(0))/255
  NEXT
  Status%=Status%+2
  PRINT
  PRINT
```

```
  ELSEIF a$="BODY" THEN
    PRINT "BODY-Chunk gefunden."
    PRINT
    a$=INPUT$(4,1)
    bytes%=(breite%-1)\8+1
    bmap%=bytes%*hoehe%
    obj$=STRING$(bytes%*hoehe%*planes%,0)
    FOR i%=0 TO hoehe%-1
      PRINT "Ich lese Zeile";i%+1
      FOR j%=0 TO planes%-1
        IF gepackt%=0 THEN
          FOR k%=1 TO bytes%
            a$=LEFT$(INPUT$(1,1)+CHR$(0),1)
            MID$(obj$,j%*bmap%+i%*bytes%+k%,1)=a$
          NEXT
        ELSE
          pointer%=1
          WHILE pointer%<bytes%+1
            a%=ASC(INPUT$(1,1)+CHR$(0))
            IF a%<128 THEN
              FOR k%=pointer% TO pointer%+a%
                a$=LEFT$(INPUT$(1,1)+CHR$(0),1)
                MID$(obj$,j%*bmap%+i%*bytes%+k%,1)=a$
              NEXT
              pointer%=pointer%+a%+1
            ELSEIF a%>128 THEN
              a$=LEFT$(INPUT$(1,1)+CHR$(0),1)
              FOR k%=pointer% TO pointer%+257-a%
                MID$(obj$,j%*bmap%+i%*bytes%+k%,1)=a$
              NEXT
              pointer%=pointer%+256-a%
            END IF
          WEND
        END IF
      NEXT
    NEXT
    Status%=Status%+4

  ELSE
    PRINT a$;" gefunden."
    a=CVL(INPUT$(4,1))/4
    FOR i%=1 TO a
      a$=INPUT$(4,1)
    NEXT
    GOTO lesechunk

  END IF

checkstatus:
  IF Status%<7 GOTO lesechunk

  CLOSE 1
  PRINT
```

```
PRINT "OK, baue Object auf."
ob$=""
FOR i%=0 TO 10
  ob$=ob$+CHR$(0)
NEXT
ob$=ob$+CHR$(planes%)+CHR$(0)+CHR$(0)
ob$=ob$+MKI$(breite%)+CHR$(0)+CHR$(0)
ob$=ob$+MKI$(hoehe%)+CHR$(0)+CHR$(24)
ob$=ob$+CHR$(0)+CHR$(3)+CHR$(0)+CHR$(0)
ob$=ob$+obj$
PRINT

PRINT "Speichere Object als ";CHR$(34);
PRINT objectfile$;CHR$(34)
PRINT

OPEN objectfile$ FOR OUTPUT AS 2
  PRINT#2,ob$;
CLOSE 2
PRINT "Object gespeichert."

IF colorflag%=1 THEN
  PRINT
  PRINT "Speichere Farbdatei:"
  OPEN colorfile$ FOR OUTPUT AS 3
    PRINT#3,CHR$(planes%);
    PRINT " 1.Byte = Anzahl der Bitplanes"
    FOR i%=0 TO 2^planes%-1
      PRINT i%*3+2;".Byte = rot  (";i%;")*255"
      PRINT#3,CHR$(r(i%)*255);
      PRINT i%*3+3;".Byte = gruen(";i%;")*255"
      PRINT#3,CHR$(g(i%)*255);
      PRINT i%*3+4;".Byte = blau (";i%;")*255"
      PRINT#3,CHR$(b(i%)*255);
    NEXT
  CLOSE 3
END IF

SCREEN 1,320,200,planes%,1
WINDOW 2,,,0,1
FOR i%=0 TO 2^planes%-1
  PALETTE i%,r(i%),g(i%),b(i%)
NEXT

OBJECT.SHAPE 1,ob$
OBJECT.PLANES 1,2^planes%-1,0

FOR i=0 TO 300 step .1
  OBJECT.X 1,i
  OBJECT.Y 1,(i\2)
  OBJECT.ON
NEXT
```

```
WINDOW CLOSE 2
SCREEN CLOSE 1

RUN
```

Variablen-Liste:

Status	gibt Aufschluß über gelesene Chunks
a	Hilfsvariable
b	Feld, Blauanteil einer Farbe
bmap	Größe einer Bitplane des BOB in Byte
breite	Breite des BOB in Pixel
brush	Name des IFF-ILBM-Files
bytes	Breite des BOB in Bytes
colorfile	Name der Farbdatei
colorflag	Flag, Abspeichern der Farbdatei ja/nein
farben	Anzahl im IFF-File gespeicherter Farben
g	Feld, Grünanteil einer Farbe
gepackt	Packstatus: 0=nicht gepackt 1=ByteRun1
hoehe	Höhe des BOB in Pixel
i	Schleifenvariable
j	Schleifenvariable
k	Schleifenvariable
l	Chunklänge
ob	Object-String
obj	Image-String
objectfile	Datei, in die ob$ abgespeichert wird
planes	Anzahl der Bitplanes des BOB (Tiefe)
pointer	Zählvariable für gelesene Byte einer Zeile
r	Feld, Rotanteil einer Farbe

So, soweit das Programm. In der optionalen Farbdatei liegen die Daten wie folgt:

```
1.Byte = Anzahl der Bitplanes des Objects
```

Jetzt folgen 2^(Byte1) Farbwerte, also bei einer Plane:

```
2.Byte = Rot-Anteil der Hintergrundfarbe * 255
3.Byte = Grün-Anteil der Hintergrundfarbe * 255
4.Byte = Blau-Anteil der Hintergrundfarbe * 255
```

```
5.Byte = Rot-Anteil der 1. Farbe * 255
6.Byte = Grün-Anteil der 1. Farbe * 255
7.Byte = Blau-Anteil der 1. Farbe * 255
```

Noch schnell ein paar Worte zum IFF-ILBM-Format:

Ein derartiges File setzt sich aus mehreren hintereinander abgespeicherten Dateien zusammen, die man Chunks nennt. Jeder Chunk ist dabei gleich aufgebaut:

```
1. Chunk-Name  = 4 Byte langer String, z.B.: "BODY"
2. Chunk-Länge = 4-Byte-Integer, also LONG-Format
3. Chunk-Daten = #Chunk-Länge Bytes
```

Der Aufbau des Headerchunk, mit dem jedes IFF-File beginnt, ist zwar ähnlich, nicht aber genauso aufgebaut:

```
1. File-Art   = "FORM", der Header von IFF-Files
2. File-Länge = Long-Wert
3. Daten-Typ  = in diesem Falle "ILBM" (interleaved bitmaps)
```

Die wichtigsten Chunks in Kurzform:

Der BMHD-Chunk:

```
 1. Long = "BMHD" (BitmapHeader-Chunk)
 2. Long = Chunklänge
 3. Word = Breite der Grafik in Pixel
 4. Word = Höhe der Grafik in Pixel
 5. Word = X-Position der Grafik
 6. Word = Y-Position der Grafik
 7. Byte = Anzahl der BitPlanes des Screens
 8. Byte = Maskierung
 9. Byte = Crunch-Art
10. Byte = ??
11. Word = transparente Farbe
12. Byte = X-Aspect
13. Byte = Y-Aspect
14. Word = Breite des Screens in Pixel
15. Word = Höhe des Screens in Pixel
```

Der CMAP-Chunk:

```
1. Long = "CMAP" (ColorMap)
2. Long = Chunklänge
3. Byte = Farbe 0 Rotwert  *255
4. Byte = Farbe 0 Grünwert *255
```

```
5. Byte = Farbe 0 Blauwert *255
6. Byte = Farbe 1 Rotwert  *255
7. ...
```

Der CRNG-Chunk von DeLuxe Paint:

```
1. Long = "CRNG" (ColorCycle-Chunk von DPaint - 4 mal)
2. Long = Chunklänge
3. Word = bisher immer 0
4. Word = Geschwindigkeit
5. Word = Aktiviert oder nicht
6. Byte = untere Farbe
7. Byte = obere Farbe
```

Der CCRT-Chunk:

```
1. Long = "CCRT" (ColorCycle-Chunk von Graphicraft)
2. Long = Chunklänge
3. Word = Richtung
4. Byte = Startfarbe
5. Byte = Endfarbe
6. Long = Sekunden
7. Long = micro-Sekunden
```

Der wichtigste, der BODY-Chunk:

```
1. Long = "BODY" (BitMaps)
2. Long = Chunklänge
3. = 1. Zeile der 1. Bitplane (eventuell gepackt - s. BMHD)
   = 1. Zeile der 2. Bitplane (sofern vorhanden)
   ...
   = 1. Zeile der letzten Bitplane
4. = 2. Zeile ...
```

Noch ein paar Zeilen zum ByteRun1-Crunch-Algorythmus:

Es wird niemals mehr als eine Zeile einer Bitplane auf einmal gepackt. Daher kann das Entpacken auch zeilenweise geschehen. Die Kodierung besteht zunächst aus einem CodeByte. Ist dieses Byte größer als der Wert 128, wird das nächste Byte 258-Code-Byte Mal wiederholt, also mindestens dreimal. Da in FOR-NEXT-Schleifen jedoch auch der Startwert der Schleifenvariable mitdurchlaufen wird, müssen wir bei solchen Konstruktionen noch den Wert 1 abziehenm, also eigentlich folgendes erhalten: FOR i=startwert TO startwert+258-Codebyte-1. Daher schrieb ich oben gleich 257-Codebyte. Die zweite Codierung gilt für

Code-Bytes, die kleiner als 128 sind. Hier werden die nächsten CodeByte+1 Bytes nicht codiert übernommen. Der Code-Wert 128 selbst wird hierbei überlesen. Zusammenfassend können wir sagen, daß sowohl bei der ersten, wie auch bei der zweiten Codierungsart maximal 128 Byte auf einmal codiert sein können. Da die Breite eines 640*x - Screens jedoch nur 80 Byte beträgt, kann so durchaus eine Zeile einer Bitplane auf einen Schlag codiert werden.

3.2.6 Floodfill einmal anders

Wie Sie sicherlich wissen, hat der Amiga die Möglichkeit, mit einer Geschwindigkeit von einer Million Pixel pro Sekunde so komplizierte Vorgänge wie das Ausfüllen von Flächen in irgendeiner Farbe durchzuführen. Das geschieht von AmigaBASIC aus zumeist mit dem Befehl PAINT. Dieser Befehl hat nun aber einen entscheidenden Nachteil. Er füllt ab den angegebenen Koordinaten den Bildschirm nur bis zu einer bestimmten Farbe. Dadurch wird seine Anwendung zum Beispiel in einem selbstgeschriebenen Malprogramm stark eingeschränkt. Besser wäre es gewesen, die AmigaBASIC-Entwickler hätten die Möglichkeit eingeräumt, durch einen bestimmten Parameter den PAINT-Befehl mit dem Füllen bis zu irgendeiner anderen Farbe zu beauftragen. Besonders verwunderlich ist, daß es eine Betriebssystem-Routine gibt, die dieses leistet. Zudem handelt es sich um eine Routine der graphics-library, deren Programme bekanntlich immer im Speicher sind und daher nicht einmal von der Workbench nachgebootet werden müßte.

Die Routine heißt Flood und kann von AmigaBASIC aus folgendermaßen aufgerufen werden:

```
CALL Flood&(Rastport,Modus,x,y)
```

Hier also eine S U B-Routine, die Flood nutzt:

```
REM ############################
REM #  F L O O D F I L L  Amiga  #
REM #-------------------------#
REM #  PAINT bis zu irgendeiner  #
REM #  anderen Farbe...          #
REM #-------------------------#
REM # (W) 1987 by Stefan Maelger #
REM ############################

LIBRARY "graphics.library"

SCREEN 1,640,255,2,2
WINDOW 2,"FLOODFILL",,0,1

LOCATE 2,2
PRINT "Floodfill-Demo"

CIRCLE (200,80),150,2
CIRCLE (400,80),150,3

FLOODFILL 200,80,1
FLOODFILL 300,80,1
FLODDFILL 400,80,1

LIBRARY CLOSE

LOCATE 4,2
PRINT "PRESS ANY KEY"

WHILE INKEY$=""
WEND

STOP

SUB FLOODFILL(x%,y%,farbe%) STATIC
  PSET (0,0),0
  PAINT (0,0),0
  COLOR farbe%
  rastport&=WINDOW(8)
  ToAnyColorMode%=1
  CALL Flood&(rastport&,ToAnyColorMode%,x%,y%)
END SUB
```

Der Einfachheit halber initialisieren wir die Routine mit einem einfachen PAINT, das nichts weiter bewirkt - außer daß der AmigaBASIC-Interpreter vorher schon einmal eine Bitplane einrichten muß, die der Flood-Befehl für seine Arbeit benötigt.

3.2.7 Windows manipulieren

Daß sich mit Windows eine ganze Menge anstellen läßt, sollte Ihnen spätestens seit dem DB-Buch Supergrafik bekannt sein. Daß man auch ausgefallenere, nichtsdestoweniger aber eminent effiziente Dinge mit BASIC-Windows tun kann, darüber soll Ihnen dieses Kapitel einen kleinen Einblick vermitteln, der sicherlich nicht der Weisheit letzter Schluß ist, Ihnen aber Anlaß genug sein sollte, kräftig drauflos zu experimentieren.

3.2.7.1 Borderless: BASIC-Windows, die aus dem Rahmen fallen

Nachdem ich in einer bekannten Zeitschrift das Superlisting eines "AmigaBASIC-Profis" sah, in dem er mühsam minutenlang die Adressen in den Bitmaps ausfindig machte, um schließlich den Border (=den Rahmen) des Windows Byte für Byte wegzupoken, entschloß ich mich, Ihnen diese Routine vorzustellen:

```
' ####################################
' # BORDERLESS fuer AmigaBASIC-Windows #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
  LIBRARY "intuition.library"
  CLS
  PRINT "Wie gefällt Ihnen denn dieses Fenster?"
  PRINT
  warte 2
  PRINT "Und ohne Border (Rahmen) ?"
  PRINT

  killborder

  wartetaste
  remake
  LIBRARY CLOSE
  end


SUB remake STATIC
  WINDOW CLOSE 1
  WINDOW 1
```

```
END SUB

SUB warte(sekunden%) STATIC
  t=TIMER+sekunden%
  WHILE t>TIMER
  WEND
END SUB

SUB wartetaste STATIC
  WHILE INKEY$=""
  WEND
END SUB

SUB killborder STATIC
  borderless&   =2^11
  gimmezerozero&=2^10
  window.base&=WINDOW(7)
  window.modi&=window.base&+24
  Modus&=PEEKL(window.modi&)
  Modus&=Modus& AND(2^26-1-gimmezerozero&)
  Modus&=Modus& OR borderless&
  POKEL window.modi&,Modus&
  CALL RefreshWindowFrame(window.base&)
END SUB
```

3.2.7.2 Gadgets an und ausschalten

Hat es Ihnen gefallen? Nein? Wieso, haben Sie etwa die Gadgets gestört? Dann nichts wie weg mit ihnen! Wie? Tja...

```
' ####################################
' # GADGETon/off in AmigaBASIC-Windows #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
LIBRARY "intuition.library"

  PRINT "Lassen wir die Gadgets verschwinden!"
  SaveGadgetPointer Gaddy&
  warte 5
  UnlinkGadgets
  warte 10
  PRINT "Und nun holen wir sie uns wieder."
  warte 5
  SetGadgets Gaddy&
  LIBRARY CLOSE
  WINDOW CLOSE 1
```

```
  WINDOW 1
  END
SUB warte(sekunden%) STATIC
  t=TIMER+sekunden%
  WHILE t>TIMER
  WEND
END SUB
SUB SaveGadgetPointer(Pointer&) STATIC
  window.base&   =WINDOW(7)
  gadget.pointer&=window.base&+62
  Pointer&=PEEKL(gadget.pointer&)
END SUB
SUB UnlinkGadgets STATIC
  window.base&   =WINDOW(7)
  gadget.pointer&=window.base&+62
  POKEL gadget.pointer&,0
  CALL RefreshWindowFrame(window.base&)
END SUB
SUB SetGadgets(Pointer&) STATIC
  window.base&   =WINDOW(7)
  gadget.pointer&=window.base&+62
  POKEL gadget.pointer&,Pointer&
  CALL RefreshWindowFrame(window.base&)
END SUB
```

3.2.7.3 BorderDraw, der Formen-Zauberer

Jetzt wollen wir einmal von Intuition einen lustig bunten Border zeichnen lassen. Dazu müssen wir jedoch wissen, wie die Border-Struktur aussieht, deren Adresse wir mit der DrawBorder-Routine übergeben müssen. Eigentlich recht simpel:

1. Word	horizontaler Abstand von der im Aufruf der Routine angegebenen X-Koordinate (so eine Art erweitertes STEP - man braucht nur einmal eine Form zu definieren und kann sie dann in einem beliebigen Abstand noch einmal zeichnen)
2. Word	vertikaler Abstand der Y-Koordinaten
3. Byte	Zeichenfarbe (die von BASIC bekannten Nummern)
4. Byte	Hintergrundfarbe
5. Byte	Zeichenmodus (JAM1 = 0)
6. Byte	Anzahl der X-Y-Koordinatenpaare
7. Long	Adresse der Koordinaten-Tabelle
8. Long	Adresse der nächsten Struktur oder der Wert 0

Spätestens bei Punkt 7 werden Sie bemerkt haben, daß wir noch eine Koordinatentabelle benötigen. Diese ist ganz einfach aufgebaut:

Die Tabelle besteht nur aus Words. Dabei kommt immer zuerst die X-Koordinate, dann die Y-Koordinate eines Punktes. Dadurch benötigt also ein Punkt vier Bytes oder zwei Words. Das wars auch schon, sie sehen, ich hatte Recht mit meiner Behauptung, alles wäre ganz einfach.

Eines noch: Wenn Sie beim Aufruf der Routine anstelle des Rahmen-Rastports den Window-Rastport einsetzen (WINDOW(8)), können Sie auch beliebig komplizierte Strukturen im BASIC-Window zeichnen lassen. Dabei gibt es nur eine Schwierigkeit: Der Zeichen-Cursor des Windows steht hinterher auf dem letzten Punkt der letzten Struktur. Ein nachfolgender PRINT-Befehl würde an dieser Position mit der Ausgabe beginnen! Erst NACH einem PRINT aktualisiert AmigaBASIC die Cursor-Position! Schieben Sie bei einem solchen Akt also ein PRINT hinter den Aufruf von DrawBorder. Dann funktioniert's nämlich wieder.

```
' ####################################
' #  DRAWBORDER  - der Formenzeichner  #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
LIBRARY "intuition.library"

PRINT "Ich setze den Koordinaten-String zusammen"

breite%=PEEKW(WINDOW(7)+8)-1
hoehe%=PEEKW(WINDOW(7)+10)-1
xlinks%=0
yoben%=0
xy$=MKI$(xlinks%)+MKI$(yoben%)
xy$=xy$+MKI$(xlinks%)+MKI$(hoehe%)
xy$=xy$+MKI$(breite%)+MKI$(hoehe%)
xy$=xy$+MKI$(breite%)+MKI$(yoben%)
Paare%=4
xOffset%=0
yOffset%=0
farbe%=0
```

```
PRINT "Ich zeichne den Border"

Setborder xy$,Paare%,farbe%,xOffset%,yOffset%

FOR i%=3 to 1 STEP -1
  PRINT "Ich warte ein wenig"
  t=TIMER+10:WHILE t>TIMER:WEND
  PRINT "Ich zeichne in Farbe";i%
  Setborder xy$,Paare%,i%,xOffset%,yOffset%
NEXT

LIBRARY CLOSE
END

SUB Setborder(xy$,anzahl%,farbe%,x%,y%) STATIC
  window.base&=WINDOW(7)
  borderrastport&=PEEKL(window.base&+58)
  IF borderrastport&=0 THEN EXIT SUB
  a$=MKI$(0)    'horizontaler Abstand
  a$=a$+MKI$(0) 'vertikaler   Abstand
  a$=a$+CHR$(farbe%) 'Zeichenfarbe
  a$=a$+CHR$(0)      'Hintergrund (unbenutzt)
  a$=a$+CHR$(0)      'Modus: JAM1
  a$=a$+CHR$(anzahl%)   'Anzahl x-y-Paare
  a$=a$+MKL$(SADD(xy$)) 'Zeiger auf Koordinaten
  a$=a$+MKL$(0)         'Zeiger auf nächste Struktur
  CALL DrawBorder(borderrastport&,SADD(a$),x%,y%)
  ' -----letzte Parameter: relative X- und Y-Koordinate
END SUB
```

Sie werden sicherlich die tollsten Anwendungen für diese Routine finden, auch wenn das Programm sehr einfach aussieht.

3.2.7.4 ChangeBorderColor -jetzt wird's bunt

Kommen wir zu einem recht wertvollen optischen Schmankerl. Sie werden mit der nächsten Routine die Farben des Rahmens (inklusive der Titelleiste) verändern können. Ach, das können Sie auch? Sie meinen, man bräuchte ja lediglich mit dem PALETTE-Befehl die Farben 0 und 1 zu ändern? Das stimmt, zugegeben, aber ich werde Ihnen jetzt zeigen, was Sie tun müs-

sen, um dabei die Hintergrund- und Zeichenfarbe des Windows nicht mit zu verändern! Natürlich das Ganze wie gewohnt in Form eines SUB-Befehls:

```
' ####################################
' # C H A N G E  B O R D E R C O L O R #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
LIBRARY "intuition.library"

PRINT "Hat es Sie schon einmal gestört, daß"
PRINT "die Zeichenfarbe des Borders immer aus"
PRINT "Farbregister 0 und der Hintergrund immer"
PRINT "aus Register 1 gezeichnet wird ?"
PRINT

PRINT "Das können wir nämlich im Gegensatz zu dem"
PRINT "Window-Befehl selbst bestimmen!"

LOCATE 10,1:PRINT "Vordergrund:"
LOCATE 13,1:PRINT "Hintergrund:"
t=TIMER+15:WHILE t>TIMER:WEND
FOR i=0 TO 3
  LINE (i*30,136)-STEP(30,20),i,bf
  LINE (i*30,136)-STEP(30,20),1,b
NEXT

FOR h%=0 TO 3
  FOR v%=0 TO 3
    ChangeBorderColor v%,h%
    LOCATE 10,14:PRINT v%
    LOCATE 13,14:PRINT h%
    t=TIMER+5
    WHILE t>TIMER
    WEND
NEXT v%,h%

ChangeBorderColor 1,0

LIBRARY CLOSE
END

SUB ChangeBorderColor(DetailPen%,BlockPen%) STATIC
  window.base&=WINDOW(7)
  Detail.Pen& =window.base&+98
  Block.Pen&  =window.base&+99
  POKE Detail.Pen&,DetailPen%
```

```
  POKE  Block.Pen&,BlockPen%
  CALL RefreshWindowFrame(window.base&)
END SUB
```

3.2.7.5 Monocolor-Workbench

Nun kommen wir zu einem wirklich nützlichen Programm. Sie hätten gern 16 KByte Speicher mehr (ohne etwas dafür bezahlen zu müssen)? Sie möchten beim editieren von BASIC-Programmen, daß endlich einmal das Listing in angemessener Geschwindigkeit über den Bildschirm jagt, anstatt endlos langsam dahinzukriechen? Sie möchten, daß überhaupt fast alles doppelt so schnell geht wie bisher? Okay, nehmen wir der Workbench eine Bitplane und Sie haben erreicht, was Sie wollen:

```
' ####################################
' #        MONOCOLOR WORKBENCH       #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger   #
' ####################################
'
LIBRARY "intuition.library"
LIBRARY "graphics.library"

Setplanes 1

LIBRARY CLOSE
SYSTEM

SUB Setplanes(planes%) STATIC
  IF planes%<1 OR planes%>6 THEN EXIT SUB
  rastport&      =WINDOW(8)
  bitmaps&       =PEEKL(rastport&+4)
  current.planes%=PEEK(bitmaps&+5)
  window.base&   =WINDOW(7)
  screen.base&   =PEEKL(window.base&+46)
  screen.breite% =PEEKW(screen.base&+12)
  screen.hoehe%  =PEEKW(screen.base&+14)
  IF current.planes%>planes% THEN
   POKE bitmaps&+5,planes%
   FOR kill.plane%=current.planes% TO planes%+1 STEP -1
    plane.ad&=PEEKL(bitmaps&+4+4*kill.plane%)
    CALL FreeRaster(plane.ad&,screen.breite%,screen.hoehe%)
    CALL RemakeDisplay
    CALL RefreshWindowFrame(WINDOW(7))
```

```
    CLS
   NEXT
  END IF
END SUB
```

3.2.7.6 Neuer SCREEN-Befehl für alle Grafikmodi

Wie Sie gesehen haben, gibt FreeRaster den Speicherbereich frei, den die zweite Plane belegte. Da wir jetzt auch wissen, wo die Adressen und wo die Anzahl der Bitplanes liegen, sollte es uns auch nicht schwerfallen, weitere Planes hinzuzufügen.

Wie Sie wissen, haben die Programmierer von AmigaBASIC die Modi Hold-And-Modify (4096 Farben gleichzeitig) und Halfbright (Slang: Halfbrite = 64 Farben gleichzeitig) völlig links liegenlassen. Diese benötigen sechs Bitplanes, und die können bekanntlich mit AmigaBASIC-Befehlen nicht so einfach erzeugt werden - nimmt man einmal den LIBRARY-Befehl heraus. Eine Einbindung des Bildschirmrandes in die Zeichenfläche, somit die Erzeugung von Screens höherer Auflösung (OVERSCAN-Modus z.B. 704 mal 576 Pixel) ist ebenso nicht vorgesehen. Wer seinem Amiga tolle Grafikeffekte entlocken will braucht dringend Abhilfe. Deshalb schrieb ich eine SUB-Routine, die den SCREEN-Befehl etwas erweitert und Ihnen HAM, Halfbright und Over-Scan bietet:

```
'########################################
'# Neuer SCREEN-Befehl (w)'88 by SMmagic #
'#--------------------------------------#
'# OVERSCAN  HOLD-AND-MODIFY  HALFBRIGHT #
'########################################


Demo:
  DEFLNG a-z                    'alle Variablen sind Longs
  DECLARE FUNCTION AllocMem LIBRARY
  LIBRARY"exec.library"
  DECLARE FUNCTION ViewAddress LIBRARY
  LIBRARY"intuition.library"

  newSCREEN 2,320,256,6,5       'Screen im HAM-Modus
  WINDOW 2,"HoldAndModify",,,2
  FOR g%=0 TO 15                'Alle 4096 Farben zeigen
   r%=0
```

```
 b%=0
 LINE(0,g%*10)-STEP(0,9),0
 LINE(1,g%*10)-STEP(0,9),g%+48
 FOR x%=0 TO 7
  FOR r%=1 TO 15
   LINE(x%*32+r%+1,g%*10)-STEP(0,9),r%+32
  NEXT
  b%=b%+1
  LINE(x%*32+17,g%*10)-STEP(0,9),b%+16
  FOR r%=14 TO 0 STEP -1
   LINE(x%*32+17+15-r%,g%*10)-STEP(0,9),r%+32
  NEXT
  b%=b%+1
  IF b%<16 THEN LINE(x%*32+33,g%*10)-STEP(0,9),b%+16
 NEXT
NEXT

t=TIMER+10
WHILE t>TIMER
WEND

WINDOW CLOSE 2
newSCREEN.CLOSE 2

newSCREEN 3,320,256,6,7   'Screen im Halfbright-Modus
WINDOW 3,"Halfbrite",,,3
FOR i%=0 TO 63            'alle 64 Farben zeigen
 LINE(i%*4+1,0)-(i%*4+3,240),i%,bf
NEXT

t=TIMER+10
WHILE t>TIMER
WEND

WINDOW CLOSE 3
newSCREEN.CLOSE 3

newSCREEN 4,352,288,2,9   'Screen im OverScan-Modus
WINDOW 4,"OverScan",,,4

FOR i%=0 TO 350 STEP 16   'Bis in den Rand malen
 FOR j%=0 TO 3
  LINE(i%+j%*4,0)-(i%+j%*4,270),j%
 NEXT
 IF i%<280 THEN
  FOR j%=0 TO 3
   LINE(0,i%+j%*4)-(350,i%+j%*4),j%
  NEXT
 END IF
NEXT

t=TIMER+10
WHILE t>TIMER
```

```
  WEND

  WINDOW CLOSE 4
  newSCREEN.CLOSE 4

  LIBRARY CLOSE
  STOP


SUB newSCREEN(n%,b%,h%,t%,m%)STATIC
 SHARED overscan.x%,overscan.y%
  IF m%>16 OR m%<1 THEN ERROR 5
  IF t%>6 OR t%<1 THEN ERROR 5
  IF h%>1024 OR h%<1 THEN ERROR 5
  IF b%>1024 OR b%<1 THEN ERROR 5
  t2%=t%+(t%=6)
  mm%=m%+8*(m%>8)
  mx%=1-(mm%=2)-(mm%=4)
  m2%=mx%-2*((mm%=3)+(mm%=4)+(mm%=6)+(mm%=8))
  IF m%>8 AND overscan.x%=0 THEN
    overscan.x%=mx%*16
    overscan.y%=(1-(m2%>2))*16
    prefs=AllocMem(120,2)
    IF prefs=0 THEN ERROR 7
    GetPrefs prefs,120
    POKE prefs+118,(PEEK(prefs+118)-overscan.x%)AND 255
    POKE prefs+119,(PEEK(prefs+119)-overscan.y%)AND 255
    SetPrefs prefs,120,-1
    FreeMem prefs,120
  END IF
  SCREEN n%,b%,h%,t2%,m2%
  IF t%>5 THEN
    view=ViewAddress
    vp=PEEKL(view)
    bitmap=PEEKL(vp+44)
    plsz=PEEKW(bitmap)*PEEKW(bitmap+2)
    plane=AllocMem(plsz,65538&)
    IF plane=0 THEN ERROR 7
    POKEL bitmap+28,plane
    POKE bitmap+5,6
    IF mm%=5 OR mm%=6 THEN POKEW vp+32,2^11
    IF mm%=7 OR mm%=8 THEN POKEW vp+32,2^7
    RemakeDisplay
  END IF
END SUB

SUB newSCREEN.CLOSE(n%)STATIC      'NUR BEI OVERSCAN
 SHARED overscan.x%,overscan.y%    'ERFORDERLICH!!!
  IF overscan.x%<>0 THEN
    prefs=AllocMem(120,2)
    IF prefs=0 THEN ERROR 7
    GetPrefs prefs,120
    POKE prefs+118,(PEEK(prefs+118)+overscan.x%)AND 255
```

```
    POKE prefs+119,(PEEK(prefs+119)+overscan.y%)AND 255
    SetPrefs prefs,120,-1
    FreeMem prefs,120
  END IF
  SCREEN CLOSE n%
END SUB
```

Na, ist Ihnen etwas aufgefallen? Wir malen mit den Farben 0 bis 63! Von wegen AmigaBASIC unterstützt diese Modi nicht! AmigaBASIC unterstützt lediglich das Errichten der Screens nicht. Sollten Sie einmal in diesen Modi arbeiten wollen, so müssen Sie noch einiges wissen. Zuvor aber zu den Parametern des neuen SCREEN-Befehls. Diese habe ich weitgehend den Original-Parametern angepaßt, so daß Ihnen die Umstellung nicht allzu schwer werden wird:

Neuer SCREEN-Befehl, Bedeutung der Parameter:

```
SCREEN n,Breite,Höhe,Tiefe,Modus
Neu: Tiefe - jetzt auch 6 Planes möglich
Neu: Modus - 1 = 320 mal 256 Normale Modi
             2 = 640 mal 256 High Resolution
             3 = 320 mal 512 Interlace
             4 = 640 mal 512 Interlace + HiRes
             5 = 320 mal 256 HoldAndModify
             6 = 320 mal 512 HoldAndModify + Interlace
             7 = 320 mal 256 Halfbrite
             8 = 320 mal 512 Halfbrite + Interlace
             9 = 352 mal 288 OverScan
            10 = 704 mal 288 OverScan + HiRes
            11 = 352 mal 576 OverScan + Interlace
            12 = 704 mal 576 OverScan + HiRes + Interlace
            13 = 352 mal 288 HoldAndModify + OverScan
            14 = 352 mal 576 HoldAndModify+Interlace+OverScan
            15 = 352 mal 288 Halfbrite + OverScan
            16 = 352 mal 576 Halfbrite + OverScan + Interlace
```

Alle anderen Parameter entsprechen denen des normalen SCREEN-Befehls. Sollten Sie nicht mit OverScan arbeiten, genügt der Befehl SCREEN CLOSE. Bei OverScan muß new-SCREEN.CLOSE aufgerufen werden, damit das Display wieder an die alte Stelle gerückt wird. Der Parameter ist die Screen-Nummer, ebenso wie bei SCREEN CLOSE.

Der OverScan-Modus

Dieser Modus unterscheidet sich von den bekannten Grafik-Modi nicht sonderlich. Die einzige Änderung besteht darin, daß alle Koordinaten nach links oben verschoben sind und der sichtbare Bereich um etliche Pixel größer geworden ist.

Der Halfbright-Modus

'Half' heißt auf deutsch 'halb', 'bright' soviel wie 'hell'. Eine sehr treffende Bezeichnung, wie ich meine, denn hier sind die Farben 0 bis 31 genau die 32 Farben, die es auch bei 5 Bitplanes sind, während die Farben 32 bis 63 die halben Helligkeitswerte der unteren Farben haben. Die neu hinzugekommenen Farben sind direkt abhängig von den 'normalen' Farben. Ist beispielsweise die Farbe 1 ein helles Rot, ist die Farbe 33 ein dunkles Rot (halb so hell wie Farbe 1). Um die zu einer normalen Farbe gehörige Halfbright-Farbe zu ermitteln, muß lediglich der Wert 32 zur Registernummer addiert werden:

```
zugehörige Halfbright-Farb-Nummer = Farb-Nummer + 32
```

Man beachte, daß es nur möglich ist, 16 verschiedene Farbabstufungen pro Farbanteil (Rot/Grün/Blau) einzustellen. Da auch die Halfbright-Farben nur diese Farbstufen annehmen können, kann es für mehrere verschiedene 'Normal'-Farben die gleiche Halfbright-Farbe geben:

Normalfarbe	Halfbright-Farbe
PALETTE x, 15/15, 13/15, 9/15	x+32 = 7/15, 6/15, 4/15
14/15, 13/15, 9/15	7/15, 6/15, 4/15
15/15, 12/15, 9/15	7/15, 6/15, 4/15
14/15, 12/15, 9/15	7/15, 6/15, 4/15
15/15, 13/15, 8/15	7/15, 6/15, 4/15
14/15, 13/15, 8/15	7/15, 6/15, 4/15
15/15, 12/15, 8/15	7/15, 6/15, 4/15
14/15, 12/15, 8/15	7/15, 6/15, 4/15

```
FarbanteilHB = INT ( Farbanteil * 15 / 2 ) / 15
```

Hier kann man gut sehen, daß jeweils acht verschiedene Farbwerte die gleiche Halfbright-Farbe ergeben. Um es noch einmal zu betonen: Der PALETTE-Befehl kann nur die unteren 32 Farben einstellen. Die einzustellenden Farben sollten Sie immer so wählen, daß verschiedene Halfbright-Farben dabei herauskommen.

Der Hold'n'Modify-Modus

Bei HAM wird es schwierig. Hier sind nur die Farben 0-15 fest einstellbar. Malt man einen Punkt in einer dieser Farben, so erscheint er auch immer in der mit PALETTE eingestellten Farbe:

```
Farben 0 - 15 sind normal mit PALETTE bestimmbar
```

Bei den Farben 16-31 ist es schon etwas anders. Hier werden zunächst die RGB-Werte des Pixels links neben dem zu malenden Pixel übernommen (Hold), und dann der Blauanteil geändert (Modify). Der neue Blauanteil ergibt sich dann so:

```
Blauanteil = (Farbnummer - 16) / 15
```

Die Farben 32-47 verändern nur den Rotanteil:

```
Rotanteil = (Farbnummer - 32) / 15
```

Mit den Farben 48-63 läßt sich schließlich der Grünanteil modifizieren:

```
Grünanteil = (Farbnummer - 48) / 15
```

Auf diese Weise kommt man nach spätestens 3 Pixeln auf die gewünschte Farbe.

Um den Grünanteil eines Pixel auf den Wert 13/15 zu bringen, muß man mit der Farbe 13+48=61 malen, bei einer Änderung des Rotanteils auf 7/15 mit der Farbe 7+32=39.

3.2.7.7 Das Koordinatenproblem

Wie Sie ja wissen, ist der Punkt mit den Koordinaten (0,0) nicht etwa in der linken oberen Ecke des Screens zu finden, sondern liegt unterhalb des Titel-Borders und rechts neben dem linken Teil des Borders. Etwas ganz merkwürdiges geschieht nun, wollen Sie ein Window ohne Titel direkt über die Titelleiste eines Windows mit Titel, zum Beispiel das BASIC-Standard-Ausgabefenster, plazieren.

Nehmen wir einmal an, wir wünschen uns ein Window, das genau acht Pixel hoch sein soll. Nun müssen wir den Befehl eingeben:

```
WINDOW 2,,(0,0)-(311,-2),16,-1
```

Ist Ihnen etwas aufgefallen? Die Y-Koordinaten gehen von 0 bis -2. Hier liegt nämlich wieder einmal ein Fehler im System vor. Das erste Koordinatenpaar (0,0) wird tatsächlich richtig interpretiert, das zweite Koordinatenpaar jedoch zumindest beim Y-Wert falsch, da sich nur in diesem Falle der Interpreter auf die relativen Koordinaten des darunterliegenden BASIC-Standard-Fensters bezieht. Fordern Sie also ein Window mit dem Befehl:

```
WINDOW 2,,(0,0)-(311,8),16,-1
```

so erhalten Sie ein Fenster, das 18 Pixel hoch ist! Wir müssen in einem solchen Fall also zunächst die Höhe des Titel-Borders (10 Pixel) abziehen, um an das Screen-Koordinatensystem anzugleichen (8 - 10 = -2).

Soll ein Window nur die Titelleiste des Standard-Fensters bedecken, ergeben sich die Koordinaten wie folgt:

y2=10	(Höhe unseres neuen Windows)
y2=y2-10	(Höhe der Titelleiste abziehen zum Angleichen der Koordinaten)
y2=y2-4	(Höhe des oberen und unteren Borders des neuen Windows abziehen)

also:

```
WINDOW 2,,(0,0)-(311,-4),16,-1
```

3.2.8 Intuition macht das Leben leicht - mit HoldAndModify, Halfbrite und OverScan

Wer nur mit Betriebssystem-Routinen auf sein Display zugreifen möchte, was schließlich etwa zehnmal so schnell geht wie mit BASIC-Befehlen, beim PRINT-Befehl sogar 1000mal so schnell, der kann sich auch von Intuition einen Screen oder ein Window öffnen lassen. Dazu legen wir im folgenden Programm eine sogenannte NewScreen-Struktur an, in der Intuition die nötigen Daten übergeben werden. Ein besonderer Punkt ist noch anzumerken: Der letzte Screen wird im OVERSCAN-Modus betrieben, d.h. daß der Bildschirm-Rand mit zur Zeichenfläche gehört.

Ermöglicht wird dies durch eine Manipulation der sogenannten View-Struktur, die angibt, wo die Zeichenfläche beginnt. Zu diesem Zweck laden wir einen Teil der Preferences-Struktur, der gerade noch die in Preferences einstellbaren ViewOffsets (+118 und +119) enthält. Diese Offsets lesen wir aus, um später das Original-Display wiederherzustellen und ziehen von x- und y-Koordinate einen Wert ab, der den Zeichenflächen-Anfang garantiert außerhalb Ihres Monitors positioniert (linke obere Ecke). Die neuen Werte werden dann in der Preferences-Struktur abgelegt und mit SetPrefs gesetzt. Die Screengröße muß natürlich entsprechend vergrößert werden. Zu Demonstrationszwecken zeichnen wir nun etliche Linien, damit Sie sehen, daß wirklich jeder Millimeter des Bildschirmrandes zur Verfügung steht.

Den Effekt des OVERSCAN haben Sie mit Sicherheit bereits in vielen Demos oder Intros zu sehen bekommen, meist in Verbindung mit scrollenden Schriften, die Sie natürlich auch einbauen können. OverScan funktioniert auch mit jedem anderen Screen oder Window - Voraussetzung ist jedoch immer entweder die

GetPrefs/SetPrefs-Anwendung oder die direkte Manipulation der Copper-Liste, deren Anfangsadresse in dem Offset +56 von der Basisadresse der Graphics.Library zu finden ist. Hier das Programm, das auch unter V1.3 seinen Dienst tut:

```
  DEFLNG a-z
DECLARE FUNCTION AllocMem LIBRARY
  LIBRARY":bmaps/exec.library"
DECLARE FUNCTION OpenScreen LIBRARY
  LIBRARY":bmaps/intuition.library"
  LIBRARY":bmaps/graphics.library"
GrafikDemo:
  WINDOW CLOSE 1
  NewScreenSize%=32
  PreferencesSize%=120
  MemorySize%=NewScreenSize%+PreferencesSize%
  MEMF.CHIP=2
  MEMF.CLEAR=65536&
  MEMF=MEMF.CHIP+MEMF.CLEAR
  NewScreen=AllocMem(MemorySize%,MEMF)
  IF NewScreen=0 GOTO failed
    xSize%=320
    ySize%=256
    zSize%=5
    ScreenQuiet%=256
    CustomScreen%=15
    ViewMode%=0
    ScreenType%=ScreenQuiet%+CustomScreen%
    POKEW NewScreen+4,xSize%
    POKEW NewScreen+6,ySize%
    POKEW NewScreen+8,zSize%
    POKEW NewScreen+14,ScreenType%
  Preferences=NewScreen+NewScreenSize%
    GetPrefs Preferences,PreferencesSize%
    ViewX%=118
    ViewY%=119
    OldX%=PEEK(Preferences+ViewX%)
    OldY%=PEEK(Preferences+ViewY%)
    OverScan%=0
Demo:
  ScreenBase=OpenScreen(NewScreen)
  IF ScreenBase=0 GOTO failed
  RastPort=ScreenBase+84
  ViewPort=ScreenBase+44
  IF OverScan%>0 GOTO NewPrefs
  SetRGB4 ViewPort,0,0,0,0
  SetRGB4 ViewPort,1,15,15,15
  IF ViewMode%=0 THEN
    yStep%=8
    GOSUB ShowColors
    GOSUB ShowSData
```

```
  ELSEIF ViewMode%=128 THEN
    yStep%=4
    GOSUB ShowColors
    GOSUB ShowSData
  ELSEIF ViewMode%=2048 THEN
    GOSUB ShowHAM
  END IF
warte:
  zeit!=TIMER+7
  WHILE zeit!>TIMER
  WEND
  READ ViewMode%
  IF ViewMode%>-1 THEN
    READ OverScan%
    xSize%=320+OverScan%
    ySize%=256+OverScan%
    POKEW NewScreen+4,xSize%
    POKEW NewScreen+6,ySize%
    READ zSize%
    POKEW NewScreen+8,zSize%
    POKEW NewScreen+12,ViewMode%
    CloseScreen ScreenBase
    GOTO Demo
  END IF
  POKE Preferences+ViewX%,OldX%
  POKE Preferences+ViewY%,OldY%
  SetPrefs Preferences,120,-1
  CloseScreen ScreenBase
  FreeMem NewScreen,MemorySize%
  LIBRARY CLOSE
  WINDOW 1,,,,-1
  END
NewPrefs:
  POKE Preferences+ViewX%,OldX%-16
  POKE Preferences+ViewY%,OldY%-16
  SetPrefs Preferences,120,-1
  SetAPen RastPort,1
  FOR i%=1 TO 11
   Move RastPort,0,0
   Draw RastPort,i%*32,287
   Move RastPort,0,287
   Draw RastPort,i%*32,0
   Move RastPort,351,0
   Draw RastPort,i%*32,287
   Move RastPort,351,287
   Draw RastPort,i%*32,0
  NEXT
  SetDrMd RastPort,0
  SetAPen RastPort,2
  READ a$
  Move RastPort,144,146
  Text RastPort,SADD(a$),LEN(a$)
  GOTO warte
```

```
ShowColors:
  FOR i%=0 TO 2^zSize%-1
    SetAPen RastPort,i%
    RectFill RastPort,0,i%*yStep%,xSize%-1,(i%+1)*yStep%-1
  NEXT
  RETURN
ShowSData:
  READ a$
  col%=2^zSize%-1
  SetAPen RastPort,col%
  SetBPen RastPort,0
  SetDrMd RastPort,1
  Move RastPort,0,6
  Text RastPort,SADD(a$),LEN(a$)
  RETURN
ShowHAM:
  SetDrMd RastPort,1
  FOR g%=0 TO 15
    b%=0
    Move RastPort,0,g%*10
    SetAPen RastPort,0
    Draw RastPort,0,g%*10+9
    Move RastPort,1,g%*10
    SetAPen RastPort,g%+48
    Draw RastPort,1,g%*10+9
    FOR x%=0 TO 7
      FOR r%=1 TO 15
        SetAPen RastPort,r%+32
        Move RastPort,x%*32+r%+1,g%*10
        Draw RastPort,x%*32+r%+1,g%*10+9
      NEXT
      b%=b%+1
      Move RastPort,x%*32+17,g%*10
      SetAPen RastPort,b%+16
      Draw RastPort,x%*32+17,g%*10+9
      FOR r%=14 TO 0 STEP -1
        SetAPen RastPort,r%+32
        Move RastPort,x%*32+32-r%,g%*10
        Draw RastPort,x%*32+32-r%,g%*10+9
      NEXT
      b%=b%+1
      IF b%<16 THEN
        SetAPen RastPort,b%+16
        Move RastPort,x%*32+33,g%*10
        Draw RastPort,x%*32+33,g%*10+9
      END IF
    NEXT
  NEXT
  READ a$
  Move RastPort,0,180
  SetAPen RastPort,1
  SetBPen RastPort,0
  Text RastPort,SADD(a$),LEN(a$)
```

```
  GOTO warte
DATA "Farben:   32  Modus: Normal"
DATA 128,0,6
DATA "Farben:   64  Modus: Halfbright"
DATA 2048,0,6
DATA "Farben: 4096  Modus: HoldAndModify"
DATA 0,32,2,"OVERSCAN",-1
```

3.3 Fading (Ein- und Ausblenden von Grafiken)

Hiermit können Sie viele interessante Effekte erzielen. Zum Beispiel werden so Texte ein- oder ausgeblendet, oder manche Grafiken verändern kontinuierlich ihre Farbe (das heißt "cycle"). Alles wunderbare Fähigkeiten, die ein Programm erst ansprechend machen.

Wie vielleicht einige unter Ihnen, die den Begriff "Fading" noch nicht gehört haben, möchte ich ihn kurz erläutern. "Fading" ist das englische Wort für "Blenden" und wird im Zusammenhang mit dem Ein- oder Ausblenden benutzt. Eigentlich wird der Begriff in der Musik verwendet, viele moderne Lieder werden zum Schluß ausgeblendet. Im Englischen nennt man dies "Fade-Out". Das Gegenstück dazu wäre das "Fade-In". Nun kann man natürlich auch die beiden Ausdrücke auf die Optik beziehen. Ich denke dabei z.B. an das Ein- und Ausblenden von Bildern oder Schriftzügen. Genau dies soll jetzt beschrieben werden.

3.3.1 Fading - die Grundidee

Wie alle weiteren Programme dienen die nun folgenden natürlich nur als Beispiel. Sie können dann die Routinen ausbauen und in eigene Werke einsetzen. Das erste Programm zeigt die Grundidee. Es bietet die Möglichkeit, jede beliebige Farbe der Palette vom Schwarz ausgehend aufzublenden und diese dann wieder verdunkeln zu lassen. Damit wäre eigentlich schon die gesamte Programmleistung beschrieben. Sehen Sie sich einmal das dazugehörige Listing an:

```
'************************************
'*                                  *
'* Farbflächen ein- oder ausblenden *
'* -------------------------------- *
'*                                  *
'* Autor  : Wolf-Gideon Bleek       *
'* Datum  : Juni '87                *
'*          Mai  '88                *
'* Grueße : Schulze                 *
'* Version: 1.0                     *
'* Betriebssystem: V1.2 & V1.3      *
'*                                  *
'************************************
Variablen:
DEFINT a-z
In    =1          'Modus-Definition
Out   =-1
Anzahl=7
DIM SHARED Rot!(Anzahl),Grn!(Anzahl),Bla!(Anzahl)
HauptProgramm:
GOSUB BildschirmAufbauen
Blenden:
 GOSUB FarbenFestlegen
 CALL Fade (0,7,16,In)
 CALL Fade (0,7,16,Out)
GOTO Blenden
END
FarbenFestlegen:
FOR i=1 TO Anzahl
  Rot!(i)=RND
  Grn!(i)=RND
  Bla!(i)=RND
NEXT i
RETURN
BildschirmAufbauen:
SCREEN 2,640,256,3,2
WINDOW 1,"Farbtest",(0,0)-(623,200),0,2
FOR i=0 TO Anzahl
   PALETTE i,0,0,0
NEXT i
Breite=640/Anzahl
FOR j=0 TO 20
   FOR i=1 TO Anzahl
      x=RND*600
      y=RND*150
      LINE (x,y)-(x+Breite,y+Breite/2),i,bf
   NEXT i
NEXT j
RETURN
SUB Fade (Start,Anzahl,Schritte,Modus) STATIC
Anf=0 : Ende=Schritte
IF Modus=-1 THEN
  Anf=Schritte : Ende=0
```

```
END IF
FOR j=Anf TO Ende STEP Modus
   Faktor!=j/Schritte
   FOR i=Start TO Start+Anzahl
      PALETTE i,Rot!(i)*Faktor!,Grn!(i)*Faktor!,Bla!(i)*Faktor!
   NEXT i
NEXT j
END SUB
```

Variablenfelder

Bla()	Feld für Blauwerte
Grn()	Feld für Grünwerte
Rot()	Feld für Rotwerte

Variablen

Anf	Anfangsstadium der Farben
Anzahl	Anzahl der Farben
Im SUB:	Anzahl der zu blendenden Farben
Breite	Ausdehnung der Beispielflächen
Ende	Endstadium der Farben
Faktor	Farbanteil zum jetzigen Zeitpunkt
In	Pointer für das Aufblenden
Modus	Modusangabe: Ein- oder Ausblenden
Out	Pointer für das Ausblenden
Schritte	Anzahl der Schritte für den Vorgang
Start	Erste Farbnummer
i, j	Laufvariablen
x, y	Koordinaten der Beispielfelder

Programmbeschreibung

Die oben gegebenen Anmerkungen reichen nicht unbedingt aus, um alles zu verstehen. Hier folgen die genaueren Angaben. Das Programm definiert eine Funktion, die jede beliebige Farbe der Palette ein- oder ausblendet. Auch zusammenhängende Farbgruppen können gemeinsam 'geblendet' werden. Zuerst werden zwei Variablen für die Art des Fading mit Werten belegt. Somit brauchen Sie sich nicht die Zahlen zu merken, sondern Sie können einfach den Namen benutzen. Als nächstes legen wir die Auflösung mit sieben Farben fest (zuzüglich dem Hintergrund).

Dann wird für jede Farbe ein Feld definiert, auf das auch die SUB-Routine zugreifen kann. In diesem Feld werden die Farbwerte gespeichert, die beim Fading erreicht werden sollen.

Zur Demonstration wird daraufhin im Unterprogramm "Bildschirmaufbauen" ein neuer Screen geöffnet - mit der vorher eingestellten Farbtiefe. Auf ihm wird das Ausgabe-Window mit allerhand bunten Rechtecken vollgezeichnet.

Nun zum Hauptteil, der dann vom Programm abgearbeitet wird. Es wird ein Unterprogramm angesprungen, das die Farbfelder mit "zufälligen" Werten versorgt. Danach wird die wichtigste SUB-Routine gleich zweimal hintereinander aufgerufen. Als Parameter werden ihr die Nummer der ersten Farbe und deren Anzahl, die Abstufung und der Pointer übergeben. Letztes sagt aus, ob von Schwarz zur gewünschten Farbe (=einblenden) oder ausgeblendet werden soll. Die Abstufung bestimmt, in wie vielen Einzelschritten das geschehen soll, und die ersten beiden Werte dürften wohl klar sein.

Kommen wir nun zur Besprechung der eigentlichen Routine. Der Anfangswert wird in Abhängigkeit zum Pointer gesetzt, entweder 0 für Schwarz oder der Wert von "Schritte" für "vollständig" dargestellt. Auch das Ende wird dadurch bestimmt. Die Schleife, in der die Farbwerte in Teilschritten ausgegeben werden, berechnet für jede Abstufung einen Faktor und weist dann in einer inneren Schleife mit der PALETTE-Anweisung die neuen Farben zu. Dies wird so lange wiederholt, bis entweder die gewollte Farbe erreicht ist oder alles auf Schwarz abgedunkelt wurde.

3.3.2 Fade-Over

Dies ist eine Variante des obigen Programms. Hier wird nicht immer stur von Schwarz zur Farbe und dann wieder zum Schwarz geblendet, sondern man legt die Ausgangs- und Endfarbe frei fest. Sie können so jeden Übergang simulieren. Die Farben des Regenbogens dürften nun kein Problem mehr sein.

```
'*********************************
'*                               *
'*    Farbflächen umblenden      *
'*    ---------------------      *
'*                               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  : Juni '87             *
'*          Mai  '88             *
'* Grueße : Herrchen & Frauchen  *
'* Version: 1.0                  *
'* Betriebssystem: V1.2 & V1.3   *
'*                               *
'*********************************
Variablen:
 DEFINT a-z
 Anzahl=7
 DIM SHARED Rot!(Anzahl,1),Grn!(Anzahl,1),Bla!(Anzahl,1)
HauptProgramm:
GOSUB BildschirmAufbauen
Blenden:
 GOSUB FarbenFestlegen
 CALL Fade (0,7,8)
GOTO Blenden
END
FarbenFestlegen:
FOR i=1 TO Anzahl
   Rot!(i,0)=Rot!(i,1)
   Grn!(i,0)=Grn!(i,1)
   Bla!(i,0)=Bla!(i,1)
   Rot!(i,1)=RND
   Grn!(i,1)=RND
   Bla!(i,1)=RND
NEXT i
RETURN
BildschirmAufbauen:
SCREEN 2,640,256,3,2
WINDOW 1,"Farbtest",(0,0)-(623,200),0,2
FOR i=0 TO Anzahl
   PALETTE i,0,0,0
NEXT i
Breite=640/Anzahl
FOR j=0 TO 20
   FOR i=1 TO Anzahl
      x=RND*600
      y=RND*150
      LINE (x,y)-(x+Breite,y+Breite/2),i,bf
   NEXT i
NEXT j
RETURN
SUB Fade (Start,Anzahl,Schritte) STATIC
FOR j=0 TO Schritte
   FOR i=Start TO Start+Anzahl
      Rdiff!=(Rot!(i,1)-Rot!(i,0))/Schritte*j
```

```
      Gdiff!=(Grn!(i,1)-Grn!(i,0))/Schritte*j
      Bdiff!=(Bla!(i,1)-Bla!(i,0))/Schritte*j
      PALETTE i,Rot!(i,0)+Rdiff!,Grn!(i,0)+Gdiff!,Bla!(i,0)+Bdiff!
    NEXT i
NEXT j
END SUB
```

Programmbeschreibung

Die Grundstruktur des Programms ist eigentlich gleich geblieben. Jedoch sind kleine, aber feine Änderungen gemacht worden. Bei der Variablendefinition fällt auf, daß die beiden Pointer In und Out nicht mehr benötigt werden. Das ist klar, denn es wird immer zur neuen Farbe geblendet. Deswegen ist auch im Hauptprogramm ein Aufruf der Fade-Routine weggelassen worden. Es werden nur immer neue Farben ausgesucht und zu diesen wird dann überblendet.

Weil die Farbfelder jetzt eine weitere Dimension bekommen haben, mit der gekennzeichnet wird, ob es die Ausgangsfarbe (0) oder die Zielfarbe (1) ist, wird beim Festlegen der neuen Farbwerte der letzte neue Wert in das Register für den Ausgangswert kopiert, und die Zielwerte werden dann neu definiert. So kann das Programm immer auf den momentanen Zustand zugreifen, denn eine Abfragefunktion existiert nicht.

Die Fading-SUB-Routine geht nun in beliebig großen Schritten von der einen Farbe zur anderen. Dafür wird die Differenz durch die Schritte geteilt und mit der Zahl der schon vollzogenen Schritte multipliziert. Das Ergebnis jeder einzelnen RGB-Farbe wird zum entsprechenden Wert addiert. Wenn die äußerste Schleife durchlaufen ist, sind die neuen Farben erreicht.

3.3.3 Fading für jeden RGB-Anteil

Die letzte Fading-Möglichkeit ist eigentlich aus der ersten entstanden. Warum muß denn die Farbe gleich vollständig eingeblendet werden? Durch den PALETTE-Befehl haben wir auch die Möglichkeit, jede der RGB-Farben einzeln aufzublenden.

Deshalb erscheinen die Farben hier erst rot, dann wird Grün dazugemischt, und erst zum Schluß wird auch Blau hinzugegeben.

```
'**********************************
'*                                *
'* Farbflächen über RGB umblenden *
'* ------------------------------ *
'*                                *
'* Autor  : Wolf-Gideon Bleek     *
'* Datum  : Juni '87              *
'*          Mai  '88              *
'* Grueße : Jana & Svolli         *
'* Version: 1.0                   *
'* Betriebssystem: V1.2 & V1.3    *
'*                                *
'**********************************
Variablen:
 DEFINT a-z
 In    =1          'Modus-Definitoin
 Out   =-1
 Anzahl=7
 DIM SHARED Rot!(Anzahl),Grn!(Anzahl),Bla!(Anzahl)
HauptProgramm:
GOSUB BildschirmAufbauen
Blenden:
 GOSUB FarbenFestlegen
 CALL Fade (0,7,16,In)
 CALL Fade (0,7,16,Out)
GOTO Blenden
END
FarbenFestlegen:
FOR i=1 TO Anzahl
   Rot!(i)=RND
   Grn!(i)=RND
   Bla!(i)=RND
NEXT i
RETURN
BildschirmAufbauen:
SCREEN 2,640,256,3,2
WINDOW 1,"Farbtest",(0,0)-(623,200),0,2
FOR i=0 TO Anzahl
   PALETTE i,0,0,0
NEXT i
Breite=640/Anzahl
FOR j=0 TO 20
   FOR i=1 TO Anzahl
      x=RND*600
      y=RND*150
      LINE (x,y)-(x+Breite,y+Breite/2),i,bf
   NEXT i
```

```
NEXT j
RETURN
SUB Fade (Start,Anzahl,Schritte,Modus) STATIC
Schritte=Schritte/2
Anf=0 : Ende=Schritte
IF Modus=-1 THEN
  Anf=Schritte : Ende=0
END IF
AnfZu=Anf/Schritte
EndZu=Ende/Schritte
FOR j=Anf TO Ende STEP Modus
   Faktor!=j/Schritte
   FOR i=Start TO Start+Anzahl
      PALETTE i,Rot!(i)*Faktor!,Grn!(i)*AnfZu,Bla!(i)*AnfZu
   NEXT i
NEXT j
FOR j=Anf TO Ende STEP Modus
   Faktor!=j/Schritte
   FOR i=Start TO Start+Anzahl
      PALETTE i,Rot!(i)*EndZu,Grn!(i)*Faktor!,Bla!(i)*AnfZu
   NEXT i
NEXT j
FOR j=Anf TO Ende STEP Modus
   Faktor!=j/Schritte
   FOR i=Start TO Start+Anzahl
      PALETTE i,Rot!(i)*EndZu,Grn!(i)*EndZu,Bla!(i)*Faktor!
   NEXT i
NEXT j
END SUB
```

Programmbeschreibung

Bis auf die SUB-Routine gleicht das Listing dem ersten dieses Abschnittes. Ich empfehle deswegen, diesen Teil zu kopieren. Sie ersparen sich viel Tipparbeit. Als erstes kürzt die Routine die angegebene Schrittezahl um die Hälfte. Dies wurde deshalb gemacht, damit Sie bei allen Programmen etwa gleiche "Geschwindigkeitseinstellungen" machen können. Denn hier wird die gleiche Schleife dreimal durchlaufen. Es dauert so fast dreimal solange! Zu Beginn wird wieder der Startwert für die Blendschleife gesucht. Entweder wird mit dem Schwarzwert begonnen oder aber mit der Farbe, das stellen Sie mit dem Modus-Pointer ein.

Weil die PALETTE-Anweisung immer alle Farbwerte braucht, muß z.B. in der ersten Schleife, die nur den Rotwert beeinflußt, bei den beiden anderen Werten der Anfangszustand gesetzt wer-

den. In den anderen Schleifen braucht das Programm aber die Endwerte, da der Rotwert schon abgehandelt wurde. Dafür berechnet das Unterprogramm am Anfang zwei Faktoren (AnfZu, EndZu). Sonst aber läuft hier alles genau wie im ersten Programm ab.

3.4 Schnelle Vektorgrafik

Bei der Vektorgrafik werden alle Objekte nicht vollständig durch ihre Flächen dargestellt, sondern nur ihre Kanten, die Vektoren. Dadurch wird eine schnelle Darstellung ermöglicht, denn die aufwendige Zeichenarbeit für die großen Oberflächen entfällt ganz und ist nur auf die Eckpunkte und die dadurch resultierenden Kanten beschränkt.

3.4.1 Gittermodelle darstellen

Für die Verarbeitung eines 3D-Körpers speichern wir seine Eckpunkte als dreidimensionale Koordinaten. Zusätzlich wird noch eine Verbindungsvorschrift aufgestellt, nach der alle Koordinaten-Tripel verbunden werden. Hat man alle diese Daten, so müssen sie vom Raum auf die Bildschirmfläche projiziert werden, denn die Darstellung erfolgt auf einer Fläche. In dem folgenden Programm wurde eine zentrische Abbildung auf der Bildschirmebene gewählt. Als Information für die Künstler und Zeichner unter Ihnen: Alle Objekte werden durch eine Ein-Fluchtpunkt-Perspektive abgebildet.

Da die Ebene, unser Bildschirm, durch ihre z-Koordinate eindeutig festgelegt ist, wird dieser Wert bei allen Punkten uninteressant. Darauf baut nun die Überlegung auf, das Gitternetz abzubilden. Um die x- und y-Koordinaten auf dem Bildschirm zu finden, stellen wir uns einen Raum vor, in dem sich der Körper befindet. Weiterhin denken wir uns irgendwo im Raum einen Punkt, er wird im folgenden "Fluchtpunkt" genannt.

Zwischen dem Körper und dem Fluchtpunkt liegt unser Bildschirm als Ebene, die durch ihren z-Wert bekannt ist. Jetzt ziehen wir von jedem Eckpunkt unseres Körpers eine Strecke zum Fluchtpunkt. Da, wo sich diese Strecke mit der Bildschirmebene schneidet, finden wir die gesuchten x- und y-Werte für diesen Eckpunkt und damit seine Lage auf dem Bildschirm!

Wie soll nun ein Programm aufgebaut sein, das diese Abbildung vornimmt? Am wichtigsten sind sicherlich die Daten der Eckpunkte. Damit wir erst einmal damit arbeiten können und nicht zuviel Aufwand und Umstand betreiben, habe ich sie in DATA-Zeilen abgelegt. Zusätzlich zu den Eckpunktkoordinaten müssen noch die Verbindungsvorschriften vorhanden sein, auch diese legen wir in DATA-Zeilen ab. In späteren Versionen des Programms können Sie zusätzlich von Speicherroutinen einbauen, um mit verschiedenen Körpern und verschiedenen Bewegungen zu arbeiten.

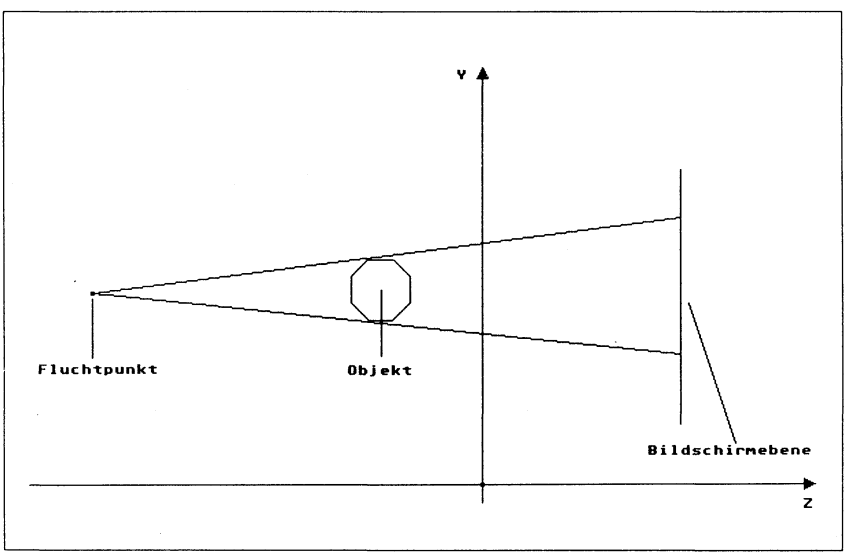


Abb. 1 Fluchtpunktperspektive

Wenn das Programm alle Raumkoordinaten kennt, kann es mit der Berechnung der Bildschirmkoordinaten beginnen. Dazu verwendet es die Streckenformel im dreidimensionalen Raum:

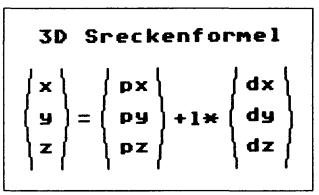


Abb. 2 3D-Streckenformel

Um die Formel benutzen zu können, müssen Sie folgendes wissen: Die gesuchten Bildschirmkoordinaten heißen x und y. Der z-Wert ist uns bekannt, weil wir ihn festlegen. Die p-Koordinaten wissen Sie, weil sie zu dem Punkt gehören, mit dem wir die Berechnung durchführen wollen. Bleiben nur noch die d-Werte. Es sind die Differenzen der einzelnen Koordinaten des Punktes und des Fluchtpunktes (px-fx, py-fy, pz-fz). Die kennen Sie auch, wenn Sie sie berechnet haben!

```
'*********************************
'* 3D - Netzgrafik               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  :  8. Mai 1987         *
'*           14. Mai 1988        *
'* Grueße : Franky               *
'* Version: 1.0                  *
'* Betriebssystem: V1.2 & V1.3   *
'*********************************
Variablen:
 RESTORE WuerfelDaten
 DEFINT B, V
 Punkte  = 25          ' Anzahl der Objektpunkte
 ZEbene  =-25          ' Bildschirmebene
 PktAnz  =  0          ' Anzahl der Objektpunkte
 Verbind =  0          ' Anzahl der Verbindungen
 OPTION BASE 1
 DIM P(Punkte,3)       ' Raumkoordinaten
 DIM B(Punkte,2)       ' Bildkoordinaten
 DIM V(Punkte*1.8,2)   ' Verbindungsvorschrift
 DIM D(3)              ' Differenz
 DIM F(3)              ' Fluchtpunkt (x,y,z)
 F(1)=-70              ' Fluchtpunkt x
```

```
 F(2)=-50              ' y
 F(3)=240              ' z
Hauptprogramm:
 PRINT "Fluchtpunkt (x,y,z) ? ";F(1)","F(2)","F(3)
 PunkteLesen:
 Grund=PktAnz          ' Basis fr Verbindungen
Loop:
 READ px,py,pz
 IF px<>255 THEN
   PktAnz=PktAnz+1
   P(PktAnz,1)=px
   P(PktAnz,2)=py*-1
   P(PktAnz,3)=pz
   GOTO Loop
 END IF

VerbindLesen:
 READ v1,v2
 IF v1<>255 THEN
   Verbind=Verbind+1
   V(Verbind,1)=Grund+v1
   V(Verbind,2)=Grund+v2
   GOTO VerbindLesen
 END IF
 READ Ende
 IF Ende<>0 THEN GOTO PunkteLesen
BildBerechnen:
 FOR i=1 TO PktAnz
    FOR j=1 TO 3
       D(j)=F(j)-P(i,j)
    NEXT j
    lambda=(ZEbene-P(i,3))/D(3)
    B(i,1)=P(i,1)+lambda*D(1)
    B(i,2)=P(i,2)+lambda*D(2)
 NEXT i
BildAusgeben:
 FOR i=1 TO Verbind
    x1=B(V(i,1),1)+50
    x2=B(V(i,2),1)+50
    y1=B(V(i,1),2)+100
    y2=B(V(i,2),2)+100
    LINE (x1,y1)-(x2,y2)
 NEXT i
END
WuerfelDaten:
 REM  x,y,z
 DATA  32, 20, 20
 DATA -32, 20, 20
 DATA -32,-20, 20
 DATA  32,-20, 20
 DATA  32, 20,-20
 DATA -32, 20,-20
 DATA -32,-20,-20
```

```
 DATA  32,-20,-20
 DATA 255,0,0
 REM  p1,p2
 DATA 1,2
 DATA 2,3
 DATA 3,4
 DATA 4,1
 DATA 1,5
 DATA 5,6
 DATA 6,7
 DATA 7,8
 DATA 8,5
 DATA 4,8
 DATA 3,7
 DATA 2,6
 DATA 255,0,1

PyramidenDaten:
 DATA -32, 25,-20
 DATA  32, 25,-20
 DATA  32, 25, 20
 DATA -32, 25, 20
 DATA   0, 65,  0
 DATA 255,0,0
 DATA 1,2
 DATA 2,3
 DATA 3,4
 DATA 4,1
 DATA 5,1
 DATA 5,2
 DATA 5,3
 DATA 5,4
 DATA 255,0,0
```

Variablenfelder

P()	Raumkoordinaten
B()	int, Bildkoordinaten
D()	Differenzen der Abbildung
F()	Fluchtpunktkoordinaten
V()	int, Verbindungsvorschrift für alle Objekte

Variablen

Ende	gelesener Wert, der bei Ende = 0 ist
Grund	Bezug für die Verbindungen eines Objektes
PktAnz	Anzahl aller abzubildenden Punkte
Punkte	maximale Anzahl der Objektpunkte
Verbind	Anzahl der Verbindungen

ZEbene	z-Koordinate der Bildschirmebene
i, j	Laufvariablen
lambda	Faktor der Koordinatenumrechnung
px, py, pz	Koordinaten eines Punktes im Raum
v1	erster Punkt einer Verbindung
v2	zweiter Punkt einer Verbindung
x1, y1	Bildkoordinaten für die Ausgabe (1. Punkt)
x2, y2	Bildkoordinaten für die Verbindung (2. Punkt)

Programmbeschreibung

Zuerst, bei der Variablendefinition, wird der DATA-Pointer auf den Anfang der Punktdaten gesetzt. In diesem Fall sind es die Koordinaten eines Würfels. Dann werden alle Variablen, die mit einem B oder V anfangen, als Integer eingestuft. Sie werden gleich sehen, warum. Damit später die Felder für die Punkte dimensioniert werden können, hält das Programm in der Variablen "Punkte" fest, wie viele Punkte maximal gespeichert werden sollen. Außerdem wird die Lage der Bildschirmebene im Raum durch die z-Koordinate eingestellt. Dann wird die Anzahl der gelesenen Punkte und Verbindungen erst einmal auf Null gesetzt.

Nun folgt die Dimensionierung der benötigten Variablenfelder. Es sind dies einmal das Feld P, in dem die Punktkoordinaten gespeichert werden (deswegen mit dem Index 3), dann das Feld B (Integer), in dem die späteren Bildschirmkoordinaten zu jedem Raumpunkt abgelegt werden. Außerdem das Feld V (auch integer), das immer zwei Punktnummern enthält, die angeben, welche Punkte miteinander verbunden werden sollen. Und als letztes das Feld D mit drei Elementen, das für die Differenzen bei der Punktberechnung gebraucht wird. Auch die Position des Fluchtpunktes wird in einem Feld F abgelegt, anstatt mit einem Buchstaben als Index (FPx, FPy, FPz), weil somit die Berechnungen automatisiert werden können. Sehen wir uns nun das Hauptprogramm an:

Um die Grafik später besser beurteilen zu können, werden in der nächsten Zeile die Fluchtpunkt-Koordinaten ausgegeben. Dann folgt die Punktlese-Routine. In ihr wird zuerst der Pointer "Grund" auf die erste Nummer des zu lesenden Punktes gesetzt.

Er dient zur Verarbeitung von mehreren Objekten, so brauchen Sie später nicht auszuzählen, welche Nummer der erste Punkt im siebten Objekt hat, Sie schreiben einfach eine Eins. In der Schleife werden Raumkoordinaten gelesen und es wird überprüft, ob der px-Wert gleich 255 ist. Dies ist die Kennung dafür, daß alle Punkte eines Objektes gelesen sind und nun die Verbindungsvorschriften folgen. Sonst wird der neue Punkt in die Tabelle eingetragen, und von neuem werden Koordinaten gelesen.

Die Schleife zum Lesen der Verbindung arbeitet eigentlich nach dem gleichen Prinzip. Zuerst werden die Nummern der beiden Punkte gelesen, die verbunden werden sollen. Entsprechen sie der Endkennung, so wird die Schleife verlassen, andernfalls werden die beiden Zahlen in das Feld eingetragen, und alles fängt von vorne an. Als letztes wird eine Zahl aus den Daten gelesen, die Auskunft darüber gibt, ob noch ein weiterer Körper folgt. Dieser Fall ist gegeben, wenn der Wert ungleich null war.

Sind beide Schleifen endgültig abgeschlossen, werden die Bildschirmpunkte des Objektes berechnet. Dies geschieht in einer Schleife, die einfach die Liste Punkt für Punkt durchgeht und für jeden die Bildschirmwerte ausrechnet. Das Verfahren ist oben schon beschrieben, trotzdem möchte ich noch einmal darauf eingehen. Nachdem die Differenzen der Fluchtpunktwerte und des aktuellen Punktes in dem Feld D abgelegt worden sind, wird der Lambda-Faktor berechnet. Darauf setzt das Programm den gewonnenen Wert in die Gleichungen für die x- und y-Werte ein. Fertig!

Zum Schluß folgt noch die Ausgabe des Gitternetzes. Hierfür durchläuft eine Schleife alle Verbindungen und sucht sich für jede die benötigten Punktkoordinaten heraus. Dabei kann es durchaus vorkommen, daß ein vorher berechneter Punkt gar nicht gebraucht wird, weil er in keiner Verbindung auftaucht. Da das Objekt in der Nähe des Nullpunktes definiert wurde, muß es jetzt noch, um es sichtbar darzustellen, in die Bildschirmmitte verschoben werden. Dann wird Linie für Linie gezeichnet.

3.4.2 Gittermodelle bewegen

Die Bewegung ist grundsätzlich nur eine Aneinanderreihung von stehenden Bildern. Deshalb können wir aufbauend auf die Darstellung programmieren und nur die Raumkoordinaten für jedes Bild leicht beeinflussen. Leider stellte sich aber heraus, daß es damit nicht getan ist, denn die Bewegung wird dann viel zu langsam.

Für eine flüssige Bewegung auf dem Bildschirm darf während der Ausgabe zuerst einmal nicht gerechnet werden, alle Werte müssen schon vorher feststehen. Außerdem reicht die einfache Art mit vielen LINE-Befehlen nicht aus. Da es eine Betriebssystemroutine gibt, die für die Ausgabe zusammengehöriger Linien zuständig ist, wollen wir diese verwenden.

3.4.3 Beschleunigung durch Betriebssystem-Routinen

Die Entwickler des Amiga-Betriebssystems haben sich viele Gedanken über die Anwendungen gemacht, die später auf dem Supercomputer laufen werden. Sicherlich kannten sie auch die Netzgrafik. Denn nur mit ihr ist eine Echtzeitbetrachtung unter abgespeckten Bedingungen möglich. Deshalb schufen sie eine Grafikroutine, die alle Punkte einer Liste nacheinander verbindet. Genau diese Routine ist die letzte Möglichkeit für uns, noch eine schnellere Darstellung zu bekommen. Durch sie lassen wir unser Gitternetz zeichnen. Dafür legen wir wie immer zuerst die Eckpunkte durch ihre Raumkoordinaten ab, die später auf den Bildschirm projiziert werden. Die Eckpunkte werden für die Bewegung im Raum bewegt, auch dies bleibt beim alten. Damit wir die durch die schnelle Routine gewonnene Zeit nicht wieder verlieren, berechnet das Programm vor der Ausgabe alle Szenen und legt diese in ein Feld ab.

Nun kommt aber das erste Problem! Die Routine erwartet eine Liste der Bildschirmkoordinaten, die in ihrer angegebenen Reihenfolge auch verbunden werden. Dies ist ein großer Nachteil

und zugleich ein Vorteil. Einmal brauchen nicht immer Koordinatenpaare gespeichert zu werden, zum anderen muß aber die Figur so aufgebaut sein, daß sie mit einer durchgehenden Linie gezeichnet werden kann. Wenn nicht, so werden manche Kanten mehrmals abgeschritten, was eigentlich unnötig ist. Aber es gibt eben Objekte, bei denen sich mit nur einer Endloslinie nichts machen läßt.

Wegen dieser Vorschrift des Betriebssystems müssen Sie die Verbindungsvorschrift ändern. Sie geben jetzt nicht mehr Koordinatenpaare ein, sondern Sie müssen auf den Kanten des Objektes entlangwandern (keine vergessen!) und tragen nacheinander die Nummern der Eckpunkte ein, die Sie in Gedanken überschritten haben.

Wenn das Programm diese Daten hat, kann es mit den Berechnungen beginnen. Zuerst erfolgt das Verschieben des Körpers im Raum mit dem Ablegen der Bildschirmkoordinaten, dann der neue Grafiktransfer. In diesem Teil werden die vorhandenen Bildschirmwerte in eine lange Liste eingetragen, die später der Betriebssystemroutine in Teilabschnitten übergeben wird.

Ist die Liste vollständig erstellt, geht es in der Ausgabeschleife weiter. Hier werden wieder alle Szenen durchlaufen, und ein entsprechender Pointer zeigt in der Liste auf die Daten für die aktuelle Szene. Dann werden die Werte an die Routine übergeben, die diese Ausgabe übernimmt. Um auch gleich wieder das Bild zu löschen, wird die Farbe auf Hintergrund gesetzt und das Objekt wird von neuem gezeichnet. Wenn alle Bilder ausgegeben worden sind, springt das Programm zum Anfang der Ausgabe und alles startet wieder.

```
'*********************************
'* 3D - Netzgrafik               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  :    Juni 1987         *
'*          14. Mai 1988         *
'* Version: 1.5                  *
'* Betriebssystem: V1.2 & V1.3   *
'*********************************
LIBRARY ":bmaps/graphics.library"
RESTORE
```

```
OPTION BASE 1
Variablen:
 DEFINT B,V,G
 READ Punkte              ' Anzahl der Objektpunkte
 READ Verbind             ' Zähler der Verbindungen
 ZEbene=25                ' Bildschirm Ebene
 Szenen=50                ' Anzahl der Szenen
 DIM P(Punkte,3)          ' Raumkoordinaten
 DIM B(Szenen,Punkte,2)   ' Bildkoordinaten
 DIM G(Verbind*2*Szenen)
 DIM V(Verbind)           ' Verbindungsvorschrift
 DIM D(3)                 ' Differenz
 DIM F(3)                 ' Fluchtpunkt (x,y,z)
 F(1)=-70                 ' Fluchtpunkt x
 F(2)=-50                 ' y (neg. wg. nichtkathes. Koord.Sys.)
 F(3)=180                 ' z
PRINT "Fluchtpunkt (x,y,z) : ";F(1)","F(2)","F(3)
PunkteLesen:
 RESTORE PyramidenDaten   ' Objekt
 FOR i=1 TO Punkte
  READ px,py,pz
   P(i,1)=px
   P(i,2)=py*-1           ' Transfer in anderes Koordinatensystem
   P(i,3)=pz
 NEXT i

VerbindLesen:
 FOR i=1 TO Verbind
   READ V(i)
 NEXT i
BildVorausBerechnen:
 FOR sz=1 TO Szenen
    FOR i=1 TO Punkte
       FOR j=1 TO 3
          D(j)=F(j)-P(i,j)
       NEXT j
       P(i,3)=P(i,3)+3
       P(i,2)=P(i,2)-2
       P(i,1)=P(i,1)+2
       Lambda=(ZEbene-P(i,3))/D(3)
       B(sz,i,1)=P(i,1)+Lambda*D(1)+200
       B(sz,i,2)=P(i,2)+Lambda*D(2)+200
    NEXT i
 NEXT sz
GraphikTransfer:
 FOR j=0 TO Szenen-1
    FOR i=1 TO Verbind*2 STEP 2
       G(i+j*Verbind*2)=B(j+1,V(i/2+.5),1)
       G(i+1+j*Verbind*2)=B(j+1,V(i/2+.5),2)
    NEXT i
 NEXT j
BildAusgeben:
```

```
 FOR i=0 TO Szenen-1
    Pointer=Verbind*2*i
    FOR j=1 TO 0 STEP-1
       COLOR j
       CALL Move(WINDOW(8),G(1+Pointer),G(2+Pointer))
       CALL PolyDraw(WINDOW(8),Verbind-1,VARPTR(G(3+Pointer)))
    NEXT j
 NEXT iq
GOTO BildAusgeben
GraphikDaten:
 DATA 5,10
 ' Punkte,Verbindungen
PyramidenDaten:
 DATA -32, 25,-20
 DATA  32, 25,-20
 DATA  32, 25, 20
 DATA -32, 25, 20
 DATA   0, 65,  0
Verbindungen:

 DATA 2,1,5,4,3,5,2,3,4,1
 DATA 4,1
```

Variablenfelder

B()	Bildschirmkoordinaten
D()	Differenz der Koordinatenumrechnung
F()	Fluchtpunktkoordinaten
G()	Zusammenfassung: Koordinaten aller Szenen
P()	Raumkoordinaten aller Punkte
V()	Verbindungsvorschrift

Variablen-Liste

Lambda	Faktor der Koordinatenberechnung
Pointer	Zeiger auf die Koordinatenliste einer Szene
Punkte	Anzahl aller Objektpunkte
Szenen	Anzahl der zu berechnenden Szenen
Verbind	Anzahl der Verbindungselemente
ZEbene	z-Position der Ebene, auf die abgebildet wird
i, j	Laufvariablen
px, py, pz	Raumkoordinaten eines Eckpunktes
sz	Schleifenzähler für die Szenen

Programmbeschreibung

Bevor die Variablen definiert werden, öffnet das Programm die graphics-Library. Hierdurch können Sie die Grafikroutinen ansprechen, die für die Gitternetzausgabe gebraucht werden. Dann werden alle Variablen, die mit B, V oder G beginnen, als Integer deklariert. Dies erspart das ständige Anhängen von Typenzeichen. Für die Gitternetzausgabe ist das neue Feld G hinzugekommen, in ihm werden alle Koordinaten in ihrer Verbindungsreihenfolge abgelegt. Und zwar immer ein 2-Byte-Integerwert für die x-Koordinate und ein 2-Byte-Integerwert für die y-Koordinate.

Neu im Programm ist die Punkt- und Verbindungsleseschleife. Sie gehen jetzt von festen Werten aus, die am Anfang des Programms am Kopf der DATA-Zeilen gelesen werden. Somit ersparen Sie sich die Endkennung, und das Programm wird etwas schneller. Das Verbindungsfeld ist nur noch mit einer Dimension definiert, da es keine Paare, sondern die Kette der zu verbindenden Punkte speichert.

Nach der Berechnung müssen die so gewonnenen Daten in ein Format gebracht werden, das die Betriebssystemroutine verarbeiten kann. Die Routine mit dem Namen PolyDraw braucht einmal eine Tabelle, in der nacheinander die x- und y-Werte als kurze Integer-Zahlen stehen. Andererseits muß sie auch wissen, wie viele Elemente sie davon verwenden soll. Die Tabelle kann also ruhig sehr lang sein, es werden nur so viele Daten gelesen, wie vorgeschrieben. Damit wird ein Pointer für das Ende gespart. Wir legen die Grafikdaten für alle Szenen in das eine Feld und übergeben der Routine immer die Adresse des ersten Elements für die nächste Szene, außerdem geben wir ihr noch die Zahl der Eckpunkte mit auf den Weg. Alles andere erledigt PolyDraw von selbst.

Die Ausgabe erfolgt in einer neuen Schleife. Sie wird entsprechend der Anzahl der Szenen durchlaufen. In ihr wird erst ein Zeiger berechnet, der auf das erste Element für die Ausgabe des Gitternetzes zeigt. Dann folgt eine zweite Schleife, die zweimal

durchlaufen wird. Zuerst zeichnet sie das Netz, indem der Grafik-Cursor auf den Anfangspunkt gesetzt wird und die Poly-Draw-Routine ihre Aufgabe erledigt.

Beim zweiten Durchlauf ist der Wert der Laufvariablen von Eins auf Null gesetzt, und somit schaltet der COLOR-Befehl von der ersten Zeichenfarbe auf die Hintergrundfarbe. Obwohl jetzt wieder das Gitternetz gezeichnet wird, wird es für den Betrachter effektiv gelöscht. Dieser Vorgang wiederholt sich solange, bis alle Szenen geplottet worden sind. Dann fängt die Ausgabeschleife wieder von vorne an.

3.4.4 3D-Bilder für die Rot-Grün-Brille

Beim Experimentieren mit Mehrfluchtpunktsystemen und dem zufälligen Betrachten eines 3D-Bildes kam mir die Idee, eine Rot-Grün-Brille einzusetzen. Zuerst soll das Prinzip erklärt werde, nach dem die 3D-Brille funktioniert.

Da der Mensch zwei Augen hat, sieht er auch zwei verschiedene Bilder. Aus diesen Bildern "errechnet" das Gehirn die unterschiedlichen Raumlagen aller Gegenstände. Wie soll man nun mit einem Bild die beiden Informationen für das Gehirn bereithalten? Da tritt die Brille in Aktion! Die farbigen Folien der 3D-Brille (meist sind es rote und grüne) filtern aus dem Bild die beiden Einzelbilder heraus.

Durch das rote Glas kann man nur das rote Licht sehen. Versuchen Sie es! Und dementsprechend können Sie durch das grüne Glas nur alles Grüne sehen. Das Problem, daß an manchen Stellen beide Farben vorhanden sein können, hat man durch konsequentes Mischen beider Farben gelöst. Denn sie werden durch die Farbfilter wieder getrennt. So kann man zwar nur einfarbige Bilder darstellen, aber der Effekt ist großartig.

Wie wurde diese Möglichkeit in ein Programm umgesetzt? Die dargestellten 3D-Bilder sind natürlich wieder Gitternetze, wie auch in den vorigen Programmen. Das Programmierprinzip be-

ruht nun auf der Tatsache, daß jedes Auge einen eigenen Fluchtpunkt für sein Bild haben muß. Weil beide Augen etwas auseinander liegen, müssen folglich auch die Fluchtpunkte etwas versetzt sein. Im Programm wird jetzt nicht mehr ein Gitternetz berechnet und gezeichnet, sondern zwei Bilder mit auf der Horizontalen verschobenen Fluchtpunkten. Wie besprochen wird ein Bild in Rot und das andere in Grün geplottet. Alle Stellen, an denen sich beide Farben überlagern, werden in Braun, der additiven Mischfarbe, dargestellt.

Damit die Bedienung auch richtig komfortabel gestaltet ist, habe ich Schieberegler für die Farbeinstellung gewählt. So können Sie die Farbintensität von Rot, Grün und Braun selber einstellen (für verwöhnte Augen!). Außerdem können Sie die Lage der Fluchtpunkte verändern, um einen optimalen 3D-Effekt zu erzielen. Wenn Sie alles nach Ihren Wünschen eingestellt haben, drücken Sie eine Taste, und das Programm gibt Ihnen alle Werte aus. Somit können Sie die Werte fest einbauen oder für eigene 3D-Programme nutzen.

```
'*********************************
'*                               *
'* 3D - Netzgrafik               *
'* ----------------------------- *
'*                               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  : 24. Mai 1987         *
'*          14. Mai 1988         *
'* Gruesse : Lars                *
'* Version: 2.0                  *
'* Betriebssystem: V1.2 & V1.3   *
'*                               *
'*********************************
LIBRARY ":bmaps/graphics.library"
RESTORE WuerfelDaten
DEFINT B,V
OPTION BASE 1
Variablen:
Punkte  = 25          ' Anzahl der Objektpunkte
ZEbene  =-25          ' Bildschirmebene
PktAnz  =  0          ' Anzahl der Objektpunkte
Verbind =  0          ' Anzahl der Verbindungen
ClickAnz=0
clickmoeg=20
DIM SHARED ClickTab(clickmoeg,4)
DIM SHARED ClickWrt(clickmoeg)
```

```
DIM SHARED ClickArt(clickmoeg)
DIM P(Punkte,3)          ' Raumkoordinaten
DIM B(2,Punkte,2)        ' Bildkoordinaten
DIM V(Punkte*1.8,2)      ' Verbindungsvorschrift
DIM D(3)                 ' Differenz
DIM F(2,3)               ' Fluchtpunkt (x,y,z)
F(1,1)=-40               ' 1. Fluchtpunkt x
F(1,2)=-50               ' y
F(1,3)=240               ' z
F(2,1)=-80               ' 2. Fluchtpunkt x
F(2,2)=-50               ' y
F(2,3)=240               ' z
TextAusgabe:
CLS
LOCATE 1,40
PRINT "Fluchtpunkt 1 (x,y,z) :"
LOCATE 2,40
PRINT "Fluchtpunkt 2 (x,y,z) :"
GOSUB KoordinatenAusg
Farbeinstellungen:
PALETTE 0,.6,.55,.4      ' Hintergrund = hell-beige
PALETTE 1,.4,.35,0       ' Neutrale Farbe = dunkel Braun
PALETTE 2,.7,0,0         ' Rot 70%
PALETTE 3,0,.65,0        ' Grn 65%
Schieberegler:
Text$="Rot"
DefMove 40!,8!,100!,70!,2!
Text$="Grün"
DefMove 45!,8!,100!,65!,2!
Text$="Braun"
DefMove 50!,8!,100!,40!,2!
Text$="FPkt1"
DefMove 60!,8!,100!,40!,2!
Text$="FPkt2"
DefMove 65!,8!,100!,80!,2!
PunkteLesen:
Grund=PktAnz             ' Basis für Verbindungen
Loop:
 READ px,py,pz
 IF px<>255 THEN
   PktAnz=PktAnz+1
   P(PktAnz,1)=px
   P(PktAnz,2)=py*-1
   P(PktAnz,3)=pz
   GOTO Loop
 END IF

VerbindLesen:
 READ v1,v2
 IF v1<>255 THEN
   Verbind=Verbind+1
   V(Verbind,1)=Grund+v1
   V(Verbind,2)=Grund+v2
```

```
   GOTO VerbindLesen
 END IF
 READ Ende
 IF Ende<>0 THEN GOTO PunkteLesen
BildBerechnen:
 FOR k=1 TO 2              ' 2 Fluchtpunkte
    FOR i=1 TO PktAnz      ' Alle Punkte
       FOR j=1 TO 3        ' Diff von x,y,z
          D(j)=F(k,j)-P(i,j)
       NEXT j
       lambda=(ZEbene-P(i,3))/D(3)
       B(k,i,1)=P(i,1)+lambda*D(1)
       B(k,i,2)=P(i,2)+lambda*D(2)
    NEXT i
 NEXT k
BildAusgeben:
 LINE (0,0)-(300,200),0,bf    ' Fläche löschen
 FOR j=1 TO 2
    COLOR 1+j
    IF j=2 THEN CALL SetDrMd&(WINDOW(8),7)
       FOR i=1 TO Verbind
       x1=B(j,V(i,1),1)+100
       x2=B(j,V(i,2),1)+100
       y1=B(j,V(i,1),2)+70
       y2=B(j,V(i,2),2)+70
       LINE (x1,y1)-(x2,y2)
    NEXT i
 NEXT j
 CALL SetDrMd&(WINDOW(8),1)
 COLOR 1
Interrupt:
 ON MOUSE GOSUB CheckTab
 ON TIMER (.5) GOSUB ColorSet
 TIMER ON
 MOUSE ON
Warten:
 IF ClickWrt(4)*-1<>F(1,1) THEN
    F(1,1)=ClickWrt(4)*-1
    NeuZeichnen:
    GOSUB KoordinatenAusg
    GOTO BildBerechnen
 END IF
 IF ClickWrt(5)*-1<>F(2,1) THEN
    F(2,1)=ClickWrt(5)*-1
    GOTO NeuZeichnen
 END IF
 IF INKEY$="" THEN GOTO Warten
 OBJECT.OFF
 TIMER OFF
 MOUSE OFF
 LOCATE 15,1
 PRINT "Rotwert :";ClickWrt(1);"%"
 PRINT "Grünwert:";ClickWrt(2);"%"
```

```
 PRINT "Braunwert aus :"
 PRINT ClickWrt(3);"% Rot und "ClickWrt(3)*.875;"% Grün"
 PRINT "Fluchtpunktwerte der X-Koordinaten:"
 PRINT "F1 ";ClickWrt(4)*-1;" und F2 ";ClickWrt(5)*-1
 END
KoordinatenAusg:
 LOCATE 1,63
 PRINT F(1,1)","F(1,2)","F(1,3)
 LOCATE 2,63
 PRINT F(2,1)","F(2,2)","F(2,3)
 RETURN
CheckTab:
IF ClickAnz=0 THEN RETURN
FOR i=1 TO ClickAnz
   mstat=MOUSE(0)
   mx=MOUSE(1)-6
   my=MOUSE(2)
   IF mx>=ClickTab(i,1) THEN
     IF my>=ClickTab(i,2) THEN
       IF mx<=ClickTab(i,3) THEN
         IF my<=ClickTab(i,4) THEN

           ClickWrt(i)=(my-ClickTab(i,2))
           OBJECT.Y i,ClickTab(i,2)+ClickWrt(i)+12

         END IF
       END IF
     END IF
   END IF
NEXT i
IF MOUSE(0)=-1 THEN CheckTab
RETURN
ColorSet:
 Rot=ClickWrt(1)/100
 Gruen=ClickWrt(2)/100
 Zeichen=ClickWrt(3)/100
 PALETTE 2,Rot,0,0
 PALETTE 3,0,Gruen,0
 PALETTE 1,Zeichen,(Zeichen*.875),0
RETURN
SUB DefMove (sx,sy,yd,po,mo) STATIC
  SHARED ClickAnz

x=sx*8   'Koordinaten fuer Line *10 bei 60 Zeichen
y=sy*8
LINE (x,y)-(x+20,y+8+yd),,B
'Extras erwuenscht?
IF mo AND 1 THEN  ' Skalierung
  FOR sk=y TO y+yd+8 STEP (yd+8)/16 '16 Einheiten
     LINE (x,sk)-(x+2,sk)
     LINE (x+20,sk)-(x+18,sk)
  NEXT sk
END IF
```

```
IF mo AND 2 THEN   ' Text
  SHARED Text$
  sy=sy-LEN(Text$)
  FOR txt=1 TO LEN(Text$)
     LOCATE sy+txt,sx+2
     PRINT MID$(Text$,txt,1)
  NEXT txt
END IF
'Clickwerte in Tabelle eintragen
ClickAnz=ClickAnz+1
ClickTab(ClickAnz,1)=x
ClickTab(ClickAnz,2)=y
ClickTab(ClickAnz,3)=x+20
ClickTab(ClickAnz,4)=y+yd
ClickArt(ClickAnz)=1      '1 steht fuer Schieber
ClickWrt(ClickAnz)=po     'vom Benutzer gewaehlter Anfangswert
' Als Schieber sollte ein Sprite in Kreisform definiert werden!
OPEN "Schieber" FOR INPUT AS ClickAnz
OBJECT.SHAPE ClickAnz,INPUT$(LOF(ClickAnz),ClickAnz)
CLOSE ClickAnz
OBJECT.X ClickAnz,x-1
OBJECT.Y ClickAnz,ClickTab(ClickAnz,2)+ClickWrt(ClickAnz)+12
OBJECT.ON ClickAnz
END SUB
WuerfelDaten:
REM  x,y,z
DATA  32, 20, 20
DATA -32, 20, 20
DATA -32,-20, 20
DATA  32,-20, 20
DATA  32, 20,-20
DATA -32, 20,-20
DATA -32,-20,-20
DATA  32,-20,-20
DATA 255,0,0
REM  p1,p2
DATA 1,2
DATA 2,3
DATA 3,4
DATA 4,1
DATA 1,5
DATA 5,6
DATA 6,7
DATA 7,8
DATA 8,5
DATA 4,8
DATA 3,7
DATA 2,6
DATA 255,0,1
PyramidenDaten:
DATA -32, 25,-20
DATA  32, 25,-20
DATA  32, 25, 20
```

```
DATA -32, 25, 20
DATA   0, 65,  0
DATA 255,0,0
DATA 1,2
DATA 2,3
DATA 3,4
DATA 4,1
DATA 5,1
DATA 5,2
DATA 5,3
DATA 5,4
DATA 255,0,0
```

Variablenfelder

B	Bildschirmkoordinaten
D	Differenzen der Koordinatenumrechnung
F	Koordinaten der beiden Fluchtpunkte
ClickArt	Kennung, hier Schieberegler
ClickTab	Koordinaten der Schieberegler
ClickWrt	Wert eines Reglers
P	Raumkoordinaten der Punkte
V	Verbindungsvorschrift

Variablen

ClickAnz	Anzahl der definierten Klickfelder
Ende	DATA-Endkennung für "keine Daten mehr"
Gruen	Farbgrünwert
Grund	Basis für Verbindungen bei mehreren Objekten
PktAnz	Tatsächliche Anzahl der Punkte
Punkte	Maximale Anzahl der Punkte
Rot	Farbrotwert
Text	Textausgabe bei Schiebereglerdefinition
Verbind	Anzahl der Verbindungen
ZEbene	z-Koordinate der Bildschirmebene
Zeichen	Zeichenfarbe für "Braun"
clickmoeg	Maximale Anzahl der Clickfelder
i, j, k	Laufvariablen
lambda	Faktor für die Koordinatenumrechnung
mo	Modusangabe für Extras beim Schieberegler
mstat	Mausstatus
mx, my	Mauskoordinaten
po	Startposition des Schiebereglers
px, py, pz	Punktkoordinaten
sk	Laufvariable Skalierung

sx, sy	Koordinaten Textausgabe
txt	Laufvariable Textausgabe
v1, v2	Verbindungspunkte
x, y	Position des Schiebereglers
x1, y1	Koordinaten mit einem Fluchtpunkt
x2, y2	Koordinaten zum anderen Fluchtpunkt
yd	Ausdehnung des Schiebereglers

Programmbeschreibung

Zuerst wird die graphics-Library geöffnet, denn in ihr befinden sich die Einsprünge für wichtige Grafikroutinen! Dann wird der DATA-Zeiger auf die Würfeldaten gerichtet, und nachdem die Felder, die mit B oder V beginnen, als Integer definiert wurden, wird das Minimum für die Feldindizes auf 1 gesetzt. Die benutzten Variablen entsprechen den bisher verwendeten, allerdings sind die der Schieberegler hinzugekommen (siehe dort). Bei den Feldern sind natürlich die Schiebereglervariablen neu dabei, aber auch die bekannten Variablen haben sich etwas geändert.

Das Feld, in dem der Fluchtpunkt abgelegt war, ist jetzt mit einem zusätzlichen Index versehen. Er entspricht der Nummer des Fluchtpunktes und macht die spätere Verarbeitung wesentlich einfacher. Wie Sie sehen, liegen die beiden Punkte 40 Einheiten auseinander. Dieser Wert hat sich bei den gegebenen Entfernungen zwischen Fluchtpunkt und Projektionsfläche als ideal erwiesen. Aber Sie können natürlich noch alles verändern!

Bei der Ausgabe des Fluchtpunktes mußte ein neuer Weg beschritten werden. Einmal ist ein weiterer hinzugekommen, der berücksichtigt werden will. Außerdem müssen später noch, falls eine Veränderung erfolgt, die neuen Werte ausgegeben werden. Deshalb wurden diese in ein Unterprogramm gelegt. Neu sind auch die Farbeinstellungen. Alle vier Farben werden gebraucht. Der Hintergrund kann bei einer zu krassen Einstellung den 3D-Effekt behindern, deshalb wurde auch er berücksichtigt. Die anderen drei Farben sind vorher schon erklärt worden.

Als nächstes folgen die Definitionen der Schieberegler. Die Werte der ersten drei Regler stehen für die Farben, die beiden anderen Schieber ermöglichen in einem kleinen Bereich das Einstellen der Fluchtpunkte auf ihrer Horizontalen. Danach arbeiten die alten Punkt- und Verbindungsleseroutinen wieder ganz normal. Nur beim Berechnen des Bildes hat sich wieder etwas verändert. Die Schleife wird von einer weiteren umklammert, die beide Fluchtpunkte durchzählt. Dieser Zähler ist auch bei den Bildschirmkoordinaten als Index hinzugekommen.

Vor der Bildschirmausgabe wird erst der Bereich gelöscht, in dem das Objekt erscheinen könnte. Dann werden für beide Fluchtpunkte die Bilder in den entsprechenden Farben ausgegeben. Wenn das Gitternetz des zweiten Punktes gezeichnet wird, setzt das Programm zusätzlich noch einen neuen Zeichenmodus. Aus der Tabelle im Kapitel 4 können Sie ersehen, daß damit alle Modi gleichzeitig aktiviert werden. So wird beim Zeichnen mit der zweiten Farbe die rote Linie jetzt mit brauner Farbe überschrieben. Am Ende der Schleife wird der Zeichenmodus wieder auf den Normalzustand gesetzt und die Zeichenfarbe wird auf 1 eingestellt.

Dann wird ein Mouse- und ein Timer-Interrupt eingeschaltet. Der erste dient zur Abfrage der Schieberegler, der zweite setzt die Farben neu, wenn sie verändert wurden. In der Warteschleife überprüft das Programm, ob jemand den Fluchtpunkt eins oder zwei verändert hat. Wenn ja, so wird der Wert übertragen, und das Bild wird neu berechnet. Ansonsten wird auf einen Tastendruck gewartet. Ist dieser erfolgt, schaltet das Programm alle Objekte, Schieberegler, die Maus- und die Timer-Abfrage aus und gibt dann alle eingestellten Werte auf dem Bildschirm aus.

3.5 Die Zeichensätze des Amiga

Erwähnt haben wir sie schon, diese Zeichensätze, und damit vielleicht einigen Lesern bereits den Mund wässrig gemacht. Ja, es funktioniert. Auch in AmigaBASIC-Programmen können ver-

schiedene Zeichensätze gleichzeitig miteinander verwendet werden. Normalerweise stehen dazu zwei Quellen zur Verfügung:

1. Die ROM-Zeichensätze, die von Anfang an im Amiga residieren.
2. Die Diskettenzeichensätze, die im Verzeichnis "Fonts" (engl. = Zeichensätze) auf der Workbench-Diskette gelagert sind.

Das folgende Programm "beschert" Ihnen den neuen SUB-Befehl "Zeichensatz", mit dem Sie sowohl auf die ROM- als auch auf die RAM-Zeichensätze zugreifen können. Sein Aufruf sieht so aus:

```
Zeichensatz "Name", höhe%
```

Um zu erfahren, welche Zeichensätze unter welchen Namen auf der Workbench-Diskette gespeichert sind, empfehlt sich ein dezentes Nachschauen, zum Beispiel so:

```
FILES "SYS:fonts"
```

Neben diesen Zeichensätzen können Sie den ROM-Zeichensatz "topaz" in den Höhen 8 und 9 benutzen. Wenn Sie mit dem ROM-Zeichensatz arbeiten wollen, ist es von größter Wichtigkeit, den Namen topaz in Kleinbuchstaben anzugeben, denn die Funktion OpenFont() ist bezüglich Groß-/Kleinschreibung sehr pingelig und würde unter dem Namen "Topaz" oder "TOPAZ" nicht den ROM-Zeichensatz, sondern allenfalls den 11 Punkt hohen Disk-Zeichensatz "Topaz" laden. Hier das Listing:

```
'###################################
'#                                 #
'# Programm: Zeichensaetze         #
'# Autor:    tob                   #
'# Datum:    12.8.87               #
'# Version:  1.0                   #
'#                                 #
'###################################
DECLARE FUNCTION OpenDiskFont& LIBRARY
DECLARE FUNCTION OpenFont& LIBRARY
LIBRARY "diskfont.library"
LIBRARY "graphics.library"
demo:       '* demonstriert einige Zeichensaetze
```

