Commodore
Sachbuch

Horst-Rainer Henning

Programmieren mit AMIGA-BASIC

Grafik ★ Sprites ★ Sprachausgabe ★ Sequentielle Dateien ★ Fenstertechnik ★ Musik ★ Tips & Tricks

Enthalten:
3½"-Diskette mit allen Programmbeispielen.

Programmieren mit Amiga-BASIC

Horst-Rainer Henning

Programmieren mit Amiga-BASIC

- Grafik
- Sprites
- Sprachausgabe
- Sequentielle Dateien
- Fenstertechnik
- Musik
- Tips&Tricks

Markt&Technik Verlag AG

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Henning, Horst-Rainer:
Programmieren mit AMIGA-BASIC : Grafik, Sprites, Sprachausg., sequentielle Dateien, Fenstertechnik, Musik, Tips&Tricks / Horst-Rainer Henning. –
Haar bei München : Markt-und-Technik-Verlag, 1987. – & 1 Diskette
ISBN 3-89090-434-3
(Commodore-Sachbuch)

15 14 13 12 11 10 9 8 7 6 5
90 89 88

ISBN 3-89090-434-3

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Kösel, Kempten
Printed in Germany

Inhaltsverzeichnis

Vorwort

Kaum ein Computer wurde, lange bevor er überhaupt auf den Markt kam, mit so vielen Vorschußlorbeeren bedacht wie der Amiga von Commodore. Gespannt wartete man eine geraume Zeit auf sein Erscheinen. Als schon niemand mehr so recht daran glauben wollte, da kam er mit einem kräftigen Donnerschlag. Die unwahrscheinlichsten Gerüchte bestätigten sich. Ausgerüstet mit einer Supergrafik, von der Computer-Besitzer bisher nur träumen konnten. Eine Million Bildpunkte pro Sekunde setzen! Eine Fläche wird nicht mehr gefüllt. Sie ist einfach ausgemalt. Neben der Supergrafik ein Super-Sound, und sprechen kann er auch. Obwohl er ein vollwertiger 16-Bit-Personalcomputer ist, dazu noch multitaskingfähig, kann ihn jedes Kind bedienen. Was kann man alles damit anfangen! Zuerst natürlich die üblichen Büroaufgaben, wie jeder andere PC auch. Aber daneben CAD, Spiele, Musik, Video-Bearbeitung, Datenübertragung und und und ...

Es ist gut, daß Commodore diesem Super-Computer auch ein Super-Basic mitgegeben hat. Über 80 Kbyte Speicherbereich benötigt das Basic-Programm! Lieber Leser, Sie werden fragen, hat er denn keine Nachteile? Doch er hat, leider! Denn es gibt für ihn bisher nur wenig Software und noch weniger Literatur, vor allem in deutscher Sprache. Diesen Mißstand zu lindern, ist auch Sinn und Zweck dieses Buches.

Was können Sie von diesem Buch erwarten? Es wird Sie in die Programmierung des Amiga-Basic einführen. Sie brauchen keine Angst zu haben, daß Sie mit theoretischem Wissen erschlagen werden. Die praktische Anwendung steht im Vordergrund. Am besten ist es, wenn Sie das Buch direkt neben dem Computer liegen haben. Etwa 100 Programme und viele Beispiele zeigen Ihnen die praktische Anwendung von über 200 Befehlen. Trotzdem soll es Ihnen auch als Nachschlagewerk dienen. Am Ende des Buches finden Sie eine Übersicht über alle besprochenen Basic-Befehle und Anweisungen und wo Sie diese wiederfinden können.

Hier noch einige technische Hinweise: Die einzelnen Befehle sind im Text durch Fettschrift gekennzeichnet. Es wird Wert darauf gelegt, daß nur eine kurze Beschreibung der Befehle das Wichtigste zusammenfaßt. Einzelheiten werden dort angegeben, wo sie

für die Programmierung erforderlich sind. Befehls-Optionen, die nur wahlweise für einen Befehl benötigt werden, sind kursiv gedruckt. Im Text aufgeführte Sprungmarken oder Unter-Routinen sind ebenfalls in kursiver Schrift dargestellt. Die Namen der Variablen sind, soweit sie im Text vorkommen, in fetter Schrift gedruckt.

Alle diese Programme liegen diesem Buch als 3 1/2-Zoll-Amiga-Diskette bei. Der Inhalt der Diskette, und wie Sie die einzelnen Programme starten, wird in Anhang 2 beschrieben.

Nun noch ein paar Worte zur Betriebssystem-Software des Amiga. Dazu muß ich ein wenig ausholen. Der erste Amiga, der Amiga 1000, hat nur ein sehr kleines ROM (fest eingebaute Software) erhalten. Diese Software reicht gerade aus, um die Funktionen bereitzustellen, daß die Kickstart-Diskette gelesen werden kann. Mit dieser Kickstart-Diskette wird dann das Betriebssystem in einen 256 Kbyte großen RAM-Speicherbereich geschrieben, der danach nicht mehr überschrieben werden kann. Dieser WOM-Bereich (Write Once Memory) verhält sich also wie ein ROM und wird in den Dokumentationen von Commodore auch oft fälschlicherweise als ROM bezeichnet. Der wesentliche Unterschied liegt darin, daß ein ROM auch nach dem Abschalten des Computers erhalten bleibt, das WOM dagegen nicht. Darum muß bei Amiga 1000 bei jedem Neustart zuerst die Kickstart-Diskette eingelegt werden.

Die ersten Amiga wurden mit dem Betriebssystem der Versionsnummer V1.1 ausgeliefert. Ende 1986 hat Commodore für den Amiga eine neue Version des Betriebssystems mit der Versionsnummer V1.2 herausgebracht. Das ist für den Anwender natürlich sehr erfreulich, da dadurch die alte Version verbessert ist und einige Fehler beseitigt sind. Es zeigt, daß ein von Diskette ladbares Betriebssystem seine Vorteile hat. Andererseits haben sich aber auch kleinere Änderungen für die Programmierung ergeben. Die wesentlichste Neuerung ist die Anpassung des Betriebssystems und des Basic-Interpreters an die deutsche Fernsehnorm, die etwas höhere Bilder ermöglicht. Diese Höhe können Sie auch von Basic voll ausnutzen. Die Programme dieses Buches sind bis auf eine Ausnahme so geschrieben, daß sie auf beiden Betriebssystemen laufen. Die Ausnahme betrifft das Programm zur Ermittlung der maximalen Fenstergröße. Wenn Sie die erste Betriebssystem-Version besitzen, ist das also kein Nachteil. Für die Erstellung eigener Programme sollten Sie sich aber die neue Version besorgen.

Wenn Sie einen Amiga besitzen, der keine Kickstart-Diskette benötigt, brauchen Sie sich über das alles keine Gedanken zu machen. Das Betriebssystem Ihres Amiga ist im ROM-Bereich untergebracht. Sie haben die Version des Betriebssystems, die höhere Bilder ermöglicht.

Und nun, viel Spaß!

1
Grundlagen des Amiga-Basic

Dieses Buch kann allein aus Platzgründen nicht sämtliche Konzepte der Programmiersprache Basic in Form eines Lehrbuchs erläutern und gleichzeitig den Umgang mit sämtlichen Fähigkeiten des Amiga, die ihn aus der Masse der Computer herausheben, beschreiben. Sämtliche Kapitel gehen deshalb von gewissen Grundkenntnissen aus, die der Leser in der Sprache Basic und in der Bedienung des Amiga besitzen sollte.

1.1 Grundsätzliches zur Bedienung des Amiga

Bevor Sie mit Amiga-Basic arbeiten können, müssen Sie natürlich – zumindest oberflächlich – mit der Bedienung des Amiga selbst vertraut sein. Falls Sie diese Kenntnisse noch nicht besitzen, sollten Sie zunächst einmal aufhören, in diesem Buch weiterzulesen und sie sich verschaffen. Hierzu sollten Sie ein Handbuch heranziehen, in dem die Bedienung des Amiga beschrieben wird (zum Beispiel »Das Amiga-Handbuch«, erschienen im Verlag Markt & Technik, Best.-Nr. 90228), und sich vor allem ein paar Stunden Zeit nehmen, mit dem Amiga herumzuspielen. Danach sollten Sie zumindest grundsätzlich über die folgenden Punkte Bescheid wissen:

- Was die »Maus« ist, und wie Sie damit den »Mauszeiger« bewegen
- Wie Sie ein Objekt auf dem Bildschirm »anklicken«
- Wie Sie ein Objekt auf dem Bildschirm mit der Maus verschieben
- Was ein »Pull-down-Menü« ist, und wie Sie einen Befehl daraus auswählen
- Was Dialogfenster (Requester) sind, und wie Sie damit umgehen
- Was »Fenster« sind, und wie Sie sie verschieben, vergrößern, verkleinern und schließen
- Wie Sie den Inhalt von Fenstern verschieben

- Was ein »Screen« ist, und wie Sie damit umgehen
- Wie Sie ein Disketten-Icon öffnen
- Wie Sie eine Schublade öffnen
- Wie Sie eine neue Schublade anlegen
- Wie Sie ein Objekt von einer Schublade in eine andere legen
- Wie Sie Disketten, Schubladen, Programme und Dokumente löschen und umbenennen
- Wie Sie ein Programm starten

1.2 Die Betriebsarten des Amiga-Basic

Nun können Sie endlich Ihre ersten Kontakte mit dem Basic des Amiga knüpfen. Der Basic-Interpreter stellt Ihnen hierzu drei verschiedene Betriebsarten oder Betriebsmodi zur Verfügung.

Im Direkt-Modus werden die Basic-Befehle sofort (direkt) nach der Eingabe ausgeführt. Um im Programm-Modus arbeiten zu können, müssen Sie zunächst eine Liste mit Anweisungen erstellen. Diese Liste nennt man Programm. Sobald dieses Programm gestartet ist, befinden Sie sich im Basic-Programm-Modus. Die dritte Betriebsart nennt man Editier-Modus. Solange Sie Programme eingeben und ändern, befinden Sie sich im Editier-Modus.

Um dies alles ausprobieren zu können, müssen Sie aber zunächst wissen, wie Sie ins Basic gelangen.

1.2.1 Amiga-Basic laden

Nehmen Sie dazu die Workbench-Diskette aus dem internen Laufwerk und legen Sie statt dessen die Diskette Amiga-Extras ein. Öffnen Sie das Disketten-Icon, das daraufhin erscheint, durch einen Doppelklick mit der Maus. In dem neuen Fenster, das sich dann öffnet, finden Sie vier Piktogramme (Icons); den Amiga-Tutor, den Mülleimer, eine Schublade BASICDemos und Amiga-Basic. Der Basic-Interpreter »verbirgt« sich hinter dem Icon Amiga-Basic. Es handelt sich dabei um ein Programm wie jedes andere auch (zum Beispiel TextCraft oder Deluxe Paint). Anders als bei anderen Computern, ist Basic in den Amiga nämlich nicht »fest eingebaut«.

Zum Laden von Amiga-Basic stehen Ihnen drei Möglichkeiten zur Verfügung:

- Sie laden ein beliebiges Basic-Programm, indem Sie einen Doppelklick auf dessen Icon machen. Amiga-Basic wird dabei automatisch mitgeladen. Wenn Sie wollen, können Sie das jederzeit mit einem Programm aus der Schublade BASICDemos ausprobieren.

- Sie aktivieren mit einem Doppelklick auf das Icon Amiga-Basic den Basic-Interpreter, ohne ein Programm dabei zu laden.

• Die dritte Möglichkeit bietet uns das CLI. Ich habe lange überlegt, ob ich diesen Weg mit aufführen soll. Sollte Ihnen einmal der zur Verfügung stehende Speicher nicht ausreichen, ist dies allerdings der Weg, welcher am wenigsten Speicherplatz beansprucht. Dagegen spricht eventuell, daß Sie wahrscheinlich mit dem CLI keine Erfahrung haben. Das CLI ist eine eigene Schnittstelle zum Betriebssystem und hat eine große Anzahl eigener Befehle. Sie befinden sich dort nicht mehr in der sehr sicheren Workbench, die Sie vor Ihrer eigenen Unachtsamkeit schützt. Mit einer einzigen falschen Eingabe können Sie den Inhalt einer Diskette zerstören. Wenn Sie trotzdem nicht darauf verzichten wollen, achten Sie bitte genau auf die folgenden Ausführungen:

Aktivieren Sie das CLI-Icon durch einen Doppelklick. Nun öffnet sich das CLI-Fenster. Geben Sie jetzt über die Tastatur ein:

```
1>df0:Amiga-Basic
```

(die fettgedruckten Buchstaben brauchen Sie nicht einzugeben – diese gibt das CLI auf den Bildschirm) und drücken dann die RETURN-Taste. Wenn Sie später Amiga-Basic wieder verlassen, landen Sie wieder in diesem CLI-Fenster. Um dieses Fenster zu schließen (es besitzt kein Schließ-Gadget), müssen Sie den folgenden Befehl eingeben:

```
1>endcli
```

und wieder die RETURN-Taste drücken. Das CLI-Fenster verschwindet daraufhin, und Sie befinden sich in der Workbench.

Wählen Sie zunächst einmal den sicheren Weg: Fahren Sie mit dem Mauspfeil über das Amiga-Basic-Icon und drücken Sie zweimal kurz auf die Auswahltaste der Maus. Das Disketten-Laufwerk beginnt zu summen, und nach einem kurzen Ladevorgang sehen Sie zwei neue Fenster am Bildschirm: ein großes und ein halb so großes, welches das erste Fenster zur Hälfte verdeckt. Das größere der beiden Fenster nennt man Ausgabe-Fenster, weil in ihm die Ausgaben der meisten Programme erfolgen. Das kleinere Fenster (welches Sie natürlich wie jedes andere Fenster in seiner Größe verändern können) ist das List-Fenster. In diesem Fenster wird später das gerade bearbeitete Programm aufgelistet. Sie geben hier neue Programm-Befehle ein oder ändern die bereits vorhandenen.

1.2.2 Der Direkt-Modus

Wie Sie bereits in der Einleitung zum Kapitel 1.2 erfahren haben, unterscheidet man beim Basic zwischen Direkt- und Programm-Modus. Im Direkt-Modus geben Sie über die Tastatur direkt Anweisungen an das Amiga-Basic. Sie können damit Programme laden und speichern, den Programmablauf steuern und Fehler lokalisieren. Bei anderen Computern erkennt der Basic-Interpreter den Unterschied zwischen Direkt- und Programm-Modus daran, daß einem direkten Befehl (der sofort ausgeführt werden soll) keine Zeilennummer vorangeht.

Da Amiga-Basic in der Regel ohne solche Nummern auskommt, hat man hier einen anderen Weg gewählt: Die beiden Modi arbeiten in unterschiedlichen Fenstern. Die direkten Befehle werden im Ausgabefenster eingegeben, und alle Befehle, die in ein Programm aufgenommen werden sollen, müssen im List-Fenster eingeben werden.

Um einen ersten Befehl im Direkt-Modus eingeben zu können, aktivieren Sie bitte das Ausgabefenster, indem Sie mit der Maus mitten ins Fenster klicken. Der Cursor erscheint unter dem Text mit der Basic-Version. Mit »Cursor« ist der senkrechte orangefarbene Strich gemeint. Er markiert immer die Stelle, an welcher neu eingegebener Text auftaucht. Geben Sie nun ein:

```
print "Guten Tag, ich bin der Amiga von Commodore"
```

und drücken Sie die RETURN- oder ENTER-Taste. Wenn Sie einen Schreibfehler korrigieren müssen, können Sie das mit der Taste BACKSPACE (Rückwärts-Leertaste) tun. Wenn Sie die ganze Zeile löschen wollen, drücken Sie die Tasten-Kombination CTRL-C. Jetzt sollte auf Ihrem Bildschirm stehen:

```
Guten Tag, ich bin der Amiga von Commodore
OK
```

Mit PRINT haben Sie Ihrem Amiga den ersten Befehl im Direkt-Modus erteilt und dabei einen der am häufigsten verwendeten Basic-Befehle kennengelernt.

> **PRINT** ***Liste von Ausdrücken***
> zeigt den Wert der Liste von Ausdrücken im aktuellen Fenster

Sie haben in dem Beispiel den Befehl gegeben, einen Text (Guten Tag, ich bin der Amiga von Commodore) auszugeben. (Texte, die ausgegeben werden sollen, müssen übrigens immer in Anführungszeichen eingegeben werden.) Genausogut können Sie mit PRINT aber auch eine Rechenaufgabe lösen lassen. Hierzu brauchen Sie nur statt des Textes die entsprechende Formel hinter PRINT zu schreiben. Und wenn Sie PRINT ohne einen Ausdruck dahinter eingeben, wird eine Leerzeile ausgegeben.

Die Eingabe im Direkt-Modus ist übrigens nicht auf einen Befehl begrenzt. Sie können so viele Befehle eingeben, wie auf eine Basic-Zeile mit 255 Zeichen Länge passen. Die einzelnen Anweisungen müssen dabei durch einen Doppelpunkt (:) getrennt werden:

```
PRINT "Befehlszeile":PRINT 5/9*7:PRINT :PRINT 86-34
```

Drücken Sie RETURN und Sie erhalten:

```
Befehlszeile
3.888889

52
OK
```

Sie sehen an diesen Beispielen, daß ein Befehl im Direkt-Modus sofort nach Betätigung der RETURN-Taste ausgeführt wird, ohne abgespeichert zu werden. Die Resultate werden sofort angezeigt.

1.2.3 Der Programm-Modus

Kommen wir nun zu dem Modus, mit dem Sie meistens arbeiten werden, dem Programm-Modus. Ein Programm ist eine Liste von Befehlen oder Anweisungen, die Amiga-Basic in der Reihenfolge abarbeitet, in der Sie sie eingegeben haben, es sei denn, bestimmte Befehle für die Ablaufsteuerung sorgen für eine andere Reihenfolge. Diese Liste können Sie abspeichern, jederzeit wieder laden, ändern und ausführen lassen (starten). Sobald ein Programm gestartet wurde, befindet es sich im Programm-Modus, bis es beendet ist. Wenn der Programmierer dafür gesorgt hat, können Sie es nicht einmal anhalten. Befehle im Direkt-Modus sind nicht möglich, während ein Programm läuft.

Bevor Sie ein Programm laufen lassen können, müssen Sie erst eines schreiben. Dazu aktivieren Sie bitte das List-Fenster. Sollte es nicht sichtbar, oder geschlossen sein, rufen Sie SHOW LIST aus dem Windows-Menü auf. Geben Sie dann das folgende Programm ein. Es gibt einige Zeilen mit dem Zeichen # auf den Bildschirm:

```
a$= "###"
FOR i = 1 TO 20
b$=b$+a$
PRINT b$
NEXT
```

Starten Sie das Programm mit dem Menü-Punkt Start aus dem Run-Menü. Wenn Ihnen bei der Eingabe ein Fehler unterlaufen sein sollte, stehen Ihnen die komfortablen Editier-Eigenschaften des Amiga-Basic zur Verfügung.

1.2.4 Programme lesen und editieren

Wenn Ihre Programme einmal die Länge von einigen Bildschirmseiten erreicht haben, werden Sie zu schätzen wissen, was Amiga-Basic alles zur Bearbeitung Ihres Programmtextes zu bieten hat. Die Möglichkeiten lassen sich durchaus mit denen eines Textverarbeitungsprogrammes vergleichen. Die Einzelheiten sollten Sie am besten gleich an einem längeren Programmtext ausprobieren. Laden Sie dazu bitte das Programm LIST-ME aus der Schublade BASICDemos (siehe Kapitel 1.3.3).

Zuerst sollen Sie erfahren, wie Sie an eine bestimmte Stelle im Listing gelangen können. Aktivieren Sie dazu das List-Fenster, indem Sie mit der Maus irgendwo in dieses Fenster klicken. Schon haben Sie die erste Frage, wie Sie an eine bestimmte Stelle im Listing gelangen, gelöst. An der Stelle, an der sich der Mauspfeil befand, als Sie die Auswahltaste der Maus drückten, befindet sich jetzt der Text-Cursor.

An dieser Stelle können Sie nun einen Text einfügen. Probieren Sie es ruhig aus. Da Sie das Programm nicht abspeichern werden, können Sie gar nichts falsch machen. Sie können aber auch mit den Cursor-Tasten zu der gewünschten Stelle der Programmliste fahren. Die Cursor-Taste mit dem Pfeil nach links bewegt den Cursor nach links. Probieren Sie es am besten gleich aus. Lassen Sie Ihren Finger ein wenig länger auf

dieser Taste. Sobald Sie die linke Fensterbegrenzung erreicht haben, ertönt ein Warnton. Weiter geht es also nicht. Jetzt drücken Sie die Cursor-Taste mit dem Pfeil nach rechts. Keine Sorge, wenn der Cursor an der rechten Fensterbegrenzung angelangt ist, werden Sie nicht von einem hellen Ton erschreckt. Aber etwas anderes passiert. Der ganze Text verschiebt sich nach links. Das geschieht so lange, wie noch Text vorhanden ist. Den Cursor können Sie aber noch weiterbewegen. Erst nach einer Zeilenlänge von 255 Zeichen ertönt das Warnsignal wieder. Die rechte Grenze ist die maximale Länge einer Basic-Zeile.

Sie müssen aber nicht jedesmal mit dem Cursor eine ganze Zeile entlangfahren, um an Ihr Ziel zu gelangen. Vier Tasten-Kombinationen helfen Ihnen, Zeit zu sparen, indem Sie schnell größere Abstände überspringen. Bei der Kombination zweier Tasten merken Sie sich bitte, daß Sie zunächst die zuerst genannte Taste drücken und festhalten müssen und dann erst die zweite Taste betätigt werden darf.

- ALT-Pfeil nach rechts: setzt den Cursor an das Zeilenende
- ALT-Pfeil nach links: setzt den Cursor an den Zeilenanfang
- SHIFT-Pfeil nach rechts: bringt den Cursor an den Anfang des letzten Viertels einer Programmzeile
- SHIFT-Pfeil nach links: bringt den Cursor an das Ende des ersten Viertels einer Programmzeile

Die beiden Cursor-Tasten »Pfeil nach oben« und »Pfeil nach unten« bringen den Cursor zeilenweise nach oben oder nach unten. Aber das haben Sie inzwischen wahrscheinlich schon vermutet. Wenn Sie mit diesen beiden Tasten an den oberen oder unteren Fensterrand gelangen, ist dort nicht Endstation. Das Listing beginnt dann nach oben oder unten zu scrollen. Erst wenn Sie am Programmanfang oder -ende angekommen sind, hören Sie wieder den Ton, als Signal dafür, daß es nicht weitergeht. Bei einem langen Programm ist diese Methode zu umständlich. Auch hier hilft Ihnen der Amiga-Basic mit vier Tasten-Kombinationen:

- ALT-Pfeil nach oben: setzt den Cursor in die erste Programmzeile
- ALT-Pfeil nach unten: setzt den Cursor in die letzte Programmzeile
- SHIFT-Pfeil nach oben: blättert fensterweise rückwärts durchs Listing
- SHIFT-Pfeil nach unten: blättert fensterweise vorwärts durchs Listing

Nachdem Sie jetzt problemlos an jede Stelle des Programmes gelangen können, werden wir einige Editier-Versuche starten. Damit Sie beim Ändern besser arbeiten können, verbreitern Sie zunächst einmal das List-Fenster so weit wie möglich. Gehen Sie dann mit den Cursor-Tasten oder mit der Maus an eine beliebige Stelle des Programms, die Sie ändern wollen. Die Zeichen links vom Cursor können Sie mit der BACKSPACE-Tasté löschen. Aber seien Sie vorsichtig, daß Sie dabei nicht mehr löschen, als Sie eigentlich wollten. Bei unserem Übungsprogramm spielt das zwar keine Rolle, aber in Ihrem eigenen Programm können Sie damit einigen Schaden anrichten. Bleiben Sie zu lange auf der BACKSPACE-Taste, wiederholt der Amiga die Taste schneller, als Basic die Zeichen löschen kann. So kommt es, daß immer noch ein Zeichen nach dem anderen gelöscht wird, obwohl Sie die Taste längst losgelassen haben.

Zeichen rechts vom Cursor können Sie mit der DEL-Taste löschen. Ansonsten verhält sich DEL genauso wie BACKSPACE. Natürlich können Sie aber an der Stelle, an der sich der Cursor befindet, nicht nur Text löschen, sondern auch neuen Text eingeben. Probieren Sie es aus! Wenn Sie eine Zeile einfügen wollen, gehen Sie mit dem Cursor einfach ans Zeilenende der vorhergehenden Zeile und drücken RETURN. Schon haben Sie eine neue, leere Zeile.

Als nächstes werden Sie erfahren, wie Sie bestimmte Bereiche des Textes auswählen. Bewegen Sie dazu den Mauspfeil vor einen Buchstaben, drücken Sie die Auswahltaste und fahren bei gedrückter Auswahltaste einige Buchstaben nach rechts. Lassen Sie dort die Taste los. Die so überstrichenen Buchstaben erscheinen jetzt rot unterlegt. Sie können auf diese Weise einzelne Buchstaben/Zahlen auswählen oder auch ganze Zeilen. Um ein einzelnes Wort auszuwählen, kennt Basic sogar noch eine Abkürzung: Fahren Sie mit dem Mauszeiger über das Wort und klicken zweimal schnell hintereinander. Sie können auch mehrere Zeilen auf einmal auswählen. Klicken Sie in die erste Zeile des Bereichs, den Sie auswählen wollen und fahren dann bei gedrückter Maustaste nach unten. In der Zeile, in der Sie die Auswahl beenden wollen, lassen Sie die Taste dann los. Lassen Sie sich dabei nicht vom oberen oder unteren Rand des Fensters beeindrucken. Fahren Sie einfach weiter nach unten oder oben, bis der Text anfängt, sich zu verschieben.

Einen so selektierten (ausgewählten) Bereich können Sie mit der BACKSPACE-Taste löschen oder mit den Menü-Punkten des Edit-Menüs bearbeiten.

Bei dem Menü-Punkt *Cut* wird der ausgewählte Bereich im List-Fenster gelöscht und in einem Zwischenspeicher abgelegt. Die Tastenkombination »Amiga-X« hat dabei die gleiche Funktion wie *Cut*. (Wenn Sie hier und in den folgenden Kapiteln eine Tastenkombination mit einer Amiga-Taste finden, ist – sofern nichts anderes erwähnt wird – immer die rechte Amiga-Taste gemeint.)

Am besten probieren Sie diese Möglichkeit im Listing des Programmes LIST-ME gleich einmal aus. Ein auf diese Weise ausgeschnittener Bereich ist übrigens nicht verloren, sondern befindet sich in der sogenannten Zwischenablage. Sie können deren Inhalt an einer anderen Stelle einsetzen, indem Sie den Befehl *Paste* auswählen. Ist zu diesem Zeitpunkt ein Bereich selektiert, ersetzt der Inhalt des Zwischenspeichers den ausgewählten Bereich. Auch für *Paste* gibt es eine Tastenkombination: »Amiga-P«.

Der Menü-Punkt *Copy* funktioniert wie *Cut*, jedoch wird der selektierte Bereich nicht gelöscht. Auch dieser Befehl hat natürlich eine Tastatur-Abkürzung, und zwar »Amiga-C«.

Wie Sie bei den bisher besprochenen Menü-Punkten sehen konnten, gibt es für die meisten davon die zusätzliche Möglichkeit, den Befehl mit einer Tasten-Kombination aufzurufen. Einige Menübefehle können aber noch auf eine weitere Art erteilt werden. Damit man dies besser demonstrieren kann, laden Sie bitte das Programm Picture aus der Schublade BASICDemos.

Daß Sie mit *Show List* das List-Fenster öffnen oder nach vorne holen können, wurde ja bereits erwähnt. Die Tasten-Kombination »Amiga-L« bewirkt das gleiche. Sie können aber auch im Direkt-Modus LIST eingeben, und Sie öffnen damit ebenfalls das List-Fenster, selbst wenn kein Programm und damit kein Listing vorhanden ist:

LIST ***Zeile-Zeile,Dateiangabe***
aktuelles Basic-Programm wird in Ausgabeeinheit gelistet

Wenn Sie nur einen bestimmten Bereich sehen wollen, müssen Sie angeben, zwischen welchen Sprungmarken dieser liegen soll. (Sprungmarken werden später noch erläutert.) Geben Sie für das Beispiel-Programm Picture folgendes ein:

```
list -CheckMouse
```

Das Programm wird nun nur bis zur Sprungmarke *CheckMouse* aufgelistet. Sogar im Programm-Modus können Sie LIST anwenden. Ändern Sie dafür die dritte Programmzeile in:

```
CLS:LIST
```

und starten Sie dann das Programm. Sie erhalten daraufhin nach dem Start gleich das Listing und können dabei das Programm verfolgen.

Aber auch für das Ändern von Programmen gibt es einen Basic-Befehl. Wenn Ihnen bestimmte Teile Ihres Programmes nicht mehr gefallen, können Sie diese mit folgendem Befehl löschen, ohne ihn dafür mit der Maus auswählen zu müssen:

DELETE ***Marke1-Marke2***
löscht den angegebenen Zeilenbereich

Probieren Sie einmal im Direkt-Modus:

```
delete CheckMouse-MovePicture
```

Der Bereich zwischen den Sprungmarken *CheckMouse* und *MovePicture*, einschließlich der beiden Sprungmarken selbst, ist nun gelöscht.

1.2.5 Programmausführung steuern

Das umfangreichste unter den Amiga-Basic-Menüs ist das *Run*-Menü. Sechs Menüpunkte helfen Ihnen, genau zu bestimmen, wie das Programm ausgeführt werden soll. Damit erhalten Sie das wichtigste Werkzeug, um Programmierfehler zu lokalisieren.

Mit Start wird, wie Sie ja bereits wissen, das gerade bearbeitete Programm gestartet. Die Tastenkombination »Amiga-R« erfüllt den gleichen Zweck. Es gibt aber auch einen Amiga-Basic-Befehl, der ein Programm lädt und startet:

RUN
RUN Dateiangabe,*R*
startet ein Programm, bzw. liest es zunächst ein und startet es dann

Wenn der Parameter *R* angegeben ist, bleiben bereits geöffnete Dateien offen. Ohne *R* werden offene Dateien geschlossen, bevor das Programm gestartet wird.

Tippen Sie jetzt bitte ins Ausgabe-Fenster ein:

```
run "BASICDemos/Picture"
```

Das Programm Picture wird daraufhin geladen und sofort gestartet.

Dieses Programm ist so aufgebaut, daß es nicht beendet wird. Sie müssen also von außen eingreifen, um es anzuhalten. Mit dem Menüpunkt Stop können wir ein laufendes Programm unterbrechen. Amiga-Basic kehrt in den Direkt-Modus zurück. Für diesen Befehl stehen sogar zwei Tastenkombinationen, «Amiga-». (Punkt) oder CTRL-C, zur Verfügung, die dasselbe tun. Sie können auch einen Basic-Befehl gleichen Namens (und gleicher Funktion) in ein Programm einbauen:

STOP
unterbricht Programmausführung und kehrt in den Direkt-Modus zurück

Dies ist ein Befehl, den Sie hauptsächlich in der Testphase eines Programmes einsetzen werden. Sie können so im Direkt-Modus den aktuellen Wert der Variablen feststellen und, falls erforderlich, ändern. Fügen Sie nun bitte hinter der LINE-Anweisung im Programm Picture ein STOP ein und wählen Sie erneut den Befehl START. Das Programm wird dann angehalten, nachdem das weiße Rechteck gezeichnet ist.

Da das Programm dadurch nur unterbrochen wird, können Sie es jederzeit weiterlaufen lassen, als wäre nichts geschehen. Mit dem Menü-Punkt *Continue* oder mit dem Basic-Befehl CONT im Direkt-Modus setzen Sie ein unterbrochenes Programm fort.

CONT
setzt ein Programm am Unterbrechungspunkt fort

Sobald an dem Programm eine Änderung vorgenommen wurde, kann es allerdings nicht mehr fortgesetzt werden. In diesem Fall und dann, wenn programmtechnisch eine Fortsetzung nicht möglich ist, wird dies durch einen Requester (Can't continue) angezeigt. Sie müssen das Programm neu starten.

Mit *Suspend* aus dem Run-Menü oder mit den Tastenkombinationen »Amiga-S« und CTRL-S können Sie ein Basic-Programm vorübergehend anhalten, ohne daß es dabei unterbrochen wird. Es kehrt dabei nicht in den Direkt-Modus zurück und somit kann es auch nicht manipuliert werden. Es hält so lange an, bis Sie eine andere Taste als »Amiga-S« drücken. Bitte probieren Sie diese neuen Befehle anhand des Programms Picture aus. (Haben Sie den STOP-Befehl entfernt?)

Jetzt kommen wir zu den wichtigsten Menüpunkten aus dem Run-Menü für die Fehlersuche. Fehler lassen sich am besten finden, wenn man anhand des Listings verfolgen kann, wie das Programm abläuft. Für Amiga-Basic ist das kein Problem. Wählen Sie einfach *Trace On* aus dem Menü *Run.* Bei sichtbarem List-Fenster wird nun jede Anweisung, während Basic sie ausführt, mit einem roten Rechteck eingerahmt. *Trace Off* schaltet die Programmablaufverfolgung wieder aus. Das Rechteck, mit welchem der letzte Befehl markiert wurde, verschwindet. Auch für diese beiden Menü-Befehle gibt es zwei Basic-Befehle, welche im Direkt- und Programm-Modus eingesetzt werden können:

TRON
TROFF
schalten die Programmablauf-Verfolgung ein oder aus

Mit diesem Befehlspaar haben Sie die Möglichkeit, Ihr Basic-Programm genau zu verfolgen. Unterbrechen Sie nun das Programm Picture. Rufen Sie zuerst *Trace On* auf und dann die Option *Start.* Anschließend aktivieren Sie das List-Fenster. Sie können nun im Ausgabefenster die Entstehung der Ellipse und im List-Fenster die Abarbeitung der einzelnen Befehle erkennen. Durch die Verfolgung des Programmablaufes wird das Basic-Programm auch deutlich langsamer. Sie können deshalb den Ablauf sehr gut – in Zeitlupe – verfolgen. So können Sie eventuelle Fehler schnell eingrenzen.

Amiga-Basic bietet aber auch eine noch feinere Kontrolle über den Programmablauf, sozusagen ein Super-Trace, an. Der Befehl *Step* führt jeweils genau eine Zeile Ihres Programms aus und hält dann an. Auch hier wird, bei geöffnetem List-Fenster, die gerade ausgeführte Anweisung im Listing rot eingerahmt. Die Tastatur-Kombination »Amiga-T« hat dieselbe Wirkung wie *Step* im Run-Menü.

Auch diesen Menü-Punkt sollten Sie nun am Programm Picture ausprobieren. Unterbrechen Sie dazu das Programm und geben Sie im Direkt-Modus END ein. Rufen Sie dann Step aus dem Run-Menü auf und öffnen Sie das List-Fenster. Jedesmal, wenn Sie nun erneut Step aufrufen, wird ein neuer Befehl des Listings ausgeführt und mit einem roten Rechteck markiert. Schneller als die Auswahl eines Menüpunktes ist natürlich die Tastenkombination »Amiga-T«.

1.3 Programme laden und speichern

Das Basic-Menü Project enthält die wichtigsten Menü-Punkte, die Sie für eine ordentliche Verwaltung Ihrer Programm-Dateien auf der Diskette benötigen. (Für alle diese Punkte gibt es auch Basic-Befehle.) Damit Sie nicht versehentlich Ihr bestes Basic-Programm verlieren, wenn Sie Basic verlassen, fragt Sie Amiga-Basic mit einem Requester, ob Sie das aktuelle Programm nicht vorher noch abspeichern wollen. Ihre Antwort geben Sie, indem Sie in eines der drei Felder mit der gewünschten Antwort klicken.

1.3.1 Ein Pfad führt durch die Dateien

Bevor Sie sich Gedanken darüber machen können, wie das Laden und Speichern von Programmen vor sich geht, müssen Sie erst wissen, wie Sie mit Dateien und Datei-Verzeichnissen umgehen können. Legen Sie dazu für einen Augenblick die Fenster von Amiga-Basic nach hinten und schauen sich in dem Fenster der Extras-Diskette um. Sie finden dort die Schublade BASICDemos, aus der Sie schon einige Programme geladen haben. Diese Schublade, wie auch alle anderen Schubladen, entspricht einem Datei-Verzeichnis. Wenn sich in dieser Schublade eine weitere Schublade befinden würde, wäre dieses Datei-Verzeichnis ein Unterverzeichnis unter dem zuerst genannten Datei-Verzeichnis. Das Beispiel können Sie, wenn Sie Lust dazu haben, gedanklich beliebig weit fortführen. In der Schublade finden Sie Programme und Dateien (z.B. ball). Das sind alles Dateien innerhalb des Datei-Verzeichnisses der Schublade. Das oberste Datei-Verzeichnis ist immer das Disketten-Verzeichnis, welches man auch als Wurzel (root) bezeichnet.

Eine komplette Beschreibung des Ortes, an dem eine Datei zu finden ist, wird als Pfadname der Datei bezeichnet und setzt sich im wesentlichen wie folgt zusammen:

Disk-Name oder Laufwerk: ein oder mehrere Datei-Verzeichnisse/Namen

Kehren Sie nun ins Basic zurück. Wenn Sie wissen wollen, was sich auf Ihrer Diskette befindet, geben Sie folgenden Befehl im Direkt-Modus ein:

> **FILES** ***Dateiangabe***
> Inhalt eines Datei-Verzeichnisses wird gezeigt

Wenn Sie keine Dateiangabe angeben, erhalten Sie alle Dateien des aktuellen Datei-Verzeichnisses aufgelistet. Wenn Sie die Dateien in einem anderen Dateiverzeichnis sehen wollen, müssen Sie dafür einen Pfadnamen angeben, wie es eben besprochen wurde. Mit

```
files "df0:BASICDemos"
```

erhalten Sie zum Beispiel den Inhalt der Schublade BASICDemos. Mit

```
files "Workbench:Demos"
```

können Sie den Inhalt der Schublade Demos von der Diskette Workbench betrachten. Wenn Sie viel mit einem bestimmten Datei-Verzeichnis arbeiten, das nicht das aktuelle ist, kann es recht umständlich werden, bei jeder Datei-Operation immer den ganzen Pfadnamen eingeben zu müssen. Ein Basic-Befehl nimmt Ihnen diese Arbeit ab, indem es jedes gewünschte Datei-Verzeichnis zum aktuellen Verzeichnis macht:

CHDIR Pfad
macht Pfad zum aktuellen Dateiverzeichnis

Wenn Sie die Schublade BASICDemos zum aktuellen Datei-Verzeichnis machen wollen, geben Sie im Direkt-Modus ein:

```
chdir "df0:BASICDemos"
```

1.3.2 New

Mit *New* aus dem *Project*-Menü löschen Sie das Programm im Hauptspeicher samt allen Variablen. Das Programm-Listing im List-Fenster wird dabei ebenfalls gelöscht. Nach der Ausführung von *New* löscht der Basic-Interpreter auch noch das Ausgabefenster und kehrt in den Direkt-Modus zurück. Sollten Sie die Programmablaufüberwachung aktiviert haben, wird auch diese abgeschaltet. *New* werden Sie immer dann einsetzen, wenn Sie ein neues Programm beginnen wollen. Sie können an Stelle der Menü-Option *New* auch einen Basic-Befehl im Direkt-Modus eingeben:

NEW
Programm wird mitsamt allen Variablen aus dem Speicher gelöscht

1.3.3 Laden eine Programmes

Das Laden eines Programmes kann auf verschiedene Arten realisiert werden. Sie werden in diesem Kapitel drei davon kennenlernen. Am einfachsten laden Sie ein Programm, wenn Sie sein Programm-Icon auf Ihrer Basic-Diskette mit einem Doppelklick starten.

Diese Methode ist allerdings die zeitraubendste, wenn Sie sich bereits im Basic befinden, da Sie dazu Amiga-Basic verlassen müssen.

Die zweite Möglichkeit bietet das *Project*-Menü. Mit dem Befehl OPEN können Sie ein Programm von der Diskette in den Hauptspeicher laden. Vorher werden alle zu diesem Zeitpunkt geöffneten Dateien geschlossen und das gerade bearbeitete Programm gelöscht. Probieren Sie das gleich einmal aus. Es öffnet sich ein Kommunikations-Fenster (Requester) und fordert Sie zur Eingabe eines Programmnamens auf:

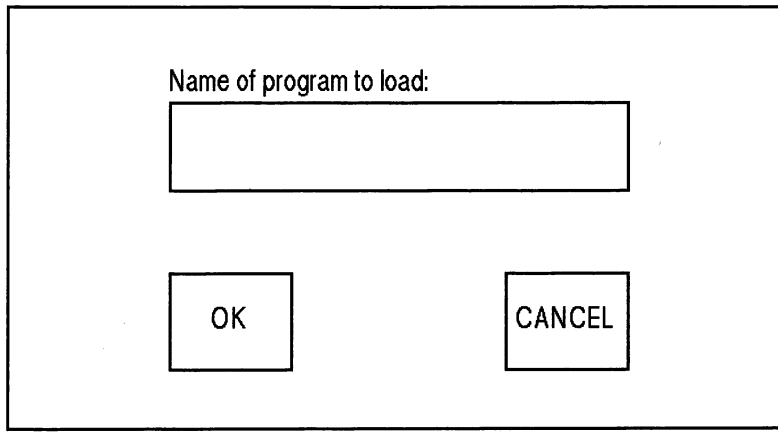


Bild 1.1: *Der Open-Requester*

Wir wollen jetzt das Programm Picture aus der Schublade BASICDemos laden. Klicken Sie dazu in das Feld für den Namen des Programms und geben Sie ein:

```
BASICDemos/Picture
```

Drücken Sie dann RETURN oder klicken Sie mit der Maus in das OK-Feld. Das Programm wird nun geladen. Haben Sie das List-Fenster geöffnet, wird Ihnen sofort das Listing gezeigt.

Die dritte Möglichkeit, ein Programm zu laden, bietet wieder einmal ein Basic-Befehl:

LOAD Dateiangabe,*R*
lädt (und startet) ein Programm von der Diskette oder Festplatte

Wird der Parameter R eingegeben, so wird das Programm geladen und gestartet. Andernfalls kehrt der Interpreter nach dem Laden in den Direktmodus zurück.

Beispiel: load "BASICDemos/Picture"
oder: load "df0:BASICDemos/Picture",R

1.3.4 Save

Mit *Save* aus dem Project-Menü speichern Sie das aktuelle Programm unter dem Namen, mit welchem Sie es vorher geladen haben. Dabei wird das Programm auch in dem Format (siehe nächstes Kapitel) gespeichert, in dem es vorher geladen wurde. Wenn Sie ein neues Programm erstellt haben oder ein Programm unter einem anderen Namen abspeichern wollen, können Sie diese Option nicht verwenden. Sie müssen dann den Befehl *Save As* aufrufen.

Dieser Befehl hat noch andere Anwendungsmöglichkeiten. Das ursprüngliche Programm, welches sich auf der Diskette befindet, wird durch ein neues Programm, das Sie unter gleichem Namen abspeichern, überschrieben. Falls Sie das alte Programm erhalten wollen, wenn Sie eine neue Version davon abspeichern, wählen Sie die Option *Save As*. In diesem Zusammenhang noch ein Tip: Öffnen Sie vor dem Abspeichern immer das List-Fenster. Sie sehen dadurch, ob sich das Programm noch im Speicher befindet. Wenn Sie einmal, durch einen Fehler im Programm, den Basic nicht entdeckt hat, Ihr eigenes Programm gelöscht haben, merken Sie das am fehlenden Listing. Falls Sie dieses Programm abspeichern, speichern Sie »Nichts« ab und verlieren Ihr altes Programm!

1.3.5 Save As

Der Menü-Punkt *Save As* aus dem Project-Menü funktioniert genau wie *Open*, mit dem einen Unterschied, daß der Name, den Sie hier eingeben, diesmal nicht dazu dient, ein Programm zu laden, sondern es unter diesem Namen abzuspeichern. Wenn Sie *Save As* aufrufen, erscheint folgendes Kommunikations-Fenster:

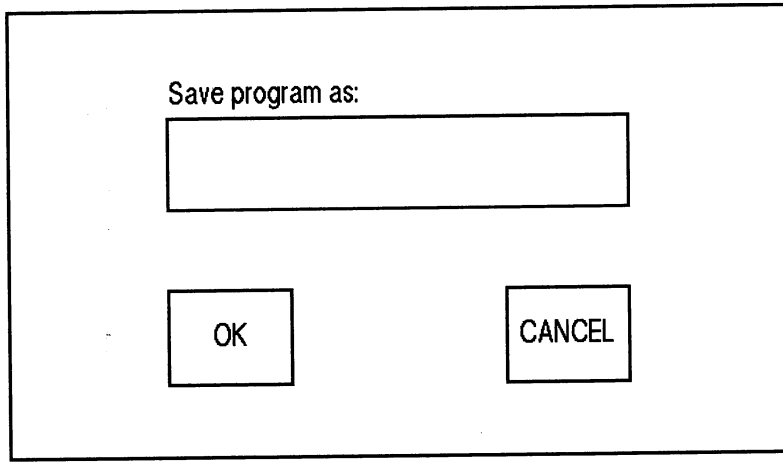


Bild 1.2: *Der Save As-Requester*

Im Textfeld wird immer der Name des zuletzt geladenen Programmes vorgeschlagen, in unserem Fall BASICDemos/Picture. Ist das Programm neu erstellt worden oder wurde zwischenzeitlich *New* ausgewählt, bleibt das Textfeld leer. Klicken Sie nun mit der Maus in dieses Feld und ändern das Wort >Picture< in >Picture3<. Wenn Sie RETURN drücken oder in das OK-Feld klicken, wird die Eingabe ausgeführt. Das Programm wird gespeichert. Wenn Sie sich davon überzeugen wollen, legen Sie die Fenster von Basic nach hinten und überprüfen Sie den Inhalt der Schublade BASICDemos.

Mit *Save As* werden Programme in einem komprimierten Format gespeichert. Wenn Sie ein Programm in einem anderen Format speichern wollen, müssen Sie den Basic-Befehl SAVE im Direkt-Modus eingeben:

SAVE Dateiangabe,*A,P,B*
speichert ein Basic-Programm in wählbarem Format auf Disk

Die einzelnen Optionen bedeuten:

B Programm wird binär gespeichert. Dies ist Standard und kann daher weggelassen werden.

P Programm wird geschützt gespeichert. Mit dieser Option ist Vorsicht geboten. Ein einmal so abgespeichertes Programm kann dann nicht mehr geändert werden.

A Programm wird in Form von ASCII-Zeichen (Text) gespeichert.

Sicherlich werden Sie jetzt denken, was soll dieser Unsinn, ein Programm wahlweise binär oder als ASCII-Zeichenketten zu speichern. Der Grund liegt im Speicherbedarf, welcher für ein binär abgespeichertes Programm am geringsten ist. Für den später erklärten MERGE-Befehl benötigen Sie die Option A. Und P ist sinnvoll, wenn Sie das Programm vor Veränderungen durch andere Personen oder versehentliche Änderungen schützen wollen.

1.3.6 Quit

Mit dem Befehl QUIT können Sie schießlich Ihre Arbeit mit Amiga-Basic beenden. QUIT schließt alle offenen Dateien und Basic-Fenster. Sie landen danach wieder auf der Workbench; oder im CLI-Window, falls Sie von dort gestartet sind. Statt QUIT können Sie auch den Basic-Befehl SYSTEM verwenden

SYSTEM
Das Programm kehrt zum Arbeitstisch (oder CLI) zurück

1.3.7 Umbenennen und Löschen von Programmen

Wenn Sie einem Programm einen anderen Namen geben wollen, steht Ihnen hierzu der Menü-Punkt *Rename* der Workbench oder ein Basic-Befehl zur Verfügung:

NAME Dateiname2 AS Dateiname2
die Datei Dateiname2 erhält den neuen Namen Dateiname2

Im Laufe der Zeit werden sich bei Ihnen auch einige Programme ansammeln, die Sie nicht mehr aufheben wollen. Auch diese könnten Sie mit einem simplen Befehl von der Diskette löschen:

KILL Dateiangabe
löscht die angegebene Datei auf Diskette oder Festplatte

1.4 Ablaufsteuerung

Die Qualität eines Basic-Dialektes läßt sich an der Anzahl seiner Steueranweisungen zur Festlegung des Programmablaufes erkennen. Amiga-Basic kennt sehr viele davon. Aber nur die Anzahl allein garantiert noch keine effektive Programmierung. Wichtig ist dabei auch, wie Programmteile zu Blocks zusammengefaßt werden können, und wie eine Programmzeile aufgebaut ist bzw. vom Basic verarbeitet wird. Beim Amiga-Basic können Sie Zeilennummern eingeben. Sie können es aber genausogut bleiben lassen. Der Basic-Interpreter arbeitet die Zeilen in der Reihenfolge ab, wie sie eingegeben wurden (es sei denn, Steueranweisungen verlangen etwas anderes) und nicht in der Reihenfolge der Numerierung. Auch werden die Zeilen nicht nach den angegebenen Nummern sortiert.

Wenn Sie also zuerst die Zeile 500 und dann die Zeile 10 eingeben, bleiben sie im Listing so hintereinander stehen. Die Zeilennummer stellt nur ein mögliches Sprungziel für Verzweigungs-Operationen dar. Besser ist es aber, wenn Sie für diesen Zweck eine Sprung-Marke verwenden.

1.4.1 Labels

Sprungziele in einem Programm können wir mit lesbaren Texten, genannt Label, versehen. Solche Programm-Marken oder Sprungmarken können aber auch zur besonderen Markierung von wichtigen Programmabschnitten (Strukturierung) eingesetzt werden. Eine Marke muß immer am Anfang einer Zeile stehen. Ihre Länge ist auf 40 Zeichen begrenzt, das erste Zeichen muß ein Buchstabe und das letzte Zeichen ein Doppelpunkt sein. Der Doppelpunkt am Ende eines Labels muß sein, da er einer Sprungmarke erst als solche kenntlich macht. Ähnliche Texte ohne Doppelpunkt hält Basic für Namen von Variablen. Aus einem ähnlichen Grund dürfen im Namen einer Sprungmarke und zwischen Bezeichnung und Doppelpunkt keine Leerstellen stehen. Ansonsten können Sie die Sprungmarken aber völlig beliebig (allerdings sinnvoll) benennen. Durch eine sinnvolle-Bezeichnung wissen Sie immer, was das Programm an dieser Stelle tun wird (oder wenigstens soll). Außerdem sparen Sie sich zusätzliche Erläuterungen.

1.4.2 GOTO, GOSUB

Eine Programm-Marke können Sie mit der GOTO-Anweisung anspringen:

GOTO Marke
unbedingte Programmverzweigung zu einer Sprungmarke/Zeile

Immer, wenn der Basic-Interpreter auf einen GOTO-Befehl stößt, sucht er die Zeile, die mit der entsprechenden Marke beginnt, und setzt in dieser das Programm fort. Mit dem folgenden Beispiel schreiben Sie zum Beispiel ununterbrochen einen Text in die linke obere Bildschirmecke:

```
start:
CLS
PRINT "Amiga Computer"
GOTO start
```

Diesen bekanntesten Befehl für die Ablaufsteuerung sollten Sie aber so selten wie möglich einsetzen. Er verführt nämlich dazu, kreuz und quer im Programm herumzuspringen, so daß Sie schnell die Übersicht verlieren können. Wenn Sie später einmal das Programm verändern wollen, kann es passieren, daß Sie sich selbst nicht mehr auskennen. Sie finden im folgenden noch weit bessere Befehle, die Ihnen helfen, eine ordentliche, übersichtliche Programm-Struktur aufzubauen.

Im Beispiel haben Sie aber auch noch einen weiteren, neuen Befehl kennengelernt:

CLS
löscht das aktuelle Fenster, bringt den Cursor in HOME-Position (links oben)

Den zweiten Befehl für die Steuerung des Programmablaufes haben Sie wahrscheinlich auch schon einmal kennengelernt:

GOSUB Sprungmarke
springt in eine Subroutine, die später wieder zu der Zeile zurückkehrt, in der das GOSUB stand

RETURN ***Sprungmarke***
kehrt von einer Subroutine zurück an die Stelle, von der aus sie (mit GOSUB) aufgerufen wurde

Sie können mit GOSUB also zu einer Subroutine (GOSUB gleich GO to SUBroutine) springen, von welcher Sie wieder eine Subroutine aufrufen können und so weiter. Wichtig ist, daß jede Subroutine oder eine Weiterverzweigung mit einem RETURN beendet wird. Kehren Sie also ohne RETURN an den Ausgangspunkt zurück, bringen Sie den Programmablauf ganz schön durcheinander. Deshalb sind auch GOTOs, die eine Subroutine verlassen, mit äußerster Vorsicht zu genießen.

Eine Subroutine setzt man in der Regel dann ein, wenn ein Programmteil öfter benötigt wird. Das verkürzt Ihr Programm, spart Ihnen unnütze Schreiberei und ist außerdem übersichtlicher.

Subroutinen legen Sie am besten an das Programmende oder zusammen an den Programmanfang und sorgen dafür, daß diese nicht unbeabsichtigt (ohne GOSUB) durchlaufen werden können.

Das folgende Programm zeigt, wie mit GOSUB eine Zeitschleife aufgerufen wird, die den Programmanlauf verlangsamt. Die FOR-NEXT-Schleife für die Verzögerung wird im nächsten Abschnitt näher beschrieben. (Im weiteren Verlauf des Buches werden Sie viele Beispiele für Subroutinen finden, welche sicherlich sinnvollere Anwendungen zeigen.)

```
REM langsamer
'P1.4.2-1

start1:
PRINT "Die Ausgabe dieses Textes"
GOSUB zeit
PRINT "wird verzoegert, da zwischen"
GOSUB zeit
PRINT "jeder Zeile eine Zeitschleife"
GOSUB zeit
PRINT "mit einer Subroutine"
GOSUB zeit
PRINT "aufgerufen wird!"
END

zeit:
FOR t = 1 TO 10000
NEXT t
RETURN
```

In diesem Programm finden Sie eine Steueranweisung, welche verhindert, daß die Subroutine hinter PRINT »aufgerufen wird!« unkontrolliert (ohne GOSUB) angelaufen wird:

END
beendet Programm, schließt alle geöffneten Dateien

Steht der letzte Befehl in der letzten Programmzeile eines Programmes, brauchen Sie keine END-Anweisung einzugeben. Folgen hinter dem »eigentlichen« Ende eines Programms aber noch Subroutinen, ist END unbedingt nötig.

Die ersten beiden Zeilen des Beispielprogramms zeigen keine Wirkung, sondern werden vom Programm überlesen. Es handelt sich um Kommentare (engl. REMarks):

REM ***Kommentartext***
Bemerkungen nach einem REM werden überlesen

Anstelle von REM kann auch das Hochkomma ' (ungeshiftet, links von der RETURN-Taste) verwendet werden. Nach REM werden in einer Programmzeile keine Befehle mehr ausgeführt.

Eine interessante Abart der GOTO- und GOSUB-Anweisungen gestattet uns, programmiert verschiedene Ziele anzuspringen:

ON Zahl GOSUB *Sprungmarke,Sprungmarke,...*
ON Zahl GOTO *Sprungmarke,Sprungmarke,...*
verzweigt abhängig von der Zahl zu der jeweiligen Sprungmarke

Als Zahl sind Werte von 1 bis 255 zulässig. Ist die Zahl zum Beispiel drei, wird in die dritte Sprungmarke verzweigt. Dieses Befehls-Paar ist ideal für die Programmierung von Menü-Verzweigungen. Sie finden es daher in diesem Buch in fast jedem Programm, das Menüs verwendet, eingesetzt.

Folgendes Beispiel soll die Funktion dieses Befehls demonstrieren:

```
REM Textverzweigung
'P1.4.2-2
'Bei Programmbeginn Variable x=0

start:
x=x+1
ON x GOTO text1,text2,text3,text4,ende

text1:
PRINT "Text der ersten Verzweigung"
GOTO start

text2:
PRINT "Text der zweiten Verzweigung"
GOTO start

text3:
PRINT "Text der dritten Verzweigung"
GOTO start

text4:
PRINT "Text der vierten Verzweigung"
GOTO start

ende:
END
```

Beim ersten Durchlauf erhält *x* den Wert 1 und verzweigt in die Anweisung, welche als erste Position hinter ON *x* GOTO steht. Das Programm schreibt den ersten Text und kehrt durch GOTO an den Start zurück. Dort erhält *x* den Wert 2 usw.

1.4.3 FOR-NEXT-Schleifen

FOR Variable=x TO y STEP z

...

NEXT *Variable,Variable,...*

Schleife mit einer festgelegten Anzahl von Durchläufen zur mehrfachen Bearbeitung von Befehlen und Funktionen

Mit dieser Schleife kann man schon allerhand anfangen. Die Variable erhält beim ersten Schleifendurchlauf den Startwert x zugewiesen. Mit STEP legen Sie die Schrittweite für die Durchläufe fest. Wenn Sie zum Beispiel

```
FOR t = 1 TO 10 STEP 2:PRINT t:NEXT t
```

eingeben, wird die Schleife nur fünfmal durchlaufen. Die Variable *t* erhält dabei nacheinander die Werte 1, 3, 5, 7, 9. Die Schleife wird dann verlassen, da beim nächsten Durchgang der Wert 10 (also *y* = Grenzwert) überschritten würde. Wird die Schrittweite *z* nicht angegeben, wird dafür der Wert 1 angenommen. Sie können die Schleifenwerte auch mit fallenden Werten durchlaufen lassen. Dafür muß natürlich der Startwert größer als der Grenzwert sein. Sogar Zahlen mit Kommastellen sind zulässig. Die folgende Schleife wird zum Beispiel sechsmal durchlaufen. Sie erhalten für die Variable *j* die Werte 9, 7.5, 6, 4.5, 3 und 1.5.

```
FOR j = 9 TO 1.5 STEP -1.5:PRINT j:NEXT
```

Sie können mehrere FOR-NEXT-Schleifen ineinanderschachteln. Dabei müssen Sie für die Variablen nur verschiedene Namen verwenden.

Im folgenden Beispiel schachteln wir zwei Schleifen:

```
REM Plus
'P1.4.3-1
FOR j=1 TO 100
 FOR i=1 TO 20
 PRINT j"+"i"="j+i
 NEXT i
 FOR t = 1 TO 10000:NEXT t
 CLS
NEXT j
```

Bei jedem Schleifendurchlauf der äußeren Schleife wird die innere Schleife 20mal durchlaufen, insgesamt also 20 * 100 Mal. Auf dem Bildschirm sehen Sie die Addition der Variablen j der äußeren und der Variablen i der inneren Schleife. Die dritte Schleife dient der Verzögerung, damit das Ergebnis der PRINT-Anweisung abgelesen werden kann.

Das letzte Beispiel zur FOR-NEXT-Schleife zeichnet ein Dreieck:

```
REM Dreieck
'P1.4.3-2

start:
CLS
x=40
FOR y = 1 TO 24
 LOCATE y,x
 FOR i = 1 TO 2*y
 PRINT "X";
 NEXT i
 x=x-1
NEXT y
FOR t= 1 TO 1000:NEXT t'Zeitschleife
GOTO start
```

Die Schleifen-Variable *y* enthält die Werte für die Zeilen. Die Variable *i* der inneren Schleife ist für die Anzahl der Zeichen auf den einzelnen Zeilen verantwortlich, nimmt aber dazu *y* der äußeren Schleife zur Hilfe. Die Positionierung des Cursors erfolgt durch folgende Anweisung:

LOCATE Zeile,Spalte
positioniert den Textcursor im aktuellen Fenster

Die Werte für Zeile und Spalte müssen ganzzahlige Werte sein, die größer als Null sind. Ist Ihnen bei unseren ersten kleinen Programmen aufgefallen, daß einige Zeilen nach innen versetzt sind? Amiga-Basic gestattet bei Schleifen diese formatierte Eingabe. Sie sehen dadurch mit einem Blick auf Ihr Listing immer sofort, welche Befehle zum Innern einer Schleife gehören, auch bei mehrfach geschachtelten Schleifen. Diese Einrückungen, die Sie mit der TAB-Taste erzielen können, erhöhen die Überschaubarkeit eines Programmes ganz beträchtlich.

Noch etwas wird Ihnen vielleicht aufgefallen sein. Wenn Sie das Programm abgeschrieben und die darin enthaltenen Basic-Wörter klein geschrieben hatten, stellten Sie bestimmt mit Verwunderung fest, daß diese beim Weiterschreiben in der nächsten Zeile automatisch in Großschrift erschienen. Alle Basic-Befehle können Sie so im Programm sofort erkennen. Es ist dabei völlig egal, ob Sie diese groß, klein oder in gemischter Groß- und Kleinschreibung eingeben.

Diese scheinbare Spielerei kann Ihnen bei der richtigen Eingabe der Basic-Befehle sehr helfen. Sie sollten immer einen Blick auf das bisher Geschriebene werfen. Wenn die Basic-Wörter nicht in Großbuchstaben erscheinen, nachdem Sie die RETURN-Taste betätigt haben, haben Sie einen Eingabefehler gemacht. In diesem Fall prüfen Sie bitte, ob Sie alle Befehle in der entsprechenden Zeile richtig geschrieben haben, und ob hinter jedem eine Leerstelle kommt.

Hinter fast allen Basic-Befehlen muß eine Leerstelle folgen.

Damit Sie sich bei der Programmierung nicht mit der Groß- und Kleinschreibung herumärgern müssen, haben die Schöpfer von Amiga-Basic noch ein weiteres Bonbon eingebaut. Tippen Sie einmal in Ihr Programm bei der Zeile "GOTO start" das "s" als Großbuchstabe ein, und Sie werden sehen, daß bei der Sprungmarke das Wort Start auch groß erscheint. Diesen praktischen Effekt können Sie auch bei den Variablen und Feldvariablen beobachten. Ein und dieselbe Variable wird also im ganzen Programm immer gleich geschrieben. Das gilt gleichermaßen für Eingaben im Direkt-Modus im Ausgabefenster.

1.4.4 Die IF-Anweisung

Jetzt kommen wir zu den wichtigsten Anweisungen von Basic. Mit diesen können Sie verschiedene Bedingungen des Programmes oder der Eingabe auswerten und je nach Resultat verschiedene Entscheidungen treffen. Schauen wir uns zuerst die Befehle an:

IF Ausdruck GOTO Label ELSE Anweisung

Wenn Ausdruck wahr ist, findet ein Sprung zu Label GOTO statt. Falls nicht, wird die Anweisung hinter ELSE ausgeführt

IF Ausdruck THEN Anweisung1 ELSE Anweisung2

Wenn Ausdruck wahr ist, wird Anweisung1 hinter THEN, wenn Ausdruck falsch ist, Anweisung2 hinter ELSE ausgeführt

Die IF-THEN-ELSE-Anweisung darf in dieser Form allerdings höchstens eine Zeile (255 Buchstaben) lang werden. Das kann manchmal »eng« werden. Amiga-Basic kennt deshalb noch eine Form der IF-THEN-ELSE-Anweisung, bei der zwischen THEN und ELSE beliebig viele Anweisungen stehen können.

```
IF Ausdruck1 THEN
Zeilen mit Anweisungen
ELSEIF Ausdruck2 THEN
Zeilen mit Anweisungen
ELSEIF Ausdruck3 THEN
...
ELSE
Zeilen mit Anweisungen
END IF
...
```

Bei dieser Form des IFs überprüft Basic mehrere Bedingungen (Ausdrücke). Ist eine davon wahr, werden die Befehle in den folgenden Zeilen ausgeführt, bis ein ELSEIF, ELSE oder END IF auftaucht. Dann springt Basic hinter die END-IF-Anweisung. Ist keiner der Ausdrücke wahr, werden hingegen die Befehle zwischen ELSE und END IF ausgeführt. Für alle verschiedenen Versionen der IF-Anweisung gilt ein Ausdruck dann als wahr, wenn sein Wert nicht gleich Null ist und als unwahr oder falsch, wenn der Wert Null ist (siehe Kapitel 1.6.2).

Bevor die einzelnen Anweisungen mit ein paar Beispielen verdeutlicht werden, noch ein Hinweis: ELSEIF muß zusammengeschrieben werden! Im Basic-Handbuch steht dies zwar anders – dort steht zwischen ELSE und IF eine Leerstelle – das ist aber falsch!

Schauen wir uns nun zunächst die erste Form der IF-Anweisung an:

```
c=1:IF c GOTO ausgeben ELSE PRINT "falsch"
END
ausgeben: PRINT "Wert ist richtig"
```

Wir fragen also:

Wenn c wahr ist, gehe nach *ausgeben*, andernfalls schreibe "falsch"

Da wir c=1 (wahr) vorgegeben haben, springt das Programm zu *ausgeben*. Probieren Sie es einmal mit c=0. Das zweite Beispiel verwendet die zweite Form der IF-Anweisung:

```
a=1:b=0
PRINT "a=1, b=0"
IF a THEN PRINT "a=wahr" ELSE PRINT "a=falsch"
IF b THEN PRINT "b=wahr" ELSE PRINT "b=falsch"
```

Diese beiden einfachen Beispiele sind sicherlich leicht zu verstehen. Mit dem nächsten Beispiel realisieren wir einen etwas komplizierteren IF-Block. Wir wollen den Bildschirm, bis auf ein Loch in der linken oberen Ecke, mit Zeichen füllen. Dabei müssen wir im Programm einige Bedingungen auswerten (die Vergleichs-Operatoren finden Sie im Abschnitt 1.5 erklärt):

```
REM freilassen
'P1.4.4-1

anfang:
CLS
x=1:y=1 'Spalte=1:Zeile=1

start:
IF y>10 THEN 'Wenn Zeile groesser 10 dann
  IF y>23 THEN ' Wenn Zeile groesser 23 dann
       GOTO anfang ' gehe zu anfang (Neustart)
  ELSEIF x>80 THEN ' Wenn Spalte groesser 80 dann
       PRINT :y=y+1:x=1 ' naechste Zeile:Zle=Zle+1:Spalte=1
  ELSE ' andernfalls
       PRINT "#";:x=x+1 ' Schreibe "#";:Spalte=Spalte+1
  END IF ' Ende innerer IF-Block
```

```
ELSEIF y<11 THEN 'Wenn Zeile kleiner 11 dann
  IF x>80 THEN ' Wenn Spalte groesser 80 dann
       PRINT :y=y+1 :x=1 ' naechste Zeile:Zle=Zle+1:Spalte=1
  ELSEIF x>20 THEN ' Wenn Spalte groesser 20 dann
       PRINT "#";:x=x+1 ' Schreibe "#";:Spalte=Spalte+1
  ELSE ' andernfalls
       PRINT " ";:x=x+1 ' Schreibe "#";:Spalte=Spalte+1
  END IF ' Ende innerer IF-Block
END IF 'Ende aeusserer IF-Block
GOTO start
```

Wenn Sie die Erklärungen in den einzelnen Zeilen aufmerksam lesen, werden Sie schnell durchschauen, wie ein solcher Block von IF-Anweisungen aufgebaut werden kann. Ihren eigenen Versuchen, vielleicht sogar mit noch größeren Schachtelungstiefen, steht nun nichts mehr im Wege.

1.4.5 Die WHILE-WEND-Schleife

Diese Schleife bietet uns interessante Wege zur Programmgestaltung:

WHILE Ausdruck
Zeilen mit Anweisungen
WEND
...
beliebig viele Anweisungen werden in einer Schleife ausgeführt, solange die Bedingung von Ausdruck wahr ist

Wenn Basic auf eine WHILE-Anweisung trifft, wird überprüft, ob Ausdruck wahr ist. Wenn ja, werden alle Anweisungen bis WEND ausgeführt. Falls nicht, werden diese Anweisungen übersprungen und das Programm fährt hinter WEND fort. Jedesmal, wenn Basic WEND erreicht, springt es zurück zur zugehörigen WHILE-Anweisung und überprüft diese erneut. Ist der Ausdruck wieder wahr, werden die Anweisungen erneut ausgeführt und so weiter. Innerhalb einer WHILE-WEND-Anweisung werden üblicherweise Variablen so verändert, daß die Bedingung der WHILE-Anweisung (Ausdruck) irgendwann einmal falsch wird – sonst läuft die Schleife ewig weiter.

WHILE-WEND-Anweisungen können – wie IF-Anweisungen – auch geschachtelt werden. Schauen Sie sich aber zunächst einmal das folgende einfachere Beispiel an:

```
REM Alphabet
a=65
WHILE a<91
 PRINT CHR$(a);
 a=a+1
WEND
```

Dieses kleine Programm schreibt uns das große Alphabet auf den Bildschirm. Die Variable *a* erhält zu Beginn den Wert 65. Bei jedem Schleifendurchlauf erhöht sich dieser Wert um 1. Sobald *a* den Wert 91 erreicht hat, wird die Schleife verlassen. Die Zahlen in der Variablen stellen Code-Zahlen für den internationalen Zeichen-Code ASCII dar. CHR$ ergibt zu einer solchen Zahl den entsprechenden Buchstaben.

Nach diesem einfachen Programm folgt nun ein Beispiel, in dem die WHILE-WEND-Anweisungen in geschachtelter Form auftauchen:

```
REM Pfeil
'P1.4.5-1

start:
x=1:y=1
WHILE y<21
  WHILE x<79
  LOCATE y,x
       PRINT ">";
       FOR i=1 TO 10:NEXT
       PRINT CHR$(8)
       PRINT CHR$(32)
       x=x+1
  WEND
   BEEP
  y=y+1
  WHILE x>1
       LOCATE y,x
       PRINT "<";
       FOR i=1 TO 10:NEXT
       PRINT CHR$(8)
       PRINT CHR$(32)
       x=x-1
  WEND
  BEEP
  y=y+1
WEND
GOTO start
```

Das Programm läßt einen kleinen Pfeil von der linken zur rechten Bildschirmseite wandern. Dort »wendet« der Pfeil mit einem Piepser und läuft eine Zeile tiefer zur linken Bildschirmseite zurück. Das Ganze wiederholt sich, bis der untere Bildschirmrand erreicht ist.

Die äußere WHILE-WEND-Schleife zählt die Variable *y* für jede zweite Zeile um eins weiter. Hat *y* den Wert 21 erreicht, wird diese Schleife verlassen und das Programm beginnt von vorne. Die erste innere WHILE-WEND-Schleife positioniert den Cursor und schreibt einen Pfeil (>). Anschließend sorgt eine kleine Zeitschleife dafür, daß der Pfeil etwas länger (und besser) zu sehen ist. Der folgende ASCII-Code 8 entspricht der BACK-SPACE-Taste. Damit wird der Pfeil gelöscht. Der ASCII-Code 32 schreibt einen

Leerschritt, damit sind wir wieder auf der alten Position. Die Variable *x* wird um 1 erhöht. Hat diese Variable den Wert 79 für den rechten Bildschirmrand, wird die WHILE-WEND-Schleife verlassen. Es wird ein Piepser ausgelöst und eine Zeile weitergezählt. Die zweite innere WHILE-WEND-Schleife bringt einen linken Pfeil an den linken Bildschirmrand. Fehlt uns nur noch die Erklärung für das akustische Signal:

BEEP
erzeugt einen Warnton und kurzes Aufblitzen des Bildschirmes

Sie sehen, auch die WHILE-WEND-Schleife ist recht einfach einzusetzen. Was Sie damit sonst noch alles anstellen können, erfahren Sie im nächsten Kapitel.

1.4.6 Wer erweckt Dornröschen?

Wollen Sie den Programmablauf kennenlernen, mit dem Sie ein, sagen wir einmal, zehn Seiten langes Basic-Programm steuern können? Ja? Gut, ich will es Ihnen nicht vorenthalten:

```
WHILE 1:SLEEP :WEND
```

Ich hoffe, Sie sind nicht enttäuscht. Diese Zeile – obwohl nur wenige Buchstaben lang – steuert den ganzen Ablauf eines beliebig großen Programms! Sie müssen nur vorher das Programm auf bestimmte Unterbrechungen vorbereiten. Schauen wir zuerst, was es mit dem Schlafen auf sich hat:

SLEEP
Programm wartet, bis eines der folgenden Ereignisse eintritt:
- Betätigung der Auswahltaste der Maus
- Betätigung der Tastatur-Taste
- Kollision grafischer Objekte
- Auswahl eines Menü-Titels
- Ablauf einer vorgegebenen Zeit

Diese Liste von Ereignissen, welche den Schlaf der Schleife stören können, sind bis auf eine Ausnahme (Tastatur-Taste) sogenannte Unterbrechungsereignisse. Dieses monströse Wort steht für ein gewaltiges Potential an Möglichkeiten für die Ablaufsteuerung von Programmen. Mit solchen Ereignissen können Sie Ihre eigenen Programme mit Pull-down-Menüs steuern, die Maustaste abfragen und vieles mehr.

Folgende Ereignisse können programmiert werden:

- TIMER reagiert auf das Eintreten einer vorgegebenen Zeit

- MOUSE reagiert auf die Auswahltaste der Maus
- MENU reagiert auf die Menütaste der Maus
- BREAK reagiert auf die rechte Amigataste-Punkttaste
- COLLISION reagiert auf die Kollision grafischer Objekte

Die Steuerung übernehmen folgende Anweisungen:

<Ereignis> ON	schaltet Reaktionsfähigkeit ein
<Ereignis> OFF	schaltet Reaktionsfähigkeit aus
<Ereignis> STOP	unterbricht Reaktionsfähigkeit vorübergehend

Statt <Ereignis> setzen Sie dabei im Programm eines der Schlüsselworte aus obiger Aufstellung ein. Zuerst müssen Sie jedoch festlegen, wohin das Programm bei einem Ereignis verzweigen soll:

```
ON <Ereignis> GOSUB Label
```

In der Subroutine wird dann festgelegt, was bei dem Ereignis passieren soll. Anschließend aktiviert man die Unterbrechungsreaktionsfähigkeit mit "<Ereignis> ON". Beide Anweisungen zusammen können so aussehen:

```
ON MENU GOSUB Sprungmarke: MENU ON
```

Sobald "<Ereignis> ON" ausgeführt ist, prüft der Basic-Interpreter vor jeder neuen Anweisung, die ausgeführt wird, ob das vorgegebene Ereignis eingetreten ist. Ist dies der Fall, verzweigt das Programm (ohne GOTO und GOSUB) zu der hinter "<Ereignis> ON" angegebenen Sprungmarke. Bei dieser Sprungmarke muß eine Subroutine beginnen, die mit einem RETURN endet. Damit wird dann an die Stelle zurückgesprungen, an der das Ereignis von Basic festgestellt wurde, und das Programm wird dort fortgesetzt.

Dieses Kapitel sollte nur einen Einblick in die Technik der Unterbrechungsereignisse vermitteln. Weitere Einzelheiten und vor allem Beispiele finden Sie bei den Erläuterungen zu den einzelnen Anweisungen. Wie Sie die verschiedenen Ereignisse wirklich zur Steuerung Ihrer Programme einsetzen können, wird anhand dieser Beispiele viel deutlicher, als es trockene Erklärungen zeigen können.

1.5 Variablen

In fast allen Programmen, welche Sie bisher kennengelernt haben, sind Ihnen Buchstaben oder Wörter begegnet, die alle mit Zahlen in einem Zusammenhang standen. Mehrmals fanden Sie in den Erläuterungen dazu das Wort Variable, das sich auf diese Namen bezog. Was es damit auf sich hat, erfahren Sie in diesem Kapitel.

1.5.1 Damit das Kind einen Namen bekommt

Bisher haben Sie gehört, daß ein Text, der den PRINT-Befehl ausgeben soll, in Anführungszeichen gesetzt werden muß. Probieren Sie einmal aus, was passiert, wenn wir die Anführungszeichen weglassen:

```
print Commodore
```

Statt des Wortes Commodore erhalten Sie:

```
0
OK
```

Sind Sie überrascht? Geben Sie nun einmal ein:

```
Commodore=1+5
print Commodore
```

Dieses Mal erhalten Sie die Zahl Sechs, obwohl Sie doch eigentlich das Wort Commodore sehen wollten.

Bestimmt haben Sie sich schon einmal in Ihrem schulischen, privaten oder geschäftlichen Bereich mit Formeln herumgeschlagen. Dabei werden für Werte, die verschieden sein können, also variabel sind, Buchstaben eingesetzt. Genau damit haben wir es hier zu tun, mit VARIABLEN. Bei dem kleinen Beispiel hat der Basic-Interpreter das Wort Commodore als Variable aufgefaßt und hat damit gerechnet. Variablennamen können auf dem Amiga nämlich nicht nur ein oder zwei Buchstaben lang sein – wie bei vielen anderen Computern üblich. Amiga-Basic ist in der Lage, 40 Zeichen lange Variablennamen voneinander zu unterscheiden. Sie dürfen nur keinen Basic-Befehl als Variablennamen verwenden. Als Teil des Namens ist er allerdings zulässig. Geben Sie, um das zu überprüfen, einmal ein:

```
cls=3
```

Wenn Sie bei der Fehlermeldung des Amiga-Basic erschrocken sind, drehen Sie den Lautstärkeregler an Ihrem Monitor zurück. Diesen Ton werden nicht nur Einsteiger öfter hören. Sie erhalten die Fehlermeldung Syntax Error. Löschen Sie nun die Fehlermeldung, indem Sie in das OK-Feld klicken. Probieren Sie nun:

```
clsausfuehren=3
```

Dieses Mal akzeptiert Amiga-Basic den Variablennamen mit einem freundlichen OK.

1.5.2 Variablen-Typen

Amiga-Basic unterscheidet grundsätzlich zwischen

- Zeichenketten- oder String-Variablen und
- numerischen Variablen.

Für alle Texte stehen die String-Variablen zur Verfügung. Sie dürfen aus maximal 32767 einzelnen Zeichen bestehen und werden durch ein $ als letztes Zeichen im Namen kenntlich gemacht:

```
text1$="bitte warten"
a$="Das Ergebnis lautet: "
```

Bevor Sie einer Zeichenketten-Variablen einen Text zugewiesen haben, enthält diese eine leere Zeichenkette mit der Länge Null:

```
a$=""
```

Numerische Variablen haben, bevor ihnen ein Wert zugewiesen wurde, den Wert Null. Den Wert, den eine Variable hat, können Sie entweder im Direkt-Modus, also durch eine Tastatureingabe oder durch einen Basic-Befehl ändern, indem Sie der Variablen einen neuen Wert zuweisen. Für die Zuweisung eines Wertes an eine Variable gibt es den folgenden Basic-Befehl:

LET Variable=Ausdruck
weist den Wert eines Ausdrucks einer Variablen zu

Das Befehlswort LET können Sie, wie Sie an den ersten Beispielen gesehen haben, auch weglassen. Das Gleichheitszeichen = reicht völlig aus, eine Zuweisung zu bewirken. (LET gibt es eigentlich nur noch aus historischen Gründen; das heißt, frühere Basic-Interpreter auf anderen Computern benötigten diesen Befehl.) Sie werden es deshalb kaum in einem Programm finden. Die numerischen Variablen unterscheidet man auch noch danach, wie genau der Wert sein muß. In der folgenden Übersicht sind alle wichtigen Punkte, die Variablentypen betreffen, zusammengefaßt. Sie soll Ihnen auch als kleines Nachschlagewerk bei später auftretenden Unklarheiten dienen.

VARIABLEN TYPEN

Typ	Bezeichnung	Kennung	Speicher-bedarf	Gültig für
INT	kurze Ganzzahl	%	2 Byte	Zahlen ohne Dezimalpunkt von -32768 bis +32767
LNG	lange Ganzzahl	&	4 Byte	Zahlen ohne Dezimalpunkt von-2147483648bis+...647
SNG	einfache Genauigkeit	!	4 Byte	bis zu 7 Stellen, von 1.18*10^-38bis3.4*10^38
DBL	doppelte Genauigkeit	#	8 Byte	bis zu 16 Stellen, von 2.23*10^-308bis1.79*10^308
STR	Zeichenkette	$	5 Byte	bis zu 32767 Zeichen, ein-
		+	Länge	geschlossen in ""

Tabelle 1: *Variablentypen und ihre Kennungen*

Möchten Sie eine Variable als kurze Ganzzahl kennzeichnen, müssen Sie also zum Beispiel schreiben:

```
Zeile%=20
```

Hat ein Variablenname keine besondere Endung, nimmt Amiga-Basic an, daß es sich um eine Zahl mit einfacher Genauigkeit handelt.

Wenn Sie in Ihrem Programm nur einen Typ, oder hauptsächlich einen Typ von Variablen einsetzen wollen, können Sie Basic vorschreiben (definieren), daß alle Variablennamen, die keine Endung besitzen, nicht als Zahl mit einfacher Genauigkeit, sondern als Variablen des gewünschten Typs aufgefaßt werden sollen.