```
            LOCATE 4,1
            Zeichensatz "Sapphire", 19
            PRINT "Dies ist Sapphire 19 Punkt!"
            Zeichensatz "Diamond", 20
            PRINT "...ein anderer Zeichensatz..."
            Zeichensatz "Garnet", 16
            PRINT "...und wieder einer! Amiga kennt noch mehr!"
            Zeichensatz "ruby", 12
            PRINT "Aber dies soll erst einmal genuegen!"
            Zeichensatz "topaz", 8
            LIBRARY CLOSE
            END
SUB Zeichensatz (welcher$, hoehe%) STATIC
    f.old&    = PEEKL(WINDOW(8)) + 52
    f.pref%   = 0
    welcher0$ = welcher$ + ".font" + CHR$(0)
    tAttr&(0) = SADD(welcher0$)
    tAttr&(1) = hoehe%*2^16 + f.pref%
    f.neu&    = OpenFont&(VARPTR (tAttr&(0)))
    f.check%  = PEEKW (WINDOW(8) + 60)
    IF f.neu& = 0 THEN
      f.neu&  = OpenDiskFont&(VARPTR (tAttr&(0)))
    ELSEIF f.check% <> hoehe% THEN
      CALL CloseFont(f.neu&)
      f.neu&  = OpenDiskFont&(VARPTR (tAttr&(0)))
    END IF
    IF f.neu& <> 0 THEN
      CALL CloseFont(f.old&)
      CALL SetFont(WINDOW(8), f.neu&)
    ELSEIF UCASE$(welcher$) = "UNDO" THEN
      CALL CloseFont(f.old&)
      CALL SetFont(original&)
    ELSE
      BEEP
    END IF
END SUB
```

Variablen

welcher$	Name des Zeichensatzes
welcher0$	wie welcher$, jedoch mit CHR$(0) abgeschlossen
hoehe%	Höhe des Zeichensatzes in Punkten
f.old&	Adresse des bisher aktiven Zeichensatzes
f.pref%	Preference-Bits; hier=0
tAttr&()	TextAttr-Struktur; Variablenfeld wird als Speicher verwendet
f.neu&	Adresse des neu geöffneten Zeichensatzes
f.check%	tatsächliche Höhe des neuen Zeichensatzes

Programmbeschreibung

Um einen Zeichensatz öffnen zu können, muß zunächst eine TextAttr-Struktur ausgefüllt werden. Diese ist in dem Variablenfeld tAttr& gespeichert. Mit der Adresse auf den Anfang dieses Feldes (VARPTR) wird die Grafik-Routine OpenFont() aufgerufen. Diese versucht, einen Zeichensatz zu finden, der den Angaben in der TextAttr-Struktur am nächsten kommt. Normalerweise gibt es im ROM lediglich den Zeichensatz "topaz" in den Höhen 8 und 9, aber wenn andere Zeichensätze noch nicht geschlossen worden sind, können auch diese mittels OpenFont() aktiviert werden.

Da OpenFont() flexibel genug ist, einen den Angaben ähnlichen Zeichensatz zu laden, wenn der gewünschte nicht zu finden ist, kann nicht immer davon ausgegangen werden, daß der von OpenFont() gefundene Zeichensatz auch wirklich der richtige ist. Deshalb wird in check% die Höhe des gefundenen Zeichensatzes mit der geforderten Höhe in hoehe% verglichen. Sind die Werte ungleich, wird der fälschlicherweise geöffnete Zeichensatz geschlossen, und OpenDiskFont() startet eine zweite Suchaktion, diesmal auf der Diskette.

Ist auf dem einen oder anderen Wege ein Zeichensatz gefunden worden (f.old& <> 0), wird mittels CloseFont() der bisher aktive Zeichensatz geschlossen und der neu geöffnete via SetFont() aktiviert. Andernfalls ertönt ein kurzer Warnton, und der alte Zeichensatz bleibt aktiviert.

Weitergehende Informationen zum Thema "Zeichensätze", insbesondere die Programmierung gänzlich eigener Zeichen, würde den Rahmen dieses auf kurze Tips und Tricks ausgelegten Buches sprengen. Weitere Informationen können Sie aber einschlägiger Spezialliteratur entnehmen. Zum Beispiel geht das im DATA BECKER Verlag erschienene Buch "Amiga Supergrafik" gerade auf diese Problematik ein und bietet sowohl BASIC- als auch "C"-Programme hierzu.

3.6 SuperPrint - schneller und komfortabler

Der wohl schwächste Befehl des AmigaBASIC heißt "PRINT". Seine Ausführung ist unendlich langsam, Text kann bei langen Sätzen das Fenster verlassen, und es gibt keine Editorbefehle. Gehen wir die Mängelliste der Reihe nach durch. Die Ausführung eines durchschnittlichen PRINT-Befehls geht so langsam, daß der Anwender ihm dabei zusehen kann. Wird eine ganze Bildschirmseite voller Text ausgegeben, kann das bereits einige Sekunden dauern.

PRINT bemerkt nicht, wenn das Ende einer Bildschirmzeile erreicht ist. Lange Textstrings werden in diesen Fällen nicht in der folgenden Bildschirmzeile fortgesetzt, sondern verlassen einfach das Fenster; dem Anwender fehlt ein Stück Information. Gerade bei Fenstern mit variabler Größe ist dies ärgerlich, weil das Fenster während des Programmablaufs verkleinert werden kann. Selbst in der Länge abgestimmte Textausgaben funktionieren dann nicht mehr.

Schließlich ist PRINT nur in der Lage, Text auszugeben. PRINT kann keine Editorbefehle ausführen, wie zum Beispiel "Bildschirm löschen", "Cursor hoch", "Insert Zeile" etc. Da PRINT einer der meistgenutzten Befehle des AmigaBASIC ist, wollen wir Ihnen nun eine Lösung all dieser Probleme präsentieren. Die Lösung ist überraschend simpel: Wir aktivieren das systeminterne "Console Device". Diese Systemkomponente dient der Ein- und Ausgabe von Text. Einmal aktiviert, übernimmt das Console Device selbständig all die Aufgaben, die BASICs PRINT nicht schafft: Text wird blitzschnell ausgegeben, an die jeweilige Breite des Fensters angepaßt, und als Clou können sogar eine Vielzahl von Editor-Kommandos in den Text integriert werden!

Leider ist es nicht ganz so leicht, das Console Device für eigene Zwecke einzusetzen, denn dazu muß es wie ein I/O-Gerät behandelt werden. Eine Vielzahl von Exec-Funktionen sind dazu nötig. Wenn diese Arbeit aber erst einmal erledigt ist, steht Ihnen ein PRINT-Befehl einer höheren Dimension zur Verfügung.

Mit seiner Hilfe werden Ihre Programme nicht nur schneller, viele Anwendungen werden sich mit den neuen Editor-Sequenzen wesentlich leichter programmieren lassen. Das folgende Programm besteht im Wesentlichen aus den SUBs "CreatePort", "RemovePort", "CreateStdIO", "RemoveStdIO", "OpenConsole", "CloseConsole", "SystemEin", "SystemAus" sowie "ConPrint". Glücklicherweise brauchen Sie sich nur mit "ConPrint" auseinanderzusetzen. Hier zunächst das Programm:

```
'##############################
'#                            #
'# Programm: Console Device   #
'# Autor:    tob              #
'# Datum:    4. 8. 87         #
'# Version:  1.0              #
'#                            #
'##############################
DECLARE FUNCTION OpenDevice% LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION AllocSignal% LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
DECLARE FUNCTION DoIO& LIBRARY
LIBRARY "exec.library"
init:    '* Kontroll-Sequenzen definieren
         C1$ = CHR$(155)  'Control Sequence Introducer
         C2$ = CHR$(8)    'Backspace
         C3$ = CHR$(10)   'Line Feed
         C4$ = CHR$(11)   'VTab
         C5$ = CHR$(12)   'Form Feed
         C6$ = CHR$(13)   'CR
         C7$ = CHR$(14)   'SHIFT IN
         C8$ = CHR$(15)   'SHIFT OUT
         C9$ = CHR$(155) + "1E" 'RETURN
demo:    '* Demonstration
         ConPrint C1$+"20CGuten Morgen!"+C9$
         ConPrint "Es war einmal ein ganz normaler Tag,";
         ConPrint " an dem wir uns auf den Weg zur Scheune begaben",
         ConPrint " und daraufhin einen grossen Baer sahen!"
SystemAus
SUB ConPrint (text$) STATIC
    SHARED c.io&
    IF c.io& = 0 THEN : SystemEin
    POKEL c.io& + 36, LEN(text$)
    POKEL c.io& + 40, SADD(text$)
    e& = DoIO&(c.io&)
END SUB
SUB SystemAus STATIC
    SHARED c.io&
    CloseConsole c.io&
```

```
END SUB
SUB SystemEin STATIC
    SHARED c.io&, c.c$
    OpenConsole c.io&
    POKEW c.io& + 28, 3
END SUB
SUB OpenConsole (result&) STATIC
    CreatePort "basic.con", 0, c.port&
    IF c.port& = 0 THEN ERROR 255
    CreateStdIO c.port&, c.io&
    POKEL c.io& + 36, 124
    POKEL c.io& + 40, WINDOW(7)
    dev$     = "console.device" + CHR$(0)
    c.error% = OpenDevice%(SADD(dev$), 0, c.io&, 0)
    IF c.error% <> 0 THEN ERROR 255
    result& = c.io&
END SUB
SUB CloseConsole (io&) STATIC
    port& = PEEKL (io& + 14)
    CALL CloseDevice(io&)
    RemovePort  port&
    RemoveStdIO io&
END SUB
SUB CreateStdIO (port&, result&) STATIC
    opt&    = 2^16
    result& = AllocMem&(48, opt&)
    IF result& = 0 THEN ERROR 7
    POKE  result& +  8, 5
    POKEL result& + 14, port&
    POKEW result& + 18, 50
END SUB
SUB RemoveStdIO (io&) STATIC
    IF io& <> 0 THEN
      CALL FreeMem(io&, 48)
    ELSE
      ERROR 255
    END IF
END SUB
SUB CreatePort (port$, pri%, result&) STATIC
    opt&  = 2^16
    byte& = 38 + LEN(port$)
    port& = AllocMem&(byte&, opt&)
    IF port& = 0 THEN ERROR 7
    POKEW port&, byte&
    port& = port& + 2
    sigBit% = AllocSignal%(-1)
    IF sigBit% = -1 THEN
      CALL FreeMem(port&,byte&)
      ERROR 7
    END IF
    sigTask& = FindTask&(0)
    POKE  port& + 8 , 4
    POKE  port& + 9 , pri%
```

```
    POKEL port& + 10, port& + 34
    POKE  port& + 15, sigBit%
    POKEL port& + 16, sigTask&
    POKEL port& + 20, port& + 24
    POKEL port& + 28, port& + 20
    FOR loop% = 1 TO LEN(port$)
      char%   = ASC(MID$(port$, loop%, 1))
      POKE port& + 33 + loop%, char%
    NEXT loop%
    CALL AddPort(port&)
    result& = port&
END SUB
SUB RemovePort (port&) STATIC
    byte&   = PEEKW(port& - 2)
    sigBit% = PEEK (port& + 15)
    CALL RemPort(port&)
    CALL FreeSignal(sigBit%)
    CALL FreeMem(port&-2, byte&)
END SUB
```

Wie Sie sehen, funktioniert das neue ConPrint fast genauso wie das herkömmliche PRINT:

```
ConPrint "auszugebender Text"
```

jedoch mit einem wesentlichen Geschwindigkeitsunterschied. Das ist aber noch nicht alles: Lange Sätze werden bereits auf die Breite des Fensters zugeschnitten. Sind Sie länger als das Fenster breit ist, werden sie in der nächsten Zeile fortgesetzt. Außerdem werden Ihnen sicherlich die Editorsequenzen C1$ bis C9$ aufgefallen sein:

C1$	CSI (Control Sequence Introducer)
C2$	Backspace, ein Zeichen nach links
C3$	Line Feed, eine Zeile nach unten
C4$	VTab, eine Zeile nach oben
C5$	Form Feed, Bildschirm löschen
C6$	CR, zurück an den Anfang der augenblicklichen Zeile
C7$	SHIFT IN, Großschrift
C8$	SHIFT OUT, Normalschrift
C9$	RETURN, schließt eine Zeile ab

Dies sind die einfachen Editor-Sequenzen. Sie fügen sie in den laufenden Text durch "+"-Zeichen ein. Zum Beispiel so:

```
ConPrint "Guten Tag, liebe Welt!"+C9$
```

Das Console Device kann aber noch wesentlich mehr. Die folgenden Editor-Sequenzen beginnen immer mit dem Control Sequence Introducer (CSI), den Sie in C1$ finden. Hier diese Editor-Sequenzen:

C1$ +	Bedeutung
"[n]@"	Füge [n] Zeichen in diese Zeile ein
"[n]A"	Cursor [n] Zeilen hoch
"[n]B"	Cursor [n] Zeilen runter
"[n]C"	Cursor [n] Zeichen nach rechts
"[n]D"	Cursor [n] Zeichen nach links
"[n]E"	Cursor [n] Zeilen runter + an den Anfang
"[n]F"	Cursor [n] Zeilen hoch + an den Anfang
"[n];[n]H"	Cursor nach Zeile [n], Spalte [n]
"J"	ab Cursor Bildschirm löschen
"K"	ab Cursor Zeile löschen
"L"	Füge Zeile ein
"M"	Lösche Zeile
"[n]P"	Lösche Zeichen ab Cursor zur Rechten
"[n]S"	Scroll [n] Zeilen hoch
"[n]T"	Scroll [n] Zeilen runter
"20h"	Set Mode
"20l"	Reset Mode
"[n];[n];[n]m"	Grafik-Modus Stil: 0 = normal 1 = fett 3 = kursiv 4 = unterstrichen 7 = invers Vordergrundfarbe: 30 - 37 Hintergrundfarbe: 40 - 47
"[n]t"	Höhe des Fensters in Rasterzeilen
"[n]u"	Länge einer Zeile in Pixel
"[n]x"	[n] Zeichen einrücken
"[n]y"	[n] Zeilen am oberen Rand freihalten

3.6.1 Das Console Device unter GFA

Eine erfreuliche Nachricht für die Freunde des GFA-BASIC: Das soeben mehr oder weniger komplex implementierte Console-Device für das AmigaBASIC existiert bei GFA bereits, denn der normale GFA-Befehl PRINT entspricht zu 100% dem neuen AmigaBASIC-Befehl SuperPrint. Sämtliche Steuercodes lassen sich also unter GFA in Zusammenhang mit ganz normalen PRINT-Anweisungen umsetzen. Die Steuercodes finden sich auf den vorangegangenen Seiten. Hier ein kleines Beispiel:

```
PRINT CHR$(155)+"3;4;31;42mkursiv, unterstrichen, farbig!"
```

Unter optimaler Ausnutzung aller vorhandenen Steuercodes lassen sich so komplexere Anwendungen wie Textverarbeitungen oder individuelle Eingaberoutinen programmtechnisch leicht umsetzen. Zu beachten ist, daß die genannten Steuercodes selbstverständlich nur auf dem Bildschirm als Ausgabegerät die gewünschte Wirkung zeigen und auf dem Drucker keine Wirkung haben.

4. Professionelle Gestaltung eigener Anwenderprogramme

In diesem Kapitel werden wir Ihnen zeigen, wie Sie als Programmierer alle Möglichkeiten ausnutzen können, um Ihre Programme wesentlich anwenderfreundlicher zu gestalten. Die Gestaltung eines Anwenderprogramms ist für die spätere Bedienung und besonders für den Bediener sehr wichtig. Gerade bei solch einem Computer wie dem Amiga ist es wichtig, daß alle Eingaben, Auswahlen und Einstellungen besonders bedienungsfreundlich gestaltet sind.

Oft wird dies über Piktogramme oder andere Bedienelemente erledigt. Auf jeden Fall sollte die Bedienung größtenteils mit der Maus möglich sein, schließlich ist sie nicht nur dazu da, das Programm zu starten und dann in der Ecke zu liegen. Deshalb sehen Sie jetzt, wie man diese Funktionen einfach programmiert und auch in jedes eigene Programm einbauen kann.

4.1 Alternativen zu PullDown-Menüs

Weil mit den Menüs nicht alles zu machen ist, wird jetzt nach Alternativen gesucht. Bloß: Was sind unsere Alternativen? "Warum in die Ferne schweifen, siehe, das Gute liegt doch so nahe." Legen Sie Ihre Workbench-Diskette in Ihr Laufwerk, und öffnen Sie das Hauptinhaltsverzeichnis. Hier findet man im Normalfall das Programm "Preferences", mit dem alle wichtigen Voreinstellungen gemacht werden können. Wenn Sie das Programm aktivieren, werden Sie alle Elemente einer benutzerfreundlichen Bedienung vor sich sehen.

Da sind einmal die Schieberegler, mit denen die Farben und die Zeit eines Doppelklicks eingestellt werden können. Außerdem findet man Wahltabellen wie z.B. für die Einstellung der Zeichen pro Zeile. Auch im Teil "Change Printer" werden Tabellen

en masse benutzt. Weiterhin findet man überall Kästen, die beim Anklicken sofort eine Aktion auslösen, z.B. "Save", "Use" und "Cancel" auf dem Hauptbildschirm.

Alle diese eben genannten Bedienelemente wollen wir im folgenden programmieren. Dazu benötigen wir eine Ausgabegrundlage auf dem Amiga. Dies ist im Normalfall ein Window, das sich über BASIC sehr leicht öffnen läßt. Jedoch gehen wir einen etwas anderen Weg, bei dem uns mehr Möglichkeiten offenstehen und über den wir keine Eingriffe in das AmigaBASIC selbst machen müssen.

Hier ist deshalb das erste Programm, das nichts weiter tut, als ein Window auf der Workbench-Screen zu öffnen. Dafür benutzt es die Intuition.Library, die wir auch weiterhin für alle anderen Modifikationen nutzen werden:

```
'*********************************
'*                               *
'* Window über Intuition öffnen *
'* ----------------------------- *
'*                               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  : 22. Mai '88          *
'* Grueße : Denis "Angle"        *
'* Version: 1.1                  *
'* Betriebssystem: V1.2 & V1.3   *
'*                               *
'*********************************
  OPTION BASE 1
  DEFLNG a-z

  LIBRARY ":bmaps/exec.library"
  DECLARE FUNCTION AllocMem LIBRARY
  LIBRARY ":bmaps/intuition.library"
  DECLARE FUNCTION OpenWindow LIBRARY

  MList      = 0&
HauptProgramm:
  GOSUB OpenAll

  ' Hauptteil
  FOR i = 1 TO 10000 : NEXT i

  GOSUB CloseAll

END
```

```
OpenAll:
  Titel$  = "Mein erstes BASIC-Window"
  WinDef NWindow, 100, 100, 460, 150, 32+64+512&, 15&+4096&, 0&, Titel$
  WinBase = OpenWindow(NWindow)
  IF WinBase = 0 THEN ERROR 7
RETURN

CloseAll:
  CloseWindow(WinBase)
  CALL UnDef
RETURN
'-----------------------------------------
SUB DefChip(Buffer, Size) STATIC
 SHARED MList
  Size=Size+8
  Buffer=AllocMem(Size,65538&)
  IF Buffer>0 THEN
    POKEL Buffer,MList
    POKEL Buffer+4,Size
    MList=Buffer
    Buffer=Buffer+8
  ELSE
    ERROR 7
  END IF
END SUB

SUB UnDef STATIC
 SHARED MList
undef.loop:
  IF MList>0 THEN
    Adresse = PEEKL(MList)
    Groesse = PEEKL(MList+4)
    FreeMem MList, Groesse
    MList   = Adresse
    GOTO undef.loop
  END IF
END SUB

SUB WinDef(bs, x%, y%, b%, h%, IDCMP, f, gad, T$) STATIC
  Size = 48+LEN(T$)+1
  DefChip bs,Size
  POKEW bs   ,x%       ' LeftEdge
  POKEW bs+ 2,y%       ' TopEdge
  POKEW bs+ 4,b%       ' Width
  POKEW bs+ 6,h%       ' Height
  POKEW bs+ 8,65535&   ' Detail- BlockPen
  POKEL bs+10,IDCMP    ' IDCMPFlags
  POKEL bs+14,f        ' Flags
  POKEL bs+18,gad      ' FirstGadget
  POKEL bs+26,bs+48    ' Title
```

```
  POKEW bs+46,1        ' ScreenType
  FOR i%=1 TO LEN(T$)
     POKE bs+47+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB
```

Programmbeschreibung

Die wichtigsten Elemente des Programms finden wir am Ende. Hier sind drei SUB-Routinen, die alle äußerst wichtige Aufgaben erfüllen. Mit DefChip() fordert man einen Speicherbereich in der gewünschten Größe an. Die wird dann über AllocMem(), die Betriebssystemfunktion der Exec.Library, besorgt. Gleichzeitig verwaltet die Routine in den ersten 8 Bytes zwei Werte, mit deren Hilfe eine Speicherliste aufgebaut wird. Dadurch hat UnDef() leichtere Arbeit beim Freigeben des Speichers. Dazu geht es mit Hilfe des Zeigers MList alle Bereiche durch und gibt sie nacheinander frei.

Die beiden Unterroutinen sind aber nur Hilfswerkzeug. Die eigentlich wichtige finden wir erst danach. WinDef() erstellt unter Hinzunahme alle geforderten Daten eine sog. NewWindow-Struktur. Diese wird von Intuition benötigt, um ein neues Window zu öffnen. WinDef() legt die bekannten Daten aber nur in einen neuen Speicherbereich. Alles weitere erledigt eine Teilroutine des Hauptprogramms.

Nachdem wir jetzt die Aufgaben der Unterroutinen kennen, wollen wir uns das Hauptprogramm ansehen. Hier werden zuerst die beiden Libraries Intuition und Exec geöffnet. Aus beiden benötigen wir einige Funktionen. Im Hauptteil wird dann das Unterprogramm OpenAll angesprungen. Hier wird die Definition einer NewWindow-Struktur über WinDef() aufgerufen und dann wird diese gerade erstellte Struktur mittels OpenWindow() an Intuition weitergeleitet. Wenn ein neues Window geöffnet werden konnte, erhält man von dieser Funktion einen Zeiger auf die Window-Struktur zurück. Diese neue Struktur enthält alle lebensnotwendigen Daten für ein Window.

Nach der Rückkehr in die Hauptschleife wird erst einmal in der leeren FOR-TO-NEXT-Schleife einige Zeit gewartet, damit Sie

beim Testen auch das Window einen Augenblick betrachten können. Dann springt der Interpreter in ein weiteres Unterprogramm. Es trägt den bezeichnenden Titel CloseAll und schließt alles bisher Geöffnete. Es erledigt sozusagen die Aufräumarbeiten. Bei genauerer Betrachtung finden wir dort wirklich nur den Aufruf zum Schließen des Windows und zur Freigabe des belegten Speichers.

Damit haben wir die Grundlage für unser professionell zu gestaltendes Programm. In dieses Window können wir jetzt nach und nach die bedienfreundlichen Elemente, wie in einem Baukastensystem einsetzen.

4.1.1 Die erste Select-Box

Das erste bedienfreundliche Element, das wir bei Preferences feststellen, sind die Klick-Felder. Mit einem Druck auf die linke Maustaste kann man bestätigen, daß die Einstellungen so richtig sind (OK, Use), oder man hat es sich doch anders überlegt und möchte alles wieder rückgängig machen. Dazu betätigt man einfach "Cancel".

Alles dies sind grafisch klar erkennbare Felder mit einer eindeutig definierten Position - somit haben sie einen hohen Wiedererkennungswert - die durch einen Klick sofort eine Aktion auslösen. Dies können wir auch in BASIC programmieren. Dazu nutzen wir die Anwendung von Gadgets über Intuition, die wir ganz leicht in unser Window einbinden können. Mit der folgenden SUB-Routine ist sofort ein neues Klickfeld definiert:

```
SUB GadgetDef(bs, nx, x%, y%, b%, h%, f%, a%, T%, i, txt, si, n%) STATIC
        DefChip bs,44&      ' Gadget-Struktur Länge
        POKEL bs   ,nx      '*NextGadget
        POKEW bs+ 4,x%      ' LeftEdge
        POKEW bs+ 6,y%      ' TopEdge
        POKEW bs+ 8,b%      ' Width
        POKEW bs+10,h%      ' Height
        POKEW bs+12,f%      ' Flags
        POKEW bs+14,a%      ' Activation
        POKEW bs+16,T%      ' GadgetType
        POKEL bs+18,i       ' GadgetRender
```

```
    POKEL bs+26,txt      '*GadgetText
    POKEL bs+34,si       ' SpecialInfo
    POKEW bs+38,n%       ' GadgetID
END SUB
```

Die Routine trägt wieder eine Reihe von Werten in einen Speicherbereich ein. Diesen Vorgang wollen wir uns genauer ansehen. In bs finden wir die Basisadresse unseres Speicherbereichs, es handelt sich hier um den Rückgabewert der Speicherallozierungsroutine. Nx bezeichnet die Startadresse des nächsten Klickbereiches und dient der Verkettung mehrerer. Wir brauchen diesen Wert später für mehrere Klickbereiche. Mit x%, y%, b% und h% definieren wir die Ausmaße des Gadgets. Position und Ausdehnung in Breite und Höhe können bestimmt werden.

Mit f% und a% werden zwei Flags definiert, auf die wir später noch zurückkommen werden. T% kennzeichnet den Typ, der bei unserem ersten Beispiel auf 1 gesetzt wird. Im weiteren Verlauf werden wir einen anderen Typ kennenlernen. Mit i und txt werden zusätzliche grafische Informationen untergebracht. So kann man unter i ein Bild oder einen Rand definieren, der zusätzlich gezeichnet wird, und unter txt einen beschreibenden Text. Si wird bei anderen Gadget-Typen benutzt, die mit den Informationen dieser Struktur nicht auskommen. Als letztes legen wir einen Identifikationswert in n%. Es handelt sich hierbei um eine Nummer, mit deren Hilfe wir später bei mehreren Gadgets eine Unterscheidung treffen können.

Nach dieser ausführlichen Beschreibung alle Werte kommen wir zu unserem ersten Beispiel. Setzen Sie dazu diese Sub-Routine in das Beispielprogramm vom Anfang ein, und ergänzen Sie in dem OpenAll-Unterprogramm folgende Zeile:

```
GadgetDef Gadget, 0&, 50, 50, 20, 10, 0, 1, 1, 0&, 0&, 0&, 1
```

Damit beauftragen wir GadgetDef mit der Definition eines neuen Gadgets. Dieses hat die folgenden Eigenschaften:

- Die Adresse der Gadget-Struktur werden wir nach Abschluß der Routine in Gadget finden.

- Es werden keine weiteren Gadgets eingebunden.
- Die Position liegt bei 50,50.
- Das Gadget ist 20 Pixel breit und 10 Pixel hoch.
- Es handelt sich um ein Gadget, das beim Anklicken erst reagiert, wenn auch im Klickbereich losgelassen wird.
- Das Gadget ist vom Typ Boolean und kann nur aktiviert werden.
- Es gibt keine Grafik und keinen Text.
- Es wird keine Zusatz-Struktur benötigt.
- Das Probe-Gadget hat die Nummer 1.

Diese gerade definierte Gadget-Struktur binden wir in das neue Window ein:

```
WinDef NWindow, 100, 100, 460, 150, 32+64+512&, 15&+4096&, Gadget,Titel$
```

So! Nun könnten Sie zwar das Programm starten, doch gebrauchen können wir es noch nicht. Das Programm ist nämlich noch nicht in der Lage, die Aktivierung des Gadgets abzufragen. Wenn Sie jetzt RUN eingeben, wird zwar das Fenster erscheinen und nach einigem Herumklicken findet man auch den Bereich, in dem das Gadget definiert ist, doch bisher können wir keine Reaktion erwarten. Dazu müssen wir erst noch eine weitere Unterroutine schreiben, die vom Hauptprogramm angesprungen wird, wenn eine Nachricht vorliegt.

Hier ist deshalb zuerst die neue Hauptprogrammabfrage, die darauf wartet, daß am Nachrichtenkanal des neuen Windows eine neue Nachricht anliegt. Dann verzweigt sie in ein ebenfalls neues Unterprogramm zur Auswertung:

```
  Lauf       = 1
HauptProgramm:
  GOSUB OpenAll
  ' Hauptteil
  MainLoop:
    IF Lauf = 1 THEN
      IntuiMsg = GetMsg(UserPort)
      IF IntuiMsg > 0 THEN GOSUB IntuitionMsg
```

```
      GOTO MainLoop
    END IF

  GOSUB CloseAll

END
```

Mit der Funktion GetMsg() aus der Exec.Library ist es möglich, den MessagePort eines Windows auf eine Nachricht hin zu testen. Dieser MessagePort liegt in der Window-Struktur, die wir von OpenWindow() zurückgegeben bekommen haben. Dort muß die neue Variable initialisiert werden:

```
UserPort = PEEKL(WinBase+86)
```

Nun können wir uns endlich die Unterroutine ansehen, die die Auswertung einer ankommenden Nachricht übernimmt. Sie unterscheidet zuerst den Typ der Nachricht, da wir hier verschiedene Auswertungsverfahren anwenden müssen. Die erste Nachricht, die wir dann wirklich auswerten können, ist die CLOSEWINDOW-Message. Bei der Betätigung eines Gadgets können wir jetzt nur die Nummer ausgeben, die wir ja in die Struktur geschrieben haben.

```
IntuitionMsg:
  MsgTyp    = PEEKL(IntuiMsg+20)
  Item      = PEEKL(IntuiMsg+28)
  GadgetNr% = PEEK(Item+39)
  CALL ReplyMsg(IntuiMsg)

  IF (MsgTyp = GADGETDOWN) THEN
     'sofortige Aktivierung
     PRINT "DOWN Gadget-Nr.:";GadgetNr%
  END IF

  IF (MsgTyp = GADGETUP) THEN
     'relverify Modus
     PRINT "UP   Gadget-Nr.:";GadgetNr%
  END IF

  IF (MsgTyp = CLOSEW) THEN
     'System-Gadget Window schließen
     PRINT "CLOSEWINDOW"
     Lauf = 0
  END IF
RETURN
```

Wenn Sie aus den bisherigen Programmteilen das Programm zusammensetzen, müßte nach dem Start wieder das Window erscheinen. Nun kommt unser erster Test. Klicken Sie dazu zuerst im linken oberen Viertel des Windows. Irgendwo wird das neue Gadget erkennbar sein. Nach jedem Loslassen erscheint in dem Ausgabe-Window des AmigaBASIC die Gadget-Nummer. Hoffentlich!

Beim zweiten Test klicken Sie bitte das Close-Gadget des Windows an. Zuerst erscheint der Text im Ausgabe-Window, dann wird das Window geschlossen. Damit haben wir die Vorarbeit geleistet!

4.1.2 Grafik bringt die Erkenntnis

Eine sehr wichtige Eigenschaft fehlt unserem Gadget bisher noch: Kein Mensch kann es erkennen! Diesen kleinen Mißstand wollen wir deshalb sofort beheben. Bei der Besprechung der SUB-Routine GadgetDef haben wir schon die Variablen txt und i kennengelernt. Dabei handelt es sich um die elementaren Grafikelemente Intuitions. Mit txt können wir eine IntuiText-Struktur in das Gadget einbinden und mit i eine Grafik oder Linien. Alle drei Möglichkeiten haben ihre Bedeutung und sollen deshalb genutzt werden.

Sehen wir uns dazu zuerst den Text an, mit dem die meisten Aufgaben am einfachsten gelöst werden können. Dazu benötigen wir eine eigens von Intuition benutzte IntuiText-Struktur. Sie enthält Einstellungen zu Position, Farbe, Zeichensatz und -art. Und natürlich auch den Text. Hier ist das SUB-Programm, das einen Speicherbereich mit den geforderten Daten initialisiert:

```
SUB IntuiText(bs, c1%, x%, y%, T$, nx) STATIC
  Size=20+LEN(T$)+1   ' Strukturlänge + Textlänge + Nullbyte
  DefChip bs,Size
  POKE  bs   ,c1%     ' FrontPen
  POKE  bs+ 2,1       ' DrawMode
  POKEW bs+ 4,x%      ' LeftEdge
  POKEW bs+ 6,y%      ' TopEdge
  POKEL bs+12,bs+20   ' IText
```

```
    POKEL bs+16,nx        ' NextText
    FOR i%=1 TO LEN(T$)
       POKE bs+19+i%,ASC(MID$(T$,i%,1))
    NEXT
END SUB
```

Um die Angelegenheit nicht zu kompliziert zu machen, wurden die Parameter wieder auf das Nötige gekürzt. Nach der obligatorischen Startadresse der Struktur stellen wir die Zeichenfarbe des Textes und die Position in Pixel ein. Des weiteren übergeben wir natürlich den Text selbst. Da auch hier wieder das Linken mehrerer Texte zugelassen ist, kann man als letzten Wert einen Zeiger auf eine weitere IntuiText-Struktur übergeben.

Die Routine setzt selbständig den Zeichenmodus auf JAM2. Damit ist garantiert, daß auch der Hintergrund überschrieben wird. Sie können also sicher sein, daß man den Text später lesen kann. Möchten Sie trotzdem auch die zweite Farbe beeinflussen können, ergänzen Sie ganz einfach einen Wert in der Parameterliste und POKEn sie den neuen Farbwert in bs+1! Um jetzt z.B. das Gadget mit einem Text zu versehen, kann man zuerst einen Text über das neue Unterprogramm erstellen und diesen dann in die Gadget-Definition einbinden:

```
TestTxt$ = "Test-Text"
IntuiText Text, 2, 10, 2, TestTxt$, 0&
GadgetDef Gadget, 0&, 50, 50, 90, 15, 1, 1, 1, 0&, Text, 0&, 1
```

Als nächstes wollen wir uns mit den Linien beschäftigen. Sie dienen hauptsächlich der Abgrenzung von Klickflächen und zur Unterteilung. Da auch die Ränder mit in die Gadget-Struktur eingebunden werden, brauchen wir dazu eine eigene Struktur. Es handelt sich hierbei um die Border-Struktur, die neben der Farbe und der Position auch eine Koordinatentabelle benötigt. Diese Koordinaten selbst werden im Speicher nach der Struktur abgelegt.

Für diese Border-Struktur haben wir ein etwas anderes SUB-Programm entwickelt. Es setzt nicht nur die Werte in den Speicherbereich ein, sondern es berechnet auch anhand der

Werte die Koordinaten für die Tabelle. Somit wird es ganz einfach, einen Kasten um ein Gadget zu definieren. Sehen Sie dazu hier die Funktion:

```
SUB Border(bs, x%, y%, c%, b%, h%) STATIC
  DefChip bs,48&       ' Strukturlänge + Koordinatentabelle
  POKEW bs  ,x%        ' LeftEdge
  POKEW bs+2,y%        ' TopEdge
  POKE  bs+4,c%        ' FrontPen
  POKE  bs+7,8         ' Count
  POKEL bs+8,bs+16     '*XY
  FOR i%=0 TO 1
     POKEW bs+22+i%*4,h%-1
     POKEW bs+24+i%*4,b%-1
     POKEW bs+32+i%*4,1
     POKEW bs+38+i%*4,h%-1
     POKEW bs+40+i%*4,b%-2
  NEXT
END SUB
```

Wenn wir jetzt die Routine mit den entsprechenden Werten aufrufen und die Basisadresse der Struktur wie auch bei der IntuiText-Struktur in die Definition des Gadgets einsetzen, dann wird auch der Rand mit dem Gadget kombiniert. Auch dazu möchte ich ein Beispiel geben:

```
Border Rand, 0, 0, 3, 90, 15
GadgetDef Gadget, 0&, 50, 50, 90, 15, 0, 1, 1, Rand, Text, 0&, 1
```

Das Gadget nutzt gleichzeitig zum Rand auch noch den vorher schon bestimmten Text. Diesen können Sie natürlich auch noch ergänzen. Abschließend möchte ich noch einmal das gesamte Listing zeigen, das jetzt alle bisher besprochenen Unterprogramme und die gezeigten Definitionen enthält. Sie können daran noch Fehler erkennen, die Sie vielleicht beim Zusammensetzen der Module gemacht haben.

```
'*********************************
'*                               *
'* Boolean-Gadgets über Intuition *
'* ----------------------------- *
'*                               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  : 23. Mai '88          *
'* Grueße : Uuuuwe               *
'* Version: 1.2                  *
```

```
'* Betriebssystem: V1.2 & V1.3    *
'*                                *
'**********************************
  OPTION BASE 1
  DEFLNG a-z

  LIBRARY ":bmaps/exec.library"
  DECLARE FUNCTION AllocMem LIBRARY
  DECLARE FUNCTION GetMsg LIBRARY
  LIBRARY ":bmaps/intuition.library"
  DECLARE FUNCTION OpenWindow LIBRARY

  GADGETDOWN = 32&
  GADGETUP   = 64&
  CLOSEW     = 512&

  MList      = 0&
  Lauf       = 1

HauptProgramm:
  GOSUB OpenAll
  ' Hauptteil
  MainLoop:
    IF Lauf = 1 THEN
      IntuiMsg = GetMsg(UserPort)
      IF IntuiMsg > 0 THEN GOSUB IntuitionMsg
      GOTO MainLoop
    END IF

  GOSUB CloseAll
END

OpenAll:
  Border Rand, 0, 0, 3, 90, 15
  TestTxt$ = "Test-Text"
  IntuiText Text, 2, 10, 2, TestTxt$, 0&
  GadgetDef Gadget, 0&, 50, 50, 90, 15, 0, 1, 1, Rand, Text, 0&, 1
  Titel$   = "Mein erstes BASIC-Window"
  WinDef NWindow,100,100,460,150,32+64+512&,15&+4096&,Gadget,Titel$
  WinBase  = OpenWindow(NWindow)
  IF WinBase = 0 THEN ERROR 7
  RastPort = PEEKL(WinBase+50)
  UserPort = PEEKL(WinBase+86)
RETURN

CloseAll:
  CALL CloseWindow(WinBase)
  CALL UnDef
RETURN

IntuitionMsg:
  MsgTyp    = PEEKL(IntuiMsg+20)
```

```
  Item      = PEEKL(IntuiMsg+28)
  GadgetNr% = PEEK(Item+39)
  CALL ReplyMsg(IntuiMsg)

  IF (MsgTyp = GADGETDOWN) THEN
     'sofortige Aktivierung
     PRINT "DOWN Gadget-Nr.:";GadgetNr%
  END IF

  IF (MsgTyp = GADGETUP) THEN
     'relverify Modus
     PRINT "UP   Gadget-Nr.:";GadgetNr%
  END IF

  IF (MsgTyp = CLOSEW) THEN
     'System-Gadget Window schließen
     PRINT "CLOSEWINDOW"
     Lauf = 0
  END IF
RETURN
'-----------------------------------------
SUB DefChip(Buffer,Size)STATIC
 SHARED MList
  Size=Size+8
  Buffer=AllocMem(Size,65538&)
  IF Buffer>0 THEN
    POKEL Buffer,MList
    POKEL Buffer+4,Size
    MList=Buffer
    Buffer=Buffer+8
  ELSE
    ERROR 7
  END IF
END SUB
SUB UnDef STATIC
 SHARED MList
undef.loop:
  IF MList>0 THEN
    Adresse = PEEKL(MList)
    Groesse = PEEKL(MList+4)
    FreeMem MList, Groesse
    MList   = Adresse
    GOTO undef.loop
  END IF
END SUB

SUB WinDef(bs, x%, y%, b%, h%, IDCMP, f, gad, T$) STATIC
  Size = 48+LEN(T$)+1
  DefChip bs,Size
  POKEW bs   ,x%      ' LeftEdge
  POKEW bs+ 2,y%      ' TopEdge
  POKEW bs+ 4,b%      ' Width
  POKEW bs+ 6,h%      ' Height
```

```
  POKEW bs+ 8,65535&   ' Detail- BlockPen
  POKEL bs+10,IDCMP    ' IDCMPFlags
  POKEL bs+14,f        ' Flags
  POKEL bs+18,gad      ' FirstGadget
  POKEL bs+26,bs+48    ' Title
  POKEW bs+46,1        ' ScreenType
  FOR i%=1 TO LEN(T$)
     POKE bs+47+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB

SUB GadgetDef(bs, nx, x%, y%, b%, h%, f%, a%, T%, i, txt, si, n%) STATIC
  DefChip bs,44&       ' Gadget-Struktur Länge
  POKEL bs   ,nx       '*NextGadget
  POKEW bs+ 4,x%       ' LeftEdge
  POKEW bs+ 6,y%       ' TopEdge
  POKEW bs+ 8,b%       ' Width
  POKEW bs+10,h%       ' Height
  POKEW bs+12,f%       ' Flags
  POKEW bs+14,a%       ' Activation
  POKEW bs+16,T%       ' GadgetType
  POKEL bs+18,i        ' GadgetRender
  POKEL bs+26,txt      '*GadgetText
  POKEL bs+34,si       ' SpecialInfo
  POKEW bs+38,n%       ' GadgetID
END SUB

SUB IntuiText(bs, c1%, x%, y%, T$, nx) STATIC
  Size=20+LEN(T$)+1    ' Strukturlänge + Textlänge + Nullbyte
  DefChip bs,Size
  POKE  bs   ,c1%      ' FrontPen
  POKE  bs+ 2,1        ' DrawMode
  POKEW bs+ 4,x%       ' LeftEdge
  POKEW bs+ 6,y%       ' TopEdge
  POKEL bs+12,bs+20    ' IText
  POKEL bs+16,nx       ' NextText
  FOR i%=1 TO LEN(T$)
     POKE bs+19+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB

SUB Border(bs, x%, y%, c%, b%, h%) STATIC
  DefChip bs,48&       ' Strukturlänge + Koordinatentabelle
  POKEW bs  ,x%        ' LeftEdge
  POKEW bs+2,y%        ' TopEdge
  POKE  bs+4,c%        ' FrontPen
  POKE  bs+7,8         ' Count
  POKEL bs+8,bs+16     '*XY
  FOR i%=0 TO 1
     POKEW bs+22+i%*4,h%-1
     POKEW bs+24+i%*4,b%-1
```

```
      POKEW bs+32+i%*4,1
      POKEW bs+38+i%*4,h%-1
      POKEW bs+40+i%*4,b%-2
  NEXT
END SUB
```

Nachfolgend finden Sie noch vier Gadgets, die man immer in einem Window für Einstellungen gebrauchen kann. Zu OK oder Cancel, die die Funktion entweder bestätigen oder abbrechen, sind noch Reset, alle Werte werden auf den Ausgangszustand gesetzt, und Undo gekommen. Mit Undo wird der letzte veränderte Wert wieder zurückgesetzt.

Dies ist eine sehr bedienfreundliche Funktion, die es besonders Unentschlossenen einfacher macht: Man hat schon alles eingestellt und möchte nur noch eine Veränderung rückgängig machen, weiß aber nicht, wie es eingestellt war. Dann hilft Undo! Sehen Sie dazu die Gadget-, Text- und Rand-Definitionen und im zweiten Teil die Abfrageroutinen mit den Verzweigungen für die vier Gadgets.

Die Gadget-Definitionen:

```
Border Rand, -1, -1, 1, 67, 14
IntuiText OKTxt, 1, 26, 2, "OK", 0&
IntuiText CancelTxt, 1, 10, 2, "Cancel", 0&
IntuiText ResetTxt, 1, 14, 2, "Reset", 0&
IntuiText UndoTxt, 1, 20, 2, "Undo", 0&
GadgetDef UndoGad, 0&, 380, 52, 65, 12, 0, 1, 1, Rand, UndoTxt, 0&, 1
GadgetDef ResetGad,UndoGad,380,68,65,12,0,1,1,Rand,ResetTxt,0&,2
GadgetDef OKGad, ResetGad, 380, 84, 65, 12, 0, 1, 1, Rand, OKTxt, 0&, 3
GadgetDef CancGad,OKGad,380,100,65,12,0,1,1,Rand,CancelTxt,0&,4
Titel$   = "Ein Beispiel für bedienfreundliche Gadgets"
WinDef NWindow,100,100,460,150,32+64+512&,15&+4096&,CancGad,Titel$
```

Die Abfrage-Routinen:

```
IF (MsgTyp = GADGETDOWN) THEN
   'sofortige Aktivierung
   PRINT "DOWN Gadget-Nr.:";GadgetNr%
END IF

IF (MsgTyp = GADGETUP) THEN
```

```
   'relverify Modus
   PRINT "UP   Gadget-Nr.:";GadgetNr%
   IF GadgetNr% = 1 THEN
      GOSUB UNDO   ' alten Wert eintragen
   END IF
   IF GadgetNr% = 2 THEN
      gsoub RESET  ' alle Werte wieder auf Urzustand
   END IF
   IF GadgetNr% = 3 THEN
      GOSUB OK     ' Werteeingabe beendet
   END IF
   IF GadgetNr% = 4 THEN
      GOSUB CANCEL ' Werteeingabe abgebrochen
   END IF
END IF

IF (MsgTyp = CLOSEW) THEN
   'System-Gadget Window schließen
   PRINT "CLOSEWINDOW"
   Lauf = 0
END IF
```

4.1.3 Wahltabellen

Bei der Gestaltung eines Windows mit Auswahl-Gadgets ist es nicht empfehlenswert, immer nur einzelne Gadgets zu verwenden. Man sollte vielmehr darauf achten, Gruppen zu bilden, die alle eine Auswahl aus einem bestimmten Themenkreis darstellen. Meist ist es dann so, daß nur eines der Gadget-Gruppe ausgewählt werden kann. Als Beispiel könnte die Auswahl des Zeichensatzes dienen. Nur ein Zeichensatz kann zur Zeit aktiv sein. In einer Wahltabelle wählt man den entsprechenden aus, und der vorher gewählte Zeichensatz wird inaktiv.

Vollständige Tabellen

Die Programmierung läuft nun etwas anders ab als bei den bisher gesehenen Gadgets. Zwar erstellt man die Strukturen, Grafiken und Texte genauso, wie wir es gewohnt sind, jedoch muß bei der Auswahl einiges beachtet werden! So müssen wir nach der Aktivierung des Gadgets dieses auch aktiviert lassen. Intuition hebt den Status gleich wieder auf, deshalb invertieren wir den Bereich erneut. Dann aber dauerhaft. Dazu nutzen wir die Graphics.Library, die uns mit zwei Befehlen zur Hilfe steht.

Außerdem muß unsere Auswertungsroutine in der Lage sein, ein zuvor ausgewähltes Gadget wieder auszuschalten. Alles dies werden wir uns jetzt ansehen. Dazu nehmen wir zuvor einige Gadgets als Grundlage:

```
Border Rand, -1, -1, 1, 100, 14
IntuiText To60, 1, 10, 2, "Topaz 60", 0&
IntuiText To80, 1, 10, 2, "Topaz 80", 0&
IntuiText PC60, 1, 26, 2, "PC 60", 0&
IntuiText PC80, 1, 26, 2, "PC 80", 0&
GadgetDef To60Gad, 0&, 80, 61, 98, 12, 0, 1, 1, Rand, To60, 0&, 1
GadgetDef To80Gad, To60Gad, 80, 74, 98, 12, 0, 1, 1, Rand, To80, 0&, 2
GadgetDef PC60Gad, To80Gad, 80, 87, 98, 12, 0, 1, 1, Rand, PC60, 0&, 3
GadgetDef PC80Gad, PC60Gad, 80, 100, 98, 12, 0, 1, 1, Rand, PC80, 0&, 4
Titel$   = "Font-Auswahl"
WinDef NWindow,100,100,460,150,32+64+512&,15&+4096&,PC80Gad,Titel$
```

Alle Gadgets benutzen eine Border-Struktur, die die Tabellengliederung später erreicht. Weiterhin haben wir vier Texte, die jeden einzelnen Zeichensatz kennzeichnen. Über Pointer haben wir alle Gadgets in einer Liste verbunden. Diese Liste wird mit ihrem Kopf in das Window eingepaßt.

Nach dem Programmstart haben wir eine Tabellengrafik im Window, aus der man seinen gewünschten Zeichensatz auswählen kann. Zur Kennzeichnung des gerade aktiven Zeichensatzes invertieren wir einfach den Kasten. Sehen Sie dazu die Abfrageroutine der Gadget-Nachrichten:

```
IF (MsgTyp = GADGETUP) THEN
   'relverify Modus
   PRINT "UP   Gadget-Nr.:";GadgetNr%
   IF aktiv THEN
      SetDrMd RastPort, 2
      RectFill RastPort, 80&, 13*aktiv+48&, 177&, 13*aktiv+59&
      SetDrMd RastPort, 1
   END IF
   aktiv = GadgetNr%
   SetDrMd RastPort, 2
   RectFill RastPort, 80&, 13*aktiv+48&, 177&, 13*aktiv+59&
   SetDrMd RastPort, 1
END IF
```

Die Abfrage geht davon aus, daß dem Programm über die Variablen bekannt ist, welcher der Zeichensätze gewählt wurde. Dann wird bei einer erneuten Wahl der Textkasten über die Grafikbefehle wieder in den Normalzustand gebracht.

Als nächstes wird ausgewertet, welcher neue Zeichensatz vom Benutzer ausgewählt wurde. Die Koordinaten des Kastens werden berechnet, und er wird grafisch hervorgehoben. Dann kehrt die Auswertung wieder zum Hauptprogramm zurück.

Auf diese sehr einfache Art lassen sich Tabellen für alle gängigen Zwecke errichten. Der Programmierer muß dafür nur die entsprechende Anzahl von Gadgets bereitstellen und die Berechnungen für die Invertierung der Grafik anpassen. Dies sollte aber keine Schwierigkeit darstellen, da bei einer Tabelle alle Abstände gleichmäßig sind und sich somit auch einfach berechnen lassen.

Scroll-Tabellen

Problematisch wird die Tabellenform bei sehr vielen Punkten oder einer vorher nicht definierten Anzahl von Elementen. Hierfür empfehlen sich sog. Scroll-Tabellen, bei denen man nur eine kleine Auswahl der Elemente sieht, diese aber hoch- oder runterschieben kann.

Ein gutes Beispiel dafür ist die Scroll-Tabelle aus den Preferences für die Druckertreiberauswahl. Hierbei ist vorher nicht bekannt, wie viele Druckertreiber auf der Diskette zu finden sein werden. Man regelt ganz einfach den Ausschnitt über zwei Pfeile, die beim Anklicken den Ausschnitt verschieben.

Und genau da setzt unsere Arbeit ein. Wir entwerfen eine Grafik, in die über eine Ausgaberoutine der Ausschnitt der Tabelle geschrieben wird. Weiterhin fügen wir zwei Gadgets ein, jeweils eins für einen Pfeil. Bei Betätigung der Gadgets verschiebt die Auswertungsroutine den Ausschnitt entsprechend und gibt erneut die Tabelle aus. Als ausgewählt betrachtet man immer das

mittlere von drei Elementen. Hier sehen Sie zuerst die beiden Pfeil-Gadgets ohne Grafiken. Der Aufbau der Grafik wird über extra darauf folgende Ausgabebefehle erledigt:

```
  Border Rand, -1, -1, 1, 200, 14
  Border Kasten, 0, -1, 1, 50, 21
  GadgetDef Hoch, 0&, 51, 60, 48, 18, 0, 1, 1, 0&, 0&, 0&, 1
  GadgetDef Runter, Hoch, 51, 80, 48, 18, 0, 1, 1, 0&, 0&, 0&, 2
  Titel$   = "Scroll-Tabelle"
  WinDef NWindow,100,100,460,150,32+64+512&,15&+4096&,Runter,Titel$
  WinBase  = OpenWindow(NWindow)
  IF WinBase = 0 THEN ERROR 7
  RastPort = PEEKL(WinBase+50)
  UserPort = PEEKL(WinBase+86)
  DrawBorder RastPort, Rand, 100&, 60&
  DrawBorder RastPort, Rand, 100&, 73&
  DrawBorder RastPort, Rand, 100&, 86&
  DrawBorder RastPort, Kasten, 50&, 60&
  DrawBorder RastPort, Kasten, 50&, 79&

  x = 50      : y = 60
 'x-            y- Werte
  x(1) =17+x : x(2) =16+y
  x(3) =34+x : x(4) =16+y
  x(5) =34+x : x(6) =10+y
  x(7) =40+x : x(8) =10+y
  x(9) =25+x : x(10)=2+y
  x(11)=10+x : x(12)=10+y
  x(13)=17+x : x(14)=10+y
  x(15)=17+x : x(16)=16+y

  Move RastPort, 17+50&, 16+60&
  PolyDraw RastPort, 8&, VARPTR(x(1))

  y = 62
'y- Werte
  x(2) =20+y
  x(4) =20+y
  x(6) =26+y
  x(8) =26+y
  x(10)=34+y
  x(12)=26+y
  x(14)=26+y
  x(16)=20+y

  Move RastPort, 17+50&, 20+62&
  PolyDraw RastPort, 8&, VARPTR(x(1))
```

```
FOR i = 1 TO 5
   READ Tabelle$(i)
   IntuiText ITxt(i), 1, 0, 0, Tabelle$(i), 0&
NEXT i
TabOut Aktiv
```

Nachdem über DrawBorder() die Ausgabe der Border-Strukturen direkt abgewickelt wurde, tritt PolyDraw() aus der Graphics.Library in Aktion. Diese Funktion erlaubt es, Koordinatentabellen über verbundene Linien auszugeben. Ein Beispiel, die 3D-Netzgrafik, finden Sie im Kapitel 3 AmigaBASIC. Hier verwenden wir die Routine nur, um die Pfeilgrafik auszugeben.

Danach liest OpenAll aus DATA-Zeilen 5 Texte ein, die später in unserer Tabelle stehen werden. Die Ausgabe erledigt die SUB-Routine TabOut. Sehen Sie hier deshalb noch einmal das ganze Programm mit dieser neuen Ausgaberoutine und den DATA-Zeilen:

```
'************************************
'*                                  *
'* Scroll-Tabellen-Gadgets          *
'* -----------------------          *
'*                                  *
'* Autor  : Wolf-Gideon Bleek       *
'* Datum  : 31. Mai '88             *
'* Grueße : Myrna                   *
'* Version: 1.2                     *
'* Betriebssystem: V1.2 & V1.3      *
'*                                  *
'************************************
  OPTION BASE 1
  DEFLNG a-w
  DEFINT x
  LIBRARY ":bmaps/exec.library"
  DECLARE FUNCTION AllocMem LIBRARY
  DECLARE FUNCTION GetMsg LIBRARY
  LIBRARY ":bmaps/intuition.library"
  DECLARE FUNCTION OpenWindow LIBRARY
  LIBRARY ":bmaps/graphics.library"

  GADGETDOWN = 32&
  GADGETUP   = 64&
  CLOSEW     = 512&

  MList      = 0&
  Lauf       = 1
  Aktiv      = 2
```

```
  DIM x(16)
  DIM SHARED Tabelle$(5), ITxt(5)

HauptProgramm:
  GOSUB OpenAll
  ' Hauptteil
  MainLoop:
    IF Lauf = 1 THEN
      IntuiMsg = GetMsg(UserPort)
      IF IntuiMsg > 0 THEN GOSUB IntuitionMsg
      GOTO MainLoop
    END IF

  GOSUB CloseAll

END
OpenAll:
  Border Rand, -1, -1, 1, 200, 14
  Border Kasten, 0, -1, 1, 50, 21
  GadgetDef Hoch, 0&, 51, 60, 48, 18, 0, 1, 1, 0&, 0&, 0&, 1
  GadgetDef Runter, Hoch, 51, 80, 48, 18, 0, 1, 1, 0&, 0&, 0&, 2
  Titel$   = "Scroll-Tabelle"
  WinDef NWindow,100,100,460,150,32+64+512&,15&+4096&,Runter,Titel$
  WinBase  = OpenWindow(NWindow)
  IF WinBase = 0 THEN ERROR 7
  RastPort = PEEKL(WinBase+50)
  UserPort = PEEKL(WinBase+86)
  DrawBorder RastPort, Rand, 100&, 60&
  DrawBorder RastPort, Rand, 100&, 73&
  DrawBorder RastPort, Rand, 100&, 86&
  DrawBorder RastPort, Kasten, 50&, 60&
  DrawBorder RastPort, Kasten, 50&, 79&

  x = 50     : y = 60
 'x-           y- Werte
  x(1) =17+x : x(2) =16+y
  x(3) =34+x : x(4) =16+y
  x(5) =34+x : x(6) =10+y
  x(7) =40+x : x(8) =10+y
  x(9) =25+x : x(10)=2+y
  x(11)=10+x : x(12)=10+y
  x(13)=17+x : x(14)=10+y
  x(15)=17+x : x(16)=16+y

  Move RastPort, 17+50&, 16+60&
  PolyDraw RastPort, 8&, VARPTR(x(1))

  y = 62
 'y- Werte
  x(2) =20+y
  x(4) =20+y
```

```
  x(6) =26+y
  x(8) =26+y
  x(10)=34+y
  x(12)=26+y
  x(14)=26+y
  x(16)=20+y

  Move RastPort, 17+50&, 20+62&
  PolyDraw RastPort, 8&, VARPTR(x(1))

  FOR i = 1 TO 5
     READ Tabelle$(i)
     IntuiText ITxt(i), 1, 0, 0, Tabelle$(i), 0&
  NEXT i
  TabOut Aktiv
RETURN

CloseAll:
  CALL CloseWindow(WinBase)
  CALL UnDef
RETURN

IntuitionMsg:
  MsgTyp    = PEEKL(IntuiMsg+20)
  Item      = PEEKL(IntuiMsg+28)
  GadgetNr% = PEEK(Item+39)
  CALL ReplyMsg(IntuiMsg)

  IF (MsgTyp = GADGETDOWN) THEN
     'sofortige Aktivierung
     PRINT "DOWN Gadget-Nr.:";GadgetNr%
  END IF

  IF (MsgTyp = GADGETUP) THEN
     'relverify Modus
     PRINT "UP   Gadget-Nr.:";GadgetNr%
     IF GadgetNr% = 1 AND Aktiv<>4 THEN Aktiv=Aktiv+1 : TabOut(Aktiv)
     IF GadgetNr% = 2 AND Aktiv<>1 THEN Aktiv=Aktiv-1 : TabOut(Aktiv)
  END IF

  IF (MsgTyp = CLOSEW) THEN
     'System-Gadget Window schließen
     PRINT "CLOSEWINDOW"
     Lauf = 0
  END IF
RETURN
'-----------------------------------------
SUB DefChip(Buffer,Size)STATIC
 SHARED MList
  Size=Size+8
  Buffer=AllocMem(Size,65538&)
  IF Buffer>0 THEN
    POKEL Buffer,MList
```

```
    POKEL Buffer+4,Size
    MList=Buffer
    Buffer=Buffer+8
  ELSE
    ERROR 7
  END IF
END SUB

SUB UnDef STATIC
 SHARED MList
undef.loop:
  IF MList>0 THEN
    Adresse = PEEKL(MList)
    Groesse = PEEKL(MList+4)
    FreeMem MList, Groesse
    MList   = Adresse
    GOTO undef.loop
  END IF
END SUB

SUB WinDef(bs, x%, y%, b%, h%, IDCMP, f, Gad, T$) STATIC
  Size = 48+LEN(T$)+1
  DefChip bs,Size
  POKEW bs   ,x%       ' LeftEdge
  POKEW bs+ 2,y%       ' TopEdge
  POKEW bs+ 4,b%       ' Width
  POKEW bs+ 6,h%       ' Height
  POKEW bs+ 8,65535&   ' Detail- BlockPen
  POKEL bs+10,IDCMP    ' IDCMPFlags
  POKEL bs+14,f        ' Flags
  POKEL bs+18,Gad      ' FirstGadget
  POKEL bs+26,bs+48    ' Title
  POKEW bs+46,1        ' ScreenType
  FOR i%=1 TO LEN(T$)
     POKE bs+47+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB

SUB GadgetDef(bs, nx, x%, y%, b%, h%, f%, a%, T%, i, txt, si, n%) STATIC
  DefChip bs,44&       ' Gadget-Struktur Länge
  POKEL bs   ,nx       '*NextGadget
  POKEW bs+ 4,x%       ' LeftEdge
  POKEW bs+ 6,y%       ' TopEdge
  POKEW bs+ 8,b%       ' Width
  POKEW bs+10,h%       ' Height
  POKEW bs+12,f%       ' Flags
  POKEW bs+14,a%       ' Activation
  POKEW bs+16,T%       ' GadgetType
  POKEL bs+18,i        ' GadgetRender
  POKEL bs+26,txt      '*GadgetText
  POKEL bs+34,si       ' SpecialInfo
  POKEW bs+38,n%       ' GadgetID
END SUB
```

```
SUB IntuiText(bs, c1%, x%, y%, T$, nx) STATIC
  Size=20+LEN(T$)+1    ' Strukturlänge + Textlänge + Nullbyte
  DefChip bs,Size
  POKE  bs   ,c1%      ' FrontPen
  POKE  bs+ 2,1        ' DrawMode
  POKEW bs+ 4,x%       ' LeftEdge
  POKEW bs+ 6,y%       ' TopEdge
  POKEL bs+12,bs+20    ' IText
  POKEL bs+16,nx       ' NextText
  FOR i%=1 TO LEN(T$)
     POKE bs+19+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB

SUB Border(bs, x%, y%, C%, b%, h%) STATIC
  DefChip bs,48&       ' Strukturlänge + Koordinatentabelle
  POKEW bs  ,x%        ' LeftEdge
  POKEW bs+2,y%        ' TopEdge
  POKE  bs+4,C%        ' FrontPen
  POKE  bs+7,8         ' Count
  POKEL bs+8,bs+16     '*XY
  FOR i%=0 TO 1
     POKEW bs+22+i%*4,h%-1
     POKEW bs+24+i%*4,b%-1
     POKEW bs+32+i%*4,1
     POKEW bs+38+i%*4,h%-1
     POKEW bs+40+i%*4,b%-2
  NEXT
END SUB

'austauschen gegen ein PropInfo
SUB STRINGINFO(bs,max%,buff$) STATIC
  IF LEN(buff$)>max% THEN
     nmax%=LEN(buff$)
  ELSE
     nmax%=max%
  END IF
  IF (nmax% AND 1) THEN nmax%=nmax%+1
  Size=36+2*(nmax%+4)
  DefChip bs,Size
  POKEL bs,bs+36
  POKEL bs+4,bs+40+nmax%
  POKEW bs+10,max%+1
  IF buff$<>""THEN
    FOR i%=1 TO LEN(buff$)
      POKE bs+35+i%,ASC(MID$(buff$,i%,1))
    NEXT
  END IF
END SUB

SUB TabOut(Aktiv) STATIC
SHARED RastPort
```

```
    COLOR 0,0
    FOR i = 0 TO 2
      SetAPen RastPort, 0
      RectFill RastPort, 101&, 13*i+60&, 296&, 13*i+71&
    NEXT i
    COLOR 1,0
    FOR i = Aktiv-1 TO Aktiv+1
       IF i>0 AND i<5 THEN
          POKEW ITxt(i)+6, 62+(i-Aktiv+1)*13
          PrintIText RastPort, ITxt(i), 110&, 0&
       END IF
    NEXT i
    SetDrMd RastPort, 2
    RectFill RastPort, 101&, 73&, 296&, 84&
    SetDrMd RastPort, 1
END SUB

DATA Scroll-Tabelle
DATA Schieberegler
DATA Tabelle
DATA Gadget
DATA Systemgadgets
```

4.1.4 Schieberegler

Wie schon oben erwähnt, sind Schieberegler eines von vielen Bedienelementen. Sie eignen sich ganz besonders, um Werte einzustellen, die in einem bestimmten Bereich liegen, oder Objekte zu positionieren. Als bekanntestes Beispiel gilt hier sicherlich die Farbeinstellung. Jeder Regler kann einen Wert zwischen 0 und 15 annehmen, der die Intensität eines Farbwertes darstellt. Natürlich könnte man auch drei INPUT-Befehle schreiben, die jeweils auf den Wertebreich überprüft werden. Aber mit den Schiebereglern ist die Einstellung doch viel bequemer. Sie können sogar durch leichte Rechts- und Linksbewegung die Rot-, Grün- und Blauwerte stufenlos einstellen.

Auch hierzu bietet die Intuition.Library wieder Unterstützung. Es handelt sich hierbei um erweiterte Gadgets! Der Klickbereich ist bei Schiebereglern der Bereich, in dem man den Schieber hin oder her bewegen kann. Dies kann sowohl horizontal, vertikal als auch in beide Richtungen gleichzeitig sein. Die Grafik, die den Schieberegler an sich verkörpert, wird über die Gadget-Grafik organisiert oder aber von Intuition als Standard vorgege-

ben. Zur Kenntlichmachung ist es üblich, daß von Intuition der Schiebebereich mit einem Kasten umrandet wird. Wie bei allem, so können wir auch hier diese Funktion evtl. abstellen.

Für die oben angesprochene Erweiterung der Gadget-Struktur brauchen wir eine sog. PropInfo-Struktur, die in den Zeiger SpecialInfo eingebunden wird. Hierfür habe ich wieder eine SUB-Routine, die die Speicher- und Parameterarbeit übernimmt:

```
SUB PropInfo(bs, Flags%, HPot%, VPot%, HBody%, VBody%) STATIC
  DefChip bs,22&
  POKEW bs   ,Flags%
  POKEW bs+ 2,HPot%
  POKEW bs+ 4,VPot%
  POKEW bs+ 6,HBody%
  POKEW bs+ 8,VBody%
END SUB
```

Wir tragen in die Info-Struktur mehrere Werte ein, die wir zuerst besprechen sollten. Mit Flags können wir einstellen, ob sich der Schieber horizontal (2) oder vertikal (4) bewegen lassen soll. Weiterhin weisen wir Intuition mit autoknob (1) an, daß keine eigene Grafik für den Schieber vorhanden ist, sondern daß Intuition eine eigene zu zeichnen hat. H- und VPot bezeichnen jeweils die Ausgangsposition des Schiebers. Mit 0 befindet er sich rechts bzw. unten, und mit &HFFFF befindet er sich links bzw. oben. Nach einer Verschiebung durch den Benutzer können wir hier auch die neue Position auslesen. In H- und VBody gibt man die Schrittweite des Schiebers an, mit der er bei einem Klick in die Box springen soll. Beide Werte werden als Teile vom Ganzen (&HFFFF) gerechnet! Alle weiteren Werte, die noch in der Struktur zu finden sind, stellt Intuition ein und sollen uns hier nicht weiter kümmern.

Für die Verwendung eines AutoKnobs, den von Intuition grafisch unterstützten Schieber, benötigen wir weiterhin noch einen 8 Byte langen Speicherbereich, der die Position (x- und y-Koordinaten) und die Breite enthält. Werden alle vier Werte nicht gesetzt, so macht dies die Initialisierungsroutine. Für ein Gadget brauchen wir jetzt also zwei weitere Strukturen:

```
PropInfo PropI, 1+2, 0, 0, &HFFF, 0
IntuiText Text, 2, -80, 2, "Schieber:", 0&
DefChip Buffer, 8&
GadgetDef Gadget, 0&, 150, 30, 100, 10, 0, 1+2, 3, Buffer, Text, PropI,1
```

Aus diesen neuen und allen bekannten Möglichkeiten lassen sich jetzt auch mehrere Proportional-Gadgets zusammenbauen. Als Beispiel habe ich hier ein Listing, das drei solcher Gadgets in das Window einbindet und die Abfrage dahingehend auswertet, daß es nach dem Loslassen des Schiebers die Werte entsprechend in ein Farbregister einträgt. Somit können Sie gleich das Ergebnis der Einstellung betrachten.

```
'************************************
'*                                  *
'* Proportional-Gadgets             *
'* --------------------             *
'*                                  *
'* Autor  : Wolf-Gideon Bleek       *
'* Datum  : 23. Mai '88             *
'* Grueße : Daniel & Nils           *
'* Version: 1.2                     *
'* Betriebssystem: V1.2 & V1.3      *
'*                                  *
'************************************
  OPTION BASE 1
  DEFLNG a-z

  LIBRARY ":bmaps/exec.library"
  DECLARE FUNCTION AllocMem LIBRARY
  DECLARE FUNCTION GetMsg LIBRARY
  LIBRARY ":bmaps/intuition.library"
  DECLARE FUNCTION OpenWindow LIBRARY
  GADGETDOWN = 32&
  GADGETUP   = 64&
  CLOSEW     = 512&

  MList      = 0&
  Lauf       = 1

HauptProgramm:
  GOSUB OpenAll
  ' Hauptteil
  MainLoop:
    IF Lauf = 1 THEN
      IntuiMsg = GetMsg(UserPort)
      IF IntuiMsg > 0 THEN GOSUB IntuitionMsg
      GOTO MainLoop
    END IF
  GOSUB CloseAll
```

```
END

OpenAll:
  IntuiText RotTxt, 2, -80, 2, "Rot", 0&
  IntuiText GrnTxt, 2, -80, 2, "Grün", 0&
  IntuiText BlaTxt, 2, -80, 2, "Blau", 0&
  PropInfo Prop1, 1+2, 0, 0, &HFFF, 0
  PropInfo Prop2, 1+2, 0, 0, &HFFF, 0
  PropInfo Prop3, 1+2, 0, 0, &HFFF, 0
  DefChip Puffer(1), 8&
  DefChip Puffer(2), 8&
  DefChip Puffer(3), 8&
  GadgetDef RotGad,0&,150,30,114,10,0,1+2,3,Puffer(1),RotTxt,Prop1,1
GadgetDef GrnGad,RotGad,150,45,114,10,0,1+2,3,Puffer(2),GrnTxt,Prop2,2
GadgetDef BlaGad,GrnGad,150,60,114,10,0,1+2,3,Puffer(3),BlaTxt,Prop3,3
  Titel$   = "Farbeinstellungen"
  WinDef NWindow,100,100,460,150,32+64+512&,15&+4096&,BlaGad,Titel$
  WinBase  = OpenWindow(NWindow)
  IF WinBase = 0 THEN ERROR 7
  RastPort = PEEKL(WinBase+50)
  UserPort = PEEKL(WinBase+86)
RETURN

CloseAll:
  CALL CloseWindow(WinBase)
  CALL UnDef
RETURN

IntuitionMsg:
  MsgTyp    = PEEKL(IntuiMsg+20)
  Item      = PEEKL(IntuiMsg+28)
  GadgetNr% = PEEK(Item+39)
  CALL ReplyMsg(IntuiMsg)

  IF (MsgTyp = GADGETDOWN) THEN
     'sofortige Aktivierung
     PRINT "DOWN Gadget-Nr.:";GadgetNr%
  END IF

  IF (MsgTyp = GADGETUP) THEN
     'relverify Modus
     PRINT "UP   Gadget-Nr.:";GadgetNr%;
     PRINT " Pos:";PEEKW(Puffer(GadgetNr%))
     Rot = PEEKW(Puffer(1))
     Grn = PEEKW(Puffer(2))
     Bla = PEEKW(Puffer(3))
```

```
      PALETTE 1, Rot/100, Grn/100, Bla/100
   END IF

   IF (MsgTyp = CLOSEW) THEN
      'System-Gadget Window schließen
      PRINT "CLOSEWINDOW"
      Lauf = 0
   END IF
RETURN
'-----------------------------------------
SUB DefChip(Buffer,Size)STATIC
 SHARED MList
  Size=Size+8
  Buffer=AllocMem(Size,65538&)
  IF Buffer>0 THEN
    POKEL Buffer,MList
    POKEL Buffer+4,Size
    MList=Buffer
    Buffer=Buffer+8
  ELSE
    ERROR 7
  END IF
END SUB

SUB UnDef STATIC
 SHARED MList
undef.loop:
  IF MList>0 THEN
    Adresse = PEEKL(MList)
    Groesse = PEEKL(MList+4)
    FreeMem MList, Groesse
    MList   = Adresse
    GOTO undef.loop
  END IF
END SUB

SUB WinDef(bs, x%, y%, b%, h%, IDCMP, f, Gad, T$) STATIC
  Size = 48+LEN(T$)+1
  DefChip bs,Size
  POKEW bs   ,x%       ' LeftEdge
  POKEW bs+ 2,y%       ' TopEdge
  POKEW bs+ 4,b%       ' Width
  POKEW bs+ 6,h%       ' Height
  POKEW bs+ 8,65535&   ' Detail- BlockPen
  POKEL bs+10,IDCMP    ' IDCMPFlags
  POKEL bs+14,f        ' Flags
  POKEL bs+18,Gad      ' FirstGadget
  POKEL bs+26,bs+48    ' Title
  POKEW bs+46,1        ' ScreenType
  FOR i%=1 TO LEN(T$)
     POKE bs+47+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB
```

```
SUB GadgetDef(bs, nx, x%, y%, b%, h%, f%, a%, T%, i, Txt, si, n%) STATIC
  DefChip bs,44&       ' Gadget-Struktur Länge
  POKEL bs   ,nx       '*NextGadget
  POKEW bs+ 4,x%       ' LeftEdge
  POKEW bs+ 6,y%       ' TopEdge
  POKEW bs+ 8,b%       ' Width
  POKEW bs+10,h%       ' Height
  POKEW bs+12,f%       ' Flags
  POKEW bs+14,a%       ' Activation
  POKEW bs+16,T%       ' GadgetType
  POKEL bs+18,i        ' GadgetRender
  POKEL bs+26,Txt      '*GadgetText
  POKEL bs+34,si       ' SpecialInfo
  POKEW bs+38,n%       ' GadgetID
END SUB

SUB IntuiText(bs, c1%, x%, y%, T$, nx) STATIC
  Size=20+LEN(T$)+1    ' IntuiText-Struktur Länge + Textlänge + Nullbyte
  DefChip bs,Size
  POKE  bs   ,c1%      ' FrontPen
  POKE  bs+ 2,1        ' DrawMode
  POKEW bs+ 4,x%       ' LeftEdge
  POKEW bs+ 6,y%       ' TopEdge
  POKEL bs+12,bs+20    ' IText
  POKEL bs+16,nx       ' NextText
  FOR i%=1 TO LEN(T$)
     POKE bs+19+i%,ASC(MID$(T$,i%,1))
  NEXT
END SUB

SUB Border(bs, x%, y%, c%, b%, h%) STATIC
  DefChip bs,48&       ' Border-Struktur Länge + Koordinatentabelle
  POKEW bs  ,x%        ' LeftEdge
  POKEW bs+2,y%        ' TopEdge
  POKE  bs+4,c%        ' FrontPen
  POKE  bs+7,8         ' Count
  POKEL bs+8,bs+16     '*XY
  FOR i%=0 TO 1
     POKEW bs+22+i%*4,h%-1
     POKEW bs+24+i%*4,b%-1
     POKEW bs+32+i%*4,1
     POKEW bs+38+i%*4,h%-1
     POKEW bs+40+i%*4,b%-2
  NEXT
END SUB

SUB PropInfo(bs, Flags%, HPot%, VPot%, HBody%, VBody%) STATIC
  DefChip bs,22&
  POKEW bs   ,Flags%
  POKEW bs+ 2,HPot%
```

```
  POKEW bs+ 4,VPot%
  POKEW bs+ 6,HBody%
  POKEW bs+ 8,VBody%
END SUB
```

Ausblicke

Aus all den in diesem Kapitel gegebenen Informationen können Sie nun einiges machen. Wir haben versucht, ein Schema zu entwickeln, das sich gut in das des Betriebssystems, speziell von Intuition, einpaßt. Alle hier vorgestellten Programme sind modular aufgebaut und ermöglichen es so auf einfachste Weise, in eigene und neue Programme aufgenommen zu werden. Dafür gehen Sie folgenden Weg:

Schreiben Sie jede SUB-Routine in ein neues Verzeichnis in ein extra ASCII-Programm-File. Setzen Sie am besten vor jede Routine eine Kommentarliste, die Informationen zu jedem Parameter enthält. Benötigen Sie jetzt einen Intuition-Aufruf, dann laden Sie das SUB-File über MERGE hinzu und ergänzen in dem OpenAll-Unterprogramm einfach den Aufruf. Am besten spielen wir dies an einem konkreten Fall durch!

Als Abschluß dieses Kapitels folgt eine vollständige Beschreibung für ein Programm, mit dem es möglich ist, alle vier Standard-Farbregister zu modifizieren. Dazu brauchen wir alle bisher besprochenen Unterroutinen. Das Programm ruft dann folgende Gadgets ins Leben. Wir benötigen die vier Grund-Gadgets für die Rücksetzung der Einstellungen oder zur Korrektur.

Die drei Proportional-Gadgets werden für die Rot-, Grün- und Blauwert-Einstellungen gebraucht und sind auch unabkömmlich. Außerdem müssen noch vier neue Gadgets entworfen werden, die zu einer Tabelle zusammenzufügen sind, über die man die gerade einzustellende Farbe selektieren kann. Sind alle Gadgets in das Window eingebunden, mit den richtigen Unterstützungsstrukturen ausgestattet und im Window mit ausreichender Gliederung positioniert, so kann die Abfrageroutine entwickelt werden.

Die Abfrage muß zuerst einmal über die Farbtabelle in einer Variablen speichern, welche Farbe gerade beeinflußt oder gelöscht werden kann. Auf diese Variable werden alle weiteren Gadgets zurückgreifen. Mit den vier Grund-Gadgets erreicht man einerseits die Abbruch- oder Bestätigungsfunktion. Andererseits kann man alle Farben auf ihren Ursprungswert oder nur eine darauf setzen. Dafür ist die Farbnummer entscheidend!

Als letztes kommen die elementaren Proportional-Gadgets, ohne deren Hilfe eine Farbveränderung natürlich nicht zustande käme. Die Veränderungen der Regler werden von der Abfrageroutine an die Farbwerte der einzelnen Farben entsprechend der Farbnummer weitergeleitet. Die Farben müssen dann über Palette gleich umgesetzt werden, damit der Benutzer sofort einen Überblick über die Veränderung hat.

Wenn Sie unter Berücksichtigung aller Grundregeln, die auf den vorhergehenden Seiten genannt wurden, so eine Farbpalette zusammenbauen, haben Sie sicherlich die Kenntnis erworben, wie man professionell programmiert und sein Programm gestaltet.

4.2 Rubberbanding

Im bisherigen Verlauf dieses Kapitels haben Sie erfahren, wie man die wichtigsten Elemente einer professionellen Programmgestaltung programmiert. Sie sollten sich aber nicht davon abhalten lassen, neue Wege zu suchen. Jedes neue Problem erfordert eine speziell abgestimmte Lösung!

In diesem Teil möchte ich noch auf eine Funktion eingehen, die Sie sicher ständig benutzen. Gemeint ist das Rubberbanding, was zu deutsch etwa "Gummibanderln" heißt. Sie können damit ganz einfach durch Ziehen oder Drücken an einem Band, welches beliebig dehnbar ist (daher Gummiband), die Größe von Windows einstellen. Intuition versieht das entsprechende Window dann mit einem Sizing-Gadget. Aber uns interessiert die Programmierung und damit die Anwendung in BASIC.

Der Trick besteht darin, daß die zu verschiebende Linie nicht einfach gezeichnet wird, sondern im "complement"-Modus gezogen wird. Somit erreicht man durch nochmaliges Zeichnen, daß der Hintergrund wieder vollständig hergestellt ist. Man erspart dem Programmierer und dem Programm so eine aufwendige Speicherung des Hintergrundes und die Restauration. Gewöhnlich verwendet man das Rubberbanding für die Einstellung von Arbeitsbereichen (siehe Windows). Aber auch das Ziehen von Rechtecken in Grafikprogrammen läßt sich so sehr komfortabel gestalten.

4.2.1 Flächenbestimmung von Rechtecken

Das Ihnen hier vorliegende Programm ist nur als Mittel zum Zweck geschrieben, denn es erfüllt keinen Sinn. Es soll Ihnen nur zeigen, wie man diese Funktion programmiert. Sie können dann die Mausabfrage in eigene Anwendungen übernehmen. Wenn Sie es starten, so wird ein leeres Window mit dem Mauscursor vor Ihnen liegen. Drücken Sie nun an einer beliebigen Position im Window die linke Taste, und bewegen Sie bei gedrückter Taste die Maus, so sehen Sie das noch frei verstellbare Rechteck. Erst wenn Sie die Taste wieder loslassen, wird das Rechteck mit der Zeichenfarbe 1 auf den Bildschirm gebracht.

```
'*********************************
'*                               *
'* Großziehen mit Rubbers        *
'* ----------------------------- *
'*                               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  : Juni '87             *
'* Version: 1.0                  *
'* Betriebssystem: V1.2 & V1.3   *
'*                               *
'*********************************
LIBRARY ":graphics.library"
ON MOUSE GOSUB SetPoint
MOUSE ON
 WHILE INKEY$<>" "
   SLEEP
 WEND
MOUSE OFF
```

```
END
SetPoint:
 MStat=MOUSE(0)
 IF MStat<>-1 THEN RETURN

 xStart=MOUSE(3)
 yStart=MOUSE(4)
 CALL SetDrMd&(WINDOW(8),2)
 NewPosition:

 mx=MOUSE(1)
 my=MOUSE(2)

 LINE (xStart,yStart)-(mx,my),,b

 WHILE MOUSE(0)=-1
   IF mx<>MOUSE(1) OR my<>MOUSE(2) THEN
     LINE (xStart,yStart)-(mx,my),,b
     GOTO NewPosition
   END IF
 WEND

 CALL SetDrMd&(WINDOW(8),1)
 LINE (xStart,yStart)-(mx,my),,b
 RETURN
```

Variablen

MStat	Mausstatus
mx, my	Mauskoordinaten
xStart	x-Anfangsposition des Rechtecks
yStart	y-Anfangsposition des Rechtecks

Programmbeschreibung

Weil die Führungslinien im Zeichenmodus "complement" geplottet werden, muß die "graphics"-Library geöffnet werden. Sie enthält die Aufrufsequenz. Zu Anfang wird die Mausabfrage auf die Unterroutine "SetPoint" gesetzt. Dann wartet das Programm auf einen Tastendruck. Hat es diesen erhalten, so schaltet es die Mausabfrage wieder ab und beendet sich selbst.

Die Mausabfrage ist der Kern dieses Programms. Sehen wir uns diese Programmzeilen etwas genauer an. Der Mausstatus wird in einer Variablen gesichert. Wenn er kennzeichnet, daß der Anwender die linke Maustaste nicht gedrückt hält, wird aus dem Unterprogramm wieder zurückgesprungen. Ansonsten merkt sich

das Programm die Position als Startwert, und der Zeichenmodus wird auf "complement" gesetzt. Dann zeichnet die Routine das Rechteck und wartet auf eine Mausbewegung.

Erfolgt diese, so wird das Rechteck gelöscht, indem es nochmal gezeichnet wird, und mit der neuen Mausposition wird wieder von vorne begonnen. Erst wenn der Benutzer die Maustaste losläßt, wird die Schleife verlassen. Dann schaltet das Programm wieder den normalen Zeichenmodus ein, und das Rechteck wird endgültig ausgegeben.

4.2.2 Punkte verbinden (Formenbestimmung)

Nicht nur die Möglichkeit, mit dem Rubberbanding Flächengrößen zu bestimmen, bleibt dem Programmierer offen. Sie können noch ganz andere Dinge damit anstellen. Denken Sie sich zum Beispiel eine mathematische Funktion, von der ein beliebiger Punkt ausgewählt werden soll. Das Programm zeichnet die Funktion, und Sie sollen den gewünschten Punkt markieren, damit von zwei weiteren Eckpunkten Verbindungen gezogen werden können (zu welchem Zweck, ist ja nicht so wichtig). Auch hier kann man eine Abwandlung des Rubberbanding anwenden. Sehen Sie sich dazu einmal das folgende Programm an:

```
'*********************************
'* Verbinden mit Rubbers       *
'* Autor  : Wolf-Gideon Bleek  *
'* Datum  : Juni '87           *
'* Version: 1.0                *
'* Betriebssystem: V1.2 & V1.3 *
'*********************************
LIBRARY ":graphics.library"
BasisGrafik:
 LINE (100,180)-(540,180)

 FOR i=100 TO 540
    x=(i-100)/2.444444
    y=SIN(x*3.1415/180)*100
    LINE -(i,180-y)
 NEXT i
ON MOUSE GOSUB SetPoint
MOUSE ON
 WHILE INKEY$<>" "
```

```
    SLEEP
  WEND
MOUSE OFF
END
SetPoint:
 MStat=MOUSE(0)
 IF MStat<>-1 THEN RETURN
 CALL SetDrMd&(WINDOW(8),2)
 NewPosition:
 mx=MOUSE(1)
 CALL Connect(mx)
 WHILE MOUSE(0)=-1
   IF mx<>MOUSE(1) THEN
     CALL Connect(mx)
     GOTO NewPosition
   END IF
 WEND
 CALL SetDrMd&(WINDOW(8),1)
 CALL Connect(mx)
 RETURN
SUB Connect (x) STATIC
 IF x<100 THEN x=100
 IF x>540 THEN x=540
 xw=(x-100)/2.444444
 yw=SIN(xw*3.1415/180)*100
 LINE (100,180)-(x,180-yw)
 LINE -(540,180)
 PSET (x,180-yw)
END SUB
```

Variablen

MStat	Mausstatus
i	Laufvariable
mx	Mausposition
x, y	Grafikkoordinaten
xw, yw	Koordinaten im Sub-Programm

Wenn Sie das Programm starten und dann die linke Maustaste drücken, so sehen Sie, wie auf diesem Bogen ein Punkt wandert, mit dem die Bogenendpunkte verbunden werden. Eine Abwandlung des Thaleskreises! Wie programmiert man dies nun?

Programmbeschreibung

Der Aufbau gleicht eigentlich dem ersten Listing. Hinzugekommen ist noch die Ausgabe des Bogens, der über eine kleine Sinus-Spielerei errechnet wird. Dann sehen Sie aber die gleiche

Warteschleife. Größere Änderungen sind aber im Unterprogramm geschehen: Nach dem Kontrollieren des Mausstatus wird nur noch die x-Position des Cursors überprüft. Von ihr abhängig wird eine SUB-Routine aufgerufen. Sie zeichnet die beiden Verbindungslinien. Auch das Warten beschränkt sich auf die Abfrage einer x-Verschiebung. Dann wird, wie beim ersten Programm, das Gezeichnete gelöscht und mit einer neuen Position geplottet.

Sehen Sie sich ruhig mal die Verbindungsroutine an. Zuerst werden die x-Werte in einen bestimmten Bereich gebracht, denn nicht für jede x-Position existiert ein grafischer Funktionswert. Danach berechnet das Programm anhand "sehr komplizierter Formeln" die Punktkoordinaten und zeichnet die Linien. Fertig!

4.2.3 Positionierung von Objekten

Diese letzte Variante entstand aus der Idee, ein Zeichenprogramm für zweidimensionale Netzgrafik zu schreiben. Wenn Sie in so einem Programm mehrere Objekte gezeichnet haben, kann es leicht vorkommen, daß Ihnen die Lage von einigen nicht mehr gefällt. Am einfachsten wäre es nun, wenn man mit der Maus in das Objekt fährt, die Maustaste gedrückt hält und es solange frei bewegen kann. Ähnliches macht nun das folgende Programm.

Zuerst errechnet es die Eckpunkte eines Kreises. Eigentlich hat ein Kreis keine Ecken, aber er soll doch nur vereinfacht dargestellt werden und, dafür müssen 11 Punkte eben reichen. Vielleicht schreiben Sie das 2D-Netzgrafikprogramm und bauen diese kleine Routine ein? Den Kreis können Sie nach Drücken der linken Taste solange bewegen, bis Sie die Taste wieder loslassen. Dann ist er fixiert.

```
'*********************************
'*                               *
'* Objekte mit Rubbers           *
'* ----------------------------- *
'*                               *
'* Autor  : Wolf-Gideon Bleek    *
'* Datum  : Juni '87             *
'* Version: 1.0                  *
'* Betriebssystem: V1.2 & V1.3   *
'*                               *
'*********************************
LIBRARY ":graphics.library"
ObjektDefinieren:
 DIM SHARED Ob%(10,1)
 Pi=3.141593
 FOR i=0 TO 360 STEP 36
    x=COS(i*Pi/180)*30
    y=SIN(i*Pi/180)*15
    Ob%(i/36,0)=x
    Ob%(i/36,1)=y
 NEXT i

ON MOUSE GOSUB SetObjekt
MOUSE ON
 WHILE INKEY$<>" "
   SLEEP
 WEND
MOUSE OFF
END
SetObjekt:
 MStat=MOUSE(0)
 IF MStat<>-1 THEN RETURN

 CALL SetDrMd&(WINDOW(8),2)
 NewPosition:
 mx=MOUSE(1)
 my=MOUSE(2)
 CALL DrawObjekt(mx,my)
 WHILE MOUSE(0)=-1
   IF mx<>MOUSE(1) OR my<>MOUSE(2) THEN
     CALL DrawObjekt(mx,my)
     GOTO NewPosition
   END IF
 WEND
 CALL SetDrMd&(WINDOW(8),1)
 CALL DrawObjekt(mx,my)
RETURN
SUB DrawObjekt(x,y) STATIC
 PSET (Ob%(0,0)+x,Ob%(0,1)+y)
 FOR i=1 TO 10
```

```
    LINE -(Ob%(i,0)+x,Ob%(i,1)+y)
 NEXT i
 LINE -(Ob%(10,0)+x,Ob%(10,1)+y)
END SUB
```

Variablenfelder

Ob	Feld für die Kreispunkte

Variablen

MStat	Mausstatus
Pi	3.141593
i	Laufvariable
mx, my	Mauskoordinaten
x, y	Koordinaten des Kreises

Programmbeschreibung

Nach dem Öffnen der "graphics"-Library werden in das Feld Ob% die x- und y-Werte eingelesen. Sie werden über eine Schleife errechnet, die für 11 Punkte auf dem Kreisrand die Koordinaten "raussucht". Der Rest des Hauptprogramms ist wie bekannt gestaltet.

Wichtige Änderungen sind erst bei der Mausabfrage zu finden. Zuerst wird wieder nachgesehen, ob die linke Maustaste gedrückt wird, denn sonst springt BASIC gleich wieder ins Hauptprogramm. Ist sie noch gedrückt, so wird der Zeichenmodus gesetzt, und an der momentanen Position wird das Objekt gezeichnet. Dafür trägt die neue SUB-Routine Sorge, die ich gleich noch näher beschreibe. Danach arbeitet das Programm in der Warteschleife weiter, die wie immer erst verlassen wird, wenn Sie die Maustaste loslassen. Allerdings springt das Programm noch einmal vor die Schleife, wenn Sie die Position der Maus verändern, denn dann muß das alte Gitternetz gelöscht werden, damit es am neuen Ort wieder gezeichnet werden kann.

Die SUB-Routine zum Zeichnen des Objektes ist darauf eingestellt, daß im Feld Ob% 11 Koordinatenpaare gespeichert sind. Zuerst zeichnet es den ersten Punkt und verbindet dann alle

weiteren mit dem LINE-Befehl in ihrer Reihenfolge. Vom letzten wird dann noch zum allerersten eine Linie gezogen. Somit sind alle Punkte zu einem Kreis verbunden.

4.3 Intuition-Programmierung unter GFA-BASIC

Auch das neue GFA-BASIC, das wesentlich schneller und komfortabler in Programmierung und Anwendung ist als das Amiga-BASIC, bietet die Möglichkeit, Vorteile und Bedienkomfort von Intuition zu nutzen.

Dazu stehen neben schon vorgefertigten Befehlen auch die vom Amiga her bekannten Libraries jedem Programmierer offen. Allerdings werden sie auf eine etwas andere Weise als beim AmigaBASIC zugänglich gemacht. Die vier wichtigsten stehen dem Programmierer schon ohne Vorarbeit zur Verfügung. Als Grundlage dienen die FD-Files, wie wir es schon am Anfang des AmigaBASIC-Kapitels erklärt haben. Auch hier müssen diese von Commodore vorgefertigten Files erst mit Hilfe eines Umsetzungs-Programms in die für GFA-BASIC nötige Form gebracht werden.

Sehen Sie hier das Listing des im GFA-Handbuch abgedruckten Umsetzungs-Programms in einer modifizierten Form. Es wurde erweitert und funktioniert nun für alle Libraries, die von GFA-BASIC unterstützt werden.

```
' Programm zur Konvertierung von FD-Files
' in GFA-BASIC Format
'
' (mod) by Wolf-Gideon Bleek
'
DIM a$(15)                          ! Register-Variable definieren
INPUT "Welche Library";lib$         ! Eingabe des Library-Namens
IF lib$=""                          ! Abbruch-Möglichkeit für die Eingabe
  END
ENDIF
OPEN "i",#1,"df0:fd1.3/"+lib$+"_lib.fd"   ! FD-File zum Lesen öffnen
OPEN "o",#2,"df1:"+lib$+"_lib.GFA"  ! MERGE-File auf Diskette erzeugen
DO UNTIL EOF(#1)                    ! HAUPTSCHLEIFE
  LINE INPUT #1,a$                  ! 1 Befehlszeile einlesen
  IF LEFT$(a$)="#"                  ! auf internen Befehl überprüfen
```

```
IF LEFT$(a$)="#"                    ! auf internen Befehl überprüfen
  SELECT CVL(MID$(a$,3))
  CASE "base"                       ! Basis-Identifikation
    SELECT UPPER$(MID$(a$,8))
    CASE "_DOS"                     ! DOS-Library
      l$="_DosBase"
    CASE "_LAY"                     ! Layers-Library
      l$="_LayersBase"
    CASE "_INT"                     ! Intuition-Library
      l$="_IntBase"
    CASE "_GFX"                     ! Graphics-Library
      l$="_GfxBase"
    CASE "_SYS"                     ! Exec-Library
      l$="{4}"
    DEFAULT
      l$=MID$(a$,9)+"%"
    ENDSELECT
  CASE "bias"                       ! Basis-Adresse
    o%=VAL(MID$(a$,8))
  CASE "publ"
  CASE "priv"
  ENDSELECT
ELSE IF LEFT$(a$)<>"*"              ! ungleich Kommentar
  PRINT #2,"PROCEDURE ";            ! Funktion: neue Prozedur beginnen
  n$=LEFT$(a$,INSTR(a$,"("))
  nl$=LEFT$(n$,INSTR(a$,"(")-1)
  PRINT #2,nl$;                     ! Name der Prozedur als Funktion
  nr$=MID$(n$,LEN(nl$)+2)
  i%=0
  WHILE LEN(nr$)>1
    a%=INSTR(nr$,",")
    IF a%=0
      a%=LEN(nr$)
    ENDIF
    a$(i%)=LEFT$(nr$,a%-1)
    INC i%
    nr$=MID$(nr$,a%+1)
  WEND
  IF i%                              ! wenn mit Parametern
    a%=1
    WHILE a%<i%
      b%=0
      WHILE b%<a%
        IF a$(a%)=a$(b%)
          a$(a%)=a$(a%)+STR$(a%)
          a$(b%)=a$(b%)+STR$(b%)
        ENDIF
        INC b%
      WEND
      INC a%
    WEND
    PRINT #2,"(";                    ! Klammer auf für Parameter
    j%=0
```

```
      WHILE j%<i%-1
        PRINT #2,a$(j%);"%,";          ! Parametername
        INC j%
      WEND
      PRINT #2,a$(j%);"%)";            ! letzter Parameter
    ENDIF
    PRINT #2
    r$=MID$(a$,LEN(n$)+2)
    i%=0
    WHILE LEN(r$)
      SELECT ASC(r$)
      CASE "D"                         ! Datenregister
        PRINT #2,"m68%(";MID$(r$,2,1);")=";a$(i%);"%"
      CASE "A"                         ! Adressregister
        PRINT #2,"m68%(";VAL(MID$(r$,2,1))+8;")=";a$(i%);"%"
      ENDSELECT
      INC i%
      r$=MID$(r$,4)
    WEND
    PRINT #2,"m68%(14)=";l$            ! Prozessorvariablen
    PRINT #2,"RCALL ";l$;"-";o%;",m68%()"  ! Routine aufrufen
    ADD o%,6
    PRINT #2,"RETURN"                  ! Prozedur verlassen
  ENDIF
LOOP
CLOSE #2                               ! Files wieder schließen
CLOSE #1
Listing FD_Konvert (GFA-BASIC)
```

Programmbeschreibung

Nun ist es uns möglich, auf fast die gleiche Weise, wie wir es schon am Anfang des Kapitels getan haben, Intuition über BASIC zu Programmieren. Hinzugekommen ist aber die Bedienfreundlichkeit und die Geschwindigkeit des GFA-BASIC.

4.3.1 Die fertigen Bedien-Elemente

Sicherlich ist es ein beruhigendes Gefühl, wenn man weiß, daß auch in GFA-BASIC alle Programmiersaiten gezupft werden können, doch manchmal ist man froh, wenn die aufwendigen Betriebssystem-Eingriffe erspart bleiben und mit weniger Aufwand das gleiche Ziel erreicht werden kann.

Deshalb bietet GFA-BASIC neben der freien Betriebssystem-Programmierung auch zwei sehr wichtige und komplexe BASIC-Befehle. Der erste heißt ALERT und ähnelt sehr stark dem System-Requester, der hier aber mit drei Auswahlmöglichkeiten eine erweiterte Version darstellt, somit also noch individueller gestaltet werden kann. Lassen Sie sich aber nicht durch den Namen irritieren! Alert wurde vom Atari ST übernommen und hat nichts mit der Guru-Meditation zu tun.

Mit dem zweiten Befehl wird endlich das große Problem der File-Auswahl behoben, das jedem Programmierer ein Dorn im Auge war. Mit einem einzigen Aufruf wird eine äußerst komfortable File-Auswahl ermöglicht, die dem Programm zum Schluß nur noch den vom Benutzer ausgewählten File-Namen übergibt.

Die Programmierung der File-Select-Box stellt kein großes Problem dar. Sie wird mit nur wenigen Parametern - einer Überschrift, dem Text für das OK-Feld und der Rückgabevariablen - aufgerufen, leistet dafür aber hervorragende Arbeit. Sehen Sie am besten das Beispielprogramm, anhand dessen sich die Funktion viel leichter erklären läßt:

```
' ********************************
' *      File - Select - Test    *
' *      --------------------    *
' *                              *
' * Demonstration zum Aufruf     *
' * einer File-Select-Box        *
' *                              *
' * F|r Amiga Tips & Tricks DB   *
' * (p) im März 1989 by Wgb      *
' * Version: 1.2/1.3             *
' * GFA:     3.0                 *
' *                              *
' ********************************
'
FILESELECT "Bitte wählen Sie ein File","Bestätige","RAM:",pfad$
'
IF pfad$=""                                ! Keine Auswahl erfolgt
  PRINT "Abbruch"
ELSE                                       ! Auswahl auswerten
  '
  zeigerd=1
  WHILE MID$(pfad$,zeigerd,1)<>":" AND zeigerd<=LEN(pfad$)
```

```
    zeigerd=zeigerd+1
  WEND
  device$=LEFT$(pfad$,zeigerd)
  '
  IF LEN(device$)=LEN(pfad$)
    file$=device$
    device$=""
  ELSE
    zeigerf=LEN(pfad$)
WHILE MID$(pfad$,zeigerf,1)<>"/" AND MID$(pfad$,zeigerf,1)<>":" AND
zeigerf>=1
      zeigerf=zeigerf-1
    WEND
    file$=MID$(pfad$,zeigerf+1)
    verzeichnis$=MID$(pfad$,zeigerd+1,zeigerf-zeigerd)
    '
  ENDIF
  PRINT "Device     : ";device$
  PRINT "Verzeichnis: ";verzeichnis$
  PRINT "File-Name  : ";file$
ENDIF
```

Listing: GFA File-Select-Box

Programmbeschreibung

Mit dem ersten Befehl wird die File-Select-Box aufgerufen. Wir übergeben die Überschrift der Box ("Bitte wählen Sie ein File"), den Text ("Bestätige"), den Pfad, von dem zuerst das Verzeichnis geladen werden soll ("RAM", damit's schneller geht), und die Variable, in der wir später unser Ergebnis erhalten. Danach erfolgt die Auswertung:

1. Ist die Variable leer, so hat der Benutzer das Abbruch-Gadget betätigt, die File-Auswahl wurde verweigert.
2. Ist ein Inhalt in der Variablen, dann wird dieser auf Device, Verzeichnis- und File-Angabe hin untersucht. Das Programm liefert zum Schluß die Ausgabe jedes einzelnen der drei Bestandteile. Dies erleichtert später die Bearbeitung von Verzeichnis- oder Diskettenwechseln oder die Änderung des Suffixes am File-Namen.

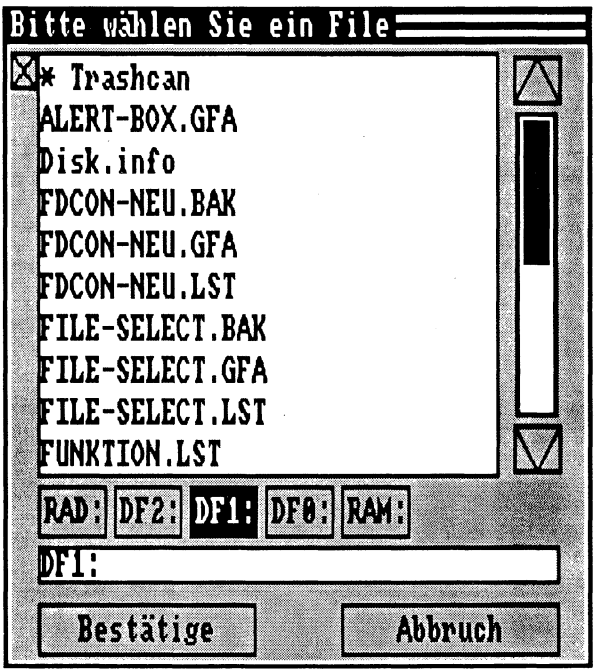


Abb. 3: File-Select-Box

Als nächstes soll die Alert-Box betrachtet werden, die vom Atari ST mit GFA-BASIC übernommen wurde. Der vom Amiga bekannte System-Requester bietet zwei Gadgets zur Auswahl. Da es oftmals drei Antworte-Möglichkeiten auf eine Frage gibt, stellt dieser Alert auch drei Gadgets zur Verfügung. Das folgende Listing verwendet auch diesen Befehl zur Demonstration:

```
' ********************************
' *    Alert-Box Testprogramm    *
' *    ----------------------    *
' *                              *
' * Für Amiga Tips & Tricks DB   *
' * (p) im März 1989 by Wgb      *
' * Version: 1.2/1.3             *
' * GFA:     3.0                 *
' *                              *
' ********************************
'
```

```
ALERT 0,ERR$(100),1,"OK",var
ALERT 0,"Wie geht es Ihnen|Lieber Amiga
User",1,"Gut|Mittel|Schlecht",var
'
IF var=1
  PRINT "Das freut micht, denn mir geht es ja auch immer gut!"
ELSE IF var=2
  PRINT "Man kann sich als Mensch ja nicht jeden Tag gut fühlen."
ELSE IF var=3
  PRINT "Das ist schade, ich hoffe, ich kann Ihnen helfen!"
ENDIF
```

Listing: GFA Alert-Box

Programmbeschreibung

Der Befehl ALERT ist sehr einfach aufzurufen. Im Ganzen werden fünf Parameter übergeben. Der erste steht für ein Symbol, das in dem Alert-Fenster erscheinen soll. Das war beim Atari ST so üblich. Doch weil der Amiga diese Symbole nicht kennt, hat die dort stehende Zahl keine Bedeutung. Wählen Sie einfach immer 0, damit später bei einer Implementierung keine Probleme auftreten.

Als nächstes folgt der Erläuterungs-Text, der aus fünf Zeilen á 40 Zeichen bestehen kann. Jede Zeile wird mit einem "|"-Strich von der folgenden getrennt. Überstehende Zeichen oder Zeilen werden einfach abgetrennt, weil sie nicht dargestellt werden können.

Für die maximal drei Antwort-Gadgets wird wieder eine String-Variable definiert, in der entsprechend den Gadgets bis zu drei Texte getrennt durch das oben verwendete Zeichen angegeben werden müssen. Vor dieser Variablen wird noch die Nummer des Gadgets angegeben, bei dem die Bestätigung noch zusätzlich über die RETURN-Taste geschehen kann.

Nun fehlt nur noch die Angabe der Variablen, in der die Nummer des ausgewählten Gadgets übertragen werden soll. Fertig ist dieser Befehl!

Im Programm selbst rufen wir zuerst den Befehl mit der Funktion ERR$() auf, da diese je nach Wahl der Nummer einen

Fehlertext liefert, der für die Alert-Box vorbereitet ist. Unter der Nummer 100 findet man die System-Meldung von GFA-BASIC. Als zweites Beispiel ist nach dem Aufruf der Alert-Box eine Auswertung programmiert, die Sie angepaßt in eigene Programme übernehmen können.

5. AmigaBASIC Intern

AmigaBASIC, das sich auf jeder Extras-Disk befinden sollte, zeichnet sich bekanntlich nicht nur durch einen sehr mächtigen Befehlssatz aus, sondern auch durch eine recht dürftige Dokumentation, die sich im wesentlichen auf eine zumeist unvollständige Beschreibung der Befehle beschränkt. Wer vor der Bekanntschaft mit seinem Amiga bereits einen C64 hatte, wird sich sicherlich an die vielen kleinen Utilities erinnern, mit deren Hilfe man Programme verändern, Programme erzeugen oder nur aus ihnen Daten herauslesen konnte. Unmöglich auf dem Amiga, werden Sie jetzt sagen, schließlich wird jedes Programm an eine andere Adresse geladen usw.

Sie haben durchaus recht, wenn Sie denken, daß eine Manipulation im Speicher derart kompliziert ist, daß ein solches Manipulationsprogramm ungeahnte Ausmaße annehmen müßte. Glücklicherweise verfügen wir aber über Hilfsmittel, die Programm-Veränderungen zulassen. Ich meine die Laufwerke wie das eingebaute Diskettenlaufwerk df0:, die allseits beliebten RAM-Disks ram: und RAD: und eventuell angeschlossene Zusatzlaufwerke. Schließlich werden Programme auf Disketten abgespeichert. Die Möglichkeiten, die uns diese Laufwerke eröffnen, gehen sogar so weit, daß es machbar ist, laufende Programme sich selbst verändern zu lassen und sie in dieser geänderten Form weiterlaufen zu lassen, doch dazu später mehr.

Man kann AmigaBASIC-Programme auf drei verschiedene Arten abspeichern. Dieses macht unsere folgenden Exkursionen etwas komplizierter, die Anwendungen, die noch folgen, jedoch um so leichter. Bevor ich allerdings die Programmierung der Utilities beschreibe, möchte ich, daß Sie die dazu notwendigen Kenntnisse über den Aufbau der drei verschiedenen File-Typen erlangen, damit es Ihnen möglich ist, die Utilities für Ihren persönlichen Bedarf umzuschreiben bzw. eigene zu entwickeln. Durch die zugegebenermaßen große Menge Theorie, die jetzt folgen wird, werden wir uns also durchbeißen müssen.

5.1 FileMonitor der Superlative

Zunächst möchte ich Ihnen einen File-Monitor vorstellen, der es Ihnen gestatten wird, jedes beliebige Programm hexadezimal und als ASCII-Text anzuschauen und zu verändern. Diesen oder einen beliebigen anderen FileMonitor werden Sie in den nachfolgenden Kapiteln häufig brauchen, um die entsprechenden Manipulationen durchführen zu können und die beschriebenen Punkte zu verdeutlichen. Hier das Programm:

```
OPTION BASE 1
DEFLNG a-z
ON ERROR GOTO FAILED
DECLARE FUNCTION ALLOCMEM LIBRARY
DECLARE FUNCTION GETMSG LIBRARY
LIBRARY":bmaps/exec.library"
LIBRARY":bmaps/graphics.library"
DECLARE FUNCTION OPENSCREEN LIBRARY
DECLARE FUNCTION OPENWINDOW LIBRARY
LIBRARY":bmaps/intuition.library"
DECLARE FUNCTION LOCK LIBRARY
DECLARE FUNCTION EXAMINE LIBRARY
DECLARE FUNCTION EXNEXT LIBRARY
DECLARE FUNCTION IOERR LIBRARY
DECLARE FUNCTION XOPEN LIBRARY
DECLARE FUNCTION XREAD LIBRARY
DECLARE FUNCTION XWRITE LIBRARY
DECLARE FUNCTION SEEK LIBRARY
LIBRARY":bmaps/dos.library"
LPRINT
PRINT"----------FILEMONITOR-V1.0----------"
PRINT"- '88 by DATA BECKER (w)'88 by S. M."
PRINT
PRINT"Program starts in a few seconds."
PRINT"Please stand by...(no Multitasking"
PRINT"during initialization!)"
DIM SHARED borders(14),itxt(25),gadgets(24),sinfo(2)
bfec01=12577793&
clearentry$=SPACE$(30)
clearstring$=STRING$(80,0)
INITIALIZE
DIRECTORY
start%=-1
gesperrt%=0
WHILE (-1)
  qualifier%=PEEK(bfec01)
  IF (qualifier%>&H60)AND(qualifier%<&H68)THEN
    IF qualifier%AND 1 THEN GOSUB tastendruck
  END IF
```

```
    intuimsg=GETMSG(userport)
    IF intuimsg>0 THEN GOSUB IntuitionMsg
  WEND
IntuitionMsg:
  MsgTyp=PEEKL(intuimsg+20)
  IF MsgTyp=2097152& THEN
    IF start% THEN RETURN
    ascii.i%=PEEKW(intuimsg+24)
    IF ascii.i%>0 GOTO tastendruck
  END IF
  Item=PEEKL(intuimsg+28)
  GadgetNr%=PEEK(Item+39)
  IF MsgTyp=32 THEN
    IF(GadgetNr%=10)OR(GadgetNr%=14)THEN gesperrt%=-1
    RETURN
  END IF
  IF MsgTyp<>64 THEN RETURN
  gesperrt%=0
  IF GadgetNr%<0 THEN ERROR 255
  IF GadgetNr%<5 THEN
    COPYMEM SADD(clearstring$),sinfo(1)+36,80
    POKEL sinfo(1)+36,CVL("DF"+CHR$(47+GadgetNr%)+":")
    lasttype%=0
    DIRECTORY
  ELSEIF GadgetNr%<10 THEN
    SETFILEACTDIR
    IF lasttype%=1 THEN
      STATUS "Loading Block"
      OPENFILE
      IF oldhandle>0 THEN :STATUS "Edit"
      RETURN
    END IF
    DIRECTORY
  ELSEIF GadgetNr%=10 THEN
    gesperrt%=0
    DIRECTORY
    RETURN
  ELSEIF GadgetNr%=11 THEN
    IF dirstart%>4 THEN dirstart%=dirstart%-5:DISPLAYDIR
  ELSEIF GadgetNr%=12 THEN
    IF anzahl%>dirstart%+5 THEN
      dirstart%=dirstart%+5
      DISPLAYDIR
    END IF
  ELSEIF GadgetNr%=13 THEN
    DIRECTORY
  ELSEIF GadgetNr%=14 THEN
    gesperrt%=0
    newoffset=PEEKL(sinfo(2)+28)*488
    IF (newoffset>=newflen)OR(newoffset<0) THEN
      POKEL sinfo(2)+36,CVL("0"+MKI$(0)+CHR$(0))
      STATUS "illegal Input"
      DISPLAYBEEP scrbase
```

```
    RETURN
  END IF
  oldpos=SEEK(oldhandle,newoffset,-1)
  currentoffset=newoffset
  STATUS "reading Block"
  READBLOCK
  RETURN
ELSEIF oldhandle=0 THEN
  POKEL sinfo(2)+36,CVL("0"+MKI$(0)+CHR$(0))
  STATUS "no File selected"
  DISPLAYBEEP scrbase
  RETURN
ELSEIF GadgetNr%=15 THEN
  COPYMEM fundo,fbuffer,488
  DISPLAYBUFFER
ELSEIF GadgetNr%=16 THEN
  STATUS "reading again"
  oldpos=SEEK(oldhandle,-gelesen,0)
  READBLOCK
ELSEIF GadgetNr%=17 THEN
  IF currentoffset<newflen-488 THEN
    STATUS "reading next Sec"
    currentoffset=currentoffset+488
    READBLOCK
  END IF
ELSEIF GadgetNr%=18 THEN
  IF currentoffset>487 THEN
    STATUS "reading last Sec"
    currentoffset=currentoffset-488
    oldpos=SEEK(oldhandle,-gelesen-488,0)
    READBLOCK
  END IF
ELSEIF GadgetNr%=19 THEN
  STATUS "writing Buffer"
  oldpos=SEEK(oldhandle,-gelesen,0)
  wr=XWRITE(oldhandle,fbuffer,gelesen)
ELSEIF GadgetNr%=20 THEN
  DUMPFILE
ELSEIF GadgetNr%=21 THEN
  DUMPBUFFER
ELSEIF GadgetNr%=22 THEN
  edmode%=0
  STATUS "switched to HEX"
ELSEIF GadgetNr%=23 THEN
  edmode%=1
  STATUS "switched to ASCII"
ELSEIF GadgetNr%=24 THEN
  STATUS "ARE YOU SURE? Y/N"
  t%=0
  WHILE (t%<>&H9D)AND(t%<>&H93)
    t%=PEEK(bfec01)
  WEND
  IF t%=&H9D THEN
```

```
      STATUS "You ARE sure! BYE"
      GOTO FAILED
    END IF
  END IF
  STATUS "OKAY"
  RETURN
tastendruck:
  ascii$=UCASE$(CHR$(ascii.i%))
  IF edmode%=1 GOTO ASCIImode
  wert%=INSTR("0123456789ABCDEF",ascii$)-1
  IF qualifier%=&H67 THEN
    offset%=offset%-24
    IF offset%<0 THEN offset%=gelesen-1:nibble%=1
    CURSORAUS
    CURSORAN
    RETURN
  ELSEIF qualifier%=&H65 THEN
    offset%=offset%+24
    IF offset%>=gelesen THEN offset%=0:nibble%=0
    CURSORAUS
    CURSORAN
    RETURN
  ELSEIF qualifier%=&H63 THEN
    IF nibble%=0 THEN
      nibble%=1
    ELSE
      nibble%=0
      offset%=offset%+1
      IF offset%>=gelesen THEN offset%=0
    END IF
    CURSORAUS
    CURSORAN
    RETURN
  ELSEIF qualifier%=&H61 THEN
    IF nibble%=1 THEN
      nibble%=0
    ELSE
      nibble%=1
      offset%=offset%-1
      IF offset%<0 THEN offset%=gelesen-1
    END IF
    CURSORAUS
    CURSORAN
    RETURN
  END IF
  IF wert%>=0 THEN
    IF nibble%=0 THEN andi%=15:muls%=16:GOTO mk
    andi%=240
    muls%=1
mk: a%=(PEEK(fbuffer+offset%)AND andi%)+wert%*muls%
    POKE fbuffer+offset%,a%
    CURSORAUS
    MOVE rastport,o.x%,o.y%+6
```

```
    SETAPEN rastport,1
    SETBPEN rastport,0
    TEXT rastport,SADD("0123456789ABCDEF")+wert%,1
    MOVE rastport,(o.b%+54)*8,o.y%+6
    SETAPEN rastport,0
    SETBPEN rastport,1
    TEXT rastport,fbuffer+offset%,1
    IF nibble%=0 THEN nibble%=1:GOTO mk2
    nibble%=0
    offset%=offset%+1
    IF offset%>=gelesen THEN offset%=0
mk2:
    CURSORAN
    RETURN
  END IF
  RETURN
ASCIImode:
  IF qualifier%=&H67 THEN
    offset%=offset%-24
    IF offset%<0 THEN offset%=gelesen-1:nibble%=1
    CURSORAUS
    CURSORAN
    RETURN
  ELSEIF qualifier%=&H65 THEN
    offset%=offset%+24
    IF offset%>=gelesen THEN offset%=0:nibble%=0
    CURSORAUS
    CURSORAN
    RETURN
  ELSEIF qualifier%=&H63 THEN
    offset%=offset%+1:IF offset%>=gelesen THEN offset%=0
    CURSORAUS
    CURSORAN
    RETURN
  ELSEIF qualifier%=&H61 THEN
    offset%=offset%-1
    IF offset%<0 THEN offset%=gelesen-1
    CURSORAUS
    CURSORAN
    RETURN
  END IF
  IF ascii$<>CHR$(0) THEN
    wert%=ascii.i%
    POKE fbuffer+offset%,wert%
    CURSORAUS
    MOVE rastport,o.x%+(o.m%=0)*nibble%,o.y%+6
    SETAPEN rastport,1
    SETBPEN rastport,0
    TEXT rastport,SADD(RIGHT$("0"+HEX$(wert%),2)),2
    SETAPEN rastport,0
    SETBPEN rastport,1
    MOVE rastport,(o.b%+54)*8,o.y%+6
    TEXT rastport,fbuffer+offset%,1
```

```
    offset%=offset%+1
    IF offset%>=gelesen THEN offset%=0
    CURSORAN
    RETURN
  END IF
  RETURN
FAILED:
  UNDEF
  IF scrbase>0 THEN
    IF winbase>0 THEN
      CLOSEWINDOW winbase
      IF oldhandle>0 THEN :XCLOSE oldhandle
    END IF
    CLOSESCREEN scrbase
  END IF
  LIBRARY CLOSE
  SYSTEM
SUB DUMPBUFFER STATIC
 SHARED fbuffer,HEXBUFF,currentlongs,currentoffset
  ausgabe$=SPACE$(1134)
  HEXBUFF currentlongs-1,fbuffer,SADD(ausgabe$)
  STATUS "printing"
  FOR i%=0 TO 20
    LPRINT RIGHT$("      "+STR$(currentoffset+i%*20),8);": ";
    LPRINT MID$(ausgabe$,i%*54+1,54)
 NEXT
 LPRINT
END SUB
SUB DUMPFILE STATIC
 SHARED gelesen,oldhandle,currentoffset
  savedoffset=currentoffset
  oldpos=SEEK(oldhandle,0,-1)
  currentoffset=0
df.loop:
  READBLOCK
  DUMPBUFFER
  currentoffset=currentoffset+488
  IF gelesen=488 GOTO df.loop
  currentoffset=savedoffset
  oldpos=SEEK(oldhandle,currentoffset,-1)
  READBLOCK
END SUB
SUB CURSORAN STATIC
 SHARED o.x%,o.y%,edmode%,o.m%,rastport,offset%,nibble%
 SHARED o.b%
  z%=INT(offset%/24)
  o.b%=offset%-z%*24
  l%=INT(o.b%/4)
  o.x%=(o.b%*2+l%-(edmode%=0)*nibble%)*8
  o.y%=z%*8+2
  SETAPEN rastport,3
  SETDRMD rastport,3
  RECTFILL rastport,o.x%,o.y%,o.x%+7-(edmode%=1)*8,o.y%+7
```

```
  RECTFILL rastport,(o.b%+54)*8,o.y%,(o.b%+54)*8+7,o.y%+7
  SETDRMD rastport,1
  o.m%=edmode%
END SUB
SUB CURSORAUS STATIC
 SHARED o.x%,o.y%,o.m%,o.b%,rastport
  SETAPEN rastport,3
  SETDRMD rastport,3
  RECTFILL rastport,o.x%,o.y%,o.x%+7-(o.m%=1)*8,o.y%+7
  RECTFILL rastport,(o.b%+54)*8,o.y%,(o.b%+54)*8+7,o.y%+7
  SETDRMD rastport,1
END SUB
SUB OPENFILE STATIC
 SHARED oldhandle,scrbase,currentoffset,actdir,newflen
 SHARED numblocks
  IF oldhandle>0 THEN :XCLOSE oldhandle
  oldhandle=XOPEN(actdir,1005)
  IF oldhandle=0 THEN
    STATUS "File Open Error"
    DISPLAYBEEP scrbase
    EXIT SUB
  END IF
  numblocks=newflen/488
  w=CVL(RIGHT$("  "+STR$(numblocks),4))
  POKEL itxt(12)+20,w
  currentoffset=0
  READBLOCK
END SUB
SUB READBLOCK STATIC
 SHARED oldhandle,fbuffer,fundo,gelesen,currentlongs
 SHARED currentoffset
  gelesen=XREAD(oldhandle,fbuffer,488)
  IF gelesen<488 THEN
    v$=STRING$(488-gelesen,0)
    COPYMEM SADD(v$),fbuffer+gelesen,LEN(v$)
  END IF
  x=currentoffset/488
  w=CVL(LEFT$(MID$(STR$(x),2)+MKL$(0),4))
  POKEL sinfo(2)+36,w
  currentlongs=(gelesen+3)/4
  COPYMEM fbuffer,fundo,488
  DISPLAYBUFFER
END SUB
SUB DISPLAYBUFFER STATIC
 SHARED HEXBUFF,currentlongs,fbuffer,rastport,gelesen
 SHARED start%,offset%,nibble%
  ASCIIbuffer$=SPACE$(1134)
  HEXBUFF currentlongs-1,fbuffer,SADD(ASCIIbuffer$)
  SETAPEN rastport,0
  RECTFILL rastport,0,0,639,190
  SETAPEN rastport,1
  SETBPEN rastport,0
  FOR i%=0 TO 20
```

```
    MOVE rastport,0,i%*8+8
    TEXT rastport,SADD(ASCIIbuffer$)+i%*54,54
  NEXT
  SETAPEN rastport,0
  SETBPEN rastport,1
  l%=24
  FOR i%=0 TO 20
    MOVE rastport,432,i%*8+8
    IF i%=20 THEN l%=8
    TEXT rastport,fbuffer+i%*24,l%
  NEXT
  start%=0
  offset%=0
  nibble%=0
  CURSORAN
END SUB
SUB SETFILEACTDIR STATIC
 SHARED GadgetNr%,actdir,scrbase,clearstring$,newflen
 SHARED dirstart%,dirbuff,lasttype%
  vergl1$=STRING$(31,0)
  vergl2$=STRING$(80,0)
  COPYMEM actdir,SADD(vergl2$),79
  COPYMEM itxt(GadgetNr%)+20,SADD(vergl1$),30
  l2%=INSTR(vergl2$,CHR$(0))-1
  l1%=INSTR(vergl1$,CHR$(0))-1
  IF lasttype%=1 THEN
    l2%=INSTR(vergl2$,":")
sfad.loop:
    l3%=INSTR(l2%+1,vergl2$,"/")
    IF l3%>l2% THEN l2%=l3%:GOTO sfad.loop
  END IF
  IF(l1%+l2%)>78 THEN
    STATUS "FileName Too Long"
    DISPLAYBEEP scrbase
    EXIT SUB
  END IF
  v$=LEFT$(vergl2$,l2%)
  IF(lasttype%>1)AND(RIGHT$(v$,1)<>":")THEN v$=v$+"/"
  lasttype%=PEEK(dirbuff+(dirstart%+GadgetNr%-5)*36+31)
  v$=LEFT$(v$+vergl1$+clearstring$,79)
  COPYMEM SADD(v$),actdir,79
  newflen=PEEKL(dirbuff+(dirstart%+GadgetNr%-5)*36+32)
END SUB
SUB DISPLAYDIR STATIC
 SHARED dirstart%,anzahl%,clearentry$,dirbuff,winbase
  FOR i%=5 TO 9
    COPYMEM SADD(clearentry$),itxt(i%)+20,30
  NEXT
  i%=0
  IF anzahl%<=dirstart% GOTO displaydir.show
  REFRESHGADGETS gadgets(23),winbase,0
displaydir.loop:
  a=dirbuff+(i%+dirstart%)*36
```

```
  COPYMEM a,itxt(i%+5)+20,30
  POKE itxt(i%+5),PEEK(a+31)
  i%=i%+1
  IF (i%<5)AND(anzahl%>(dirstart%+i%))GOTO displaydir.loop
displaydir.show:
  REFRESHGADGETS gadgets(23),winbase,0
END SUB
SUB DIRECTORY STATIC
 SHARED anzahl%,dirstart%,actdir,lasttype%
 SHARED fileinfo,clearentry$,dirbuff,newflen
  STATUS "Examining Entry"
  dirlock=LOCK(actdir,-2)
  IF dirlock=0 THEN
    STATUS "File not found"
    EXIT SUB
  END IF
  e=EXAMINE(dirlock,fileinfo)
  IF e=0 THEN
    UNLOCK dirlock
    STATUS "Examine Error"
    EXIT SUB
  END IF
  IF PEEKL(fileinfo+120)<0 THEN
    newflen=PEEKL(fileinfo+124)
    UNLOCK dirlock
    OPENFILE
    lasttype%=1
    EXIT SUB
  END IF
  lasttype%=3
  anzahl%=0
  dirstart%=0
  FOR i%=5 TO 9
    COPYMEM SADD(clearentry$),itxt(i%)+20,30
  NEXT
  STATUS "reading Directory"
directory.loop:
  e=EXNEXT(dirlock,fileinfo)
  IF e=0 THEN
    e=IOERR
    IF e<>232 THEN
      STATUS "Directory invalid"
      anzahl%=0
    ELSE
      STATUS "Okay"
    END IF
    UNLOCK dirlock
    DISPLAYDIR
    EXIT SUB
  END IF
  a=dirbuff+anzahl%*36
  COPYMEM fileinfo+8,a,30
  IF PEEKL(fileinfo+120)<0 THEN c%=1 ELSE c%=3
```

```
  POKE a+31,c%
  POKEL a+32,PEEKL(fileinfo+124)
  anzahl%=anzahl%+1
  IF anzahl%<72 GOTO directory.loop
  UNLOCK dirlock
  STATUS "Okay"
  DISPLAYDIR
END SUB
SUB INITIALIZE STATIC
 SHARED HEXBUFF,fbuffer,fundo,nscreen,dirbuff,fileinfo
 SHARED actdir,scrbase,winbase,viewport,rastport
 SHARED userport
  FORBID
  DEFCHIP HEXBUFF,60&
  DEFCHIP fbuffer,488&
  DEFCHIP fundo,488&
  DEFCHIP nscreen,88&
  DEFCHIP dirbuff,2592&
  DEFCHIP fileinfo,252&
  DEFCHIP darts,68&
  borders(13)=darts+28
  borders(14)=darts+48
  FOR i%=0 TO 14
    READ i$
    POKEW HEXBUFF+i%*4,VAL("&H"+LEFT$(i$,4))
    POKEW HEXBUFF+i%*4+2,VAL("&H"+RIGHT$(i$,4))
  NEXT
  FOR i%=0 TO 6
    READ i$
    POKEW darts+i%*4,VAL("&H"+LEFT$(i$,4))
    POKEW darts+i%*4+2,VAL("&H"+RIGHT$(i$,4))
  NEXT
  POKE darts+29,10
  POKE darts+31,3
  POKE darts+33,7
  POKE darts+35,7
  POKE darts+37,1
  POKEW darts+42,256
  COPYMEM darts+28,darts+48,20
  FOR i%=0 TO 1
    POKEL darts+i%*20+38,darts+i%*14
  NEXT
  FOR i%=1 TO 12
    READ a%,b%,c%,d%,e%,f%
    BORDER borders(i%),a%,b%,c%,d%,e%
    IF f%>0 THEN POKEL borders(i%)+12,borders(f%)
  NEXT
  FOR i%=1 TO 4
    INTUITEXT itxt(i%),1,6,3,"DF"+CHR$(47+i%)+":",0&
  NEXT
  FOR i%=5 TO 9
    INTUITEXT itxt(i%),1,8,0,SPACE$(30),0&
  NEXT
```

```
  FOR i%=10 TO 25
    READ a%,b%,c%,d$,e%
    IF e%>0 THEN f=itxt(e%) ELSE f=0
    INTUITEXT itxt(i%),a%,b%,c%,d$,f
  NEXT
  STRINGINFO sinfo(1),79,"DF0:"
  STRINGINFO sinfo(2),4,"0"+STRING$(15,0)
  actdir=sinfo(1)+36
  d=0
  FOR i%=1 TO 24
    READ e%,f%,G%,h%,j%,k%,l%,m%,n%,o%
    IF o%>0 THEN a=sinfo(o%) ELSE a=0
    IF n%>0 THEN b=itxt(n%) ELSE b=0
    IF m%>0 THEN c=borders(m%) ELSE c=0
    GADGET gadgets(i%),d,e%,f%,G%,h%,j%,k%,l%,c,b,a,i%
    d=gadgets(i%)
  NEXT
  POKEL nscreen+4,41943296&
  POKE nscreen+9,2
  POKE nscreen+12,192
  POKEW nscreen+14,&H10F
  nwindow=nscreen+32
  POKEL nwindow+4,41943296&
  POKEW nwindow+8,259
  POKE nwindow+11,32
  POKE nwindow+13,96
  POKE nwindow+15,1
  POKE nwindow+16,24
  POKEL nwindow+18,d
  POKE nwindow+47,15
  POKEW nscreen+82,&HFFF
  POKE nscreen+84,15
  POKEW nscreen+86,&HFD0
  PERMIT
  scrbase=OPENSCREEN(nscreen)
  IF scrbase=0 THEN ERROR 7
  POKEL nwindow+30,scrbase
  winbase=OPENWINDOW(nwindow)
  IF winbase=0 THEN ERROR 7
  rastport=PEEKL(winbase+50)
  viewport=scrbase+44
  userport=PEEKL(winbase+86)
  LOADRGB4 viewport,nscreen+80,4
END SUB
SUB STATUS(t$)STATIC
 SHARED winbase
  t$=LEFT$(t$+SPACE$(17),17)
  COPYMEM SADD(t$),itxt(22)+20,17
  REFRESHGADGETS gadgets(23),winbase,0
END SUB
SUB DEFCHIP(Buffer,size)STATIC
 SHARED MList
  size=size+8
```

```
  Buffer=ALLOCMEM(size,65538&)
  IF Buffer>0 THEN
    POKEL Buffer,MList
    POKEL Buffer+4,size
    MList=Buffer
    Buffer=Buffer+8
  ELSE
    ERROR 7
  END IF
END SUB
SUB UNDEF STATIC
 SHARED MList
undef.loop:
  IF MList>0 THEN
    Buffer=PEEKL(MList)
    size=PEEKL(MList+4)
    FREEMEM MList,size
    MList=Buffer
    GOTO undef.loop
  END IF
END SUB
SUB GADGET(bs,nx,x%,y%,b%,h%,f%,a%,t%,i,txt,si,n%)STATIC
  DEFCHIP bs,44&
  POKEL bs,nx
  POKEW bs+4,x%
  POKEW bs+6,y%
  POKEW bs+8,b%
  POKEW bs+10,h%
  POKEW bs+12,f%
  POKEW bs+14,a%
  POKEW bs+16,t%
  POKEL bs+18,i
  POKEL bs+26,txt
  POKEL bs+34,si
  POKEW bs+38,n%
END SUB
SUB INTUITEXT(bs,c1%,x%,y%,t$,nx)STATIC
  size=20+LEN(t$)+1
  DEFCHIP bs,size
  POKE bs,c1%
  POKE bs+2,1
  POKEW bs+4,x%
  POKEW bs+6,y%
  POKEL bs+12,bs+20
  POKEL bs+16,nx
  COPYMEM SADD(t$),bs+20,LEN(t$)
END SUB
SUB BORDER(bs,x%,y%,c%,b%,h%)STATIC
  DEFCHIP bs,48&
  POKEW bs,x%
  POKEW bs+2,y%
  POKE bs+4,c%
  POKE bs+7,8
```

```
  POKEL bs+8,bs+16
  FOR i%=0 TO 1
    POKEW bs+22+i%*4,h%-1
    POKEW bs+24+i%*4,b%-1
    POKEW bs+32+i%*4,1
    POKEW bs+38+i%*4,h%-1
    POKEW bs+40+i%*4,b%-2
  NEXT
END SUB
SUB STRINGINFO(bs,max%,buff$)STATIC
  IF LEN(buff$)>max% THEN nmax%=LEN(buff$) ELSE nmax%=max%
  IF(nmax%AND 1)THEN nmax%=nmax%+1
  size=36+2*(nmax%+4)
  DEFCHIP bs,size
  POKEL bs,bs+36
  POKEL bs+4,bs+40+nmax%
  POKEW bs+10,max%+1
  IF buff$<>""THEN
    COPYMEM SADD(buff$),bs+36,LEN(buff$)
  END IF
END SUB
DATA 48E7F0C0,4CEF0308,001C5303,22187407,E9991001,0200000F
DATA 06000030,0C00003A,65040600,000712C0,51CAFFE6,12FC0020
DATA 51CBFFDA,4CDF030F,4E750000,10003800,7C00FE00,38003800
DATA 38003800,38003800,FE007C00,38001000
DATA 0,0,2,43,13,0,-6,-3,2,268,45,0,-6,-3,3,268,13,0
DATA 0,0,2,28,13,0,0,-45,2,28,13,4,0,-15,2,28,13,5
DATA -62,-3,2,172,13,0,0,0,2,65,13,0,0,0,2,109,13,0
DATA 0,15,2,218,13,9,0,0,2,60,13,0,0,0,2,43,28,0
DATA 3,-56,0,"Block:",0,3,40,0,"of:",10,1,72,0,"   0",11
DATA 3,6,3,"OK",0,1,6,3,"UNDO",0,1,6,3,"PRINT BUFFER",0
DATA 1,6,3," PRINT FILE",0,1,17,3,"READ",0,1,17,3,"NEXT",0
DATA 1,17,3,"BACK",0,1,13,3,"WRITE",0,3,6,18,"Status:",15
DATA 1,70,18,"reading Directory",21,1,9,3,"ASCII",0
DATA 1,9,3," HEX",0,1,6,10,"QUIT",0
DATA 0,198,43,13,0,3,1,1,1,0,0,213,43,13,0,3,1,1,2,0
DATA 0,228,43,13,0,3,1,1,3,0,0,243,43,13,0,3,1,1,4,0
DATA 52,201,256,8,0,3,1,2,5,0,52,209,256,8,0,3,1,0,6,0
DATA 52,217,256,8,0,3,1,0,7,0,52,225,256,8,0,3,1,0,8,0
DATA 52,233,256,8,0,3,1,0,9,0,52,246,256,8,0,3,4,3,0,1
DATA 317,198,28,13,4,3,1,13,0,0,317,228,28,13,4,3,1,14,0,0
DATA 317,243,28,13,0,3,1,6,13,0
DATA 416,201,40,8,0,2051,4,7,12,2
DATA 529,198,43,13,1,3,1,1,14,0
DATA 575,198,65,13,0,3,1,8,17,0,575,213,65,13,0,3,1,8,18,0
DATA 575,228,65,13,0,3,1,8,19,0,575,243,65,13,0,3,1,8,20,0
DATA 354,213,109,13,0,3,1,9,16,0
DATA 354,228,109,13,0,3,1,10,22,0
DATA 466,213,60,13,0,3,1,11,24,0
DATA 466,228,60,13,0,3,1,11,23,0
DATA 529,213,43,28,128,3,1,12,25,0
```

5.1.1 Arbeiten mit dem FileMonitor

Zunächst muß darauf aufmerksam gemacht werden, daß dieser Monitor sehr viel Chip-RAM benötigt - andere Programme sollten daher nicht gleichzeitig betrieben werden. Des weiteren wird gleich zu Anfang des Programms der eingestellte Druckertreiber geladen. Verfügen Sie noch nicht über einen solchen, ist das erste LPRINT wegzulassen.

Voreinstellungen: Für die integrierte Directory-Routine sind unter anderem vier Gadgets vorhanden, mit denen die am häufigsten benutzten Laufwerke per Anklicken eingestellt werden können. In diesem Falle handelt es sich um die Drives DF0 bis DF3. Sollten Sie andere Laufwerke in die Schnell-selektier-Gadgets eintragen wollen, ändern Sie in den DATA-Zeilen die entsprechenden Namen ab, wobei darauf zu achten ist, daß der Name inklusive des Doppelpunktes nicht länger als 4 Zeichen ist. Sie können auch vor dem Laden dieses Monitors mit ASSIGN den gewünschten Laufwerken (Beispiel: RAD) die Laufwerksbezeichnungen DF0 bis DF3 zuweisen.

Funktionen der Gadgets

Die vier Gadgets am linken Rand dienen dem schnellen Anwählen des Hauptverzeichnisses eines Laufwerkes. So genügt das Anklicken von "DF0:", um das aktuelle Directory auf das interne Laufwerk umzulegen, dieses einzulesen und zur Anzeige zu bringen.

Der große Kasten ist zur Anzeige von jeweils fünf Directory-Einträgen bestimmt. Hier werden Dateien und Programme in weiß und Directories in gelb angezeigt. Das Anklicken eines Directories führt zum sofortigen Auslesen des Directories und dessen Anzeige. Klickt man ein File an, wird der erste Datenblock der gewünschte Dateien zur Anzeige gebracht und kann editiert werden.

Das längliche Gadget unter dem großen Kasten ist ein String-Gadget, das der Anzeige des aktuellen Directories und des eventuell selektierten Files dient. Klickt man es an, erscheint ein

Cursor, mit dessen Hilfe man von Hand Pfade und Dateinamen angeben kann. Dies ist besonders empfehlenswert, wenn es sich um lange Pfade handelt oder auf ein nicht in den vier linken Gadgets angegebenes Laufwerk zugegriffen werden soll. Eingegebene Directories und Files werden ebenso schnell behandelt, wie das beim Anklicken eines Eintrages im großen Kasten der Fall ist.

Die Pfeile rechts neben dem großen Kasten sind für das Scrollen des Directories gedacht. Hier wird das angezeigte Directory um jeweils fünf Einträge gescrollt. Das OK-Feld hat die gleiche Funktion wie das Aktivieren und anschließende Desaktivieren des String-Gadgets: Der angegebene Eintrag wird entsprechend behandelt.

Die Anzeige "Block: #### of: ####" zeigt Ihnen bei einer angewählten Datei die aktuelle Datenblocknummer des angezeigten Blocks und die letzte Datenblocknummer der Datei an. Bei der ersten Zahl handelt es sich um ein Integer-Gadget, in das Sie nach dem Aktivieren durch Anklicken die gewünschte Datenblocknummer eingeben können, die zur sofortigen Anzeige des gewünschten Blockes führt.

Mit den beiden PRINT-Gadgets läßt sich entweder der Editorpuffer oder das gesamte File hexadezimal auf dem Drucker ausgeben. Danach wird automatisch der zuletzt editierte Block wieder zur Anzeige gebracht. Die Status-Anzeige zeigt alle Fehler und aktuellen Operationen an.

Mit den Gadgets "ASCII" und "HEX" können Sie den Editor von der hexadezimalen Anzeige auf die ASCII-Anzeige umschalten, was gerade beim Ändern von Texten, wie z.B. beim Eindeutschen der Menüs von AmigaBASIC eine sinnvolle Option ist. Das Quit-Gadget beendet, verbunden mit einer Sicherheitsabfrage, die über direkten Hardware-Zugriff gelöst wurde, dieses Programm.

Die Gadgets "READ", "NEXT" und "BACK" dienen dem nochmaligen Einlesen des aktuellen Blocks, dem Einlesen des näch-

sten und des vorhergehenden Datenblocks. Das WRITE-Gadget schreibt den Editor-Buffer auf die Diskette zurück. Hierbei wird aus Geschwindigkeitsgründen keine Sicherheitsabfrage gemacht. Sollte es einmal zu einem versehentlichen Abspeichern kommen, wählen Sie zunächst die UNDO-Funktion an und danach erneut das WRITE-Gadget.

Die UNDO-Funktion versetzt den Editor-Puffer (das sind die Daten, die angezeigt und editiert werden können) zurück in den Zustand, in dem sich der Editor-Puffer genau nach dem Einlesen des aktuellen Blocks befand. Im Klartext heißt das für Sie: Sobald Sie einen Block einlesen, wird dessen Inhalt in einen Undo-Puffer kopiert.

Der Editor akzeptiert neben allen Zeichen, die auf der Tastatur eingegeben werden können, auch die Cursor-Tasten. Grundsätzlich sind zwei Cursors zu sehen, einer im hexadezimalen und einer im ASCII-Anzeigefeld. Dies ist sinnvoll, da man dadurch immer genau weiß, welcher Hex-Code zu welchem angezeigten Zeichen gehört. Den aktuellen Editor-Modus erkennt man an der Breite des Cursors im Hex-Display: Ist dieser einen Nibble breit (ein Hex-Zeichen), befindet der Editor sich im Hexadezimal-Modus, bei doppelter Cursor-Breite im ASCII-Modus.

Das Programm ist Multitasking-fähig. Mit den bekannten Tastenkombinationen (linke Sondertaste + M oder +N) kann zwischen Monitor und Workbench umgeschaltet werden. Letzte Anmerkung: Es können jeweils nur 72 Einträge eines Directories eingelesen werden. Sollten Sie mehr wünschen, muß der Directory-Puffer entsprechend größer dimensioniert und die Abfrage im DIRECTORY-SUB angepaßt werden. Ansonsten kommen Sie auch durch direkte Eingabe an jedes File heran.

5.1.2 Patching, arbeiten mit dem FileMonitor

Unter Patching versteht man das Ändern vorhandener Programme durch das Manipulieren bestimmter Bytes eines abgespeicherten Files. So ist es möglich, jedes erreichbare Programm

seinen Wünschen entsprechend anzupassen. Eines sollten Sie dabei allerdings berücksichtigen: Das Ändern von Copyrights oder die Weitergabe gepatchter Programme verstößt gegen eine ganze Reihe von Gesetzen. Um sich nicht strafbar zu machen, sollten Sie daher nur für den eigenen Gebrauch patchen.

5.1.3 AmigaBASIC eindeutschen

Wie wäre es denn mit einem BASIC-Interpreter, der mit deutschen Menüs und deutschen Fehlermeldungen aufwarten kann? Zu diesem Zweck laden Sie den FileMonitor und klicken in dem großen Kasten AmigaBASIC an. Suchen Sie nun durch Anklicken des NEXT-Gadgets solange, bis Sie englische Texte oder gar die Menü-Texte gefunden haben. Fahren Sie mit dem Cursor auf den ersten Buchstaben des Textes.

Vorsicht: Um keine Programm-Daten zu zerstören, dürfen Sie die vorhandene Text-Länge nicht überschreiten. Zeichen, die zwischen zwei Wörtern stehen, dürfen nur dann überschrieben werden, wenn sie im Hex-Display mit dem Code 20 (=dezimal 32 =SPACE) angezeigt werden. Da die Übersetzung der Menüs unter Berücksichtigung der maximal möglichen Zeichen nicht gerade einfach ist, hier das deutsche Menü meines BASIC-Interpreters:

```
"Projekt"   "Edit "   "Los"          "Fenster"
"Neu    "   "Raus "   "Start    "    "Listing    "
"Öffnen "   "Kopie"   "Stop     "    "Ausgabe    "
"Sichern"   "Rein "   "Weiter   "
"    als"             "Abbruch  "
"Ende   "             "Trace an "
                      "Trace aus"
                      "Schritt  "
```

5.1.4 Andere Programme patchen

Die Anwendungsmöglichkeiten des Patching sind nahezu unbeschränkt. So können Sie beispielsweise das ED-Fenster auf volle PAL-Größe bringen, indem Sie die entsprechenden Größenan-

gaben im Programm ändern. Bevor Sie sich allerdings an die Veränderung von Programmen machen, sollten Sie sich genau informieren, ob es nicht einen einfacheren Weg gibt. Gerade die Betriebssystemversionen des Amiga und die Funktionen der einzelnen Tools machen uns deutlich, wie stark auf volle Kompatibilität zu vorhergehenden Versionen Wert gelegt wird.

So ist es schon vorgekommen, daß ein Programm als Utility zu einem anderen Programm (DPaint) herauskam, das mit einer gepatchten Version nicht mehr lauffähig war. Ein sehr schönes Beispiel dafür, wie es auch anders geht, ist das Shell-Icon der neuen Workbench V1.3. Wählen Sie hier im WB-Menü den Punkt Info an, sehen Sie ganz unten die Maßangaben für das zu öffnende Fenster, die Sie nun einfach Ihren Wünschen entsprechend ändern können.

5.2 Aufbau der AmigaBASIC-Files

Wie Sie sicherlich aus dem AmigaBASIC-Handbuch wissen, kann man hinter dem SAVE-Befehl angeben, wie man ein Programm abspeichern möchte. Es gibt drei verschiedene Möglichkeiten:

```
SAVE "Test",a
```

speichert das Programm als ASCII-File.

```
SAVE "Test",b
```

speichert das Programm normal ab.

```
SAVE "Test",p speichert das Programm geschützt ab.
```

Bevor Sie ein Programm abspeichern, sollten Sie sich darüber im klaren sein, was Sie später mit diesem File vorhaben. Dazu müssen Sie natürlich wissen, wozu man in dieser oder jener Situation eine bestimmte Art eines Files braucht.

Beginnen wir mit dem ASCII-File. Sie benötigen ASCII-Files, um zwei Programme durch den Befehl MERGE oder CHAIN

MERGE zu verbinden. Wenn Sie ein Programm als ASCII-File gespeichert haben, können Sie es später (z.B. nach erneutem Laden) immer wieder als ASCII-, Binär- oder Protected-File speichern.

Der Nachteil von ASCII-Files (und von modularem Programmieren überhaupt) ist der große Speicherplatzbedarf. Das tritt besonders bei sehr langen und häufig verwendeten Variablennamen auf. Doch dazu später mehr. Das Binär-File ist kurz, Befehle und Variablen sind in Token übersetzt. Auch ein binäres File kann jederzeit als ASCII-, Binär- oder Protected-File gespeichert werden.

Sollten Sie einmal ein Programm "protected" gespeichert haben, entdeckten dann eine Kleinigkeit, die Sie schnell noch korrigieren wollten, so haben Sie sich wahrscheinlich erstmal tüchtig die Haare gerauft. Im Gegensatz zu anderen Computern hält das Wort protected beim Amiga, was es verspricht: Was man einmal geschützt gespeichert hat, sieht man garantiert nicht wieder. Deshalb empfiehlt es sich, vorher eine Sicherheitskopie des Programms anzufertigen.

5.2.1 Typ feststellen

Erinnern wir uns daran, was wir eigentlich vorhaben. Wir wollen AmigaBASIC-Programme manipulieren, ob sie bereits auf Diskette vorliegen und ob es sich um ein BASIC-Programm handelt.oder Zwischenspeicherung auf sich selbst einwirken sollen. Sobald Sie den Aufbau von AmigaBASIC-Files kennen, sollte das auch keine allzugroßen Schwierigkeiten mehr bereiten.

Ein Problem stellt sich aber: Stellen Sie sich einmal vor, Sie haben ein Programm geschrieben, das in der Lage ist, aus einem diskettenresidenten Programm ein neues AmigaBASIC-Programm zu generieren. Dieses Programm erwartet eine Auswahl der User, welches Programm sie modifizieren möchten. Nachdem wir nun festgestellt haben, daß das Programm tatsächlich auf der

eingelegten Diskette vorhanden ist, muß sich dem Programmierer geradezu die Frage aufdrängen: Ist das denn tatsächlich ein AmigaBASIC-File?

5.2.1.1 BASIC-Check

Um Sie nicht völlig im Dunkeln tappen zu lassen, hier eine Routine, die das nachprüft:

```
goto start
REM ####################################
REM #  B  A  S  I  C  -  C  H  E  C  K   #
REM #------------------------------------#
REM #     (W) 1987 by Stefan Maelger     #
REM ####################################
REM SUB-Routine zum Prüfen, ob ein File
REM ein AmigaBASIC-Programm ist
start:
    DECLARE FUNCTION xOpen& LIBRARY
    DECLARE FUNCTION xRead% LIBRARY
    DECLARE FUNCTION Seek%  LIBRARY
    LIBRARY "dos.library"
main:
    CLS
    LOCATE 2,2
    PRINT "Name des AmigaBASIC-Programms:"
    LOCATE 4,1
    PRINT ">";:LINE INPUT Filename$
    BASICcheck Filename$,Flag%
    LOCATE 6,2
    IF Flag% THEN
        PRINT "TATSACHE! Ein AmigaBASIC-Programm"
    ELSE
        PRINT "Leider kein AmigaBASIC-Programm..."
    END IF
    LIBRARY CLOSE
    END
SUB BASICcheck (Filename$,ok%) STATIC
    File$ = Filename$+".info"+CHR$(0)
    Default.Tool$ = SPACE$(12)
    OpenOldFile% = 1005
    OffsetEOF% = 1
    Offset% = -12
OpenFile:
    File.handle& = xOpen&(SADD(File$),OpenOldFile%)
    IF File.handle& = 0 THEN
        CLS
        LOCATE 2,2
```

```
        PRINT "Ich finde ";Filename$;" nicht!"
        BEEP
        EXIT SUB
    ELSE
        OldPosition%=Seek%(File.handle&,Offset%,OffsetEOF%)
        GotThem%=xRead%(File.handle&,SADD(Default.Tool$),12)
        IF GotThem%<12 THEN
            CLS
            LOCATE 2,2
            PRINT "READ-ERROR"
            BEEP
            EXIT SUB
        ELSE
            IF INSTR(Default.Tool$,":AmigaBASIC")>0 THEN
                ok%=-1
            ELSE
                ok%=0
            END IF
        END IF
        CALL xClose(File.handle&)
    END IF
END SUB
```

Variablen

Filename$	Name des vermeintlichen AmigaBASIC-Programms.
Flag%	=-1: Das File ist ein AmigaBASIC-Programm.
ok%	Bezeichnung der SUB-Variablen von Flag%
File$	Name des ".info"-Files von Filename$ + CHR$(0)
Default.Tool$	12-Byte-String, der die letzten 12 Byte von File$ aufnimmt.
OpenOldFile%	Angabe darüber, daß ein vorhandenes File geöffnet werden soll (1006 = neues File öffnen).
OffsetEOF%	Cursor der File-Leseroutine auf das Ende des Files setzen (-1 = Anfang, 0 = derzeitige Position).
Offset%	Wert, um den der File-Cursor von OffsetEOF% an verschoben wird.
File.handle&	Adresse des File-Handlers (0 = File wurde nicht geöffnet).
OldPosition%	Alter Offset des File-Cursors.
GotThem%	Anzahl tatsächlich gelesener Bytes.

Zum Programm

Haben Sie schon einmal im Workbench-Menü den Punkt Info angewählt? Wenn ja, dann fiel Ihnen sicherlich der Punkt Default-Tool auf. Das Default-Tool ist das Programm, das beim Anklicken eines Icons als erstes gestartet wird. So ist es dann auch nicht weiter verwunderlich, daß bei AmigaBASIC-Pro-

grammen dort der Eintrag ":AmigaBASIC" zu finden ist. Fraglich ist nun natürlich, woher diese Information kommt. Zu jedem Programm (zumindest zu jedem AmigaBASIC-Programm) existiert ein File, das den gleichen Namen wie das Programm selbst trägt, allerdings mit dem Zusatz .info. Im wesentlichen besteht ein solches .info-File aus der Bitmap für das Icon und am Ende dem Default-Tool.

Da wir nun wissen, wo sich die Information versteckt hält, ob es sich um ein AmigaBASIC-Programm handelt oder nicht, brauchen wir natürlich nur noch das zugehörige .info-File zu öffnen, den Lese-Cursor auf das File-Ende zu setzen und mit einem Offset von -12 Bytes ab dieser Position die 12 Byte des Default-Tools einzulesen. Warum 12 Byte? Nun, der Eintrag selbst scheint zwar nur 11 Byte zu haben, doch muß man wissen, daß das AmigaDOS nur Namen akzeptiert, die mit CHR$(0) abgeschlossen sind. Daher das 12. Byte.

Kleiner Tip am Rande: Einige Programme, die Icons manipulieren oder neue Icons schaffen, sind nicht ganz korrekt programmiert. Dieser kleine Programmierfehler kann dazu führen, daß unser "hochgeschätztes" Default-Tool verschoben wird. Im Ernstfall können Sie diesen Fehler ausschalten, indem Sie einfach die Anzahl der einzulesenden Bytes erhöhen (auch vom String!).

5.2.1.2 HeaderCheck - Wie wurde das Programm gespeichert?

Jetzt wissen wir also, wie wir feststellen können, ob es sich bei einem File um ein AmigaBASIC-Programm handelt oder nicht. Eine solche Routine sollte nie in einem Programm fehlen, das ein anderes AmigaBASIC-Programm verändern kann. Als nächstes sollten wir feststellen, um was für einen Programm-Typ es sich bei dem angepeilten Programm handelt. Um das erkennen zu können, ist es wichtig zu wissen, wie der AmigaBASIC-Interpreter die verschiedenen Programme unterscheidet.

Wir kommen dabei bereits zum ersten Byte eines AmigaBASIC-Programms, dem Header-Byte. Das Header-Byte zeigt dem AmigaBASIC-Interpreter, um was für ein Programm es sich handelt. Und das geht folgendermaßen. Bei binär gespeicherten Programmen, und dazu zählt auch ein protected gespeichertes Programm, ist ein Byte vor das File gehängt worden, das sogenannte Header-Byte. Sicherlich werden Sie jetzt nach den ASCII-Files fragen. Tatsächlich ist bei ASCII-Files kein Header-Byte vorhanden! Beachten Sie das bitte beim Programmieren. Wieso das so ist? Nun, bei ASCII-Files ist kein Header-Byte nötig. Warum das so ist, werden Sie spätestens dann merken, wenn Sie den Aufbau von ASCII-Files kennen. Es ist nämlich kaum möglich, daß ausgerechnet die Werte der Header-Bytes für binäre Programme am File-Anfang auftreten. Merken Sie sich am besten folgendes:

- Liegt am File-Anfang das Byte $F5 (=245 dezimal), handelt es sich um ein normal gespeichertes Programm, ein Binär-Programm.
- Finden Sie dagegen das Byte $F4 (=244 dezimal), ist es ein protected gespeichertes Binär-Programm.
- Liegt am File-Anfang weder $F5 noch $ F4 vor, so handelt es sich um ein Binär-Files.

Wie immer, haben wir auch hier eine kleine Routine, die Ihnen die Überprüfung abnimmt. Da Manipulationsprogramme wohl kaum ohne die dos.library-Routinen xRead und xWrite auskommen werden, wurden diese Routinen verwendet. Beachten Sie jedoch, daß die folgende Routine nur dann funktionieren kann, wenn ein AmigaBASIC-Programm vorliegt.

```
GOTO start
' ####################################
' # H E A D E R - C H E C K #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger    #
' ####################################
'
' SUB-Routine zum Auslesen des File-
' Headers eines AmigaBASIC-Programms, um
' den File-Typ zu bestimmen.
```

```
' ------------------------------------
'
start:
  DECLARE FUNCTION xOpen& LIBRARY
  DECLARE FUNCTION xRead% LIBRARY
  LIBRARY "dos.library"
main:
  ProgrammTyp$(0)="ASCII-File"
  ProgrammTyp$(1)="Binär-File"
  ProgrammTyp$(2)="Protected-Binär-File"
  LINE INPUT "Filename: >";Filename$
  HeaderCheck Filename$,Ergebnis%
  LOCATE 10,1
  PRINT "Das Programm ";CHR$(34);
  PRINT Filename$;CHR$(34);
  PRINT " ist ein ";ProgrammTyp$(Ergebnis%)
  LOCATE 15,1
  LIBRARY CLOSE
  END
SUB HeaderCheck(Filename$,Ergebnis%) STATIC
    File$=Filename$+CHR$(0)
    OpenOldFile%=1005
    handle&=xOpen&(SADD(File$),OpenOldFile%)
    IF handle&=0 THEN ERROR 53
    s$=" "
    Byte&=1
    gelesen&=xRead%(handle&,SADD(s$),Byte&)
    CALL xClose(handle&)
    Ergebnis%=0
    d%=ASC(s$)
    IF d%=&HF5 THEN
      Ergebnis%=1
    ELSEIF d%=&HF4 THEN
      Ergebnis%=2
    END IF
END SUB
```

Variablen

ProgrammTyp$()	die Programmarten im Klartext
Filename$	Name des zu prüfenden AmigaBASIC-Programms
Ergebnis%	0=ASCII; 1=binär; 2=protected
File$	Filename$ + abschließendes CHR$(0) für DOS
OpenOldFile%	ein bereits existierendes File öffnen
handle&	Adresse des File-Handlers
s$	String, in den das erste Byte gelesen wird
Byte&	Anzahl einzulesender Bytes
gelesen&	Anzahl tatsächlich gelesener Bytes
d%	ASCII-Wert von s$

5.2.2 ASCII-Files

Der Aufbau von ASCII-Files ist denkbar einfach. Nehmen Sie sich einmal die Zeit, ein eingegebenes Programm mit der folgenden Zeile abzuspeichern:

```
SAVE "Test",A
```

Beispielprogramm:

```
a=1
PRINT a
```

Laden Sie nun den File-Monitor, der in Kapitel 5.1 beschrieben wurde, oder einen beliebigen anderen File-Monitor. Laden Sie dieses ASCII-File und sehen Sie sich die Daten an. Wenn man die rechte Seite der Ausgabe betrachtet, so sieht man das Programm im Klartext. So würde dort beispielsweise stehen:

```
a=1.PRINT a..
```

und als Hex-Dump:

```
61 3D 31 0A  50 52 49 4E  54 20 61 0A  0A (hexadezimal)
```

Rechnen wir diese hexadezimalen Zahlen um, erhalten wir:

```
97 61 49 10  80 82 73 78  84 32 97 10  10 (dezimal)
```

Schlagen Sie nun im AmigaBASIC-Handbuch die ASCII-Zeichencode-Tabellen auf. Sie sehen, das Programm wurde im Klartext gespeichert. LF (10) heißt im übrigen LINE FEED, also auf gut deutsch: nächste Zeile. Wollen Sie von einem Programm aus ein ASCII-File ausdrucken, genügen daher folgende Zeilen:

```
LINE INPUT file$
OPEN file$ FOR INPUT AS 1
  WHILE NOT EOF(1)
    PRINT INPUT$(1,1);
  WEND
CLOSE 1
```

Toll, nicht wahr? Nehmen Sie sich jetzt die Workbench zur Hand, laden Sie das Shell, sofern Sie Version 1.3 besitzen, oder das CLI (System-Schublade). Nun geben Sie ein:

```
ed Diskname:Test
```

Für "Diskname" geben Sie den Namen der Diskette ein, auf der sich das Testprogramm befindet. Jetzt sollte das Programm auf dem Bildschirm erscheinen. Man kann also den komfortablen ED-Editor zum Editieren von ASCII-Programmen verwenden. Der einzige Nachteil dabei ist, daß man die Programme nicht gleich austesten kann. Da man sie vorher jedoch in jedem Fall speichern sollte - und schließlich einen Multitasking-Computer besitzt - hat man eine wirklich gute Möglichkeit, an einen neuen Programm-Editor zu kommen. Doch das nur am Rande.

Sollten Sie jetzt auf die Idee kommen, einfach per OPEN-FOR-OUTPUT-Anweisung ein neues Programm zu erzeugen, so ist das sicherlich kein schlechter Einfall. Die Tücke des Objektes ist hierbei jedoch, daß Sie Schwierigkeiten haben können, wenn Sie Ihr neues Programm später per Anklicken aus dem Directory laden wollen. Das File Name.info hat schließlich nicht ":AmigaBASIC" als Default-Tool. Um auf jeden Fall vorbereitet zu sein, sollte man einfach ein neues .info-File erzeugen:

```
SAVE "Dummy": KILL file$+".info"
NAME "Dummy.info" AS file$+".info"
KILL "Dummy"
```

Anwendungen zum Thema ASCII-Files finden Sie unter Kapitel 5.3.

5.2.3 Binär-Files

Wir werden jetzt ausführlich den Aufbau von binär gespeicherten Programmen beschreiben. Der Aufbau dieser Files ist besonders wichtig, da er die einzige Form des Programmaufbaus darstellt, die der AmigaBASIC-Interpreter direkt abarbeiten kann. Alle anderen File-Arten müssen also zunächst in das binäre

Format überführt werden, bevor sie ausgeführt werden können. Die Bedeutung des ersten Bytes eines binären Programms kennen Sie bereits: das Header-Byte ($F5 für Binär-Files).

5.2.3.1 Aufbau einer AmigaBASIC-Zeile

Mit dem zweiten Byte des Programms beginnt bereits die erste Programmzeile. Wir sollten daher den Aufbau einer Zeile betrachten. Das erste Byte einer Zeile ist der Zeilen-Header. Dieses Byte kann zwei Werte haben, den Wert Null oder den Wert 128 ($80). Beginnt eine Zeile mit dem Wert 0, handelt es sich um eine Zeile ohne Zeilennummer, beginnt sie mit dem Wert 128, ist es eine Zeile mit Zeilennummer. Label sind in diesem Zusammenhang nicht von Bedeutung. Wir werden sie später behandeln.

Das zweite Byte einer Zeile ist der Offset zur nächsten Zeile. Da ein AmigaBASIC-Programm nach jedem Laden an einer anderen Speicherstelle beginnen kann, wäre es etwas zu aufwendig, würde man mit Pointern auf die nächste Zeile arbeiten. Es reicht doch völlig aus, wenn an einer festgelegten Stelle einer Zeile die Gesamtlänge derselben angegeben ist. Der Interpreter braucht sich nur die Adresse zu merken, an der die Zeile beginnt, und dazu die Nummer des gerade bearbeiteten Bytes. Muß sich - beispielsweise durch einen Sprungbefehl - der Interpreter viele Zeilen vorwärts bewegen, wird jedesmal nur die Zeilenlänge der aktuellen Zeile zu der Anfangsadresse derselben addiert.

Hier wird auch ersichtlich, weshalb eine Programmzeile nicht länger als 255 Byte sein kann. Schließlich steht nur ein Byte für die Zeilenlänge zur Verfügung. Sie werden sicherlich Ihre eigenen Programme sehr übersichtlich eingeben, was eine eventuelle

Fehlersuche oder das Verständnis des Programmablaufs positiv unterstützt. Dann sehen Ihre Programme wahrscheinlich ähnlich aus wie dieses:

```
Viele.FOR.NEXT.Schleifen:                                    0
                                                             0
  FOR ersteSchleife=1 TO 100                                 2
    FOR zweiteSchleife=1 TO 10                               4
      FOR dritteSchleife=1 TO 50                             6
        LPRINT FNstefan (x,y,z)                              8
  NEXT dritteSchleife,zweiteSchleife,ersteSchleife           2
```

Die Zahlen auf der rechten Seite gehören nicht zum Programm. Ich habe mit diesen Zahlen den Wert angegeben, den das dritte Byte der jeweiligen Programmzeile annimmt. Prüfen Sie dieses ruhig mit einem File-Monitor nach. Das dritte Byte wird nur für den Befehl LIST und für Editierzwecke benötigt. Es gibt den Abstand des ersten Befehls zum linken Rand an. Damit ist jetzt die Frage geklärt, ob es die Programmlänge oder die Programm-Ablaufgeschwindigkeit beeinflußt, wenn man wie oben die Programmzeilen bei Verschachtelungen einrückt. Sie sehen: Die einzige Veränderung ist der Wert des dritten Byte. Programmieren Sie also ruhig weiter übersichtlich.

Bis zu dieser Stelle ist der Aufbau einer Zeile mit Zeilennummer genauso wie der Aufbau einer Zeile ohne Zeilennummer. An dieser Stelle haben wir den Kopf einer Zeile ohne Zeilennummer bereits vollständig behandelt. Daher noch einmal zur Übersicht:

Kopf einer Zeile ohne Zeilennummer:

Byte-Nr.	Wert	Bedeutung
1	00	Es folgt eine Zeile ohne Zeilennummer.
2	xx	Länge der Zeile in Byte (mit Kopf und Ende).
3	xx	Abstand des ersten Befehls vom linken Rand (nur für LISTen des Programms).

Bei Zeilen mit Zeilennummer folgen nun weitere zwei Bytes. Der Zeilenkopf einer solchen Zeile ist also fünf Byte lang. Jetzt folgt noch die Zeilennummer. Sie wird im High-Low-Format

angegeben. Ist die Zeilennummer beispielsweise 10000, so folgen nun die Bytes $27 und $10 (39 und 16 dezimal, also 39*256 + 16 = Zeilennummer). Auch hier noch einmal eine Übersicht:

Kopf einer Zeile mit Zeilennummer:

Byte-Nr.	Wert	Bedeutung
1	128	Es folgt eine Zeile mit Zeilennummer.
2	xx	Länge der Zeile in Byte (mit Kopf und Ende).
3	xx	Abstand des ersten Befehls vom linken Rand.
4	xx	High-Byte der Zeilennummer.
5	xx	Low-Byte der Zeilennummer.

Jetzt ist der Aufbau der Zeile wieder bei beiden Zeilentypen gleich. Es folgen die Token, das heißt, die zu ein oder zwei Byte kodierten Befehle. Was mit den Labeln ist? Geduld, Geduld, deren Kodierung kommt noch. Abgeschlossen wird jede BASIC-Zeile mit dem Wert Null, also mit einem weiteren Byte. Zusammenfassend können wir also feststellen: Eine Programmzeile besteht aus:

- Zeilenkopf mit oder ohne Zeilennummer
- Token (Befehle, Label, Variablen und Werte)
- Ende-Byte mit dem Wert 0

5.2.3.2 Leerzeilen

Mit dem Wissen, das Sie jetzt haben, können Sie bereits angeben, wie Leerzeilen gespeichert sind. Unter Leerzeilen verstehe ich Zeilen, in denen weder ein Befehl noch eine Zeilennummer auftritt. Sehen wir uns das Problem gemeinsam an:

- Das erste Byte, also das Zeilentyp-Byte, muß den Wert Null haben, da keine Zeilennummer folgt.
- Das dritte Byte, also der Abstand vom linken Rand, ist meist auch Null.
- Mit dem vierten Byte müßten die Token folgen. Da diese Zeile jedoch leer ist, folgt nun der Zeilenende-Code, nämlich ein Null-Byte.

Damit wäre unsere Zeile abgeschlossen, und wir können in dem zweiten Byte die Zeilenlänge mit vier Byte angeben. Eine Leerzeile sieht also folgendermaßen aus:

```
$00 - $04 - $xx - $00
```

Wie hieraus ersichtlich ist, verlängert jede leere Zeile nicht nur das Programm um vier Byte, sondern beeinflußt auch die Ablaufgeschwindigkeit des Programms nachteilig, da der Interpreter jedesmal diese Zeile nach Befehlen absuchen und den Anfang der nächsten Zeile errechnen muß. Sie sollten daher Leerzeilen aus Ihren Programmen entfernen, auch wenn dadurch die Laufgeschindigkeit nur unerheblich erhöht wird. Denn wie der Bauer schon sagt: Kleinvieh macht auch Mist. Ein Programm, das diese Arbeit für Sie erledigt, finden Sie im Kapitel 5.3.

5.2.3.3 Die letzte Zeile

Da jedes Programm einmal ein Ende haben muß, folgt auf die letzte Zeile des Programms zunächst ein Null-Byte zur Anzeige, daß keine Zeilennummer folgt. Darauf folgt das Zeilenlänge-Byte, das ebenfalls auf Null gesetzt ist, des weiteren ein Zeilenende-Code, also noch eine Null. Es kommt vor, daß weitere Bytes folgen, je nachdem, wie das Programm editiert wurde. Diese Bytes können die "wildesten" Werte annehmen (!). Vorsicht ist also geboten, wollen Sie die Variablentabelle einlesen.

5.2.3.4 Die Variablentabelle

Variablen-Namen können in AmigaBASIC sehr lang sein, z.B.:

```
Anzahl.der.eingelesenen.Bytes%
```

Es wäre geradezu haarsträubend, würde man bei jedem Auftauchen einer Variablen ihren vollen Namen abspeichern. Um derartig lange Variablen-Namen verwalten zu können, ohne die

Ablaufgeschwindigkeit von AmigaBASIC unendlich träge werden zu lassen, mußten sich die Programmierer dieses BASIC-Dialektes also etwas einfallen lassen:

Tritt in einem Programm eine Variable auf, so erkennt der Interpreter das an einem speziellen Token. Dieses Token hat immer den Wert $01. Auf dieses Token folgt eine Nummer im altbekannten High-Low-Format. Richtig, Sie haben es erraten. Der Interpreter numeriert einfach alle Variablen der Reihe nach durch und benutzt beim Programm-Ablauf folglich nur noch Variablennummern. Um beim Auflisten des Programms wieder die vollen Variablennamen ausgeben zu können, müssen diese natürlich irgendwo gespeichert werden. Das geschieht am Programm-Ende in Form einer Variablentabelle. Ein Eintrag in dieser Tabelle hat folgendes Format:

```
1. Byte: Länge des Variablen-Namen in Byte.
folgende Bytes: Variablen-Name in ASCII-Code.
```

Verwenden Sie zum Beispiel in Ihrem Programm die Variablen a%, Zeichenkette$ und Adresse&, so würde die Variablentabelle folgendermaßen aussehen:

Hexadezimal ASCII

```
01 61 .a
0C 5A 65 69 63 68 65 6E 6B 65 74 74 65 .Zeichenkette
07 41 64 72 65 73 73 65 .Adresse
```

Hiermit würde das letzte Byte Ihres Programms das Byte $65 sein. Wie Sie aber auch bemerkt haben dürften, steht in der Variablentabelle nicht, ob es sich bei der Variablen um eine String-Variable, eine der Floating-Point-Variablen oder eine der Integer-Variablen handelt. Diese Angabe finden Sie - sofern sie im Listing aufgeführt wird - hinter der Variablen-Nummer, die dem Token $01 folgt. Bleiben wir bei dem obigen Beispiel, so würde die Variable a% in folgender Form im Programm stehen:

Byte-Nr.	Wert	Bedeutung
1	1	Variablennummer folgt
2	0	High-Byte der Variablennummer
3	0	Low-Byte der Variablennummer
4	37	ASCII-Code des Zeichens "%"

Sie sehen des weiteren an dieser Tabelle, daß die erste vorkommende Variable die Nummer Null bekommt. Leider haben die Programmierer von AmigaBASIC die Geschichte mit den Variablen noch etwas kompliziert. So ist es zwar richtig, wenn ich oben gesagt habe, daß die Variablen der Reihe nach durchnumeriert werden, ich habe Ihnen allerdings verschwiegen, daß mit der "Reihe" die Reihenfolge gemeint ist, in der Sie die Variablen beim Eintippen des Programms eingeben! Wenn Ihnen jetzt Böses "schwant", haben Sie damit leider recht. Damit Sie das ganz große Problem beziehungsweise den großen "Bug" erkennen, gehen Sie bitte wie folgt vor:

1. Laden Sie AmigaBASIC.
2. Geben Sie folgendes ein:

```
Der.grosse.Fehler%=0
Blablabla%=Der.Fehler%
Hallo%=0
```

3. Ändern Sie die Zeile Blablabla%=... in:

```
Blablabla%=Der.grosse.Fehler%
```

4. Speichern Sie das Programm binär ab, und sehen Sie es sich mit einem File-Monitor an.

Im Programm taucht die Variable Der.Fehler% überhaupt nicht mehr auf. In der Variablentabelle wird sie aber aufgeführt und auch abgespeichert! Sollten Sie einmal ein langes Programm schreiben, das Sie bei der folgenden Fehlersuche häufig ändern müssen, oder haben Sie etwa einen Tippfehler bei einem BASIC-Befehl gemacht (dann wird er auch sofort in die Variablentabelle übernommen), kann es durchaus sein, daß Ihr Programm infolge von Variablenmüll gleich etliche KByte länger ist. Über die Herabsetzung der Ablaufgeschwindigkeit wollen wir hier gar nicht erst reden...

Wie Sie diesen Fehler ausmerzen können, sehen Sie in dem Kapitel 5.3.6. Eine weitere Tücke, die die Programmierer von AmigaBASIC eingebaut haben, ist die Tatsache, daß alle Subprogramm-Namen, ihre Aufrufe und auch alle Betriebssystemroutinen, die Sie mit LIBRARY und/oder DECLARE FUNCTION eingebunden haben, als Variablen abgelegt werden - ebenso in der Tabelle wie im Programm-Text selbst.

AmigaBASIC kann diese Namen nur nach einer vollständigen Syntax-Überprüfung als Funktionen oder SUB-Erweiterungen erkennen! Das macht dem BASIC-Interpreter nicht viel aus, da er nach dem Laden des Programms sowieso zuallererst eine vollständige Prüfung des Programms vornimmt und daher diese Arbeit gleich mit erledigen kann. Deshalb dauert es auch immer erst ein wenig, bis ein geladenes Programm endlich startet.

5.2.3.5 Labelhandling

Bis jetzt war alles schön und gut, doch wo bleiben die Label, werden Sie sich fragen. Nun, Label werden ganz ähnlich verwaltet wie Variablen. Auch hier ergab sich für die Entwickler von AmigaBASIC das Problem mit den langen Namen, die Label annehmen können. Die Lösung, die sie fanden, sieht folgendermaßen aus: Label sind spezielle Variablen, die sich nur dadurch von anderen Variablen unterscheiden, daß sie angesprungen werden können.

Das bedeutet nun, daß ein im Programm auftauchendes Label ebenso in die Variablentabelle einsortiert wird wie eine normale Variable. Nun muß der BASIC-Interpreter nur noch erkennen können, daß es sich bei einer solchen Variablen um ein Label handelt, für das kein Speicherplatz angelegt werden muß. Dies wird ganz einfach durch ein anderes Spezial-Token gelöst, durch Verwendung von $02 anstelle von $01 im Programm. Trifft der Interpreter also auf das Token $02, so ist die folgende Nummer im High-Low-Format die Nummer eines Labels. Ein Beispiel:

Byte-Nr.	Wert	Bedeutung
1	2	Label-Nummer folgt
2	xx	High-Byte der Label-Nummer
3	xx	Low-Byte der Label-Nummer

Fände der Interpreter $02 $00 $09 im Programm, so wüßte er, daß es sich um ein Label handelt, dessen Name an zehnter Stelle in der Variablentabelle zu finden ist (Sie erinnern sich: Variablen werden von Null an durchnumeriert).

5.2.3.6 Label anspringen

Jetzt soll so ein Label natürlich irgendwie angesprungen werden können, ansonsten wäre es ebenso überflüssig wie ein REMark. Ich werde Ihnen das Verfahren anhand des BASIC-Befehls GOTO erklären, obwohl es selbstverständlich ebenso für GOSUB gilt. Beispiel:

```
GOTO division
```

Nehmen wir an, daß "division" an 3. Stelle in der Variablentabelle steht, so fände der Interpreter im Programm folgendes vor:

Byte-Nr.	Wert	Bedeutung
1	151	Token für GOTO (siehe Anhang
2	32	Space (wird mit abgespeichert!)
3	3	Token = ein Label soll angesprungen werden
4	0	immer Null
5	0	High-Byte der Nummer in der Variablentabelle
6	2	Low-Byte der Nummer

Sie sehen, wir haben ein neues Token kennengelernt - $03. Der Interpreter braucht sich nun nur noch ein passendes $02-$00-$02 zu suchen und an dieser Stelle mit der Programm-Abarbeitung fortfahren.

5.2.3.7 Zeilennummern anspringen

Das eben gezeigte Verfahren läßt sich natürlich nicht auf Zeilennummern anwenden, da diese nicht in der Variablentabelle stehen. Ein neues Token muß also her. Beispiel:

```
GOTO 10000
```

Byte-Nr.	Wert	Bedeutung
1	151	Token von GOTO (siehe Anhang)
2	32	Space
3	14	Token = springe zu folgender Zeilennummer
4	0	immer Null
5	39	High-Byte der Zeilennummer (39*256)
6	16	Low-Byte der Zeilennummer (+16 = 10000)

Das Token $0E besagt also, daß in allen Zeilen, deren Header-Byte $80 ist, die Bytes 4 und 5 mit den obigen Bytes 5 und 6 verglichen werden müssen, um das Sprungziel zu finden.

5.2.3.8 Werte in AmigaBASIC-Programmen

Wir kommen nun zu der Ablage von Werten in AmigaBASIC-Programmen. Sie geben beispielsweise ein:

```
Amiga=1
```

Uns soll an dieser Stelle interessieren, in welcher Form die "1" im Programm gespeichert wird. Entgegen der Methode anderer BASIC-Dialekte, in denen Zahlen ganz einfach durch ihren ASCII-Code vertreten sind, und weil während des Programmablaufs eine ständige Umrechnerei Zeit kostet, wird bei AmigaBASIC die Zahl bzw. der Wert gleich in dem wahrscheinlich benötigten Format abgelegt. Für jedes Format, beispielsweise Floating-Point- oder Oktal-Zahlen, muß nun natürlich ein neues Token her. Keine Angst, wir werden uns das ganze Verfahren Schritt für Schritt ansehen.

Dummerweise ist der Entscheidungsprozeß, ob dieses oder jenes Format gewählt wird, nicht davon abhängig, was für ein Format die Variable, der der Wert zugewiesen werden soll, schlußendlich benötigt. Aber warum sollte man BASIC-Programme auch beschleunigen?!?

Sehen wir uns obiges Beispiel noch einmal an. Bei der Zahl 1 handelt es sich unzweifelhaft um eine Integer-Zahl. Der nächste wichtige Punkt ist der, daß es ein einstelliger Wert ist. Wenn jetzt noch dazukommt, daß der Wert positiv ist, erfährt er eine Sonderbehandlung:

Ganzzahlige positive Werte im Bereich von 0 bis 9 werden ohne Token im Programm gespeichert. Dabei wird nicht der ASCII-Code verwendet! Auch ein direktes Speichern des Wertes kommt nicht in Frage (da "0" beispielsweise "Zeilenende" bedeutet und "1" "Variablennummer folgt" etc.). Vielmehr werden die Werte wie folgt kodiert:

Hex.	dez.	Wert (dez.)
$11	17	0
$12	18	1
$13	19	2
...	...	...
$19	25	8
$1A	26	9

Findet der Interpreter folglich ein Byte zwischen 17 und 26, so zieht er lediglich den Wert 17 ab und erhält so den richtigen Wert. Bleiben wir zunächst bei positiven Integer-Werten. Liegen diese im Bereich zwischen 10 und 255, so genügt ein Byte für das Abspeichern. Hier benötigen wir wieder ein Token, an dem der Interpreter erkennt, daß das folgende Byte nicht etwa ein Befehls-Token oder ein anderes Token ist. Das Format ist:

Byte-Nr.	Wert	Bedeutung
1	15	Es folgt ein positiver Int-Wert zwischen 10 und 255.
2	xx	Wert zwischen 10 und 255.

Integer-Werte können nun natürlich noch größer werden oder vorzeichenbehaftet sein. Dann wird dieses Format benutzt:

Byte-Nr.	Wert	Bedeutung
1	28	Es folgt ein vorzeichenbehafteter 2-Byte-Integerwert.
2	xx	High-Byte (Bit 7 = Vorzeichenbit)
3	xx	Low-Byte

Integer-Werte, die sich hiermit nicht mehr darstellen lassen, z.B.: Werte größer als 32767, werden im Long-Integer-Format dargestellt:

Byte-Nr.	Wert	Bedeutung
1	30	Token: vorzeichenbehafteter 4-Byte-Integer-Wert folgt.
2-5	xx	4-Byte-Integer-Zahl, Bit 7 in Byte 2 ist das Vorzeichen-Bit.

Sollte es sich bei dem Wert um eine Fließkommazahl handeln, wird folgendes Format verwendet:

Byte-Nr.	Wert	Bedeutung
1	29	Token: 4-Byte-Fließkommazahl folgt.
2-5	xx	4-Byte-Float (Genauigkeit = 7 Stellen).

Nun das Ganze mit doppeltgenauen Fließkommazahlen:

Byte-Nr.	Wert	Bedeutung
1	31	Token: 8-Byte-Floating-Point folgt.
2-9	xx	8-Byte-Float (Genauigkeit = 16 Stellen).

Sollten Sie der Auffassung sein, daß das alles war, haben Sie sich getäuscht. Geben Sie doch einmal in einem Programm ein:

```
a=&hff
```

Nach dem Verlassen dieser Zeile korrigiert der Amiga gleich:

```
a=&HFF
```

Daran können wir schon sehen, daß der Amiga auch diese Werte erkennen muß. Aber sehen Sie selbst:

Byte-Nr.	Wert	Bedeutung
1	12	Token: Hexadezimalzahl folgt
2	xx	High-Byte
3	xx	Low-Byte

Dann sind da noch die Oktalzahlen wie &O123456. Sie werden zunächst ins 2-Byte-Format umgerechnet und treten dann so auf:

Byte-Nr.	Wert	Bedeutung
1	11	Token: Oktalzahl folgt
2+3	xx	Oktalzahl (6 Stellen genau)

In Zusammenhang mit Werten über Strings zu reden erscheint vielleicht etwas merkwürdig, trotzdem an dieser Stelle eine Bemerkung darüber: Strings werden im ASCII-Klartext gespeichert. Um Speicherplatz zu sparen, wird bei einer direkten Wertzuweisung kein neuer Speicherplatz reserviert, in die dann der String aus dem Programm übertragen wird, sondern es werden einfach die Zeiger, die auf die Anfangsadresse der Strings weisen, auf den Klartext im Programm gesetzt. Das läßt sich sofort beweisen:

```
a$="----------------------------"
b$="He, ich habe mich verändert!"
FOR i=1 TO LEN(b$)
  POKE SADD(a$)+i-1,ASC(MID$(b$,i,1))
NEXT
LIST
```

Sollten Sie das Programm nicht ganz verstanden haben, empfehle ich Ihnen, die Beschreibung des Befehles SADD in Ihrem AmigaBASIC-Handbuch einmal durchzulesen. Lassen Sie das Programm laufen und vergleichen Sie anschließend das Listing mit dem, was Sie eingegeben haben. Sie werden vorfinden:

```
a$="He, ich habe mich verändert!"
b$="He, ich habe mich verändert!"
FOR i=1 TO LEN(b$)
```

Zum einen sehen Sie, wie gefährlich es sein kann, ein Programm zu starten, ohne daß es vorher abgespeichert wurde, zum anderen haben Sie die erste Möglichkeit eines selbstmodifizierenden Programms: In a$ könnte doch zum Beispiel der Name eines Windows stehen. Der Programmbenutzer könnte nun nach Preferences-Voreinsteller-Manier während des Programmlaufes den neuen Namen eingeben, der gepoket wird, und wenn das Programm sich dann mittels SAVE selbst abspeichert, liegt es in veränderter Form auf Disk vor. Hierbei sind Ihrer Kreativität kaum Grenzen gesetzt.

5.2.3.9 Besondere Token

Bei den Befehls-Token (größer als 127) treten einige Besonderheiten auf, die Sie unbedingt beachten sollten. $8E (ELSE) tritt im Programm niemals alleine auf. Der Interpreter kann das Ende eines Befehles nur dann feststellen, wenn entweder der Code $00 für Zeilenende oder der Code $3A, der Doppelpunkt, erreicht wird. Da nach einem IF ... THEN ... nicht unbedingt ein ELSE folgen muß, wird IF-THEN vom Interpreter als ein Befehl abgehandelt.

Folgt nun ein ELSE, so werden Sie mit einem File-Monitor feststellen können, daß der BASIC-Interpreter einen Doppelpunkt vor $8E gehängt hat, der beim Listen des Programms nie auftaucht. Wer also bisher vor jedes ELSE einen Doppelpunkt gesetzt hat, wird mit einem File-Monitor erkennen, daß vor $8E zwei Doppelpunkte stehen, von denen nur einer benötigt wird, der beim Listen nicht sichtbare nämlich.

Ein ähnliches Phänomen tritt bei REMarks auf. Auch hier stellt der Interpreter immer einen Doppelpunkt davor. Merkwürdigerweise tut er das auch, wenn vor dem REMark kein anderer Befehl steht. So kann eine Zeile derartig aussehen:

```
' * 1. *
00 0E 00|3A|AF E8|20 2A 20 31 2E 20 2A|00
  Kopf  | :| '  |   *  1  .  *|Ende
```

Eine weitere Merkwürdigkeit tritt auf, wenn Sie ein Programm erzeugen und dabei den Befehl WHILE beziehungsweise sein Token $BE verwenden. Belassen Sie es dabei, meldet der Amiga beim Programmablauf einen ERROR 22 (Missing operand). Erstellt man mit AmigaBASIC ein Programm, so hängt der Interpreter grundsätzlich ein $EC hinter den scheinbaren Ein-Byte-Token $BE. Sie sollten das auch immer tun - dann funktioniert es nämlich. Also immer $BE+$EC verwenden.

Was noch sehr wichtig ist

Es existiert ein merkwürdiges Token, das niemals gelistet werden kann - und kaum jemand benutzt es nicht. Sie wissen sicherlich, daß das Aufrufen von SUB-Routinen nur direkt nach THEN oder ELSE mit dem Befehl CALL durchgeführt werden muß. Ansonsten kann man den Namen des SUB-Programms anstelle eines BASIC-Befehls verwenden. Schließlich haben die SUB-Programme nur einen Sinn: Sie ermöglichen das Programmieren von Befehlserweiterungen in BASIC. Wer um diesen Umstand weiß, verwendet - außer zum Aufruf von Betriebssystem-Routinen - nie den Befehl CALL, dafür aber (oft ohne es zu wissen) dieses merkwürdige Token. Im Gegensatz zu CALL steht es hinter den Zeigern auf die Variablentabelle. Es handelt sich hierbei um das Doppel-Token $F8-$D1.

Außerdem wäre noch etwas zu dem Befehl DATA zu sagen. Auf ein DATA folgt grundsätzlich alles im (ASCII-)Klartext, ebenso wie nach einem REM, da der Interpreter nunmal keine hellseherischen Fähigkeiten besitzt und daher nicht wissen kann, ob Sie zum Beispiel aus folgender Zeile in Variablen einlesen wollen, wie Float oder Integer, oder ob dies die Zeichenketten für eine Stringvariable sind:

```
DATA &hffe2,123,&06666
```

5.2.3.10 Subprogramme

Wie kam es überhaupt dazu, daß Subprogramme in AmigaBASIC implementiert wurden? Dazu wird jeder ehemals stolze Besitzer

eines C64 oder ähnlichen Computers ein Lied singen können. Etwa nicht? Dann möchte ich an dieser Stelle noch einmal daran erinnern. Der erste Punkt ist sicherlich der, daß damit modulares Programmieren erst möglich wird. Sicherlich trägt zwar auch der Befehl MERGE oder CHAIN dazu bei, jedoch müssen bei deren Methoden die Variablennamen immer gleich sein, sofern sie übergeben werden sollen.

Des weiteren muß auch der Name des nächsten Programmteils feststehen, es sei denn, man lädt das nächste Programm als Variable nach, die dann aber leider auch feststehen muß. An SUB-Routinen können jedoch beliebige Variablen(namen) übergeben werden, da in der Klammer vor dem STATIC Platzhalter definiert werden, in die die übergebenen Werte übertragen werden.

Es ist daher ratsam, jedes Subprogramm einzeln zu editieren und als ASCII-File abzuspeichern, um es bei Bedarf nach dem Editieren eines Programms mittels MERGE im Direktmodus oder im Programm (was unglaublich viel Zeit kostet, von wegen der Syntax-Prüfung und so...) einfach an das speicherresidente BASIC-File anzuhängen. Die Aufrufkonventionen (z.B.: welche Betriebssystemroutinen vorher als Funktion deklariert werden müssen etc.) sollten Sie sich allerdings aufschreiben beziehungsweise mit einer Dateiverwaltung archivieren.

Der zweite Punkt war wohl der, daß bei den bisher erhältlichen Computern ständig jemand beklagte, wie unvollständig doch der Befehlssatz sei und wie schwer es doch für einen BASIC-Programmierer wäre, den Befehlssatz zu erweitern. Nun, Befehlserweiterungen lassen sich auf dem Amiga nicht nur in Maschinensprache oder C programmieren (eigene Libraries), sondern mit SUB-Programmen auch in BASIC. Beispielsweise:

```
PRINTAT 10,20,"Blabla"
SUB PRINTAT (x,y,Text$) STATIC
  LOCATE y,x
  PRINT Text$
END SUB
```

Der dritte Punkt ist ein Anschlag auf Programmierer in anderen Sprachen wie Pascal oder ähnlichen. Wozu komplizierte Sprachen

erlernen, wenn BASIC das auch kann - und auf dem Amiga nicht gerade langsam. Jetzt sagen natürlich schon wieder einige, das wäre mit Pascal nicht zu vergleichen, da sich die SUB-Programme nicht aufrufen können. Das ist nur insofern richtig, als Variablen nicht wieder in Platzhaltervariablen überführt werden (was sich programmiertechnisch allerdings auch lösen läßt).

Geht es nur darum, daß ein Befehl sich selbst bis zu einem gewissen Punkt immer wieder selbst durchläuft, mehrere Iterationen also durchgeführt werden, so hilft ein einfaches Label am Anfang der Routine, zu der dann innerhalb des SUBs dauernd verzweigt werden kann. Programmintern werden SUB-Routinen wie Variablen behandelt. Nur aus dem Zusammenhang heraus kann der Amiga sie als Unterroutinen erkennen.

Wichtige Besonderheiten

Was macht Ihr Manipulationsprogramm, wenn es auf die Codefolge $20-$F8-$8F-$20 trifft? Blättern Sie einmal in der Token-Liste im Anhang nach. Unzweifelbar handelt es sich hier um das $F8-Doppel-Token END, eingeschlossen von zwei Spaces. Ist das Programm hier zu Ende? Was, wenn jetzt die Codes $F8-$BE folgen? Richtig, das ist der Code für SUB! Und von dieser Art gibt es noch genügend Beispiele (INPUT in "OPEN x$ FOR INPUT...").

Sie sehen, ein Token gibt nicht allein Aufschluß über das, was tatsächlich geschieht. Erst der Zusammenhang, in dem das Token zu anderen Token steht, macht die Art der Ausführung aus (hatten wir das nicht auch bei der Verwaltung der Namen von SUB-Programmen?). Das gilt übrigens auch für PRINT# und?# - die Token sind gleich!

5.2.3.11 Andere Token

Was, noch mehr? Ja, leider. Wenn Sie fleißig mitnotiert haben, werden Sie gewisse Löcher in der Token-Reihenfolge mit Werten unter 128 festgestellt haben. Nicht, daß diese Token unge-

nutzt sind, beileibe nicht! Sollten Sie Ihr gerade editiertes Programm nicht mit dem ersten Befehl im Direkt-Modus sichern, hat sich Ihr Programm auch schon verändert. Ihnen ist sicher schon aufgefallen, daß der Interpreter bereits bei der Eingabe einige schwerwiegende Fehler entdeckt, und zwar in dem Moment, in dem Sie einen Direkt-Modus-Befehl eingeben, anstelle dessen Ausführung dann der Fehler-Requester erscheint.

Ausgenommen hiervon ist aus Sicherheitsgründen der SAVE-Befehl. Mir ist es schon passiert, daß sich AmigaBASIC in der Fehler-Anzeige-Routine aufgehängt hat und nur noch den gleichen Fehler anzeigte, ohne wieder Befehle anzunehmen (keine Angst, ich habe 20mal auf OK geklickt).

Es wird also eine einfache Programmüberprüfung eingeleitet, bei der das Programm bereits geändert wird. Und zwar handelt es sich bei den "Loch-Token" um Token, die der Programmablaufsteuerung dienen. So ist beispielsweise $8 zuständig für die Aufnahme von Sprungoffsets bei IF-THEN-Verzweigungen, die aber nicht unbedingt gleich in Programme eingebaut werden. Um für Manipulationsprogramme die Aufgabe nicht unnötig zu erschweren - es gibt eine ganze Reihe Sonderformen - treffen wir folgende Vereinbarung:

1. Bei Manipulationsprogrammen oder Programmen zum Auslesen von Daten aus anderen Programmen, die ein binäres File-Format benötigen, ist folgendermaßen vorzugehen:
   - Speichern des zu bearbeitenden Files als ASCII-Datei.
   - Laden und mit dem ersten weiteren Befehl gleich wieder als Binär-File speichern.

2. Bei ASCII-Files ist keine Sonderbehandlung nötig, da mit dem Abspeichern die Programmsteuerungscodes nicht abgespeichert werden.

5.3 Nützliche Programme zur Manipulation von AmigaBASIC

Die nachfolgenden Unterkapitel stellen Ihnen einige Programme zur Verfügung, mit denen Sie Ihre BASIC-Programme bearbeiten können.

5.3.1 DATA-Generator

Dieses Programm demonstriert, wie man von einem Programm aus ein AmigaBASIC-ASCII-Programm erzeugen kann. Nun ist es zwar eleganter, Daten in einem File auf Diskette abzuspeichern und dann jedesmal wieder zu laden, dennoch gibt es Projekte, bei denen man ohne DATA-Zeilen nicht mehr auskommt. So gibt es bei einem guten Programm immer die Möglichkeit, es einmal in einer Zeitschrift abdrucken zu lassen und dafür natürlich auch etwas Geld (für Erweiterungen natürlich!) zu bekommen.

Was aber, wenn das Programm nicht ohne Sprites, BOBs, Maschinenspracheroutinen oder ähnlichem auskommt? Da gibt es gar keine Frage: DATA-Zeilen müssen her. Nun, das vorliegende Programm erzeugt DATA-Zeilen von jedem beliebigen File. Für andere Anwendungszwecke steht es Ihnen natürlich frei, das Programm nach eigenen Wünschen umzuschreiben.

Das erzeugte ASCII-File läßt sich ganz einfach an das Programm, das die DATAs benötigt, durch MERGE anhängen. Um die DATA-Zeilen nicht unnötig lang zu machen, werden die Werte als Hexadezimalzahlen ausgegeben. Eine Leseroutine für die DATAs wird mit abgespeichert. Ungewöhnlich an der Einleseroutine ist für eifrige Leser des AmigaBASIC-Handbuches sicherlich die Umwandlung von Hexadezimalzahlen in Dezimalzahlen.

Hier ist das BASIC-Handbuch schlichtweg falsch! Natürlich ist es für den Interpreter völlig egal, ob Sie a%=255 oder a%=&HFF schreiben. Ebenso funktioniert das selbstverständlich auch bei

VAL und ähnlichen Funktionen! Die umständlichen Umrechenroutinen - in 64er BASIC gehalten - die man immer wieder in Amiga-Listings findet, entsprechen keinesfalls dem Standard von AmigaBASIC. Sie können daher ohne weiteres schreiben:

```
daten: DATA ff,ec,0,1,f
RESTORE daten:FOR i=1 TO 5:READ a$:x(i)=VAL("&H"+a$):NEXT
```

Man bemerke: Nur ein Befehl anstelle eines ganzen Subprogramms! Nun aber zum Listing:

```
GOTO start
' ####################################
' # D A T A - G E N E R A T O R  Amiga #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
' "dos.bmap" und "exec.bmap" muessen auf
' Disk vorhanden sein!
' ------------------------------------
' Betriebssystemroutinen als Funktionen
' deklarieren
'
start:
  DECLARE FUNCTION xOpen&    LIBRARY
  DECLARE FUNCTION xRead%    LIBRARY
  DECLARE FUNCTION AllocMem& LIBRARY
  DECLARE FUNCTION Examine&  LIBRARY
  DECLARE FUNCTION Lock&     LIBRARY
' ------------------------------------
' Bibliotheken öffnen
'
  LIBRARY "exec.library"
  LIBRARY "dos.library"
' ------------------------------------
' Eingaben
'
sourcefile:
  CLS
  LINE INPUT "Name des Source-Files: ";source$
  PRINT
  PRINT "Diskette einlegen (RETURN)"
  WHILE A$<>CHR$(13)
    A$=INKEY$
  WEND
  LOCATE 3,1:PRINT "Checking File...          "
  CHDIR "df0:"
  CheckFile source$,Bytes&
  IF Bytes&=0 THEN
```

```
    LOCATE 3,1:PRINT "File not found...":BEEP
    A=TIMER+3 :WHILE A>TIMER:WEND
    GOTO sourcefile
  ELSEIF Bytes&=-1 THEN
    LOCATE 3,1:PRINT "Directories kann ich nicht..."
    BEEP :A=TIMER+3:WHILE A>TIMER:WEND
    GOTO sourcefile
  END IF
  LOCATE 3,1:PRINT "File gefunden. Länge=";Bytes&;" Byte"
' ------------------------------------
' Buffer einrichten
'
  PublicRAM&=65537&
  Buffer&=AllocMem&(Bytes&,PublicRAM&)
  IF Buffer&=0 THEN
    LOCATE 5,1:PRINT "Nicht genug Speicher vorhanden."
    LOCATE 7,1
    PRINT "Programm kann mit RUN wieder gestartet werden."
    BEEP :END
  END IF
' ------------------------------------
' File in Buffer laden
'
  source$=source$+CHR$(0)
  Opened&=xOpen&(SADD(source$),1005)
  IF Opened&=0 THEN
    LOCATE 5,1:PRINT "Ich kann das File nicht öffnen!"
    BEEP :A=TIMER+3:WHILE A>TIMER:WEND
    GOTO sourcefile
  END IF
  gelesen%=xRead%(Opened&,Buffer&,Bytes&)
  CALL xClose(Opened&)
' ------------------------------------
' Eingabe Target-File
'
targetfile:
  LOCATE 9,1:PRINT "Name des BASIC-ASCII-Files,"
  FOR i=11 TO 17 STEP 2
    LOCATE i,1:PRINT SPACE$(80)
  NEXT
  LOCATE 11,1:LINE INPUT "das erzeugt werden soll: ";target$
  LOCATE 13,1:PRINT "Target-Disk einlegen (RETURN)"
  A$="" :WHILE A$<>CHR$(13):A$=INKEY$:WEND
  CHDIR "df0:"
  LOCATE 15,1:PRINT "Checking Disk..."
  CheckFile target$,vorhanden&
  IF vorhanden&=-1 THEN
    LOCATE 15,1:PRINT "Das ist der Name eines Directory!"
    BEEP :A=TIMER+3:WHILE A>TIMER:WEND
    GOTO targetfile
  ELSEIF vorhanden&<>0 THEN
    LOCATE 15,1:PRINT "Es ist bereits ein File mit diesem"
    LOCATE 17,1:PRINT "Namen vorhanden. File löschen? (J/N)"
```

```
warte:
    A$=INKEY$ :IF A$<>"" THEN A$=UCASE$(A$)
    IF A$="J" GOTO weiter
    IF A$<>"N" GOTO warte
    GOTO targetfile
  END IF
weiter:
' ------------------------------------
' DATA-ASCII-File erzeugen
'
  LOCATE 19,1:PRINT "ASCII-File wird erzeugt."
  LOCATE 21,1:PRINT "Bitte etwas Geduld..."
  OPEN target$ FOR OUTPUT AS 1
    anzahl&=0
    PRINT#1,"RESTORE datas";CHR$(10);
    PRINT#1,"datastring$=";CHR$(34);CHR$(34);CHR$(10);
    PRINT#1,"FOR i=1 TO ";STR$(Bytes&);CHR$(10);
    PRINT#1,"READ a$";CHR$(10);
    PRINT#1,"a$=";CHR$(34);"&H";CHR$(34);"+a$";CHR$(10);
    PRINT#1,"datastring$=datastring$+CHR$(VAL(a$))";
     PRINT#1,CHR$(10);
    PRINT#1,"NEXT";CHR$(10);
    PRINT#1,"datas:";CHR$(10);
Zeile:
    PRINT#1,"DATA ";
    zahl=0
Wert:
    PRINT#1,HEX$(PEEK(Buffer&+anzahl&));
    zahl=zahl+1 :anzahl&=anzahl&+1
    IF anzahl&<Bytes& THEN
      IF zahl<20 THEN
        PRINT#1,",";
        GOTO Wert
      ELSE
        PRINT#1,CHR$(10);
        GOTO Zeile
      END IF
    END IF
    PRINT#1,CHR$(10);CHR$(10);
  CLOSE 1
' ------------------------------------
' .info-file ändern
'
  SAVE "DATA-GENINFO"
  weg$=target$+".info"
  KILL weg$
  NAME "DATA-GENINFO.info" AS target$+".info"
  KILL "DATA-GENINFO"
  CLS
  PRINT "fertig."
  CALL FreeMem(Buffer&,Bytes&)
  END
' ------------------------------------
```

```
' SUBROUTINE
'
SUB CheckFile(Filename$,Length&) STATIC
  ChipRAM&=65538&
  InfoBytes&=252
  Info&=AllocMem&(InfoBytes&,ChipRAM&)
  IF Info&=0 THEN ERROR 7
  File$=Filename$+CHR$(0)
  DosLock&=Lock&(SADD(File$),-2)
  IF DosLock&=0 THEN
    Length&=0
  ELSE
    Dummy&=Examine&(DosLock&,Info&)
    Length&=PEEKL(Info&+4)
    IF Length&>0 THEN
      Length&=-1
    ELSE
      Length&=PEEKL(Info&+124)
    END IF
  END IF
  CALL UnLock(DosLock&)
  CALL FreeMem(Info&,InfoBytes&)
END SUB
```

Variablen

A	String, Hilfsvariable
AllocMem	EXEC-Routine, Speicher reservieren
Buffer	Adresse des reservierten Speichers
Bytes	Länge der zu bearbeitenden Datei
CheckFile	SUB-Routine, prüft, ob File vorhanden, wenn ja, ob Directory, wenn nicht, Länge holen
ChipRAM	Option für AllocMem: 2^16 (65536) = Bereich löschen, 2^1 (2) = Chip-RAM-Bereich
DosLock	File-Handle der Checkfile-Routine
Dummy	nicht genutzte Variable
Examine	DOS-Routine, untersucht File
File	Filename mit abschließender Null für DOS
Filename	Name der zu bearbeitenden Datei
FreeMem	EXEC-Routine, gibt Speicherbereich frei
Info	Adresse der File-Info-Struktur
InfoBytes	Länge der File-Info-Struktur
Length	Länge des Files
Lock	DOS-Routine, sperrt File gegen Zugriffe von anderen Programmen und besorgt Handle
Opened	Adresse des File-Handlers der Source-Datei
PublicRAM	Option für AllocMem: 2^16 (65536) = Bereich löschen, 2^0 (1) = Public-Bereich

UnLock	DOS-Routine, hebt Lock auf
anzahl	Zähler für geschriebene DATA-Werte
gelesen Anzahl	tatsächlich gelesener Bytes
i	Schleifenvariable
source	Ausgangsdatei
target	Zieldatei im ASCII-Format für DATAs
vorhanden Flag:	existiert File?
weg	Hilfsvariable
xClose	DOS-Routine, schließt File
xOpen	DOS-Routine, öffnet File
xRead	DOS-Routine, liest aus File
zahl	Zähler für Bytes in einer DATA-Zeile

5.3.2 Cross-Reference-Liste

Dieses Programm demonstriert das Auslesen von Werten aus binär gespeicherten AmigaBASIC-Programmen. Um den Rahmen dieses Buches nicht zu sprengen, wird keine vollständige Syntax-Prüfung des Programms vorgenommen. Dadurch ergibt sich, daß eventuell vom Interpreter eingebaute Programmablaufsteuerungsmarken sowie Speichermüll zwischen dem Programmrumpf und der Variablentabelle vor dem Einsatz dieses Programms aus dem zu bearbeitenden File entfernt werden müssen. Um das zu erreichen, gehen Sie folgendermaßen vor:

1. Laden des zu bearbeitenden Files
2. SAVE "filename",A
3. AmigaBASIC beenden
4. AmigaBASIC neu laden
5. LOAD "filename"
6. SAVE "filename",B

Haben Sie dies getan, so können Sie sich mit folgendem Programm eine Kreuzverweis-Liste auf dem Drucker ausgeben lassen. Dabei werden sowohl Label als auch Zeilennummern in der Reihenfolge, in der sie im Programm auftauchen, ausgegeben. Zu diesen Sprungmarken wird durch "<--" markiert angegeben, von welchen Labeln oder Zeilennummern aus diese Sprungmarke angesprungen wird.

Erfolgt der Ansprung von einer Stelle im Programm, an der noch keine Sprungmarke definiert ist, zum Beispiel von der ersten Programmzeile aus, so wird dies mit dem geklammerten Pseudo-Label "(Programm-Anfang)" kenntlich gemacht. Dann folgen noch durch "-->" gekennzeichnet die Sprungmarken, die von dieser Sprungmarke aus angesprungen werden. Zu beachten ist hierbei, daß weder Betriebssystemaufrufe noch SUB-Routinen berücksichtigt werden, da diese von dem AmigaBASIC-Interpreter wie Variablen abgelegt werden. Trotzdem haben Sie hiermit ein wunderbares Werkzeug, um Ihre Programme zu dokumentieren.

```
' ####################################
' # C r o s s R e f e r e n c e  Amiga #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
' Dieses Programm erstellt auf Ihrem
' Printer eine Kreuzverweis-Liste
' ( Cross-Reference-Chart ), die es
' Ihnen erlaubt, jedes BINÄR gespeicherte
' AmigaBASIC-Programm zu dokumentieren.
' ---------------------------------------
' Da SUB-Routinen vom Interpreter wie in
' AmigaBASIC programmierte AmigaBASIC-
' Befehls-Erweiterungen gehandhabt werden,
' bleiben deren Aufrufe unberuecksichtigt.
' ---------------------------------------
'
'----Speicher reservieren, Druckertreiber ----
'----laden,Bibliothek öffnen und Variablen----
CLEAR,45000&
LPRINT
DECLARE FUNCTION xOpen& LIBRARY
DECLARE FUNCTION xRead% LIBRARY
DECLARE FUNCTION Seek%  LIBRARY
LIBRARY ":dos.library"
DIM Cross$(5000),names$(1000)
LOCATE 2,2
PRINT CHR$(187);" Cross Reference Amiga ";CHR$(171)
LOCATE 5,2
PRINT "Name des binären AmigaBASIC-Programms:"
LOCATE 7,2
LINE INPUT Filename$
CHDIR "df0:"
BASICcheck Filename$,Result%
LOCATE 10,2
IF Result%=-1 THEN
```

```
  PRINT "Ich kann kein Info-File finden."
 ELSEIF Result%=0 THEN
  PRINT "Lese-Fehler!"
 ELSEIF Result%=1 THEN
  PRINT "Das ist kein AmigaBASIC-Programm."
 END IF
 IF Result%<>2 THEN
  BEEP
  WHILE INKEY$=""
  WEND
  RUN
 END IF
 PRINT CHR$(34);Filename$;".info";CHR$(34)
 PRINT
 PRINT " weist dieses Prg als AmigaBASIC-File aus."
 OpenFile Filename$,handle&
 LOCATE 14,2
 IF handle&=0 THEN
  PRINT "AAAaargh! Ich finde ";CHR$(34);
  PRINT Filename$;CHR$(34);" nicht!!!"
  BEEP
  WHILE INKEY$="":WEND:RUN
 ELSE
  PRINT "File geöffnet."
 END IF
 LOCATE 16,2
 HeaderCheck handle&,Header$
 IF ASC(Header$)<>&HF5 THEN
  PRINT "Sorry, ich kann nur binär-Files"
  BEEP
  WHILE INKEY$="":WEND:RUN
 ELSE
  PRINT "File hat binäres Format"
  PRINT :PRINT "Bitte etwas Geduld. ";
  PRINT "Ich melde mich wieder..."
 END IF
 pointer%=-1
main:
 GetLine handle&,Current$
 IF LEN(Current$)<4 THEN
  PRINT
  PRINT " Ende des Binär-Codes erreicht"
  PRINT :PRINT " Lese Variablentabelle."
  GOTO Vartab
 END IF
 IF ASC(Current$)=128 THEN
  pointer%=pointer%+1
  Cross$(pointer%)=CHR$(128)+MID$(Current$,4,2)
  Current$=MID$(Current$,6)
 ELSE
  Current$=MID$(Current$,4)
 END IF
GetToken:
```

```
 Token%=ASC(Current$+CHR$(0))
 IF Token%=0 GOTO main
'----BefehlsToken?----
 IF Token%>127 THEN
  IF Token%=175 OR Token%=141 GOTO main
  IF Token%=190 OR Token%>247 THEN
   Current$=MID$(Current$,3)
  ELSE
   Current$=MID$(Current$,2)
  END IF
  GOTO GetToken
 END IF
'----String?----
 IF Token%=34 THEN
  Byte%=INSTR(2,Current$,CHR$(34))
  IF Byte%=0 GOTO main
  Current$=MID$(Current$,Byte%+1)
  GOTO GetToken
 END IF
'----Folgt 2-Byte-Wert?----
 IF Token%=1 OR Token%=11 OR Token%=12 OR Token%=28 THEN
  Current$=MID$(Current$,4)
  GOTO GetToken
 END IF
'----Folgt 1-Byte-Wert?----
 IF Token%=15 THEN Current$=MID$(Current$,3):GOTO GetToken
'----Folgt 4-Byte-Wert?----
 IF Token%=29 OR Token%=30 THEN
  Current$=MID$(Current$,6)
  GOTO GetToken
 END IF
'----Folgt 8-Byte-Wert?----
 IF Token%=31 THEN Current$=MID$(Current$,10):GOTO GetToken
'----Ist es ein Label?----
 IF Token%=2 THEN
  pointer%=pointer%+1
  Cross$(pointer%)=LEFT$(Current$,3)
  Current$=MID$(Current$,4)
  GOTO GetToken
 END IF
'----Ist es ein Sprungziel?----
 IF Token%=3 OR Token%=14 THEN
  pointer%=pointer%+1
  Cross$(pointer%)=CHR$(Token%)+MID$(Current$,3,2)
  Current$=MID$(Current$,5)
  GOTO GetToken
 END IF
 Current$=MID$(Current$,2)
 GOTO GetToken
Vartab:
 p2%=-1
notforever:
 GetLength handle&,bytes%
```

```
 IF bytes%=0 GOTO goon
 GetName handle&,Current$,bytes%
 p2%=p2%+1
 names$(p2%)=Current$
 GOTO notforever
goon:
 IF pointer%=-1 THEN
  PRINT
  PRINT "Ich habe kein Label und keine Zeilennummer"
  PRINT
  PRINT "entdecken können!"
  BEEP
  WHILE INKEY$="":WEND:RUN
 ELSEIF p2%=-1 THEN
  PRINT
  PRINT "Hmm - keine Variablentabelle"
  BEEP
  WHILE INKEY$="":WEND:RUN
 ELSE
  PRINT :PRINT " Gebe Daten aus."
 END IF
 LPRINT ">>> CrossReference Amiga <<<"
 LPRINT "---------------------------"
 LPRINT "Programm: ";Filename$
 LPRINT
 FOR i=0 TO pointer%
  ascii%=ASC(Cross$(i))
  IF ascii%=2 THEN
   LPRINT names$(CVI(MID$(Cross$(i),2)));":"
   FOR j=0 TO pointer%
    IF ASC(Cross$(j))=3 THEN
     IF CVI(MID$(Cross$(j),2))=CVI(MID$(Cross$(i),2)) THEN
      k=j
      WHILE k>-1
       k=k-1
       IF k>-1 THEN
        IF ASC(Cross$(k))=2 THEN
         LPRINT "  <-- ";
         LPRINT names$(CVI(MID$(Cross$(k),2)))
         k=-2
        ELSEIF ASC(Cross$(k))=128 THEN
         LPRINT "  <-- ";CVI(MID$(Cross$(k),2))
         k=-2
        END IF
       END IF
      WEND
      IF k=-1 THEN LPRINT "  <--(Programm-Anfang)"
     END IF
    END IF
   NEXT j
  ELSEIF ascii%=3 THEN
   LPRINT "  --> ";names$(CVI(MID$(Cross$(i),2)))
  ELSEIF ascii%=14 THEN
```

```
   LPRINT "  --> ";CVI(MID$(Cross$(i),2))
  ELSEIF ascii%=128 THEN
   LPRINT CVI(MID$(Cross$(i),2))
   FOR j=0 TO pointer%
    IF ASC(Cross$(j))=14 THEN
     IF CVI(MID$(Cross$(j),2))=CVI(MID$(Cross$(i),2)) THEN
      k=j
      WHILE k>-1
       k=k-1
       IF k>-1 THEN
        IF ASC(Cross$(k))=2 THEN
         LPRINT "  <-- ";
         LPRINT names$(CVI(MID$(Cross$(k),2)))
         k=-2
        ELSEIF ASC(Cross$(k))=128 THEN
         LPRINT "  <-- ";CVI(MID$(Cross$(k),2))
         k=-2
        END IF
       END IF
      WEND
      IF k=-1 THEN LPRINT "  <--(Programm-Anfang)"
     END IF
    END IF
   NEXT j
  END IF
 NEXT i
 PRINT :PRINT "Fertig."
 BEEP
 WHILE INKEY$="":WEND:RUN

SUB GetName(handle&,Current$,bytes%) STATIC
  Current$=SPACE$(bytes%)
  Length%=xRead%(handle&,SADD(Current$),bytes%)
END SUB
SUB GetLength(handle&,bytes%) STATIC
  Current$=CHR$(0)
readit:
  Length%=xRead%(handle&,SADD(Current$),1)
  IF Length%=0 THEN
    CALL xClose(handle&)
    bytes%=0
    EXIT SUB
  END IF
  bytes%=ASC(Current$)
  IF bytes%=0 THEN readit
  IF bytes%>60 THEN readit

END SUB

SUB GetLine(handle&,Current$) STATIC
  Current$=STRING$(3,0)
  Length%=xRead%(handle&,SADD(Current$),3)
  OldPos%=Seek%(handle&,-3,0)
```

```
  LoL%=ASC(MID$(Current$,2,1))
  IF LoL%=0 THEN
    EXIT SUB
  ELSE
    Current$=STRING$(LoL%,0)
    Length%=xRead%(handle&,SADD(Current$),LoL%)
  END IF
END SUB

SUB HeaderCheck(handle&,Header$) STATIC
  Header$="J"
  OldPos%=Seek%(handle&,0,-1)
  gotit%=xRead%(handle&,SADD(Header$),1)
END SUB

SUB OpenFile(Filename$,handle&) STATIC
  file$=Filename$+CHR$(0)
  handle&=xOpen&(SADD(file$),1005)
END SUB

SUB BASICcheck(Filename$,Result%) STATIC
  file$=Filename$+".info"+CHR$(0)
  Default.Tool$=SPACE$(20)
  handle&=xOpen&(SADD(file$),1005)
  IF handle&=0 THEN
    Result%=-1
  ELSE
    OldPos%=Seek%(handle&,-20,1)
    gotit%=xRead%(handle&,SADD(Default.Tool$),20)
    IF gotit%<20 THEN
      Result%=0
    ELSE
      IF INSTR(Default.Tool$,"AmigaBASIC")>0 THEN
        Result%=2
      ELSE
        Result%=1
      END IF
    END IF
    CALL xClose(handle&)
  END IF
END SUB
```

Variablen

BASICcheck	SUB-Routine zum Prüfen des Default-Tools
Byte	Pointer auf Byte im String
Bytes	Hilfsvariable, z.B. Stringlänge
Cross	Stringfeld zur Zwischenspeicherung der Sprungmarken und Sprünge
Current	String, eingelesene BASIC-Zeile
Default.Tool	String, in den das Default-Tool gelesen wird

Filename	String, Name des zu bearbeitenden Files
GetLength	SUB-Routine, holt die Label-Länge
GetLine	SUB-Routine, liest Zeile ein
GetName	SUB-Routine, liest Label-Name ein
Header	String, File-Header-Byte
HeaderCheck	SUB-Routine, die die Abspeicherart prüft
Length	Anzahl tatsächlich gelesener Bytes
LoL	Zeilenlänge
OldPos	alte Pointer-Position im File
OpenFile	SUB-Routine, öffnet File
Result	Flag, Ergebnis der Untersuchung
Seek	DOS-Routine, verschiebt Schreib-Lese-Zeiger im File
Token	ASCII-Wert des nächsten Bytes einer Zeile
ascii	Wert des Codes in Cross$
file	String, Filename mit Null abgeschlossen für DOS-Routinen
gotit	tatsächlich gelesene Bytes
handle	Adresse auf den File-Handler
i	Schleifenvariable
j	Schleifenvariable
k	Schleifenvariable
names	Stringfeld, Namen der Sprungmarken
p2	Hilfsvariable
pointer	Hilfsvariable
xClose	DOS-Routine, schließt File
xOpen	DOS-Routine, öffnet File
xRead	DOS-Routine, liest aus File an Speicheradresse

5.3.3 Leerzeilen-Killer

Da wir wissen, wie Leerzeilen aufgebaut sind, sollte es uns nun leichtfallen, diese zu entfernen. Das folgende Programm erledigt diese Arbeit für uns. Vorher muß jedoch jeglicher Speichermüll beseitigt werden. Wie das vor sich geht, sehen Sie im Text zur Cross-Reference-Liste.

Wichtig: Wenn Sie das Programm abtippen, könnte Ihnen ein winziger Fehler unterlaufen, der Ihr Originalprogramm zerstören würde. Verwenden Sie daher nur Kopien Ihrer Originale, und prüfen Sie nach der Bearbeitung die Lauffähigkeit. Das vorliegende Programm ändert das Ausgangsfile ab. Des weiteren

wird aus Gründen der Speicher-Ersparnis während der Bearbeitung das gerade aktuelle Fenster geschlossen. Sollte noch ein kleiner Fehler im Programm sein, wie beispielsweise eine Endlosschleife, kommen Sie daher nicht mehr an Ihr Programm heran. Ausweg: Solange das Programm noch nicht einwandfrei läuft, lassen Sie das List-Fenster offen - aber nicht aktiviert! Es ist jedoch ganz normal, wenn sich das Programm eine ganze Zeit lang nicht zurückmeldet - je nach Länge des zu bearbeitenden Files.