DEFTyp Buchstabe-*Buchstabe*

Legt für alle Variablennamen ohne Typ-Endung und mit bestimmten Anfangsbuchstaben fest, welchen Typ sie haben sollen

Wenn Sie zum Beispiel in Ihrem Programm nur mit Variablen als kurze Ganzzahl arbeiten wollen, definiert Ihnen die Anweisung DEFINT A-Z alle Variablen als kurze Ganzzahl. (Die Buchstaben können wahlweise groß oder klein geschrieben werden.) Hätten Sie nur einen Buchstaben, zum Beispiel A angegeben, so würden alle Variablen, welche mit dem Buchstaben A beginnen, als kurze Ganzzahl definiert. Würden Sie auf diese Anweisung verzichten, bliebe Ihnen nichts anderes übrig, als hinter jeder Variablen die Kennung anzugeben. Denken Sie auch daran, daß Variablen verschiedenen Typs verschieden viel Speicher benötigen. Variablen mit einer Genauigkeit, die eigentlich nicht erforderlich ist, fressen deshalb überflüssig viel Speicherplatz. Wenn Sie innerhalb Ihres Programmes doch Variablen mehrerer Typen benötigen, so steht dem trotz der DEF-Anweisung nichts im Wege. Diese gilt ja nur für Variablennamen ohne Typ-Kennung am Ende. Wenn Sie zum Beispiel mit DEFINT alle Zahlen zu kurzen Ganzzahlen definiert haben, und Sie benötigen eine Zahl mit einfacher Genauigkeit, schreiben Sie hinter den Namen der entsprechenden Variablen einfach immer ein Ausrufezeichen (!).

Nach so viel Theorie aber nun ein kleines Beispiel:

```
REM Variablen
'P1.5.2-1
text1$="und "
text2$="Die Addition "
text3$="Die Division "
text4$="Die Multiplikation "
text5$="Die Potenzierung "
text6$="von "
text7$="mit "
text8$="durch "
text9$="ergibt "
FOR i= 1 TO 10
```

```
  a=i+2
  CLS
  PRINT text2$ text6$ a text1$ i text9$ a+i:PRINT
  PRINT text4$ text6$ a text7$ i text9$ a*i:PRINT
  PRINT text3$ text6$ a text8$ i text9$ a/i:PRINT
  PRINT text5$ text6$ a text7$ i text9$ a^i:PRINT
  FOR t=1 TO 20000:NEXT t 'Zeitschleife
NEXT i
```

Nachdem zu Programmbeginn den einzelnen String-Variablen bestimmte Zeichenketten zugewiesen wurden, läuft das Programm in einer Schleife ab und weist der Variablen a bei jedem Durchgang neue Werte zu. Lassen Sie nun das Programm einmal laufen. Beim letzten Bild erhalten Sie bei der Potenzierung eine etwas komisch aussehende Zahl

```
6.191736E+10.
```

Vielleicht haben Sie ein ähnliches Ergebnis schon einmal auf Ihrem Taschenrechner erhalten. Wenn eine Zahl länger wird als für den Variablen-Typ vorgesehen, wendet der Amiga diese Exponential-Darstellung an. Das Zeichen E ist dabei wie folgt aufzufassen:

```
6.191736E+10 = 6.191736*10^10 = 61917360000
```

Die Zahl hinter E ist also als die Anzahl der Stellen aufzufassen, um die Sie das Komma in der Zahl vor dem E nach rechts schieben müssen, um das wirkliche Ergebnis der Rechnung zu erhalten. Stoßen Sie einmal auf eine Zahl mit einem Minus hinter dem E, dann ist die Zahl für eine genaue Darstellung zu klein. Auch hierzu schauen wir uns ein Beispiel an:

```
4.123654E-8 = 4.123654*10^-8 = 0.00000004123654
```

Die Zahl hinter E ist in diesem Fall als die Anzahl der Stellen aufzufassen, um die Sie das Komma in der Zahl vor dem E nach links verschieben müssen.

Die verschiedenen Datentypen geben Ihnen bei der Programmierung ein hohes Maß an Flexibilität. Sie können für eine Variable stets den Typ wählen, der am besten geeignet ist – zum Beispiel am wenigsten Speicher benötigt. Nun kann es sich in Ihren Programmen aber ergeben, daß Sie numerische Variablen, die mit einer Genauigkeit abgespeichert sind, in einer anderen Genauigkeit benötigen. Diese Umwandlung eines Wertes von einer Genauigkeitsstufe in eine andere können Sie mit den sogenannten "Konvertierungsbefehlen" erreichen.

v=CSNG(Ausdruck)
wandelt den Wert von Ausdruck in eine einfachgenaue Zahl um

Wenn Sie zum Beispiel

```
a#=13.12394913:PRINT CSNG(a#)
```

eingeben, erhalten Sie

```
13.12395
```

v=CDBL(Ausdruck)
wandelt den Wert von Ausdruck in eine doppeltgenaue Zahl um

Wenn Sie zum Beispiel

```
a=25.1:PRINT CDBL(a)
```

eingeben, erhalten Sie

```
25.10000038146973
```

Die Zahl kann natürlich nach der Wandlung nicht genauer werden als sie vorher war. Die zusätzlich angezeigten Dezimalstellen sind zum Zeitpunkt der Wandlung ohne Bedeutung.

Auch für den umgekehrten Weg gibt es zwei Funktionen:

v=CINT(Ausdruck)
wandelt den Wert von Ausdruck (eventuell durch Rundung) in eine kurze Ganzzahl um

Wenn Sie zum Beispiel

```
PRINT CINT(34.5)CINT(35.5)CINT(12345.123456789#)
```

eingeben, erhalten Sie

```
34  36  12345
```

v=CLNG(Ausdruck)
wandelt den Wert von Ausdruck (eventuell durch Rundung) in eine lange Ganzzahl um

Wenn Sie zum Beispiel

```
PRINT CLNG(34.5)CLNG(35.5)CLNG(123456789.123456789#)
```

eingeben, erhalten Sie

```
34  36  123456789
```

Die beiden Funktionen CINT und CLNG runden auf, wenn der ganzzahlige Teil ungerade ist. Ist er gerade, wird abgerundet.

1.5.3 Eine Besonderheit – die Feldvariablen

Damit ist das Thema Variablen aber noch nicht erschöpft. Es gibt nämlich auch noch die Feldvariablen, auch indizierte Variablen genannt. Eine einzige Feldvariable (auch kurz »Feld« genannt) enthält eine vorher definierte Anzahl von numerischen Werten oder Strings. Man bezeichnet einen einzelnen Wert aus dem Feld auch als Feldelement. Es reicht nicht, nur den Namen der Feldvariablen anzugeben, wenn man den Wert eines Feldelementes benötigt. Statt dessen muß man hinter den Namen der Feldvariablen in Klammern schreiben, den wievielten Wert man haben möchte. Diese Zahl in Klammern hinter dem Feldnamen nennt man auch »Index«. Folgendes Schaubild zeigt, wie eine Feldvariable namens b mit 8 Feldelementen aufgebaut ist.

b(0)	b(1)	b(2)	b(3)	b(4)	b(5)	b(6)	b(7)

Tabelle 1.1: *Der Aufbau einer Feldvariablen*

Wenn Sie den Feldelementen Werte zuweisen (z.B. b(0)=78 und b(6)=43), bleiben die Werte so lange gespeichert, bis Sie diese ändern oder das Feld löschen. In dieser Hinsicht unterscheiden sich Feldvariablen nicht von anderen Variablen. Damit der Amiga den Speicherbedarf von Feldern einplanen kann, müssen Felder vor der Anwendung »dimensioniert« werden:

DIM SHARED Variable(Elemente),*Variable(Elemente)*
legt die maximale Anzahl von Elementen, die Feldvariablen enthalten dürfen, fest, und reserviert den Speicherbedarf dafür

Bevor Sie Feldvariablen benützen, müssen Sie immer erst die DIM-Anweisung anwenden. Der Computer erlaubt maximal 11 Elemente ohne Dimensionierung. Wenn Sie hinter DIM den Zusatz SHARED verwenden, werden die Variablen als »globale Felder« dimensioniert. Sie bleiben dann auch erhalten, wenn das Programm, in dem diese Anweisung steht, ein anderes (Unterprogramm) aufruft. Ohne SHARED werden in einem solchen Fall alle Variablen gelöscht.

Bevor Sie sich ein kleines Programm zu diesem Thema ansehen, sollen Sie nun noch erfahren, wie Sie Werte in die Variablen eines Programms bekommen. Im wesentlichen

stehen Ihnen dazu zwei Möglichkeiten offen: Werte werden vom Programm errechnet oder die Werte werden beim Benutzer des Programms erfragt und von diesem über die Tastatur eingegeben. Da im folgenden Programm der dritte Weg gewählt wurde, sollten Sie sich zunächst die dafür vorgesehene Anweisung anschauen:

INPUT *"Text"*; Variable1, *Variable2*,...
INPUT ; Variable1, *Variable2*,...
gibt Text aus und liest dann Variablenwerte von der Tastatur ein

Der Text in der Input-Anweisung dient als sogenannter »Prompt«. Er sagt dem Benutzer, was er eingeben soll. Danach folgt eine Liste von Variablen, die die eingegebenen Werte aufnehmen. Wenn Sie statt des Strichpunkts ein Komma zur Trennung von Text und Variable 1 verwenden, erscheint nur der blinkende Eingabecursor auf dem Bildschirm. Verwenden Sie ein Semikolon, folgt hinter dem Text noch ein Fragezeichen. Werden mehrere Variablen mit einem INPUT eingelesen, so müssen auch die Eingaben — wie die Variablennamen – durch ein Komma getrennt werden. Im ersten Programm mit der INPUT-Anweisung wird zum Beispiel mit einer Anweisung nach einer kurzen Ganzzahl p% und nach einem String o$ gefragt.

```
REM Postleit
'P1.5.3-1
DIM post$(999)

start:
PRINT "Ende = 0,0"
INPUT "bitte dreistellige Postleitzahl, Ort eingeben:
";p%,o$
CLS
IF p%>999 OR p%<0 GOTO start
post$(p%)=o$
IF p%>0 THEN start

abfrage:
PRINT "Ende = 0"
INPUT "bitte Postleitzahl fuer gesuchten Ort eingeben: ";p%
CLS
IF p%>999 OR p%<0 THEN
 GOTO abfrage
ELSEIF post$(p%)="" THEN
 PRINT "Fuer die Postleitzahl "p%"wurde kein Ort ein-
gegeben"
ELSE
 PRINT "Die Ort mit der Postleitzahl "p%"heisst: "post$(p%)
END IF
IF p%>0 GOTO abfrage
END
```

Zuerst dimensionieren wir ein Feld post$ mit 1000 Stringelementen. Mit INPUT werden Sie dann nach bis zu 999 verschiedenen Postleitzahlen mit den zugehörigen Orten gefragt. Unter der Feldvariablen post$ werden die einzelnen Orte gespeichert. Wenn Sie mit der Eingabe fertig sind, geben Sie 0,0 ein. Nun können Sie durch Eingabe der Postleitzahl den zugehörigen Ort abrufen. Findet das Programm dabei einen Leer-String, weiß es, daß kein Name eingegeben wurde.

Felder können auch mehrere Dimensionen haben. Es sind bis zu 255 Dimensionen möglich. Jede Dimension darf 32767 Elemente aufweisen. Versuchen Sie bitte nicht, sich diese Anzahl bildlich vorzustellen. Das menschliche Gehirn wird maximal mit drei Dimensionen fertig. Der Amiga kennt solche Probleme nicht. Nachfolgend finden Sie die schematische Darstellung eines zweidimensionalen Feldes. Man spricht hier auch von Tabellen, da man sich die beiden Dimensionen als Spalten und Zeilen vorstellen kann:

c(0,0)	c(0,1)	c(0,2)	c(0,3)	c(0,4)	c(0,5)	c(0,6)	c(0,7)
c(1,0)	c(1,1)	c(1,2)	c(1,3)	c(1,4)	c(1,5)	c(1,6)	c(1,7)
c(2,0)	c(2,1)	c(2,2)	c(2,3)	c(2,4)	c(2,5)	c(2,6)	c(2,7)
c(3,0)	c(3,1)	c(3,2)	c(3,3)	c(3,4)	c(3,5)	c(3,6)	c(3,7)

Tabelle 1.2: *Eine zweidimensionale Feldvariable*

Im nächsten Programm wird gleich ein zweidimensionales Feld angewendet. Da die Eingabe vieler Werte von Hand doch recht mühsam ist, werden dieses Mal die Werte der Variablen berechnet. Das Feld drei% enthält die Flächen, die rechtwinklige Dreiecke mit Höhen und Breiten zwischen 1 und 20 haben. Der erste Index des Feldes steht dabei für die Grundlinie des Dreieckes und der zweite für die Höhe.

```
REM Dreiecksflaeche
'P1.5.3-2
PRINT "ich rechne"
DIM drei(20,20)
FOR i%=1 TO 20
 FOR j%=1 TO 20
 drei(j%,i%)=j%*i%/2
NEXT j%,i%
start:
INPUT "bitte Zahlen von 1 bis 20 eingeben - Basis, Hoehe:
";b%,h%
CLS
IF b%>20 OR h%>20 GOTO start
PRINT "Ein Dreieck mit der Basis von ";b%
PRINT "und einer Hoehe von "h%;
PRINT "hat einen Inhalt von: ";
PRINT drei(b%,h%)
GOTO start
```

Nach der Dimensionierung werden in den beiden geschachtelten FOR-NEXT-Schleifen die möglichen Dreiecksflächen berechnet. Sie können dann die Basis und die Höhe eingeben, und das Programm gibt die dazugehörige Dreiecksfläche aus. Bei dieser Anfrage wird die Dreiecksfläche nicht berechnet, sondern in dem zweidimensionalen Feld nachgesehen. Das Programm hält übrigens nicht von selbst an, sondern muß mit dem Menübefehl STOP abgebrochen werden.

Als letztes Beispiel wollen wir uns an ein dreidimensionales Feld wagen. Ein solches Feld können Sie sich am besten als einen Quader vorstellen. Die ersten beiden Dimensionen bilden die Grundfläche und die dritte die Höhe des Quaders. Da das noch recht gut vorstellbar ist, berechnen wir wieder alle möglichen Quader-Inhalte mit Kantenlängen zwischen 1 und 20 und legen die Ergebnisse in einem dreidimensionalen Feld ab:

```
REM Quader
'P1.5.3-3
PRINT "ich rechne"
DIM Quader%(20,20,20)
FOR i%=1 TO 20
 FOR j%=1 TO 20
 FOR k%=1 TO 20
 Quader%(k%,j%,i%)=k%*j%*i%
NEXT k%,j%,i%

start:
INPUT "bitte Zahlen von 1-20 eing.-
Laenge,Breite,Hoehe:";l%,b%,h%
CLS
IF 1%>20 OR b%>20 OR h%>20 GOTO start
PRINT "Ein Quader mit den Kantenlaengen",
PRINT l%;"*";b%;"*";h%;
PRINT "hat einen Inhalt von: ";
PRINT Quader%(l%,b%,h%)
GOTO start
```

Sie sehen, daß das Programm nicht komplizierter ist als das Beispiel der Flächenberechnung. Hier berechnen wir nach der Dimensionierung in den geschachtelten FOR-NEXT-Schleifen die Quader-Inhalte. Sie können die Werte für Länge, Breite und Höhe eingeben, worauf der Inhalt des Quaders auf dem Bildschirm erscheint. Ich hoffe, daß Sie an den Beispielen sehen, daß auch mehrdimensionale Felder einfach zu programmieren sind.

1.6 Basic 1 * 1

Zum Schluß dieses Abschnittes über die Fundamente des Basic sollen Sie mit Hilfe von Basic ein wenig in die Mathematik hineinschnuppern. Dieses Gebiet wurde im bisherigen Verlauf dieses Buches schon einige Male gestreift. Jetzt sollen die ersten Eindrücke etwas vertieft werden.

1.6.1 Der Ober sticht den Unter

In einer Berechnung, die ein Basic-Programm ausführen soll, können Sie die Ihnen aus der Mathematik bekannten Rechenoperationen verwenden. Wenn Sie in einem Programm etwas berechnen wollen, müssen Sie wissen, in welcher Reihenfolge der Interpreter die arithmetischen Operatoren abarbeiten wird. In der folgenden Aufstellung sind die wichtigsten Operatoren in der mathematischen Hierarchie zusammengefaßt. Dabei hat dann die Potenzierung die höchste und die Addition mit der Subtraktion die niedrigste Priorität.

Operator	Rechenart	Beispiel
^	Potenzierung	V^4
-	Negation (negatives Vorzeichen)	-V
* und /	Multiplikation und Gleitpunktdivision	V*8 und V/34
\	Ganzzahldivision	V\4
+ und -	Addition und Subtraktion	V+7 und V-9

(Logische Operatoren werden in Kapitel 1.6.2 erläutert.)

Tabelle 1.3: *Mathematische Operatoren*

Folgen mehr Operatoren der gleichen Hierarchie-Stufe hintereinander, werden diese von links nach rechts abgearbeitet. Diese Regel können Sie durch den Einsatz von Klammern aufheben. Bei mehreren ineinanderliegenden Klammern wird immer der Inhalt der innersten zuerst bearbeitet.

Beispiele:

Wollen Sie die Summe von 3 und 2 mit 5 potenzieren, ist folgender Ansatz falsch:

```
PRINT 3+2^5
35
```

Wegen der Operator-Hierarchie wird nämlich zuerst 2^5 ausgeführt = 32 und 3 addiert. Um das zu verhindern, müssen Sie eine Klammer verwenden:

```
PRINT (3+2)^5
3125
```

Wenn Sie den Wert 17 durch das Ergebnis der Subtraktion von 11-5 teilen wollen, ist folgender Ansatz ebenso falsch:

```
PRINT 17/11-5
-3.454545
```

Hier wird nämlich erst 17 durch 11 dividiert = 1.545 455 und davon 5 abgezogen. Aber auch in diesem Fall sorgt eine Klammer für das richtige Ergebnis:

```
PRINT 17/(11-5)
2.833333
```

Bislang wurden Operatoren nur auf numerische Werte angewendet. Aber auch für Strings gibt es Operatoren. Der häufigste davon ist die Verkettung zweier oder mehrerer Strings, die Sie mit dem Pluszeichen (+) erzielen können. Hierzu ein Beispiel:

```
a$="uper":b$="computer ":c$"Amiga"
PRINT "S"+a$+b$+c$
```

Ergebnis: Supercomputer Amiga

1.6.2 Logische Operatoren

Wenn Sie etwas tiefer in die Programmierung eingestiegen sind und Befehle zur Ablaufsteuerung, wie IF und WHILE, verwenden, benötigen Sie Bedingungen (Wahrheitswerte). Hinter IF muß zum Beispiel immer eine Bedingung folgen, die dem Interpreter die Entscheidung ermöglicht, wo er das Programm fortsetzen soll; hinter THEN oder hinter ELSE beziehungsweise END IF. Prinzipiell ist zwar jeder numerische Ausdruck oder eine numerische Variable eine Bedingung (ein Wert von Null gilt als FALSCH und ein Wert ungleich Null als WAHR). Zur Formulierung solcher Bedingungen sollten Sie aber wenigstens die wichtigsten logischen Operatoren und Vergleichsoperatoren kennen. In diesem Abschnitt werden zunächst die logischen Operatoren vorgestellt, im nächsten dann die Vergleichsoperatoren.

OPERATOR	Operand 1	Operand 2	Ergebnis
NOT	wahr	-	falsch
AND	wahr	wahr	wahr
	wahr	falsch	falsch
	falsch	wahr	falsch
	falsch	falsch	falsch
OR	wahr	wahr	wahr
	wahr	falsch	wahr
	falsch	wahr	wahr
	falsch	falsch	falsch
XOR	wahr	wahr	falsch
	wahr	falsch	wahr
	falsch	wahr	wahr
	falsch	falsch	falsch

Tabelle 1.4: *Logische Operatoren*

Mit den logischen Operatoren können Sie mit Hilfe der IF-Anweisungen den Programmablauf steuern. Probieren Sie zunächst einmal NOT aus:

```
a=5:IF NOT 6 THEN PRINT "FALSCH" Ergebnis: FALSCH
a=5:IF NOT 5 THEN PRINT "WAHR" Ergebnis: WAHR
```

Die häufigsten Vergleiche werden mit AND und OR durchgeführt. Das folgende Programm verwendet im Ansatz des IF-Blockes AND und zeigt beim Programmablauf die Ergebnisse gemäß der Wahrheitstabelle des Operators AND:

```
REM und
'P1.6.2-1

start:
INPUT "a,b ";a,b
IF a THEN
   IF a AND b THEN
   PRINT "a=wahr, b=wahr, Ergebnis: wahr = "a AND b"
   ELSE
   PRINT "a=wahr, b=falsch, Ergebnis: falsch = "a AND b"
   END IF
ELSEIF b THEN
   PRINT "a=falsch, b=wahr, Ergebnis: falsch = "a AND b"
ELSE
   PRINT "a=falsch, b=falsch, Ergebnis: falsch = "a AND b"
END IF
GOTO start
```

Das folgende Programm verwendet den Operanden OR:

```
REM oder
'P1.6.2-2

start:
INPUT "a,b ";a,b
IF a OR b THEN
   IF a THEN
        IF b THEN
             PRINT "a=wahr, b=wahr, Ergebnis: wahr = "a OR b"
        ELSE
        PRINT "a=wahr, b=falsch, Ergebnis: wahr = "a OR b"
        END IF
   ELSE
        PRINT "a=falsch, b=wahr, Ergebnis: wahr = "a OR b"
   END IF
ELSE
   PRINT "a=falsch, b=falsch, Ergebnis: falsch = "a OR b"
END IF
GOTO start
```

Sie können mit dem Programm die Wahrheitstabelle des Operators OR ausgeben. Sie werden OR immer dann einsetzen, wenn nur eine von zwei Bedingungen erfüllt sein muß:

```
IF unter%<20 OR ober%>100 THEN PRINT "falsche Eingabe"
```

1.6.3 Vergleichs-Operatoren

Zum Schluß unseres kleinen Basic-Einmaleins nehmen wir uns noch die einfacheren Vergleichsoperatoren zur Brust. Vergleichsoperatoren vergleichen zwei Ausdrücke miteinander; auf Gleichheit, Ungleichheit, größer, kleiner und so weiter. Das Ergebnis eines solchen Vergleichs ist dann entweder wahr (Wert -1) oder falsch (Wert 0). Üblicherweise werden Vergleiche zum Beispiel in IF-THEN- oder WHILE-WEND-Anweisung verwendet. Die folgende Tabelle zeigt alle Vergleichsoperatoren auf einen Blick:

Operator	Bedeutung	Beispiel
=	gleich	Wert1 = Wert2
<	kleiner als	Wert1 < Wert2
>	größer als	Wert1 > Wert2
<> oder ><	ungleich	Wert1 <> Wert2
<= oder =<	gleich oder kleiner als	Wert1 =< Wert2
>= oder =>	gleich oder größer als	Wert1 >= Wert2

Tabelle 1.5: *Vergleichsoperatoren*

Zum besseren Verständnis sollten Sie einige Beispiele für die Verwendung von Vergleichsoperatoren gleich im Direkt-Modus ausprobieren:

Eingabe	Ergebnis
print 27-12=15	-1 (richtig)
print 25><12	-1 (richtig)
print 30<>30	0 (falsch)
a=5:b=7:print a>b	0 (falsch)
c=112:ca=14:printc>=ca	-1 (richtig)
a=5:if a then ?"wahr"else ?"falsch"	wahr
a=0:if a then ?"wahr"else ?"falsch"	falsch

Tabelle 1.6: *Vergleichsoperatoren im Direkt-Modus*

Ähnlich wie numerische Werte können Sie mit den Vergleichsoperatoren aber auch Zeichenketten vergleichen. Dabei werden jeweils zwei Strings zeichenweise über den ASCII-Code miteinander verglichen:

Eingabe	Ergebnis
print "a">"b"	0 (falsch)
print "D"<"H"	-1 (richtig)
print "Pater"<"Peter"	1 (richtig)

Tabelle 1.7: *Vergleichsoperatoren mit Zeichenketten*

Zum Abschluß dieses Abschnittes wollen wir uns nach soviel Theorie ein kleines Programm gönnen. Es demonstriert die Anwendung von Vergleichsoperatoren. Da es aber einen Befehl benutzt, welchen wir erst im nächsten Kapitel kennenlernen, habe ich es an den Schluß gesetzt. Tippen Sie es einfach ein und erfreuen Sie sich am Ergebnis.

```
REM Vergleich
REM P0.6.3-1 Vergleichsoperatoren
a=0:b=10
LINE (0,b)-(a,b)

rechts:
LINE -(a,b):a=a+5
IF a<600-x THEN rechts

runter:
LINE -(a,b):b=b+5
IF b<180-x THEN runter

links:
LINE -(a,b):a=a-5
IF a>10+x THEN links

rauf:
LINE -(a,b):b=b-5
IF b>15+x THEN rauf
x=x+5
IF b>= 90 THEN END ELSE rechts
```

2
Grafik

Wenn Sie eine Zeitlang mit Basic herumgespielt haben, ist Ihnen bestimmt mehrmals die Idee gekommen, ein Bild, das Sie mit einem Zeichenprogramm erstellt haben, in einem eigenen Basic-Programm einzubauen. Aber da beginnen bereits die Schwierigkeiten, denn außer der Betrachtung des hübschen Bildes am Bildschirm konnten Sie nichts damit anfangen. Dabei wäre es doch so schön, könnte man die Zeichnungen je nach Bedarf verändern oder anpassen. Bald können Sie das. In diesem Kapitel lernen Sie nach und nach alle für das Zeichnen von Grafik-Bildern erforderlichen Befehle und Anweisungen kennen; anhand kleinerer und etwas größerer Programme. Ich bin überzeugt davon, daß Sie sogar Vergnügen daran finden werden. Es ist auch nicht daran gedacht, daß Sie nur stur Listings abtippen sollen. Vielmehr sollen Sie möglichst viele Anregungen für eigene Programme erhalten. Aus diesem Grunde wurden für die einzelnen Befehlserläuterungen immer neue Programmbeispiele erdacht, obwohl es bestimmt einfacher gewesen wäre, vorhergegangene Programme zu erweitern.

Nun aber genug der Vorrede. Es geht los.

2.1 Der erste Grafik-Befehl

LINE *STEP* (x1,y1)-*STEP* (x2,y2), *Farbe, bf*
zeichnet eine Linie oder ein Viereck auf dem Bildschirm

LINE zeichnet eine Linie auf dem Bildschirm vom Punkt (x1,y1) zum Punkt (x2,y2). Welche Farbe die Linie haben soll, gibt die Option *Farbe* an. Verwenden Sie zum Abschluß des Befehls die Option *bf*, wird das Rechteck, dessen Diagonäle die Linie bildet, komplett mit der angegebenen Farbe ausgefüllt. Die Punkte werden normalerweise

als absolute Positionen (Koordinaten) in Klammern angegeben. Schreiben Sie vor der Klammer aber STEP, werden diese Zahlen darin nicht als absolute Positionen, sondern als Abstände des gesuchten Punktes von der aktuellen Position des Grafik-Cursors aufgefaßt.

Diesen vielseitigen Befehl haben Sie sicher schon einmal ausprobiert. Einen Wehrmutstropfen bedeutet allerdings die Aussage im Amiga-Basic-Handbuch, daß die Option Farbe nur die 4 Grundfarben haben darf. Ich kann Sie beruhigen. Die Angabe ist, so wie sie dort steht, falsch. Sie werden später mit LINE in 32 verschiedenen Farben zeichnen. Nun soll aber anhand eines kleinen Programmes die Geschwindigkeit des Amiga beim Linienzeichnen erst einmal getestet werden.

```
REM Linie
'P2.1-1 LINE, GOTO

start:
LINE (0+x,0+x)-(620-x,90),3
LINE -(10+x,184-x),2
LINE -STEP (0,-186+x),1
x=x+2
GOTO start
```

Aktivieren Sie nun Amiga-Basic durch einen Doppelklick mit der linken Maus-Taste und tippen Sie dieses kleine Programm ein. Achten Sie beim Ausprobieren darauf, daß es ohne äußere Einwirkung nicht anhält. Sobald die letzte Zeile erreicht ist, springt es nämlich zum Anfang zurück. Sie müssen es deshalb mit dem Menü-Befehl *Stop* aus dem Run-Menü anhalten, sobald das Bild komplett ist.

Mit den drei LINE-Anweisungen haben Sie gleich drei verschiedene Anwendungsmöglichkeiten dieses Befehles eingesetzt. Die erste LINE-Anweisung zeichnet eine schräge Linie von der linken, oberen Bildschirmecke bis zur Mitte der rechten Fensterbegrenzung. Die Variable x hat bei jedem Programmdurchlauf einen anderen Wert und versetzt so die einzelnen Linien jedesmal ein wenig. Der zweite LINE-Befehl setzt die erste Linie an ihrem Endpunkt fort. Bei beiden Anweisungen sind die tatsächlichen, absoluten Bildschirm-Koordinaten angegeben. Bei dem letzten LINE-Befehl ist dies anders. Durch STEP wird die aktuelle Position des Grafik-Cursors zur Startposition. Die Linie wird also in x-Richtung um null Bildpunkte und in y-Richtung um minus 186 Pixel gezeichnet. (Bewegungen in x-Richtung haben nach rechts ein positives Vorzeichen und nach links ein negatives. Auf der y-Achse haben Bewegungen nach unten ein positives und nach oben ein negatives Vorzeichen.)

Nachdem schon das erste Programm ein solch tolles Muster zustande gebracht hat, testen Sie nun einmal, wie sich der LINE-Befehl beim Zeichnen von Rechtecken anstellt:

```
REM Tunnel
REM P2.1-2 LINE, IF THEN ... ELSE
CLS
f=3
```

```
lauf:
x1=200:y1=95:x2=x1+260:y2=y1+5

tunnel:
LINE (200,95)-(460,100),f,bf
LINE (x1,y1)-(x2,y2),f,b
x1=x1-4:y1=y1-2:x2=x2+4:y2=y2+2
if x1<=0 THEN farbe
GOTO tunnel

farbe:
IF f=3 THEN f=1 ELSE f=3
GOTO lauf
```

In der vierten Zeile erhält die Variable *f* den Wert 3. Das *f* steht hier für Farbe. Wenn die Farben in den Preferences auf der Workbench nicht verstellt wurden, haben die vier Standardfarben folgende Bedeutung:

0 = Blau (Hintergrund)
1 = Weiß (Vordergrund)
2 = Schwarz
3 = Rot/Orange

Unter der Sprungmarke *lauf* erhalten die beiden Variablen für die Rechteckgröße ihre Startwerte. Das Programm läuft als nächstes die Sprungmarke *tunnel* an. Der erste LINE-Befehl setzt ein ausgefülltes Rechteck in die Mitte des Bildschirmes. Die beiden ersten Zahlen hinter dem LINE-Befehl sind die Werte für die obere linke Ecke des Rechteckes (200 Bildpunkte von der linken oberen Ecke des Bildschirmes nach rechts und 95 Bildpunkte von der linken oberen Ecke nach unten). Die anderen beiden Zahlen in der zweiten Klammer sind die Werte für die untere rechte Ecke des Rechteckes. Es folgt die Variable *f* für die Farbe des Rechtecks und die Option *bf*, die dafür sorgt, daß ein Rechteck statt einer Linie gezeichnet wird. Wird nur das *b* vorgegeben, so zeichnet der Amiga nur den Rand des Rechtecks, das zusätzliche *f* füllt das Rechteck farbig aus.

Als letztes im Programm folgen zwei IF-Anweisungen. Bei der ersten Anweisung wird geprüft, ob das Rechteck inzwischen so groß geworden ist, daß es den Bildschirmrand überschreiten würde. Ist dies der Fall, verzweigt THEN zu der Sprungmarke *farbe*. Der zweite IF-Befehl trägt die Verantwortung für den Farbwechsel:

Wenn Farbe = Rot dann Farbe = Weiß andernfalls Farbe = Rot

```
IF f = 3 THEN f = 1 ELSE f = 3
```

Doch nun sollten Sie, falls Sie es nicht schon längst getan haben, die trockene Theorie zur Seite legen. Gehen Sie mit der Maus über das Run-Menü und wählen Sie *Start*.

Sieht doch recht hübsch aus, unser zweites Programm! Oder finden Sie es auch dürftig, daß für ein Programm auf einem Supercomputer mit 4096 möglichen Farben nur ganze 4 Farben zur Verfügung stehen? Recht haben Sie! Bevor Sie diesem Mißstand zu Leibe rücken können, müssen Sie sich aber erst ein wenig mit dem beschäftigen, was wir dauernd vor Augen haben, nämlich den Bildschirm unseres Computers.

2.2 Fenster auf- und zumachen

Nachdem Sie den Amiga ja zumindest ein paar Tage kennen, wissen Sie bestimmt, daß mit dem Wort Fenster in der Überschrift nicht Ihr Wohnungsfenster gemeint ist. Doch ehe wir ein neues Fenster öffnen, sollte uns erst einmal interessieren, wie groß denn das Fenster ist, in welchem wir die ersten Programme gezeichnet haben. Sie wissen das schon? Wenn Sie der Meinung sind, daß es 640 Punkte horizontal und 200 Pixel vertikal sind, steht Ihnen eine kleine Enttäuschung bevor. Lassen Sie einfach den Amiga nach der richtigen Lösung suchen:

```
REM Bildschirm
REM P2.2-1ermitteln der Bildschirmgroesse
CLS ' Bildschirm loeschen
FOR i = 0 TO 615
PSET (i,10)
NEXT i
LINE (0,11)-(625,14),3,bf
FOR i = 0 To 185
PSET (20,i)
NEXT i
LINE (22,0)-(25,200),3,bf
FOR i = 1 TO 10000 : NEXT i ' Zeitschleife
END
```

Das Wichtigste in diesem Programm sind die beiden folgenden neuen Befehle:

PRESET *STEP* (x,y),Farbe
PSET *STEP* (x,y),Farbe
setzt einen Bildpunkt an einer bestimmten Bildschirmposition

x und y sind die absoluten, oder bei STEP die relativen Koordinaten des Punktes. Wird bei PRESET keine Farbe mit angegeben, verwendet Basic die Hintergrundfarbe des Bildschirms. Damit kann diese Anweisung zum Löschen von Linien verwendet werden:

```
FOR i = 100 TO 200: PSET(i,50):NEXT i
FOR i = 200 TO 100 STEP-1: PRESET(i,50):NEXT i
```

Das Beispiel zeichnet eine Linie und löscht sie anschließend wieder.

Das Programm P2.2-1 setzt jeweils eine senkrechte und eine waagrechte weiße Linie an den linken beziehungsweise oberen Rand des Fensters. Direkt neben beiden Linien wird dann ein schmales rotes Rechteck gezeichnet, dessen Seitenlänge auf dem Bildschirm nicht mehr darstellbar ist. Wenn Sie genau hinsehen, können Sie erkennen, daß die weiße Linie jeweils um einen Pixel länger ist, als das entsprechende Rechteck hoch oder breit ist.

Dieses Programm zeigt, daß maximal 617 Punkte waagrecht und maximal 187 Punkte senkrecht im Ausgabe-Fenster darstellbar sind.

Das ist uns aber zu wenig. Wir werden gleich versuchen, uns noch ein paar Bildpünktchen dazu zu klauen. Doch dafür sollten Sie noch etwas über die Anweisungen SCREEN und WINDOW erfahren. Diese beiden Anweisungen gehören zusammen wie Mann und Frau, oder besser, wie der Computer und sein Bildschirm:

SCREEN Nummer, Breite, Höhe, Tiefe, Modus
öffnet einen neuen Bildschirm mit bestimmten Eigenschaften
SCREEN CLOSE Nummer
schließt den Bildschirm mit der angegebenen *Nummer*

Der Wert für *Nummer* muß zwischen eins und vier liegen und dient als Kennung für den neuen Schirm. Diese Kennung muß auch in der WINDOW-Anweisung verwendet werden, um festzulegen, in welchem Bildschirm ein Fenster liegen soll, das neu am Bildschirm erscheint. Die Werte für die *Breite*, *Höhe*, *Tiefe* und den *Modus* hängen voneinander ab und sind nicht beliebig einzusetzen. Der Wert für die Tiefe legt die Anzahl der Bit-Ebenen und damit der in diesem Bildschirm darstellbaren Farben fest:

Tiefe	darstellbare Farben
1	2
2	4
3	8
4	16
5	32

Tabelle 2.1: *Bit-Ebenen und Farben*

Der Wert für den Modus legt die Auflösung des Bildschirmes fest:

Modus	Auflösung
Modus 1	320 Pixel/Zeile bei 256 (200) Zeilen (ohne Zeilensprung)
Modus 2	640 Pixel/Zeile bei 256 (200) Zeilen (ohne Zeilensprung)
Modus 3	320 Pixel/Zeile bei 512 (400) Zeilen (mit Zeilensprungverf.)
Modus 4	640 Pixel/Zeile bei 512 (400) Zeilen (mit Zeilensprungverf.)

Tabelle 2.2: *Mögliche Auflösungen eines neuen Bildschirms. Die Werte in Klammern gelten für die Version 1.1 des Betriebssystems*

Beim Zeilensprungverfahren ist zwar die Anzahl der horizontalen Bildzeilen verdoppelt. Dafür werden auf dem Bildschirm aber jede Sekunde nur halb soviele neue Bilder gezeichnet wie ohne Zeilensprungverfahren. Dies führt zu einem deutlich sichtbaren, sehr störenden Flimmern des Bildes. Nun aber zum Gegenpart von SCREEN:

WINDOW Nummer,*Titel,(x1,y1)-(x2,y2),Typ,Screen*

öffnet ein neues Fenster mit bestimmten Eigenschaften

WINDOW OUTPUT Nummer

schickt alle Ausgaben des Basic-Programms in das Fenster mit der angegebenen Nummer

WINDOW CLOSE Nummer

schließt das Fenster mit der angegebenen Nummer (entfernt es vom Bildschirm)

Die erste Befehlsform erzeugt ein neues Fenster und zeigt es auf dem Bildschirm. Die *Nummer* (zwischen 1 und 4) gibt dem neuen Fenster eine Kennung, unter der Sie es später wieder ansprechen können. Verwenden Sie WINDOW nur mit *Nummer* und lassen die restlichen Angaben weg, aktiviert er ein bereits erzeugtes Fenster und bringt es in den Vordergrund. (x1,y1)-(x2,y2) gibt die linke obere und rechte untere Ecke des neuen Fensters an und SCREEN bestimmt, in welchem Bildschirm es auftaucht (siehe SCREEN). Es dürfte klar sein, daß die Abmessungen eines Fensters immer kleiner sein müssen als der entsprechende Bildschirm. *Typ* legt schließlich fest, wie man das Fenster mit Hilfe der Maus manipulieren kann. Dies wird später noch anhand von Beispielen illustriert.

WINDOW OUTPUT aktiviert ein Fenster, bringt es aber nicht in den Vordergrund. (Die Ausgabe von Grafiken kann also auch im Hintergrund ablaufen.) WINDOW CLOSE löscht das Fenster mit der angegebenen Kennung.

Ihr neu erworbenes Wissen werden Sie nun zuerst bei einem Programm zur Ermittlung der Schirmgröße einsetzen. Bei der SCREEN-Anweisung nehmen wir für die Tiefe den Wert 2 (maximal vier Farben) und für den Modus ebenfalls den Wert 2. Modus 2 bedeutet die Darstellung von 640 Bildpunkten pro Zeile bei 256 Zeilen. Für die WINDOW-Anweisung nehmen wir als Kennung 2, da die Kennung 1 bereits für das Basic-Ausgabefenster vergeben ist. Den Titel lassen wir weg und als *Typ* nehmen wir die Zahl 0. Die Bildschirm-Koordinaten lassen wir ebenfalls wegfallen. In diesem Fall taucht das Fenster mit der maximalen Größe auf. Das überarbeitete Programm sieht dann wie folgt aus:

```
REM Bildschirm 2
'P2.2-2 SCREEN WINDOW
SCREEN 2,640,256,2,2
WINDOW 2,,,0,2
LINE (0,70)-(629,79),1,bf          maximale Breite -2
LINE (0,80)-(631,89),3,bf          maximale Breite 632 Pixel
```

```
LINE (20,0)-(39,250),1,bf          maximale Hoehe -1
LINE (40,0),-(59,251),3,bf         maximale Hoehe 252 Pixel
FOR i = 1 TO 32: LOCATE i, 15:PRINT i;:NEXT'max. 31
** Textzeilen
tas: a$=INKEY$ : IF a$=" " THEN tas
WINDOW CLOSE 2: SCREEN CLOSE 2     schliesst Fenster und
** Bildschirm
```

Das Programm arbeitet nicht mit der Betriebssystem-Version 1.1. Wenn Sie mit dieser ersten Version arbeiten, müssen Sie alle y-Werte, die größer als 0 sind, um 56 reduzieren. Mit folgender Zeile, im Direkt-Modus eingegeben, können Sie die maximale Höhe Ihres Bildschirmes ermitteln:

```
PRINT PEEKW (PEEKL(WINDOW(7)+46)+14)
```

Diese Anweisung zu erklären, würde den Rahmen dieses Buches sprengen. Dazu müßten wir in die Tiefen der System-Software des Amiga hinabsteigen.

Wenn Sie dieses Programm laufen lassen, kehren Sie nach Betätigung einer beliebigen Taste ins Ausgabefenster zurück. Dafür sorgt die Verwendung der folgenden Anweisung am Programmende.

INKEY$
prüft, ob ein Zeichen eingegeben wurde und liefert dieses als Ergebnis zurück

Wenn der Tastaturpuffer leer ist, wird ein leerer Text ("") übergeben. Das Programm stellt dies fest und beginnt dann wieder von vorn; so lange, bis eine Taste gedrückt wird.

Daß Sie mit SCREEN wirklich neue, von der Workbench unabhängige Bildschirme erzeugen können, zeigt das folgende Programm:

```
REM ZweiSCREENS
'P2.2-3 FRE SPC TAB PTAB
frei1=FRE(-1)
SCREEN 1,320,200,5,1
SCREEN 2,320,200,5,1
WINDOW 2,"Fenster auf SCREEN1",(30,30)-(220,170),15,1
WINDOW 3,"Fenster auf SCREEN2",(10,30)-(280,170),15,2
PRINT SPC(3) "System-Speicher "
PRINT "vorher"TAB(10)"nachher"
PRINT frei1;PTAB(60);FRE(-1)
start:
WINDOW OUTPUT 2 'wenn ohne OUTPUT dann Umschaltung
LINE (20,5)-(150,100),fa,bf
fa=fa+1
WINDOW 3
LINE (150,10)-(275,170),fa,bf
```

```
fa=fa+1
IF fa>30 THEN fa=1
ta$=INKEY$:IF ta$<>"" THEN ende
GOTO start

ende:
SCREEN CLOSE 1
SCREEN CLOSE 2
END
```

Bevor Sie sich das Programm im einzelnen betrachten, tippen Sie es erst einmal ab und starten es. Es erscheint ein Fenster, welches ungefähr in der Mitte des Bildschirmes sitzt. In dem Fenster wechselt ein Rechteck unablässig seine Farben (was auf die Dauer recht nervig wirkt). Am oberen Rand des Bildschirmes sehen Sie einen weißen Streifen. Fahren Sie mit dem Pfeil der Maus auf diesen weißen Balken und drücken die linke Maus-Taste. Fahren Sie nun, bei gedrückter Taste, bis etwa in die Mitte des Bildschirmes. Und siehe da, wie durch Zauberei erscheint dahinter ein weiteres Fenster, in welchem ebenfalls ein Rechteck andauernd seine Farben wechselt. Was Sie jetzt im Augenblick sehen, sind zwei SCREENS gleichzeitig auf dem Bildschirm. Aber auch dieser Bildschirm hat oben einen weißen Balken. Wiederholen Sie bitte den Vorgang von vorhin und ziehen Sie diesen Bildschirm um etwa ein Drittel nach unten. Dahinter befindet sich noch einmal ein Bildschirm, nämlich der SCREEN der Workbench. Er zeigt im Moment das Basic-Fenster verdeckt. Wenn Sie dieses schließen, sehen Sie die Workbench mit allen ihren Icons. Um das Programm wieder zu verlassen, brauchen Sie nur eine Taste zu drücken. (Hierzu wird natürlich wieder das inzwischen bekannte INKEY eingesetzt.)

In der ersten Zeile des Programms nach den Bemerkungen (REMs) wird der noch verfügbare System-Speicher in der Variablen frei abgelegt:

v=FRE(x)
liefert je nach *x* Aussagen über freie Speicherplätze (Bytes)

Bei FRE handelt es sich um eine Funktion. Eine Funktion liefert Informationen (Zahlen oder Texte), in diesem Fall über freie Speicherplätze:

x=1	noch verfügbarer System-Speicher
x=-2	noch verfügbarer Amiga-Basic-Stapel-Speicher
x>-1	noch verfügbarer Amiga-Basic-Programm-Speicher

Im Programm **zwei SCREENS** fragen wir zu Beginn nach dem verbliebenen System-Speicher. Anschließend definieren wir zwei Bildschirme mit einer Auflösung von 320*200 Pixels mit einer Tiefe von 5. Zwei Fenster, für jeden Bildschirm eines, werden erzeugt. Auf dem Fenster 3 wird der Text über den freien Speicherplatz vor und nach der Aktivierung der SCREENS und WINDOWS angezeigt. Bei dem Label *start* beginnt der eigentliche Programm-Ablauf. Mit WINDOW OUTPUT 2 wird Fenster 2 aktiviert. Der

nachfolgende LINE-Befehl wirkt unsichtbar im Hintergrund. Wenn Sie den Teil OUTPUT des Befehles löschen, wird laufend zwischen den beiden SCREENS hin- und hergeschaltet. Probieren Sie es ruhig aus. Die Zeile WINDOW 3 sorgt dafür, daß Fenster 3 im Vordergrund bleibt und dort das farbenfrohe Rechteck zu sehen ist. Bevor GOTO *start* die Schleife schließt, wird aber noch einmal abgefragt, ob eine Taste gedrückt wurde. War dies der Fall, werden beide Bildschirme nacheinander mit SCREEN CLOSE wieder geschlossen.

Kehren wir zur Textausgabe des Beispielprogramms zurück. Ist Ihnen dabei etwas besonders aufgefallen? Vier Besonderheiten sind zu finden. Die erste ist der enorme Speicherbedarf für dieses kleine Programm. Es frißt 86 Kbyte allein vom System-Speicher! Also Vorsicht bei Experimenten mit SCREEN! Hiermit können Sie spielend den Amiga zum Absturz bringen. Wenn Sie zum Beispiel in die SCREEN-Anweisungen eine Auflösung von 640*200 Bildpunkten unterbringen wollen, stürzt das Programm ab und Sie haben plötzlich kein vernünftiges Bild mehr vor sich. Da hilft nur eines: Diskette raus und Neustart.

Die drei anderen Besonderheiten im Beispielprogramm sind drei verschiedene Befehle für die Verwendung eines Tabulators:

SPC(n)
gibt *n* Leerstellen aus

Dieser Befehl sorgt für Zwischenräume in einer Textzeile:

```
PRINT "Amiga"SPC(5)"Basic"
```

ergibt auf Bildschirm, Drucker, sequentieller Datei

```
Amiga  Basic
```

TAB(n)
setzt den Text-Cursor auf die Position *n* Stellen rechts vom linken Rand des Fensters (die vertikale Position bleibt gleich)

Mit diesem Befehl können Sie eine formatierte Ausgabe erzielen:

```
PRINT "Computer"TAB(25)"Seite 20"
PRINT "COMPUTER-Zubehör"TAB(25)"Seite 22"
PRINT "Drucker"TAB(25)"Seite 25"
```

ergibt:

```
Computer              Seite 20
Computer-Zubehör      Seite 22
Drucker               Seite 25
```

Die letzte Tabulator-Funktion setzt einen grafischen Tabulator:

PRINT PTAB(n)
setzt den Text-Cursor auf die Position *n* Bildpunkte rechts vom linken Rand des Fensters (die vertikale Position bleibt gleich)

Die Funktion dieses Befehls können Sie an unserem letzten Programm recht gut erkennen. Unter dem Wort nachher finden Sie die Zahl über den restlichen Speicherplatz. Die Zahlen stehen nicht direkt unter den Buchstaben, sondern etwas versetzt darunter.

Bevor Sie zum nächsten Kapitel übergehen, sollten Sie ruhig noch etwas mit SCREEN und WINDOW herumprobieren. Beachten Sie aber bitte besonders bei SCREEN den nötigen Speicherbedarf! Falsche Angaben führen hier oft zum Programmabsturz.

2.3 Wohin mit den Programmen?

Nachdem Sie einige eigene Beispielprogramme entwickelt haben, werden Sie diese bestimmt auch einmal abspeichern wollen. Wenn Sie sich Ihre Amiga-Basic-Diskette einmal genauer ansehen, werden Sie aber feststellen, daß diese ganz schön voll ist. Wie Sie mit Info aus dem Workbench-Menü erfahren, sind von insgesamt 1758 Blocks bereits 1515 belegt. Bei 512 Kbyte pro Block bleiben etwa noch 124 Kbyte für Ihre Programme übrig. Falls Sie meinen, das wäre doch eine ganze Menge, kennen Sie den Amiga aber schlecht. Die meisten besonderen Fähigkeiten des Amiga sind ungeheuer speicherhungrig. Die verbleibenden 124 Kbyte reichen deshalb gerade zur Speicherung von einigen kleinen Programmen und drei (3; kein Schreibfehler) Grafik-Bildern.

Natürlich kann man seine Basic-Programme mit der richtigen Datei-Spezifikation auf jeder beliebigen Diskette speichern und von dort auch wieder laden. Aber erstens ist es recht umständlich, immer den vollständigen Pfadnamen samt Diskettenlaufwerk anzugeben, und zweitens wollen wir ja die Vorteile der Pull-down-Menüs nutzen. Wenn Sie den Amiga Tutor auf der Basic-Diskette löschen, hilft Ihnen das auch nicht viel weiter, wie Sie gerne ausprobieren können. Sie werden deshalb nun erfahren, wie Sie eine Programmier-Diskette anfertigen können, auf der genügend Platz für Ihre eigenen Entwicklungen verbleibt.

Verwenden Sie für die nachfolgend beschriebenen Diskettenoperationen bitte niemals die Original-Diskette. Diese gehört kopiert und zur Sicherheit im hintersten Winkel Ihres Schreibtisches versteckt. Auch in der Disketten-Box, mit der Sie täglich arbeiten, hat sie nichts zu suchen. Wenn Sie einmal beim Programmieren einen groben Fehler einbauen, oder ein lieber Mitmensch stolpert über das Anschlußkabel Ihres Computers, können die Daten unwiederbringlich dahin sein. Zugegeben, so etwas passiert nicht täglich. Aber wenn es geschieht, dann soll es auf einer Kopie der Original-Diskette passieren.

So, nach diesem Ausflug in die Philosophie des Datenschutzes wieder zurück zur Amiga-Basic-Diskette. Der Grund, warum auch das Löschen des Tutors nichts nützt, liegt an einer ganzen Batterie von Unterprogrammen und Hilfsdateien für den Tutor. Wenn jemand zuviel Zeit hat, kann er sich die Arbeit machen und alle diese Unterprogramme mit dem KILL-Befehl löschen. Wir wählen den kürzeren und einfacheren Weg über eine neue, leere Diskette.

Nehmen Sie jetzt bitte eine leere Diskette, oder eine, deren Inhalt Sie nicht mehr benötigen. Legen Sie diese in den Diskettenschacht und warten, bis deren Icon (eventuell namens DFO:BAD) erscheint und klicken dieses an. Wählen Sie nun aus dem *Disc*-Menü *Initialize-Disk*. (Der Computer verlangt eventuell zwischendurch nach der Workbench-Diskette.) Wenn das rote Licht erloschen ist, entnehmen Sie diese wieder und schieben die neue Diskette hinein. Der Amiga fragt Sie, ob Sie auch wirklich initialisieren wollen (so einen gefährlichen Befehl glaubt er erst nach dem zweiten oder dritten Mal). Klicken Sie jetzt Continue an.

Sobald dieser Vorgang beendet ist, öffnen Sie die Diskette mit einem Doppelklick auf das Icon, und Sie sehen eine – bis auf den Papierkorb – leere und voll aufnahmefähige Diskette vor sich. Ändern Sie jetzt den Namen der Diskette (mit *Rename*) in BFA um. Lassen Sie das Disketten-Fenster geöffnet und legen die Kopie der Amiga-Basic-Diskette ein. Öffnen Sie nun auch dieses Disketten-Fenster und transportieren Sie mit der Maus das Amiga-Basic-Icon in die neue Diskette. Keine Sorge, wenn Sie ein Icon von einer Diskette auf eine zweite bringen, kopiert der Amiga automatisch dessen Inhalt, löscht aber nicht das Icon auf der Originaldiskette. Amiga-Basic bleibt Ihnen also auch auf der anderen Diskette erhalten. Der Computer fordert Sie anschließend zu einigen Diskettenwechseln auf, bis das Programm auf die neue Diskette kopiert ist.

Die gleiche Prozedur wiederholen Sie am besten gleich anschließend mit der Basic-Demo-Schublade. Wenn Sie wollen, können Sie auf der BFA-Diskette die Demo-Schublade auch noch ausleeren. Lassen Sie aber bitte den Spriteeditor mit den beiden Bibliotheks-Routinen darin. Wenn Sie jetzt noch einmal mit Info die neue Diskette BFA überprüfen, werden Sie sehen, daß Ihnen nun 1513 Blocks, also etwa 774 Kbyte zur Verfügung stehen. Das reicht eine Weile. Da Sie ein vorausdenkender Mensch sind, werden Sie sich von der neu erstellten Diskette gleich eine Kopie machen, damit Sie das nächste Mal, wenn Sie eine neue Basic-Diskette benötigen, gleich eine parat haben. Verwenden Sie für solche Kopien der gerade angefertigten Diskette aber bitte andere Namen.

Nachdem Sie sich so flott eine neue Diskette angelegt haben, bleibt noch ein bißchen Zeit für ein kleines, aber sehr hübsches Programm, das zeigt, wie man mit wenig Aufwand und dem LINE-Befehl eine recht große Wirkung erzielen kann.

```
REM Kelch
'P2.3-1 COS ATN WRITE SWAP
a=1:b=2:c=3
```

```
lauf1:
FOR i=1 TO 359
   SWAP a,b:SWAP b,c
   x=200*COS(i*3.14/180)
   LINE (300+x,10)-(300,170),a
NEXT
te1$="Die Winkel der Linien wurden mit einer "
te2$="COS-Funktion errechnet."
WRITE te1$,te2$

zeit:
FOR ti=1 TO 5000:NEXT ti

lauf2:
CLS
FOR i=1.57 TO -1.57 STEP -.01
   SWAP a,b:SWAP b,c
   x=i*100
   LINE (300+x,10)-(300,170),a
NEXT
te3$="ATN-Funktion errechnet."
WRITE te1$,te3$
```

Dieses Programm zeichnet zwei kelchförmige Linienbündel und demonstriert dabei die Verwendung einiger neuer Funktionen und Befehle. Die Winkel der Linien wurden zum Beispiel mit den Funktionen COS und ATN berechnet:

v=COS(x)
übergibt den Cosinus des Wertes *x* (Winkel im Bogenmaß)

Sie müssen also zuerst den Wert *x* im Bogenmaß umrechnen. Bei einem Winkel von 43 Grad sieht die dafür nötige Rechnung folgendermaßen aus:

```
v=COS(43*pi/180)
```

Wenn Sie für pi=3.141 593 einsetzen, erhalten Sie als Ergebnis den numerischen Wert der trigonometrischen COS-Funktion 0.7313536.

v=ATN(x)
berechnet den Arcustangens zum Wert *x*

Sie erhalten durch diese Funktion die Länge des Kreisbogens im Einheitskreis r=1. Um diese in Winkelgrade umzurechnen, können Sie folgende Formel einsetzen:

```
Winkel=180/pi*ATN(x)
```

Wenn v=3.14 ergibt, erhalten Sie: Winkel=180/3.14*3.14=180 Grad

In unserem Programm sind beide Funktionen ohne besonderen mathematischen Hintergrund eingesetzt. Sie zeigen nur, wie man grafische Effekte mit mathematischen Funktionen erzielen kann. Während des Programmablaufes sorgen zwei SWAP-Anweisungen dafür, daß jede Linie in einer anderen Farbe gezeichnet wird:

SWAP Variable1,Variable2
vertauscht die Werte zweier Variablen gleichen Typs

Es kann also keine String-Variable mit einer numerischen Variable getauscht werden. Am Schluß dieses Kapitels finden Sie eine Eingabe-Routine, die eine mögliche Anwendung von SWAP demonstriert. Für die Ausgabe des Textes im Programm Kelch haben wir eine neue Anweisung verwendet:

WRITE ***Liste von Ausdrücken***
zeigt die Werte der Ausdrücke im aktuellen Fenster, ähnlich wie PRINT eine solche Liste im Ausgabe-Fenster ausgeben würde

Die Ausdrücke müssen durch Kommata oder Strichpunkte voneinander getrennt werden. Im Gegensatz zum PRINT-Befehl werden die Anführungszeichen von Strings mit ausgegeben. Auch haben Komma und Semikolon anders als bei PRINT hier keine Tabulator-Steuerfunktion und werden beide gleich behandelt.

2.4 Mehr Farbe ins Spiel

Mit diesem Abschnitt beginnen wir das Programm-Projekt des ersten Kapitels. Wie schon die Überschrift besagt, fangen wir gleich mit der Farbe an. Was dem Maler seine Palette, ist dem Amiga die PALETTE-Anweisung von Basic:

PALETTE Farbkennung,Rot,Grün,Blau
definiert für eine Farbkennung einen Farbton aus den drei Grundfarben Rot, Grün und Blau

Farbkennung muß eine Zahl zwischen 0 und 31 sein und entspricht einer der 32 Farben, die der Amiga gleichzeitig am Bildschirm darstellen kann. Die Angaben für Rot, Grün und Blau bestimmen, wieviel die jeweilige Farbe zum resultierenden Farbton beiträgt.

Sind die Angaben für Rot, Grün und Blau alle gleich 1.0, ergibt das ein helles Weiß; sind sie alle Null, erhalten Sie Schwarz. Es gibt 16 mögliche Werte zwischen 0.00 und 1.00 für jeden dieser Anteile. Das ergibt 16*16*16=4096 verschiedene Farbtöne. Warum ausgerechnet die Zahl 16? Für jeden Farbanteil steht ein halbes Byte (= ein Nibbel) zur Verfügung. Mit 4 Bit lassen sich 16 verschiedene Werte-Intensitäten darstellen.

Wenn Sie aber wirklich mit 32 Farben programmieren wollen und Ihnen die dafür voreingestellten Farben nicht gefallen, bedeutet das, daß Sie auch 32 PALLETE-Befehle eintippen müssen. Am besten fangen Sie gleich damit an. Halt! Vorher legen Sie bitte eine leere Schublade mit dem Namen Unterprogramme auf Ihrer BFA-Diskette an.

```
REM P.2.4=1

Farbtafel:
   PALETTE 0,1,1,1 'weiss
   PALETTE 1,.8,0,0 'braunrot
   PALETTE 2,1,0,0 'dunkelrot
   PALETTE 3,.93,.2,0 'feuerwehrrot
   PALETTE 4,1,.4,0 'hellrot
   PALETTE 5,1,.6,0 'rotbraun
   PALETTE 6,1,.8,0 'sand
   PALETTE 7,1,1,0 'gelb
   PALETTE 8,.6,1,.1 'giftgruen
   PALETTE 9,.47,.8,.13 'gruen
   PALETTE 10,.4,.6,0 'dunkelgruen
   PALETTE 11,.2,.4,0 'olivgruen
   PALETTE 12,0,.4,0 'schwarzgruen
   PALETTE 13,0,.6,.67 'graugruen
   PALETTE 14,0,.8,.6 'mittelgruen
   PALETTE 15,0,1,.6 'hellgruen
   PALETTE 16,.2,1,.93 'hellblau
   PALETTE 17,.2,.73,1 'blau
   PALETTE 18,.13,.4,1 'dunkelblau 1
   PALETTE 19,.4,0,1 'dunkelblau 2
   PALETTE 20,0,0,.6 'schwarzblau
   PALETTE 21,.33,.13,.87'nachtblau
   PALETTE 22,.6,.2,1 'blaulila
   PALETTE 23,1,0,1 'lila
   PALETTE 24,.93,.53,.73 'dunkellila
   PALETTE 25,1,.73,.73 'braunlila
   PALETTE 26,.8,.53,.47 'braun
   PALETTE 27,.6,.4,.33 'dunkelbraun
   PALETTE 28,.4,.2,0 'braungrau
   PALETTE 29,.4,.4,.4 'dunkelgrau
   PALETTE 30,.6,.6,.6 'mittelgrau
   PALETTE 31,0,0,0 'schwarz
RETURN
```

Die hier vorgegebenen Farbwerte stellen sicherlich nicht das Nonplusultra an Farbgebung für 32 Farben dar. Für unser Programm-Projekt sollten Sie diese aber, um Mißverständnisse auszuschließen, beibehalten. Wenn Sie eigene Programme erstellen, können Ihnen diese Farben als ein kleiner Anhaltspunkt für Ihre eigenen Farbschöpfungen dienen. Bitte beachten Sie dabei, daß es keinen Sinn hat, beliebige Dezimalzahlen für die Farbanteile einzusetzen. Für die jeweils 16 verschiedenen Intensitäten müssen Sie die folgenden Werte verwenden:

```
0.00 0.07 0.13 0.20 0.27 0.33 0.40 0.47
0.53 0.60 0.67 0.73 0.80 0.87 0.93 1.00
```

Speichern Sie nun bitte das Programm mit dem folgenden Befehl im Direkt-Modus (beachten Sie die Option a):

```
save "Unterprogramme/Farbtafel",a
```

Nachdem Sie so viel Arbeit damit hatten, wollen Sie sicher endlich die 32 Farben sehen. Löschen Sie dazu zunächst einmal das Programm Farbpalette, denn dieses ist allein nicht lauffähig, und geben Sie folgendes neue Programm ein:

```
REM Farbpalette - Darstellung von 32 Farben
'P.2.4-2
SCREEN 1,320,200,5,1
WINDOW 2,"bitte Taste druecken",(0,0)-(310,180),0,1
GOSUB Farbtafel
CLS
PRINT "FARBPALETTE 32 FARBEN"
FOR i=0 TO 31
  a=10+(i*5)
  LINE (0,a)-(310,a+4),i,bf
NEXT i
GOTO Taste

aus:
SCREEN CLOSE 1
END

Taste:
t$=INKEY$
IF t$<>"" THEN aus
SLEEP
GOTO Taste
```

Die Subroutine *Taste* wartet auf einen Tastendruck und stellt eine Alternative zu dem bereits in früheren Programmen vorgekommenen Einzeiler dar. In der FOR-Schleife darüber werden 32 Rechtecke in unterschiedlichen Farben gezeichnet.

Nun aber das Programm von Anfang an: Mit SCREEN erstellen Sie zunächst einen neuen Bildschirm der Tiefe 5 – also für 32 Farben. Dann wird in diesem SCREEN ein neues Fenster erzeugt. Als Fenstertitel wird eine Anweisung an den Benutzer verwendet. Diese Art Mitteilungen zu verwenden, ist recht einfach und hat den Vorteil, daß im

Fenster selber kein Text ausgegeben werden muß. Sie können also laufend das Bild ändern, ja sogar das Fenster löschen, und trotzdem bleibt die Information sichtbar.

Anschließend an WINDOW finden Sie einen Sprung in eine Subroutine namens Farbtafel. Dieser Programmteil ist jedoch (noch) nicht im Programm, was Sie sofort ändern sollten. Geben Sie dazu im Direktmodus ein:

```
merge "BFA:Unterprogramme/Farbtafel"
```

und schon befindet sich das Unterprogramm am Ende des Hauptprogrammes. Sollte dies bei Ihnen nicht der Fall sein, liegt es wahrscheinlich daran, daß Sie das Programm vorhin nicht mit der Option A abgespeichert haben.

MERGE Dateiangabe
hängt ein Programm (im ASCII-Format) an das bereits bestehende Programm an

Nun können Sie endlich das Programm starten. Und da ist sie, die 32-Farben-Premiere! Haben Sie Appetit auf mehr? Dann können Sie gleich im nächsten Abschnitt weitermachen. Vorher schauen Sie sich bitte auch einmal die 32 Standard-Farben an. Setzen Sie dazu die Zeile *GOSUB Farbtafel* mit einem Hochkomma außer Funktion, und starten Sie das Programm erneut.

2.5 Regenbogen

Bevor Sie darangehen, das eben begonnene Programm weiterzuentwickeln, sollten Sie zunächst einmal versuchen, ein Bild mit diesen 32 Farben aufzubauen. Zur Vorbereitung laden Sie bitte das Programm Farbtafel und ändern es ein wenig ab:

- tauschen Sie die Farben 1 und 10
- löschen Sie das RETURN am Programmende
- und schreiben dafür GOTO Anfang an diese Stelle

Das geänderte Programm speichern Sie dann bitte als »Farbtafel1« mit der Option A als ASCII-Datei ab. Nun können Sie mit dem neuen Werk beginnen. Am besten tippen Sie, wie gehabt, erst das Listing ein und gehen dann anhand des folgenden Textes das Programm Punkt für Punkt durch.

```
REM Regenbogen
REM P.2.5-1
DEFINT A-Z
SCREEN 1,320,200,5,1
WINDOW 2,"bitte Taste druecken",(0,0)-(310,180),16,1
CHAIN MERGE "Unterprogramme/Farbtafel1",10
```

```
Anfang:
CLS
COLOR 1,2
LINE (0,0)-(310,180),17,bf
LINE (0,90)-(310,180),26,bf
za=2
WHILE za<50
   kx=10+za:ky=140-(za/2): GOSUB baum
   za=za+10
WEND
za=3
WHILE za<82
   kx=30+za:ky=170-(za/3):GOSUB baum
   za=za+12
WEND
LINE (150,120)-(210,165),6,bf
LINE (150,100)-(210,120),2,bf
za=4
WHILE za<50
   LINE (150+za,125)-(160+za,135),18,bf
   za=za+14
WEND
za=4
WHILE za<50
   LINE (150+za,145)-(160+za,155),18,bf
   za=za+14
WEND
za=0
WHILE za<70
   kx=230+za:ky=100+za:GOSUB lbaum
   za=za+20
WEND
FOR i=0 TO 31
   A=208+(i*1)
   CIRCLE (160,A),250,i,.3,2.9
NEXT i
GOTO Taste

aus:
SCREEN CLOSE 1
END

Taste:
t$=INKEY$
IF t$<>"" THEN aus
SLEEP
GOTO Taste
```

```
baum:
AREA (kx,ky)
AREA (kx+20,ky)
AREA (kx+10,ky-40)
AREA (kx,ky)
AREAFILL
RETURN

lbaum:
fa=14
CIRCLE(kx,ky),15,fa
PAINT(kx,ky),fa
RETURN
10 REM
```

Wie ein Hasenbraten mit Speck, ist dieses Programm mit neuen Befehlen gespickt. Ein paar sollten Ihnen aber natürlich auch schon bekannt sein. Die Zeile DEFINT A—Z definiert alle Variablen als kurze Ganzzahl. Die SCREEN- und WINDOW-Anweisungen kennen Sie ebenfalls schon. Aber dann kommt ein ganz raffinierter Trick. Bestimmt haben Sie schon entdeckt, daß wieder die Subroutine Farbtafel im Programm fehlt und trotzdem aufgerufen wird. Nur diesmal brauchen Sie die entsprechende Datei nicht im Direktmodus mit MERGE anhängen. Das lassen Sie vom Programm selbst machen!

CHAIN *MERGE* Datei-Angabe, *Zeile, ALL, DELETE Bereich*
Lädt ein anderes Basic-Programm und startet es

Für diesen Befehl gibt es eine Reihe von Optionen, die hier aber nicht alle aufgeführt werden sollen. In unserem Programm Regenbogen bedeutet das MERGE hinter CHAIN, daß das Basic-Programm Farbtafel 1, aus der Schublade Unterprogramme, ab Zeile 10 angefügt und dann gestartet wird. Deshalb lautet auch die letzte Zeile des Programms 10 REM. Interessant ist auch die Möglichkeit, mit CHAIN ein anderes Programm aufzurufen und diesem Programm Werte zu übergeben. Wollen Sie diese Möglichkeit nutzen, brauchen Sie nur den CHAIN Befehl wie folgt zu verwenden:

```
CHAIN "Programm-Name",,ALL
```

Durch die Option ALL bleiben die Werte aller Variablen beim Aufruf des neuen Programms erhalten und können in diesem weiterverwendet werden. Wollen Sie nur einen Teil der Variablen an das neue Programm weiterleiten, müssen Sie diese einzeln am Anfang des Programms angeben:

COMMON Variable1,*Variable2,...*
sorgt dafür, daß die genannten Variablen bei einem CHAIN-Befehl erhalten bleiben

Am besten läßt sich die Wirkungsweise dieses Befehls an zwei kleinen Programmen demonstrieren:

```
REM Abloesung
'P.2.5-2
a=50:b=15:c=15:d=150:e=30:f=999
PRINT "a= "a
PRINT "b= "b
PRINT "c= "c
PRINT "d= "d
PRINT "e= "e
PRINT "f= "f;:PRINT "Variable f wird nicht uebergeben"
PRINT "Jetzt ist Programm >Abloesung< im Speicher"
FOR t=1 TO 10000:NEXT
COMMON a,b,c,d,e
CHAIN "ausfuehren"
REM ausfuehren
'P.2.5-3
CLS
PRINT "Jetzt ist Programm >ausfuehren< im Speicher"
PRINT "a+12= "a+12
PRINT "b*c= "b*c
PRINT "d/e= "d/e
PRINT "f= "f
PRINT "Variable f wurde nicht uebergeben, daher Wert Null"
```

Im Programm >>Ablösung<< geben wir den Variablen a-f verschiedene Werte. In der COMMON-Anweisung nennen wir jedoch lediglich die Variablen a-e. Denken Sie bitte daran, das Programm >>Ausführen<< im ASCII-Format zu speichern. Wenn Sie das Programm Ablösung starten, werden Ihnen die Werte der Variablen auf dem Bildschirm angezeigt. Nach einer kurzen Warteschleife wird durch den CHAIN-Befehl das Programm >>Ausführen<< geladen und dabei das Programm >>Ablösung<< gelöscht. Mit den übergebenen Variablen werden einfache Rechenaufgaben durchgeführt und angezeigt. Die Variable f hat nun den Wert Null, obwohl sie zuvor im Programm >>Ablösung<< einen anderen Wert besaß.

Kehren wir wieder zu unserem Programm >>Regenbogen<< zurück. Zwei Zeilen hinter dem CHAIN-Befehl finden wir wieder eine neue Anweisung:

COLOR ***Vordergrund, Hintergrund***

legt als Vorder- und Hintergrundfarbe zwei Farben aus der Palette fest

Mit dieser Anweisung können Sie sich aus der PALETTE jede beliebige Farbe für den Vordergrund und/oder den Hintergrund aussuchen:

```
COLOR 10,25
```

Die Farbe 10 wird dann für Schrift und Grafik-Befehle, in denen Sie keine Farbe angeben, verwendet. Die Farbe 25 bildet den Hintergrund und wird zum Beispiel auch vom PRESET-Befehl verwendet, um Punkte zu löschen.

Die beiden folgenden Zeilen zeichnen dann ein Rechteck, und damit wird auch das Versprechen aus dem ersten Kapitel eingelöst. Wie Sie sehen können, werden die Rechtecke in den Farben 17 und 26 der Farbtafel gezeichnet und nicht nur in den Farben 0 bis 3. Dies gilt gleichermaßen für die CIRCLE-Anweisung, die bald einen Regenbogen aus 32 Farben zaubern wird.

In den ersten beiden WHILE-WEND-Schleifen wird wiederholt die Subroutine *baum* aufgerufen. Nachdem die beiden Nadelbaumreihen gezeichnet sind, wird ein Häuschen konstruiert. Schnell noch ein paar Laubbäume, und dann kommt der Regenbogen!

CIRCLE STEP (x,y), r, *Farbe, Start, Ende, Ratio*

Zeichnet einen Kreis oder eine Ellipse um den Mittelpunkt x,y mit Radius r

Wird nur CIRCLE (x,y),r und Farbe angegeben, sollte ein Kreis herauskommen. Mit Start und Ende kann man festlegen, daß nur ein Teil des vollen Kreisbogens von 360 Grad gezeichnet wird, nämlich der Bogen zwischen den Winkeln Start und Ende (die natürlich wieder im Bogenmaß angegeben werden müssen). Möchte man keinen Kreis, sondern eine Ellipse zeichnen, kann man mit Ratio angeben, wie groß das Verhältnis zwischen Höhe und Breite sein soll. Die Variable r gibt in diesem Fall nicht »den« Radius, sondern nur einen, nämlich den größten der beiden Radien (horizontal oder vertikal) an.

Unser Regenbogen setzt sich aus 32 Ellipsen-Abschnitten zusammen, die die folgende Anweisung zeichnet. Veränderlich an dieser Anweisung sind nur die Angaben für das Zentrum der Ellipse und die zu verwendende Farbe.

```
CIRCLE (160, a ),          250 ,      i ,    3 ,  2.9
       x, y=32*versetzt,   Radius,    Farben,Start,Ende
```

Die Subroutine *baum* zeichnet – wie der Name sagt – einen Baum und verwendet hierzu zwei neue, sehr leistungsfähige Grafik-Befehle:

AREA STEP (x,y)

definiert eine Seite eines Vielecks (Polygons)

Der erste AREA-Befehl in einem Programm setzt den ersten Eckpunkt des Polygons fest. Die folgenden AREA-Befehle ziehen jeweils Linien zu den angegebenen Punkten und fügen so neue Kanten zu dem Polygon hinzu. (Dies sollte idealerweise so lange fortgeführt werden, bis das Polygon geschlossen ist; der letzte AREA-Befehl sollte also zurück zum Anfang führen.)

AREAFILL *Modus*
schließt das zuvor mit AREA definierte Polygon ab und malt es mit einem Muster aus

Wichtig bei der Verwendung dieser beiden Anweisungen ist, daß die in den einzelnen AREA-Anweisungen vorgegebenen Koordinaten zu einem geschlossenen Vieleck führen, da dieses sonst nicht ausgemalt werden kann. Wird bei AREAFILL Modus auf 1 gesetzt, invertiert Basic das Innere des Polygons. Wird kein Modus angegeben, so wird das Vieleck in der PALETTE Farbe 1 ausgemalt. (Genauer gesagt, wird das Polygon mit einem Muster gefüllt. Ändert man dieses Muster aber nicht mit dem Pattern-Befehl, ist es eine gleichmäßig gefärbte Fläche.) Leider stehen zum Ausmalen eines Polygons nur zwei Farben zur Verfügung. Das war auch der Grund, warum die PALETTE Farben 1 und 10 vertauscht werden mußten. Sie hätten sonst nämlich braune Bäume erhalten. Das war mir zu realistisch.

Und somit sind wir dann schon beim letzten neuen Befehl des Regenbogen-Programms. In der Unterroutine *lbaum* wird ein Kreis gezeichnet und ausgemalt:

PAINT STEP(x,y), *Innenfarbe, Rand*
füllt eine geschlossene Fläche, in der der Punkt (x,y) liegt, mit Innenfarbe; als Begrenzung der Fläche dienen Punkte der Farbe Rand

Für diesen Befehl gilt, wieder entgegen dem Handbuch, daß die Fläche mit jeder beliebigen der 32 Farben gefüllt werden kann. Leider kann diese Fläche aber nur in der Farbe gefüllt werden, mit der sie umrandet ist. Wenn Sie eine Fläche mit einer anderen Farbe eingerahmt haben als Sie dann zum Füllen verwenden, läuft diese Farbe in das ganze Bild. Also Vorsicht! Es nützt auch nichts, wenn Sie für Rand die gleiche Farbe angeben.

Bevor Sie das Programm starten (wenn Sie das nicht längst getan haben), speichern Sie es bitte ab und legen dafür anschließend eine Schublade mit dem Namen Grafik-Beispiele an. So, nach dieser Leistung haben Sie sich jetzt aber eine kleine Entspannung verdient. Das folgende Programm zeichnet einige Hasen, die nur aus Ellipsen bestehen:

```
REM hasen
'P.2.5-4
PALETTE 0,.53,.93,.13
PALETTE 1,.8,.53,.53
PALETTE 3,.6,.53,0!
x=-20:y=100:r=10
FOR i = 12 TO 50 STEP 7+i^6
 GOSUB hase
```

```
NEXT
END

hase:
a=x+20+i^2/4 :b=y-r*2-i/4
CIRCLE (a,y+i/2),r+i/2,3,,,.9
PAINT (a,y+i/2),3
CIRCLE (a,y-r),r+i/2
PAINT (a,y-r)
CIRCLE (a-i/2,b),r+i/2,,,,5
PAINT (a-i/2,b)
CIRCLE (a+i/2,b),r+i/2,,,,5
PAINT (a+i/2,b),
CIRCLE (a,y+i),i/4,2
PAINT (a,y+i),2
RETURN
```

Zum Abschluß dieses Kapitels noch ein paar Worte zum Thema Abspeichern.

Wenn Sie längere Programme schreiben oder abtippen, empfiehlt es sich, diese zwischendurch immer wieder einmal abzuspeichern. Mit dem Menü-Befehl *Save* geht das einfach und schnell. Es wäre doch schade, wenn Ihnen durch eine kleine Unachtsamkeit die Arbeit von Stunden verlorenginge. Und das kann bei einem Programmierfehler schneller passieren, als Sie vielleicht annehmen.

2.6 Dame

In diesem Kapitel werden Sie die wohl leistungsfähigsten und ungewöhnlichsten Grafik-Befehle von Amiga-Basic kennenlernen: Befehle zum Verschieben ganzer Bildschirmteile. Wäre es nicht phantastisch, wenn Sie dem Amiga sagen könnten: "Nimm den Bildausschnitt in der Mitte des Bildes und lege ihn in die rechte untere Ecke". Genau das geht und es ist sogar recht einfach: GET Bildschirmbereich und PUT Bildschirmbereich.

GET (x1,y1)-(x2,y2),Feld-Variable *(Index,...,Index)*
speichert einen rechteckigen Bildschirmausschnitt in einem numerischen Feld.
PUT *STEP* (x,y),Feld-Variable *(Index,...,Index),Modus*
legt ein abgespeichertes Bild aus einem numerischen Feld wieder auf den Bildschirm

Die Koordinaten-Angaben für GET sind wie bei der Konstruktion eines Rechteckes die obere linke Ecke und die untere rechte Ecke des zu speichernden Bildausschnittes. Die Werte x und y für den PUT-Befehl markieren hingegen die obere linke Ecke des Bereichs, in dem das gespeicherte Bild auf dem Bildschirm auftauchen soll.

Bevor ich mich jetzt lang und breit über die theoretischen Möglichkeiten dieser beiden Superbefehle auslasse, schauen Sie sich einfach das folgende Beispiel an:

```
REM Wald
' P.2.6-1 GET PUT
DIM bild%(167)
PALETTE 0,.8,.6,.47
PALETTE 1,.53,.8,.13
AREA (0,40):AREA (20,40):AREA (10, 0)
AREAFILL
GET (0,0)-(20,40),bild%
FOR j=0 TO 150 STEP 30
  FOR i=0 TO 600 STEP 20
      PUT (i,j),bild%,PSET
NEXT i,j
END
```

In diesem Programm wird Baum als kurzes Ganzzahlenfeld dimensioniert. Genaugenommen, wird allerdings nicht der Baum, sondern ein rechteckiger Bildschirmausschnitt definiert, wie man aus der GET-Anweisung ersehen kann. Wieviele Elemente dieses Feld haben muß, kann man aus den Abmessungen des zu speichernden Bildausschnittes und der Tiefe des Bildschirms nach folgender Formel berechnen:

```
6+(y2-y1+1)*2*INT((x2-x1+16)/16)*tiefe/2
```

Wobei (x1,y1) die linke obere, (x2,y2) die linke untere Ecke des abzuspeichernden Rechtecks und *tiefe* die Anzahl der verwendeten Bitebenen ist. Da wir in diesem Programm nur mit vier Farben arbeiten, ist *tiefe* gleich zwei. Weil wir die Dimensionierung als kurzes Ganzzahlenfeld (ein Element =2 Byte) vorgenommen haben, wird die Farbtiefe durch 2 geteilt. Setzen Sie nun einfach die Werte unseres Baumes in die Formel ein und Sie werden sehen, daß sie harmloser ist, als sie aussieht.

```
6+(40-0+1)*2*INT((20-0+16)/16)*2/2 und wir erhalten 167
```

Wäre die Feld-Variable als langes Ganzzahlenfeld bild& dimensioniert worden, würden sogar schon 83 Elemente genügen.

Verfolgen wir den weiteren Ablauf des Programms Wald. Mit der PALETTE-Anweisung wird zunächst als Hintergrundfarbe Braun und als Vordergrundfarbe Grün gewählt. Mit AREA und AREAFILL zeichnet das Programm dann einen einfachen Baum. Jetzt tritt der neue Befehl GET in Aktion. Der Bildschirmbereich von der oberen linken Ecke bis zu den Koordinaten x=20 und y=40 wird in das Feld Bild% geholt. In zwei in sich geschachtelten FOR-NEXT-Schleifen werden schließlich mit PUT fast 200 Bäume aus diesem Feld wieder auf den Bildschirm plaziert. Beeindruckend ist dabei die rasante Geschwindigkeit, mit der der PUT-Befehl arbeitet.