```
' ####################################
' #      Leerzeilen-Killer  Amiga      #
' #------------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
' "dos.bmap" und "exec.bmap" muessen auf
' Disk vorhanden sein
' ------------------------------------
'
  DECLARE FUNCTION AllocMem& LIBRARY
  DECLARE FUNCTION Lock&     LIBRARY
  DECLARE FUNCTION Examine&  LIBRARY
  DECLARE FUNCTION xOpen&    LIBRARY
  DECLARE FUNCTION xRead&    LIBRARY
  DECLARE FUNCTION xWrite&   LIBRARY
  LIBRARY ":exec.library"
  LIBRARY ":dos.library"
  WINDOW CLOSE WINDOW(0)
  WINDOW 1,"Leerzeilen-Killer",(0,0)-(250,50),16
Allocation.1:
  COLOR 3,1:CLS
  info&=AllocMem&(252&,65538&)
  IF info&=0 THEN
    ALLOCERR
    GOTO Allocation.1
  END IF
Source:
  REQUEST "SOURCE"
  SELECT box%
  IF box% THEN CALL FreeMem(info&,252):SYSTEM
  CHDIR "df0:"
GetFilename:
  LINPUT Filename$
  GETINFO Filename$,info&,Length&
  IF Length&<1 THEN
    IF Length&=-1 THEN
```

```
      DIRERR
    ELSEIF Length&=0 THEN
      FILEERR
    END IF
    GOTO GetFilename
  END IF
Allocation.2:
  COLOR 3,1:CLS
  buffer&=AllocMem&(Length&,65537&)
  IF buffer&=0 THEN
    ALLOCERR
    GOTO Allocation.2
  END IF
  LOADFILE Filename$,buffer&,Length&
  IF Filename$="" THEN
    CALL FreeMem(buffer&,Length&)
    LOADERR
    GOTO GetFilename
  END IF
  IF PEEK(buffer&)<>&HF5 THEN
    CALL FreeMem(buffer&,Length&)
    FORMERR
    GOTO GetFilename
  END IF
  NEWFILE Filename$,handle&
  IF handle&=0 THEN
    CALL FreeMem(buffer&,Length&)
    CALL FreeMem(info&,252&)
    OPENERR
    SYSTEM
  END IF
  Bytes&=1
  DWRITE handle&,buffer&,Bytes&
  IF Bytes&=0 THEN
    CALL xClose(handle&)
    CALL FreeMem(buffer&,Length&)
    CALL FreeMem(info&,252&)
    WRITEERR
    SYSTEM
  END IF
  pointer&=buffer&+1
GetLength:
  Bytes&=PEEK(pointer&+1)
  IF Bytes&=4 THEN
    pointer&=pointer&+4
    GOTO GetLength
  ELSEIF Bytes&>4 THEN
    DWRITE handle&,pointer&,Bytes&
    IF Bytes&=0 THEN
      CALL xClose(handle&)
      CALL FreeMem(buffer&,Length&)
      CALL FreeMem(info&,252&)
      WRITEERR
```

```
      SYSTEM
    END IF
    pointer&=pointer&+Bytes&
    GOTO GetLength
  ELSE
    Bytes&=Length&-(pointer&-buffer&+1)
    DWRITE handle&,pointer&,Bytes&
    IF Bytes&=0 THEN
      CALL xClose(handle&)
      CALL FreeMem(buffer&,Length&)
      CALL FreeMem(info&,252&)
      WRITEERR
      SYSTEM
    END IF
  END IF
  CALL xClose(handle&)
  CALL FreeMem(buffer&,Length&)
  CALL FreeMem(info&,252&)
  LIBRARY CLOSE
  COLOR 3,1:CLS:LOCATE 2,2:PRINT "Ready."
  WHILE INKEY$="":WEND
  SYSTEM
SUB WRITEERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Write-error."
  ShowCont
END SUB
SUB DWRITE(handle&,adr&,Length&) STATIC
  written&=xWrite&(handle&,adr&,Length&)
  IF written&<>Length& THEN Length&=0
END SUB
SUB OPENERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Can't open File."
  ShowCont
END SUB
SUB NEWFILE(Filename$,handle&) STATIC
  File$=Filename$+CHR$(0)
  handle&=xOpen&(SADD(File$),1005)
END SUB
SUB FORMERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Not a binary File."
  ShowCont
END SUB
SUB LOADERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Load-error."
  ShowCont
END SUB
SUB LOADFILE(Filename$,buffer&,Length&) STATIC
  File$=Filename$+CHR$(0) :handle&=xOpen&(SADD(File$),1005)
  IF handle&=0 THEN
    Filename$=""
  ELSE
    inBuffer&=xRead&(handle&,buffer&,Length&)
    CALL xClose(handle&)
```

```
    IF inBuffer&<>Length& THEN Filename$=""
  END IF
END SUB
SUB FILEERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  File not found."
  ShowCont
END SUB
SUB DIRERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2
  PRINT "ERROR:  File is a Directory."
  ShowCont
END SUB
SUB GETINFO(Filename$,info&,Length&) STATIC
  File$=Filename$+CHR$(0) :DosLock&=Lock&(SADD(File$),-2)
  IF DosLock&=0 THEN
    Length&=0
  ELSE
    Dummy&=Examine&(DosLock&,info&)
    IF PEEKL(info&+4)>0 THEN
      Length&=-1
    ELSE
      Length&=PEEKL(info&+124)
    END IF
  END IF
  CALL UnLock(DosLock&)
END SUB
SUB LINPUT(Filename$) STATIC
  COLOR 3,1:CLS:WINDOW 2,"Filename:",(0,0)-(250,10),0
  WINDOW OUTPUT 1:LOCATE 5,2
  PRINT "Name of a binary saved File";
  LINE INPUT Filename$:WINDOW CLOSE 2
END SUB
SUB SELECT(box%) STATIC
Check:
  WHILE MOUSE(0)=0:WEND:x=MOUSE(1):y=MOUSE(2)
  IF y>27 AND y<43 THEN
    IF x>9 AND x<38 THEN box%=0:EXIT SUB
    IF x>177 AND x<238 THEN box%=-1:EXIT SUB
  END IF
  GOTO Check
END SUB
SUB ALLOCERR STATIC
COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Allocation denied."
  ShowCont
END SUB
SUB ShowCont STATIC
  LOCATE 4,2:PRINT "Press SPACE to continue,"
  LOCATE 5,7:PRINT "ESCAPE to exit.";
  WHILE a$<>CHR$(32) AND a$<>CHR$(27)
    a$=INKEY$
  WEND
  IF a$=CHR$(27) THEN SYSTEM
END SUB
```

```
SUB REQUEST(disk$) STATIC
  COLOR 3,1:CLS
  LOCATE 2,2:PRINT "INSERT ";disk$;" DISK INTO DRIVE"
  LOCATE 3,14:PRINT "DF0:":LOCATE 5,3:PRINT "OK";
  LOCATE 5,24:PRINT "CANCEL";:LINE(10,28)-(37,42),3,b
  LINE(178,28)-(237,42),3,b
END SUB
```

Variablen

ALLOCERR	SUB: Fehler beim Reservieren von Speicher
AllocMem	EXEC-Routine: reserviert Speicherplatz
Bytes	Länge
DIRERR	SUB: Fehler - keine Datei
DWRITE	SUB: schreibe in File
DosLock	Handle von Lock
Dummy	nicht benötigt
Examine	DOS-Routine: untersucht File
FILEERR	SUB: Fehler s.o.
FORMERR	SUB: Fehler s.o.
File	Filename + Null für DOS-Routinen
Filename	Programm
FreeMem	EXEC-Routine: Speicher freigeben
GETINFO	SUB: File-check
LINPUT	SUB: Eingabe
LOADERR	SUB: Fehler s.o.
LOADFILE	SUB: lädt Programm
Length	Länge des Programms
Lock	DOS-Routine: "saugt" sich am File fest
NEWFILE	SUB: erstellt neues File
OPENERR	SUB: Fehler s.o.
REQUEST	SUB: Zeichne primitiven Requester
SELECT	SUB: Auswahl per Mausklick
ShowCont	SUB: zeige Möglichkeiten
UnLock	DOS-Routine: löst Saugfuß
WRITEERR	SUB: Fehler
a	Hilfsvariable
adr	Adresse
b	Hilfsvariable
box	Hilfsvariable
buffer	Adresse des reservierten Speichers
disk	Diskette
handle	Adresse des File-Handlers
inBuffer	gelesene Bytes
info	Adresse der File-Info-Struktur
pointer	Hilfsvariable

written	geschriebene Bytes
x	Hilfsvariable
xClose	DOS-Routine: schließt File
xOpen	DOS-Routine: öffnet File
xRead	DOS-Routine: liest aus File
xWrite	DOS-Routine: schreibt in File
y	Hilfsvariable

5.3.4 REMarks entfernen

Für dieses Programm gilt das gleiche wie für den Leerzeilen-Killer! Beachten Sie bitte den dortigen Text.

```
' ####################################
' #   K i l l - R e m a r k    Amiga   #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
' "dos.bmap" und "exec.bmap" muessen auf
' Disk vorhanden sein
' ------------------------------------
'
  DECLARE FUNCTION AllocMem& LIBRARY
  DECLARE FUNCTION Lock&     LIBRARY
  DECLARE FUNCTION Examine&  LIBRARY
  DECLARE FUNCTION xOpen&    LIBRARY
  DECLARE FUNCTION xRead&    LIBRARY
  LIBRARY ":exec.library"
  LIBRARY ":dos.library"
  WINDOW CLOSE WINDOW(0)
  WINDOW 1,"Kill-Remark",(0,0)-(250,50),16
Allocation.1:
  COLOR 3,1:CLS
  info&=AllocMem&(252&,65538&)
  IF info&=0 THEN
    ALLOCERR
    GOTO Allocation.1
  END IF
Source:
  REQUEST "SOURCE"
  SELECT box%
  IF box% THEN CALL FreeMem(info&,252):SYSTEM
  CHDIR "df0:"
GetFilename:
  LINPUT filename$
  GETINFO filename$,info&,Length&
  IF Length&<1 THEN
    IF Length&=-1 THEN
      DIRERR
```

```
    ELSEIF Length&=0 THEN
      FILEERR
    END IF
    GOTO GetFilename
  END IF
Allocation.2:
  COLOR 3,1:CLS
  buffer&=AllocMem&(Length&,65537&)
  IF buffer&=0 THEN
    ALLOCERR
    GOTO Allocation.2
  END IF
  LOADFILE filename$,buffer&,Length&
  IF filename$="" THEN
    CALL FreeMem(buffer&,Length&)
    LOADERR
    GOTO GetFilename
  END IF
  IF PEEK(buffer&)<>&HF5 THEN
    CALL FreeMem(buffer&,Length&)
    FORMERR
    GOTO GetFilename
  END IF
  NEWFILE filename$
  Bytes&=1
  DWRITE buffer&,Bytes&
  pointer&=buffer&+1
GetLength:
  Bytes&=PEEK(pointer&+1)
  IF Bytes&=4 THEN
    pointer&=pointer&+4
    GOTO GetLength
  ELSEIF Bytes&>4 THEN
    IF PEEK(pointer&)=128 THEN offs&=6 ELSE offs&=4
    IF PEEK(pointer&+offs&)<>175 THEN
      DWRITE pointer&,Bytes&
    END IF
    pointer&=pointer&+Bytes&
    GOTO GetLength
  ELSE
    IF ((pointer&-buffer&+1)MOD 2)=1 THEN
      pointer&=pointer&-1
    END IF
    Bytes&=Length&-(pointer&-buffer&+1)+1
    DWRITE pointer&,Bytes&
  END IF
  CLOSE 1
  OPEN filename$+"-RL.info" FOR OUTPUT AS 1
  OPEN filename$+".info" FOR INPUT AS 2
  PRINT#1,INPUT$(LOF(2),2);
  CLOSE 2,1
  KILL filename$+"-RL.info.info"
```

```
  CALL FreeMem(buffer&,Length&)
  CALL FreeMem(info&,252&)
  LIBRARY CLOSE
  COLOR 3,1:CLS:LOCATE 2,2:PRINT "Ready."
  WHILE INKEY$="":WEND
  SYSTEM
SUB WRITEERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Write-error."
  ShowCont
END SUB
SUB DWRITE(adr&,Length&) STATIC
  FOR i&=1 TO Length&
    PRINT#1,CHR$(PEEK(adr&-1+i&));
  NEXT
END SUB
SUB OPENERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Can't open File."
  ShowCont
END SUB
SUB NEWFILE(filename$) STATIC
  File$=filename$+"-RL"
  OPEN File$ FOR OUTPUT AS 1
END SUB
SUB FORMERR STATIC
COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Not a binary File."
  ShowCont
END SUB
SUB LOADERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Load-error."
  ShowCont
END SUB
SUB LOADFILE(filename$,buffer&,Length&) STATIC
  File$=filename$+CHR$(0) :handle&=xOpen&(SADD(File$),1005)
  IF handle&=0 THEN
    filename$=""
  ELSE
    inBuffer&=xRead&(handle&,buffer&,Length&)
    CALL xClose(handle&)
    IF inBuffer&<>Length& THEN filename$=""
  END IF
END SUB
SUB FILEERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  File not found."
  ShowCont
END SUB
SUB DIRERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2
  PRINT "ERROR:  File is a Directory."
  ShowCont
END SUB
SUB GETINFO(filename$,info&,Length&) STATIC
  File$=filename$+CHR$(0) :DosLock&=Lock&(SADD(File$),-2)
  IF DosLock&=0 THEN
```

```
    Length&=0
  ELSE
    Dummy&=Examine&(DosLock&,info&)
    IF PEEKL(info&+4)>0 THEN
      Length&=-1
    ELSE
      Length&=PEEKL(info&+124)
    END IF
  END IF
  CALL UnLock(DosLock&)
END SUB
SUB LINPUT(filename$) STATIC
  COLOR 3,1:CLS:WINDOW 2,"Filename:",(0,0)-(250,10),0
  WINDOW OUTPUT 1:LOCATE 5,2
  PRINT "Name of a binary saved File";
  LINE INPUT filename$:WINDOW CLOSE 2
END SUB
SUB SELECT(box%) STATIC
Check:
  WHILE MOUSE(0)=0:WEND:x=MOUSE(1):y=MOUSE(2)
  IF y>27 AND y<43 THEN
    IF x>9 AND x<38 THEN box%=0:EXIT SUB
    IF x>177 AND x<238 THEN box%=-1:EXIT SUB
  END IF
  GOTO Check
END SUB
SUB ALLOCERR STATIC
  COLOR 1,3:CLS:LOCATE 2,2:PRINT "ERROR:  Allocation denied."
  ShowCont
END SUB
SUB ShowCont STATIC
  LOCATE 4,2:PRINT "Press SPACE to continue,"
  LOCATE 5,7:PRINT "ESCAPE to exit.";
  WHILE a$<>CHR$(32) AND a$<>CHR$(27)
    a$=INKEY$
  WEND
  IF a$=CHR$(27) THEN SYSTEM
END SUB
SUB REQUEST(disk$) STATIC
  COLOR 3,1:CLS
  LOCATE 2,2:PRINT "INSERT ";disk$;" DISK INTO DRIVE"
  LOCATE 3,14:PRINT "DFO:":LOCATE 5,3:PRINT "OK";
  LOCATE 5,24:PRINT "CANCEL";:LINE(10,28)-(37,42),3,b
  LINE(178,28)-(237,42),3,b
END SUB
```

Variablen

ALLOCERR	Fehler-SUB
AllocMem	exec, reserviert Speicher
Bytes	Länge
DIRERR	Fehler-SUB
DWRITE	Fehler-SUB
DosLock	Handle von Lock
Dummy	nicht benötigter Wert
Examine	DOS, untersucht File
FILEERR	Fehler-SUB
FORMERR	Fehler-SUB
File	Filename + 0 für DOS
FreeMem	exec, gibt Speicher frei
GETINFO	File-Analyse-SUB
LINPUT	Eingabe-SUB
LOADERR	Fehler-SUB
LOADFILE	SUB, lädt File in Puffer
Length	File-Länge
Lock	DOS, Saugfuß auf File
NEWFILE	SUB, Ziel-File öffnen
OPENERR	Fehler-SUB
REQUEST	Rückfrage bei Fehler, SUB
SELECT	Auswahl-SUB
ShowCont	Menu-SUB
UnLock	DOS, Saugfuß lösen
WRITEERR	Fehler-SUB
a	Hilfsvariable
adr	Adresse
b	Hilfsvariable
box	Hilfsvariable
buffer	Adresse des File-Buffers
disk	Diskette
filename	Programmname
handle	Adresse des File-Handlers
i	Hilfsvariable
inBuffer	tatsächlich gelesene Bytes
info	Adresse der File-Info-Struktur
offs	Offset
pointer	Hilfsvariable
x	dito
xClose	DOS, schließt File
xOpen	DOS, öffnet File
xRead	DOS, liest File
y	Hilfsvariable

5.3.5 Variablen auflisten

Kennen Sie das: Sie sehen sich das Listing eines älteren BASIC-Programms an, weil Sie gern wüßten, wie Sie damals dieses oder jenes Problem gelöst haben. Da es in der menschlichen Natur liegt, nicht mehr zu arbeiten als nötig, haben Sie entweder auf eine ausführliche Dokumentation verzichtet, oder Sie finden diese unter meterdicken Stapeln von Programmausdrucken nicht mehr wieder.

Gerade beim modularen Programmieren, bei dem Sie viele Kurzroutinen auf einer Diskette gesammelt haben, um sie nun in Ihr Hauptprogramm einzubinden, ist eine Dokumentation dieser Kurzroutinen unerläßlich. Auch verlangen viele Zeitschriften, in denen Sie Ihre Listings abdrucken lassen möchten, ausführliche Dokumentationen. An dieser Stelle fällt natürlich viel Arbeit für den Programmierer an. Nun, wir Programmierer wären keine solchen, würden wir nicht einfach ein Programm schreiben, das uns den Großteil dieser Arbeit abnimmt.

Das vorliegende Programm ist daher nicht nur als Beispiel anzusehen, wie man die Variablenliste ausliest und dabei Label-Namen ausschließt, sondern im Gegenteil äußerst brauchbar für Dokumentationszwecke. So ist es beispielsweise vorstellbar, ein mit diesem Programm erzeugtes ASCII-File einfach an das betreffende Programm anzuhängen (MERGE) und so die Zettelwirtschaft abzubauen bzw. das lästige Eingeben in eine Dateiverwaltung zu vermeiden.

Dazu muß allerdings an dieser Stelle gesagt werden, daß von dem Variablen-List-Programm weder angegeben wird, ob es sich bei einem Variablennamen um ein SUB-Programm der eine Betriebssystemroutine handelt, noch wird angegeben, um welchen Variablentyp es sich handelt. Es gibt da so viele Möglichkeiten, allein schon mit den Befehlen DEFxxx, z.B. DEFINT a-c. So verwenden Sie im Programm beispielsweise die Variable Anton$, die dann in der Liste unter Anton steht. Sollten Sie sich daran

stören, daß das Programm beim Sortieren nicht zwischen Groß- und Kleinschreibung unterscheidet, entfernen Sie die vier UCASE$() nach dem Label "ausgabe:".

Wichtig: Beachten Sie die Konventionen für binär gespeicherte Programme, die in den vorigen Kapiteln dargestellt wurden!

```
' ####################################
' # V a r i a b l e n - L i s t  Amiga #
' #----------------------------------#
' #     (W) 1987 by Stefan Maelger     #
' ####################################
'
' "dos.bmap" und "exec.bmap" muessen auf
' Disk vorhanden sein
' ------------------------------------
'
  CLEAR,50000&
  DECLARE FUNCTION AllocMem& LIBRARY
  DECLARE FUNCTION Lock&     LIBRARY
  DECLARE FUNCTION Examine&  LIBRARY
  DECLARE FUNCTION xOpen&    LIBRARY
  DECLARE FUNCTION xRead&    LIBRARY
  LIBRARY ":exec.library"
  LIBRARY ":dos.library"
  WINDOW CLOSE WINDOW(0)
  DIM varname$(2000),var%(2000),fehler$(5)
  FOR i=0 TO 5:READ fehler$(i):NEXT
DATA "File contains no binary."
DATA "Read-Error.","File open error."
DATA "File is a directory.","File not found."
DATA "Allocation denied."
nextTry:
  REQUEST "Plug Disk into Drive df0.",1,"OK","",flag%
  WINPUT filename$
  CHECKFILE filename$,buffer&
  IF buffer&<0 THEN
    e%=6+buffer&
    REQUEST fehler$(e%),2,"CANCEL","QUIT",flag%
    IF flag%=2 THEN LIBRARY CLOSE:SYSTEM
    GOTO nextTry
  END IF
  pointer&=buffer&+1
ReadLine:
  SETPOINTER pointer&,flag%
  IF flag%=1 GOTO ReadNames
ReadToken:
  CHECKTOKEN pointer&,number%
  IF number%<0 GOTO ReadLine
```

```
  var%(number%)=1:GOTO ReadToken
ReadNames:
  current%=0
searching:
  IF PEEK(pointer&)=0 OR PEEK(pointer&)>&H60 THEN
    pointer&=pointer&+1:GOTO searching
  END IF
getlength:
  length%=PEEK(pointer&)
  IF length%=0 GOTO ausgabe
  FOR i%=1 TO length%
    pointer&=pointer&+1
 varname$(current%)=varname$(current%)+CHR$(PEEK(pointer&))
  NEXT
  current%=current%+1
  pointer&=pointer&+1:GOTO getlength
ausgabe:
  flag%=1:begi%=0:ende%=current%-2
  WHILE flag%=1
    flag%=0
    FOR i%=begi% TO ende%
      IF UCASE$(varname$(i%))>UCASE$(varname$(i%+1)) THEN
        SWAP varname$(i%),varname$(i%+1)
        SWAP var%(i%),var%(i%+1)
        flag%=1
      END IF
    NEXT
    begi%=begi%+1:flag%=0
    FOR i%=ende% TO begi% STEP -1
      IF UCASE$(varname$(i%))<UCASE$(varname$(i%-1)) THEN
        SWAP varname$(i%),varname$(i%-1)
        SWAP var%(i%),var%(i%-1)
        flag%=1
      END IF
    NEXT
    ende%=ende%-1
  WEND
ausgabe2:
  BEEP
  REQUEST "List to Screen?",2,"YES","NO",sflag%
  REQUEST "List to Printer?",2,"YES","NO",pflag%
  REQUEST "Save as ASCII-File?",2,"YES","NO",fflag%
  IF sflag%=2 AND pflag%=2 AND fflag%=2 GOTO ausgabe2
  IF sflag%=1 THEN WINDOW 2,"Variables:",(0,0)-(240,180),31
  IF fflag%=1 THEN
    OPEN filename$+".V" FOR OUTPUT AS 1
    PRINT#1,CHR$(10);"Variablen-Liste:";
    PRINT#1,CHR$(10);"---------------";CHR$(10);CHR$(10);
  END IF
  IF pflag%=1 THEN
    LPRINT "Variablen-Liste von:"
    LPRINT filename$:LPRINT
  END IF
```

```
  FOR i%=0 TO current%-1
    IF var%(i%)=1 THEN
      IF sflag%=1 THEN PRINT varname$(i%)
      IF pflag%=1 THEN LPRINT varname$(i%)
      IF fflag%=1 THEN PRINT#1,varname$(i%);CHR$(10);
    END IF
  NEXT
  IF fflag%=1 THEN CLOSE 1
  REQUEST "Ready.",1,"OK","",flag%
  LIBRARY CLOSE
  SYSTEM

SUB CHECKTOKEN(a&,n%) STATIC
PeekToken:
  t%=PEEK(a&):a&=a&+1
  IF t%=0 THEN strflag%=0:n%=-1:EXIT SUB
  IF strflag%=1 AND t%<>34 GOTO PeekToken
  IF t%>127 THEN
    IF t%>247 THEN a&=a&+1
    GOTO PeekToken
  ELSEIF t%=1 THEN
    n%=CVI(CHR$(PEEK(a&))+CHR$(PEEK(a&+1))):a&=a&+2:EXIT SUB
  ELSEIF t%=2 OR t%=11 OR t%=12 OR t%=28 THEN
    a&=a&+2:GOTO PeekToken
  ELSEIF t%=15 THEN
    a&=a&+1:GOTO PeekToken
  ELSEIF t%=29 OR t%=30 THEN
    a&=a&+4:GOTO PeekToken
  ELSEIF t%=31 THEN
    a&=a&+8:GOTO PeekToken
  ELSEIF t%=3 OR t%=14 THEN
    a&=a&+3:GOTO PeekToken
  ELSEIF t%=34 THEN
    IF strflag%=1 THEN strflag%=0 ELSE strflag%=1
    GOTO PeekToken
  ELSE
    GOTO PeekToken
  END IF
END SUB

SUB SETPOINTER(a&,f%) STATIC
  IF PEEK(a&+1)=0 THEN f%=1 ELSE f%=0
  IF PEEK(a&)=0 THEN a&=a&+3 ELSE a&=a&+5
END SUB

SUB CHECKFILE(a$,f&) STATIC
  i&=AllocMem&(252&,65538&)
  IF i&=0 THEN
    f&=-1:EXIT SUB
  ELSE
    b$=a$+CHR$(0):l&=Lock&(SADD(b$),-2)
    IF l&=0 THEN
      f&=-2:EXIT SUB
```

```
    ELSE
      s&=Examine&(l&,i&)
      IF PEEKL(i&+4)>0 THEN
        f&=-3:CALL UnLock(l&):EXIT SUB
      ELSE
        f&=PEEKL(i&+124):CALL UnLock(l&)
        CALL FreeMem(i&,252&):v&=f&+3
        c&=AllocMem&(v&,65537&)
        IF c&=0 THEN
          f&=-1:EXIT SUB
        ELSE
          h&=xOpen&(SADD(b$),1005)
          IF h&=0 THEN
            f&=-4:EXIT SUB
          ELSE
            r&=xRead&(h&,c&,f&):CALL xClose(h&)
            IF r&<>f& THEN
              f&=-5:EXIT SUB
            ELSE
              f&=c&
              IF PEEK(f&)<>&HF5 THEN f&=-6:EXIT SUB
            END IF
          END IF
        END IF
      END IF
    END IF
  END IF
END SUB
SUB WINPUT (a$) STATIC
  WINDOW 1,"Input: Filename",(0,0)-(240,8),0
  LINE INPUT a$
  WINDOW CLOSE 1
END SUB
SUB REQUEST(a$,m%,b$,c$,b%) STATIC
  WINDOW 1,"System Request",(0,0)-(240,40),22
  COLOR 0,1:CLS:LOCATE 2,(30-LEN(a$))\2:PRINT a$;:COLOR 1,0
  IF m%=1 THEN
    l%=LEN(b$)/2:LOCATE 4,15-l%:PRINT " ";b$;" ";
  ELSEIF m%=2 THEN
    LOCATE 4,2:PRINT " ";b$;" ";:LOCATE 4,27-LEN(c$)
    PRINT " ";c$;" ";
  END IF
maus:
  WHILE MOUSE(0)<>0:WEND
  WHILE MOUSE(0)=0:WEND
  x%=(MOUSE(1)+8)\8:y%=(MOUSE(2)+8)\8:b%=0
  IF y%=4 THEN
    IF m%=1 THEN
      IF x%>14-l% AND x%<17+l% THEN b%=1
    ELSEIF m%=2 THEN
      IF x%>1 AND x%<LEN(b$)+4 THEN b%=1
      IF x%>26-LEN(c$) AND x%<30 THEN b%=2
    END IF
```

```
  END IF
  IF b%>0 THEN
    WINDOW CLOSE 1
    EXIT SUB
  END IF
  GOTO maus
END SUB
```

Eine Variablenliste dürfte sich hier wohl erübrigen. Mittels dieses Programms wurden übrigens die Variablen-Listen einiger Programme in diesem Buch erstellt. Sie sehen, wie praktisch es ist, sich eine solche Liste an Ihr Programm anzufügen.

5.3.6 Variablenmüll beseitigen

Haben Sie sich manchmal gefragt, weshalb ein binär abgespeichertes BASIC-Programm, das Sie geladen und etwas verkürzt haben, nach einem erneuten Abspeichern meist noch länger ist als vorher, nie aber kürzer? Traten beim Programmlauf eines BASIC-Programms schon einmal Fehler auf, bei denen im List-Fenster um ein paar Leerzeichen ein orangefarbenes Kästchen gemalt wurde und der Interpreter auch noch von Ihnen verlangte, daß Sie einen Syntax-Error bestätigen? Ärgern Sie sich auch, wenn Sie sich mit einem File-Monitor ein binäres Programm anschauen und vor lauter Müll, der da im Programm abgespeichert ist, kaum noch etwas erkennen können? Wird vielleicht Ihr Superprogramm nach jedem Editieren, das dem Zwecke der Programmbeschleunigung dient, immer langsamer? Oder treten ständig Internal-Errors bei dem Versuch auf, ein binäres Programm zu manipulieren?

Sollte eines dieser Problemchen auf Sie zutreffen, dann habe ich eine Lösung parat. In der Tat verpfuscht der AmigaBASIC-Interpreter von Abspeichern zu Abspeichern das binäre Programm mehr und mehr. Ständig fügt er weiteren Müll dazu - und das an Stellen im Programm, die Sie kaum für möglich halten. Das alles liegt daran, daß der gesamte Speicherbereich, der für das Programm inklusive Variablentabelle vorgesehen ist, einfach so, wie er ist, abgespeichert wird, ohne vorher zu überprüfen, ob tatsächlich alles, was in diesem Bereich liegt, zu dem Programm

gehört. Meist ist das nicht der Fall, da vergessen wurde, wichtige Pointer des Interpreters hin und wieder einmal zu aktualisieren - und das nicht einmal vor dem Abspeichern (oder Neuladen) eines Programms.

Es gibt da aber wie immer ein Hintertürchen: Wenn Sie ein Programm als ASCII-File abspeichern, so wird einfach genau das in dieses File geschrieben, was auch im List-Fenster zu sehen ist, zu gut deutsch: alles im (ASCII-)Klartext, nur getrennt durch Line-Feed (CHR$(10)=nächste Zeile). Speichern Sie daher Ihr Programm einfach mit dem Zusatz ",A" ab. Nun dürfen Sie es aber nicht gleich wieder laden! Der ganze Interpreter-Müll ist schließlich noch im Speicher, und selbst ein NEW kann hier nichts retten, da die Pointer nicht aktualisiert werden. Es bleibt also nur eines: Quit anwählen und sich damit abfinden, den Interpreter-Block noch einmal zu laden. Danach aber sofort das Programm laden und auch sofort wieder speichern. Den Zusatz ",b" dürfen Sie allerdings nicht vergessen, da Sie mit dem Laden eines ASCII-Files die Voreinstellung der SAVE-Routine geändert haben. Das auf diese Weise entstandene Binärprogramm ist jetzt absolut sauber.

Merken Sie sich!

1. Am besten ist es, wenn Sie unfertige Programme immer zunächst als ASCII-Files abspeichern. Erst nach der Fertigstellung den Interpreter neu laden und das Programm nach dem Laden als erstes gleich als binäres File wieder abspeichern.

2. Treten mit einem Programm Probleme auf, die logisch nicht mehr erklärbar sind, ist zur Rettung des Programms denkbar, es zunächst als ASCII-File zu speichern, danach wie unter Punkt 1 zu verfahren. Klappt dies nicht mehr... (verfahren Sie also lieber nach 1).

3. Das Schlimmste, was Sie tun können, ist das binäre Speichern eines Programms, das bereits einen Testlauf hinter sich hatte und dabei mit einer Fehlermeldung ausgestiegen ist (und möglichst vorher noch Betriebssystemroutinen nutzte). Dann fabriziert der Interpreter den meisten Müll.

4. Läuft Ihr Programm nicht, liegt es häufig nicht an Programmier- oder Denkfehlern Ihrerseits, sondern an den Entwicklern von AmigaBASIC.

5.3.7 Selbstmodifizierende Programme

Wir wollen jetzt daran gehen, Programme sich während des Programm-Ablaufes selbst verändern zu lassen. Zu diesem Zweck kann ich hier zwei Möglichkeiten aufzeigen, dies zu bewerkstelligen. Die erste Möglichkeit ist das direkte Verändern des Programms im Speicher durch POKEn. Wenn Sie nicht nur Strings ändern wollen, dann gehört dazu eine sehr gute Kenntnis von binären Programmen.

Das Prinzip ist relativ einfach: An irgendeiner Stelle im Programm weisen Sie einer Stringvariablen eine Zeichenkette zu. Wichtig ist, daß diese Stringvariable vor dem Eingriff keinerlei Veränderungen unterworfen wurde, wie A$=A$+CHR$(0). Solange das nicht der Fall war, werden die Variablen-Pointer direkt auf die Zeichenkette in Ihrem Programm gelenkt. Hier zunächst das Beispiel für die Änderung eines Strings im Programm. In dieser Routine wird ein Fenster geöffnet, das die genannte Zeichenkette als Titelleiste trägt. Durch Anwahl des Menüpunktes "Ändern" kann ein neuer Window-Titel eingegeben werden, der dann nach einem erneuten Laden bzw. Starten des Programms sofort erscheint. Eine Art von Preferences für Window-Titel also.

```
REM ##################################################
REM #    S e l b s t m o d i f i z i e r u n g  I    #
REM #-------------------------------------------------#
REM #        (W) 1987 by Stefan Maelger, Hamburg       #
REM ##################################################
REM * Hier steht der änderbare String:
Titel$="Selbstmodifizierung I"
SCREEN 1,320,200,2,1
WINDOW 2,Titel$,,16,1
MENU 1,0,1,"ÄNDERN"
MENU 1,1,1,"TITEL"
ON MENU GOSUB checkmenu
MENU ON
WHILE Maelger=0
```

```
SLEEP
WEND
WINDOW CLOSE 2
SCREEN CLOSE 1
END
checkmenu:
IF MENU(1)=1 AND MENU(0)=1 GOTO neuertitel
RETURN
neuertitel:
PRINT "Bitte neuen Titel eingeben"
PRINT LEN(Titel$);"Zeichen."
LINE INPUT neu$
neu$=LEFT$(neu$+SPACE$(LEN(Titel$)),LEN(Titel$))
REM * Hier wird der String geändert:
FOR i=1 TO LEN(neu$)
  POKE SADD(Titel$)+i-1,ASC(MID$(neu$,i,1))
NEXT
REM * Programm wieder abspeichern (den neuen Titel)
PRINT "Voreinstellung wird gesichert."
SAVE "Programmname"
PRINT "Voreinstellung ist gesichert."
PRINT "Laden oder starten Sie dieses"
PRINT "Programm nun erneut."
t=TIMER+15:WHILE t>TIMER:WEND
Maelger=1
RETURN
```

Sie sehen, wie einfach das geht. Anstelle von "Programmname" speichern Sie natürlich das Programm unter dem Namen, den es zuvor hatte. Bei dieser Methode läßt sich natürlich auch ein Befehl ändern, dazu muß man jedoch ganz genau über binäre Programme Bescheid wissen, denn jedes Programm liegt binär im Speicher! Da das so ist, funktioniert die obige Methode natürlich auch bei protected gespeicherten Files. Hierbei müssen Sie es dann nur wieder als ",p"-Programm speichern.

Wir kommen nun zu der zweiten Methode, der ASCII-File-Methode. Auch hier wird ein Programm völlig verändert. Der Clou bei dieser Geschichte ist, daß es völlig problemlos ist, ganze Programmteile zu ändern. In einem Programm, das in binärem Format vorliegt, kann das POKEn natürlich böse Folgen haben, zumal es nicht so leicht ist, Befehle zu ersetzen. Wir gehen daher den Umweg über ASCII-Files, die sich ganz einfach erzeugen lassen. Zunächst wieder das Prinzip.

Erst wird festgelegt, wie der zu ersetzende Programmteil später aussehen soll. Bei Benutzereingaben sollten Sie eine eingehende Syntaxprüfung vornehmen. Im laufenden Programm werden nun die zu ändernden Programmzeilen gelöscht (DELETE von - bis). Dann wird das Programm als ASCII-File auf einer Diskette abgelegt, wenn die Veränderung bleibend sein soll, natürlich unter seinem eigentlichen Namen, ansonsten bietet sich aus Geschwindigkeitsgründen die RAM-Disk an. Nun öffnen wir das gerade gespeicherte Programm zum Anfügen (OPEN x$ FOR APPEND AS y) und verfahren wie bei dem DATA-Generator mit der Erzeugung des neuen Programmteils.

Um dieses Programm nun aber auch im Speicher zu haben, genügt ein "RUN filename$" oder ein "LOAD filename$,r". Dummerweise jedoch startet damit das Programm völlig neu, so daß es je nach Anwendung günstiger sein kann, den neuen Programmteil als ASCII-File in der RAM-Disk zu erzeugen und ihn mit CHAIN MERGE anzuhängen. Man kann aber auch wie oben beschrieben verfahren und damit die Variablenwerte erhalten. Dann kann man das Startlabel bestimmen und das geänderte Programm mittels "CHAIN blabla,Zeile,ALL" neu starten. Hier wieder ein kleines Beispiel:

```
REM ##################################################
REM #    S e l b s t m o d i f i z i e r u n g  II   #
REM #--------------------------------------------------#
REM #        (W) 1987 by Stefan Maelger, Hamburg       #
REM ##################################################
REM * Auflösung des Screens holen
GOSUB variablesLabel
SCREEN 1,breite%,hoehe%,tiefe%,modus%
WINDOW 2,"Hallo!",,0,1
PRINT "Breite in Pixel:";breite%
PRINT "Höhe   in Pixel:";hoehe%
PRINT "Tiefe in Planes:";tiefe%
PRINT
PRINT "Bitte geben Sie die neue"
PRINT "Breite ein:";
INPUT breiteneu%
IF breiteneu%<20 OR breiteneu%>640 THEN
  breiteneu%=breite%
END IF
INPUT "Neue Höhe :";hoeheneu%
IF hoeheneu%<10 OR hoeheneu%>512 THEN
  hoeheneu%=hoehe%
```

```
END IF
INPUT "Neue Tiefe:";tiefeneu%
IF tiefeneu%<1 OR tiefeneu%>5 THEN
  tiefeneu%=tiefe%
END IF
PRINT
modus%=1
IF breiteneu%>320 THEN modus%=2
IF hoeheneu%>256 THEN modus%=modus%+2
IF modus%=4 AND tiefeneu%>2 THEN
  tiefeneu%=2
ELSEIF modus%>1 AND tiefeneu%>4 THEN
  tiefeneu%=4
END IF
DELETE variablesLabel-variablesLabelende
SAVE "Programmname",A
OPEN "Programmname" FOR APPEND AS 1
PRINT#1,"variablesLabel:";CHR$(10);
PRINT#1,"breite%=";STR$(breiteneu%);CHR$(10);
PRINT#1,"hoehe%=";STR$(hoeheneu%);CHR$(10);
PRINT#1,"tiefe%=";STR$(tiefeneu%);CHR$(10);
PRINT#1,"modus%=";STR$(modus%);CHR$(10);
PRINT#1,"RETURN";CHR$(10);
PRINT#1,"variablesLabelende:";CHR$(10);
CLOSE 1
WINDOW CLOSE 2
SCREEN CLOSE 1
LOAD "Programmname",R
END
variablesLabel:
breite%= 320
hoehe%= 200
tiefe%= 2
modus%= 1
RETURN
variablesLabelende:
```

Wunderbar, nicht wahr? Besonders geeignet ist dieses Verfahren wohl für jede Art von Grafikprogrammen, z.B. die Eingabe von benutzerdefinierten Funktionen bei einem Funktionsplot, den Palette-Werten bei einem Malprogramm usw.

Lassen Sie Ihrer Phantasie ruhig freien Lauf. Und überlegen Sie sich einmal, wenn Sie ein Programm schreiben, wie angenehm es sein kann, wenn es nach dem Laden gleich auf die individuellen Wünsche des Benutzers eingestellt ist. Das erspart Zeit und Ärger, und Ihr Programm wird sicherlich dadurch aufgewertet. Zumal Superprogramme wie Deluxe Paint II nicht einmal diesen

Luxus bieten (!). Und seien wir ehrlich: Wenn man für einige wenige Funktionen von DPaint Maschinenspracheroutinen einbaut, sollte es keine Schwierigkeit bedeuten, dieses Programm von BASIC aus zu übertreffen.

Wer mehr über die Grafikmöglichkeiten von AmigaBASIC im Zusammenhang mit Betriebssystemroutinen wissen möchte, dem sei hier wärmstens das DATA BECKER Buch "Amiga Supergrafik" empfohlen.

6. Die Workbench

6.1 Arbeiten mit der Workbench

Die Benutzeroberfläche des Amiga läßt in der Bedienung eigentlich keine Wünsche offen. Alle wichtigen Operationen lassen sich durch Icons realisieren. Diese bildlichen Aktionen machen Texteingaben größtenteils unnötig. Deshalb ist dem Verständnis keine Barriere gesetzt, wie es ja oft bei Sprachen vorkommt.

Allerdings gibt es einige Funktionen, die nicht oft gebraucht werden. Sie geraten in Vergessenheit, obwohl damit manches Problem viel einfacher hätte gelöst werden können! In diesem Abschnitt wird deshalb gezeigt, wie man effektiv alle Möglichkeiten ausnutzen kann und somit nicht selten Zeit und umständliche Wege spart.

6.1.1 Tastaturtricks

Die Bedienung von String-Gadgets

Man merkt oft gar nicht, wie oft man mit den Mitteln Intuitions in Berührung kommt. Doch viel zu häufig weiß man diese Mittel gar nicht richtig auszunutzen und macht sich somit Umstände, die nicht nötig wären. Auf der Workbench kommt man nur bei einer Gelegenheit mit den String-Gadgets in Konflikt. Das ist der Moment, wenn man einem File oder einer Diskette einen neuen Namen geben möchte. Nach dem Aufrufen der Rename-Funktion erscheint der waagerechte Kasten, in dem schon der alte Name steht. Nun können wir einen neuen eingeben, und dabei helfen uns folgende Tasten:

Für ganz Hartnäckige, denen der ganze Name nicht mehr gefällt, ist eine Tastenkombination vorgesehen, die einen davor bewahrt, jedes Zeichen einzeln mit DEL zu löschen. Es handelt sich dabei um die Kombination Amiga-X (Delete). Probieren Sie es ruhig einmal aus! Sofort ist das Eingabefeld gelöscht, und neuer Text

kann eingegeben werden. Nun gibt es Leute, denen fällt erst ein, daß sie doch ein ganz anderes Programm umbenennen wollten, nachdem sie den ganzen Text gelöscht haben. Das ist Pech! Aber auch an jene wurde gedacht. Mit einem kurzen Druck auf Amiga-Q ruft man nämlich den Inhalt eines Zwischenspeichers ab, in dem der ursprüngliche Name gesichert ist. Diese Funktion nennt man Undo.

Für alle mit einem Hang zu langen File-Namen wurden auch zwei Tastenkombinationen eingeführt. Die eine, SHIFT + Cursor rechts, setzt den Cursor an das Ende des Textes. Umgekehrt verfährt SHIFT + Cursor links. Dann befindet sich unser Schreiberling wieder am Anfang der Zeile.

Auto-Requester mit der Tastatur beantworten

Auch die Auto-Requester der Workbench, sie werden als System-Requester bezeichnet, können mittels der Tastatur beantwortet werden. Und zwar wurde jedem der beiden Gadgets eine Tastenkombination zugeteilt. Mit Amiga-B wählt man das Cancel-Gadget an und mit Amiga-V das Retry-Gadget. Somit können Sie ganz einfach mit der linken Hand den Requester beantworten.

Die Entwicklung des Amiga

Jeder, der etwas geschaffen hat, möchte dafür gerühmt werden. Das ist auch ein Grund, warum z.B. auf dem Buch-Cover unsere Namen vermerkt sind. Genauso ging es natürlich auch den Entwicklern des Amiga. Viele haben mitgeholfen und wollen deshalb auch erwähnt werden.

Es haben sich deshalb alle, wie man so schön sagt, verewigt. Wie kommen wir an diese Verewigung heran? Nun, dazu gibt es eine Tastenkombination, die man gedrückt halten muß, während die Workbench aktiv ist. Und zwar müssen beide ALT- und beide SHIFT-Tasten gedrückt werden. Nachdem Sie dieses Kunststück geschafft haben, müssen Sie mit einem der noch freien Finger in Richtung der Funktionstasten langen. Alle 10 sind mit neuen Texten belegt!

Hier ist eine Aufstellung der Texte, die wir bisher herausgefunden haben (Wenn es verschiedene Texte auf verschiedenen Amiga-Modellen gab, so sind diese bei der gleichen Funktionstaste zu finden):

F1:	"System Software: Carl, Neil & Kodiak"
F2:	"Graphics Software: Dale, Bart, Jim & RJ"
	"Graphics Software: Dale, Bart, Jim & =RJ="
F3:	"QA: Jon, Bruce, Stan, Kim & Jerry"
F4:	"LG Support: Karen, Dave, Cheryl & Nancy"
	"LG Support: Caryn, Dave, Victor, Terry, Cheryl & Nancy"
F5:	"CBM Software: Andy, Barry & Eric"
	"CBM Software: Andy, Barry, Dave & Eric"
F6:	"Pics: Sheryl & Jack"
F7:	"Docs: Rick, Mitch, Peggy & Rob"
F8:	"Chips: Jay, Akio, Glenn, Edwin, Marc & Dave"
F9:	"HW: Dave, Bill, ChrisR & Josh"
F10:	"Moral Support: Joe Pillow & The Dancing Fools"

6.1.2 Der Mülleimer

Beim Arbeiten wird - wie überall - nicht nur Brauchbares produziert! Daran haben natürlich auch die Entwickler des Betriebssystems gedacht, haben einen Mülleimer (Trashcan) geschaffen. Doch finden Sie diesen Weg nicht auch etwas umständlich?! Erst das Icon anklicken, dann auf den Mülleimer bewegen. Aber damit ist die Prozedur noch nicht ausgestanden! Denn der Speicher auf der Diskette ist ja noch lange nicht freigegeben. Erst durch das Auswählen des Menüs "Disk" kann mit "Empty Trash" - welch liebevolle Bezeichnung - endlich der Platz frei gemacht werden.

Es geht aber auch wesentlich einfacher! Doch der oben beschriebene Weg wurde aus verständlichen Gründen gewählt. Durch die vielen Schritte, die gegangen werden müssen, ist es so gut wie ausgeschlossen, daß durch Unachtsamkeit etwas gelöscht wird, das nicht gelöscht werden soll. Außerdem glaubte man, daß 880 KByte Diskettenkapazität es erlauben, auch nicht benö-

tigte Daten noch eine Weile zu speichern. Doch nach der Anschaffung des Amiga ist oft nicht viel Geld für Qualitätsdisketten übriggeblieben. Diskettenspeicherplatz wird dadurch kostbar!

Nun aber endlich zum einfacheren Weg. Erinnern Sie sich noch an den Punkt "Discard" im Workbench-Menü? Hier liegt die schnellere Version der Löschfunktion "begraben"! Man markiert einfach die zu löschenden Files mit einem Mausklick und wählt dann "Discard". Man wird noch einmal durch einen System-Requester darauf hingewiesen, daß diese Funktion endgültig ist, doch dann geht's rund. Diskette frei! File weg! Einfacher und schneller.

6.1.3 Mehrfachaktivierung

Sicherlich bekommt jeder Besitzer des Amiga einmal die Idee, bei jedem Window auf akkurat geordnete Icons zu achten. Legen Sie dazu Ihre BASIC-Diskette ins Laufwerk. In der Schublade BASICDEMOS liegen ca. 25 Icons, deren Namen leider so unglücklich lang sind, daß mit CleanUp nichts mehr zu erkennen ist.

Nun hat man einiges an Arbeit vor sich! Nach dem Ordnen in einer Tabelle muß jedes Icon einzeln angeklickt werden und dann noch der Punkt "Snapshot" aus dem "Special-Menü" ausgeführt werden. Nach vorsichtigen Hochrechnungen sitzen Sie sehr lange daran! Auch hier gibt es natürlich einen viel einfacheren Weg, doch diesen überliest man sehr leicht im Anwender-Handbuch! Deshalb soll diese arbeitserleichternde Funktion auf keinen Fall vergessen werden.

Wenn beim Anklicken gleichzeitig auf die Shift-Taste gedrückt wird, so ist nicht nur das neue Icon aktiv, sondern auch das alte Objekt bleibt gewählt. Praktisch können so alle erreichbaren Icons gleichzeitig aktiviert werden. Man muß nur darauf achten, daß auf alle die gleiche Funktion ausgeübt werden kann. Eine

Löschfunktion kann z.B. nicht durchgeführt werden, wenn sich unter den Icons ein Symbol für eine Diskette befindet. Doch um das obige Beispiel nicht aus den Augen zu verlieren:

Nach dem richtigen Positionieren aller BASIC-Icons wählt man diese nacheinander mit der Mehrfachaktivierung an. Darauf wird der Menüpunkt "Snapshot" ausgewählt, und für jedes Icon speichert die Workbench die Position ab. Dies dauert zwar etwas, ist aber einfacher, als diese Operation an jedem Symbol einzeln durchzuführen.

Will man mehrere Programme kopieren, hilft diese Möglichkeit der Mehrfachaktivierung auch weiter. Denn man kann so alle Icons gleichzeitig über die Screen und damit auch in Windows anderer Disketten bewegen! Einziger Nachteil, der nicht verschwiegen werden soll, ist, daß der Diskettenwechsel für jedes Programm einzeln durchgeführt werden muß.

Wollen Sie auch noch den häufigen Diskettenwechsel bei einem Laufwerk umgehen, so empfehle ich folgende Methode: Kopieren Sie zuerst die "Empty"-Schublade der Workbench auf die Quelldiskette. Das sollten Sie bei jeder Diskette gleich nach dem Formatieren machen! Dann bewegen Sie alle Icons, die kopiert werden sollen, in diese Schublade. Auch hierfür empfehle ich die Mehrfachaktivierung, wenn es viele sind.

Beachten Sie nur, daß die Position Ihres Mauscursors entscheidend ist für das Ziel. Der Rest der Icons kann überall hin verschoben werden: außerhalb des Windows oder sogar außerhalb des Screens. Wichtig ist nur die Stelle, an der der Pfeil steht. Sind alle Programme in der Schublade verstaut, so bewegen Sie diese ganz einfach auf oder in die Zieldiskette. Dafür brauchen Sie nur noch einen Kopierdurchgang.

6.1.4 Infos

Die Info-Funktion im Workbench-Menü soll Auskünfte über Programme und Daten-Files geben. Doch welche Informationen

kann man ihr überhaupt entnehmen? Wozu kann man diese Informationen ändern? Dies sind einige Fragen, auf die das Anwender-Handbuch keine Antwort gibt! Ein Grund mehr, hier darüber ausführlich zu berichten:

6.1.4.1 Das Info-Feld

Der Aufruf erfolgt durch das Anklicken eines Icons irgendeiner Art - die verschiedenen Arten werden später noch beschrieben - woraufhin der Menüpunkt "Info" angewählt wird. Nachdem von der Workbench-Diskette nachgeladen wurde, gibt der Amiga ein Fenster mit allen notierten Informationen aus. Ist einmal "Info" aufgerufen, so muß später die Workbench-Diskette nicht wieder eingelegt werden.

Das Info-Feld ist in mehrere Bereiche aufgeteilt. Links oben findet man allgemeine Angaben zum File bzw. zur Diskette: der Name, der Typ und die Größe in zwei verschiedenen Einheiten. Unter Stack ist vermerkt, wieviel Bytes das File für sich als Arbeits- oder Datenspeicher beansprucht. Der Typ ist entscheidend für die Eigenschaften des Icons, was gleichzusetzen ist mit denen des Files/der Diskette. Möglich ist hier Disk, Drawer, Tool, Project oder Garbage.

Disk steht für die Disketten-Icons, die als einzige nicht in Directory-Windows bewegt werden dürfen. Interessant ist nur die Tatsache, daß auch dieses Icon wie jedes andere grafisch neu definiert werden kann! Nostalgiefreaks könnten doch mal die gute alte 5¼-Zoll-Diskette zeichnen.

Drawer repräsentiert Schubladen oder Unter-Directories. Auf Drawer bewegte Programme werden so ganz einfach in ein anderes Inhaltsverzeichnis gelegt, wenn sie sich auf der gleichen Diskette befanden. Diese Operation dauert aber sehr lange! Deshalb mein Tip: Besser ist es, den Rename-Befehl des AmigaDOS zu verwenden. Geben Sie den ersten Namen mit voller Pfadbezeichnung ein, und geben Sie beim neuen Namen eine andere

Pfadfolge an, so finden Sie das Programm ab jetzt dort. Vergessen Sie aber nicht, das Icon auch mit dem neuen Pfad zu versehen.

Als Tool wird jedes Programm bezeichnet. Programme können ebenso wie Schubladen sowohl in Windows als auch auf dem Arbeitstisch liegen. Tools sind die einzigen Icons, die durch Anklicken ausgeführt werden!

Unter Project versteht der Amiga jedes File, das Daten enthält, die von einem Programm abgespeichert wurden. Als Beispiel kann hier ein Text des Notepads oder einer anderen Textverarbeitung angeführt werden. Aber auch BASIC-Programme oder die .bmap-Files gelten als Projekte!

Der letzte File-Typ ist Garbage. Dies ist eine besondere Art von Schublade. Denn normalerweise können alle Schubladen auch in andere gelegt werden. Doch ein Garbage - das entspricht dem Mülleimer - kann nur im Haupt-Directory stehen. Auch kann es nicht auf den Arbeitstisch bewegt werden! Damit soll verhindert werden, daß der Anwender lange nach dem Mülleimer sucht, denn im Haupt-Directory ist er ja immer schnell gefunden.

Auf der rechten Seite findet man oben einen Kasten, der über den Status des Files Auskunft gibt. Hiermit sind die Zugriffsmöglichkeiten des Benutzers gemeint (s.a. AmigaDOS-Handbuch 2-43/List). Da in der Version 1.2 nur der Löschschutz gesetzt werden kann, ist ein Verändern der Lese-, Schreib- und Ausführmöglichkeit nicht vorgesehen. Platz dafür ist allerdings reichlich vorhanden! Wenn man sich Informationen zu einer Diskette holt, ist hier angezeigt, ob sie mit dem Schieber schreibgeschützt wurde. Durch Anklicken kann dieser Zustand aber nicht verändert werden. Hier muß schon die Mechanik bewegt werden.

Für alle stolzen Amiga-Besitzer, die schon die Kickstart- und Workbench-Version 1.3 haben, möchte ich an dieser Stelle noch die neuen Flags erklären. Zuerst einmal sind jetzt das Read- und Write-Flag mit übernommen worden. Man kann also jetzt

einstellen, ob ein File weiterhin gelesen oder beschrieben werden darf. Allerdings wird diese Funktion nur mit dem neuen Kickstart abgefragt. Deshalb können 500er- oder 2000er-Eigentümer dies nicht ohne neues Kickstart-ROM nutzen, auch wenn es sich über die Workbench einstellen läßt.

Ganz neu ist hingegen das Archive-Flag. Es gibt den Weg frei für ein neues, wesentlich benutzerfreundlicheres Backup. Dazu wird bei jedem Schreibzugriff auf ein File das Archive-Flag gelöscht. Benutzt man dann ein Backup-Programm, das Sicherheitskopien von einige Files macht, so wird bei allen kopierten Files dieses Flag gesetzt. Rufen Sie dann erneut das Backup-Programm auf, so kopiert es nur die Files, bei denen das Flag wieder gelöscht wurde. Damit ist das Kopieren von schon gesicherten Files ausgeschlossen! Eine große Zeitersparnis.

Unter den Status-Flags wird in einer Zeile der Kommentar des Benutzers zu diesem File angezeigt. Unter AmigaDOS kann mit FileNote ein bis zu 80 Zeichen langer Text angefügt werden. Diese Funktion wird bei Disketten unterdrückt, da sie nicht mit einem Kommentar versehen werden können.

Das Info-Feld gibt nicht nur zu Programmen oder Disketten Informationen aus. Auch Text- oder Daten-Files können hiermit detaillierter betrachtet werden. Dem Anwender wird mit "Default Tool" angezeigt, welches Programm dieses File erstellt hat oder womit die Diskette kopiert wurde. Außerdem weiß so die Workbench gleich, welches Programm vorher geladen werden muß, wenn das File aufgerufen wird.

In der letzten Zeile stehen die "Tool Types". Damit sind Informationen gemeint, die dem Hauptprogramm übergeben werden. Das Notepad braucht z.B. Angaben darüber, welche Zeichensätze verwendet wurden und welche Größe das Fenster beim Schreiben hatte. Dies wird gespeichert, damit der Anwender später alles genau so wieder vorfindet, wie er es auf Diskette geschrieben hat.

6.1.4.2 Praktische Beispiele zum Info-Feld

Wenn Sie wollen, können Sie gerne einmal die vorhandenen Informationen ändern. Am besten ist es, wenn Sie dazu mit dem Notepad einen Text schreiben und abspeichern. Was Sie unter Comment eingeben, ist ziemlich egal, es ist nur ein Kommentar. Viel wichtiger ist z.B. Default Tool:

Wie oben schon erwähnt, findet man hier den Namen des Programms, mit dem das Daten-File erstellt wurde und das bei einem Aufruf nachgeladen werden soll. Gerade hier liegt die Tücke im Detail! Ein Beispiel: Ich habe zum Arbeiten eine Workbench-Diskette names "User" und eine namens "CLI" angelegt. Die erste entspricht fast der normalen Workbench.

Die zweite habe ich so modifiziert, daß nach dem Start alle wichtigen CLI-Befehle ins RAM kopiert werden und das DOS-Window nicht automatisch verlassen wird. Auf beiden Disketten befindet sich aber das Notepad. Habe ich nun mit der User-Disk einen Text geschrieben, so steht unter Default Tool:

```
"Workbench User:Utilities/Notepad"
```

Will ich den Text also laden, so muß ich die User-Disk einlegen, auch wenn ich gerade mit der CLI-Disk arbeite, die auch das Notepad enthält. Um diesen Mißstand zu beheben, könnte ich nun ganz einfach den Text "Workbench User:" in "SYS:" umwandeln. Schon wird beim Anklicken die aktuelle Workbench angefordert. Denn SYS: ist die allgemeine Bezeichnung für die Diskette, von der das System gebootet wurde. Gleiches gilt für Tool Types. Dort stehen alle Informationen, die das Programm benötigt. Auch diese können Sie nach Belieben ändern. Zuerst einmal die Bedeutungen:

Name	Beispiel	Bedeutung
FILETYPE	notepad	Zeigt an, daß es ein Text ist.
FONT	topaz. 8	Gibt den Global-Font an.
WINDOW	0,0,50,50	Gibt die Window-Koordinaten an.
FLAGS	NOGLOBAL	Gibt einige Flags an.

Die Zeile FILETYPE muß immer so aussehen, damit das Programm sein File identifiziert. Mit FONT geben Sie den Namen des Global-Fonts an, hier können Sie jeden wählen, der auf der Workbench-Diskette zu finden ist. Geben Sie nach dem Punkt die Höhe an. Nach Window folgen die X- und Y-Koordinaten des Eingabefensters, danach wird die Ausdehnung angegeben. Schreiben Sie hier andere Werte, wenn erwünscht. Der Punkt FLAGS wird sicher neu für Sie sein, denn beim normalen Abspeichern ist dieser nicht zu finden. Hinter diesem Wort können einige Einstellungen gemacht werden, die sonst üblicherweise nach dem Laden getätigt werden. Die möglichen Parameter sind:

Parameter Bedeutung

NOGLOBAL	Schaltet Global-Font-Funktion aus.
GLOBAL	Schaltet Global-Font-Funktion ein.
NOWRAP	Schaltet Wordwrapping aus.
WRAP	Schaltet Wordwrapping ein.
NOFONTS	Die Font-Tabelle wird nicht erstellt.
FORMFEED	Schaltet Formfeed bei Druckereinstellungen ein.
DRAFT	Stellt auf Normaldruck.

Setzen Sie nach Herzenslust ein, was Ihnen gerade gefällt. Sicherlich gibt es die eine oder andere interessante Möglichkeit für Ihre Anwendung. Man findet diese Einstellungen übrigens nicht nur beim Notepad. Auch andere Programme, die Daten in Files ablegen, benutzen dieses Verfahren. Wie Sie selber mit Ihren Programmen diese Parameter auf Diskette abspeichern und noch vieles mehr machen können, lesen Sie bitte im Kapitel über Icons nach.

6.1.5 Viele Wege führen nach ...

Rom! So heißt zwar der Spruch, und auf dem Amiga gelten fast die gleichen Gesetze. Das heißt, viele Wege führen zum Ziel, und so ist es dann auch. Möchten Sie eine Diskette kopieren, so gibt es drei Wege, dies zu tun:

1. Sie bewegen das Icon der Originaldiskette auf das der Kopiediskette, dann wird das bei der Quelldiskette unter De-

faultTool eingetragene Programm geladen und ausgeführt. Dies ist in den allermeisten Fällen natürlich DiskCopy, und wir bekommen eine Kopie der Diskette. Bei diesem Weg ist es allerdings nötig, daß Sie schon die Zieldiskette kennen, denn ohne diese können wir ja schlecht die Quelldiskette darauf bewegen. Ein etwas umständlicher Vorgang!

2. Sie klicken das Icon der zu kopierenden Diskette an und wählen dann im Menü Duplicate aus. Auch hier wird das Programm DiskCopy geladen und es führt den Kopiervorgang aus. Der Nachteil hierbei ist, daß man in jedem Fall die Diskette wechseln muß, denn DiskCopy arbeitet in diesem Zustand nicht mit zwei Laufwerken!

3. Dieser dritte Weg ist den meisten unbekannt. Dazu klicken Sie wieder das Icon der Quelldiskette an, und nun halten Sie Shift gedrückt und klicken doppelt auf das Icon von DiskCopy. Auch so lassen sich Kopien erstellen!

Hinweis: Wenn Sie versuchen sollten, die RAM-Disk mit DiskCopy zu kopieren, werden Sie vom Programm mit der Begründung 'rausgeschmissen', daß es sich gar nicht um eine Diskette handelt. Anders ist dies bei der resetfesten RAM-Disk RAD:! Sie wird anstandslos kopiert, mit dem kleinen Nachteil, daß dieser Kopiervorgang nur über das Laufwerk RAD: abgewickelt werden kann, was bedeutet, daß wir RAD: auf sich selbst kopieren. Außerdem stürzt die Workbench nach dem Vorgang ab, mit dem Guru-Eintrag, daß eine Speicherliste fehlerhaft sei.

Formatieren leichtgemacht

Einige Gedanken zum Format-Befehl und seiner Wirkung. Das Formatieren von Disketten kennt jeder Anwender. Einige wissen vielleicht nicht, daß man diesen Formatierungsvorgang auch über den bei DiskCopy beschriebenen Weg der Mehrfachaktivierung erreichen kann, doch dies ist nur nebensächlich. Haben Sie sich schon einmal Gedanken darüber gemacht, was passiert, wenn wir die RAM-Disk formatieren? Nun, zunächst einmal wird wie üblich das Formatierungsprogramm geladen. Dieses

wartet auf eine Bestätigung und stellt erst dann fest, daß die RAM-Disk gar nicht formatiert werden kann. Welch schwaches Bild! Nur bei Mehrfachaktivierung wird gleich erkannt, daß die RAM-Disk gar keine richtige Diskette ist.

Anders verhält sich das bei der resetfesten RAM-Disk, denn diese ist ja als richtige Diskette konzipiert. Sie hat Zylinder und Blöcke und belegt deswegen auch ständig den Speicher in vollem Maße. Hier ist eine Formatierung möglich! Wählen wir den Punkt aus, so wird blitzschnell die Diskette gelöscht und verifiziert. Doch von diesem schönen Vorgang bleibt ein Directory-Window unberührt. Es zeigt noch immer den Inhalt an. Erst wenn wir dieses schließen, bemerkt der Amiga den neuen Zustand, doch dann verschwindet auch das Disketten-Icon!

Wir kommen an die resetfeste RAM-Disk über die Workbench nicht mehr heran. Hier hilft uns nur noch der Weg über das CLI. Entweder wir geben DiskChange ein, oder aber wir formatieren über das DOS. Dann wird die Diskette wieder über ein Symbol dargestellt, und die normale Arbeit ist wieder möglich. Dieses Problem trat übrigens beim zweiten Versuch nicht mehr auf! Man sieht, daß sich das Betriebssystem noch nicht ganz einig ist.

Neuerungen auf der Workbench

Mit der neuesten Version der Workbench ist es nun auch möglich, die Namen der beiden RAM-Disks zu verändern. Sie können jetzt nach Herzenslust den Namen von RAM-Disk auf z.B. WGB ändern. Gleiches gilt auch für die resetfeste Variante. Es ist dann für einen Außenstehenden nicht mehr ersichtlich, auf welches Speichermedium nun zugegriffen wird. Allerdings macht das System diesen Namenswechsel nur mit, wenn Sie ihn über die Workbench tätigen. Wird er aus dem CLI gestartet, dann behalten die Diskettensymbole auf der Workbench ihren Namen, und beim Öffnen wird man gebeten, die entsprechende Diskette einzulegen, da sie nirgendwo vorhanden ist. Unter diesen Um-

ständen kommt man dann nicht mehr an die RAM-Disks heran. Sie sollten sich also hüten, die RAM-Disks über Relable mit einem neuen Namen zu versehen, es sei denn, Sie wollten jemanden ärgern.

6.1.6 Arbeitserleichterung

Wenn man noch nicht im Besitz einer Festplatte ist, kann es schon einmal vorkommen, daß die Ladezeiten bei einem Diskettenlaufwerk die Geduld eines jeden Benutzers herausfordern. Deshalb überlegt man sich bei Zeiten, ob es Möglichkeiten gibt die Ladezeiten zu verkürzen.

Die Lösung setzt bei dem Umstand an, daß ein Verzeichnis-Aufbau genau dann besonders lange auf der Workbench dauert, wenn sich viele Files darin befinden. Noch zusätzlich wird die Wartezeit verlängert, wenn es in dem Verzeichnis Files gibt, die gar nicht mit einem Icon versehen sind. Deshalb muß man diese Situation umgehen. Als Beispiel soll ein professionelles Programm dienen, das nicht über die Startup-Sequence aufgerufen wird (also mit einem Auto-Boot), sondern über einen Workbench-Klick.

Dazu muß man zuerst das Disketten-Icon der Diskette öffnen. Und hier beginnt schon unser Problem. Neben dem sehnlichst gesuchten befinden sich noch viele andere Icons im Verzeichnis, deren Ladevorgang kostbare Zeit in Anspruch nimmt. Wie umgeht man diese Situation? Man muß dazu schon am Kern des Problems ansetzen und den Programm-Aufruf verlagern. Dazu reicht ein einfaches Project-Icon, bei der Programm-Aufruf eingetragen ist. Aber nun der Reihe nach:

1. Möglichkeit

Damit Sie das Verzeichnis irgendeiner Programm-Diskette nicht öffnen müssen, erstellen Sie sich auf Ihrer Workbench- oder Arbeits-Diskette ein Verzeichnis, das alleine Project-Icons enthält. Mit diesen Project-Icons wird über das Default-Tool der CLI-

Befehl IconX aufgerufen, der es ermöglicht, CLI-Befehle über die Workbench aufzurufen. Mit den Tool-Types kann dann das Window angegeben werden, in dem die Ausgabe der CLI-Befehle erfolgt. (Dieses können wir später wieder schließen.)

Nun sehen Sie hier noch ein konkretes Beispiel, bei dem BECKERtext ohne Öffnen des Verzeichnisses der Programm-Diskette gestartet werden kann:

1. Kopieren Sie mit

```
copy sys:prefs/printer.info sys:BECKERtext.info
```

das Project-Icon vom Preferences-Unterpunkt "Printer" in das Hauptverzeichnis unter dem Namen "BeckerText" und tragen Sie bei Default-Tool dieses neuen Icons

```
C:IconX
```

und bei Tool-Types

```
WINDOW=CON:0/0/150/50/BeckerText
```

ein. Jetzt erscheint im Workbench-Hauptverzeichnis dieses Icon, das Sie noch geeignet positionieren können.

2. Wir erstellen über den CLI-Befehl

```
1> ed sys:BECKERtext
```

eine CLI-Sequenz mit den nötigen Befehlen zum Aufruf des Programms BECKERtext:

```
CD DF1:
RUN DF1:BECKERtext  ; Programm starten
WAIT 40 SECS        ; warten bis Installation abgeschlossen
RUN SYS:Clock analog=535,0,105,65 24HOUR SECONDS DATE
CD SYS:
```

Hier wird neben dem Aufruf des Programms selbst noch ein Komfort-Element ergänzt. Sicherlich ist die Anzeige einer Uhr von Interesse und hiermit für den Anwender ohne zusätzlichen

Aufruf gegeben. Die Warte-Funktion ist nötig, weil das Programm aus dem Standard-Device die Default-Einstellungen liest. Die Zeit von 40 Sekunden entspricht etwa der Dauer des Programm-Aufrufs von Diskette.

2. Möglichkeit

Mit dieser Methode können Sie es sich natürlich auch ersparen, tiefer in die Verzeichnisse einer Diskette zu wandern. Sie kopieren einfach in das Haupt-Verzeichnis wiederum ein Projekt-Icon und tragen als Default-Tool das Programm ein. Somit wird viel Wartezeit mit wenig Struktur-Arbeit ersetzt. Diese Methode arbeitet nach dem ähnlichen Verfahren, bei dem z.B. über ein Text-File die Textverarbeitung aufgerufen wird, nur daß wir hier keinen Text haben. Das sollte Ihr Programm aber nicht unbedingt stören.

3. Möglichkeit

Ein anderer Trick zum Aufräumen der Workbench ist schnell über das CLI durchgeführt. Hat man besonders viele Icons aus Verzeichnis-Fenstern herausgenommen und diese wieder geschlossen, so fällt es später schwer, diese wieder richtig einzuordnen. Meist bleiben dann bis zum Ausschalten oder einem Reset alle Icons mit den Disketten auf der Workbench und schlucken viel Speicher und Rechenzeit. Dies läßt sich aber ändern, indem Sie einfach im CLI

```
1> loadwb
```

eingeben. Alle einzelnen Icons aus Verzeichnissen werden sofort entfernt und nur die Windows bleiben bestehen. Gleichzeitig werden alle erreichbaren Disketten sortiert und rechts geordnet abgebildet.

Da die Icons verschwinden, die auf der Screen lagen und nicht in irgendeinem Verzeichnis aufbewart sind, müssen Sie zuvor Verzeichnis-Fenster schließen, aus denen Icons entfernt wurden, wenn Sie diese dort wiederhaben möchten!

6.2 Systemdaten selbst einstellen

```
struct Preferences <intuition/intuition.h>
                   <intuition/preferences.h> /* Ab Version 1.3 */
{
0x00  00  BYTE FontHeight;    /* Zeichensatz: 60 oder 80 Zeichen */
0x01  01  UBYTE PrinterPort; /* PrinterPort: seriell oder parallel */
0x02  02  USHORT BaudRate;    /* BaudRate: zwischen 110 und 19200 */
0x04  04  struct timeval KeyRptSpeed; /* Tastaturwiederholungsrate */
0x0C  12  struct timeval KeyRptDelay; /* Verzögerungszeit bis zu einer
                                         Wiederholung */
0x14  20  struct timeval DoubleClick; /* Länge des Zeitintervalls
                                         Doppelklick */
0x1C  28  USHORT PointerMatrix[36L];  /* Daten für Mauszeiger-Grafik */
0x64 100  BYTE XOffset;           /* Offset des HotSpot */
0x65 101  BYTE YOffset;
0x66 102  USHORT color17;         /* Farben des Mauszeigers */
0x68 104  USHORT color18;
0x6A 106  USHORT color19;
0x6C 108  USHORT PointerTicks;    /* Übersetzung der Mausbewegung */
0x6E 110  USHORT color0;          /* Farben der Workbench */
0x70 112  USHORT color1;
0x72 114  USHORT color2;
0x74 116  USHORT color3;
0x76 118  BYTE ViewXOffset;       /* Relative Position der Workbench-
                                     Screen zum View */
0x77 119  BYTE ViewYOffset;
0x78 120  WORD ViewInitX;         /* Initialisierungswerte für View */
0x7A 122  WORD ViewInitY;
0x7C 124  BOOL EnableCLI;         /* CLI ein- oder ausgeschaltet */
0x7E 126  USHORT PrinterType;     /* PrinterTyp */
0x80 128  UBYTE PrinterFilename[30L]; /* Name des Printers bei CUSTOM */
0x9E 158  USHORT PrintPitch;      /* Schriftart: Pica, Elite, Fine */
0xA0 160  USHORT PrintQuality;    /* Druckqualität: Draft, NLQ */
0xA2 162  USHORT PrintSpacing;    /* Druckabstand: 6 LPI oder 8 LPI */
0xA4 164  UWORD PrintLeftMargin; /* Linker und rechter Druckrand */
0xA6 166  UWORD PrintRightMargin;
0xA8 168  USHORT PrintImage;      /* Positive oder negative Darstellung
                                     der Grafik */
0xAA 170  USHORT PrintAspect;     /* Druckaspekt: horizontal, vertikal */
0xAC 172  USHORT PrintShade;      /* Druckart: schwarz/weiß, Graustufen,
                                     Farbe */
0xAE 174  WORD PrintThreshold;    /* Graustufe */
0xB0 176  USHORT PaperSize;       /* Papiergröße: Flaggen */
0xB2 178  UWORD PaperLength;      /* Papierlänge in Zeilen */
0xB4 180  USHORT PaperType;       /* Papiertyp: endlos, Einzelblatt */
0xB6 182  UBYTE SerRWBits;        /* Serielle Einstellungen:
                                     Read-/Write-Bits */
0xB7 183  UBYTE SerStopBuf;       /* Anzahl Stop-Bits, Puffergröße */
0xB8 184  UBYTE SerParShk;        /* Parität und Shake */
0xB9 185  UBYTE LaceWB;           /* Workbench im Interlace-Modus:
                                     ein, aus */
```

```
0xBA 186  UBYTE WorkName[30L];   /* Zwischenspeicherung des
                                    Druckernamens */
0xD8 216  BYTE RowSizeChange;
0xD9 217  BYTE ColumnSizeChange; /* Ende Version 1.2 */
0xDA 218  UWORD PrintFlags;      /* Neue Grafik-Einstellungen  */
0xDC 220  UWORD PrintMaxWidth;
0xDE 222  UWORD PrintMaxHeight;
0xE0 224  UBYTE PrintDensity;
0xE1 225  UBYTE PrintXOffset;
0xE2 226  UWORD wb_Width;        /* Breite, Höhe, Tiefe der Workbench */
0xE4 228  UWORD wb_Height;
0xE6 230  UBYTE wb_Depth;        /* Version 1.3 */
0xE7 231  UBYTE ext_size;        /* Länge einer eingebundenen
                                    Erweiterung */
0xE8 232
};
```

Preferences_FontHeight

```
TOPAZ_EIGHTY        8L
TOPAZ_SIXTY         9L
```

Preferences_LaceWB

```
LACEWB              0x01L
```

Preferences_PrinterPort

```
PARALLEL_PRINTER    0x00L
SERIAL_PRINTER      0x01L
```

Preferences_BaudRate

```
BAUD_110            0x00L
BAUD_300            0x01L
BAUD_1200           0x02L
BAUD_2400           0x03L
BAUD_4800           0x04L
BAUD_9600           0x05L
BAUD_19200          0x06L
BAUD_MIDI           0x07L
```

Preferences_PaperType

```
FANFOLD             0x00L
SINGLE              0x80L
```

Preferences_PrintPitch

PICA	0x000L
ELITE	0x400L
FINE	0x800L

Preferences_PrintQuality

DRAFT	0x000L
LETTER	0x100L

Preferences_PrintSpacing

SIX_LPI	0x000L
EIGHT_LPI	0x200L

Preferences_PrintImage

IMAGE_POSITIVE	0x00L
IMAGE_NEGATIVE	0x01L

Preferences_PrintAspect

ASPECT_HORIZ	0x00L
ASPECT_VERT	0x01L

Preferences_PrintShade

SHADE_BW	0x00L
SHADE_GREYSCALE	0x01L
SHADE_COLOR	0x02L

Preferences_PaperSize

US_LETTER	0x00L
US_LEGAL	0x10L
N_TRACTOR	0x20L
W_TRACTOR	0x30L
CUSTOM	0x40L

Preferences_PrinterType

CUSTOM_NAME	0x00L
ALPHA_P_101	0x01L
BROTHER_15XL	0x02L
CBM_MPS1000	0x03L
DIAB_630	0x04L

```
DIAB_ADV_D25        0x05L
DIAB_C_150          0x06L
EPSON               0x07L
EPSON_JX_80         0x08L
OKIMATE_20          0x09L
QUME_LP_20          0x0AL
HP_LASERJET         0x0BL
HP_LASERJET_PLUS    0x0CL
```

Preferences_SerialBuffer

```
SBUF_512            0x00L
SBUF_1024           0x01L
SBUF_2048           0x02L
SBUF_4096           0x03L
SBUF_8000           0x04L
SBUF_16000          0x05L
```

Preferences_SerRWBits

```
SREAD_BITS          0xF0L
SWRITE_BITS         0x0FL
```

Preferences_SerStopBuf

```
SSTOP_BITS          0xF0L
SBUFSIZE_BITS       0x0FL
```

Preferences_SerParShk

```
SPARITY_BITS        0xF0L
SPARITY_NONE        0L
SPARITY_EVEN        1L
SPARITY_ODD         2L
SHSHAKE_XON         0L
SHSHAKE_RTS         1L
SHSHAKE_NONE        2L
```

6.3 Betriebssystem-Editor

Seit der totalen Umstellung Commodores auf die Betriebssystem-Version 1.3 erfreuen sich die Kickstart-Umschalte-Platinen wachsender Beliebtheit. Alle Amigas werden jetzt mit dem neuen Betriebssystem ausgeliefert, was zur Folge hat, daß sich die Programmentwickler auf die neuen Möglichkeiten ein-

schießen. Da zwar 1.2-Programme auch auf V1.3 laufen, V1.3-Programme manchmal jedoch nicht auf 1.2-Amigas, möchte natürlich jeder Amiga-Besitzer das neue Betriebssystem besitzen. Wer mit dem Gedanken gespielt hat, sich lediglich die (irgendwann erscheinenden) Update-Sets von Commodore zu kaufen, ist spätestens seit der Ankündigung von Kickstart 1.4 verunsichert. Freuen können sich eigentlich nur die Amiga-1000-Besitzer, die jedes Betriebssystem laden können. Um dies auch den anderen Amiga-Besitzern zu ermöglichen, gibt es jetzt von einer bekannten Hard- und Software-Firma ein sogenanntes Bootmodul, mit dem jederzeit ein anderes Betriebssystem geladen werden kann, sofern dies in Form einer Kickstart-Diskette vorliegt.

Gründe genug, ein Programm zu schreiben, das die Möglichkeit bietet, solche Kickstart-Disketten herzustellen. Darüber hinaus gibt es jedoch noch einige Kleinigkeiten, die an den existierenden Betriebssystemversionen auszusetzen sind. Zum einen ist da sicherlich die ständig als Damokles-Schwert über dem User schwebende Virengefahr, das zwangsläufige Ausführen des Bootblock-Programms nach einem Reset. Dies ließe sich durch eine Änderung des Betriebssystems abschaffen, ohne auf Programme mit Bootblock-Loader verzichten zu müssen. Zum anderen haben wir da die manchmal fehlerhaft arbeitende PAL-Test-Routine, das viel zu langsame Zugreifen auf Diskettenlaufwerke, das immer noch nicht auf PAL-Format vergrößerte CLI-Window und diverse Betriebssystem-Routinen, deren Aufruf fast zwangsläufig zur Guru Meditation führt. Wer sich gut im Betriebssystem auskennt, wird erkannt haben, daß es noch viel zu tun gibt, um ein nahezu perfektes Betriebssystem zu erhalten. Um diese Änderungen vornehmen zu können, habe ich Ihnen einen Betriebssystem-Editor in Assembler geschrieben, der hier aufgrund seiner erheblichen Länge in total gepackter Form als DATA-Loader abgedruckt ist:

```
OPEN "KICKSTART-EDITOR V1.5" FOR OUTPUT AS 1
zeile%=0
checksum%=0
zloop:
  zeile%=zeile%+1
  wort%=0
```

```
  StrOut$=""
iloop:
    wort%=wort%+1
      READ a$
      IF a$<>"x" THEN
        wert%=VAL("&H"+a$)
        checksum%=(checksum% XOR wert%)
        checksum%=(checksum% XOR wort%)
        StrOut$=StrOut$+MKI$(wert%)
      ELSE
        wort%=10
      END IF
      IF wort%<10 GOTO iloop
      READ b$
      wert%=VAL("&H"+b$)
      IF wert%=checksum% THEN
        LOCATE 1,1
        PRINT "Zeile";zeile%;" - Checksumme okay"
        PRINT #1,StrOut$;
        IF a$<>"x" GOTO zloop
        CLOSE 1
        PRINT
        PRINT "Fertig."
        PRINT
        END
     ELSE
        PRINT
        PRINT "Checksummen-Fehler in DATA-Zeile";zeile%
        PRINT
        BEEP
        CLOSE 1
        KILL "KICKSTART-EDITOR V1.5"
        END
      END IF

DATA 0000,03F3,0000,0000,0000,0003,0000,0000,0000,0002,03F9
DATA 0000,009B,0000,0588,0000,0735,0000,03E9,0000,009B,02A6
DATA 6106,4EF9,0000,0000,48E7,FFFE,2C78,0004,4BFA,019A,FC57
DATA 41FA,FFE6,2050,D1C8,D1C8,2250,D3C9,D3C9,5888,5889,4041
DATA 48E7,00C0,6100,0192,4CDF,0300,2F09,5188,2248,2011,5BF8
DATA 4EAE,FF2E,2657,508B,201B,2E00,E788,223C,0001,0000,5701
DATA 4EAE,FF3A,2B40,000C,508B,2007,5380,226D,000C,221B,EEA4
DATA 5889,5481,22C1,51C8,FFF6,2F0B,2C07,5386,266D,000C,18B3
DATA 284B,4A9B,223C,0001,0000,201B,588B,0800,001E,6704,4FDF
DATA 08C1,0001,E588,4EAE,FF3A,28C0,588C,51CE,FFE0,265F,EB35
DATA 4286,4285,0C6B,03E9,0002,6700,0090,0C6B,03EA,0002,8CAE
DATA 6700,0086,0C6B,03EB,0002,6700,009E,0C6B,03EC,0002,8CBA
DATA 6700,00A6,0C6B,03F2,0002,4E71,4A9B,4285,5286,BE86,4EE3
DATA 6EC6,225F,5189,2011,4EAE,FF2E,43FA,FF12,206D,000C,5EE0
DATA 2010,5880,E488,2280,5287,5387,671A,2250,2028,0004,8415
DATA E588,2280,2028,0008,5880,E488,2340,0004,5088,60E2,CC10
DATA 42A9,0004,43FA,FEE6,206D,000C,2050,5088,2288,226D,63F6
DATA 000C,2007,E788,4EAE,FF2E,4CDF,7FFF,4E75,6128,4A9B,4318
```

```
DATA 201B,E588,204B,D7C0,226D,000C,2206,E789,2271,1800,AC94
DATA 5089,4EAE,FD90,6000,FF48,6106,508B,6000,FF40,4A85,3428
DATA 6702,5286,7A01,4E75,4A9B,206D,000C,2006,E788,2270,BAD7
DATA 0800,5089,201B,6724,221B,E789,206D,000C,2870,1800,70E9
DATA 508C,5380,204C,2449,221B,D5C1,2212,D1C1,2488,51C8,06A2
DATA FFF0,60D8,6000,FEFA,0000,1620,0000,1CD4,0000,0000,0D8F
DATA 0000,0000,0000,0000,D1D5,2B48,0010,246D,0004,D5C9,06A9
DATA 2020,E288,6602,616E,6532,7208,7601,E288,6602,6162,47B5
DATA 654C,7203,4244,6164,3602,D644,7207,E288,6602,614E,0454
DATA E392,51C9,FFF6,1502,51CB,FFEE,6038,7208,7808,60DE,F833
DATA 7202,6140,0C02,0002,6D12,0C02,0003,67EA,7208,6130,F2BB
DATA 3602,323C,0008,600A,323C,0009,D242,5442,3602,611C,75A7
DATA 534A,14B2,2000,51CB,FFF8,B3CA,6D00,FF90,4E75,2020,F368
DATA 44FC,0010,E290,4E75,5341,4242,33C8,00DF,F180,E288,2A76
DATA 6608,2020,44FC,0010,E290,E392,51C9,FFEA,4E75,FFFF,3612
DATA 0000,03EA,0000,0588,181C,F008,3038,0040,0831,0120,E106
DATA B025,E035,3001,B955,0F86,5040,4340,103D,E008,1490,C06A
DATA 0408,7802,0490,0103,9E00,8141,0040,9B80,204A,30A0,2D93
DATA 7557,81CA,584E,8130,8BF3,0008,1010,70AA,0009,D818,332B
DATA 6CBF,D720,5025,000A,207B,605C,10A6,0014,1448,0010,9C5D
DATA 2010,8004,0982,2011,100C,14F8,01F8,0730,82C4,6029,F500
DATA 1F88,021E,0010,539A,72AF,D460,B822,A048,179E,6834,7A18
DATA 0D82,BC1A,05A2,097F,80DC,8494,C3F4,05AF,F096,610E,94DD
DATA 25F7,0B4E,0F53,0000,8058,6014,CFF3,1D41,6006,A614,41D0
DATA 5856,E3E1,7820,8601,A03C,2180,7215,D04B,6FF1,1727,5F79
DATA 57F3,1520,166C,0EB1,26A9,5A8F,9AB3,1B84,4E00,7E9B,C8F6
DATA C008,507E,41E6,0780,80C3,0042,8050,8080,E373,4082,3D4D
DATA 2503,00D3,7305,1863,42E6,6623,4297,0370,2DB0,148A,2FE8
DATA 6C06,4A06,A458,0028,2928,3888,22D9,A505,0264,9C3E,A5B5
DATA 8321,4A20,8197,0300,0340,AE81,E280,2881,9145,881F,90B2
DATA F01C,2760,390B,4224,0528,18B1,036F,00B7,83FF,3281,93D5
DATA 4080,2708,21C2,E860,6042,6080,0440,4A0E,1880,C4D0,AF28
DATA ED00,4113,03D0,8368,03D1,46A5,812B,0177,0358,1153,54AB
DATA 0B54,060F,880A,8420,0882,BA05,42A5,01E8,BB13,880E,9706
DATA 941E,07F8,7BB6,0EA2,7022,08C8,070B,01FD,0249,6081,6D2B
DATA A8EA,0120,BA04,2061,0008,123E,6504,040B,7214,1823,4781
DATA 29C8,0A02,67CC,3085,4166,0265,83CA,0124,660C,1883,8C6B
DATA C8FC,C026,D4FC,2281,C361,045B,41E2,1103,C09E,01C0,2442
DATA 81F8,0F99,A95A,8F89,B308,6A1F,2078,08EA,1F23,8320,E17D
DATA 8D80,805E,00D1,5400,1A70,3D10,70A0,61B3,801F,A6F0,A8E5
DATA 0C56,9FC2,A191,E12B,9EF0,BB3B,8300,3F93,2201,2076,E0EF
DATA 0128,02C4,C023,006A,9804,E07C,B752,588E,1354,E4EA,435B
DATA FEC2,0847,28F4,0A02,0805,0206,D605,0110,0281,0303,4BB7
DATA 0280,9003,0250,801D,60A0,5400,2815,C40A,0580,00E6,86AB
DATA 0080,3030,01E4,048D,280B,3985,B000,B0F8,EB37,0A23,431B
DATA B0E0,04E0,00CA,1041,A996,83E9,A414,10B4,0050,2266,5B72
DATA 1040,1C64,4410,3679,2979,2238,5A93,8096,81C2,8018,F5AA
DATA B668,2002,8004,D600,E213,2980,1007,F3B5,1627,9678,9DB1
DATA 1C08,32C3,3890,09F8,0485,3894,0708,8480,8C10,F108,4098
DATA 1B40,D221,F400,4710,4100,F480,00E7,0219,A9F4,83C8,A7A0
DATA 13E2,0B02,4621,0E01,1286,2808,0042,0553,B094,0040,7820
DATA 4690,A049,1814,2B50,F910,0369,1A80,D048,9320,F420,FA07
DATA 96C7,E82E,9AA3,38B0,3171,2212,089B,32E1,4046,12FA,5D53
```

```
DATA 80CB,7542,6C60,6007,3084,569F,F8F9,874C,1183,0C04,A09F
DATA 0C5E,C817,580A,56A3,E47C,CA10,1D40,43C0,6040,DC10,A6C8
DATA 1000,3C25,F408,7002,1041,9002,2058,3904,5029,01EB,C631
DATA 2116,03D5,F8E8,65A0,F8E9,8017,FFFC,071E,15B0,6548,8955
DATA 0040,1F14,E24C,F248,1050,1EEA,825F,0030,C104,2102,EADD
DATA 0704,0039,05D0,804A,0040,2700,2011,4010,08B4,4109,668D
DATA 0080,4A80,4024,C020,1110,1044,0FB0,2017,4010,0A10,C871
DATA 082A,04A8,104C,04B0,2014,C010,0850,0804,B801,9700,1F55
DATA 1C0C,2607,8818,0B70,0223,F768,040A,7802,0558,0102,2B24
DATA 8C02,4B00,6E04,828E,0241,5A01,20A5,414F,A3E2,5091,CA7E
DATA 0059,0402,415A,2000,B4E4,EB7E,A386,C135,0180,BE80,2D5D
DATA C05B,0060,2E80,5415,F02A,0AA0,1523,F448,300B,C817,7405
DATA 8B10,2D00,4298,04C0,10AC,CF70,2628,7480,DD04,2126,E510
DATA 0360,8001,0847,C12D,8FCA,010F,F4A4,B0B4,839F,9086,76DC
DATA 6284,FFCF,F085,C000,90A2,4380,110F,0000,1A39,9416,971B
DATA 324C,8094,0315,27CA,2988,0511,0A45,0F02,60AF,01C0,49A6
DATA 50C0,E02F,A070,16D0,6304,3B08,E804,0434,0442,2A06,D59A
DATA 0124,6677,4030,8E33,0BA0,4208,0981,4094,80C0,44B5,B809
DATA 0F91,003F,E008,2067,8008,1010,0809,6215,A8F8,9904,BC3B
DATA 2483,480B,5BE1,D074,9C15,0C00,9000,28A8,24C0,3420,6370
DATA 6C02,207E,4879,8B2B,02DE,0C82,8E04,C606,2ACC,E184,6143
DATA BF40,E8C4,CA21,0B98,037D,7018,4F81,840C,0006,0300,4C9B
DATA 0302,B214,B801,BBFF,0A82,3453,FD40,242A,4CC0,0C28,592B
DATA 4003,053C,0182,3604,2D06,5482,DC81,802E,0754,06E1,0F07
DATA 2220,681C,C61C,4BFA,0810,C019,0860,D81E,D215,00E1,0255
DATA 4C02,74A4,B0B4,B998,0305,4010,019F,683B,7703,24B4,4AD2
DATA 83C4,DC21,00EE,0640,02AB,8210,0264,E03E,1083,31E8,5018
DATA 25B1,F940,20C0,0714,606E,A014,606C,A480,0536,A7E2,0D74
DATA 0C81,1CC7,2218,001F,0CCD,2DB2,1D70,5D82,42F0,7020,6C63
DATA 27E0,B2AD,984C,0071,417F,FFBB,007F,BF6A,2170,4240,03F9
DATA B500,E0C3,447E,61A9,8582,89E1,41B5,81E0,A1B6,A3E4,BD82
DATA BCC2,1C4B,EA12,B01C,62C8,1881,8087,A25A,0F8A,B93A,A92A
DATA FF9D,9A5B,2075,E818,1CD3,4C30,0181,9980,1200,0730,D958
DATA 02FA,4412,FA02,9070,0287,0815,7030,0827,337A,1404,A032
DATA 9198,8120,6399,718C,0219,6C56,8301,9C02,4103,0C06,9EDD
DATA 4443,00C1,5B41,1260,989D,9810,903D,80C1,0860,186B,930F
DATA 60D3,0C30,4CCE,CAA4,1000,1D52,E220,3000,C864,1C40,72DB
DATA 4FC0,002D,8648,E05E,09D8,2501,0C03,FE0A,8804,0205,0FFA
DATA F660,043C,00FF,7F0E,09B5,80E6,96A0,1410,8583,520F,5E33
DATA 02B9,1008,9808,04CD,0AD4,7C8C,8402,4A07,A174,0600,CF65
DATA 9F4E,2127,0314,206B,9CC1,E844,8C1F,5990,88C0,C18D,BA3F
DATA F504,B0E1,0311,8220,6851,0041,087A,5A21,920A,8C05,5AA4
DATA 23D0,4010,5889,144C,1C19,8583,98E1,D041,EE04,53EA,197E
DATA 5388,0180,838F,4060,3A86,5808,8A20,8423,2FB8,0C00,C7A7
DATA 66B1,AF1D,4FAC,0E0D,2E58,0690,0611,0062,1416,5A04,2F08
DATA A600,4800,140C,16F0,1708,4070,0064,E10C,2101,7E66,2A88
DATA 7220,1600,1E0B,4000,1048,3C86,1206,1366,3992,3280,3614
DATA 5987,D03B,A480,8107,4173,E680,E4C6,0C86,4924,A411,38A2
DATA CCC1,5A21,CC3E,0C36,8920,8102,01C1,FE7F,A904,7E60,4EB9
DATA 25CE,0204,D1D2,05CF,097E,0304,42E0,0BCC,61E5,EE00,71D6
DATA 21C6,4BC1,4818,007D,C4CB,ED02,0018,554A,D683,B05C,49FB
DATA CE08,96C7,E500,879C,2E94,001C,9D7F,DD20,9861,0480,8195
DATA 8288,1580,80AC,0040,43C0,2025,8010,1370,0809,A615,C8E3
```

```
DATA BFF8,B908,0492,1801,20F9,CAE9,CAB8,03A4,1600,3C02,DB85
DATA AF0C,8369,8224,8000,D0E0,C309,0711,4302,4086,2820,CA93
DATA C007,E131,0000,A060,0010,5052,2188,0D80,4061,2381,5464
DATA 881F,C082,9032,8100,DE02,00F9,C06E,320B,6404,024A,4710
DATA 0301,9501,4303,7501,008E,8080,60C0,603C,6038,A071,A7A2
DATA 1170,2011,B010,0938,0805,DC04,1440,0585,2382,0514,CCB2
DATA 0D20,A204,03B0,0201,8C01,0491,C508,0106,2162,0041,0F91
DATA 81C9,0041,81E7,0021,81C0,5031,0820,580B,A071,028A,2CF5
DATA 0660,ED02,01F1,0100,BA80,987C,C008,D0CB,4040,3A60,8F72
DATA 3818,7008,3039,F008,3033,F00C,3036,8010,09A0,0805,76E4
DATA 0804,022C,020A,C0C6,020C,0E94,050C,0ED4,070C,0CBC,B2FB
DATA 038B,0FC8,020C,0DD8,028C,0F38,068C,0F2A,20AC,8041,1598
DATA 1844,4022,81D8,1043,0201,0500,D848,40C2,81E8,32C0,F0CD
DATA 82D0,B380,8303,B704,0103,0030,5F44,00E2,503E,4307,B83D
DATA 6920,FA80,116E,5D51,A083,36D0,7481,0F2A,BA01,4200,7250
DATA 2502,2882,9E29,A103,DEB7,069A,3499,ABA0,4572,4E80,0C17
DATA 1E5A,D0D3,C014,5C9A,8BBD,9380,3489,F16B,03EA,864E,0660
DATA 8334,450D,01AA,581B,02EC,0F46,070C,8D07,0968,0490,13BA
DATA 1471,337B,1B22,E25C,0082,3636,2325,44EC,4C83,AA58,7A63
DATA 3AD2,0899,3A86,9670,3805,781D,1765,1538,45CB,C5C8,2693
DATA 5E86,A002,6964,044A,D4D0,8AA5,8DA1,F93B,13E3,0B62,875C
DATA 9218,9280,DD4B,1353,2132,B4CA,2E4E,AED6,2EAE,4E00,3C19
DATA F35A,0F4A,5A1B,825C,0BA8,BAD2,1742,0F82,0426,86CE,3317
DATA 0426,E515,2703,B04C,A1AB,99BB,89B5,6B00,F262,0C87,6120
DATA C50D,20F0,85A9,B1B5,B3AB,ADAD,803F,85ED,2841,D0DF,5380
DATA 65CE,8006,54DC,3CB5,0819,A54C,0D62,A448,C8D0,C9D8,DB5D
DATA BC8B,3818,0214,1C2B,604A,2479,5292,609A,54D2,D0EC,B3FF
DATA 9381,11BD,BD89,F513,8B4B,7361,6200,D292,C2D2,CA2A,3912
DATA 824A,2AB4,2296,CED6,A62E,10F7,1027,743E,2F01,0486,84E0
DATA 8725,4C00,DC82,C858,1851,3010,2601,2286,2E43,8408,DD99
DATA 7F0D,21CE,02C4,9C04,22BB,02B2,EA96,3636,C714,0845,2E69
DATA 5C85,889A,1BEC,1319,8208,CD08,28B8,06D0,B275,37E0,1675
DATA 0111,AB4B,58D2,06C2,ECED,0801,044E,E1CE,F684,04E8,1134
DATA 01E5,AD0D,3C00,652D,4C8C,81C6,A047,6830,42C5,C01A,6318
DATA 5876,2974,3271,75A5,1134,B177,B407,A29A,A31D,2B00,6B06
DATA 0901,90ED,00A5,7701,4981,B77B,4382,CA2E,A666,8623,D256
DATA B5C9,9565,6D0D,CD49,4B90,12CA,1A98,DA87,5270,0331,9AB1
DATA FA70,253E,A889,0ABA,0583,C905,6D2D,CC42,C8AA,5839,1ABD
DATA 0ED3,2021,5276,3DAD,6CEE,4455,4298,D280,9193,8393,F13C
DATA 8080,1C70,A450,4550,4088,5458,5A54,4948,1FC2,F281,6648
DATA 1DA9,A9B1,E543,0152,AE2E,0082,C3A5,4C08,46ED,8309,D2AF
DATA 3AB9,D90B,1345,DC0B,08FA,7603,645D,534B,0237,5327,E6A7
DATA 1B7B,2700,7608,139A,985A,989D,9007,54B1,B1B1,05A0,CF25
DATA 422A,E042,686C,E04E,E964,ED6C,0E91,630B,7E00,2941,DBB7
DATA 0039,5149,3900,137B,507C,4907,5D17,5740,8A96,F676,CF7B
DATA 4440,83E6,0684,1C2D,064B,39D0,DA59,193A,9894,F000,8613
DATA 6462,6661,6563,6760,E4E4,1216,1115,1087,8C91,B1A0,4C49
DATA 1D33,02A2,08D2,9645,95D4,29C0,3CA8,818D,177B,2722,FC2C
DATA A280,0018,4403,83B4,0080,44D0,437A,2E54,B733,B630,B175
DATA A469,62E0,4664,EA68,4F5F,0DEB,E1BD,503D,2867,6450,64F8
DATA 5316,C464,2EC0,F812,29BD,9999,0E13,B03E,8538,FF1B,5179
DATA 5B14,1694,EC05,0ACA,1AB9,1814,3674,35A0,45EA,E455,5AFB
DATA 42DC,0C98,1796,0F38,0001,8A18,BA9D,55C0,B153,40DA,98D7
```

```
DATA 6897,011C,90E0,2E97,3029,2AEE,682E,41C0,9087,855A,69D4
DATA 664E,A696,8400,2266,A59C,04E1,ECE0,6088,D028,9BCB,6997
DATA A5B9,8B81,132B,1B34,90A7,B03C,14E8,0885,4D8C,8A06,B4C7
DATA 2820,CD80,5C38,1130,2A04,293B,DB99,3A1A,DADA,98D0,BCD2
DATA 033B,03E2,46C0,E0BE,C1F0,C0F5,27CB,F05D,80E8,3D95,7190
DATA 73B1,3023,713D,2CEA,E838,4150,2654,B232,AA6C,E81E,10A2
DATA 5EC0,3C81,29A1,455B,00A2,7400,DA42,1083,BA02,F4C5,EEB6
DATA 0D01,CB80,4100,7D1F,4E12,E026,9433,B3B0,2197,2126,9D25
DATA 6EE8,3B5A,9113,1060,23B4,B53B,0411,BD99,99D8,8253,7D63
DATA D465,2CEC,8529,50C1,AD57,271F,E36B,022F,575B,0251,6E0F
DATA 134B,177B,707F,2A00,A743,8C2D,0501,F410,6097,8081,0A22
DATA 49DE,DCC9,D0D6,D6C8,1EC2,26A6,8430,EB59,9170,1A27,457A
DATA 6B48,7C50,3808,09C1,B100,0A86,0700,8231,530B,1B12,150E
DATA A25C,0235,0CED,0A0B,5816,93B0,2255,3475,7533,3607,2D38
DATA EAEA,9676,26F6,EE04,6F66,6676,A676,20E8,990C,D814,57CB
DATA C40E,5816,1023,9551,9191,1BAD,AB28,7357,02B1,535D,4D05
DATA 422A,2203,054D,2C08,8A69,8276,0304,2C00,6B48,07D6,4FE7
DATA B671,7432,7175,A316,981B,101A,AB68,62E5,CC58,1518,C700
DATA 1113,4A4B,0B49,138B,1581,0382,2685,259C,0C88,009B,9D98
DATA 19D0,D810,C89D,D803,9440,8090,4A8C,81F1,5302,5216,9274
DATA 3683,2409,0D0F,8B98,1812,7531,7533,B127,29A8,5A18,DCE5
DATA D917,9345,E14E,503B,442A,E666,622C,0827,7572,7630,ECCC
DATA A04B,66FE,6362,6D10,3EE0,1558,42BC,04A8,C26F,0185,8A46
DATA 0801,0949,0608,042A,0608,D518,E958,0014,9DED,CD01,E397
DATA C7AB,6B60,092C,4B5E,C014,9D80,3100,6943,62D6,CE2E,A40A
DATA 864E,2EA4,8525,130F,9594,2D08,1A85,941D,0B4B,185A,BFD4
DATA 9939,D898,5134,B177,B270,0416,5614,9492,0750,6062,0DEF
DATA 5443,4842,2076,9194,342B,2042,2864,1B52,C070,E06D,A745
DATA D40C,A181,C102,8A86,76E6,4474,3C5C,9D0D,9C01,44C1,D244
DATA 9C9A,0219,94B3,B6B6,1A2C,4E07,680A,4207,9165,5743,D6C9
DATA 8041,16A0,2D63,0B42,C266,0754,5100,2543,8296,0240,57A5
DATA 9701,A204,B2F6,26AE,3604,E6A6,66AE,40FD,1530,3CAB,2999
DATA 4899,3A10,D670,0284,2181,D13B,4392,D696,0E43,8449,605B
DATA 4818,D86C,3C00,C46D,0DCD,9D81,D30B,E858,2AE2,EF6C,66D8
DATA 3B14,030B,D903,1354,5917,006B,223A,7605,C4D4,076B,5A67
DATA 3806,93B0,032C,C006,0954,F072,0306,3200,22A6,6E96,B6E0
DATA 2480,A553,677B,5727,6353,0019,4B23,6074,E938,1790,B1D1
DATA 5030,2F60,2C45,C4F0,64A9,0B0D,8980,1CA1,ADA9,803F,F12C
DATA 8489,499C,8988,888B,2A1E,54B3,B732,2C04,1C93,B412,702B
DATA 2449,6281,8079,6B43,4F00,7553,4B17,5327,0267,5763,AAB5
DATA 0B53,3B00,1218,7E09,B01C,12E0,1812,1D20,60E1,B0F0,8123
DATA 2001,7274,3636,B134,B676,B3A1,3533,74B6,3530,0787,22BA
DATA 3020,6320,3020,6D00,3020,7A20,3028,3286,221C,200D,662E
DATA 05E4,1809,480C,049B,20FC,02F0,9410,863C,0820,0528,0A77
DATA 010B,7006,2665,0E25,F3F7,B1AF,1060,0A01,F140,2043,DA0B
DATA 6006,A90E,E026,2A40,3CC5,E880,5E92,8005,0E4E,AFEC,721E
DATA 2C1D,6210,2800,1730,8180,0D3F,2400,0661,BF40,0043,12F5
DATA 2E0B,4C02,809F,1870,B097,EBD6,03B9,10A6,6284,FFCF,3D0D
DATA F03A,64C5,17A8,A25B,7F84,861E,000F,4071,0120,0C01,A8CF
DATA 0E06,0B96,6045,1804,0081,860A,001A,1104,6418,2A80,0C18
DATA 2650,8070,1105,5868,4E55,C25C,F060,3A8C,262A,D1F8,5269
DATA C0A0,9615,1422,A44E,1414,0FE0,0A05,E805,0270,0281,4DBE
DATA 3C44,3500,6020,4303,4E00,4082,004A,8014,8090,EE00,879E
```

```
DATA 4858,4024,2480,06A0,6D03,C006,8124,A068,A008,2994,A81C
DATA 1108,0022,E094,3984,310F,88E0,3AA0,040A,5878,9400,2B10
DATA 78C0,EB40,3020,E018,1980,0C0A,1806,0451,F801,0D43,943C
DATA A005,A206,2600,29FD,C200,42D6,0021,3403,8172,29A0,85EF
DATA E038,0070,1071,6070,1100,3808,31B3,083C,5200,060C,5126
DATA CC08,87AC,A092,21E0,B022,2058,1F70,2C47,E5F0,5810,8556
DATA C783,400B,012E,112E,344E,140E,502E,C028,1270,1408,A4EB
DATA 800A,04B0,0502,E002,815A,0140,ED80,A04D,8050,2380,AB5D
DATA 2814,8014,0B14,12FD,F8DF,15BD,F80E,A691,F9E2,21A8,7108
DATA 0340,8776,14C0,1808,100C,081C,21F9,120C,0448,0EC2,D892
DATA 0190,C87E,6443,A8FC,2B0A,DCFC,628A,5401,57A0,1000,5B15
DATA 850A,280C,152E,0C24,425C,E042,544B,E006,0E4E,B7ED,40E2
DATA E65F,6908,1C1D,8268,0342,C8C2,2460,14A4,7ECC,A0E8,74AB
DATA 20F7,02FA,2B81,C0AE,6702,A012,A113,001E,2916,D89C,2A15
DATA 3801,4806,8344,0C4C,0483,A001,A706,40A1,014A,0612,916C
DATA 0D02,09AC,0082,4A02,CA06,F8C0,C30A,1000,C206,400A,BC89
DATA 0830,818B,0030,5D40,602A,A00C,5061,C00C,1400,0648,2A4A
DATA 56BF,E07A,96C7,E51F,E7BA,881C,1200,4000,FF1D,2080,0D67
DATA 8E73,4082,7E01,80AF,3A1E,A202,0003,82C1,0302,02C0,262F
DATA B801,6210,158F,4B4F,F2D2,0809,D027,C205,2388,0192,6816
DATA 0943,E004,A818,9E0D,440C,921C,0120,9800,9075,C048,A842
DATA 2E88,8001,DFE4,289D,2825,F3F8,B0E0,0A16,1538,01C9,8463
DATA D5FC,8A81,0231,C4BF,3F64,45C1,50F2,F700,1052,00E6,D078
DATA 2000,E412,B8C0,3004,7B8B,9576,E1F0,0487,F86A,15AF,7AEA
DATA F8EC,32AE,15BF,F94B,93BB,FBE4,95BD,F8A9,2493,F003,8D8C
DATA 7120,22D3,0614,275C,9604,2DF4,00D9,01E0,0149,2598,6124
DATA DC93,8069,039D,4BD0,C0FB,0012,6A1B,C3F6,4DA4,4021,1119
DATA 7C34,E247,4040,1D12,818E,018E,31E0,9044,C009,3031,03AF
DATA 0028,2944,F330,4A19,4072,2818,AC6A,4490,CA85,110C,C8F8
DATA 1262,0C12,0404,0000,A106,AA00,0344,10A4,DA11,CA24,DA54
DATA 4712,B801,4227,D021,0720,8240,2026,EE4E,AFE4,A696,F530
DATA DFE7,EA40,081D,627E,0080,8155,E004,B8B0,2059,7808,2BCF
DATA 7F27,89C0,0FD1,7807,2AEC,0EC0,40D1,83B0,10C5,400E,1D73
DATA 4148,A880,1012,1300,33F0,0482,3140,0244,9008,0464,67B8
DATA B025,D482,D048,3056,6200,B424,4468,105E,01E8,F24C,92BC
DATA 3499,201E,905E,B054,AD47,C4F9,8818,10A8,3AD4,25CF,482F
DATA 0F29,6041,CCBE,D048,3683,E668,0040,1E1C,E24C,0050,1711
DATA 02C4,800F,4842,1DC0,58C0,CD03,0042,6F0E,1841,C434,E6A9
DATA 2A00,9F11,427F,EDEE,55C0,96C7,E618,541D,5218,A281,7DB9
DATA 6440,85CB,E801,2E23,1607,E824,81C8,0428,0413,0090,255B
DATA 0082,6011,0010,5882,C006,0001,116C,7484,1104,6097,C92D
DATA F7C8,2897,C00C,B8A5,C00A,1839,357E,DE89,D1CF,2547,A99C
DATA 09D2,060C,02BA,0142,066C,0382,063D,056E,7E68,1D68,C00C
DATA C301,4944,F804,C061,4580,C023,2004,9805,483A,021B,05A4
DATA 0213,5027,D070,4035,3A24,E680,3412,4830,E002,E006,675C
DATA E001,4630,0C07,3AA1,E620,200E,3824,0970,1202,1B98,0920
DATA 1202,1AA8,1202,1928,123C,1BFB,CC12,2C19,4091,80A0,2356
DATA 418B,A241,4231,3A82,4385,FE11,070A,0620,26EE,4EAF,6CDB
DATA E3E6,96DF,E7E0,1CFA,92B7,1F0B,A25F,0F4E,7E00,80E4,3CBA
DATA 1321,8B22,1D04,738E,0693,A103,1603,A503,2011,01C8,FF71
DATA 0E0B,E411,0A09,83D8,1307,1001,C1F4,0221,8A21,9107,4744
DATA 3490,8E87,18C1,16A1,9931,2E25,0073,890E,01B1,C044,0CA4
DATA 0012,90E0,2512,865F,E1E2,87EC,0AA0,A923,784C,4282,C053
```

```
DATA 6346,2C0C,400A,1C60,1631,A515,2496,6496,1D42,53C0,6EDE
DATA 08A4,9720,ADCF,C613,8103,8011,04A4,97CB,CA5F,A512,67BD
DATA 0EE2,BCCC,39E4,5879,0004,4924,5522,9107,01D1,0A91,3240
DATA B7CC,4134,840C,3F50,8501,DC20,7522,8414,2985,2C8E,D2F3
DATA 600E,E509,501F,FFE0,C56C,100C,429A,181B,281A,0ADC,5527
DATA 5520,9E67,0040,AA38,3031,D038,5031,0137,01E5,FE58,7AA1
DATA 6885,88FC,0554,303E,216E,6524,2D14,B048,0059,6401,12F7
DATA 0078,625B,7F8F,F2F2,3012,8018,1031,8028,5892,5010,D533
DATA 9825,6101,0014,001C,6150,5E60,A0A8,2C82,5391,050A,C995
DATA 8000,4310,4801,C330,F802,0391,4900,6094,8099,3A08,E929
DATA 400F,ACBE,6304,0408,6E15,BFF9,1906,0464,0302,2115,8C06
DATA 4DA6,10F8,BFC0,6129,44F3,3022,4551,1851,80D1,304B,96F1
DATA 55F8,D870,0032,0250,1290,6407,0899,8089,0955,C683,2841
DATA C88E,1B20,43C4,2150,19E8,40B1,4274,20B5,1CBF,CFC5,7192
DATA 1DA0,0A35,0CBF,C7D4,8D40,0D59,8A24,3883,0301,8280,1E58
DATA 2474,E4EA,FE63,A804,C609,CAB8,12D3,FCBF,987D,29D2,DBD8
DATA 0391,1009,C9DD,FD86,4390,5806,F020,1E60,4010,3208,7BDE
DATA 8004,0220,0201,3001,0084,0080,4A00,4023,0020,1380,D276
DATA 1008,2008,0490,0402,2802,0134,0100,8600,804B,0040,CCD2
DATA 2380,2013,C010,0810,0822,0228,1054,04D0,20B8,0044,3938
DATA 2205,3020,12C1,101B,8104,2001,C29C,8080,4140,4024,CBB1
DATA A020,1350,1008,6808,04B4,0402,3A02,0103,0100,9180,ABFD
DATA 8044,C040,2660,2010,B010,0958,0804,EC04,020E,0201,B0C5
DATA 2701,008B,8080,4BC0,4148,1FC0,8048,0040,2600,2012,8496
DATA 8010,09C0,0804,9004,0228,0201,3401,0086,0080,4700,E663
DATA 4020,4020,1120,1009,9008,0428,0402,3402,0106,0100,4367
DATA 9D0A,800E,8540,3A04,0234,2A00,1A15,000D,0A80,8C08,5588
DATA 0468,540A,6040,2720,2012,5010,09A8,0804,B404,027A,8551
DATA 0201,2301,0089,8080,4CC0,4025,42A0,7702,0127,0100,1D33
DATA 8B80,804C,8540,7E04,022C,0201,3601,0082,0E40,F408,2156
DATA 0404,0402,4202,0B20,2604,0230,0201,6401,00B3,0080,2A7E
DATA 4B80,402B,C020,1510,1008,C808,058C,0402,5602,0147,7A25
DATA 0104,501F,0201,2A01,0550,0D02,0150,0104,D0B9,020A,D980
DATA 6116,1004,0C82,4F00,2018,09E4,0000,03EB,0000,0735,C639
DATA 0000,03F2,x,C5C8
```

Was noch fehlt, ist ein passendes Icon. Am besten kopieren Sie sich im Shell mit

```
COPY AmigaBASIC.info TO "KICKSTART-EDITOR V1.5.info"
```

ein passendes Icon dazu.

6.3.1 Die Bedienung des Programms und besondere Features

Das Programm ist voll menügesteuert. Mit der Tastatur können DIREKT Änderungen im Kickstart durchgeführt werden. Der Editor akzeptiert im Hexadezimal-Modus die Tasten 0-9 und a-f sowie die Cursor-Tasten. Im ASCII-Modus wird jeder Tastendruck (mit Ausnahme der Cursor-Tasten) als eingegebener Wert interpretiert. In beiden Modi kann mit den Cursor-Tasten im Betriebssystem herumgefahren werden. Um den Cursor an eine beliebige Stelle zu bringen, kann diese sowohl im ASCII-, als auch im HEX-Feld mit der Maus angeklickt werden.

Die Menüs im einzelnen:

"Programm"	
"Info"	Autor, (c)-Inhaber und Version werden angezeigt
"Ende"	Entspricht der Betätigung des Schließknopfes vom Editor-Fenster und führt nach einer Sicherheitsabfrage zum Programmende.
"Kickstart"	
"Lese ROM"	Das aktuelle Betriebssystem wird nach einer Sicherheitsabfrage in den Editor-Buffer kopiert und steht damit zur Bearbeitung zur Verfügung.
"Laden"	Ein Betriebssystem wird von einer Kickstart-Diskette in den Editor-Buffer geladen.
"Speichern"	Die Checksumme des Betriebssystems wird berechnet und eine Diskette als bootfähige Kickstart-Disk formatiert.
"Editor"	
"ASCII"	Umschaltung auf Eingabe von Text. Der Cursor im HEX-Feld wird zwei Zeichen breit (1Byte).
"HEX-Modus"	Umschaltung auf Eingabe hexadezimaler Zahlen. Der Cursor im HEX-Feld wird auf Nibble-Breite verkleinert.
"Lade Datei"	Lädt ein mit einem beliebigen Assembler geschriebenes Programm an eine beliebige Stelle des Betriebssystems. Dabei werden die vorher in diesem Bereich liegenden Daten überschrieben. Ein solches Programm muß mit folgenden Daten beginnen: 1. DC.B "KICK" — Anfangs-Kennung für Editor 2. DC.L Adresse — Wert > = $FC0000 (wohin) 3. DC.L Länge — Anzahl folgender Bytes 4. Routine — Ihr Programm

"Zeige Adr."	Das Betriebssystem wird ab der einzugebenden Adresse angezeigt.
"Z. Anfang"	Das Betriebssystem wird ab $FC0000 angezeigt
"Extras"	
"Zeige RES"	Es werden nacheinander alle Betriebssystem-Module mit sämtlichen Daten in einem Requester angezeigt. Die Anwahl von "Okay" führt zur Anzeige ab der gewählten Residentstruktur im Editor-Fenster.
"Linke RES"	Automatische Beschleunigung der Reset-Routine durch Verketten der Resident-Module.
"Suche Z."	Byteweises Suchen nach einer anzugebenden Zeichenfolge. Wenn gefunden, Okay anklicken, um den Bereich anzuzeigen.

Verändern der Größe und Position des CLI-Einschalt-Windows
Wählen Sie im "Extras"-Menü den Punkt "Suche Zeichenfolge". In dem erscheinenden Requester geben Sie ein:

```
CON:0/0/
```

Der Editor meldet sich mit einem Requester und der Anzeige von "CON:0/0/640/200/AmigaDOS" im ASCII-Feld. Bestätigen Sie mit "Okay", schalten Sie den Editor auf ASCII-Modus um und ändern Sie die Angabe nach Ihren Wünschen (beispielsweise vergrößert "CON:0/0/640/256/SMmagic!" das Einschaltfenster auf volle PAL-Größe nebst Titeländerung). Ein Stück weiter unten sehen Sie die Einschaltmeldung. In diese lassen sich auch Steuercodes einbetten, die etwa Schriftart oder Aussehen des Windows bestimmen, s. MACROs mit ALIAS.

6.3.2 Ersetzen der fehlerhaften PAL-Test-Routine

Die PAL-Test-Routine hat die Aufgabe, festzustellen, ob 200 (NTSC-Norm) oder 256 Zeilen (PAL-Norm) dargestellt werden können, um welche eingebauten Video-Chips es sich bei Ihrem Gerät handelt. Hier haben die Betriebssystem-Entwickler einen schwerwiegenden Fehler eingebaut. So kommt es auf die Position des Elektronenstrahls an, ob die Routine funktioniert oder nicht. Handelt es sich um ein PAL-Gerät, so soll diese Routine im Register D0 den Wert 4 zurückliefern, anderenfalls den Wert 1. Zu

finden ist die recht lange Routine bei Kickstart 1.3 im Bereich $FCB00C bis exklusive $FCB04C (Kick 1.2: $FCB058 bis excl. $FCB096). Da es für einzubauende Erweiterungen sinnvoll ist, so viel Platz wie möglich zu sparen, ersetzen wir die ganze Routine einfach durch 4 Byte:

```
CODE                          Programm
-------------------------------------------------------------
(4B49434B00FCB00C00000004)    DC.L "KICK",$FCB00C,Ende-Start

                          Start:
7004                        MOVEQ #4,D0
4E75                        RTS
                          Ende:
```

Sie können die Codes auch direkt im Hex-Modus eingeben (nur nicht die in Klammern stehenden!). Wir erhalten hiermit ab der Adresse $FCB010 einen verfügbaren ROM-Bereich von 60 Bytes und immer einen Bildschirm in voller PAL-Auflösung.

6.4 Virus-festes Betriebssystem

Das Problem bei der Erkennung von Boot-Disks ist der Einsprung in das im Bootblock liegende Programm (die DOS-Initialisierungs-Routine). Auf diesem Weg gelangen die meisten Viren in den Computer und richten mehr oder weniger viel Schaden an. Leider verwenden heute viele Programme einen sogenannten Bootblock-Loader, durch den das Programm (nur über den Einsprung in diesen Loader) geladen und gestartet wird. Wir können daher nicht einfach diesen Einsprung abschalten, sondern müssen dem Benutzer die Entscheidung überlassen, ob das im Bootblock liegende Programm gestartet werden soll. Im Regelfall heißt das, daß das DOS zunächst ohne Einsprung initialisiert wird. Kann ein Programm ohne den Bootblock nicht gestartet werden, führt man einen Reset durch und läßt den Bootblock anspringen. Programmiert wird so eine Routine folgendermaßen:

1. Nach der Testroutine "Handelt es sich um eine Bootdisk?" wird in eine eigene Routine eingesprungen.

2. Abfrage beim User (am einfachsten über die Maustasten).
3. Einsprung in Bootblock oder Vortäuschen eines normalen Bootblocks (wie von INSTALL).

Wir müssen daher zunächst die Strap-Routine umleiten. Für unsere neue Routine benötigen wir einige Bytes, die wir durch die neue PAL-Test-Routine gewonnen haben. Auf diese Adresse lenken wir den Strap um:

```
CODE                        Programm
-------------------------------------------------------
(4B49434B00FE85C200000006)   DC.L "KICK",$FE85C6,Ende-Start

                            Start:
              4EF900FCB010    JMP $FCB010
                            Ende:
```

Auch hier lassen sich die Änderungen leicht von Hand vornehmen (bei Kickstart 1.2 sind die Adressen $FE8A2C und $FCB058). Überschrieben wurde der Einsprung in den Bootblock (JSR 12(A4)) und der anschließende Test (TST.L D0).

Fehlt unsere Routine:

```
CODE                        Programm
-------------------------------------------------------
(4B49434B00FCB0100000003A)   DC.L "KICK",$FCB010,Ende-Start

                            Start:
         0839000600BFE001     BTST #6,$BFE001 ;Maustaste?
                  671C        BEQ.S Einsprung ;JA--->
            43F900FC52E4      LEA $FC52E4,A1  ;V1.2: $FC5278
                4EAEFFA0      JSR -96(A6)     ;FindResident
                  4A80        TST.L D0
                  670A        BEQ.S NixDOS
                  2040        MOVEA.L D0,A0
                20680016      MOVEA.L 22(A0),A0
                  7000        MOVEQ #0,D0
                  6008        BRA.S Verzweige
                            NixDOS:
                  70FF        MOVEQ #-1,D0
                  6004        BRA.S Verzweige
                            Einsprung:
                4EAC000C      JSR 12(A4)
                            Verzweige:
```

```
          4A80    TST.L D0
          6706    BEQ.S AllesOkay
4EF900FE85CE      JMP $FE85CE      ;V1.2: $FE8A34
                AllesOkay:
4EF900FE85F0      JMP $FE85F0      ;V1.2: $FE8A56
                Ende:
```

Bei Betriebssystem V1.2 bitte die Adressen angleichen (s. Remarks). Haben Sie Ihr Betriebssystem derart ausgestattet, ist es nicht mehr möglich, einen Boot-Virus in Ihrem Amiga zu finden (es sei denn, Sie halten ständig beim Reset die linke Maustaste gedrückt. Dann nämlich wird das im Bootblock befindliche Programm ausgeführt).

6.5 Betriebssystem-Version patchen

Nun haben Sie so wunderschöne Änderungen vorgenommen, und äußerlich hat sich nicht viel getan. Wie wäre es daher mit einer eigenen Betriebssystem-Nummer? Zunächst die Änderung der Hand, die Sie zum Einlegen der Workbench auffordert (bewußt binär):

```
DC.L "KICK"
DC.L $FE8C5E
DC.L 60 ;toll, näh, rechnen kann ich auch
DC.W  %0000111110011111,%1100000111000000,%0011111111110000
DC.W  %0000011100000111,%0000001111000000,%0011111111110000
DC.W  %0000011100001110,%0000011111000000,%0011100000000000
DC.W  %0000011100011100,%0000000111000000,%0011100000000000
DC.W  %0000011100111000,%0000000111000000,%0011111110000000
DC.W  %0000011101110000,%0000000111000000,%0011111111100000
DC.W  %0000011111100000,%0000000111000000,%0000000011110000
DC.W  %0000011111000000,%0000000111000000,%0000000011110000
DC.W  %0000011110000000,%0000011111110011,%0011111111100000
DC.W  %0000011100000000,%0000011111110011,%0011111110000000
```

Für alle, die PROFIMAT noch nicht besitzen. Ab Adresse $FE8C5E eingeben:

```
0F9F C1C0 3FF0 0707 03C0 3FF0 070E 07C0 3800 071C 01C0 3800
0738 01C0 3F80 0770 01C0 3FE0 07E0 01C0 00F0 07C0 01C0 00F0
0780 07F3 3FE0 0700 07F3 3F80
```

Aber auch die Titelleiste der Workbench muß noch geändert werden. Wählen Sie den Punkt "Suche Zeichenfolge" und tippen Sie:

```
Workbench release
```

Der Editor zeigt Ihnen ab Adresse $FECD4A:

```
Workbench release 1.2.  %ld free memory
```

Ändern Sie es im ASCII-Modus so, wie Sie wollen. Möchten Sie auch weiterhin die Anzahl freier Bytes gezeigt bekommen, müssen Sie die Angabe %ld in dieser Zeichenkette haben. Diese Angabe ist ein Code für die Exec-Function RAWDOFMT, die anstelle des %ld ein Langwort (l) dezimal (d) ausgibt. Als Assembler-Freak hätten Sie es lieber Hexadezimal? Dann geben Sie ein: '$%lx'. Sollen auch die führenden Nullen angezeigt werden und acht Stellen ausgegeben werden, schreiben Sie:

```
SMmagic Workbench 1.5  $%08lx Byte free
```

7. Icons

Die gesamte grafisch gestaltete Benutzeroberfläche, die Workbench des Amiga, benötigt Symbole für die Darstellung von Programmen, Datenfiles, Inhaltsverzeichnissen und Disketten. Sogar die Fenster sind im eigentlichen Sinne Symbole. Wir wollen uns hier aber nur mit den Icons beschäftigen. Diese Icons stellen durch ihre Bilder Objekte dar, mit denen der Benutzer arbeiten kann. Zur Vereinfachung braucht er nicht mehr den Namen eines Programms, das gestartet werden soll, einzugeben, sondern nur das dazugehörige Icon anzuklicken. So erspart man sich besonders bei Files, die in sehr verschachtelten Directories liegen, die Eingabe des gesamten Pfadnamens, wie es ja im CLI mehr oder weniger immer nötig ist.

7.1 Die verschiedenen Icon-Typen

Nun ergibt sich aber ein Problem: Alle Symbole können für verschiedene Objekte stehen. Hierbei ist nicht nur der Unterschied zwischen den einzelnen Programmen gemeint, sondern man hat ja auch zwischen Directories und Disketten zu unterscheiden. Deshalb möchten wir folgendes anmerken: Das Symbol, das der Benutzer für eine Sache definiert, sollte mit dem Zweck übereinstimmen, muß es aber nicht! Sie sind also nicht gezwungen, für Unterverzeichnisse immer ein Schubladen-Icon zu zeichnen.

Im Gegenteil! Sie können sogar, aus welchem Grund auch immer, ein Directory mit einem Programmsymbol belegen. So verwirren Sie zwar den Benutzer, die Verwaltung läuft aber immer noch korrekt. Aus diesem Grund sollten wir uns darauf einigen, daß eine Diskette mit einem diskettenähnlichen Objekt, also einem Diskettensymbol belegt wird usw. Unter dieser Voraussetzung können die einzelnen Typen genauer besprochen werden. Wie Sie aus dem Kapitel über die Info-Funktion der Workbench-Diskette wissen, gibt es folgende Icon-Typen:

Die Icon-Typen

Name	Bezeichnung	Objekt	Nummer
Disketten-Icon	WBDISK	Grundstock der Diskette	1
Verzeichnis-Icon	WBDRAWER	Inhaltsverzeichnis	2
Programm-Icon	WBTOOL	Startbares Programm	3
Datei-Icon	WBPROJECT	Daten zu einem Programm	4
Mülleimer-Icon	WBGARBAGE	Das Müllverzeichnis	5
Device-Icon	WBDEVICE	Icon für allgemeines Gerät	6
Kickstart-Icon	WBKICK	Die Kickstart Diskette	7

Dabei hat die Workbench für nicht grafisch definierte Icons noch eine Sondereinlage bereit. Diese Icons stellen bestimmte Zustände von Laufwerken dar. Es gibt hier folgende Typen:

Modus	Beschreibung
DF0:NDOS	Es handelt sich um eine formatierte Diskette, die aber nicht das DOS-Format besitzt.
DF0:BAD	Die Diskette ist nicht formatiert.
DF0:COPY	Es handelt sich um eine angefangene Kopie.
DF0:BUSY	Das Laufwerk ist beschäftigt und kann nicht angesteuert werden.

Da es verschiedene Icon-Typen gibt, kann man folgern, daß die Workbench noch einige zusätzliche Informationen verwaltet. Sehen Sie dazu folgendes:

- Die Informationen eines Disketten-Icons entsprechen denen eines Schubladen-Icons. Das Schubladen-Icon speichert zusätzlich zum Bild, das ja alle Icons zwangsläufig haben müssen, noch Daten zu dem Fenster, das bei einem Doppelklick geöffnet wird.

- Die Datei-Icons gleichen eigentlich den Programm-Icons, nur wird bei ihnen auch noch das Programm, mit dem sie erstellt wurden, namentlich festgehalten. Dies dient dazu, daß man durch einfaches Doppelklicken auf das Dateisymbol der Workbench mitteilt, daß zuerst das Hauptprogramm geladen werden soll und diesem dann anschließend der Inhalt der Datei übergeben wird.

- Der Mülleimer ist eigentlich nur eine Sonderform der Schublade. Ihn kann man im Gegensatz dazu jedoch nicht von einem Inhaltsverzeichnis zum anderen bewegen und auch nicht auf der Workbench ablegen.

7.2 Der Aufbau eines Icons

Kommen wir zu den Strukturen, nach denen Sie dann Ihre eigenen Icons herstellen können. Die Icon-Daten findet man, indem man sich ein Inhaltsverzeichnis mit dem CLI genauer ansieht. Dort ist zu jedem File, zu dem auch ein Icon existiert, noch ein weiteres File gleichen Namens vorhanden, dem das Suffix ".info" anhängt. Dieses File enthält alles, was die Workbench an Informationen braucht. Sehen wir uns den Inhalt deshalb genauer an.

7.2.1 Die DiskObject-Struktur

Zuerst beginnt jedes Iconfile, egal, um welchen Typ es sich handelt, mit einer sog. DiskObjekt-Struktur. Hier findet man allgemeine Informationen. Dazu diese Tabelle:

DiskObject-Struktur

Bezeichnung	Parameter	Bytes	Offset
do_Magic	magische Zahl	2	0
do_Version	Versionsnummer	2	2
do_Gadget	Klickstruktur	4	4
gg_LeftEdge	Klickber. links	2	8
gg_TopEdge	Klickber. oben	2	10
gg_Width	Klickber. Breite	2	12
gg_Height	Klickber. Höhe	2	14
gg_Flags	Invert-Flag	2	16
gg_Activation	$0003	2	18
gg_Type	$0001	2	20
gg_GadgetRender	Zeiger1 Bilddaten	4	22
gg_SelectRender	Zeiger2 Bilddaten	4	26
gg_IntuiText	"not used??"	4	30
gg_MutualExclude	"not useable!"	4	34
gg_SpecialInfo	"not useable!"	4	38

gg_GadgetID	"for own use!"	2	42
gg_UserData	"your Pointer!"	4	44
do_Type	Icon-Typ	1	48
do_DefaultTool	Text-Struktur	4	50
do_ToolTypes	Text-Struktur	4	54
do_CurrentX	aktuelle x-Position	4	58
do_CurrentY	aktuelle y-Position	4	62
do_DrawerData	WindowStruktur	4	66
do_ToolWindow	Eigenes Programm-Window	4	70
do_StackSize	Speicherplatzreservierung	4	74

Zu Anfang muß die "magic number" $E310 stehen! Sie teilt dem System mit, daß es sich um ein Icon handelt. Danach folgt die Versionsnummer, die bisher immer $0001 beträgt. Sie finden in der obigen Tabelle übrigens zu jedem Eintrag noch eine weitere Zahl, die angibt, in wieviel Bytes der Wert angegeben werden muß. Das ist einerseits wichtig zum Auslesen, andererseits brauchen Sie den Wert, um den Variablentyp zu bestimmen, wenn Sie nicht in C programmieren.

Nach den ersten beiden Daten folgen vier unbenutzte Bytes, die normalerweise für eine weitere Gadget-Klickstruktur verwendet werden. Und nun wird es schon etwas komplizierter: Das eigentliche Symbol wird noch in zwei zu unterscheidende Bereiche unterteilt. Da gibt es einmal den gesamten Grafikbereich. Zum anderen muß man aber noch extra angeben, in welchem Teil überhaupt auf das Anklicken reagiert werden soll. Dazu folgen die x- und y-Offsets zur eigentlichen Position, die die linke obere Ecke des Klickfeldes bestimmen. Danach stehen die Breite und die Höhe.

Wichtig ist auch zu wissen, daß unter dem Klickbereich der Text gedruckt wird, der jeweils unter einem Icon steht. Zieht man nämlich den Klickbereich höher, so wird der Text vielleicht sogar in die Grafik geschrieben. Jetzt wird es wieder einmal ganz wichtig. Wir befinden uns in einer Gadget-Struktur. Mit dem nächsten Wert geben wir an, in welcher Weise sich das Bild verändern soll, wenn es durch einen Klick aktiviert wurde. Es stehen drei Möglichkeiten zur Auswahl:

1. Als erstes können Sie wählen, daß das gesamte rechteckige Feld, in dem sich das Bild befindet, einfach invertiert wird. Diese Möglichkeit ist zwar einfach, aber nicht gerade genial. Hierfür schreiben Sie eine 4 in das Flag.
2. Nicht alles, sondern nur das Gezeichnete wird invertiert. Das wirkt schon etwas besser und macht nicht so einen klobigen Eindruck. Das Nonplusultra ist es aber auch nicht. Für diesen Modus steht eine 5.
3. Erst die letzte Version gibt Hoffnung auf eine ausgefeilte grafische Gestaltung. Wählen Sie diesen Typ, so wird anstatt des ersten Symbols ein vollständig neues Bild dargestellt. Somit können Sie jede beliebige Änderung vornehmen, was Sie mit einer 6 in dem Flag anzeigen.

Als nächstes folgen in unserer DiskObject-Struktur zwei konstante Zahlenwerte $0003 und $0001. Der erste steht für die Aktivierungsart, im zweiten wird ein Boolean-Gadget markiert. Danach folgen die Zeiger auf die Icon-Grafikdaten. Je nachdem, ob Sie das Umschalten zwischen zwei Grafiken gewählt haben, muß auch der zweite Zeiger initialisiert werden.

Darauf folgen 18 Bytes, die das System für normale Gadgets benötigt. Allerdings wäre die Benutzung hier ziemlich sinnlos. Am besten ist es, wenn Sie diese mit Nullen auffüllen. Wichtig wird es bei dem nächsten Parameter. Er entscheidet, welcher Icon-Typ dem Benutzer zur Verfügung steht. Setzen Sie deshalb die Nummern ein, die Sie der oben aufgelisteten Tabelle entnehmen. Weil dies aber in einem Byte angegeben werden soll und der Prozessor nur gerade Adressen adressieren kann, steht gleich danach ein Füllbyte.

Je nachdem, welcher Typ gewählt wurde, müssen nun die einzelnen Zeiger gesetzt werden. Zunächst gilt das für den Zeiger auf die DefaultTool-Struktur. Wie diese aufgebaut sein muß, sehen Sie später. Dann der Zeiger auf die ToolTypes-Struktur. Auch dazu später mehr.

Für die Positionierung wird in der DiskObject-Struktur als weiteres die aktuelle x- und y-Position gespeichert. Man hat aber auch die Möglichkeit, durch die Koordinatenangaben von $80000000, $80000000 der Workbench mitzuteilen, daß sie nach einer geeigneten Position suchen soll. Dieser Wert wird NO_ICON_POSITION genannt. Somit verdeckt ein selbst erzeugtes Icon nicht irgendwelche anderen, die sich an der gleichen Stelle befinden. Es folgt ein Zeiger auf die Fensterdaten, falls diese erforderlich sind, und ein Zeiger auf die ToolWindow-Struktur.

Zum Schluß kommt noch die Stack-Tiefe, damit die Workbench weiß, wieviel Speicher für dieses Programm oder diese Daten reserviert werden muß. Dabei hat der Wert eines Datenfiles eine höhere Priorität als der eines Hauptprogramms. Damit wird bezweckt, daß man z.B. für besonders große Datenfelder das File gleich entsprechend reservieren kann.

7.2.2 Die Drawer-Struktur

Nachdem Sie nun über den Aufbau der allgemeinen DiskObject-Struktur informiert sind, beschäftigen wir uns mit den einzelnen Typen. Behandelt wird zuerst die Drawer-Struktur, die der einer Diskette gleich ist. Aber auch die Inhaltsverzeichnisse und der Mülleimer benutzen diese Struktur. Sie enthält alle nötigen Daten zum Öffnen eines neuen Directory-Fensters. Dazu folgende Tabelle:

DrawerData-Struktur

Bezeichnung	Parameter	Bytes	Offset
wi_LeftEdge	Linke Ecke	2	0
wi_TopEdge	Obere Ecke	2	2
wi_Width	Breite	2	4
wi_Height	Hoehe	2	6
wi_DetailPen	Zeichenfarbe 1	1	8
wi_BlockPen	Zeichenfarbe 2	1	9
wi_IDCMPFlags	Gadget-Flags	4	10
wi_Flags	Window-Flags	4	14

wi_FirstGadget	Gadget-Struktur	4	18
wi_CheckMark	Abhaken	4	22
wi_Title	Titeltext	4	26
wi_Screen	Screenpointer	4	30
wi_BitMap	Window-Bitmap	4	34
wi_MinWidth	min. Breite	2	38
wi_MinHeight	min. Hoehe	2	40
wi_MaxWidht	max. Breite	2	42
wi_MaxHeight	max. Hoehe	2	44
wi_Type	$0001	2	46
actx-pos	aktuelle x-Position	4	48
acty-pos	aktuelle y-Position	4	52

Wie Sie vielleicht schon erkannt haben, handelt es sich um eine vollständige NewWindow-Struktur, die um die Koordinaten für die aktuelle Position erweitert wurde. Für diejenigen, denen diese noch nicht so bekannt ist, hier die Erläuterungen:

Am Anfang stehen die Koordinaten der linken oberen Ecke und die Ausdehnung. Wurde das Window vom Benutzer verschoben und danach geschlossen, die Diskette aber nicht aus dem System entfernt, so wird das Directory-Window nicht an der angegebenen Position geöffnet, sondern an der, die die aktuellen Koordinaten angeben.

Des weiteren folgen Angaben zur farblichen Gestaltung. Mit den Werten legt man fest, in welcher Farbe die Linien und Blöcke eines Windows gezeichnet werden. Hier stehen im Normalfall jeweils $FF für -1, was bedeutet, daß die Farben denen der Screen angepaßt werden sollen. Somit ist eine durchgehend gleiche Gestaltung gewährleistet.

Die nächsten Bytes beinhalten einige Zeiger und Flags, die eigentlich systemintern verwaltet werden. Zuerst die IDCMP-Flags. Mit ihnen wird die Reaktion auf irgendwelche Aktionen am Window festgelegt. Über die Window-Flags bestimmt man die Eigenschaften des Directory-Fensters. Darauf folgen fünf Zeiger auf Strukturen oder Speicherbereiche, deren Veränderung wesentliche Kenntnisse des Betriebssystems voraussetzt.

Da sich alle Windows in ihrer Größe beliebig einstellen lassen, müssen dementsprechend auch die Minimum- und Maximumwerte feststehen. Diese werden mit MinWidth, MinHeight, MaxWidth und MaxHeight eingestellt. Als letztes Datum der Window-Struktur geben wir an, daß dieses Window auf der Workbench erscheinen soll. Wie oben schon erwähnt, wird die Struktur mit den aktuellen Koordinaten des Fensters abgeschlossen. So kann man sich für das momentane Arbeiten mit bestimmten Inhaltsverzeichnissen eigene Positionen einrichten, die wichtige Arbeitsflächen nicht verdecken.

7.2.3 Die Image-Struktur

Die Image-Struktur wird bei jedem Icon benötigt. Sie enthält die gesamte Grafik und kommt, wenn es angefordert wird, sogar zweimal in einem File vor.Image

Bezeichnung	Parameter	Bytes	Offset
im_LeftEdge	Linke Ecke	2	0
im_TopEdge	Obere Ecke	2	2
im_Width	Breite	2	4
im_Height	Hoehe	2	6
im_Depth	Tiefe	2	8
im_ImageData	Bitplanes-Zeiger	4	10
im_PlanePick	Bild-Daten	1	14
im_PlaneOnOff	Verwendung	1	15
im_NextImage	Nächstes Bild	4	16

Der Aufbau ist denkbar einfach: Nach ein paar Informationen zur Größe und Lage folgen mehrere Bitmaps, die die eigentlichen Bilder enthalten. Die Anzahl der Bitmaps hängt von der Tiefe des Screens ab. Normalerweise hat die Workbench eine Tiefe von zwei Bitmaps, entsprechend sollte auch das Image aufgebaut sein.

Obwohl die Position des Icons schon angegeben worden ist, und zwar in der DiskObject-Struktur, wird diese Angabe beim Image noch einmal wiederholt. Die Position ist nun aber ein Offset zu

dieser Angabe. Über die Breite, Höhe und Anzahl der Bitplanes brauchen wohl keine Worte mehr verloren zu werden, da es sich hier genau wie bei den anderen Strukturen verhält.

Die nächsten vier Bytes sind ein Zeiger auf die tatsächlichen Grafikdaten. Dabei können durch die beiden weiteren Parameter noch einige Veränderungen vorgenommen werden. So gibt man unter PlanePick an, welche der vorhandenen Bitplanes überhaupt verwendet werden sollen, um das Bild darzustellen. Und mit PlaneOnOff entscheidet man, was mit den Bitplanes geschehen soll, die nicht zur Darstellung benutzt werden. Der letzte Parameter ist ein Zeiger auf eine andere Image-Struktur. So hat man die Möglichkeit, mehrere Objekte zu einem Gefüge zusammenzubinden. Diese Funktion wird aber auf der Workbench nicht unterstützt.

Auf die Image-Struktur folgen die Bytes der einzelnen Bitplanes. Zuerst Bitplane 1, dann Bitplane 2 und, wenn noch mehr Bitplanes erforderlich, auch noch Bitplane 3 usw. Die Anzahl der Bytes errechnet sich aus der angegebenen Breite, aufgerundet auf das nächste Vielfache von 16, durch 8 mal Pixel-Anzahl der Höhe. Diese Bytes benötigt man, um eine Bitplane abzubilden. Für jede weitere werden natürlich genauso viele gebraucht.

7.2.4 Der DefaultTool-Text

Im Gegensatz zu der Image-Struktur, die bei jedem Icon gebraucht wird, benötigt man den DefaultTool-Text nur bei den Disketten und den Datenfiles, da hier angegeben wird, mit welchem Programm das Objekt erstellt wurde und welches Programm für die Bearbeitung oder Verarbeitung gebraucht wird. Deshalb steht bei Disketten immer der Text "SYS:System/DiskCopy", weil beim Übereinanderlegen zweier Disketten dieses Systemprogramm aufgerufen werden muß, damit die eine auf die andere kopiert werden kann. Wenn Sie diesen Text entfer-

nen, kann man die Diskette nicht mehr über diese Funktion kopieren. Aber damit sind wir schon bei der Anwendung. Zuerst sollten wir uns mit dem Aufbau beschäftigen. Dazu hier die Parameter:

DefaultTool

Bezeichnung	Parameter	Bytes
char_num	Zeichenzahl	4
chars	Zeichen	x
char_end	Null-Byte	1

Wie Sie sicherlich gleich erkannt haben werden, besteht diese Liste nur aus einem wirklich festen Datum, der Anzahl der Zeichen. Alles andere ist davon abhängig. Es bleibt nur noch zu erwähnen, daß der Text mit einem Nullbyte abgeschlossen werden muß, damit das Ende eindeutig identifiziert werden kann.

7.2.5 Der ToolTypes-Text

Wie schon bei der Besprechung der Info-Funktion der Workbench angeschnitten, bietet die Texteingabe unter Tool Types eine wunderbare Möglichkeit, dem Hauptprogramm zusätzliche Informationen zu übergeben. So kann es sich z.B. bei einem Text um das IFF-Format handeln. Das Programm benötigt aber noch weitere Informationen, die in dem IFF-Format nicht berücksichtigt werden. Diese kann man dann einfach hier hineinpacken, und das File kann gleichzeitig noch von allen anderen Programmen, die das IFF-Format voraussetzen, benutzt werden.

ToolTypes

Bezeichnung	Parameter	Bytes
string_num	Textanzahl	4

Genau wie der DefaultTool-Text ist auch die Größe des ToolTypes-Feldes ziemlich schwer abzuschätzen. Damit eine Auswertung aber auch nicht ins Leere greift, steht am Anfang die Anzahl der Zeichenketten. Für die Berechnung muß man erst

die tatsächliche Zahl um eins erhöhen und dann mit vier multiplizieren. Diese Zahl findet man also, wenn man das File liest. Will man die Daten auswerten, muß man entsprechend umgekehrt verfahren.

Danach folgt ein Feld mit Zeichenketten, die genau wie unter DefaultTool mit der Länge beginnen und mit einem Nullbyte enden. Wie viele Zeichenketten es sind, haben Sie ja vorher aus der verschlüsselten Zahl berechnet.

7.2.6 Der Icon-Analyzer

Bisher haben Sie zwar einiges über die Strukturen kennengelernt, aus denen sich ein Icon zusammensetzt. Trotzdem können Sie dies nicht praktisch erfahren, indem Sie z.B. ein .info-File näher untersuchen. Dafür habe ich nun folgendes Programm in BASIC geschrieben. Es liest nach Angabe des Namens das File ein und zeigt nacheinander die Parameternamen mit den entsprechenden Werten dieses Files. Am einfachsten ist eine Analyse, wenn Sie sich die Liste auf dem Drucker ausgeben lassen. Sie brauchen es aber nicht.

```
' Icon - Analyser V 1.3
'
' (p) by Wgb im Mai 1988
' (c) by DATA BECKER
'
' Autor  : Wolf-Gideon Bleek
' Grüße  : a-ha "there's never a forever thing"
' Version: 2.0
' Kickstart 1.2/1.3

DEFLNG a-z
DIM Groesse(200,2)

LIBRARY ":bmaps/icon.library"
DECLARE FUNCTION GetDiskObject LIBRARY
DECLARE FUNCTION PutDiskObject LIBRARY

RESTORE DiskObjekt
INPUT "Bitte geben Sie den Namen des Icon-Files ein"; File$
IF File$ = "" THEN END
Adresse = GetDiskObject(SADD(File$))
IF Adresse = 0 THEN END
```

```
PRINT "Das Icon '";File$;".info' liegt ab";Adresse

WBObjektOut(Adresse)
IF Groesse(10,0) THEN WBImageOut(Groesse(10,0))
IF Groesse(11,0) THEN WBImageOut(Groesse(12,0))
IF Groesse(22,0) THEN WBDrawerOut(Groesse(22,0))
FreeDiskObject(Adresse)
END
'-------------------------
SUB WBDrawerOut(Adresse) STATIC
 PRINT
 PRINT "DrawerData - Struktur:"

 RESTORE DrawerData

 CALL DataOut(Adresse, 20)
END SUB

SUB WBImageOut(Adresse) STATIC
 PRINT
 PRINT "Image - Struktur:"

 RESTORE Image

 CALL DataOut(Adresse, 9)
END SUB

SUB WBObjektOut(Adresse) STATIC
 PRINT
 PRINT "DiskObject - Struktur:"

 RESTORE DiskObjekt

 CALL DataOut(Adresse, 25)
END SUB

SUB DataOut(Adresse, Daten%) STATIC
 SHARED Zaehler, Groesse()
 Start = Adresse
  FOR i = 1 TO Daten%
    READ Text$, Erkl$, Bytes
    Groesse(Zaehler,1) = Start
    Groesse(Zaehler,2) = Bytes
    IF Bytes = 1 THEN
      Wert = PEEK(Start)
      Start = Start + 1
    ELSEIF Bytes = 2 THEN
      Wert = PEEKW(Start)
      Start = Start + 2
    ELSEIF Bytes = 4 THEN
      Wert = PEEKL(Start)
      Start = Start + 4
```

```
    END IF
    PRINT LEFT$(Text$,15);TAB(20);LEFT$(Erkl$,25);TAB(50);
    PRINT "$";RIGHT$("0000000"+HEX$(Wert), Bytes*2);
    PRINT TAB(60);Start-Adresse-Bytes;TAB(65);Zaehler
    Groesse(Zaehler,0) = Wert
    Zaehler = Zaehler + 1
  NEXT i
END SUB


'--------------------------
DiskObjekt:
  DATA do_Magic,magische Zahl,2
  DATA do_Version,Versionsnummer,2
  DATA do_Gadget,weitere Klickstruktur,4
  DATA gg_LeftEdge,Klickbereich links,2
  DATA gg_TopEdge,Klickbereich oben,2
  DATA gg_Width,Klickbereich Breite,2
  DATA gg_Height,Klickbereich Höhe,2
  DATA gg_Flags,Invert-Flagge,2
  DATA gg_Activation,festgelegt $0003,2
  DATA gg_Type,festgelegt $0001,2
  DATA gg_GadgetRender,Zeiger auf 1. Bilddaten,4
  DATA gg_SelectRender,Zeiger auf 2. Bilddaten,4
  DATA gg_IntuiText,unbenutzt,4
  DATA gg_MutualExclude,unbenutzt,4
  DATA gg_SpecialInfo,unbenutzt,4
  DATA gg_GadgetID,unbenutzte Gadget-ID,2
  DATA gg_UserData,unbenutzter Gadget-Pointer,4
  DATA do_Type,Icontyp,2
  DATA do_DefaultTool,Text-Struktur,4
  DATA do_ToolTypes,Text-Struktur,4
  DATA do_CurrentX,aktuelle x-Position,4
  DATA do_CurrentY,aktuelle y-Position,4
  DATA do_DrawerData,Window-Struktur,4
  DATA do_ToolWindow,Eigenes Programm-Window,4
  DATA do_StackSize,Speicherplatzreservierung,4
DrawerData:
  DATA wi_LeftEdge,Linke_Ecke,2
  DATA wi_TopEdge,Obere_Ecke,2
  DATA wi_Width,Breite,2
  DATA wi_Height,Hoehe,2
  DATA wi_DetailPen,Zeichenfarbe_1,1
  DATA wi_BlockPen,Zeichenfarbe_2,1
  DATA wi_IDCMPFlags,Gadget_Flaggen,4
  DATA wi_Flags,Window_Flaggen,4
  DATA wi_FirstGadget,Gadget_Struktur,4
  DATA wi_CheckMark,Abhaken,4
  DATA wi_Title,Titeltext,4
  DATA wi_Screen,Screenpointer,4
  DATA wi_BitMap,Window_BitMap,4
  DATA wi_MinWidth,min._Breite,2
  DATA wi_MinHeight,min._Hoehe,2
```

```
 DATA wi_MaxWidht,max._Breite,2
 DATA wi_MaxHeight,max._Hoehe,2
 DATA wi_Type,$0001,2
 DATA actx-pos,aktuelle_x-Position,4
 DATA acty-pos,aktuelle_y-Position,4
Image:
 DATA im_LeftEdge,Linke_Ecke,2
 DATA im_TopEdge,Obere_Ecke,2
 DATA im_Width,Breite,2
 DATA im_Height,Hoehe,2
 DATA im_Depth,Tiefe,2
 DATA im_ImageData,BitPlanes_Zeiger,4
 DATA im_PlanePick,Bild_Daten,1
 DATA im_PlaneOnOff,Verwendung,1
 DATA im_NextImage,Nächtes_Bild,4
DefaultTool:
 DATA char_num,Zeichenzahl,4
ToolTypes:
 DATA string_num,Textanzahl,4
```

Programmbeschreibung

Der Icon-Analyzer nutzt für seine Arbeit die Betriebssystemroutinen, die wir in der Icon.Library finden. Dort sind mehrere Funktionen zum Laden, Speichern und Löschen von Icons. Wir nutzen diese drei. Die dann angesteuerten Unterprogramme werten die Datenfelder aus und geben die Werte in einer Tabelle auf den Bildschirm aus. Dazu gibt es die allgemeine Routine DataOut(), die über die Startadresse und die DATA-Zeilen alles erledigt. Diese Routine wird aber nicht direkt angesprungen, sondern von den leichter verständlichen Programmteilen zur Ausgabe der Drawer-, Image- oder WBObjekt-Struktur.

7.3 Eigene Icons erstellen

Nachdem Sie nun ausführlich über die Strukturen informiert sind, aus denen ein Icon besteht, sollten wir daran gehen, dies auch zu nutzen und Icons nach eigenen Bedürfnissen zu schreiben. Es ist allerdings einfacher, ein bereits bestehendes Icon nach seinen Wünschen umzuändern, als ein vollkommen neues herzustellen. Aber diese Methode wird ja z.B. auch beim Icon-Editor der Workbench verwendet.

7.3.1 Zwei Bilder für ein Icon

Hier werden Sie kennenlernen, wie man über die Alternate-Image-Funktion in der Lage ist, beim Anklicken nicht nur das Bild zu invertieren, sondern sogar ein neues Bild darzustellen. Dies ist eine allgemeine Methode, die Sie auf jeden Icon-Typ anwenden können. Später werden Sie dann noch andere Möglichkeiten kennenlernen, die sich z.B. nur auf Schubladen beziehen.

Die Änderung geht davon aus, daß nur der Pointer auf die zweite Image-Struktur gesetzt werden muß und auch noch die entsprechenden Daten angefügt werden. Dieses Problem läßt sich einfacher lösen als man denkt, denn auf der neuen Extras-Diskette befindet sich ein solches Programm. Es erledigt die Arbeit des Zusammenbindens. Wir müssen mit einem Programm, z.B. dem Icon-Editor, zwei Icons erstellen. Wichtig ist nur, daß sie beide die gleiche Größe haben. Nach Angabe des Namens werden sie dann zu einem verschweißt.

Damit können Sie die witzigsten Effekte erzielen: z.B. einen Mülleimer, aus dem schon der Abfall quillt, wenn man ihn anklickt, oder lassen Sie irgend etwas Unerwartetes passieren. Schon eine Schublade, die scheinbar geöffnet wird, sobald sie angeklickt wird, ist bestimmt schöner als die langweiligen auf der Workbench.

Mit IconMerge zum Ziel

IconMerge ist ein Utility, das es uns erlaubt, die Grafiken zweier Icons zu einem neuen zu verschmelzen. Dabei wird die des ersten beibehalten, jedoch beim Anklicken nicht mehr invertiert, sondern die des zweiten gezeigt. Hierfür ruft man ganz einfach IconMerge mit folgenden Parametern über das CLI auf:

```
IconMerge File1 File2 ZielFile
```

Sie brauchen bei den Filenamen nicht das Suffix .info anzuhängen. Dies erledigt IconMerge selbständig. Beim Arbeiten mit IconMerge ist darauf zu achten, daß die beiden Grafiken in

etwa die gleiche Größe haben, denn sonst wird die eine nicht durch die andere vollkommen überdeckt, und dann kann es zu Schwierigkeiten in der Darstellung kommen, die sich in einem Kuddelmuddel äußern. Achten Sie also besonders auf einheitliche Größe.

Wollen Sie ein Icon aus zwei Bildern wieder in ein einzelnes verwandeln, dann hängen Sie noch die Option -s für Split hinten ran. Sie erhalten dann aus einem File zwei neue Icons. Somit ist es auch möglich, das zweite Bild für ein eigenständiges Icon zu nutzen oder einfach nur die Reihenfolge zu tauschen:

```
IconMerge QuellFile File1 File2 -s
```

7.3.2 Text im Bild

Eine weitere Möglichkeit, etwas anders zu machen, ist das Hochziehen des Bildtextes. Wie Sie aus der DiskObjekt-Struktur schon kennen, wird außer dem eigentlichen Grafikbereich des Icons noch der Klickbereich unterschieden. Dieser Klickbereich wird ganz zu Anfang in der DiskObjekt-Struktur in den Parametern 4-7 angegeben. Und als erwähnenswerte Eigenschaft ist zu erkennen, daß genau unter dem Klickbereich die Icon-Unterschrift gedruckt wird. Verkürzt man nun die Höhe des Klickbereiches, so zieht man gleichzeitig auch den Text hoch. So können Sie bei geschickter Bildgestaltung den Text mit in die Grafik einbauen und haben nicht wie sonst den Titel darunter stehen.

7.3.3 Der Icon-Editor

Für diese Änderungen, die ja nur einige Bytes des .info-Files betreffen, benötigen wir ein Programm, das es erlaubt, bestimmte Bytes anzusprechen und diese zu verändern. Außerdem muß man das Geänderte wieder abspeichern können.

Dafür verwenden wir größtenteils das oben beschriebene Analyzer-Programm. Als Ergänzungen werden einige Zeilen eingefügt. Hier sehen Sie noch einmal das ganze Listing. Um Tipparbeit zu ersparen, empfiehlt es sich, die neuen Zeilen in das Analyzer-Listing einzufügen und es unter dem Namen "Icon-Editor" abzuspeichern.

```
DIM DiskObject$(26,3),DiskObject(26)
DIM DrawerData$(20,3),DrawerData(20)
DIM Image$(2,9,3),Image(2,9)
DIM DefaultTool$(2,3),DefaultTool(2)
DIM Adresse(100,3)
ON TIMER(.5) GOSUB KeyTest
TIMER ON
DEF FNSize%(Im)=Image(Im,4)*2*INT((Image(Im,3)+15)/16)
WIDTH 75 : Adr=1 : Adrnr=1
INPUT "Pfadname:";Pfad$
INPUT "Filename:";File$
OPEN Pfad$+File$+".info" FOR INPUT AS 1
  summary$=INPUT$(LOF(1),1)

CLOSE 1
summary$=summary$+STRING$(40,0)
LstBytes:
nr=0 : lst=0
GOSUB LoadHeader
  IF DiskObject(18)=1 THEN
    GOSUB LoadDrawer
    GOSUB LoadImage
    GOSUB LoadDefaultTool
    GOSUB LoadToolTypes
  END IF
  IF DiskObject(18)=2 OR DiskObject(18)=5 THEN
    GOSUB LoadDrawer
    GOSUB LoadImage
    GOSUB LoadToolTypes
  END IF
  IF DiskObject(18)=3 THEN
    GOSUB LoadImage
    GOSUB LoadToolTypes
  END IF
  IF DiskObject(18)=4 THEN
    GOSUB LoadImage
    GOSUB LoadDefaultTool
    GOSUB LoadToolTypes
  END IF
PRINT
PRINT "End of File!"
WHILE ende=0
  SLEEP
```

```
  IF lst=1 THEN GOTO LstBytes
WEND
END
KeyTest:
 IF INKEY$<>" " THEN RETURN
 WINDOW 2,"Eingabe",(0,0)-(631,53),6
Start:
 PRINT "Adresse:"Adr,Adresse(Adrnr,3)
 INPUT "Befehl: ",Befehl$
 ComKey$=LEFT$(Befehl$,1)
 ComTxt$=MID$(Befehl$,2)
 ComWrt#=VAL(ComTxt$)
 IF ComKey$="#" THEN
   FOR TestI=1 TO nr
      IF Adresse(TestI,1)=ComWrt# THEN Adr=ComWrt# : Adrnr=TestI
   NEXT TestI
   GOTO Start
 END IF
 IF ComKey$="e" THEN ende=1
 IF ComKey$="s" THEN
   IF LEN(ComTxt$)>0 THEN File$=ComTxt$
   OPEN ":mod. Icons/"+File$+".info" FOR OUTPUT AS 1
   PRINT#1,summary$
   CLOSE 1
   KILL ":mod. Icons/"+File$+".info.info"
   GOTO Start
 END IF
 IF ComKey$="a" THEN
   bytes$="" : Wert#=ComWrt#
   FOR KeyI=Adresse(Adrnr,2)-1 TO 1 STEP -1
      a=INT(Wert#/256^KeyI)
      Wert#=Wert#-a*256^KeyI
      bytes$=bytes$+CHR$(a)
   NEXT KeyI
   bytes$=bytes$+CHR$(Wert#)
   MID$(summary$,Adr,Adresse(Adrnr,2))=bytes$
   Adresse(Adrnr,3)=ComWrt#
   GOTO Start
 END IF
 IF ComKey$="l" THEN lst=1
 WINDOW CLOSE 2
RETURN
LoadHeader:
  RESTORE DiskObject
  po=1 : PRINT
  PRINT "struct DiskObject" : PRINT
  FOR I=1 TO 26
     GetBytes
DiskObject$(I,1),DiskObject$(I,2),DiskObject$(I,3),DiskObj6ect(I)
  NEXT I
RETURN
LoadDrawer:
  RESTORE DrawerData
```

```
  PRINT
  PRINT "struct DrawerData" : PRINT
  FOR I=1 TO 20
     GetBytes
DrawerData$(I,1),DrawerData$(I,2),DrawerData$(I,3),DrawerD6ata(I)
  NEXT I
RETURN
LoadImage:
  Im=1
  GOSUB GetImage
  IF DiskObject(12)<>0 THEN Im=2 : GOSUB GetImage
RETURN
GetImage:
  RESTORE Image
  PRINT
  PRINT "struct Image" : PRINT
  FOR I=1 TO 9
     GetBytes Image$(Im,I,1),Image$(Im,I,2),Image$(Im,I,3),Image(Im,I)
  NEXT I
  bytes=FNSize%(Im)
  PRINT
  PRINT "BitPlanes" : PRINT
  WIDTH 60
  FOR j=1 TO Image(Im,5)
     PRINT
     PRINT "Bitplane";j
     FOR I=1 TO bytes
        a$=HEX$(ASC(MID$(summary$,po,1)))
        IF LEN(a$)<2 THEN a$="0"+a$
        PRINT a$;
        IF I/2=INT(I/2) THEN PRINT " ";
        po=po+1
     NEXT I
     PRINT
  NEXT j
  WIDTH 75
RETURN
LoadDefaultTool:
  RESTORE DefaultTool
  PRINT
  PRINT "DefaultTool" : PRINT
  GetBytes DefaulTool$(1,1),DefaultTool$(1,2),DefaultTool$(1,3),
DefaultTool(1)
  IF DefaultTool(1)>80 THEN DefaultTool(1)=DefaultTool(1)/16
  GetString DefaultTool(1)/16
RETURN
LoadToolTypes:
  RESTORE ToolTypes
  PRINT
  PRINT "ToolTypes" : PRINT
  IF po>LEN(summary$) THEN RETURN
  GetBytes ToolTypes$(1,1),ToolTypes$(1,2),ToolTypes$(1,3),ToolTypes(1)
  FOR I=1 TO ToolTypes(1)/4-1
```

```
     RESTORE DefaultTool
     ToolTypes$(2,3)=""
     GetBytes
ToolTypes$(2,1),ToolTypes$(2,2),ToolTypes$(2,3),ToolTypes(62)
     IF ToolTypes(2)>80 THEN ToolTypes(2)=ToolTypes(2)/16
    GetString ToolTypes(2)
  NEXT I
RETURN
SUB GetString (length) STATIC
SHARED po,summary$
 ts=po : a=1
 IF length=0 THEN EXIT SUB

 WHILE a<>0
    a=ASC(MID$(summary$,po,1))
    a$=HEX$(a)
    IF LEN(a$)<2 THEN a$="0"+a$
    PRINT a$;" ";
    po=po+1
 WEND
 PRINT
 PRINT MID$(summary$,ts,po-ts-1)
END SUB
SUB Decimal (he$,Dec) STATIC
 Dec=0
 FOR I=1 TO LEN(he$)
    a=ASC(MID$(he$,LEN(he$)+1-I,1))-48
    IF a>9 THEN a=a-7
    Dec=Dec+16^(I-1)*a
 NEXT I
END SUB
SUB GetBytes (eng$,deu$,Wert$,Dec) STATIC
SHARED po,summary$,Adresse(),nr
     READ eng$,deu$,bytes
     PRINT eng$;TAB(20);deu$;TAB(47);
     a$=MID$(summary$,po,bytes)
     IF bytes=1 THEN Wert=ASC(a$)
     IF bytes=2 THEN Wert=CVI(a$)
     IF bytes=4 THEN
       Wert$=""
       FOR j=1 TO 4
          a=ASC(MID$(a$,j,1))
          h$=HEX$(a)
          IF LEN(h$)<2 THEN h$=h$+"0"
          Wert$=Wert$+h$
       NEXT j
     ELSE
        Wert$=HEX$(Wert)
     END IF
     PRINT "$";Wert$;TAB(57);
     Decimal Wert$,Dec
     PRINT Dec;TAB(71);po
     nr=nr+1
```

```
    Adresse(nr,1)=po : Adresse(nr,2)=bytes : Adresse(nr,3)=Dec
    po=po+bytes

END SUB
DiskObject:
DATA do_Magic,magische_Zahl,2
DATA do_Version,Versionsnummer,2
DATA do_Gadget,Klickstruktur,4
DATA gg_LeftEdge,Klickber._links,2
DATA gg_TopEdge,Klickber._oben,2
DATA gg_Width,Klickber._Breite,2
DATA gg_Height,Klickber._Höhe,2
DATA gg_Flags,InvertFlagge,2
DATA gg_Activation,$0003,2
DATA gg_Type,$0001,2
DATA gg_GadgetRender,Zeiger1_Bilddaten,4
DATA gg_SelectRender,Zeiger2_Bilddaten,4
DATA gg_IntuiText,"not used??",4
DATA gg_MutualExclude,"not useable!",4
DATA gg_SpecialInfo,"not useable!",4
DATA gg_GadgetID,"for own use!",2
DATA gg_UserData,"your Pointer!",4
DATA do_Type,Icontyp,1
DATA nothing,Fuellbyte,1
DATA do_DefaultTool,TextStruktur,4
DATA do_ToolTypes,TextStruktur,4
DATA do_CurrentX,aktuelle x-Position,4
DATA do_CurrentY,aktuelle y-Position,4
DATA do_DrawerData,WindowStruktur,4
DATA do_ToolWindow,Eigenes_Programmwindow,4
DATA do_StackSize,Speicherplatzreservierung,4
DrawerData:
DATA wi_LeftEdge,Linke_Ecke,2
DATA wi_TopEdge,Obere_Ecke,2
DATA wi_Width,Breite,2
DATA wi_Height,Hoehe,2
DATA wi_DetailPen,Zeichenfarbe_1,1
DATA wi_BlockPen,Zeichenfarbe_2,1
DATA wi_IDCMPFlags,GadGet_Flaggen,4
DATA wi_Flags,Window_Flaggen,4
DATA wi_FirstGadget,Gadget_Struktur,4
DATA wi_CheckMark,Abhaken,4
DATA wi_Title,Titeltext,4
DATA wi_Screen,Screenpointer,4
DATA wi_BitMap,Window_BitMap,4
DATA wi_MinWidth,min._Breite,2
DATA wi_MinHeight,min._Hoehe,2
DATA wi_MaxWidht,max._Breite,2
DATA wi_MaxHeight,max._Hoehe,2
DATA wi_Type,$0001,2
DATA actx-pos,aktuelle_x-Position,4
DATA acty-pos,aktuelle_y-Position,4
```

```
Image:
DATA im_LeftEdge,Linke_Ecke,2
DATA im_TopEdge,Obere_Ecke,2
DATA im_Width,Breite,2
DATA im_Height,Hoehe,2
DATA im_Depth,Tiefe,2
DATA im_ImageData,BitPlanes_Zeiger,4
DATA im_PlanePick,Bild_Daten,1
DATA im_PlaneOnOff,Verwendung,1
DATA im_NextImage,Nächtes_Bild,4
DefaultTool:
DATA char_num,Zeichenzahl,4
ToolTypes:
DATA string_num,Textanzahl,4
```

Programmbeschreibung

Das Programm besteht in den wesentlichen Teilen aus Unterprogrammen zur Auswertung der Icon-Daten, wie auch der Icon-Analyzer über die Betriebssystemroutinen. Allerdings gehen wir hier den Weg über BASIC, damit die Modifikation einfacher wird. Hinzugekommen ist die Angabe der Bytenummer hinter allen veränderbaren Parametern. Zusätzlich erscheint ein Fenster, wenn Sie die Space-Taste betätigen. In diesem Editor haben Sie mit einfachen Befehlen die Möglichkeit, das File datenweise zu verändern. Ihnen stehen folgende Befehle zur Verfügung:

 # num Geben Sie für "num" die Adresse ein, an der Sie Änderungen vornehmen möchten. Somit wählen Sie die Position, an der eigene Bytes eingesetzt werden sollen.

a zahl Setzt an die aktuelle Adresse den unter "zahl" angegebenen Wert ein. Je nachdem, welches Byteformat erwartet wird, wandelt die Routine die angegebene Zahl um.

s nam Speichert die Bytes des .info-Files im Verzeichnis ":mod. Icons" ab. Das Verzeichnis sollte vorher angelegt werden. Gibt man hinter s keinen Namen an, so wird der Name verwendet, der beim Laden angegeben wurde, ansonsten verwendet das Programm den neuen.

l Sind alle Strukturen aufgelistet und hat man einige Änderungen durchgeführt, so trägt man dem Programm mit diesem Befehl auf, die Listen ein weiteres Mal durchzugehen und alle Werte aufzuzeigen.

e Hiermit wird das Programm beendet. Der Befehl "e" muß eingegeben werden, da sich das Programm auch nach Ausgabe aller Strukturen in einer Warteschleife befindet. Wenn Sie den Editor verlassen wollen, so geben Sie einfach auf die Frage nach dem Befehl nichts ein und betätigen RETURN.

Ich möchte an dieser Stelle noch erwähnen, daß der Editor sicherlich nicht einer der komfortabelsten ist. Leider würde ein bequemerer Editor wesentlich mehr Platz einnehmen, der nicht unbedingt frei ist, und hiermit lassen sich auch alle Ergebnisse erzielen.

7.3.4 Farbliche Veränderung

Vielleicht wissen Sie schon, daß jedes Fenster, das geöffnet wird (oder auch die Workbench) in allen seinen Farben bestimmt werden kann. Nun ist man bei der Programmierung der Workbench etwas abgestumpft und verwendet dort nur die Farben Schwarz, Weiß, Orange und Blau, was zwar ein einheitliches Bild verursacht, aber nicht gerade interessant aussieht.

Um dem entgegenzuwirken, müssen wir vor dem Öffnen eines Fensters eingreifen und die Daten, besonders die der Farben, verändern. Wie Sie sicherlich schon erkannt haben, läuft das auf eine Veränderung der Drawer-Struktur des Icons hinaus, denn diese stimmt fast genau mit der eines normalen Fensters überein.

Sie können nun unter dem Punkt "Zeichenfarbe 1/2" beim Auflisten mit dem Icon-Editor jeweils den Wert $FF oder 255 erkennen. Ich hatte oben schon erwähnt, daß dieser für eine An-

passung an die Screen-Farben steht, was wir ja gerade ändern wollen. Schreiben Sie dafür einen Wert zwischen 0 und 3 in das entsprechende Byte.

Am besten ist es, Sie probieren einmal jede Möglichkeit aus. Wundern Sie sich aber nicht, wenn Sie die gespeicherten .info-Files öffnen wollen. Nachdem das Window für kurze Zeit erschienen ist, verschwindet es sofort wieder! Das liegt daran, daß mit dem Window kein Unterverzeichnis erstellt wurde, das aber für ein Schubladen-Icon ziemlich wichtig ist. Wechseln Sie deshalb in CLI, und erstellen Sie mit MAKEDIR für jedes .info-File auch ein Verzeichnis.

Danach können Sie sich alle neuen Fensterfarben genau ansehen. Wie Sie sicherlich gleich erkennen werden, sind manche Kombinationen nicht sehr hilfreich, da die Farben z.B. wichtige Texte unkenntlich machen können. Allerdings erhöhen manche Verbindungen sichtlich den Kontrast, und einige heben sich besonders gut von all den anderen ab. So kann man leicht optische Prioritäten setzen.

8. Kontrollierte Fehlerbehandlung

Für große Programme ist die "kontrollierte Fehlerbehandlung" unumgänglich. Doch zuerst müssen Sie natürlich wissen, wie man überhaupt Fehler "behandeln" kann, und noch wichtiger ist es zu wissen, an welcher Stelle im Programm die Fehler überhaupt auftreten. Das letztere kann man eigentlich gar nicht herausfinden, doch es gibt ein paar Regeln, die man beachten muß, damit ein Programm fehlerfrei abläuft.

8.1 Wann treten im Programm Fehler auf?

Eigentlich sollten im Programm gar keine Fehler auftreten, doch wie Sie sicher schon beim Arbeiten mit eigenen oder gekauften Programmen gemerkt haben, entsprechen einige Programme nicht unbedingt diesem Grundsatz. Damit wir uns allmählich an unser Problem "ranpirschen", wollen wir die Fehler in zwei Gruppen unterteilen.

Einmal sind dies die Fehler, die der Programmierer verursacht hat. Da gibt es z.B. Zeilen, in denen ein Syntax-Error auftritt, weil man vergessen hat, eine Klammer im Rechenausdruck zu schließen. Dieser Fehlertyp tritt oft auf, wenn man vor dem Benutzen noch mal ganz schnell eine Veränderung vornimmt. Der Flüchtigkeitsfehler wird meist nicht sofort entdeckt und tritt irgendwann bei der Arbeit auf. Um dies zu vermeiden, empfiehlt sich ein genaues Testen des Programms. Wie geht man dabei vor?

Schreiben Sie zuerst eine Liste der Programmteile, die benutzt werden müssen. Achten Sie besonders auf Programmzeilen, die nur unter bestimmten Bedingungen durchlaufen werden. Hier gilt besondere Aufmerksamkeit, denn ein dort versteckter Fehler kann nur gefunden werden, wenn Sie sorgfältig arbeiten oder den Zustand herstellen, unter dem die Zeilen abgearbeitet werden. Wenn Sie das Programm also testen, haken Sie jeden Teil

ab, den Sie durch Aufrufen abgearbeitet haben. Sind alle Programmteile "abgehakt", so können Sie davon ausgehen, daß keine Syntax-Errors vorliegen.

Doch es gibt noch mehr Fehlerquellen! Ein häufiger Fehler ist z.B. das Auftreten eines "Subscript out of Range"-Errors. Er fällt beim Arbeiten mit bis zu 10 Elementen des Feldes gar nicht auf, doch dann hat er verheerende Folgen! Um auch dieses auszuschließen, sollten Sie eine Liste der verwendeten Arrays erstellen und darauf achten, daß alle in der richtigen Größe definiert sind. Wenn Sie OPTION BASE verwenden, ist besondere Aufmerksamkeit vonnöten. Um die Kontrolle einfacher zu gestalten, empfiehlt es sich, die Dimensionierung am Anfang oder in einem bestimmten Teil des Programms vorzunehmen. Sie ersparen sich damit langes Suchen der Definitionszeilen.

Eine weitere Fehlerquelle sind die Rechenfehler. Bei fast jeder Berechnung in Ihrem Programm kann ein Fehler entstehen. Nehmen wir an, in Ihrem Programm kommen Funktionen vor - seien es nun selbst definierte oder von der Programmiersprache vorgegebene - dann kann es immer passieren, daß Sie durch irgendeinen mißlichen Umstand den Wertebereich nicht einhalten, und so wird ein Overflow- oder ähnlicher Error ausgegeben. Das Programm stürzt ab! Außerdem sollten Sie sich hüten, durch Null zu teilen. Deshalb empfiehlt es sich, vor jeder Division zu überprüfen, ob der Divisor ungleich Null ist, sonst erleben Sie böse Überaschungen! Allgemein gesagt, sollten Sie immer beim Dividieren, Potenzieren oder bei Funktionsaufrufen mißtrauisch sein und gegebenenfalls lieber die Werte über IF-Zeilen (in denen natürlich wieder ganz andere Fehler auftreten können!) überprüfen.

8.1.1 Fehler beim Diskettenzugriff

Wer kennt das nicht: Sie schreiben die perfekte Daten- und Adreßverwaltung, der Benutzer gibt den Namen seiner Datei ein, die geladen werden soll, - hat dabei aber noch die Programmdiskette im Laufwerk liegen - und beim Lesen: ein "File

Not Found Error"! Im Gegensatz zur Workbench, die durch einen Requester auffordert, die richtige Diskette einzulegen, meldet sich das AmigaBASIC mit einem Error.

Aber es kann Ihnen auch durchaus passieren, daß ein File mit dem angegebenen Namen existiert. Nur leider wurde es von einem anderen Programm erstellt und hat deswegen ein ganz anderes Format. Das beste, was Ihnen hier noch passieren kann, ist, daß die Datenwerte konfus eingelesen werden, das Programm aber weiterarbeitet. In den meisten Fällen gibt es aber einen "Type Mismatch"-Error oder ähnliches als Strafe.

Viel ärgerlicher ist es, wenn zwar die richtige Diskette eingelegt ist, das gesuchte File sich aber in einem ganz anderen Directory befindet. Auch hier tritt ein "File Not Found"-Error auf!

8.1.2 Fehler bei Benutzereingaben

In der von mir als Beispiel herangezogenen Datenverwaltung müssen sicherlich einmal Werte eingegeben werden. Doch für jeden Wert gibt es Einschränkungen! Zahlen sollen in einem bestimmten Bereich liegen oder keine Nachkommastellen aufweisen, Texte dürfen nur eine bestimmte Länge haben oder nur aus bestimmten Zeichen bestehen. All diese Bedingungen werden von dem normalen INPUT-Befehl nicht berücksichtigt. Er nimmt bei Zahlenangaben auch Text an, und das führt meist zu dem bekannten "Redo From Start", was manchmal den ganzen Bildschirm durch Scrolling durcheinanderbringt oder wichtige Texte überschreibt.

Aber auch die Möglichkeit, bei der Texteingabe nur bestimmte Zeichen zuzulassen, wird nicht unterstützt. Überschreitet der Anwender z.B. die Länge einer Texteingabe, so wird diese vielleicht vom Programm später abgeschnitten, doch man selber ist noch in dem Glauben, daß die Daten gesichert wurden. So können wichtige Informationen verlorengehen!

8.1.3 Fehler durch oder bei der Menüauswahl

Hierbei wird die Fehlersuche schon schwieriger. Bei den Benutzermenüs legen Sie als Programmierer sämtliche Unterprogramme und Funktionen fest. Der Programmbenutzer wählt aus diesen aus, und das Programm reagiert darauf. Aber gerade da liegt die Anfälligkeit der Menüs!

Es kann durchaus vorkommen, daß ein oder mehrere Punkte der Menüs gerade unter den herrschenden Umständen vollkommen unsinnig sind. Eine Ausführung dieser Funktionen wäre gleichzusetzen mit dem mutwilligen Provozieren eines Absturzes.

Ein harmloses Beispiel wäre die Abspeicherfunktion bei unserer fiktiven Datenbank, wenn man noch keine Daten eingegeben hat. Das hätte zwar keinen Absturz zur Folge, trotzdem hätten Sie dann unter Umständen einen leeren Datensatz in Ihrer Datei, den man meist nur schwer wieder löschen kann!

8.2 Von der Möglichkeit, einen Fehler aufzufangen

Sie haben vielleicht schon davon gehört, daß es die Möglichkeit geben soll, einem Fehler und somit einem Absturz nicht tatenlos zuzusehen, sondern diesen sogar beheben zu können! Gehen wir also daran, für die oben beschriebenen Probleme Lösungen zu finden, die nicht nur einen Fehler beseitigen, sondern ihn sogar verhindern! Denn man sollte nie nach der Beseitigung eines vorhandenen Problems trachten, die Devise lautet: vorbeugen.

Wie schon erwähnt, können Sie bei "einfachen" Fehlern wie z.B. einem "Division By Zero"-Error durch eine Abfrage diesen Fehler unterbinden. Denn wenn Sie selber durch das Programm den Fehler liefern, ist das wesentlich benutzerfreundlicher als eine Unterbrechung, wobei der Anwender vor dem neuen Problem steht, selbst programmieren zu müssen. In Ihrer Fehlerabfrage können Sie einen Text ausgeben lassen, der darauf hinweist. Entweder Sie geben die Möglichkeit vor, die Daten dafür

neu einzugeben, oder Sie springen schlimmstenfalls an den Anfang des Programms. Auf jeden Fall ist dadurch der Abbruch umgangen.

Bei dieser rüden Methode, einfach wieder von vorne anzufangen, können Sie aber auch die Error-Abfrage des BASIC verwenden. Mit ON ERROR GOTO springt das System immer an die angegebene Stelle, wenn ein Fehler auftritt. Es liegt dann an Ihnen, das Programm verzweigen zu lassen, damit die Ursachen genauer aufgeschlüsselt und vielleicht nach einer möglichen Behebung wieder hinter die fehlerhafte Stelle gesprungen werden kann.

Die Betriebssystem-Requester sind eine andere sehr komfortable Lösung der Fehlerbehebung. Sie kennen das ja: Einmal die falsche Diskette im Laufwerk, und schon erscheint das Window in der oberen Ecke. "Bitte richtige Diskette einlegen!", fordert der Computer Sie auf. Wenn Sie nicht gewillt sind und "Cancel" anwählen, "spuckt" er die entsprechende Fehlermeldung aus. Ein Requester (deutsch: Frager) ist also nur die letzte Chance, etwas gegen die Fehlermeldung zu unternehmen, andernfalls erfolgt sie doch. Aber damit wird dem nicht immer perfekten Menschen wenigstens die Möglichkeit zugestanden, sich einen Fehler zu erlauben. Manchmal - bei nur einem Diskettenlaufwerk - ist es gar nicht anders zu bewerkstelligen, als diese Prozedur über sich ergehen zu lassen.

Aber nicht nur der erneute Versuch, den Fehler zu beheben oder den fehlerhaften Teil zu umgehen, kann alleiniger Ausweg sein. Ich sagte vorhin, daß es am wichtigsten sei, einen Fehler gar nicht erst entstehen zu lassen, deshalb gibt es auch eine Möglichkeit, bei der das Programm es einfach nicht zuläßt, überhaupt einen Fehler zu erzeugen. So kann man beispielsweise eine Routine, die Fehler erzeugen könnte, nur unter Bedingungen anspringen lassen, die die Fehler nicht gestatten.

8.2.1 Freundliche Aufforderung an den Benutzer

Lassen Sie uns aber nach der vielen grauen Theorie endlich mal an die Programmierung gehen! Wenn ein Fehler auftritt, so ist der Anwender ziemlich verschreckt und ärgert sich natürlich über die Unterbrechung. Immerhin sind die meisten Anwender keine Programmierprofis oder sie kennen sich kaum in der Materie aus. Deshalb müssen wir behutsam mit unserem Benutzer umgehen. Er soll schonend auf seinen Fehler vorbereitet werden. Erst dann fordert das Programm ihn auf, das zu tun, was den Fehler korrigieren könnte.

Solch eine freundliche Aufforderung kennen Sie schon. Wenn Sie mit der Workbench arbeiten, erscheint bei einer Fehlbedienung des Diskettenlaufwerks (falsche Diskette) ein Requester. Dieser Requester fragt meist freundlich nach, ob Sie nicht die falsche Diskette eingelegt haben, und fordert Sie auf, jetzt bitte die richtige einzulegen.

Diese Art der Aufforderung können wir auch selbst programmieren! Sie werden sicherlich sagen: Gut, dann öffne ich ein Fenster, schreibe den Text hinein und frage die Maustasten ab. Aber das ist viel zu umständlich! Wir können einfach dem Betriebssystem mitteilen, daß wir auch einen Requester haben möchten, und dann macht es uns einen. Allerdings braucht es dazu Informationen, doch dazu komme ich später noch.

Bevor wir den Requester programmieren, muß erst geklärt sein, wann wir ihn überhaupt brauchen. Es bietet sich geradezu an, ihn bei den gleichen Anlässen zu benutzen, bei denen es auch die Workbench tut. Da wäre als erstes ein nicht gefundenes File zu nennen. Doch dabei gibt unser BASIC eine Fehlermeldung aus! Sie müssen also zuerst die Fehlermeldung verhindern und dann Ihren Requester in Aktion treten lassen, der noch einmal alles zum Guten wenden kann. Befassen wir uns erst mit dem Fehler, der auftritt, wenn das File nicht gefunden wird.

Da das Öffnen des Files immer gleich die Fehlermeldung "File Not Found" ausgibt, wenn das File nicht auf der Diskette oder

im Inhaltsverzeichnis ist, ist ein einfaches Lesen durch OPEN nicht möglich. Bei einer sequentiellen Datei gibt es aber einen Ausweg. Wir können das File mit der Option APPEND öffnen. Entweder das File existierte, dann wird ein Zeiger definiert, der das Anhängen von Daten erlaubt, oder das File existiert nicht, und die Datei wird neu geöffnet.

In jedem Fall gibt es aber nun die neue Datei. Durch die Funktion LOF (Length of File) können Sie nun abfragen, ob das File vorher existierte. Denn dann hat es mindestens die Länge von einem Zeichen (wenn nicht mehr), sonst aber ist die Länge gleich null. In diesem Fall sollten Sie das neu geöffnete File wieder löschen, weil es nicht gebraucht wird. Sehen Sie sich einmal das oben beschriebene Verfahren in einem Programm an:

```
' Test, ob sich ein File auf der Diskette
' befindet
'
' « by Wgb im August '87
'
FileName$=":AmigaBASIC2"
 Hauptprogramm:
 Again:

 PRINT "Gesuchtes File: ";
 WRITE FileName$

 CALL CheckFile (FileName$)
 IF exist=1 THEN
   PRINT "File existiert!"
   PRINT "File Header beginnt ab Block";blk&;"auf Disk."
 ELSE
  PRINT "File nicht gefunden!"
 END IF

END
' -------------------------------
SUB CheckFile (File$) STATIC
 SHARED exist
 OPEN File$ FOR APPEND AS 255
   exist=(LOF(255)>1)
 CLOSE 255
 IF exist=0 THEN KILL File$
END SUB
```

Um festzustellen, ob sich eine Datei auf der Diskette befindet, gibt es aber auch noch eine andere Methode! Sie ist gleichzeitig

eleganter und komplizierter. Weil sie aber Aufschluß über das Betriebssystem gibt, möchte ich sie Ihnen nicht vorenthalten. Auch hierfür können wir ein Unterprogramm anlegen, das einen entsprechenden Wert zurückliefert. Entweder 1, dann existiert das File, oder 0 für "nicht gefunden".

Die Funktion "Lock" selbst muß im Programm als Funktion definiert werden. Dann wird ihr die Speicheradresse des Namens übergeben, der mit einem Nullbyte endet. Zusätzlich braucht die Routine noch die Information, wie sie auf das File zugreifen soll. Da wir es mit einem Multitasking-Computer zu tun haben, können wir wählen, ob der Zugriff alleine uns vorbehalten ist (Acces Mode = Exclusive Write (-1)), oder ob Sie auch anderen das Lesen gestatten wollen (Acces Mode = Shared Acces (-2)). Die erste Möglichkeit ist dafür gedacht, daß beim Schreiben von Daten natürlich nur einer mit dem File arbeiten kann. Wogegen die zweite es erlaubt, daß die Daten von mehreren Programmen gleichzeitig gelesen werden können.

Unsere neue Routine braucht nur den Shared Access, deswegen übergeben wir den Wert -2. Der von der Funktion zurückgelieferte Wert ist gleich null, wenn sie kein File gefunden hat, andernfalls war etwas auf der Diskette. Den Wert müssen wir uns merken, denn nur mit ihm können wir den Zugriff auf die Daten wieder aufgeben.

Den Zugriff, den Sie sich mit dieser Routine verschafft haben, sollten Sie aber nach der Feststellung wieder aus der Liste streichen. Denn immerhin nimmt er Speicherplatz und Zeit in Anspruch. Sie erreichen dies mit der "UnLock"-Funktion. Ihr wird nur der Wert übergeben, den "Lock" zurückliefert.

```
' Test, ob sich ein File auf der Diskette
' befindet
'
' « by Wgb im August '87
'
DECLARE FUNCTION DosLock& LIBRARY
LIBRARY ":dos.library"
FileName$=":AmigaBASIC2"
 Hauptprogramm:
 Again:
```

```
 PRINT "Gesuchtes File: ";
 WRITE FileName$
 CALL CheckFile (FileName$)
 IF exist=1 THEN
   PRINT "File existiert!"
   PRINT "File Header beginnt ab Block";blk&;"auf Disk."
 ELSE
  PRINT "File nicht gefunden!"
 END IF
LIBRARY CLOSE
END
' -------------------------------
SUB CheckFile (File$) STATIC
 SHARED exist,blk&
 File$=File$+CHR$(0)
 accessRead%=-2
 lock&=DosLock&(SADD(File$),accessRead%)
 IF lock&=0 THEN
   exist=0
 ELSE
   exist=1
   blk&=PEEKL(lock&*4+4)
 END IF
 CALL DosUnLock(lock&)
END SUB
```

Nachdem nun geklärt ist, wie man feststellt, ob sich eine Datei auf der Diskette oder in dem aktuellen Inhaltsverzeichnis befindet, können wir zu der Besprechung übergehen, wie man in BASIC einen Requester schreibt. Ich hatte oben schon erwähnt, daß das Ganze natürlich auch über BASIC-Befehle möglich ist. Ich möchte dies allerdings nicht in Betracht ziehen. Sie werden mit etwas Ehrgeiz ein entsprechendes Unterprogramm schreiben können. Ich möchte Ihnen jetzt erklären, wie Sie die schon vorhandene Betriebssystemroutine verwenden. Dort steht ein Modul, das alle Aufgaben für einen Requester erledigt.

Es ist die sogenannte AutoRequest-Funktion. Ihr übergeben wir den Text, die Antwortmöglichkeiten, und schon erledigt sie den Rest. Auch hierfür schreiben wir ein Unterprogramm, das alle anfallenden Arbeiten erledigt. Dieses Unterprogramm gibt uns nach seinem Aufruf einen Wert zurück, von dem das Hauptprogramm Verzweigungen abhängig machen kann.

```
' Test, ob sich ein File auf der Diskette
' befindet
'
' « by Wgb im Juni '87
'
DECLARE FUNCTION AllocRemember& LIBRARY
DECLARE FUNCTION AutoRequest& LIBRARY
DECLARE FUNCTION DosLock& LIBRARY
LIBRARY ":intuition.library"
LIBRARY ":dos.library"
FileName$=":AmigaBASIC2"
 Hauptprogramm:
 Again:

 PRINT "Gesuchtes File: ";
 WRITE FileName$
 CheckFile FileName$
 IF exist=1 THEN
   PRINT "File existiert!"
   PRINT "File Header beginnt ab Block";blk&;"auf Disk."
 ELSE
  Request FileName$
  IF res&=1 THEN GOTO Again
  PRINT "File nicht gefunden!"
 END IF
LIBRARY CLOSE
END
' --------------------------------
SUB CheckFile (File$) STATIC
 SHARED exist,blk&
 TestFile$=File$+CHR$(0)
 accessRead%=-2
 lock&=DosLock&(SADD(TestFile$),accessRead%)
 IF lock&=0 THEN
   exist=0
 ELSE
   exist=1
   blk&=PEEKL(lock&*4+4)
 END IF
 CALL DosUnLock(lock&)
END SUB
' ----------------------------------
SUB Request (FileName$) STATIC
 SHARED add&,st$,res&,offs%
 Quest$(0)="Please insert volume containing"
 Quest$(1)="File "+FileName$
 Quest$(2)="It's not found!"
 yes$="Retry"
 no$="Cancel"
 bt%=2
 wid%=8*38
 hi%=8*9
 offs%=0
```

```
 opt&=2^0+2^16
 req&=AllocRemember&(0,400,opt&)
 IF req&=0 THEN ERROR 7
 add&=req&
 t1&=add&
 FOR loop2=0 TO bt%-1
   st$=Quest$(loop2)
   MakeHeader add&,st$,1,5,offs%+3
   offs%=offs%+8
 NEXT loop2
 st$=Quest$(bt%)
 MakeHeader add&,st$,0,5,offs%+3

 st$=yes$
 t2&=add&
 MakeHeader add&,st$,0,5,3

 st$=no$
 t3&=add&
 MakeHeader add&,st$,0,5,3
 res&=AutoRequest&(WINDOW(7),t1&,t2&,t3&,0,0,wid%,hi%)
 CALL FreeRemember(0,-1)
END SUB
SUB MakeHeader (ptr&,Text$,md%,le%,te%) STATIC
 SHARED add&
 Text$=Text$+CHR$(0)
 POKE ptr&,1
 POKE ptr&+1,0
 POKE ptr&+2,2
 POKEW ptr&+4,le%
 POKEW ptr&+6,te%
 POKEL ptr&+8,0
 POKEL ptr&+12,SADD(Text$)
 IF md%=0 THEN
   POKEL ptr&+16,0
 ELSE
   POKEL ptr&+16,ptr&+20
 END IF
 add&=ptr&+20
END SUB
```

Programmbeschreibung

Zuerst muß die Unterroutine "wissen", welche Texte sie ausgeben soll. Da gibt es einmal den allgemeinen Text, der auf die Sachlage hinweist, und außerdem folgen zwei weitere Texte, die anwählbar sind. Die letzten beiden Texte werden in einer Zeile ausgegeben und auch umrahmt. Daß man sie auch anklicken kann, ist wohl selbstverständlich. Unter Berücksichtigung der Texte, die Sie ausgeben lassen wollen, müssen Sie die Größe des

Fensters berechnen. Ist das Fenster nicht groß genug, so werden die Texte einfach "verschluckt" oder überschrieben, was nicht gerade gut aussieht.

Es muß nun für die Texte, die im Speicher in einer bestimmten "Struktur" abgelegt werden müssen, ein Speicherbereich reserviert werden. Dafür benutzen wir die Betriebssystemfunktion AllocRemember. Es besteht die Möglichkeit, den Speicherbereich nach bestimmten Kriterien zu wählen.

```
PUBLIC 20
CHIP 21
FAST 22
CLEAR 216
```

Die Speicherart, die wir benötigen, ist beliebig, nur sollte sie vorher gelöscht werden. Falls kein Speicher gefunden wurde, so wird vorsichtshalber eine Fehlermeldung ausgegeben. Nehmen wir nun an, daß genügend Speicher gefunden wurde. Dann können wir endlich die Texte in den reservierten Teil schreiben. Und zwar muß dies in einem Format geschehen, das leider nur für C-Programmierer einfach ist. In BASIC erfordert es ein wenig "POKErei". Ich möchte dieses Verfahren nicht vollständig erläutern. Wichtig für Sie ist nur die Benutzung und das Einbauen in eigene Programme.

Die erste Schleife bringt den Informationstext, der über den Anwählfeldern steht, in unseren Speicherbereich. Dann werden die beiden Antwortfelder im gleichen Format in unser RAM geschrieben. Ist alles gelaufen, so kann die AutoRequest-Funktion gestartet werden. Dafür übergeben wir ihr die Adresse des ersten Textes, des Textes der beiden Antwortfelder und die Maße unseres Requesters. Sie liefert einen Wert zurück, und zwar das Resultat 1, wenn das erste Feld angewählt wurde.

Mit diesem Wert können nun im Hauptprogramm Verzweigugen erfolgen. Entweder wird das Laden noch einmal versucht, weil z.B. der Benutzer inzwischen die Diskette gewechselt hat, oder er bricht das Laden ab, weil z.B. der Dateiname falsch eingegeben wurde, und verzweigt deshalb wieder ins Hauptprogramm.

8.2.2 Fehler vermeiden durch Abfrage während der Benutzereingabe

Nachdem Sie nun bei Diskettenfehlern einen Weg kennen, mit dem Sie den Programmabbruch verhindern können, wenden wir uns einem anderen Thema zu. Ihre Programme werden sicherlich nicht nur auf die Diskette zugreifen und ab und zu einen Requester ausgeben. Viel wichtiger sind die Eingaben vom Anwender. Aber gerade hier treten möglicherweise viele Fehler und Probleme auf.

Die einfachste und beste Lösung ist es nun, eine eigene Eingaberoutine zu schreiben, die die gewünschten Daten einliest, nur bestimme Zeichen annimmt und keine Fehler "ausspuckt". Damit die Routine auch "weiß", welche Zeichen sie zulassen soll und welche nicht, wird ihr beim Aufruf ein String übergeben, der alle erlaubten Zeichen enthält. Auch die Anzahl der einzugebenden Zeichen wird mitgeliefert. Den Rest erledigt dann das Unterprogramm. Erst wenn die Return-Taste gedrückt wird, wird das Unterprogramm verlassen.

Da die meisten Eingaben an einer bestimmten Position im Window gemacht werden sollen, ist vorgesehen, die Koordinaten anzugeben. So ersparen Sie sich den LOCATE-Befehl. Außerdem können Sie wie bei jedem INPUT-Kommando einen Text vorher ausgeben. Das alles erledigt die Eingaberoutine.

```
' Input Routine
'
' « by Wgb Mai '87
'
Hauptprogramm:
 DEFINT a-z
 KAlpha$="abcdefghijklmnopqrstuvwxyzßöäü"
 GAlpha$="ABCDEFGHIJKLMNOPQRSTUVWXYZßÖÄÜ"
 NAlpha$="01234567890+-*/.,="
 ZAlpha$=" ,.?!-/;:'"
 Moegl$=KAlpha$+GAlpha$+ZAlpha$
 GetInput "Nachname: ",NName$,Moegl$,10,10,20,0
 GetInput "Vorname : ",VName$,Moegl$,10,12,20,0
 WRITE NName$,VName$
 END
```

```
SUBRoutinen:
 SUB GetInput (Text$,In$,Moegl$,x,y,Zeichen,Pointer) STATIC
  Xold=POS(0)
  Yold=CSRLIN
  Laenge=0
  LOCATE y,x
  PRINT Text$;
  x=x+LEN(Text$)
  Lesen:
    Cursor x+Laenge,y
    GetInkey i$
    IF i$=CHR$(13) THEN GOTO Ende
    IF i$=CHR$(8) THEN GOTO RubOut
    IF Zeichen=Laenge THEN GOTO Lesen
    f=INSTR(Moegl$,i$)
    IF f=0 THEN
      BEEP
      GOTO Lesen
    END IF
    PRINT i$;
    In$=In$+i$ : Laenge=Laenge+1
  GOTO Lesen
  RubOut:
   IF Laenge=0 THEN GOTO Lesen
   Laenge=Laenge-1
   PRINT " ";
   In$=LEFT$(In$,Laenge)
   GOTO Lesen
  Ende:
   PRINT " ";
   LOCATE Yold,Xold
   IF Pointer AND 1 = 1 THEN
     l=LEN(In$)
     In$=In$+SPACE$(Zeichen-l)
   END IF
 END SUB
 SUB Cursor (x,y) STATIC
  COLOR 3
  LOCATE y,x
  PRINT "_";
  LOCATE y,x
  COLOR 1

 END SUB
 SUB GetInkey (Key$) STATIC
  KeyRead:
    Key$=INKEY$
  IF Key$="" THEN GOTO KeyRead
 END SUB
```

Programmbeschreibung

Sie sollten beim Gebrauch dieser neuen Eingaberoutine vor dem Hauptprogramm einige Strings definieren, in denen Gruppen von Zeichen liegen. Zum Beispiel einen String mit kleinen, einen anderen mit großen Buchstaben, den nächsten mit Zahlen und einen letzten mit Sonderzeichen. Mit diesen Strings können Sie später einfacher angeben, welche Zeichen zugelassen sind. Es bleibt natürlich Ihnen überlassen, ob Sie so verfahren. Es ist durchaus möglich, beim neuen INPUT-Befehl eine Konstante zu verwenden.

Dem GetInput-Befehl selbst werden der Text, die Variable, in der der Text abgelegt werden soll, dann eine Zeichenkette mit allen erlaubten Zeichen, die Position, die Anzahl der erlaubten Zeichen und als letztes ein Pointer übergeben. Dieser Pointer legt fest, ob der Eingabetext unter Umständen mit Leerzeichen aufgefüllt werden soll, wenn nicht genügend Zeichen eingegeben wurden. Dafür muß er auf 1 gesetzt werden. Es ist leider nicht möglich, Zahlen zu verarbeiten. Aber dies können Sie mit einer einfachen Kombination erreichen:

```
GetInput "Zahl: ",Zahl$,NumChar$,10,10,8
Zahl=VAL(Zahl$)
```

Wenn im String NumChar$ nur Ziffern waren, können Sie sicher sein, daß kein Nullwert in "Zahl" steht, solange es nicht gewünscht ist. Außerdem kann auf keinen Fall ein "Redo from Start"-Error auftreten!

Die SUB-Routine speichert am Anfang die momentane Cursorposition. Weil diese Position beim Verlassen wiederhergestellt wird, ist gewährleistet, daß keine Ausgaben beeinflußt werden. Dann wird der Text an der bestimmten Position ausgegeben und die Anfangsposition für die Eingabe berechnet. Die Länge des eingegebenen Textes ist noch auf Null gesetzt.

In der Leseschleife wird an der aktuellen Eingabeposition der Cursor ausgegeben, und die Routine wartet auf einen Tastendruck. Ein erhaltenes Zeichen wird auf Steuerfunktionen untersucht. Ist es z.B. die Backspace-Taste, so wird - wenn möglich -

das letzte eingegebene Zeichen entfernt. Ein Druck auf die Return-Taste wird auch erkannt, und sofort wird zum Ende der Routine gesprungen.

Ansonsten wird geprüft, ob überhaupt ein weiteres Zeichen erlaubt ist. Nur wenn dies der Fall ist, sieht die Routine im Vorgabestring nach, ob das Zeichen zu finden ist. Wenn nicht, so wird ein Warnsignal durch BEEP ausgegeben und an den Anfang der Leseschleife gesprungen. Ist es zugelassen, wird es ausgegeben und zum Eingabestring addiert. Anschließend wartet die Routine auf die Eingabe eines weiteren Zeichens.

Sie können diese Routine natürlich nach Belieben erweitern. Es ist zum Beispiel noch nicht möglich, mit dem Cursor im Text zu "wandern". Jetzt kann man nur das letzte Zeichen löschen und ggf. neue Zeichen anfügen. Es wäre aber durchaus interessant, Zeichen in der Mitte einzufügen oder zu löschen!

Auch können noch keine Eingaben vorgegeben werden. Dies wäre praktisch, wenn ein Wert sehr oft vorkommt und nur selten geändert wird. Diese Berücksichtigung müßten Sie dann in den Kopf der SUB-Routine einbauen, denn die Länge der Vorgabe muß festgestellt werden, und sie muß auch auf dem Bildschirm ausgegeben werden.

Bisher hatte man die Möglichkeit, eine Abfrage ohne Eingabe abzuschließen, indem man nur Return drückte. Sie könnten aber auch den Pointer so erweitern, daß z.B. wenn das 2. Bit gesetzt ist, nur mit mindestens einem eingegebenen Zeichen abgeschlossen werden kann. Alles das sind Funktionen, die man irgendwann gebrauchen könnte. Sie müssen abwägen, wann welche nötig ist. Ich wünsche Ihnen viel Spaß beim Erweitern.

8.3 Fehler beheben durch Korrektur falsch ausgewählter Funktionen

Dieser Teil soll sich mit der Korrektur beschäftigen. Vorher möchte ich aber grundsätzlich feststellen, daß die Korrektur

immer nur die letzte Möglichkeit ist, eine fehlerhafte Eingabe zu berichtigen. Manchmal läßt sich dieser Weg nicht umgehen, da in einigen Fällen eine Echtzeitüberprüfung in BASIC einfach zu langsam wäre. Auf die selbstgeschriebene Eingaberoutine bezogen, würde das bedeuten, daß nach jedem Zeichen vielleicht drei Sekunden gerechnet wird, um festzustellen, ob es erlaubt war, dieses Zeichen einzugeben. Damit wäre keinem geholfen.

Angenommen, es sei nur erlaubt, eins von hundert möglichen Wörtern einzugeben. Dann wäre es unklug, bei jedem neuen Zeichen zu überprüfen, ob dieses in der Kombination zugelassen ist. Erst wenn die Eingabe abgeschlossen wurde, wird mit allen Wörtern verglichen, und man kann, wenn nötig, eine Fehlermeldung ausgeben lassen und wieder zur Eingabe verzweigen.

Manchmal werden z.B. Einstellungen vorgenommen, bei denen man nicht sagen kann, ob sie zusammenpassen, wenn noch nicht alle Werte bekannt sind. Auch hier ist die Überprüfung nur am Schluß möglich. Stellt das Programm dann fest, daß ein Wert nicht hineinpaßt, so wird dieser korrigiert. Danach sollte das Programm nicht direkt fortfahren. Es muß den Benutzer wieder in die Korrektur einschalten, denn es kann sein, daß sein Wunsch vollkommen verfehlt wurde.

Sie sehen schon, daß wir es hier mit einem sehr komplexen Thema zu tun haben. Besonders schwierig wird die ganze Angelegenheit dadurch, daß man leider kein Patentrezept anbieten kann, nach dem die Benutzereingaben ohne Fehler abgewickelt werden könnten. Jedes Programm bietet andere Möglichkeiten und dadurch andere Fehlerquellen. Als Programmierer müssen Sie deshalb besonders darauf achten, daß alle Stellen, an denen Fehler auftreten könnten, gut gesichert sind. Das heißt, daß dort Überprüfungen stattfinden. Schlimmstenfalls sollte besser die Funktion abbrechen, als daß später das Programm abbricht.

8.3.1 Auswahl unmöglicher Kombinationen grafisch ausschließen

Hinter dieser pompösen Überschrift steht die ganz einfache Möglichkeit, bei PullDown-Menüs einige Punkte, die im Moment nicht ausgewählt werden sollen, in Schattenschrift darzustellen. Programmiertechnisch ist das doch ganz einfach, denn der MENU Befehl bietet doch alle Möglichkeiten, sagen Sie bestimmt. Da haben Sie zwar recht, doch kann man dies noch einfacher programmieren.

Wußten Sie z.B., daß bei einer Desaktivierung des Menünamens, sprich Überschrift, das ganze Menü nicht mehr aktiv ist? Damit können Sie sich bei langen Menüs viel Arbeit sparen, alle Punkte mit "MENU Nr.,Pkt.,0" zu desaktivieren. Aber es kann durchaus vorkommen, daß Sie zwar viele Punkte in einem Menü inaktiv machen müssen, aber eben nicht alle davon betroffen sind. Da hilft nun folgendes Programm!

Es ist eine SUB-Routine mit dem Namen "Able", die in einem bestimmten Menü ab dem angegebenen Punkt eine bestimmte Anzahl von Punkten mit der gewünschten Funktion versieht. Sie können also damit einen ganzen Block desaktivieren, aktivieren oder mit dem Haken versehen. Die Funktion ist praktisch ein Ersatz für den MENU-Befehl.