Dieses kleine Programm war aber nur die Vorspeise zu den vielen Leckerbissen, die das Befehlspaar GET und PUT bietet. Mit dem nächsten Programm bekommen Sie das erste Menü serviert. Wir werden gleich ein recht komfortables Dame-Spiel auf unseren Bildschirm zaubern. Ein Spiel braucht doch bewegliche Steine? Das geht doch nur mit BOBs

oder Sprites?, werden Sie sich vielleicht fragen. Nein, wir brauchen dazu keine Sprites, über die wir erst an anderer Stelle sprechen werden. Das schaffen allein unsere beiden neuen Superbefehle GET und PUT. Fangen wir gleich mit dem Herzstück des Programmes an: dem Spielablauf.

```
REM Dame
'P.2.6-2
DEFINT a-z
DIM stein%(71)
PALETTE 0,.4,.8,1
aus=1
ON MOUSE GOSUB mauskontrolle
MOUSE ON
WHILE aus
   GOSUB ein
   WHILE ende
   SLEEP
   WEND
WEND
END
SYSTEM
```

Nach dem bekannten Befehl DEF werden zunächst alle Variablen als kurze Ganzzahl festgelegt. Dann wird das Feld Stein mit 71 Elementen dimensioniert. Dame spielt man mit schwarzen und weißen Steinen. Daran wollen wir uns natürlich auch halten. Da man aber die schwarze Farbe auf dem voreingestellten, dunkelblauen Hintergrund kaum erkennen kann, setzen wir mit der PALETTE-Anweisung den Hintergrund auf eine hellblaue Farbe.

Jetzt kommt ein sehr interessanter Befehl: ON MOUSE. Er ermöglicht es uns, das Hauptprogramm zu einem solch kurzen Programmablauf zusammenzufassen, wie Sie das im ersten Programmabschnitt sehen.

ON MOUSE GOSUB Sprungmarke

wird die linke Maustaste gedrückt, verzweigt das Programm zu der angegebenen Sprungmarke (dies kann eine Zeilennummer oder ein Label sein)

MOUSE ON

die Wirkung von ON MOUSE wird aktiviert

MOUSE OFF

macht ON MOUSE unwirksam

MOUSE STOP

unterbricht die Wirkung von ON MOUSE vorübergehend

Wie Sie sehen können, gehören diese Befehle zusammen. Wenn Sie in Ihrem Programm nach dem ON MOUSE GOSUB das MOUSE ON vergessen, passiert rein gar nichts. Bei MOUSE STOP merkt sich der Interpreter, ob die linke Maus-Taste gedrückt wurde. Sobald wieder MOUSE ON befohlen wird, verzweigt das Programm zu der entsprechenden Routine. Hinter diesen Mausbefehlen finden sich im Programm zwei ineinandergeschachtelte WHILE-Schleifen. Mit ihnen steuern wir in unserem Programm zwei verschiedene Wahlmöglichkeiten für den Programmbenutzer. Die innere Schleife kann nur durch die linke Maustaste wachgerüttelt werden (SLEEP). Die äußere WHILE-Schleife wird aktiviert, sobald *ende* nicht mehr wahr ist, das heißt, einen Wert von Null erhält. In diesem Programm wird ein Neustart durchgeführt, wobei der Wert für *ende* wieder auf logisch wahr (<>0) gesetzt wird. Sobald auch die äußere Schleife verlassen wird, endet das Programm mit dem Befehl END.

Beschäftigen wir uns aber nun etwas näher mit der ersten Subroutine:

```
ein:
CLS
CALL spielfeld
br=0:farbe=1: anf=11:CALL start(br,farbe,anf)
br=52:farbe=1: anf=34:CALL start(br,farbe,anf)
br=104:farbe=1: anf=11:CALL start(br,farbe,anf)
br=260:farbe=2: anf=11: CALL start(br,farbe,anf)
br=312:farbe=2: anf=34:CALL start(br,farbe,anf)
br=364:farbe=2: anf=11:CALL start(br,farbe,anf)
LINE (0,0)-(50,15),2,bf
LOCATE 1,2:PRINT "neu"
LINE (51,0)-(99,15),3,bf
LOCATE 1,8:PRINT "Ende"
ende=1
RETURN
```

Nach der Zeile, in der der Bildschirm mit CLS gelöscht wurde, können Sie schon wieder einen neuen Befehl entdecken: CALL. Wie der Name schon sagt (engl. to call = rufen), ruft man damit etwas auf. Sie können mit CALL zum Beispiel ein Basic-Unterprogramm (nicht zu verwechseln mit einer Subroutine, die mit GOTO oder GOSUB aufgerufen wird), aber auch Unterprogramme in Maschinensprache und Bibliotheks-Routinen aufrufen. Wir lassen es zunächst einmal beim Aufruf eines Basic-Unterprogramms bewenden.

CALL Sprungmarke (Argumentenliste)
ruft ein Basic-Unterprogramm auf und übergibt diesem eine Liste von Werten (Argumenten)

Was unterscheidet ein Unterprogramm von einer Subroutine? Ein Unterprogramm wird von Basic aus fast so verwaltet, als wäre es ein eigenes Programm. Unterprogramme ver-

walten selbständig ihre Variablen, so daß gleiche Variablen-Namen im Hauptprogramm und im Unterprogramm verschiedene Variablen bezeichnen. Verwenden Sie zum Beispiel eine Variable i im Hauptprogramm und diese hat den Wert 4, so hat ein i, das in einem Unterprogramm verwendet wird, beim Start des Unterprogramms wieder den Wert 0. Weisen Sie dieser Variablen im Unterprogramm einen Wert zu, bleibt er im Unterprogramm erhalten. Nach dem Verlassen des Unterprogramms gibt Basic aber auf alle Fälle wieder 4 aus, wenn Sie PRINT i eingeben. Die beiden im Haupt- und im Unterprogramm haben nämlich bis auf den gleichen Namen nichts miteinander zu tun. Sie haben damit die Möglichkeit, sich eine Bibliothek mit immer wiederkehrenden Routinen anzulegen, ohne daß Sie dabei auf die Variablen achten müssen, die innerhalb der Bibliotheksroutinen verwendet werden, wie Sie es bei Subroutinen tun müssen.

Ein Unterprogramm, das mit CALL aufgerufen werden kann, wird durch die folgenden Anweisungen definiert.

SUB Name *(Variablenliste)* STATIC

Anfang eines Basic-Unterprogrammes

END SUB

Ende eines Basic-Unterprogrammes

EXIT SUB

Verlassen eines Basic-Unterprogrammes vor Erreichen des Endes

Im Programm Dame werden bei CALL Spielfeld keine Programm-Variablen an das Unterprogramm übergeben. Diese werden später noch behandelt. Es folgen gleich sechs Aufrufe an ein Unterprogramm namens *start*, bei denen dem Unterprogramm drei verschiedene Variablen als Argumente mit auf den Weg gegeben werden: br=breite vom linken Spielfeldrand; die Farbe Schwarz oder Weiß und *anf* für die Positionierung der einzelnen Spielsteine in y-Richtung. Anschließend setzen wir den Text neu in ein schwarzes Rechteck links oben im Fenster und daneben Ende in ein rotes Rechteck. Der letzte Befehl dieser Subroutine setzt die Variable *ende* wieder auf logisch wahr. Schauen Sie sich nun zunächst das erste Unterprogramm etwas genauer an:

```
SUB spielfeld STATIC
fa=1
FOR j=0 TO 161 STEP 23
  FOR i=100 TO 412 STEP 104
  LINE (i,j)-(i+51,j+22),fa,b
  LINE (i+1,j+1)-(i+50,j+21),fa,b
  fa=fa+1
  IF fa>2 THEN fa=1
  LINE (i+52,j)-(i+103,j+22),fa,b
  LINE (i+53,j+1)-(i+102,j+21),fa,b
  fa=fa+1
  IF fa>2 THEN fa=1
```

```
   NEXT i
   fa=fa+1
   IF fa>2 THEN fa=1
NEXT j
END SUB
```

Die Variable *fa* enthält die Farbe, mit der unser Spielfeld auf den Bildschirm kommt. Aus Gründen, die weiter unten noch besprochen werden, können die Spielsteine nicht auf zwei verschiedene Untergrundfarben gesetzt werden. Damit der Spieler aber trotzdem weiß, wo er seine Steinchen hinsetzen soll, zeichnen wir ihm Rechtecke mit der jeweiligen Farbe seiner Spielsteine. Da ein Rechteck in normaler Strichstärke nicht besonders gut wirkt, zeichnen wir zwei Rechtecke ineinander. Das geschieht durch die beiden LINE-Befehle, die direkt untereinanderstehen. Die beiden darauffolgenden ineinander geschachtelten FOR-Schleifen laufen folgendermaßen ab: erste Reihe, von links nach rechts, weißes Rechteck, Farbwechsel, schwarzes Rechteck, Farbwechsel usw., bis acht Rechtecke gezeichnet sind. Dann wird die innere Schleife verlassen, die Farbe fa gewechselt, so daß nun mit einem schwarzen Rechteck begonnen wird, es geht zurück in die innere Schleife. Das ganze passiert achtmal und das Spielfeld (acht Zeilen) ist fertig.

Das zweite Unterprogramm setzt die zwei mal zwölf Spielsteine (weiße und schwarze Kreise) auf die Startpositionen:

```
SUB start(b,f,a) STATIC
FOR i=a TO (a+138) STEP 46
   CIRCLE (126+b,i),15,f,,,.52
   PAINT (126+b,i),f
NEXT i
END SUB
```

Da alle nötigen Werte dem Unterprogramm mitgegeben werden, fällt es sehr kurz aus. Mit der CIRCLE-Anweisung werden auf jedes zweite Feld, von oben nach unten die Steine gezeichnet. Da bei CIRCLE der voreingestellte Wert 1 für Ratio anstelle von Kreisen nur unförmige Eier zeichnet, wurde dieser Wert auf .52 korrigiert. Nun kann das Spiel beginnen. Werfen Sie zuvor einen Blick auf den Kern der Programmsteuerung. Alles was Sie mit der Maus machen können, wird hier verarbeitet:

```
mauskontrolle:
GOSUB posholen
IF posx <50 AND posy<15 THEN ende=0:RETURN
IF posx <100 AND posy<15 THEN ende=0:aus=0:RETURN
GOSUB poskorr
GET (x,y)-(x+30,y+16),stein
PUT(x,y),stein
WHILE MOUSE(0)<>0
   GOSUB posholen
   posx=posx-15:posy=posy-8
   PUT(posx,posy),stein
   FOR i=1 TO 100 : NEXT i 'Zeitschleife
   PUT(posx,posy),stein
```

```
WEND
GOSUB poskorr
PUT(x,y),stein
CALL spielfeld
RETURN
```

Zu Beginn wird die Position der Maus erfragt und dann überprüft, ob der Spieler in eines der beiden Felder neu oder Ende geklickt hat. Das Programm geht im ersten Fall auf Neustart und beendet im zweiten Fall das Programm, indem die Variablen »ende und/oder aus« auf logisch falsch (0) gesetzt werden. Dadurch wird die WHILE-WEND-Schleife verlassen.

Wenn weder Ende noch neu angeklickt wurde, wird eine korrigierte Position berechnet, mit welcher dann der Spielstein eingelesen wird (GET). Daraufhin wird er wieder sofort auf die gleiche Position gesetzt, was ihn auf Nimmerwiedersehn verschwinden läßt.

Die Position der Maus und die Frage, ob die Maustaste gedrückt wurde, werden mit einer weiteren Funktion MOUSE festgestellt, die bislang noch nicht erläutert wurde.

v=MOUSE(n)
liefert je nach n verschiedene Informationen über den Stand der Maustaste und Mausposition

Um zu wissen, ob sich am Zustand der Maus etwas geändert hat, fragt man MOUSE(0) ab. Ergibt diese Funktion einen Wert <> 0, dann ist eine Änderung des Mauszustands eingetreten. Die anderen Möglichkeiten für n (1-6) werden bei deren Auftreten im Programm erläutert.

Die WHILE-WEND-Schleife wird so lange durchlaufen, wie die Maustaste gedrückt bleibt. Bei jedem Schleifendurchlauf wird die aktuelle Position geholt, auf die linke obere Ecke des eingelesenen (GET) Bildausschnittes gesetzt und dieser dann mit PUT wieder auf den Bildschirm gebracht. Nach einer kurzen Pause wird der Spielstein durch nochmaliges PUT wieder gelöscht. Da der Amiga bei den meisten Grafikbefehlen sehr schnell ist, würde man den Spielstein bei einem so schnellen Wechsel zwischen GET und PUT gar nicht erkennen. Deshalb wird das Programm an dieser Stelle durch eine Zeitschleife etwas gebremst. Sobald die linke Maustaste losgelassen wird, wird die WHILE-WEND-Schleife verlassen und wieder die korrigierte Position der Maus ermittelt. An dieser korrigierten Position wird der Stein auf das Spielfeld plaziert. Zu guter Letzt wird das Unterprogramm Spielfeld noch einmal aufgerufen, damit das Spielfeld wieder restauriert wird. Betrachten Sie nun die Subroutine *posholen*, die die Mausposition feststellt:

```
posholen:
muss=MOUSE(0)
posx=MOUSE(1)
posy=MOUSE(2)
RETURN
```

Zuerst steht eine Funktion, die scheinbar keinen Sinn hat, sie muß im Programm nicht weiterverarbeitet werden. Dieser Funktionsaufruf ist aber notwendig, damit die beiden darauffolgenden überhaupt eine Information liefern! MOUSE(1) liefert dann die x-Koordinate und MOUSE(2) die y-Koordinate zum Zeitpunkt des letzten MOUSE(0) -Aufrufes. Wenn Sie vergessen, vorher MOUSE(0) aufzurufen, ergibt beides Null. Nachdem Sie jetzt schon wissen, wie man von Basic aus mit der Maus umgeht, nun zur letzten Subroutine des Programms:

```
poskorr:
GOSUB posholen
x=(INT((posx-48)/52)*52)+74
y=(INT(posy/23)*23)+11
x=x-15
y=y-8
RETURN
```

Auch in dieser Subroutine wird zu Beginn die Mausposition abgefragt und dann in zwei Formeln weiterverarbeitet. Hier wird eine Position, die irgendwo in einem Quadrat des Spielfeldes liegt, in Koordinaten umgerechnet, die exakt in der Mitte dieses Quadrats liegen.

Um alle verbleibenden Fragen zum Programm zu klären, tippen Sie es ein und starten Sie es, sobald Sie es abgespeichert haben. Nachdem das Spielfeld fertig gezeichnet ist, können Sie mit der Maus aktiv ins Geschehen eingreifen. Klicken Sie mit der Maus einen Spielstein an und halten Sie die Maustaste fest. Der Stein »springt« Ihnen dann unter den Mauszeiger und bleibt mitten in dem Feld liegen, in welchem Sie die Maustaste wieder loslassen. Es ist zum Bewegen eines Steins nicht notwendig, daß Sie ihn exakt in der Mitte anklicken. Hauptsache, Ihr Mauszeiger befindet sich irgendwo innerhalb des betreffenden Quadrats, wenn Sie die Maustaste drücken. Die exakte Positionierung übernimmt das Programm. Geschlagene Steine können links und rechts des Spielfeldes abgelegt werden. Wenn ein Spiel, das Sie gegen sich selbst oder mit einem Partner spielten, beendet ist, und Sie wollen weitermachen, klicken Sie einfach das Feld neu an. Schon bekommen Sie ein frisches Spielfeld mit den Spielsteinen auf den Startpositionen. Wenn Sie das Spiel satthaben, klicken Sie einfach Ende oder wählen den Menü-Befehl *Stop*.

Haben Sie schon einmal probiert, einen weißen auf einen schwarzen Stein zu legen? Falls nicht, versuchen Sie es einmal. Sie erhalten statt des weißen oder schwarzen Steins nun einen roten Stein (der sich gut als Dame verwenden läßt). Sollten Sie allerdings auf die Idee kommen, zwei gleichfarbige Steine aufeinanderzulegen, verschwinden diese wie weggezaubert. Wir wollen nun einmal versuchen, diesem Phänomen auf den Grund zu gehen.

Schuld an diesem Verhalten ist der Modus der PUT-Anweisung. Wenn Sie sich unser Dame-Spiel noch einmal anschauen, werden Sie feststellen, daß dieser Modus nirgends verwendet wird. Aber wie bei den meisten Befehlen ist auch hier ein bestimmter Modus voreingestellt, der verwendet wird, wenn keiner angegeben wird: XOR. Außerdem gibt es noch die Modis PSET, PRESET, AND und OR. Interessant sind für uns vorläufig nur PSET und der Standardwert XOR, der im Dame-Spiel ausschließlich verwendet wurde.

Bei der ausschließenden Oder-Verknüpfung, die XOR bewirkt, werden zwei Wahrheitswerte (Bits) wie folgt verknüpft:

```
1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0
```

Wie Sie sehen, liefern zwei falsche und zwei wahre Operanden ein unwahres Ergebnis. Nur wenn die beiden Operanden ungleich sind, ergibt dies eine 1 (= wahr). Im Klartext bedeutet dies für die Spiel-Steine, daß immer dann, wenn Sie zwei gleichfarbige Steine aufeinandersetzen, der Stein verschwindet (da alle Punkte gleich 0 werden). Das gleiche Prinzip wurde ja auch beim Verschieben eines Steines genutzt. Hierzu wurde, mit einer kleinen Pause, die durch eine Zeitschleife bewirkt wird, der Stein zweimal nacheinander auf die gleiche Stelle gesetzt. Der Stein erscheint dann und verschwindet wieder, um eventuell an anderer Stelle wieder aufzutauchen.

2.7 Gewitter

Bisher haben Sie mit dem GET-Befehl Grafiken kurzzeitig in einer Variablen ablegen und mit dem PUT-Befehl wieder auf den Bildschirm zaubern können. Doch was machen Sie, wenn Sie Bilder längere Zeit aufheben wollen? Diesem Problem soll in diesem Abschnitt zu Leibe gerückt werden. Selbstverständlich ist es aber wieder in ein hübsches Beispiel-Programm eingepackt. Es beginnt mit den folgenden Zeilen:

```
REM Gewitter
REM P.2.7-1
CLEAR ,45000&
```

Das fängt ja gut an! Gleich in der ersten Programmzeile taucht ein neuer Befehl auf.

> **CLEAR** *,Programmspeicher,Stapelspeicher*
> löscht alle Variablen und legt die Speicheraufteilung neu fest

Wenn Sie Amiga-Basic aufrufen, erhalten Sie im Ausgabefenster einige Daten. In der letzten Zeile dieser Aufstellung steht: 25 000 Byte free in Basic. Diese 25 000 Byte reserviert Basic für Ihr Programm. Für den Stapelspeicher sind es nur 4789 Byte, aber das ist meist völlig ausreichend. Die 25 000 Byte für das Programm sind aber zu wenig. Im Programm Gewitter wird mit der GET-Anweisung der komplette Bildschirm im Speicher abgelegt. Hierzu werden allein schon über 38 Kbyte Speicherplatz benötigt. Da das Programm selbst auch noch Speicherplatz braucht, werden mit CLEAR 45 000 Byte reserviert.

```
SCREEN 1,320,200,5,1
WINDOW 2,">O<pen >S<ave >D<raw >E<rase >Q<uit",
** (0,0)(310,180),0,1
GOSUB farbtafel

start:
IF tim THEN TIMER OFF:tim=0
CLS
zei=0
PRINT "Bild einlesen = O"
PRINT "Bild speichern = S"
PRINT "Bild zeichnen = D"
PRINT "Bild loeschen = E"
PRINT "Programmende = Q"
```

Dieses Programm wird durch Tastendrücke gesteuert. Damit jederzeit ersichtlich ist, welche Taste was bewirkt, werden die entsprechenden Tasten sowohl mit PRINT ausgegeben, wie auch als Fenster-Titel permanent sichtbar gemacht. Die Befehle hinter der Sprungmarke *start* schreiben das Menü mit PRINT auf den Bildschirm, während hinter der Sprungmarke *auswahl* auf die Tastatureingabe gewartet und zu den einzelnen Unterprogrammen verzweigt wird.

```
auswahl:
 A$ = INKEY$
 IF A$ <>"" THEN A$ = UCASE$(A$)
 IF A$ = "O" THEN einlesen
 IF A$ = "S" THEN speichern
 IF A$ = "D" THEN zeichnen
 IF A$ = "E" THEN start
 IF A$ = "Q" THEN ende
GOTO auswahl

ende:
SCREEN CLOSE 1
END

einlesen:
IF tim THEN TIMER OFF:tim=0
CLS
INPUT "Bitte Filenamen: ",filename$
IF filename$="" THEN einlesen
OPEN filename$ FOR INPUT AS 1
GOSUB bilddim
LOCATE 15,15 : PRINT "bitte 50 sec warten"
i=0
WHILE NOT EOF (1)
 Bild&(i) = CVL(INPUT$(4,1))
 i=i+1
WEND
CLS
```

```
PUT (0,0),Bild&,PSET
ERASE Bild&
CLOSE #1
GOTO auswahl
```

Die Subroutine *einlesen* bittet den Programmbenutzer zuerst um den Namen eines Bildes, welches eingelesen und dargestellt werden soll. Der eingegebene Text wird an die String-Variable filename$ übergeben. Wenn der Benutzer keinen Namen eingibt und nur die RETURN-Taste drückt, verzweigt das Programm zurück zu *einlesen.* Sobald ein Name vorliegt, wird versucht, die entsprechende Datei einzulesen:

OPEN "Dateiname" *FOR Modus* AS #Dateinummer
öffnet eine Datei zur Datenein- oder -ausgabe

Für Modus stehen zur Auswahl:

- OUTPUT für sequentielle Ausgabe,
- APPEND zum Anhängen an eine sequentielle Datei,
- INPUT für sequentielle Eingabe.

In diesem Beispiel wird eine Datei mit der Nummer 1 für die sequentielle Eingabe geöffnet.

Nach dem Aufruf der Subroutine *bilddim* und der Ausgabe einer Meldung auf dem Bildschirm folgt dann eine WHILE-Schleife, die es in sich hat. Abbruchbedingung für diese Schleife ist die Funktion EOF.

v=EOF(Dateinummer)
liefert dann den logischen Wert wahr, wenn beim Lesen einer sequentiellen Datei das Datei-Ende erreicht wurde

Die WHILE-Schleife wird also so lange durchlaufen, bis das Ende der Daten erreicht ist. Sie können deshalb auch Dateien einlesen, deren Länge Ihnen unbekannt ist. Die Zeile innerhalb der WHILE-Schleife ist dafür verantwortlich, daß die Daten eingelesen und in den Elementen von bild& abgelegt werden. Sie verwendet dazu zwei neue Funktionen.

v$ = INPUT$(Länge,#Dateinummer)
übergibt eine Zeichenkette bestimmter Länge von einer sequentiellen Datei (oder von der Tastatur) an eine String-Variable

Dann wird die Funktion CVL verwendet, um einen String der Länge 4 (also 4 Buchstaben = 4 Byte) in eine lange Ganzzahl umzuwandeln. CVL ist nur eine von diversen Funktionen, die Basic für solche und ähnliche Zwecke zur Verfügung stellt. Für die Umwandlung des Inhaltes von Strings in numerische Werte können Sie die folgenden Funktionen verwenden:

v=ASC(v$)	1-Byte-Zeichen (ASCII-Code) wird Dezimalwert
v=CVI(v$)	2-Byte-Zeichenkette wird kurze Ganzzahl
v=CVL(v$)	4-Byte-Zeichenkette wird lange Ganzzahl
v=CVS(v$)	4-Byte-Zeichenkette wird einfachgenaue Zahl
v=CVD(v$)	8-Byte-Zeichenkette wird doppeltgenaue Zahl

Die Datenübernahme erscheint auf den ersten Blick recht kompliziert. Da Daten in einer sequentiellen Datei grundsätzlich als Folge von ASCII-Zeichen gespeichert werden, auch wenn es sich dabei um Zahlen handelt, müssen die Zahlen bei der Eingabe zunächst als Zeichenketten eingelesen und dann umgewandelt werden. Wegen der vier verschiedenen Zahlendarstellungen werden auch vier Funktionen für diesen Zweck benötigt. Die fünfte Funktion, welche an erster Stelle aufgeführt ist, brauchen Sie dann, wenn Sie nur ein einzelnes Byte einlesen wollen.

Mit PUT bringen wir das eingelesene Bild dann auf den Bildschirm und löschen es mit ERASE aus dem Speicher:

ERASE Feld-Variable,*Feld-Variable*
löscht die genannten Feld-Variablen und gibt notwendigen Speicherplatz frei

Die Subroutine *einlesen* ist, wie Sie gesehen haben, dafür verantwortlich, eine Datei einzulesen und ihren Inhalt dann als Bild auf den Bildschirm zu bringen. Enthält diese Datei kein Bild, kann das übrigens interessante Effekte ergeben. Das Pendant zu *einlesen*, mit dem Bilder in eine Datei geschrieben werden können, ist *speichern*. Nur Bilder, die mit *speichern* oder einem ähnlich funktionierenden Programm auf der Diskette abgelegt wurden, können mit *einlesen* wieder zurückgeholt werden.

```
speichern:
IF tim THEN TIMER OFF:tim=0
GOSUB bilddim
GET (0,0)-(310,180),Bild&
CLS
INPUT "Bitte Filenamen: ",filename$
IF filename$<>"" THEN
 OPEN filename$ FOR OUTPUT AS 1
 LOCATE 15,15 : PRINT "bitte 40 sec warten"
 FOR i = 0 TO bildinhalt& -1
 PRINT #1,MKL$(Bild&(i));
 NEXT i
END IF
```

```
CLOSE #1
ERASE Bild&
GOTO start
```

Beim Speichern geht das Programm exakt in der umgekehrten Reihenfolge vor. Zuerst wird mit GET das Bild in eine Feldvariable geholt, dann fragt das Programm nach dem Datei-Namen und öffnet eine Datei mit diesem Namen für die sequentielle Ausgabe. Die Ausgabe der Daten an die Diskette erfolgt dann mit einer FOR-Schleife, da die Länge der Datei ja (von der Größe des Feldes her) klar ist. Auch hierbei werden wieder zwei neue Befehle verwendet.

PRINT #Datei-Nummer,*USING n$;* Ausdruck1, Ausdruck2, ...
gibt die Werte der Ausdrücke an eine sequentielle Datei aus
n$ ist dabei ein String, der exakt angibt, wie die Zahlen oder Strings ausgegeben werden sollen. Diese Möglichkeit wird im Beispielprogramm aber nicht verwendet

Bevor eines der Elemente von bild& aber ausgegeben wird, muß es zunächst einmal in einen String umgewandelt werden. Hierfür wird die Konvertierungsfunktion MKL$ verwendet. Für die Umwandlung von numerischen Werten in Zeichenketten stehen Ihnen die folgenden Funktionen zur Verfügung:

v$=CHR$(n)	Wert des ASCII-Codes eines Zeichens wird 1-Byte-Zeichenkette
v$=MKI$(A%)	kurze Ganzzahl wird 2-Byte-Zeichenkette
v$=MKL$(A&)	lange Ganzzahl wird 4-Byte-Zeichenkette
v$=MKS$(A!)	einfachgenaue Zahl wird 4-Byte-Zeichenkette
v$=MKD$(A#)	doppeltgenaue Zahl wird 8-Byte-Zeichenkette

Sie sehen, daß im Programmteil *speichern* nun der umgekehrte Weg gegangen wird und die Zahlen in Zeichenketten umgewandelt werden, damit sie abgespeichert werden können. Auch hier wird zum Schluß der Routine mit ERASE das Feld bild& wieder aus dem Speicher gelöscht, da es sehr viel Platz benötigt. Wieviel Platz es benötigt, ergibt sich aus der Dimensionierung des Feldes (mit DIM), die in diesem Programm einmal nicht am Anfang, sondern in einer separaten Subroutine erfolgt.

```
bilddim:
bildinhalt&=(9510)
DIM Bild&(bildinhalt&)
RETURN
```

Die Zahl für die Dimensionierung wurde nach der Formel aus Kapitel 2.6 berechnet. bild& hat 9510 Elemente und benötigt, da jedes der Elemente eine lange Ganzzahl ist, damit 38040 Byte.

Damit Sie aber nicht unbedingt erst ein Bild einlesen müssen, um etwas auf dem Bildschirm zu sehen, wird im Programmteil *zeichnen* ein recht einfaches Bild (drei Hochhäuser mit einer Reihe von Fenstern sowie ein Blitz auf der linken Seite) auf den

Bildschirm gemalt. Dieses Bild können Sie dann abspeichern oder durch ein anderes ersetzen, indem Sie das andere Bild einlesen.

```
zeichnen:
IF tim THEN TIMER OFF:tim=0
CLS
LINE (0,0)-(310,180),19,bf
br = 30
FOR h = 0 TO 2
   LINE (br,40)-(br+62,180),28+h,bf
   FOR i = br+3 TO br+53 STEP 10
        FOR j = 45 TO 165 STEP 10
             LINE (i,j)-(i+5,j+5),19,bf
        NEXT j
   NEXT i
   br=br+100
NEXT h
FOR i = 0 TO 1
   LINE (10+i,10)-(15+i,20),18
   LINE -(8+i,25),18
   LINE -(17+i,35),18
   LINE -(12+i,39),18
   LINE -(27+i,50),18
   LINE -(20+i,70),18
   LINE -(28+i,95),18
NEXT i
ON TIMER(5) GOSUB lau:TIMER ON :tim=1
GOTO auswahl

lau:
FOR j = 1 TO 5
   GOSUB fat
   FOR i=1 TO 10 : NEXT i
   GOSUB farbtafel
NEXT j
RETURN
```

Werfen Sie doch mal einen kleinen Blick auf die in diesem Programm verwendete Palette (Farbtafel). Fällt Ihnen dort etwas auf? Richtig! Die Farben 18 und 19 haben die gleichen Farbwerte. Wenn wir das Bild normal aufrufen, kann man deshalb den Blitz gar nicht sehen, da er dieselbe Farbe wie der Himmel hat. Das Erscheinen des Blitzes ist in dem Programmabschnitt *lau* versteckt, der in regelmäßigen Zeitabständen angesprungen wird. Aber wie kann man es realisieren, daß eine Routine nur in bestimmten Zeitabständen angesprungen wird? Eine Möglichkeit wäre es, eine exakt abgestimmte Zeitschleife zu schreiben. Dies ist aber aufwendig und überflüssig, da Amiga-Basic einen entsprechenden Befehl vorsieht.

ON TIMER(n) GOSUB Sprungmarke oder Zeilennummer
alle n Sekunden verzweigt das Programm zur Sprungmarke/Zeile

Damit die Unterbrechungsreaktionsfähigkeit (ein schlimmes Wort) in Kraft gesetzt wird, muß jedoch noch ein zweiter Befehl aufgerufen werden.

TIMER ON
ON TIMER wird aktiviert
TIMER OFF
die Wirkung ON TIMER wird abgeschaltet
TIMER STOP
die Wirkung von ON TIMER wird vorübergehend unterbrochen

Bei TIMER STOP merkt sich der Interpreter, ob die mit ON TIMER bestimmte Zeit inzwischen abgelaufen ist. Sobald wieder TIMER ON im Programm-Ablauf erscheint, verzweigt das Programm sofort zu der entsprechenden Routine. Wenn Sie jedoch TIMER OFF befehlen, wird die »Stoppuhr« angehalten und beginnt erst wieder zu laufen, wenn Sie mit TIMER ON aktivieren. Am Ende dieses Kapitels finden Sie noch ein Beispiel zu TIMER, mit dem Sie einige Zeit-Experimente durchführen können.

In der Subroutine lau wird nach Ablauf des Timers in die zweite Farbtafel fat gesprungen. Dadurch ändert sich die Farbe 18 in Gelb, die Farbe 19 in Blau und schon erscheint der Blitz. Anschließend wird die normale Farbtafel wiederhergestellt. Wichtig hierbei ist vor allem, daß der Timer abgeschaltet wird, wenn sich das Programm in einem anderen Programmteil befindet. Läuft der Timer während des Speicherns eines Bildes, kann das die Daten ganz schön durcheinanderbringen. Das Bild werden Sie jedenfalls nicht wiedererkennen. Mit der Programmzeile

```
IF tim THEN TIMER OFF: tim=0
```

wird der Timer abgeschaltet und dies in der Variablen tim vermerkt.

```
farbtafel:
 PALETTE 0,1,1,1 'weiss
 PALETTE 2,1,0,0 'dunkelrot
 PALETTE 3,.93,.2,0 'feuerwehrrot
 PALETTE 1,.4,.6,0 'dunkelgruen
 PALETTE 18,.2,0,.8 'dunkelblau 2
 PALETTE 19,.2,0,.8 'dunkelblau 2
 PALETTE 28,.4,.2,0 'braungrau
 PALETTE 29,.4,.4,.4 'dunkelgrau
 PALETTE 30,.6,.6,.6 'mittelgrau
RETURN
```

```
fat:
 PALETTE 18,1,1,0 'gelb
 PALETTE 19,.2,.75,1 'blau
RETURN
```

Obwohl das Programm einen SCREEN im 32-Farben-Modus benutzt, werden nur 9 Farben verwendet. Sie haben also noch genug Möglichkeiten, das Bild farbiger zu gestalten. Probieren Sie es ruhig.

Die hier vorgestellte Methode, Grafiken abzuspeichern und einzulesen, besticht durch ihre Einfachheit. Sie hat jedoch den Nachteil, daß solche Dateien nur von Basic-Programmen, und nicht von Graficraft oder Deluxe Paint verwendet werden können. Wie Sie Grafiken in einem passenden Format für Graficraft oder Deluxe Paint abspeichern und einlesen können (im sogenannten IFF-Format), erfahren Sie in Kapitel 3.3.

Bevor wir uns aber ans Werk machen, das gerade vorgestellte Programm weiter auszubauen, hier das versprochene Übungsprogramm zur TIMER-Anweisung:

```
REM Mauer
'P.2.7-2 TIMER-Funktion
LINE (0,0)-(630,190),,bf
x2=20:y2=5
y=180:x=x-24
t=TIMER
ON TIMER(.2) GOSUB zeit:TIMER ON
WHILE 1
  SLEEP
WEND

ende:
TIMER OFF
LOCATE 1,60:PRINT "FEIERABEND"
END

zeit:
x=x+24: IF x>623 THEN x=-12:y=y-8
IF x>611 THEN x=0:y=y-8
IF y<0 THEN ende
LINE (x,y)-(x+x2,y+y2),3,bf
LOCATE 1,1:PRINT "Programm-Dauer:"TIMER-t"Sekunden "
RETURN
```

Das Programm mauert eine Wand aus roten Ziegelsteinen auf den Bildschirm. Jedesmal wenn die n Sekunden, die mit der Anweisung ON TIMER (n) bestimmt wurden, abgelaufen sind, wird ein neuer Ziegelstein zur Mauer hinzugefügt. n darf dabei maximal 86400 (= 24 Stunden) betragen. Ich glaube aber nicht, daß Sie soviel Zeit haben. Bei diesem Wert von n läuft das Programm nämlich fast zwei Jahre! Sie können aber auch Werte unter einer Sekunde einsetzen; die Mauer wächst dann sehr schnell. Probieren Sie ruhig ein paar verschiedene Angaben für n aus. In der ersten Zeile des Bildschirms erhalten Sie während des Programmablaufs fortwährend die bisherige Laufzeit

ausgedruckt, so daß Sie Ihre Experimente gut verfolgen können.

Doch wie kann das Programm feststellen, wie lange es schon läuft? Natürlich wieder durch eine Funktion.

v=TIMER
übergibt die Anzahl Sekunden, die seit Mitternacht oder seit dem Einschalten des Rechners vergangen sind

Im Programm Mauer wird nun zu Programmbeginn diese Zeit mit der TIMER-Funktion geholt und in der Variablen t gespeichert. Bei jedem Aufruf der Subroutine zeit durch den Timer wird t dann von der aktuellen Zeit abgezogen und das Ergebnis auf dem Bildschirm ausgegeben.

2.8 PAINTAmiga

Nachdem Sie nun fast alle wesentlichen Grundzüge der Grafikprogrammierung auf dem Amiga kennengelernt haben, können Sie auch größere Projekte angehen. Das folgende Programm ist ein solches Projekt: ein Mal-Programm, das es gewiß nicht mit Deluxe Paint aufnehmen will, aber einen guten Eindruck davon vermittelt, zu was auch das gute alte Basic auf dem Amiga fähig ist. Da das Programm einen ziemlich großen Umfang angenommen hat, werde ich darauf verzichten, es durch Erklärungen zu unterbrechen. Es wird auch so schwierig genug für Sie werden, es fehlerfrei abzutippen. Die nötigen Erläuterungen zur Programmierung bekommen Sie nach dem Listing ab Abschnitt 2.8.1. Bevor Sie sich gleich daranmachen, es abzuschreiben und auszuprobieren, beachten Sie bitte auch die Anmerkungen in Kapitel 2.8.1. Programmzeilen, welche länger als die Druckzeilen sind, wurden mit zwei Sternen am Anfang der Zeile markiert. Mit "**" gekennzeichnete Zeilen gehören also zur vorhergehenden Zeile und müssen durchgehend eingetippt werden. Und schließlich sollten Sie, bevor Sie nicht ganz sicher sind, daß das Programm fehlerfrei läuft, die Programmzeile "ON BREAK GOSUB ..." nicht eintippen, beziehungsweise ein REM davorsetzen. Diese Zeile verhindert, daß das Programm während des Laufs unterbrochen wird, was in einem fehlerhaften Programm verhängnisvoll sein kann.

```
REM PAINTAmiga
REM P.2.8
CLEAR,100000&:DEFLNG a-z:SCREEN 1,320,200,5,1: WINDOW
** 2,,,0,1
DIM rot!(31),gruen!(31),blau!(31):GOSUB Farbtafel

anfang:
GOSUB MenuEin
xb=WINDOW(2):yb=WINDOW(3)
```

```
DIM var(106):p$="SP/pins?":GOSUB Pinselstandard
GOSUB bilddim1
GET(0,10)-(310,190),Bild1
zur=0:pinse=1:ko=0:Beginn=1:arbeit=0
```

Ablauf:

```
ON MENU GOSUB MenuKontrolle : MENU ON
ON MOUSE GOSUB MausKontrolle : MOUSE ON
ON BREAK GOSUB BreakKontrolle : BREAK ON
GOSUB titel:ende = -1
WHILE ende
  SLEEP
WEND
SCREEN CLOSE 1: MENU RESET:CLS:ERASE Bild1:END
REM MAUSROUTINEN
REM ************
```

MausKontrolle:

```
IF arbeit THEN RETURN
GOSUB posholen:GOSUB zurueck:startx=posx:starty=posy
IF Farbeneu THEN GOSUB neufarbeholen:RETURN
IF Farbwahl THEN GOSUB Farbeholen : RETURN
IF Lupenwahl THEN GOSUB Lupeholen : RETURN
IF Pinselwahl THEN Pinselholen ELSE GOSUB pinselausf
ON Werkzeugart GOSUB
** Stift,Linie,Rechteck,Ellipse,Kreis,fuellen,
** Radierer,Feld,Kopie,blind
Beginn=0: RETURN
blind: RETURN
```

Lupeholen:

```
pins1=0: IF lu THEN Lupenbild
GOSUB posholen:IF posy<11 THEN RETURN
IF posx<90 AND posy >y5 AND posy<(y5+71) THEN RETURN
klx=posx-8:kly=posy-6
GET (klx-1,kly-1)-(klx+20,kly+12),pins1
LINE (klx,kly)-(klx+19,kly+11),31,b
LINE (klx-1,kly-1)-(klx+20,kly+12),0,b
a1=0: GOSUB fensterauf
WINDOW OUTPUT 2
FOR j=(kly+1) TO (kly+12)
  a2=0
  FOR i=(klx+1) TO (klx+18)
       farb=POINT(i,j)
       WINDOW OUTPUT 3
       i1=(a2)*4:j1=(a1)*4
       LINE (i1,j1)-(i1+3,j1+3),farb,bf
       WINDOW OUTPUT 2
       a2=a2+1
```

```
  NEXT i
  a1=a1+1
NEXT j
lu=1: RETURN
```

Lupenbild:

```
Werkzeugart=10:pinsl=0
WINDOW OUTPUT 3
GOSUB posholen:IF posx>80 THEN lupezu
gx=INT(posx/4)*4:gy=INT(posy/4)*4
LINE (gx,gy)-(gx+3,gy+3),farbe,bf
WINDOW OUTPUT 2
kx=INT(posx/4)+1:ky=INT(posy/4)+1
PSET (kx+klx,ky+kly),farbe:RETURN
```

lupezu:

```
WINDOW OUTPUT 2
GET (klx+1,kly+1)-(klx+18,kly+10),pins2
PUT (klx-1,kly-1),pins1,PSET
PUT (klx+1,kly+1),pins2,PSET
ERASE pins1:ERASE pins2:ERASE pins3:ERASE pinsf
GOSUB fensterzu:GOSUB Pinselstandard
x5=0:y5=0:Lupenwahl=0:xb=310:yb=180: RETURN
```

neufarbe:

```
GOSUB fensterauf
LINE (0,0)-(68,28),farbe,bf
GOSUB rgbsetzen:Farbeneu=-1: RETURN
```

rgbsetzen:

```
LOCATE 1,1 : PRINT "+ + +";
LOCATE 3,1 : PRINT "R G B";
LOCATE 5,1 : PRINT "- - -";:RETURN
```

neufarbeholen:

```
GOSUB posholen
IF posx>70 OR posy>40 THEN Farbeneu=0:GOSUB fensterzu:IF
** pinsl THEN GOTO pinselfarbe ELSE RETURN
IF posx>40 THEN
   IF posy<15 THEN
   blau!(farbe)=blau!(farbe)+.067
   IF blau!(farbe)>1 THEN blau!(farbe)=1
   ELSE
        blau!(farbe)=blau!(farbe)-.067
        IF blau!(farbe)<0 THEN blau!(farbe)=0
   END IF
END IF
IF posx>20 THEN
   IF posy<15 THEN
        gruen!(farbe)=gruen!(farbe)+.067
```

```
      IF gruen!(farbe)>1 THEN gruen!(farbe)=1
   ELSE
      gruen!(farbe)=gruen!(farbe)-.067
      IF gruen!(farbe)<0 THEN gruen!(farbe)=0
   END IF
END IF
IF posx<21 THEN
   IF posy<15 THEN
      rot!(farbe)=rot!(farbe)+.067
      IF rot!(farbe)>1 THEN rot!(farbe)=1
   ELSE
      rot!(farbe)=rot!(farbe)-.067
      IF rot!(farbe)<0 THEN rot!(farbe)=0
   END IF
END IF
PALETTE farbe,rot!(farbe),gruen!(farbe),blau!(farbe): RETURN
```

Farbeholen:

```
posix=MOUSE(3):posiy=MOUSE(4)
IF posiy>10 THEN RETURN
fbe=INT(posix/9)
IF fbe>31 THEN RETURN
farbe=fbe:COLOR farbe:GOSUB titel:Farbwahl=0
IF pinsl THEN GOTO pinselfarbe ELSE RETURN
```

pinselfarbe:

```
GOSUB fensterauf:GOSUB varhoL:PUT (10,10),var,OR
FOR iy = 0 TO 19
   FOR ix = 0 TO 16
      va=POINT(ix+10,iy+10)
      IF va THEN PSET(ix+10,iy+10),farbe
   NEXT ix
NEXT iy
IF pinsl=1 THEN
   GET (10,10)-(26,29),pins1
ELSEIF pinsl=2 THEN
   GET (10,10)-(26,29),pins2
ELSEIF pinsl=3 THEN
   GET (10,10)-(26,29),pins3
ELSEIF pinsl=4 THEN
   GET (10,10)-(26,29),pinsf
END IF
x5=0: GOSUB fensterzu:p1=1: RETURN
```

```
Pinselholen:
posix=MOUSE(3):posiy=MOUSE(4)
IF posix>70 OR posiy>40 THEN 12
IF posiy<18 THEN pinsl=INT(ABS(posix/15)) :p1=1: GOTO 12
IF posiy<32 THEN pinse=2 ELSE pinse=3:Pinselwahl=0 : CLS:
** RETURN
12 y5=0:GOSUB fensterzu
xb=310:yb=180:Pinselwahl=0: RETURN

pinselausf:
IF pinse=2 THEN pinsellesen
IF pinse=3 THEN pinselkonstr
RETURN

pinsellesen:
GOSUB fensterauf:FILES "WP/":PRINT "Name eingeben"
INPUT p$:IF p$="" THEN pinse=1: GOTO fensterzu
p$="WP/"+p$:GOSUB wahlpins:pinse=1
GOSUB fensterzu:RETURN

pinselkonstr:
Werkzeugart=10
LINE (0,8)-(19,8),31: LINE -(19,31),31: LINE (0,31)-
** (18,38),5,bf
LOCATE 1,1 : PRINT "loeschen";:LOCATE 5,1 : PRINT "ok";
GOSUB posholen:IF posx>16 THEN RETURN
IF posy<38 AND posy>28 THEN xb=15 : yb=19 :PL$="WP/":
** bildinhalt=89:DIM Bild(bildinhalt): GOSUB Speichernals2 :
** pinse=1:p1=1: xb=310:yb=180 : GOTO fensterzu
IF posy>37 THEN RETURN
IF posy<9 THEN CLS: GOTO pinselkonstr
PSET (posx,posy),31
WHILE MOUSE(0)<>0
  GOSUB posholen:LINE -(posx,posy),31
WEND:RETURN

Stift:
GOSUB posholen:IF posy<11 THEN RETURN
IF pinsl THEN Stiftpinsel
arbeit=-1
PSET (posx,posy),farbe
WHILE MOUSE(0)<>0
  GOSUB posholen:LINE -(posx,posy),farbe
WEND:arbeit=0:RETURN

Stiftpinsel:
arbeit=-1
IF p1 THEN GOSUB varhoL
PUT (posx-8,posy-8),var,OR
WHILE MOUSE(0)<>0
```

```
   GOSUB posholen:PUT (posx-8,posy-8),var,OR
WEND:arbeit=0:RETURN
```

varhoL:

```
IF pinsl = 1 THEN
   FOR i=0 TO 106: var(i) = pins1(i) : NEXT i
ELSEIF pinsl = 2 THEN
   FOR i=0 TO 106: var(i) = pins2(i) : NEXT i
ELSEIF pinsl = 3 THEN
   FOR i=0 TO 106: var(i) = pins3(i) : NEXT i
ELSEIF pinsl = 4 THEN
   FOR i=0 TO 106: var(i) = pinsf(i) : NEXT i
END IF
p1=0: RETURN
```

Linie:

```
arbeit=-1:blo=0: GOSUB untermaus
LINE (startx,starty)-(posx,posy),farbe:arbeit=0:RETURN
```

untermaus:

```
GOSUB posholen:IF posy<11 THEN RETURN
WHILE MOUSE(0)<>0
   GOSUB posholen:GOSUB bilddim:GET (0,10)-(310,190),Bild
   IF blo THEN
        LINE (startx,starty)-(posx,posy),farbe,b
   ELSE
        LINE (startx,starty)-(posx,posy),farbe
   END IF
   FOR i = 1 TO 50 : NEXT i
   PUT (0,10),Bild,PSET:ERASE Bild
WEND:RETURN
```

Rechteck:

```
arbeit=-1:blo=1: GOSUB untermaus
LINE (startx,starty)-(posx,posy),farbe,b:arbeit=0:RETURN
```

Ellipse:

```
arbeit=-1
blo=1: GOSUB untermaus
IF posx>startx THEN
   rx=(posx-startx)/2:mx=startx+rx
ELSE
   rx=(startx-posx)/2:mx=startx-rx
END IF
IF posy>starty THEN
   ry=(posy-starty)/2:my=starty+ry
ELSE
   ry=(starty-posy)/2:my=starty-ry
END IF
IF rx THEN el!=ABS(ry/rx) ELSE arbeit=0:RETURN
```

```
IF ry>rx THEN rx=ry
IF rx<1 THEN arbeit=0:RETURN
CIRCLE (mx,my),rx,farbe,,,el!:arbeit=0:RETURN
```

Kopie:

```
IF ko THEN Kopie1
arbeit=-1
blo=1: GOSUB untermaus
sx=startx:sy=starty:px=posx : py=posy
xb=ABS(startx-posx):yb=ABS(starty-posy)
GOSUB bilddim:GET (sx,sy)-(px,py),Bild
ko=-1:arbeit=0: RETURN
```

Kopie1:

```
arbeit=-1
GOSUB posholen:IF posy<11 THEN 9
WHILE MOUSE(0)<>0
  GOSUB posholen:PUT (posx,posy),Bild
  FOR i=1 TO 100 : NEXT i 'Verzoegerung
  PUT (posx,posy),Bild
WEND
PUT (posx,posy),Bild
9 ERASE Bild
xb=310:yb=180:ko=0:arbeit=0:RETURN
```

Kreis:

```
GOSUB posholen:IF posy<11 THEN RETURN
arbeit=-1
WHILE MOUSE(0)<>0
  GOSUB posholen: GOSUB bilddim
  GET (0,10)-(310,190),Bild
  x2=posx-startx:y2=starty-posy
  R=SQR(x2^2+y2^2):R=CINT(R)
  CIRCLE (startx,starty),R,farbe,,,1.05
  FOR i=1 TO 40 : NEXT i
  PUT (0,10),Bild,PSET:ERASE Bild
WEND
CIRCLE (startx,starty),R,farbe,,,1.05:arbeit=0: RETURN
```

Feld:

```
GOSUB posholen:IF posy<11 THEN RETURN
arbeit=-1
WHILE MOUSE(0)<>0
  GOSUB posholen
  LINE (startx,starty)-(posx,posy),farbe
WEND:arbeit=0:RETURN
```

```
fuellen:
GOSUB posholen:IF posy<11 THEN RETURN
arbeit=-1
WHILE MOUSE(0)<>0
  GOSUB posholen:PAINT (posx,posy),farbe
WEND:arbeit=0:RETURN

Radierer:
arbeit=-1
blo=1: GOSUB untermaus
LINE (startx,starty)-(posx,posy),0,bf:arbeit=0:RETURN
REM Menueroutinen
REM ************

MenuEin:
MENU 1,0,1,"Projekt" :MENU 1,1,1,"New" :MENU 1,2,1,"Open"
MENU 1,3,1,"Save" :MENU 1,4,1,"Save As" :MENU 1,5,1,"Quit"
MENU 2,0,1,"Werkzeug":MENU 2,1,1,"Stift" :MENU 2,2,1,"Linie"
MENU 2,3,1,"Rechteck":MENU 2,4,1,"Ellipse" :MENU
** 2,5,1,"Kreis"
MENU 2,6,1,"fuellen" :MENU 2,7,1,"Radierer":MENU
** 2,8,1,"Flaeche"
MENU 2,9,1,"Kopie" :MENU 3,0,1,"Farben" :MENU 3,1,1," "
MENU 4,0,1,"SPEZIAL" :MENU 4,1,1,"zurueck" :MENU
** 4,2,1,"Pinsel"
MENU 4,3,1,"Lupe ob" :MENU 4,4,1,"Lupe un" :MENU 4,5,1,"SW-
** HC"
MENU 4,6,1,"neuFarb"
RETURN

MenuKontrolle:
IF arbeit THEN RETURN
MenuTitel = MENU(0)
MenuNr = MENU(1)
ON MenuTitel GOTO
** MenuFiles,MenuWerkzeug,MenuFarben,MenuSpezial

MenuFiles:
arbeit=-1
ON MenuNr GOSUB Neu,Einlesen,speichern,speichernals,FileQuit
arbeit=0
RETURN

MenuWerkzeug:
Werkzeugart=MenuNr:GOTO titel

MenuFarben:
FA = 0 : far=0
WHILE FA<280
  LINE (0+FA,0)-(9+FA,8),far,bf
```

```
  FA=FA+9:far=far+1
WEND
Farbwahl=-1: RETURN

MenuSpezial:
arbeit=-1
ON MenuNr GOSUB zck,pinsel,Lupeo,lupeu,HCSW,neufarbe
arbeit=0
RETURN

zurueck:
IF zur THEN PUT (0,10),Bild1,PSET : zur=0 :Werkzeugart=10:
** RETURN
ERASE Bild1:GOSUB bilddim1
GET (0,10)-(310,190),Bild1:RETURN

zck:
zur=-1: RETURN

pinsel:
GOSUB fensterauf:pi = 0
WHILE pi<70
  LINE (0+pi,0)-(15+pi,18),31,b:pi=pi+15
WEND
CIRCLE (8,8),1,31
PUT (14,0),pins1,PSET:PUT (29,0),pins2,PSET
PUT (44,0),pins3,PSET:PUT (59,0),pinsf,PSET
LOCATE 4,1 : PRINT"holen";:LOCATE 5,1 : PRINT "fertigen";
xb=310:yb=180:Werkzeugart=1:Pinselwahl=-1: RETURN

fensterauf:
IF fens THEN RETURN
GOSUB bilddim2:GET (0,8+y5)-(80,53+y5),Bild2
WINDOW 3,,(0,10+y5)-(70 ,40+y5),0,1:fens=1: RETURN

fensterzu:
WINDOW CLOSE 3: PUT (0,8+y5),Bild2,PSET
ERASE Bild2
fens=0:RETURN

Lupeo:
y5=0:GOTO Lupe

lupeu:
y5=100

Lupe:
Lupenwahl=-1:lu=0: RETURN

FileQuit:
GOSUB Neu:ende = 0: RETURN
```

```
Neu:
IF Beginn THEN RETURN
GOSUB bilddim:GET (0,10)-(310,190),Bild:CLS:BEEP
PRINT "Das aktuelle Bild ist nicht gespeichert!"
PRINT "Bild speichern.... = S":PRINT "Bild loeschen.....
** = L"
10 a$ = INKEY$ : IF a$ = "" THEN 10
a$ = UCASE$(a$)
IF a$ = "S" THEN PUT (0,10),Bild,PSET : ERASE Bild : GOSUB
** speichernals:GOSUB Farbtafel: RETURN
IF a$ = "L" THEN CLS : ERASE Bild :GOSUB Farbtafel: RETURN
BEEP:GOTO 10

BreakKontrolle:
RETURN

Pinselstandard:
DIM pins1(106):DIM pins2(106)
DIM pins3(106):DIM pinsf(106)
OPEN "SP/pins1" FOR INPUT AS 1:i=0:WHILE NOT EOF (1)
pins1(i) = CVL(INPUT$(4,1)):i=i+1:WEND:CLOSE #1
OPEN "SP/pins2" FOR INPUT AS 1:i=0:WHILE NOT EOF (1)
pins2(i) = CVL(INPUT$(4,1)):i=i+1:WEND:CLOSE #1
OPEN "SP/pins3" FOR INPUT AS 1:i=0:WHILE NOT EOF (1)
pins3(i) = CVL(INPUT$(4,1)):i=i+1:WEND:CLOSE #1

wahlpins:
i=0:OPEN p$ FOR INPUT AS 1:WHILE NOT EOF (1)
pinsf(i) = CVL(INPUT$(4,1)):i=i+1:WEND:CLOSE # 1: RETURN
REM allgemeine Routinen
REM ********************

posholen:
muss=MOUSE(0):posx=MOUSE(1):posy=MOUSE(2):RETURN

bilddim:
bildinhalt =10000:DIM Bild(bildinhalt):RETURN

bilddim1:
bildinhalt1 = 9510: DIM Bild1(bildinhalt1):RETURN

bilddim2:
bildinhalt2 = 700: DIM Bild2(bildinhalt2):RETURN

titel:
LOCATE 1,1:PRINT "PAINTAmiga M&T by H-R Henning ********";
:RETURN

Farbtafel:
RESTORE
FOR i=0 TO 31
  READ rot!(i),gruen!(i),blau!(i)
```

```
  PALETTE i,rot!(i),gruen!(i),blau!(i)
NEXT
farbwerte=-1
DATA 1,1,1, .4,.6,0, 1,0,0, .93,.2,0
DATA 1,.4,0, 1,.6,0, .4,.4,.4, 1,1,0
DATA .6,1,.13, .53,.8,.13, .8,0,0, .2,.4,0
DATA 0,.4,0, 0,.6,.67, 0,.8,.6, 0,1,.6
DATA .2,1,.93, .2,.73,1, .13,.4,1, .4,0,1
DATA 0,0,.6, .33,.13,.87, .6,.2,1, 1,0,1
DATA .93,.53,.73, 1,.73,.73, .8,.53,.53, .6,.4,.33
DATA .4,.2,0, 1,.8,0, .6,.6,.6, 0,0,0
RETURN

HCSW:
REM !!!!!!!!!!!!!HIER SPAETER EINFUEGEN!!!!!!!!!!!!!!!!!!
RETURN

einlesen1:
CLS:COLOR 31 :FILES PL$
INPUT "Bitte Filenamen: ",filename$
IF filename$="" THEN GOSUB titel : RETURN
filenam$=PL$+filename$
OPEN filenam$ FOR INPUT AS 1:GOSUB bilddim
LOCATE 15,15 : PRINT "bitte 50 sec warten":i=0
WHILE NOT EOF (1)
  Bild(i) = CVL(INPUT$(4,1))
  i=i+1
WEND
CLS:PUT (0,10),Bild,PSET:ERASE Bild:CLOSE #1:RETURN

Speichernals1:
GOSUB bilddim

Speichernals2:
GET (0,10)-(xb,yb+10),Bild:CLS:PRINT "Name eingeben: "
INPUT filename$
GOTO 20

speichern1:
GOSUB bilddim:GET (0,10)-(xb,yb+10),Bild:CLS
20 REM
IF filename$<>"" THEN
  filenam$=PL$+filename$
  OPEN filenam$ FOR OUTPUT AS 1
  LOCATE 15,15 : PRINT "bitte 40 sec warten"
  FOR i = 0 TO bildinhalt -1
      PRINT #1, MKL$(Bild(i));
  NEXT i
END IF:CLS:CLOSE #1:ERASE Bild:GOTO titel
Einlesen: PL$="":GOTO einlesen1
```

```
speichernals: PL$="":GOTO Speichernals1
speichern: PL$="":GOTO speichern1
```

Die Druck-Routine ist in diesem Listing nicht ausgedruckt. Da hierfür detaillierte Erläuterungen notwendig sind, wurde sie herausgenommen. Sie finden sie zusammen mit diesen Erläuterungen in Kapitel 3.2.

Das Programm hat in seiner vollständigen Version eine Länge von über 15 Kbyte und ist schon recht groß und unübersichtlich. Zum besseren Verständnis sind hier die wichtigsten Funktionen in einem vereinfachten Ablaufschema verdeutlicht:

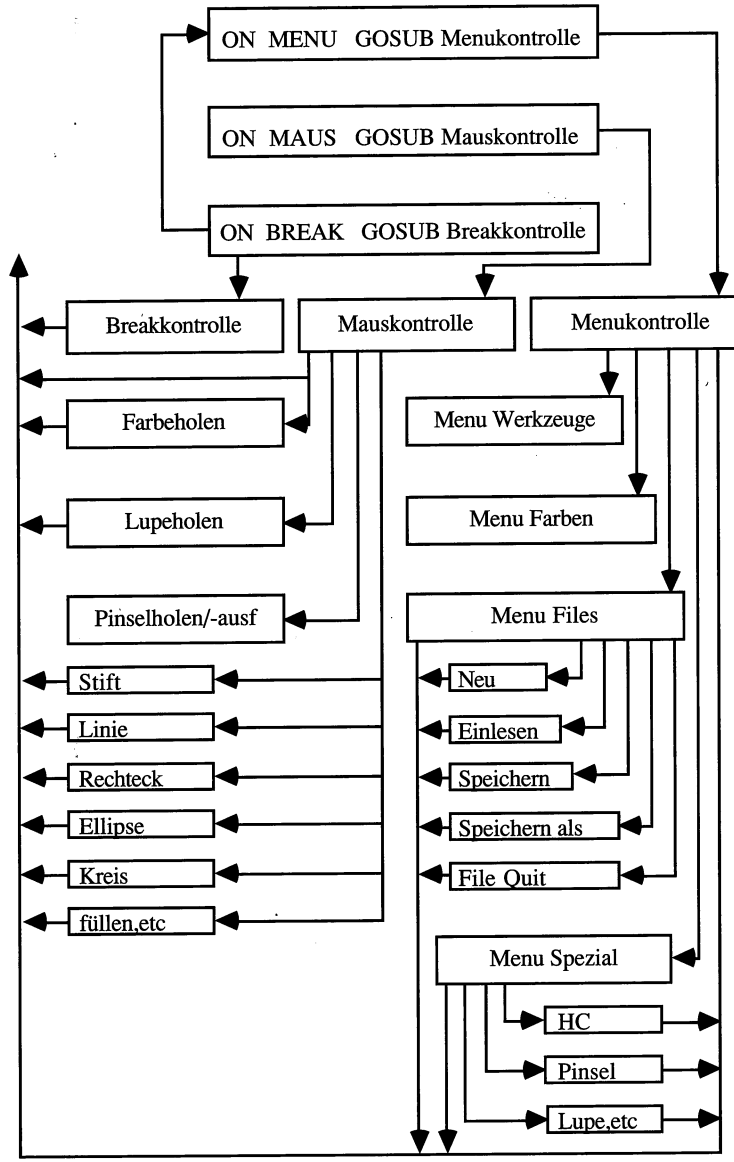


Bild 2.1: *Ablaufschema PAINTAmiga*

2.8.1 PAINTAmiga im Detail

Bevor das Programm im Detail beschrieben wird, zunächst ein grober Überblick über den Programmablauf in wenigen Worten. Das Programm wartet (SLEEP) in einer WHILE-WEND-Schleife, bis ein Unterbrechungs-Ereignis eintritt. Basic verzweigt dann wegen der ON-Anweisungen automatisch zu der Routine, die für dieses Ereignis zuständig ist. Die drei Ereignisse, die eine Unterbrechung hervorrufen können, sind die rechte Maustaste (ON MENU GOSUB), die linke Maustaste (ON MOUSE GOSUB) und die Tasten für die Programmunterbrechung (ON BREAK GOSUB). Letztere bewirkt nichts weiter, als daß das Programm unbeeinträchtigt weiterläuft, da in der entsprechenden Unterroutine nur RETURN steht. Deshalb läßt sich das Programm nicht unterbrechen.

Beim Anwählen eines Programm-Punktes mit der rechten Maustaste in den einzelnen Pull-down-Menüs springt das Programm in die Subroutine *MenuKontrolle.* Dort erfolgt die Weiterverzweigung in die für die einzelnen Menü-Befehle zuständigen Subroutinen. In diesen werden hauptsächlich die Vorbereitungen dafür getroffen, daß mit der Maus (linke Taste) gearbeitet werden kann. Wenn Sie einmal einen Blick auf diese Routinen werfen, werden Sie feststellen, daß bis auf die Hardcopy-Routine, überall »Zeiger« gesetzt werden. Diese Zeiger sorgen dann dafür, daß bei dem zweiten Hauptverteiler der Routine MausKontrolle in die richtige Unterroutine verzweigt wird.

Nun aber zu den Details: Am Anfang des Programms gibt es nichts Besonderes; die hier benutzten Befehle sollten Ihnen inzwischen vertraut sein. Dann darauffolgend aber gleich vier neue Anweisungen:

ON MENU GOSUB Sprungmarke
verzweigt zur Sprungmarke, wenn ein Menüpunkt gewählt wird

ON BREAK GOSUB Sprungmarke
verzweigt zur Sprungmarke, wenn CTRL-C, Amiga-. gedrückt wird oder der Stopp-Befehl aus dem Run-Menü gewählt wird

Damit diese beiden Anweisungen wirksam oder unwirksam werden können, werden – wie schon von ON MOUSE und ON TIMER bekannt – weitere Befehle benötigt:

BREAK ON
macht ON BREAK wirksam

BREAK OFF
macht ON BREAK unwirksam

BREAK STOP
hebt die Wirkung ON BREAK vorübergehend auf

MENU ON
macht ON MENU unwirksam
MENU OFF
macht ON MENU unwirksam
MENU STOP
hebt die Wirkung ON MENU vorübergehend auf

(Die im Programm auf die ON-Befehle folgende Anweisung MENU RESET wird zusammen mit der Subroutine MenuEin erläutert.)

Kommen wir nun zu den Subroutinen des Programms im einzelnen.

Mauskontrolle: Holt die Startposition der Maus, welche für die verschiedenen Werkzeugarten benötigt wird und verzweigt dann in die einzelnen Unterroutinen.

Lupeholen: Wenn die Lupe bereits eingeschaltet ist, wird nach Lupenbild verzweigt. Dann wird geprüft, ob der Benutzer im Bereich des späteren Lupenfensters geklickt hat. Da dieses zum Absturz führen würde, wird der Mausklick ignoriert. Das zu vergrößernde Rechteck wird dann abgespeichert und schließlich mit einem Rahmen versehen. Nun wird das Fenster 3 geöffnet und anschließend wieder Fenster 2 aktiviert.

Dann werden die einzelnen Bildpunkte im markierten Bereich gelesen und mit 16facher Vergrößerung im Lupenfenster gezeichnet; das heißt, aus jedem Punkt des Originals wird ein 4 Punkte breites Quadrat in der Lupe. Hierzu wird die folgende Funktion verwendet.

v=POINT(x,y)
liefert die Farbkennung (gemäß PALETTE) des Punktes bei den angegebenen Koordinaten (x,y)

Lupenbild: Zuerst wird geprüft, ob die Maus rechts vom Lupenfenster geklickt wurde; wenn ja, wird nach Lupezu verzweigt, wenn nicht, wird die Mausposition umgewandelt, damit der vergrößerte Punkt an die richtige Position kommt. Dann werden der vergrößerte und der normal große Punkt nacheinander gesetzt.

Lupezu: Der neu gezeichnete Bildausschnitt wird abgespeichert, der Zustand davor wiederhergestellt und dann der neue Bildausschnitt gesetzt. Dadurch verschwindet auch das um den vergrößerten Ausschnitt gezeichnete Rechteck. Dann werden die Standardpinsel gelöscht, da diese zur Zwischenspeicherung der Bildausschnitte mißbraucht wurden, und wieder neu eingelesen.

neufarbe: Es wird ein Fenster geöffnet und mit der aktuellen Pinsel-Farbe gefüllt. Dann wird der Text "RGB" mit jeweils einem Pluszeichen (+) darüber und einem Minuszeichen (-) darunter ausgegeben.

neufarbeholen: Je nach der Position des Grafik-Cursors im Augenblick des Mausklicks wird einer der drei Farbanteile Rot, Grün und Blau der aktuellen Pinselfarbe um 0.067 (ein Sechzehntel) erhöht oder erniedrigt. Die Positions-Abfrage wird durch eine geschachtelte IF-THEN-ELSE-Konstruktion bewerkstelligt. Mit PALETTE wird die neue Farbeinstellung dann wirksam gemacht.

Farbeholen: MOUSE(3) liefert die x-Koordinate und MOUSE(4) die y-Koordinate des letzten Mausklicks. Hat der Anwender in die Palette geklickt, wird diese Farbe zur aktuellen Pinselfarbe.

pinselfarbe: Fenster 3 wird geöffnet, der aktuelle Pinsel gezeigt und in der zuletzt gewählten Farbe dargestellt. Nachdem der geänderte Pinsel abgespeichert ist, wird Fenster 3 wieder geschlossen.

pinselholen: Wird die Maustaste betätigt, während der Mauszeiger außerhalb des Pinselfensters ist, wird das Fenster wieder geschlossen. Beim Anklicken im Bereich y<18 wird entsprechend der x-Position berechnet, welcher Pinsel gewählt wurde. Wählt der Anwender das Einlesen oder Konstruieren eines Pinsels, wird ein entsprechender Wert in der Variablen pinse abgelegt.

pinsellesen: Die bereits früher abgespeicherten Wahlpinsel (frei wählbare Pinsel) werden im Fenster 3 aufgelistet. Der Befehl FILES "WP" bedeutet, daß der Inhalt der Schublade WP im aktuellen Fenster aufgelistet wird. Das Unterprogramm fragt anschließend nach dem Namen des einzulesenden Pinsels. Wird RETURN betätigt, ohne einen Namen einzugeben, wird das Pinselfenster geschlossen und ins Hauptprogramm zurückgesprungen. In der Unterroutine wahlpins wird der gewünschte Pinsel dann eingelesen.

pinselkonstr: Nachdem die Werkzeugart auf 10 (kein Werkzeug) gesetzt wurde, wird ein Rechteck gezeichnet, in dem der neue Pinsel erstellt werden kann. Darauf folgen vier IF-THEN-Anweisungen. Je nachdem, wo der Mausklick erfolgt, wird die Eingabe ignoriert, der Pinsel abgespeichert oder der bereits gezeichnete Pinsel wieder gelöscht. Bei der Speicherung eines neuen Pinsels gilt für den File-Namen das gleiche, was schon im vorhergegangenen Abschnitt zum Thema Einlesen gesagt wurde. In der WHILE-WEND-Schleife, die darauf folgt, wird schließlich der neue oder geänderte Pinsel gezeichnet.

Stiftpinsel: Wenn die Variable p1 wahr ist, wird der aktuelle Pinsel in die Variable var geholt und so mit PUT auf den Bildschirm plaziert, daß der Mauszeiger im Zentrum des Pinsels steht. Der Modus für PUT ist in diesem Fall OR. Der Pinsel wird also mit dem bereits vorhandenen Bild verknüpft. (Siehe hierzu auch Kapitel 1.6). Probieren Sie ruhig einmal einen anderen Modus aus (zum Beispiel PSET). Vielleicht gefällt Ihnen das Ergebnis besser.

Das Programm PAINTAmiga bietet dem Benutzer eine Anzahl von Werkzeugen, die er beim Malen verwenden kann. Für fast all diese Werkzeuge ist es nötig, auf dem Bildschirm eine Linie oder ein Rechteck zu markieren, während der Benutzer die Maus bei gedrückter linker Taste bewegt. Diese Markierung übernimmt die Routine *untermaus.*

untermaus: Zuerst wird die aktuelle Mausposition festgestellt und der augenblickliche Bildschirminhalt mit Hilfe von GET in der Variablen *Bild* gespeichert. Danach wird eine

Linie oder ein Rechteck in der aktuellen Farbe gezeichnet und nach einer kleinen Pause wieder gelöscht, indem der alte Bildschirminhalt mit PUT wiederhergestellt wird. Sobald die Maustaste nicht mehr gedrückt ist, wird die WHILE-Schleife verlassen.

Kopie: Zur Realisation dieses Werkzeugs werden zwei Routinen benötigt. Die erste speichert den Inhalt des Rechteckes aus *untermaus* ab. Da die Größe dieses Rechteckes variabel ist, wird dabei genügend Platz für den ganzen Bildschirm reserviert. Der Zeiger *ko* wird dann auf -1 (logisch wahr) gesetzt, damit das Programm beim nächsten Durchlauf in die Unterroutine *Kopie1* gelangt.

Kopie1: Die zweite Kopierroutine wird wieder durch einen Mausklick aktiv. Das in Kopie abgespeicherte Bild folgt hier den Bewegungen des Mauszeigers, solange die Maustaste gedrückt wird. Bei jedem Durchlauf der WHILE-Schleife wird der PUT-Befehl zweimal aufgerufen. Beim ersten Mal erscheint das Bild und wird dann nach kurzer Wartezeit durch die zweite PUT-Anweisung wieder gelöscht. Erst wenn die Maustaste gelöscht wird, wird die WHILE-Schleife verlassen und das Bild dann ein letztes Mal mit PUT an seinem neuen Standort plaziert.

MenuEin: Auch die Einbeziehung von Pull-down-Menüs in eigene Programme ist bei Amiga-Basic ganz einfach zu bewerkstelligen. Hierzu werden im wesentlichen die folgenden Basic-Befehle benötigt:

MENU Kennung,Menüpunkt,Status,*Titel*
erzeugt einen neuen Menü-Punkt im Menü mit der Nummer Kennung oder ein neues Menü mit dem genannten Titel

MENU RESET
löscht alle vom Programm definierten Menüs aus der Menü-Leiste und zeigt wieder die Menüs von Amiga-Basic

Betrachtet man Unterroutine MenuEin, so sieht man, daß als Kennungen die Zahlen eins bis vier auftauchen. Es werden also vier Menüs definiert. (Maximal kann Basic zehn Menüs verwalten.) Die Anzahl der Menüpunkte in den einzelnen Menüs ist unterschiedlich und liegt in unserem Programm zwischen null und neun. (Basic läßt bis zu neunzehn Punkte zu.)

Wenn man als Menüpunkt die Nummer Null angibt, wird ein neues Menü erzeugt, ansonsten bekommt ein schon vorhandenes Menü einen neuen Punkt. Status ist in diesem Programm immer gleich 1, das heißt, alle Menüs und Menüpunkte sind aktiv.

MenuFarben: Eine WHILE-WEND-Schleife zeichnet 32 kleine Rechtecke in die Menüleiste, die die 32 möglichen Farben enthalten.

zurueck: Dieses winzig kleine Unterprogramm kann für den Anwender eine riesengroße Wirkung haben. Wie schon der Name dieser Routine sagt, holt es ein eventuell zerstörtes Bild zurück. Besser gesagt, es bringt das Bild wieder in den Zustand zurück, in dem es

vor der letzten Änderung war. Natürlich wird hierzu wieder GET und PUT verwendet. In der Routine *anfang* wird bei jedem Programmdurchlauf mit einer GET-Anweisung der komplette Bildschirminhalt als Bild1 gespeichert. Bei jeder Mausbetätigung wird die Unterroutine *zurück* aufgerufen. Wenn der Menüpunkt *zurück* nicht ausgewählt wurde, dann wird Bild1 zunächst gelöscht, und dann wird der jetzige Zustand des Bildschirms wieder in Bild1 abgelegt. Wünscht der Anwender jedoch, das vorhergehende Bild zurückzuholen, wird es einfach mit einem PUT-Befehl aus Bild1 wieder auf den Bildschirm gebracht.

pinsel: Öffnet Fenster3 und stellt die fünf möglichen Pinsel im neuen Fenster zur Wahl. Als weitere Wahlmöglichkeit werden die Punkte *holen* (zum Einlesen eines Wahlpinsels) und *fertigen* (zum Konstruieren eines neuen Wahlpinsels) im Pinselfenster ausgegeben.

fensterauf: Der Bereich des Bildes, in dem das neue Fenster auftauchen wird, wird als Bild2 gespeichert. Das Fenster3 wird dann mit einer WINDOW-Anweisung geöffnet.

fensterzu: Fenster 3 wird durch die Anweisung WINDOW CLOSE 3 veranlaßt. Danach wird der dahinterliegende Teil des Bildes mit dem PUT-Befehl wiederhergestellt. (Dieser Teil wurde ja in *fensterauf* in der Variablen Bild2 gespeichert.)

Neu: Wenn noch keine Zeichnung vorliegt, erfolgt ein Sprung zurück an den Anfang des Hauptprogramms. Andernfalls wird der Bildschirm zunächst einmal als *bild* gespeichert. Untermalt von einem akustischen Signal, wird der Anwender dann gefragt, ob er das Bild wirklich löschen und ein neues beginnen will. Das Programm wartet dann auf eine Tastatureingabe (S oder L). Es ist dabei egal, ob der Benutzer die Antwort als Klein- oder Großbuchstabe gibt. Mit der folgenden Funktion werden nämlich vor der Überprüfung alle Buchstaben in Großbuchstaben umgewandelt.

v$=UCASE$(x$)
wandelt die Zeichenkette x$ in Großbuchstaben um

Pinselstandard: Hier werden die drei Standardpinsel und ein Wahlpinsel von der Diskette eingelesen. Da diese Subroutine in zwei Routinen aufgeteilt ist, kann die Unterroutine *wahlpins* auch einzeln angesprungen werden. Von dieser Möglichkeit machen wir bei der Änderung des letzten Pinsels Gebrauch.

Farbtafel: Hier werden die Farbanteile für die Farben der Palette aus DATA-Zeilen gelesen und den drei indizierten Variablen rot!(), gruen!() und blau!() zugewiesen. Diese Variablen werden dann in den darauffolgenden PALETTE-Anweisungen verwendet, um die Farben zu definieren.

Speichernals: Diese Routine speichert das aktuelle Bild auf der Diskette ab. Sie entspricht derjenigen aus dem letzten Kapitel. Wählt der Anwender nur *Save* und nicht *Save As*, so wird er nicht nach einem Datei-Namen gefragt. Statt dessen wird der Dateiname, der seit dem letzten Einlesen eines Bildes in der Variablen *filename$* gespeichert ist, verwendet.

Ich hoffe, daß Sie das Programm ohne Fehler eingegeben haben. Überprüfen Sie es sicherheitshalber noch einmal anhand des Listings! Starten Sie es aber noch nicht sofort, sondern lesen Sie erst den nächsten Abschnitt. Einige Funktionen arbeiten nur nach bestimmten Vorarbeiten.

2.8.2 PAINTAmiga – es läuft

Wenn Sie vorhaben, PAINTAmiga intensiver zu nutzen, schlage ich vor, daß Sie dafür eine eigene Diskette anlegen. (Ein abgespeichertes Bild verschlingt nahezu 40 Kbyte.) Ob Sie das tun oder mit der Diskette BFA weiterarbeiten – die folgenden Punkte müssen Sie auf jeden Fall berücksichtigen. Amiga-Basic muß auf derselben Diskette vorhanden sein, auf der sich auch das Programm PAINTAmiga befindet. Legen Sie dann zwei leere Schubladen an und benennen Sie diese (mit *Rename* aus dem Workbench-Menü Project) SP (für Standardpinsel) und WP (für Wahlpinsel).

Weiterhin ist der normale Mauszeiger für exaktes Arbeiten in Grafiken denkbar ungeeignet. Sie sollten sich deshalb einen anderen Mauszeiger erstellen, der auch für fast alle anderen Anwendungen sehr gut zu gebrauchen ist. Legen Sie dazu die Workbench-Diskette ein und öffnen Sie deren Icon mit einem Doppelklick. Starten Sie das Programm Preferences und wählen Sie den Schalter *Edit Pointer* an. Erstellen Sie dann den folgenden Mauszeiger.

```
oooo21ooo12oooo
oooo21ooo12oooo
oooo21ooo12oooo
oooo21ooo12oooo
222221ooo122222     o = blau
111111ooo111111     1 = braun
oooooooooooooooo    2 = schwarz
ooooooo3ooooooo     3 = rot
oooooooooooooooo
111111ooo111111
222221ooo122222
oooo21ooo12oooo
oooo21ooo12oooo
oooo21ooo12oooo
oooo21ooo12oooo
```

Bild 2.2: *Schema für einen verbesserten Mauszeiger*

Auf das rote Quadrat in der Mitte setzen Sie zum Schluß bitte den heißen Punkt. Klicken Sie dazu den Schalter *Set Point* an und dann den Punkt. (Lassen Sie sich nicht durch das längliche Format der Schemazeichnung täuschen. In Wirklichkeit wird der Mauszeiger genau quadratisch.) Betätigen Sie nun den Schalter OK und dann (im Hauptfenster von Preferences) den Schalter *Save*.

Jetzt haben Sie den richtigen Mauszeiger und die Schubladen für das Programm PAINTAmiga, können aber immer noch nicht anfangen! Die Schubladen mit den Pinseln sind ja noch leer. Wenn Sie jetzt das Programm starten, sucht es in der Schublade *SP*

nach den Standardpinseln, findet sie nicht und bricht das Programm mit der Fehlermeldung *File not found* ab. Wir müssen also zuerst darangehen, die Standard-Pinsel einzugeben. Brauchen wir dafür nicht ein Zeichenprogramm? Sie haben recht, aber Sie verfügen ja auch schon über ein Zeichenprogramm. Sie müssen es zu diesem speziellen Zweck nur etwas abändern.

Folgende Programmzeilen müssen für die Konstruktion der Standard-Pinsel geändert werden:

```
bei anfang GOSUB Pinselstandard in REM GOSUB Pinselstandard
bei pinsel
** PUT (14,0),pins1,PSET.. in REM PUT (14,0),pins1, PSET..
   PUT (44,0),.......... in REM PUT (44,0)..........
bei pinselkonstr
** IF posy<38 AND posy>28........:PL$="WP/":.......
in IF posy<38 AND posy>28........:PL$="SP/":.......
```

Geben Sie diese Änderungen ein und starten Sie PAINTAmiga. Warten Sie, bis die Titelleiste PAINTAmiga by.... erscheint. Wählen Sie dann aus dem Pull-down-Menü SPEZIAL den Menüpunkt *Pinsel* aus. Es öffnet sich das Pinselfenster. In diesem neuen Fenster klicken Sie mit der Maus auf "fertigen" und klicken dann noch ein zweites Mal. Es erscheint nun in diesem Fenster ein fast quadratisches Zeichenfeld. In diesem Feld zeichnen Sie als erstes einen kleinen gefüllten Kreis. Wenn Ihnen die Zeichnung nicht gefällt, können Sie diese durch einen Klick auf das Wort "löschen" verschwinden lassen, und Sie können von vorn anfangen. Nachdem Sie Ihr Kunstwerk beendet haben, klicken Sie in das OK-Feld. Die Zeichnung verschwindet aus dem Fenster, und Sie werden aufgefordert, einen Namen einzugeben. Schreiben Sie bitte pins1 und drücken Sie die RETURN-Taste. Prima! Schon ist der erste Pinsel fertig. Diese Prozedur wiederholen Sie nun, zeichnen einen großen gefüllten Kreis und nennen ihn *pins2*. Beim dritten Durchgang zeichnen Sie ein Kreuz und speichern es als *pins3* ab. So, noch einmal das Ganze mit einem Fragezeichen und als File-Name *pins?*.