```
' Menu-Punkte streichen
'
' « by Wgb im Juni '87
'
DEFINT a-z
Hauptprogramm:
 GOSUB MenuDefinieren
 PRINT "Alle Menues aktiv!"
 Warten 5

 PRINT "Disk-Menue inaktiv."
 Able 1,0,0,0
 Warten 5
 PRINT "Zeichenart gesetzt."
 Able 2,4,0,2
 Warten 5
 PRINT "Nur einfarbig Zeichnen."
 Able 3,1,5,0
```

```
 Able 3,1,0,1
 Warten 5
 PRINT "Nur Pinsel aus der Grafik."
 PRINT "Sonst aber wieder alles."
 Able 4,1,4,0
 Able 1,0,0,1
 Able 3,1,5,1
 Warten 5
 PRINT "Programm kann nur noch beendet werden."
 Able 1,0,0,0
 Able 2,0,0,0
 Able 3,0,0,0
 Able 4,0,0,0
 Able 5,1,2,0
 WHILE INKEY$=""
  SLEEP
 WEND
 MENU RESET
END
MenuDefinieren:
 RESTORE MenuDaten
 READ Anzahl
 FOR i=1 TO Anzahl
    READ Punkte,Laenge
    FOR j=0 TO Punkte
       READ Punkt$
       IF j>0 THEN
         Punkt$=LEFT$(Punkt$+SPACE$(Laenge),Laenge)
         IF i=2 OR i=3 THEN
           Punkt$="  "+Punkt$
         END IF
       END IF
       MENU i,j,1,Punkt$
    NEXT j
 NEXT i
RETURN
SUB Able (MenuNr,Punkt,Anzahl,Art) STATIC
 FOR i=Punkt TO Punkt+Anzahl
    MENU MenuNr,i,Art
 NEXT i
END SUB
SUB Warten (Sekunden) STATIC
 Zeit&=TIMER+Sekunden
 WHILE TIMER<Zeit&
 WEND
 PRINT
END SUB
MenuDaten:
 DATA 5
 DATA 7,15,Disk
 DATA Neu,Laden,Laden unter
 DATA Speichern,Speichern unter
 DATA DiskKommando,Inhalt
```

```
DATA 7,9,Zeichnen
DATA Freihand,Linie,Linien
DATA Kreise,Rechtecke,Polygome
DATA Fuellen
DATA 6,10,Farben
DATA Einfarbig,Mehrfarbig,Palette
DATA Schatten,Wischen,Transparent
DATA 6,15,Pinsel
DATA Laden,Laden unter,Speichern
DATA Speichern unter, Löschen,Holen
DATA 4,11,Extras
DATA Workbench,Koordinaten
DATA Ausblenden,Ende
add&=ptr&+20
```

Programmbeschreibung

Zuerst werden alle Variablen als Integer definiert. Sie werden natürlich sofort einwenden: "Hier sind doch nur ganz wenige Variablen verwendet, was soll dann diese Deklaration?" Ich muß gestehen, daß es mir so in Fleisch und Blut übergegangen ist, diese Zeile zu Anfang zu schreiben, daß es manchmal vorkommen kann, daß sie eigentlich nicht so sinnvoll oder nötig ist. Aber beachten sie folgendes: Durch die Definition erhöht sich die Rechengeschwindigkeit, denn alle Operationen werden in echter Integer-Arithmetik durchgeführt.

Außerdem gibt es keine Probleme mehr beim Aufrufen der SUB-Routinen. Gibt man dort Konstanten an, die ganzzahlig sind, so wird der Fehler "Type Mismatch" ausgegeben, da das Unterprogramm Variablen vom Typ Real erwartet. Dann müssen Sie entweder hinter die Konstante in der Aufrufzeile ein Ausrufungszeichen setzen oder den Variablentyp im Unterprogramm anpassen. Nach der Variablendefinition springt das Hauptprogramm in das Unterprogramm "MenuDefinieren", in dem aus den DATA-Zeilen am Ende des Listings die Menütexte eingelesen werden.

Betrachten wir dazu das Unterprogramm selbst. Nachdem der Zeiger der DATA-Zeilen auf die Menüdaten gesetzt wurde, wird die Anzahl der zu definierenden Menüs gelesen. Entsprechend der Anzahl wird die äußerste Schleife durchlaufen. In

dieser Schleife wird dann aus den Daten gelesen, wie viele Menüpunkte jedes einzelne Menü hat und welche Längen für die Texte genommen werden sollen.

Der letzte Wert ist wichtig, denn nach der Definition eines Menüs kann man das entsprechende Feld öffnen. Es hat eine maximale x-Ausdehnung, die dem längsten Text angepaßt ist. Aktivieren kann man die einzelnen Punkte aber nur dort, wo auch wirklich die Buchstaben stehen. Wird jede Zeile, in der nicht die maximale Anzahl von Zeichen steht, mit Leerzeichen aufgefüllt, so kann man auch am Ende jeden Menüpunkt anwählen, da Leerzeichen dann auch aktiviert werden können.

Damit Sie sich ein Bild davon machen können, fahren Sie doch einmal die Menüpunkte nach dem Starten der jetzigen Version ab, und setzen Sie dann vor die Zeile, in der Punkt$ mit SPACE$ aufgefüllt wird, ein REM-Zeichen. Wenn Sie dann die Menüpunkte anwählen, werden Sie ganz sicher den Unterschied merken!

Als letztes müssen wir uns noch die innere Schleife des Unterprogramms ansehen. Dort werden entsprechend der vorher gelesenen Zahl die Menüpunkte aus den DATA-Zeilen geholt und mit der MENU-Funktion definiert. Da bei den Menüs 2 und 3 noch Haken vor den Punkten erscheinen können, werden in einer Abfrage zwei Leerzeichen vor die Texte dieser Menüs gesetzt. Allerdings wird dies genauso wie das Anhängen von Leerzeichen nur dann durchgeführt, wenn die Nummer des Menüpunktes größer null ist, denn die Überschrift des Menüs muß nicht korrigiert werden!

Kehren wir nun ins Hauptprogramm zurück. Dort wird der Hinweistext ausgegeben, daß alle Menüs aktiv sind. Damit der Benutzer dies auch überprüfen kann, wird in eine SUB-Routine verzweigt, die die angegebenen Sekunden wartet. Erst dann kehrt sie ins Hauptprogramm zurück.

Der Aufbau dieser Routine ist relativ einfach. Zuerst wird die Zeitzahl berechnet, die nach der angegebenen Zahl von Sekun-

den erreicht sein muß. Dann wird in einer Schleife so lange gewartet, wie die aktuelle Zeit noch vor der gesetzten ist. Fertig!

Im Hauptprogramm wird danach ein weiterer Text ausgegeben, der mitteilt, daß nun das Disk-Menü inaktiv sei. Dafür ist dann auch der Aufruf unserer wichtigen Unterroutine verantwortlich. Die Parameter sagen aus, daß im ersten Menü der nullte Punkt und null weitere auf Null gesetzt werden sollen. Im Klartext heißt das, daß die Überschrift des ersten Menüs inaktiv gemacht werden soll. Damit sind gleichzeitig alle anderen Punkte des Menüs inaktiv (s.o.).

Die SUB-Routine, die dafür zuständig ist, ist sehr einfach gestaltet, aber trotzdem arbeitserleichternd aufgebaut. Entsprechend der übergebenen Parameter wird die Schleife durchlaufen, die im angegebenen Menü alle Punkte auf die spezifizierte Art setzt. Das war alles!

Das Hauptprogramm wird dann weiter abgearbeitet. Es werden verschiedene Möglichkeiten der Nutzung dieses Unterprogramms gezeigt. Dafür ist es am einfachsten, wenn Sie das Programm starten und sich zu den Texten die entsprechenden Menüs ansehen.

9. Maschinensprache

AmigaBASIC ist zwar eine wunderbare Programmiersprache, jedoch nicht die schnellste. Sicher, das Programmieren in Assembler ist nicht unbedingt leichter, mit einem sehr guten Assembler ist es aber gerade auf dem Amiga unglaublich einfach, die tollsten Dinge in Maschinensprache zu programmieren. Der Grund hierfür liegt darin, daß zu allen BASIC-Befehlen eine entsprechende Betriebssystemroutine existiert, die dann allerdings mehr als tausendmal so schnell ausgeführt werden kann, als dies in BASIC der Fall ist. Wesentlich vereinfacht ist die Programmierung fast aller erdenklicher Features, die in BASIC nicht so ohne weiteres möglich sind, da für beinahe jeden Wunsch die richtige Systemroutine vorhanden ist.

Nun wird gerade heute ständig von der Programmiersprache C gesprochen, die es zumindest auf ein Zehntel der Geschwindigkeit von Assembler bringen kann. Leider muß hierzu bemerkt werden, daß viele Dinge mit C überhaupt nicht oder nur durch extrem zeitaufwendige Routinen möglich sind. Hinzu kommt, daß das Hintergrundwissen, das man für C besitzen muß, nicht weniger ist, als es in Assembler der Fall ist.

Ein erstklassiger Assembler wie "PROFIMAT Amiga", auf den DATA BECKER wirklich stolz sein kann, schlägt mit seinen vielen Möglichkeiten jeden C-Compiler um ein Vielfaches. Dabei sollte der Interessierte auch den Preisunterschied bedenken. Ich kaufte mir einen C-Compiler für tausend D-Mark. Nachdem ich für knapp hundert Mark den PROFIMAT erstand und mich von seiner Leistungsfähigkeit überzeugen konnte, hatte ich nichts Eiligeres zu tun, als den primitiven C-Compiler wieder zu verkaufen. Da man auf dem PROFIMAT Programme erstellen kann, die aufgrund ihrer MaKros und Sonderfunktionen kaum noch von BASIC-Programmen unterschieden werden können, mußte ich die Routinen der folgenden Kapitel in einfachem, auf jeden Assembler übertragbaren Code schreiben.

9.1 Super-Handler für Division-By-Zero

Eine leider allzuhäufige Ursache für eine Guru-Meditation ist die Division durch den Wert Null. Dazu sollten wir uns zunächst einmal ansehen, was in einem solchen Fall geschieht. Der Prozessor selbst ist es, der eine Exception auslöst, eine Ausnahmebehandlung also. Hierbei wird zunächst das Status-Register vom 68000 zwischengespeichert. Danach schaltet der Prozessor automatisch in den Supervisor-Modus um. Dabei wird das Trace-Bit gelöscht, um den Einzelschritt-Modus unmöglich zu machen. Jetzt wird der Programmzähler (PC) auf den Supervisor-Stack gelegt, danach folgt das Statusregister, das Instruction-Register in dem Zustand, in dem es beim Fehler war, und zu guter Letzt die Zugriffsadresse und das sogenannte Superstate-Word, in dem der Prozessorzustand erklärt ist.

Um nach einer Exception-Routine mit dem Programm weiterzuarbeiten, das den Fehler auslöste, genügt der Befehl RTE (ReTurn from Exception). Nun werden Fehler leider allzu häufig von einigen übereifrigen Programmierern einfach dadurch abgefangen, daß der jeweilige Exception-Vektor auf ein RTE zeigt. Eine ebenso unsinnige Variante finden Sie in Form des CLI-Befehls SetAlert, der nichts weiter tut, als auf das Setzen des Fehler-Flags zu warten und danach je nach SuperState-Word entscheidet, ob ein RTE erfolgt oder der böse Guru herbeigerufen wird.

Was ich hiermit klarmachen möchte, ist folgendes: Stellen Sie sich ein Programm vor, in dem der Nenner so klein geworden ist, daß er nicht mehr in einem Word dargestellt werden kann (Die 68000-Befehle DIVU und DIVS verarbeiten nur Words), der Nenner also Null ist. Hier tritt nun die Exception auf, da der 68000 nicht durch 0 dividieren kann. Stellen wir uns weiterhin vor, daß bei der Exception einfach mit RTE in unser Programm zurückgekehrt wird. Dieses Zurückkehren geschieht zu der Adresse hinter dem Befehl, der den Fehler auslöste. Wunderbar? Mitnichten! Was passiert denn normalerweise, wenn man durch einen sehr kleinen Wert teilt, beispielsweise durch 0,0001? Das Ergebnis wird sehr sehr hoch werden.

Dummerweise enthält unser Register, in das das Ergebnis hätte abgelegt werden müssen, nun aber einen sehr kleinen Wert. Die Folge ist, daß sämtliche nachfolgenden Berechnungen auch falsch sein müssen, womit bald die nächste Exception ausgelöst wird (im günstigsten Fall nur bei einer Division). Das Programm würde allerdings nur noch Blödsinn fabrizieren, womit dem Anwender keinesfalls geholfen sein dürfte. All diese Fehler wollen wir jedoch nicht machen. Nehmen wir an, der Nenner wäre nicht Null, sondern nur unendlich klein (Sie kennen diese mathematische Feinheit?). Damit ist die Division wieder zulässig. Für "unendlich klein" können wir schreiben:

$$\frac{1}{\text{Nenner} = \text{unendlich groß}}$$

Bei der Division durch diesen Nenner hilft uns die Regel: Dividiere durch einen Bruch, indem Du mit dem Inversen multiplizierst:

```
Zähler = x
Inverser Nenner = unendlich groß

Ergebnis = Zähler * Inverser Nenner
         =   x * unendlich groß
         = unendlich groß
```

Setzen wir jetzt für "unendlich groß" den größten möglichen Wert ein, den die Befehle DIVS und DIVU verarbeiten können, muß folglich in das Ergebnisregister eben dieser Wert - und die Rechnung ist trotz Exception richtig. Daher werden wir nun eine Exception-Routine schreiben, die den richtigen Wert in das entsprechende Register legt, bevor sie mit RTE zurückkehrt.

Zu diesem Zweck muß zunächst herausgefunden werden, welcher Befehl die Exception auslöste, denn der höchstmögliche Wert ist bei DIVU $FFFF und bei DIVS $7FFF. Weiterhin muß ermittelt werden, in welches Datenregister dieser Wert übergeben werden muß. Die Nummer des Datenregisters finden wir direkt im OpCode wieder, wie folgende Tabelle zeigt:

Befehl	Bits des Befehls-Codes
	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DIVU	1 0 0 0 x x x 0 1 1 y y y y y y
DIVS	1 0 0 0 x x x 1 1 1 y y y y y y

Die Bits 9-11 (x) geben die Nummer des Datenregisters an, in die das Ergebnis abgelegt wird. Die Bits 0-5 (y) geben die Adressierungsart an, die uns hier (Gott sei Dank) nicht zu interessieren braucht. Zunächst das Programm:

```
Zielcode Zeile Quelltext
          1    ;Division by Zero - Handler; by SM'88
          2
          3 Init_Trap:                ;Handler installieren
2C780004  4  Move.l 4,A6              ;ExecBase nach A6
203C      5  Move.l #Div_End-Div,D0   ;Handler-Länge nach D0
00000030
7201      6  MoveQ #1,D1              ;MEMF_Public
4EAEFF3A  7  Jsr -198(a6)             ;AllocMem aufrufen
23C0      8  Move.l D0,New_Trap       ;Adresse retten
00000036
6718      9  Beq.s Init_End          ;Ende, wenn Fehler
2240     10  Move.l D0,A1            ;Adresse nach A1
41F9     11  Lea Div,A0              ;CodeStart-Adresse
0000003A
702F     12  MoveQ #(Div_End-Div)-1,D0  ;CodeLänge-1
         13 Copy_Code:                 ;Handler kopieren
12D8     14  Move.b (A0)+,(A1)+       ;Byte kopieren
51C8FFFC 15  DBra D0,Copy_Code        ;Nächstes Byte
21F9     16  Move.l New_Trap,20       ;Neuer Trap-Vektor
00000036
0014
         17
         18 Init_End:                ;Adieu
7000     19  MoveQ #0,D0             ;Nenner = 0 !!!
82C0     20  Divu D0,D1              ;Go Ahead, Make My Day
4E75     21  rts                     ;Ende
         22
         23 New_Trap:                ;Neuer Trap-Vektor
00000000 24  dc.l 0                  ;1 LongWord
         25
         26 Div:                     ;DivisionByZeroHandler
48E7FFFE 27  MoveM.l D0-A6,-(sp)     ;Alle Register (s.u.)
206F003E 28  Move.l 62(sp),A0        ;PC vom Stack holen
3028FFFE 29  Move.w -2(A0),D0        ;Letzten Befehl holen
223C     30  Move.l #$ffff,D1        ;Größte Zahl für Divu
0000FFFF
08000008 31  Btst #8,D0              ;War es ein Divu-Befehl?
6706     32  Beq.s GoOn              ;Dann weitermachen
223C     33  Move.l #$7fff,D1        ;Sonst Divs-Wert
```

```
00007FFF
         34 GoOn:              ;Ermitteln des Datenregisters
EE48     35  Lsr.w #7,D0           ;Befehl bitweise scrollen
0280     36  AndI.l #28,D0         ;Register ausklammern
0000001C
2F810800 37  Move.l D1,0(sp,d0.l) ;Register auf Stack ändern
4CDF7FFF 38  MoveM.l (sp)+,D0-A6  ;Geänderte Register laden
4E73     39  Rte                  ;Return from Exception
         40 Div_End:              ;Label: SizeOf
         41
         42  End
```

Wenn Sie sich die ganze Zeit über gefragt haben, wie der neue Ergebniswert in das Datenregister gelangt, so finden Sie hier die Erklärung: Nach dem Ablegen sämtlicher Register auf dem Stack, bei dem immer zuerst das höchste Adreßregister abgelegt wird, dann die anderen Register in abfallender Reihenfolge, wird in Zeile 37 ganz einfach die mit 4 multiplizierte Registernummer als Offset für den Zugriff auf den Stack verwendet. Sollte im übrigen irgend etwas nicht geklappt haben, so folgt der Guru auf dem Fuße.

Wir sind nämlich einfach so frech und lösen nach der Installation des neuen Trap-Vektors eine Division-By-Zero-Exception aus (in Zeile 20 wird bewußt durch 0 dividiert!), daher am Rande Clint Eastwoods Lieblingsspruch... Damit auch die BASIC-Enthusiasten auf ihre Kosten kommen, hier eine Kurzroutine, die das Programm als CLI-Befehl ablegt (sollte ja auch in keiner Startup-Sequence fehlen):

```
OPEN "sys:c/DIVZERO" FOR OUTPUT AS 1
FOR i=1 TO  176
 READ a$
 a%=VAL("&H"+a$)
 PRINT #1,CHR$(a%);
NEXT
CLOSE 1
KILL "sys:c/DIVZERO.info"
datas:
DATA 0,0,3,F3,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
DATA 40,0,0,1B,0,0,3,E9,0,0,0,1B,2C,78,0,4,20,3C,0,0
DATA 0,30,72,1,4E,AE,FF,3A,23,C0,0,0,0,36,67,18,22,40,41,F9
DATA 0,0,0,3A,70,2F,12,D8,51,C8,FF,FC,21,F9,0,0,0,36,0,14
DATA 70,0,82,C0,4E,75,0,0,0,0,48,E7,FF,FE,20,6F,0,3E,30,28
```

```
DATA FF,FE,22,3C,0,0,FF,FF,8,0,0,8,67,6,22,3C,0,0,7F,FF
DATA EE,48,2,80,0,0,0,1C,2F,81,8,0,4C,DF,7F,FF,4E,73,0,0
DATA 0,0,3,EC,0,0,0,3,0,0,0,0,0,0,0,12,0,0,0,1C
DATA 0,0,0,2A,0,0,0,0,0,0,3,F2,0,0,3,F2
```

9.2 Achtung: Viren-Alarm!

Computerviren verbreiten sich mit rasender Geschwindigkeit. Die bekanntesten Vertreter dürften auf dem Amiga der SCA-Virus und der BYTE-BANDIT-Virus sein. Früher konnte man ganz einfach sagen: Verwende nur gekaufte Software, keine geklaute, und Du bleibst von Viren verschont. Dem ist heute leider nicht mehr so. Wir haben also ein Problem, das man nicht ernst genug nehmen kann.

Die erste Frage, die sich bei diesem Thema stellt, ist die, wer überhaupt auf die teuflische Idee kommen kann, anderen solche Virus-Programme "unterzujubeln". Die Namen der Viren sagen es aus: Sie kommen aus dem Bereich der "Cracker", die den lieben langen Tag nichts anderes zu tun haben, als ehrliche Programmierer um ihr wohlverdientes Geld zu bringen. Mit Raubkopien, mit denen zumindest fast jeder Jugendliche irgendwann einmal in Kontakt gekommen sein dürfte (traurig, traurig...) verbreiteten sich die Viren immer weiter. So kam es dann auch, wie es kommen mußte: Plötzlich wurden Originalspiele (!) mit Viren ausgeliefert! Nachdem die Spiele-Firma, die ich hier nicht nennen möchte, dieses mitgeteilt bekam, zog sie ihre Konsequenzen - die Viren sollen über Public-Domain-Programme ins Werk gekommen sein.

Wie vermehrt sich ein Virus? Im allgemeinen befindet sich das Killer-Programm im Bootblock einer Diskette. Wird von dieser gebootet, lädt das Betriebssystem zunächst die beiden ersten Sektoren einer Disk (Boot-Sektoren). Normalerweise befindet sich hier die Initialisierungsroutine für die DOS-Library. Diese Routine wird vom Befehl INSTALL auf die Disk geschrieben, um sie bootfähig zu machen. Die Routine wird direkt angesprungen, was genau der Umstand ist, den die Viren sich zunutze machen.

Sie kopieren sich in irgendeinen Speicherbereich, verbiegen System-Vektoren auf sich und führen danach die DOS-Initialisierung durch. Somit gelangen sie unbemerkt ins System. Legt man jetzt eine Boot-Disk in ein Laufwerk, schreibt sich der eine Virus bereits in den Bootblock, während der andere Typ dies erst beim Reset tut.

Leider starten inzwischen viele Programme über einen Loader im Bootblock, der von den Viren einfach überschrieben wird, was der Zerstörung der Disk gleichkommt. Zudem treten bei verseuchten Computern diverse, von den Viren geschaffene Störungen auf, die sich allerdings erst dann bemerkbar machen, wenn das Virus der Meinung ist, genug Disketten verseucht zu haben.

Der einzige Feind der Viren ist zur Zeit der INSTALL-Befehl, der den verseuchten Bootblock einfach wieder überschreibt. Heute bekannte Viren kennen jedoch ihren Feind - wird von INSTALL auf den Bootblock geschrieben, klinken sie sich in den Schreibvorgang mit ein und erneuern sich auf der Diskette. Einem derart verseuchten System ist mit normalen Mitteln nicht mehr zu helfen.

9.2.1 Der ultimative Virus-Killer

Das folgende Programm sollte in die Startup-Sequence jeder Boot-Disk gelegt werden. Es überprüft die Systemvektoren, die von den Viren im allgemeinen verwendet werden können, und dreht ihnen durch das Löschen dieser Vektoren "den Saft ab". War das System verseucht, erfolgt eine entsprechende Meldung. In diesem Falle müssen Sie sämtliche Bootdisks sofort mit dem INSTALL-Befehl reinigen. Durch das Abschalten der Virus-Programme im Speicher können diese sich dann nicht nach jedem INSTALL gleich wieder auf Disk zurückschreiben.

```
Zeile Quelltext
 1   ;VIRUS-KILLER V1.0; by SM'87
 2
 3  start:
 4   move.l 4,a6       ;EXECBASE nach a6
 5   moveq #0,d1       ;Flags: kein Virus vorhanden
 6   tst.l 46(a6)      ;Test, ob Cool-Capture verbogen ist
 7   beq.s noSCA       ;Der SCA-Virus war's nicht
 8   clr.l 46(a6)      ;Cool-Capture löschen
 9   addq.b #1,d1      ;Bit 0 setzen
10  noSCA:
11   cmpi.w #$FC,148(a6) ;Vertical Blank Interrupt normal?
12   beq.s noVBI       ;nein?
13   addq.b #2,d1      ;Bit 1 setzen
14   bra.s ClearTag    ;KickTag-Pointer löschen
15  noVBI:
16   tst.l 550(a6)     ;KickTag-Pointer verbogen?
17   beq.s GoOn        ;nein?
18   addq.b #4,d1      ;Bit 2 setzen
19  ClearTag:
20   clr.l 550(a6)     ;Pointer löschen
21   clr.l 554(a6)     ;Pointer löschen
22  GoOn:
23   move.b d1,Virusflag ;Flag retten
24   lea dosname,a1      ;Adresse des Libnamen
25   moveq #0,d0         ;Version ist egal
26   jsr -552(a6)        ;OpenLibrary
27   move.l d0,dosbase   ;Library-Basis retten
28   beq fehler          ;Verzweigen wenn Fehler
29   move.l d0,a6        ;DOSaufruf vorbereiten
30   jsr -60(a6)         ;Output-Handle holen
31   move.l d0,Outputhandle ;und retten
32   beq fehler          ;Verzweigen, wenn Fehler
33   move.l #title,d2    ;Überschrift nach d2
34   move.l #titleend-title,d3 ;Text-Länge
35   jsr writeout        ;Text ausgeben
36   tst.b Virusflag     ;Virusflag testen
37   bne.s Virusfound    ;Verzweigen bei Virus
38   move.l #clean,d2    ;Beruhigende Meldung
39   move.l #cleanend-clean,d3 ;Länge
40   jsr writeout        ;Text ausgeben
41   bra fehler          ;Programm beenden
42  Virusfound:          ;Virus war aktiv
43   btst #0,Virusflag   ;Cool-Capture?
44   beq.s notsca        ;Nee
45   move.l #scaV,d2     ;Meldung
46   move.l #scaVend-scaV,d3 ;Länge
47   jsr writeout        ;Text ausgeben
48  notsca:
49   btst #1,Virusflag   ;VB-Interrupt?
50   beq.s notvbi        ;Nee
51   move.l #bbVvbi,d2   ;Meldung
52   move.l #bbVvbiend-bbVvbi,d3 ;Länge
```

```
53   jsr writeout          ;ausgeben
54   bra.s bbfound         ;nächste Meldung
55  notvbi:
56   btst #2,Virusflag     ;Kicktag?
57   beq.s fehler          ;Nee
58   move.l #bbVtag,d2     ;Meldung
59   move.l #bbVtagend-bbVtag,d3 ;Länge
60   jsr writeout          ;ausgeben
61  bbfound:
62   move.l #bbv,d2        ;Meldung
63   move.l #bbvend-bbv,d3 ;Länge
64   jsr writeout          ;ausgeben
65  fehler:
66   move.l 4,a6
67   move.l dosbase,d0
68   beq.s quit
69   move.l d0,a1
70   jsr -414(a6)
71  quit:
72   moveq #0,d0
73   rts
74  writeout:
75   move.l outputhandle,d1
76   jmp -48(a6)
77  dosname:dc.b "dos.library",0
78  title:dc.b $c,$9b,"1;31;42m - Virus-Killer"
79        dc.b " V1.0 - ",10,13
80        dc.b "(w)  1988 by S. Maelger",10,13,10
81        dc.b $9b,"0;31;40m"
82  titleend:align
83  clean:dc.b "Keine Anzeichen für Virus"
84        dc.b "-Infektion gefunden !",10,13,10
85  cleanend:align
86  scaV:dc.b "Reset-Vektor Cool-Capture wurde"
87       dc.b " benutzt !",10,13
88       dc.b "SCA-Virus wird vermutet !",10,13
89       dc.b "Virus im Speicher vernichtet.",10,13,10
90  scaVend:align
91  bbVvbi:dc.b "Vertical Blank Interrupt wurde "
92         dc.b "benutzt !",10,13
93  bbVvbiend:align
94  bbV:dc.b "Byte-Bandit-Virus wird vermutet !",10,13
95      dc.b "Virus im Speicher vernichtet.",10,13,10
96  bbVend:align
97  bbVtag:dc.b "KickTagPointer wies nicht mehr auf"
98         dc.b " Betriebssystem !",10,13
99  bbVtagend:align
00  dosbase:dc.l 0
01  outputhandle:dc.l 0
02  Virusflag:dc.b 0
03   end
```

Wie schon zuvor nun der BASIC-Loader:

```
OPEN "sys:c/VIRUSCHECK" FOR OUTPUT AS 1
FOR i=1 TO  828
 READ a$
 a%=VAL("&H"+a$)
 PRINT #1,CHR$(a%);
NEXT
CLOSE 1
KILL "SYS:C/VIRUSCHECK.INFO"
datas:
DATA 0,0,3,F3,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,40,0,0,AB,0,0
DATA 3,E9,0,0,0,AB,2C,78,0,4,72,0,4A,AE,0,2E,67,6,42,AE,0
DATA 2E,52,1,C,6E,0,FC,0,94,67,4,54,1,60,8,4A,AE,2,26,67,A
DATA 58,1,42,AE,2,26,42,AE,2,2A,13,C1,0,0,2,A8,43,F9,0,0,1
DATA 12,70,0,4E,AE,FD,D8,23,C0,0,0,2,A0,67,0,0,AA,2C,40,4E
DATA AE,FF,C4,23,C0,0,0,2,A4,67,0,0,9A,24,3C,0,0,1,1E,26,3C
DATA 0,0,0,46,4E,B9,0,0,1,8,4A,39,0,0,2,A8,66,16,24,3C,0,0
DATA 1,64,26,3C,0,0,0,31,4E,B9,0,0,1,8,60,0,0,6A,8,39,0,0,0
DATA 0,2,A8,67,12,24,3C,0,0,1,96,26,3C,0,0,0,66,4E,B9,0,0,1
DATA 8,8,39,0,1,0,0,2,A8,67,14,24,3C,0,0,1,FC,26,3C,0,0,0
DATA 2A,4E,B9,0,0,1,8,60,1C,8,39,0,2,0,0,2,A8,67,24,24,3C,0
DATA 0,2,6A,26,3C,0,0,0,35,4E,B9,0,0,1,8,24,3C,0,0,2,26,26
DATA 3C,0,0,0,43,4E,B9,0,0,1,8,2C,78,0,4,20,39,0,0,2,A0,67
DATA 6,22,40,4E,AE,FE,62,70,0,4E,75,22,39,0,0,2,A4,4E,EE,FF
DATA D0,64,6F,73,2E,6C,69,62,72,61,72,79,0,C,9B,31,3B,33,31
DATA 3B,34,32,6D,20,2D,20,56,69,72,75,73,2D,4B,69,6C,6C,65
DATA 72,20,56,31,2E,30,20,2D,20,A,D,28,77,29,20,20,31,39,38
DATA 38,20,62,79,20,53,2E,20,4D,61,65,6C,67,65,72,A,D,A,9B
DATA 30,3B,33,31,3B,34,30,6D,4B,65,69,6E,65,20,41,6E,7A,65
DATA 69,63,68,65,6E,20,66,FC,72,20,56,69,72,75,73,2D,49,6E
DATA 66,65,6B,74,69,6F,6E,20,67,65,66,75,6E,64,65,6E,20,21
DATA A,D,A,0,52,65,73,65,74,2D,56,65,6B,74,6F,72,20,43,6F
DATA 6F,6C,2D,43,61,70,74,75,72,65,20,77,75,72,64,65,20,62
DATA 65,6E,75,74,7A,74,20,21,A,D,53,43,41,2D,56,69,72,75,73
DATA 20,77,69,72,64,20,76,65,72,6D,75,74,65,74,20,21,A,D,56
DATA 69,72,75,73,20,69,6D,20,53,70,65,69,63,68,65,72,20,76
DATA 65,72,6E,69,63,68,74,65,74,2E,A,D,A,56,65,72,74,69,63
DATA 61,6C,20,42,6C,61,6E,6B,20,49,6E,74,65,72,72,75,70,74
DATA 20,77,75,72,64,65,20,62,65,6E,75,74,7A,74,20,21,A,D,42
DATA 79,74,65,2D,42,61,6E,64,69,74,2D,56,69,72,75,73,20,77
DATA 69,72,64,20,76,65,72,6D,75,74,65,74,20,21,A,D,56,69,72
DATA 75,73,20,69,6D,20,53,70,65,69,63,68,65,72,20,76,65,72
DATA 6E,69,63,68,74,65,74,2E,A,D,A,0,4B,69,63,6B,54,61,67
DATA 50,6F,69,6E,74,65,72,20,77,69,65,73,20,6E,69,63,68,74
DATA 20,6D,65,68,72,20,61,75,66,20,42,65,74,72,69,65,62,73
DATA 73,79,73,74,65,6D,20,21,A,D,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,3,EC,0,0,0,16,0,0,0,0,0,0,0,30,0,0,0,36,0,0,0,42,0
DATA 0,0,52,0,0,0,5C,0,0,0,68,0,0,0,6E,0,0,0,76,0,0,0,82,0
DATA 0,0,8E,0,0,0,96,0,0,0,A2,0,0,0,AA,0,0,0,B2,0,0,0,BE,0
DATA 0,0,C8,0,0,0,D0,0,0,0,DC,0,0,0,E2,0,0,0,EE,0,0,0,F8,0
DATA 0,1,A,0,0,0,0,0,0,3,F2,0,0,3,F2
```

Na, war das 'ne Datenwüste?

9.3 ASSEMBLER und BASIC

Um in BASIC Assembler-Routinen aufzurufen, muß einer Long-Variablen die Anfangsadresse der Routine übergeben werden. Wir demonstrieren dies gemeinerweise an der Reset-Routine des Betriebssystems, die bei $FC0000 beginnt. Leider gibt es mit BASIC immer wieder Schwierigkeiten bei den Long-Variablen, die meist so aussehen, daß der BASIC-Interpreter fast ausschließlich mit Fließkommavariablen rechnet und eine Zahl erst später in Longs umwandelt.

Dabei tritt häufig der Fehler auf, daß die normalen Fließkommavariablen nur über ein paar Stellen genau sind, so daß beim Umrechnen in Longs das Ergebnis um Werte zwischen 1 und 5 zu niedrig ist. Wer dies umgehen möchte, muß entweder auf Maschinensprache-Routinen ausweichen, die fehlerfreie und superschnelle Long-Arithmetik garantieren, oder er geht den Weg über Strings:

```
SMreset&=CVL(CHR$(0)+CHR$(&HFC)+MKI$(0))
```

Wesentlich eleganter ist es, bei häufiger Verwendung von Systemroutinen und Assembler-Programmen vorher mit DEFLNG alle Variablen, die kein Kennzeichen tragen, als Long zu deklarieren:

```
DEFLNG a-z
SMreset=CVL(CHR$(0)+CHR$(&HFC)+MKI$(0))
SMreset
```

Vorsicht! Wenn Sie obiges Beispiel eingeben und starten, springt der BASIC-Interpreter die Reset-Routine an. Dies sollte nur als Beispiel für das Anspringen einer Assembler-Routine sein. Wollen Sie tatsächlich in einem BASIC-Programm einen Reset auslösen, gibt es eine viel bessere Methode, die sich zudem durch eine größere Geschwindigkeit auszeichnet:

```
POKEL 32,CVL(CHR$(0)+CHR$(&HFC)+MKI$(0))
```

Aber das nur am Rande, denn unser eigentliches Thema ist Assembler. Um den nächsten Schritt in Richtung BASIC-Erweite-

rungen einzuschlagen, schreiben wir eine Kurzroutine, die die Power-LED von hell auf dunkel umschaltet und umgekehrt. Jagen Sie Ihren Bekannten damit ruhig einen gehörigen Schreck ein:

```
Code             Mnemonic
0879000100BFE001 BCHG #1,$BFE001 ;Helligkeit umschalten
4E75             RTS             ;Das war's schon
```

Jetzt ergibt sich für uns die Frage, wo wir den Code unterbringen können. Das ist gar nicht so leicht, wie es den Anschein hat, denn die Adresse, an der der Code beginnt, muß in jedem Fall gerade sein, da sonst ein Guru-#3 auftritt (Adressierungsfehler). Der 68000-Prozessor kann nur Befehle an geraden Adressen abarbeiten, folglich muß sich auch jeder Befehl an einer solchen befinden. Dummerweise läßt BASIC hier nicht mit sich reden und legt seine Variablen byteweise dort in den Variablen-Puffer, wo gerade genug Platz vorhanden ist. Um daher absolut sicher zu gehen, daß die Adresse gerade ist, sollten Sie Speicherplatz vom System abzwacken. Das kann mit der Exec-Routine AllocMem geschehen. Wir schreiben folglich:

```
DEFLNG a-z
DECLARE FUNCTION AllocMem LIBRARY
LIBRARY"exec.library"
SMmagic=AllocMem(10,1) '10 Byte, Speicherbereich Public-RAM
FOR i%=0 to 4
 READ Power$
 POKEW SMmagic+i%*2,VAL("&H"+Power$)
NEXT
DATA 879,1,BF,E001,4E75
PRINT "Oh, siehe da, die Power-LED flackert !"
FOR i%=1 TO 20          '20 mal umschalten
 a!=TIMER+.5            'Verzögerung 0,5 Sekunden
 WHILE a!>TIMER:WEND    'Warten
 SMmagic                'Assembler-Routine aufrufen
NEXT
FreeMem SMmagic,10      'Speicherplatz wieder freigeben
LIBRARY CLOSE           'SYS schließen
```

Bei derart einfachen Routinen, die noch dazu kein einziges Register verwenden, haben wir noch keine Probleme. Diese treten vermehrt dann auf, wenn unser Programm bestimmte Zugriffe

durchführt. So akzeptiert jeder Assembler anstandslos folgendes Beispiel:

```
MOVE.L D0,Datenregister0
...
Datenregister0:dc.l 0
```

Hier wird ein Label im Programm definiert, an welches das Datenregister D0 abgelegt werden soll. Würden Sie die entsprechenden Codes wie bei dem Power-LED-Programm in den Speicher poken, führt es garantiert zum Herbeieilen des Guru. Der Grund hierfür ist die Adressierungsart, die bei obigem MOVE.L verwendet wird. Es wird nämlich absolut adressiert. Die absolute Adressierung kann aber nicht verwendet werden, weil wir nie im voraus wissen, wo das Programm schließlich liegt.

Der Code müßte erst an die jeweilige Anfangsadresse angeglichen werden, was wir uns natürlich ersparen wollen. Folglich muß bei einer Vorgehensweise wie im Power-LED-Programm PC-relative Adressierung erfolgen. Ein guter Assembler hat eine Einstellmöglichkeit für Programmzähler-relativen Code. In jedem Fall müßten Sie dann jedoch schreiben:

```
LEA Datenregister0(PC),A0
MOVE.L D0,(A0)
...
```

Ihr Programm könnte also nicht alle Assembler-Befehle nutzen, und die Programmierung würde dadurch sehr aufwendig werden. Es gibt zwar auch in den Listings, die in Amiga-Zeitschriften abgedruckt werden, lediglich einige PC-relativ adressierte Programme, doch wozu haben Sie sich Tips & Tricks gekauft? Ganz einfach: Ich werde Ihnen jetzt eine Möglichkeit zeigen, wie Sie erstens jeden Assembler-Befehl benutzen können, zweitens keinen Speicherplatz von Hand belegen müssen, drittens keine aufwendigen Laderoutinen benötigen und viertens sogar C-Programme aufrufen können (!):

9.3.1 Assembler- und C-Programme von BASIC aus nachladen und aufrufen

Um Sie nicht nur einfach mit der Tatsache zu konfrontieren, daß das möglich ist, sehen wir uns einmal an, wie das Ganze bei beliebigen (nicht BASIC-) Programmen ausschaut. Ob es sich um Ihr Lieblingsspiel, eine Textverarbeitung oder den BASIC-Interpreter handelt, immer sind diese Programme absolut adressiert. Denken wir etwas darüber nach, kommen wir zu dem Schluß, daß die Adressen, die irgendwo in den Programmcode weisen, vor dem Laden nur als Offsets vom Programmanfang aus vorhanden sein können. Während beziehungsweise kurz nach dem Laden muß das Betriebssystem die wirklichen Adressen berechnen und so das Programm geradezu selbständig umprogrammieren.

Um es vorwegzunehmen: Genau das ist auch der Fall. Damit das Betriebssystem, genauer gesagt eine Routine der DOS-Library, weiß, welche Befehle umcodiert werden müssen, wird von Ihrem Assembler mindestens ein sogenanntes Linkmodul mit abgespeichert, das diese Informationen enthält. Im Normalfall besteht ein abgespeichertes Assembler-Programm aus einem Code-Segment (Ihr Programm), einem Data-Segment und einem BSS-Segment (direkt beim Laden für das Programm zu reservierender freier Speicherplatz), sowie ein oder mehreren Link-Segmenten.

Was das Betriebssystem tut, müßten wir uns allerdings auch zunutze machen können. Selbständig gestartet werden soll unser Programm allerdings nicht, lediglich die Laderoutine wollen wir benutzen. Diese ist günstigerweise in der DOS-Library enthalten und kann problemlos von BASIC aus aufgerufen werden. Sie heißt LoadSeg und benötigt lediglich den mit einem Null-Byte abgeschlossenen Namen des Programms. Jede Speicherreservierung und alle anderen anfallenden Arbeiten werden von LoadSeg gleich mit erledigt. Die Syntax sieht so aus:

```
HauptSegment=LoadSeg ( SADD( "Filename" + CHR$(0) ) )
```

Um den reservierten Speicherplatz wieder freizugeben, benötigen wir die Routine UnloadSeg, der der Rückgabewert von LoadSeg übergeben werden muß:

```
UnloadSeg HauptSegment
```

Schwierigkeiten gibt es jedoch, wenn man sich den Wert ansieht, den wir von LoadSeg übergeben bekamen. Da das DOS bekanntlich in BCPL geschrieben wurde (eine Compiler-Sprache), handelt es sich auch um einen BCPL-Wert. Dieser Wert würde in der Sprache BCPL nun auf das Haupt- bzw. Code-Segment weisen. Für unsere Anwendungen müssen wir ihn jedoch in eine Adresse umformen. Das DOS tut so, als ob der Speicher nur aus hintereinandergelegten Longs bestehen würde. Folglich handelt es sich bei dem Rückgabewert um die Nummer des Longs vom Speicheranfang aus.

Um eine richtige Adresse zu erhalten, muß dieser Wert mit 4 (4 Bytes pro Long) multipliziert werden. Diese Adresse zeigt jedoch immer noch nicht genau auf unser Programm. Im ersten Long des Hauptsegmentes ist der BCPL-Zeiger auf das nächste Segment abgelegt, welches wieder mit einem BCPL-Zeiger auf ein weiteres Segment beginnt und so weiter. Hinter diesem Pointer steht bereits unser Programm, so daß wir folgendes schreiben müssen, um eine derart geladene Maschinenroutine aufzurufen:

```
DEFLNG a-z
DECLARE FUNCTION loadseg LIBRARY
LIBRARY"dos.library"
bcpl.segstart=loadseg(SADD("Profimat:asm.prg"+CHR$(0)))
Segment=bcpl.segstart*4
Routine=Segment+4
Routine
unloadseg bcpl.segstart
LIBRARY CLOSE
```

Um sich alle Startadressen der einzelnen Segmente ausgeben zu lassen, fügen Sie vor dem Aufruf von "Routine" ein:

```
PRINT "Haupt-Programm beginnt bei Adresse";Routine
i%=1
WHILE Segment<>0
```

```
    PRINT i%;". Segment beginnt bei Adresse";Segment+4
    Segment=PEEKL(Segment)*4
  WEND
```

Kommen wir zur Parameter-Übergabe. BASIC verfährt dabei mit Assembler-Routinen ebenso wie die Compilersprache C mit Programmteilen. Die Parameter werden in Form von Longs auf den Stack gelegt, wobei im folgenden Beispiel zuerst Parameter P3 auf dem Stack landet, danach P2 und zum Schluß P1. So ist der erste Parameter, der schließlich im aufgerufenen Programm vom Stack geholt wird, Parameter P1. Nach den Parametern wird die Routine mit JSR (Jump to Sub-Routine) direkt angesprungen, wodurch als letztes die Rücksprungadresse auf dem Stack landet.

```
Routine P1,P2,P3
```

Auf dem Stack liegt die Rücksprungadresse der Routine:

```
Stackpointer + 12 = P3 (Long)
Stackpointer +  8 = P2 (Long)
Stackpointer +  4 = P1 (Long)
Stackpointer +  0 = Rücksprungadresse (durch JSR)
```

Wer mit obigem Programm eine C-Routine nachgeladen hat, kann sie genauso aufrufen, wie dies bei einem Assembler-Programm der Fall ist. Wenn Sie können, sollten Sie jedoch immer auf Assembler zurückgreifen, da der von C generierte Maschinencode ganz erheblich langsamer ist als ein reines Maschinenprogramm. Das geht sogar so weit, daß Compiler-BASIC-Dialekte immer wesentlich schneller als C sind, da die Betriebssystem-Routinen genau so aufgerufen werden, wie man BASIC-Befehle schreibt, der Zugriff daher immer direkt erfolgt, und nicht vorher gewaltige Speicherkopierereien stattfinden müssen.

Teile des Betriebssystems (es werden immer weniger) sind in C programmiert, und gerade das ist es, was den Amiga so langsam macht. Weiterhin ist der Speicherplatzbedarf von C-Programmen sehr hoch. So sind nur wenige Prozente des Betriebssystems in C geschrieben, machen jedoch den größten Teil des vom Betriebssystem benötigten Speicherplatzes aus. Das erklärt auch, warum

das Betriebssystem trotz Beschleunigung und Verbesserung immer kleiner wird (C-Routinen werden mehr und mehr durch Assembler-Routinen ersetzt).

Im C-Programm stehen die Parameter in der Klammer in der gleichen Reihenfolge, wie sie von BASIC aus übergeben wurden. Wie kommt man nun von Assembler aus an sie heran? Dazu muß vorher noch erwähnt werden, daß Sie nicht so ohne weiteres die Register ändern dürfen. Diese sollten Sie gleich am Programmanfang irgendwo ablegen, wozu sich der Stack geradezu aufdrängt:

```
START: MOVEM.L D0-A6,-(A7)  ;Register retten: Damit befinden
                            ;sich 15 Register und die Rück-
                            ;sprungadresse auf dem Stack.
                            ;Der erste Parameter liegt somit
                            ;ab SP+(16*4) = 64(SP)
       MOVEM.L 64(SP),D0-D2 ;Die drei Parameter holen.
 ...
ENDE:  MOVEM.L (A7)+,D0-A6 ;Register vom Stack holen
       RTS
```

Fragt sich natürlich, wie die Rückgabe von Werten an das BASIC-Programm ablaufen muß. Zu diesem Zweck geben wir bei wenigen Rückgabewerten mit der VARPTR- oder SADD-Funktion die Adresse einer Variablen als Parameter an, holen im Maschinenprogramm die Adresse vom Stack und schreiben den Return-Wert hinein. Haben Sie viele Werte zurückzugeben, reicht eine Adresse einer Feldvariablen (Vorsicht bei Strings - dort erhält man höchstens die Adresse der Stringdeskriptoren, die jeweils aus 5 Bytes bestehen).

9.3.2 BASIC-Erweiterung ColorCycle

Um die Parameter-Übergabe an ein Maschinenprogramm zu verdeutlichen, hier eine Erweiterung, mit der Sie wie in DPaint die Farben rotieren können (sogar um ein Vielfaches schneller):

```
;----------------------------------------;
; BASIC-Erweiterung ColorCycle  SMmagic'88 ;
;----------------------------------------;
; Syntax: AdresseRoutine WINDOW(7),von,bis ;
; von=Startfarbe; bis=Endfarbe           ;
; Farbverschiebung immer  "von" --> "bis"  ;
; von < bis: Rotation nach oben          ;
; von > bis: Rotation nach unten         ;
;----------------------------------------;
Cycle:   MOVEM.L D0-A6,-(SP) ;Register auf Stack retten
         MOVE.L 4,A6         ;ExecBase nach A6 holen
         LEA GFXNAME,A1      ;Library-Name nach A1
         MOVEQ #0,D0         ;Version ist egal
         JSR -552(A6)        ;OpenLibrary aufrufen
         TST.L D0            ;Test, ob Basis vorhanden
         BEQ.S Exit          ;Wenn nicht, dann Ende
         MOVE.L D0,A6        ;GfxBase nach A6
         MOVE.L 64(SP),A0    ;WindowBase besorgen
         MOVE.L 46(A0),A0    ;ScreenBase ermitteln
         ADD.L #44,A0        ;ViewPort des Screens in A0
         MOVE.L 4(A0),A1     ;ColorTable nach A1
         MOVE.L 4(A1),A1     ;ColorMap ermitteln
         LEA CTab,A2         ;Unser Buffer nach A2
         MOVEQ #15,D0        ;15 Longs (32 Words)
CopyCT:  MOVE.L (A1)+,(A2)+  ;ColorMap kopieren
         DBRA D0,CopyCT      ;(wegen nicht geänderter F.
         MOVEM.L 68(SP),D0-D1 ;Start- und Endfarbe holen
         ANDI.W #31,D0       ;darf nicht höher als 31 sein
         ANDI.W #31,D1       ;dito
         LSL.B #1,D0         ;*2 (als Offset verwenden)
         LSL.B #1,D1         ;dito
         LEA CTab,A1         ;Adresse unseres Buffers
         MOVE.W (A1,D1.W),D2 ;letzte Farbe retten
         CMP.B D0,D1         ;Rotationsrichtung ermitteln
         BEQ.S ClLib         ;beide Farben gleich ???
         BGT.S Up            ;Farben nach oben rotieren?
Down:    MOVE.W 2(A1,D1.W),(A1,D1.W) ;Farbe nach unten
         ADDQ.B #2,D1        ;Offset erhöhen
         CMP.B D0,D1         ;Ende erreicht?
         BNE.S Down          ;nein? Dann nächste Farbe
         BRA.S SetLC         ;Schluß
Up:      MOVE.W -2(A1,D1.W),(A1,D1.W) ;Farbe von unten
         SUBQ.B #2,D1        ;Offset erniedrigen
         CMP.B D0,D1         ;unten angekommen?
         BNE.S Up            ;nein? Dann nächste Farbe
SetLC:   MOVE.W D2,(A1,D1.W) ;gerettete Farbe ablegen
         MOVEQ #32,D0        ;32 Farben setzen
         JSR -192(A6)        ;LoadRGB4 (a0=VP,a1=Ctab,d0)
ClLib:   MOVE.L A6,A1        ;GfxBase nach A1
         MOVE.L 4,A6         ;ExecBase holen
         JSR -414(A6)        ;Library schließen
```

```
Exit:    MOVEM.L (SP)+,D0-A6 ;Register vom Stack holen
         RTS                 ;Feierabend
GFXNAME: DC.B "graphics.library",0,0 ;Library-Name
CTab:    DS.W 32                     ;32 Words Buffer
         END                 ;Bis hierher assemblieren
```

Der Code wurde so gehalten, daß Sie entweder PC-relativ oder normal assemblieren können. Hier ein Demo-Programm in BASIC, das unsere Routine benutzt. Dabei wird von normaler Assemblierung ausgegangen (LoadSegment):

```
'Erst einmal Routine laden:
DEFLNG a-z
DECLARE FUNCTION loadseg LIBRARY
LIBRARY":bmaps/dos.library"
a=loadseg(SADD("profimat:ColorCycle"+CHR$(0)))
prg=a*4+4
'Ein wenig Grafik
FOR i%=0 TO 3
 LINE (0,i%*40)-STEP(80,40),i%,bf
NEXT
'Demo: Farben vorwärts und rückwärts rotieren
FOR i%=0 TO 50
 t!=TIMER+.2
 WHILE t!>TIMER
 WEND
 prg WINDOW(7),1,3
NEXT
FOR i%=0 TO 50
 t!=TIMER+.2
 WHILE t!>TIMER
 WEND
 prg WINDOW(7),3,1
NEXT
'Speicherplatz freigeben
unloadseg a
LIBRARY CLOSE
```

Hier nun der BASIC-Loader, der Ihnen dieses File auf Disk anlegt:

```
OPEN "COLORCYCLE" FOR OUTPUT AS 1
FOR i=1 TO  300
 READ a$
 a$="&H"+a$
 PRINT#1,CHR$(VAL(a$));
NEXT
CLOSE 1
DATA 0,0,3,F3,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,40,0,0,3A,0,0
```

```
DATA 3,E9,0,0,0,3A,48,E7,FF,FE,2C,78,0,4,43,F9,0,0,0,94,70,0
DATA 4E,AE,FD,D8,4A,80,67,76,2C,40,20,6F,0,40,20,68,0,2E,D1
DATA FC,0,0,0,2C,22,68,0,4,22,69,0,4,45,F9,0,0,0,A6,70,F,24
DATA D9,51,C8,FF,FC,4C,EF,0,3,0,44,2,40,0,1F,2,41,0,1F,E3,8
DATA E3,9,43,F9,0,0,0,A6,34,31,10,0,B2,0,67,26,6E,E,33,B1,10
DATA 2,10,0,54,1,B2,0,66,F4,60,C,33,B1,10,FE,10,0,55,1,B2,0
DATA 66,F4,33,82,10,0,70,20,4E,AE,FF,40,22,4E,2C,78,0,4,4E
DATA AE,FE,62,4C,DF,7F,FF,4E,75,67,72,61,70,68,69,63,73,2E
DATA 6C,69,62,72,61,72,79,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,EC
DATA 0,0,0,3,0,0,0,0,0,0,0,A,0,0,0,32,0,0,0,52,0,0,0,0,0,0,3
DATA F2,0,0,3,F2
DATA BECKER
```

9.3.3 BASIC-Erweiterung Zzz

Wenn der Amiga irgendeine Diskettenoperation durchführt, bei der er nicht gestört werden möchte, erscheint das berühmte Wölkchen mit den "Z". Auch professionelle Programme haben manchmal einen solchen veränderten Mauszeiger aufzuweisen, der mal mehr, mal weniger gut gelungen ist. Auch wir wollen in BASIC unsere Maus schlafen schicken, beispielsweise bei Diskettenoperationen oder beim Einlesen von Daten. Dazu hier eine Erweiterung, die uns die nötige Arbeit abnimmt:

```
;-------------------------------------------------;
;BASIC-Erweiterung Zzz                  SMmagic'88;
;-------------------------------------------------;
;Syntax: Zzz WINDOW(7),OnOff                      ;
;OnOff gerade oder 0:    Maus geht in Schlafstellung ;
;OnOff ungerade (1) :    Maus wieder normal       ;
;-------------------------------------------------;
ZZZ:    MOVEM.L D0-A6,-(SP) ;Register retten
        MOVE.L 4,A6         ;Basisadresse von Exec holen
        LEA INTNAME,A1      ;Adresse des Library-Namen
        MOVEQ #0,D0         ;Version ist egal
        JSR -552(A6)        ;OpenLibrary
        TST.L D0            ;Na, hat's geklappt?
        BEQ.S ENDE          ;Nicht? Dann Ende
        MOVE.L D0,A6        ;IntuitionBase laden
        MOVE.L 64(SP),A0    ;WindowBase holen
        MOVE.L 68(SP),D0    ;OnOff-Flag holen
        BTST #0,D0          ;Bit 0 testen
        BEQ.S SLEEP         ;Gerade Zahl? Gute Nacht
        JSR -60(A6)         ;ClearPointer aufrufen
        BRA.S EXIT          ;Und beenden
```

```
SLEEP:  LEA MAUS,A1          ;Adresse der Pointerdaten
        MOVEQ #22,D0         ;Höhe nach d0
        MOVEQ #16,D1         ;Breite nach d1
        MOVEQ #0,D2          ;xoffset löschen
        MOVE.L D2,D3         ;yoffset löschen
        JSR -270(A6)         ;SetPointer aufrufen
EXIT:   MOVE.L A6,A1         ;IntuitionBase nach a1
        MOVE.L 4,A6          ;ExecBase holen
        JSR -414(A6)         ;CloseLibrary aufrufen
ENDE:   MOVEM.L (SP)+,D0-A6 ;Register vom Stack holen
        RTS                  ;Zurück ins BASIC-Programm
INTNAME:DC.B "intuition.library",0 ;Library-Name
MAUS:   DC.L 0,$3000300,$7A007A0,$1FF01FF0,$3FF03FF0
        DC.L $30F83FF8,$3DFC3FFC,$7BFC7FFC,$30FE3FFE
        DC.L $3F863FFE,$1FEF1FFF,$3FDE3FFE,$1F861FFE
        DC.L $FFC0FFC,$3F803F8,$E000E0,$3800380,$7E007E0
        DC.L $3400340,0,$600060,$700070,$200020,0
        END                  ;Ende der Daten
Ein Aufruf könnte etwa so aussehen:
ZZZ WINDOW(7),0      'Wölkchen erscheint
ZZZ WINDOW(7),1      'Zeiger wieder normal
```

Dazu auch gleich der BASIC-Generator:

```
OPEN "ZZZ" FOR OUTPUT AS 1
FOR i=1 TO  260
 READ a$
 a$="&H"+a$
 PRINT#1,CHR$(VAL(a$));
NEXT
CLOSE 1
DATA 0,0,3,F3,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,40,0,0,31,0,0,3
DATA E9,0,0,0,31,48,E7,FF,FE,2C,78,0,4,43,F9,0,0,0,50,70,0,4E
DATA AE,FD,D8,4A,80,67,32,2C,40,20,6F,0,40,20,2F,0,44,8,0,0,0
DATA 67,6,4E,AE,FF,C4,60,12,43,F9,0,0,0,62,70,16,72,10,74,0
DATA 26,2,4E,AE,FE,F2,22,4E,2C,78,0,4,4E,AE,FE,62,4C,DF,7F,FF
DATA 4E,75,69,6E,74,75,69,74,69,6F,6E,2E,6C,69,62,72,61,72,79
DATA 0,0,0,0,0,3,0,3,0,7,A0,7,A0,1F,F0,1F,F0,3F,F0,3F,F0,30
DATA F8,3F,F8,3D,FC,3F,FC,7B,FC,7F,FC,30,FE,3F,FE,3F,86,3F,FE
DATA 1F,EF,1F,FF,3F,DE,3F,FE,1F,86,1F,FE,F,FC,F,FC,3,F8,3,F8
DATA 0,E0,0,E0,3,80,3,80,7,E0,7,E0,3,40,3,40,0,0,0,0,0,60,0
DATA 60,0,70,0,70,0,20,0,20,0,0,0,0,0,0,0,0,3,EC,0,0,0,2,0,0
DATA 0,0,0,0,0,A,0,0,0,30,0,0,0,0,0,0,3,F2,0,0,3,F2
DATA BECKER
```

Auch dieses File ist mit der Load-Segment-Routine zu laden.

9.3.4 Neue BASIC-Erweiterungen

Die Basic-Erweiterung (folgendes Listing) wird mit der DOS-Routine LoadSeg geladen:

```
DECLARE FUNCTION LOADSEG& LIBRARY
LIBRARY"dos.library"
SEGLIST&=LOADSEG&(SADD("BASICERW"+CHR$(0)))
IF SEGLIST&>0 THEN
  RESULT&      =SEGLIST&*4+4
  ALLOCBUFFER&=RESULT&+4
  FREEBUFFER& =RESULT&+8
  FREEALL&     =RESULT&+12
  BUBBLESORT& =RESULT&+16
  BLOAD&       =RESULT&+20
  BSAVE&       =RESULT&+24
ELSE
  ERROR 255
END IF
```

Entfernt wird sie mit:

```
UNLOADSEG SEGLIST&
```

Die ersten drei Funktionen bieten Ihnen eine komfortable Speicherverwaltung. Für die erste Funktion benötigen Sie die gleichen Parameter wie für die Betriebssystem-Routine AllocMem. Im Gegensatz zu dieser merkt sich unsere Funktion allerdings Adresse und Bereichslänge. Um beispielsweise 1 KByte Chip-Memory zu reservieren, schreiben Sie:

```
AllocBuffer& 1024&,65539&
Adresse&=PEEKL(Result&)
```

Sie können beliebig viele Speicherbereiche reservieren. Freigegeben werden diese durch den Aufruf von FreeAll:

```
FreeAll&
```

Nun sind manche Speicherreservierungen nur kurzzeitig nötig, der belegte Speicher soll nach der Nutzung wieder ans System zurückgegeben werden, um erneute Nutzung zu ermöglichen. In diesem Fall können Sie einen Bereich gezielt wieder freigeben:

```
FreeBuffer& Adresse&
```

Die vierte Routine sortiert Ihnen ein eindimensionales String-Feld bzw. einen Teil eines Stringfeldes. Dabei können Sie bestimmen, ab welchem Byte die Strings verglichen werden sollen und ob zwischen Groß- und Kleinschreibung unterschieden werden soll:

```
BubbleSort& VARPTR(a$(Anfang%)),Anzahl&,Offset&,0&
```

...sortiert Ihnen Anzahl& Feldelemente, beginnend mit dem Element a$(Anfang%), wobei die Strings ab dem Byte Offset& miteinander verglichen werden, und zwar ohne Unterschiede zwischen Groß- und Kleinbuchstaben zu machen.

Die nächste Funktion lädt eine Datei oder einen Teil dieser Datei an eine bestimmte Speicheradresse (s. Assembler-Listing):

```
BLOAD& SADD("Dateiname"+CHR$(0)),Adresse&,Lang&
```

Geben Sie für Lang& den Wert 0 an, wird die gesamte Datei geladen. Die Umkehrfunktion folgt gleich darauf:

```
BSAVE& SADD("Dateiname"+CHR$(0)),Adresse&,Lang&
```

Die angegebene Datei wird neu angelegt, etwa bereits vorhandene Dateien gleichen Namens werden vorher gelöscht.

```
;-------------------------------------------------------;
; Aus Result kann jederzeit das letzte Ergebnis gePEEKLt ;
; werden.                                               ;
;-------------------------------------------------------;

Result:              DC.L    0

Funktionen:          DC.L    AllocBuffer
                     DC.L    FreeBuffer
                     DC.L    FreeAll
                     DC.L    BubbleSort
                     DC.L    BLOAD
                     DC.L    BSAVE

;-------------------------------------------------------;
; Memory-Handling                                       ;
;-------------------------------------------------------;
```

```
;-------------------------------------------------------;
; Speicherplatz reservieren                             ;
;-------------------------------------------------------;
; Input:          Parm #1= Anzahl Bytes                 ;
;                 Parm #2= Speichertyp                  ;
; (Speichertypen:      1 = nicht verschiebbar           ;
;                      2 = Chip-Memory [untere 512 KByte] ;
;                      4 = Fast-Memory [nix Sound/Grafik] ;
;                  65536 = vorher löschen)              ;
;-------------------------------------------------------;
; Output:  Result = Speicheradresse (oder 0 bei Fehler) ;
;-------------------------------------------------------;

AllocBuffer:            MOVEM.L D0-A6,-(A7)
                        MOVEM.L 64(A7),D0/D1
                        MOVEA.L 4,A6
                        CLR.L   Result
                        ADDI.L  #12,D0
                        MOVE.L  D0,-(A7)
                        JSR     -198(A6)
                        MOVE.L  (A7)+,D1
                        TST.L   D0
                        BEQ.S   \out_of_memory
                        MOVEA.L D0,A6
                        MOVE.L  Memory_Liste,(A6)+
                        MOVE.L  #Memory_Liste,(A6)+
                        MOVE.L  D1,(A6)+
                        MOVE.L  A6,Result
\out_of_memory:         MOVEM.L (A7)+,D0-A6
                        RTS

;-------------------------------------------------------;
; Bestimmten Speicherbereich wieder freigeben           ;
;-------------------------------------------------------;
; Input:          Parm #1= Speicheradresse              ;
;-------------------------------------------------------;

FreeBuffer:             MOVEM.L D0-A6,-(A7)
                        MOVEA.L 64(A6),A1
                        LEA     -12(A1),A1
                        TST.L   (A1)
                        BEQ.S   \nothing_next
                        MOVEA.L (A1),A6
                        MOVE.L  4(A1),4(A6)
\nothing_next:          MOVEA.L 4(A1),A6
                        MOVE.L  (A1),(A6)
                        MOVE.L  8(A1),D0
                        MOVEA.L 4,A6
                        JSR     -210(A6)
                        MOVEM.L (A7)+,D0-A6
                        RTS
```

```
;-------------------------------------------------------;
; Alle reservierten Speicherbereiche freigeben          ;
;-------------------------------------------------------;

FreeAll:              MOVEM.L D0-A6,-(A7)
                      LEA     Memory_Liste,A1
                      MOVEA.L 4,A6
                      BRA.S   \check_list
\freemem_loop:        MOVEA.L (A1),A1
                      MOVE.L  A1,-(A7)
                      MOVE.L  8(A1),D0
                      JSR     -210(A6)
                      MOVEA.L (A7)+,A1
\check_list:          TST.L   (A1)
                      BNE.S   \freemem_loop
                      CLR.L   Memory_Liste
                      MOVEM.L (A7)+,D0-A6
                      RTS

;-------------------------------------------------------;
; Strings                                               ;
;-------------------------------------------------------;

;-------------------------------------------------------;
; Stringfeld sortieren                                  ;
;-------------------------------------------------------;
; Input:   Parm #1= Adresse des String-Descriptors des  ;
;                   ersten Feldelements [VARPTR(a$(x%))];
;          Parm #2= Anzahl der Elemente, die sortiert   ;
;                   werden sollen                       ;
;          Parm #3= Byte, ab dem die Strings verglichen ;
;                   werden sollen                       ;
;          Parm #4= Flag:   0 => UCASE,                 ;
;                         <>0 => unterscheide Groß- und  ;
;                                Kleinschreibung         ;
;-------------------------------------------------------;

BubbleSort:           MOVEM.L D0-A6,-(A7)
                      MOVEA.L 64(A7),A0
                      MOVEM.L 68(A7),D0-D2
                      SUBQ.W  #2,D0
                      BMI     \no_sort
                      SUBQ.W  #1,D1
                      BMI     \no_Sort
                      LEA     0,A1
                      MOVEA.L A1,A2
\sort_loop:           MOVEM.L D0/A0,-(A7)
                      MOVEQ   #0,D3
\string_loop:         BSR.S   \read_descriptor
                      CMP.W   D1,D4
                      BLT.S   \no_swap
                      MOVE.L  D4,D5
                      MOVEA.L A4,A5
```

```
                        ADDQ.L  #5,A0
                        BSR.S   \read_descriptor
                        CMP.W   D1,D4
                        BLT.S   \swap_descriptors
                        ADD.L   D1,A4
                        ADD.L   D1,A5
                        MOVE.L  D4,D6
                        CMP.W   D5,D6
                        BLE.S   \flag_check
                        MOVE.W  D5,D6
\flag_check:            TST.L   16(A7)
                        BEQ.S   \ucase_compare
                        BRA.S   \fast_compare
\fast_loop:             CMPM.B  (A4)+,(A5)+
                        BLT.S   \no_swap
                        BGT.S   \swap_descriptors
\fast_compare:          DBRA    D6,\fast_loop
                        BRA.S   \len_check
\ucase_loop:            EXG     D1,A1
                        EXG     D2,A2
                        MOVE.B  (A4)+,D2
                        BSR.S   \to_upper_case
                        MOVE.B  D2,D1
                        MOVE.B  (A5)+,D2
                        BSR.S   \to_upper_case
                        EXG     D1,A1
                        EXG     D2,A2
                        CMPA.W  A1,A2
                        BLT.S   \no_swap
                        BGT.S   \swap_descriptors
\ucase_compare:         DBRA    D6,\ucase_loop
\len_check:             CMP.W   D4,D5
                        BLE.S   \no_swap
\swap_descriptors:      MOVEQ   #4,D3
\swap_loop:             MOVE.B  (A0,D3.W),D6
                        MOVE.B  -5(A0,D3.W),(A0,D3.W)
                        MOVE.B  D6,-5(A0,D3.W)
                        DBRA    D3,\swap_loop
\no_swap:               DBRA    D0,\string_loop
                        MOVEM.L (A7)+,D0/A0
                        TST.B   D3
                        BEQ.S   \no_sort
                        DBRA    D0,\sort_loop
\no_sort:               MOVEM.L (A7)+,D0-A6
                        RTS
\read_descriptor:       CLR.L   -(A7)
                        CLR.L   -(A7)
                        MOVE.B  (A0),2(A7)
                        MOVE.B  1(A0),3(A7)
                        MOVE.B  2(A0),5(A7)
                        MOVE.B  3(A0),6(A7)
                        MOVE.B  4(A0),7(A7)
                        MOVEM.L (A7)+,D4/A4
```

```
                        RTS
\to_upper_case:         CMPI.W  #97,D2
                        BLT.S   \was_not_of_lcase
                        CMPI.W  #122,D2
                        BLE.S   \push_it_up
                        CMPI.W  #224,D2
                        BLT.S   \was_not_of_lcase
                        CMPI.B  #255,D2
                        BEQ.S   \was_not_of_lcase
\push_it_up:            SUBI.W  #32,D2
\was_not_of_lcase:      RTS


Memory_Liste:           DC.L    0

;-------------------------------------------------------;
; Datei an bestimmte Adresse laden                      ;
;-------------------------------------------------------;
; Input:        Parm #1=SADD(Dateiname$+CHR$(0))        ;
;               Parm #2=Adresse                         ;
;               Parm #3=Länge (0=ganze Datei)           ;
;-------------------------------------------------------;
; Output:       Result =Länge oder 0 (Fehler)           ;
;-------------------------------------------------------;

BLOAD:                  MOVEM.L D0-A6,-(A7)
                        MOVEA.L 4,A6
                        LEA     DOSname(PC),A1
                        MOVEQ   #0,D0
                        JSR     -552(A6)
                        TST.L   D0
                        BEQ.S   \noload
                        MOVEA.L D0,A6
                        MOVE.L  64(A7),D1
                        MOVE.L  #1005,D2
                        JSR     -30(A6)
                        MOVE.L  D0,Result
                        BEQ.S   \closeDOS
                        MOVE.L  D0,D1
                        MOVE.L  72(A7),D3
                        BNE.S   \loadit
                        MOVEQ   #0,D2
                        MOVEQ   #1,D3
                        JSR     -66(A6)
                        MOVE.L  Result,D1
                        MOVEQ   #0,D2
                        MOVEQ   #-1,D3
                        JSR     -66(A6)
                        MOVE.L  D0,D3
\loadit:                MOVE.L  Result,D1
                        MOVE.L  68(A7),D2
                        JSR     -42(A6)
                        MOVE.L  Result,D1
```

```
                        MOVE.L  D0,Result
                        JSR     -36(A6)
\closeDOS:              MOVEA.L A6,A1
                        MOVEA.L 4,A6
                        JSR     -414(A6)
\noload:                MOVEM.L (A7)+,D0-A6
                        RTS

;-------------------------------------------------------;
; Speicherbereich auf Disk ablegen                      ;
;-------------------------------------------------------;
; Input:        Parm #1=SADD(Dateiname$+CHR$(0))        ;
;               Parm #2=Adresse                         ;
;               Parm #3=Länge                           ;
;-------------------------------------------------------;
; Output:       Result =tatsächlich gespeicherte Bytes  ;
;-------------------------------------------------------;

BSAVE:                  MOVEM.L D0-A6,-(A7)
                        MOVEA.L 4,A6
                        LEA     DOSname(PC),A1
                        MOVEQ   #0,D0
                        JSR     -552(A6)
                        TST.L   D0
                        BEQ.S   \nosave
                        MOVEA.L D0,A6
                        MOVE.L  64(A7),D1
                        MOVE.L  #1006,D2
                        JSR     -30(A6)
                        MOVE.L  D0,Result
                        BEQ.S   \closeDOS
                        MOVE.L  D0,D1
                        MOVEM.L 68(A7),D2/D3
                        JSR     -48(A6)
                        MOVE.L  Result,D1
                        MOVE.L  D0,Result
                        JSR     -36(A6)
\closeDOS:              MOVEA.L A6,A1
                        MOVEA.L 4,A6
                        JSR     -414(A6)
\nosave:                MOVEM.L (A7)+,D0-A6
                        RTS

DOSname:                DC.B    "dos.library",0
```

Hier das BASIC-Listing, das Ihnen obige Programmdatei generiert:

```
OPEN "BASICerw" FOR OUTPUT AS 1
zeile%=0
checksum%=0
```

```
zloop:
  zeile%=zeile%+1
  wort%=0
  StrOut$=""
iloop:
    wort%=wort%+1
      READ a$
      IF a$<>"x" THEN
        wert%=VAL("&H"+a$)
        checksum%=(checksum% XOR wert%)
        checksum%=(checksum% XOR wort%)
        StrOut$=StrOut$+MKI$(wert%)
      ELSE
        wort%=10
      END IF
      IF wort%<10 GOTO iloop
      READ b$
      wert%=VAL("&H"+b$)
      IF wert%=checksum% THEN
        LOCATE 1,1
        PRINT "Zeile";zeile%;" - Checksumme okay"
        PRINT #1,StrOut$;
        IF a$<>"x" GOTO zloop
        CLOSE 1
        PRINT
        PRINT "Fertig."
        PRINT
        END
     ELSE
        PRINT
        PRINT "Checksummen-Fehler in DATA-Zeile";zeile%
        PRINT
        BEEP
        CLOSE 1
        KILL "BASICerw"
        END
      END IF

DATA 0000,03F3,0000,0000,0000,0001,0000,0000,0000,0000,03F9
DATA 0000,00A2,0000,03E9,0000,00A2,0000,0000,0000,001C,0007
DATA 0000,005E,0000,008E,0000,00BC,0000,01A6,0000,0222,03E4
DATA 48E7,FFFE,4CEF,0003,0040,2C78,0004,42B9,0000,0000,969F
DATA 0680,0000,000C,2F00,4EAE,FF3A,221F,4A80,6716,2C40,2D45
DATA 2CF9,0000,01A2,2CFC,0000,01A2,2CC1,23CE,0000,0000,2244
DATA 4CDF,7FFF,4E75,48E7,FFFE,226E,0040,43E9,FFF4,4A91,3CA1
DATA 6708,2C51,2D69,0004,0004,2C69,0004,2C91,2029,0008,7A47
DATA 2C78,0004,4EAE,FF2E,4CDF,7FFF,4E75,48E7,FFFE,43F9,6E05
DATA 0000,01A2,2C78,0004,600E,2251,2F09,2029,0008,4EAE,4009
DATA FF2E,225F,4A91,66EE,42B9,0000,01A2,4CDF,7FFF,4E75,8F42
DATA 48E7,FFFE,206F,0040,4CEF,0007,0044,5540,6B00,008A,6A19
DATA 5341,6B00,0084,43F8,0000,2449,48E7,8080,FFFE,226E,2091
DATA 0040,43E9,FFF4,4A91,6708,2C51,2D69,0004,0004,2C69,9C0F
DATA 0004,2C91,2029,0008,2C78,0004,4EAE,FF2E,4CDF,7FFF,3E6C
```

```
DATA 4E75,48E7,FFFE,43F9,0000,01A2,2C78,0004,600E,2251,EB73
DATA 2F09,2029,0008,4EAE,FF2E,225F,4A91,66EE,42B9,0000,1949
DATA 01A2,4CDF,7FFF,4E75,48E7,FFFE,206F,0040,4CEF,0007,BE6B
DATA 0044,5540,6B00,008A,5341,6B00,0084,43F8,0000,2449,DF9A
DATA 48E7,8080,42A7,1F50,0002,1F68,0001,0003,1F68,0002,4A03
DATA 0005,1F68,0003,0006,1F68,0004,0007,4CDF,1010,4E75,58B1
DATA 0C42,0061,6D16,0C42,007A,6F0C,0C42,00E0,6D0A,0C02,3711
DATA 00FF,6704,0442,0020,4E75,0000,0000,48E7,FFFE,2C78,8197
DATA 0004,43FA,00CC,7000,4EAE,FDD8,4A80,6760,2C40,222F,2257
DATA 0040,243C,0000,03ED,4EAE,FFE2,23C0,0000,0000,673E,F07F
DATA 2200,262F,0048,6618,7400,7601,4EAE,FFBE,2239,0000,0323
DATA 0000,7400,76FF,4EAE,FFBE,2600,2239,0000,0000,242F,90D1
DATA 0044,4EAE,FFD6,2239,0000,0000,23C0,0000,0000,4EAE,6EB1
DATA FFDC,224E,2C78,0004,4EAE,FE62,4CDF,7FFF,4E75,48E7,1A2A
DATA FFFE,2C78,0004,43FA,0050,7000,4EAE,FDD8,4A80,673E,64C1
DATA 2C40,222F,0040,243C,0000,03EE,4EAE,FFE2,23C0,0000,DFBB
DATA 0000,671C,2200,4CEF,000C,0044,4EAE,FFD0,2239,0000,454C
DATA 0000,23C0,0000,0000,4EAE,FFDC,224E,2C78,0004,4EAE,9769
DATA FE62,4CDF,7FFF,4E75,646F,732E,6C69,6272,6172,7900,157D
DATA 0000,03EC,0000,0014,0000,0000,0000,0004,0000,0008,1682
DATA 0000,000C,0000,0010,0000,0014,0000,0018,0000,002C,16B5
DATA 0000,0046,0000,004C,0000,0054,0000,0094,0000,00B2,16C6
DATA 0000,01CE,0000,01E6,0000,01F6,0000,0204,0000,020A,171D
DATA 0000,024A,0000,025E,0000,0264,0000,0000,0000,03F2,1694
DATA 0000,03F2,x,1565
```

10. I/O - Kommunikation mit der Außenwelt

Unter I/O versteht man normalerweise den Kontakt zwischen dem Amiga und seinen Peripheriegeräten. Zu diesen Geräten gehören zum Beispiel Drucker, Joysticks und Diskettenlaufwerke. Selbst das eingebaute Laufwerk wird wie ein externes Gerät angesprochen, denn dem Amiga ist es natürlich ganz gleich, wo sich diese Geräte befinden.

Für den fortgeschrittenen Anwender bleibt die Frage, wie seine Programme direkt mit diesen Geräten Verbindung aufnehmen können. Dazu gibt es im Amiga ein einheitliches I/O-System. Für jedes Gerät gibt es dazu ein Software-Modul, das für alle Geräte einheitliche Steuercodes in die gerätetypischen Codes übersetzt.

Diese Software-Module besitzen das einheitliche Suffix ".device". Sie befinden sich zum Teil im schreibgeschützten Kickstart-Speicher, zum Teil auf der Workbench-Diskette. Um I/O zu betreiben, ist es nötig, einen sogenannten I/O-Request-Block zu schaffen. Dabei handelt es sich praktisch um ein beliebiges Stück reservierten Speichers. Dieses Stück wird wie folgt definiert:

```
add& = Anfangsadresse des Speichers
```

add&+	Typ	Bedeutung
0	L	Zeiger auf vorangegangene Node
4	L	Zeiger auf folgende Node
8	L	Typ
9	B	Priorität
10	L	Zeiger auf Namensstring
14	L	Zeiger auf Message-Port
18	W	Länge der Message in Bytes
20	L	Zeiger auf Device-Block
24	L	Zeiger auf Unit-Block
28	W	I/O-Kommando
30	B	Flags
31	B	I/O-Error-Nummer

32	L	Actual-Feld
36	L	Länge-Feld
40	L	Data-Feld
44	L	Offset-Feld

B=Byte, W=Word, L=Longword

Neben dieser Struktur, auf die wir gleich zu sprechen kommen, muß ein sogenannter "Message-Port" eingerichtet werden. Dieser ist sozusagen ein Stück Speicher. Den I/O-Request-Block kann man sich als "Brief" vorstellen. Obwohl in einem Multitasking-System wie dem Amiga scheinbar viele Programme gleichzeitig ablaufen, kann der Prozessor in Wirklichkeit natürlich immer nur ein einziges bearbeiten.

Möchte nun ein Programm eine Nachricht zu einem anderen, "gleichzeitig" laufenden Programm senden, dann wird dies durch "Messages", sogenannte Nachrichten, realisiert.

Unser I/O-Request-Block ist eine solche Nachricht. Unser Programm, der BASIC-Interpreter AmigaBASIC, möchte schließlich eine Nachricht an ein I/O-Gerät senden, das als eigenständiges Programm parallel zu unserem Programm läuft. Dazu sendet es diesen Message-Block an die Adresse des anderen Tasks. In Wirklichkeit wird der Datenblock natürlich nicht im Speicher bewegt, sondern bleibt an seinem festen Platz. Statt dessen bekommt der fremde Task ausschließliche Kontrolle über diesen Speicher.

Sobald ein I/O-Request-Block an einen anderen Task verschickt ist, darf unser eigenes Programm also nicht mehr auf diesen Speicher zugreifen. Erst wenn der andere Task die Message verarbeitet hat, wird er die Kontrolle über diesen Speicher wieder an unser eigenes Programm zurückgeben.

Wir wollen uns aber nicht weiter mit den technischen Hintergründen aufhalten, denn eine genaue Beschreibung aller Abläufe würde den Umfang dieses Buches sprengen. Sollten Sie sich für dieses Spezialthema interessieren, empfehlen wir Ihnen weitergehende Literatur auf dem Gebiet der Amiga-Systemprogram-

mierung. Auf den folgenden Seiten wollen wir Ihnen anhand einiger Beispiele zeigen, wie Sie auch ohne große Programmierkenntnisse auf Diskettenlaufwerke und Drucker direkt zugreifen können.

10.1 Das Trackdisk-Device: Direkter Zugriff auf Disketten

Das "trackdisk.device" ist zuständig für ein jedes der bis zu vier 3½-Zoll-Diskettenlaufwerke. Mit seiner Hilfe können die Daten, die auf Diskette gespeichert sind, direkt eingesehen und manipuliert werden.

Jedes dieser Diskettenlaufwerke besitzt zwei Schreib-/Leseköpfe (für jede Seite einen). Die Diskette selbst wird pro Seite in 80 "Zylinder" eingeteilt. Jeder Zylinder wiederum besteht aus 11 Sektoren. Jeder Sektor beinhaltet 512 nutzbare Datenbytes sowie 16 Sektor-Verwaltungsbytes. Damit stehen insgesamt

```
     2 Köpfe *
    80 Zylinder *
    11 Sektoren *
   512 Bytes =
-------------------
901120 Bytes (880 KByte)
```

freier Datenspeicher sowie 28160 Bytes (28 KByte) für den Benutzer nicht zur Verfügung stehender Speicher zur Verfügung. Kommen wir zur Programmierung: Dazu stellt Ihnen das folgende Programm sechs High-Level-SUBs sowie vier Sub-Level-Routinen zur Verfügung. Normalerweise benötigen Sie aber nur die sechs ersten SUBs.

OpenDrive öffnet ein beliebiges Diskettenlaufwerk für Sie. Als Argument verlangt dieses SUB die Nummer des Laufwerks (0 = internes, 1, 2, 3 = externe(s) Laufwerk(e)), mit dem Sie arbeiten möchten. CreateBuffer ist eine Allzweckroutine zur Besorgung von Speicherplatz. Als Argument verlangt diese Routine die Variable, in der die Anfangsadresse des allokierten Speichers abgelegt werden soll, sowie die Größe des gewünschten Buffers in

Bytes. DiscardBuffer gibt den zuvor mit CreateBuffer reservierten Speicher wieder frei. Einziges Argument: die Anfangsadresse des Buffers. WorkDrive sendet einen I/O-Befehl an ein beliebigen, zuvor geöffneten Drive. CloseDrive schließt ein Diskettenlaufwerk nach Gebrauch. MotorOff schaltet den Motor des Diskettenlaufwerkes ab.

Mit dem folgenden Programm können Sie einen beliebigen Drive öffnen und anschließend jeden der 1760 Sektoren untersuchen. Die dort gefundenen Daten werden sowohl dezimal wie auch sedezimal (hex) dargestellt:

```
'################################
'#                              #
'# Programm: Disk - Monitor     #
'# Autor:    tob                #
'# Datum:    8. 8. 87           #
'# Version:  1.0                #
'#                              #
'################################

DECLARE FUNCTION OpenDevice% LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION AllocSignal% LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
DECLARE FUNCTION DoIO% LIBRARY

LIBRARY "exec.library"
LIBRARY "graphics.library"

var:        '* Variable
            DIM SHARED reg&(3,1)

main:       '* Demonstrationsprogramm
            PRINT TAB(20);"DISK MONITOR"
            PRINT
            LINE INPUT "Zugriff auf welches Drive (0 - 3)? ...."; dr$
            dr% = VAL(dr$)

            OpenDrive dr%
            CreateBuffer d0&, 512&

            LINE INPUT "Welchen Sektor (0 - 1759)? ....";sec$
            sec% = VAL(sec$)
            WorkDrive dr%, 2, sec%, d0&
            MotorOff  dr%

            WHILE sec$ <> "ende"
              CLS
```

```
            PRINT "Sektor ";sec%
            PRINT
            c% = 3
            FOR loop1% = 0 TO 512 - 1 STEP 25
              FOR loop2% = 0 TO 24
                check% = PEEK( d0& + loop1% + loop2%)
                h$     = HEX$(check%)
                IF LEN(h$) = 1 THEN
                  h$ = "0" + h$
                END IF
                he$  = he$ + h$
                IF check% < 31 THEN
                  d$ = d$ + "?"
                ELSE
                  d$ = d$ + CHR$(check%)
                END IF
                IF loop2% + loop1% = 512 - 1 THEN
                  loop2% = 24
                END IF
              NEXT loop2%
              LOCATE c%, 1
              c% = c% + 1
              out$ = he$ + "  " + d$
              CALL Text(WINDOW(8), SADD(out$), LEN(out$))
              he$ = ""
              d$  = ""
            NEXT loop1%
            LOCATE 1,20
            LINE INPUT "Welchen Sektor (0 - 1759, ende)?..";sec$
            sec% = VAL(sec$)
            WorkDrive dr%, 2, sec%, d0&
            MotorOff  dr%
          WEND

          DiscardBuffer d0&
          CloseDrive dr%
          CLS
          PRINT "All OK."

          LIBRARY CLOSE
          END

SUB OpenDrive (nr%) STATIC
    IF reg&(nr%, 0) = 0 THEN
      CreatePort "disk.io", 0, port&
      IF port& = 0 THEN ERROR 255
      CreateStdIO port&, io&
      dev$ = "trackdisk.device" + CHR$(0)
      er%  = OpenDevice% (SADD(dev$), nr%, io&, 0)
      IF er% <> 0 THEN
        RemoveStdIO io&
        RemovePort  port&
        io&   = 0
```

```
        port& = 0
        ERROR 255
      ELSE
        reg&(nr%, 0) = io&
        reg&(nr%, 1) = port&
      END IF
    ELSE
      io&    = reg&(nr%, 0)
      port&  = reg&(nr%, 1)
    END IF
END SUB

SUB CloseDrive (nr%) STATIC
    IF reg&(nr%, 0) <> 0 THEN
      io&   = reg&(nr%, 0)
      port& = reg&(nr%, 1)
      CALL CloseDevice(io&)
      RemoveStdIO io&
      RemovePort  port&
      reg&(nr%, 0) = 0
      reg&(nr%, 1) = 0
    END IF
END SUB

SUB MotorOff (nr%) STATIC
    io& = reg&(nr%, 0)
    IF io& <> 0 THEN
      POKEW io& + 28, 9
      POKEL io& + 36, 0
      e% = DoIO% (io&)
    ELSE
      BEEP
    END IF
END SUB

SUB CreateBuffer (add&, size&) STATIC
    IF size& > 0 THEN
      size& = size& + 4
      opt&  = 2^16
      add&  = AllocMem& (size&, opt&)
      IF add& <> 0 THEN
        add&  = add& + 4
        POKEL add& - 4, size&
      END IF
    ELSE
      BEEP
    END IF
END SUB

SUB DiscardBuffer (add&) STATIC
    IF add& <> 0 THEN
      size& = PEEKL (add& - 4)
      add&  = add& - 4
```

```
      CALL FreeMem (add&, size&)
    END IF
END SUB

SUB WorkDrive (nr%, command%, sector%, buffer&) STATIC
    td.sector% = 512
    io&        = reg&(nr%, 0)
    td.offset& = sector%*td.sector%
    IF io& <> 0 THEN
      POKEW io& + 28, command%
      POKEL io& + 36, td.sector%
      POKEL io& + 40, buffer&
      POKEL io& + 44, td.offset&
      er% = DoIO% (io&)
    ELSE
      BEEP
    END IF
END SUB

'--- sub level routines for advanced use only ---

SUB CreateStdIO (port&, result&) STATIC
    opt&    = 2^16
    result& = AllocMem&(62, opt&)
    IF result& = 0 THEN ERROR 7
    POKE  result& +  8, 5
    POKEL result& + 14, port&
    POKEW result& + 18, 42
END SUB

SUB RemoveStdIO (io&) STATIC
    IF io& <> 0 THEN
      CALL FreeMem(io&, 62)
    ELSE
      ERROR 255
    END IF
END SUB

SUB CreatePort (port$, pri%, result&) STATIC
    opt&  = 2^16
    byte& = 38 + LEN(port$)
    port& = AllocMem&(byte&, opt&)
    IF port& = 0 THEN ERROR 7
    POKEW port&, byte&
    port& = port& + 2
    sigBit% = AllocSignal%(-1)
    IF sigBit% = -1 THEN
      CALL FreeMem(port&,byte&)
      ERROR 7
    END IF
    sigTask& = FindTask&(0)
```

```
    POKE  port& + 8 , 4
    POKE  port& + 9 , pri%
    POKEL port& + 10, port& + 34
    POKE  port& + 15, sigBit%
    POKEL port& + 16, sigTask&
    POKEL port& + 20, port& + 24
    POKEL port& + 28, port& + 20
    FOR loop% = 1 TO LEN(port$)
      char%   = ASC(MID$(port$, loop%, 1))
      POKE port& + 33 + loop%, char%
    NEXT loop%
    CALL AddPort(port&)
    result& = port&
END SUB

SUB RemovePort (port&) STATIC
    byte&   = PEEKW(port& - 2)
    sigBit% = PEEK (port& + 15)
    CALL RemPort(port&)
    CALL FreeSignal(sigBit%)
    CALL FreeMem(port&-2, byte&)
END SUB
```

Variablen

reg&()	Dieses Feld beinhaltet wichtige interne I/O-Adressen, wie z.B. I/O-Request und -Port
dr%	Nummer des Diskettenlaufwerks (0 - 3)
d0&	512 Bytes großer Zwischenbuffer
sec%	Nummer des Sektors (0 - 1759)
loop1%	Schleife
loop2%	dito
check%	ausgelesenes Zeichen (dezimal)
h$	ausgelesenes Zeichen (sedezimal)
he$	ausgelesene Zeile (sedezimal)
d$	ausgelesene Zeile (dezimal)
c%	aktuelle Bildschirmzeile

OpenDrive()

nr%	Nummer des zu öffnenden Drives (0 - 3)
port&	Adresse des Message Ports
io&	Adresse des I/O-Blocks
dev$	"trackdisk.device", nullterminiert
er%	I/O-Fehler; 0 = kein Fehler

CreateBuffer()

size&	Größe des Buffers in Bytes
opt&	Optionen; 216 = CLEAR MEMORY
add&	Adresse des gefundenen Speichers

WorkDrive()

td.sector%	= 512; Bytes pro Sektor
io&	Adresse des I/O-Blocks
td.offset&	Byte-Offset von Sektor 0; Vielfaches von 512
er%	I/O Error Code

CreatePort()

port$	Name des neuen Ports
pri%	Priorität des neuen Ports (-128 bis 127)
result&	Adresse des gefundenen Ports (Output)
opt&	Speicheroption; 216 = CLEAR MEMORY
byte&	Größe des benötigten Speicherplatzes
sigBit%	Signalbit
sigTask&	Adresse auf AmigaBASIC-Task-Handler
char%	ASCII-Code des gerade gelesenen Zeichens

Programmbeschreibung

Zunächst wird festgelegt, mit welchem Diskettenlaufwerk der Benutzer arbeiten möchte. OpenDrive öffnet diesen Drive. Dabei wird intern zunächst überprüft, ob dieser Drive schon geöffnet ist, ob also in reg&() schon ein Eintrag vorliegt. Ist das nicht der Fall, dann wird mittels CreatePort ein Message-Port namens "disk.io" (der Name spielt keine Rolle) eingerichtet. Seine Anfangsadresse liegt in port&. Konnte kein Port geschaffen werden (port& = 0), dann kommt es zum Fehler, ansonsten wird ein I/O-Block geschaffen.

Diese Aufgabe übernimmt CreateStdIO. Ihr wird der bereits existierende Port (als Adresse) übergeben. In io& liegt anschließend die Anfangsadresse des I/O-Blocks. Nun wird der Drive mittels der Exec-Funktion OpenDevice%() geöffnet. Liefern diese Routinen einen Wert ungleich 0, dann konnte der Drive nicht geöffnet werden. Mögliche Gründe: Ein anderer

Task kontrolliert gerade der Drive, ein Open wurde noch nicht durch Close aufgehoben, der Drive existiert nicht oder ist nicht angeschlossen). In solch einem Fall werden Port und I/O-Block wieder aufgelöst, die Variablen werden auf Null gesetzt und es kommt zu einer Fehlermeldung. Ansonsten werden die Adressen des neuen Ports und des neuen I/O-Blocks in reg&() hinterlegt.

Anschließend wird ein Buffer geschaffen, der groß genug ist, die Daten eines Disk-Sektors aufzunehmen (Mindestgröße). Dieser 512 Bytes große Speicher wird mittels CreateBuffer reserviert, seine Anfangsadresse steht in d0& zur Verfügung. Nun wird der Benutzer gefragt, für welchen Sektor er sich interessiert (sec%). Das SUB WorkDrive liest diesen Sektor in den Buffer d0& (CMD READ, das Lesekommando, ist =2). Intern füllt dieses SUB den I/O-Requestblock mit den nötigen Werten und ruft dann die Exec-Funktion DoIO%() auf, die den Befehlsblock an den Diskdrive sendet.

Nachdem WorkDrive seine Arbeit erledigt hat, muß der Diskettenmotor ausgeschaltet werden. WorkDrive schaltet diesen Motor zwar selbständig ein, jedoch nicht wieder aus. Grund: Folgen mehrere Aufrufe des WorkDrive-Kommandos dicht aufeinander, dann wäre es töricht, jedesmal den Disk-Motor ein- und wieder auszuschalten. Das SUB MotorOff schaltet darum den Motor aus. Dazu wird das Motor-Kommando (= 9) in den I/O-Block geschrieben, der dann mittels DoIO%() abgesendet wird.

Jetzt liegen die Daten des gewünschten Sektors ab d0& im Speicher. Zwei Schleifen lesen die Werte aus dem Buffer und bringen sie in dezimaler und sedezimaler Darstellung auf den Bildschirm. Anschließend wird nach einem weiteren Sektor gefragt. Entweder gibt der Benutzer eine Zahl (0 - 1759) ein, oder aber das Wort "ende" für Abbruch. Im ersten Fall wird wie oben beschrieben wieder ein Sektor eingelesen, im zweiten Fall wird zunächst der Buffer freigegeben, dann der Diskdrive mittels CloseDrive geschlossen.

Dabei wird zunächst geprüft, ob der angegebene Drive überhaupt offen war (Eintrag in reg&()). Falls ja, werden die

Adressen der I/O- und des Port-Blocks ausgelesen. Mit Hilfe von RemoveStdIO und RemovePort werden diese Strukturen freigegeben, nachdem der Drive durch die Exec-Funktion CloseDevice() geschlossen wurde. Anschließend werden die Einträge in reg&() ausgelöscht.

10.1.1 Die I/O-Kommandos des Trackdisk-Devices

Wollen Sie eigene Programme entwerfen, sollten Sie natürlich alles aus dem WorkDrive-SUB herausholen, was es zu bieten hat. Insgesamt stehen Ihnen diese Befehle zur Verfügung:

a) Lesen von Daten

Kommando: 2
Befehlsaufruf: WorkDrive nummer%, 2, sector%, buffer&

Ist Ihr Buffer größer als 512 Bytes, dann können Sie natürlich auch mehr als einen Sektor gleichzeitig laden. Dazu muß innerhalb des SUBs der Eintrag in I/O-Feld 36 geändert werden: Statt td.sector% ein Vielfaches, z.B. 5 * td.sector%, wenn Ihr Buffer soviel Daten fassen kann.

b) Schreiben von Daten

Kommando: 3
Befehlsaufruf: WorkDrive nummer%, 3, sector%, buffer&

Schreibt den Inhalt des Buffers in den angegebenen Sektor auf Disk.

Achtung: Wenn Sie nicht genau über interne Disk-Organisation Bescheid wissen, können Sie mit diesem Befehl leicht ganze Disketten zerstören.

Wollen Sie nur einige Daten eines Sektors ändern, dann lesen Sie zunächst den Sektor in den Buffer (Kommando 2), verändern den Buffer Ihren Wünschen gemäß und schreiben ihn dann zurück. Auch hier läßt sich mehr als ein Sektor gleichzeitig ausschreiben (siehe: Schreiben von Daten).

c) Motor ein- und ausschalten

Kommando: 9
Befehlsaufruf: WorkDrive nummer%, 9, 0, 0

Das I/O-Feld 36 muß dabei manipuliert werden: 0 = Motor aus, 1 = Motor ein. IO_Actual liefert den vorherigen Zustand.

d) Diskette formatieren

Kommando: 11
Befehlsaufruf: WorkDrive nummer%, 11, track%, trackbuf&

Dieser Befehl schreibt immer nur einen gesamten Track auf Diskette. Ein Track besteht aus 11 Sektoren. track% muß demnach ein Vielfaches von 11 sein. Der Trackbuffer muß sein für 11 Sektoren groß genug. Der Befehl ignoriert alle zuvor auf diesem Track gespeicherten Daten und kann sogar Harderrors überschreiben.

10.1.2 Simultanes Arbeiten mit mehreren Laufwerken

Auch das ist gar kein Problem. Die SUBs des vorangegangenen Programms sind so konzipiert, daß bis zu vier Diskettenlaufwerke gleichzeitig angesprochen werden können. Dazu muß lediglich jeder Drive mit Hilfe des OpenDrive-Befehls geöffnet und natürlich auch einzeln wieder geschlossen werden. Außerdem muß für jedes Laufwerk ein eigener Buffer eingerichtet werden, es sei denn, Sie möchten Daten kopieren. Dann kann natürlich ein einziger Buffer verwendet werden.

10.1.3 Der Aufbau der Sektoren

Sicherlich geben die bloßen Daten eines Sektors nur wenig Aufschluß über den wahren Inhalt einer Diskette. Deshalb hier der Aufbau der Sektoren (Die Zahlen links beziehen sich auf Longwords, stehen also für Vier-Byte-Felder):

Root-Block (Sektor 880)

0	Typ (= 2)
1	0
2	0
3	Hashtable-Größe (512 - 224)
4	0
5	Checksumme
6 bis 77	Hashtable; Hier liegen die Sektornummern der im Haupt-Directory liegenden Dateien oder Unter-Directories
78	= FFFFFFFF (-1), wenn Bitmap gültig
79 bis 104	Hier liegen die Sektornummern der Sektoren, die die Bitmap enthalten (normalerweise ein Sektor). Jedes Bit der Bitmap entspricht einem Sektor der Diskette und zeigt an, ob der Sektor frei ist (Bit gesetzt) oder bereits belegt (Bit gelöscht).
105	Tag des Datums, an dem die Diskette zum letzten Mal verändert wurde
106	Minuten
107	Ticks (1/50 Sekunden)
108 bis 120	Name der Diskette; BCPL-String; erstesByte gibt an, wieviel Zeichen der String enthält (max. 30)
121	Tage des Datums, an dem diese Diskette initialisiert wurde
122	Minuten
123	Ticks
124	0
125	0
126	0
127	Root-ID; = 1

User-Directory-Block

0	Typ (= 2)
1	Header-Key (Nummer dieses Sektors)
2	0
3	0
4	0
5	Checksumme
6 bis 77	Hashtable; Sektornummern der Dateien oder Unter-Directories in diesem Directory (bzw. Sektornummern der entsprechenden Header)
78	reserviert
79	Protection-Bits (EXEC, DEL, READ, WRITE)
80	0
81 bis 104	Kommentarstring, BCPL-String
105	Tag des Datums, an dem dieses Directory geschaffen wurde
106	Minuten
107	Ticks

108 bis 123	Name dieses Directories; BCPL-String
124	Nächster Eintrag mit gleichem Hash-Wert
125	Sektornummer des Eltern-Directory
126	0
127	User-Directory (= 2)

File-Header-Block

0	Typ (= 2)
1	Sektornummer dieses Sektors
2	Gesamtanzahl der Datensektoren für diese Datei
3	Anzahl der benutzten Data Block Slots
4	Sektornummer des ersten Datenblocks
5	Checksumme
6 bis 77	Sektornummern der Datenblöcke
78	unbenutzt
79	Protection-Bits
80	Gesamtgröße der Datei in Bytes
81 bis 104	Kommentar als BCPL-String
105	Tag des Datums, an dem diese Datei geschaffen wurde
106	Minuten
107	Ticks (1/50 Sekunden)
108 bis 123	Name der Datei als BCPL-String
124	Nächster Eintrag mit gleichem Hash-Wert
125	Sektornummer des Eltern-Directories
126	0 oder Sektornummer des ersten Erweiterungsblocks (File List Blocks)
127	File Typ (= FFFFFFFD)

File-List-Block

0	Typ (= 1)
1	Sektornummer dieses Sektors
2	Anzahl der Datenblöcke in der Liste
3	wie oben
4	erster Datenblock
5	Checksumme
6 bis 77	Liste der Sektornummern der Datenblöcke
78 bis 123	unbenutzt
124	0
125	Sektornummer des Eltern-Directories
126	nächster Erweiterungsblock
127	Typ: FFFFFFFD

Datenblock

0	Typ: 8
1	Sektornummer des File Header Sektors
2	Sequenz dieses Datenblocks
3	Anzahl der Daten in Bytes
4	Sektornummer des nächsten Datenblocks
5	Checksumme
6 bis 127	Daten

10.2 Memory Handling

Das Speichersystem des Amiga ist ausgesprochen flexibel. Es gibt so gut wie keine festen, unveränderlichen Adressen, sondern ein Adreßmanager weist Speicherplatz individuell je nach Bedarf und Situation zu. Demnach gibt es beim Amiga nicht wie bei älteren Commodore-Rechnern bestimmte Speicherbereiche, die "immer" freigegeben sind für Benutzeranwendungen (der Kassettenpuffer war oft solch ein beliebtes Plätzchen).

Solch eine Speicherorganisation wäre auch unsinnig, denn der Amiga ist bekanntlich ein Multitasking-Computer, in dem sich mehrere Programme den Speicher teilen müssen. Zwangsläufig benötigt man dazu eine "Schalt- und Waltzentrale", die die Speicheraufteilung kontrolliert. Möchte man nun selber, aus welchem Grunde auch immer, ein Stück vom Speicherkuchen haben, dann gibt es dazu viele verschiedene Methoden. Hier die gebräuchlichsten:

10.2.1 Speicher durch Variablen reservieren

Jedesmal, wenn Sie einer Variablen einen Wert zuweisen, nehmen Sie dazu ein Stück aus dem allgemeinen Speicher und reservieren dieses Stück für diesen Wert. Es hängt nun davon ab, welcher Art Ihre Variable ist, um zu bestimmen, wieviel Speicherplatz benötigt wird. Eine lange Integer-Variable wie zum Beispiel f& reserviert 4 Bytes. Nun können Sie diesen Speicher

auch für andere Zwecke benutzen. Die Anfangsadresse erhalten Sie durch den BASIC-Befehl VARPTR, in unserem Fall also durch den Aufruf:

```
VARPTR (f&)
```

Benötigen Sie mehr als vier Bytes, dann können natürlich ganze Variablenfelder (DIM f&(100) reserviert 400 Bytes) oder Strings verwendet werden (a$ = SPACE$(100) reserviert 100 Bytes). Die Anfangsadresse des Strings erhalten Sie durch den Aufruf:

```
SADD (a$)
```

Es ist hier jedoch anzumerken, daß die Anfangsadresse des String-Speichers variabel ist! Bei jeder neuen String-Definition können alte Strings im Speicher verschoben werden. Deshalb ist vor jedem Zugriff auf diesen Speicher die Anfangsadresse erneut festzustellen, und deshalb eignet sich dieser Speicher auch nicht für feste Datenstrukturen, die man Maschinensprache-Routinen zugänglich machen möchte. Für solche Zwecke ist die folgende Methode wesentlich praktischer:

10.2.2 Speicher gezielt reservieren

Ein Exec-Befehl, AllocMem(), liefert Ihnen soviel Speicherplatz, wie Sie brauchen, sofern die angeforderte Größe frei ist. Sie können dabei zwischen folgenden Optionen wählen:

Public memory	2^0
Chip memory	2^1 (für DMA und Special Purpose Chips)
Fast memory	2^2 (für alle anderen Anwendungen)
Clear memory	2^{16} (löscht den Speicher automatisch)

Die folgenden beiden SUBs reduzieren die Arbeit bei der Speicherreservierung auf ein Minimum. Hier das Listing:

```
'##################################
'#                                #
'# Programm: Memory Handler       #
'# Autor:    tob                  #
'# Datum:    12.8.87              #
'# Version:  2.0                  #
'#                                #
'##################################

DECLARE FUNCTION AllocMem& LIBRARY

LIBRARY "exec.library"

demo:       '* besorgt 4500 Bytes
            PRINT "Speicher vor Festlegung der 4500 Bytes: ";
            PRINT FRE(-1)

            GetMemory mem&, 4500&

            PRINT "Aktueller Speicherstand danach: ";
            PRINT FRE(-1)

            FreeMemory mem&

            PRINT "Endstand des Speichers: ";
            PRINT FRE(-1)

            LIBRARY CLOSE
            END


SUB GetMemory (add&, size&) STATIC
    IF size& > 0 THEN
      opt&  = 2^16
      size& = size& + 4
      add&  = AllocMem&(size&, opt&)
      IF add& <> 0 THEN
        POKEL add&, size&
        add& = add& + 4
      END IF
    END IF
END SUB