Beenden Sie nun PAINTAmiga mit *Quit* aus dem Project-Menü. Sie befinden sich nun wieder im Amiga-Basic. Löschen Sie nun das Programm mit *New* aus dem Speicher. Die ungeänderte Version befindet sich ja bei Ihnen auf der Diskette. Ihrem Tatendrang steht nun nichts mehr im Wege; Sie können PAINTAmiga benutzen.

Starten Sie nun das Programm. Bis es sich meldet, habe ich noch etwas Zeit für ein paar Worte. Damit Sie auf dem Bildschirm auch das Ergebnis bekommen, das Sie sich vorstellen, sollten Sie immer mit Ruhe vorgehen. Denken Sie daran, daß Sie mit einem Basic-Programm und nicht mit einem handoptimierten Programm in Maschinensprache arbeiten. PAINTAmiga ist langsam! Lassen Sie Ihren Finger deshalb immer so lange auf der Maustaste, bis eine Reaktion eintritt – auch wenn das eine halbe Sekunde und mehr dauern sollte. Und wenn Sie die Maustaste loslassen, lassen Sie die Maus bitte einen kleinen Moment an der alten Position stehen. Wenn Sie diese Tips beherzigen, werden Sie sicher einige nette Bilder mit PAINTAmiga erstellen können.

2.8.3 PAINTAmiga-Handbuch

Nach den ersten Tests stellen Sie vielleicht fest, daß Sie nicht alle Funktionen des Programms sofort durchschauen. Deshalb an dieser Stelle ein kleines Handbuch, das die wichtigsten Befehle und ihre Wirkung erläutert.

Das erste Menü Project sollte Ihnen bereits von anderen Programmen her geläufig sein. Ich gehe auf diesen Punkt deshalb nicht weiter ein. Interessanter ist schon das Werkzeug-Menü.

Stift: Nachdem Sie den Menüpunkt *Stift* gewählt haben, können Sie freihandzeichnen. Das heißt, Sie können beliebige Kleckse und Kurven auf das Bild plazieren, indem Sie die Maustaste drücken und dann die Maus bewegen. Dies ist übrigens auch der einzige Modus des Programms, in dem die Pinsel arbeiten. Alle anderen Werkzeuge benutzen immer den kleinen, punktförmigen Pinsel.

Linie: Dieses Werkzeug dient zum Zeichnen gerader Linien. Klicken Sie den Punkt an, an welchem die Linie beginnen soll, und fahren Sie mit gedrückter Maustaste langsam zu dem Punkt, an welchem die Linie aufhören soll. Dort lassen Sie die Taste los, und schon zieht das Programm die Linie.

Rechteck: Klicken Sie in eine Ecke des gewünschten Rechtecks und bewegen Sie den Mauszeiger bei gedrückter Maustaste in die gegenüberliegende Ecke. Wenn Sie die Maustaste loslassen, erscheint das neue Rechteck im Bild. Mit dem Menüpunkt *füllen* können Sie das Rechteck später ausmalen. Wechseln Sie zuvor aber auf keinen Fall die Farbe (siehe unten). Sonst läuft Ihnen das ganze Bild mit der neuen Farbe voll. Beim Füllen brauchen Sie nur in das Rechteck hineinzuklicken. Das Füllen funktioniert gleichermaßen mit dem Kreis und der Ellipse.

Ellipse: Dieser Befehl funktioniert genauso wie das Rechteck. Wenn Sie die Maustaste loslassen, wird nur innerhalb der Rechteckbegrenzung eine Ellipse gezeichnet.

Kreis: Drücken Sie die Maustaste im Mittelpunkt des gewünschten Kreises, bewegen Sie dann die Maus, und lassen Sie die Maustaste los, wenn der Kreis den gewünschten Radius hat.

Radierer: Gehen Sie bei diesem Befehl vor, als wollten Sie ein Rechteck zeichnen. Sobald Sie die linke Maustaste loslassen, wird der Inhalt des Rechtecks gelöscht (mit der Hintergrundfarbe Weiß ausgemalt).

Fläche: Dieses Werkzeug zu beschreiben, ist nicht ganz einfach. Probieren Sie es aus: Klicken Sie an beliebiger Stelle und bewegen Sie dann die Maus bei gedrückter Taste.

Kopie: Markieren Sie auch diesmal wieder ein Rechteck. Lassen Sie dann die Maustaste los und warten Sie einen Moment. Drücken Sie jetzt erneut die Taste und halten Sie sie fest. Eine Kopie des eben markierten Bereichs klebt nun am Mauszeiger und folgt all seinen Bewegungen. Sobald Sie die Maustaste loslassen, bleibt diese Kopie im Bild liegen.

Das Menü *Farben* dient zur Wahl der Pinselfarbe. Es hat nur einen leeren Befehl (ohne Namen). Wählen Sie diesen namenlosen Befehl und sofort erscheinen in der Menü-Zeile

nebeneinander alle 32 Farben. Sie brauchen jetzt nur noch mit dem Mauszeiger auf die gewünschte Farbe zu klicken.

Wir kommen jetzt zu dem Interessantesten, dem Spezial-Menü.

zurueck: Wenn Ihnen Ihre letzte Änderung am Bild nicht gefällt, brauchen Sie nur "zurück" zu wählen und mit der Maus irgendwo auf Ihr Bild zu klicken, und schon ist diese Änderung nicht mehr zu sehen. Diese Funktion können Sie nicht zweimal hintereinander anwählen.

Pinsel: Bei der Wahl dieses Punktes öffnet sich ein neues Fenster in der oberen linken Ecke des Bildschirms. In diesem Pinsel-Fenster haben Sie mehrere Wahlmöglichkeiten. Klicken Sie auf einen der fünf Pinsel, die darin erscheinen, haben Sie einen neuen Pinsel gewählt. Das Fenster schließt sich dann wieder. Klicken Sie auf den »Text holen«, wird der Bildschirm des Pinsel-Fensters gelöscht. Nach einem zweiten Mausklick werden im Pinsel-Fenster dann alle gespeicherten Wahlpinsel aufgelistet. Anschließend müssen Sie den Namen des gewünschten Pinsels eingeben. Sobald Sie ihn richtig eingegeben haben, wird er als fünfter Pinsel eingelesen. Wenn Sie es sich anders überlegt haben, brauchen Sie nur RETURN zu drücken, und das Pinsel-Fenster wird geschlossen.

Die dritte Möglichkeit, die Ihnen im Pinsel-Fenster zur Verfügung gestellt wird, ist "fertigen". Hier erscheint nach dem zweiten Mausklick ein Rechteck, in welchem Sie Ihren neuen Pinsel konstruieren können. Sie können in diesem Rechteck arbeiten, als würden Sie das Werkzeug Stift im Hauptbild benutzen. Gefällt Ihnen Ihr neuer Pinsel nicht, brauchen Sie nur auf "löschen" zu klicken und können von vorn beginnen. Sind Sie mit Ihrem Werk zufrieden, klicken Sie OK an, und Sie werden nach dem Namen gefragt, unter welchem dieser neue Pinsel abgespeichert werden soll. Auch hier können Sie es sich noch einmal überlegen und einfach RETURN drücken. Der Pinsel wird dann nicht gespeichert. Wollen Sie das Pinsel-Fenster schließen, ohne einen neuen Pinsel abzuspeichern, können Sie jederzeit mit der Maus außerhalb des Fensters klicken, damit sich dieses wieder schließt.

Lupe ob: Suchen Sie sich diese Funktion heraus, passiert zunächst scheinbar gar nichts. Sobald Sie aber die linke Maustaste im Bild betätigen, wird der Bereich um den Mauszeiger herum durch ein Rechteck eingerahmt und ein Fenster öffnet sich. Darin wird der Inhalt des eingerahmten Bereiches 16fach vergrößert dargestellt. Wenn Sie nun innerhalb der Lupe mit der Maus klicken, erscheint dort ein kleines Rechteck und im Originalbild wird ein Punkt in der aktuellen Farbe gesetzt. Sie können damit sehr präzise Feinarbeiten ausführen. Sobald Sie fertig sind, klicken Sie einfach rechts vom Lupenfenster in das Bild, und die Lupe verschwindet wieder.

Lupe un: Diese Funktion entspricht *Lupe ob*. Das Lupen-Fenster wird jedoch in der unteren Bildschirmhälfte geöffnet. Im allgemeinen ist zu beachten, daß der Bereich unter dem Lupen-Fenster natürlich nur für die Vergrößerung markiert werden darf.

SW-HC: Dieser Befehl ist vorgesehen für den Grafik-Ausdruck, der in dieser Version von PAINTAmiga noch nicht realisiert ist.

neuFarb: Mit diesem Befehl können Sie die aktuelle Pinselfarbe ändern. Wenn Sie diesen Menü-Punkt anwählen, öffnet sich ein Fenster, das zwei Balken in der aktuellen Pinselfarbe enthält. Sie können nun den Rot-, Grün- oder Blau-Anteil dieser Farbe vergrößern, indem Sie auf das Plus über R, G oder B klicken. Ein Klick auf das Minus unter R, G oder B verringert den entsprechenden Anteil. Sie können also, wenn Sie Lust haben, sämtliche 4096 Farben, zu denen der Amiga fähig ist, sehen und einstellen.

Und nun,viel Spaß und schöne Bilder!

3
Dateien

Die Arbeit mit Daten, die sogenannte Dateiverwaltung, spielt heute im Umgang mit Firmen und Behörden eine stetig wachsende Rolle, und Computer sind dabei natürlich kaum wegzudenken; – kaum jemand legt noch größere Datenmengen in Karteikästen ab. In diesem Abschnitt sollen Sie deshalb die Grundlagen der Dateiverwaltung auf einem Computer kennenlernen. Keine Angst, Sie sollen hier keine perfekten Datenbanksysteme aufbauen. Es geht wirklich nur um die wesentlichen Grundlagen der Dateiverwaltung. Unter anderem werden Sie erfahren, wie Sie von Basic aus Bilder in einem Format auf der Diskette abspeichern können, das auch von anderen Programmen, wie Deluxe Paint und Grafikcraft, verstanden wird.

3.1 Text drucken

Vielleicht fragen Sie sich nun, was das Wort »Drucken« in der Überschrift eines Abschnitts zum Thema Dateien zu suchen hat. Die Erklärung ist einfach: Der Amiga behandelt den Drucker, als wäre er eine Datei, in die man zwar etwas schreiben, aus der man aber nie wieder etwas lesen kann.

Bevor Sie etwas in eine Datei schreiben können, müssen Sie sie normalerweise »öffnen«. Das gilt genauso für den Drucker. Einige Befehle schicken ihre Ausgaben aber immer nur zum Drucker und niemals auf den Bildschirm oder in eine Datei. Für diese Befehle muß die »Datei Drucker« auch nicht erst geöffnet werden.

Wenn Sie bei der Fehlersuche in einem Programm nicht weiterkommen, ist es oft leichter, den Fehler in einem ausgedruckten Listing als am Bildschirm zu suchen. Ein solches Listing liefert Ihnen zum Beispiel einer dieser speziellen Druckerbefehle.

LLIST ***Bereich-Bereich***

Ein Basic-Programm wird auf dem Drucker gelistet

LLIST verhält sich genauso wie der Befehl LIST (der in Kapitel 1 beschrieben wurde), nur daß das Listing zum Drucker geschickt wird. LLIST druckt das ganze Programm, LLIST-500 druckt die Zeilen bis 500 und so weiter.

Mit PRINT geben Sie alle Ausgaben auf den Bildschirm. Es gibt aber auch einen anderen Befehl, der sich genauso verhält, seine Ausgaben aber druckt und nicht anzeigt.

LPRINT ***Liste von Ausdrücken***

LPRINT USING v$;Liste von Ausdrücken

gibt die Werte der Liste von Ausdrücken auf dem Drucker aus

Ein Drucker ist – wie der Bildschirm – nicht beliebig breit. Wird die Liste, die Sie mit einer LPRINT-Anweisung ausdrucken lassen, breiter als 80 Zeichen, geht der Drucker deshalb automatisch in eine neue Zeile. Die Breite von 80 Zeichen ist aber nur eine Voreinstellung, die Sie jederzeit ändern können.

WIDTH ***LPRINT Breite, Druckzone***

WIDTH #Dateieinr, Breite, Druckzone

WIDTH Gerät, Breite, Druckzone

legt fest, nach welcher Breite beim Drucker, einer Datei oder einem bestimmten Gerät der automatische Zeilenvorschub erfolgt

Die letzte Angabe, Druckzone, einer WIDTH-Anweisung legt die Breite der Druckzonen fest. Druckzonen verhalten sich ähnlich wie Tabulatoren auf einer Schreibmaschine. Wenn Sie die Elemente der Liste hinter Print oder LPRINT mit Kommata abtrennen, springt Basic jedesmal zum Ende der jeweiligen Druckzone, bevor es den nächsten Wert ausgibt. Listen können so ordentlicher gestaltet werden.

Sie können die Wirkung der WIDTH-Anweisung auch am Bildschirm überprüfen. Geben Sie einmal im Direktmodus WIDTH 20 ein und dann PRINT "12345678901234567890 12345". Sie werden feststellen, daß die Ausgabe dieses Textes bereits bei der 20sten Stelle abbricht.

Das folgende Beispielprogramm zeigt eine sehr einfache Anwendung des LPRINT-Befehls. Es listet den kompletten Zeichensatz, den Ihr Drucker eingebaut hat, auf.

```
REM Zeichen
REM P.3.1-1
LPRINT "Zeichensatz"
LPRINT:LPRINT
FOR n=0 TO 128 STEP 128
  GOSUB druck
NEXT
LPRINT
END

druck:
FOR i = 33+n TO 127+n
  LPRINT CHR$(i);
  IF LPOS(x)>20 THEN LPRINT CHR$(13)
NEXT
RETURN
```

Dieses Programm druckt die Zeichen mit den ASCII-Codes von 33 bis 127 und von 161 bis 255 aus. Dabei wurde die Zeilenbreite auf 20 Zeichen begrenzt – aber nicht mit WIDTH, sondern durch einen IF-Befehl, der prüft, ob bereits die 20ste Stelle in der Zeile erreicht wurde. Ist dies der Fall, wird ein Wagenrücklauf-Code CHR$(13) ausgegeben.

v=LPOS(n)
liefert die Position des zuletzt gedruckten Zeichens

Der Parameter n ist ein Wert ohne Bedeutung, der aber trotzdem nicht fortfallen darf.

Nachdem Sie nun wissen, wie Texte an den Drucker gesendet werden, kommt der interessantere Teil, der Grafik-Druck.

3.2 Hardcopy

Was Sie mit Ihrem Drucker anstellen können, hängt natürlich in erster Linie vom Typ ab. Voraussetzung für den Grafik-Druck ist ein Matrix-Drucker, ein Typenrad-Drucker kann nur zur Textausgabe verwendet werden.

Die nachfolgend aufgeführten Programme und Anweisungen sind mit dem Drucker EPSON EX-800 getestet worden. Der EPSON EX-800 ist ein 9-Nadel-Matrixdrucker, der mit einem Zusatz-Farbkit auch farbig drucken kann. (Er ist auch zum EPSON FX-80 kompatibel, solange man nur in einer Farbe druckt.) Blättern Sie bitte nicht gleich weiter, wenn Sie einen anderen Drucker besitzen. Moderne Nadeldrucker sind sich alle so ähnlich, daß programmtechnische Unterschiede nur in Details auftreten. Wenn Sie die Struktur im folgenden beschriebenen Programm verstanden haben, können Sie diese

deshalb meist problemlos an andere Drucker anpassen. Hierzu brauchen Sie meist wenig mehr als ein Drucker-Handbuch und etwas Zeit.

Beim Grafikdruck gibt es für Sie als Basic-Programmierer nur zwei wichtige Neuerungen: die Steuer-Codes, die den Drucker auf Grafikdruck umschalten und die Ansteuerung der einzelnen Druckernadeln. Die Steuer-Codes werden in den folgenden Programmbeispielen ausgiebig kommentiert; dadurch sollen Sie diese Programme leicht an andere Drucker anpassen können.

Doch zuerst werden wir uns der Nadelsteuerung zuwenden. Im Druckkopf des Druckers liegen die Nadeln oder Stifte, welche den eigentlichen Druck besorgen, senkrecht übereinander. Der Ausdruck der Grafik erfolgt aber wie im Text-Modus Zeile für Zeile. Im Grafik-Modus werden für den Druckvorgang die oberen acht Nadeln des Druckkopfes eingesetzt. Das bedeutet, daß man dem Drucker für jede Spalte einer solchen Grafikzeile sagen muß, welche der acht Nadeln das Farbband bearbeiten sollen und welche nicht. Diese Angabe muß man dem Drucker üblicherweise in Form einer Zahl zwischen 0 und 255 machen. Diese ominöse Zahl 255, der man beim Programmieren öfters begegnen wird, bedeutet 8 Bit oder ein Byte.

Zur Ermittlung der Zahl, die an den Drucker geschickt werden muß, wird jeder der oberen acht Nadeln des Druckers eine Zahl zugeordnet. Die Nadeln erhalten die folgenden Werte:

o 2^7 = 128
o 2^6 = 64
o 2^5 = 32
o 2^4 = 16
o 2^3 = 8
o 2^2 = 4
o 2^1 = 2
o 2^0 = 1

Um eine bestimmte Kombination von Nadeln drucken zu lassen und die anderen zurückzuhalten, müssen Sie die Zahlen neben den Nadeln, die drucken sollen, addieren. Wenn Sie also keine Nadel einer Reihe ansprechen wollen, geben Sie den Wert 0, und wenn alle Nadeln drucken sollen, zählen Sie alle Nadelwerte zusammen und erhalten 255. Um die Sache noch etwas zu verdeutlichen, hier noch ein paar Beispiele:

+128	o	+128	o
o	+64	+64	o
o	o	o	+32
+16	o	o	+16
o	+8	o	+8
o	o	+4	o
+2	o	o	+2
o	+1	+ 1	o
Summe:146	Summe:73	Summe:197	Summe:58

So, jetzt sollen Sie Ihr frisch erworbenes Wissen gleich an einem Beispiel ausprobieren. (ACHTUNG, dieses Beispiel funktioniert nur, wenn der Drucker, den Sie verwenden, am parallelen Port des Amiga angeschlossen ist!)

```
REM Amiga-K
REM P.3.2-1
REM Grafikdruck in einfacher Dichte=K
OPEN "PAR:" FOR OUTPUT AS#2
PRINT#2,CHR$(27)+"A"+CHR$(8);
FOR anzahl = 1 TO 10
  PRINT#2,CHR$(27)+"K"+CHR$(17)+CHR$(0);
  FOR spalte = 1 TO 17
       READ W
       PRINT #2,CHR$(W);
  NEXT spalte
  RESTORE
NEXT anzahl
PRINT#2,CHR$(27)+"@"
CLOSE#2
END
DATA 16,24,12,22,27,14,14,27,54,108
DATA 216,48,96,192,128,0,0
```

In der ersten Zeile eröffnen wir eine Datei mit der Nummer 2. Die Datei erhält den Namen "PAR:", das bedeutet, daß alle Daten, die wir in diese Datei schicken, zur parallelen Schnittstelle und damit zu dem Drucker, der dort angeschlossen ist, gehen. Laut Basic-Handbuch soll man hierfür eigentlich "LPT1:" verwenden. Das funktioniert auch für den Ausdruck von Texten. Für den Grafikdruck muß man aber "PAR:" verwenden.

In der zweiten Zeile wird mit PRINT#2 ein Steuercode zum Drucker geschickt, der den Zeilenabstand so verändert, daß die Grafikzeilen später dicht an dicht, und nicht mit einem weißen Streifen dazwischen, ausgedruckt werden.

Die darauffolgende FOR-Schleife schickt zehnmal hintereinander dieselbe Grafikzeile zum Drucker. Zuvor werden aber noch eine Reihe von Steuercodes zum Drucker geschickt, die ihm sagen, daß dahinter jeweils 17 Byte (17 Spalten) mit Grafikdaten folgen.

In der inneren FOR-Schleife werden die 17 Zahlen, die an den Drucker geschickt werden und die steuern, welche Nadeln in jeder Spalte gedruckt werden, mit dem Befehl READ aus den DATA-Befehlen am Ende des Programms gelesen.

READ Variable,*Variable,...*
liest Daten aus DATA-Zeilen und weist sie Variablen zu

Die Werte, die READ liest, müssen mit DATA-Befehlen bereitgelegt werden.

DATA Konstante, ***Konstante,...***

legt Konstanten im Speicher ab, die das Programm mit READ lesen kann

Zum besseren Verständnis zwischendurch ein kleines Beispiel:

```
REM farbe
REM P.3.2-2
LPRINT "neue Grundfarben"
LPRINT "================"
LPRINT
n=-1
WHILE n
   READ n,o,p,q,f$
   PALETTE n,o,p,q
   PRINT "Farbe"n;"wurde "f$
WEND
DATA 1,1,1,.13,gelb, 2,.6,.6,.6,grau
DATA 3,.33,.87,0,gruen, 0,.47,.87,1,hellblau
```

Das Programm liest Daten für die Farbanteile von vier der Farben in der Palette aus DATA-Befehlen ein und verwendet sie dann in der PALETTE-Anweisung. Beachten Sie bitte die Reihenfolge von numerischen Werten und Strings, die bei den DATAs und der READ-Anweisung übereinstimmen muß.

Nun aber zurück zum Druckprogramm. In dessen innerer Schleife werden also die DATA-Werte eingelesen und spaltenweise ausgedruckt. Jedesmal, wenn eine Zeile abgeschlossen ist, wird der Befehl RESTORE aufgerufen.

RESTORE ***Marke***

der Lesezeiger für READ wird zurückgesetzt

READ beginnt normalerweise mit der ersten DATA-Anweisung, Daten zu lesen, und geht die DATA-Anweisungen dann nacheinander durch. Mit RESTORE kann man dafür sorgen, daß die nächste READ-Anweisung wieder bei der ersten DATA-Anweisung oder bei der DATA-Anweisung, die hinter Marke steht, beginnt,

Der letzte Druckbefehl in diesem Programm versetzt den Drucker wieder in den »Normalzustand« und leert den Druckerpuffer. Zu guter Letzt wird die Datei geschlossen und das Programm ist fertig.

Wenn alles glattgeht, wird dieses Programm zehnmal das Amiga-Zeichen drucken. Wenn Sie Spaß daran gefunden haben, können Sie ruhig noch ein paar Bildchen kreieren. Die Zeichnungen können auch mehrere Spalten hoch sein. Das Programm muß dann nur um eine weitere Schleife ergänzt werden.

Nach diesen Programmen sind Sie so gut vorbereitet, daß Sie sich gleich auf das Grafik-Hardcopy-Programm stürzen können. Wegen der geringen Geschwindigkeit des Grafikausdruckes von Basic aus wird dieses Programm so einfach wie möglich gehalten. Im Klartext heißt das, daß ohne Schattierungen gedruckt wird; jeder Punkt ist entweder schwarz oder weiß. Für diese simpelste Lösung benötigt das Programm dann schon etwa zehn Minuten, wenn eine Grafik ausgedruckt werden soll, die 310 Punkte breit und 180 Punkte hoch ist. Das ist natürlich nicht gerade berauschend. Trotzdem ist das Programm sinnvoll, da es so universell geschrieben ist, daß man es ohne große Änderung in vielen Basic-Programmen einsetzen kann, die Grafiken ausdrucken müssen.

Die erste Frage, die sich beim Ausdruck eines Fensters stellt, ist die Fenstergröße. Diese kann mit einer speziell dafür gedachten Funktion ermittelt werden.

v=WINDOW(n)
liefert je nach dem Wert von n (0-8) verschiedene Information über das aktuelle Fenster

Format	übergebene Information	Beispiel
WINDOW(0)	Kennung des gewählten Ausgabefensters, welches den Titel nicht in Geisterschrift angezeigt	Eingabe direkt: PRINT WINDOW(0) Ergebnis : 1 Fenster 1 ist das AMIGA Ausgabefenster
WINDOW(1)	Kennung des aktuellen Ausgabefensters, in welchem Print- und grafische Befehle wirken	Eingabe direkt: PRINT WINDOW(1) Ergebnis : 1 siehe 1. Beispiel
WINDOW(2)	Breite des aktuellen Ausgabefensters in Bildpunkten	Programm: WINDOW 2,,,1 While 1 Locate 1,1 Print "Breite= "window(2); Print " Hoehe= "WINDOW(3) WEND Das Ergebnis erhält man, wenn mit der Maus die Fenstergröße verändert wird
WINDOW(3)	Höhe des aktuellen Ausgabefenster in Bildpunkten	
WINDOW(4)	X-Koordinate im aktuellen Ausgabefenster, bei der das nächste Zeichen ausgegeben wird, in Bildpunkten	Programm: FOR i = 1 TO 5 Print i;WINDOW(4);WINDOW(5); NEXT Ergebnis: 1 24 6 2 104 6 3 192 6... oder wenn letzter ; fehlt: 1 24 6 2 24 14 3 24 22
WINDOW(5)	Y-Koordinate im aktuellen Ausgabefenster, bei der das nächste Zeichen ausgegeben wird, in Bildpunkten	
WINDOW(6)	Maximalzahl der für das aktuelle Ausgabefenster erlaubten Farben	Eingabe direkt: Print WINDOW(6) Ergebnis:3 (ohne Hintergrundfarbe)
WINDOW(7)	Ein Adressanzeiger auf den INTUITION WINDOW-Datensatz für das aktuelle Ausgabefenster	Programm: Beispiel WINDOW 2,,,8 WHILE 1 IF WINDOW(7)=0 THEN BEEP:END Wenn das Fenster geschlossen wird zeigt WINDOW(7) den Wert 0
WINDOW(8)	Adresszeiger auf den RASTPORT-Datensatz für aktuelles Ausgabefenster	Ein Beispiel finden sie im nächsten Kapitel

Bild 3.1: *Erläuterungen für die Window-Befehle*

```
REM SW-Hardcopy
REM P.3.2-4
breite=WINDOW(2)
br=INT(breite/256)
b=breite-(br*256)
hoehe=WINDOW(3)
h=INT(hoehe/8)+1
OPEN"PAR:"FOR OUTPUT AS #2 'Datei für Drucker öffnen
PRINT#2,CHR$(27)+"A"+CHR$(8); '8 Druckzeichen hoch
GOSUB nadel
PRINT#2,CHR$(27)+"@" 'Drucker auf Ausgangswerte setzen
CLOSE #2
END
nadel:
y=0
FOR gs=0 TO h ' Grafikzeilen
  IF breite>480 THEN
       PRINT #2, CHR$(27) + "L" + CHR$(b) + CHR$(br);
  ELSE
       PRINT#2,CHR$(27)+"K"+CHR$(b)+CHR$(br);
  END IF
  FOR x=0 TO breite 'Spalten
       FOR z=7 TO 0 STEP-1 '8 Zeilen
  IF POINT(x,y1+y) THEN einzelwert=2^z ELSE einzelwert=0
            wert=wert+einzelwert
            y1=y1+1
            W=wert
       NEXT z
       PRINT#2,CHR$(W);
       wert=0:y1=0
  NEXT x
  PRINT#2,CHR$(10)
  y=y+8
NEXT gs
RETURN
```

Wie Sie sehen, ist das Programm recht kurz und einfach. Am Anfang werden zunächst einige Zahlen festgestellt oder berechnet, die für den Ausdruck benötigt werden. Dann wird die Druckerdatei geöffnet, die Subroutine *nadel* aufgerufen und die Druckerdatei wieder geschlossen. Die Hauptarbeit leistet *nadel.* Diese besteht aus drei großen ineinander geschachtelten FOR-Schleifen. Die äußere wird einmal für jede Druckzeile, die nächste einmal für jede Druckspalte und die innere einmal für jeden Punkt der Spalte durchlaufen. Je nachdem, ob die Breite der zu druckenden Grafik größer ist als 480 Punkte oder nicht, wird der Drucker zu Beginn der äußeren Schleife in einen Modus geschaltet, in dem er entweder maximal 480 oder 960 Punkte auf der vollen Breite des Papiers ausgeben kann.

Interessant ist vor allem die innere FOR-Schleife. Nach jedem Durchlauf durch diese Schleife (einmal für jede Spalte der Druckzeile) werden acht übereinanderliegende Punkte als eine Zahl (ein ASCII-Code) zum Drucker geschickt. Die innere Schleife ermittelt, welche Zahl das sein muß. Dazu wird diese Zahl (*wert*) zunächst auf Null gesetzt. Dann wird für jeden der acht Punkte der Spalte mit der Funktion POINT festgestellt, ob der entsprechende Punkt am Bildschirm gesetzt ist oder nicht. Dann wird, wenn der Punkt gesetzt ist, eine Zahl auf die Variable *wert* addiert, die der Höhe dieses Punktes in der Druckspalte entspricht. Für den obersten Punkt werden 128 addiert, für den siebten 64 und so weiter bis zum untersten Punkt der Druckzeile, der der 1 entspricht.

Die innere FOR-Schleife wird dann beendet und die so berechnete Zahl als ein Zeichen zum Drucker geschickt. Das geschieht für jede Spalte einer Grafikzeile einmal, dann ist auch die mittlere FOR-Schleife einmal durchlaufen. An deren Ende wird auch der Code CHR$(10) zum Drucker gesendet, der für den Zeilenvorschub sorgt. Die äußere FOR-Schleife fährt dann in der nächsten Druckzeile (acht Bildschirmpunkte tiefer) fort.

Beim Eingeben des Programmes achten Sie bitte darauf, daß Sie bei den PRINT-Befehlen das Semikolon am Ende nicht vergessen, da sonst jede Grafik-Spalte in einer neuen Zeile ausgedruckt wird!

Nachdem das Ausdrucken in einer Farbe so schön geklappt hat, war an dieser Stelle eigentlich eine Hardcopy-Funktion für farbige Grafiken vorgesehen. Das Programm war fertig und arbeitete auch korrekt. Sogar der Text für das entsprechende Kapitel war bereits geschrieben. Das Programm hatte nur einen kleinen Haken: für einen Ausdruck benötigte es über eine Stunde. Das wollte ich Ihnen doch nicht antun. Sie müssen sich deshalb, wenn Sie farbige Bilder drucken wollen, eines professionellen Programmes bedienen. Wie man das Zeichenprogramm GrafiCraft (und andere Programme) dazu bekommt, Bilder drucken zu lassen, die mit einem Basic-Programm erstellt wurden, erfahren Sie im nächsten Abschnitt.

3.3 IFF, Chunks und solche Sachen

Wenn Sie nach dieser Überschrift vermuten, daß Sie versehentlich ein chinesisches Kochbuch aufgeschlagen haben, darf ich Sie beruhigen. Sie lesen ein Buch über Amiga-Basic. IFF ist eine Abkürzung für INTERCHANGE FILE FORMAT und bedeutet auf deutsch etwa soviel wie »austauschbares Datei-Format«. IFF ist inzwischen zum Standard von Grafiken und Musik auf dem Amiga geworden. Wir verdanken diesen Standard der Software-Firma Electronic Arts aus den USA. Diese Firma hat beim Erscheinen des Amiga ein System entwickelt, Daten jeder Art so abzuspeichern, daß sie von jedem Programm verarbeitet werden können. Fast alle Programme auf dem Amiga arbeiten heute mit diesem Format – zumindest wenn sie Grafiken abspeichern und einlesen.

Dieser Standard für den Datenaustausch definiert je nach Art der zu speichernden Daten verschiedene Datei-Typen. Durch spezielle Code-Wörter am Anfang der jeweiligen Datei werden die einzelnen Dateien als einem bestimmten Typ zugehörig kenntlich gemacht.

Der Code FTXT steht für den formatierten Text einer Textdatei, 8SVX für 8 Bit Sampled Voice einer Datei für digitalisierte Klänge, SMUS für Simple Musical Score einer Musik-Datei und ILBM für Interleaved Bit Map (eine Grafik-Datei), wobei diese Zusammenfassung natürlich keinen Anspruch auf Vollständigkeit erhebt. In diesem Kapitel werden wir uns nur mit den Grafik-Dateien vom Typ ILBM beschäftigen. Es wird also nur der Umfang des IFF-Standards besprochen, der für die Programmierung in Basic besonders wichtig ist.

3.3.1 Vom Umgang mit Speicherstellen

Ein wichtiger Gesichtspunkt für das Projekt IFF-Dateien ist der richtige Umgang mit Speicherstellen, mit den fast schon legendären Befehlen PEEK und POKE. Im Gegensatz zu anderen Computern kommt man auf dem Amiga zwar fast völlig ohne PEEK und POKE aus. Für diesen Sonderfall kommen Sie aber kaum umhin, diese recht gefährlichen Befehle zu verwenden.

Dazu eine Warnung vorweg: Gerade der Befehl POKE ist sehr gefährlich. Sie können damit eine beliebige Stelle im Speicher Ihres Amiga ändern, egal was dort bislang gespeichert ist. Liegt dort aber zum Beispiel ein Teil des Basic-Interpreters, kann dies katastrophale Effekte ergeben. Jedes Programm, das ein POKE enthält, sollten Sie deshalb extrem sorgfältig eingeben und überprüfen, bevor Sie es starten.

Nun aber zur Definition von PEEK und POKE:

v=PEEK(Adresse)
liefert den Inhalt des Speichers an der genannten Adresse als ganze Zahl im Bereich zwischen 0 und 255 (ein Byte)

POKE Adresse,Wert
ändert den Inhalt des Speichers an der genannten Adresse, indem es dort die Zahl Wert als 8-Bit-Binärwert (ein Byte) ablegt

Das folgende Beispiel demonstriert diese beiden neuen Befehle:

```
REM reinraus
'P.3.3-1
PRINT "Speicherstellen"
PRINT :PRINT
FOR i=10000 TO 10015
   sp=PEEK(i)
   PRINT "SPEICHERSTELLE: "i,"INHALT: "sp
   POKE i,PEEK(i)
NEXT
```

Das Programm liest die Speicherstellen 10000 bis 10015 und zeigt deren Inhalt auf dem Bildschirm. Mit der Zeile POKE i,PEEK(i) wird der Wert an einer Stelle des Speichers gelesen und wieder zurückgeschrieben. Diese Zeile hat natürlich keinerlei Wirkung, sondern soll nur den POKE Befehl demonstrieren.

PEEK und POKE haben jedoch einen schwerwiegenden Nachteil. Sie lesen und schreiben nur Zahlen zwischen 0 und 255 (ein Byte). Im folgenden werden aber oft größere Zahlen benötigt. Wichtiger sind deshalb die beiden Befehle, die mit einer ganzen Zahl arbeiten:

v=PEEKW(Adresse)
liefert den Inhalt des Speichers an der genannten Adresse als kurze Ganzzahl (2 Byte)

Adresse muß dabei unbedingt eine gerade Zahl sein.

POKEW Adresse,Wert
schreibt den Wert als 16-Bit-Binärwert (2 Byte) in den Speicher an die genannte Adresse

POKEW ist das Gegenstück zu PEEKW. Die Adresse muß auch hier eine gerade Zahl sein.

Amiga-Basic kennt aber noch mehr Versionen von PEEK und POKE:

v=PEEKL(Adresse)
liefert den Inhalt des Speichers an der genannten Adresse als lange Ganzzahl (4 Byte).

POKEL Adresse,Wert
schreibt den Wert als 32-Bit-Binärwert (4 Byte) in den Speicher an die genannte Adresse

Die Anweisung POKEL ist wieder das Pendant zu der Funktion PEEKL. Die Adresse muß bei diesen beiden Befehlen ebenfalls eine gerade Zahl sein.

Das folgende Programm liest die Inhalte von Variablen verschiedener Typen direkt mit PEEK-Befehlen:

```
REM VarSpeicher
'P.3.3-2
PRINT "Speicherstellen von Variablen"
```

```
PRINT :PRINT
PRINT "Variable","Typ",,"Speicher","Speicherinhalt"
n%=3456:n&=98765&
PRINT
PRINT "n%= "n%,"kurze Ganzzahl",VARPTR(n%),PEEKW(VARPTR(n%))
PRINT
PRINT "n&= "n&,"lange Ganzzahl",VARPTR(n&),PEEKL(VARPTR(n&))
```

Zunächst werden in diesem Programm die beiden Variablen n% und n& definiert. Diese Variablen werden vom Interpreter an einer bestimmten Stelle im Speicher abgelegt. Mit der Funktion VARPTR stellt das Programm fest, an welcher Stelle.

v=VARPTR(Variable)
liefert die Adresse, an der der Wert der genannten Variablen im Speicher abgelegt wird

Bei den letzten beiden Beispielen wurden die neuen Befehle nur dazu benutzt, den Inhalt von Speicherstellen zu zeigen. Natürlich können Sie mit Ihrem neuen Wissen aber auch diese Speicherstellen manipulieren. Das folgende Beispiel zeigt nebenbei auch noch, wie VARPTR auf die Elemente eines Feldes angewendet wird.

```
REM FeldSpeicher
'P.3.3-3
CIRCLE (50,25),50,3,,,.51
PAINT (50,25),3
DIM bild%(720)
GET (0,0)-(100,50),bild%
FOR i = 0 TO 300
sp=VARPTR(bild%(3+i))
POKEW sp,0
NEXT
CLS:PUT (0,0),bild%:ERASE bild%
DIM bild&(360)
GET (0,0)-(100,50),bild&
FOR i = 350 TO 250 STEP-1
sp=VARPTR(bild&(i))
POKEL sp,0
NEXT
CLS:PUT (0,0),bild&
```

Dieses Programm zeichnet in der linken oberen Ecke des Bildschirmes einen roten Kreis. Mit GET wird dieser Bildschirmbereich dann in die Feld-Variable Bild% geholt. In der ersten FOR-Schleife werden mit VARPTR die Adressen der ersten 300 Feld-Elemente ermittelt und diese mit POKEW gelöscht (auf Null gesetzt). Dadurch wird die Farbe der entsprechenden Punkte geändert. Wenn anschließend Bild% mit PUT wieder auf den Bildschirm gebracht wird, ist ein Teil des Kreises schwarz geworden. Der ganze Vorgang

wird dann ein zweites Mal mit langen Ganzzahlen und der Feld-Variablen Bild& wiederholt. Diesmal werden die Bits der oberen Bitmap teilweise überschrieben. Das Ergebnis ist ein Kreis, der rot, schwarz und blau geworden ist. Sollte Sie dieses Thema interessieren, so finden Sie im Kapitel 5.5 weitere Informationen dazu.

Im letzten Beispiel zu den Befehlen zur Speichermanipulation werden Strings behandelt:

```
REM StringSpeicher
'P.3.3-4
a$="BMIGA"
PRINT "Name der Zeichenkette: ";
PRINT a$
v=SADD(a$)
POKE v,65
PRINT "erste Speicherstelle geändert: ";
PRINT a$
REM Achten Sie nach dem Programm-Ablauf
REM auf den String a$ in der dritten Zeile!"
END
```

Zuerst wird der String-Variablen a$ das Wort BMIGA zugewiesen und ihr Inhalt auf dem Bildschirm gezeigt. Mit einer neuen Funktion wird dann die Adresse des Strings ermittelt und das erste Zeichen darin geändert.

v=SADD(Zeichenkettenausdruck)
liefert die Adresse, an der das erste Zeichen eines Strings abgespeichert ist

In diese Speicherstelle wird mit POKE der ASCII-Code des Buchstabens A geschrieben. Nun wird die Variable a$ wieder ausgegeben und tatsächlich ist nun das Wort Amiga zu lesen.

Damit ist der Ausflug in die Welt der Speicherstellen zunächst einmal beendet, und wir wenden uns wieder dem eigentlichen Thema dieses Abschnitts zu.

3.3.2 Eine IFF-Grafik einlesen

Eine IFF-Datei besteht aus einem Grundbaustein, genannt FORM und einer Reihe von Einzelbausteinen, welche Chunks genannt werden. Die Anzahl der Chunks ist nicht fest vorgegeben (variabel). Ebenso haben die Chunks keine vorgegebene Länge. Damit das jeweilige Programm weiß, wieviele Daten in dem Chunk enthalten sind, sind zu Beginn des Chunks sein Name und die Länge der darauf noch folgenden Daten verzeichnet. Anhand dieser Informationen kann das Programm in eine entsprechende Routine verzweigen und dort die Daten verarbeiten. Die folgende Abbildung zeigt das Grundschema einer IFF-Datei.

Grundbaustein:	4 Byte	String "FORM"	
	4 Byte	numerisch	FORM-Länge
	4 Byte	String	FORM-Kennung
1.Chunk	4 Byte	String	Chunk-Name
	4 Byte	numerisch	Chunk-Länge "X"
	X Byte		Chunk-Inhalt
	(eventuell ein Füll-Byte)		
2.Chunk	4 Byte	String	Chunk-Name
	4 Byte	numerisch	Chunk-Länge "X"
	X Byte		Chunk-Inhalt
	(eventuell ein Füll-Byte)		
3.Chunk	usw.		

Tabelle 3.2: *Der Aufbau einer IFF-Datei*

Das Füll-Byte hinter dem Chunk-Inhalt ist dann erforderlich, wenn der Chunk-Inhalt aus einer ungeraden Zahl von Bytes besteht. Es sorgt dafür, daß jeder neue Chunk bei einer geraden Adresse beginnt.

Das war jetzt eine ganze Menge Theorie und sie wird Ihnen nicht viel sagen. Diese graue Theorie soll nun mit einem kleinen Beispiel etwas erhellt werden. Sie kennen sicher das Bild der Mona Lisa auf der Diskette des Zeichenprogrammes GrafiCraft. Sie werden nun ein Programm erstellen, mit dem Sie sich die Chunks dieses Bildes genauer betrachten können.

Vorher möchte ich aber noch einmal kurz zusammenfassen, was Sie dazu über die Funktionen wissen müssen, die benötigt werden, um Variablen verschiedener Typen einzulesen, beziehungsweise in eine Datei zu schreiben.

Ausdruck n >>> speichern	>>> lesen	
Zahl <256	PRINT #1,CHR$(n);	n=ASC(INPUT$(1,1)
kurze Ganzzahl	PRINT #1,MKI$(n%);	n=CVI(INPUT$(2,1)
lange Ganzzahl	PRINT #1,MKL$(n&);	n=CVL(INPUT$(4,1)
einfachgenaue	PRINT #1,MKS$(n!);	n=CVS(INPUT$(4,1)
doppeltgenaue	PRINT #1,MKD$(a#);	n=CVD(INPUT$(8,1)
n$="abc"	PRINT #1,n$;	n$=INPUT$(3,1)

Tabelle 3.3: *Nötige Konvertierungen bei der Ein- und Ausgabe*

Tippen Sie nun bitte das folgende Programm ein und speichern Sie es unter dem Namen IFFlesen ab. Wenn Sie keinen Drucker angeschlossen haben, ersetzen Sie bitte jedes LPRINT durch PRINT.

```
REM IFFlesen
REM P.3.3-1
CLEAR ,45000&

formlesen:
CLS:INPUT "Bitte Filenamen: ",filename$
IF filename$="" THEN formlesen
OPEN filename$ FOR INPUT AS 1
Namen$=INPUT$(4,1) :LPRINT "Grundbaustein: "Namen$
IF Namen$="FORM" THEN weiter
PRINT "KEINE IFF-DATEI"
END

weiter:
Formlaenge=CVL(INPUT$(4,1)) :LPRINT "Form-Laenge: "
** Formlaenge
FormKennung$=INPUT$(4,1) :LPRINT "Form-Kennung: "
** FormKennung$
LPRINT

chunklesen:
WHILE NOT EOF(1)
   chunkname$=INPUT$(4,1) :LPRINT "Chunk-Name: "chunkname$
   chunklaenge=CVL(INPUT$(4,1)):LPRINT "Chunk-Laenge: "
** chunklaenge
   GOSUB byteslesen
WEND
CLOSE#1:END

byteslesen:
IF chunklaenge/2-FIX(chunklaenge/2)>0 THEN chunklaenge=
** chunklaenge+1
LPRINT "Chunk-Inhalt: "
FOR i = 1 TO chunklaenge
   LPRINT ASC(INPUT$(1,1));
   IF i>100 THEN
   rest$=INPUT$(20000,1)
   rest$=INPUT$(chunklaenge-20000-i,1)
   i=chunklaenge
   END IF
NEXT
LPRINT
RETURN
```

In diesem Programm werden die Daten genau in der Reihenfolge eingelesen, wie es oben in dem Schema für IFF-Dateien gezeigt wurde. Zuerst wird eine Datei mit dem eingegebenen Namen geöffnet. Dann werden die ersten vier Buchstaben eingelesen. Sind diese nicht gleich "FORM", ist es keine IFF-Datei und das Programm wird beendet. Nach dem Einlesen des Grundbausteins liest eine WHILE-Schleife die einzelnen Chunks ein. Nachdem die Chunklänge festgestellt wurde, verzweigt das Programm in die Routine

byteslesen. In diesem Programmteil erhöhen wir die Länge zunächst um ein Füll-Byte, wenn sie ungerade ist. Hierzu wird unter anderem die Funktion FIX verwendet.

v=FIX(x)
liefert eine Zahl, die sich ergibt, wenn man die Nachkommastellen abschneidet (ohne Rundung)

Nachdem die genaue Länge feststeht, werden die Chunk-Daten in einer FOR-Schleife eingelesen und die ersten 100 Byte davon (als Zahlen) ausgegeben. Dann werden die restlichen Bytes in zwei »Häppchen« eingelesen und ignoriert. Zur Ausgabe eines Bytes als Zahl wird die Funktion ASC verwendet.

v=ASC(x$)
liefert den ASCII-Code (zwischen 0 und 255) des ersten Zeichens (Bytes) von x$

Bevor Sie das Programm starten, sollten Sie das Directory (Dateiverzeichnis) ändern. Sie können dann leichter, aber mit Vorsicht, auch noch andere Bilder als IFF-Dateien untersuchen. Geben Sie bitte im Direktmodus ein:

```
CHDIR "GrafiCraft:"
```

Wenn Sie Ihrer GrafiCraft-Diskette einen anderen Namen gegeben haben, müssen Sie "GrafiCraft" selbstverständlich durch diesen anderen Namen ersetzen.

Schalten Sie nun Ihren Drucker an und starten Sie das Programm IFFlesen. Geben Sie den Namen MonaLisa ein, wenn Sie nach dem »Filenamen« gefragt werden, und harren Sie der Dinge, die da kommen. Wenn Ihr Programm keinen Tippfehler hat, erhalten Sie folgenden Ausdruck:

```
Grundbaustein: FORM
Form-Laenge: 40156
Form-Kennung: ILBM
Chunk-Name: BMHD
Chunk-Laenge: 20
Chunk-Inhalt: 1 64 0 200 0 0 0 0 5 0 0 0 0 0 10 11 1 64
** 0 200
Chunk-Name: CMAP
Chunk-Laenge: 96
Chunk-Inhalt: 240 240 240 240 160 64 64 48 48 80 48 ...
... bis 112 48 128 96 64 80 80 48 48 48 48 0 0 0
Chunk-Name: BODY
Chunk-Laenge: 40000
Chunk-Inhalt: 0 16 0 0 128 0 0 0 0 10 0 0 0 0 0 0 0 ...
Chunk-Name: CAMG
```

```
Chunk-Laenge: 4
Chunk-Inhalt: 0 0 64 0
```

Die genaue Struktur einer IFF-Datei können Sie so schon feststellen. Jetzt brauchen wir nur noch herauszufinden, welche Informationen in welchem Chunk verborgen sind. Übrigens, haben Sie sich auch den Aufbau anderer Bilder angesehen? Im wesentlichen sind alle Bilder gleich aufgebaut. Bei den Bildern Skier und NightFlight findet sich ein weiterer Chunk mit dem Namen CCRT. Dieser ist für die Farb-Rotation in diesen Bildern verantwortlich und soll uns nicht weiter interessieren.

Im Grundbaustein FORM einer IFF-Datei steht als Formlänge immer die Gesamtlänge der jeweiligen Datei von 40156 Byte und als Formkennung ILBM, für »InterLeaved BitMap«. Daran kann man erkennen, daß es sich um eine Grafik handelt.

Der erste Chunk ist der BitMap Header BMHD. Wie Sie selbst nachzählen können, ist er 20 Byte lang. Da die Bytes durch das Programm einzeln ausgelesen wurden, müssen manche davon wieder zu Worten (Zahlen aus zwei Byte) zusammengefaßt werden.

Wort	2 Byte	320	Bildbreite
Wort	2 Byte	200	Bildhöhe
Wort	2 Byte	0	X-Position Bildschirm
Wort	2 Byte	0	Y-Position Bildschirm
Byte		5	Anzahl der Bitplanes (für 32 Farben)
Byte		0	verwendete Maske = nicht belegt
Byte		0	gepackte Grafik = nicht belegt
Byte		0	für zukünft. Erw. = nicht belegt
Wort	2 Byte	0	für Maske = nicht belegt
Byte		10	Pixel-Breite Verhältnis = 10:11
Byte		11	Pixel-Höhe bei 320x200 Screen
Wort	2 Byte	320	Bildschirmbreite
Wort	2 Byte	200	Bildschirmhöhe

Tabelle 3.4: *Aufbau eines BMHD-Chunks*

Die vorstehende Liste gibt wieder, wie dieser Chunk wirklich aussieht, wenn die Felder, die aus mehr als einem Byte bestehen, berechnet werden.

Damit haben Sie alle Informationen über die Grafik beisammen. Die Entwicklung eines Programms zum Einlesen von IFF-Grafiken kann beginnen. Nach den ermittelten Werten aus dem Chunk BMHD kann bereits eine Routine erstellt werden, zu der das Programm verzweigt, wenn es auf einen BMHD-Chunk trifft. Diese Routine liest den BMHD-Chunk ein und analysiert ihn. Sie finden sie im folgenden Listing ab dem Label bitmapdaten.

```
REM Teil1
REM
```

```
einlesen:
CLS:GOSUB namelesen

einlesen4:
GOSUB dimen
OPEN filename$ FOR INPUT AS 1
Namen$=INPUT$(4,1)
IF Namen$="FORM" THEN weiter
PRINT "KEINE IFF-DATEI"
FOR i=1 to 10000:NEXT i:RETURN

weiter:
Formlaenge&=CVL(INPUT$(4,1))
FormKennung$=INPUT$(4,1)

chunklesen:
IF EOF(1) THEN zeigen
chunkname$=INPUT$(4,1)
chunklaenge&=CVL(INPUT$(4,1))
IF chunklaenge&/2-FIX(chunklaenge&/2)>0 THEN
** chunklaenge& = chunklaenge&+1
IF chunkname$="BMHD" THEN bitmapdaten
IF chunkname$="CMAP" THEN farbwert
IF chunkname$="BODY" THEN bildwerte
rest$=INPUT$(chunklaenge&,1)
GOTO chunklesen

zeigen:
CLOSE#1:RETURN

bitmapdaten:
bildbreite=CVI(INPUT$(2,1))
bildhoehe=CVI(INPUT$(2,1))
xpos=CVI(INPUT$(2,1))
ypos=CVI(INPUT$(2,1))
ebenen=ASC(INPUT$(1,1))
rest$=INPUT$(7,1)
schirmbreite=CVI(INPUT$(2,1))
schirmhoehe=CVI(INPUT$(2,1))
IF schirmbreite=320 THEN modus=1 ELSE modus=2
IF schirmhoehe=400 THEN modus=modus+2
GOSUB fensterholen:IF ebenen=ebenen1 THEN chunklesen
wind=wind+1
SCREEN 4,schirmbreite,schirmhoehe,ebenen,modus
WINDOW wind,filename$,,0,4
GOTO chunklesen
```

```
namelesen:
LOCATE 1,1:INPUT "Bitte File-Namen: ",filename$
IF filename$="" THEN namelesen
RETURN

dimen:
IF farbwerte OR dimension THEN RETURN
DIM rot! (31), gruen! (31), blau! (31)
dimension=-1:RETURN

fensterholen:
breite=WINDOW(2)
hoehe=WINDOW(3)
f7=WINDOW(6)
farben=f7+1
FOR n=1 TO 5
  IF 2^n=farben THEN ebenen1=n
NEXT n
IF hoehe>210 THEN BEEP:RETURN
IF breite>320 THEN br=640 ELSE br=320
RETURN
```

Der Anfang dieses Programmteils entspricht dem Programm IFFlesen. Überzählige Chunks, das heißt Chunks, welche nicht weiterverarbeitet werden können, werden in der String-Variablen rest$ eingelesen und ignoriert. Sobald ein BMHD-Chunk gefunden wird, ruft das Programm die Subroutine *bitmapdaten* auf. Diese liest die einzelnen Werte für Bildbreite, Bildhöhe und so weiter in der oben beschriebenen Reihenfolge ein, sofern sie von Interesse sind und überliest den (uninteressanten) Rest. Aus der Schirmbreite und der Schirmhöhe ergibt sich der Modus für den SCREEN, in dem das Bild später dargestellt wird.

Das Programm springt dann in die Routine *fensterholen*. Hier wird mit WINDOW(2) und WINDOW(3) die Breite und die Höhe des aktuellen Ausgabefensters ermittelt und mit WINDOW(6) die Anzahl der erlaubten Farben. Da dieser Wert die Hintergrundfarbe außer acht läßt, wird dann die Variable *farben* noch um eins erhöht. Aus dieser Variable wird dann in einer FOR-Schleife die Anzahl der für die Darstellung benötigten Bit-Ebenen berechnet und in der Variablen *ebenen1* abgelegt. Im Hauptprogramm wird dann anhand dieser Werte geprüft, ob die Zahl der nötigen Bit-Ebenen für das Bild in der Datei mit der Zahl der Bitebenen des aktuellen Fensters übereinstimmt. Ist dies der Fall, springt das Programm zu *chunklesen* zurück. Hat das aktuelle Fenster aber eine andere Ebenen-Zahl, wird ein neuer Bildschirm mit der passenden Zahl von Bitebenen und ein Fenster darin geöffnet.

Der zweite Chunk heißt CMAP, für ColorMAP, und beschreibt die Farben, die die Palette enthalten muß, damit dieses Bild korrekt dargestellt wird. Jede Farbe wird in diesem Chunk durch drei aufeinanderfolgende Bytes beschrieben, die jeweils den Rot-, Grün- und Blau-Anteil der entsprechenden Farbe enthalten. Dieser Chunk ist bei dem Bild der Mona Lisa 96 Byte lang. 96 Byte geteilt durch drei Byte pro Farbe ergibt 32 Farben. Dieses Ergebnis war zu erwarten, denn bereits im BMHD-Chunk waren für dieses

Bild fünf Bit-Ebenen (also 32 Farben) verzeichnet. Mit diesem Wissen können Sie nun den zweiten Teil der Lese-Routine programmieren, der den Namen *farbwert* bekommt.

```
REM Teil2

farbwert:
FOR f6= 0 TO chunklaenge&/3-1
rot!(f6)=(ASC(INPUT$(1,1)))/240
gruen!(f6)=(ASC(INPUT$(1,1)))/240
blau!(f6)=(ASC(INPUT$(1,1)))/240
NEXT f6
IF chunklaenge&/3-FIX(chunklaenge&/3)>0 THEN
rest$=INPUT$(1,1)

farbe:
FOR f6= 0 TO (2^ebenen)-1
PALETTE f6,rot!(f6),gruen!(f6),blau!(f6)
NEXT f6
GOTO chunklesen
```

In einer FOR-Schleife werden hier nacheinander die Farbanteile der 32 Farben eingelesen. Jeder wird nach dem Einlesen durch 240 geteilt, bevor er in dem entsprechenden Feld angelegt wird. Die Notwendigkeit für die Division ergibt sich ganz einfach daraus, daß im CMAP-Chunk für jeden Farbwert ein Byte zur Verfügung steht und davon nur die oberen 4 Bit genutzt werden. Der Amiga benötigt ja nur 4 Bit pro Farbwert. Der größtmögliche Farbwert ist 240 (wenn die oberen 4 Bit gesetzt sind). Die Division durch 240 macht aus der ganzen Zahl, die im CMAP-Chunk abgespeichert ist, den entsprechenden Bruch zwischen 0 und 1, den die PALETTE-Anweisung erwartet. Die 16 möglichen Werte der Bytes im CMAP-Chunk ergeben nach der Division durch 240 genau die 16 erlaubten Werte für die Farbanteile in der PALETTE-Anweisung.

Nach dem Einlesen der Farbanteile in drei Felder werden die Elemente dieser Felder in einer FOR-Schleife dazu verwendet, die aktuelle PALETTE aufzubauen. Damit ist die Bearbeitung des CMAP-Chunks abgeschlossen.

Nun muß nur noch der dritte und letzte Chunk, der BODY-Chunk, bearbeitet werden, der die eigentlichen Bild-Daten enthält. Das Format, in dem diese Daten abgespeichert sind, ist jedoch ein anderes als GET es erzeugt und in einem Feld ablegt – sonst wäre es ja überflüssig, sich überhaupt mit IFF zu beschäftigen. Und zwar kommen in einem BODY-Chunk zuerst die Daten für die erste Bildzeile der Bit-Ebene 1, anschließend die Daten für die Ebene 2 und so weiter, bis zur letzten (fünften) Bit-Ebene. Dann kommen die Daten für die zweite Zeile der ersten Bit-Ebene, für die zweite Zeile der zweiten Bit-Ebene und so weiter.

Der Aufbau des Bildes ist damit geklärt. Nun ist nur noch die Frage offen, wo diese Daten hingelegt werden müssen, damit sie am Bildschirm zu sehen sind. Wo liegen die Bitebenen des Bildschirms? Diese Frage läßt sich leider nicht allgemeingültig beantworten, denn beim Amiga gibt es kaum feste Adressen, an denen bestimmte Daten zu finden sind. Die Adresse, an der die Daten für den Bildschirminhalt stehen, kann sich

zum Beispiel jederzeit ändern. Glücklicherweise gibt es aber Mittel und Wege, diese Adresse, ausgehend von einem Fenster, zu ermitteln.

Zu Hilfe kommt hier wieder einmal die WINDOW-Funktion. WINDOW(8) liefert die Adresse des sogenannten »Rastports« des aktuellen Ausgabefensters.

```
adzeiger=WINDOW(8)
```

Dieser Rastport ist eine komplizierte Datenstruktur mit einer ganzen Reihe von Feldern. Die meisten sind für das hier vorgestellte Programm aber uninteressant. Das zweite im Rastport (das also 4 Byte hinter der ersten Adresse beginnt) enthält die Adresse einer weiteren Datenstruktur, einer sogenannten BitMap.

```
zwadresse=PEEKL(adzeiger+4)
```

In einem BitMap-Datensatz kommen zunächst zwei Langworte mit Daten, die hier nicht interessieren und dann (8 Byte hinter der Startadresse der Struktur) die Liste mit den Adressen der einzelnen Bit-Ebenen.

```
adresse=zwadresse+8
```

Die Adresse der zweiten Bit-Ebene ist wieder vier Byte weiter zu finden (bei zwadresse+12). Damit ist das Ziel erreicht: Die Adresse der ersten Bit-Ebene findet man mit den folgenden vier Anweisungen.

```
adzeiger=WINDOW(8)
zwadresse=PEEKL(adzeiger+4)
adresse=zwadresse+8
ebeneadr=PEEKL(adresse)
```

Das Ganze muß natürlich nicht vier Zeilen umfassen und läßt sich auch zusammenfassen. Bitte geben Sie im List-Fenster ein:

```
FOR i=0 to 4 STEP 4
   PRINT PEEKL(PEEKL(WINDOW(8)+4)+8+i)
NEXT
```

Sie erhalten zwei Zahlen, von denen die zweite um 20480 höher ist als die erste Zahl. Das sind die Startadressen der beiden Bit-Ebenen Ihres derzeitigen Bildschirmes.

Die Differenz der beiden Startadressen ergibt sich aus der Anzahl der benötigten Bytes pro Ebene.

Breite 640 * Höhe 256 / 8 Bit = 20480 Byte

(Eigentlich müssen die beiden Bit-Ebenen nicht unbedingt direkt aufeinanderfolgen; ihr Abstand kann also auch größer sein. Meistens ist dies aber nicht der Fall, da der Amiga beim Öffnen eines neuen Bildschirms die dafür benötigten Bit-Ebenen direkt hintereinanderlegt.) Nach dieser Reise in den Speicher-Irrgarten können Sie nun auch die dritte Routine *bildwerte* des Programms zum Lesen von IFF-Dateien erstellen. Diese liest die Bild-Daten von BODY-Chunk direkt in die Bit-Ebenen des Bildschirms ein.

```
REM Teil3

bildwerte:
bytprozeile=bildbreite/8
adzeiger&=WINDOW(8)
zwadresse&=PEEKL(adzeiger&+4)
adresse&=zwadresse&+8
FOR h=0 TO bildhoehe-1
  FOR eb=0 TO ebenen-1
      ebeneadr&=PEEKL(adresse&+4*eb)
      FOR z=0 TO bytprozeile/4-1
      POKEL ebeneadr&+4*z+bytprozeile*h,CVL(INPUT$(4,1))
NEXT z,eb,h
GOTO chunklesen
```

Zuerst wird die Bildbreite durch 8 geteilt und das Ergebnis in der Variablen bytprozeile (Anzahl von Bytes in einer ein Punkt hohen Zeile des Bildschirms) gespeichert. Dann wird in die Variable adresse& die Adresse der ersten Bit-Ebene des aktuellen Bildschirms geholt. Die äußere FOR-Schleife durchläuft nun die Zeilen (bildhoehe) des Bildes, die mittlere die einzelnen Bit-Ebenen (ebenen) und die innere schreibt die Daten für eine Zeile einer Bit-Ebene aus dem Chunk in den Speicher.

Nun sollen Sie aber auch sehen, was das fertige Programm leistet. Hängen Sie dazu nun bitte die Programm-Abschnitte Teil1, Teil2 und Teil3 aneinander, und speichern Sie das entstandene Programm unter dem Namen *Bildlesen* ab. Starten Sie das Programm, und es fragt Sie zunächst nach einem Datei-Namen. Geben Sie daraufhin den folgenden Namen (oder den Pfad-Namen eines anderen Bildes) ein:

GrafiCraft:MonaLisa

Ist das nicht ein Erfolg? Langsam rollt das Bild der Mona Lisa, Zeile für Zeile vom oberen Bildrand herab. Die Farben entsprechen exakt dem Original auf der GrafiCraft-Diskette (dafür wird ja nach dem Einlesen des CMAP-Chunks gesorgt). Probieren Sie das Programm ruhig noch an einigen anderen Bildern aus.

3.3.3 Speichern im INTERCHANGE FILE FORMAT

Jetzt, wo die erste Hürde erfolgreich genommen ist, wird es Ihnen auch nicht schwerfallen, ein Programm zu schreiben, das Dateien im IFF-Standard erzeugen kann. Sie

brauchen dazu eigentlich fast nur die Lesebefehle (INPUT$) aus den ersten drei Teilen des Programms in Schreibbefehle (PRINT) zu ändern. Das entsprechende Programm folgt im nächsten Listing. Es ist mit Teil4 überschrieben, was Ihnen schon einen Tip gibt, was damit gemacht werden soll. Es wird später hinter den letzten Teil3 des Programmes zum Einlesen von IFF-Dateien gehängt.

```
REM Teil4

speichernals:
GOSUB namelesen

speichern:
IF filename$="" THEN filename$="Namenlos"
GOSUB fensterholen
bytes&=br*hoehe/8*ebenen1
IF farbwerte=0 THEN zus=52 ELSE zus=52+8+farben*3
GOSUB dimen
OPEN filename$ FOR OUTPUT AS 1

form:
PRINT #1,"FORM";MKL$(bytes&+zus);"ILBM";

bmhd:
PRINT #1,"BMHD";MKL$(20);
PRINT #1,MKI$(br);MKI$(hoehe);MKI$(0);MKI$(0);CHR$(ebenen1);
PRINT #1,CHR$(0);MKL$(0);
IF br=640 THEN PRINT #1,CHR$(5);ELSE PRINT #1,CHR$(10);
PRINT #1,CHR$(11);MKI$(br);MKI$(200);

cmap:
IF farbwerte=0 THEN body
PRINT #1,"CMAP";MKL$(farben*3);
FOR f=0 TO f7
   PRINT #1,CHR$(rot!(f)*240);
   PRINT #1,CHR$(gruen!(f)*240);
   PRINT #1,CHR$(blau!(f)*240);
NEXT f

body:
PRINT #1,"BODY";MKL$(bytes&);
adzeiger&=WINDOW(8)
zwadresse&=PEEKL(adzeiger&+4)
adresse&=zwadresse&+8
bytprozeile=br/8
FOR h=0 TO hoehe-1
   SOUND 2000,2,50
   FOR ebe=0 TO ebenen1-1
        ebeneadr&=PEEKL(adresse&+4*ebe)
        FOR z=0 TO bytprozeile/4-1
        PRINT #1,MKL$(PEEKL(ebeneadr&+4*z+bytprozeile*h));
NEXT z,ebe,h
```

```
camg:
PRINT #1,"CAMG";MKL$(4);MKL$(16384);
CLOSE #1
RETURN
```

Die Routine verwendet den Namen »Namenlos«, wenn kein File-Name angegeben wurde. Der Grund liegt in der möglichst vielfältigen Einsatzmöglichkeit dieses Programmes. Es kann so auch Programme ohne vorgegebenen Namen abspeichern.

Im ersten Teil des Programmes erhalten die Variablen, die für die einzelnen Chunks benötigt werden, ihre Werte zugewiesen. Die Daten für die Bildbreite, Bildhöhe und die Anzahl der Bit-Ebenen werden mit der WINDOW-Funktion in der Subroutine *fensterholen*, die bereits an anderer Stelle besprochen wurde, festgestellt. In der Variablen *bytes* wird die Anzahl der Bytes, die das Grafikbild enthält, aus der Bildbreite *br*, Bildhöhe *hoehe*, und der Anzahl der Bit-Ebenen ebenen1 errechnet. Die Variable *zus* erhält die Anzahl der Bytes, die zusätzlich zu den Bytes des Grafikbildes für den Grundbaustein FORM in die Datei geschrieben werden müssen: 52 (ohne Berücksichtigung von CMAP). Die Variable *farbwerte* wird als Zeiger eingesetzt, der vom Programm, das *speicherals* aufruft, gesetzt werden muß. Ist dieser Zeiger logisch wahr, wird auf *zus* noch die Länge des CMAP-Chunkes aufaddiert, die natürlich von der Anzahl der Bit-Ebenen abhängt. Anschließend wird die Subroutine *dimen* aufgerufen. Ist der Zeiger *dimension* falsch, werden dort die Feldvariablen Rot, Gruen und Blau dimensioniert und *dimension* auf -1 (logisch wahr) gesetzt.

Nun kann die Datei filename$ geöffnet werden. Zuerst wird der Grundbaustein FORM in die Datei geschrieben. Dann folgt der BMHD-Chunk. Zuerst wird der Chunk-Name "BMHD" und die Chunk-Länge 20 in die Datei geschrieben. Anschließend folgen die Werte für die Bildbreite br, Bildhöhe *hoehe* und so weiter.

Im Programmteil *cmap* wird wieder zunächst der Zeiger *farbwerte* abgefragt. Ist das Basic-Programm auf diese IFF-Lese- und Schreib-Routine vorbereitet, werden aus dem Hauptprogramm die Farbwerte aus den Feldern für die roten, grünen und blauen Anteile der einzelnen Farben übernommen und ausgegeben (mit 240 multipliziert; siehe oben). Ist *farbwerte* falsch (=0), wird die Routine *cmap* einfach übersprungen, also kein CMAP-Chunk geschrieben. In einem solchen Fall wird beim späteren Einlesen der Datei die Palette einfach nicht geändert.

Als nächstes kommt nun der BODY-Chunk an die Reihe. Wie üblich, wird zuerst der Name "BODY" und die Länge des Chunks (bytes) in die Datei geschrieben. Die Adressen für die Bit-Ebenen werden wieder – wie in Teil3 – mit WINDOW(8) ermittelt. Die äußere FOR-Schleife durchläuft nun wieder die einzelnen Zeilen des Bildes, die mittlere die Bit-Ebenen und die innere schreibt eine Zeile einer Bit-Ebene in 4-Byte-Häppchen in die Datei. In der äußeren FOR-Schleife erklingt für jede abgespeicherte Zeile ein Signalton. Durch dieses akustische Signal wissen Sie, wann die Speicherung beendet ist. Wenn Sie dieses Geräusch stört, können Sie den SOUND-Befehl einfach wegfallen lassen. Zum Schluß wird noch ein CAMG-Chunk geschrieben, der eine exakte Kopie desselben Chunks in der Mona-Lisa-Datei ist, und die Datei geschlossen.

Damit haben Sie die beiden Utilities zum Lesen und Schreiben von IFF-Grafik-Dateien komplett. Sie können sie einzeln einsetzen oder mit einem kleinen Programmanfang als Komplettprogramm:

```
REM iffLesenSpeichern
'P.3.3-6
GOTO einlesen
REM ######Einschub zum Anhaengen an unvorbereitete Programme
GOSUB iffroutinen
END

iffroutinen:
ta$=INKEY$:IF ta$="" THEN iffroutinen
IF ta$=CHR$(13)THEN speichern
REM ######Einschub ende ##########
REM Teil1 anhaengen
REM Teil2 anhaengen
REM Teil3 anhaengen
REM Teil4 anhängen
```

Speichern Sie dieses Programm dann unter dem Namen iffLesenSpeichern ab. Sie haben damit ein äußerst vielseitiges Werkzeug für die verschiedensten Anwendungen. Es ist so ausgelegt, daß es fast in jedem Programm eingesetzt werden kann.

Hängen Sie zunächst iffLesenSpeichern an das Zeichenprogramm PAINTAmiga aus dem letzten Kapitel. Bei PAINTAmiga müssen dazu die drei letzten Zeilen

```
Einlesen: ...
speichernals: ...
speichern: ...
```

durch REM-Befehle außer Kraft gesetzt oder gelöscht und dahinter dann die Zeilen von iffLesenSpeichern angehängt werden. Wenn Sie PAINTAmiga nun benutzen, werden Ihre Bilder automatisch im IFF-Format gespeichert und eingelesen. Das wäre an sich nichts Besonderes und es würde Ihnen nicht viel nützen. Sie können aber auch die Bilder sofort auf eine GraphiCraft-Diskette speichern. (Dateiname: "Graphicraft:<Name>"). Mit GraphiCraft können Sie sich das Bild dann farbig ausdrucken lassen. Umgekehrt können Sie ein GraphiCraft-Bild einlesen und mit PAINTAmiga weiter bearbeiten. Die beiden Programme sind damit fähig, Daten auszutauschen (kompatibel)! Beachten Sie aber bitte, daß Sie die alten Bilder, die Sie vor dieser Änderung mit PAINTAmiga erstellt haben, nun nicht mehr einlesen können.

Das Programm iffLesenSpeichern speichert auch Teilbilder, wie Sie mit dem Programm Adventure aus Kapitel 5.1 überprüfen können. Ich bin sicher, daß Sie mit diesem Programm viele eigene Ideen verwirklichen können.

3.4 Sequentielle Dateien

Nachdem Sie im ersten Teil dieses Kapitels vor allem erfahren haben, wie Sie Dateioperationen (und PEEK/POKE) für grafische Anwendungen einsetzen können, wollen wir uns nun etwas näher mit anderen Dateioperationen beschäftigen. Dieser Abschnitt behandelt die drei grundlegenden Ein- und Ausgabemöglichkeiten von sequentiellen Dateien.

Amiga-Basic sieht zwei Arten von Dateien vor, sequentielle und Direktzugriffs-Dateien. Die sequentielle Datei ist recht einfach zu programmieren, hat aber einige Nachteile. Die Daten sind bei diesem Datei-Typ hintereinander auf der Diskette abgelegt. Es handelt sich dabei üblicherweise nicht einfach um einen Strom von Bytes, sondern um eine Folge von »Datensätzen«. (Die IFF-Chunks sind zum Beispiel solche Datensätze). Suchen Sie nach einem bestimmten Datensatz und dieser kommt am Ende der Datei, müssen Sie erst die ganze Datei einlesen (überlesen), bevor Sie an den gesuchten Satz gelangen. Eine Änderung dieses Datensatzes ist ebenfalls sehr umständlich. Solange es sich um kurze Dateien handelt, ist das aber kein allzu großes Problem. Bedenken Sie aber, daß eine Diskette 880 KB Daten und somit auch rechte große Dateien enthalten kann. Das Hauptanwendungsgebiet von sequentiellen Dateien sind deshalb kleine Dateien, die man problemlos mit einem Schlag einlesen kann und die reinen Textdateien (diese sind nämlich nur ein Strom von Bytes).

Bei sequentiellen Dateien unterscheiden wir zwischen drei verschiedenen Zugriffsarten: Anlegen (Schreiben), Anfügen und Lesen. Bei einer sequentiellen Datei müssen Sie schon beim Öffnen entscheiden, welche dieser Zugriffsarten Sie verwenden wollen. Sie können also nicht etwas in eine Datei schreiben und gleichzeitig etwas daraus lesen. Außerdem dürfen Sie eine bereits existierende Datei nicht einfach zum Schreiben öffnen. In diesem Falle würden die bereits darin vorhandenen Daten nämlich gelöscht. Am Schluß dieses Abschnitts finden Sie eine kleine Liste mit einigen Punkten, die bei der Arbeit mit sequentiellen Dateien beachtet werden müssen. Die beiden Programmbeispiele in diesem und im nächsten Abschnitt berücksichtigen aus Gründen des Platzbedarfs nicht alle diese Kriterien. Die Kfz-Datei (siehe unten) hingegen geht exakt danach vor.

Die erste Zugriffsart ist das Erzeugen/Schreiben einer Datei. Der entsprechende Basic-Befehl dafür lautet:

```
OPEN "Name" FOR OUTPUT AS #Dateiummer
oder kürzer
OPEN "O", #Dateiummer, "Name"
```

Dadurch wird eine Datei mit dem genannten Namen neu angelegt oder eine vorhandene zum Überschreiben geöffnet, die später mit der Kennung #Dateinummer angesprochen werden kann. Dies soll nun durch ein kleines Beispiel verdeutlicht werden.

Anlegen:

```
OPEN"O",#1,"video"
```

```
Eintrag:
CLS
PRINT "Irrtum = 0 *** Ende = E"
PRINT :PRINT
LINE INPUT"Filmtitel? ";tit$
IF tit$="0" THEN eintrag
IF tit$="E" THEN beenden
LINE INPUT"Kassette? ";kas$
IF kas$="0" THEN eintrag
IF kas$="E" THEN beenden
WRITE#1,tit$,kas$
PRINT
GOTO eintrag

Beenden:
CLOSE#1
GOTO menue

Ende:
END
```

Das Programm erfragt, nach dem Öffnen der Datei, Titel und Nummer eines Videofilms und legt diese Angaben in den Variablen *tit$* und *kas$* ab. Mit der WRITE#-Anweisung werden diese beiden Variablen in die Datei »video« geschrieben. Solange nicht E für Ende eingegeben wird, kann ein Datensatz nach dem anderen abgespeichert werden. Dann wird die Datei mit CLOSE geschlossen. Danach erscheint (im Moment noch) eine Fehlermeldung, da der Programmteil *menue* noch fehlt. Dieser wird weiter unten noch erläutert.

Neben OPEN verwendet das Programm die beiden neuen Befehle WRITE und CLOSE.

WRITE #Dateinummer, Liste von Ausdrücken
schreibt die Werte der Ausdrücke in die Datei mit der angegebenen Nummer

CLOSE *#Dateinummer, #Dateinummer, ...*
Schließt die Dateien mit den angegebenen Nummern; danach können keine Daten mehr in diese Dateien geschrieben oder daraus gelesen werden.

Um weitere Daten an eine schon bestehende Datei anzuhängen, müssen Sie die zweite Zugriffsart, das Anhängen, benutzen. Hierzu muß die Datei mit dem folgenden Befehl geöffnet werden:

```
OPEN "NAME" FOR APPEND AS #Dateiummer
oder kürzer
OPEN "A", #Dateiummer , "Name"
```

Das folgende Programmfragment demonstriert, wie dieser Befehl verwendet wird.

```
Ergaenzen:
OPEN"A",#1,"video"

Anhaengen:
CLS
PRINT "Irrtum = 0 *** Ende = E"
PRINT :PRINT
LINE INPUT"Filmtitel? ";tit$
IF tit$="0" THEN anhaengen
IF tit$="E" THEN zurueck
LINE INPUT"Kassette? ";kas$
IF kas$="0" THEN anhaengen
IF kas$="E" THEN zurueck
PRINT #1,tit$
PRINT #1,kas$
GOTO anhaengen

Zurueck:
CLOSE#1
les=1
GOTO menue
```

Wie Sie erkennen können, gleicht das Anfügen an eine Datei, bis auf das Öffnen, dem Anlegen und anschließenden Schreiben in eine neue Datei. Die beiden Variablen *tit$* und *kas$* werden diesmal mit der PRINT#-Anweisung an die Datei übergeben. PRINT und WRITE tun in diesem Fall exakt dasselbe.

Die dritte und letzte Zugriffsart ist das Lesen von Daten aus einer bereits vorhandenen sequentiellen Datei. Auch hierfür muß die Datei besonders geöffnet werden. Dazu dient der folgende Befehl :

```
OPEN "Name" FOR INPUT AS #Dateinummer
oder kürzer
OPEN "I", #Dateinummer, "Name"
```

Das folgende Programmfragment zeigt, wie diese Zugriffsart in einem Programm verwendet werden kann.

```
Einlesen:
OPEN"I",#1,"video"
nr=0
WHILE NOT EOF(1)
   INPUT#1,tit$(nr),kas$(nr)
   nr=nr+1
WEND
CLOSE#1
les=0
RETURN
```

```
Lesen:
CLS
IF les THEN GOSUB einlesen
FOR i = 0 TO nr-1
  PRINT tit$(i),kas$(i)
  GOSUB taste
NEXT i
GOTO menue

Taste:
ta$=INKEY$ : IF ta$="" THEN taste
RETURN
```

Nachdem die Datei zum Lesen geöffnet wurde, werden hier mit einer WHILE-Schleife so lange Daten eingelesen, bis das Ende der Datei (EOF = End Of File) erreicht ist. Die eingelesenen Daten werden in den Feldvariablen *tit$()* und *kas$()* abgelegt und dann mit PRINT am Bildschirm gezeigt. Zum Einlesen dient der Befehl INPUT, der in ähnlicher Form bereits zum Einlesen einer IFF-Datei verwendet wurde.

INPUT #Nummer,Variable,*Variable,*
liest Daten aus der sequentiellen Datei mit der Kennung *Nummer*, weist diese dann einer oder mehreren *Variablen* zu

Wenn Sie die eingelesenen Daten nicht nur auf dem Bildschirm, sondern auch schwarz auf weiß haben wollen, können Sie die folgende kleine Routine anfügen.

```
Drucken:
IF les THEN GOSUB einlesen
FOR i = 0 TO nr-1
  LPRINT tit$(i),kas$(i)
NEXT i
GOTO menue
```

Damit Sie diese, zugegeben etwas primitive, Dateiverwaltung benutzen können, brauchen Sie nur noch den Programmteil *menu* zu schreiben und die Feldvariablen *tit$* und *kas$* zu dimensionieren. Genau das enthält das folgende Programmfragment. Geben Sie es ein und hängen Sie die anderen Programmfragmente zum Erzeugen, Anhängen und Lesen von Daten dahinter. Speichern Sie es dann unter dem Namen Video-Datei ab. (Beachten Sie bitte, wenn Sie das Programm verwenden wollen, daß es in der hier vorgestellten Form nur mit maximal 100 Datensätzen fertig wird; länger sind die drei Feldvariablen nicht.)

```
REM Video-Datei
'P.3.4-1
CLEAR
```

```
start:
DIM tit$(100),kas$(100)
les=1

menue:
CLS
PRINT ,,"VIDEO DATEI"
PRINT :PRINT ,,"M E N U E"
PRINT :PRINT ,,"1 = Datei anlegen"
PRINT :PRINT ,,"2 = Datei ergaenzen"
PRINT :PRINT ,,"3 = Datei lesen"
PRINT :PRINT ,,"4 = Datei drucken"
PRINT :PRINT ,,"5 = E N D E"

wahl:
ta$=INKEY$ : IF ta$ = "" THEN wahl
ta=VAL(ta$)
IF ta<1 OR ta>5 THEN BEEP:GOTO wahl
ON ta GOTO anlegen,ergaenzen,lesen,drucken,ende
```

Wie Sie an diesem Programm sehen können, ist ein einfaches Dateiverwaltungs-Programm doch wirklich ganz leicht zu erstellen.

Im folgenden Schema wird die grobe Struktur gezeigt, die ein Programm haben sollte, das mit einer sequentiellen Datei arbeitet. Diese entspricht nicht ganz dem Programm Video-Datei, da dieses aus Vereinfachungsgründen etwas anders geschrieben wurde:

-Felder dimensionieren
-abfragen, ob Datei bereits existiert
-wenn nein,
    Datei zum Schreiben öffnen
    Datensatz mit fiktiven Daten hineinschreiben
    Datei schließen
-Menü zur Verzweigung in die einzelnen Routinen
    -neue Daten
        Datei zum Anfügen (APPEND) öffnen
        Daten mit PRINT# in Datei schreiben
        Datei schließen
    -vorhandene Daten lesen
        Datei zum Lesen (INPUT) öffnen
        Daten mit INPUT# einlesen
        Datei schließen.

Besonders wichtig an diesem Schema sind die ersten Schritte: das Überprüfen, ist die Datei schon vorhanden, und dann gegebenenfalls das Erzeugen der Datei. Mit dieser Vorgehensweise vermeiden Sie ein versehentliches Überschreiben einer schon vorhandenen Datei. Sie erfahren später in diesem Kapitel, wie Sie feststellen können, ob eine Datei auf der Diskette vorhanden ist oder nicht.

3.5 Kochbuch

Wie Sie bereits gehört haben werden, setzt man sequentielle Dateien vor allem zum Verarbeiten von Texten ein. In diesem Abschnitt werden Sie nun ein Programm erstellen, mit dem Sie kleine Texte verwalten können. Ich habe dem Programm den Namen »Kochbuch« gegeben, da Kochrezepte kaum in einer Zeile einzugeben sind. Sie können dasselbe Programm aber natürlich auch für andere Zwecke verwenden.

Wenn Sie das Programm aktiv nutzen wollen, beachten Sie bitte die Hinweise am Ende des letzten Abschnitts. Im Programm wurde aus Platzgründen darauf verzichtet, die Datei vor dem Überschreiben zu schützen. Wichtiger erschien mir, zu zeigen, daß auch die Ausgabe formatierter Texte in Amiga-Basic kein Problem ist. Und bevor Sie beginnen mit dem Programm zu arbeiten, legen Sie bitte auch noch eine leere Schublade mit dem Namen REZEPTE auf der Diskette BFA an. Diese wird die Rezepte aufnehmen, die das Programm verwaltet.

Nun aber zunächst das Listing des Programms Kochbuch. Erläuterungen dazu finden Sie im Anschluß.

```
REM Koch-Datei
REM P.3.5-1
REM ******************************************************
REM Wichtig! Vor Erststart Schublade "REZEPTE" anlegen
REM ******************************************************
CLEAR

start:
DIM ger$(100),text$(100)
les=1
WIDTH 70

menue:
CLS
PRINT ,,"K O C H B U C H"
PRINT :PRINT ,,"M E N U E"
PRINT :PRINT ,,"1 = Kochbuch anlegen"
PRINT :PRINT ,,"2 = Kochbuch ergaenzen"
PRINT :PRINT ,,"3 = Inhaltsverzeichnis"
PRINT :PRINT ,,"4 = Rezept lesen"
PRINT :PRINT ,,"5 = Rezept drucken"
PRINT :PRINT ,,"6 = E N D E"

wahl:
ta$=INKEY$ : IF ta$ = "" THEN wahl
ta=VAL(ta$)
IF ta<1 OR ta>6 THEN BEEP:GOTO wahl
ON ta GOTO anlegen,ergaenzen,inhalt,lesen,drucken,ende
```

```
beenden:
CLOSE#1
GOTO menue

ende:
END

anlegen:
OPEN"O",#1,"koch"

eintrag:
zck=0
GOSUB ein
IF zck THEN zurueck
WRITE#1,ger$
PRINT
GOTO eintrag

ergaenzen:
OPEN"A",#1,"koch"

anhaengen:
zck=0
GOSUB ein
IF zck THEN zurueck
PRINT #1,ger$
GOTO anhaengen

ein:
CLS
PRINT "Irrtum = 0 *** Ende = E"
PRINT :PRINT
LINE INPUT"Gericht ? ";ger$
IF ger$="0" THEN ein
IF ger$="E" THEN zck=1:RETURN
PRINT "bitte eingeben >"
tx$="":lg=0

tet:
GOSUB taste
tx$=tx$+ta$
lg=lg+1
PRINT ta$;
IF ta$=CHR$(8) THEN
  lg=lg-2
  tx$=LEFT$(tx$,lg)
END IF
IF ta$ <>CHR$(13) THEN tet
filename$="REZEPTE/"+ger$
lg=lg+5
OPEN"O",#2,filename$,lg
```

```
WRITE#2,tx$
CLOSE#2
RETURN

zurueck:
CLOSE#1
les=1
GOTO menue

inhalt:
GOSUB inhzeig
GOSUB taste
GOTO menue

inhzeig:
CLS
IF les THEN GOSUB einlesen
PRINT :PRINT "I N H A L T S V E R Z E I C H N I S"'
ze=3: sp=1
IF les THEN GOSUB einlesen
FOR i = 0 TO nr
   LOCATE ze,sp
   PRINT ger$(i)
   sp=sp+20
   IF sp=81 THEN sp=1:ze=ze+1
NEXT i
RETURN

lesen:
GOSUB holen
zl=0 : GOSUB format
FOR i=0 TO zl
   PRINT text$(i)
NEXT i
GOSUB taste
GOTO menue

holen:
GOSUB inhzeig
PRINT :LINE INPUT "bitte Rezept eingeben: ";rez$
filename$="rezepte/"+rez$
OPEN "I",#2,filename$
LINE INPUT #2,tx$
CLOSE#2
FOR i=0 TO 100
 text$(i)=""
NEXT i
RETURN
```

```
taste:
ta$=INKEY$ : IF ta$="" THEN taste
RETURN

einlesen:
OPEN"I",#1,"koch"
nr=0
WHILE NOT EOF(1)
   INPUT#1,ger$(nr)
   nr=nr+1
WEND
CLOSE#1
les=0
RETURN

format:
mges=LEN(tx$)
IF mges<61 THEN
   text$(zl)=tx$
   RETURN
END IF
br=60
ww=1
WHILE ww
   le$=MID$(tx$,br,1)
   IF le$=CHR$(32) THEN ww=0 ELSE ww=1
   br=br-1
        IF br=1 THEN ww=0
WEND
IF br=1 THEN br=59
br=br+1
text$(zl)=LEFT$(tx$,br)
mges=mges-br
tx$=RIGHT$(tx$,mges)
zl=zl+1
GOTO format
RETURN

drucken:
GOSUB holen
zl=0 : GOSUB format
LPRINT rez$
ste=LEN(rez$)
srn$=STRING$(ste,42)
LPRINT srn$
FOR i=0 TO zl
   LPRINT text$(i)
```

```
NEXT i
LPRINT
LPRINT
GOTO menue
```

Der Anfang des Programms bis zum Label *ein* sollte Ihnen eigentlich schon geläufig sein; Video-Datei sah nicht wesentlich anders aus. Danach wird es aber wieder interessant. Die Rezepte, die abgespeichert werden sollen, werden sicherlich unterschiedlich lang werden. Jedes Rezept wird deshalb in einer eigenen Datei abgelegt. Damit es wiedergefunden werden kann, wird in einer zusätzlichen Datei ein Verzeichnis der vorhandenen Rezepte gespeichert. Diese Datei bekommt den Namen "koch".

Im Programmteil hinter dem Label *anlegen* wird die Datei "koch" angelegt und dann die Subroutine *ein* aufgerufen. Diese liest zunächst einen Rezeptnamen und dann das zugehörige Rezept ein. Der Rezeptname wird zuerst mit der Anweisung LINE INPUT erfragt und an die Zeichenkettenvariable ger$ übergeben.

LINE INPUT;*"Text"*; Stringvariable
liest ähnlich wie INPUT max. 255 Zeichen von der Tastatur ein und weist sie der Variablen zu; im Gegensatz zur normalen INPUT-Anweisung kann man mit Hilfe von LINE INPUT aber auch Kommata oder Leerzeichen eingeben

Der mit LINE INPUT eingelesene Rezeptname wird gleichzeitig der Dateiname für die Datei, die das entsprechende Rezept enthält, und wird später in der Datei "koch" verzeichnet. Dann wird der Text des Rezeptes buchstabenweise eingelesen. Jedesmal wenn eine Taste gedrückt wird (dies ermittelt die Subroutine taste), hängt das Programm den entsprechenden Buchstaben an das Ende des Textes in der Variablen *tx$*. Die Variable *lg* zählt dabei die Anzahl der eingegebenen Buchstaben mit.

Fast alle Buchstaben werden so abgespeichert, wie sie eingegeben werden. Einzige Ausnahme sind BACKSPACE und RETURN. Stellt ein BACKSPACE (=CHR$(8)) fest, wird diese Taste zwar zuerst abgespeichert, dann aber werden die letzten beiden Buchstaben von *tx$* gelöscht, und somit nicht nur das BACKSPACE, sondern auch der Buchstabe davor aus dem eingegebenen Text entfernt. BACKSPACE wirkt also genauso, wie Sie es von anderen Programmen her kennen. Sobald der Anwender hingegen die RETURN-Taste betätigt, wird die Eingabe beendet und der Text mit dem Namen des Rezeptes als Datei gespeichert. Die Variable *lg*, welche immer als Zähler mitläuft, dient dabei zur Bestimmung der Länge des abzuspeichernden Datensatzes.

Zur Überprüfung, ob ein BACKSPACE eingegeben wurde, wird die Funktion CHR$ verwendet.

v$=CHR$(Zahl)
liefert den Buchstaben mit dem ASCII-Code *Zahl* (zwischen 0 und 255)

Zum Abschneiden der letzten beiden Buchstaben verwendet das Programm die LEFT$-Funktion.

v$=LEFT$(String, N)
liefert einen Text, der aus den ersten *N* Buchstaben von *String* besteht

Hierzu noch ein Beispiel: Geben Sie die folgenden beiden Befehle im Direktmodus ein:

```
A$="Basic-Programm":PRINT LEFT$(A$,5)
```

und Sie erhalten die folgende Ausgabe (die ersten 5 Buchstaben von A$)

```
Basic
```

Die Unterroutine *holen* liest einen gespeicherten Text wieder ein. Sie zeigt zunächst die Liste der verfügbaren Rezepte (durch einen Aufruf von *inhzeig*) und fragt dann nach dem Namen des gewünschten Rezeptes. Dieser Name wird dann eingelesen, um den Namen der Schubalde Rezepte erweitert und damit dann die Datei geöffnet, die das Rezept enthält. Deren Inhalt wird mit LINE INPUT in die Variable *tx$* gelesen und dann werden noch die Elemente des Feldes *text$* gelöscht, das später den formatierten Text des Rezeptes aufnehmen wird. Dadurch wird verhindert, daß Sie Reste des letzten eingelesenen Rezeptes gezeigt bekommen.

Nun kommt die schönste Routine des ganzen Programmes: *format.* In ihr wird der Text des Rezeptes linksbündig und 60 Zeichen breit formatiert. Ich hoffe, Sie bekommen dabei Geschmack auf weitere Experimente in dieser Richtung. In dieser kleinen Routine finden Sie die Grundlagen dazu. Zuerst benötigen wir die Anzahl der Zeichen des zu formatierenden Textes *tx$*.

v=LEN(x$)
liefert die Länge der Zeichenkette *x$*

Dazu wieder ein Beispiel:
Wenn Sie die beiden folgenden Anweisungen im Direktmodus eingeben

```
A$="Zeichensatz":PRINT LEN(A$)
```

erhalten Sie als Ergebnis die Zahl 11.

Format hat nun die Aufgabe, den Text des Rezeptes in *tx$* in Abschnitte zu zerlegen, die höchstens 60 Zeichen groß sind. Wenn die Länge des Textes kleiner ist als 61 Zeichen, ist *format* schon fertig. Ansonsten bleibt noch einiges zu tun. Zunächst wird versucht, den Text an der 60sten Stelle zu brechen. Hierzu wird die Variable *br* auf 60 Zeichen gesetzt. Die folgende WHILE-Schleife sucht dann nach einem Leerzeichen vor der 60sten Stelle und packt alle Zeichen des Textes, die davor stehen, in ein Element des Feldes *text$*. Später werden diese Zeichen dann aus *text$* gelöscht. Dieser Vorgang wird nun einfach so oft wiederholt, bis *text$* leer ist. Wieviele Zeilen auf diese Weise bereits erstellt wurden, hält dabei permanent die Variable *zl* fest. Beim Aufbrechen des unter Umständen langen Rezeptes in kurze Zeilen werden eine ganze Reihe neue Funktionen verwendet. Dies soll nun auch erläutert werden.

v$=MID$(String,n,m)
liefert den Teil von *String*, der bei Position *n* beginnt und *m* Zeichen lang ist

Dazu wieder ein Beispiel:
Wenn Sie die beiden folgenden Anweisungen im Direktmodus eingeben

```
A$="Disketten-Station":PRINT MID$(A$,4,5)
```

gibt Amiga-Basic aus

```
kette
```

Zum Abschneiden des vorderen Teils des Rezeptes (in *tx$*) wird die Funktion RIGHT verwendet, die den rechten Teil eines Strings liefert.

v$=RIGHT$(String,n)
liefert einen Text, der die letzten *n* Zeichen des *Strings* enthält

Auch hierzu wieder ein Beispiel:

Wenn Sie die beiden folgenden Anweisungen im Direktmodus eingeben

```
A$="Bleistift":PRINT RIGHT$(A$,5)
```

erhalten Sie :

```
stift
```

Zum Abschluß des Programms soll aber nun auch noch eine Möglichkeit eingebaut werden, Rezepte zu Papier zu bringen. Da hierzu keine besonderen Drucker-Funktionen benötigt werden, kann dies direkt mit LPRINT gemacht werden. Vorher wird das Rezept aber mit *holen* eingelesen und dann wie gehabt auf 60 Zeichen Breite formatiert. Damit der Ausdruck etwas besser aussieht, wird der Name des Rezeptes in der zweiten Zeile mit einer Reihe von Sternen (*) unterstrichen. Dazu ermittelt das Programm mit LEN die

Anzahl der Zeichen des Rezeptnamens und bildet dann mit Hilfe der STRING$ -Funktion in der Variablen *srn$* einen Text mit entsprechend vielen Sternchen.

v$=STRING$(n, Zahl)
v$=STRING$(n, String)
liefert einen Text, der aus *n* Zeichen mit dem ASCII-Code *Zahl* oder einen Text, der aus *n* Wiederholungen des *Strings* besteht

Auch dazu natürlich wieder ein Beispiel:
Wenn Sie die beiden folgenden Anweisungen im Direktmodus eingeben

```
A$="STRING$(5,7):PRINT A$
```

hören Sie fünf Piepstöne (das Zeichen mit dem ASCII-Code 7 ist auf dem Bildschirm nicht sichtbar, sondern läßt bei der Ausgabe einen Piepston erklingen).

Und wenn Sie die beiden folgenden Anweisungen im Direktmodus eingeben

```
A$="Maus":PRINT STRING$(10,A$)
```

erhalten Sie

```
MMMMMMMMMM
```

Wenn Ihnen inzwischen eigene Anwendungsmöglichkeiten eingefallen sind, steht Ihnen nichts mehr im Wege, das Programm entsprechend zu modifizieren und zu verwenden. Beachten Sie bitte die Warnung am Ende des Abschnitts 3.4, damit Sie nicht versehentlich bestehende Daten löschen. Noch ein paar Vorschläge für eigene Verbesserungen: Das Menü können Sie auch als Pull-down-Menü anlegen, wie wir es bei dem Zeichenprogramm PAINTAmiga gelernt haben. Bei längeren Texten sollte am Ende der Seite der Druck unterbrochen und eventuell eine Seiten-Numerierung vorgenommen werden. Sie sehen, Ihrem Tatendrang steht ein weites Betätigungsfeld offen.

3.6 Zahlensalat

Irgendwann, wenn Sie bereits viele Daten in einer Datei abgespeichert haben, wird auch bei Ihnen die Stunde kommen, in der Sie diese Daten ordentlich sortiert vor sich sehen wollen. Wie Sie sich bestimmt denken können, haben sich über dieses Problem schon etliche gescheite Leute den Kopf zerbrochen. Etliche Bücher sind allein über dieses Thema geschrieben worden. Aber Sie brauchen keine Angst zu haben, daß Sie den Inhalt dieser Bücher kennen müssen, um eine Datei zu sortieren. Allerdings sollten Sie wenigstens ein paar Grundkenntnisse des Sortierens haben, um diese bei Bedarf einsetzen zu können, oder wenn nötig auch auszubauen.

Für die folgenden Sortierversuche soll uns ein eindimensionales Feld mit 100 Elementen genügen. Um etwas zum Sortieren zu haben, werden zunächst den Elementen der Feldvariablen a() die Zahlen von 300 bis 100 zugewiesen – in umgekehrter Reihenfolge und in Zweierschritten. Um das Ganze etwas komplizierter zu machen, werden willkürlich 4 Werte verteilt, die außerhalb dieses Bereiches liegen.

```
REM sort
REM P.3.6-1
DIM a(100)
j=0
FOR i = 300 TO 100 STEP -2
  a(j)=i
  j=j+1
NEXT i
a(10)=1111
a(60)=5
a(70)=911
a(80)=2
```

Das Feld sieht also vor der Sortierung wie folgt aus:

300 298 296 294 292 290 288 286 284 282 1111 278 276 274 272 270
268 266 264 262 260 258 256 254 252 250 248 246 244 242 240 und so weiter.

Dieses Feld wird nun nach aufsteigenden Zahlen sortiert, wobei die Zeit, die der Amiga dazu benötigt, gestoppt wird. Letzteres kann der Amiga selbst am besten; Sie brauchen keine Stoppuhr hervorzuholen.

```
v1$= TIME$
GOSUB sortieren1
v2$=TIME$
WIDTH 60
FOR s1 = 0 TO 100
  PRINT a(s1);
NEXT s1
GOSUB zeit
PRINT
PRINT "Die Sortierung benoetigte";T7;"Sekunden"
END
```

In v1$ wird vor dem eigentlichen Sortiervorgang (in der Subroutine *sortieren1*) die aktuelle Zeit gespeichert und in v2$ dann nach dem Sortiervorgang die neue Zeit. Die Subroutine *zeit* bildet dann die Differenz und legt sie in T7 ab. Zur Feststellung der aktuellen Zeit verwendet das Programm die Funktion TIME$.

v$=TIME$
liefert die aktuelle Zeit als Zeichenkette der Form: "hh.mm.ss" (wobei "hh" die Stundenangabe, "mm" die Minuten und "ss" die Sekunden enthält)

In der Routine *zeit* werden aus diesem String dann die numerischen Werte für Sekunden und Minuten ermittelt. Hierzu werden die entsprechenden Teilstrings mit MID$ und RIGHT$ extrahiert und mit VAL in Zahlen umgewandelt.

v=VAL(String)
liefert den Wert, den *String* hat, wenn man ihn als Zahl auffaßt

Aus diesen beiden Zahlen kann dann problemlos die Differenz zwischen v1$ und v2$ in Sekunden ermittelt werden und damit erhalten wir die Zeit, die der Amiga zum Sortieren gebraucht hat. (Beachten Sie bitte, wie der Fall berücksichtigt wird, daß zwischen den beiden Zeiten ein Stundenwechsel liegt.)

```
zeit:
T1=VAL(RIGHT$(v1$,2))
T2=VAL(MID$(v1$,4,2))
T3=T2*60+T1
T4=VAL(RIGHT$(v2$,2))
T5=VAL(MID$(v2$,4,2))
T6=T5*60+T4
T7=T6-T3
IF SGN(T7)<0 THEN T7=(3600+T6)-T3
RETURN
```

Die erste Sortierroutine *sortieren1* ist noch sehr simpel. Es werden jeweils zwei benachbarte Zahlen verglichen. Ist die erste Zahl größer, so werden die beiden Zahlen getauscht und die größere kommt damit eine Stelle weiter nach hinten. Um es anders auszudrücken: Das Programm vergleicht a(0) mit a(1), a(1) mit a(2), a(2) mit a(3) und so weiter, bis die größte Zahl an letzter Stelle steht. Dann folgt der zweite Durchgang, in dem die zweitgrößte Zahl an die zweitletzte Stelle kommt. (Dieses Sortierprinzip wird auch Bubble-Sort genannt, weil dabei die großen Zahlen wie Luftblasen im Wasser aufsteigen.) Die Veränderung des Zahlenfeldes soll nun einmal am Beispiel dargestellt werden. Dabei sind die Schleifendurchgänge der inneren FOR-NEXT-Schleife der Routine *sortieren1* mit einem Stern (*) gekennzeichnet.

1. Durchlauf der äußeren Schleife:

1* 298 300 296 294 292 290 288 286 282 1111 ...
2* 298 296 300 294 292 290 288 286 282 1111 ...
3* 298 296 294 300 292 290 288 284 282 1111 ...
11* 298 296 294 292 290 288 286 284 282 300 278 1111 ...
...
100* 298 296 294 ... 106 104 102 100 1111

2. Durchlauf der äußeren Schleife:

...
100* 296 294 292 ... 104 102 100 911 1111

3. Durchlauf der äußeren Schleife:

...
100* 294 292 290 ... 102 100 300 911 1111

100. Durchlauf der äußeren Schleife:

...
100* 25 100 102 ... 296 298 300 911 1111

Nun aber die Subroutine *sortieren1* selbst, die dieses Sortierprinzip in Basic realisiert:

```
sortieren1:
FOR s2 = 0 TO 99
FOR s = 0 TO 99
 b=a(s)
 IF a(s)>a(s+1) THEN a(s)=a(s+1) : a(s+1)=b
NEXT s
NEXT s2
RETURN
```

Die erste Sortierroutine benötigt eine Zeit von etwa 60 bis 61 Sekunden für das zuvor angelegte Feld *a.*, und das ist zweifellos recht lange. Deshalb nun ein zweiter Versuch. Dieses Mal suchen wir uns die kleinste Zahl aus dem Feld *a* heraus und merken uns die Zahl in der Variablen *b* und die Position dieses Elements in der Variablen *c*. Dann werden alle Elemente unter der Position *c* eine Stelle nach oben geschoben, wodurch die unterste Position frei wird. An diese kommt dann die Zahl b (das kleinste Element). Im zweiten Durchgang der Schleife wird das Feld dann nur noch vom zweiten Element nach dem zweitkleinsten Element durchsucht – das kleinste ist schließlich schon gefunden – und dieses dann an die zweite Stelle im Feld gesetzt. Dieser Vorgang wird durch die äußere Schleife so lange fortgeführt, bis das letzte Element durchsucht wird.

1. Durchgang der äußeren Schleife:

1* 298 300 296 294 292 ...
2* 296 300 298 294 292 ...
3* 294 300 298 296 292 ...
...
100* 2 300 298 296 ... 108 106 104 102 100

2. Durchgang der äußeren Schleife:

...
100* 25 300 298 ... 110 108 106 104 102 100

3. Durchgang der äußeren Schleife:

...
100* 25 100 300 298 ... 110 108 106 104 102

100. Durchgang der äußeren Schleife:

...
100* 25 100 102 104 ... 298 300 911 1111

Die Routine sortieren2, die dieses Sortierprinzip realisiert, sieht dabei wie folgt aus:

```
sortieren2:
FOR d=0 TO 99
   b=a(d)
   FOR s= d TO 100
        IF a(s)<b THEN b=a(s): c=s
   NEXT s
   FOR t=c TO d+1 STEP-1
        a(t)=a(t-1)
   NEXT t
   a(d)=b
NEXT d
RETURN
```

Bevor Sie die zweite Sortierroutine ausprobieren, denken Sie bitte daran, daß Sie im Hauptprogramm die Zeile GOSUB *sortieren1* in GOSUB *sortieren2* umändern müssen. Sie werden dann feststellen, daß die Sortierung desselben Feldes *a* mit *sortieren2* nun nur noch 27 bis 28 Sekunden benötigt.

Auch das ist noch recht viel. Schaffen wir es ein zweites Mal, die Laufzeit zu halbieren? Die Subroutine *sortieren3* schafft es. Es handelt sich dabei um eine kleine Modifikation von sortieren2. In dieser Routine kostete die zweite innere FOR-NEXT-Schleife sehr viel Zeit. Dies ist aber überflüssig, weil es gar nicht nötig ist, die noch nicht sortierten Feldelemente eine Position nach oben zu schieben. Es genügt, das jeweils im noch nicht sortierten Teil gefundene größte Element nach vorne zu holen und an dessen alte Position das Element zu setzen, das dabei vorne verdrängt wurde. Die sich aus dieser kleinen Änderung ergebende Subroutine sieht dann wie folgt aus.

```
sortieren3:
FOR d=0 TO 100
b=a(d)
 FOR s= d TO 100
 IF a(s)=<b THEN b=a(s): c=s
 NEXT s
a(c)=a(d)
a(d)=b
NEXT d
RETURN
```

Und tatsächlich! Die Subroutine *sortieren3* schafft den Sortiervorgang in 14 bis 15 Sekunden. Im übernächsten Abschnitt wird sie dann auch wirklich praktisch eingesetzt.

3.7 Wer arbeitet, macht auch Fehler...

Ein wichtiger Gesichtspunkt bei der Programmierung eines Anwenderprogrammes ist die Berücksichtigung möglicher Eingabefehler. Ein Programm sollte bei einer falschen Eingabe nicht abstürzen können! Wenn Sie allerdings alles berücksichtigen wollen, was ein Anwender falsch machen kann, wird der Aufwand dafür ziemlich groß. Es lohnt sich aber immer, wenigstens die häufigsten Fehler abzufangen.

Auch bei der Fehlerbehandlung unterstützt Sie Amiga-Basic erheblich. Die Anweisungen ON ERROR, ERROR, RESUME und die System-Variablen ERR und ERL machen die Fehlerbehandlung fast zu einem Kinderspiel. Um die Problemlösung zu verdeutlichen, wird der Umgang mit diesen Befehlen natürlich wieder an einem Beispiel gezeigt. Am besten tippen Sie es zunächst einmal ein.

```
REM Fehler
REM P3.7-1
1 ON ERROR GOTO 20
2 CLS
3 PRINT "Geschwindigkeitsberechnung"
3 PRINT :PRINT
3 PRINT "Fuer Ende bitte >0< bei Wegstrecke eingeben"
PRINT :PRINT
5 INPUT "bitte Wegstrecke in km eingeben";w
6 IF w=0 THEN END
10 INPUT "bitte benoetigte Zeit in Minuten eingeben";k
IF k>24*60 OR k<0 THEN ERROR 151
12 k=k/60
13 g=w/k
14 PRINT "Ihre Geschwindigkeit betrug";k;"Km/h"
FOR i= 1 TO 10000 : NEXT i
GOTO 2
20 IF ERL=13 THEN 25
```

```
20 IF ERL=13 THEN 25
21 IF ERR=151 THEN PRINT "Zeit ausserer halb des gueltigen
Bereiches" ** : GOTO zeit
22 PRINT ERR
23 END
25 PRINT "Division durch 0 ist nicht gestattet"

zeit:
FOR i=1 TO 10000 : NEXT i
RESUME 2
```

Erschrecken Sie bitte nicht, wenn Sie die Zeilennumerierung sehen. Obwohl Amiga-Basic so etwas normalerweise nicht nötig hat, werden die Nummern in diesem Fall gebraucht. Zum einen möchte ich Ihnen noch einmal zeigen, daß es Amiga-Basic völlig egal ist, ob und wie die Zeilen numeriert sind, und zum anderen benötigen diese Programme für einen wichtigen Aspekt Zeilennummern. Wichtig ist dabei nur, daß keine Zeilennummer mehrfach vorkommt.

Schauen Sie sich das Programm einmal genauer an. In Zeile 1 steht die ON-ERROR-Anweisung:

ON ERROR GOTO Sprungmarke

sorgt dafür, daß beim Auftreten eines Programmfehlers zu der eingegebenen *Sprungmarke* verzweigt wird

Im Gegensatz zu den Unterbrechungs-Ereignissen (ON BREAK und ähnliche), muß diese Anweisung nicht extra aktiviert werden (BREAK ON). Sobald in einem Programm ein Fehler gefunden wird, verzweigt das Programm an die gewünschte Stelle, ohne die normalerweise erscheinende Fehlermeldung auszugeben! Sie können damit also auf alle vorhersehbaren Fehler reagieren. Ein Sonderfall dieser Anweisung ist ON ERROR GOTO 0, was die normale Fehlerbehandlung wieder in Kraft setzt.

Die folgenden Zeilen 2 bis 10 sind nicht weiter interessant. Das Programm fragt nach den Werten für Weg und Zeit. Jetzt aufgepaßt! In der nächsten Zeile simulieren wir einen Fehler. Sie haben richtig gelesen. Für jeden Programmschritt können Sie, wenn Sie Lust dazu haben, eine eigene Fehlermeldung ausgeben lassen:

ERROR Nummer

simuliert einen Programmfehler mit dem Fehler-Code *Nummer*

Sie können bei dieser Anweisung sowohl die von Basic definierten Fehler-Codes (Sie finden sie im Handbuch), als auch eigene Fehler-Codes verwenden. Diese müssen dann aber außerhalb der standardmäßigen Fehler-Codes (zum Beispiel über 150) liegen. In unserem Beispiel wird der Fehler-Code 151 definiert, wenn die eingegebene Zeit kleiner als 0

Minuten oder größer als 24 Stunden ist. Starten Sie einmal das Programm und geben für die Zeit den Wert minus 25 ein. Das Programm meldet sich dann mit der Meldung *Zeit ausserhalb des gueltigen Bereiches.*

Betrachten Sie nun das Programm ab Zeile 20, also der Zeile, zu welcher bei der Fehlerbehandlung verzweigt wird (ON ERROR GOTO 20). Zuerst wird gefragt, ob die System-Variable ERL=13 ist:

v=ERL
liefert die Nummer der Zeile, in der der letzte Fehler aufgetreten ist

Um diese Variable nutzen zu können, mußten im Programm Zeilennummern verwendet werden. Nur so kann der genaue Ort des Fehlers eingegrenzt werden. Wenn demnach in Zeile 13 ein Fehler aufgetreten ist, springt das Programm nach Zeile 25. Was passiert nun in Zeile 13? Wir berechnen die Geschwindigkeit, indem wir den Weg durch die Zeit teilen. Geben wir für die Zeit den Wert 0 ein, wird in der folgenden Formel durch 0 dividiert und Sie würden normalerweise die Fehlermeldung *Division by zero* erhalten und das Programm würde abgebrochen. Das Beispielprogramm würde nach Zeile 25 verzweigen und die Meldung *Division durch 0 ist nicht gestattet* ausgeben. Nach einer kleinen Zeitschleife, die dem Benutzer Gelegenheit zum Lesen des Textes gibt, finden Sie die Anweisung RESUME.

RESUME Sprungmarke
setzt ein Programm nach einem Fehler ab der genannten Sprungmarke fort

Das Beispielprogramm beginnt deshalb wieder in Zeile 2, ohne daß es – wie bei einem solchen Fehler sonst üblich – abgebrochen wird. Probieren Sie das am besten gleich einmal aus. In der Zeile 22 werden die Fehler abgefangen und deren Fehler-Codes ausgedruckt, welche im Programm nicht vorgesehen sind.

v=ERR
liefert den Fehler-Code des letzten aufgetretenen Fehlers

Eine Überprüfung von ERR sollten Sie in jeder Fehlerbehandlungs-Routine vornehmen. Bei unvorhergesehenen Fehlern sollten Sie ein Programm lieber abbrechen, bevor es zu Komplikationen kommt.

Mit diesem kurzen Beispiel haben Sie das Wichtigste über die Möglichkeiten der Fehlerbehandlung erfahren. Beim Lesen dieses Buches werden Sie aber noch öfter auf die eine oder andere Fehlerbehandlungs-Routine stoßen.

3.8 Meine Autos

Nachdem Sie sich in den letzten Kapiteln die Grundlagen für die Anwendung sequentieller Dateien erarbeitet haben, dürfen Sie Ihr geballtes Wissen nun in ein etwas größeres Projekt einbringen, eine Art Buchhaltung für Ihre Autokosten.

Eigentlich wäre es doch ganz schön, wenn Sie immer genau darüber Bescheid wüßten, was Ihr Auto tatsächlich an Kosten verursacht. Es gibt ja sogar Bücher, in denen man diese Kosten fein säuberlich nach Sparten getrennt eintragen kann und in mühseliger Rechenarbeit solche Daten erhält. Sie als stolzer Amiga-Besitzer werden aber solche aufwendigen Prozeduren nicht auf sich nehmen wollen. Sie lassen rechnen – den Amiga beziehungsweise ein Basic-Programm.

Sie haben in Ihrem Haushalt mehr als ein Auto? Sie haben sogar einen kleinen Fuhrpark? Macht nichts, auch dafür ist das Programm ausgelegt. Und selbstverständlich werden Sie die Daten auch sortiert ausgeben können. Da der Amiga über eine hervorragende Grafik verfügt, und weil es sehr übersichtlich ist, Zahlen grafisch darzustellen, wird auch diese Möglichkeit nicht fehlen. Wenn Ihnen bei der Beschreibung des Programms der Mund wäßrig geworden ist, können Sie gleich loslegen.

Wir beginnen zunächst mit dem soeben Erlernten und aktivieren am Anfang des Programms eine Fehlerbehandlungs-Routine. Neben der Dimensionierung auf 200 Datensätze (bitte bei Bedarf ändern) werden dann die beiden Zeiger *offen* und *sort* auf 0 (logisch falsch) gesetzt. Wenn später im Verlauf des Programms die gewählte Kfz-Datei benötigt wird, zeigt der Zeiger *offen* an, ob diese bereits eingelesen ist. Ähnlich verhält es sich mit *sort*. Er zeigt uns an, ob die aktuelle Datei bereits sortiert ist. Geben Sie nun das folgende Listing ein. (Denken Sie dabei daran, daß Zeilen mit zwei Sternen ** am Anfang zur vorhergehenden Zeile gehören und in einer Zeile zusammen mit dieser eingegeben werden müssen.)

```
REM KFZ-DATEI
REM P.3.8-1
CLEAR
ON ERROR GOTO fehler
offen=0
sort=0
COLOR 1,0
DIM tit$(11)
d=200
DIMtag$(d),monat$(d),jahr$(d),km$(d),liter$(d),preis$(d),
** kosten$(d),kostenart$(d)

fehler:
IF ERR <>53 THEN CLS: PRINT "Programmabbruch wegen
** Fehlernummer: "; ERR : RESUME ende
RESUME anlegen
```

Als nächstes folgt bei der Sprungmarke *start* das Öffnen der Kfz-Datei »name« zum Lesen. In der Datei »name« werden die Kraftfahrzeuge abgespeichert, über die Buch geführt werden soll. Wenn das Programm das erste Mal läuft, existiert diese Datei natürlich noch nicht. Der OPEN-Befehl liefert dann einen Fehler, wodurch das Programm zur Routine *fehler* und von dort nach *anlegen* springt. Dort wird dann die Datei »name« angelegt und in einer WHILE-Schleife die Namen der Fahrzeuge eingelesen. Die Variable j dient dabei als Zähler für die Anzahl der Namen. Ist kein Name gespeichert (j<2), wird der Anwender aufgefordert, diesen einzugeben. Wenn nur ein Name ermittelt wird (j<3), dann wird zum *menü* verzweigt. Bei mehreren Namen, also auch bei Dateien, wird dafür extra ein Auswahl-Menü erstellt.

```
start:
OPEN "name" FOR INPUT AS#2
CLS
j=0
WHILE NOT EOF(2)
  INPUT #2,Na$(j)
  j=j+1
WEND
men=j
CLOSE#2
IF j<2 THEN
  PRINT "Es existiert keine Datei! Waehlen Sie > neues
**Fahrzeug hinzufuegen <"
  FOR ti=1 TO 5000:NEXT ti
  menz=1
END IF
if menz then menz=0 : GOTO menue
IF j<3 THEN kfz$=Na$(1) : GOTO menue
END IF
FOR i=1 TO men
  tit$(i+1)=Na$(i)
NEXT i
tit$(0)=Na$(0)
tit$(1)=""
FOR i = men+1 TO 11
  tit$(i)=""
NEXT i
y2=(j+1)*16
GOSUB zeigen
dt=(y1-19)/16
kfz$=tit$(dt+1)
```

Kommt das Programm zur Sprungmarke *menue*, wird die Subroutine *menuezeigen* aufgerufen, die die zur Verfügung stehenden Funktionen zeigt. Je nach Eingabe des Benutzers wird dann zur gewünschten Funktion verzweigt. Bei *ende* wird geprüft, ob eine Änderung vorgenommen wurde. In diesem Fall erfolgt in der Subroutine *speichern* die Speicherung der geänderten Daten auf Diskette.

```
menue:
GOSUB menuezeigen
COLOR 1,0
me=(y1-19)/16
ON me GOTO eintragen,lesen,aendern,loeschen,ergaenzen,
**  verbrauchswerte,ende
ende:
IF offen THEN GOSUB speichern
END
REM SYSTEM
```

```
menuezeigen:
tit$(0)= "F A H R Z E U G M E N U E"
tit$(1)= "Fahrzeug "+kfz$
tit$(2)= "Datensatz eintragen"
tit$(3)= "Datensaetze lesen"
tit$(4)= "Datensatz aendern"
tit$(5)= "Datensatz loeschen"
tit$(6)= "neues Fahrzeug hinzufuegen"
tit$(7)= "Verbrauchswerte"
tit$(8)= "Ende"
FOR i = 9 TO 11
  tit$(i)=""
NEXT i
y2=9*16
GOTO zeigen
```

Wenn der Anwender dieses Programmes nur einen einzigen Datensatz eintragen will, sollte dieses natürlich möglichst schnell gehen. Deshalb stehen zwei Routinen zum Eintragen zur Verfügung. Die erste, *eintragen*, ist für den eben genannten schnellen Eintrag gedacht. Die Datei wird hier zum Anfügen (APPEND) geöffnet und die neuen Daten hinter die vorhandenen gehängt. Es wird dann sofort ins Menü zurückgesprungen, von wo aus der Benutzer das Programm beenden kann. Arbeitet man längere Zeit mit dem Programm (das Programm merkt es daran, daß die Datei bereits eingelesen wurde und in den dafür vorgesehenen Feldvariablen (siehe unten) zur Verfügung steht), so wird der neue Datensatz durch die Routine *eintragenoffen* in diesen Feldern gespeichert und nicht sofort auf die Diskette geschrieben. Die eigentliche Eingabe der Werte erfolgt in der Subroutine *eingabe*, die weiter unten noch beschrieben wird.

```
eintragen:
IF offen THEN eintragenoffen
OPEN kfz$ FOR APPEND AS #1
s=1
GOSUB eingabe
PRINT #1,tag$(s)
PRINT #1,monat$(s)
PRINT #1,jahr$(s)
PRINT #1,km$(s)
PRINT #1,liter$(s)
```

```
PRINT #1,preis$(s)
PRINT #1,kosten$(s)
PRINT #1,kostenart$(s)
CLOSE#1
sort=1
GOTO menue
```

eintragenoffen:

```
Satz=Satz+1
s=Satz
GOSUB eingabe
sort=1
GOTO menue
```

Wir kommen nun zur Subroutine *anlegen*, die weiter oben ja schon bei der Fehlerbehandlung erwähnt wurde. In dieser Subroutine wird die Datei »name«, die die Fahrzeuge enthält, über die Buch geführt werden soll, angelegt. Diese Datei wird als Datei#2 angelegt und geöffnet (#1 wird für die Kosten-Daten benötigt). Wenn Sie sich die Routine genauer ansehen, werden Sie feststellen, daß beim Anlegen zunächst nur ein Pseudo-Eintrag getätigt wird, der später gleichzeitig die erste Zeile für die Menüausgabe wird (siehe unten). Der tatsächliche Eintrag des ersten Fahrzeugs, wie auch aller weiteren Fahrzeuge, erfolgt im Anfügemodus.

anlegen:

```
CLS
CLOSE#1
PRINT "KFZ-DATEI WIRD ANGELEGT"
OPEN "name" FOR OUTPUT AS #2
  Na$="Uebersicht der Fahrzeugdateien"
  WRITE #2,Na$
CLOSE#2
GOTO 22
```

ergaenzen:

```
IF offen THEN GOSUB speichern
CLS
PRINT "KFZ-DATEI WIRD ERGAENZT"
22 OPEN "name" FOR APPEND AS #2
PRINT "Irrtum=0"
LINE INPUT "Bitte geben Sie das KFZ-Kennzeichen ein: ";Na$
IF Na$="0" OR Na$= "" THEN
  CLOSE#2
  GOTO start
END IF
PRINT #2,Na$
CLOSE#2
```

```
ersteintrag:
kfz$=Na$
OPEN kfz$ FOR OUTPUT AS #1
s=1
GOSUB eingabe
WRITE #1,tag$(s),monat$(s),jahr$(s),km$(s),liter$(s),
preis$(s),kosten$(s),kostenart$(s)
CLOSE#1
Satz=1
sort=0
GOTO menue
```

Das Löschen eines Datensatzes ist auch nicht weiter schwierig. Hierzu wird zunächst die Nummer des Datensatzes erfragt, der gelöscht werden soll. Weiß der Benutzer diese Nummer nicht, hat er die Möglichkeit, diese Funktion zu verlassen, um sie festzustellen. Das Löschen selbst findet dann nicht direkt in der Datei, sondern in den Feldvariablen statt, die die Datensätze aufnehmen. Hierzu wird die komplette Datei gegebenenfalls eingelesen. In einer FOR-Schleife holt sich das Programm dann immer den nächsthöheren Satz und speichert ihn, angefangen von dem Satz, der gelöscht werden soll, eine Stelle tiefer ab. Vorher wird die Variable, die die Anzahl der Sätze enthält (Satz), um eins reduziert.

```
loeschen:
IF offen THEN loeschenoffen
GOSUB einlesen

loeschenoffen:
CLS
INPUT"Bitte Satznummer (Irrtum=0): ",Saz
IF Saz=0 THEN menue
Satz=Satz-1
FOR s=Saz TO Satz
   r=s+1
   tag$(s)=tag$(r)
   monat$(s)=monat$(r)
   jahr$(s)=jahr$(r)
   km$(s)=km$(r)
   liter$(s)=liter$(r)
   preis$(s)=preis$(r)
   kosten$(s)=kosten$(r)
   kostenart$(s)=kostenart$(r)
NEXT s
sort=1
GOTO menue
```

Es folgt nun die *eingabe* eines Datensatzes. Die ersten vier Eingaben sind Zwangseingaben, das heißt, das Programm liest erst dann die fünfte Angabe, wenn die ersten vier Eingaben erfolgt sind. Sie können dieses Programmverhalten natürlich jederzeit abändern. Unbedingt erforderlich ist allerdings der vierte Eintrag (Kilometerstand),

da er zum Sortieren der Daten benötigt wird. Bei den restlichen Eingaben nimmt es das Programm nicht so genau. Wenn der Anwender hier nur RETURN drückt, wird der entsprechenden Variablen einfach "0" zugewiesen.

```
eingabe:
CLS
PRINT"bitte fuer Komma einen Punkt eingeben!!!"
COLOR 3
PRINT "Bei Irrtum bitte in >Art der Kosten< eine Null
** eingeben"
COLOR 1
LOCATE 5,1
INPUT"Tag.............. ",tag$(s)
IF tag$(s)="" THEN BEEP: GOTO eingabe
INPUT"Monat............ ",monat$(s)
IF monat$(s)="" THEN BEEP: GOTO eingabe
INPUT"Jahr............. ",jahr$(s)
IF jahr$(s)="" THEN BEEP: GOTO eingabe
INPUT"Kilometerstand. ",km$(s)
IF km$(s) = "" THEN BEEP: GOTO eingabe
INPUT"tanken liter... ",liter$(s)
IF liter$(s) = "" THEN liter$(s) = "0"
INPUT"tanken Preis... ",preis$(s)
IF preis$(s) = "" THEN preis$(s) = "0"
INPUT"sonstige Kosten DM",kosten$(s)
IF kosten$(s) = "" THEN kosten$(s) = "0"
INPUT"Art der Kosten... ",kostenart$(s)
IF kostenart$(s) = "0" THEN GOTO eingabe
IF kostenart$(s) = "" THEN kostenart$(s) = "0"
sort=1
RETURN
```

Zu den beiden folgenden Routinen *speichern* und *einlesen* gibt es eigentlich nichts zu sagen. Sie sollten Ihnen aus den vorangehenden Abschnitten schon gut bekannt sein.

```
speichern:
OPEN kfz$ FOR OUTPUT AS #1
FOR s=1 TO Satz
   WRITE #1,tag$(s),monat$(s),jahr$(s),km$(s),liter$(s),
   preis$(s),
   **    kosten$(s),kostenart$(s)
NEXT s
CLOSE#1
RETURN

einlesen:
OPEN kfz$ FOR INPUT AS #1
s=1
WHILE NOT EOF(1)
```

```
   INPUT #1,tag$(s),monat$(s),jahr$(s),km$(s),liter$(s),
   preis$(s),
   **  kosten$(s),kostenart$(s)
   s=s+1
WEND
CLOSE#1
Satz=s-1
offen=1
RETURN
```

lesen:

```
IF offen THEN lesenoffen
GOSUB einlesen
```

lesenoffen:

```
if sort THEN GOSUB sortieren
CLS
PRINT "Satz ";"Datum ";"Kilometer ";
PRINT " ";"liter";"Preis";
PRINT "l/100 km ";"Kosten ";
PRINT "Kostenart"
PRINT : a$=SPACE$(2)
s=1 : km$(0)="1"
WHILE s<=Satz
 k2=VAL(km$(s)):k1=VAL(km$(s-1)):li=VAL(liter$(s))
 vb=li/((k2-k1)/100)
 PRINT USING"###";s;:PRINT a$;
 PRINT USING"##.";VAL(tag$(s));
 PRINT USING"##.";VAL(monat$(s));
 PRINT USING"####";VAL(jahr$(s));:PRINT a$;
 PRINT USING"######";k2;:PRINT " ";
 PRINT USING"[]";liter$(s);preis$(s);
 PRINT USING"##.##";vb;:PRINT a$;
 PRINT USING"#####.##";VAL(kosten$(s));:PRINT a$;
 PRINT kostenart$(s)
 s=s+1
2 ta$=INKEY$ :IF NOT ta$=CHR$(32) THEN 2
WEND
PRINT :PRINT
PRINT "Ende der Eintragungen"
3 ta$=INKEY$ :IF NOT ta$=CHR$(32) THEN 3
GOTO menue
```

Der Programmteil *lesen* steht für das Anzeigen der einzelnen Datensätze am Bildschirm. Nach der Prüfung der beiden Zeiger *offen* und *sort* und dem eventuellen Einlesen der Datei in die Feldvariablen wird zunächst eine Kopfzeile auf dem Bildschirm ausgegeben. Dabei wird die Zeichenketten-Variable *a$* als aus zwei Leerstellen bestehend definiert, wozu die Funktion SPACE$ verwendet wird.

v$=SPACE$(Anzahl)
liefert einen String aus *Anzahl* Leerstellen

Danach werden die Daten formatiert ausgegeben. Die vielfältigste Anweisung für die formatierte Bildschirmausgabe ist PRINT USING.

PRINT USING Format;Liste
gibt die Ausdrücke in der *Liste* gemäß dem *Format* auf den Bildschirm

Die Möglichkeiten für die Formatierung von Texten und Zahlen mit *Format* sind so groß, daß dafür ein eigenes Kapitel nötig wäre. Da das Basic-Handbuch hier ausreichend Beispiele aufführt, beschränke ich mich auf die Beispiele im Programm.

Die PRINT-USING-Anweisungen stehen in einer WHILE-Schleife, die nach jedem Druck auf die Leertaste eine neue Zeile auf dem Bildschirm ausgibt. An erster Stelle wird die Satznummer (s) dreistellig ausgegeben (Format "###"). Hat die Zahl weniger als drei Stellen, werden diese rechtsbündig angezeigt. (Wenn sie mehr als drei Stellen enthält, wird die Formatierung hinfällig. Es werden dann alle Stellen mit einem %-Zeichen davor ausgegeben.)

Tag und Monat verwenden das Steuerzeichen "##." Es gilt das gleiche wie vorher (also zweistellige Ausgabe der jeweiligen Zahl), nur wird nach dem Wert ein Punkt angezeigt. Eine weitere Format-Variante finden Sie bei der Ausgabe der Variablen vb (Verbrauch pro Liter). Das Steuerzeichen "##.##" gibt zwei Stellen vor und nach dem Dezimalpunkt aus. Weitere Stellen nach dem Dezimalpunkt werden auf zwei Stellen gerundet. Falls der Wert kleiner als eins ist, wird links vom Dezimalpunkt eine Null (0) ausgegeben.

Sind alle Werte der Datei auf dem Bildschirm angezeigt, erscheint nach dem Betätigen der Leertaste (CHR$(32)) der Text für das Ende der Eintragungen. Nach einem weiteren Tastendruck springt das Programm zum Menü zurück.

Nicht ganz so einfach ist die Änderung eines Datensatzes. Sinnvollerweise sollten hierbei, während der Eingabe der neuen, die alten Werte erscheinen, die durch einfaches Drücken der RETURN-Taste übernommen werden können. Genau das wird im Programm auch realisiert.

Zuerst wird nach der Nummer des zu ändernden Datensatzes gefragt (Saz). Danach werden dieselben Texte wie bei der Eingabe ausgegeben, dahinter aber sofort der augenblickliche Wert des jeweiligen Feldes. Mit LOCATE wird der Cursor dann an den Anfang des alten Feldwertes gesetzt, und der Benutzer wird aufgefordert, diesen Wert zu überschreiben. Der eingegebene Text wird dann mit LINE INPUT eingelesen. Drückt der Benutzer statt einem neuen Wert nur RETURN, wird ein Leerstring ("") an LINE INPUT übergeben. Dies wird überprüft und die Leereingabe dann durch den (in merk$ zwischengespeicherten) alten Wert des Feldes ersetzt.

```
aendern:
IF offen THEN aendernoffen
GOSUB einlesen

aendernoffen:
CLS
INPUT"Bitte Satznummer (Irrtum=0): ",Saz
IF Saz=0 THEN menue
s=Saz : p = 22
CLS
PRINT"bitte fuer Komma einen Punkt eingeben!!!"
merk$=tag$(s)
LOCATE 5,1:PRINT"Tag..... ";" ";merk$;:LOCATE 5,p:
** LINE INPUT tag$(s)
IF tag$(s)="" THEN tag$(s)=merk$
merk$=monat$(s)
LOCATE 6,1:PRINT"Monat..... ";" ";merk$;:LOCATE 6,p:
** LINE INPUT monat$(s)
IF monat$(s)="" THEN monat$(s)=merk$
merk$=jahr$(s)
LOCATE 7,1:PRINT"Jahr..... ";" ";merk$;:LOCATE 7,p:
** LINE INPUT jahr$(s)
IF jahr$(s)="" THEN jahr$(s)=merk$
merk$=km$(s)
LOCATE 8,1:PRINT"Kilometerstand... ";" ";merk$;:LOCATE 8,p:
** LINE INPUT km$(s)
IF km$(s)="" THEN km$(s)=merk$
merk$=liter$(s)
LOCATE 9,1:PRINT"tanken liter..... ";" ";merk$;:LOCATE 9,p:
** LINE INPUT liter$(s)
IF liter$(s)="" THEN liter$(s)=merk$
merk$=preis$(s)
LOCATE 10,1:PRINT"tanken Preis..... ";" ";merk$;:LOCATE
** 10,p: LINE INPUT preis$(s)
IF preis$(s)="" THEN preis$(s)=merk$
merk$=kosten$(s)
LOCATE 11,1:PRINT"sonstige Kosten DM";" ";merk$;:LOCATE
** 11,p: LINE INPUT kosten$(s)
IF kosten$(s)="" THEN kosten$(s)=merk$
merk$=kostenart$(s)
LOCATE 12,1:PRINT"Art der Kosten... ";" ";merk$;:LOCATE
** 12,p: LINE INPUT kostenart$(s)
IF kostenart$(s)="" THEN kostenart$(s)=merk$
sort=1
GOTO menue
```

Der erste Programmteil, der nicht nur Daten speichert und ausgibt, sondern diese auswertet, ist *verbrauchswerte.* Diese Subroutine errechnet zuerst für jedes Jahr den maximalen Kilometerstand, summiert die verbrauchten Liter Treibstoff und die dafür

bezahlten Beträge und addiert auch alle sonstigen Kosten.

```
verbrauchswerte:
IF offen THEN verbrauchswerteoffen
GOSUB einlesen

verbrauchswerteoffen:
IF sort THEN GOSUB sortieren
CLS
s=1
ja=VAL(RIGHT$(jahr$(s),2)):kmanfang=VAL(km$(s))
datum=VAL(RIGHT$(jahr$(Satz),2)) :dat1=datum-(ja-1)
FOR j=1 TO dat1+1
  liter(j)=.1:preis(j)=.1:kosten(j)=.1:ges(j)=.1:ges1=.1:
** iter1=.1
NEXT j
Zaehler=1
zr=ja

DLF:
WHILE Zaehler
  j=zr-(ja-1)
  IF VAL(RIGHT$(jahr$(s),2))=zr THEN
  km(j)=VAL(km$(s))
  liter(j)=liter(j)+VAL(liter$(s)):preis(j)=preis(j)+VAL
  **(preis$(s)
  kosten(j)=kosten(j)+VAL(kosten$(s))
  ELSE
  Zaehler=0
  END IF
  s=s+1
  IF s>Satz THEN Zaehler=0
WEND
zr=zr+1
IF zr>datum THEN WTR
s=s-1
Zaehler=1
GOTO DLF
```

Anschließend geht es an den Aufbau eines Diagramms, das die ermittelten Werte in übersichtlicher Form darstellt. Zuerst werden die Beschriftungen ausgegeben und ein Gitternetz gezeichnet. Die einzelnen Jahreswerte, die *verbrauchswerte* errechnet hat, werden zu einem Gesamtwert summiert und errechnet. In einer FOR-Schleife werden dann die Werte auf dem Bildschirm ausgegeben und die Verbrauchskurve gezeichnet. Der durchschnittliche Verbrauch wird in Form von kleinen roten Rechtecken (als dicke gestrichelte Linie) zum Schluß gezeichnet. Ein Druck auf die Leertaste bringt den Benutzer dann wieder zurück ins Menü.

```
WTR:
CLS
PRINT "l/100 km":PRINT
FOR h=16 TO 2 STEP-2
  PRINT " "; h: PRINT
NEXT h
LOCATE 19,1
PRINT "Jahr"
PRINT "Verbrauch"
PRINT "Ges.Kosten"
PRINT "Kosten/km"
LINE (90,142)-(560,142),3
LINE (90,10)-(90,142),3
FOR h=120 TO 560 STEP 40
  LINE (h,10)-(h,142)
NEXT h
FOR h=20 TO 132 STEP 16
  LINE (80,h)-(560,h)
NEXT h
po=14
FOR i=ja TO datum
  LOCATE 19,po
  PRINT i;
  po=po+5
NEXT i
PALETTE 2,.3,.9,0
LOCATE 19,72:PRINT "Gesamt";
po=14 : km(0)=kmanfang : posiy(0)=142 : x4=80
FOR j=1 TO dat1
  liter1=liter1+liter(j)
  vb(j)=liter(j)/((km(j)-km(j-1))/100)
  ges(j)=preis(j)+kosten(j) :ges1=ges1+ges(j)
  koprokm(j)=ges(j)/(km(j)-km(j-1))
  LOCATE 20,po : PRINT USING"##.##";vb(j)
  LOCATE 21,po : PRINT USING"####";ges(j)
  LOCATE 22,po : PRINT USING"##.##";koprokm(j)
  posiy(j)=148-(vb(j)*8)
  IF posiy(j)>142 THEN posiy(j)=142
  LINE (x4,posiy(j-1))-(x4+40,posiy(j)),2
  po=po+5 : x4=x4+40
NEXT j
vb1=liter1/((km(dat1)-kmanfang)/100)
koprokm1=ges1/(km(dat1)-kmanfang)
LOCATE 20,72:PRINT USING"##.##";vb1
LOCATE 21,72:PRINT USING"#####";ges1
LOCATE 22,72:PRINT USING"#.###";koprokm1
posiy1=148-(vb1*8)
IF posiy1>142 THEN posiy1=142
```

```
IF posiy1<0 THEN posiy1=0
FOR h=80 TO 560 STEP 20
  LINE (h,posiy1-1)-(h+15,posiy1+1),3,bf
NEXT h
4 ta$=INKEY$ :IF NOT ta$=CHR$(32) THEN 4
GOTO menue
```

Nun ist das Programm weitgehend fertig und es bleiben nur noch einige Hilfsroutinen zu erklären. Die Unterroutine *zeigen* ist so ausgelegt worden, daß sie für die Darstellung beider verwendeten Menüs genutzt werden kann. In einem roten Rechteck zeigt sie zunächst die beiden Überschriften tit$(0) und tit$(1). Eine FOR-Schleife gibt dann bis zu neun Menüpunkte untereinander auf dem Bildschirm aus. Etwas versetzt erscheint darunter *Auswahl mit Leertaste*.

Nach dem Wechsel der Zeichenfarbe wird dann der Bereich links von den Menüpunkten gelöscht und links neben dem ersten Menüpunkt mit AREA und AREAFILL ein rotes Dreieck gezeichnet. Während das Programm auf eine Tastatureingabe wartet, springt es immer wieder zur Zeile 5 am Anfang der Subroutine. Dadurch wird der Hinweispfeil gelöscht und neu gezeichnet – er blinkt also. Durch die Farbänderung im Programmablauf wechselt außerdem der Schriftzug *Auswahl mit Leertaste* laufend die Farbe.

Drückt der Anwender nun auf die Leertaste, wird die Unterroutine *pfeil* angesprungen. Dort wird die Variable *y1* um 16 erhöht (der nächste Menüpunkt ist angewählt). Daraufhin springt auch der Hinweispfeil neben den nächsten Menüpunkt. Ist der letzte Menüpunkt erreicht (dies wird in der Variablen *y2* festgehalten), wird *y1* zurückgesetzt. Sobald der Anwender RETURN (CHR$(13)) betätigt, kehrt die Subroutine zum aufrufenden Programm (hinter der Sprungmarke *menue* am Anfang des Programms) zurück. Dieses berechnet aus *y1*, welcher Menüpunkt gewählt wurde, und ruft dann die Subroutinen für die verschiedenen Funktionen auf.

```
zeigen:
CLS
y1=35
ta=0
LINE (120,0)-(480,24),3,bf
COLOR 1
LOCATE 1,23: PRINT tit$(0)
LOCATE 3,29: PRINT tit$(1)
FOR i = 2 TO 10
  LOCATE 2*i+1,29
  PRINT tit$(i)
NEXT i
fa=1
5 LOCATE 22,29:COLOR fa:PRINT "Auswahl mit Leertaste";
```

```
fa=fa+2:
IF fa=5 THEN fa=1
FOR h = 5 TO 17 STEP 2
  LOCATE h,18
```

```
   PRINT " "
NEXT h
AREA(160,y1-4):AREA(160,y1+4)
AREA(190,y1):AREA(160,y1-4):AREAFILL 1
ta$=INKEY$:IF ta$="" THEN 5
IF ta$=CHR$(32) THEN GOSUB pfeil: GOTO 5
IF ta$=CHR$(13) THEN RETURN
GOTO 5
```

pfeil:

```
y1=y1+16
IF y1>y2 THEN y1=35
RETURN
```

Die letzte Routine unseres Programms ist für die Sortierung der Daten verantwortlich. Sie ist genauso aufgebaut, wie Sie bereits im Kapitel 3.6 gesehen haben. Da hier aber mehrere Felder parallel sortiert werden, sieht sie etwas unübersichtlich aus. Wenn Sie genau hinsehen, werden Sie aber feststellen, daß es sich im wesentlichen wirklich um die Routine sortieren3 handelt, die Sie bereits kennen. Sortierkriterium ist in diesem Fall übrigens der Kilometerstand, der mit *VAL(km$(s))* in eine Zahl umgewandelt wird – er wurde ja als Text abgespeichert.

sortieren:

```
sp=Satz+1
FOR dk=0 TO Satz
bk=VAL(km$(dk))
FOR s= dk TO Satz
   ak=VAL(km$(s))
  IF ak=<bk THEN bk=ak : c=s
   NEXT s
   km$(sp)=km$(c)
   km$(c)=km$(dk)
   km$(dk)=km$(sp)
   REM gefundenen Satz zwischenspeichern
   tag$(sp)=tag$(c):monat$(sp)=monat$(c):jahr$(sp)=jahr$(c):
   ** liter$(sp)=liter$(c)
   preis$(sp)=preis$(c):kosten$(sp)=kosten$(c):
   **  kostenart$(sp)=kostenart$(c)
   REM unteren Satz an Stelle des gefundenen Satzes
   tag$(c)=tag$(dk):monat$(c)=monat$(dk):jahr$(c)=jahr$(dk):
   ** liter$(c)=liter$(dk)
   preis$(c)=preis$(dk):kosten$(c)=kosten$(dk):
   ** kostenart$(c)=kostenart$(dk)
   REM zwischengespeicherten Satz nach unten
   tag$(dk)=tag$(sp):monat$(dk)=monat$(sp):jahr$(dk)=jahr$(sp):
   ** liter$(dk)=liter$(sp)
   preis$(dk)=preis$(sp):kosten$(dk)=kosten$(sp):
   ** kostenart$(dk)=kostenart$(sp)
NEXT dk
```

```
sort=0
RETURN
```

An dieser Stelle möchte ich Ihnen nur noch wünschen, daß Sie das Programm fehlerfrei abschreiben, und daß es Ihnen gefällt und Ihnen Anregungen für eigene Projekte verschafft. Mit diesem Wunsch wird dann auch die Behandlung der sequentiellen Dateien abgeschlossen, und wir wenden uns den »Direktzugriffs-Dateien« zu.

3.9 Direkter Zugriff – schnell und vielseitig

In diesem Kapitel werden Sie eine Art von Dateien kennenlernen, die für viele Anwendungsfälle weit besser geeignet ist als die sequentiellen Dateien. Direktzugriffs-Dateien bieten wesentlich kürzere Zugriffszeiten für beliebige Datensätze in einer Datei. Wenn Sie eine Direktzugriffs-Datei verwenden, können Sie einen beliebigen Datensatz aus der Datei (dessen Nummer Sie allerdings kennen müssen) einlesen, ändern und wieder abspeichern. Dazu ist es nicht nötig, alle davorliegenden Datensätze zu lesen. Sie brauchen wirklich nur den Datensatz einzulesen, den Sie gerade für Ihre Arbeit benötigen.

Der Aufbau einer Datei für den direkten Zugriff erscheint auf den ersten Blick komplizierter als der einer sequentiellen Datei. Wenn Sie ein bißchen in die Materie eingestiegen sind, werden Sie feststellen, daß der Schein trügt. Sie müssen nämlich nicht mehr überlegen, ob die Datei angelegt, ob Daten angefügt oder ausgelesen werden und den passenden OPEN Modus wählen. Es genügt statt dessen, die Datei am Programmanfang zu öffnen und am Schluß mit einem einfachen CLOSE sämtliche Dateien wieder zu schließen.

Bei so vielen Vorteilen hat die ganze Sache natürlich auch einen Haken! Bei jeder Daten-Operation benötigen wir bei der Direktzugriffs-Datei eine Datensatznummer. Diese Nummer brauchen wir, damit wir direkt auf die Daten zugreifen können. Das ist auch der Grund, warum Sie im privaten und geschäftlichen Bereich immer irgendwelche Nummern verfolgen. Da sich ein Mensch kaum diese ganzen Nummern merken kann, ist es Aufgabe der Programmierer, diese Nummern möglichst nun in einem Programm zu verwenden und im Programm ein Hilfsmittel zu schaffen, das eine Umsetzung der Nummern in eine verständlichere Form ermöglicht.

Der zweite Nachteil des direkten Zugriffs ist die Verschwendung von Speicherplatz. Die Datensätze in einer Direktzugriffs-Datei haben immer eine feste Länge, damit sie Basic problemlos finden kann. Diese feste Länge wird für jeden Datensatz reserviert, egal wieviel davon wirklich genutzt wird.

Wenn Sie eine Datei mit einer Satzlänge von 1024 Byte verwenden und einen Datensatz abspeichern, der nur 20 Byte enthält, bleiben in dieser Datei 1004 Byte ungenutzt und werden verschenkt!

Die eben erwähnte Satzlänge muß beim Öffnen einer Direktzugriffs-Datei bereits angegeben werden. Dies geschieht im OPEN-Befehl, den es für Direktzugriffs-Dateien nur in einer Form gibt (es gibt ja keine unterschiedlichen Zugriffsarten wie bei den sequentiellen Dateien).

OPEN"R",#Datei-Nummer,"Datei-Name",Datensatzlänge

Auf die Datensatzlänge kann in diesem Fall nicht, wie bei der sequentiellen Datei, verzichtet werden. Zusammen mit der Datensatznummer kann mit dieser Länge direkt die Position auf der Diskette gefunden werden, an der ein Datensatz gespeichert ist. Die Aufteilung der Daten innerhalb eines Datensatzes in einzelne Felder geschieht mit der FIELD-Anweisung:

FIELD #DateiNo, L1 AS Var1, L2 AS Var2,...
ordnet den Daten in einem Datensatz in der Datei mit der Kennung *DateiNo* eine Reihe von String-Variablen zu. Die ersten *L1* Buchstaben werden *Var1*, die nächsten *L2* Buchstaben *Var2* zugeordnet und so weiter. Die ersten *L1* Buchstaben des zuletzt gelesenen Datensatzes befinden sich nach einer solchen Anweisung immer in der Variablen *Var1*, die nächsten *L2* Buchstaben in *Var2* und so weiter

WICHTIG: Die Summe der einzelnen Feldlängen darf nicht länger sein als die Datensatzlänge, welche wir in der OPEN Anweisung festgelegt haben.

WICHTIG: Den Zeichenketten-Variablen aus einer FIELD-Anweisung darf im Programm kein anderer Wert zugewiesen werden (auch nicht über INPUT)!

Also: Nach der FIELD-Anweisung *FIELD#1,5 AS A$,7 AS B$* sind die folgenden Anweisungen nicht zulässig (ergeben allerdings keinen Fehler):

```
C$=A$
INPUT A$
LET D$=A$
```

Übertragen Sie nun das soeben Gelernte in ein Beispiel:

```
oeffnen:
REM Programm1
OPEN"R",#1,"Artikel",20
FIELD #1, 4 AS num$, 16 AS bez$
'RETURN
```

```
schreiben:
CLS
INPUT "Datensatz (0=Ende)";satz%
IF satz%=0 THEN END 'start
INPUT "Artikel-Nummer (4 Stellen): ";nu%
INPUT "Artikel-Bezeichnung: ";nam$
LSET num$=MKI$(nu%)
LSET bez$=nam$
PUT #1,satz%
GOTO schreiben
```

In der Routine *oeffnen* wird eine Direktzugriffs-Datei mit einer Datensatzlänge von 20 Byte geöffnet. Die Datensätze werden dann mit einer FIELD-Anweisung in zwei Felder aufgeteilt. Das erste (vierstellige) Feld wird eine Artikelnummer aufnehmen und mit der String-Variablen *num$* gekoppelt. Die restlichen 16 Byte sind für eine Artikelbezeichnung vorgesehen, die mit der String-Variablen *bez$* gekoppelt ist.

In der Routine *schreiben* wird dann zunächst die Nummer eines Datensatzes erfragt. Diese Nummer wird für alle Direktzugriffs-Operationen benötigt. Dann wird ein Datensatz eingegeben und den beiden Variablen *nu%* und *nam$* (nicht *num$* und *bez$*) zugewiesen. Mit der LSET-Anweisung werden die Daten der beiden Variablen dann an die FIELD-Variablen *num$* und *bez$* übergeben

LSET FIELD-Variable = Variable
legt den Wert von *Variable* linksbündig in der *FIELD-Variablen* ab, damit diese Daten mit der PUT-Anweisung in eine Direktzugriffs-Datei geschrieben werden können

Ähnliches tut auch die RSET-Anweisung. Diese Anweisung habe ich allerdings nur der Vollständigkeit halber mitaufgeführt. In den folgenden Programmen wird immer die LSET-Anweisung verwendet.

RSET FIELD-Variable = Variable
legt den Wert von *Variable* rechtsbündig in der *FIELD-Variablen* ab, damit diese Daten mit der PUT-Anweisung in eine Direktzugriffs-Datei geschrieben werden können

Damit numerische Werte, wie in diesem Beispiel *nu%*, in einer Direktzugriffs-Datei abgelegt werden können, müssen sie in eine Zeichenkette umgewandelt werden. Das wird in diesem Fall von *MKS$(nu%)* erledigt. (Die Umwandlung der verschiedenen Zahlenformate wurde bereits im Kapitel 2.7 ausführlich besprochen.)

In der vorletzten Zeile dieses Beispielprogramms wird der so gewonnene Datensatz mit der zugehörigen Datensatznummer auf die Diskette geschrieben.

PUT #Dateinummer,#Datensatznummer
schreibt den Inhalt der FIELD-Variablen als einen Datensatz in die Datei mit der Kennung #Dateinummer

Probieren Sie nun bitte das Programm aus und schreiben Sie einige Datensätze in die Datei. Das ist nötig, damit das folgende Programm, das das Einlesen von Datensätzen aus einer Direktzugriffs-Datei demonstriert, auch etwas zu lesen hat.

```
oeffnen:
OPEN"R",#1,"Artikel",20
FIELD #1,4 AS num$,16 AS bez$
REM RETURN

lesen:
REM Programm2
CLS
INPUT "Datensatz (0=Ende)";satz%
IF satz% =0 THEN END 'start
GET #1,satz%
PRINT
PRINT USING "Artikel-Nummer:#,###";CVI(num$)
PRINT
PRINT "Artikel-Bezeichnung: "bez$
FOR i=1 TO 10000: NEXT i
GOTO lesen:
```

Auch in diesem Beispiel wird zunächst die Datensatznummer erfragt und mit dieser dann der Datensatz von der Diskette geholt. Hierzu dient die Funktion GET.

GET #Dateinummer, Satznummer
liest aus der Datei mit der Kennung *Dateinummer* den Datensatz mit der Nummer *Satznummer* in die entsprechenden FIELD-Variablen.

Die so mit GET gelesenen Daten stehen nun in den FIELD-Variablen bereit und können mittels PRINT ausgegeben werden. Vergessen dürfen wir allerdings nicht, die numerischen Variablen wiederherzustellen. Und da das Schreiben und Lesen von Daten üblicherweise zusammengehören, werden wir nun die beiden Programmstücke zu einem Programm zusammenfassen.

```
REM Teile
REM P.3.9-1
GOSUB oeffnen

start:
CLS
PRINT "Lesen = l"
PRINT "Schreiben = s"
PRINT "Ende = e"

taste:
ta$=INKEY$:IF ta$=""THEN taste
IF ta$="s" THEN schreiben
IF ta$="l" THEN lesen
IF ta$="e" THEN ende
GOTO taste

ende:
CLOSE#1
END
Programm1
Programm2
```

Wo in diesem Programm *Programm1* und *Programm2* steht, müssen Sie nun noch die beiden ersten Programme dieses Abschnitts anhängen und können mit dem Programm arbeiten. In der Subroutine *oeffnen* aktivieren Sie bitte die RETURN-Anweisung, indem Sie das REM davor löschen. Außerdem soll in beiden Programmen die Zeile *IF satz% = 0 THEN END* in *IF satz% = 0 THEN start* geändert werden. Jetzt können Sie nach Herzenslust ein einfaches Programm zur Benutzung einer Direktzugriffs-Datei ausprobieren.

Zum Abschluß dieses Kapitels werden wir die wichtigsten Schritte, aus denen ein Programm besteht, das mit einer Direktzugriffs-Datei arbeitet, noch einmal kurz zusammenfassen:

Öffnen einer Datei für direkten Zugriff (Option "R")
Unterteilung des Datensatzes mit der FIELD-Anweisung
Schreiben:
Daten eingeben
Übertragen der Daten in die FIELD-Variablen mit LSET oder RSET
Daten PUT auf Diskette speichern
Lesen:
Einlesen der Daten in die FIELD-Variablen mit GET
Ausgeben oder Verarbeiten der Daten aus den FIELD-Variablen
Schließen der Datei mit CLOSE

Als grober Überblick soll diese Zusammenstellung zunächst genügen. Einige zusätzliche Feinheiten werden Sie im folgenden Abschnitt noch kennenlernen.

3.10 Terminkalender

Haben Sie schon einmal den Geburtstag Ihrer Frau oder Freundin bzw. Ihres Mannes oder Freundes vergessen? Erhielten Sie schon einmal eine Mahnung, weil Sie eine wichtige Zahlung übersehen hatten? Mit dem folgenden Programm können Sie solche Probleme vermeiden. Voraussetzung ist natürlich, daß Sie mit diesem Programm all Ihre Termine speichern und ab und zu einmal nachsehen, »was denn so anliegt«. Platz für Eintragungen ist jedenfalls reichlich vorhanden. Zu jedem Tag des Jahres können Sie 260 Zeichen eintragen. Und, wenn Sie wollen, können Sie sogar Termine eingeben, die erst nächstes Jahr oder noch später benötigt werden.

Nachdem Sie im letzten Kapitel gelesen haben, daß für jede Operation mit Direktzugriffs-Dateien eine Datensatznummer benötigt wird, fragen Sie sich nun vielleicht, wie Sie ohne solche Nummern bei dem Programm Terminkalender an die Daten herankommen sollen. Die Lösung für dieses Problem ist recht einfach: Zu jedem Termin gibt es eine Datumsangabe. Das Programm fragt nach diesem Datum und rechnet es in Kalendertage um. Diese werden dann zu Datensatznummern.

Eine Warnung vorweg: Das Programm legt für jedes Jahr, zu dem Sie Termine eintragen, eine eigene Datei an. Bei der Eingabe des Datums erfolgt eine Prüfung auf die Richtigkeit. (Jahreseingaben von 1986 bis 1999 sind möglich.) Für jeden Tag eines jeden Jahres wird in der entsprechenden Datei Platz für einen Datensatz reserviert (261 Byte). Wenn Sie für 14 verschiedene Jahre Einträge machen, sind dies 14 Dateien zu 95526 Byte oder etwa 93 Kbyte. Damit reicht die Kapazität einer Diskette schon nicht mehr aus. Für die Abspeicherung von Terminen ein oder zwei Jahre im voraus sollte aber genügend Platz auf einer Diskette sein.

Ein weiterer wichtiger Aspekt des Programms ist die Markierung von leeren Datensätzen. Diese etwas zeitaufwendige Prozedur ist für eine geregelte Datenverarbeitung unerläßlich. Dabei wird beim Anlegen einer neuen Datei jeder Datensatz zunächst mit einem speziellen Zeichen als unbelegt markiert (siehe unten).

```
REM Termin-Kalender
REM
ON ERROR GOTO fehler
DEFINT a-z
DIM gz(12)
CALL kopf
GOSUB grundzahl

start:
GOSUB datum
GOSUB aufloesung
IF jahr1=jahr THEN menue
GOTO pruefen
```

```
lauf:
CLOSE
ja$=STR$(jahr)
GOSUB oeffnen

menue:
CALL kopf
PRINT,, "lesen ..... bitte > l < druecken"
PRINT,, "schreiben ..... bitte > s < druecken"
PRINT,, "ende ..... bitte > e < druecken"

menue1:
ta$ = INKEY$
IF ta$ <>"" THEN ta$ = UCASE$(ta$) ELSE GOTO menue1
CALL kopf
IF ta$="L" THEN GOSUB lesen:GOTO menue2
IF ta$="S" THEN GOSUB schreiben:GOTO menue2
IF ta$="E"THEN ende
GOTO menue1

menue2:
jahr=jahr1
GOTO start
```

Im Programmteil *menue* werden wie üblich die zur Verfügung stehenden Routinen zur Wahl gestellt und dann zu ihnen verzweigt. Nachdem die gewünschte Funktion ausgeführt wurde, geht es über die Subroutine *menue2* dann zurück zum *start*.

```
ende:
CLOSE
END

datum:
REM Programm Nr.2
CALL kopf

datum1:
LOCATE 12,1
PRINT"Bitte geben Sie das gewuenschte Datum ein > TTMMJJ"
LOCATE 12,48:LINE INPUT datum$
tag= VAL(LEFT$(datum$,2))
IF tag>31 OR tag<1 THEN datum1
monat=VAL(MID$(datum$,3,2))
IF monat>12 OR monat<1 THEN datum1
jahr=VAL(MID$(datum$,5,2))
IF jahr>99 OR jahr<86 THEN datum1
RETURN
```

```
grundzahl:
FOR i = 1 TO 12
READ gz(i)
NEXT i
DATA 0,31,60,91,121,152,182,213,244,274,305,335
RESTORE
RETURN

aufloesung:
grza=gz(monat)
satz=grza+tag
RETURN
```

In der Routine *datum*, die gleich hinter *start* aufgerufen wird (siehe oben), wird zunächst ein Datum erfragt. Damit die Möglichkeiten für Eingabefehler verringert werden, wird der Tag daraufhin überprüft, ob er größer als 31 und kleiner als 1 ist, der Monat, ob er größer als 12 und kleiner als 1 ist. Die Jahre werden auf den Bereich von 1986 bis 1999 beschränkt. (Wenn Sie vermeiden wollen, daß die Kapazität Ihrer Diskette überschritten wird (siehe oben), können Sie die Grenzen für das Jahr nach Bedarf abändern.) Bei Eingabefehlern verzweigt das Programm zum Anfang der Routine.

Die Routine *grundzahl* wird einmal zu Beginn des Programmablaufs durchlaufen. Sie legt in der Feldvariablen *gz* ab, mit welchem Tag (vom Beginn des Jahres gezählt) die zwölf Monate eines Jahres beginnen. Dies wird vor allem dazu benötigt, im Programmabschnitt *aufloesung* die Satznummer des zu einem Datensatz gehörenden Datensatzes zu ermitteln. Hierzu wird die Tagesangabe im Datum auf die entsprechende Grundzahl (*gz*) des jeweiligen Monats addiert.

```
SUB kopf STATIC
CLS
IF z=0 THEN GOSUB init
PUT (0,0),tit%
LOCATE 2,57:PRINT "MARKT&TECHNIK VERLAG"
LOCATE 3,57:PRINT "by Horst-R. Henning"
LOCATE 12,1
GOTO zur

init:
x=20 : y=10
FOR i = 0 TO 5
  LINE (i,i)-(i+350,i)
  LINE -(i+350,i+50)
  LINE -(i,i+50)
  LINE -(i,i)
  LINE (i+120,i+25)-(i+300,i+25)
  LINE (x+i+15,y+i)-(x+i+15,y+30+i),3
  LINE (x+i,y+i)-(x+30+i,y+i),3
  FOR j = 0 TO 30 STEP 15
       LINE (x+50+i,y+j+i)-(x+80+i,y+j+i),3
```

```
   NEXT j
   LINE (x+50+i,y+i)-(x+50+i,y+30+i),3
NEXT i
LOCATE 3,20:PRINT "TERMINKALENDER"
DIM tit%(5158)
GET (0,0)-(355,55),tit%
z=1
RETURN
```

zur:

```
END SUB
```

In dem Unterprogramm *kopf* wird der Titelbildschirm des Programmes gezeichnet. Den größten Teil dieser Arbeit leistet die Routine init, die leider auch recht langsam ist. Deshalb verzeigt *kopf* nur dann zur Unterroutine init, wenn der Titel noch nie gezeichnet wurde (was in der Variablen *z* vermerkt wird). Ist der Titel einmal gezeichnet, wird er mit GET abgespeichert und kann später jederzeit mit PUT wieder auf den Bildschirm gebracht werden.

pruefen:

```
vorh$ = ""
OPEN "jahre" FOR INPUT AS #2
WHILE NOT EOF(2)
   INPUT #2,ja$
   IF VAL(ja$)=jahr THEN vorh$=ja$
WEND

CLOSE#2
IF vorh$<> "" THEN lauf
REM ergaenzen
ja$=STR$(jahr)
OPEN "jahre" FOR APPEND AS#2
   PRINT #2,vorh$
CLOSE#2
GOTO aktivieren
```

fehler:

```
IF ERR <>53 THEN CALL kopf: PRINT "Programmabbruch wegen
** Fehlernummer: ";ERR : RESUME ende
RESUME anlegen
```

Bei *start*, gleich zu Beginn des Programmes, wurde zur Unterroutine *pruefen* verzweigt. Sie wird hier erklärt. In *pruefen* wird die sequentielle Datei »jahre« geöffnet, in welcher die aktivierten Jahre abgespeichert werden, für die bereits Termine abgespeichert wurden (die also als Dateien auf der Diskette existieren). Ist diese Datei nicht vorhanden, wird (über ON ERROR am Programmanfang) in die Routine *fehler* verzweigt. Ist der Fehler-Code gleich 53 (File not found), wird zu *anlegen* verzweigt (siehe unten), wo eine Datei »jahre« erzeugt wird.