SUB FreeMemory (add&) STATIC
    IF add& > 0 THEN
      add&  = add& - 4
      size& = PEEKL (add&)
      CALL FreeMem(add&, size&)
    END IF
END SUB
```

Programmbeschreibung

Das Prinzip dürfte aus dem Beispiel hervorgehen: Sie benutzen die GetMemory-Routine, um ein beliebig großes Speicherstück für Ihre Zwecke zu reservieren. Dabei liefern Sie zwei Variablen mit: erstens die Adreßvariable, in der Sie nach dem Aufruf die Anfangsadresse auf das Speicherstück wiederfinden (bzw. 0, wenn soviel Speicher nicht verfügbar war), und zweitens die Größe des gewünschten Speichers. Für ein 1000 Byte großes Stück geht das also so:

```
GetMemory myMem&, 1000&
```

Nach dem Aufruf finden Sie in der Variablen myMem& die Anfangsadresse auf den Speicher:

```
PRINT myMem&
```

Wenn Sie, vermutlich am Programmende, den Speicher nicht mehr brauchen, ist es Ihre Pflicht, ihn ans System zurückzugeben, damit er anderen Programmen oder dem System zur Verfügung gestellt werden kann. Dazu dient dieser Aufruf:

```
FreeMemory myMem&
```

Sie brauchen hier die Größe Ihres Speicherstücks nicht mehr anzugeben, denn GetMemory hat in Wirklichkeit ein um vier Byte größeres Speicherstück reserviert und in den überzähligen Byte die Größe abgelegt.

10.3 Das Printer.Device

Das Printer-Device bietet dem BASIC-Programmierer die Möglichkeit, einfach und ohne großen eigenen Programmieraufwand größtmöglichen Nutzen aus dem angeschlossenen Drucker zu ziehen. Erwähnenswert ist hier die einfach zu bewerkstelligende Ausführung einer Hardcopy-Routine, die den Inhalt eines Fensters grafisch auf den Drucker ausgibt.

10.3.1 Auslesen der Druckparameter

Auf dem Markt streiten sich eine wahre Fülle verschiedenster Drucker um die Gunst des Anwenders. Sie sind verschieden teuer, verschieden gut, aber vor allen Dingen haben sie alle ihre speziellen Stärken und Schwächen. So besticht der Typenraddrucker durch ein exzellentes Druckbild, ohne jedoch Grafiken ausdrucken zu können, der Matrixdrucker hingegen durch seine Farbgrafiken und der Laserdrucker schließlich durch Druckgeschwindigkeit und Auflösung.

Es ist sicherlich leicht einzusehen, daß alle diese Drucker auf ihre ganz besondere Weise von einem Programm angesprochen werden wollen. Diese Bürde nimmt Ihr Amiga Ihnen jedoch ab: Nachdem Sie in den Preferences (auf der Workbench-Diskette) denjenigen Drucker angegeben haben, den Sie zu benutzen gedenken, übernimmt der Amiga automatisch die Umwandlung der allgemeinen Druckbefehle in die druckerspezifischen Steuercodes, so daß Ihre Programme völlig universell mit jedem beliebigen Druckertyp lauffähig sind.

Welche Qualitäten der Drucker aufweist, der augenblicklich in den Preferences spezifiziert ist, das läßt sich mit dem folgenden Programm leicht feststellen, das auch gleich einen Einblick geben soll, wie der Drucker über das Printer-Device abgesprochen wird:

```
'*************************************
'* Programm: Drucker-Daten auslesen
'* Datum: 28. Mai 1988
'* Autor: tob
'* Version: 1.3
'*************************************
PRINT "Suche die .bmap-Dateien!"
'EXEC-LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION DoIO& LIBRARY
DECLARE FUNCTION OpenDevice% LIBRARY
DECLARE FUNCTION AllocSignal% LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
LIBRARY "exec.library"
init:     '
          GetPrinterData
```

```
        PRINT "Drucker-Name        : "; prt.name$
        PRINT "Drucker-Typ         : "; prt.typ$
        PRINT "Farbtuechtigkeit    : "; prt.farb$
        PRINT "Zeichen pro Zeile   : "; prt.columns%
        PRINT "Anzahl Zeichensaetze: "; prt.charsets%
        PRINT "Anzahl Rasterreihen : "; prt.rows&
        PRINT "Max. Anz. Dots horiz: "; prt.xdots&
        PRINT "Max. Anz. Dots vert.: "; prt.ydots&
        PRINT "Dichte: Dots/Inch h.: "; prt.xdotspi&
        PRINT "Dichte: Dots/Inch v.: "; prt.ydotspi&

        END

SUB GetPrinterData STATIC
    SHARED prt.DRPReq&
    SHARED prt.typ$, prt.farb$, prt.name$
    SHARED prt.columns%, prt.charsets%
    SHARED prt.rows&, prt.xdots&, prt.ydots&
    SHARED prt.xdotspi&, prt.ydotspi&

    DIM prt.color$    (9)
    DIM prt.printer$  (3)

    prt.color$  (1) = "Schwarz-Weiss"
    prt.color$  (2) = "Gelb-Magenta-Tuerkis"
    prt.color$  (3) = "Gelb-Magenta-Tuerkis oder Schwarz-Weiss"
    prt.color$  (4) = "Gelb-Magenta-Tuerkis-Schwarz"
    prt.color$  (5) = "Blau-Gruen-Rot-Weiss"
    prt.color$  (6) = "Schwarz-Weiss Invers"
    prt.color$  (7) = "Blau-Gruen-Rot"
    prt.color$  (8) = "Blau-Gruen-Rot oder Schwarz-Weiss"
    prt.color$  (9) = "Blau-Gruen-Rot-Weiss"

    prt.printer$(0) = "s/w Textdruck"
    prt.printer$(1) = "s/w Grafik"
    prt.printer$(2) = "farbiger Textdruck"
    prt.printer$(3) = "Farbgrafik"


    OpenPrinter

    prt.printerdata&  = PEEKL (prt.DRPReq& + 20)
    prt.extendeddata& = (PEEKL (prt.printerdata& + 92) + 12)
    prt.name$         = ""
    prt.name&         = PEEKL (prt.extendeddata&)
    prt.printer%      = PEEK  (prt.extendeddata& + 20)
    prt.color%        = PEEK  (prt.extendeddata& + 21)
    prt.columns%      = PEEK  (prt.extendeddata& + 22)
    prt.charsets%     = PEEK  (prt.extendeddata& + 23)
    prt.rows&         = PEEKW (prt.extendeddata& + 24)
    prt.xdots&        = PEEKL (prt.extendeddata& + 26)
    prt.ydots&        = PEEKL (prt.extendeddata& + 30)
    prt.xdotspi&      = PEEKW (prt.extendeddata& + 34)
```

```
      prt.ydotspi&      = PEEKW (prt.extendeddata& + 36)

      prt.typ$          = prt.printer$ (prt.printer%)
      prt.farb$         = prt.color$   (prt.color%)

      count = NULL
      char  = PEEK (prt.name& + count)

      WHILE char <> NULL
        prt.name$ = prt.name$ + CHR$ (char)
        count     = count + 1
        char      = PEEK (prt.name& + count)
      WEND

      ClosePrinter
END SUB
SUB OpenPrinter STATIC
      SHARED mem.chunk&
      SHARED prt.DRPReq&

      mem.clear&  = 2^16          'Speicher vor Gebr. loesch.
      mem.DRPReq% = 62            '62 Bytes fuer DRPStruktur
      mem.port%   = 37            '37 Bytes fuer Port-Strukt.
      mem.label%  = 4             '4  Bytes fuer Organisation
      mem.size%   = mem.DRPReq% + mem.port% + mem.label%

      mem.chunk&  = AllocMem& (mem.size%, mem.clear&)
      IF mem.chunk& = NULL THEN
        ERROR 7                   'OUT OF MEMORY ERROR
      END IF

      prt.label&  = mem.chunk&
      prt.DRPReq& = mem.chunk& + mem.label%
      prt.port&   = mem.chunk& + mem.label% + mem.DRPReq%
      prt.name$   = "printer.device" + CHR$(0)

      POKEL prt.label&, mem.size% 'Speichergroesse ablegen

      status% = OpenDevice% (SADD(prt.name$), 0, prt.DRPReq&, 0)
      IF status% <> NULL THEN
        PRINT "Drucker ist nicht frei."
        CALL FreeMem (mem.chunk&, mem.size%)
        EXIT SUB
      END IF
END SUB

SUB ClosePrinter STATIC
      SHARED mem.chunk&
```

```
      mem.size%   = PEEKL (mem.chunk&)
      prt.DRPReq& = mem.chunk& + 4
      CALL CloseDevice (prt.DRPReq&)
      CALL FreeMem (mem.chunk&, mem.size%)
END SUB
```

Variablen

prt.DRPReq&	I/O DumpRastPort-Struktur (Anfangsadresse darauf)
prt.typ$	Druckerkategorie
prt.farb$	Farbtüchtigkeit
prt.name$	Druckername
prt.columns%	Zeichen pro Zeile
prt.charsets%	Anzahl der verfügbaren Zeichensätze
prt.rows&	Anzahl der Drucknadeln
prt.xdots&	Max. Punkte in X-Richtung
prt.ydots&	Max. Punkte in Y-Richtung
prt.xdotspi&	Horiz. Auflösung in Punkten/Inch
prt.ydotspi&	Vert. Auflösung in Punkten/Inch

GetPrinterData():

prt.color$()	Feld der versch. Farbtypen
prt.printer$()	Feld der versch. Druckertypen
prt.printerdata&	Anfangsadresse der PrinterData-Struk.
prt.extendeddata&	Anfangsadresse der ExtendedData-Str.
prt.name&	Anfangsadresse des Namensstrings
prt.printer%	Code-Nr. des Druckertyps
prt.color%	Code-Nr. des Farbtyps
count	Zähler
char	gerade ausgelesenes Zeichen

OpenPrinter():

mem.chunk&	Anfangsadresse reserv. Speicher
mem.clear&	= 2^16; Speicher vor Gebrauch auf 0
mem.DRPReq%	= 62; 62 Bytes für Strukt. reserv.
mem.port%	= 38; 38 Bytes für Strukt. reserv.
mem.label%	= 4; 4 Bytes für Organisat. reserv.
mem.size%	benötigte Speichergröße in Bytes
prt.label&	Anfangsadr. Labelspeicher
prt.DRPReq&	Anfangsadr. DumpRastport-Struktur
prt.port&	Anfangsadr. Port-Struktur
prt.name$	Name des Devices
status%	0 = alles ok

Programmbeschreibung

Wenn Sie genau hinsehen, so werden Sie entdecken, daß das vorliegende Programm aus insgesamt drei SUB-Routinen besteht:

```
GetPrinterData
OpenPrinter
ClosePrinter
```

Für den Anwender ist lediglich das SUB GetPrinterData interessant. Es ruft intern die beiden anderen SUBs auf. Die benötigten Informationen finden sich in einer Struktur namens PrinterExtendedData. Um an diese zu gelangen, ist es zunächst notwendig, den Drucker via Printer.Device zu öffnen. Dies geschieht durch das SUB OpenPrinter.

Zunächst wird hierbei mittels der Exec-Funktion AllocMem() Speicherplatz für zwei Strukturen reserviert: eine Port- und eine DumpRastPort-Struktur. Außerdem werden vier Labelbytes mitreserviert, in denen später die absolute Speichergröße für FreeMem() abgelegt wird.

Ist dieses Vorhaben bewerkstelligt, wird der Drucker durch die Exec-Funktion OpenDevice() geöffnet. Dieser Aufruf liefert einen Statusreport zurück. Ist dieser Wert ungleich null, so konnte der Drucker nicht geöffnet werden. Mögliche Gründe: Er wurde von einem anderen Task benutzt oder nicht ordnungsgemäß geschlossen.

Ist der Drucker geöffnet, so findet sich in der DumpRastPort-Struktur ein Zeiger auf eine Struktur namens PrinterData. In ihr wiederum liegt der Zeiger auf die gesuchte PrinterExtended-Data-Struktur, in der die benötigten Daten gespeichert sind.

Die Daten werden ausgelesen und in den einschlägigen Variablen abgelegt. Anschließend kann der Drucker wieder geschlossen werden. Dies geschieht mittels eines Aufrufs der ClosePrinter-Routine und hat einen ernsten Hintergrund: Wird der Drucker zwar geöffnet, nicht aber durch dasselbe Programm wieder geschlossen, so bleibt er auf alle Ewigkeiten, mindestens jedoch bis zum nächsten Reset, unansprechbar.

10.3.2 Grafik-Dumps mit dem Printer-Device

Selbstverständlich war dies nur ein kleiner Vorgeschmack. Das nun folgende Programm soll Sie in die Lage versetzen, den Inhalt Ihres augenblicklich aktuellen BASIC-Fensters als Grafik auf einen Drucker auszugeben. Doch damit nicht genug: Alle Möglichkeiten des Printer-Devices sind unterstützt, sämtliche Spezial-Flags einschließlich der der neuen Betriebssystemversion 1.3 sind anwendbar. Damit können Sie Ihren Fensterinhalt beliebig verkleinern, vergrößern, verzerren, zentrieren sowie vieles mehr. Zunächst das Programm:

```
'************************************
'* Programm: Grafik-Dump
'* Datum: 28. Mai 1988
'* Autor: tob
'* Version: 1.3
'************************************
PRINT "Suche die .bmap-Dateien!"
'EXEC-LIBRARY
DECLARE FUNCTION AllocMem& LIBRARY
DECLARE FUNCTION DoIO& LIBRARY
DECLARE FUNCTION OpenDevice% LIBRARY
DECLARE FUNCTION AllocSignal% LIBRARY
DECLARE FUNCTION FindTask& LIBRARY
LIBRARY "exec.library"
init:   '
 CIRCLE (100,100),100
 PRINT STRING$ (100,"U")

 special.nothing   = 0 'keine Spezialeffekte
 special.milcols   = 1 'X-Dimension in 1/1000 Inch
 special.milrows   = 2 'Y-Dimension in 1/1000 Inch
 special.fullcols  = 4 'Maximale X-Ausdehnung
 special.fullrows  = 8 'Maximale Y-Ausdehnung
 special.fraccols  =   16 'Bruchteil der max. X -Ausdehnung
 special.fracrows  =   32 'dito, jedoch Y-Ausdehnung
 special.center    =   64 'Grafik wird zentriert ausgegeben
 special.aspect    =  128 'korrigiert X-Y-Verhaeltnis
 special.density1  =  256 'Aufloesung 1 (gering)
 special.density2  =  512 'Aufloesung 2
 special.density3  =  768 'Aufloesung 3
 special.density4  = 1024 'Aufloesung 4
 special.density5  = 1280 'Aufloesung 5
 special.density6  = 1536 'Aufloesung 6
 special.density7  = 1792 'Aufloesung 7 (hoch)
 special.noformfeed   = 2048 'kein Papierausschubnach Dump
 special.trustme   = 4096 'keinen Reset ausgeben
 special.noprint   = 8096 'nur Kalkulation, kein Druck
```

```
 Hardcopy (special.center + special.density4), 100&, 100&

 'für Schwarzweiß-Drucker den Bildschirm auf s/w

 PALETTE 0,1,1,1
 PALETTE 1,0,0,0

 Hardcopy (special.aspect + special.fullcols + special.fullrows), 0&, 0&


 END

SUB Hardcopy (flags, x&, y&) STATIC
 SHARED prt.DRPReq&

 OpenPrinter

 POKEL prt.DRPReq& + 52, x&
 POKEL prt.DRPReq& + 56, y&
 POKEW prt.DRPReq& + 60, flags
 InitDRPReq

 fehler% = DoIO& (prt.DRPReq&)

 fehler$  (0)   = "KEIN FEHLER."
 fehler$  (1)   = "DRUCK WURDE VOM BENUTZER ABGEBROCHEN."
 fehler$  (2)   = "DRUCKER KANN KEINE GRAFIK AUSGEBEN."
 fehler$  (3)   = "./."
 fehler$  (4)   = "DRUCKGROESSE UNMOEGLICH."
 fehler$  (5)   = "./."
 fehler$  (6)   = "KEIN SPEICHERPLATZ FUER INTERNE VARIABLEN."
 fehler$  (7)   = "KEIN SPEICHERPLATZ FUER DRUCKPUFFER."

 resultat$   = fehler$ (fehler%)

 PRINT resultat$

 ClosePrinter
END SUB
SUB OpenPrinter STATIC
 SHARED mem.chunk&
 SHARED prt.DRPReq&
 SHARED prt.port&

 mem.clear&  = 2^16    'Speicher vor Gebrauch loeschen
 mem.DRPReq% = 62   '62 Bytes fuer DumpRastport-Struktur
 mem.port%   = 38   '38 Bytes fuer Port-Struktur
 mem.label%  = 4    '4  Bytes fuer Organisation
 mem.size%   = mem.DRPReq% + mem.port% + mem.label%

 mem.chunk&  = AllocMem& (mem.size%, mem.clear&)
```

```
    IF mem.chunk& = NULL THEN
   ERROR 7        'OUT OF MEMORY ERROR
    END IF

    prt.label&  = mem.chunk&
    prt.DRPReq& = mem.chunk& + mem.label%
    prt.port&   = mem.chunk& + mem.label% + mem.DRPReq%
    prt.name$   = "printer.device" + CHR$(0)

    POKEL prt.label&, mem.size% 'Speichergroesse ablegen

    status% = OpenDevice% (SADD(prt.name$), 0, prt.DRPReq&, 0)
    IF status% <> NULL THEN
   PRINT "Drucker ist nicht frei."
   CALL FreeMem (mem.chunk&, mem.size%)
   EXIT SUB
    END IF
END SUB
SUB InitDRPReq STATIC
    SHARED prt.DRPReq&
    SHARED prt.port&
    SHARED p.sigBit%

    f.fenster& = WINDOW(7)
    f.rastport&   = PEEKL (f.fenster& + 50)
    f.breite%  = PEEKW (f.fenster& + 112)
    f.hoehe%   = PEEKW (f.fenster& + 114)
    f.screen&  = PEEKL (f.fenster& + 46)
    f.viewport&   = f.screen& + 44
    f.colormap&   = PEEKL (f.viewport& + 4)
    f.vp.modi% = PEEKW (f.viewport& + 32)

    p.sigBit% = AllocSignal%(-1)
    IF p.sigBit% = -1 THEN
   PRINT "Kein Signalbit frei!"
   CALL FreeMem(p.io&,100)
   EXIT SUB
    END IF
    p.sigTask& = FindTask&(0)

    POKE  prt.port&+8,4
    POKEL prt.port&+10,prt.port&+34
    POKE  prt.port&+15,p.sigBit%
    POKEL prt.port&+16,p.sigTask&
    POKEL prt.port&+20,prt.port&+24
    POKEL prt.port&+28,prt.port&+20
    POKE  prt.port&+34,ASC("P")
    POKE  prt.port&+35,ASC("R")
    POKE  prt.port&+36,ASC("T")

    CALL AddPort(prt.port&)

    POKE  prt.DRPReq& +  8, 5
```

```
   POKEL prt.DRPReq& + 14, prt.port&
   POKEW prt.DRPReq& + 28, 11
   POKEL prt.DRPReq& + 32, f.rastport&
   POKEL prt.DRPReq& + 36, f.colormap&
   POKEL prt.DRPReq& + 40, f.vp.modi%
   POKEW prt.DRPReq& + 48, f.breite%
   POKEW prt.DRPReq& + 50, f.hoehe%

   IF PEEKL (prt.DRPReq& + 52) = 0 THEN
  POKEL prt.DRPReq& + 52, breite%
   END IF

   IF PEEKL (prt.DRPReq& + 56) = 0 THEN
  POKEL prt.DRPReq& + 56, hoehe%
   END IF
END SUB

SUB ClosePrinter STATIC
   SHARED mem.chunk&
   SHARED prt.port&
   SHARED p.sigBit%

   mem.size%   = PEEKL (mem.chunk&)
   prt.DRPReq& = mem.chunk& + 4
   CALL CloseDevice (prt.DRPReq&)
   CALL RemPort  (prt.port&)
   CALL FreeSignal  (p.sigBit%)
   CALL FreeMem  (mem.chunk&, mem.size%)
END SUB
```

Variablen

(soweit nicht identisch mit Vorprogramm)

fehler%	Fehlernummer des I/O-Vorgangs
fehler$()	Klartext-Fehlermeldungen
resultat$	Aktuelle Fehlermeldung

Programmbeschreibung

Wie Sie sicher sofort erkannt haben, beinhaltet auch dieses Programm die SUBs OpenPrinter und ClosePrinter, die Sie ja bereits im vorangegangenen Programm kennengelernt haben. Neu hinzugekommen sind die SUBs Hardcopy sowie InitDRPReq. Für den Anwender spielt lediglich das SUB Hardcopy eine Rolle. Es sorgt dafür, daß der Inhalt des augenblicklich aktuellen BASIC-Fensters als Grafik zum Drucker gesendet wird, und

ruft dazu selbständig die anderen SUBs auf. Zum genauen Aufruf-Format dieses SUBs kommen wir gleich.

Um eine Grafik ausdrucken zu können, muß zunächst der Drucker geöffnet werden. Dies erledigt wie im vorangegangenen Programm das SUB OpenPrinter. Anschließend werden Breite und Höhe des auszudruckenden Bildes in die DumpRastPort-Request-Struktur eingepoked. Gleiches passiert mit den Spezialbits. Nun wird InitDRPReq aufgerufen. Diese Routine füllt den Rest der Struktur mit den Standardwerten und den Zeigern auf das BASIC-Fenster.

Nun kann die Exec-Funktion DoIO& aufgerufen werden, die den I/O-Request zum Drucker sendet. Ist der Druck beendet oder kann aus irgendeinem Grunde der Befehl nicht oder nur teilweise ausgeführt werden, so liefert diese Funktion an die Variable status% einen Fehlercode zurück. Dieser wird im folgenden in Klartext umgewandelt und auf den Bildschirm ausgegeben. Schließlich wird der Drucker wieder via ClosePrinter geschlossen, der Zugriff ist beendet.

Anwendung

Die Hardcopy-Funktion ist außerordentlich vielseitig und macht Gebrauch von sämtlichen Möglichkeiten, die das Printer-Device anzubieten hat. Der Aufruf des SUBs sieht folgendermaßen aus:

```
Hardcopy flags, Breite&, Höhe&
      flags:  Spezialflags
      Höhe:   Höhe des Ausdrucks
      Breite: Breite des Ausdrucks
```

Flags

special.nothing

Der Druck wird ohne spezielle Verfremdungen oder Effekte vorgenommen.

special.milcols

Die Druckbreite wird nicht - wie normalerweise - in Druckpunkten angegeben, sondern in 1/1000 Inch (1 Inch entspricht ca. 2,5 cm).

Hardcopy special.milcols, 9000, 400

Dieser Aufruf würde eine Grafik drucken, die eine Breite von 9,000 Inches (ca. 22,5 cm) und eine Höhe von 400 Druckpunkten aufweisen würde.

special.milrows

Wie special.milcols, jedoch bezügl. Druckhöhe.

special.fullcols

Unabhängig vom angegebenen Wert wird der Druck so breit wie hardwaremäßig möglich durchgeführt.

special.fullrows

Wie special.fullcols, jedoch bezügl. Druckhöhe.

special.fraccols

Die angegebene Breite wird als x/65535tel der Maximalbreite interpretiert.

special.fracrows

Dito, jedoch Druckhöhe.

special.center

Die Grafik wird auf dem Drucker zentriert ausgegeben. Dies geschieht unabhängig von den angegebenen Druckdimensionen.

special.aspect

Jeweils durch Korrektur der Höhe oder Breite wird das Originalverhältnis zwischen Höhe und Breite gewahrt.

special.density1-7 (V1.3)

Druckgenauigkeit; 1 = niedrig (Default), 7 = hoch

special.noformfeed (V1.3)

Unterbindet den Papierausschub speziell bei Laserdruckern, um Text und Grafik mischen zu können.

special.trustme

Kein Reset an Drucker senden.

special.noprint (V1.3)

Alle Angaben werden verarbeitet und alle Druckdimensionen errechnet, aber es kommt zu keinem Druck. Dient der vorherigen Kontrolle der Parameter.

Hinweise zur Betriebssystemversion 1.3

Neben den gekennzeichneten zusätzlichen Befehlen besitzt das V1.3-Betriebssystem zudem völlig überarbeitete Druckertreiber, die gegenüber der Version 1.2 einen immensen Geschwindigkeitsvorteil von bis zu 1000% aufweisen. Es lohnt sich also allemal, eine Workbench-Version 1.3 zu beschaffen (die V1.3-Druckertreiber laufen unabhängig von der Kickstartversion, also auch mit Kickstart V1.2 auf Amiga 500 und 2000).

11. Hardware-Basteleien

Warum soll man sich eigentlich immer nur äußerlich mit dem Amiga beschäftigen? Nicht nur die Software, sondern auch die Hardware hat es in sich! Denn unsere kleine Freundin steckt voller Überraschungen, die einen sind praktisch, die anderen unerwartet. Wir haben Ihnen gleich zu Anfang den Mund wässerig gemacht, und Sie sitzen schon mit angeheiztem Lötkolben neben dem Buch, trotzdem müssen ein paar Vorbemerkungen fallen:

1. Sie sollten sich bewußt sein, daß jede noch so qualifizierte Hardware-Änderung an Ihrem Gerät die Garantie verletzt und somit bei einem unverschuldeten Defekt kein Händler mehr kostenfrei repariert!
2. Alle Veränderungen, die hier beschrieben werden, sind natürlich von uns ausgetestet, wie es auch auf einer der ersten Seiten dieses Buches steht, doch ist es für einen Autor nicht immer möglich, alle Platinenversionen eines Computers aufzutreiben. Deshalb sei gleich gewarnt! Wenn nach theoretischem Durcharbeiten einer Schaltungsänderung Unstimmigkeiten vorliegen, lassen Sie lieber die Hände davon! Wir können nicht alle Gerätetypen kennen.
3. Bei jedem Eingriff sollten Sie eine gewisse Erfahrung von Elektronik und von Löt- oder Bastelarbeiten haben. Dies ist kein Grundkurs in Schaltungstechnik und will es auch nicht sein. Es wird also stillschweigend vorausgesetzt, daß Sie wissen, was Lötzinn ist, daß Sie auch mit einem Lötkolben umgehen können und daß Sie eine Schraube von einem Schraubenzieher unterscheiden können.

Nach dieser etwas harten Einleitung steigen wir gleich in die wundersamen Innereien unseres Lieblingscomputers ein. Dazu werden wir uns zuerst die Speicher-Erweiterungen ansehen und mit einigen kleineren Änderungen dazu bringen, daß sie nicht irgendwelche Programme stören, die sich damit nicht vertragen. Im nächsten Teil geht es dann an die Floppies. Immerhin kann

man auch diese abschalten, und manchmal braucht man diese Funktion ganz dringend. Als großes Bonbon rüsten wir so ganz nebenbei unseren Amiga mit einem 68010 um! Für die Nachdenklichen unter Ihnen gibt es einen HALT-Schalter und eine Taktfrequenz-Bremse, mit der die Geschwindigkeit des Amiga stufenlos herunterreguliert werden kann. Für die Kompatiblen unter Ihnen gibt es eine Bauanleitung für eine Prozessorumschaltung zwischen 68000'er und 68010. Und die Kommunikativen unter Ihnen finden hier die Bauanleitung für das serielle Verbindungskabel zum Betrieb einer Null-Modem-Schaltung und Datenkommunikation zwischen zwei Computern (das müssen nicht unbedingt beides Amigas sein!).

Zum klareren Aufbau geben wir für jede Umbau-Anleitung eine Liste mit den benötigten Teilen, eine Liste für evt. benötigtes Handwerkszeug und eine circa-Preisangabe, damit Sie in etwa über die anfallenden Kosten informiert sind.

11.1 Speichererweiterungen abschalten

Speichererweiterungen bieten nicht nur, wie man auf den ersten Blick denken mag, Vorteile! Schon oft kam ich zu dem Punkt, an dem ich meinen Amiga "verfluchte", weil durch diesen zusätzlichen Speicher manche Programme nicht liefen. Im eigentlichen Sinne liegt das Problem auch nicht bei der Erweiterung, sondern bei den Programmen, die einfach nicht qualifiziert angeben, welchen Speicher sie benötigen und so prompt den falschen bekommen. Dann versucht z.B. der Sound-Chip auf Daten zuzugreifen, die aber im FAST-RAM liegen, und schon ist der Absturz perfekt.

Die nächsten beiden Hardware-Tricks beziehen sich auf diesen Umstand. Es ist zwar in beiden Fällen eine Softwarelösung möglich: Für das komplette Abschalten des FAST-RAMs gibt es ein Programm im AMIGA Intern von DATA BECKER. Doch oftmals hat man einfach nicht die Diskette zur Hand. Da ist es dann einfacher, wenn man den Amiga ausschaltet, den Schalter umlegt und einen geschröpften Amiga wieder einschaltet.

11.1.1 Die 2000a-Platine

Haben Sie also einen Amiga 2000, so vergewissern Sie sich bitte zuerst, ob Sie auch die A-Platine vor sich haben. Denn nur hier kann eingegriffen werden. Dafür suchen Sie auf der Expansions-Platine nach einem PAL mit der Bezeichnung U3. Es gibt daneben auch noch U1 und U6, doch die interessieren uns hier nicht. Dieser PAL hat die ehrenvolle Aufgabe, die Freigabe eines Speicherbereichs zu organisieren, und genau dabei setzt unsere Lösung an: Wir werden diesem Chip einfach den Auftrag geben, den erweiterten Speicher nicht freizugeben! Und zwar läuft die Freigabe über zwei Pins ab, die sich -OVR und -SELECT nennen. Der erste hat die Position 19 und der zweite die Position 17.

Für unser Unterfangen ist es nun nötig, daß beide Pins nicht mehr mit der Platine verbunden sind, das heißt, es darf kein Strom fließen. Dafür gibt es nun zwei Möglichkeiten. Die erste und einfachste besteht darin, daß Sie die Leiterbahnen von diesen Pins mit einem gekonnten Schnitt durchtrennen. Dann besorgen Sie sich einen 2poligen Schalter und lassen über diesen die Verbindung laufen. Ist der Schalter geschlossen, so erkennt das System ganz normal das RAM. Wird er jedoch geöffnet - dafür muß der Computer ausgeschaltet sein und dies auch mindestens 5-10 Sekunden bleiben - dann ist der Speicher nicht vorhanden, und alle Programme laufen einwandfrei.

Die zweite Methode ist etwas eleganter, dafür aber auch aufwendiger. Besorgen Sie sich dafür noch einen Sockel (20 Pole genau wie der Chip), in den unser PAL gesteckt wird. Die beiden Pins, die wir eben erst auf der Platine erreicht haben, werden hier geschickt umgeknickt. Jetzt stecken Sie unseren Sockel in die alte Fassung ein. Die Verbindung ist durchtrennt und mit etwas Kabel und einem 2poligen Schalter läßt sich diese wieder herstellen. So wird nichts zerstört, und das Abschalten ist zusätzlich erlaubt.

Den Schalter empfehle ich mit einem nicht zu langen Kabel an das Gehäuse zu führen und dort durch ein Bohrloch zu stecken.

So laufen keine Überlandleitungen auf dem Tisch entlang, die zu Störungen führen könnten. Außerdem macht sich ein fest installierter Schalter besser als ein lose herumliegender!

Material:

- 1 Schalter 2 * AN
- ca. 30-40 cm zwei adrige Litze
- (1 Sockel 20 polig)
- Lötzinn

Werkzeug:

- Lötkolben
- scharfes Messer oder Schraubenzieher
- (evt. Zange)

Preis:

ca. DM 8,-

11.1.2 Die 500er-Platine

Ganz anders verläuft der Umbau beim Amiga 500. Hier beziehen wir uns auf die 512-KByte-Karte, die man von Commodore erhalten kann. Sie beinhaltet zusätzlich zu dem FAST-RAM noch eine akkugepufferte Uhr. Diese Uhr bleibt vom Eingriff natürlich unberührt. Zum Umbau schalten Sie bitte Ihren Amiga aus und öffnen den Erweiterungsschacht. Nachdem Sie die Karte herausgezogen haben - hier ist wie immer Vorsicht geboten - legen Sie sie vor sich hin. Sie sehen jetzt auf die Lötseite der Platine und haben oberhalb die Steckerleiste.

Für uns ist jetzt Pin 32 der Steckerleiste interessant. Verfolgen Sie ihn bis auf die Platine. Dort, wo eine Leiterbahn von der Lötstelle wegführt, setzt unsere Arbeit an. Irgendwo muß jetzt diese Leiterbahn unterbrochen werden. Nehmen Sie dafür ein scharfes Küchenmesser oder einen spitzen Schraubenzieher. An-

schließend nehmen Sie zwei Stück Litze und löten an jedem Schnittende je eine an. Die beiden anderen Enden werden dann über den Schalter verbunden. Fertig!

Sie können jetzt zur Probe die Speicherkarte wieder einbauen und die Workbench booten. Ist der Schalter in der AUS-Position, werden Sie wieder in die alten Tage zurückversetzt, in denen es noch keine Erweiterung gab. Schalten Sie aber den Amiga wieder aus und legen den Schalter um, muß nach erneutem Booten wieder der alte Speicher vorhanden sein. Ansonsten haben Sie etwas falsch gemacht. So werden z.B. sehr häufig manche Lötstellen "kalt" gelötet, so daß sie nicht richtig Kontakt geben. Dann empfiehlt es sich, diese noch einmal nachzulöten. Andere Fehler könnten aber eigentlich nicht auftreten.

Material:

- 1 Schalter 1 * AN
- ca. 30-40 cm zweiadrige Litze
- Lötzinn

Werkzeug:

- Lötkolben
- scharfes Messer oder Schraubenzieher

Preis:

ca. DM 7,-

11.2 Floppylaufwerke abschalten

Nicht nur der Speicher bringt manche Programme zum Absturz, auch ein zusätzlich angeschlossenes Laufwerk, das meistens autokonfigurierend ist und dann seinen Arbeitsspeicher fordert, bringt Probleme. Diese liegen meistens darin begründet, daß das DOS alle File-Daten nur über Chip-RAM abwickeln kann. Deshalb wird für jedes Laufwerk dort ein Pufferspeicher von etwa 30 KByte gebraucht.

Dieser Speicher wird von manchen Programmen dringend benötigt, steht dann aber nicht mehr zur Verfügung. Die Folge davon ist, daß dieses Programm sich verabschiedet oder gar nicht gestartet werden kann. Als Lösung bietet es sich an, das störende Laufwerk einfach abzuschalten. Dies wollen wir jetzt auch tun:

- Das externe Laufwerk wird abgeklemmt

Besitzt das externe Laufwerk von Haus aus noch keinen Schalter, dann kann man diesen leicht einbauen. Mit Hilfe des Schalters werden wir dann bei einem Reset des Computers die Leitung unterbrechen, die dem Amiga meldet, daß ein weiteres Laufwerk angeschlossen ist. Hierbei handelt es sich um die Leitung am Pin 21 des Steckers.

Wird sie unterbrochen, mit der Litze an beiden Enden verlängert und mit dem Schalter wieder verbunden, ist schon die ganze Arbeit getan. Es ist hierbei egal, ob Sie den Schalter am Stecker oder am Gehäuse der Floppy unterbringen. Für die Montage am Gehäuse müssen Sie aber erst die ankommende Leitung identifizieren. Und dazu muß zuvor sowieso der Stecker geöffnet werden.

Material:

- 1 Schalter 1 * AN
- ca. 10-20 cm einadrige Litze
- Lötzinn

Werkzeug:

- Lötkolben
- scharfes Messer
- Schraubenzieher

Preis:

ca. DM 7,-

11.3 Umrüstung auf den MC 68010

War es nicht schon immer Ihr Wunsch, Ihren Amiga für wenig Geld um einige Prozente schneller zu machen? Hiermit möchten wir Ihnen diesen Wunsch erfüllen! Dazu rüsten wir Ihren Amiga um! Der alte 68000er wird ausgetauscht und ersetzt durch einen 68010. Dieser neue Motorola-Chip ist 99.99% kompatibel zum alten. Er hat nur eine Macke, auf die wir aber noch eingehen und mit der man sehr gut leben kann.

Nun aber zu den Vorteilen! Der neue Prozessor kann durch einfaches Austauschen ersetzt werden. Das heißt, Sie brauchen weder lästige Lötarbeiten noch aufwendige Korrekturen durchzuführen. Weiterhin bietet der 68010 bei einigen Prozessorbefehlen eine Geschwindigkeitssteigerung um bis zu 80%. Das kann sich doch hören lassen. Wenn man dies auf die Allgemeinheit umrechnet, denn die Programme nutzen ja nicht nur die Befehle, die schneller geworden sind, sondern auch die, bei denen sich nichts geändert hat, dann hat man immer noch einen Zuwachs um glatte 16%. Auch diese Zahl kann sich hören lassen.

Weiterhin haben wir die Möglichkeit, mit einem Zusatzprogramm, zu dem Sie am Ende dieser Umbauanleitung Informationen finden, den neuen Chip 100% kompatibel zu machen, womit sich jedes Problem von selbst erledigt.

Nun aber endlich zum Einbau unseres neuen Prozessors. Dafür müssen Sie sich ihn erst einmal besorgen. Das sollte aber kein Problem darstellen. In vielen gewerblichen Kleinanzeigen von bekannten Computerzeitschriften findet man immer wieder Angebote dieses Chips. Oder Sie kennen einen Elektronik-Shop in Ihrer Nähe, der ihn sicherlich auch auf Lager hat. Er wird momentan zu einem Preis von ca. 40,- DM gehandelt, was ihn besonders beliebt macht. Haben Sie sich den neuen Prozessor besorgt, kann es an die Arbeit gehen.

Dafür schrauben Sie zuerst Ihren Amiga auf. Je nachdem, welches Amiga-Modell Sie besitzen, ist dieser Vorgang mehr oder

minder strapaziös. Sie sollten sich aber in jedem Fall genau die Reihenfolge und die Verbindungen merken, damit nachher beim Zusammenbau keine Gedächtnislücken entstehen und Ihr Amiga nur noch offen in Betrieb genommen werden kann.

Wenn Sie sich nun zur Hauptplatine durchgewuselt haben, wird es wohl kein Problem sein, den alten Hauptprozessor zu finden. Es ist immerhin der größte und klotzigste. Er läßt sich ganz einfach durch seine Aufschrift identifizieren. Immerhin steht dort "68000" oder ein ähnlicher Text. Diesen Chip müssen wir nun entfernen.

Zum Ausbau des Hauptprozessors gibt es noch einige Worte zu sagen. Zum ersten ist uns noch keine Amiga-Platine unter die Augen gekommen, bei der der 68000er nicht gesockelt war. Wenn Sie doch einen fest eingelöteten Prozessor vor sich haben, können wir nur hoffen, daß Sie genügend Löterfahrung haben oder aber einen Freund kennen, der diese hat. Denn bei einer solchen Masse von Pins kann man mit einfachem "Rumbraten" nur Zerstörung anrichten. Am besten ist die Verwendung einer extra dafür vorgesehenen Lötplatte, mit der man alle Pins gleichzeitig heiß machen kann, so daß der Chip während des Lötens herausgezogen werden kann.

Im folgenden wollen wir aber davon ausgehen, daß wir einen gesockelten 68000er vor uns haben. Dieser muß zuerst aus seinem Sockel entfernt werden. Dazu gibt es spezielle Zangen, die an allen Beinen gleichzeitig greifen. So eine Anschaffung lohnt aber nur, wenn Ihnen der Chip sehr am Herzen liegt oder Sie keinen Mut haben, es auch anders zu versuchen. Der gleiche Effekt läßt sich nämlich auch mit einem flachen Schraubenzieher erzielen.

Dazu setzen Sie ihn zuerst auf Schmalseite zwischen Sockel und Unterseite und versuchen den Prozessor durch leichtes Drehen, zu lockern. Gleiches wiederholen Sie auf der anderen Seite usw. Auf diese Weise kann man am einfachsten Chips aller Art, die mit besonders vielen Beinen gesegnet sind, aus der Fassung bringen. Merken Sie sich vor dem Herausnehmen, in welche

Richtung die Einkerbung an einem Ende des Chips zeigte. In die gleiche Richtung muß auch die Einkerbung des neuen Prozessors zeigen!

Als nächstes muß der neue 68010 eingesetzt werden. Bevor wir uns damit beschäftigen, sollte noch eine Warnung ausgesprochen werden. Diese Wunderwerke der Technik sind sehr empfindlich. Auch nur der kleinste Spannungsunterschied versetzt ihnen den Todesstoß. Deshalb haben Sie immer Verbindung zu einem geerdeten Leiter. Dazu kann man z.B. mit einer Hand Kontakt mit dem Chassis halten. Nehmen Sie nun den 68010 aus seinem Schaumstoffkissen, und setzen Sie unseren in den Ruhestand getretenen 68000er dorthin. Beim Eindrücken aller Pins ist darauf zu achten, daß keines auch nur einen halben Milimeter aus seinem Loch heraussieht. Biegen Sie gegebenenfalls die ganze Pinseite durch leichtes Drücken auf das Schaumstoffkissen um. Unser Mammutbeiner kann dann mit einem kräftigen Druck auf beiden Enden gleichzeitig in die Fassung gepreßt werden. Fertig!

Jetzt können Sie den Amiga wieder zusammenbauen. Achten Sie darauf, daß alle Verbindungen wieder richtig zusammengesetzt werden und daß keine Schraube oder irgend etwas anderes fehlt! Es soll schon Leute gegeben haben, die nach einem Umbau ein ganzes Chip-Set übrig hatten! Nun kommt die Einschaltprobe. Alles sollte sich wie normal verhalten, doch müßten Sie einen Geschwindigkeitszuwachs bemerken. Wenn nicht alles zur Zufriedenheit verlaufen ist, kann dies aus zwei Gründen so sein:

Zuerst ist es immer mal möglich, daß Sie beim Zusammenbau vielleicht irgendwelche Kabel nicht richtig oder gar nicht zusammengesteckt haben. Dies sollten Sie als erstes kontrollieren. Die zweite Möglichkeit ist nicht sehr angenehmen, kann aber durchaus aufgetreten sein. Beim Tragen von Kleidung entsteht schon durch die kleinste Reibung eine elektrische Ladung. Diese Ladung vertragen unsere beiden Chips nicht. Haben Sie nun ein Pin des Chips mit dem Finger angefaßt, so kann es angehen, daß dabei eine zu hohe Ladung von Ihnen auf den Prozessor übertragen wurde. Diese Ladung zerstört in den meisten Fällen einen

Mikrochip. Es hilft nichts, Sie müssen zum Test einen neuen 68010 kaufen. Durch das Einsetzen des alten 68000ers kann man auch feststellen, ob nicht gar der ganze Amiga defekt ist!

Zum Abschluß hier die versprochenen Informationen zum Programm, das nach dem Start jede Exception 4 abfängt, so daß der 68010 nicht abstürzt. Dieses Programm dürfen wir hier nicht abdrucken, da es sich um Public-Domain-Software handelt. Also verweisen wir hier auf die Fish-Disk 18. Auf ihr befindet sich das Programm DeciGEL, das alle Ausgleicharbeiten übernimmt. Die gleiche Aufgabe erledigt auch das Programm SetAlert, das auf den neuen Workbench-Disketten V1.3 mitgeliefert und schon in der Startup-Sequence gestartet wird. Außerdem ist dem neuen Amiga Profimat von DATA BECKER auch so ein Programm beigelegt.

Material:

- M68010-Prozessor 8 MHz
- (Sockel für den Prozessor, wenn noch nicht vorhanden)
- (Lötzinn)

Werkzeug:

- (Lötkolben)
- Schraubenzieher (zum Hebeln des alten Prozessors)

Preis:

ca. DM 45,- (50.-)

11.4 Laute Lüfter stören sehr!

Haben Sie auch einen Amiga 2000? Zu Anfang war ich richtig stolz, daß dieses "PC-Geräusch" mein Zimmer erfüllte, doch nach und nach wurde dieses Dröhnen, das dem eines Großraumflugzeuges gleicht, ziemlich entnervend. Deshalb beschloß ich, etwas dagegen zu unternehmen. Dafür standen mir zwei Wege offen. Einmal wurde mir von meinem Vater, er ist Fern-

sehtechnikermeister, geraten, einfach die Leistung des Ventilators zu drosseln. Doch weil ich bei solchen guten Ratschlägen immer etwas vorsichtig bin, habe ich eine elegantere Lösung gewählt.

Bei dem eingebautem Ventilator handelt es sich um einen Papst Multi-Fan 8312M. Das M am Ende der Nummer bedeutet so viel wie mittel(mäßig laut). Nach Wälzen von Prospekten und Händlerinformationen gelang es einem Bekannten von mir, ein ähnliches Modell herauszufinden. Dieses hat die gleichen Daten, mit dem Unterschied, daß die Geräuschentwicklung um fast 50% gemindert ist. Welch ein Erfolg!

Wenn es Ihnen gelingt, dieses Modell aufzutreiben - und das ist gar nicht so einfach - werden Sie keine Probleme beim Einbau haben. Denn alle Schrauben und Anschlüsse liegen gleich. Einziger Nachteil der ganzen Aktion ist der Preis unseres Modells Papst Multi-Fan 8312L. Denn für das gute Stück müssen Sie um die 80,- DM zahlen!

Wenn Sie sich jetzt von Ihrem Schock erholt haben, sollten Sie aber nicht vergessen, daß dadurch immerhin einige ruhige Stunden ermöglicht werden. Für die, denen das trotzdem zu teuer ist, möchte ich noch kurz die Methode meines Vaters beschreiben.

Er hat mir geraten, ganz einfach die positive Leitung des Lüfters durchzutrennen, damit ich einen Regelwiderstand von 50Ohm 5W einsetzen kann (der Regelbereich sollte so von 0 bis 100 Ohm liegen!). Dadurch wird nicht mehr so viel Strom durchgelassen wie vorher, und der Lüfter kann auch nicht mehr so schnell blasen. Mit Hilfe des Reglers ist es außerdem möglich, den Widerstand ganz auszuschalten, was dem alten Zustand gleicht. Dreht man ihn weiter auf, wird der Lüfter langsamer und somit auch leiser. Richten Sie dabei die Geschwindigkeit nicht nach der Lautstärke, sondern nach der Wärmeentwicklung. Diese steigt bei Erweiterungskarten an!

Zum Schluß sei noch eine Warnung mit auf den Weg gegeben: Der Lüfter des Amiga ist ja nur deswegen so laut, weil er so

viel Leistung aufbringen muß. Immerhin rechnet man damit, daß Sie alle freien Slots mit Steckkarten versehen. In diesem Fall wird die vorhandene Leistung des Lüfters wirklich gebraucht! Dann dürfen Sie ihn natürlich nicht bremsen oder durch einen schwächeren ersetzen. Nur wenn Ihr Amiga nicht voll ausgelastet ist, kann man zu diesen Methoden greifen.

Material:

- Lüfter
- Potentiometer
- Kabel
- Lötzinn

Werkzeug:

- Lötkolben
- Seitenschneider
- Schraubenzieher

Preis:

ca. DM 85,-

11.5 Den Amiga aus dem Takt bringen

Nicht immer kann Geschwindigkeit von Vorteil sein! Der Amiga ist mit seinen 7.16 MHz ein sehr schneller Computer. Dies macht die Arbeit sehr angenehm, die Fehlersuche oder filigrane Einstellungen aber für den Menschen unüberschaubar. Es geht aber auch anders. Wir haben uns zwei kleine Schaltungen überlegt, mit denen es zum einem möglich ist, den Amiga vollkommen anzuhalten, oder seine Geschwindigkeit zu drosseln.

Wie immer haben wir natürlich gleich darauf geachtet, daß dabei nicht zu große Kosten entstehen, daß die Schaltungen einfach nachzubauen und auch leicht zu verstehen sind. Denn schließlich sollen Sie auch über die Hintergründe unserer Ideen informiert werden.

11.5.1 Stop! Der Amiga hält an

Es sind nicht nur imaginäre Situationen, in denen man sich wünschte, seinen Amiga anhalten zu können. Ob es nun bei einem Spiel ist, das keine PAUSE-Taste besitzt, und von dem man kurze Zeit abgehen muß, oder ob es ein Anwenderprogramm ist, dessen Arbeiten man einfrieren möchte. Auch bei der Programmierung bietet es sich an manchen Stellen an, wenn man für eine Überlegung das Programm - oder besser den ganzen Amiga - anhält. Die hier vorgestellte Schaltung erledigt diese Aufgabe mit wenig Aufwand!

Wir benutzen dafür eine vom Prozessor angebotene Leitung, die wohl beim 68000er als auch beim 68010 vorhanden ist. Sie trägt die Bezeichnung HALT und liegt an Pin 17. Verbindet man dieses Pin mit der an Pin 16 liegenden Masse, so hält der Prozessor so lange an, bis die Masse wieder entfernt wird. Dies nutzen wir aus, indem wir die beiden Pins über einen Druckschalter miteinander verbinden.

Damit die Kabel nicht direkt am Prozessor angelötet werden müssen - durch die Lötarbeiten könnte er beschädigt werden - nehmen wir dazu eine separate Fassung, die zwischen der Platinenfassung und dem Prozessor gesteckt wird.

Löten Sie je ein Litzekabel an die Pins 16 und 17 der zusätzlichen Fassung. Sie sollten etwa die Länge von 10 bis 15 Zentimetern haben. Dies hängt einmal vom Amiga ab (500er, 1000er oder 2000er) und von der Position, an der Ihr Schalter angebracht oder heraushängen soll. Messen Sie den Weg vorher großzügig ab! Auf der anderen Seite werden die Kabel mit dem Schalter verbunden. Die Polung ist dabei vollkommen egal. Fertig!

Nehmen Sie nun nach dem Öffnen des Amiga den Prozessor vorsichtig aus seiner Fassung (nähere Beschreibungen dazu finden Sie in einem vorhergehenden Kapitel über die Aufrüstung mit einem 68010-Prozessor). Setzen Sie jetzt die Zwischenfassung richtig herum ein (man achte auf die Kerbe!) und darauf

den Prozessor. Da der Verlust einer Fassung leichter zu verschmerzen ist als der eines Prozessors, sollten Sie vielleicht bei einer unzugänglichen Position erst den Prozessor in die Zwischenfassung setzen und erst dann diese in der Computer.

Zum Ausprobieren bietet es sich an ein Demo-Programm oder ein Spiel zu laden. Dieses können Sie dann auf Knopfdruck anhalten. Viel Spaß!

Hinweis: Achten Sie beim Start des Amiga darauf, daß der Schalter ausgeschaltet ist, denn sonst kann Ihr Amiga nicht booten.

Material:

- Druckschalter 1 x an (einrastend)
- IC-Fassung für Prozessor (64 Beine)
- Lötzinn
- zweiadriges Litzekabel

Werkzeug:

- Seitenschneider
- flachen und breiten Schraubenzieher
- Lötkolben

Preis:

maximal ca. DM 8,-

11.5.2 Die Bremse im Huckepack

Nicht immer reicht es aus, den Amiga nur anzuhalten. Viel brauchbarer ist es, wenn man zwischen mehreren Geschwindigkeitsstufen wählen kann. Mit der nun vorgestellten Schaltung ist dies ohne weiteres möglich.

Wir überlassen es dabei einer kleinen taktgebenden Schaltung, den Prozessor auf den HALT-Status zu setzen. Dies geschieht in

wählbarer Anzahl innerhalb einer Sekunde mehrmals. Durch das Potentiometer können Sie bestimmen, wie oft der Prozessor auf HALT gelegt werden soll. Damit wird zwar die Arbeitsgeschwindigkeit des Prozessors verändert, nicht aber der Systeminterne Takt.

Das verwendete IC wird mit 5V und Masse aus dem Amiga mit Spannung versorgt. Herausgeführt ist eine Leitung, die wir wieder mit dem HALT-Pin verbinden. Hier liegt periodisch Masse an. Die Frequenz wird über einen Schwingkreis gesteuert, der mit dem Potentiometer innerhalb eines festgelegten Bereichs geregelt werden kann.

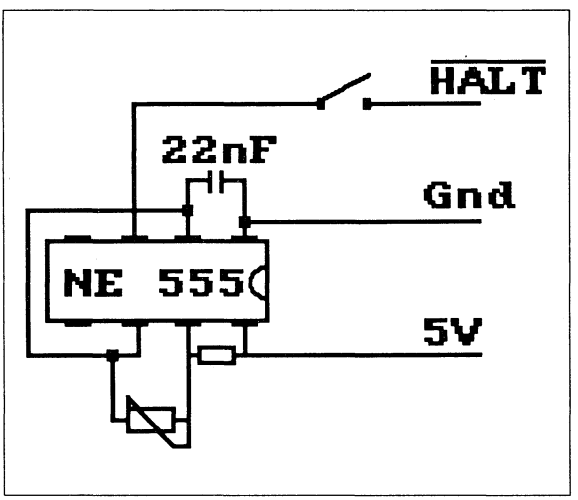


Abb. 4: Schaltung für Takt-Frequenz-Bremse

Für den Aufbau der oben gezeigten Schaltung verwenden Sie am besten die in der Material-Liste aufgeführte Lochraster-Platine. Dort können Sie leicht das IC einlöten. Achten Sie bei Platinen mit Lötstreifen darauf, daß jedes Bein in einer separaten Reihe steht. So müssen Sie nur in der Mitte unter dem IC die Bahnen durchtrennen. Dann können alle weiteren Bauteile ohne Mühe ergänzt werden. Der Ein-Schalter und das Potentiometer sollten über Kabel mit der Platine verbunden werden, damit diese nicht

starr mit den Bauteilen zusammenhängen. Sie können dann an geeigneter Stelle im Gehäuse je eine Bohrung für den Schalter und das Potentiometer vornehmen. Masse, 5 Volt und die HALT-Leitung werden wieder über die Prozessor-Fassung erreicht.

Hinweis: Um Geld zu sparen, können Sie auch die Halt-Schaltung und diese zusammen mit einer Prozessor-Fassung kombinieren. Sie sparen damit 4,50 DM, die für eine präzise Fassung nicht zuviel verlangt sind.

Material:

- Lochraster-Platine
- Schalter 1 x ein
- Kondensator 22 nF
- Widerstand 820 Ohm
- Potentiometer 10 kOhm
- IC NE 555 (Zeitgeber)
- IC-Fassung für Prozessor (64 Beine)
- Litze
- Lötzinn

Werkzeug:

- Lötkolben
- Seitenschneider
- Schraubenzieher

Preis:

ca. DM 10,50 (mit Fassung)

11.6 Prozessor-Umschaltung macht doppelt kompatibel

Sicherlich sind Sie begeistert, wenn durch den Einbau eines 68010 alles viel schneller, sprich: leichter von der Hand geht.

Die Workbench-Arbeit wird flüssiger, die Programme leisten mehr. Nun gibt es aber besonders auf dem Gebiet der Spiele einige, die sich nicht mit dem einen inkompatiblen Befehl vertragen und alsbald abstürzen.

Hier greift unsere Idee! Warum soll man nicht auch zwischen zwei Prozessoren umschalten können, wie man es auch beim Kickstart mittlerweile kann? Hierbei sind aber nicht - wie man zuerst denken mag - aufwendige Platinenarbeiten nötig, sondern es läßt sich mit wenigen Bauteilen realisieren. Das Verfahren ist einfach! Wir löten auf den in der Platinen-Fassung befindlichen Prozessor einen weiteren Sockel (natürlich nicht, während er im Amiga steckt, das versteht sich unter Elektronikern von selbst), in den dann der 68010 eingesteckt werden kann. Hierbei müssen allerdings einige Beine zur Seite gebogen werden, denn hier setzt die Schaltung an. Aber gehen wir die Schaltung der Reihe nach durch:

Nach dem Aufschrauben des Amiga (gleich welches Modell) entfernen Sie den 68000er vorsichtig aus seinem Sockel (wie wurde schon mehrmals oben beschrieben). Um eine Schaltung später zu ermöglichen, müssen einige Beine des Prozessors zur Seite gebogen werden. Es sind im einzelnen Pin 14, 16, 49 und 53. Beachten Sie, daß keines der Beine wirklich abbricht. Sollte dies doch geschehen, muß später versucht werden, einen Draht direkt an die Bruchstelle zu löten, was einige Schwierigkeiten verursacht. Jetzt können Sie den 68000-Prozessor in eine der beiden neuen Fassungen stecken. Achten Sie auf gleiche Richtung der Kerbe des Prozessors und der Fassung. Die abgebogenen Beine stehen frei.

Die zweite Fassung wird genau wie der Prozessor bearbeitet, d.h. die oben genannten Pins sind diesmal zu entfernen (knipsen Sie die Beine unterhalb mit einem Seitenschneider ab). Alle anderen Pins werden, nachdem Sie diese Fassung auf den 68000er gesetzt haben, angelötet. Achten Sie auch hier wieder auf Gleichgerichtetheit der Kerben. Nun sind die freistehenden Pins der Prozessoren oder Fassungen über Litzekabel in geeigneter Weise zu verbinden, um eine Umschaltung möglich zu machen.

Verwenden Sie zur Verkabelung am besten mehrfarbiges Litzekabel, damit die Leitungen leichter unterschieden werden können. Am bequemsten ist 6poliges Flachbandkabel, das die Arbeit sehr übersichtlich gestaltet. Zuerst verbinden Sie mit einem kurzen Litze-Stück die abgebogenen vier Pins des 68000ers paarweise: Pin 14 mit 49 und Pin 16 mit 53. Genauso wird die für den 68010 vorbereitete Fassung bearbeitet. Auch hier müssen die entfernten, also nicht mit dem darunter liegenden Prozessor verbundenen Pins in gleicher Weise untereinander verdrahtet werden.

Nun beginnt die Arbeit am Schalter. Löten Sie am 68000er von Masse-Pin (16 bzw. 53) eine Leitung zum Schalter-Pin 1a (siehe auch untenstehende Abbildung). Ein zweites Kabel wird vom 5V-Pin (14/49) zum Schalter-Pin 2a geführt. Das gleiche führen Sie mit zwei weiteren Leitungen an der Fassung für den 68010-Prozessor durch und verbinden diese am Schalter mit Pin 1b bzw. 2b. Es fehlt nur noch eine Brücke zwischen Schalter-Pin 1a und 3b, die die Masse des 68000ers beim Umschalten auf den 68010'er auf 5V legt, damit diese inaktiv bleibt.

Zur Schalter-Versorgung wird von der untersten Fassung, in die Sie den 68000er gesteckt haben, am Pin 14 bzw. 49 (5V) abgenommen und an die Schalter-Pins 2 und 3 gelegt. Weiterhin verbinden Sie Pin 16 bzw. 53 (Masse) der unteren Fassung mit dem Schalter-Pin 1. Fertig!

68000	Schalter			68010	
16/53	1a	1	1b	16/53	
14/49	2a	2	2b	14/49	
			3b	1a	(untereinander)
		2		3	(untereinander)
16/53		1			(untere Fassung)

Abb. Schalter-Verbindung

Zum Einbau dieser Huckepack-Schaltung stecken Sie nun endlich den 68010 vorsichtig (ohne daß ein Bein abbricht oder ab-

knickt) in die obere Fassung (achten Sie auf die Kerbe) und dieses Gebilde wiederum in Ihren Amiga. Auch hier ist die Richtung der Pins über die Kerbe zu beachten.

Zur Durchführung eines Tests schalten Sie den Amiga nach dem Zusammenbau ein und legen keine Workbench-Diskette ein. Ist der Amiga funktionsfähig, sollte das Disketten-Symbol auf dem Bildschirm erscheinen. Ansonsten müßten noch einmal alle Verbindungen auf ihre Richtigkeit überprüft werden. Schalten Sie zum Test des zweiten Prozessors den Amiga wieder aus, legen Sie den Schalter um und schalten Sie wieder ein. Auch hier sollte das Diskettensymbol erscheinen.

Hinweis: Sie zerstören Ihren Amiga, wenn Sie zwischen den Prozessoren umschalten und unseren Liebling nicht zuvor vom Netz getrennt haben. Beachten Sie dies immer! Bei unserem Test wurde ferner folgendes Problem aufgeworfen: der Prozessor erhitzt sich beim Betrieb sehr stark. Je nachdem, welchen der beiden Sie eingeschaltet haben, ist es der untere 68000er oder der oben gelegene 68010. Letzterem macht es wenig aus, da die Wärme frei entweichen kann. Nur dem 68000'er sollte es ganz schon warm ums Herz werden. Deshalb kann es sein, daß er wegen Überhitzung (wie man im Fachjargon sagt) "stirbt". Bei unserer Arbeit ist dies jedoch nicht geschehen. Um dem vorzubeugen, können Sie z.B. ein Kühlblech aufbringen und mit Wärmeleitpaste am unteren Prozessor befestigen. Es war aber während der Testphase nicht nötig.

Material:

- M68000 8 MHz
- M68010 8 MHz
- 2 Fassungen 64 polig
- Schalter 3 x um
- Litze (möglichst Flachbandkabel sechsadrig)
- Lötzinn

Werkzeug:

- Lötkolben
- Seitenschneider
- Schraubenzieher

Preis:

ca. DM 80,- (inclusive beider Prozessoren, sonst DM 12,-)

11.7 Das Null-Modem zur Datenkommunikation

Ein Null-Modem ist die Verbindung zwischen zwei Computern über ein ununerbrochenes Kabel. Man unterscheidet bei der Modem-Übertragung zwischen Telefon-Modems, die mit der Telefon-Leitung Daten verschicken, und dem hier beschriebenen Null-Modem, das mit einem Verbindungskabel auskommt und praktisch jeden Computer mit jedem verbinden kann. Einzige Bedingung für diesen Lebensbund ist eine RS-232 Schnittstelle, die unser Amiga auf jeden Fall besitzt.

Als Beispiel für unsere Anwendung habe ich einen Amiga 2000 und einen AT-Rechner genommen. Die Lötarbeiten sind dabei ganz einfach. Man benötigt ein mindestens sechsadriges Kabel, das die Länge von 5 Metern nicht überschreiten sollte, da man sonst ohne geeignete Verstärkung mit Datenverlusten rechnen muß. Zusätzlich sind noch zwei RS-232-Buchsen zu besorgen, die unsere gewünschte Verbindung herstellten. Die einzelnen Leitungen des Kabels werden nach folgendem Schema verbunden:

Serieller Port A	Serieller Port B
1	3
3	1
4/5	8
6	20
8	4/5
20	6

Sie sehen hier schon auf den ersten Blick, daß es sich um eine kreuzweise Verbindung handelt, die keine größeren Schwierigkeiten bereiten sollte. Beachten Sie bei den Buchsen die kleinen eingestanzten Nummern an jedem Lötpin, so bekommen Sie keine Probleme beim Zählen. Prüfen Sie auf jeden Fall vor dem Anschluß noch einmal theoretisch die Verbindungen, da Sie sonst böse Überraschungen erleben könnten.

Hier noch eine Informationstabelle, was an welcher Leitung anliegt:

Pin-Nummer	Signal
1	Schutz-Masse
2	Daten losschicken
3	Daten empfangen
4	Anfrage zu senden
5	Bereit zum Senden
6	Data Set Ready
7	Signal-Masse
8	Carrier Detect
20	Data Terminal ready

Haben Sie nun zwei Rechner verbunden, können Sie mit den ersten Probeeinstellungen beginnen. Dazu ist zuerst mit Preferences die serielle Schnittstelle am Amiga einzustellen. Ich verwende ohne Probleme folgende Werte:

```
Baud Rate       1200
Buffer Size     512 (bei AUX: unbenutzt)
Read Bits       8
Write Bits      8
Stop Bits       2
Parity          None
Handshaking     RTS/CTS
```

Seit der Workbench 1.3 ist es empfehlenswert, mit dem seriellen Device AUX: zu arbeiten, da hier die Datenübertragung ungepuffert geschieht. Bedenken Sie, daß es vor dem ersten Benutzen geMounted werden muß! Auch beim zweiten Rechner sollten die gleichen Einstellungen gemacht werden. Ist dies ein Amiga,

übernehmen Sie die oben angegebenen Werte. Als Beispiel soll dazu noch die Einstellung bei einem PC-kompatiblen Rechner gelten:

```
mode com1:1200,n,8,2,
```

Zur Datenübertragung selbst schicken Sie bei dem einen Rechner Daten ab, indem Sie z.B. ein File in das entsprechende Device kopieren:

```
Amiga: 1> copy testfile to AUX:

PC:    c:\> copy testfile com1:
```

Der Empfang läuft beim Amiga über ein CON:-Fenster, in das die Daten der Schnittstelle geschickt werden. Sie können es auf folgende Weise öffnen:

```
1> run copy aux: to con:0/0/640/100/Datenempfang
```

Beim PC-kompatiblen Rechner sollten Sie ein Terminal-Programm wie z.B. VTERM verwenden, da hier neben den Parametern der Empfang protokolliert wird.

Material:

- Kabel sechsadrig (ca. 2 Meter)
- RS232-Buchsen
- Lötzinn

Werkzeug:

- Lötkolben
- Seitenschneider
- Schraubenzieher

Preis:

ca. DM 15,-

12. Das Workbench-Equipment

Zusätzlich zu den lebensnotwendigen Datenfiles wie Fonts, Druckertreiber und den unbedingt nötigen Programmen wie CLI-Befehle und den Libraries findet man auf der Workbench-Diskette noch einiges mehr. Gleiches gilt auch für die Extras-Diskette, die viele Ergänzungen und zusätzliche Unterstützung bietet. Dieses Kapitel will sich nun mit allen Ergänzungen beschäftigen, die das Leben auf der Workbench leichter machen sollen. Hierfür betrachten wir beide Disketten. Zuerst wollen wir uns den Programmen widmen, ohne die manche Aufgabenstellung unmöglich zu lösen wäre. Im zweiten Teil geht es dann auch an die Datenfiles, in denen einige versteckte Informationen liegen. Doch nun stürzen wir uns wie Tarzan in das Abenteuer eines unerkundeten Dschungels!

12.1 Mit Vorliebe: Preferences

Das Programm Preferences ist sicherlich eines der wichtigsten Programme, die man mit dem Amiga geliefert bekommt. Schon alleine der Umstand, daß man alle Daten, die die Arbeitsumgebung prägen und bestimmen, ein- und verstellen kann, macht dieses Programm so wichtig! Nun liegt hinter diesem ganzen Lob aber auch ein bitterer Geschmack. Immerhin nimmt das Programm in seiner neuesten Version fast 55 KByte in Anspruch. Das ist schon mal ein Grund, auf Preferences zu verzichten, denn die Ladezeit läßt einen schon ab und zu verzweifeln.

Hierfür gibt es mehrere Auswege. Der erste besteht darin, daß man Preferences vielleicht in die (ab Version 1.3 resetfeste) RAM-Disk legt und so immer schnellen Zugriff hat. Dafür spricht der große Bedienkomfort. Es ist aber viel öfter so, daß manche Programme wichtige Einstellungen benötigen, wie z.B. den 80er Zeichensatz oder eine bestimmte Farbeinstellung. Dafür

lohnt es sich dann gar nicht, wenn man extra Preferences lädt. Und außerdem ist es gar nicht möglich, daß man immer wieder genau die gleichen Farben einstellt!

12.1.1 Daten lesen und setzen

An dieser Stelle greift nun das Hilfsprogramm ein. Es ermöglicht über die Intuition-Funktionen einen Zugriff auf die Daten und zeigt an zwei Beispielen, wie einfach es ist, die Struktur auszulesen. Dabei muß beachtet werden, daß seit Version 1.3 des Betriebssystems und der Workbench die Preferences-Struktur um einige Datenfelder ergänzt und verlängert wurde. Dieser Umstand wurde natürlich berücksichtigt! Trotzdem läuft das Programm auch mit Kickstart 1.2.

Wenn Sie wissen wollen, wie die Preferences-Struktur aussieht, schauen Sie bitte im Kapitel 6 nach, wo ein Programm zum Einstellen der Config.Sys-Daten vorgestellt wird. Sie wissen nach dem Studium der dort aufgeführten Struktur zwar, welche Daten von der Preferences-Struktur verwaltet werden, doch eine Programmier-Anwendung dafür haben wir immer noch nicht! Deshalb möchte ich Ihnen ein Programm anbieten, das es ermöglicht, die aktuellen Einstellungen zu lesen und andererseits dieses zu verändern und dann zu sichern.

Bei dieser Arbeit helfen uns zwei Intuition-Funktionen. Die eine kopiert die aktuellen Werte in einen von uns reservierten Speicherbereich. Dann sind wir am Zuge und können mit geschickten Eingriffen die Daten verändern. Mit der anderen Funktion wird es uns erlaubt, diese korrigierten Parameter in den Systemeintrag zu übernehmen, womit jetzt die Einstellungen allgemeingültig sind.

```
'*********************************
'*                               *
'* Preferences-Daten einstellen *
'* ---------------------------- *
'*                               *
'* Autor  : Wolf-Gideon Bleek   *
'* Datum  : 15. Mai '88         *
```

```
'* Grueße : Brigitte & Verena   *
'* Version: 1.1                 *
'* Betriebssystem: V1.2 & V1.3  *
'*                              *
'********************************

LIBRARY ":bmaps/exec.library"
DECLARE FUNCTION AllocMem& LIBRARY
LIBRARY ":bmaps/intuition.library"

Hauptprogramm:
  ' Prefrences laden
  GetPreferences Prefs&, 220&
  IF Prefs& = 0 THEN GOTO Ende

  ' Farben neu eintragen

  Colour0 = 110
  Colour1 = 112
  Colour2 = 114

  POKEW Prefs& + Colour0, 1*15+16*4+256*15
  POKEW Prefs& + Colour1, 1*15+16*15+256*0
  POKEW Prefs& + Colour2, 1*0+16*15+256*8
  ' Preferences speicher
  SetPreferences Prefs&
Ende:
  FreePreferences Prefs&
  LIBRARY CLOSE
END

SUB GetPreferences (Adresse&, Size&) STATIC
  Adresse& = AllocMem&(Size&+4, 65536&)
    IF Adresse& <> 0 THEN
      POKEL Adresse&, Size&
      Adresse& = Adresse&+4
      CALL GetPrefs(Adresse&, Size&)
    ELSE
      Adresse& = 0
    END IF
END SUB
SUB SetPreferences (Adresse&) STATIC
    IF Adresse& <> 0 THEN
      Size& = PEEKL(Adresse&-4)
      CALL SetPrefs(Adresse&, Size&, -1)
    END IF
END SUB
SUB FreePreferences (Adresse&) STATIC
    IF Adresse& <> 0 THEN
      Size& = PEEKL (Adresse&-4)
      CALL FreeMem(Adresse&-4, Size&+4)
    END IF
END SUB
```

Programmbeschreibung

Das Programm bietet drei SUB-Routinen an, die alle Funktionen unterstützen. Die erste, GetPreferences, reserviert für die Datenstruktur den benötigten Speicher und kopiert in diesen die Daten selbst. Falls ein Fehler auftritt, liefert sie den Wert 0 zurück. Dann kann im Programm jede Einstellung modifiziert werden. Achten Sie dabei auf die richtigen Startadressen, wie sie auch in der Tabelle stehen, und auf die richtige POKE-Länge. POKE für Byte-Werte, POKEW für Worte, d.h. 2-Byte-Werte und POKEL für Langworte, d.h. 4-Byte-Werte. Die Länge entnehmen Sie auch aus der Tabelle, indem Sie die Differenz zwischen Basis und nächster Adresse bilden.

Die zweite Funktion, SetPreferences, übergibt die Daten aus dem Speicher an das System und teilt dabei allen anderen laufenden Programmen diese Veränderung mit (das kennzeichnet die -1!). Wollen Sie nicht, daß auch andere Utilities von der Änderung erfahren, dann müssen Sie den Wert 0 einsetzen.

Mit der letzten Funktion FreePreferences wird der Speicher wieder freigegeben. Das ist wichtig und kennzeichnet guten Programmierstil. Immerhin sollte man alles, was man nicht mehr benötigt, für andere Programme freigeben, damit nicht irgendwann Speichermangel eintritt.

12.1.2 Die neuen Preferences (Version 1.3.10)

In Zusammenhang mit der Betriebssystemverbesserung hielt man es auch für nötig, die Einstellungen und Veränderungen bei den Preferences zu erweitern. Hinzugekommen sind hauptsächlich Druckerbeeinflussungen. So hat man als erstes einmal alle Druckertreiber wesentlich beschleunigt. Eine Hardcopy, die früher an die 10 Minuten dauerte, braucht jetzt nur noch knapp 2 Minuten. Wenn das kein Fortschritt ist!

Weiterhin hat man sich viele Gedanken über den Grafikausdruck gemacht, der bisher zwar unterstützt wurde, aber doch sehr stiefmütterlich. Nun kann man wirklich alles einstellen. Funktio-

sind ab Version 1.3 Wirklichkeit. Die anderen kleinen Änderungen möchte ich jetzt noch kurz vorziehen, bevor ich ausführlichst auf die Grafikeinstellungen eingehen werde. Beim ersten Blick auf das Titelbild fällt gar nicht auf, was alles geändert wurde. So ist z.B. der CLI-Schalter "abgeschaltet worden".

Man kann jetzt immer im CLI arbeiten und braucht sich dafür nicht erst die Erlaubnis zu holen. Eine Veränderung, so meine ich, die lange überfällig war. Eine interne Verbesserung, die keiner auf den ersten Blick erkennen wird, wurde bei der Uhr vorgenommen. So ist jetzt jede Einstellung auch gleich in die akkugepufferte Systemuhr übernommen!

Das ist ein großer Vorteil für alle 500er- und 2000er-Eigentümer. Der lästige Weg über das CLI entfällt.

Quereinstieg in Preferences

Eine wirklich gute Verbesserung ist mit dem Quereinstieg in die Preferences gelungen. Kennen Sie das auch? Man hat sich einen neuen Drucker zugelegt und will jetzt die richtigen Einstellungen ausprobieren. Dazu wählt man immer wieder Preferences und verändert auf der Pinter-Seite die Daten. Nach und nach wird dieser Vorgang aber ziemlich lästig, denn immer müssen Sie auf der Hauptseite erst anwählen, daß Sie auf die Druckerseite gehen möchten. Das ist jetzt vorbei.

Dafür wurde Preferences in eine Schublade gepackt und um drei neuen Gesellen ergänzt. Wir finden dort Pointer, Printer und Serial. Alle drei sind nichts weiter als Pseudo-Icons, die kein Programm repräsentieren, sondern nur Preferences aufrufen. Jedoch übergeben Sie in Tool Types die Variable PREFS, die entsprechend des Icons auf POINTER, PRINTER oder SERIAL gesetzt wird. Durch diese Übergabe wird Preferences angewiesen, gleich in die gewünschte Seite zu gehen. So erspart man

sich immerhin einen Handgriff! Wenn Sie einmal Preferences über das CLI starten möchten, so brauchen Sie auf diesen Komfort nicht zu verzichten! Geben Sie nach dem Programmnamen nur den Flag-Namen an.

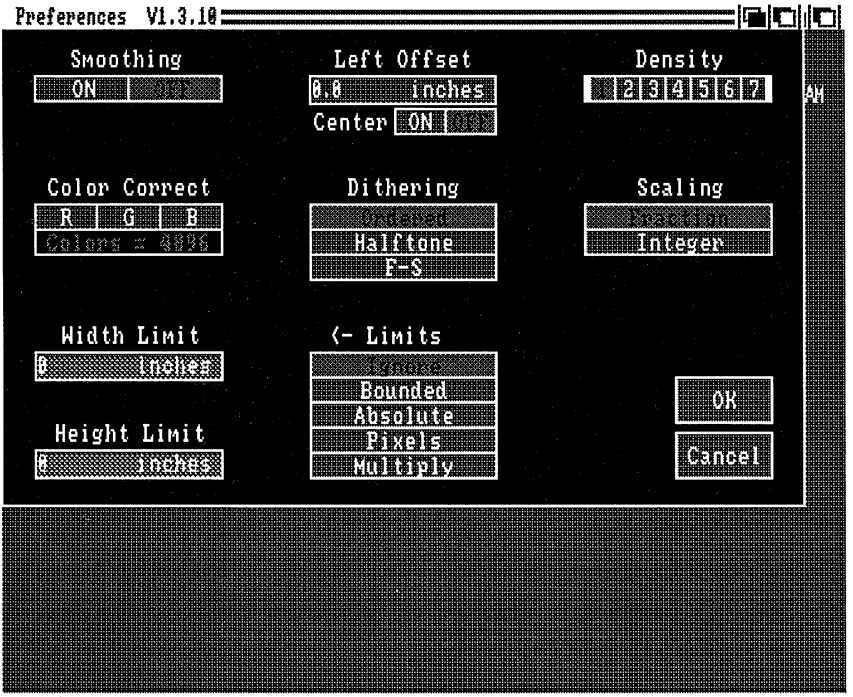
Neue Grafikeinstellungen

Gehen wir jetzt einen Schritt tiefer in die Voreinstellungen und sehen uns dazu das Grafik-Fenster an. Dort wurde der locker geschwungene Pinsel durch ein weiteres Klickfeld ersetzt. Doch bevor wir in "Graphic 2" in die neuen Grafikeinstellungen wandern, werfen wir zunächst einen Blick auf die alten Einstellungen. Hier ist nämlich ganz nebenbei eine neue Grauabstufung hinzugekommen. Sie wird mit "Gray Scale 2" bezeichnet.

Die Anwendung dieser Funktion ist allerdings recht selten. Und zwar bezieht sie sich auf sog. Hedley-Monitore. Das sind Monitore, die nur sieben Graustufen in der Darstellung erlauben, und damit das Bild dem des Bildschirms möglichst ähnlich wird, wurde dieser Parameter implementiert. Doch kommen wir jetzt zu der neuen Grafikseite. Wir haben hier eine komplett neue Auswahl an Veränderungen, die alle dazu neu eingerichtet wurden, daß die Grafikausgabe naturgetreuer und besser wird.

Antialiasing/Smoothing

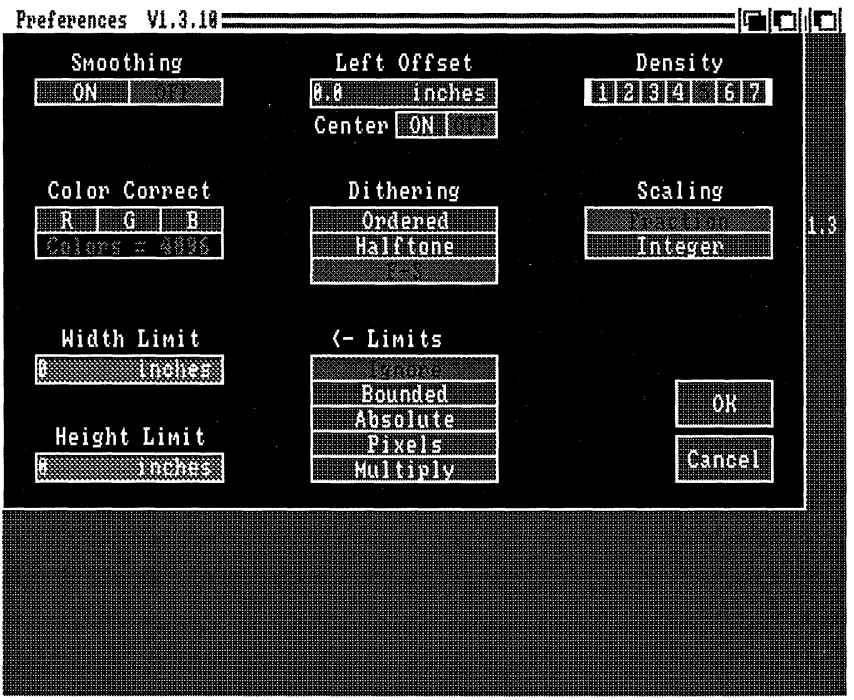
Als erstes findet man oben links in der Ecke die Funktion "Antialiasing" oder auch "Smoothing". Damit bezeichnet man eine Weichzeichnung von schräg laufenden Linien. Diese sind bisher immer in Treppenabstufungen ausgegeben worden. Mit Hilfe von Antialiasing werden diese Treppen in weiche Übergänge umgewandelt.



Ein sehr guter Anwendungsbereich liegt nach Commodore in der Ausgabe von Text über die Hardcopy-Funktion. Dann wird der Text nämlich nicht mehr in Kästchen dargestellt, sondern hat markante Konturen.

Position

Gehen wir nun eine Einstellung weiter nach rechts. Dort können wir bestimmen, ob die Grafik nicht vielleicht eingerückt werden soll. Oder es ist sogar möglich, die Grafik zentriert ausgegeben zu bekommen. Dann ist die Angabe eines Offsets natürlich gesperrt.



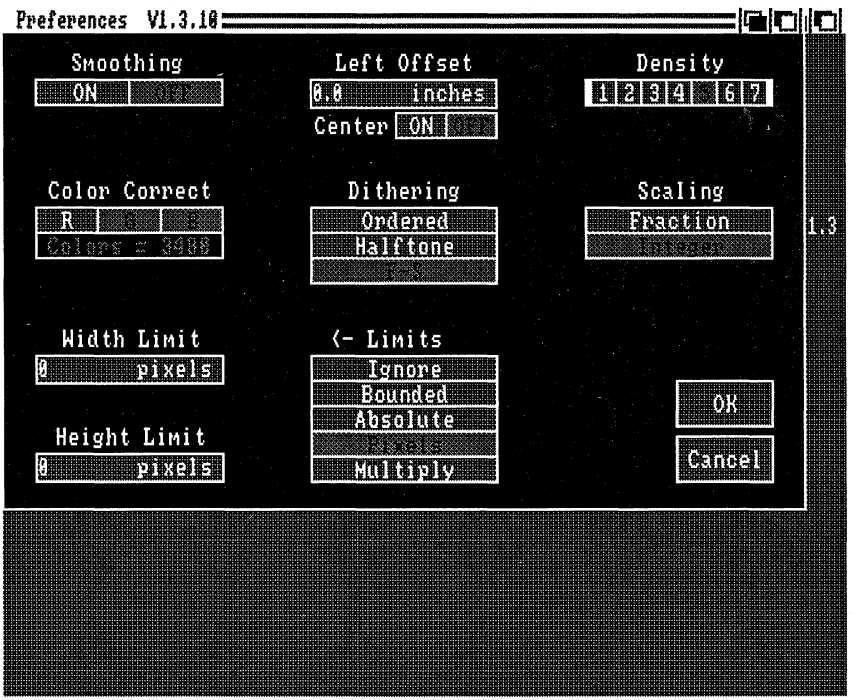
Density

Ganz rechts finden wir in der oberen Reihe das Feld für "Densitiy", was auf deutsch soviel wie Dichte heißt. Bei der Grafikausgabe ist es nicht unwichtig, daß man die Dichte einstellen kann. Doch bedeutet eine höhere Dichte auch ein Mehr an Zeit. Somit ist es hilfreich, wenn über Preferences einfach eine Dichte vorgegeben wird und man dadurch die Grafikausgabe beschleunigen kann. Je geringer die Einstellung ist, desto schneller erfolgt der Ausdruck.

Color Correct

Mit "Color Correct" kommen wir in einen neuen Bereich. Da jede Grafik aus maximal 4096 Farben bestehen kann, ist eine Umsetzung in Grau- oder Farbstufen nötig, bei der auf jeden

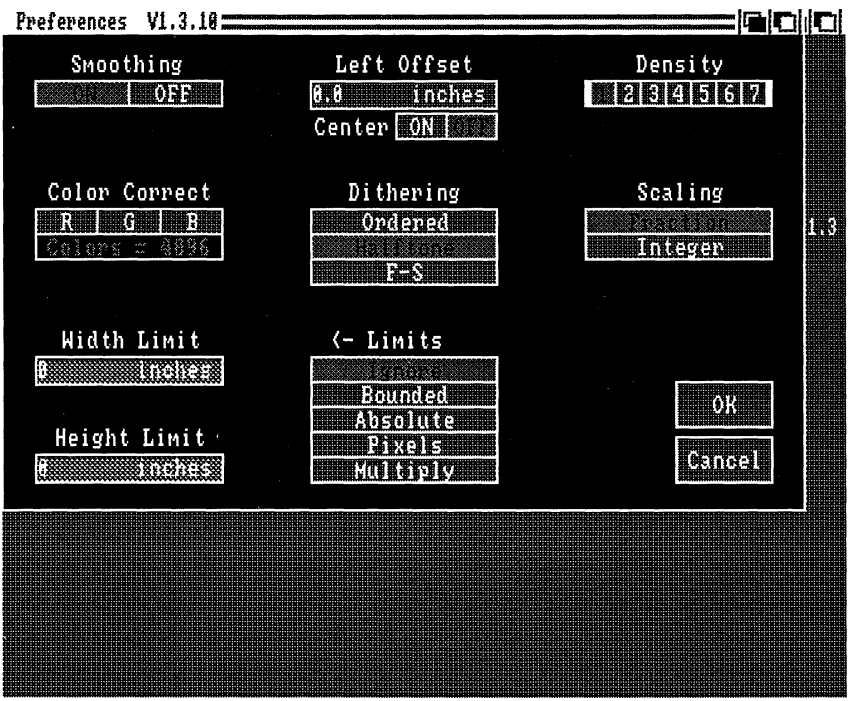
Fall ein Verlust eintritt. Es gibt nun einen Algorithmus, der immer ganz stur seine Umsetzung betreibt. Doch in manchen Fällen spielen Teilbereiche der Farben eine nicht so große Rolle wie andere. Hier wäre es angebracht, wenn bei der Aufteilung z.B. die Rottöne gar nicht erst beachtet werden, so daß alle Graustufen nun auf eine viel geringere Zahl aufgeteilt werden. Dadurch erhält man oftmals eine bessere Ausgabe.



Dithering

Auch "Dithering" bezieht sich auf die Grauabstufung. Es gibt dafür ja mehrere Verfahren. So kann man jeder Farbe ein entsprechend grobes Muster an Punkten zuordnen. Dieses Muster kann einmal geordnet sein, dafür verwendet man die Einstellung "Ordered". Oder aber das Muster soll mehr in Richtung einer Rasterung, wie sie auch bei Fotos betrieben wird, gehen, dann wählt man dafür "Halftone" aus. Als letzte Möglichkeit bleibt

noch "F-S". Dahinter verbirgt sich das "Floyd-Steinberg"-Verfahren, bei dem nicht einfach Punktmuster über die Flächen verteilt werden, sondern unterschiedlich starke Schwingungen. Sie sollten sich einen Ausdruck ruhig einmal ansehen, es ist wirklich sehr interessant und zum Erzeugen von Marmortapeten eines der besten Hilfsmittel!



Scaling

Bisher war die Größe einer Grafik auf dem Papier immer gleich. Egal, welches Format der Screen hatte, das Bild nutzte die ganze Breite des Blattes. Aber warum? Eben! Und deshalb gibt es nun zwei weitere Modi. Der erste "Fraction" stellt den alten Zustand dar, bei dem die Grafik dem Papier in der Größe angepaßt wurde. Mit "Integer" wählt man verschiedenes aus. Dabei spielen auch "Width Limit" und "Height Limit" eine Rolle. Und zwar geben beide Werte an, mit wie vielen Punkten jeder

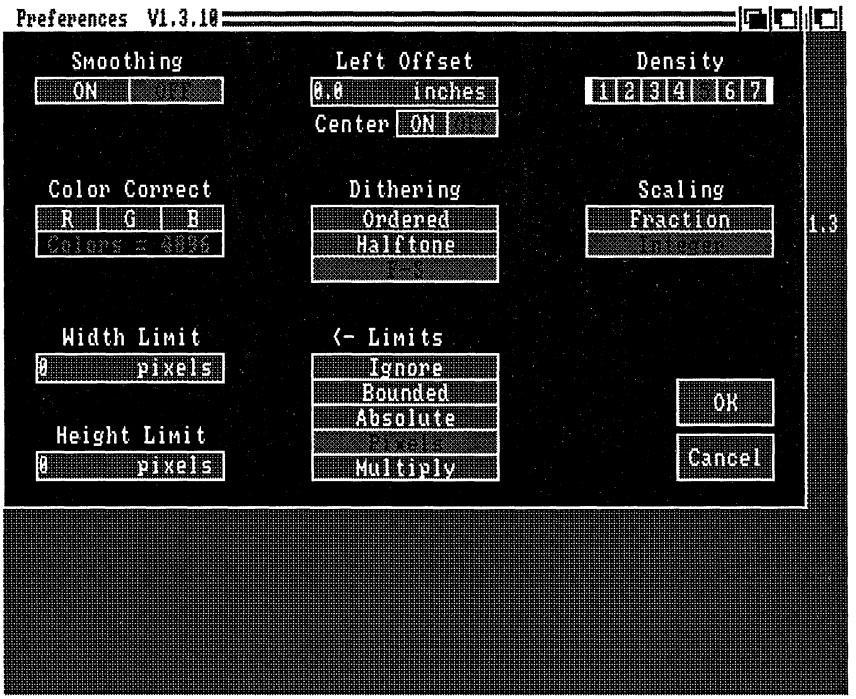
Grafikpunkt auf dem Drucker ausgegeben werden soll. Steht bei beiden Eintragungen eine 1, so erscheint die Grafik beim Ausdruck 1 zu 1. Entsprechend kann die Grafik in jeder Richtung verdoppelt, verdreifacht oder vervielfacht werden. Ein besonderer Wert ist 0! Dann nutzt die Routine nämlich wieder die ganze Größe des Blattes, als wenn "Fraction" eingestellt wäre.

Limits

Als letztes Auswahlfeld haben wir die Einstellungen für die Verwendung der angegebenen Limits. In der Limit-Tabelle kann man aus fünf verschiedenen Verwendungen wählen. "Ignore" macht die Werte unwichtig und läßt die Grafik wieder im normalen Format ausgeben. Unter "Bounded" stellt man die maximal zu erwartenden Maße ein. Das heißt, daß die Grafik z.B. nicht größer als 1 x 2 werden soll. Dann werden 10 und 20 in die Limits eingetragen. Allerdings kann es durchaus sein, daß manche Grafik kleiner ausgegeben wird, weil sie sonst verzerrt wäre. Für eine absolute Größeneinstellung gibt es "Absolute". Dann trägt man die Höhe und Breite in Zehntel-Inches in den Limit-Variablen ein.

Gleiches gilt auch für "Pixels". Hier werden die Werte nur nicht als Zehntel-Inches verstanden, sondern als Druckerpixel. Ein wenig interessanter ist aber die "Multiply"-Einstellung. Hiermit kann man das ausgegebene Bild in jeder Richtung beliebig vervielfachen. Dazu gibt man in die Limits einfach den Faktor ein.

Damit wären alle neuen Einstellungen der Preferences beschrieben. Ich hoffe, Sie haben einen Drucker und können gleich ans Probieren gehen, denn nur die Praxis macht richtig Spaß! Achten Sie darauf, daß Sie beim Kauf eines neuen Druckers einen zu den bisher vorhandenen Druckertreibern Kompatiblen nehmen, damit Sie auch wirklich alles ausnutzen können!



12.2 Die Utilities auf der Workbench-Diskette

Die Workbench beinhaltet neben den Systemdaten auch noch nützliche und hilfreiche Programme. Diese sind in der uns vorliegenden neuen Version 1.3 sogar noch um einiges ergänzt worden. Doch auch das Bestehende wurde nicht auf dem Pfad des Altwerdens gelassen. Neue Parameter und Einstellungen und der häufig unbekannte Weg über das CLI sind hinzugekommen!

Die Clock auch übers CLI

Die Einstellung der Uhr über die Menüs ist zwar komfortabel gelöst, jedoch der Weisheit letzter Schluß ist das auch noch nicht, denn jedesmal, wenn man die Workbench wieder neu bootet, sind alle Einstellungen in ihrem Ausgangszustand. Bietet

es sich dann nicht an, die Uhr über die Startup-Sequence aufzurufen? Denn dort können wir alle Einstellungen beim Aufruf erledigen, was nicht einmal über das .info-File der Workbench erlaubt ist! Dabei können Sie auf die folgenden Flags zurückgreifen:

```
Type: ANALOG, DIGITAL, DIGITAL2
Mode: 12HOUR, 24HOUR
Special: SECONDS, DATE
```

Mit

```
Clock DIGITAL2 24HOUR DATE
```

bekommen Sie während des Arbeitens mit der Workbench immer Informationen über Uhrzeit und Datum!

Mit Cmd zu einer Umleitung

Mit diesem neuartigen Programm ist es möglich, die Ausgabe, die auf ein externes Laufwerk gehen sollte, in ein Datenfile umzuleiten. Dabei schaltet es sich zwischen das Ausgabegerät und die Datenleitung und fängt die Daten ab. Hierbei haben wir als Benutzer einige Möglichkeiten der Einstellung. Dazu übermitteln wir mittels des .info-Files oder der Parameter beim CLI-Befehl die Flags und Daten.

Das wichtigste dabei ist natürlich, welches Gerät "abgehört" werden soll und in welches File die Daten geschrieben werden sollen. Als Device können wir entweder serial oder parallel angeben. Somit ist es z.B. möglich, die Ausgabe auf den Drucker in ein File umzuleiten.

Weiterhin können noch zusätzliche Flags den Vorgang genauer bestimmen. So schreibt das GraphicDump-Programm zu Anfang einen Reset, damit der Drucker in den Ursprungszustand gelangt. Dies ist zwar ganz interessant, doch wird das Programm danach einen Augenblick warten. Und dies geschieht später nicht, wenn wir das File an das Device weiterschicken. Deshalb kann man mit der SKIP-Option (-s) diese Anweisung überlesen und nicht in das File schreiben.

Normal ist es, daß nach dem Schreiben der Daten das Programm Cmd sich selbständig wieder abschaltet. Diese Vorsichtsmaßnahme wurde aus zwei Gründen getroffen. Erstens kann es leicht geschehen, daß das Speichermedium, z.B. die RAM-Disk, irgendwann voll ist, und man bekäme dann eine Meldung! Zweitens will man die Druckerdaten ja trennen können, genau wie man es auch mit dem Papier macht. Also wählt man beim zweiten Mal lieber ein anderes Datenfile an. Hierfür gibt es die MULTIPLE-Option (-m). Sie heben diese durch erneutes Starten von Cmd wieder auf!

Die letzte Option NOTIFY (-n) entscheidet, ob das Programm Cmd Nachrichten an den Benutzer sendet, die ihn über den Zustand der Arbeit informieren. Ich möchte Ihnen diese Option empfehlen, damit man immer auf dem neuesten Stand der Datenübertragung ist, denn so erfährt man vielleicht auch von Datensendungen, die man gar nicht kannte. Sehen Sie hier noch einmal alle Parameter des .info-Files und die des CLIs im Überblick:

DEVICE	Names des Devices (serial/parallel)
FILE	Name des Files, zu dem die Daten gesendet werden
SKIP	Überspringen der Initialisierungsdaten (TRUE/FALSE)
MULTIPLE	Mehrere Datensendungen (TRUE/FALSE)
NOTIFY	Nachrichten an den Benutzer (TRUE/FALSE)

```
Cmd DeviceName FileName [opt s m n]
```

Wenn Sie einmal ganz schnell die Informationen brauchen, können Sie über das CLI mit Cmd Help eine Information aufrufen.

Grafiken nicht nur so, sondern anders

Was viele Anwender des Amiga nicht wissen, sind die Parameter von Graphic-Dump. Eine Grafik muß nicht immer in der Standardgröße auf das Papier kommen. Man kann ohne große Schwierigkeiten die Ausgabe auf vier Größen einstellen: TINY, SMALL, MEDIUM und LARGE, oder aber man wählt die Pixelanzahl!

12.3 Die Utilities auf der Extras-Diskette

Auch auf der Extras-Diskette finden wir einige Tools, die nicht unbeachtet bleiben sollen. Sehen Sie hierzu eine Auswahl:

Mit ClockPtr immer die Uhrzeit

ClockPtr ist eine Workbench-Uhr, die nicht wie die normale Clock ein Window beansprucht. ClockPtr nutzt für die Darstellung der Zeit und des Datums den Mauscursor. Hierfür muß man den blauen Workbench-Hintergrund anklicken, und der Cursor verwandelt sich in eine Uhr, die Stunden und Minuten anzeigt. Möchte man auch die genauen Sekunden wissen, dann bewegt man die Maus in die linke obere Ecke. Befindet sich der Cursor dort, dann werden nur die Minuten mit den Sekunden angezeigt. Als letztes kann man auch noch das Datum erfahren. Hierfür muß man an den linken Rand der Workbench-Screen fahren. Man erhält das Datum in amerikanischer Schreibweise!

Wieviel Speicher? Zwei Möglichkeiten!

Zur Anzeige des freien Speichers gibt es drei Möglichkeiten. Zum einen die Anzeige in der Workbench-Screen. Sie hat allerdings den Nachteil, daß jedes Window diese verdecken kann, und außerdem ist die Anzeige nur aktiv, wenn auch die Workbench aktiviert wird. Deshalb gibt es zwei Utility-Programme auf der Extras-Diskette, die den freien Speicher anzeigen.

FreeMap

FreeMap nutzt für die Anzeige des Speichers eine neue Screen, die man im unteren Bereich hin und her schieben kann. Man sieht hier sowohl Chip- als auch Fast-RAM. Dies ist eine Neuerung, denn bisher vertrug sich das Programm nur mit dem Chip-RAM. Über diese Speicheranzeige ist auch wunderbar erkennbar, wie der Speicher in Stücke gerissen wird, wenn mehrere einzelne Programme laufen. Allerdings ist diese grafische Ausgabe natürlich auch platzintensiver.

PerfMon

PerfMon ist ein Monitorprogramm, das neben dem freien Chip-Speicher auch noch den Fast-RAM-Speicher anzeigt. Als große Besonderheit findet man über den beiden Anzeigen noch eine Kurve für die Prozessorauslastung. Hier ist interessant zu sehen, inwieweit der 68000er ausgenutzt wird oder nicht. Ein Strich zeigt die maximale und damit ideale Auslastung an. Einziger Nachteil des Programms: Es verbraucht über sein Window und den Datenspeicher einiges an Chip-RAM und auch an Textfläche auf der Screen.

Palette

Die Farben der Workbench-Screen sind von Hause aus fertig eingestellt. Es ist aber jederzeit möglich, diese durch das Programm Preferences zu verändern. Doch gilt diese Veränderung mehr für die Ewigkeit als für eine aktuelle Einstellung. Manche Programme, die auf der Workbench arbeiten, kommen mit den Standard-Farben nicht ganz zurecht, und deshalb bietet es sich doch an, die Farben dem Programm anzupassen.

Aber halt! Hier stößt man auf ein Problem. Ist das Programm schon geladen, steht meist wenig Speicher zur Verfügung, der es nicht erlaubt, "mal eben" Preferences zu laden. Dafür gibt es jetzt ein extra Programm. Es heißt Palette und ist klein und handlich. Alle vier Farben können über die bekannten drei Regler für Rot, Grün und Blau eingestellt werden. Das Ergebnis sieht man sogleich im Window und auch auf der Screen.

Als Zusatz bekommt man noch den hexadezimalen Farbwert, der es auch erlaubt, die Farbeinstellung in Zahlen festzuhalten und so neue Farbwerte für Programme einzustellen. Dies ist besonders bei den Befehlen der Graphics-Library interessant, weil hier meistens nur Ausprobieren zum richtigen Ergebnis führt. Jetzt können Sie nach Herzenslust herumprobieren, bis der gewünschte Farbton eingestellt ist. Sie schreiben sich den Wert dazu auf und setzen ihn in die Grafik-Funktion ein.

KeyToy 2000

Das KeyToy ist ein Demo-Programm, mit dem Sie die aktuelle Tastaturbelegung erfragen können. Wozu, werden Sie fragen. Doch nicht immer weiß man genau, wo welches Zeichen zu erreichen ist. Das gilt besonders für ausländische Tastaturbelegungen. Dann lädt man dieses Programm, das individuell nach der Einstellung mit Setmap die Tasten darstellt. Durch Auswahl der Sondertasten wie Shift, Alt und Ctrl ergeben sich die anderen Belegungen, die die oft unbekannten Sonderzeichen enthalten.

Wenn Sie wissen wollen, welche Zeichen bei welcher Tastaturbelegung wo liegen, gehen Sie wie folgt vor:

1. Öffnen Sie ein CLI-Fenster und verlagern Sie die Lage des Gerätes DEVS: von der Workbench-Diskette auf die Extras-Diskette mit

   ```
   1> assign DEVS: Extras:devs
   ```

   Damit können wir die zusätzlichen Tastaturbelegungen über den Befehl Setmap ansprechen.

2. Suchen Sie sich jetzt mit

   ```
   1> dir DEVS:
   ```

   eine der vielen Belegungen heraus. Als Beispiel soll usa0 gelten, da dort am leichtesten der Umtausch von y und z zu erkennen ist.

3. Laden Sie nun KeyToy2000 mit einem Doppelklick und schauen Sie sich die Belegung genau an. Besonders interessant sind die Zeichen bei der Auswahl der Alt-Taste.

4. Tippen Sie nun im CLI

   ```
   1> setmap usa0
   ```

   ein, um die andere Tastaturbelegung zu aktivieren. Jetzt haben wir in dem KeyToy2000-Fenster zwar immer noch die gleiche Belegung, wenn Sie nun aber den nächsten Schritt durchführen ...

5. Laden Sie erneut das KeyToy2000 und schieben Sie das Fenster unter das zuerst geöffnete.

   ... werden Sie merken, welche Unterschiede zwischen den beiden Tastaturen liegen.

13. Sammelsurium der Tips & Tricks

Das lange Arbeiten mit dem Amiga bringt viele Erfahrungen mit sich. In dem Konzept dieses Buches haben wir uns vorgenommen, dem Leser alle diese gesammelten Erfahrungen geordnet und gut verpackt zu vermitteln. Dafür sind nun schon mehr als 13 Kapitel zusammengetragen. Doch unsere Tips & Tricks sind damit nicht erschöpft. Oft hat man Kleinigkeiten, die es nicht lohnen, in einem eigenen Kapitel erörtert zu werden, die aber trotzdem wissenswert und interessant sind.

Dafür haben wir dieses neue Kapitel geschaffen. Es enthält viele kleine Informationen, sozusagen Mini-Tips & Tricks. Doch auch damit läßt sich viel anfangen! Folgen Sie uns jetzt in die Welt der kleinen, aber feinen Erfahrungen, mit denen man oft mehr anfangen kann als mit großen und pompös aufgemachten Nichtigkeiten.

13.1 Tips zum CLI: schneller, bunter, besser

Tips & Tricks Nr. 01 **Mit Backups gerettet**

Beim Arbeiten im CLI passiert es oft, daß man z.B. die Startup-Sequence editiert und durch einen Reset ausprobiert. Dann stellt man plötzlich entgeistert fest, daß das, was man gerade gelöscht hat, lebensnotwendig ist. Doch es ist passiert und weil es viel zu viele Zeilen waren, kann man sie nicht rekonstruieren. Dennoch gibt es einen letzten Ausweg, der bei vielen schon in Vergessenheit geraten ist:

Beide Editoren legen im Verzeichnis T: (für temporary) sogenannte Backup-Files ab, die den Text vor dem Editieren des letzten Files beinhalten. Sie brauchen jetzt dort nur das File "ed-backup" zu kopieren, und schon ist die alte Startup-Sequence wieder hergestellt! Aus diesem Grund sollte man auf keinen Fall

das T:-Verzeichnis im RAM ablegen, denn dort wäre es nach einem Reset gelöscht, und die geplante Hilfe wäre zunichte.

Tips & Tricks Nr. 02 — CLI-Ausgabe unterbrechen

Die Listen- oder Textausgabe im CLI-Fenster läßt sich jederzeit durch Drücken irgendeiner Zeichentaste unterbrechen. Dann wartet der CON-Handler solange, bis dieses Zeichen wieder mit BACKSPACE entfernt wird. Erst dann setzt er die Ausgabe fort. Man kann sich so in Ruhe z.B. mit Space und Backspace eine Liste ansehen und diese immer wieder anhalten!

Tips & Tricks Nr. 03 — Neue Schrifttypen im CLI

Wer hat es nicht schon in einem Vorspann irgendwelcher Public-Domain-Disketten gesehen? Da erscheinen während des Bootens Texte im DOS-Fenster, von denen man sonst nur träumt. Ohne Schwierigkeiten werden die Farben verändert, die Schriften kursiv, fett oder unterstrichen dargestellt. Dies ist alles auch ohne weiteres mit dem ganz normalen Echo-Kommando möglich!

Dazu verwendet man zusätzlich zum gewünschten Text noch einige Steuerzeichen. Diese werden mit einer festgelegten Sequenz eingeleitet. Und zwar ist das "*e[" (Die Besitzer einer Deutschen Tastatur finden die eckige Klammer über der TAB-Taste oder tippen ALT-ü). Die Kombination "*e" kann durch die ESC-Taste ersetzt werden! Nach dieser Einleitung folgen mehrere Zahlen, die durch Semikolon getrennt werden können. So steht z.B. 4 für Unterstreichen und 42 für schwarze Hintergrundfarbe. Alle anderen Werte entnehmen Sie den nachfolgenden Tabellen. Ist die Steuersequenz abgeschlossen und soll Text ausgegeben werden, dann notiert man ein "m" als Abschluß der Steuersequenz.

Schriftarten

Typ	Zahl	Bemerkung
normal	0	Zustand beim Fenster-Öffnen
fett	1	
kursiv	3	Schrägschrift
unterstrichen	4	
invers	7	negativ

Vordergrundfarben

Farbe	Zahl	Bemerkung
normal	30	Einstellung durch Preferences
weiß	31	
schwarz	32	
orange	33	

Hintergrundfarben

Farbe	Zahl	Bemerkung
normal	40	Einstellung durch Preferences
weiß	41	
scharz	42	
orange	43	

Tips & Tricks Nr. 04 Die CTRL-Tastenkombinationen

Viele Kleinigkeiten lassen sich über Tastenkombinationen erledigen. So gibt es auch welche in Verbindung mit CTRL. Doch die meisten Anwender kennen diese nur zum Teil oder gar nicht. Wir wollen hier deshalb nicht darauf verzichten, alle uns bekannten Kombinationen zu erklären. Die ersten vier CTRL-Sequenzen sind den meisten durch das DOS-Buch bekannt. Mit ihnen werden laufende Befehle abgebrochen.

CTRL-C	bricht einen CLI-Befehl ab.
CTRL-D	bricht eine laufende Batchdatei ab.
CTRL-E	führt einen Abbruch höherer Priorität durch.
CTRL-F	führt einen Abbruch höherer Priorität durch.

Alle anderen sind sicherlich von großem Interesse, da sie fast keiner kennt, man sie aber sehr gut gebrauchen kann.

CTRL-H	wie Delete-Taste
CTRL-J	wie Tabulator-Taste
CTRL-K	wie Cursor-up
CTRL-L	löscht das Window (wie ESC-C)
CTRL-M	wir Return-Taste
CTRL-N	aktiviert einen neuen (alternativen) Zeichensatz
CTRL-O	hebt den Zeichensatz wieder auf (gleiches macht auch Return)
CTRL-X	löscht den Inhalt der aktuellen Zeile

Tips & Tricks Nr. 05 — Copy aber mehr!

Zum Kopieren von Files verwendet man im CLI überlicherweise das Copy-Kommando. Allerdings sind hier viele Varianten gar nicht bekannt. Eine Möglichkeit der Ausnutzung bietet sich besonders zum Kopieren mehrerer Files an, die nichts miteinander zu tun haben. Dann kann nämlich nicht über die Joker-Funktion gearbeitet werden. Der Tip ist nun ganz einfach: Es wird ja vom System angeboten, daß es sich aus verschiedenen File-Enden das entsprechende herraussucht:

```
Copy :test.(C|H|O) to DF1:
```

Wir nutzen dies etwas anders, indem wie den ganzen File-Namen als Ende ansehen und so ein Mehrfachkopieren auslösen. Dies erspart das mehrmalige Laden des Copy-Kommandos.