Wenn die Datei »jahre« jedoch schon existiert und ein Datum in einem Jahr eingegeben wurde, das hier noch nicht verzeichnet ist, wird dieses Jahr mit APPEND hinten an die bestehende Datei »jahre« angehängt und dann »aktiviert« (in der Routine *aktivieren*).

anlegen:

```
OPEN "jahre" FOR OUTPUT AS #2
 ja$=STR$(jahr)
 WRITE #2,ja$
CLOSE #2
```

aktivieren:

```
CALL kopf
PRINT,, "Das Jahr";jahr;"wird aktiviert"
GOSUB oeffnen
LSET da$=CHR$(0)
FOR n= 1 TO 366
   PUT #1,n
   LOCATE 15,1:PRINT ,,"Datensatz ";n
NEXT n
CLOSE #1
GOTO lauf
```

oeffnen:

```
CLOSE
nam$="Jahr"+ja$
OPEN"R",#1,nam$,261
FIELD#1,1 AS da$,65 AS ge$,65 AS er$,65 AS te$,65 AS so$
RETURN
```

In der Routine *aktivieren* wird zunächst eine neue Jahresdatei geöffnet und dann in jeden Datensatz diese Datei CHR$(0) (der Buchstabe mit dem ASCII-Code Null) als Anfangsmarkierung geschrieben. Dieser Vorgang dauert einige Minuten, ist aber zu verkraften, da jedes Jahr nur einmal so markiert werden muß. In der Routine *oeffnen,* die von *anlegen* aufgerufen wird, wird aber die neue Datei nicht nur angelegt und geöffnet. Die 261 Byte langen Datensätze der Jahresdatei werden zugleich mit einer FIELD-Anweisung in ein 1Byte (Anfangsmarkierung) und vier 65 Byte lange Felder aufgeteilt. Diese vier Felder sollen zur Aufnahme von Geburtstagen, Erinnerungen, Terminen und sonstigen Informationen zu einem Datum dienen.

lesen:

```
GET#1,satz
PRINT,, "Terminkalender vom ";datum$:PRINT
IF da$=CHR$(0) THEN PRINT "kein Eintrag vorhanden":GOTO
** taste
IF ge$="0" THEN ge$=""
PRINT "Geburtstage: ";ge$:PRINT
IF er$="0" THEN er$=""
PRINT "Erinnerungen: ";er$:PRINT
IF te$="0" THEN te$=""
```

```
PRINT "Termine: ";te$:PRINT
IF so$="0" THEN so$=""
PRINT "sonstiges: ";so$
GOSUB taste
RETURN
```

Die Routine *lesen* holt den Datensatz, dessen Nummer zuvor aus dem eingegebenen Datum berechnet wurde, von der Diskette in die FIELD-Variablen und bringt ihn dann mit einer Folge von PRINT-Anweisungen auf den Bildschirm. Beginnt der Datensatz mit CHR$(0), wird der Text "kein Eintrag vorhanden" angezeigt. Etwas schwieriger gestaltet sich das Schreiben eines neuen Termins auf die Diskette. Wenn ein neuer Satz einfach mit PUT auf die Diskette geschrieben wird, werden alle alten Daten in diesem Satz zerstört. Das wollen wir aber nicht. Außerdem wäre es natürlich beim Eintragen eines neuen Termins angenehm, zu sehen, was denn zu diesem Datum schon vermerkt ist. Schauen wir uns die Lösung dieses Problems etwas genauer an.

In der Routine *schreiben* wird zunächst, wie in *lesen,* der Datensatz mit GET in die FIELD-Variablen geholt. Ist der Satz noch nicht benutzt, also mit einem CHR$(0) versehen, werden alle Felder mit 0 gefüllt und der Satz dann in die Datei geschrieben (GOSUB 111), wobei das erste Byte mit CHR$(1) markiert wird. Dann wird die Routine *schreiben* erneut aufgerufen – wobei natürlich kein CHR(0) mehr am Anfang des Datensatzes steht. *schreiben* gibt daraufhin den alten Inhalt des Termins am Bildschirm aus. Mit der INSTR-Funktion wird dann versucht, noch freie Stellen im Terminkalender zu finden.

v=INSTR(*Start*, String,Such)
sucht und liefert als Ergebnis die Position, an der ein Suchtext *Such* das erste Mal in einem anderen Text *String* vorkommt (wird *Start* fortgelassen, beginnt die Suche beim ersten Buchstaben des Textes, sonst bei der ersten Position *Start*)

Hierzu ein Beispiel. Wenn Sie die folgende Zeile im Direktmodus eingeben:

```
a$="Anger 9":b$="9":c=INSTR(a$,b$):IF a THEN ?  a
**ELSE?"nicht gef."
```

erhalten Sie: 7

Schreiben sucht also nach zwei Leerzeichen, in der Annahme, daß danach kein Text mehr in dem untersuchten Feld steht. Die ersten Zeichen bis zu diesen Leerzeichen werden in der Variablen *ar$* gespeichert und auf dem Bildschirm ausgegeben. Dahinter kann der Anwender dann neuen Text eingeben. Dieser neue Text wird mit dem alten zusammen in einer von vier Text-Variablen *(geb$, eri$, ter$ und son$)* gespeichert und am Ende der Routine in die FIELD-Variablen gesetzt. Diese werden dann mit PUT gespeichert. Dieser Ablauf spielt sich einmal für die vier Felder jedes Datensatzes ab; Geburtstage, Erinnerungen, Termine und Sonstiges.

```
schreiben:
GET#1,satz
IF da$=CHR$(0) THEN geb$="0":eri$="0":ter$="0":son$="0":
** GOSUB 111 :GOTO schreiben
PRINT,, "Terminkalender vom ";datum$:PRINT
PRINT "Bitte maximal die jeweilige Zeilenlaenge eingeben."
su$=" ":xb=INSTR(ge$,su$):xc=xb+15:ar$=LEFT$(ge$,xb)
LOCATE 15,1:PRINT "Geburtstage "; ge$:LOCATE 15,xc:LINE
** INPUT geb$
IF LEFT$(ar$,1)="0" THEN ar$=""
geb$=ar$+geb$
IF geb$="" THEN geb$="0"
xb=INSTR(er$,su$):xc=xb+15:ar$=LEFT$(er$,xb)
LOCATE 16,1:PRINT "Erinnerungen: "; er$:LOCATE 16,xc:LINE
** INPUT eri$
IF LEFT$(ar$,1)="0" THEN ar$=""
eri$=ar$+eri$
IF eri$="" THEN eri$="0"
xb=INSTR(te$,su$):xc=xb+15:ar$=LEFT$(te$,xb)
LOCATE 17,1:PRINT "Termine: "; te$:LOCATE 17,xc:LINE INPUT
**ter$
IF LEFT$(ar$,1)="0" THEN ar$=""
ter$=ar$+ter$
IF ter$="" THEN ter$="0"
xb=INSTR(so$,su$):xc=xb+15:ar$=LEFT$(so$,xb)
LOCATE 18,1:PRINT "sonstiges: "; so$:LOCATE 18,xc:LINE
** INPUT son$
IF LEFT$(ar$,1)="0" THEN ar$=""
son$=ar$+son$
IF son$="" THEN son$="0"

111: LSET da$=CHR$(1)
LSET ge$=geb$
LSET er$=eri$
LSET te$=ter$
LSET so$=son$
PUT #1,satz
RETURN

taste:
te$=INKEY$: IF te$=""THEN taste
RETURN
```

Zum Schluß dieses Programms möchte ich Ihnen noch wünschen, daß Sie von nun an keinen Termin mehr versäumen. Wie schon am Anfang gesagt: das wird nur dann funktionieren, wenn Sie das Programm gelegentlich aufrufen und Ihre Termine eintragen und überprüfen.

4
Sprites, Bobs & Sound

Schon die Überschrift dieses Kapitels klingt ein wenig wie eine Zauberformel. Viele von Ihnen werden schon auf dieses Kapitel gewartet haben; vielleicht haben Sie schon ein wenig darin herumgeblättert. Denn gerade mit diesen Stichworten sind die speziellen Fähigkeiten des Amiga ganz besonders verbunden. Dabei sind diese Fähigkeiten durchaus nicht nur für die Spiele gedacht. Auch ernsthafte Einsatzmöglichkeiten dafür gibt es zuhauf. Das beste Beispiel hierfür ist der kleine Pfeil, der den Mausbewegungen folgt und ein Sprite ist.

Oder, was würden Sie von einem Frage- und Antwortspiel halten, in dem der Computer, dargestellt durch ein kleines Männchen, Ihnen Fragen stellt? »Fragen stellt« heißt dabei richtige Sprache und nicht nur Ausgabe von Text auf dem Bildschirm. Und wie wäre es, wenn Sie bei einem Eingabefehler auf das Mißgeschick angesprochen werden, oder wenn statt einer Meldung eine mündliche Erklärung abgerufen werden kann? Nun, der Amiga macht's möglich. Wie alle Amerikaner hat er zwar Schwierigkeiten mit der Aussprache fremder Sprachen, aber er kann sogar deutsch sprechen.

Freilich kann man die grafischen und musikalischen Fähigkeiten des Amiga nicht nur für nüchterne ernsthafte Aufgaben einsetzen. Eine passende musikalische Untermalung bei einem Abenteuerspiel kann die erwünschte Spielstimmung wesentlich steigern. Und was wäre schließlich der Krieg im Weltraum ohne die entsprechenden Sound-Effekte? Warum muß man vor dem Computer sitzen bleiben, wenn dieser mit einer längeren Aufgabe beschäftigt ist? Besser ist es doch, wenn man inzwischen eine Tasse Kaffee trinken kann und durch eine musikalische Tonfolge zurückgerufen wird, sobald die Arbeit beendet ist. Möglichkeiten dazu bietet der Amiga mit seinen vier Tonkanälen genug.

Das alles und noch einiges mehr können Sie mit Ihrem Amiga anstellen. Es wäre vermessen zu sagen, daß Sie die ganze Palette von Möglichkeiten, die im Amiga steckt, in diesem Kapitel wiederfinden werden. Darüber könnten noch viele Bücher geschrieben

werden. Einige Grundlagen zu diesem Thema, die sich in der Programmiersprache Basic verwirklichen lassen, finden Sie jedoch in diesem Kapitel.

4.1 Hallo – hier spricht der Amiga!

Recht ungewöhnlich für einen Computer ist, daß er sprechen kann. Noch besser ist es, daß dies sogar von Basic aus möglich ist, und daß beim Amiga nur ein einziger Befehl dafür benötigt wird! Das ist ja kinderleicht, werden Sie jetzt sagen. Das wäre es auch, wenn der Amiga nicht ein Amerikaner wäre. Und Amerikaner sprechen im allgemeinen recht schlecht deutsch (fast so schlecht, wie die meisten Deutschen englisch sprechen). Abhilfe kann hier der Befehl SAY schaffen, der gewissermaßen international spricht.

SAY Codes, *Modus*
gibt die Phonem-Codes im String Code als Sprache aus, wobei die Betonung vom Inhalt des numerischen Feldes *Modus* abhängig ist

Wenn Sie im Amiga-Basic-Handbuch unter »SAY« nachschlagen, finden Sie dort die Behauptung, daß mit dieser Anweisung »...verständliche Sprache beliebiger Nationalität...« ausgegeben werden kann. Das stimmt auch, da mit einer Lautschrift, auch phonetische Schrift genannt, sämtliche Sprachen der Erde darstellbar sind. Voraussetzung dafür ist allerdings, daß dabei auch sämtliche Sprachmöglichkeiten berücksichtigt werden, und das haben die Schöpfer der Amiga-Sprachsynthese leider nicht getan. Man sollte einmal ausprobieren, wie der Amiga chinesisch spricht. Betrachten wir uns nun die SAY-Anweisung, oder besser die Phonem-Codes, etwas genauer. Um etwas mit SAY zu sagen, muß man zunächst eine Zeichenkette bilden, die aus den Phonem-Codes zusammengesetzt ist, die den Text ergeben, den man hören möchte. Eine genaue Beschreibung dieser Phonem-Codes finden Sie im Anhang des Basic-Handbuches. Probieren Sie nun aber einfach die SAY-Anweisung aus, indem Sie den folgenden Befehl im Direktmodus eintippen. (Das System verlangt daraufhin eventuell, daß Sie die Workbench-Diskette einlegen.)

```
SAY "AY AEM AX KAA5MAXDOHR PERSINUL KUMPYUW5TER"
```

Hätten Sie bei der Eingabe gewußt, was danach aus dem Lautsprecher kommt?

```
I am a Commodore personal Computer
```

So einfach geht das. Sie meinen, daß es gar nicht einfach ist, für jeden Buchstaben oder jede Buchstabenkombination den richtigen Phonem-Code zu finden? Da stimme ich Ihnen voll und ganz zu! Aber auch die Programmierer von Amiga-Basic müssen dieser Meinung gewesen sein. Sie haben deshalb die TRANSLATE-Funktion geschaffen, die uns diese Arbeit abnimmt:

v$=TRANSLATE$(englischer Text)
liefert einen String, der die zum übergebenen *englischen* Text passenden Phonem-Codes enthält, der also diesen Text ausspricht, wenn man ihn SAY übergibt

Sie brauchen also nur einen englischen Text dieser Funktion anzuvertrauen und schon sind Sie die Sucherei nach den passenden Phonem-Codes los. Probieren Sie es doch gleich einmal aus:

```
REM englisch, translate
'P.4.1-1
a$=TRANSLATE$("If you have never used a computer before,
read
** all of this book.")
b$=TRANSLATE$("In addition, work through all the
** exercises.")
c$=TRANSLATE$("The best way to learn Basic is to use it.")
d$=a$+b$+c$
SAY d$
```

Sie sehen, TRANSLATE und SAY funktionieren auch mit längeren Texten einwandfrei. Laut Basic-Handbuch kann man mit dieser Funktion eine Phonem-Zeichenkette von bis zu 32767 Zeichen erzeugen. Das sind acht bis neun Schreibmaschinenseiten. Die Ausgabe eines solchen Textes dürfte den Amiga eine ganze Weile beschäftigen.

Falls er Ihnen nicht gefällt, können Sie aber auch den Tonfall, den Sie soeben bei der Sprachausgabe gehört haben, jederzeit nach Belieben verändern. Ob Sie eine Frauenstimme sprechen lassen wollen, oder die monotone Stimme eines Roboters bevorzugen, ob Sie eine gemächliche Sprechweise bevorzugen, oder lieber ein Stimmen-Stakkato hören, fast alles ist möglich. Um dies auszuprobieren, nehmen Sie unser letztes Beispiel und modifizieren Sie es ein wenig.

```
REM englisch2, translate, modus
'P.4.1-2
a$=TRANSLATE$("If you have never used a computer before,
** read all of this book.")
b$=TRANSLATE$("In addition, work through all the
** exercises.")
c$=TRANSLATE$("The best way to learn Basic is to use it.")
a%(0)=65    REM <110> Frequenz(65-320 Hz)
a%(1)=0     REM <0> Modulation, Betonung=0, Monoton=1
a%(2)=120   REM <150> Sprechgeschwindigkeit(40-400
** Worte/min.)
a%(3)=0     REM <0> 0=maennlich, 1=weiblich
a%(4)=22200      REM <22200> Stimmqualitaet(5000-22200)
a%(5)=64    REM <65> Lautstaerke(0-64)
a%(6)=10    REM <10> Tonkanal(0-11)
```

```
a%(7)=0      REM <0>Synchronisation 0=abwarten,1=parallel
a%(8)=0      REM <0> Steuerung fuer Synchronisation(0-2)
d$=a$+b$+c$
SAY d$,a%
```

Die Modi für die SAY-Anweisung sind in dem Beispiel den Feldern des Ganzzahlenfelds a% zugewiesen worden. Wenn Sie später eigene Programme entwickeln, können Sie die Zahlen genausogut in DATAs ablegen und am Programmanfang einlesen. In diesem Programm habe ich davon Abstand genommen, damit die einzelnen Werte leichter abgeändert werden können. Außerdem finden Sie neben den einzelnen Feld-Elementen deren Bedeutung für die Aussprüche als Kommentar. Die voreingestellten Werte, also die Werte die dann gelten, wenn Sie keinen Modus angeben, finden Sie in spitzen Klammern am Anfang der betreffenden Bemerkungen.

Sie können jetzt nach Herzenslust mit den verschiedenen Elementen des Modus herumprobieren. Ändern Sie einmal a%(3) in 1, a%(0) in 250 und a%(4) in 22350 um. Jetzt spricht eine Frauenstimme zu Ihnen. Geben Sie wieder die vorherigen Werte ein und weisen Sie a%(1) eine 1 zu. Der Text hört sich nun monoton, eben wie von einem Computer kommend, an. Bei der Änderung der Stimmqualität gehen Sie besser behutsam vor und weichen nicht sehr weit von dem voreingestellten Wert ab. Die Stimme hört sich sonst grauenhaft an. Werte unterhalb der Standard-Vorgabe hören sich dumpf und grollend und Werte darüber hoch und quäkend an. Wenn Sie eine Stereo-Anlage angeschlossen haben, können Sie natürlich auch die Einstellungen für die verschiedenen Kanalzuordnungen ausprobieren.

Nachdem Sie inzwischen, mehr oder weniger erfolgreich, Ihre Experimente mit der englischen Sprache abgeschlossen haben, wenden wir uns der deutschen Sprachausgabe zu. Wie bereits eingangs erwähnt, ist das ein Problem, das sich nicht mit einer einzigen Basic-Anweisung lösen läßt. Es bleibt Ihnen nichts anderes übrig, als die einzelnen Worte nach der Phonem-Code-Tabelle im Anhang H.11 des Basic-Handbuches zusammenzusuchen. Leider sind dabei nicht alle deutschen Laute vertreten, so daß Sie auf ähnlich klingende Laute ausweichen müssen. Es ist recht schwierig und zeitaufwendig, dem Computer verständliche Worte zu entlocken. Und um einen amerikanischen Akzent kommen wir dann immer noch nicht herum.

```
REM deutsch, modus
'P.4.1-3
a$="BIHDXEH DRUHKEHRX AYNQSHAXLTEHN" 'bitte Drucker ein-
** schalten
b$="IXZH BIHN AYN KUMPYUW5TER" 'ich bin ein Computer
a%(0)=65 'Frequenz(65-320)
a%(1)=0 'Modulation, Betonung=0, Monoton=1
a%(2)=120 'Sprechgeschwindigkeit(40-400)
a%(3)=0 '0=maennlich, 1=weiblich
a%(4)=22200 'Stimmhoehe(5000-22200)
a%(5)=64 'Lautstaerke(0-65)
a%(6)=10 'Tonkanal(0-11)
a%(7)=0 'Synchronisation 0=abwarten,1=weitere Bearbeitung
```

```
a%(8)=0 'Steuerung fuer Synchronisation(0-2)
c$=b$+a$
SAY c$,a%
```

Mit etwas Gewöhnung kann man den Text verstehen. Probieren Sie ruhig einige andere Werte für die Elemente des Feldes a% aus; vielleicht können Sie so die Sprachqualität verbessern. Das Ganze ist natürlich reine Übungssache. Wenn Sie sich einige wichtige Lautkombinationen notieren, können Sie damit schon eine Menge machen.

Versuchen Sie einmal, ob die TRANSLATE-Funktion vielleicht auch bei der deutschen Sprachausgabe nützlich sein kann.

```
REM deutsch2, translate, modus
P.4.1-4
a$=TRANSLATE$("bitt_a Drooker ayn shalten.")
b$=TRANSLATE$("ish bin ayin Computer.")
a%(0)=65 'Frequenz(65-320)
a%(1)=0 'Modulation, Betonung=0, Monoton=1
a%(2)=120 'Sprechgeschwindigkeit(40-400)
a%(3)=0 '0=maennlich, 1=weiblich
a%(4)=22200 'Stimmhoehe(5000-22200)
a%(5)=64 'Lautstaerke(0-64)
a%(6)=10 'Tonkanal(0-11)
a%(7)=0 'Synchronisation 0=abwarten,1=weitere Bearbeitung
a%(8)=0 'Steuerung fuer Synchronisation(0-2)
c$=b$+a$
SAY c$,a%
```

Wie Sie hören, ist auch dieser Text einigermaßen verständlich. Der ganze Trick dabei ist der, daß Sie versuchen müssen, einen Text zu bilden, der deutsch klingt, wenn ein Amerikaner ihn wie einen englischen Text (also fehlerhaft) ausspricht. Manchmal genügt schon die Änderung eines einzigen Wortes in einem Satz. Schreiben Sie einmal die nachfolgenden Sätze in die Text-Variablen a$ und b$:

```
"Der bower holt das Korn."
"Der Mowrer holt den Hammer."
```

Die beiden Sätze sind recht gut zu verstehen. Damit sind aber noch nicht alle Möglichkeiten der deutschen Sprachausgabe erschöpft. Die dritte habe ich bis zum Schluß aufgehoben, da sie etwas aufwendiger ist. Dafür können Sie diese bequem in eigene Programme einbauen. Außerdem brauchen Sie nicht vor jedem Wort, das gesprochen werden soll, die Workbench-Diskette einzulegen.

Wenn Sie im Amiga-Basic-Handbuch den Anhang H12 aufschlagen, finden Sie ein Programm für die deutsche Sprachausgabe. Um dieses Programm verwenden zu können, müssen Sie zuerst den Text, den Sie hören wollen, in die Datei test.txt eingeben. Für die Einbindung in ein Programm ist das nicht gerade ideal. Und wenn Sie die DATA-Anweisungen des Programms ansehen, werden Sie vielleicht schon den Fehlerteufel im Nacken spüren. Aber keine Sorge, wir lösen das viel eleganter.

Dieses Programm befindet sich nämlich auch in der Demos-Schublade auf Ihrer Amiga-Basic-Diskette. Der Name des Programms ist *speechd.* Machen Sie nun bitte eine Kopie dieses Programms und legen Sie es aus der »Demos-Schublade.« Ändern Sie dann den Namen in »deutsch« und öffnen Sie das Icon mit einem Doppelklick der Maus. Sie befinden sich dann wieder im Amiga-Basic. Es erscheint ein Requester, der Sie dazu auffordert, die Workbench-Diskette einzulegen. Diesen Wunsch erfüllen Sie aber nicht, sondern klicken statt dessen mit der Maus in das Feld Cancel. Daraufhin meldet sich Basic mit der Fehlermeldung *Illegal function call* und rahmt eine Programmzeile rot ein. Klicken Sie nun in das OK-Feld der Fehlermeldung und dann noch einmal ins Listing, damit die rote Umrandung verschwindet. Jetzt liegt das Programm vor Ihnen und Sie können es bearbeiten.

Zuerst löschen Sie nun den Teil des Programms, der sich mit der Tastatureingabe befaßt. Dieser Teil ist sehr umfangreich. Damit Sie nicht versehentlich zuviel löschen, sollten Sie sehr behutsam vorgehen. Zusätzlich vergleichen Sie bitte im Anhang H12 des Basic-Handbuchs, ob Sie den Beginn der Rest-Listings erreicht haben. Löschen Sie nun, beginnend mit dem Anfang des Programms, zuerst die Routine *start.* Es folgen die Routinen *hand* und *DrawControls.* Die letzten beiden Programmteile, die Sie löschen müssen, sind *HandleMouse* und *InitSpeech.* Jetzt haben Sie es fast geschafft. Der erste Programmteil, den Sie benötigen, ist *initdeutsch.*

ACHTUNG!! Dieses Programm jetzt noch nicht abspeichern! Sonst geht es Ihnen so, wie es mir ergangen ist. In diesem Programm steckt nämlich noch ein böser Fehler. Wenn Sie das Programm abspeichern und dann erneut laden, funktioniert es plötzlich nicht mehr. Es hat mich einige Stunden Arbeit und fast meinen Verstand gekostet, den Fehler zu finden. In den DATA-Anweisungen werden nämlich auch die deutschen Umlaute verwendet. Und das verträgt Amiga-Basic schlecht. Das hat der Programmierer offensichtlich vergessen. Jetzt, da Sie den Fehler kennen, ist es leicht, ihn zu beheben. Löschen Sie einfach in den DATA-Anweisungen *phoneme* die siebte Zeile von unten, die wie folgt aussieht:

```
DATA "",2, "H",ER, "",ER
```

Nun können Sie das Rumpfprogramm abspeichern. (Wegen urheberrechtlicher Bedenken wird es in diesem Buch aber nicht abgedruckt.) Wenn Sie in einem der folgenden Listings lesen *Hier bitte Restprogramm deutsch einsetzen,* ist damit dieses Rumpfprogramm gemeint, daß Sie jetzt vor sich sehen. Dieses Programm ist allein aber nicht lauffähig. Es folgt ein kurzes Programm, das diesen Zustand beseitigt.

```
REM Sprache
'P.4.1-5
REM deutsche Sprachausgabe

vorbereiten:
CLEAR,10000
CLEAR,90000&
GOSUB initdeutsch
```

```
REM ##########################################
REM hier können Sie Ihr Basic Programm einsetzen
REM ##########################################
```

sprache:

```
text$="Guten Tag, mein Name ist Amiga"
german text$,deutsch$
SAY deutsch$,how%
END
```

initdeutsch:

```
REM Hier bitte Restprogramm deutsch einsetzen
```

Jetzt sollte das Programm laufen. Bevor Sie es ausprobieren, speichern Sie es bitte unter dem Namen »deutsch« ab. Zur Demonstration, wie Sie das Programm in eigenen Anwendungen verwerten können, folgt nun ein Rechenprogramm für die vier Grundrechenarten. Damit es nicht zu aufwendig wird, rechnet es nur mit kurzen Ganzzahlen (mit der Endung %). Bitte erwarten Sie von der Sprachausgabe aber keine Wunder. Mir gefällt es, wie der Amiga spricht, – trotz aller Unzulänglichkeiten.

```
REM sprechende Rechenmaschine
'P.4.1-6
CLEAR,10000
CLEAR,90000&
ON ERROR GOTO fehler
GOSUB initdeutsch
CLS
text$="Guten Tag, mein Name ist Amiga"
text1$="Und wie heisst Du?"
GOSUB ausgeben
namen$=nam$
```

rech:

```
text$=namen$+", ich will mit Dir rechnen."
text1$="Bitte gib mir eine Rechenaufgabe:"
GOSUB ausgeben
IF nam$="Ende" THEN ende
ze%= VAL(nam$)
ze1$=STR$(ze%)
ze2=LEN(ze1$)
ze3$=MID$(nam$,ze2,1)
ze4%=VAL(MID$(nam$,ze2+1))
ze5=ASC(ze3$)
IF ze5<48 AND ze5>41 THEN
  erg%=0
  IF ze5=42 THEN erg%=ze%*ze4%:text1$="der Multiplikation
  ** lautet "
  IF ze5=43 THEN erg%=ze%+ze4%:text1$="der Adition lautet "
  IF ze5=45 THEN erg%=ze%-ze4%:text1$="der Subtraktion
  **lautet "
```

```
  IF ze5=47 THEN erg%=ze%/ze4%:text1$="der Division lautet "
ELSE
  erg%=0
END IF

ergebn:
IF erg%=0 THEN text1$="kann ich nicht berechnen."
text$=namen$+", das Ergebnis,"
GOSUB sprache
LOCATE 14,20:PRINT text$
text$=text1$
LOCATE 16,20:PRINT text$;ze%;ze3$;ze4%;"=";erg%
GOSUB sprache
text$="Wenn Du keine Lust mehr hast, gib bitte - Ende -
** ein."
CLS
LOCATE 10,20:PRINT text$
GOSUB sprache
GOTO rech

fehler:
IF ERR<>6 THEN text$="Fehler im Programm":GOSUB sprache:
RESUME ende
erg%=0:RESUME ergebn

ende:
text$="Auf Wiedersehen."
GOSUB sprache
END

ausgeben:
CLS
GOSUB sprache
LOCATE 10,20:PRINT text$
text$=text1$
GOSUB sprache
LOCATE 12,20:PRINT text$;:LINE INPUT nam$
RETURN

sprache:
german text$,deutsch$
SAY deutsch$,how%
RETURN

initdeutsch:
REM Hier bitte Restprogramm deutsch einsetzen
```

Zu Beginn wird wieder die schon bekannte ON-ERROR-Anweisung verwendet. Die Routine *rech* zerlegt dann die eingegebene Formel in ihre Bestandteile. Die Variable *ze%* erhält die linke Zahl, *ze3$* die Rechenart und *ze4%* die rechte Zahl. In einer IF-Anweisung wird das Ergebnis *erg%* berechnet und je nach Rechenart *text1$* ein Text

zugewiesen. Ist das Ergebnis außerhalb des zugelassenen Bereiches, kommt es zu einem Fehler. Diesen fängt die ON-ERROR-Anweisung ab und er springt zur Routine *fehler*. Dort erhält das Ergebnis den Wert 0. Diesen Wert bekommt *erg%* auch dann zugewiesen, wenn bei der Eingabe ein Fehler gemacht wurde. Bei *ergebn* spricht der Amiga dann das Ergebnis der Berechnung aus. Im Falle eines Fehlers sagt er: *Das Ergebnis kann ich nicht berechnen.*

Bei diesen Beispielen möchte ich es belassen, was Sie aber natürlich nicht davon abhalten soll, weiter mit der Sprachsynthese zu experimentieren. Probieren Sie doch einmal aus, ob der Amiga auch singen kann!

4.2 Mit Musik geht alles besser

Wenn man allgemein über Musikprogrammierung auf Computern spricht, meint man damit meist eine höchst komplizierte Angelegenheit. Beim Amiga-Basic ist das viel einfacher. Mit Hilfe der SOUND-Anweisung können Sie mit wenig Aufwand sogar ganze Lieder spielen.

SOUND Frequenz, Dauer, Laut, Kanal
gibt einen Ton der genannten *Frequenz, Dauer* und *Lautstärke* über den gewählten *Kanal* aus (vier Kanäle stehen zur Verfügung). SOUND »stößt« eigentlich die Klangerzeugung nur an und kehrt dann zurück zum Programm; der Ton erklingt noch, wenn das Programm längst mit anderen Dingen beschäftigt ist
SOUND WAIT
hält die Tonwiedergabe vorübergehend an
SOUND RESUME
setzt die mit SOUND WAIT gestoppte Tonwiedergabe fort

Da sich das etwas kompliziert anhört, fangen Sie nun ganz klein an und probieren Sie einen einzigen Ton aus. Geben Sie dazu im Direktmodus ein:

```
SOUND 261.63,18.2,127
```

Die erste Zahl stellt die Frequenz in Hertz (Hz) dar und darf im Bereich zwischen 20 und 15000 Hz liegen. 261.63 entspricht einem C auf der musikalischen Tonleiter. Die Dauer des Tons (in Zeittakten) bestimmt die zweite Zahlenangabe. (Eine Sekunde entspricht 18.2 Zeittakten.) Erlaubt sind Werte zwischen 0 und 77 Zeittakten. Die letzte Eintragung steht für die Lautstärke und darf zwischen 0 und 255 liegen. Da wir keinen Wert für den Kanal vorgeschrieben haben, gilt der Standardwert Null. Null bedeutet Ausgabe über den linken Tonkanal.

Sie sehen, es geht ganz einfach. Mit einem Wort und zwei bis vier Zahlen produziert man die schönsten Töne. Weil es so gut ging, wagen wir uns als nächstes gleich an eine längere Tonfolge.

```
REM Tonleiter
'P4.2-1
GOSUB freq
GOTO song

freq:
FOR i=0 TO 6    READ f(i)
NEXT i
DATA 1114,1250,1403,1487,1669,1873,2103
RETURN

song:
FOR j=0 TO 12
  READ fr
  READ da
  SOUND f(fr),da,127
NEXT j
END
DATA 0,12,1,12,2,12,3,12,4,12,5,12,6,12
DATA 5,12,4,12,3,12,2,12,1,12,0,12
```

Zunächst werden in die Feld-Variable f() die Frequenzen für die sieben Noten von C bis H gelegt. Diese werden einer DATA-Anweisung entnommen. In der zweiten Folge von DATA-Anweisungen stehen nebeneinander jeweils Dauer und Frequenz eines Tons. Das Programm erzeugt eine auf- und absteigende Tonleiter, die von der Routine *song* abgespielt wird. Beachten Sie dabei, daß der nächste Ton immer erst dann gespielt wird, wenn der vorhergehende sauber ausgespielt ist. Im folgenden Beispiel wird die Tonleiter noch weiter verbessert.

```
REM Tonleiter2
'P4.2-2
note$="CDEFGAHcdefgah"
DIM f(14)
GOSUB freq
GOTO song

freq:
FOR i=1 TO 14
  READ f(i) :REM Frequenzen einlesen
NEXT i
DATA 1114,1250,1403,1487,1669,1873,2103
DATA 2228,2500,2807,2974,3338,3746,4205
RETURN
```

```
song:
FOR j=1 TO 28
  READ fr$
  READ da
  po=INSTR(note$,fr$)          :REM Note suchen
  da=1.5*da                    :REM Notenlaenge in 1/16
  SOUND f(po),da,127
NEXT j
END

DATA C,8,D,8,E,8,F,8,G,8,A,8,H,8
DATA c,8,d,8,e,8,f,8,g,8,a,8,h,8
DATA h,4,a,4,g,4,f,4,e,4,d,4,c,4
DATA H,4,A,4,G,4,F,4,E,4,D,4,C,4
```

Die zweite Routine *song* kann nun die Töne C bis H und c bis h nach Buchstaben (und nicht nach Frequenzen) spielen, die aus DATA-Anweisungen gelesen werden. Diese Buchstaben werden zu Beginn in der Textvariablen *note$* abgelegt und in der Feldvariablen *f* die dazugehörigen Frequenzen. Auch die Tondauer braucht nun nicht mehr in Form von Zeittakten angegeben zu werden, sondern – wie in der Musik üblich – in Sechzehntelsekunden. Sie können mit diesem kleinen Programm also jede beliebige Melodie spielen lassen, wenn Sie nur eine Partitur mit den Noten und Längen vorliegen haben. Für ein längeres Musikstück brauchen Sie nur die DATA-Anweisungen und die Einleseschleife (die obere Grenze für *j*) zu verlängern. Probieren Sie doch gleich aus, wie das mit einer richtigen Melodie funktioniert.

```
REM Abendlied
'P4.2-3
note$="CDEFGAHP"
REM
GOSUB freq
GOTO song

freq:
FOR i=1 TO 8
  READ f(i) 'Frequenzen einlesen
NEXT i
DATA 130.81,146.83,164.81,185.0,196.00,220.00,246.94,0
REM
RETURN

song:
FOR j=1 TO 25
  READ fr$
  READ da
  po=INSTR(note$,fr$)'Note suchen
  da=.75*da 'Notenlaenge in 1/32
  SOUND f(po),da,127
NEXT j
END
```

```
DATA    F,16,G,16,F,16,H,16,A,16,G,32,F,16
DATA    A,15,P,1,A,15,P,1,A,15,d,16,c,16
DATA    H,32,A,15,P,1,A,15,P,1,A,15,P,1
DATA    A,16,H,16,A,16,G,40
```

Klingt doch schon ganz nett, oder? Gegenüber dem vorherigen Programm haben Sie sicherlich einige kleine Änderungen bemerkt. Zum ersten erscheinen die Frequenzen als einfachgenaue Zahlen im Listing und zum zweiten taucht eine Note »P« auf. Der Buchstabe P steht für Pause – wichtig für fast jedes Musikstück. P wird immer dann in die DATA-Anweisungen gesetzt, wenn zwei gleiche Töne aufeinanderfolgen. Der Amiga ist nämlich in einer Hinsicht etwas zu perfekt: aufeinanderfolgende Töne gleicher Höhe spielt er einfach ohne Pause hintereinander. Mit einer kleinen Pause dazwischen klingt eine Tonfolge eher nach einem normalen Instrument.

Als kleine Entspannungsübung können Sie jetzt Ihrer musikalischen Kreativität einmal freien Lauf lassen. Sicherlich finden Sie viel schönere Melodien als in den obigen Beispielen.

Zum Abschluß dieses Kapitels folgt nun aber noch ein Beispiel, das für zwei Tonkanäle ausgelegt ist.

```
REM Volksweise
'P.4.2-4
note$="CDEFGAHcdefgahP"
DIM f(15)
GOSUB freq

ablauf:
RESTORE noten
GOSUB song
GOTO ablauf

freq:
FOR i=1 TO 15
  READ f(i) 'Frequenzen einlesen
NEXT i
DATA 130.81,146.83,164.81,185.3,196.00,220.00,246.94
DATA 261.63,293.66,329.63,370.7,392.00,440.00,493.88,10
RETURN

song:
da=1
WHILE da
  SOUND WAIT
  FOR j=0 TO 1
      READ fr$,da,laut
      po=INSTR(note$,fr$)'Note suchen
      da=.75*da 'Notenlaenge in 1/32
      SOUND f(po),da,laut,j
  NEXT j
```

```
   SOUND RESUME
WEND
RETURN

noten:
DATA D,4,255,C,4,30
DATA D,4,255,D,4,30
DATA G,4,255,C,4,30
DATA G,3,255,D,3,30
DATA P,1,1,D,1,30
DATA G,4,255,C,4,30
DATA G,4,255,D,4,30
DATA D,4,255,C,4,30
DATA D,3,255,D,3,30
DATA P,1,1,D,1,30
DATA D,4,255,C,4,30
DATA D,4,255,D,4,30
DATA E,4,255,C,4,30
DATA E,3,255,D,3,30
DATA P,1,1,D,1,30
DATA E,4,255,C,4,30
DATA E,4,255,D,4,30
DATA D,4,255,C,4,30
DATA D,4,255,D,4,30
DATA P,4,1,C,4,30
DATA P,4,1,D,4,30
DATA D,4,255,C,4,30
DATA E,4,255,D,4,30
DATA F,4,255,C,4,30
DATA G,4,255,D,4,30
DATA c,4,255,C,4,30
DATA c,4,255,D,4,30
DATA D,4,255,C,4,30
DATA D,4,255,D,4,30
DATA G,4,255,C,4,30
DATA G,4,255,D,4,30
DATA P,16,1,P,16,1
DATA A,0,1,A,0,1
```

Wenn Sie etwas von Musik verstehen, werden Ihnen wahrscheinlich beim Abspielen dieses kleinen Liedes die Haare zu Berge stehen. Sinn und Zweck der Beispiele ist es jedoch nicht, Ihr Ohr zu verwöhnen, sondern Sie mit der Technik der Sound-Programmierung vertraut zu machen.

Das Programm ist so angelegt, daß es in einer Schleife das Lied wieder und wieder abspielt. Dafür verantwortlich ist die Routine *ablauf*. Mit RESTORE setzt sie den Zeiger zum Lesen der Noten auf das erste DATA-Element zurück und ruft dann die Routine *song* auf.

In *song* sehen Sie die beiden SOUND-Anweisungen SOUND WAIT und SOUND RESUME, welche oben bereits erwähnt, aber noch nicht besprochen wurden. SOUND WAIT steht zu Beginn der WHILE-Schleife. Diese Anweisung hält das Abspielen neuer Noten zurück. Die folgenden SOUND-Anweisungen legen dann neue Töne in einer Warteschlange ab, bis die Musik mit SOUND RESUME fortgesetzt wird. Dieser Trick sorgt dafür, daß alle Kanäle gleichzeitig anfangen zu spielen. Die Töne aus der Warteschlange können nun gespielt werden.

Die DATA-Elemente für jede Note enthalten in diesem Programm drei Werte: fr$ für die Tonhöhe, die Variable *da* zur Ermittlung der Spieldauer und schließlich die Variable *laut* für die Lautstärke. Die zweite Stimme in dem Programmbeispiel dient lediglich der rhythmischen Untermalung. Die DATA-Zeilen sind wieder so übersichtlich gestaltet worden, daß Sie ohne Probleme Ihre eigenen Musiknoten einsetzen können. Die Notenzahl ist vom Programm her nicht begrenzt. Beachten Sie bitte lediglich, daß Sie als letzte Note bei *da* den Wert Null einsetzen. Dieser Wert sorgt dafür, daß *da* logisch unwahr wird und die WHILE-WEND Schleife verlassen werden kann.

4.3 Töne – sanft gewellt oder eckig

Mit der SOUND-Anweisung können Sie schon allerhand anfangen. Mit der Frequenz legen Sie fest, wie oft die Wellen in der Sekunde schwingen und bestimmen so die Tonhöhe. Aber finden Sie es nicht auch recht langweilig, immer die gleiche Wellenform, also das gleiche Instrument, zu hören? Aber auch das kann man ändern. Mit der WAVE-Anweisung können Sie die Form der Wellen selbst bestimmen.

WAVE Kanal, Definition
legt die Wellenform für die Töne fest, die über einen Ton*kanal* erklingen

Die Nummer des Tonkanals wird hier genauso verwendet wie in der SOUND-Anweisung. Die Zahlen 0 und 3 gelten für den linken und die Zahlen 1 und 2 für den rechten Tonkanal. Als *Definition* für die Wellenform können Sie das Schlüsselwort SIN verwenden. Der entsprechende Tonkanal erzeugt dann Sinuswellen (die in etwa wie ein Klavier klingen). Da dieses der voreingestellte Standardwert ist, brauchen Sie ihn, solange Sie nur mit einem Tonkanal arbeiten, nicht anzugeben. Weitaus interessanter ist es, daß man statt des Schlüsselwortes SIN auch ein numerisches Feld mit 256 Elementen vorgeben kann. Jedes der Elemente diese Feldes kann einen ganzzahligen Wert von 127 bis minus 128 enthalten. Damit können Sie jede beliebige Wellenform realisieren und so auch den Klang vieler Musikinstrumente erzeugen!

Im weiteren Verlauf dieses Kapitels werden Sie einige andere Wellenformen kennenlernen. Daß diese nur einen kleinen Bereich des Möglichen abdecken, ergibt sich allein daraus, daß rein rechnerisch 256*256 Variationen möglich sind. Es bleibt Ihnen daher ein

weites Feld für eigene Experimente. Die erste Wellenform, die wir verändern werden, ist die Sinuswelle selbst. Sie kennen diese Welle sicher aus der Schule oder aus den Anzeigen von HiFi-Firmen.

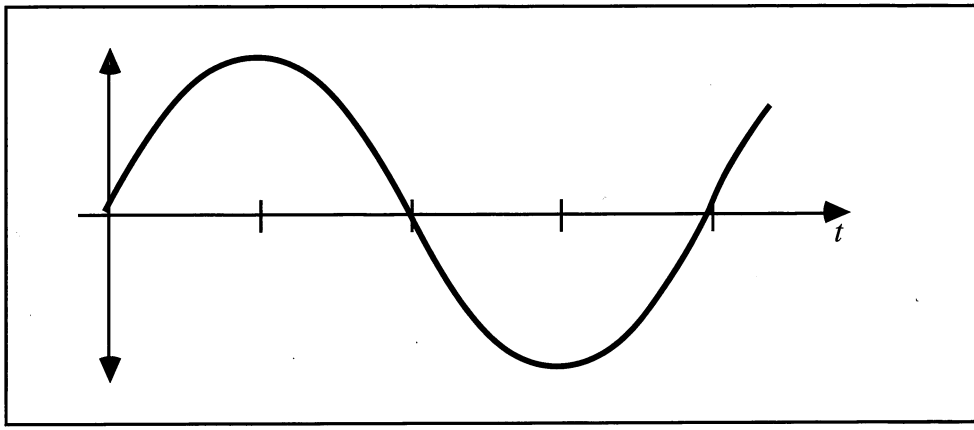


Bild 4.1: *Die Sinuswelle*

```
REM Sinuswelle
'P.4.3-1
DEFINT a-z
DIM we(255)
GOSUB sinus

ablauf:
FOR j!=20 TO 120 STEP 20
   faktor! = j! 'Schwingungsbreite max. 127
   GOSUB sinus
   FOR s = 300 TO 500 STEP 100
        SOUND s,12,255
   NEXT s
NEXT j!
END

sinus:
wert!=1.4118*3.1416/180
FOR i = 0 TO 255
   we(i)=INT(SIN(wert!*i)*faktor!)
NEXT i
WAVE 0,we
RETURN
```

Die Festlegung der Wellenform geschieht in der Routine *sinus*. Um Ihnen die Mühe zu ersparen, alle für die Form benötigten 256 Werte in DATA-Anweisungen einzugeben, berechnet das Programm diese Werte mit Hilfe der Basic-Funktion SIN und legt sie in der Feld-Variablen *we()* ab. (Beachten Sie, daß die Argumente der SIN-Funktion in

Bogenmaß und nicht in Grad angegeben werden.) Die Routine wird mit der WAVE-Anweisung beendet, die die in *we* abgelegte Wellenform dem Tonkanal Null zuweist.

Im Programmteil *ablauf* erhält die Variable *faktor* verschiedene Werte und sorgt so dafür, daß die Sinus-Schwingung verschieden weit ausschlägt. In der inneren FOR-NEXT-Schleife werden dabei jedesmal drei Töne gespielt. Wenn Sie das Programm einige Male abgespielt haben, werden Sie hören, daß bei einer engen Schwingung (kleinen Werten von *faktor*) der Ton dünn klingt. Wird die Schwingung vergrößert, klingt er immer voller. Der Klang entspricht bei voller Schwingung etwa dem einer Orgel, und bei einer engeren Schwingung klingt es etwa wie eine Flöte. Natürlich kommt es auch stark auf die Tondauer und die Frequenz an.

Die zweite Wellenform, die vorgestellt werden soll, ist die Dreieckswelle. Im Gegensatz zur Sinuswelle steigt sie geradlinig auf und ab.

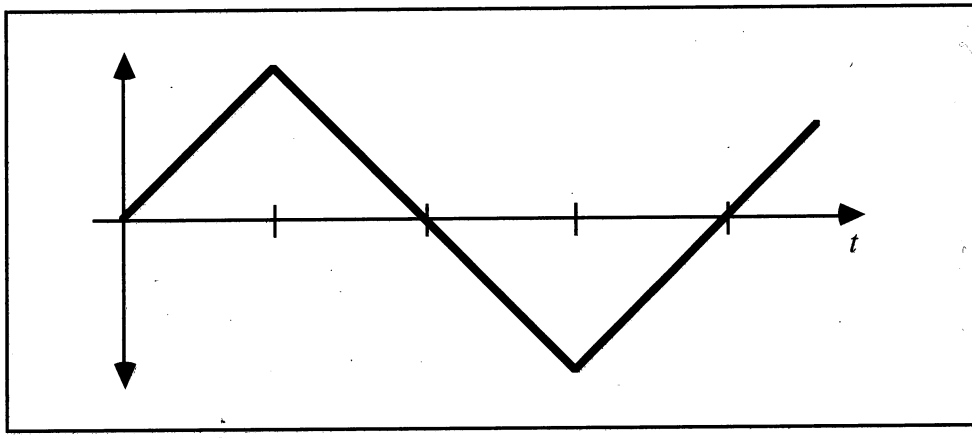


Bild 4.2: *Die Dreieckswelle*

```
REM Dreieckswelle
'P.4.3-2
DEFINT a-z
DIM we(255)

ablauf:
GOSUB drei
FOR j=300 TO 600 STEP 100
  SOUND j,12,255
NEXT j
END

drei:
FOR i=0 TO 63
  we(i)=2*i
NEXT i
FOR i=i TO 254 STEP 2
  we(i)=127-i
```

```
NEXT i
FOR i=0 TO 63
   we(i)=-127+(2*i)
NEXT i
WAVE 0,we
RETURN
```

Der Programmaufbau entspricht dem ersten Beispiel. Die Wellenform wird in der Routine *drei* festgelegt. Da die Welle kontinuierlich auf- und absteigt, ist keine komplizierte Berechnung wie bei der Sinuswelle nötig. Es werden dann vier Töne mit einer Tondauer von 12 Zeittakten gespielt. Im Vergleich zur Sinuswelle klingt der Ton nun hohler, etwa wie eine Trompete. Experimentieren Sie ruhig ein bißchen mit verschiedenen Tonhöhen. Verändern Sie die Tondauer und vergleichen Sie dabei die beiden Wellenformen.

Die nächste Welle hat auch eine dreieckige Form. Sie steigt aber nicht gleichmäßig auf und ab, sondern sanft an und steil (fast senkrecht) ab. Man nennt sie Sägezahnwelle.

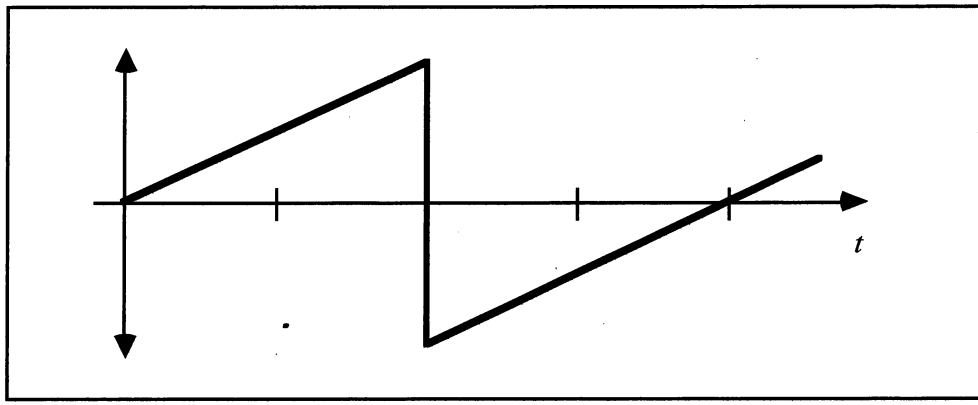


Bild 4.3: *Die Sägezahnwelle*

```
REM Saegezahnwelle
'P.4.3-3
DEFINT a-z
DIM we(255)

ablauf:
GOSUB saege
FOR j=300 TO 600 STEP 100
   SOUND j,12,255
NEXT j
END

saege:
FOR i=0 TO 127
   we(i)=i
```

```
NEXT i
FOR i=128 TO 1 STEP-1
  we(i)=-i
NEXT i
WAVE 0,we
RETURN
```

Wenn Sie die Töne der Sägezahnwelle mit denen der Dreieckswelle vergleichen, scheinen letztere mehr zu hallen. Auch die Sägezahnwelle sollten Sie sich im gesamten Freqenzbereich bei unterschiedlicher Tondauer anhören, damit Sie ein besseres Gefühl für die verschiedenen Anwendungsmöglichkeiten bekommen.

Eine weitere Wellenform ist die Pulswelle, oder auch variable Pulswelle genannt. Sie schwingt in Form eines Rechteckes. Die Breite des Rechteckes, die Pulsweite, können Sie dabei frei bestimmen.

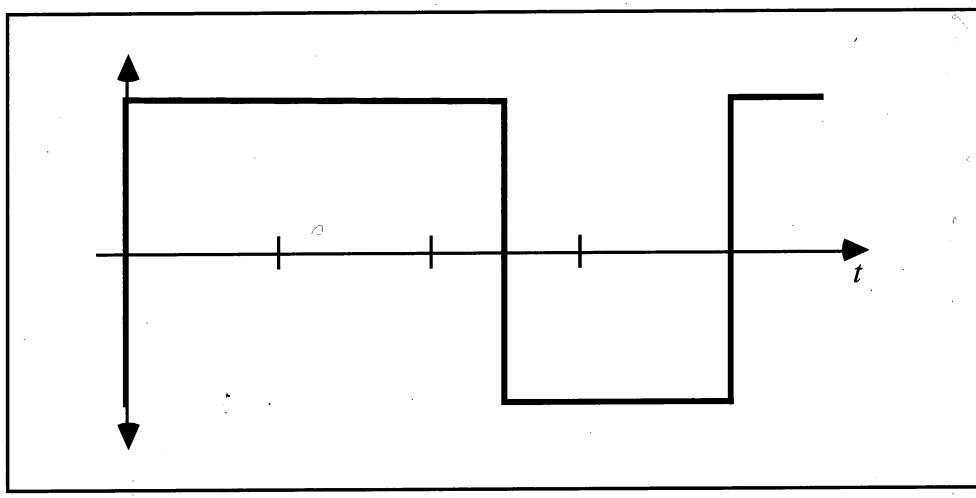


Bild 4.4: *Eine Pulswelle*

```
REM Pulswelle
'P.4.3-4
DEFINT a-z
DIM we(255)

ablauf:
FOR j=30 TO 110 STEP 20
  Faktor=j
  GOSUB puls
  FOR s=300 TO 600 STEP 100
       SOUND s,6,255
  NEXT s
NEXT j
END
```

```
puls:
b1=Faktor
b2=256-2*Faktor
FOR i=1 TO b1
  we(i)=-127
NEXT i
FOR i=1 TO b2
  we(i)=127
NEXT i
FOR i=1 TO b1
  we(i)=-127
NEXT i
WAVE 0,we
RETURN
```

Im Programmteil *ablauf* werden der Variablen *faktor* verschiedene Werte gegeben und so verschiedene Pulsbreiten festgelegt. Bevor die vier Test-Töne gespielt werden, wird in die Routine *puls* gesprungen, die die Wellenform (abhängig von *faktor*) festlegt. Wie Sie sehen, kommen in der Welle nur maximale (+127) und minimale (-127) Schwingungswerte vor. Je nach Pulsbreite ergeben sich verschiedene Klangfarben, die manchmal wie ein Spinett klingen, oft aber nur für Geräuscheffekte zu gebrauchen sind.

Die letzte Wellenform, welche in diesem Zusammenhang betrachtet werden soll, nennt man weißes Rauschen. Im Gegensatz zu den anderen Wellen ist hier die Wellenform (die Werte in den Feldern des Wellenform-Feldes) zufällig. (Die RND-Funktion, die hierzu verwendet wird, finden Sie in Kapitel 5.4 erläutert.)

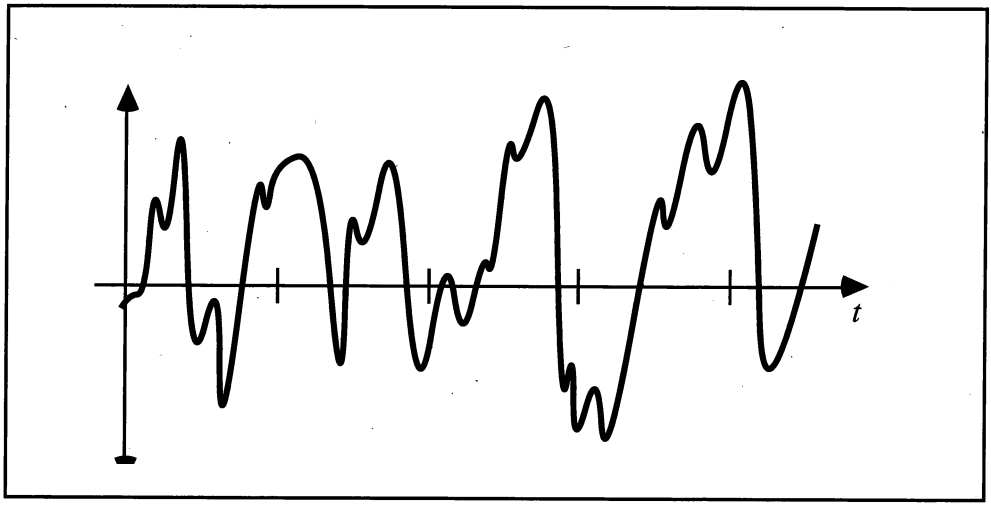


Bild 4.5: *Eine zufällige Wellenform (weißes Rauschen)*

```
REM Rauschen
P.4.3-5
DEFINT a-z
DIM we(255)
FOR j=40 TO 100 STEP 20
```

```
  Faktor=j 'max.127
  GOSUB rau
  FOR s=100 TO 600 STEP 100
  SOUND s,1,255
  NEXT s
NEXT j
END

rau:
REM weisses Rauschen
RANDOMIZE TIMER
FOR i=0 TO 63
  we(i)=2*i-INT(RND*Faktor)
NEXT i
RANDOMIZE TIMER
FOR i=1 TO 127 STEP 2
  we(i)=127-i-INT(RND*Faktor)
NEXT i
RANDOMIZE TIMER
FOR i=1 TO 127 STEP 2
  we(i)=0-i+INT(RND*Faktor)
NEXT i
RANDOMIZE TIMER
FOR i=0 TO 63
  we(i)=-127+(2*i)+INT(RND*Faktor)
NEXT i
WAVE 0,we
RETURN
```

Die mit dieser Wellenform erzeugten Töne klingen alle stark verzerrt. Sie erhalten so elektronisch klingende Geräusche und können Spezialeffekte realisieren. Durch die Variable *faktor* kann dabei wieder die Breite der Schwingungen verändert werden.

Sie haben nun die wichtigsten Wellenformen kennengelernt und konnten sich einen Eindruck von den verschiedenen Klangfarben verschaffen. Wichtig ist vor allem, daß Sie erkennen, daß dies nur ein kleiner Teil aus dem riesigen Spektrum von Möglichkeiten ist.

Nachdem in diesem Kapitel bisher Ihre Ohren nur mit Einzelbeispielen malträtiert wurden, soll zum Abschluß und zur Erholung ein kleines Liedchen erklingen. Damit die Programmierung auch nicht zu kurz kommt, wurde in jedem der vier Tonkanäle eine andere Wellenform verwendet. Herausgekommen ist eine alte Weise, die aber ganz und gar nicht alt klingt:

```
REM schwaebische Weise
P.4.3-6
note$="CDEFGAHcdefgahP"
DIM f(15)
DEFINT w
DIM we(256)
```

```
GOSUB freq
GOSUB sinu
GOSUB drei
GOSUB saege
b1=2:b2=252:GOSUB puls

ablauf:
RESTORE noten
GOSUB song
GOTO ablauf

freq:
FOR i=1 TO 15
  READ f(i) 'Frequenzen einlesen
NEXT i
DATA 130.81,146.83,164.81,185.3,196.00,220.00,246.94
DATA 261.63,293.66,329.63,370.7,392.00,440.00,493.88,0
RETURN

song:
da=1
WHILE da
  SOUND WAIT
  READ fr$,da,laut
  FOR j=0 TO 3
  po=INSTR(note$,fr$)'Note suchen
  da=.75*da 'Notenlaenge in 1/32
  SOUND f(po),da,laut,j
  NEXT j
  SOUND RESUME
WEND
RETURN

noten:
DATA D,11,255,P,1,1,D,3,255,P,1,1, D,7,255,P,1,1,
D,7,255,P,1,1
**DATA D,7,255,P,1,1,D,8,255,G,7,255,P,1,1,G,8,25
DATA E,11,255,P,1,1,E,3,255,P,1,1, E,7,255,P,1,1,
** E,7,255,P,1,1
DATA E,7,255,P,1,1, E,8,255, A,7,255,P,1,1, A,8,255
DATA H,12,255, A,4,255, H,8,255,  A,7,255,P,1,1
DATA A,8,255, G,8,255, D,15,255,P,1,1, D,7,255,P,1,1
DATA D,8,255, E,8,255, F,8,255, G,7,255,P,1,1
DATA G,7,255,P,1,1, G,16,255, D,7,255,P,1,1, D,7,255,P,1,1
DATA D,7,255,P,1,1, D,7,255,P,1,1, D,7,255,P,1,1, D,8,255
DATA G,16,255, E,7,255,P,1,1, E,7,255,P,1,1, E,7,255,P,1,1
DATA E,7,255,P,1,1, E,7,255,P,1,1, E,8,255, A,16,255
DATA H,12,205, A,4,205, H,8,205, A,7,205,P,1,1
DATA A,8,205, G,8,205, D,15,205,P,1,1, D,7,205,P,1,1
```

```
DATA D,8,205, E,8,205, F,8,205, G,7,205,P,1,1
DATA G,7,205,P,1,1, G,16,205, A,0,1

sinu:
WAVE 0,SIN
RETURN

saege:
FOR i=0 TO 127: we(i)=i: NEXT i
FOR i=128 TO 255:we(i)=i-255:NEXT i
WAVE 1,we
RETURN

drei:
FOR i=0 TO 63: we(i)=2*i: NEXT i
FOR i=1 TO 254 STEP 2: we(i)=127-i:NEXT i
FOR i=0 TO 63: we(i)=-127+(2*i): NEXT i
WAVE 2,we
RETURN

puls:
FOR i=1 TO b1:we(i)=-127:NEXT i
FOR i=1 TO b2:we(i)=127 :NEXT i
FOR i=1 TO b1:we(i)=-127:NEXT i
WAVE 3,we
RETURN
```

Bisher habe ich versucht, den verschiedenen Wellenformen einigermaßen melodische Töne zu entlocken. Ob mir das gelungen ist, müssen Sie entscheiden. Wenn Sie ein einigermaßen funktionierendes Musikgefühl haben, wird es Ihnen nicht schwerfallen, es besser zu machen.

Im nächsten Kapitel werden wir hingegen mit aller Gewalt und vereinten Kräften versuchen, dem Amiga ganz und gar unmelodische Töne zu entlocken.

4.4 Crash, Zoom & Wham – Soundeffekte

Ist es Ihnen auch schon so ergangen, daß Sie zu einem kleinen selbsterstellten Programm ein bestimmtes Geräusch suchten? Besonders bei der Programmierung von Spielen kommt man häufig in diese Verlegenheit. Mit diesem Kapitel möchte ich versuchen, diesen Mißstand etwas zu lindern. Es wird eine Anzahl von Geräuscheffekten vorgestellt, die Sie in eigene Programme einbauen können. Sie können allerdings nicht erwarten, daß Sie in diesem Kapitel für jeden Fall den passenden Sound finden. Sollte der richtige Soundeffekt nicht dabei sein, kommen Sie vielleicht durch Variation der Beispiele zur gesuchten Lösung.

Wenn Sie ein Geräusch mühevoll erarbeitet haben, oder eines der folgenden Programme verwenden, kann es trotzdem sein, daß es in Ihrem Programm nicht gut klingt. Es sind

völlig verschiedene Dinge, ob Sie so ein Geräusch solo laufen lassen oder zusammen mit einem anderen Programm. Solange Sie eine solche Geräuschfolge in einem Unterprogramm völlig separat ablaufen lassen und den Programmablauf nicht durch Unterbrechungsabfragen beeinflussen, kann nicht viel passieren. Viele Geräusche müssen aber parallel zu einem Ereignis ablaufen. Wenn Sie zum Beispiel in einem Spiel einen Laserschuß abfeuern, muß der Ton so lange präsent sein, wie die Spur des Schusses zu sehen ist. Sonst ist der ganze Effekt im Eimer. Da bleibt Ihnen nichts anderes übrig, als das Geräusch dem Programm-Timing anzupassen. Wenn Sie Glück haben, genügt es, den Wert für die Tondauer im SOUND-Befehl zu korrigieren. Wenn Sie Pech haben, steht Ihnen allerdings viel Arbeit bevor.

Die Wellenform schlechthin für die Programmierung von Geräuschen ist übrigens das (zunächst häßlich klingende) weiße Rauschen. Aber auch mit der Pulswelle lassen sich einige klangvolle Effekte verwirklichen. Im ersten Beispiel werden diese beiden Wellenformen für den typischen, durch Mark und Bein gehenden, hellen Klang einer Autohupe verwendet.

```
REM Hupe
'P.4.4-1

programmanfang:
DEFINT a-z:DIM we(255)
ru=120:GOSUB rau:WAVE 0,we:ERASE we

hupe:
SOUND 350,15,255,0
END

rau:
REM weisses Rauschen
RANDOMIZE TIMER
FOR i=0 TO 63:we(i)=2*i-INT(RND*ru):NEXT i
RANDOMIZE TIMER
FOR i=1 TO 127 STEP 2:we(i)=127-i-INT(RND*ru):NEXT i
RANDOMIZE TIMER
FOR i=1 TO 127 STEP 2:we(i)=0-i+INT(RND*ru):NEXT i
RANDOMIZE TIMER
FOR i=0 TO 63:we(i)=-127+(2*i)+INT(RND*ru):NEXT i
RETURN
```

Wenn Sie sich den Ton einige Male vorspielen, werden Sie feststellen, daß die Hupe jedesmal etwas anders klingt. Es bleibt dabei trotzdem immer eine Autohupe. Der Grund liegt in der Wellenform *rau*. Das Programm zwingt im Verlauf der Festlegung der Wellenform viermal die Zufallsfunktion RND, mit neuen Werten zu beginnen. Es ist daher in einem Programm völlig unmöglich, die Wellenform erst bei Bedarf zu generieren. Dazu braucht die Berechnung der Welle viel zuviel Zeit. Sie müssen deshalb die Wellenform bereits vorher berechnet und abgespeichert haben. Der beste Zeitpunkt dafür ist, wie in solchen Fällen immer, der Anfang des Programmes. Die Beispiel-Routinen sind daher in zwei Abschnitte aufgeteilt. Den ersten Abschnitt *programmanfang*

bauen Sie möglichst weit vorne in Ihr Programm ein. Die Routine, die das Geräusch erklingen läßt, können Sie dann an beliebiger Stelle verwenden. Und vergessen Sie bitte nicht, das Feld für die Wellenform mit einem %-Zeichen zu versehen.

Kommen wir nun zu unserem zweiten Beispiel in unserer kleinen Sammlung, *Alarmstufe1:*

```
REM Alarmstufe1
P.4.4-2

programmanfang:
DEFINT a-z:DIM we(255)
b1=64:b2=128:GOSUB puls:WAVE 0,we:ERASE we

alarmstufe1:
zeiger=9
WHILE zeiger=9
   SOUND 100,2,255,0
   SOUND 0,1,0,0
WEND
END

puls:
FOR i=1 TO b1:we(i)=-127:NEXT i
FOR i=1 TO b2:we(i)=127:NEXT i
FOR i=1 TO b1:we(i)=-127:NEXT i
RETURN
```

Sie hören das tiefe, fortlaufende Brummen eines Signalhornes. Man bezeichnet dieses Geräusch auch als »red alert«. Sie können es zum Beispiel dazu einsetzen, um ein Raumschiff in Alarmzustand zu versetzen, oder für die Alarmanlage bei einem Banküberfall. In diesem kleinen Programm ist ein Zeiger zur Programmsteuerung verwendet worden. Wenn Sie die Routine *programmanfang* einmal aufrufen, läuft sie so lange, bis sie durch ein anderes Ereignis (ON TIMER, ON COLISION) unterbrochen wird. Erst wenn *zeiger* ungleich 9 ist, hält sie an. Solange das Programm allein steht, können Sie es nur durch den Menü-Befehl *Stop* beenden.

Nun aber gleich das dritte Beispiel: Die Feuerwehr kommt!

```
REM Feuerwehr
'P.4.4-3

programmanfang:
DEFINT a-z:DIM we(255)
fakt=20:GOSUB raul:WAVE 0,we:ERASE we

feuerwehr:
wert = 8
WHILE wert=8
FOR j=700 TO 1000 STEP 10
   SOUND j,1,255,0
NEXT j
```

```
FOR j=1000 TO 700 STEP-10
  SOUND j,1,255,0
NEXT j
WEND
END
```

raul:

```
FOR d1=31 TO 255 STEP 32
RANDOMIZE TIMER
FOR i=0 TO d1/2:we(i)=127-INT(RND*fakt):NEXT i
RANDOMIZE TIMER
FOR i=d1/2 TO d1:we(i)=-127+INT(RND*fakt):NEXT i
NEXT d1
RETURN
```

Das Programm überrascht mit dem nervenzerreißenden Jaulen eines amerikanischen Feuerwehrautos. Die Tonerzeugung beruht auf dem stufenweisen (STEP 10) Herab- und Hinaufspielen der Frequenzen zwischen 700 und 1000. Das weiße Rauschen wird im Vergleich zum Programm Hupe in nur zwei Schleifen erzeugt. Außerdem ist der Ausschlag der Schwingungen auf eine Breite von 20 beschränkt.

Das nächste Programm verwendet wieder die Pulswelle.

```
REM Masch.Gewehr
'P.4.4-4
```

programmanfang:

```
DEFINT a-z:DIM we(255)
b1=1:b2=254:GOSUB puls
FOR n=1 TO 20
  fr=10
  ze=.9
  SOUND fr,ze,255,0
NEXT n
END
```

puls:

(Die Routine *puls* ist in dem Listing nicht mehr abgedruckt. Entnehmen Sie diese bitte dem Programm Alarmstufe1.) Wenn Sie das Programm laufen lassen, erklingt das kurze, trockene Knattern einer Maschinengewehrsalve. Der kurze hohle Klang wird dadurch erreicht, daß die Pulsbreite auf 1/254 der Dauer einer Welle herabgesetzt wird.

Im folgenden Programm haben wir die Pulsbreite wieder »normal« verwendet.

```
REM Abklang
'P.4.4-5
```

programmanfang:

```
DEFINT a-z:DIM we(255)
b1=64:b2=128:GOSUB puls:WAVE 0,we:ERASE we
```

```
abklang:
FOR n=40 TO 5 STEP-1
   fr=n*20
   ze=8/n
   SOUND fr,ze,255,0
NEXT n
END
```

puls:

Auch hier wurde der Programmteil *puls* nicht aufgelistet. (Wenn Sie in weiteren Programmen dieses Kapitels für eine Wellenform nur die entsprechende Sprungmarke, aber nicht die Subroutine finden, gibt es diese in einem anderen Programm und Sie müssen sie aus diesem kopieren.) Die erzeugte Tonfolge dieses Programms klingt recht effektvoll. Sie können sie für verschiedene Verwendungszwecke einsetzen. Ein Beispiel ist der Abschuß eines beliebigen Objekts in einem Schießspiel. Probieren Sie doch einmal aus, wie es klingt, wenn Sie die erste Zeile von *Abklang* wie folgt ändern:

```
FOR n=5 to 40
```

Klingt auch nicht schlecht, nicht wahr?

Auch das nächste Programm verändert die Frequenz und die Tondauer in einer bestimmten Folge:

```
REM Ufo-landet
'P.4.4-6

programmanfang:
DEFINT a-z:DIM we(255)
b1=16:b2=224:GOSUB puls:WAVE 0,we: ERASE we

ufo:
FOR n=100 TO 30 STEP-1
   fr=n*20
   ze=42/n
   SOUND fr,ze,255,0
NEXT n
END

puls:
```

Das Programm erzeugt eine recht eindrucksvolle Tonfolge, die vielleicht zur Untermalung eines landenden UFOs passen würde. Auch der Flug einer Rakete könnte durch diese Klänge begleitet werden. Das waren jetzt hintereinander drei Geräuschprogramme mit einer Pulswelle. Schauen Sie sich doch einmal die Variablen *b1* und *b2* an, die das Aussehen der drei Pulswellen bestimmen. Sie sehen, welche erstaunlichen Effekte durch die Variation der Pulsbreite erzeugt werden können.

Das folgende Programm verwendet hingegen wieder die Wellenform weißes Rauschen.

```
REM Motor
'P.4.4-7

programmanfang:
DEFINT a-z:DIM we(255)
ru=120:GOSUB rau:WAVE 0,we:ERASE we

motor:
zeiger =8
WHILE zeiger=8
  SOUND 20,2,255,0
WEND
END

rau:
```

Dieses Geräuschprogramm klingt einem schön rund laufenden Motor recht ähnlich. Sie sehen, daß auch mit einem minimalen Kernprogramm innerhalb der WHILE-Schleife eindrucksvolle Geräuscheffekte erzeugt werden können. Vergleichen Sie bitte einmal die einzelnen Werte mit denen im ersten Programm Hupe. Sie werden feststellen, daß außer der Frequenz und der Tondauer alle Parameter übereinstimmen. Ein besseres Argument für eigene Experimente gibt es doch gar nicht. Oder?

Für ein weiteres Programm wird wieder die simpelste Art der Wellenform, weißes Rauschen, eingesetzt.

```
REM Hochspannung
'P.4.4-8

programmanfang:
DEFINT a-z:DIM we(255)
GOSUB rau3:WAVE 0,we:ERASE we

hochspannung:
zeiger=3
WHILE zeiger=3
  SOUND 30,.5,100,0
WEND
END

rau3:
REM weisses Rauschen
FOR i=0 TO 255
  we(i)=INT(RND*127)
NEXT i
RETURN
```

Wenn Sie das Programm starten, ertönt das tiefe Brummen eines Hochspannungstransformators. Dieses Mini-Programm ist ebenfalls ein Beispiel dafür, wie mit geringem Aufwand erstaunliche Geräusche erzeugt werden können.

Die bisherigen Beispiele haben sich auf einen Tonkanal beschränkt. Sie können aber auch bis zu vier Wellenformen (in vier Kanälen) für Ihre Geräusch-Routinen festlegen. Es ist dann allerdings zu berücksichtigen, daß jeder Wellenform eine eigene Feld-Variable zuzuweisen und diese natürlich auch vorher zu definieren und mit Werten zu füllen ist.

Unser letztes Beispiel arbeitet auf die eben beschriebene Weise mit zwei Wellenformen gleichzeitig.

```
REM Laser
'P.4.4-9

programmanfang:
DEFINT a-z:DIM we(255):DIM we1(255)
b1=1:b2=254:GOSUB puls: WAVE 0,we :ERASE we
b1=8:b2=240:GOSUB puls1:WAVE 3,we1:ERASE we1

laser:
FOR n=1 TO 7
   fr=100:ze=4
   SOUND fr,ze,255,0
   SOUND fr,ze,255,3
NEXT n
END

puls:
REM Bitte hier einsetzen

puls1:
FOR i=1 TO b1:we1(i)=-127:NEXT i
FOR i=1 TO b2:we1(i)= 127:NEXT i
FOR i=1 TO b1:we1(i)=-127:NEXT i
RETURN
```

Das Programm erzeugt den für Aktion-Spiele typischen Klang eines Laserschusses. Da zwei verschiedene Pulswellen eingesetzt werden, sind alle Anweisungen bei *programmanfang* doppelt vorhanden.

Damit sind wir am Ende des Kapitels über die Geräusche. Daß die Programme nur einen kleinen Einblick in die vielfältigen Möglichkeiten des Amiga auf diesem Teilgebiet der Musikprogrammierung geben können, sagte ich Ihnen schon am Anfang. Einen kleinen Grundstock für Ihr Archiv mit Sound-Effekten haben Sie damit erhalten. Es liegt jetzt an Ihnen, soweit Sie Lust dazu haben, es kräftig auszubauen und zu verbessern. Je besser Ihr Archiv sortiert ist, um so seltener kommen Sie in die Verlegenheit, erst im Falle eines Falles herumprobieren zu müssen.

Ich hoffe, daß Sie an den Beispielen sehen konnten, daß schon geringe Änderungen bestehender Routinen völlig andere Geräusche hervorbringen können. Experimentieren Sie also ausgiebig mit verschiedenen Zeiten, Frequenzen und Wellenformen. Probieren Sie verschiedene Wellen auf mehreren Kanälen mit unterschiedlichen Parametern aus. Sie müssen sich auch nicht auf die Pulsform und das weiße Rauschen beschränken. Vielleicht bringen Ihnen die Sinuswelle, die Dreieckswelle oder die Sägezahnwelle den Erfolg.

4.5 Bewegte Bilder – Sprites & BOBs

Ein besonders faszinierender Aspekt der Computergrafik ist die Arbeit mit bewegten Bildern (der Fachmann sagt »Animation« dazu). Bisher hatten Sie dafür nur die PUT-Anweisung verwendet. Jetzt werden Sie eine große Anzahl neuer Befehle zu diesem Thema kennenlernen, die kaum Wünsche offen lassen. Trotz ihrer großen Anzahl ist die Handhabung der meisten dieser Anweisungen kinderleicht.

Amiga-Basic unterscheidet zwei Arten von bewegten Objekten:

- Sprites, die durch spezielle Hardware bewegt werden.
- BOBs, die (hauptsächlich) durch Software bewegt werden.

Wir werden uns in diesem Abschnitt zunächst mit den Sprites beschäftigen. Die Daten, die das Aussehen eines Sprites bestimmen, werden in einer String-Variablen abgelegt und können auch in einer sequentiellen Datei gespeichert werden. Diese Daten enthalten alle Angaben über das Objekt, wie Länge, Breite, Form und so weiter (aber nicht die Position des Sprites). Sie brauchen sich aber keine Gedanken darüber zu machen, wie Sie diese Informationen in eine Datei bekommen. Diese Arbeit nimmt Ihnen weitgehend der »Objekt-Editor« ab, ein Programm, das Sie in der Schublade »BASICDemos« finden.

Für die Einfärbung der Sprites stehen Ihnen vier Farben zur Verfügung. Die erste Farbe wirkt allerdings später als transparent, weshalb ein Sprite effektiv nur 3 Farben besitzen kann. Diese drei Farben müssen aber nicht mit denen aus der Palette des aktuellen Bildschirms identisch sein. Sie können auch mehr als vier Farben im Vierfarben-Modus der Workbench darstellen. Die Anzahl der darstellbaren Sprites ist auf acht begrenzt.

Öffnen Sie nun bitte die Schublade »BASICDemos« und starten Sie das Programm »ObjEdit« durch einen Doppelklick auf das entsprechende Icon. Das Programm fragt Sie dann nach einer kleinen Pause:

```
Enter 1 if you want to edit sprites
Enter 0 if you want to edit bobs >
```

Geben Sie eine 1 (für die Erzeugung von Sprites) ein und drücken Sie RETURN. Der Zeichenbereich läßt sich nur in der Höhe verstellen (y), da die Breite für Sprites auf 16 Pixel fest eingestellt ist (y=15). Wählen Sie gleich die Vergrößerung *4x4* aus dem *Enlarge*-Menü, da Sie sonst beim Setzen der Punkte kaum etwas erkennen können. Die anderen Werkzeuge aus dem *Tools*-Menü können Sie bei der Sprite-Konstruktion weitgehend vergessen, da sie in der geringen Breite kaum einsetzbar sind. Zeichnen Sie nun ein Sprite; irgend etwas, was Ihnen gerade in den Sinn kommt. Wählen Sie anschließend *Save As* aus dem *Tools*-Menü und speichern Sie das Sprite unter dem Namen »test« ab.

Verlassen Sie nun das Programm und schließen Sie die Schublade BASICDemos. Öffnen Sie sie dann wieder und nehmen Sie das Icon »test« heraus. Legen Sie es in eine Schublade, in der Sie Ihre Programme abspeichern (oder in eine neue Schublade). Geben Sie dann das folgende Programm ein und speichern Sie es in derselben Schublade unter dem Namen »sprite-test« ab.

```
REM Sprite-Test
REM P.4.5-1 OBJECT.SHAPE OBJECT.X OBJECT.ON OBJECT.VX
PALETTE 0,.5,.8,1
OPEN"test"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE 1
OBJECT.X 1,580
OBJECT.Y 1,100
OBJECT.ON
```

Starten Sie nun das Programm (nachdem Sie es abgespeichert haben). Wie Sie sehen, genügt ein solches Miniprogramm, um das erste Sprite auf den Bildschirm zu bringen. Die PALETTE-Anweisung in der zweiten Zeile sorgt dafür, daß das Sprite deutlicher sichtbar wird. Ein schwarzes Sprite hebt sich nun einmal besser vom Hintergrund ab. Betrachten Sie nun das Programm etwas näher. Nach dem Öffnen der Datei »test«, die die Sprite-Daten enthält, finden Sie die folgende neue Anweisung.

OBJECT.SHAPE Objektnummer, String
erzeugt ein neues Objekt (Sprite oder BOB) mit der Kennung *Objektnummer* und entnimmt die dazu nötigen Informationen dem übergebenen *String*

OBJECT.SHAPE Objektnummer1 , Objektnummer2
erzeugt ein neues Objekt (Sprite oder BOB) mit der Kennung *Objektnummer1* und entnimmt die dazu nötigen Informationen dem schon existierenden Objekt mit der Kennung *Objektnummer2*

Mit der OBJEKT.SHAPE-Anweisung werden zunächst alle Daten des Sprites wie Größe, Form und so weiter an das Programm übergeben. Als nächstes wird mit zwei neuen Anweisungen festgelegt, wo das Sprite auf dem Bildschirm erscheinen wird.

OBJECT.X Objektnummer, X
OBJECT.Y Objektnummer, Y
bewegt ein Objekt (Sprite oder BOB) mit der Kennung *Objektnummer* so, daß danach dessen linke obere Ecke die X-Koordinate *X* beziehungsweise die Y-Koordinate *Y* hat (wobei die jeweils andere Koordinate nicht geändert wird)

Die genannten Werte für die X- und Y-Koordinaten haben wie üblich als Bezugspunkt die linke obere Ecke des Bildschirmes. Diese Anweisungen bewegen das Sprite zwar, machen es aber noch nicht sichtbar. Hierzu dient der letzte neue Befehl in diesem Programm, OBJECT.ON.

OBJECT.ON Objektnummer1, Objektnummer2,...
OBJECT.OFF Objectnummer1, Objektnummer2,...
macht die Objekte (Sprites oder BOBs) mit den genannten Kennungen sichtbar (OBJECT.ON) oder läßt sie vom Bildschirm verschwinden (OBJECT.OFF)

Am Anfang des Kapitels wurden aber eigentlich bewegte Objekte erwähnt. Im Augenblick steht das Sprite noch bewegungslos auf dem Bildschirm; ein Zustand, der sofort geändert werden muß. Hängen Sie dazu bitte die folgenden Zeilen an das Programm »sprite-test«.

```
OBJECT.VX 1,-10
OBJECT.VY 1,-1
OBJECT.START
wert=1
WHILE wert
  SLEEP
WEND
```

Wenn Sie nun das Programm erneut starten, bewegt sich das Sprite gemächlich von der rechten zur linken Bildschirmseite und klettert dabei sogar ein kleines Stück den Bildschirm hinauf. »Schuld« daran sind drei Anweisungen, die dem Sprite eine Geschwindigkeit geben und die Bewegung in Gang setzen.

OBJECT.VX Objektnummer, Geschwindigkeit
OBJECT.VY Objektnummer, Geschwindigkeit
legt die *Geschwindigkeit* (in Bildschirmpunkten pro Sekunde) für ein Objekt (Sprite oder BOB) mit der Kennung *Objektnummer* in X- oder Y-Richtung fest

Auf der X-Achse bedeutet ein positiver Wert für *Geschwindigkeit* eine Bewegung nach rechts und ein negativer Wert eine Bewegung nach links. Auf der Y-Achse gibt eine negative Zahl eine Aufwärtsbewegung vor und ein positiver Wert bewegt das Sprite nach unten. Interessant hieran ist vor allem, daß die Bewegung völlig unabhängig vom Basic-Programm erfolgt. Das Sprite zieht so lange seine Bahn, bis es irgendwo anstößt. Sie brauchen also keine kunstvollen Schleifen zu programmieren, um ein Objekt jedesmal ein Stück weiterbewegen zu können. Der normale Programmablauf bleibt davon völlig unberührt. Bevor die Bewegung eines Objekts allerdings beginnt, müssen Sie den Startschuß dazu geben.

OBJECT.START ***Objektnummer1,Objektnummer2,...***
OBJECT.STOP ***Objektnummer1,Objektnummer2,...***
setzt alle Objekte (wenn keine Kennungen angegeben werden) oder die mit den genannten Kennungen in Bewegung beziehungsweise stoppt deren Bewegung (OBJECT-STOP)

Jetzt haben Sie schon eine ganze Menge neue Anweisungen kennengelernt. Versuchen Sie doch gleich einmal, wie deren Verwendung zusammen mit mehreren Sprites aussieht.

```
REM Bummelbahn
REM P.4.5-2
DEFINT a-z:ON ERROR GOTO fehler

anfang:
OPEN"lok1"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE #1
OPEN"lok2"FOR INPUT AS 1
OBJECT.SHAPE 2,INPUT$(LOF(1),1)
CLOSE #1
OPEN"wag"FOR INPUT AS 1
OBJECT.SHAPE 3,INPUT$(LOF(1),1)
CLOSE #1
PALETTE 0,.5,.8,1: PALETTE 1,.4,.6,0
PALETTE 2,0,0,.5: PALETTE 3,.8,.8,.8
LINE (0,123)-(615,199),1,bf
LINE (0,121)-(615,121),3
FOR i = 50 TO 600 STEP 50
  LINE (i,123)-(i-50,199),2
NEXT i
OBJECT.SHAPE 4,3: OBJECT.SHAPE 5,3
FOR i=1 TO 5:OBJECT.Y i,100:NEXT
OBJECT.X 1,490: OBJECT.X 2,490
OBJECT.X 3,522: OBJECT.X 4,554
OBJECT.X 5,586: OBJECT.ON
WHILE 1
  IF OBJECT.X(1)<=0 OR OBJECT.X(2)<=0 THEN
  FOR i=1 TO 5:OBJECT.VX i,20:NEXT:OBJECT.START
  END IF
  IF OBJECT.X(1)>=490 OR OBJECT.X(2)>=490 THEN
  FOR i=1 TO 5:OBJECT.VX i,-20:NEXT:OBJECT.START
  END IF
  IF a=20THEN OBJECT.STOP:OBJECT.X 1,OBJECT.X(2):OBJECT.
  ** ON 1:OBJECT.OFF 2 :OBJECT.START
  IF a=40 OR a=0 THEN OBJECT.STOP:OBJECT.X 2,OBJECT.X(1)
```

```
   ** :OBJECT.ON 2:a=0:OBJECT.OFF 1:OBJECT.START
   a=a+1
WEND

ende:END

fehler:
IF ERR<>53 THEN PRINT "Fehlernummer: "ERR:RESUME ende
RESUME konstruieren

konstruieren:
CLOSE:DIM spr(75):n=75:namen$="lok1":GOSUB speichern
namen$="lok2":GOSUB speichern:namen$="wag":GOSUB speichern
GOTO anfang

speichern:
FOR i = 1 TO n:READ spr(i):NEXT
OPEN namen$ FOR OUTPUT AS 1
   FOR i = 1 TO 5: PRINT #1,MKL$(spr(i));:NEXT i
   FOR i = 6 TO n:PRINT #1, MKI$(spr(i));:NEXT
CLOSE#1: RETURN

spr:
DATA 0,0,2,16,32,25,3,0,15,7,15,10,0,0,0,12,28,30,12,0,0,64
DATA 224,64,0,0,128,256,0,0,1024,0,0,0,0,0,0,0,0,0,15,7,15
DATA 10,0,0,0,12,28,30,12,0,0,64,224,64,0,0,128,256,0,0
DATA 1024,317185,7185,7199,32767,32767,-1,0,12300,255,0,3968
DATA 0,0,2,16,32,25,3,0,0,0,3,15,7,15,2,0,0,24,60,28,56,32,0
DATA 64,96,224,64,0,768,512,0,3072,0,0,0,0,0,0,0,0,0,0,3,15
DATA 7,15,2,0,0,24,60,28,56,32,0,64,96,224,64,0,768,512,
** 0,3103
DATA 7185,7185,7199,32767,32767,-1,0,12300,255,0,3968
DATA 0,0,2,16,32,25,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,32766
DATA 18726,18726,32766,32766,-1,0,12300,255,0,3968
```

Für dieses Programm, das den sinnigen Namen »Bummelbahn« trägt, brauchen Sie keine Sprites zu konstruieren. Die hierfür benötigten Daten sind nämlich im Programm selbst (in DATA-Anweisungen) enthalten. Beim ersten Start des Programms werden diese Daten in drei Dateien geschrieben und dann bei späteren Programmläufen aus diesen eingelesen. Beim Start versucht das Programm zunächst, diese Datei zu öffnen. Gelingt das nicht, wird (über die ON ERROR-Anweisung) zur Routine *fehler* verzweigt. Dort werden die Sprite-Werte aus den DATA-Anweisungen gelesen und dann unter den Dateinamen »lok1«, »lok2« und »wag« abgespeichert. Dann geht es zurück zum *anfang*, wo diesmal natürlich die Dateien gefunden werden.

Nachdem die drei Sprites (zwei Lokomotiven und ein Waggon) aus Dateien eingelesen wurden, werden die Sprites 4 und 5 durch Kopieren des dritten Sprites (Waggon) zum Leben erweckt. Dann wird ein Bahndamm mit Gleisen auf den Bildschirm gemalt. Mit

den Anweisungen OBJECT.X und OBJECT.Y wird der komplette Zug schließlich auf das Gleis gestellt und mit OBJECT.ON sichtbar gemacht. Die Bewegung der Bummelbahn und die Kontrolle derselben geschieht in der folgenden WHILE-Schleife.

In der WHILE-Schleife überprüft die erste IF-Anweisung, ob die Lok den linken Bildschirmrand erreicht hat. Ist dies der Fall, wird die Geschwindigkeit OBJECT.VX aller Sprites positiv, und der Zug bewegt sich nach rechts. Die zweite IF-Anweisung hat die gleiche Aufgabe; dieses Mal jedoch für den rechten Bildschirmrand. Die Geschwindigkeit OBJECT.VX sämtlicher Sprites wird negativ und der Zug fährt wieder vorwärts, nach links. Der interessanteste Aspekt dieses Programms steckt in den beiden letzten IF-Anweisungen der Schleife. Bei jedem Schleifendurchlauf wird die Variable a um eins erhöht. Sobald die WHILE-Schleife zwanzigmal durchlaufen wurde, wird die erste Lok mit OBJECT.OFF unsichtbar und die zweite Lokomotive durch OBJECT.ON sichtbar gemacht. Vorher wird die Position der einen Lok mit der OBJECT.X-Funktion festgestellt und an die zweite Lok übergeben.

v=OBJECT.X (Objektnummer)

v=OBJECT.Y (Objektnummer)

liefert die aktuelle X- beziehungsweise Y-Position des Objekts (Sprites oder BOBs) mit der Kennung *Objektnummer*

Das Umschalten zwischen beiden Dampflokomotiven geht so schnell vonstatten, daß man es auf dem Bildschirm kaum bemerkt. Da die beiden Loks aber unterschiedlich positionierte Rauchwolken erhalten haben, sieht es so aus, als wenn sich die Rauchwolken bewegen würden.

So, das waren die wichtigsten Aspekte von Sprites, soweit sie von Basic aus manipulierbar sind. Im Rest dieses Kapitels spielen die BOBs die Hauptrolle. Ich bin aber sicher, daß Sie nach diesem Beispiel auf den Geschmack gekommen sind und nun darauf brennen, eigene Ideen zu verwirklichen.

4.6 Kunstflug

Nachdem im letzten Kapitel recht ausführlich die Handhabung von Sprites-Erstellung behandelt worden ist, sind jetzt die BOBs an der Reihe. »BOB« ist die Abkürzung für **B**litter **OB**ject. Der Blitter ist ein Chip im Innern des Amiga, dessen Aufgabe es unter anderem ist, zusammen mit spezieller Software Bilder über den Bildschirm zu bewegen.

BOBs können in nahezu jeder Größe konstruiert werden. Weder ihre Anzahl noch die Anzahl der Farben, die sie enthalten, ist begrenzt. (BOBs können so viele oder so wenige Farben besitzen wie der Bildschirm, auf dem sie sichtbar sind.) Zusätzliche Farben auf den Bildschirm bringen, wie es mit Sprites möglich ist, können BOBs nicht. Dies sind

die wichtigsten Unterschiede der BOBs gegenüber den Sprites. Wenn Sie mit »ObjEdit« arbeiten, müssen Sie genau auf die Reihenfolge der Farben achten, da dieses Programm in der Regel mit anderen Farben arbeitet, als sie später im Programm verwendet werden.

Das Programm »ObjEdit« ist zunächst einmal für vier Farben, das heißt für zwei Bit-Ebenen eingestellt. Es läßt sich aber relativ einfach auf drei Bit-Ebenen, also auf acht Farben umstellen. Wir werden diese Möglichkeit in einem späteren Programm auch nutzen. Probieren Sie aber zunächst einmal ein vierfarbiges BOB aus. Gehen Sie dabei genauso vor, wie es im letzten Abschnitt für Sprites beschrieben wurde. Starten Sie »ObjEdit«, und geben Sie auf die Frage nach der Art des Objektes eine Null für BOBs ein. Speichern Sie das BOB unter dem Namen »bob-test« ab. Ändern Sie dann im Programm »sprite-test« (siehe oben) eine Zeile. Starten Sie das geänderte Programm und betrachten Sie Ihr Werk!

von: OPEN "test" FOR INPUT AS 1

in: OPEN "bob-test" FOR INPUT AS 1

Sind Sie ein bißchen unzufrieden mit Ihrem ersten BOB? Da haben Sie nun einen Supercomputer mit den tollsten Farbmöglichkeiten und müssen sich mit lausigen vier Farben begnügen. Da der Objekt-Editor die Möglichkeit bietet, acht Farben darzustellen, klingt da auch nicht überwältigend. Die folgenden Anweisungen zeigen Ihnen deshalb, wie Sie ihm beibringen, BOBs mit 32 Farben zu erzeugen.

Dazu bleibt Ihnen nichts anderes übrig, als das Programm »ObjEdit« zu ändern. Keine Sorge, ich habe den Änderungsaufwand so gering wie möglich gehalten. Wählen Sie zuerst das Programm ObjEdit und starten Sie es. Wenn das Programm nach der Art der Objekte, Sprites oder BOBs, fragt, unterbrechen Sie es mit *Stop* aus dem *Run*-Menü. Jetzt können Sie sich mit *Show List* aus dem Menü *Windows* das Listing zeigen lassen. Vergrößern Sie das List-Fenster so weit es geht und blättern Sie mit der Tastenkombination SHIFT-CURSOR DOWN (=Pfeil abwärts) auf die zweite Bildschirmseite. Hier müssen Sie die erste Änderung vornehmen.

Ändern Sie die folgenden Zeilen

```
WinY=185: WinX=617
'If BOBs are to be created with other than 2 bit-planes
' alter next 3 lines (only if machine has more than 256k)
' Depth=3
' scrn=1
' SCREEN scrn,640,200,Depth,2
, WINDOW 1,,(0,0)-(WinX,WinY),31,scrn
```

in:

```
WinY=170: WinX=270
'If BOBs are to be created with other than 2 bit-planes
' alter next 3 lines (only if machine has more than 256k)
Depth=5
scrn=1
```

```
SCREEN scrn,320,200,Depth,1
WINDOW 1,,(0,0)-(WinX,WinY),31,scrn
```

Die zweite Änderung betrifft die beiden Routinen *PrintColorBar* und *CheckColor:*. Diese beiden Programmpunkte liegen hintereinander ziemlich am Ende des Listings. Ändern Sie die folgenden Zeilen (die Programmzeilen, welche Sie nicht ändern müssen, sind nur angedeutet.):

```
PrintColorBar:
   COLOR ...
   LOCATE ...
   ColorBar ...
   COLOR ...
   x=70
   FOR i=0 TO maxColor
        LINE (x,ColorBar)-(x+20,y+ColorBar+10),i,bf
        LINE (x,ColorBar)-(x+20,y+ColorBar+10),1,b
        x=x+20
   NEXT i
   RETURN

CheckColor:
   IF ...
   IF ...
   i=INT((CurrentX-70)/20)
   ...
```

in:

```
PrintColorBar:
   COLOR ...
   LOCATE ...
   ColorBar ...
   COLOR ...
   x=70
   FOR i=0 TO maxColor
        LINE (x,ColorBar)-(x+6,y+ColorBar+10),i,bf
        LINE (x,ColorBar)-(x+6,y+ColorBar+10),1,b
        x=x+6
   NEXT i
   RETURN

CheckColor:
   IF ...
   IF ...
   i=INT((CurrentX-70)/6)
   ...
```

Das war dann auch schon alles. Damit Sie das Programm nicht jedesmal ändern müssen, wenn Sie 32farbige BOBs erstellen wollen, speichern Sie es am besten nun unter dem Namen »ObjEdit32« ab. Und damit Sie gleich in den Genuß der neuen Farbenpracht

kommen, konstruieren Sie jetzt bitte zwei BOBs. Starten Sie dazu das neue Programm »ObjEdit32«. Sie werden vom Programm gar nicht mehr danach gefragt, ob Sie BOBs oder Sprites erstellen wollen, sondern es erscheint sofort das Konstruktionsfenster. Da das Programm mit einem Minimum an Arbeit geändert wurde, müssen Sie unbedingt beachten:

ACHTUNG! Auf keinen Fall die Vergrößerung ENLARGE 4x4 wählen!

In diesem Falle würde nämlich alles zerstört, was Sie bis dahin gezeichnet haben. Sie könnten zwar in dem vergrößerten Feld zeichnen, aber beim Abspeichern wird nichts davon übernommen. Das ist aber auch schon alles, was Sie beachten müssen. Sie können ansonsten im Konstruktionsfenster mit allen Werkzeugen und mit allen 32 Farben arbeiten. Außerdem steht auch einer Vergrößerung oder Verkleinerung des Fensters nichts im Wege. Beachten Sie aber auch, daß die Farben in den 32 Kästchen nicht den Farben entsprechen, die hinterher im Programm verwendet werden.

Zeichnen Sie nun zwei BOBs. Das erste etwa mit den Maßen x=30 und y=28. Der Name dafür soll »bob32« sein. Als zweites BOB nehmen Sie sich gleich einen ordentlichen Brocken vor. Ich schlage vor, daß er die Maße x=93 und y=84 erhalten soll. Diesen taufen Sie auf den Namen »test32«. Um einen Eindruck der Farbenvielfalt zu bekommen, verwenden Sie möglichst viele verschiedene Farben dazu. Verlassen Sie dann das Programm, schließen Sie die Demo-Schublade und öffnen Sie sie wieder. Ziehen Sie dann die beiden Icons mit den BOB-Daten heraus und legen Sie diese zusammen mit dem folgenden Programm in einer anderen Schublade ab.

```
REM Farbenpracht
REM P.4.6-1 BOBs in 32 Farben
CLEAR ,40000&
DEFINT a-z
SCREEN 1,320,200,5,1
WINDOW 2,"bitte Taste druecken",(0,0)-(310,180),16,1
PALETTE 0,1,1,1 :PALETTE 10,.8,0,0 :PALETTE 2,1,0,0
PALETTE 3,.93,.2,0 :PALETTE 4,1,.4,0 :PALETTE 5,1,.6,0
PALETTE 6,1,.8,0 :PALETTE 7,1,1,0 :PALETTE 8,.6,1,.15
PALETTE 9,.5,.8,.15:PALETTE 1,.4,.6,0 :PALETTE 11,.2,.4,0
PALETTE 12,0,.4,0 :PALETTE 13,0,.6,.67:PALETTE 14,0,.8,.6
PALETTE 15,0,1,.6 :PALETTE 16,.2,1,.93:PALETTE 17,.2,.75,1
PALETTE 18,.15,.4,1:PALETTE 19,.4,0,1 :PALETTE 20,0,0,.6
PALETTE 21,.35,.15,.9:PALETTE 22,.6,.2,1:PALETTE 23,1,0,1
PALETTE 24,.9,.5,.75:PALETTE 25,1,.75,.75:PALETTE
** 26,.8,.55,.5
PALETTE 27,.6,.4,.35:PALETTE 28,.4,.2,0:PALETTE 29,.4,.4,.4
PALETTE 30,.6,.6,.6 :PALETTE 31,0,0,0
FOR j=0 TO 31 STEP 8
  FOR i = 0 TO 7
  LINE (i*40,j*6)-((i*40)+39,(j*6)+47),i+j,bf
  NEXT
NEXT
OPEN"bob32"FOR INPUT AS 1:OBJECT.SHAPE 1,INPUT$(LOF(1),1)
```

```
** :CLOSE #1
OPEN"test32"FOR INPUT AS 1: OBJECT.SHAPE 2,INPUT$(LOF(1),1)
** :CLOSE #1
OBJECT.X 1,260: OBJECT.Y 1,30
OBJECT.X 2,190: OBJECT.Y 2,90
OBJECT.ON
WHILE 1
   IF OBJECT.X(1)<=0 OR OBJECT.X(2)<=0 THEN
   OBJECT.VX 1,3 :OBJECT.VX 2,1:OBJECT.START
   END IF
   IF OBJECT.X(1)>=290 OR OBJECT.X(2)>=190 THEN
   OBJECT.VX 1,-3:OBJECT.VX 2,-1:OBJECT.START
   END IF
   GOSUB taste
WEND

aus:
SCREEN CLOSE 1
END

taste:
t$=INKEY$
IF t$<>"" THEN aus
RETURN
```

Wahrscheinlich wird Ihnen das Programm etwas zu bunt vorkommen. Zur Demonstration der Farbmöglichkeiten des Amiga darf es jedoch auch einmal zu farbenfroh sein. In Ihren Programmen werden Sie sicher dezentere Farbabstufungen einsetzen. Nun zu dem Programm selbst. Mit der SCREEN- und WINDOW-Anweisung wurde ein Bildschirm von 320 Bildpunkten Breite und 200 Pixeln Höhe vorgegeben. Die PALETTE-Anweisung definiert dann die 32 darin möglichen Farben. In der folgenden FOR-Schleife wird der Bildschirm mit 32 verschiedenfarbigen Rechtecken ausgemalt. Mit den inzwischen bekannten OBJECT-Anweisungen werden die beiden BOBs dann eingelesen und auf dem Bildschirm positioniert. Die folgende WHILE-Schleife sorgt dafür, daß sich die beiden BOBs gemächlich zwischen der linken und rechten Bildschirmseite hin- und herbewegen. Außerdem wird bei jedem Schleifendurchgang die Routine *taste* angesprungen, die Ihnen die Möglichkeit gibt, das Programm zu beenden. Dieses Programm zeigt die Möglichkeiten, die in BOBs stecken, schon sehr schön. Es hat jedoch auch einen Haken: es tut eigentlich wenig. Deshalb wird nun ein kleines Spiel entwickelt, in dem achtfarbige BOBs bewegt werden.

Zuerst will ich Ihnen kurz die Geschichte zu diesem Programm erzählen. Der Name dieses Spieles lautete zunächst »Wolkenflug«. Es war so gedacht, daß ein Flugzeug zwischen einer großen Anzahl von Wolken hin- und herfliegt. Hinter einer dieser Wolken war ein Kirchturm versteckt, den der Pilot (Spieler) nicht sehen kann. Aufgabe bei diesem Spiel war es nun, möglichst lange umherzufliegen, ohne auf den Turm zu treffen. Die Spielidee schien nicht schlecht. Nur spielte der Amiga nicht mit. Die Wolken-BOBs sollten doch die Kirchturmspitze verstecken. Davon konnte aber keine Rede sein. Die für

die BOBs zuständige Software paßt nicht richtig mit der deutschen Fernsehnorm zusammen. Daher flimmern die BOBs bei jeder Bewegung sehr stark. Je größer ein BOB ist, um so stärker flimmert er. Man kann deshalb nur hoffen, daß diese Software möglichst rasch verbessert wird.

So entstand als Ersatz das Programm »Kunstflug«. Der Spieler hat die Aufgabe, im Tiefflug zwischen Bäumen hin- und herzufliegen. Die Position der Bäume verändert sich, und die Geschwindigkeit des Flugzeugs erhöht sich nach jedem Durchgang. Je länger der Spieler das Flugzeug in der Luft halten kann, um so mehr Punkte erreicht er. Da die Bäume ziemlich große BOBs sind, ließ sich leider auch hier ein Flimmern nicht vermeiden – es beeinträchtigt den Spielzweck jedoch nicht so sehr wie beim Wolkenflug.

Damit das Programm nicht zuviel Platz beansprucht, habe ich auf programmierte Sprites verzichtet. Sie dürfen also zuerst auf dem Objekt-Editor zwei BOBs konstruieren. Den ObjEdit stellen wir dazu auf die Farbtiefe 3, für 8 Farben, um. Entfernen Sie dazu die vier Hochkommata am Anfang des Programmes, wie wir es vorhin auch taten. Sie brauchen allerdings diesmal keine weiteren Änderungen vorzunehmen. Die einzelnen Farben haben im Programm folgende Werte:

Farbnummer der PALETTE-Anweisung:	**Farbton**
0 bis 2	Hellblau
3	Hellrot
4	Schwarz
5	Dunkelgrün
6	Mittelgrün
7	Hellgrün

Tabelle 4.1: *Farbpalette im Programm Kunstflug*

Das erste BOB »baum« sollte die Aufsicht auf einen Baum darstellen. Verwenden Sie dafür am besten die Farben 5 bis 7. Die Größe sollte etwa 83 (X) mal 40 (Y) sein. Das andere BOB »flugz« wird ein Flugzeug mit den Abmessungen X=61 und Y=24. Das Flugzeug soll mit dem Propeller nach links gezeichnet werden. Als Farben empfehlen sich Rot und Schwarz. Legen Sie die beiden BOBs dann wie gehabt zusammen mit dem folgenden Programm in einer anderen Schublade als »BASICDemos« ab.

```
REM Kunstflug
REM P.4.6-2
DEFINT a-z:DIM we(255):GOSUB rau:WAVE 1,we:ERASE we:ton=0
ON COLLISION GOSUB zusammen:COLLISION ON
SCREEN 1,640,200,3,2
WINDOW 2,,(0,0)-(610,180),,1
PALETTE 0,.5,.8,1: PALETTE 1,.5,.8,1
PALETTE 2,.5,.8,1: PALETTE 3,1,.4,0
PALETTE 4,0,0,0: PALETTE 5,0,.8,.6
```

```
PALETTE 6,0,.9,.6: PALETTE 7,0,1,.6
OPEN"flugz"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE #1
OPEN"baum"FOR INPUT AS 1
OBJECT.SHAPE 2,INPUT$(LOF(1),1)
CLOSE #1
OBJECT.SHAPE 3,2:OBJECT.SHAPE 4,2
start:
CLS
GOSUB vorfahrt:GOSUB unterbr
GOSUB baumpos:y1=100:tempo=30:punkte=0:x1=450
OBJECT.X 1,x1:OBJECT.Y 1,y1
FOR i = 2 TO 4:OBJECT.VX i,tempo:NEXT
OBJECT.ON :OBJECT.START
ton=1:zeiger=1:COLOR 4:LOCATE 2,60:PRINT "PUNKTE: "
WHILE zeiger
  LOCATE 2,68:PRINT punkte
  IF OBJECT.X(2)>500 THEN
  FOR i = 2 TO 4:OBJECT.OFF i:NEXT
  GOSUB baumpos
  FOR i = 2 TO 4:OBJECT.VX i,50:NEXT
  FOR i = 2 TO 4:OBJECT.START i:NEXT
  END IF
  y1=y1+STICK(3):IF y1>145 THEN y1=145
  IF y1<0 THEN y1=0
  OBJECT.Y 1,y1
  IF ton THEN SOUND 20,1.4,255,1
WEND
GOTO start
zusammen:
ton=0
LOCATE 10,1:INPUT "SPIELENDE - Noch ein Spiel? j/n ";ta$
IF ta$="j" THEN zeiger=0:RETURN
SCREEN CLOSE 1:END
RETURN
unterbr:
OBJECT.HIT 1,2,28:OBJECT.HIT 2,4,2
OBJECT.HIT 3,8,2:OBJECT.HIT 4,16,2
RETURN
vorfahrt:
OBJECT.PRIORITY 1,0:OBJECT.PRIORITY 2,3
OBJECT.PRIORITY 3,1:OBJECT.PRIORITY 4,2
RETURN
```

```
baumpos:
tempo=tempo+5:IF tempo>90 THEN tempo=90
punkte=punkte+tempo
RANDOMIZE TIMER: zl=(INT(RND*4))
FOR i = 2 TO 4
po=(zl*42)+1
OBJECT.X i,0:OBJECT.Y i,po:OBJECT.ON i
zl=zl+1:IF zl>3 THEN zl=0
NEXT i
x1=x1-10:OBJECT.X 1,x1:RETURN

rau:
RANDOMIZE TIMER
FOR i=0 TO 63
  we(i)=2*i-INT(RND*120)
NEXT
RANDOMIZE TIMER
FOR i=1 TO 127 STEP 2
  we(i)=127-i-INT(RND*120)
NEXT
RANDOMIZE TIMER
FOR i=1 TO 127 STEP 2
  we(i)=0-i+INT(RND*120)
NEXT
RANDOMIZE TIMER:FOR i=0 TO 63
  we(i)=-127+(2*i)+INT(RND*120)
NEXT
RETURN
```

Beachten Sie bitte, daß Sie einen Joystick (Steuerknüppel) benötigen, um mit diesem Programm zu spielen. Wenn Sie nicht über einen Joystick verfügen, müssen Sie sich mit den folgenden Erklärungen begnügen. Gleich zu Beginn finden Sie eine neue Anweisung: ON COLLISION. Diese dient wie ON MENU oder ON MOUSE dazu, das Programm beim Eintreten bestimmter Ereignisse zu bestimmten Programmstellen zu lenken.

ON COLLISION GOSUB Sprungmarke
sorgt dafür, daß die Subroutine bei *Sprungmarke* aufgerufen wird, wenn ein Objekt (Sprite oder BOB) mit einem anderen Objekt oder dem Bildschirmrand zusammentrifft

Sie können damit genau vorgeben, was im Falle eines Zusammenstoßes eines Objektes mit einem anderen geschieht. Damit nach einer Kollision nicht weitere Unterbrechungen hervorgerufen werden (das Geschehen auf dem Bildschirm geht ja unterdessen weiter), wird bei dieser Anweisung gleichzeitig COLLISION STOP (siehe unten) ausgeführt. In diesem Programm wird bei jeder Kollision nach zusammen *verzweigt.* Hier wird der

Spieler unter anderem gefragt, ob er weiterspielen will. Drückt er auf *j*, dann setzt das Programm den Wert von *zeiger* auf Null und verläßt dadurch automatisch die WHILE-Schleife des Hauptprogrammes. Ein direkter Sprung zu *start* verbietet sich, da das Amiga-Basic völlig durcheinanderbringen würde. Auf jedes GOSUB muß nämlich ein RETURN erfolgen (und auf jedes FOR ein NEXT sowie auf WHILE ein WEND). Direkte Sprünge aus Subroutinen oder Schleifen heraus sollten Sie deshalb vermeiden und statt dessen die hier vorgestellte Lösung bevorzugen. Nun aber wieder zurück zum Programmablauf. Die ON COLLISION-Anweisung muß wie jede andere ON-Anweisung zunächst durch COLLISION ON aktiviert werden, bevor sie etwas bewirkt.

COLLISION ON

setzt eine vorangegangene ON COLLISION-Anweisung in Kraft.

COLLISION OFF

hebt die Wirkung der letzten ON COLLISION-Anweisung auf.

COLLISION STOP

hebt die Wirkung der gerade gültigen ON COLLISION-Anweisung vorübergehend auf, bis wieder COLLISION ON ausgeführt wird (dabei merkt sich Amiga-Basic alle Kollisionen in einer Art Warteschlange und arbeitet sie nach dem COLLISION ON nacheinander ab)

Nach diesen Befehlen für die Programmsteuerung wird ein Fenster in einem Screen mit acht Farben geöffnet und diese Farben werden dann mit PALETTE festgelegt. Sie werden vielleicht bemerkt haben, daß die ersten drei Farben gleich sind. Das bewirkt, daß der Fensterrahmen vollständig verschwindet. Mit OBJECT.SHAPE werden dann drei Bäume und ein Flugzeug definiert. Bei *start* beginnt der eigentliche Programmablauf mit der Festlegung zusätzlicher Eigenschaften für die BOBs, auf die ich später noch zu sprechen komme, und der Positionierung des Flugzeug-BOBs. In der darauffolgenden WHILE-Schleife wird der aktuelle Punktestand ausgegeben, bei Überschreitung einer bestimmten x-Position des Flugzeugs ein neuer Durchlauf gestartet und der Joystick abgefragt.

In der Unterroutine *unterbr* wird für jedes BOB festgelegt, welcher Zusammenstoß dieses BOBS mit einem anderen Objekt eine Unterbrechung hervorruft und welcher nicht.

OBJECT.HIT Objektnummer, *Selbstwert, Fremdwert*

legt fest, mit welchen anderen Objekten das Objekt mit der Kennung *Objektnummer* zusammenstoßen muß, um eine Unterbrechung hervorzurufen

Normalerweise führt jeder Zusammenstoß mit einem anderen Objekt oder der Fensterbegrenzung zu einer Unterbrechung (und einer Verzweigung mit ON COLLISION).

OBJECT HIT erlaubt es, diesen Kreis der gefährlichen Objekte etwas einzuschränken. Leider hat die Sache einen Haken. Um die Werte für Selbstwert und Fremdwert bestimmen zu können, müssen wir uns wieder auf die unterste Ebene des Amiga begeben, auf die Ebene der Bits. Für jeden der beiden Werte stehen 16 Bit zur Verfügung. Bei einer Kollision werden die beiden Werte logisch mit UND verknüpft. Nur wenn beide Bits gesetzt (gleich 1) sind, findet eine Unterbrechung statt. Das letzte (nullte) Bit ist für Zusammenstöße mit der Fensterbegrenzung reserviert. Ist es nicht gesetzt, spielen Zusammenstöße mit der Fensterbegrenzung keine Rolle. Damit Sie sich das besser vorstellen können, schauen Sie sich bitte einmal an, wie das in unserem Programm realisiert wurde:

	Selbstwert:	**Fremdwert:**
OBJECT.HIT 1,2,28 (Flugzeug)	00000000 00000010	00000000 00011100
OBJECT.HIT 2,4,2 (Baum)	00000000 00000100	00000000 00000010
OBJECT.HIT 3,8,2 (Baum)	00000000 00001000	00000000 00000010
OBJECT.HIT 4,16,2 (Baum)	00000000 00010000	00000000 00000010

Tabelle 4.2 : *Verwendung von OBJECT.HIT*

Kurz gefaßt kann man sagen, daß mit dem *Selbstwert* hinter OBJECT.HIT zunächst für jedes Objekt ein Bit definiert wird, das bei Zusammenstößen dieses Objekt identifiziert. In der obigen Tabelle bekommt das Flugzeug das zweite Bit von rechts (2), der erste Baum das dritte Bit (4), der zweite Baum das vierte (8) und der dritte Baum das fünfte Bit (16). Für den Bildschirmrand ist immer das am weitesten rechts stehende Bit (1) reserviert. Im Selbstwert darf immer nur ein Bit gesetzt (gleich 1) werden. Da der Selbstwert 16 Bit umfaßt, können auf diese Weise maximal 15 Objekte (plus Bildschirmrand) identifiziert werden. Der *Fremdwert* gibt an, mit welchen anderen Objekten ein Objekt (dessen Kennung OBJECT.HIT folgt) zusammenstoßen muß, um eine Unterbrechung hervorzurufen. Findet ein Zusammenstoß eines Objekts mit einem anderen statt, so wird der *Selbstwert* des anderen Objekts mit dem Fremdwert des ersten Objekts verknüpft. Eine Unterbrechung (und Verzweigung über ON COLLISION) findet nur dann statt, wenn die UND-Verknüpfung dieser beiden Zahlen eine Zahl ergibt, in der mindestens ein Bit gesetzt ist (die also ungleich Null ist).

Gleich in der nächsten Routine *vorfahrt* kommt schon wieder ein neuer Befehl vor. Dieser ist aber bei weitem nicht so kompliziert wie der OBJECT.HIT-Befehl.

OBJECT.PRIORITY Objektnummer, Priorität

legt fest, welches Objekt bei eventuellen Überdeckungen vor welchen anderen Objekten liegt (diese überdeckt). Danach verdecken Objekte mit einer höheren Priorität Objekte niedriger Priorität

Durch das Übereinanderschieben von grafischen Objekten kann man die Illusion von Räumlichkeit schaffen. Legt man die Objekt-Prioritäten nicht fest, haben alle Objekte die gleiche Priorität und überdecken sich zufällig. Der *Prioritätswert* darf zwischen -32768 und +32767 liegen. Im Programm »Kunstflug« haben alle drei Bäume einen höheren Wert als das Flugzeug. Damit wird erreicht, daß das Flugzeug hinter den Bäumen herfliegt (eine Kollision tritt natürlich trotzdem ein).

Damit sind wir auch schon bei der letzten Routine dieses kleinen Demonstrationsprogrammes. In *baumpo* wird die Geschwindigkeit des Flugzeugs in Fünferschritten bis auf 60 Pixel pro Sekunde erhöht. Damit der Spieler von der größeren Schwierigkeit auch etwas hat, wird seine Punktezahl bei jedem Durchlauf, den er unbeschadet überstanden hat, um die aktuelle Geschwindigkeit erhöht. Die Positionierung der Bäume wird dem Zufallsgenerator überlassen. Damit ist sichergestellt, daß man die Position der Bäume garantiert nicht vorhersehen kann.

Sind Ihnen bei diesen Beispielen Ideen für eigene Projekte gekommen? Ich finde auch, daß sich trotz der Flimmerei der großen BOBs interessante Möglichkeiten eröffnen. Wäre es nicht noch schöner, wenn BOBs und Sprites kombiniert werden könnten? Das ist überhaupt kein Problem. Im nächsten Abschnitt wird ein Beispiel zu diesem Thema vorgestellt.

4.6.1 Der Joystick

Sie haben sicherlich bemerkt, daß ich mich bei der Beschreibung des letzten Programms um die Erläuterung eines Befehls »herumgemogelt« habe. Ich wollte Ihnen diesen nicht vorenthalten. Damit Sie mit dem »Freudenstock«, wie Joystick (frei) übersetzt heißt, auch Ihre Freude haben, gehe ich in diesem Abschnitt nun etwas genauer darauf ein.

Der Joystick ist ein Steuerknüppel, mit dem Sie – ähnlich wie mit der Maus – Bewegungen auf den Bildschirm übertragen und Ereignisse auslösen können. Welchen Joystick Sie verwenden, hängt davon ab, was Sie mit ihm anstellen wollen. Voraussetzung ist natürlich, daß der Anschluß paßt. Probleme werden Sie aber mit dem Anschluß sicher nicht bekommen. An meinem Amiga hängt zum Beispiel gerade der uralte Joystick meines Sohnes (von einem Atari-Telespiel). Sie können an Ihrem Amiga bis zu zwei Joysticks anschließen. Einen Anschluß finden Sie direkt neben (oder hinter) dem Mausanschluß. Der zweite wird direkt in den Mausanschluß gesteckt. Wenn Sie auf die Maus nicht verzichten wollen, können Sie auch nur einen Steuerknüppel anschließen.

Damit Sie Bewegungen des Joysticks in die entsprechende Bewegung auf dem Bildschirm umsetzen können, müssen Sie natürlich wissen, welche Bewegungen der Joystick gerade ausführt. Dafür hält Amiga-Basic auch eine Funktion bereit.

v=STICK(n)
liefert den Zustand des Joysticks in Port 1 oder 2 als Zahl zwischen -1 und 1

Die Funktion STICK liefert je nach *n* die folgenden Werte :

n	Joystick und Richtung	Werte und Bewegungen
0	Stick 1(A) in x-Richtung	1=nach rechts 0=keine -1=nach links
1	Stick 1(A) in y-Richtung	1=nach oben 0=keine -1=nach unten
2	Stick 2(B) in x-Richtung	1=nach rechts 0=keine -1=nach links
3	Stick 2(B) in y-Richtung	1=nach oben 0=keine -1=nach unten

Tabelle 4.3: *Ergebnisse der Funktion STICK*

Sie werden sich nun vielleicht fragen, warum hier zur Unterscheidung der beiden Joysticks Zahlen und Buchstaben angegeben werden. Die Erklärung steht im Basic-Handbuch. Dort finden Sie in den Erläuterungen zum Joystick die Buchstaben A und B. Um die Angelegenheit weiter zu komplizieren, finden Sie an Ihrem Amiga an den entsprechenden Anschlüssen die Zeichen 1 und 2. Ich werde aber die beiden Joysticks im folgenden nur noch nach ihren Anschlüssen (Ports), also mit 1 und 2 bezeichnen.

Betrachten Sie die Tabelle mit den möglichen Ergebnissen von STICK. Wie können Sie die Richtung, in die der Anwender den Steuerknüppel gerade drückt, am besten erfahren? Sie könnten zum Beispiel die folgenden Zeilen in Ihrem Programm verwenden.

```
IF STICK(2) =1 THEN x=x+1 REM z.B.Sprite-Position
IF STICK(2) =-1 THEN x=x-1
```

Das wäre aber ungeschickt. Die STICK-Funktion liefert nämlich genau die Werte, die Sie für die Positionsänderung benötigen! Es geht also viel einfacher. Die folgende Basic-Zeile leistet exakt dasselbe wie die beiden Zeilen oben.

```
x=x+STICK(2)
```

Damit haben Sie schon zwei Richtungen abgefragt und weitergegeben. Wünschen Sie eine schnellere Bewegung, multiplizieren Sie einfach das Ergebnis der Funktion mit einem bestimmten Faktor.

```
x=x+2*STICK(2)
```

Schon hat sich die jeweilige Bewegungsänderung verdoppelt. Damit Sie einmal nach Herzenslust Ihren Steuerknüppel malträtieren können, nun ein kleines Programm:

```
REM Steuerknueppel
REM P.4.6.1-1 STICK STRIG
OPEN"flugz"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE #1
zeiger=1:OBJECT.ON :x1=100
WHILE zeiger
  OBJECT.X 1,x1
  OBJECT.Y 1,y1
  GOSUB joyabfr
```

```
  IF x1<0 THEN zeiger=0
WEND
PRINT "ENDE":OBJECT.OFF:END

joyabfr:
x1=x1+3*STICK(2)
y1=y1+3*STICK(3)
IF STRIG(3) THEN LINE (x1-80,y1+11)-(x1-5,y1+11):fe=1
IF STRIG(3) =0 AND fe THEN CLS:fe=0
RETURN
```

Als bewegtes Objekt wird hier wieder das BOB »flugz« aus dem Programm »Kunstflug« verwendet. Im Programmteil *joyabfr* werden die Werte für die x- und y-Bewegung des Joysticks festgestellt und an das BOB weitergegeben. Mit STRIG wird danach der Feuerknopf des Joysticks abgefragt.

v=STRIG(n)
übergibt den Status des Feuerknopfes an Joystick 1 oder 2

Je nach n und dem Zustand des jeweiligen Feuerknopfes ergibt STRIG dabei die folgenden Werte.

n	STRIG(n)	Bedeutung
0	1 / 0	Knopf an Stick 1 wurde gedrückt / nicht gedrückt
1	1 / 0	Knopf an Stick 1 ist gerade gedrückt / nicht gedrückt
2	1 / 0	Knopf an Stick 2 wurde gedrückt / nicht gedrückt
3	1 / 0	Knopf an Stick 2 ist gerade gedrückt / nicht gedrückt

Tabelle 4.4: *Mögliche Ergebnisse von STRIG*

Im Programm wird mit STRIG(3) gefragt, ob der Joystick 2 (am hinteren Anschluß) gerade betätigt wird. Die andere Abfragemöglichkeit STRIG(2) würde auch dann 1 ergeben, wenn der Joystick zwar seit dem letzten Aufruf von STRIG gedrückt, inzwischen aber wieder losgelassen wurde. (STRIG(3) ergibt nur dann 1, wenn der Knopf immer noch festgehalten wird.)

Nun wird es aber Zeit, daß Sie mit dem Flugzeug einige Runden drehen. Wenn Sie auf den Feuerknopf drücken, erscheint eine weiße Linie (Schuß?) vor dem Flugzeug. Die Linie verschwindet wieder, sobald Sie den Feuerknopf loslassen. Fliegen Sie ein paar Kurven mit gedrücktem Feuerknopf. So können Sie sich nämlich recht einfach als Himmelsmaler austoben. Das Fenster können Sie oben, unten und rechts verlassen. Verlassen Sie es aber auf der linken Seite, wird zunächst die innere WHILE-Schleife durchlaufen und dann das Programm beendet.

4.7 Abschuß

Nachdem Sie in den beiden letzten Abschnitten Sprites und BOBs kennengelernt haben, sollen zunächst einmal die wesentlichen Unterschiede zwischen beiden Arten von beweglichen Objekten in einer Tabelle festgehalten werden.

Sprites	**BOBs**
allein durch Hardware erzeugt	softwaremäßig gesteuert
hohe Geschwindigkeit	langsamer als Sprites
maximal acht Sprites	keine Begrenzung in der Anzahl
Breite max. 16 Pixel	unbegrenzte Größe
3 Farben + Hintergrundfarbe jedoch andere Farben als im eingesetzten Bildschirm möglich	gleiche Anzahl wie im eingesetzten Bildschirm
exakte Darstellung	z.Zt. flimmern größere BOBs

Tabelle 4.5: *Unterschiede zwischen Sprites und BOBs*

Je nach Anwendungsfall können Sie sich für eine der beiden Objekt-Arten entscheiden. Vielleicht ist Ihnen bei den bisherigen Programmen aufgefallen, daß BOBs und Sprites weitgehend gleich behandelt werden. Es steht Ihnen also nichts im Wege, beide Arten von beweglichen Objekten zusammen in einem Programm zu verwenden.

```
REM Abschuss
REM P.4.7-1
DEFINT a-z
ON ERROR GOTO fehler
DIM we(255):b1=16:b2=224:GOSUB puls
WAVE 1,we:WAVE 2,we:ERASE we
PALETTE 0,.5,.8,.15: PALETTE 1,1,1,0
PALETTE 2,.93,.2,0: PALETTE 3,.4,.4,.4
COLOR 2
PRINT " S D I - R A K E T E N A B W E H R   I M
** W E L T R A U M"
LOCATE 3,1:PRINT "Ihre Aufgabe ist es die Abwehrrakete im
** richtigen Augenblick zu starten."
PRINT "Druecken Sie dazu bitte den Feuerknopf des Joy-
**Sticks."
LOCATE 5,40:PRINT "Treffer: ":LOCATE 5,60:PRINT "Fehler: "
```

Im ersten Teil des Programms finden Sie nichts wesentlich Neues. Im Anschluß an einige Vorarbeiten werden die Wellenformen für die Tonkanäle 2 und 3 festgelegt. Die vier Farben werden dann mit PALETTE bestimmt und die »Bedienungsanleitung« wird ausgegeben. Interessanter wird es da schon in dem Programmabschnitt *anfang*. Eine neue Anweisung engt den Bewegungsraum der Objekte ein:

OBJECT.CLIP (x1,y1)-(x2,y2)
schränkt den Bereich, in dem sich Objekte bewegen können, auf das genannte Rechteck mit der linken oberen Ecke bei (x1,y1) und der rechten unteren Ecke bei (x2,y2) ein

(Normalerweise sind die Grenzen des Bewegungsraums gleich den Abmessungen des Ausgabefensters.) Außerhalb dieser Begrenzung sind also keine BOBs oder Sprites sichtbar. Außerdem gilt diese Grenze auch für die COLLISION-Anweisung als Fensterbegrenzung. Im Programm »Abschuß« wird mit dieser Anweisung der obere Fensterbereich, in dem der Text erscheint, für die Objekte gesperrt. Diese Begrenzungslinie hält ein Objekt jedoch nicht schlagartig an. Je nach der erreichten Geschwindigkeit des Objektes schießt es ein wenig über die Linie hinaus und dringt ein kleines Stück in die verbotene Zone ein.

Zur Startvorbereitung werden anschließend die Eigenschaften (Parameter) der drei benutzten Objekte festgelegt. Dabei stoßen Sie gleich auf eine weitere, bisher unbekannte Anweisung.

OBJECT.AX Objektnummer, Beschleunigungswert
OBJECT.AY Objektnummer, Beschleunigungswert
legt die Rate (Beschleunigung) fest, mit der sich die Geschwindigkeit eines Objektes (Sprites oder BOBs) mit der Kennung *Objektnummer* in X- beziehungsweise Y- Richtung ändert

Ja, Sie haben richtig gelesen! Amiga-Basic stellt Ihnen sogar einen Befehl zur Verfügung, mit dem Sie eine kontinuierliche Beschleunigung von Objekten erreichen können. Der Beschleunigungswert wird – wie die Geschwindigkeit – in Bildpunkten pro Sekunde eingegeben. In diesem Beispiel bekommt das Objekt *ang* in der X-Richtung eine Beschleunigung von 3 und in der Y-Richtung -1. Nachdem das zweite Objekt, eine angreifende Rakete, gestartet wurde, gelangt das Programm in eine WHILE-Schleife.

```
anfang:
ton=0: COLLISION OFF
c2=COLLISION(0):c3=COLLISION(c2)
c4=COLLISION(0):c5=COLLISION(c4)
```

```
OPEN"lkw"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE #1
OPEN"ang"FOR INPUT AS 1
OBJECT.SHAPE 2,INPUT$(LOF(1),1)
CLOSE #1
OPEN"abw"FOR INPUT AS 1
OBJECT.SHAPE 3,INPUT$(LOF(1),1)
CLOSE #1
ON COLLISION GOSUB zusammen:COLLISION ON
OBJECT.CLIP (0,20)-(620,190)
OBJECT.X 1,10:OBJECT.Y 1,167:OBJECT.X 2,10:OBJECT.Y 2,165
OBJECT.X 3,500:OBJECT.Y 3,165:OBJECT.VX 2,40:OBJECT.VY 2,-10
OBJECT.AX 2,3:OBJECT.AY 2,-1
OBJECT.ON : FOR i=1 TO 10000:NEXT:ton=1:n=600:OBJECT.START 2
zeiger =1
LINE (500,180)-(515,180),1
WHILE zeiger
  IF STRIG(3)=-1 THEN
  OBJECT.VX 3,0:OBJECT.VY 3,-10
  OBJECT.AY 3,-50:GOSUB rakton1
  OBJECT.START 3
  END IF
  IF ton=1 THEN GOSUB rakton
  LOCATE 5,50:PRINT tref
  LOCATE 5,70:PRINT fehl
  IF tref>=10 OR fehl>=10 THEN zeiger =0
WEND
OBJECT.CLOSE:CLS
IF tref>=20 THEN
  LOCATE 10,1:PRINT "Sie haben "tref"Raketen abgeschossen
  ** und damit den Feind zur Aufgabe gezwungen"
ELSE
  LOCATE 10,1:PRINT "Sie haben mit"fehl"Raketen ihren
  ** Vorrat verbraucht und sind verloren"
END IF
```

ende:END

In dieser WHILE-Schleife wird zuerst der Feuerknopf des Joysticks abgefragt. Wird dieser gerade gedrückt, erhält die Angriffsrakete ihre Startwerte und wird pfeifend nach oben gejagt. Anschließend wird in die Ton-Routine *rakton* verzweigt. Die Anzahl der Treffer und der Fehlschüsse wird dann angezeigt. Wenn eine der beiden entsprechenden Variablen den Wert 10 erreicht hat, wird die Schleife verlassen. Anstelle der Zahl 10 können Sie natürlich auch einen größeren Wert eingeben, um die Spieldauer zu verlängern. Sobald die Schleife verlassen wird, werden die Objekte nicht nur vom Bildschirm, sondern auch aus dem Speicher eliminiert und das Programm wird mit der Anzeige der Treffer oder Fehlschüsse beendet.

OBJECT.CLOSE Objektnummer1 ,Objektnummer2,...
gibt den von einem oder mehreren Objekten belegten Speicherplatz frei

Dieser Befehl sollte in einem Programm, das BOBs verwendet, niemals zum Abschluß fehlen. Dadurch gehen Sie sicher, daß Sie keinen Speicherplatz unnötig verschwenden.

Bislang war dies aber nur ein recht grober Überblick über die Programmfunktionen. Nun kommen wir zur ersten Subroutine *zusammen.* Sie enthält ebenfalls eine neue Funktion, die Aufschluß darüber gibt, was genau bei einer Kollision passiert ist. Amiga-Basic speichert alle Daten einer Kollision in einer Art Warteschlange ab. Jede neue Kollision wird in diese Warteschlange eingetragen und verbleibt dort so lange, bis sie mit der COLLISION-Funktion daraus entfernt wird.

v=COLLISION(n)
je nach Wert *n* werden verschiedene Objekt-Parameter übergeben

Aufruf	Ergebnis
COLLISION(0)	Kennummer des Objektes, welches zuletzt mit einem anderen Objekt zusammengestoßen ist. Der Inhalt der Kollisions-Warteschlange wird durch diesen Aufruf nicht verändert.
COLLISION(0)	Kennummer des Fensters, in dem der durch COLLISION(0) festgestellte Zusammenstoß stattgefunden hat.
COLLISION(n > 0)	Kennung des Objektes, das mit dem Objekt n zusammengestoßen ist. Gleichzeitig wird diese Information aus der Kollisions-Warteschlange gelöscht.

Tabelle 4.6: *Mögliche Anwendungen der COLLISION-Funktion*

Ergebnis	Bedeutung
-1	Objekt stieß an obere Fensterbegrenzung
-2	Objekt stieß an linke Fensterbegrenzung
-3	Objekt stieß an untere Fensterbegrenzung
-4	Objekt stieß an rechte Fensterbegrenzung
n > 0	Objekt stieß mit Objekt n zusammen

Tabelle 4.7: *Bedeutung der Ergebnisse der COLLISION-Funktion*

Soweit die Theorie, und nun zur Praxis. Normalerweise holt man sich zunächst mit COLLISION(0) die Information über das letzte Kollisions-Ereignis. Mit diesem Wert kann man dann weiterfragen. Das hier vorgestellte Programm geht genauso vor. Zuerst wird erfragt, ob ein Objekt mit der Fensterbegrenzung zusammengestoßen ist (c1<0). Das war dann ein schlechter Schuß. Die zweite Abfrage gilt den beiden Raketen, den Objekten Nummer 2 und 3. Ist das Ergebnis positiv, fand ein Abschuß einer der beiden Raketen statt. So sollte(!) es jedenfalls funktionieren.

Praktisch sieht es leider so aus, daß die Kollisions-Abfrage zu langsam reagiert. Zu langsam jedenfalls für die Geschwindigkeit, mit der die Sprites über den Bildschirm flitzen können. In diesem Programm kommt noch erschwerend dazu, daß die Treffer genau in einer Fensterecke registriert werden. Es kommen also fast regelmäßig drei Kollisions-Ereignisse zusammen. Wenn Sie bessere Treffer-Resultate erreichen wollen, sollten Sie den Wert für die Beschleunigung des Objektes *abw* reduzieren. Jetzt verstehen Sie auch, wieso bei *anfang* zweimal Werte aus der Kollisions-Warteschlange geholt werden. Die eine Abfrage dient nur dazu, ein (uninteressantes) Ereignis aus der Schlange zu entfernen.

```
zusammen:
c0=COLLISION(0):c1=COLLISION(c0)
IF c1=<0 AND c0>0 THEN
  LOCATE 5,1:PRINT "Fehlschuss ":ton=0:fehl=fehl+1
  GOTO anfang
END IF
IF c1=2 OR c1=3 AND c0>0 THEN
  LOCATE 5,1:PRINT "Treffer " :ton=0:tref=tref+1
  FOR i=1 TO 10:BEEP:NEXT:GOTO anfang
END IF
RETURN

fehler:
IF ERR<>53 THEN PRINT "Fehlernummer: "ERR:RESUME ende
RESUME konstruieren
```

```
konstruieren:
CLOSE:DIM spr(98):n=98:namen$="lkw":GOSUB speichern
namen$="ang":GOSUB speichern
n=61:namen$="abw":GOSUB speichern:GOTO anfang

speichern:
FOR i = 1 TO n:READ spr(i):NEXT
OPEN namen$ FOR OUTPUT AS 1
  FOR i = 1 TO 5: PRINT #1,MKL$(spr(i));:NEXT i
  FOR i = 6 TO n:PRINT #1, MKI$(spr(i));:NEXT
CLOSE#1: RETURN

spr:
DATA 0,0,2,35,15,24,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-6343
DATA -12685,-32768,-6343,-12685,-32768,0,0,0,0,0,0,0,0,0
DATA 0,0,0,0,0,0,0,30,0,0,255,-32768,-16384,8049,-32768
DATA 24579,-7951,-32768,12348,127,-32768,8128,127,-32768
DATA -1,-1,-16384,-1,-1,-16384,-6343,-12685,-32768
DATA -6343,-12685,-32768
DATA 0,0,2,35,15,24,3,0,0,1023,-32768,0,8191,0,1,-2,0,15
DATA -4,0,255,-8,0,2047,-64,0,16383,-512,0,16383,-8192,0
DATA 4094,0,0,2016,0,0,896,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 256,0,0,384,0,0,192,0,0,96,0,0,16,0,0,0,0,0,0,0,0
DATA 16384,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
DATA 0,0,2,16,25,25,3,0,0,0,0,0,0,896,896,1984,4064,896
DATA 896,896,896,896,896,896,896,896,896,1984,4064,8176
DATA 16376,32764,-2,256,896,896,896,896,0,0,1088,3168,0
DATA 0,0,0,0,0,0,0,0,0,1088,3168,7280,15480,31868,-898
DATA 879,2406,3855

rakton: n=n+5:IF n>680 THEN ton=0
ze=2000/n:SOUND n,ze,150,1

RETURN

rakton1:
SOUND 1100,10,255,2:ton=0
RETURN

puls: FOR i=1 TO b1:we(i)=-127: NEXT i
FOR i=1 TO b2:we(i)=127: NEXT i
FOR i=1 TO b1:we(i)=-127: NEXT i
RETURN
```

Zum Schluß folgen die Routinen zur Konstruktion und Abspeicherung der Objekte sowie die Tonausgabe, was beides bereits bei anderen Programmen besprochen wurde.

Betrachten Sie nun einmal die Gegenüberstellung von BOBs und Sprites am Kapitel-Anfang des Programmes »Abschuß«. Wir haben in diesem Programm zwei BOBs, die Angriffsrakete und die fahrbare Abschußrampe und ein Sprite, die Abwehrrakete. Der

Geschwindigkeitsvergleich fällt eindeutig zugunsten des Sprites aus. Daß beim Größenvergleich die BOBs nach Längen siegen, läßt sich nicht bestreiten. Daß die Breite bei Sprites auf 16 Pixel begrenzt sei, stimmt allerdings nicht. Das kann nur jemand behaupten, der noch nie ein Sprite gesehen hat. (Darum wohl auch der Fehler im Handbuch.) Schauen Sie sich einmal die Abwehrrakete genau an. Die ist genau doppelt so breit, also 32 Pixel. Um dies noch deutlicher zu demonstrieren, habe ich eine Zeile ins Programm eingebaut, die mit dem Ablauf überhaupt nichts zu tun hat. Eine Zeile vor der inneren WHILE-Schleife wird eine Linie gezogen, die direkt unter dem Sprite der Abwehrrakete liegt und 16 Pixel breit ist. Dieser direkte Vergleich wird auch den letzten Zweifler überzeugen, daß dieses Sprite breiter als 16 Pixel sei. Andererseits stimmt die Aussage im Handbuch aber auch wieder, da ein Sprite wirklich nur 16 Pixel breit ist. Diese Pixel werden aber doppelt so breit dargestellt, wenn ein Bildschirm mit 640 Punkten pro Zeile verwendet wird.

Ein weiterer Aspekt beim Vergleich der Objekt-Typen ist die Farbdarstellung der Sprites. Lassen Sie dazu das Programm noch einmal ablaufen und schauen Sie die Abwehrrakete genau an. Obwohl nur ein Bildschirm mit vier Farben verwendet wird, zeigt das Sprite drei zusätzliche Farben – BOBs können das nicht. Und über das deutlich sichtbare Flimmern der BOBs, selbst bei recht kleinen Objekten, brauche ich wohl auch nichts mehr zu sagen.

Die Bewegung von Objekten auf dem Bildschirm bezeichnet man häufig als »Animation«. Schauen wir doch einmal ins Lexikon, was dort unter Animation nachzulesen ist: »Animation – filmtechnisches Verfahren, unbelebten Objekten (z.B. gezeichneten Figuren) im Trickfilm Bewegung zu verleihen.« Nun, genau das haben die Programme bisher nicht fertiggebracht. Das einzige, was sie zustande gebracht haben, war eine mehr oder weniger elegante Verschiebung von nicht veränderlichen Objekten. Aber Sie werden sehen, auch die »echte« Animation ist gar nicht so schwer.

Starten Sie dazu den Objekt-Editor und wählen Sie das Editieren von Sprites. Stellen Sie die Sprite-Höhe dann auf 31 Pixel ein und schalten Sie gleich auf die 4x4-Vergrößerung. Nun versuchen Sie bitte, ein Männchen in der Seitenansicht zu zeichnen. Für die Wiedergabe macht es sich ganz gut, wenn Sie die beiden Beine in verschiedenen Farben zeichnen. Denken Sie bitte daran, daß das Sprite hinterher auf dem Bildschirm doppelt so breit wird – malen Sie also ein möglichst schlankes Männlein. Wenn Sie mit Ihrem Kunstwerk zufrieden sind, speichern Sie es bitte unter dem Namen »fg1« ab. Nun laden Sie bitte mit dem Befehl *Open* wieder das Bild. Ändern Sie das Männchen nun. Lassen Sie dabei zunächst den Oberkörper unverändert und modifizieren Sie nur die Beine. Fertigen Sie nacheinander fünf Modifikationen an, die das Männchen in verschiedenen Phasen des Gehens zeigen und speichern Sie diese unter den Namen »fg2« bis »fg6« ab.

Verlassen Sie dann den Objekt-Editor und legen Sie die sechs Icons mit den Sprites zusammen mit dem folgenden Programm in eine andere Schublade. (Wenn Sie die Icons nicht sofort sehen, brauchen Sie nur das Fenster BASICDemos einmal zu- und dann wieder aufzumachen.)

```
REM Bewegung
REM P.4.7-2
```

```
DEFINT a-z
OPEN"fg1"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1):CLOSE #1
OPEN"fg2"FOR INPUT AS 1
OBJECT.SHAPE 2,INPUT$(LOF(1),1):CLOSE #1
OPEN"fg3"FOR INPUT AS 1
OBJECT.SHAPE 3,INPUT$(LOF(1),1):CLOSE #1
OPEN"fg4"FOR INPUT AS 1
OBJECT.SHAPE 4,INPUT$(LOF(1),1):CLOSE #1
OPEN"fg5"FOR INPUT AS 1
OBJECT.SHAPE 5,INPUT$(LOF(1),1):CLOSE #1
OPEN"fg6"FOR INPUT AS 1
OBJECT.SHAPE 6,INPUT$(LOF(1),1):CLOSE #1
FOR i = 1 TO 6:OBJECT.Y i,50:NEXT
FOR i = 1 TO 6:OBJECT.HIT i,0,0:NEXT
OBJECT.X 6,9
zeiger=-1:x2=0
OBJECT.VX 6,10:OBJECT.ON 6:OBJECT.START 6
WHILE zeiger
  x2=x2+1:IF x2=7 THEN x2=1
  x3=x2-1:IF x3=0 THEN x3=6
  OBJECT.X x2,OBJECT.X(x3):OBJECT.ON x2
  OBJECT.VX x2,OBJECT.VX(x3) :OBJECT.START x2
  OBJECT.OFF x3
  IF OBJECT.X(x2)>600 THEN zeiger=0
WEND
OBJECT.CLOSE
END
```

Starten Sie dann das Programm und betrachten Sie Ihre erste Animation. Vielleicht ist sie noch nicht ganz perfekt, aber Sie können sie jederzeit verbessern, indem Sie die Sprites ändern. Das Programm selbst ist recht einfach. Mit OBJECT.HIT werden zunächst die Fremd- und Selbstwerte aller Objekte auf Null gesetzt. Theoretisch ist dieser Befehl unnötig, da ja gar keine Kollisionen abfragt werden. Praktisch aber verhindert er ein leichtes Rucken der zeitweise übereinanderliegenden Sprites. In der darauffolgenden WHILE-Schleife werden nacheinander die Sprites in der Reihenfolge fg1 bis fg6 in Bewegung gesetzt und zyklisch an- und ausgeschaltet. Der rasche Wechsel zwischen den Sprites sieht dann so aus, als würde sich ein Sprite ändern beziehungsweise bewegen. Für die reibungslose Übergabe der Bewegung von einem Sprite auf den anderen sorgt eine weitere neue Funktion.

v=OBJECT.VX (Objektnummer)

v=OBJECT.VY (Objektnummer)

liefert die Geschwindigkeit des Objekts (Sprites oder BOBs) mit der Kennung *Objektnummer* in Pixeln pro Sekunde

Im Programm wird diese Funktion so eingesetzt, daß die Geschwindigkeit des vorhergehenden Sprites der Bildfolge als Geschwindigkeit für den neuen Sprite übernommen wird. Dies kann zum Beispiel wie folgt aussehen:

```
OBJECT.VX x2, OBJECT.VX(x3): REM x2 erhält Geschw. von x3
```

Damit soll die Behandlung von Sprites und BOBs zunächst einmal abgeschlossen werden. Wenn Sie mehr über BOBs und Sprites wissen wollen, finden Sie in Kapitel 5.6 noch ein paar Leckerbissen zu diesem Thema.

4.8 Da rollt der Bildschirm

Wenn Sie Ihre Programme im List-Fenster erstellen, ist es für Sie wahrscheinlich selbstverständlich, daß sich der Bildschirminhalt in jede der vier möglichen Richtungen bewegen läßt – mit den Workbench-Fenstern ist das ja auch möglich. Dabei ist das gar nicht so selbstverständlich. Bei vielen Computern kann sich der Bildschirminhalt nur nach oben oder unten bewegen (man sagt auch »rollen« oder »scrollen« zu diesem Vorgang). Aber auch hierfür bietet Amiga-Basic den SCROLL-Befehl, der mit wenig Aufwand entsprechende Funktionen in selbstgeschriebenen Programmen realisierbar macht. Probieren Sie ihn aus.

SCROLL (x1,y1)-(x2,y2), X, Y

verschiebt den Inhalt des Rechtecks (x1,y1)-(x2,y2) im aktiven Fenster *X* Punkte nach rechts und *Y* Punkte nach unten

```
REM Roll-Test
'P.4.8-1
FOR i=1 TO 23
  PRINT "Das ist ein Test um die scroll-Moeglichkeiten zu
** demonstrieren."
NEXT
SCROLL (200,50)-(400,100),0,-50
```

Das sieht doch schon recht nett aus, oder? Ich verspreche Ihnen aber, daß Sie Ihre Meinung bald ändern werden. Sie haben an diesem Beispiel schon gelernt, daß das Scrollen so schnell vor sich geht, daß man es in einem sinnvollen Programm abbremsen muß, weil sonst überhaupt nichts mehr zu sehen ist. Bevor Sie das probieren, sollten Sie noch ein wenig mit dem ersten Programm herumexperimentieren, denn dazu ist es gedacht. Ändern Sie die Werte für die Verschiebung X- und Y-Richtung. Sie werden feststellen, daß bei einigen Richtungen die Verschiebung im angegebenen Bereich bleibt, bei anderen dagegen werden nicht vorhergesehene Bildausschnitte mitverschoben.

Leider beschränkt sich SCROLL nur auf den sichtbaren Bildschirmbereich. Sie können damit also auch keinen größeren Bereich verschieben als den, der durch die Fenstergröße vorgegeben ist – also zum Beispiel einen kleinen Teil eines sehr großen Bildes im Fenster darstellen.

Nun aber zur Verlangsamung der Rollbewegung, die für viele Zwecke unbedingt benötigt wird. Diese erreicht man am einfachsten, indem man immer nur kleine Bewegungen in die gewünschte Richtung vornimmt und zwischen diesen eventuell kleine Pausen einlegt.

```
REM Hochhaus
P.4.8-2
LINE (200,10)-(400,140),1,bf
LOCATE 3,30:PRINT "Haus der Konkurrenz"
FOR j=30 TO 130 STEP 10
  FOR i=215 TO 375 STEP 20
  LINE (i,j)-(i+10,j+5),0,bf
  NEXT
NEXT
FOR i = 1 TO 18
  SCROLL (180,0)-(420,140),0,i
NEXT
```

Die Verschiebung des Hochhauses nach unten wurde in diesem Beispiel durch eine FOR-Schleife erreicht. Die Bewegung in Y-Richtung erfolgt durch die Variable *i* des Schleifenzählers. Dadurch wird eine Beschleunigung des SCROLL-Vorganges erreicht, da der Wert dieser Variablen kontinuierlich ansteigt.

Wenn Sie versuchen sollten, das Haus durch eine entgegengesetzte Scrollbewegung zurückzuholen, haben Sie Pech gehabt. Ein einmal verschobener Bereich kann, wenn er erst einmal unsichtbar ist, nicht mehr zurückgeholt werden. Wenn aber nur soweit gescrollt wird, daß das gezeichnete Bild noch zu sehen ist, kann es natürlich anschließend wieder zurückgeholt werden. Mit dieser Idee läßt sich ein kleines Spiel aufbauen, wie das folgende Programm zeigt.

```
REM softball
'P.4.8-3
CIRCLE (20,90),20,1
PAINT (20,90),1
FOR j = 1 TO 8
  FOR i = 1 TO 10
  SCROLL (0,0)-(620,160),j,i
  NEXT
  FOR i = 1 TO 10
  SCROLL (0,0)-(620,160),j,-i
  NEXT
NEXT
```

Dieses einfache Programm läßt erahnen, welche Möglichkeiten im SCROLL-Befehl stecken. Experimentieren Sie ruhig ein wenig damit. Ändern Sie zum Beispiel einmal die Schleifenzähler in

```
FOR j = 14 TO 1 STEP -2
```

und

```
FOR i = 1 TO j
```

oder ändern Sie in den SCROLL-Anweisungen das *j* in *j/2* und in den CIRCLE-und PAINT-Anweisungen die Angaben in *(20,30)*. Der Ball springt nun den Berg hinauf und verringert so lange seine Sprunghöhe, bis er schließlich liegenbleibt. Das nächste Programmbeispiel, das wieder intensiv den SCROLL-Befehl nutzt, versetzt Sie nach Cap Canaveral (oder nach Baikonur).

```
REM Fehlstart
'P.4.8-4
DEFINT f
DIM flmust(3)
flmust(0)=&H5555
flmust(1)=&HAAAA
flmust(2)=&H5555
flmust(3)=&HAAAA
PATTERN &HFFFF,flmust
AREA (300,110)
AREA STEP (5,10)
AREA STEP (0,40)
AREA STEP (8,10)
AREA STEP (-26,0)
AREA STEP (8,-10)
AREA STEP (0,-40)
AREAFILL
FOR i = 1 TO 15 STEP .5
SCROLL (270,0)-(330,175),0,-i
NEXT
AREA (300,60)
AREA STEP (5,-10)
AREA STEP (0,-40)
AREA STEP (8,-10)
AREA STEP (-26,0)
AREA STEP (8,10)
AREA STEP (0,40)
AREAFILL 1
FOR i = 1 TO 12 STEP .5
SCROLL (270,0)-(330,180),0,i
NEXT
```

Die Rakete sieht zwar etwas komisch aus, dafür demonstriert sie aber recht schön den PATTERN-Befehl, der bislang nur einmal kurz gestreift wurde.

PATTERN Linienmuster, Feldmuster

legt ein oder zwei Muster fest, die danach von Amiga-Basic beim Ziehen von Linien beziehungsweise Füllen von Flächen statt einer »soliden« Farbe verwendet werden

Dabei muß das *Linienmuster* in Form einer kurzen Ganzzahl (16 Bit) angegeben werden. Diese 16 Bit legen fest, welche Punkte auf der Linie gezeichnet werden und welche frei bleiben. Jedes Bit im *Linienmuster*, das gesetzt (gleich 1) ist, bedeutet, daß ein Punkt in der Linie gezeichnet wird.

Als *Feldmuster* muß ein Ganzzahlenfeld angegeben werden, welches das Füllmuster enthält. Der Wert jedes Elementes dieses Feldes wird wieder als Folge von 16 Bit aufgefaßt, die jeweils eine Zeile des Musters beschreiben. Ein gesetztes Bit im Element entspricht einem farbigen Punkt im Muster, nicht gesetzte Bits (Punkte) bleiben frei. Damit ist die Breite eines Musters auch schon geklärt: 16 Bildschirmpunkte. Die Höhe des Musters hängt von der Länge des Feldes ab. Möglich sind hierfür alle Potenzen von zwei (1, 2, 4, 8, 16, 32 und so weiter).

Muster müssen also immer in Form von Zahlen angegeben werden. Dezimalzahlen eignen sich aber ziemlich schlecht dafür, da kaum eine erkennbare Beziehung zwischen Ziffern und den entsprechenden Bits besteht. Anders ist das bei Hexadezimalzahlen und Oktalzahlen, mit denen Amiga-Basic ebenfalls arbeiten kann.

Bezeichnung:	Ziffern:	Anzahl:	Kennung:	Beispiel:
Oktalzahl	0 bis 7	max. 6	&O oder &	&O777777
Hexadezimalzahl	0 bis F	max. 4	&H	&HFFFF

Tabelle 4.8: *Oktal- und Hexadezimalzahlen*

Hexadezimalzahlen entsprechen folgenden dezimalen Zahlen:

Hexadezimal:	**Dezimal:**
0 bis 9	0 bis 9
A	10
B	11
C	12
D	13
E	14
F	15
10	16
11	17
...	...
FFFF	65535

Tabelle 4.9: *Hexadezimalziffern und ihre Bedeutung*

Für Amiga-Basic ist es ganz egal, in welchem Zahlensystem Sie Zahlen angeben – Sie müssen ihm nur (mit Hilfe der Kennung) mitteilen, welches System Sie verwenden wollen. Dabei können auch in einer Formel mehrere verschiedene Zahlensysteme verwendet werden. Geben Sie doch einmal ein paar Beispiele ein:

```
dezimal       ? SQR(30625)      Ergebnis: 175
oktal         ? SQR(&73641)     Ergebnis: 175
hexadezimal   ? SQR(&H77A1)     Ergebnis: 175
```

(SQR berechnet die Quadratwurzel). Alle drei Ziffernfolgen in diesen Beispielen entsprechen derselben Zahl und führen deshalb zum selben Ergebnis. Nur die Beschreibung der Zahlen ist unterschiedlich. Das Ergebnis einer Berechnung gibt der Amiga jedoch immer dezimal aus.

Probieren Sie nun noch zwei Beispiele:

```
dezimal       ? 345/58*89       Ergebnis: 529.3965
gemischt      ? 345/&H3A*&131   Ergebnis: 529.3965
```

Sie sehen also, im Prinzip können Sie jederzeit das Zahlensystem verwenden, das am besten geeignet erscheint. Aber man kann Zahlen nicht nur in verschiedenen Zahlensystemen eingeben, sondern diese auch ausgeben. Hierzu stehen zwei Funktionen zur Verfügung, die einen Wert in einen String umwandeln, der die Zahl in dem gewünschten Zahlensystem enthält.

v$=OCT$(numerischer Ausdruck)
liefert einen String, der den Wert des *numerischen Ausdrucks*, dargestellt als Oktalzahl, enthält

v$=HEX$(numerischer Ausdruck)
liefert einen String, der den Wert des *numerischen Ausdrucks*, dargestellt als Hexadezimalzahl, enthält

Bei beiden Funktionen werden eventuell vorhandene Dezimalstellen nicht berücksichtigt, sondern nur der ganzzahlige Anteil einer Zahl umgewandelt. Der Vollständigkeit halber listet die folgende Tabelle noch einmal alle Umrechnungsmöglichkeiten in gebündelter Form auf.

Umrechnung:	Basic-Befehl:
dezimal in hexadezimal	PRINT HEX$(Dezimalzahl)
hexadezimal in dezimal	PRINT &H Hexadezimalzahl
dezimal in oktal	PRINT OCT$(Dezimalzahl)
oktal in dezimal	PRINT &O Oktalzahl
oktal in hexadezimal	PRINT HEX$(&O Oktalzahl)
hexadezimal in oktal	PRINT OCT$(&H Hexadezimalzahl)

Tabelle 4.10: *Umwandlungen zwischen den verschiedenen Zahlensystemen*

Damit haben Sie die Grundlage, leichter mit der PATTERN-Anweisung arbeiten zu können. Geben Sie zur Erholung aber zunächst einmal das folgende Programm ein.

```
REM Muster
'P.4.8-5
DEFINT f
DIM flmust(3)
flmust(0)=&H1010
flmust(1)=&H3838
flmust(2)=&H7C7C
flmust(3)=&HFEFE
PATTERN &H9249,flmust 'erste Anweisung
LINE (50,10)-(200,10),1
LINE (100,50)-(200,150),1,bf
ERASE flmust
DIM flmust(7)
flmust(0)=&O0
flmust(1)=&O1770
flmust(2)=&O3114
```

```
flmust(3)=&H3E4F
flmust(4)=&H7FFF
flmust(5)=32767
flmust(6)=6156
flmust(7)=0
PATTERN &HFFFF,flmust ' zweite Anweisung
LINE (50,20)-(200,20),3
CIRCLE (350,90),85,3
PAINT (350,90),3
AREA (400,5)
AREA STEP (200,0)
AREA STEP (0,180)
AREAFILL
ERASE flmust
END
```

Dieses Programm stellt zwei Linien mit verschiedenen Linienmustern dar und füllt drei geometrische Objekte mit zwei verschiedenen Flächenmustern aus. Wenden wir uns zuerst dem Linienmuster zu. Dieses wird mit *PATTERN &H9249* festgelegt. Die entsprechende 16-Bit-Zahl sieht wie folgt aus:

Hexadezimalzahl:	9	2	4	9	
Bitfolge:	1 0 0 1	0 0 1 0	0 1 0 0	1 0 0 1	
Dezimalwert:	2^15 + 2^12	2^9	2^6	2^3 + 2^	= 37 449

An jeder Stelle der Maske, an der eine 1 steht, ist das Bit gesetzt. An diesen Stellen wird auch ein Punkt gesetzt. Die mit der PATTERN-Anweisung erzeugte Linie sieht also etwa so aus:

Punkt, leer, leer, Punkt, leer, leer, Punkt usw.

Um den entsprechenden Dezimalwert auszurechnen, müssen Sie die einzelnen Werte, denen die Bit entsprechen, zusammenzählen: 2^15+2^12+2^9+2^6+2^3+2^0. Heraus kommt 37449. Diese Zahl hat nur wirklich nicht viel mit dem sichtbaren Strich-Punkt-Muster der Linien zu tun und verlangt einige Rechnerei. Einfacher geht es mit Hexadezimalzahlen. Hierzu werden die 16 Bit des Musters in 4 »Nibbles«, Gruppen von vier Bit, aufgeteilt. In jedem Nibble werden nun getrennt die zu den vier Bit gehörenden Dezimalzahlen (16, 8, 4, 2 und 1) addiert und die dabei herauskommende höchstens zweistellige Dezimal**zahl** (zwischen 0 und 15) in eine Hexadezimal**ziffer** (zwischen 0 und F) umgewandelt.

	4.Nibble	3.Nibble	2.Nibble	1.Nibble
Bit-Maske	1 0 0 1	0 0 1 0	0 1 0 0	1 0 0 1
Summen dezimal	9	2	4	9
Hexadezimalziffer	9	2	4	9

Aus der Bitfolge kann so ohne komplizierte Berechnungen die entsprechende Hexadezimalzahl gebildet werden – oder zumindest sind die Berechnungen wesentlich einfacher als die für die Ermittlung der Dezimalzahl nötigen. Um diesen Vorgang zu verdeutlichen, soll er noch einmal anhand des zweiten Linienmusters gezeigt werden. Dieses ergibt eine durchgezogene Linie (ohne freie Punkte).

	4.Nibble	3.Nibble	2.Nibble	1.Nibble
Bit-Maske	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
Summen dezimal	16 16	16	16	
Hexadezimalziffer	F F	F	F	

Jetzt, wo Sie wissen, wie einfach das bei Linienmustern funktioniert, können Sie sich auch an die Berechnung der Felder für die Füllflächen wagen. Hierzu nehmen Sie bitte ein kariertes Papier zur Hand. Zeichnen Sie zuerst das Pfeilmuster, das im Programm zum Füllen des Rechtecks verwendet wird. Es ist in der ersten PATTERN-Anweisung definiert. Zur besseren Übersicht werden in der folgenden Tabelle nichtgesetzte Bits als Punkte gezeichnet und gesetzte Bits mit einem x versehen.

	8 4 2 1	8 4 2 1	8 4 2 1	8 4 2 1	
0. Feld-Element	. . . x	. . . .	. . . x	. . . .	&H1010
1. Feld-Element	. . x x	x . . .	. . x x	x . . .	&H3838
2. Feld-Elemen	. x x x	x x . .	. x x x	x x . .	&H7C7C
3. Feld-Element	x x x x	x x x .	x x x x	x x x .	&HFEFE

Schon haben wir unsere erste Füllfläche konstruiert. Da das so kinderleicht war, nehmen wir uns gleich das Auto vor. Das Muster dafür ist acht Elemente hoch. Sie erinnern sich, für die Höhe der Muster können wir nur Zahlen mit Zweier-Potenzen verwenden. Damit Sie sehen, daß die Konstruktion mit Oktal-Zahlen noch leichter geht, finden Sie bei der folgenden Konstruktion beide Möglichkeiten:

hexadezimal	8 4 2 1 8 4 2 1 8 4 2 1 8 4 2 1	
0. Feld-Element	. . . . .	&H0 &O0
1. Feld-Element	. . . . . x x x x x x x x . . .	&H3F8 &O1770
2. Feld-Element	. . . . . x x . . x . . x x . .	&H64C &O3114
3. Feld-Element	. . x x x x x . . x . . x x x x	H3E4F &O37117
4. Feld-Element	. x x x x x x x x x x x x x x x	&H7FFF &O77777
5. Feld-Element	. x x x x x x x x x x x x x x x	&H7FFF &O77777
6. Feld-Element	. . . x x . . . . . x x . .	&H180C &O14014
7. Feld-Element	. . . . .	&H0 &O0
oktal	1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1	

Die Ermittlung der zu einer Bitfolge gehörenden Oktalzahl funktioniert genauso wie bei den hexadezimalen Zahlen. Nur wird die 16-Bit-Zahl hierzu in fünf Dreiergrupen (und ein einzelnes Bit links) aufgeteilt.