```
Copy C:(DIR|LIST|RENAME) to RAM:C
```

Tips & Tricks Nr. 06 — Neues CLI-Fenster mit Batchdatei

Die Überschrift hört sich wahrlich etwas konfus an. Doch ist sie so zu verstehen, wie man es liest. Wir können nämlich beim Öffnen eines neuen CLI-Fenstes einerseits die Ausmaße und den Titel angeben und andererseits auch noch das CLI anweisen, gleichzeitig eine Batchdatei auszuführen. Das sieht dann so aus:

```
NewCLI Ausgabegerät: Batchdatei
```

Als Ausgabegerät wählt man bei Version 1.2 sicherlich das CON: mit beliebigen Größenangaben. Die Batchdatei kann z.B. Initialisierungsarbeiten übernehmen. Sie können aber auch nur das Vorhaben, eine Datei ausführen zu lassen, dadurch unterstützen, daß Sie diese parallel in einem anderen Fenster laufen lassen. Beides ist möglich.

Für alle stolzen Besitzer einer Workbench 1.3 empfiehlt es sich, das Ausgabegerät als NEWCON: anzugeben, damit auch im neuen Fenster alle Editiereigenschaften vorhanden sind, oder Sie verwenden NewShell ohne Ausgabegerät.

Tips & Tricks Nr. 07 Durchgängiges Design

Die neue Workbench hat ein durchgestylteres Aussehen als die Vorgängerversion. Beim genaueren Betrachten wird man aber enttäuscht. Denn alle alten Devices werden auch mit dem alten Icon dargestellt. So hat die RAM-Disk und auch die neue RAMB0-Disk das platte Symbol. Wenn man aber immer das Symbol übernehmen will, das auch die Workbench-Diskette hat, so reichen zwei kleine Copy-Kommandos in der Startup-Sequence aus. Mit

```
Copy Disk.info to RAM:
```

und

```
Copy Disk.info to CARD:
```

ist schon alles erledigt. Wenn beide Kommandos vor LoadWB stehen, haben Sie nach dem Öffnen der Workbench ein einheitliches Design. Bei alle anderen Disketten läßt sich diese Methode natürlich auch anwenden.

Tips & Tricks Nr. 08 Zweimal CLI in einem Window

Es ist nicht nur erlaubt, zwei CLIs gleichzeitig zu betreiben (über zwei Windows), es ist auch möglich, zwei CLIs in einem Window zu haben. Das geht ganz einfach durch den Aufruf von

NewCLI *. So leitet man die Ausgabe in das schon vorhandene Window um, und es wird kein neues extra geöffnet. Sie bekommen dann immer abwechselnd den ersten und den zweiten Task. Damit erspart man sich die etwas umständliche "Herumwurschtelei" mit den Windows. Außerdem erspart man sich so das ewige Aufrufen der Programme mittels RUN.

Tips & Tricks Nr. 09 — CLI fällt aus dem Rahmen

Es gibt haufenweise Programme, die über komplizierte Wege versuchen, den Rand des CLI-Fensters auszuschalten. Jedoch haben alle bisher vergessen, daß das CLI doch über das Console-Device realisiert wird. Und dort sind schon Befehle und Kommandos vorhanden, mit denen man den Rand einstellen kann. Hier sehen Sie nun eine Aufstellung der benötigten Esc-Sequenzen, mit denen sich die Größe des Fensters ganz beliebig einstellen läßt:

Sequenz	Erklärung
ESC [n u	Setzt die Breite des Windows auf n Zeichen.
ESC [n x	Setzt den linken Rand auf n Pixel.
ESC [n y	Setzt den oberen Rand auf n Pixel.
ESC [n t	Setzt die Anzahl der Zeilen auf n.
ESC c	Setzt alles wieder auf die normalen Wert.

Der Aufruf jeder Sequenz sollte über Echo erfolgen. Dann ist es auch leicht möglich, mehrere Einstellungen über Semikolon miteinander zu verbinden. Sie sollten diesen Echo-Befehl in einer Batchdatei ablegen, so daß man mit einem Aufruf wie Execute AUS den Rand ausschalten kann. Das Gegenstück dazu wäre natürlich Execute AN.

Tips & Tricks Nr. 10 — Schneller mit Type

Oft möchte man in der Startup-Sequence einige Meldungen ausgeben. Da sind zuerst einmal Angaben über Version und Erstellungsdatum. Zum anderen haben wir aber auch Grußnachrichten und Infos zur Konfiguration. Würde man alle Texte über das Echo-Kommando ausgeben, dann müßte der Befehl jedesmal erneut geladen werden. Dies kann man sich ersparen, indem man

den ganzen auszugebenden Text in ein File schreibt, das am Anfang der Batchdatei folgendermaßen aufgerufen wird:

```
Run Type Textfile
```

13.2 Tips zum AmigaBASIC: rauf und runter!

Tips & Tricks Nr. 11 **Schluß mit den Einheits-Icons**

Bei der täglichen Arbeit mit AmigaBASIC stört nach einiger Zeit, daß alle Dateien und alle Programme mit dem gleichen Icon ausgerüstet sind. Nun gibt es zwei Methoden, Abhilfe zu schaffen:

Die erste bezieht sich auf das Programm-Icon. Hier können Sie, nachdem das Programm einigermaßen fertiggestellt ist, ein eigenes über den Icon-Editor kreieren. Dann wird dieses anstatt des vorhandenen mit dem gleichen Namen versehen und in das Verzeichnis kopiert. Damit nun aber, und das ist der Trick, dieses Icon beim nächsten Abspeichern des Programms nicht wieder überschrieben wird, setzt man einfach das DELETE-Flag, und jedes Löschen ist verboten.

Wenn es um Dateien geht, dann ist eine andere Methode einfacher! Diese Dateien sollen bei einem Programm immer ein bestimmtes Aussehen haben, wie z.B. alle Texte des Notepads. Zuerst zeichnet man wieder das gewünschte Symbol mit dem Icon-Editor. Dieses Symbol wird unter irgendeinem Namen in dem gleichen Verzeichnis abgelegt, in dem auch das Hauptprogramm liegt. Dann liest man zu Anfang über die Funktion GetDiskObject& das Icon ein. Bei jedem weiteren Speichern einer Datei wird dieses Icon dann zusätzlich zum Schließen unter gleichem Namen abgespeichert, was bewirkt, daß das AmigaBASIC-Symbol verschwindet und durch unser neues ersetzt wird. Zum Abschluß noch die Programmsegmente zum Einbauen:

Beim Programmstart

```
LIBRARY ":bmaps/icon.library"
DECLARE FUNCTION GetDiskObject& LIBRARY
DECLARE FUNCTION PutDiskObject& LIBRARY
FileName$ = ":BasicIcons/MeinIcon"+CHR$(0)
DiskAdr&  = GetDiskObject&(SADD(FileName$)
Nach dem File-Close:
File$ = File$+CHR$(0)
PutDiskObject&(SADD(File$), DiskAdr&)
```

Tips & Tricks Nr. 12 — Modular arbeiten

Man sollte es sich von Anfang an angewöhnen, in Programmen eine gewisse Modulstruktur zu erreichen, dann wird das Listing nämlich um so einfacher, und außerdem kann man es viel leichter erweitern. Ein weiterer Vorteil liegt darin, daß manche dieser Module in vielen eigenen Programmen eingebaut werden können. Somit erspart man sich eine Menge Arbeit. Sie haben dann zwei Möglichkeiten, nach denen Sie vorgehen können, diesen Teil in andere Listings zu übernehmen.

Die erste ist, Sie laden zweimal den BASIC-Interpreter und legen aus dem Quellprogramm das Modul in das Clipboard ab. Dieses Clipboard kann nun von jedem anderen Programm angesprochen werden, also unter anderen auch von dem zweiten BASIC-Programm. Also gehen Sie jetzt in das zweite Programm und holen mit der Paste-Funktion diesen eben abgelegten Teil wieder in das Listing zurück - und schon ist ein Programmblock umkopiert.

Beim zweiten Verfahren speichern Sie diese Programmblöcke in extra Files ab. Achten Sie dabei darauf, daß Sie das ASCII-Format verwenden, denn sonst ist der beschriebene Weg nicht möglich. Als weiteres können Sie dann in jedem Programm über

```
MERGE Dateiname
```

den neuen Block anbinden. Sie ersparen sich hier eine Menge Arbeit, und diese Methode läßt sich auch anwenden, wenn Sie nur wenig Speicher zur Verfügung haben.

Tips & Tricks Nr. 13 Fenster- und Screen-Name ändern!

In AmigaBASIC kann man zwar durch Öffnen eines neuen Fensters den Namen einstellen, bei schon offenen ist eine Veränderung aber nicht vorgesehen. Dabei bietet die Intuition.Library doch einen Weg. Warum sollten wir ihn dann nicht nutzen? Also öffnen Sie in Ihrem Programm zu Anfang die Library und können dann durch Aufruf von SetWindowTitles() nicht nur den Fensternamen, sondern auch noch den des Screen einstellen. Ein einfaches SUB hilft hier, denn man darf nicht vergessen, jeden String mit einem NULL-Byte abzuschließen!

```
LIBRARY ":bmaps/intuition.library"
SUB SetTitle(WinNam$, ScrNam$) STATIC
  WinNam$ = WinNam$ + CHR$(0)
  ScrNam$ = ScrNam$ + CHR$(0)
  CALL SetWindowTitles(WINDOW(7), SADD(WinNam$), SADD(ScrNam$);
END SUB
```

Tips & Tricks Nr. 14 Mehr Speicher

Im Normalfall benutzt man CLEAR, um im AmigaBASIC mehr Speicher zur Verfügung zu haben. Doch oft streikt dieser Befehl, weil er den gewünschten Speicher nicht beschaffen kann. Dies liegt alleine an einem Fehler des Programms. Es wird nämlich erst versucht, neuen Speicher zu besorgen, erst dann wird der alte wieder freigegeben. Will man also seinen Speicher nur geringfügig ändern, muß man das Doppelte zur Verfügung haben, was nicht oft der Fall ist. Es gibt aber eine sichere Methode, mit der man doch zum Ziel kommt. Dafür setzt man einfach den Bereich auf seine Mindestgröße zurück:

```
CLEAR ,1024
```

Und dann wählt man die gewünschte Zahl:

```
CLEAR ,500000
```

Wenn man sich nicht im Direktmodus befindet, sondern im Programmablauf, ist diese Methode nicht zu empfehlen. Dann arbeitet man lieber mit Sicherheit:

```
CLEAR ,25000-FRE(0)  'nur der benötigte Programmspeicher
CLEAR ,FRE(-1)-50000 'gesamter freie Speicher - Sicherheit
```

Tips & Tricks Nr. 15 BASIC-Programme editieren

Haben Sie sich auch schon so über den BASIC-Editor geärgert? Eigenwillig wie er ist, scrollt er horizontal nur mit Mühe und vertikal nur mit langsamem Tempo. Außerdem kann man nicht nach Befehlen suchen, weil kein FIND existiert. Doch eine Hilfe bietet das AmigaBASIC. Wir können Programme als ASCII-Texte behandeln, und dann ist es das einfachste von der Welt, wenn man ein bestehendes Programm mit SAVE "name",a abspeichert und dann über eine Textverarbeitung weiter editiert. Hier bieten sich z.B. TEXTOMAT oder BECKERtext von DATA BECKER an.

Beide erfüllen viele Voraussetzungen. Sie können nach Begriffen suchen oder diese durch neue ersetzen. Bei BECKERtext kann man sogar über das Lexikon einen Syntax-Check durchführen, ohne AmigaBASIC zu starten. Und wenn das Programm fertig editiert ist, kann es mit dem Interpreter zusammen ausgetestet werden. Für die kleinen Ergänzungen und Verbesserungen reicht das allemal.

Tips & Tricks Nr. 16 Schneller!

Das AmigaBASIC ist zwar von Natur aus nicht gerade langsam, doch könnte es wie jedes Programm noch schneller sein. Dies ist ohne weiteres möglich. Bedenken Sie, daß AmigaBASIC meistens über die Workbench gestartet wird. Einmal nimmt die Workbench als Programm selbst ja einen Task im Betriebssystem in Anspruch. Außerdem sind die Prioritäten nicht gerade optimal eingestellt.

Für dieses Problem bietet das Betriebssystem zwei Funktionen in der Exec.Library. Mit SetTaskPri() können wir die Priorität unseres Tasks, also des BASIC, einstellen. Es läßt sich ein Wert von 1 bis 127 wählen. Je größer man einstellt, desto schneller läuft AmigaBASIC, das heißt desto weniger Zeit bekommen andere

Programme. Besonders geeignet ist diese Methode für zeitaufwendige Berechnungen, die schneller erledigt werden sollen. Gleiches gilt auch für die Grafikausgabe, die damit auf ein maximales Maß gebracht werden kann.

Wie geht man nun dafür vor? Zuerst verschaffen wir uns Zugriff auf die Exec.Library. Dann wird über FindTask() der eigene Task gefunden und mit SetTaskPri() verändert. Zum Schluß ist es wichtig, daß die Priorität wieder zurückgesetzt wird, damit nachher noch andere Programme laufen, die so im schlimmsten Fall ganz unterbunden werden. Sehen Sie abschließend die Programmfragmente:

```
Programmbeginn:
LIBRARY ":bmaps/exec.library"
DECLARE FUNCTION FindTask& LIBRARY
BASICTask& = FindTask&()
CALL SetTaskPri(BASICTask&, 80)
Programmende:
CALL SetTaskPri(BASICTask&, 0)
```

Tips & Tricks Nr. 17 — No overflow in line buffer

Kennen Sie das? Beim Arbeiten im AmigaBASIC-Editor schnappt der Editor über. Es ist nicht mehr möglich, eine Zeile über Backspace hochzuziehen. Immer meldet er sich mit der Fehlermeldung "line buffer overflow". Doch dagegen kann man ganz leicht vorgehen. Dazu müssen wir bloß ein Leerzeichen mit der Maus markieren und mit Amiga-X in den Textpuffer übernehmen. Jetzt arbeitet der Editor wieder korrekt.

Tips & Tricks Nr. 18 — Reset!

Wenn Sie über Ihr BASIC-Programm einen Reset auslösen möchten, brauchen Sie nur CALL 16515072 aufzurufen!

13.3 Ohne Müh' und Not: Tips zum Drucker

Tips & Tricks Nr. 19 Druckerwechsel ohne Preferences

Haben Sie zwei Drucker? Dies kann sehr praktisch sein, wenn Sie sehr oft einerseits Listings ausdrucken müssen und andererseits gut aussehende Dokumente benötigen. Dafür nimmt man am besten einen Matrix- und einen Laserdrucker. Sie haben sich dann bestimmt einen Hardware-Umschalter besorgt. Schließlich wäre ein ständiges Umschalten nicht zumutbar. Hingegen zumutbar ist es, daß man ständig Preferences laden muß, um den anderen Druckertreiber einzustellen. Doch es gibt einen Ausweg aus der Misere!

Die wenigsten wissen, daß die Druckertreiber eigenständige Programme sind, die auch alleine gestartet werden können. Zwar handelt es sich nicht im eigentlichen Sinne um Programme, so daß der Zugriff aus dem CLI nicht erlaubt ist. Jedoch weiß die Workbench mit den Treiber etwas anzufangen. Deshalb sollte man sich schleunigst die benötigten Treiber aus dem Verzeichnis DEVS:Printers in das Hauptverzeichnis der Workbench-Diskette kopieren. Danach besorgt man sich zwei Icons von Typ Tools wie z.B. das Clock- oder CLI-Icon. Geben Sie den Icons die gleichen Namen wie den Druckertreibern. Wenn Sie jetzt eines der Icons von der Workbench aus anklicken, wird der entsprechende Druckertreiber initialisiert!

Tips & Tricks Nr. 20 Drucken über Umwege lohnt sich!

Möchte man in einem BASIC-Programm auch die Funktionen seines Druckers ausnutzen, so wählen die meisten den Weg über die direkte Ansteuerung und nehmen dann PAR: oder SER:. Dies ist aber eigentlich nicht im Sinne des Erfinders. Normalerweise soll jede Ausgabe an einen Drucker über das Printer-Device laufen und somit über PRT:. Doch weil kein Mensch weiß, welche Steuercodes den richtigen Effekt erreichen - Commodore schweigt sich wie immer aus - möchten wir hier Hilfestellung leisten:

Es gibt allgemeine Sequenzen, die den jeweils gewünschten Effekt auf jedem Drucker erreichen. Der entsprechende Druckertreiber setzt die Sequenzen dann in die für den Drucker notwendigen um. Somit ist gewährleistet, daß Druckroutinen für einen Drucker auch auf allen anderen laufen. In der folgenden Tabelle finden Sie die Steuercodes, mit deren Hilfe alle Drucktypen erreicht werden können. Alle Sequenzen werden mit ESC eingeleitet. Hierfür definieren Sie am besten einen String:

```
ESC$=CHR$(27)
```

Schriftart		Sequenz
Kursivschrift	ein:	ESC[3m
	aus:	ESC[23m
Fettdruck	ein:	ESC[1m
	aus:	ESC[22m
Unterstreichen	ein:	ESC[4m
	aus:	ESC[24m
Normalschrift:		ESC[0w
Elite	ein:	ESC[2w
	aus:	ESC[1w
Schmalschrift	ein:	ESC[4w
	aus:	ESC[3w
Breitschrift	ein:	ESC[6w
	aus:	ESC[5w
NLQ	ein:	ESC[2"z
	aus:	ESC[1"z
Proportionalschrift	ein:	ESC[2p
	aus:	ESC[1p
Superscript	ein:	ESC[2v
	aus:	ESC[1v
Subscript	ein:	ESC[4v
	aus:	ESC[3v

Mit LPRINT kann dann eine Steuersequenz zusammen mit dem Text geschickt werden:

```
LPRINT ESC$;"[4m";Text$;ESC$;"[24m"
```

13.4 Tips zur Arbeit mit dem Amiga

Tips & Tricks Nr. 21 **Schneller, schneller, schneller!**

Um den Amiga z.B. während langer Rechenzeiten etwas zu beschleunigen, hilft es schon, wenn Sie alle Screens herunterfahren. Dadurch wird der Grafikprozessor entlastet und somit das ganze System.

Tips & Tricks Nr. 22 **Stop!**

Wenn irgendein Task irgendwo eine Ausgabe tätigt und sie partout nicht mitkommen, dann muß man nicht immer erst das ganze Programm unterbrechen. Ein zeitweiliger Aufschub läßt sich schon durch einen Druck auf die linke Maustaste realisieren. Dann wird nämlich die Menüleiste eingeblendet, wodurch jede andere Ausgabe gestoppt wird. Erst nachdem Sie die Taste wieder loslassen, kann die Textausgabe fortgesetzt werden.

Tips & Tricks Nr. 23 **Kein Ärger mehr mit Disk-Full!**

Haben Sie das auch schon einmal erlebt? Da speichert man so lustig vor sich hin, Daten über Daten, und plötzlich kommt dieser unausstehliche Requester auf den Bildschirm, der uns ohne ein Wimpernzucken mitteilt, daß leider die Diskette voll sei. Wie reagiert man auf so etwas, wenn es sich um sehr wichtige Daten handelt?

Zuerst einmal sollte man die Ruhe bewahren. Dann besorgt man sich einen neuen Zugriff auf das CLI - am besten über das CLI-Icon. Nun sieht man sich Schritt für Schritt die angeblich volle Diskette an. Es gibt nun mehrere Möglichkeiten. Einerseits kann es wirklich sein, daß alle Daten unbedingt gebraucht werden und nichts mehr frei ist. Aber in vielen Fällen sind einige unwichtige Dinge auf der Diskette vorhanden. Was ist z.B. mit der Trashcan und ihrem Verzeichnis? Liegen hier vielleicht Programme, die schon lange weg sollten?

Selbst wenn das Verzeichnis leer ist, sollte es gelöscht werden, denn auch leere Verzeichnisse nehmen Speicher weg. Wenn Sie auf diese Weise die ganze Diskette bearbeiten, werden bestimmt noch einige Blöcke frei. Und dann zeigen Sie dem Requester, was eine Harke ist, und antworten ganz cool mit "Retry"!

Tips & Tricks Nr. 24 — Erste Hilfe vom DiskDoctor

Mir passiert so etwas ständig. Immer wieder löscht man viel zu schnell Programme oder Dateien, die man doch noch brauchte. Nach vielen verzweifelten Stunden weiß ich jetzt einen Ausweg, über den ich diese Files doch wieder an das Tageslicht hole.

Der Amiga beschreitet beim Löschen nämlich den gleichen Weg wie früher der C64. Die Datei wird nicht echt gelöscht und mit Nullen überschrieben. Nur der Eintrag im Directory und in der BitMap der Diskette wird zurückgesetzt. Nur wie bekommt man diese Einträge wieder restauriert? Dazu brauchen Sie nicht einmal einen Diskettenmonitor, denn alles dies erledigt der DiskDoctor! Nach einem Durchlauf sind alle gerade gelöschten Dateien wiederhergestellt. Allerdings darf nach dem Löschen nicht auf die Diskette geschrieben worden sein.

Tips & Tricks Nr. 25 — Ein Guru weniger!

Das ist echt wundersam! Starte ich mein Programm von der Workbench, läuft es anstandslos. Starte ich es vom CLI, schmiert es garantiert nach 5 Sekunden ab. Warum? Tja! Diese Frage stellen sich viele, und hier ist auch gleich die Antwort: Auf einem Multitasking-Computer hat jedes Programm seinen eigenen Stack, auf dem es Rücksprungadressen ablegt. Dieser Stack ist mit einem Wert vorbelegt und kann im Info-File der Workbench eingestellt werden. Auch das CLI bietet diese Möglichkeit. Jedoch merkt man von den verschiedenen Einstellungen auf der Workbench nichts, denn wer guckt schon in alle Info-Files?

Beim CLI ist das anders. Hier wird nicht der Wert des Info-Files übernommen, sondern der des CLI. Und dieser ist ziemlich klein, wenn man ihn nicht vergrößert hat. Startet man ein Pro-

gramm aus dem CLI, so übernimmt es die Stack-Größe. Diese reicht aber nicht aus, und schon stürzt unser gutes Stück ab. Also vergrößern Sie in solchen Fällen den Stack. Die Angaben dazu erfahren Sie über die Workbench-Funktion Info.

Tips & Tricks Nr. 26 **Mehrmals das gleiche**

Wenn man mit vielen Bootdisketten arbeitet, so ist es nicht gerade einfach, überall die gleichen Einstellungen zu machen. Aus Gründen des Geschmacks möchte man überall die Farben gleich, verständlicherweise auch überall den gleichen Drucker und natürlich den gleichen Zeichensatz eingestellt haben. Doch die Arbeit wird mühsam, wenn nicht unmöglich. Schließlich verzweifelt man und gibt das Vorhaben auf. Man merkt es ja spätestens beim Ausdruck, daß auf dieser Diskette noch nicht der richtige Drucker eingestellt ist.

Doch es gibt eine einfachere Lösung. Auf jeder Bootdiskette findet man im Verzeichnis DEVS: die Datei System-Configuration. Hier sind alle Daten abgespeichert, die über Preferences eingestellt werden können. Es ist nichts einfacher, als daß man dieses File auf jede weitere Bootdiskette kopiert. Schon hat man überall die gleichen Einstellungen!

13.5 Verstecktes und Unbekanntes

Tips & Tricks Nr. 27 **Versteckte Grüße im Kickstart**

Wie Sie vielleicht im Workbench-Kapitel gelesen haben, waren die Programmierer des Amiga so mutig, sich in dem Betriebssystem zu verewigen. Aber nicht nur über die Workbench lassen sich Grüße oder Nachrichten abrufen. Auch wenn Sie einmal im Speicher "rumwühlen", werden Sie feststellen, daß dieser nicht nur aus Programmcode besteht.

Folgendes kleine BASIC-Programm macht es möglich, nach der Eingabe von Start- und Endadresse den Text in diesem Bereich sichtbar zu machen.

```
Adressen:
V1.1 16653596 16653672
V1.2 16649670 16649747
INPUT "Startadresse";Start
INPUT "Endadresse  ";Ende
PRINT
FOR i = Start to Ende
   PRINT CHR$(PEEK(i));
NEXT i
```

Ich möchte Sie an dieser Stelle ermuntern, ruhig einmal ein bißchen im Speicher zu stöbern. Vielleicht finden auch Sie so einen oder einen ähnlichen Text. Wenn Sie die Start- und Endadresse eines solchen Textes haben, schreiben Sie uns! Wir würden uns sehr über Ihre Zuschrift freuen. Wenn wir dies dann in das Buch aufnehmen, ist die Namensnennung das Mindeste. Vielleicht winkt bei besonders guten Tips ein gratis "Amiga Tips & Tricks" mit Ihrem Beitrag!

Tips & Tricks Nr. 28 — Die Love Story

Wußten Sie schon, daß Ray mit ...

Aber halt! Das sollen Sie schon selbst erfahren. Es handelt sich hier nämlich um einen versteckten Text im Preferences-Programm der Version 1.2. Dazu müssen Sie eine etwas aufwendige Prozedur durchlaufen, damit Sie in der Window-Titelleiste die gewisse Nachricht erhalten. Doch immer nach der Reihe.

Starten Sie zuerst Preferences. Sie befinden sich jetzt auf der Hauptseite. Rechts sehen Sie die beiden Mausgrafiken. Beide Mäuse haben je zwei Buttons. Fahren Sie jetzt von links nach rechts alle Buttons viermal ab und klicken Sie jedesmal auf die Buttons. Nun haben Sie viermal alle vier Buttons angeklickt. Wenn Sie jetzt das Druckermenü anwählen, müssen Sie nur noch bei der Printer-Wahltabelle das Pfeil-Nach-Oben-Gadget sooft anwählen, bis Sie an den obersten Eintrag gekommen sind. Jetzt noch einmal draufgeklickt, und schon erscheint der Text in der Titelzeile!

Tips & Tricks Nr. 29 Noch eine Geheimnachricht!

Wenn Sie während des Drückens der beiden SHIFT- und ALT-Tasten irgendeine Funktionstaste niederhalten und gleichzeitig noch die Diskette aus irgendeinem Laufwerk ziehen, erscheint eine weitere Meldung auf der Workbench.

Legen Sie die Diskette wieder ein, erhalten Sie entweder wieder die gleiche Meldung oder aber einen anderen Text. Dieser andere Text ist nur noch bei den älteren Versionen zu finden, da er leichte Kritik am Amiga übt. Wenn Sie ihn auch bei neueren Versionen noch sehen wollen, müssen Sie eine Diskette einlegen, deren Icon erst geladen werden muß. Dann vergeht genügend Zeit, so daß der Text nicht gleich wieder überschrieben wird.

14. Arbeiten mit Betriebssystem V1.3

Um es so kurz wie möglich zu sagen: Nichts ist mehr so, wie man es bisher gewohnt war, alles wurde verbessert, und viele Dinge sind neu hinzugekommen. Mit wenigen, ganz speziellen Ausnahmen ist dabei die volle Kompatibilität zur Version 1.2 erhalten geblieben. Die Ausnahmen sind in der Regel Voraussetzungen, von denen bisher stillschweigend ausgegangen werden konnte, so daß beispielsweise der Font der Workbench immer der Topaz-Font war, was heute nicht mehr unbedingt der Fall sein muß. Im Klartext heißt das für den Programmierer, daß er gezwungen ist, alle Datenstrukturen mit den gewünschten Werten zu versehen und nicht, wie es früher üblich war, beispielsweise die Font-Adresse auf 0 zu setzen, ausgehend davon, daß das Betriebssystem dort den Topaz-Font hinterlegt. Jede Pfuscherei bei der Programmierung führt also bei Version 1.3 eher zu unerwünschten Effekten, als dies bei Version 1.2 der Fall war.

Doch Inkompatibilität kann wie gesagt nur in seltenen Fällen auftreten und ist dann durch einfache Änderungen der Voreinstellungen zu beseitigen. Viel interessanter sind natürlich die ganzen neuen Devices, Libraries und Handler, die Verbesserungen an vorhandenen Features und die fast unüberschaubar vielen neuen Möglichkeiten, die sich daraus für den Amiga-Besitzer ergeben. Wahrlich, nützt man jede Möglichkeit und Erweiterung für diesen erstaunlichen Computer, könnten schon heute selbst Großrechenanlagen vom Schlage einer Cray X1 nicht mehr mithalten. Sehen wir uns erst einmal im DEVS-Verzeichnis um und schauen nach, was sich dort getan hat:

14.1 Die Pipeline im Amiga - das PIPE-Device

Aha, ein neues Device also. Um die Vorzüge des Handlers besser zu durchleuchten, schauen wir uns zunächst ein bereits bekanntes Device an, das ClipBoard-Device. Dieses wird häufig auch als die Pin-Wand im Amiga bezeichnet, was gar kein schlechter

Vergleich ist. Wer im CLI beziehungsweise in der neuen Shell Datenaustausch mit anderen Tasks betreiben oder sich nur kurzfristig Daten zur Seite legen wollte, hat meist das ClipBoard-Device verwendet. Nachdem es durch den Befehl MOUNT ins System eingebunden worden war, konnte man durch Aus- und Eingabe-Umlenkung Daten zu diesem Device befördern oder sie sich wieder abholen.

Leider kam es dabei häufig zu Verzögerungen und zu Schreib-Lese-Fehlern der Workbench, die darauf zurückzuführen sind, was dieses Device mit den erhaltenen Daten anstellt. Sehen Sie sich das DEVS-Verzeichnis einmal genauer an, werden Sie darin ein Directory finden, das meist leer ist. Der Name sagt es schon: In dieses Directory schreibt das ClipBoard-Device die erhaltenen Daten und holt sie sich gegebenenfalls wieder zurück. Nun ist es nicht schwer, sich vorzustellen, daß der ständige Diskettenzugriff ganz nett Zeit kostet, zumal die Workbench dabei nicht gerade geschont wird.

Im Zuge der Speicherinflation (der Amiga kann mittels 68020-Prozessor-Erweiterung auf 4.294.967.296 Bytes adressierbaren Hauptspeicher zugreifen - auf 4 GigaByte!) sollten die meist geringen Datenmengen, die bisher via ClipBoard auf Diskette verteilt wurden, auch im RAM verwaltet werden können.

Dies bewerkstelligt das neue Device PIPE. Sie können sich dabei die Wirkungsweise etwa so vorstellen, als ob mitten im RAM eine Pipeline liegen würde, in die auf der einen Seite Daten hineingefüllt werden, die bei Bedarf auf der anderen Seite wieder herausgeholt werden können, eine Art titanisches Schieberegister, könnte man meinen. Um das PIPE-Device nutzen zu können, muß dem System erst einmal mitgeteilt werden, daß der PIPE-Handler aktiviert werden soll. Geben Sie daher ein:

```
MOUNT PIPE:
```

In der DEVS/MountList können Sie vorher je nach gewünschter Pipeline-Größe die Parameter mit dem Editor ED einstellen. Um etwas in die PIPE zu füllen, wird einfach mit dem Aus-

gabe-Umlenk-Befehl des CLI (>) dafür gesorgt, daß anstelle des Console-Devices beziehungsweise des neuen NewCon-Devices der PIPE-Handler die Daten bekommt:

```
DIR >PIPE: SYS:
```

schickt beispielsweise das Root-Directory der Boot-Disk in die Pipeline. Da nach der Eingabe dieses Befehls nichts weiter geschieht, als daß das entsprechende Laufwerk kurz anläuft, wollen wir einmal nachschauen, ob tatsächlich etwas in der Pipeline ist. Folgender Befehl leert die PIPE und gibt den Inhalt auf dem Bildschirm aus:

```
TYPE PIPE:
```

Na, haben Sie schon einmal in der Geschwindigkeit ein Directory im CLI-Fenster erscheinen sehen? Sicher nicht! Sollte die Pipeline einmal zu voll sein, macht sie sich durch eine Fehlermeldung bemerkbar.

14.2 Das Shell hält Vorträge (oder was das SPEAK-Device kann)

Reden konnte der Amiga schon immer und das mit einem für Sprachsynthese lächerlich kleinen Speicherbedarf. Ab Version 1.3 spricht nun auch das CLI. Der Befehl SAY war zwar bereits in Version 1.2 vorhanden, aber hier ist eine Spielerei ganz anderer Qualität gemeint - und wer weiß, vielleicht ist das SPEAK-Device für einen Seh-Behinderten überhaupt keine Spielerei, sondern eine große Hilfe.

Dieses Device kann eine gewisse Verwandtschaft mit dem PIPE-Device nicht verleugnen. So wird es ebenso wie PIPE ins System eingebunden, und Daten werden genauso hineingeschaufelt. Im Unterschied zur PIPE ist das SPEAK-Device allerdings ein Eimer ohne Boden: Was man hineinschüttet, kommt aus den Laut-

sprechern in Form von Sprache wieder heraus. Wir wollen das einmal ausprobieren und uns das System-Directory vorlesen lassen:

```
MOUNT SPEAK:
DIR >SPEAK: SYS:
```

Wer seine Kinder lieber dem Amiga anvertraut als selbst den Mund zu bewegen, oder wer sich gerne Geschichten vorlesen läßt, sollte gleich ganze Textdateien ausgeben lassen:

```
RUN TYPE >SPEAK: "GuteNachtGeschichte"
```

Aber auch für die Sprachsynthesegeschädigten haben wir eine Anwendung:

```
RUN TYPE >SPEAK: ExtrasD:AmigaBASIC OPT H
```

Sinnvoller ist es da, die Abfrage mit ASK hörbar zu machen:

```
ASK >SPEAK: "Do You Like Stefan?"
```

Wer häufig englische Texte im Computer hat und diese Korrekturlesen muß, hat mit SPEAK die Möglichkeit, die Augen zu schonen und nebenbei etwas anderes am Amiga zu machen, denn das Gehör wird schon feststellen, ob Fehler gemacht wurden (unser hochdifferenziertes, computergestähltes Gehirn überliest gerne Schreibfehler, weil es weiß, wie es richtig hätte heißen müssen - SPEAK nicht). Auch alle diejenigen, die den Sinn eines englischen Satzes besser verstehen können, wenn sie ihn hören, könnten dieses Device gebrauchen. Selbst das Englisch-Lernen kann durch die Möglichkeit, riesige Textmengen an SPEAK zu übergeben, gefördert werden.

14.3 Eine Muschel für's CLI - das NewCon-Devi4ce

Die wunderbarste neue Errungenschaft für alle CLI-Enthusiasten und solche, die es werden wollen, ist eindeutig das NewCon-Device, das Sie durch das Anklicken des Shell-Icons arbeiten las-

sen. Dieses Device wird bereits in der Startup-Sequence eingebunden (MOUNT NEWCON:). Die Wirkungsweise besteht zunächst, ähnlich wie bei dem Console-Device, im Öffnen eines Windows. Viel mehr tut sich nicht so ohne weiteres. Geben Sie doch einmal einen Befehl ein und warten mit dem Betätigen der LineFeed-Taste. Drücken Sie nun die linke Cursor-Taste und beobachten Sie genau, was auf dem Monitor geschieht.

Richtig, Sie können jetzt die Befehlszeile jederzeit nachträglich verändern, sprich editieren, ähnlich, wie es im List-Fenster des AmigaBASIC möglich ist. Haben Sie sich einmal bei einer längeren Befehlszeile vertippt, können Sie diesen Fehler korrigieren, ohne daß der Text nochmals eingetippt werden muß. Geben Sie jetzt folgende Befehle der Reihe nach ein:

```
DIR df0: DIRS
LIST ram: OPT A
TYPE s/startup-sequence
```

Nehmen wir an, Sie möchten sich noch einmal die Directories von Laufwerk 0 ansehen. Bisher mußte zu diesem Zweck der Befehl noch einmal eingegeben werden. Heute möchte ich Sie bitten, etwas mit den Cursortasten "auf" und "ab" herumzuspielen. Wie Sie bemerken werden, können Sie durch die Liste der bereits eingegebenen Befehle auf- und abscrollen. Da nur die Befehle gespeichert werden, ist der Speicherbedarf relativ gering. Bereits eingebene Befehle, die Sie sich mit den Cursor-Tasten zurückgeholt haben, können natürlich auch wieder editiert werden.

Nach dem Drücken der Return-Taste werden sie unten an die Liste angehängt. Die Größe der Liste ist natürlich begrenzt, ist aber selbst für CLI-Enthusiasten reichlich dimensioniert, so daß es kaum vorkommen wird, daß ein Befehl oben aus der Liste herausgeschoben und damit gelöscht wird. Shift-Cursor ist eine Möglichkeit, an den Anfang oder das Ende der editierten Befehlszeile zu gelangen. Alle anderen bekannten Tastaturtricks des CLI sind weiterhin implementiert.

14.4 Das FastFileSystem

Wer hat sich nicht bereits über das etwas träge DOS geärgert, das zudem recht verschwenderisch mit dem Speicherplatz auf unseren Disketten umspringt? Seit Version 1.3 gehört das AmigaDOS zu den Veteranen im Umgang mit externen Speichermedien. Grund hierfür ist ein neues Diskettenformat, das alle Erwartungen weit übertrifft.

Um Sie mit dem Umgang mit diesem File-System vertraut zu machen, sehen wir uns zunächst die Unterschiede zum AmigaDOS etwas genauer an. Ein Datenblock einer AmigaDOS-Diskette, in dem Ihre Programmdaten abgelegt werden, besteht aus 512 Bytes. Von diesen 512 Bytes werden nur 488 Byte für Daten verwendet, der Rest geht für Verwaltungsdaten drauf. Wird ein Programm eingelesen, es müssen durch aufwendigen Speichertransfer die Verwaltungsdaten vom übrigen Programm getrennt werden, da dieses schließlich in einem Stück benötigt wird. Hier setzt nun das FastFileSystem an. Es sorgt dafür, daß in einem Datenblock keine Verwaltungsdaten mehr benötigt werden, also alle 512 Bytes für Programmdaten zur Verfügung stehen.

Liest man jetzt mehrere Datenblöcke eines Programms ein, die günstigerweise hintereinander liegen, kann dies mit einem einzigen Lesezugriff geschehen. Da hierbei das Trennen von den Verwaltungsstrukturen entfällt, können diese Blöcke sogar direkt an die gewünschte Speicheradresse gelesen werden. Der Geschwindigkeitszuwachs dabei ist enorm. So können alle Diskettenoperationen auf die fünffache Geschwindigkeit beschleunigt ablaufen. Am interessantesten ist dabei jedoch der Speicherplatzgewinn, der durch eine kompaktere Ablage von Verwaltungsdaten einer Diskette entsteht. Rechnen Sie bei einer 20-MByte-Harddisk, die Sie auf FastFileSystem umgestellt haben, mit 1,5 MByte mehr Speicherplatz.

Die Nutzung des FFS zahlt sich daher im wahrsten Sinne des Wortes aus: Sie erhalten ein Mehr an Diskette für das gleiche Geld. Die Anschaffung der Betriebssystemversion 1.3 macht sich daher in jedem Falle nach kurzer Zeit bezahlt, auch, wenn Sie

Besitzer eines Amiga 500 oder 2000 sind und sich das Betriebssystem auf EPROM brennen lassen, wobei dieser Service im allgemeinen von den Vertreibern sogenannter Kickstart-Umschaltplatinen übernommen wird, da diese dadurch mehr Platinen absetzen können.

Haben Sie Version 1.3, ist das FastFileSystem nicht gleich aktiv. Sie müssen dem System erst mitteilen, daß Sie den Handler gleichen Namens benötigen und auf welches Speichermedium dieser angewandt werden soll. Zu diesem Zweck empfiehlt es sich, durch Eingabe von "ED DEVS/MOUNTLIST" die gewünschten Devices in die MountList einzutragen. Diese können später in der Startup-Sequence mit dem Befehl MOUNT eingebunden werden. Um Sie mit der MountList nicht im Trüben fischen zu lassen, habe ich Ihnen nebst drei Einträgen eine modifizierte Startup-Sequence vorbereitet, durch die Sie automatisch in den Genuß dieser neuen Features kommen können.

14.5 Das FastFileSystem auf Harddisks

Am meisten fällt der Speicherplatzgewinn natürlich auf Harddisks auf, da diese bereits von Hause aus mit dem größten Speicherplatz ausgerüstet sind. Hier ein Vorschlag für einen MountList-Eintrag, der Ihre Harddisk auf das FastFileSystem umstellt. Haben Sie ihn in die MountList integriert, kopieren Sie zunächst alle Dateien Ihrer Harddisk auf normale Disketten, geben den Befehl MOUNT FHD: ein, formatieren unsere "FastHardDisk" mit FORMAT DRIVE FHD: NAME "FastHardDisk" und kopieren Ihre Files auf das neue Device FHD:. Sie werden überrascht sein, wieviel mehr Dateien auf der Harddisk Platz finden.

```
FHD:       Device = hddisk.device     /* Zugriff auf HD */
      FileSystem = L:FastFileSystem   /* Na klar !      */
            Unit = 1                  /* Gerät #0 bleibt
                                         bei AmigaDOS ! */
           Flags = 0                  /* für OpenDevice */
        Surfaces = 4                  /* Diskettenseiten*/
  BlocksPerTrack = 15                 /* Je nachdem...  */
        Reserved = 2                  /* Bootblocks     */
```

```
        Interleave = 0              /* Blockorganisation*/
            LowCyl = 10             /* Ab Zylinder 10 */
           HighCyl = 800            /* Bis Zyl. 800   */
           Buffers = 11             /* Lesebuffer     */
        BufMemType = 1              /* egal (5=FastRAM)*/
           GlobVec = -1             /* Kein GlobVec   */
             Mount = 1              /* Handler sofort bei
                                       Eingabe von MOUNT
                                       laden          */
           DosType = 0x444F5301     /* Erkennungs-Code
                                       für FFS        */
#                                   /* Ende d.Eintrages*/
```

In die Startup-Sequence muß nun der Eintrag:

```
MOUNT FHD:
```

14.6 FastFileSystem auf der resetfesten RAM-Disk

Natürlich kann auch die RAM-Disk RAD: mit dem FastFileSystem arbeiten, ist dann jedoch nicht mehr resetfest. Bei einem Reset bleibt Ihnen diese Disk zwar erhalten, die Daten sind allerdings weg. Doch allein Speicherplatzersparnis und Zugriffsbeschleunigung sollten für viele Anwendungen - gerade im professionellen Bereich - die Installation des FFS rechtfertigen, zumal jederzeit die Möglichkeit besteht, sich weitere RAM-Disks "zurechtzuschustern". Hierbei braucht neben dem Namen nur die Unit-Nummer der RAM-Disk geändert zu werden. Unit 0 ist im übrigen die normale RAM-Disk. Wollen Sie Überschneidungen vermeiden, sollten Sie die nächsthöheren Nummern verwenden. Der entsprechende MountList-Eintrag sieht so aus:

```
FRAD: Device = ramdrive.device
      FileSystem = L:FastFileSystem
      Unit = 1 /* je nach Anzahl der RAM-Disks */
      Flags = 0
      Surfaces = 2
      BlocksPerTrack = 11
      Reserved = 2
      Interleave = 0
      Mount = 1
      LowCyl = 0
      DosType = 0x444F5301
```

```
        HighCyl = 79 /* Je nach vorhandenem RAM (512K->5)*/
        Buffers = 22 /* dito */
        BufMemType = 1 /* egal, wo sie liegt */
#
```

In die Startup-Sequence können Sie folgendes einfügen:

```
MOUNT FRAD:
FORMAT >NIL: <NIL: DRIVE FRAD: NAME "FFS im RAM"
```

Die beiden Umlenkbefehle auf das Nirwana-Device NIL sorgen dafür, daß Sie erstens keine Ausgabe erhalten und zweitens nicht mit der Aufforderung konfrontiert werden, eine Diskette "in's RAM einzulegen", worauf Sie mit dem Drücken der Return-Taste zur Lüge gezwungen wären (wir lassen lieber NIL für uns schwindeln...). Arbeiten Sie mit Programmen, bei denen Sie durch Anklicken nur zwischen DF0 bis DF3 wählen können (CLImate), empfiehlt es sich, den Namen in der MountList mit beispielsweise DF3 anzugeben. Ebenso muß dann natürlich in der Startup-Sequence DF3 eingebunden werden, und zwar anstelle von FRAD.

14.7 FastFileSystem auf normalen Disketten

Da das FFS zur Zeit noch nicht ins AmigaDOS eingebunden ist, kann es hier zu einigen kleinen Schwierigkeiten kommen, die ich später ausführlich erläutern werde. Zunächst wenden wir uns wieder der MountList zu, um Devices mit FFS-Betrieb zu erstellen:

```
DF2: Device = trackdisk.device
     FileSystem = L:FastFileSystem
     Unit = 0                          /* FFS für DF0: */
     Flags = 0
     Surfaces = 2
     BlocksPerTrack = 11
     Reserved = 2
     Interleave = 0
     LowCyl = 0
     HighCyl = 79
     Buffers = 11
```

```
    BufMemType = 1
    Mount = 1
    GlobVec = -1
    DosType = 0x444F5301
#
```

In die Startup-Sequence ist nun "MOUNT DF2:" einzufügen. Dummerweise tritt jetzt folgendes auf: Legen Sie eine mit "FORMAT DRIVE DF2: NAME SMmagic FFS" formatierte Diskette in Laufwerk 0, geschieht nichts, außer daß auf der Workbench die Anzeige "DF0:BAD" erscheint. Gehen Sie jetzt aber ins Shell und tippen "Diskchange DF2:" ein, erscheint auf der Workbench Ihre FFS-Diskette genauso, wie dies bei einer normalen Disk der Fall ist. Durch den Workbench-Menüpunkt INITIALIZE können Sie sie automatisch wieder im FFS-Format überformatieren.

Ebenso funktioniert alles andere wie gewohnt, so daß es sich lohnt, im FFS-Format zu arbeiten, was Ihnen im Schnitt über 40.000 Byte mehr Diskettenkapazität beschert. Entfernen Sie die Disk, verschwindet die Anzeige "DF0:BAD", das Icon der FFS-Diskette jedoch bleibt. Was hiermit eindeutig dargelegt wurde, ist, daß das FastFileSystem keine Diskettenwechsel registriert. Der Grund hierfür ist der, daß es eigentlich nur für Harddisks entwickelt worden ist, die bekanntlich nicht gewechselt werden können. Um nun nicht ständig das Shell laden und mit dem FFS "Diskchange" mitteilen zu müssen, daß die Disk gewechselt wurde, legen Sie sich mit ED eine Datei an, die den Befehl enthält:

```
DISKCHANGE DF2:
```

Kopieren Sie nun das Shell-Icon unter den Namen der Datei:

```
Copy Sys:Shell.info to "Sys:c/DiskwechselDF2"
```

Via Info-Punkt der Workbench tragen Sie in das Default-Tool-Feld "RAM:XICON" ein und löschen die übrigen Einträge. In die Startup-Sequence kommt jetzt zusätzlich:

```
RESIDENT SYS:C/DISKCHANGE PURE
COPY SYS:c/(XICON|DiskwechselDF2) TO RAM:
```

Auf diese Weise befindet sich nun in der RAM-Disk ein Icon, das nach dem Anklicken für die Ausführung unserer Batch-Datei sorgt. Das ständige Laden von der Workbench ist so nicht mehr nötig. Um noch besser arbeiten zu können, sollten Sie das Icon nach dem Hochfahren des Systems aus der RAM-Disk direkt auf die Workbench verlegen, wo es sich immer schnell anklicken läßt, ohne dauernd das RAM-Disk-Window offen halten zu müssen.

14.8 Die neuen MatheLibrarys

Wer hat nicht schon einmal von einem 68030-Prozessor mit 68882-Fließkommaarithmetik-Coprozessor geträumt? Aber bei DEN Preisen, die diese Turbo-Karten haben... Schnellere Berechnungen sind jetzt mit Version 1.3 jedoch auch mit dem 68000 zu haben. Neue Mathematik-Libraries sorgen für den nötigen Schwung - und obwohl diese für die volle Unterstützung der Fließkomma-Coprozessoren 6888x ausgelegt sind, fiel mir beim Ausprobieren mit dem 68000 vor Staunen die obligatorische Kaffeetasse aus der Hand.

Die Geschwindigkeit bei der Bearbeitung von doppelt genauen IEEE-Floats, wie BASIC sie mit den x#-Variablen verwendet, ist zigmal schneller geworden. Nach einer genaueren Analyse stellte ich fest, daß ich hier auf die schnellsten bekannten Fließkommaroutinen gestoßen bin, die es zur Zeit auf der Erde gibt. Um Ihnen nicht nur etwas vorzuschwärmen, hier gleich ein Beispiel, wie diese Routinen in BASIC aufgerufen werden können:

```
DECLARE FUNCTION IEEEDPSin# LIBRARY
LIBRARY "mathieeedoubtrans.library" 'Pfadnamen akzeptiert
                                    'BASIC hier nicht mehr.
'eventuell vorher CHDIR [Pfad zu den BMAP-Files] eingeben!
PI#=4*ATN(1)      'Zahl PI genau berechnen (Vollkreis=2*PI)
KREIS#=2*PI#
FOR I%=0 TO 359                     'Gesamten Kreis in Grad
 WINKEL#=KREIS#/I%                  'Winkel auf Basis 2*PI
 HighLong&=PEEKL(VARPTR(WINKEL#))   'Erstes Long des DFloat
 LowLong& =PEEKL(VARPTR(WINKEL#)+4) 'Zweites Long des Float
 SINUS#=IEEEDPSin#(HighLong&,LowLong&) 'Funktion aufrufen
```

```
 PSET(I%,90-INT(SINUS#*50)),1        'Punkt zeichnen
NEXT
FOR I%=0 TO 359                      'Just for Demo
 WINKEL#=KREIS#/I%                   'Mal sehen, wie schnell
 SINUS#=SIN(WINKEL#)                 'BASIC ist...
 PSET(I%,90-INT(SINUS#*50)),2        'andere Farbe
NEXT
LIBRARY CLOSE
```

Was man bei der ersten FOR-NEXT-Schleife erwarten muß, ist eine erheblich langsamere Ausführung als in der zweiten Schleife. Schließlich muß 2*360=720mal die VARPTR-Funktion aufgerufen werden, 720mal das langsame PEEKL, 360mal die Addition des Wertes 4 und 360mal der Routinen-Aufruf mit Übergabe von zwei Long-Werten, was auch nicht eben schnell geht. Mehr Zeilen und Variablenzuweisungen verzögern die erste Schleife zusätzlich. Bei all diesen genannten Einschränkungen heißt es jetzt staunen: Die erste Schleife ist genauso schnell wie die zweite! Es ist kaum zu ermessen, welch unglaubliche Geschwindigkeit erst in Maschinensprache erzielt werden kann.

Brandneu ist sie, die MathIEEEdoubtrans-Library, und ermöglicht endlich alle transzendenten Funktionen mit doppelt genauen Floats. Aber auch die MathIEEEdoubbas-Library, die die einfachen Rechenfunktionen enthält, ist derart schnell geworden, daß man bei Demonstrationen immer herunterhängenden Kinnladen begegnen wird. Gerade für den BASIC-Anwender sind die neuen transzendenten Funktionen interessant, da hier endlich einmal die Möglichkeit besteht, beispielsweise den Arcus-Sinus ohne gewaltige Umrechnungs-Programme zu ermitteln, und das mit der gleichen Geschwindigkeit wie beim Aufruf der SIN-Funktion. Daher an dieser Stelle eine Übersicht über die Funktionen:

MathIeeeDoubBas-Library

x,y,Double			doppeltgenaue Fließkommazahl (BASIC: anstelle x und y je 2 Longs)
Long			vorzeichenbehaftete Long-Integer-Zahl
Long (D0)	IEEEDPFix -30	(x) (D0/D1)	Umwandlung von Double-Float in Long-Integer.

Double (D0/D1)	IEEEDPFlt -36	(Long) (D0)	Umwandlung von Long-Integer in Double-Float.
Long (D0)	IEEEDPCmp -42	(x,y) (D0/D1,D2/D3)	Vergleicht x und y. cc für Bcc wird gesetzt. Es gilt: x > y -->1 x=y -->0 x<y -->-1
Long (D0)	IEEEDPTst -48	(x) (D0/D1)	Vergleicht x und 0. cc wird gesetzt. Ergebnis wie bei IEEEDPCmp wenn dort y=0
Double (D0/D1)	IEEEDPAbs -54	(x) (D0/D1)	Ermittelt den Absolutwert von x.
Double (D0/D1)	IEEEDPNeg -60	(x) (D0/D1)	Funktion: Double=-x
Double (D0/D1)	IEEEDPAdd -66	(x,y) (D0/D1,D2/D3)	Funktion: Double=x+y
Double (D0/D1)	IEEEDPSub -72	(x,y) (D0/D1,D2/D3)	Funktion: Double=x-y
Double (D0/D1)	IEEEDPMul -78	(x,y) (D0/D1,D2/D3)	Funktion: Double=x*y
Double (D0/D1)	IEEEDPDiv -84	(x,y) (D0/D1,D2/D3)	Funktion: Double=x/y
Double (D0/D1)	IEEEDPFloor -90	(x) (D0/D1)	Ermittelt größte Ganzzahl, die kleiner oder gleich x ist.
Double (D0/D1)	IEEEDPCeil -96	(x) (D0/D1)	Ermittelt kleinste Ganzzahl, die größer oder gleich x ist

MathIeeeDoubTrans-Library

Double (D0/D1)	IEEEDPAtan -30	(x) (D0/D1)	Ermittelt Arcus-Tangens von x
Double (D0/D1)	IEEEDPSin -36	(x) (D0/D1)	Ermittelt den Sinus von x
Double (D0/D1)	IEEEDPCos -42	(x) (D0/D1)	Ermittelt den Cosinus von x
Double (D0/D1)	IEEEDPTan -48	(x) (D0/D1)	Ermittelt den Tangens von x
Double (D0/D1)	IEEEDPSincos -54	(x,VARPTR) (D0/D1,A0)	Doppelrechnung: Sinus von x errechnen, Cosinus von x an Adresse VARPTR legen
Double (D0/D1)	IEEEDPSinh -60	(x) (D0/D1)	Ermittelt Sinus hyperbolicus von x
Double (D0/D1)	IEEEDPCosh -66	(x) (D0/D1)	Ermittelt Cosinus hyperbolicus von x
Double (D0/D1)	IEEEDPTanh -72	(x) (D0/D1)	Ermittelt Tangens hyperbolicus von x

Double	IEEEDPExp	(x)	Exponentialfunktion zur Basis e.
(D0/D1)	-78	(D0/D1)	Funktion: Double=e^x
Double	IEEEDPLog	(x)	Ermittelt nat. Logarithmus von x
(D0/D1)	-84	(D0/D1)	
Double	IEEEDPPow	(x,y)	Funktion: Double=x^y
(D0/D1)	-90	(D0/D1,D2/D3)	
Double	IEEEDPSqrt	(x)	Zieht die Quadratwurzel aus x
(D0/D1)	-96	(D0/D1)	
Float	IEEEDPTieee	(x)	Rechnet x in einfachgenaue
(D0)	-102	(D0/D1)	IEEE-Fließkommazahl um
Double	IEEEDPFieee	(Float)	Rechnet einfachgenaue Float in
(D0/D1)	-108	(D0)	doppeltgenaue um
Double	IEEEDPAsin	(x)	Ermittelt den Arcus-Sinus von x
(D0/D1)	-114	(D0/D1)	
Double	IEEEDPAcos	(x)	Ermittelt Arcus-Cosinus von x
(D0/D1)	-120	(D0/D1)	
Double	IEEEDPLog10	(x)	Ermittelt den Logarithmus zur
(D0/D1)	-126	(D0/D1)	Basis 10 von x

Für die BASIC-Programmierer sei hier nochmals darauf hingewiesen, daß die Übergabe einer Double-Float an eine Systemroutine nicht so ohne weiteres möglich ist. Es muß diese 64 Bit breite Variable in Form zweier Longs übergeben werden, die ihre Werte via PEEKL(VARPTR()) beziehen. Wie dies zu geschehen hat, ist in der Sinus-Demo dieses Kapitels beschrieben. Eine weitere unangenehme Eigenschaft von BASIC ist der nur sehr kurz gestattete Library-Name beim gleichnamigen Befehl.

So ist es zwar möglich, bei den meisten Libraries auch längere Pfadnamen anzugeben, hier jedoch ist bereits der eigentliche Name der Datei so lang, daß kein Pfad mehr angegeben werden kann. Tut man es dennoch, folgt sofort ein File-not-found-Error. Will man daher an die Mathe-Libraries heran, haben im aktuellen Verzeichnis oder im LIBS-Verzeichnis der Systemstart-Disk die entsprechenden BMAP-Files zu liegen. Im Ernstfall hilft man sich mit einem CHDIR vor dem LIBRARY-Befehl.

15. Betriebssystem-Erweiterungen

Das Betriebssystem des Amiga ist sehr flexibel aufgebaut. Fast alle seine Ein- und Ausgabe-Schnittstellen lassen sich ergänzen, erweitern oder verändern. So ist es einem versierten Programmierer möglich, neue Devices zu programmieren und in das Betriebssystem einzubinden, eigene Libraries zu kreieren und diese in Programmen zu benutzen und bestehende Devices oder Libraries zu ergänzen. Dies alles erfordert allerdings viel Arbeit.

Noch einfacher hingegen ist es, sich für die eigene Programmierung neue Hilfsfunktionen zu programmieren, mit denen man dann komplizierte Aufgaben leicht und bequem erledigen kann. In der Sprache C ist dies ohne großen Aufwand möglich und läßt sich beliebig weit ergänzen.

Dazu programmiert man einfach die benötigten Funktionen in einem separaten Programm-File und fügt sie später beim Link-Vorgang mit dem Hauptprogramm zusammen. Dazu werden diese Hilfs-Funktionen getrennt von den anderen mit dem Compiler und Assembler bearbeitet und stehen dann als sog. Objekt-Files zur Verfügung.

15.1 Lade- und Speicherfunktionen mit Pack-Algorithmus

Wir wollen als Beispiel für die einfache Ergänzung des Programmier-Wortschatzes einige Funktionen vorstellen, die wie die bekannten Lade- und Speicherfunktionen zum Datentransfer der DOS-Library arbeiten. Unsere Funktionen werden nur den zusätzlichen Vorteil haben, daß sie beim Speichern und Laden die Daten quasi nebenher packen werden. Dieser Vorgang durchsucht den Datenpuffer vor dem Abspeichern auf Diskette noch nach einer Möglichkeit, die anfallende Menge an Bytes zu kürzen. Beim Ladevorgang wird diese Kürzung, die unter anderem

Diskettenplatz spart, wieder rückgängig gemacht, so daß der Programmierer bei Verwendung unserer Funktionen überhaupt nichts davon merkt.

Dies ist auch der Ansatzpunkt für die Paramter der Funktionen. Da man normalerweise mit denen der DOS-Library arbeitet, benutzen die unseren gleiche Werte. Zum Öffnen wird auch hier der Name des Files und die Zugriffsart übergeben. Zurück erhalten wir auch hier ein Handle. Dieses mußte allerdings erweitert werden. Nun heißt die Datenstruktur nicht mehr Handle, sondern PackHandle, dies verändert die Handhabung aber in keiner Weise.

DOS:

```
Handle = Open(name, access);
```

Eigene Funktion:

```
pHandle = pOpen(name, access);
```

Die Funktionen zur Datenübertragung lehnen sich im Namen an die DOS-Funktionen an, werden nur im Namen durch das vorangestellte "p" gekennzeichnet. Es gibt eine Funktion zum Schreiben eines Pufferinhalts bestimmter Größe und eine zum Lesen von Daten gewünschter Anzahl in einen dafür bereitgestellten Puffer:

DOS:

```
Anzahl = Write(Handle, Puffer, Länge);
Anzahl = Read(Handle, Puffer, Länge);
```

Eigene Funktionen:

```
Anzahl = pWrite(pHandle, Puffer, Länge);
Anzahl = pRead(pHandle, Puffer, Länge);
```

Sowohl die DOS- als auch die eigenen Funktionen liefern beim Lese- und beim Schreibzugriff die Anzahl der wirklich gelesenen oder wirklich geschriebenen Bytes zurück. Dies kann beim

Lesen in beiden Fällen divergieren, wenn das File nicht mehr genügend Bytes enthält. Beim Schreiben ist das Verhalten der neuen Funktion grundsätzlich anders. Sie werden bei der Programmbeschreibung erkennen, woran dies liegt.

Das Packverfahren

Zur Kürzung der auf die Diskette zu schreibenden Datenmenge verwenden wir hier als Beispiel eine sehr einfaches Verfahren. Die Daten, die auf die Diskette oder einen anderen Datenträger geschrieben werden sollen, werden zuerst ist einem Puffer gesammelt, dieser hat die Größe von 512 Bytes und dient dazu, das Verfahren überhaupt praktikabel zu machen. Ist der Puffer voll oder wird das File geschlossen, wird sein Inhalt nicht ohne weiteres abgespeichert. Zuerst wird er nach wiederkehrenden Zeichen durchsucht. Sind mehr als zwei Zeichen hintereinander gleich, kann diese Folge kodiert gespeichert werden. Und zwar verwenden wir dazu ein sog. Kodierbyte - in diesem Fall 255 -, das anzeigt, daß das darauf folgende Byte mehrfach vorkommt. Wie oft, das sagt das dritte Byte. Damit haben wir für die Kodierung folgende Byte-Folge:

```
<Kodierbyte> <Byte> <Anzahl>
```

Da es aber durchaus möglich ist, daß in einem File das Byte 255, das wir als Kodierbyte gewählt haben, vorkommt, muß auch dieses Byte kodiert werden. Das bringt einen entscheidenden Nachteil mit sich: kommen viele solche Bytes vor, wird das File nicht kürzer, sondern länger. Hier ist die dafür einzufügende Byte-Folge:

```
<Kodierbyte> <255> <1>
```

Gemindert wird diese Streckung durch Folgen dieses Bytes, die durch Erhöhen der Anzahl keinen Platz verschlingen. Die Anzahl selbst kann nur einen Bereich von 0 - 255 abdecken. Sollten mehr gleiche Bytes vorkommen, muß die Kodierung wiederholt werden. Dies mag zwar verschwenderisch wirken, ist aber in den meisten Fällen besser, als wenn man zwei Bytes für die Kodierung verwendet. Dann müssen nämlich alle Kodierungen mit 4

Bytes verfaßt werden, was auf diese Art mehr Platz verbraucht. Und dieser Fall kommt sicherlich öfter vor, als daß man zu viele Bytes hat.

Die Datenverwaltung

Zur Verwaltung des benötigten Puffers verwenden wir eine neue Struktur. Sie enthält neben dem Puffer den Zeiger auf die von DOS verwendete FileHandle-Struktur und einige Positionsvariablen, mit der die Schreib- und Lese-Position im Puffer verwaltet wird.

```
struct PackHandle
   {
   char                Puffer[512];
   LONG                Position;
   LONG                Anzahl;
   struct FileHandle *Handle;
   };
```

Die Funktionen

Sehen Sie nun die Funktionen, die zur File-Arbeit verwendet werden können. Neben den Ersatz-Funktionen zum Öffnen, Schließen, Schreiben und Lesen gibt es noch eine zum Kopieren eines Speicherbereichs und die wichtigste von allen, die Pack-Funktion. Sie erledigt die eigentliche Arbeit.

```
/****************************************
*                                       *
*   Entwicklung von Funktionen zum      *
*   Packen oder kodieren von Daten      *
*                                       *
*  Editiert mit dem Editor EDwork       *
*  von Data Becker (Autor Peter Schulz) *
*                                       *
*  (c) by DATA BECKER                   *
*  (p) by Wolf-Gideon Bleek 1989        *
*                                       *
****************************************/

#include <exec/types.h>
#include <exec/memory.h>
#include "packlib.h"

/* #define TEST "Im Falle eines Falles hilft Testen einfach alles" */
```

```
void *malloc();
void Close();
LONG Write(), Read();
struct FileHandle *Open();
void free();

/*#FOLD: Funktion MemCopy */

void MemCopy(Von, Nach, Laenge)
register char *Von, *Nach;
register LONG Laenge;
   {
   while(Laenge--)
      *Nach++ = *Von++;
   }
/*#ENDFD*/

/*#FOLD: Funktion Pack() */

byte Code[] = {MARK, 0, 1, 0, 0};

LONG Pack(pHandle)
struct PackHandle *pHandle;
   {
   struct FileHandle *Handle = pHandle->Handle;
   char              *Puffer = pHandle->Puffer;
   LONG               Laenge = pHandle->Anzahl;

   LONG  Anzahl = NULL;

   LONG  ZaehlerV = NULL;
   LONG  Zaehler  = NULL;
   char  *Zeiger;

   #define Zeichen   Code[1]
   #define ZaehlerG  Code[2]

   while(Laenge--)
      {
      Zeichen = *Puffer++;

      if (*Puffer == Zeichen && ZaehlerV)
         {
         #ifdef TEST
         printf("(%ld)", ZaehlerV);
 /* zum Test Anzahl der unkodierten Bytes */
         #endif
         Zaehler += ZaehlerV;
         ZaehlerV = NULL;
         }

      while(*Puffer == Zeichen && Laenge > NULL)
```

```
        {
        Puffer ++;
        ZaehlerG ++;
        Laenge --;
        }

    if (ZaehlerG>1)
        {
        Zaehler += ZaehlerG;
        if (ZaehlerG <= 3 && Zeichen != MARK) /* Zu wenig Zeichen */
         {
         #ifdef TEST
         printf("<%d>", ZaehlerG); /* bei 3 Zeichen normal speichern */
         #endif
         ZaehlerV = (LONG)ZaehlerG;
         Code[2] = Zeichen;
         Code[3] = Zeichen;
         Anzahl += Write(Handle, &Code[1], ZaehlerV);
         }
        else
         {
         #ifdef TEST
         printf("<MARK, %d, %d>\n", Zeichen, ZaehlerG);
         #endif
         /* Kodierung auf Diskette bringen */
         Anzahl  += Write(Handle, &Code[0], 3L);
         }
        ZaehlerG = 1;
        }
    else
        {
        ZaehlerV ++;
        Anzahl += Write(Handle, &Zeichen, 1L);
        }
    }

if (ZaehlerG)
    {
    Zaehler += ZaehlerG;
    if (ZaehlerG <= 3 && Zeichen != MARK) /* Zu wenig Zeichen */
        {
        #ifdef TEST
        printf("<%d>", ZaehlerG);
        #endif
        ZaehlerV = (LONG)ZaehlerG;
        Code[2] = Zeichen;
        Code[3] = Zeichen;
        Anzahl += Write(Handle, &Code[1], ZaehlerV);
        }
    else
        {
        #ifdef TEST
        printf("<MARK, %d, %d>\n", Zeichen, ZaehlerG);
```

```
         #endif
         Anzahl  += Write(Handle, &Code[0], 3L);
         }
      }
   else
      Zaehler += ZaehlerV;

   #ifdef TEST
   printf("ANZAHL: %ld\n", Zaehler); /* Testausgabe */
   #endif

   return(Anzahl);
   }
/*#ENDFD*/
/*#FOLD: Funktion pOpen() */

struct PackHandle *pOpen(Name, Modus)
char *Name;
LONG  Modus;
   {
   struct FileHandle *Handle;
   struct PackHandle *pHandle;

   Handle = Open(Name, Modus);

   if (Handle != NULL)
      {
      pHandle = (struct PackHandle *)malloc(sizeof(struct PackHandle));
      if (pHandle == NULL)
         {
         Close(Handle);
         return(NULL);
         }
      pHandle->Handle   = Handle;
      pHandle->Position = NULL;
      pHandle->Anzahl   = NULL;
      pHandle->Modus    = Modus;
      }

   return(pHandle);
   }

/*#ENDFD*/
/*#FOLD: Funktion pClose() */

LONG pClose(pHandle)
struct PackHandle *pHandle;
   {
   LONG Anzahl = 0L;

   if (pHandle->Anzahl && pHandle->Modus == MODE_NEWFILE)
      Anzahl = Pack(pHandle);
```

```
   Close(pHandle->Handle);
   free(pHandle);

   return(Anzahl);
   }

/*#ENDFD*/
/*#FOLD: Funktion pWrite() */

LONG pWrite(pHandle, Puffer, Laenge)
struct PackHandle *pHandle;
char              *Puffer;
LONG               Laenge;
   {
   LONG Anzahl = NULL;

   if (pHandle->Modus == MODE_OLDFILE)
      return(NULL); /* Schreibzugriff bei einem Lese-File verweigert */

   while(pHandle->Anzahl+Laenge > 512L)
      {
MemCopy(Puffer,&pHandle->Puffer[pHandle->Position],512L-pHandle->Anzahl)

      Laenge    -= 512L-pHandle->Anzahl; /* noch zu Schreibende Bytes */
      Puffer    += 512L-pHandle->Anzahl; /* Position im \bergabepuffer *
      pHandle->Position = 512L;  /* Zeiger auf volle "Pulle" */
      pHandle->Anzahl   = 512L;

      Anzahl += Pack(pHandle);

      pHandle->Position = NULL;
  /* Durch Schreiben wurde der Puffer leer */
      pHandle->Anzahl   = NULL;
      }

   MemCopy(Puffer, &pHandle->Puffer[pHandle->Position], Laenge);
   pHandle->Position += Laenge;  /* Puffer um Laenge Bytes aufstocken */
   pHandle->Anzahl   += Laenge;

   return(Anzahl);
   }

/*#ENDFD*/
/*#FOLD: Funktion pRead() */

LONG pRead(pHandle, Puffer, Laenge)
struct PackHandle *pHandle;
char              *Puffer;
LONG               Laenge;
   {
   LONG Anzahl = NULL;
```

```
   if (pHandle->Modus == MODE_NEWFILE)
      return(NULL); /* Lesezugriff bei einem Schreib-File verweigert */

   pHandle->Anzahl = Read(pHandle->Handle, &pHandle->Puffer[0], 512L);

   while(Laenge)
      {
      if (pHandle->Puffer[pHandle->Position] != MARK)
         {
         #ifdef TEST
         printf("%c", pHandle->Puffer[pHandle->Position]);
         #endif
         *Puffer++=pHandle->Puffer[pHandle->Position++];
         Anzahl ++;
         Laenge --;
         }
      else
         {
         while(pHandle->Puffer[pHandle->Position+2] && Laenge)
            {
            #ifdef TEST
            printf("%c", pHandle->Puffer[pHandle->Position+1]);
            #endif
            *Puffer++=pHandle->Puffer[pHandle->Position+1];
            Anzahl ++;
            Laenge --;
            pHandle->Puffer[pHandle->Position+2] --;
            }

         if(pHandle->Puffer[pHandle->Position+2] == 0)
            pHandle->Position += 3;
         }

      if (pHandle->Position == (pHandle->Anzahl-1))
         {
         pHandle->Position = NULL;
 pHandle->Anzahl = Read(pHandle->Handle, &pHandle->Puffer[0], 512L);
         if (!pHandle->Anzahl)
            Laenge = NULL;
         }
      }

   return(Anzahl);
   }

/*#ENDFD*/

Listing: Funktionen zum Packen beim Datenspeichern
#include <libraries/dosextens.h>

typedef unsigned char   byte;
```

```
#define PUFFER_SIZE  512L;
#define MARK         (char)0xff

struct PackHandle
   {
   char               Puffer[512];
   LONG               Position;
   LONG               Anzahl;
   struct FileHandle *Handle;
   LONG               Modus;
   };
```

Listing: Include-File mit Struktur-Definition

Programmbeschreibung

Die Funktionen gliedern sich in zwei Teile: Die Hauptfunktionen, die die Fähigkeiten der DOS-Funktionen nachahmen und die Hilfsfunktionen, die die Arbeit ergänzen.

Zu den Hauptfunktionen zählen pOpen(), mit der über die gleichen Parameter wie beim DOS-Befehl ein File geöffnet werden kann, das entweder zum Lesen oder zum Schreiben genutzt werden kann. Diese Funktion verwaltet neben dem Modus noch einen Puffer, in dem die Daten zwischengespeichert werden.

Die korrespondierende Funktion pClose() schließt ein File ordnungsgemäß und gibt den zusätzlich belegten Pufferspeicher wieder frei. Sollte das File zum Schreiben geöffnet gewesen sein, wird vorher noch der Pufferinhalt gesichert.

Wichtig sind aber besonders die Funktion pWrite() und pRead(). Beide arbeiten gepuffert, das heißt, sie lesen die Daten nicht direkt in den angegebenen Speicher, sondern schreiben sie erst in den für jedes File angelegten Puffer. Von dort aus werden sie dann entweder bei Überfüllung geschrieben oder dekodiert übertragen. Beim Schreiben wird die Funktion Pack() aufgerufen, da diese Arbeit etwas komplizierter ist. Die Unterfunktion geht den Puffer Byte für Byte durch und sucht nach Vor-

kommen von mehreren gleichen Bytes. Bei Erfolg wird nur die oben beschriebene Markierung geschrieben, ansonsten gehen die Bytes direkt auf Diskette.

Die Funktion pRead() hat zwar auch Arbeit zu erledigen, kann diese aber einfacher durchführen. Sie geht den Pufferinhalt nach und nach durch und überträgt entweder jedes Byte in den Benutzer-Puffer oder überträgt das kodierte Byte entsprechend der angegebene Anzahl. Ist der Puffer leer, werden erneut Daten von der Diskette gelesen.

Ein Testprogramm

```
/*************************************
*                                    *
*  Programm zum Testen der neuen     *
*  Pack- und Codier-Funktionen       *
*                                    *
*  (c) by DATA BECKER                *
*  (p) by Wolf-Gideon Bleek 1989     *
*                                    *
*************************************/

#include <exec/types.h>
#include "packlib.h"

extern struct PackHandle *pOpen();

#define File      argv[1]
#define SrcFile   argv[1]


main(argc, argv)
int argc;
char *argv[];
   {
   struct FileHandle *Handle, *Open();
   struct PackHandle *pHandle;

   LONG Anzahl, pAnzahl;
   int i;
   char Puffer[520];

   char DestFile[31];

   strcpy(DestFile, File);
   strcat(DestFile, (char *)".pak");
```

```
   printf("Programm zum Testen den Pack-Funktionen\n");
   printf("Die Datei '%s' wird gepackt ...\n", File);

   Handle = Open(SrcFile, MODE_OLDFILE);
   if (Handle == NULL)
      {
      printf("Das File konnte nicht gevffnet werden!\n");
      exit(FALSE);
      }

   pHandle = pOpen(DestFile, MODE_NEWFILE);
   if (pHandle == NULL)
      {
      printf("Das Pack-File konnte nicht gevffnet werden!\n");
      exit(FALSE);
      }

 Anzahl  = Read(Handle, &Puffer[0], 520L); /* Puffer mit Daten füllen */
 printf("Beim Lesen %ld Daten.\n", Anzahl);

 pAnzahl = pWrite(pHandle,&Puffer[0],500L); /* Teil des Daten packen */
 printf("Beim ersten Schreiben %ld Daten.\n", pAnzahl);

pAnzahl = pWrite(pHandle,&Puffer[500],20L); /* restliche Daten packen */
 printf("Beim zweiten Schreiben %ld Daten.\n", pAnzahl);

 Close(Handle);                /* Beide Files schließen */
 pAnzahl = pClose(pHandle);
 printf("Beim Schließen des Files %ld Daten\n", pAnzahl);


   /* Gepacktes File Testen
      -------------------- */

   printf("\nVersuch zum Öffnen des Files ...\n");

   pHandle = pOpen(DestFile, MODE_OLDFILE);
   if (pHandle == NULL)
      {
      printf("Das Pack-File konnte zum Lesen nicht geöffnet werden!\n");
      exit(FALSE);
      }

   Anzahl  = pRead(pHandle, &Puffer[0], 500L);  /* 500 Bytes lesen */
   printf("\nBeim Lesen %ld Daten.\n", Anzahl);
   pClose(pHandle);
```

```
for(i=0; i<Anzahl; i++)    /* gelesene Bytes (max. 500) ausgeben */
   printf("%c", Puffer[i]);

printf("\n");
}
```

Programmbeschreibung

Dieses Programm kodiert die ersten 520 Bytes eines Files. Dazu geben Sie als Parameter beim Programmstart den File-Namen mit den nötigen Verzeichnis Angaben an. Zum Test werden noch einmal die ersten 500 Bytes des gepackten Files gelesen und ausgeben. Dies hat allerdings nur Sinn, wenn es sich um ein Text-File handelt.

Sie können mit diesem Testprogramm ausprobieren, wie effektiv dieser Pack-Algorithmus arbeitet. Die Ausbeute ist sehr unterschiedlich.

Als Anregung möchte ich Ihnen empfehlen, ein Programm zu schreiben, das in der Lage ist, jedes beliebige File vollständig zu komprimieren und eines, das wieder dekodiert. Die Veränderungen am obigen Programm sind dabei nicht schwierig.

Compiler- und Link-Anweisungen

Die hier beschriebene Verfahrensweise gilt für den Aztec-C-Compiler V3.6a, da wir meinen, daß dieser Compiler besonders einfach zu handhaben ist. Er bietet als weiteren Vorteil die Möglichkeit, auch schon mit einem Diskettenlaufwerk und nur 512 KByte RAM alle Arbeiten (wenn auch umständlich und zeitraubend) durchzuführen, während der Lattice-Compiler mindestens zwei Laufwerke oder 1 MByte Speicher benötigt.

Wichtig ist, daß Sie die oben abgedruckten Funktionen nach dem Eintippen mit dem Compiler und dem Assembler zu einem Objekt-File assembliert werden. Es reicht dafür eine einziger Aufruf:

```
cc packfunk.c
```

Wir benötigen keine weiteren Parameter, weil der Compiler automatisch den Assembler aufruft. Auch die Wahl des Speichertyps ist in diesem Fall egal.

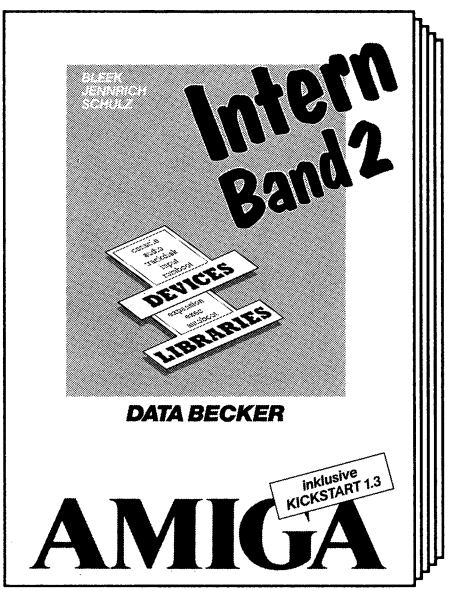
Wollen Sie diese Funktionen [pOpen(), pWrite(), pRead(), pClose()] in einem eigenen Programm verwenden, genügt es, beim Link-Vorgang den unterstrichenen Teil zu ergänzen:

```
ln testprg.o packfunk.o -lc etc.
```

Stichwortverzeichnis

Ein Muß für jeden aktiven Programmierer.

Amiga Intern Band 2 – das Buch für jeden aktiven Programmierer, der alle weiterführenden Informationen zu seiner Arbeit schnell und zuverlässig finden will. Beispielsweise braucht er eine detaillierte Dokumentation aller Library-Funktionen. Eine Dokumentation, die eine sofortige Anwendung für seine Assembler- oder C-Programme garantiert. Amiga Intern Band 2 bietet sie – zu allen bisher ausgelieferten Versionen. Also zu Kickstart 1.1, 1.2 und zur aktuellen Version 1.3! Ebenfalls vermißten viele Amiga-Programmierer die nötigen Informationen zur Parameterübergabe an Programme über CLI und über die Workbench. Amiga Intern Band 2 schließt auch diese Lücke. Dazu natürlich jede Menge zu den Standard-Austausch-Formaten, den Basis-und Grundstrukturen im System und nicht zuletzt zu den verschiedenen Amiga-Devices.



Bleek/Jennrich/Schulz
Amiga Intern Band 2
Hardcover, 895 Seiten, DM 69,-
ISBN 3-89011-268-4

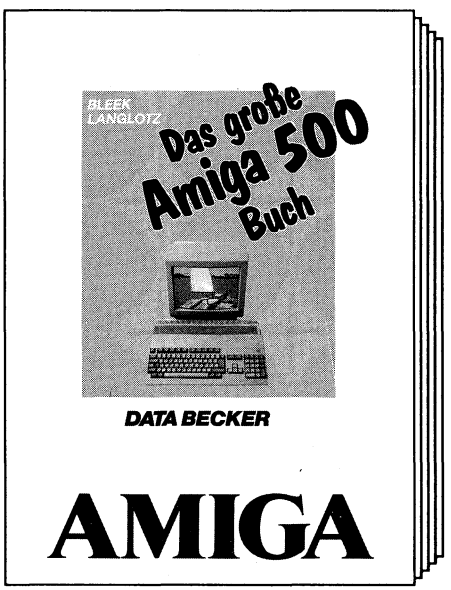
Kompetentes Detailwissen zum Amiga 2000.

Von den glorreichen Drei (Rügheimer/Spanik/Amiga) wurden die Amiga-Anwender schon immer verwöhnt: Ihre Fachbücher strotzen vor fundiertem Know-how und sind dennoch dank einer lockeren Schreibe überaus unterhaltsam. Warum sollte es also beim großen Amiga-2000-Buch anders sein? Mit viel Liebe zum Detail und der gewohnten leichtverständlichen Sprache beschreiben Sie alles, was den Amiga 2000 so interessant macht: die möglichen Speichererweiterungen, Einbau und Einrichtung einer PC-/Amiga-Harddisk, Arbeiten mit einer PC-/AT-Karte, Kickstart im RAM, jede Menge zur Janus.Library 2.0, Software-Installationstips, der Umgang mit dem AmigaDOS und und und. Selbstverständlich auch mit einer detaillierten Einführung zum Amiga 2000 und einigen wichtigen Software-Tips. Das große Amiga-2000-Buch – ein großartiges Buch zu einem großartigen Rechner.

Rügheimer/Spanik
Das große Amiga-2000-Buch
Hardcover, 736 Seiten, DM 59,-
ISBN 3-89011-199-8

So kommt der „Kleine" ganz groß raus.

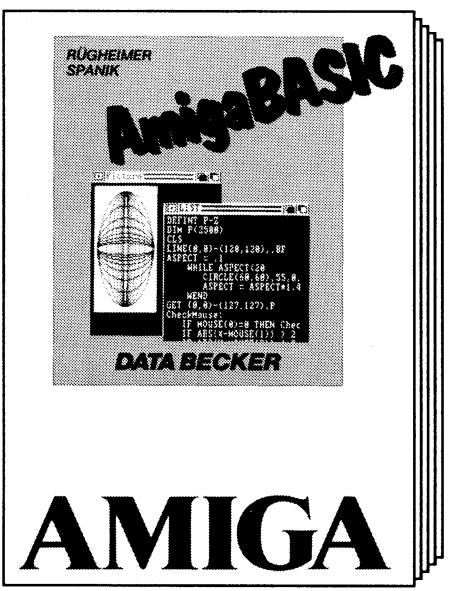
Zum Spielen alleine ist er zu schade: Lernen Sie Ihren Amiga 500 von seiner professionellen Seite kennen. Das große Amiga-500-Buch vermittelt Ihnen nicht nur den Umgang mit CLI und Workbench, sondern enthält auch ausführliche Kapitel zur Systemprogrammierung, zum professionellen Einsatz von Software und zu "Umbauten", die Sie selbst ausführen können. Ob Sie Einsteiger oder Profi sind: In diesem Buch finden Sie alle wichtigen Informationen rund um den Amiga 500 - etwa zum Virenschutz, zum Soundsampling, zu Hardware-Erweiterungen (z.B. neuer Prozessor, Harddisk, PC-Karte, Profigehäuse), zu Kickstart 1.3 und zu vielen Themen mehr. Außerdem gibt es natürlich jede Menge Tips und Tricks.



Bleek/Langlotz
Das große Amiga-500-Buch
Hardcover, 527 Seiten, DM 49,-
ISBN: 3-89011-279-X

Jede Menge Programme und Utilities.

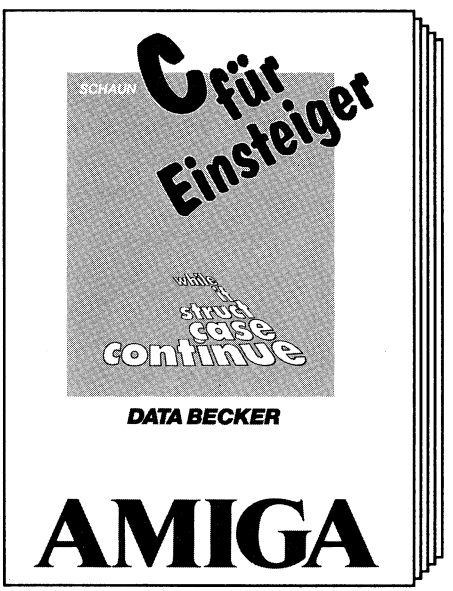
Fast 800 Seiten über AmigaBASIC - von Fans (das bekannte Duo Rügheimer/Spanik) für Fans. Im ersten Teil werden Sie Schritt für Schritt - und das vor allem auf verständliche Weise - in die Programmierung des Amiga eingeführt, im zweiten Teil finden Sie alle gelernten Befehle mit Syntax und Parameterangaben zum schnellen Nachschlagen. Dazu gibt es Programme und Utilities in Hülle und Fülle: ein Videotitel-Programm (OBJECT-Animation), ein Balken- und Tortengrafik-Programm, ein Malprogramm (mit Windows, Pulldowns, Mausbefehlen, Füllmustern und dem Einlesen sowie Abspeichern von IFF-Bildern), ein Statistikdaten-Programm, ein Sprach-Utility, ein Synthesizer-Programm u.v.a.m.



Rügheimer/Spanik
AmigaBASIC
Hardcover, 777 Seiten
inkl. Diskette, DM 59,-
ISBN: 3-89011-209-X

Programmieren in C – nicht nur für Profis.

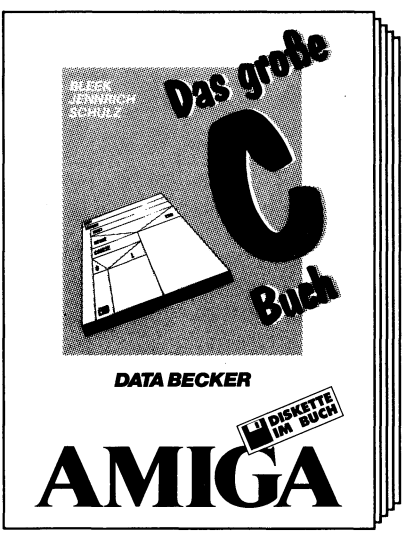
Wie einfach es auch für einen absoluten Computer-Neuling sein kann, die „Profi"-Sprache C innerhalb kürzester Zeit zu beherrschen, zeigt Ihnen das Buch „C für Einsteiger". Bereits nach einem Wochende sind Ihnen die wichtigsten Grundlagen dieser Sprache vertraut. Schnell und leichtverständlich erfahren Sie hier alles zur Programmiertechnik – vom ersten Programm bis hin zu den Routinen und den Bibliotheken. Mit dem gesamten Sprachumfang und den besonderen Features von C. Zahlreiche Übungsbeispiele veranschaulichen das Gelernte und sorgen für die nötige praktische Erfahrung. Selbstverständlich lernen Sie dabei auch die wichtigen Tips und Tricks zur Programmierung kennen. Eine detaillierte Beschreibung der beiden Compiler Lattice C und Aztek C runden das Ganze ab. C für Einsteiger – ein Einführungskurs, wie ein Neuling ihn sich nur wünschen kann.



Schaun
Amiga C für Einsteiger
272 Seiten, DM 39,-
ISBN 3-89011-107-6

Für alle, die Spaß an C gefunden haben.

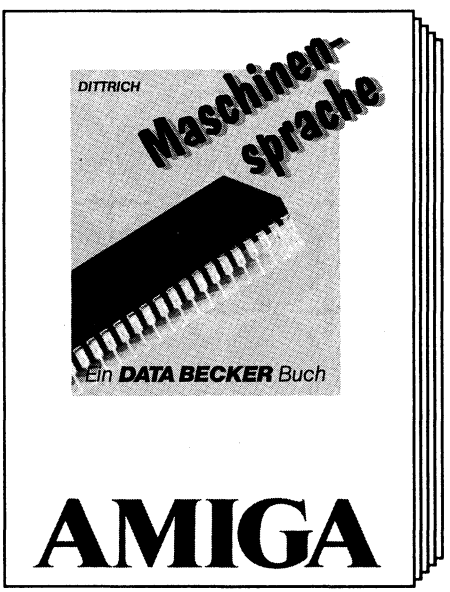
Das große C-Buch zum Amiga – ein Buch für alle engagierten Amiga-Anwender, die Spaß an C gefunden haben und nun darauf brennen, eigene, professionelle Programme zu schreiben. Hier erfahren Sie, wie ein C-Compiler arbeitet und wie Sie selbst schwierigste Probleme in C lösen. Ein kleiner Blick ins Inhaltsverzeichnis macht deutlich: Hier finden Sie das Know-how für eine optimale C-Programmierung – Funktionsweise des Aztec-Compilers, Debugging und Optimierung des Assembler-Sources, Sprungtabellen und dynamische Arrays in C, Einbinden von Assembler-Source in den C-Source, alles Wissenswerte zur Intuition-Programmierung und natürlich eine detaillierte Beschreibung der Folder-Technik. Wer mit diesem Buch arbeitet, dem werden in Zukunft bei der Programmierung höchstens noch Tippfehler unterlaufen.



Bleek/Jennrich/Schulz
Das große C-Buch zum Amiga
Hardcover, 777 Seiten
inklusive Diskette, DM 69,-
ISBN 3-89011-191-2

Maschinen-sprache auf dem Amiga.

Schreiben Sie Ihre Programme in Maschinensprache – und Sie werden sehen, was Ihr Amiga auch in Sachen Geschwindigkeit alles draufhat. Das nötige Know-how für eine schnelle, perfekte Assembler-Programmierung finden Sie in diesem Buch: Grundlagen des 68000, das Amiga-Betriebssystem, Drukkeransteuerung, Diskettenoperationen, Sprachausgabe, Windows, Screens, Register, Pull-Down-Menüs... Natürlich wird auch gleich gezeigt, was die wichtigsten Assembler leisten und wie Sie damit arbeiten. So werden SEKA, PROFMAT und ASEEM detailliert und praxisnah beschrieben. Amiga Maschinensprache – der beste Weg zu schnellen, professionellen Assemblerprogrammen.



Dittrich
Amiga – Maschinensprache
Hardcover, 288 Seiten, DM 49,-
ISBN 3-89011-076-2

Datenbank-Probleme optimal gelöst.

Ob Superbase Personal, Superbase 2 oder Superbase Professional - das große Buch zu Superbase beschreibt Ihnen detailliert, was Sie mit den einzelnen Datenbank-Versionen zu leisten vermögen. Es garantiert nicht nur einen leichten Einstieg in die Arbeit mit Superbase, sondern zeigt Ihnen auch an praxisnahen Beispielen die optimale Problemlösung. Erfahren Sie Schritt für Schritt, wie Sie Ihre erste Datei erstellen, welche verschiedenen Darstellungsmöglichkeiten es gibt, was sich hinter den Menüs verbirgt, welche besonderen Merkmale Superbase hat, warum relationale Dateien Vorteile bringen, wie mit den verschiedenen Editoren gearbeitet wird, was "DML" ist u.v.a.m.

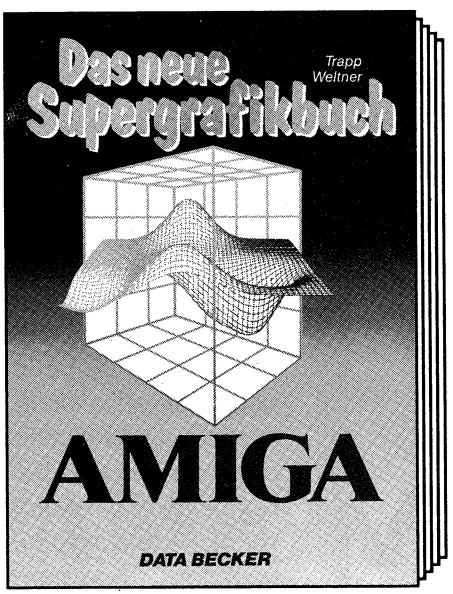
Tornsdorf
Das große Buch zu Superbase
413 Seiten, DM 39,-
ISBN: 3-89011-319-2

Machen Sie mehr aus Ihren Disketten

Bei diesem Buch geht es rund - im wahrsten Sinne des Wortes. Alles dreht sich um die zahlreichen Möglichkeiten, wie Sie aus Ihren Disketten den größten Nutzen ziehen - durch die Kenntnis der nötigen Fakten und durch die auf Diskette beiliegenden Programme. Lesen Sie, wie die Floppy unter Workbench und CLI/Shell arbeitet; wie in BASIC Daten geladen und gespeichert werden (sequentielle und relative Dateien); wie die DOS-Funktionen greifen; was bei der File-Verwaltung wichtig ist (Blocktypen, Bootblock, Checksummen, File-Header, Hash-Berechnung, Bitmap); wie man sich vor Viren schützt; welche Befehle, Strukturen und Nachrichten zum Trackdisk-Device gehören; wie Sie ohne DOS auf Disketten zugreifen u.v.a.m. Die beiliegende Diskette enthält einen Floppyspeeder, einen Disketten-Monitor und starke Kopierprogramme.

Bleek/Gelfand
Das große Floppybuch
Hardcover, 557 Seiten, inkl. Diskette, DM 59,-
ISBN 3-89011-180-7

Malprogramme, Bobs und Apfelmännchen.

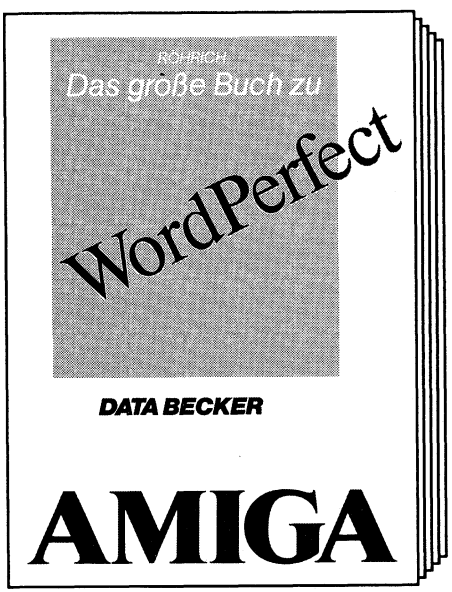


Als Amiga-Freund wissen Sie es längst: Der Amiga ist eine tolle Grafik-Maschine. Bis zu 4096 Farben gleichzeitig, 640 x 512 Bildpunkte Auflösung, Sprites, Bobs und die Geschwindigkeit des Grafikprozessors begeistern jeden Anwender. Das neue Supergrafikbuch zum Amiga ist die ideale Hilfe, um alle Funktionen schnell und sicher in den Griff zu bekommen - besonders dann, wenn man sich der vielen Beispielprogramme bedient. Erfahren Sie alles über die Grafikbefehle (von PSET bis zum Apfelmännchen), das Benutzer-Interface Intuition, den Viewport, die Zeichensätze des Amigas, die Möglichkeiten zur Hardcopy-Erstellung, das Laden von Fremdgrafiken, ein Malprogramm mit 1024 x 1024 Punkten u.v.a.m.

Trapp/Weltner
Das neue Supergrafikbuch zum Amiga
405 Seiten, DM 39,-
ISBN: 3-89011-345-1

Perfekt schreiben mit WordPerfect Amiga.

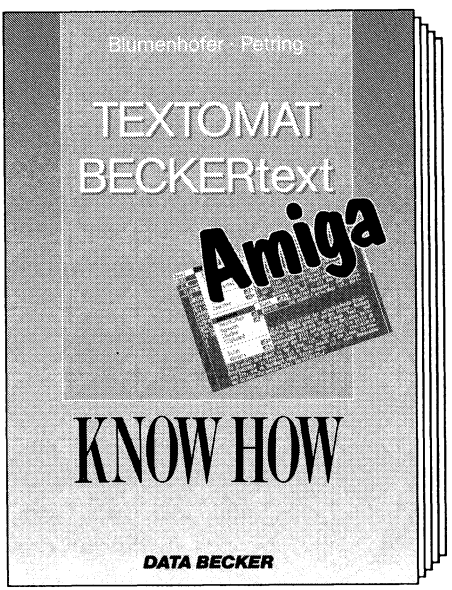
WordPerfect-Amiga ist ein Programm mit außergewöhnlichen Leistungsmerkmalen, denn gerade die vielfältigen Formatierungsarten und die Möglichkeit, fast beliebig lange Texte zu schreiben, zeichnen dieses Programm aus. Wer möglichst schnell WordPerfect-Amiga nutzen will, findet in diesem Buch alle Funktionen ausführlich beschrieben. Praxisnahe Anwendungen, die sofort zum Erfolg führen, fehlen genausowenig wie ein umfassender Nachschlageteil, der auch später das Auffinden der benötigten Informationen gewährleistet. Natürlich auch in diesem Buch: zahlreiche Tips und Tricks für die tägliche, praktische Arbeit. Das große Buch zu WordPerfect – für buchstäbliche Vielschreiber einfach ein Muß.



Polk/Röhrich
Das große Buch zu WordPerfect
316 Seiten, DM 39,-
ISBN 3-89011-305-2

Gewußt wie: perfekte Texte leichtgemacht.

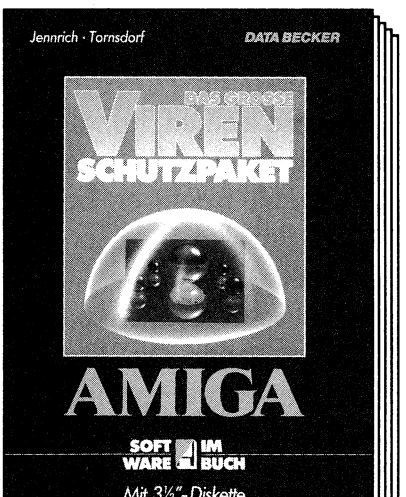
Warum BECKERtext und TEXTOMAT Amiga nicht nur mehrere 100.000mal im Einsatz, sondern auch bei Anwendern und Fachleuten gleichermaßen beliebt sind, zeigt dieses Buch: Die außergewöhnlichen Leistungsmerkmale der beiden Textverarbeitungen werden ausführlich beschrieben. Und damit die angelesenen Kenntnisse sofort umgesetzt werden können, fehlen praxisnahe Anwendungen ebensowenig wie zahlreiche Tips und Tricks. Lesen Sie, wie man Kopf- und Fußzeilen richtig einsetzt, perfekte Index- und Inhaltsverzeichnisse erstellt, Rundschreiben und Serienbriefe zu Papier bringt, Tabellen und Formulare gestaltet, Zeitungen mit Spaltentext druckt, Texte auf korrekte Rechtschreibung überprüfen läßt u.v.a.m.



Blumenhofer/Petring
TEXTOMAT & BECKERtext Know-how
286 Seiten, DM 39,-
ISBN: 3-89011-245-5

Endlich Schluß mit den Computerviren.

Schlimm genug, aber am leidigen Thema Computer-Viren kommt keiner vorbei. Speziell auf Amiga-Rechnern treten immer häufiger die sogenannten Boot-Block-Viren auf. Sorgen Sie schon im voraus für den nötigen Schutz: Im großen Amiga-Viren-Schutzpaket finden Sie Programme, die diese Viren sofort erkennen und entfernen. Sei es auf der Festplatte oder auf der Diskette. Auch zukünftige Störenfriede, beispielsweise Link-Viren, werden dabei schon berücksichtigt, denn jede Veränderung an Programmen und Daten wird sofort gemeldet.

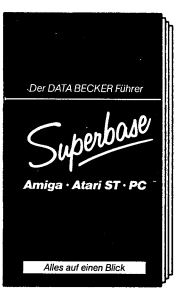


Selbst wenn ein Virus bereits den Boot-Block eines Ihrer Programme zerstört hat, läßt sich dieser ohne weiteres mit einem der mitgelieferten Hilfsprogramme wiederherstellen. Das Buch selbst bietet Ihnen detaillierte Anleitungen zu den einzelnen Anti-Viren-Programmen und natürlich auch das entsprechende Hintergrundwissen zu Verbreitung, Funktionsweise und Aufbau der verschiedenen Virenprogramme.

Bleek/Jennrich
Das große Amiga-Viren-Schutzpaket
172 Seiten, inkl. Disk., DM 69,- (unverb. Empf.)
ISBN 3-89011-802-X

EINFACH FÜHREND!

Findige Geister wissen längst: Man muß nicht alles selbst im Kopf haben; wissen sollte man nur, wo man was findet. Bei Fragen rund um den Amiga greifen Sie daher kurzerhand zu den DATA-BECKER-Führern. Denn diese kleinen, handlichen Bände sind führend, wenn es darum geht, ein bestimmtes Problem in Sekundenschnelle zu lösen. Ob zum Betriebssystem, zum Programm oder zur Programmiersprache – ein Blick genügt. Geordnet nach Sachgruppen und Stichworten oder alphabetisch mit Kurzsyntax finden Sie hier jede gesuchte Information. Die DATA-BECKER-Führer – die kompletteste und zugleich auch erfolgreichste Serie ihrer Art.



Der DATA-BECKER-Führer zu AmigaDOS & AmigaBASIC
320 Seiten, DM 24,80
ISBN 3-89011-431-8

Der DATA-BECKER-Führer zu Superbase
223 Seiten, DM 29,80
ISBN 3-89011-468-7

Das steht drin:

Amiga Tips & Tricks ist eine riesige Fundgrube für jeden Amiga-Benutzer. Denn hier zeigen die Autoren, wie man das Amiga-Betriebssystem richtig nutzt. Viele Beispielprogramme veranschaulichen die fantastischen Möglichkeiten dieses Superrechners.

Aus dem Inhalt:

CLI:	Tips & Tricks zum CLI bzw. Shell, Neue StartUp-Sequencen, Dateiverwaltung mit dem CLI, Batchprogrammierung.
BASIC:	Nutzung des Betriebssystems von AmigaBASIC und GFA-BASIC, Intuition-Programmierung unter BASIC, Programmieren von Windows, Screens, Menüs, Gadgets, ..., Variablen-Dump in AmigaBASIC, Assembler-Programme mit AmigaBASIC nutzen, Beschleunigen von Betriebssystem-Routinen, der Superdiskettenmonitor zum Abtippen, Selbstmodifizierende Programme.
Hardware:	Die Takt-Bremse (stufenloses Abbremsen des Amiga), Der Stopp-Schalter läßt den Amiga einfrieren, Prozessor-Umschaltung, Einbau einer 68010 CPU.
Workbench:	Tastaturtricks, MonoColor-Workbench für zusätzlichen Speicher, Arbeiten mit dem Betriebssystem 1.3, FastFilingSystem.

Und geschrieben haben dieses Buch:

Wolf-Gideon Bleek, Stefan Maelger und Tobias Weltner, die für Sie auf der Suche nach unerforschten Gebieten im Amiga waren. Tobias Weltner ist seit der ersten Auflage dabei. Stefan Maelger ist Kenner von AmigaBASIC und GFA-BASIC. Wolf-Gideon Bleek hat schon mit dem »Großen C-Buch« und »Intern Band 2« für Furore gesorgt. Mit Amiga Tips & Tricks stellen die Autoren ihren Erfolg auch den BASIC-Programmierern zur Verfügung.

ISB N 3-89011-211-0 DM +049.00

DM 49,–
ÖS 382,–
sFr 47,–

04900

DATA BECKER

9 783890 112114