Für welches Zahlensystem Sie sich auch entscheiden, Sie können jedenfalls mit der PATTERN-Anweisung viele effektvolle Bilder gestalten. Kehren wir nun wieder zum SCROLL-Befehl zurück. Nach den einfachen Beispielen am Anfang des Kapitels stellt das folgende Programm schon ein praktisches Beispiel dar. Man könnte es zum Beispiel recht gut für die Einleitung eigener Programme verwenden.

```
REM Vorspann
'P.4.8-6
LINE (0,0)-(615,186),3,bf
FOR i=60 TO 560 STEP 80
  x=i:y=i/5:stern x,y
NEXT
FOR i = 1 TO 10
  READ zei%(i)
NEXT i
zeit
FOR i = 1 TO 16.5 STEP .5
SCROLL (0,0)-(310,190),-i,0
SCROLL (311,0)-(620,190),i,0
NEXT
x2=0:y2=35:a=1:v%=16:fa=3:GOSUB zeichnen
zeit
FOR i = 1 TO 8 STEP .5
SCROLL (0,0)-(620,90),0,i
SCROLL (0,90)-(620,190),0,-i
NEXT
x2=0:y2=35:a=6:v%=16:fa=1:GOSUB zeichnen
zeit
FOR i = 1 TO 17.5 STEP .5
SCROLL (0,0)-(310,190),i,0
SCROLL (311,0)-(617,190),-i,0
NEXT
END

SUB stern(a,b) STATIC
AREA (a,b)
AREA STEP (10,20)
AREA STEP (50,5)
AREA STEP (-50,5)
AREA STEP (-10,20)
AREA STEP (-10,-20)
AREA STEP (-50,-5)
AREA STEP (50,-5)
AREAFILL
END SUB
```

```
SUB zeit STATIC
FOR i=1 TO 10000 :NEXT
END SUB
```

zeichnen:

```
CLS:PALETTE 2,0,0,.6:COLOR 2
FOR i = 46 TO 122: PRINT CHR$(i);:NEXT i:COLOR 1
FOR i = a TO a+4
x1=(zei%(i)-46)*8
FOR ho=0 TO 7
  FOR br=0 TO 6
  IF POINT(br+x1,ho)<>0 THEN
  LINE (v%*br+x2,v%*ho+y2)-(v%*br+x2+v%-1,ho*v%+v%-
  ** 1+y2),fa,bf
  END IF
  NEXT br
NEXT ho
x2=x2+(8*v%)
NEXT i
RETURN
DATA 65,77,73,71,65,122,101,105,103,116
```

Dieses Programm macht einen recht spektakulären Eindruck. Dabei verwendet es bis auf eine »Super-Subroutine« nichts weiter als den SCROLL-Befehl. Eine kurze Erläuterung soll aber trotzdem nicht verschwiegen werden. Zunächst füllt ein rotes Rechteck den Bildschirm vollständig aus und dann werden einige Sterne in dieses Rechteck gezeichnet. Der Stern wird in dem Unterprogramm *stern* konstruiert. Anschließend werden die DATA-Anweisungen eingelesen und das Rechteck wie ein Theatervorhang aufgerollt. Es folgt die Ausgabe eines Textes, der dann von einer gleichzeitigen Scroll-Bewegung von oben nach unten und von unten nach oben »aufgefressen« wird. Das gleiche geschieht mit dem zweiten Text, nur diesmal in horizontaler Richtung. Somit bin ich Ihnen nur noch die Erklärung der Routine *zeichnen* schuldig. Diese können Sie im Abschnitt 5.2 nachlesen.

Damit sind wir dann aber auch am Ende des dritten Kapitels angelangt. Ich hoffe, Sie hatten beim Lesen und Ausprobieren genausoviel Freude wie ich beim Schreiben und Programmieren.

5
Tips und Tricks

Wenn man ein Buch wie das vorliegende schreibt, legt man sich natürlich vorher ein bestimmtes Schema zurecht, beziehungsweise eine Gliederung, nach der man dann arbeitet. Man hat feste Vorstellungen von den einzelnen Abschnitten und Kapiteln und was darin enthalten sein muß. Im Laufe des Arbeitens ergibt sich dann der eine oder andere Gesichtspunkt, der eigentlich gar nicht in das Konzept paßt, den man aber gerne berücksichtigen möchte. Dieser 5. Abschnitt ist eine – recht ungeordnete – Sammlung solcher Punkte.

Weiterhin schreien einige Befehle des Amiga-Basic geradezu danach, daß man mit ihnen etwas anstellt, das normalerweise nicht vorgesehen ist. Für einen Programmierer wird dann die Angelegenheit erst interessant. Solche Manipulationen von Befehlen sind ebenfalls in diesem Teil des Buches zu finden.

Und schließlich läßt sich bei der großen Anzahl von Befehlen des Amiga-Basic nicht vermeiden, daß einige nicht richtig ins Konzept passen. Man könnte sie gewaltsam in ein halbwegs passendes Kapitel pressen. Besser sind jedoch diese Befehle, meiner Meinung nach, hier aufgehoben.

5.1 Disk ohne Diskette

Haben Sie schon einmal etwas von einer RAM-Disk gehört? Eine RAM-Disk ist in gewisser Hinsicht eine Diskette; allerdings keine wirkliche Diskette, die Sie in die Hand nehmen und aufheben können. Statt dessen wird diese Diskette durch Software simuliert, indem ein bestimmter Teil des RAM-Speichers reserviert und wie eine Diskette behandelt wird. Dadurch, daß sich hier keine mechanischen Teile bewegen, ist diese Diskette ungleich schneller als jede wirkliche Diskette. Sie hat allerdings auch einen großen

Nachteil: Daten, die darauf gespeichert werden, sind nach dem Abschalten des Computers oder nach einem System-Absturz nicht mehr vorhanden! Wenn Sie die schnelle Arbeitsweise der RAM-Disk nützen und trotzdem die Daten darauf behalten wollen, müssen Sie die Daten auf eine echte Diskette überspielen, bevor Sie den Amiga abschalten.

Damit Sie selbst feststellen können, wie schnell eine RAM-Disk ist, geben Sie nun bitte folgendes Programm ein und speichern Sie es unter dem Namen »RamDisk« ab.

```
REM RamDisk
'P5.1-1
WIDTH 60
a$(1)="Diesen Text werden wir nun nacheinander "
a$(2)="auf die Diskette und auf die RAM-Disk "
a$(3)="abspeichern und wieder einlesen. Mit der "
a$(4)="TIMER-Funktion werden wir die jeweils "
a$(5)="benoetigte Zeit nehmen und auf dem "
a$(6)="Bildschirm anzeigen."
FOR i = 1 TO 6
  PRINT a$(i)
  b$=b$+a$(i)
NEXT
PRINT
filename$="Testtext":GOSUB speichern
filename$="RAM:Testtext":GOSUB speichern
filename$="Testtext":GOSUB lesen
PRINT :PRINT c$
filename$="RAM:Testtext":GOSUB lesen
PRINT :PRINT c$
END

speichern:
v1=TIMER
OPEN filename$ FOR OUTPUT AS #1
 WRITE #1,b$
CLOSE #1
v2=TIMER
PRINT "Dauer der Speicherung in Sekunden: "v2-v1
RETURN

lesen:
v1=TIMER
OPEN filename$ FOR INPUT AS #1
 INPUT #1,c$
CLOSE#1
v2=TIMER
PRINT "Dauer des Lesevorganges in Sekunden: "v2-v1
RETURN
```

In diesem Programm wird nacheinander der gleiche Text einmal auf die normale Diskette

und einmal auf die RAM-Disk abgespeichert. In der Variablen *v1* wird dabei vor Beginn des Schreibvorgangs mit Hilfe der TIMER-Funktion die aktuelle Zeit festgehalten. Ist der Speichervorgang beendet, wird wieder die aktuelle Zeit festgestellt und in *v2* abgespeichert. Die Differenz der beiden Werte ist also die Zeit, die der Schreibvorgang jeweils benötigte. Beim Lesen des Textes geht das Programm dann genauso vor. Die folgende Tabelle zeigt die Ergebnisse, die ich dabei bekam (auf Ihrem Amiga können Sie eventuell leicht abweichende Zahlen erhalten).

	speichern	lesen
normale Diskette	7.50 sec	1.05 sec
RAM-Disk	0.18 sec	0.09 sec

Tabelle 5.1: *Zeitvergleich zwischen Diskette und RAM-Disk*

Wie Sie sehen, ergeben sich dabei schon recht beachtliche Unterschiede. Das nächste Programm speichert Grafiken auf der RAM-Disk ab und liest sie wieder ein. Achten Sie auch bitte besonders auf die Geschwindigkeit, mit der das Ganze abläuft.

```
REM Adventure
'P.5.1-2
WINDOW 2,,(0,0)-(618,50),0
AREA (100,50):AREA (500,50):AREA STEP (-30,-45)
AREA STEP (-200,0):AREAFILL
filename$="RAM:auto":GOSUB speichern:CLS
AREA (200,50):AREA (400,50):AREA (300,0):AREAFILL
filename$="RAM:pyramide":GOSUB speichern:CLS
AREA (200,50):AREA (300,50):AREA (300,15)
AREA (250,0) :AREA (200,15):AREAFILL
filename$="RAM:haus":GOSUB speichern:CLS
FOR i=20 TO 550 STEP 30
  LINE (i,0)-(i+5,50),3,bf
NEXT
filename$="RAM:leiter":GOSUB speichern:CLS
FOR i = 50 TO 550 STEP 50
  CIRCLE (i,25),20:PAINT(i,25)
NEXT :filename$="RAM:baelle":GOSUB speichern:CLS
WINDOW OUTPUT 1:CLS:LOCATE 10,1 :COLOR 3

spiel:
WINDOW OUTPUT 1:PRINT "Sie sehen nichts. 2-3"

eingabe:
INPUT a%:IF a%<1 OR a%>6 THEN eingabe
ON a% GOTO spiel,baelle,pyramide,leiter,auto,haus

baelle:
WINDOW 2:filename$="RAM:baelle":GOSUB einlesen4
```

```
WINDOW OUTPUT 1:PRINT "Sie sehen Baelle. 1-4":GOTO eingabe

pyramide:
WINDOW 2:filename$="RAM:pyramide":GOSUB einlesen4
WINDOW OUTPUT 1:PRINT "Sie sehen eine Pyramide. 1-4-5"
GOTO eingabe

leiter:
WINDOW 2:filename$="RAM:leiter":GOSUB einlesen4
WINDOW OUTPUT 1:PRINT "Sie sehen eine Leiter. 2-3-6"
GOTO eingabe

auto:
WINDOW 2:filename$="RAM:auto":GOSUB einlesen4
WINDOW OUTPUT 1:PRINT "Sie sehen ein Auto. 3-6"
GOTO eingabe

haus:
WINDOW 2:filename$="RAM:haus":GOSUB einlesen4
WINDOW OUTPUT 1:PRINT "Sie sehen ein haus. 5-4":GOTO
** eingabe
REM Hier Programm iffLesenSpeichern anhaengen.
```

Der Name »Adventure« für dieses Programm ist natürlich maßlos übertrieben. Das Einlesen der Bilder in einem wirklichen Adventure-Programm könnte aber auf die hier gezeigte Art und Weise praktiziert werden. Im ersten Teil des Programms werden zur Demonstration einige einfache Bilder gezeichnet. Sie können hier auch Bilder einsetzen, die Sie mit PAINTAmiga (oder Deluxe Paint oder Graficraft oder...) gezeichnet haben. Diese Bilder werden dann auf der RAM-Disk abgespeichert. Das eigentliche Programm beginnt bei der Sprungmarke *spiel.* Sie können dabei in die mit Zahlen angegebenen Bilder gelangen. Es dauert nur etwa zehn Sekunden, bis ein 50 Zeilen hohes Bild von der RAM-Disk eingelesen ist. Berücksichtigen Sie dabei bitte auch die Zeit, die Basic beziehungsweise die Routine *iffLesenSpeichern* benötigt, um das Bild zu decodieren.

5.2 Groß, größer, am größten

Das folgende Programm zeigt, wie Sie extrem große Buchstaben auf den Bildschirm des Amiga bringen können. Wenn Sie es abgetippt und ausprobiert haben, fallen Ihnen bestimmt einige Anwendungsmöglichkeiten ein. Man kann es zum Beispiel sehr gut für die Eröffnungsbilder eigener Programme verwenden (um zum Beispiel recht deutlich darauf aufmerksam zu machen, wer dieses Programmkunstwerk geschaffen hat). Das Programm ist, wie alle Programme, für eine Zeilenbreite von 80 Zeichen ausgelegt. Beim Abschreiben beachten Sie bitte, daß die "**" am Anfang einer Zeile an das Ende der vorhergehenden Zeile gehören.

Nach dem Start fragt das Programm zunächst nach der gewünschten Vergrößerung. Sie können hier Zahlen von 1 bis 20 eingeben. Bei 1 erhalten Sie keine Vergrößerung, bei 2

wird jedes Zeichen doppelt so hoch und breit dargestellt, bei 3 dreimal so groß und so weiter, bis zu 20, was wirklich riesige Buchstaben ergibt.

```
REM Das grosse ABC
'P.5.2-1

start:
GOSUB kopf
LOCATE 15,1:INPUT "Bitte geben Sie die gewuenschte
** Vergroesserung ein (1 bis 20): ";v%
IF v%>20 THEN start

lauf:
ko=1:x2=0:y2=20
GOSUB kopf
WHILE ko
  GOSUB taste
  GOSUB zeichnen
  GOSUB position
WEND
END

zeichnen:
x1=(ASC(ta$)-46)*8
IF x1=-264 OR x1=-112 THEN RETURN' return-Taste und Leer-
taste
FOR ho=0 TO 7 ' Zeichenhoehe
  FOR br=0 TO 6 ' Zeichenbreite
  IF POINT(br+x1,ho)<>0 THEN
  LINE (v%*br+x2,v%*ho+y2)-(v%*br+x2+v%-1,ho*v%+v%-
  ** 1+y2),1,bf
  END IF
  NEXT br
NEXT ho
RETURN

kopf:
CLS
FOR i = 46 TO 122 'ASCII Zeichencode
  PRINT CHR$(i);
NEXT i
COLOR 3
PRINT :PRINT "D A S G R O S S E A B C ";
COLOR 1
PRINT " RETURN = ENDE"
RETURN

taste:
ta$=INKEY$
IF ta$=CHR$(13) THEN ko=0:RETURN 'return-Taste
```

```
IF ta$=CHR$(32) THEN RETURN 'Leertaste
IF ta$<CHR$(46) OR ta$>CHR$(122) THEN taste 'ausserhalb des
** Bereiches
RETURN

position:
x2=x2+(8*v%)
IF x2+(7*v%)>617 THEN 'max. Bildschirmbreite
   y2=y2+(9*v%)   'naechste Zeile
   x2=0
   IF y2 +(8*v%)>187 THEN 'max. Bildschirmhoehe
   ko=0
   END IF
END IF
RETURN
```

Nachdem Sie die Vergößerung gewählt haben, können Sie den gewünschten Text ganz normal über die Tastatur eingeben. Da jedes Zeichen im Tastaturpuffer gespeichert wird, brauchen Sie beim Schreiben nach der Eingabe eines Buchstabens nicht darauf zu warten, bis er vergrößert erscheint, sondern können sofort den nächsten Buchstaben eintippen. Am Ende jeder Zeile springt das Programm automatisch in die nächste Zeile. Wenn der Bildschirm ausgefüllt ist, oder wenn Sie RETURN drücken, wird das Programm beendet.

Die Arbeitsweise des Programms ist prinzipiell sehr einfach. Zunächst wird der Buchstabe normal groß in ein bestimmtes Rechteck auf den Bildschirm gezeichnet. Dieses Rechteck wird dann mit POINT Punkt für Punkt abgetastet. Jeder gesetzte Punkt ergibt dann ein kleines (oder großes Rechteck) in der vergrößerten Darstellung.

Wenn Sie dieses Programm als Vorspann zu einem größeren Programm einsetzen wollen, sollten Sie den Programmteil, der sich mit der Tastatureingabe beschäftigt, entfernen. Sie können den Text, der dargestellt werden soll, dann besser in DATA-Anweisungen hinterlegen und aus diesen einlesen. Damit der Anwender nichts davon bemerkt, daß Sie den großen aus einem kleinen Buchstaben zusammensetzen, können Sie den kleinen Buchstaben auch unsichtbar auf den Bildschirm setzen. Dazu brauchen Sie nur mit der PALETTE-Anweisung vorübergehend eine Farbe der Hintergrundfarbe anzugleichen und den Buchstaben in dieser Farbe auf den Bildschirm zu schreiben:

```
PALETTE 0,0,0,.6 : PALETTE 2,0,0,.6 : COLOR 2
```

5.3 Sonderzeichen

In der Tabelle über die ASCII-Zeichen-Codes im Basic-Handbuch sind einige interessante Codes nicht aufgeführt. Diese möchte ich Ihnen aber nicht vorenthalten und werde sie deshalb in diesem Abschnitt vorstellen.

5.3.1 Zeichen-Codes bei der Tastaturabfrage

In der folgenden Übersicht sind die einzelnen Zeichen-Codes, der besseren Übersicht halber, nur mit ihren Ziffern aufgeführt. Abgefragt werden sie mit *v=CHR$(n)*. Die Zeichen-Codes können Sie in Ihren Programmen wie folgt abfragen:

```
taste:
ta$=INKEY$
IF ta$ = "" THEN taste
IF ta$ = CHR$(n) THEN Verzweigung
GOTO taste
```

Dabei ist *n* eine Ziffer aus einer der folgenden Tabellen, und mit *Verzweigung* ist eine Subroutine gemeint. Zum besseren Verständnis finden Sie auch ein Programmbeispiel am Ende dieses Kapitels.

Code	Bedeutung	Code	Bedeutung
8	BACK SPACE		
9	TAB		
13	RETURN		
27	ESC		
28	Cursor oben		
29	Cursor unten		
30	Cursor rechts		
31	Cursor links		
32	Leertaste		
127	DEL	31	CTRL+DEL
129	F1	1	CTRL+F1
130	F2	2	CTRL+F2
131	F3		
132	F4	4	CTRL+F4
133	F5	5	CTRL+F5
134	F6	6	CTRL+F6
135	F7	7	CTRL+F7
136	F8	8	CTRL+F8
137	F9	9	CTRL+F9
138	F10	10	CTRL+F10
139	HELP	11	CTRL+HELP

Tabelle 5.2: *ASCII-Codes*

5.3.2 Zeichen-Codes für die PRINT-Anweisungen

Ähnlich, wie einem Programm der Druck auf spezielle Tasten in Form besonderer ASCII-Codes mitgeteilt wird, kann es auch Zeichen ausgeben, die nicht als Buchstaben auf dem Bildschirm erscheinen, sondern spezielle Effekte ergeben. Die Anweisung PRINT CHR$(7) gibt einen Ton aus und läßt den Bildschirm aufblitzen. Diese PRINT-Anweisung entspricht also dem Aufruf des BEEP-Befehls.

Code	Wirkung
7	wie BEEP-Anweisung
8	wie Taste BACKSPACE
9	wie Taste TAB
10	wie PRINT-Anweisung allein (Zeilenvorschub)
12	wie CLS-Anweisung

Tabelle 5.3: *Spezielle ASCII-Codes für die Ausgabe mit PRINT*

Ein weiteres Problem, vor dem der Basic-Programmierer – nicht nur beim Amiga – immer wieder steht, sind die deutschen »Sonderzeichen«. So nennen jedenfalls amerikanische Softwarehäuser die Buchstaben, zu denen wir schlicht und einfach »Umlaut« sagen. Die folgende Tabelle listet auf, welche ASCII-Codes die Umlaute haben.

Code	Buchstabe	Code	Buchstabe
228	ä	196	Ä
246	ö	220	Ö
252	ü	220	Ü
223	ß		

Tabelle 5.4: *ASCII-Codes der deutschen Sonderzeichen*

Das folgende Programm demonstriert, wie deutsche »Sonderzeichen« auch in einem Basic-Programm verwendet werden können. (Amiga-Basic reagiert auf diese Zeichen ansonsten sehr unangenehm – verwenden Sie auf keinen Fall in Variablen-Namen oder anderen Texten in einem Basic-Programm Umlaute.)

```
REM Staedte raten
REM P.5.3-1
DIM J$(30)
richtig=0 : falsch=0
FOR n= 1 TO 30
  READ J$(n)
NEXT n
FOR z = 1 TO 24 STEP 6
```

```
   PRINT CHR$(12) 'statt CLS
   PRINT ,"Raten Sie die Hauptstadt von ";J$(z)
   PRINT CHR$(10);:PRINT CHR$(10);:PRINT CHR$(10); ' statt
PRINT
   PRINT ,"F1 ";J$(z+1)
   PRINT ,"F2 ";J$(z+2)
   PRINT ,"F3 ";J$(z+3)
   PRINT ,"F4 ";J$(z+4)
   PRINT CHR$(10);:PRINT CHR$(10);:PRINT CHR$(10);
   COLOR 3
   PRINT ,"bitte druecken Sie die richtige Funktionstaste"
   COLOR 1
   GOSUB taste
NEXT z
PRINT CHR$(12) 'statt CLS
PRINT CHR$(10);:PRINT CHR$(10);:PRINT CHR$(10); ' statt
** PRINT
PRINT ,"E R G E B N I S"
PRINT CHR$(10);:PRINT CHR$(10);:PRINT CHR$(10); ' statt
** PRINT
PRINT ,"Sie hatten ";richtig;" richtige Antworten"
PRINT CHR$(10);:PRINT CHR$(10);:PRINT CHR$(10); ' statt
** PRINT
PRINT ,falsch;" waren leider falsch"
END

taste:
tt=VAL(J$(z+5))
ta$=INKEY$
IF ta$="" THEN taste
IF ta$<CHR$(129) OR ta$>CHR$(132) THEN PRINT CHR$(7)
GOTO taste
IF ta$<>CHR$(tt) THEN falsch=falsch+1:RETURN
richtig=richtig+1
RETURN

DATA Deutschland,Berlin,Bonn,Hamburg,Muenchen,130
DATA Frankreich,Lyon,Biariz,Versailles,Paris,132
DATA Oesterreich,Wien,Innsbruck,Graz,Prater,129
DATA Italien,Verona,Mailand,Rom,Neapel,131
DATA Russland,Baku,Moskau,Leningrad,Kreml,130
```

Ein Rate-Programm dieser Art kann wesentlich eleganter aussehen. Als Beispiel für den Einsatz diverser Sonderzeichen erfüllt es aber seinen Zweck. Wenn Sie das Programm trotzdem als Ratespiel einsetzen wollen, brauchen Sie eigentlich nur die DATA-Anweisungen erheblich zu erweitern. Der Aufbau dieser Zeilen ist wie folgt: Das erste Wort der DATA-Zeile gibt das Land aus, und die vier folgenden Worte sind Städte, die in diesem Land liegen. Die Zahl an letzter Stelle der DATA Zeile ist der ASCII-Code für die Funktionstaste, die der Anwender drücken soll. Außerdem müssen Sie bei einer

Erweiterung des Programms die Dimensionierung des Feldes *J$* und Angaben in den beiden FOR-Schleifen entsprechend erhöhen.

5.4 Zufall

Für einige Anwendungen – insbesondere bei der Programmierung von Spielen – ist der Zufall unerläßlich. Amiga-Basic bietet zu diesem Zweck die RND-Funktion (engl. Random = Zufall).

v=RND
liefert eine zufällige Zahl zwischen 0 und 1

Probieren Sie die Funktion am besten aus, um sich einen Eindruck von der Zufälligkeit der Zahlenfolge, die wiederholte Aufrufe ergeben, zu verschaffen.

```
FOR i= 1 TO 10:PRINT RND :NEXT
```

Wenn Sie diese Zeile wiederholt eingeben, werden Sie feststellen, daß Sie bei jedem Aufruf die gleichen Zahlen erhalten. Manchmal ist das in einem Programm nicht weiter störend. Wenn es wichtig ist, bei jedem Programmdurchlauf verschiedene Zufallszahlen zu erhalten, müssen Sie eine weitere Funktion verwenden.

RANDOMIZE n
RANDOMIZE TIMER
startet den Zufallsgenerator mit einem neuen Anfangswert

Dabei ist die Angabe einer Zahl oder TIMER zwingend vorgeschrieben. Fehlt sie, stoppt Basic das laufende Programm und erfragt die Zahl direkt vom Benutzer.

```
Random number seed (-32768 to 32767)?
```

Die zuverlässigste Methode, wirklich zufällige Zahlen zu erzeugen, ist ansonsten *RANDOMIZE TIMER*. Geben Sie statt TIMER eine Zahl an, so liefert dies in der Folge für jede Zahl dieselbe Folge von Zufallszahlen. *RANDOMIZE 2* ergibt also eine andere Folge von Zahlen als *RANDOMIZE* 3, aber exakt dieselbe wie ein zweites *RANDOMIZE 2*.

In vielen Anwendungen werden Zufallszahlen innerhalb eines bestimmten Bereiches benötigt. RND liefert aber nur Zufallszahlen zwischen 0 und 1. Dies läßt sich aber mit einer einfachen Formel ändern. Angenommen, Sie wollen Zufallszahlen zwischen zwei Werten *Min* und *Max* erzeugen. Dann liefert die folgende Formel die gewünschten Zahlen

```
RANDOMIZE TIMER: z = Min + RND * (Max-Min).
```

Wenn Sie nur ganze Zahlen wünschen, können Sie das jederzeit mit der INT-Funktion erreichen. Probieren Sie Ihr neues Wissen gleich einmal an einem Beispiel aus. Das folgende Programm gibt Zufallszahlen in einem Bereich von 1 bis 49 aus, also Lottozahlen. Wenn Sie damit einen Hauptgewinn landen sollten, begnüge ich mich mit 5%.

```
REM Lotto
'P.5.4-1
PALETTE 3,0,0,0: PALETTE 2,.7,.7,.7
COLOR 2: DEFINT a-z
FOR i=3 TO 1 STEP -1:CIRCLE (122,52),19+i,i,,,.55:NEXT
PAINT (122,52),1
DIM kug(273):GET (100,30)-(144,72),kug

start:
CLS
LOCATE 7,16:PRINT "L O T T O"
FOR i=50 TO 250 STEP 50:PUT (50+i,30),kug,OR:NEXT
LOCATE 10,1:PRINT "Wenn Sie moegliche Lottozahlen haben
** wollen, druecken Sie bitte eine Taste!"
GOSUB Taste
FOR i = 1 TO 6
  GOSUB zufall:za(i)=z
  LOCATE 16,8+i*7: PRINT za(i)
  PUT (i*55+50,100),kug,OR
NEXT
LOCATE 21,10:PRINT "weiterspielen: beliebige Taste druecken"
LOCATE 22,10:PRINT "beenden: bitte F1 druecken"
GOSUB Taste
IF ta$=CHR$(129)THEN END
GOTO start

Taste:
ta$=INKEY$: IF ta$="" THEN Taste:RETURN

zufall:
za(0)=0: RANDOMIZE TIMER:z= INT(RND*(50))
FOR j =0 TO 6
IF z=za(j) THEN zufall
NEXT
RETURN
```

Im zweiten Beispiel zum Thema Zufallszahlen können Sie in einem Würfelspiel gegen Ihren Freund Amiga antreten. Da auch in diesem Programm die Zufallszahlen mit RANDOMIZE TIMER ermittelt werden, besteht absolute Chancengleichheit; der Amiga ist ein fairer Spielgegner.

```
REM wuerfel
REM P.5.4-2
PALETTE 0,.5,.8,.2: PALETTE 1,.2,.4,1: PALETTE 2,1,1,0
```

```
z=0
FOR b=10 TO 490 STEP 80
  h=5:GOSUB wuerfel:z=z+1
NEXT b
DEFINT a-z:d=251
DIM nul(d),ein(d),zwe(d),dre(d),vie(d),fuen(d),sech(d)
GET (10,5)-(70,35),nul:GET (90,5)-(150,35),ein
GET (170,5)-(230,35),zwe:GET (250,5)-(310,35),dre
GET (330,5)-(390,35),vie:GET (410,5)-(470,35),fuen
GET (490,5)-(550,35),sech
start:
CLS:LOCATE 7,25:PRINT "W U E R F E L S P I E L"
LOCATE 8,25:    PRINT "========================="
FOR i= 150 TO 350 STEP 100
  PUT (i,70),nul
NEXT
LOCATE 11,10:PRINT "Amiga:"
LOCATE 14,20:PRINT "bitte wuerfeln Sie"
LOCATE 16,10:PRINT "Wenn Sie bereit sind, druecken Sie
** bitte eine Taste!"
GOSUB Taste
h2=60:GOSUB ausgabe :spielzahl=zahl
h2=0:GOSUB ausgabe:compzahl=zahl
text$="u n e n t s c h i e d e n"
IF compzahl>spielzahl THEN text$="Der Amiga hat gewonnen"
IF spielzahl>compzahl THEN text$="Sie haben gewonnen"
COLOR 3:LOCATE 3,1:PRINT "E R G E B N I S : ";text$:
** COLOR 1
LOCATE 21,10:PRINT "weiterspielen: beliebige Taste Druecken"
LOCATE 22,10:PRINT "beenden: bitte F1 druecken"
GOSUB Taste:IF ta$=CHR$(129)THEN END
GOTO start
ausgabe:
zahl=0
FOR i=150 TO 350 STEP 100
  GOSUB zufall:zahl=zahl+z
  IF z/2-INT(z/2)=0 THEN
  IF z=2 THEN PUT (i,70+h2),zwe,PSET
  IF z=4 THEN PUT (i,70+h2),vie,PSET
  IF z=6 THEN PUT (i,70+h2),sech,PSET
  ELSE
  IF z=1 THEN PUT (i,70+h2),ein,PSET
  IF z=3 THEN PUT (i,70+h2),dre,PSET
  IF z=5 THEN PUT (i,70+h2),fuen,PSET
```

```
  END IF
NEXT i
RETURN

Taste:ta$=INKEY$
IF ta$="" THEN Taste
RETURN

wuerfel:
R=8
LINE (b,h)-(b+60,h+30),3,bf
IF z/2-INT(z/2)=0 THEN
  IF z=2 THEN dif=0:GOSUB zwei
  IF z=4 THEN GOSUB vier
  IF z=6 THEN dif=0: GOSUB zwei :GOSUB vier
ELSE
  IF z=1 THEN GOSUB eins
  IF z=3 THEN GOSUB drei
  IF z=5 THEN GOSUB fuenf
END IF
RETURN

zwei:GOSUB zwei1:GOSUB zwei2
RETURN

zwei1:CIRCLE (b+15,h+dif+15),R,1: PAINT (b+15,h+dif+15),1
RETURN

zwei2:CIRCLE (b+45,h+dif+15),R,1: PAINT (b+45,h+dif+15),1
RETURN

vier:dif=-10:GOSUB zwei:dif=10:GOSUB zwei
RETURN

eins:CIRCLE (b+30,h+15),R,1: PAINT (b+30,h+15),1
RETURN

drei:GOSUB eins:dif=10:GOSUB zwei1:dif=-10:GOSUB zwei2
RETURN

fuenf:GOSUB eins:GOSUB vier:RETURN

zufall:
RANDOMIZE TIMER:za=6:z= INT(RND*(za+1)):IF z=0 THEN zufall
RETURN
```

5.5 Kopfstand

Daß die GET-Anweisung eine der vielseitigsten Grafik-Anweisungen ist, haben Sie gewiß schon ziemlich am Anfang von Kapitel 2 festgestellt. Aber, haben Sie sich schon

einmal Gedanken darüber gemacht, wie diese Anweisung die vom Bildschirm gelesenen Daten ablegt? Mit dem numerischen Feld, das zur Speicherung der Bilddaten verwendet wird, läßt sich nämlich einiges anfangen – wenn man das exakte Format kennt. Bevor Sie sich näher damit befassen, geben Sie zunächst einmal das folgende Programm ein und starten Sie es.

```
REM Get-Manipulation
'P.5.5-1
DEFINT a-z
DIM bild(128)
LINE (17,0)-(31,62),1,bf
GET (17,0)-(31,62),bild
FOR i = 4 TO 64 STEP 2
  bild(i)=0
NEXT i
PUT (100,0),bild
```

Das Programm zeichnet zuerst ein kleines Rechteck, welches dann mit GET in das numerische Feld *bild* geholt wird. Wenn das Rechteck mit PUT wieder auf dem Bildschirm erscheint, ist plötzlich eine Leiter daraus geworden. Die Ursache dafür liegt in der FOR-Schleife, die einige Feld-Elemente in Zweierschritten auf Null setzt. Das deutet schon die Möglichkeiten an, die sich bei der Manipulation solcher Felder bieten.

Die Bild-Informationen wurden in diesem Beispiel in ein Feld mit kurzen Ganzzahlen gelegt. Wie Sie ja inzwischen wissen, ist eine kurze Ganzzahl 2 Byte (16 Bit) lang. Jedem Bit darin entspricht ein einzelner Bildpunkt (Pixel). Das bedeutet, daß man mit einem Feld-Element einen Bildausschnitt von 16 Punkten Breite und einem Pixel Höhe speichern kann. Schauen wir uns nun an, wie die ersten Feld-Elemente in unserem Beispiel aussehen. (Sie können das jederzeit überprüfen, indem Sie einfach die Felder mit PRINT ausgeben.)

0. Element	16 Bit:	0000000000001111=15	Breite
1. Element	16 Bit:	0000000000111111=63	Höhe
2. Element	16 Bit:	0000000000000010=2	Tiefe der Bitmap
3. Element	16 Bit:	1111111111111110=-2	15 Pixel gesetzt
4. Element	16 Bit:	0000000000000000=0	alle Pixel geloescht
5. Element	16 Bit:	1111111111111110=2	15 Pixel gesetzt
6. Element	16 Bit:	0000000000000000=0	alle Pixel geloescht
7. Element		...	...

Die ersten drei Feld-Elemente enthalten die Information über Breite, Höhe und Tiefe des Bildes. Die nächsten Elemente enthalten dann die eigentlichen Bild-Daten, also Bits, die angeben, ob ein Punkt gesetzt ist oder nicht. In dem obigen Beispiel wurden diese Elemente einfach auf Null gesetzt, und die entsprechenden Bildpunkte verschwanden.

Dieses sehr einfache Format stimmt aber nur dann, wenn das Programm lediglich mit zwei Farben arbeiten würde. Das ist aber meistens nicht der Fall. Der Standardbildschirm der Workbench benutzt schon vier Farben. Dadurch wird das Ganze etwas komplizierter. Wie Sie oben gelesen haben, enthält das dritte Feld-Element den Wert für die Tiefe 2. Das besagt nichts anderes, als daß für jeden Bildschirmpunkt zwei Bit übereinanderliegen, die vier Farben entsprechen.

Erstes Bit	Zweites Bit	Farbe:
0	0	Blau =Farbe 0
1	0	Weiß =Farbe 1
0	1	Schwarz =Farbe 2
1	1	Rot =Farbe 3

Tabelle 5.5: *Zusammenhang zwischen Bits und Farbwerten*

Die genannten Farben sind in diesem Beispiel wie üblich die Standardfarben. Dies gilt natürlich genauso für eine andere mit PALETTE geänderte Farbe. Wichtig für die Zuordnung von Bits zu einer bestimmten Farbe ist nur die Farbnummer. Da für jeden Bildpunkt zwei Bit benötigt werden, bleibt die Frage, wo diese beiden Bits in dem von GET erzeugten Feld stehen – sie stehen nämlich nicht hintereinander. Das Format ist glücklicherweise recht einfach. Zunächst kommen alle Daten der ersten Bit-Ebene, dann alle der zweiten Bit-Ebene (und die der dritten, wenn acht Farben verwendet werden, und so weiter). Sie können leicht nachprüfen, ob diese Information richtig ist. Ändern Sie im Programm »GET-Manipulation« die Farbe in der LINE-Anweisung von 1 auf 3. Die Leiter erscheint dann Rot und Schwarz, da Rot bedeutet, daß die Bits in beiden Bitmaps gesetzt sind (Farbzahl 3), während Schwarz bedeutet, daß nur noch das Bit in einer Bit-Ebene gesetzt ist. Probieren Sie es aus! Vielleicht sind Sie noch nicht überzeugt, da sich die schwarze Farbe auf dem dunkelblauen Untergrund schlecht hervorhebt. Fügen Sie bitte hinter der Zeile mit DIM... folgende Zeile ein:

```
PALETTE 0,1,1,0.
```

Damit haben wir die Hintergrundfarbe von Blau in Gelb geändert und die schwarz-rote Leiter ist gut zu sehen.

Überlegen Sie sich doch einmal, was Sie mit dem neuerworbenen Wissen nun anfangen. Sie können zum Beispiel das Bild auf den Kopf stellen! Dazu überlegen Sie erst einmal, wie die einzelnen Felder umgeschichtet werden müssen. Dazu brauchen Sie nur die letzte Zeile des Bildes im Feld mit der ersten zu vertauschen, die zweitletzte mit der zweiten und so weiter. Diesen Tauschvorgang kann man für jede der beiden (oder mehr) Bitmaps getrennt ausführen. Das einzige kleine Problem dabei ist, zu berechnen, im wievielten Feld die jeweilige Zeile der jeweiligen Bit-Ebene beginnt. Das folgende Programm löst dieses Problem. Es kopiert die obere Hälfte des Bildschirms in die untere Hälfte und spiegelt das Bild dabei auch gleich.

```
REM Fatamorgana
REM 'P.5.5-2
CLEAR ,35000&
DEFINT a-z
DEF FNw1=(hoehe*feldx)-(reihe*feldx)+feld+2
DEF FNw2=(reihe*feldx)+feld+2
breite=WINDOW(2)
hoehe=WINDOW(3)
hoehe=hoehe/2
feldy=hoehe+1
feldx=INT((breite+16)/16)
halbfeld=(feldy*feldx)
gesfeld=3+(halbfeld*2)
DIM bild(gesfeld)
DIM neubild(gesfeld)
```

zeichnung:

```
PALETTE 0,1,1,.2
PALETTE 1,.2,.8,1
PALETTE 2,.8,.6,.5
PALETTE 3,.5,.8,.2
AREA (200,10)
AREA (250,30)
AREA (450,40)
AREA (300,80)
AREA (150,85)
AREA (200,10)
AREAFILL
LINE (350,40)-(360,70),2,bf
FOR i= 1 TO 18
CIRCLE (345+i,50-i),30,3
NEXT i
```

kippen:

```
GET (0,0)-(breite,hoehe),bild
FOR reihe = 0 TO hoehe
  FOR feld = 0 TO feldx
  neubild(FNw1)=bild(FNw2)
  neubild(halbfeld+FNw1)=bild(halbfeld+FNw2)
  NEXT feld
NEXT reihe
FOR i= 0 TO 2
  neubild(i)=bild(i)
NEXT i
PUT (0,90),neubild
ERASE bild
ERASE neubild
END
```

Im ersten Abschnitt des Programms werden die Variablen mit Werten versehen. Alle Variablen tragen lange, aussagekräftige Namen, was völlig ausreichen dürfte, Ihnen zu sagen, was in den Variablen enthalten ist, beziehungsweise woher die Werte stammen. In der Routine *zeichnung* entsteht das Bild. Sie können hier natürlich auch Ihre eigenen Zeichnungen oder Texte einsetzen. Die Routine *kippen* vertauscht in zwei ineinandergeschachtelten FOR-Schleifen die beiden Bit-Ebenen im zuvor mit GET gelesenen Bild, wie eben beschrieben. Wenn Sie den ganzen Bildschirm kippen wollen, brauchen Sie nur die folgenden drei Zeilen zu ändern.

```
hoehe=hoehe/2          loeschen
CLEAR ,35000           wird CLEAR,60000
PUT (0,90),neubild     wird PUT (0,0),neubild
```

5.6 BOBs und Sprites einmal anders

In den Kapiteln über die BOBs und die Sprites haben Sie erfahren, was man in AmigaBasic alles mit beweglichen Objekten anstellen kann. Und das ist schon eine ganze Menge. Trotzdem gibt es noch viel mehr solche Tricks und Kniffe. Ein paar davon werden Sie in diesem Kapitel kennenlernen.

Da Sie sicherlich inzwischen einige Sprites auf dem Sprite-Editor erstellt haben, wird Ihnen die Farbgebung in diesem Programm oft recht eintönig vorgekommen sein. Der »ObjEdit« verwendet feste Farbwerte, nämlich Weiß, Schwarz und Rot. Es ist aber kein Problem, den Objekt-Editor dahingehend zu ändern, daß man auch andere Farben verwenden kann. Das sollte mit Ihren Kenntnissen sicher kein Problem sein. Doch was machen Sie, wenn Sie in einem anderen Programm merken, daß die Sprite-Farben nicht richtig passen? Da müssen Sie die Datei mit den Sprite-Daten wieder in die »BASICDemos«-Schublade legen und den Objekt-Editor erneut aufrufen. Wenn Sie dann die Farben mühevoll geändert haben, stellen Sie vielleicht erneut fest, daß sie Ihnen immer noch nicht gefallen. Die Farbänderung ist also eine recht umständliche Angelegenheit. Da bietet es sich doch direkt an, die Arbeit von einem kleinen Programm machen zu lassen.

Dazu müssen Sie aber zunächst einmal wissen, wie der String mit den Sprite-Daten, den eine solche Datei enthält, aufgebaut ist. Das folgende Programm liest diese Daten zunächst einmal ein und gibt sie dann als eine Folge von Zahlen auf dem Drucker aus. (Wenn Sie diese Daten statt dessen auf dem Bildschirm sehen wollen, brauchen Sie nur die LPRINT-Anweisungen im Programm gegen PRINT-Anweisungen auszutauschen.) Sie können die ausgedruckten Zahlen später hervorragend dazu verwenden, das Sprite oder BOB in Form von DATA-Anweisungen im Programm zu verstecken und brauchen es nicht bei jedem Start von der Diskette einzulesen.

```
REM Objekt-Daten
'P.5.6-1
DEFINT a-z
```

```
start:
PRINT "O B J E K T D A T E N":PRINT
PRINT :INPUT "DATAs? auf Drucker? j/n: ";da$
INPUT "bitte Objektnamen: ";namen$
IF namen$="" OR da$="" THEN start
OPEN namen$ FOR INPUT AS 1
   n= LOF(1)/2:n=n-5:DIM sprite(n)
   FOR i=1 TO 5:sprite(i)=CVL(INPUT$(4,1)):NEXT
   FOR i=6 TO n:sprite(i)= CVI(INPUT$(2,1)):NEXT
CLOSE #1
IF da$="n" THEN weiter
LPRINT :LPRINT :LPRINT namen$:LPRINT "Anzahl der Daten="n
FOR i = 1 TO 4: LPRINT :NEXT i
FOR i = 1 TO n: LPRINT sprite(i);:NEXT
LPRINT

weiter:
CLS:WIDTH 60
FOR i = 1 TO n: PRINT sprite(i);:NEXT
OPEN namen$ FOR INPUT AS 1:OBJECT.SHAPE 1,INPUT$(LOF(1),1)
** :CLOSE #1
PALETTE 0,.5,.8,1
OBJECT.X 1,520: OBJECT.Y 1,100
OBJECT.ON
END
```

Hinter *start* fragt das Programm zunächst, ob die Daten ausgedruckt werden sollen. Mit einem zweiten INPUT-Befehl wird dann der Name des gewünschten Objektes an das Programm übergeben. Sie können hier Dateien angeben, die sowohl Sprites als auch BOBs enthalten. Nachdem die betreffende Datei geöffnet wurde, wird mit LOF zunächst ihre Länge ermittelt.

v=LOF(Dateinummer)
liefert die Länge der Datei mit der Kennung *Dateinummer* in Bytes

Aus der Länge kann das Programm die Anzahl der Datenwerte des Objektes ermitteln. Die dafür verwendete Formel werden Sie sofort verstehen, wenn Sie die folgenden Erläuterungen gelesen haben. Mit dem hier gefundenen Wert *n* hat das Programm dann auch gleich die richtige Anzahl von Elementen für die Dimensionierung des Feldes *sprite*.

Wenn Sie wissen wollen, wie sich die Zeichenkette eines Objektes zusammensetzt, schauen Sie am besten dort nach, wo diese Zeichenkette erzeugt wird, im Programm »ObjEdit«. (Dieses Programm ist recht gut kommentiert, so daß man daraus alle für diese Anwendung wichtigen Fakten entnehmen kann.) Die ersten fünf Werte eines solchen Strings bestehen jeweils aus 4Byte langen Zahlen (langen Ganzzahlen).

1. Wert: ColorSet=CVL(INPUT$(4,1)) ist immer Null

2. Wert: DataSet =CVL(INPUT$(4,1)) ist immer Null

3. Wert: Tiefe =CVL(INPUT$(4,1)) ist immer zwei (Farbtiefe)

4. Wert: Breite =CVL(INPUT$(4,1))-1 ist immer 16 (Spritebreite)

5. Wert: Höhe =CVL(INPUT$(4,1))-1 ist variabel (Spritehöhe)

Es folgen drei Werte, die jeweils aus 2 Byte bestehen, also eine kurze Ganzzahl sind:

6. Wert: Flags =CVI(INPUT$(2,1)) ist immer 25

7. Wert: PlanePick =CVI(INPUT$(2,1)) ist immer drei

8. Wert: PlaneOnOff=CVI(INPUT$(2,1)) ist immer null

(Wenn Sie in den obigen Erläuterungen lesen *ist immer...*, so gilt das natürlich nur für die hier besprochenen Fälle.) Ab dem 9. Wert folgen dann wieder 2 Byte große Werte, bis die Datei zu Ende ist. Dieser 9. Wert ist der erste wirklich wichtige. Hier beginnen die Daten, die mit Hilfe einer GET-Anweisung bei der Sprite-Programmierung eingelesen werden. Die Daten in diesem Bereich können genauso manipuliert werden, wie es im Kapitel »Kopfstand« vorexerziert wurde. Für dieses Programm sind im Augenblick aber die letzten drei Werte noch interessanter. Sie entsprechen nicht mehr dem Feldinhalt bei der GET-Anweisung und enthalten die drei Farben des Sprites.

Jede der drei Farben wird als 16-Bit-Zahl abgespeichert, von der aber nur die rechten 12 Bit genutzt werden. Jede Farbe setzt sich auf dem Amiga ja aus drei Teilen, dem Rot-, Grün- und Blau-Anteil zusammen, und für jeden dieser Anteile steht jeweils ein Nibble (4 Bit) der 16-Bit-Zahl zur Verfügung (zusammen also 12 Bit).

	Rotes Nibble	Grünes Nibble	Blaues Nibble
Wert:	2048 1024 512 256	128 64 32 16	8 4 2 1
Bitnummer:	11 10 9 8	7 6 5 4	3 2 1 0

Tabelle 5.6: *Die drei Farbanteile einer Farbe auf drei Nibbles verteilt*

Jedes Nibble kann 16 verschiedene Werte (zwischen 0 und 15 beziehungsweise 0 und F hexadezimal) besitzen, die den 16 möglichen Intensitäten eines Farbanteils entsprechen. Dividiert man diesen Wert durch 16, erhält man die Intensitätswerte, wie sie in der PALETTE-Anweisung verwendet werden. Damit haben wir jetzt alle Grundlagen beisammen, um das folgende Programm zu schreiben, das die Farbangaben in einer Sprite-Datei ändern kann.

```
REM Sprite-Farben
'P.5.6-2
DEFINT a-z
```

```
start:
CLS:PRINT "S P R I T E M A N I P U L A T I O N -
**  Farbaenderung":PRINT
PRINT :INPUT "Ausdruck der DATAs? j/n: ";da$
PRINT :INPUT "bitte alten Objektnamen: ";namen$
PRINT :INPUT "bitte neuen Objektnamen: ";namen1$
IF namen$="" OR namen1$="" OR da$="" THEN start
PRINT :PRINT "Bitte Farbe wie bei PALETTE eingeben!
** Beispiel .8,.6,.53"
OPEN namen$ FOR INPUT AS 1
   n= LOF(1)/2:n=n-5:DIM sprite(n)
   FOR i=1 TO 5:sprite(i)=CVL(INPUT$(4,1)):NEXT
   FOR i=6 TO n:sprite(i)= CVI(INPUT$(2,1)):NEXT
CLOSE #1
FOR i=0 TO 2
   LOCATE 12+i,1:PRINT "Farbe ";i+1;": "
   LOCATE 12+i,10:INPUT a!,b!,c!
   sprite(n-2+i)=(15*c!)+(15*b!*(2^4))+(15*a!*(2^8))
NEXT i
IF da$="n" THEN weiter
LPRINT :LPRINT :LPRINT namen1$:LPRINT "Anzahl der Daten="n
FOR i = 1 TO 4: LPRINT :NEXT i
FOR i = 1 TO n: LPRINT sprite(i);:NEXT
LPRINT

weiter:
OPEN namen1$ FOR OUTPUT AS 1
   FOR i = 1 TO 5
   PRINT #1,MKL$(sprite(i));
   NEXT i
   FOR i = 6 TO n
   PRINT #1, MKI$(sprite(i));
   NEXT i
CLOSE #1
OPEN namen$ FOR INPUT AS 1: OBJECT.SHAPE 1,INPUT$(LOF(1),1)
** :CLOSE #1
OPEN namen1$ FOR INPUT AS 1: OBJECT.SHAPE 2,INPUT$(LOF(1),1)
** :CLOSE #1
PALETTE 0,.5,.8,1
OBJECT.X 1,80: OBJECT.Y 1,100
OBJECT.X 2,80: OBJECT.Y 2,140
OBJECT.ON
LOCATE 14,1:PRINT "vorher: "
LOCATE 18,1:PRINT "nachher: "
END
```

Das Programm fragt zuerst, ob die Sprite-Werte ausgedruckt werden sollen. Anschließend wird mit INPUT der alte und der neue Name des Sprites eingelesen. (Unter dem neuen Namen wird das geänderte Sprite auf die Diskette zurückgeschrieben.) Wird

der alte Name auch als neuer Name eingegeben, werden die neuen Werte unter dem ursprünglichen Namen auf Diskette gespeichert und die alte Version dabei zerstört. Dann wird das zu ändernde Sprite in das Feld *sprite* eingelesen. Nun muß der Benutzer die drei Sprite-Farben als Folge von jeweils drei RGB-Werten, genauso wie bei der PALETTE-Anweisung, eingegeben. Die einzelnen Farbbestandteile werden dann, wie eben beschrieben, umgerechnet und an die Feld-Variable *sprite* übergeben.

Falls ein Ausdruck der Zahlen für die Verwendung in DATA-Anweisungen gewünscht wurde, erfolgt dieser Ausdruck jetzt. Danach werden die geänderten Sprite-Daten auf Diskette gespeichert. Und damit der Anwender auch sieht, welchen Erfolg die Änderung gebracht hat, wird sowohl das alte wie auch das neue Sprite auf den Bildschirm gebracht und entsprechend beschriftet. Am besten probieren Sie das Programm gleich einmal mit Ihren Lieblings-Sprites aus.

Die Änderung der Objekt-Farben ist nicht nur bei Sprites möglich. Die Farben der BOBs werden ja der aktuellen Palette entnommen und können zusammen mit dieser geändert werden. Amiga-Basic stellt aber noch eine besondere Anweisung parat, mit der die Farben eines BOBs auch nachträglich (aber nur mit den Möglichkeiten, die die aktuelle PALETTE bietet) geändert werden können.

OBJECT.PLANES Objekt, Bit-Ebene, Ebene-Ein-Aus

legt fest, auf welche Bit-Ebenen des Bildes das BOB mit der Kennung *Objekt* »wirkt«, wenn es auf dem Bildschirm erscheint

Ein BOB ist ja selbst ein kleines Bild, das beim Erscheinen auf dem Bildschirm vorübergehend die entsprechenden Bildteile ersetzt. Wenn das BOB aber aus weniger Bit-Ebenen besteht als der Bildschirm, hat man die Wahl, auf welchen Bit-Ebenen des Bildschirms die Bit-Ebenen des BOBs abgebildet werden sollen. Ein vierfarbiges BOB in einem 32farbigen Bildschirm kann deshalb vier beliebige Farben aus den 32 zur Verfügung stehenden benutzen und nicht nur die ersten vier. (Haben der Bildschirm und das BOB gleichviele Bit-Ebenen, ergibt OBJECT.PLANES deshalb wenig Sinn.)

Wie die beiden Zahlen *Bit-Ebene* und *Ebene-Ein-Aus* sich gegenseitig beeinflussen, kann ohne weitere Informationen nur sehr schwierig erklärt werden. Für die praktische Anwendung empfiehlt es sich deshalb, die jeweiligen Farbänderungen an einem praktischen Beispiel zu testen. Konstruieren Sie dafür bitte ein Farb-BOB im 8-Farben-Modus. Das BOB sollte aus 8 übereinanderliegenden farbigen Quadraten in der Reihenfolge der Farbnummern bestehen. Nennen Sie das BOB »famu«. Das folgende Programm wird Ihnen helfen, die Veränderung der Farbgebung durch OBJECT.PLANES zu verstehen.

```
REM BOB-Farben
REM P.5.6-3
DEFINT a-z
SCREEN 1,640,200,3,2
```

```
WINDOW 2,"bitte Taste druecken",(0,0)-(600,180),0,1
OPEN"famu"FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1):CLOSE #1
OBJECT.SHAPE 2,1
OBJECT.ON
FOR j = 0 TO 7
FOR i = 0 TO 7
OBJECT.PLANES 2,i,j
OBJECT.X 1,200:OBJECT.Y 1,80
OBJECT.X 2,250:OBJECT.Y 2,80
LOCATE 3,1:PRINT "Bitebene: "i"Ebene-an-aus: "j
start:
ta$=INKEY$:IF ta$="" THEN start
NEXT i
NEXT j
SCREEN CLOSE 1
END
```

Das Programm zeichnet das BOB zunächst mit den unveränderten Farben und daneben dasselbe BOB mit der durch OBJECT.PLANES modifizierten Colorierung. Sobald Sie eine Taste drücken, erscheint dann eine andere Kombination von *Bit-Ebene* und *Ebene-Ein-Aus*, deren Werte auch immer auf dem Bildschirm ausgegeben werden.

Sicherlich haben Sie schon einmal bei professioneller Spiele-Software gesehen, wie sich Sprites oder BOBs (eventuell explosionsartig) auflösen. Mit Amiga-Basic gibt es sogar mehrere Möglichkeiten, ein solches Verhalten in die Tat umzusetzen! Eine davon besteht darin, fünf oder zehn BOBs beziehungsweise Sprites mit den einzelnen Stadien der Vernichtung zu programmieren und diese nacheinander auf dem Bildschirm zu zeigen. Das ist recht umständlich und erfordert einigen Programmieraufwand. Eine andere Möglichkeit ist, die jeweiligen Objekt-Daten nach und nach zu löschen. Dafür brauchen Sie keine zusätzlichen Objekte, nur ein kleines Programmstück.

```
REM aufloesen
'P.5.6-4
DEFINT a-z
start:
PRINT "S P R I T E - und B O B M A N I P U L A T I O N"
PRINT :PRINT "Sprite oder BOB-Aufloesung"
PRINT :INPUT "bitte Objektnamen: ";namen$
IF namen$="" THEN start
OPEN namen$ FOR INPUT AS 1
   n= LOF(1)/2:n=n-5:DIM sprite(n)
   FOR i=1 TO 5:sprite(i)=CVL(INPUT$(4,1)):NEXT
   FOR i=6 TO n:sprite(i)= CVI(INPUT$(2,1)):NEXT
CLOSE #1
GOSUB altsprite
n1=INT(n/10)*10:n1=n1-10
```

```
FOR i = 1 TO 5:te1$=te1$+MKL$(sprite(i)):NEXT i
fa=0:GOSUB redu
OBJECT.SHAPE 2,te$(fa)
FOR fa = 0 TO 9: GOSUB redu:NEXT fa
FOR fa=0 TO 9: GOSUB neusprite:NEXT
OBJECT.CLOSE 2:END

redu:
FOR i = 11+fa TO n1+fa STEP 10:sprite(i)=0: NEXT i
te$(fa)=te1$
FOR i = 6 TO n:te$(fa)=te$(fa)+MKI$(sprite(i)):NEXT i
RETURN

altsprite:
OPEN namen$ FOR INPUT AS 1: OBJECT.SHAPE 1,INPUT$(LOF(1),1):
** CLOSE #1
PALETTE 0,.5,.8,1
OBJECT.X 1,80: OBJECT.Y 1,100
OBJECT.ON
LOCATE 14,1:PRINT "vorher: "
LOCATE 18,1:PRINT "nachher: "
RETURN

neusprite:
OBJECT.CLOSE 2
OBJECT.SHAPE 2,te$(fa)
OBJECT.X 2,80: OBJECT.Y 2,140
OBJECT.ON 2
RETURN
```

Bei *start* wird zunächst wieder der Name eines BOBs oder Sprites erfragt und die Objekt-Daten werden von der Diskette gelesen. Damit man sehen kann, wie das Objekt unverstümmelt aussieht, wird es erst einmal mit *altsprite* auf den Bildschirm gebracht. Dann werden zehn neue Strings für die einzelnen Auflösungsstufen erzeugt. Dazu werden zunächst die ersten fünf als lange Ganzzahlen aus dem Ursprungs-Objekt im String *te1$* gespeichert. Dann werden, beginnend ab dem elften Wert, in Zehnerschritten die Objekt-Daten auf null gesetzt. Dabei werden die letzten Werte ausgenommen, damit nicht versehentlich die Farben (bei Sprites) gelöscht werden. Die so erzeugten Sprite-Daten werden dann einem Element der Feld-Variablen *te$* zugewiesen. Dieser Löschvorgang wird insgesamt zehnmal wiederholt, so daß das Feld danach 10 verschiedene Objekte verschiedener Auflösungsstufen enthält.

Der bisher besprochene Programmteil eignet sich gut dafür, in Ihren eigenen Programmen verwendet zu werden. Sie haben dann für eine entsprechende Aktion die zehn neuen Objekte zur Verfügung. Die restlichen Routinen dienen nur dazu, den hier erzielten Effekt auch zu demonstrieren – sie sollten nicht in andere Programme übernommen werden. In *neusprite* wird das alte Objekt mit OBJECT.CLOSE aus dem Speicher entfernt. An seiner Stelle erscheinen dann nacheinander (an derselben Position) die zehn Teilsprites. So löst sich das ursprüngliche Objekt nach und nach auf.

Ein anderer hübscher Effekt, den Sie sicher schon einmal in einem Programm gesehen haben, sind Objekte, die ständig ihre Farben wechseln. In dem folgenden Programm wird dieser Effekt durch »Invertieren« der Objekt-Daten erzielt. »Invertieren« bedeutet, daß sämtliche Bits umgedreht werden. Wo ein Bit eine Null enthält, wird es auf Eins gesetzt und umgekehrt. Man erhält dadurch eine Art Film-Negativ.

```
REM invertieren
REM P.5.6-5
DEFINT a-z
start:
PRINT "S P R I T E - und B O B M A N I P U L A T I O N"
PRINT :PRINT "Objekte invertieren"
PRINT :INPUT "bitte Objektnamen: ";namen$
IF namen$="" THEN start
OPEN namen$ FOR INPUT AS 1
  n= LOF(1)/2:n=n-5:DIM sprite(n):DIM bild(n)
  FOR i=1 TO 5:sprite(i)=CVL(INPUT$(4,1)):NEXT
  FOR i=6 TO n:sprite(i)= CVI(INPUT$(2,1)):NEXT
CLOSE #1
GET (0,0)-(15,(n-13)/2),bild
FOR i =9 TO n-3
  bild(i-6)=sprite(i)
NEXT
CLS
PUT (30,0),bild,PRESET
GET (30,0)-(45,(n-13)/2),bild
FOR i = 1 TO 5:te1$=te1$+MKL$(sprite(i)):NEXT i
FOR i = 6 TO 8:te1$=te1$+MKI$(sprite(i)):NEXT i
FOR i = 9 TO n-3:te1$=te1$+MKI$(bild(i-6)):NEXT i
FOR i = n-2 TO n :te1$=te1$+MKI$(sprite(i)):NEXT i
OPEN namen$ FOR INPUT AS 1
OBJECT.SHAPE 1,INPUT$(LOF(1),1)
CLOSE#1
OBJECT.SHAPE 2,te1$
PALETTE 0,.5,.8,1
OBJECT.X 1,80: OBJECT.Y 1,100
OBJECT.X 2,80: OBJECT.Y 2,100
WHILE 1
  IF wert THEN
       OBJECT.ON 1:OBJECT.OFF 2:wert=0
  ELSE
       OBJECT.ON 2:OBJECT.OFF 1:wert=1
  END IF
WEND
```

Der erste Teil dieses Programms entspricht weitgehend dem ersten Teil des vorhergehenden Programms. Die erste GET-Anweisung dient dazu, auf möglichst einfache Art die ersten drei Werte herauszufinden, die GET beim Abspeichern eines Bildes ablegt.

Anschließend werden die Objekt-Daten an die Feld-Variable *Bild* übergeben. Dieses Bild wird nun mit PUT im Modus PRESET (also invertiert) auf den Bildschirm gebracht und dann mit GET sofort gespeichert. Anschließend wird das invertierte Objekt in einem String gespeichert. Dieser beginnt zunächst mit einer Kopie des ersten Teils der originalen Objekt-Daten. Die eigentlichen Bild-Daten des Objekts werden aber durch invertierte Bits aus der Feld-Variablen *Bild* ersetzt. Damit Sie auch das Ergebnis dieses Vorgangs bewundern können, werden schließlich beide Sprites aktiviert und in einer WHILE-Schleife nacheinander ein- und ausgeschaltet.

Das waren einige originelle Möglichkeiten, um Sprites und BOBs einmal anders zu verwenden. Vielleicht haben Sie dabei auch ein paar Ideen entwickelt, die Sie ausprobieren wollen. Dann aber nichts wie ans Werk!

5.7 Automatische Dimensionierung von Feldern

Inzwischen haben Sie sicher schon eigene Programme geschrieben, bei denen Sie in die Verlegenheit kamen, die Feld-Variablen dimensionieren zu müssen. Dabei haben Sie dann auch zwei Fehlermeldungen des Interpreters kennengelernt. *Subscript out of range* erscheint dann, wenn Sie eine Variable zu niedrig dimensioniert haben. Und *Duplicate definition* erscheint, wenn Sie die gleiche Feld-Variable mehrmals dimensionieren wollen. Im folgenden Programm wird eine automatische Dimensionierung realisiert und an einem Beispiel gezeigt, das Sie vor den beiden Fehlermeldungen bewahrt.

```
REM Fluesse
REM P.5.7-1
OPTION BASE 1
DIM fl$(10)
FOR n= 1 TO 10:READ fl$(n):NEXT

start:
FOR i= 1 TO 10000:NEXT:CLS:zeig=0
INPUT "Nennen Sie mir einen Fluss, den ich noch nicht
*** kenne: ", flus$
flu$=UCASE$(flus$)
FOR n = LBOUND(fl$) TO UBOUND(fl$)
  IF flu$ =fl$(n) THEN zeig=-1:n=UBOUND(fl$)
NEXT n
IF zeig THEN PRINT "Den Fluss "flu$" kenne ich schon"
*** :GOTO start

dimensionierung:
PRINT "Danke, den Fluss "flu$" habe ich noch nicht
gekannt."
DIM fl1$(UBOUND(fl$)+1)
```

```
FOR n=LBOUND(fl$)TO UBOUND(fl$)
   fl1$(n)=fl$(n)
NEXT
fl1$(UBOUND(fl1$))=flu$
ERASE fl$
DIM fl$(UBOUND(fl1$))
FOR n=LBOUND(fl1$)TO UBOUND(fl1$)
   fl$(n)=fl1$(n)
NEXT n
ERASE fl1$
GOTO start
DATA WOLGA,DONAU,RHEIN,MOSEL,MAIN
DATA ELBE,NECKAR,NAAB,ISAR,REGEN
```

In der ersten Programmzeile wird zunächst der Index-Wert für das erste Element aller Feld-Variablen des Programmes festgelegt. Feldvariablen können mit dem Element Nummer 0 oder 1 beginnen.

OPTION BASE n
setzt den Index-Wert für das erste Element aller Feldvariablen des Programms auf *n* (*n* kann nur 0 oder 1 sein)

Dann wird zunächst einmal die Feld-Variable fl$ für 10 Elemente dimensioniert. Diese 10 Elemente werden dann aus DATA-Anweisungen eingelesen. Der Anwender muß nun den Namen eines Flusses eingeben, der der String-Variablen *flus$* zugewiesen wird. Der Name wird dann im Feld *fl$* gesucht, wobei die UCASE-Funktion dafür sorgt, daß die Großschreibung keine Rolle spielt. Wenn der Flußname gefunden wurde, geht es zurück an den *start*. Wenn nicht, wird die Routine *dimensionierung* aufgerufen. In einer FOR-Schleife werden dann die einzelnen Feldelemente der Variablen *fl$* in die Variable *fl1$* kopiert, die zuvor mit derselben Anzahl von Elementen dimensioniert wird, die *fl$* gerade enthält. Wieviele Elemente *fl$* enthält, stellt das Programm mit einer speziell dafür vorgesehenen Basic-Funktion fest.

LBOUND (Feld, *Dimension*)
liefert die Index-Nummer des ersten Elements von *Feld*. Besitzt das Feld mehrere Dimensionen, kann mit *Dimension* angegeben werden, welche davon überprüft werden soll

UBOUND (Feld, *Dimension*)
liefert die Index-Nummer des letzten Elements von *Feld.* Besitzt das Feld mehrere Dimensionen, kann mit *Dimension* angegeben werden, welche davon überprüft werden soll

Hierzu ein Beispiel: Geben Sie die folgenden Anweisungen im Direktmodus ein.

```
OPTION BASE 1:DIM fl$(20):PRINT LBOUND(fl$) UBOUND(fl$)
```

Sie erhalten dann die beiden Zahlen 1 (untere Grenze) und 20 (obere Grenze).

Nun aber zurück zum Programm: Nachdem das alte Feld in das neue Feld *fl1$* kopiert wurde, wird es mit ERASE gelöscht und dann erneut, aber mit einem Element mehr, dimensioniert. Dann werden die Elemente von *fl1$* zurück nach *fl$* kopiert und als letztes Element der neue Flußname angehängt. Schließlich wird noch *fl1$* gelöscht, und es geht zurück an den *start.*

5.8 Etwas Mathematik

Manchmal läßt es sich nicht vermeiden, in einem Programm komplizierte Berechnungen durchzuführen – und dies vielleicht auch noch wiederholt an mehreren Stellen des Programms. Es kann sehr lästig werden, immer wieder dieselbe lange, komplizierte Formel einzugeben. Da ist es sehr praktisch, wenn man Funktionen definieren und diese dann wie SIN und SQR verwenden kann.

DEF FNName (Arg1, Arg2, ...) = Funktions-Definition
definiert eine neue Funktion, die an anderer Stelle mit *Name(...)* aufgerufen werden kann

Die Namen in der Klammer hinter dem Funktionsnamen sind die »Argumente« der Funktion, die bei der Berechnung des Ergebnisses verwendet werden können. (Die Funktion SIN benötigt zum Beispiel als Argument einen Winkel, von dem der Sinus berechnet werden soll.)

Hierzu ein Beispiel: Angenommen, Sie müßten mehrfach in einem Programm die Fläche eines rechtwinkligen Dreiecks berechnen. Die folgende Anweisung definiert die Funktion *dreiecksflaeche,* die genau das tut.

```
DEF FNdreiecksflaeche(grundl,hoe)=grundl*hoe/2
```

Geben Sie nun im Direktmodus die folgenden Anweisungen ein:

```
a=20:b=7:c=FNdreiecksflaeche(a,b):PRINT a, b, c
```

und Sie erhalten

```
20      7     70
```

Das folgende Programm zeigt ein etwas größeres Beispiel für die Verwendung selbstdefinierter Funktionen im Amiga-Basic.

```
REM Kreisberechnung
REM P.5.8-1
pi=3.14159265#
DEF FNkrfl(r)=pi*r^2
DEF FNkrumf(r)=2*pi*r

start:
CLS:PRINT "KREISBERECHNUNG":PRINT
PRINT "E N D E = 0":PRINT
INPUT ;"Bitte Radius: ",r
IF r=0 THEN END
PRINT :PRINT "Flaeche: "FNkrfl(r)
PRINT "Umfang: "FNkrumf(r)
FOR i= 1 TO 10000:NEXT
GOTO start
```

In dem Programm werden für die Berechnung der Kreisfläche mit *DEF FNkrfl(r)* und des Kreisumfanges mit *DEF FNkrumf(r)* zwei Funktionen definiert und später im Programm verwendet, als wären diese Funktionen im Basic »fest eingebaut«. Und weil die Beschäftigung mit der Mathematik gerade so schön ist, schauen Sie sich doch einmal an, wie ein mathematisches Anwendungsprogramm aufgebaut sein könnte.

```
REM Mini-Mathe
REM
DEFDBL a-z
pi=3.14159265359#
DEF FNbog(r,w)=pi*r*w/180
MENU 1,0,1,"R E C H E N A R T "
MENU 1,1,1,"Quadratwurzel "
MENU 1,2,1,"Exponentialfunktion: "
MENU 1,3,1,"Logarithmus "
MENU 1,4,1,"Kreis "
MENU 1,5,1,"Kreisausschnitt "
MENU 2,0,1,"B E E N D E N "
MENU 2,1,1,"ende "
t1$="Bitte Zahl eingeben: "
t2$="Bitte Radius eingeben: "
t3$="Bitte Winkel eingeben: "
```

```
ON TIMER(1) GOSUB datum:TIMER ON
ON MENU GOSUB menuKontrolle:MENU ON
WHILE 1
  SLEEP
WEND

menuKontrolle:
menuTitel = MENU(0)
MenuNummer= MENU(1)
ON menuTitel GOTO rechnen,ende

rechnen:
ON MenuNummer GOSUB wurzel,expo,loga,kr,krausschn
RETURN

ende:
IF MenuNummer THEN END

wurzel:
tex$=t1$:GOSUB eingabe
c=ABS(a)
c=SQR(c)
PRINT "Wurzel aus "a" = "c
RETURN

expo:
tex$=t1$:GOSUB eingabe
PRINT " e hoch "a" = "EXP(a)
RETURN

loga:
tex$=t1$:GOSUB eingabe
PRINT "der natuerliche Logarithmus von "a" = "LOG(a)
RETURN

kr:
tex$=t2$:GOSUB eingabe
IF SGN(a) =-1 THEN PRINT "negative Kreise sind nicht
** berechenbar":RETURN
PRINT "Die Kreisfläche von Radius "a" betraegt: "pi*a^2
PRINT "Der Umfang des Kreises betraegt: "2*pi*a
kr1:CIRCLE (400,120),80,3,,,.52
PAINT (400,120),3
RETURN

krausschn:
tex$=t2$:GOSUB eingabe
tex$=t3$:GOSUB eingabe2
IF SGN(a) =-1 THEN PRINT "negative Kreise sind nicht
** berechenbar":RETURN
d=FNbog(a,b)
```

```
PRINT "Der Kreisbogen betraegt: "d
PRINT "Die Flaeche des Kreisausschnittes betraegt: "d*a/2
IF b>359 THEN kr1
bo=pi*b/180
CIRCLE (400,120),80,3,-.001,-bo,.52
PAINT (479,119),3
RETURN

eingabe:
CLS:LOCATE 3,1:PRINT tex$
INPUT a
IF a =0 THEN eingabe
RETURN

eingabe2:
LOCATE 6,1:PRINT tex$
INPUT b
IF b =0 THEN eingabe2
RETURN

datum:
spalte=POS(0)
zeile=CSRLIN
LOCATE 1,60:PRINT DATE$
LOCATE zeile,spalte
RETURN
```

Das Programm beginnt zunächst mit der Definition aller Zahlen als doppeltgenaue Zahlen und der Festlegung des Wertes der Kreiszahl *pi* in einer dafür reservierten Variablen. Wie eben schon einmal vorgeführt, wird dann die Funktion für den Kreisbogen definiert. Die folgenden MENÜ-Anweisungen sind Ihnen sicher noch aus früheren Kapiteln geläufig. Das Programm »schläft« dann, wie bei der Verwendung von Menüs üblich, in einer WHILE-Schleife. In der Routine *menuKontrolle* wird dann erst einmal festgestellt, welches Menü der Anwender gewählt hat.

v=MENU(n)
liefert die Nummer des zuletzt vom Benutzer gewählten Menü-Titels ($n = 0$) oder Menü-Punktes ($n = 1$)

Nachdem mit MENÜ festgestellt wurde, was der Benutzer zu tun wünscht, verzweigt *menukontrolle* zu den einzelnen Unterroutinen des Programms. Diese werden nun im einzelnen beschrieben.

wurzel: Diese Subroutine berechnet mit Hilfe der Basic-Funktion SQR die Quadratwurzel aus der eingegebenen Zahl. Damit keine Fehlermeldung bei der Verwendung von SQR erscheinen kann, wird zuvor dafür gesorgt, daß die eingegebene Zahl positiv ist.

v=ABS(x)
liefert den (nichtnegativen) Absolutwert des numerischen Ausdrucks *x*

expo: Diese Subroutine berechnet mit Hilfe der Basic-Funktion EXP die Potenz der eingegebenen Zahl zur Basis der (eulerschen) Zahl *e*.

v=EXP(x)
liefert die Potenz von *x* zur Basis *e* (v = ex)

loga: Diese Subroutine ist das Gegenstück zu *expo*. Sie berechnet mit Hilfe der Basic-Funktion LOG den natürlichen Logarithmus der eingegebenen Zahl.

v=LOG(x)
liefert den natürlichen Logarithmus von *x* zur Basis *e*

kr: Diese Subroutine dient zur Berechnung des Umfangs und der Fläche des Kreises mit dem eingegebenen Radius. Sie prüft zunächst das Vorzeichen des eingegebenen Wertes, bevor dieser weiterverarbeitet wird. Auch hierfür können wir natürlich wieder eine Basic-Funktion verwenden.

v=SGN(x)
das Vorzeichen von x wird in Form einer Konstanten übergeben

Wert von X	SIGN-Ergebnis
x<0	SGN(x) = -1
x=0	SGN(x) = 0
x>0	SGN(x) = 1

Tabelle 5.7: *Mögliche Ergebnisse der SGN-Funktion*

Ist diese Prüfung beendet und der Radius nicht negativ, wird die Kreisfläche und der Kreisumfang berechnet und der Kreis auf dem Bildschirm gezeichnet.

krausschn: Diese Subroutine berechnet in der selbstdefinierten Funktion *FNbog* die Fläche und die Bogenlänge eines Kreisausschnittes in Form eines Kuchenstücks und zeichnet ihn auf dem Bildschirm (CIRCLE kann ja nicht nur volle Kreise, sondern auch

Kreisausschnitte zeichnen). Hierfür werden zwei Eingabewerte benötigt (Radius und Winkel), die von zwei verschiedenen Subroutinen »hereingeholt« werden: *eingabe* und *eingabe2.*

Zu diesen beiden Eingabe-Routinen gibt es eigentlich nichts zu sagen; Sie sollten diese kleinen Programmstücke auch ohne weitere Erklärungen verstehen. Die nächste Routine ist da schon wesentlich »gehaltvoller«.

datum: Diese Routine enthält gleich drei neue, bislang noch nicht verwendete Befehle. Sie gibt das aktuelle Datum in der rechten oberen Bildschirmecke aus – und funktioniert nur dann, wenn Sie mit »Preferences« das korrekte Datum eingestellt haben. Dieses Datum liefert – wie üblich – eine Basic-Funktion.

v$=DATE$
liefert einen String, der das aktuelle Datum im Format *mm-tt-jjjj* (also zum Beispiel 01-07-1987 für den 7. Januar 1987) enthält

Mit zwei weiteren neuen Funktionen und einer bekannten (LOCATE) sorgt das Programm dafür, daß der Text-Cursor vor und nach der Ausgabe des Datums an derselben Stelle im Fenster steht. Hierzu wird die aktuelle Position des Cursors festgestellt, gespeichert und nach der Ausgabe des Datums wiederhergestellt.

v=POS(0)
liefert die Spalte, in der sich der Text-Cursor gerade befindet

v=CSRLIN
liefert die Zeile, in der sich der Text-Cursor gerade befindet

Mit diesem kleinen Mathematik-Programm kann man noch nicht sehr viel anfangen. Sie können es aber nach Ihren Wünschen und Bedürfnissen erweitern und verbessern. Probieren Sie doch zum Beispiel einmal, die folgende trigonometrische Funktion in das Programm zu integrieren und versuchen Sie, deren Ergebnis auch in irgendeiner Form grafisch darzustellen.

v=TAN(x)
liefert den Tangens des Winkels *x* (der im Bogenmaß angegeben werden muß)

Diese Funktion wollte ich eigentlich selbst in das Programm Mini-Mathe einbauen, habe sie dann aber glatt übersehen.

5.9 Der Rest

Eigentlich sind wir jetzt am Ende angelangt. Alles, was ich Ihnen »eigentlich« sagen wollte, ist gesagt. Einige wenige Befehle, die nicht in einem Programm behandelt werden konnten, sind aber noch übriggeblieben. Der Vollständigkeit halber sollen sie nun wenigstens kurz beschrieben werden.

DECLARE FUNCTION Name Parameterliste LIBRARY
definiert eine Funktion, die durch ein Maschinensprache-Unterprogramm realisiert wird, das sich in einer Bibliotheksdatei befindet. Diese Funktion kann nach einer solchen Definition wie eine selbstdefinierte Funktion verwendet werden

LIBRARY "Dateiname"
LIBRARY CLOSE
öffnet eine Bibliothek mit Maschinensprache-Unterprogrammen für den Zugriff von Amiga-Basic. Ein solcher Befehl ist vor dem DEFINE-FUNCTION-Befehl nötig, um Amiga-Basic mitzuteilen, in welcher Bibliotheks-Datei die dort definierte Funktion zu finden ist

Diese beiden Befehle stellen auf recht elegante Weise ein Bindeglied zwischen der Maschinensprache und Basic her. Jede Maschinensprache-Bibliothek, die sich an bestimmte Standards hält, kann auf diese Weise von Basic genutzt werden. Solche Bibliotheken besitzen eine Art Inhaltsverzeichnis, in dem verzeichnet ist, wie die einzelnen Routinen heißen und wo sie in der Bibliothek zu finden sind. Um auf eine Bibliothek von Basic aus zugreifen zu können, muß dieses Inhaltsverzeichnis erst in eine sogenannte ».bmap-Datei« übertragen werden. Eine solche Übertragung (Konvertierung) können Sie mit dem Hilfsprogramm ConvertFD, welches sich in der Schublade »BASICDemos« befindet, durchführen. Nur wenn eine solche .bmap-Datei zu einer Bibliothek vorliegt, können die darin enthaltenen Routinen genutzt werden.

Zwei fertige .bmap-Dateien befinden sich bereits in der Schublade »BASICDemos« (dos.library und graphics.library). Eine davon wird im folgenden Beispielprogramm auch genutzt. Sind Sie an weiteren Informationen zu diesem Thema interessiert, so finden Sie diese im Kapitel 6 und Anhang F des Basic-Handbuches. Ohne weitere Handbücher zum Amiga (die es zum Beispiel vom Addison-Wesley-Verlag gibt) werden Sie aber kaum weiterkommen. Nur in diesen Handbüchern finden Sie nämlich die Informationen, welche Routinen in welchen Bibliotheken zur Verfügung stehen.

```
REM Biblio1
'P.5.9-1
DEFINT a-z
LIBRARY "graphics.library"
```

```
linie:
IF MOUSE(0)=0 THEN linie
x=MOUSE(3)
y=MOUSE(4)
   WHILE MOUSE(0)<>0
   muss=MOUSE(0)
   x1=MOUSE(1)
    y1=MOUSE(2)
   CALL SetDrMd& (WINDOW(8),3)
   LINE (x,y)-(x1,y1),3,bf
   LINE (x,y)-(x1,y1),3,bf
   CALL SetDrMd& (WINDOW(8),1)
   WEND
   LINE (x,y)-(x1,y1),3,bf
END
```

(Dieses Programm nutzt die .bmap-Datei »graphics.library«, die sich in der Schublade »BASICDemos« befindet.) Starten Sie das Programm und drücken Sie die linke Maustaste. Halten Sie dann die Taste fest und bewegen Sie die Maus. Das, was Sie nun auf dem Bildschirm sehen, wäre allein mit Amiga-Basic – zumindest in dieser Geschwindigkeit – kaum möglich.

Ein weiterer Basic-Befehl, der noch nicht an anderer Stelle besprochen wurde, ist die LOC-Funktion.

v=LOC(Dateinummer)
liefert die Nummer des Datensatzes in der Direktzugriffsdatei mit der Kennung *Dateinummer*, auf den zuletzt mit einem GET- oder PUT-Befehl zugegriffen wurde

Hierzu ein Beispiel: Wenn Sie die Anweisung *IF LOC(1)>100 THEN pruefen* in ein Programm einbauen, bewirkt sie einen Sprung zur Marke pruefen, wenn die letzte Operation in der Direktzugriffsdatei Nummer 1 mit einem Datensatz hinter dem einhundertsten Datensatz gearbeitet hat.

Die letzte, bisher noch nicht beschriebene Anweisung, ist die SHARED-Anweisung innerhalb von Unterprogrammen, die besonders bei der Programmierung großer Programme, die viele Unterprogramme benutzen, sehr wichtig ist.

SHARED Variable1 ,Variable2,...
macht aus den genannten Variablen »globale Variablen«, die sowohl im Unterprogramm wie auch in den anderen Teilen des Programms denselben Wert haben

Damit ist natürlich nicht gemeint, daß solche Variablen im Programm einen festen, unveränderlichen Wert haben. Vielmehr ist es ja so, daß zwei Variablen gleichen Namens im Hauptprogramm und im Innern eines Unterprogramms normalerweise zwei verschiedene Objekte sind. Wird der Variablen *a* in einem Unterprogramm der Wert 4 zugewiesen und das Unterprogramm dann beendet, muß *PRINT a* im Hauptprogramm oder einem anderen Unterprogramm nicht unbedingt *4* ergeben. Statt dessen wird *a* hier den Wert haben, der zuletzt im Hauptprogramm der Variablen *a* zugewiesen wurde – auch wenn diese Zuweisung **vor** der Zuweisung *a=4* im Unterprogramm passiert ist. Mit der SHARED-Anweisung kann dieser Effekt auf Wunsch verhindert werden. Gleiche Namen im Haupt- und Unterprogramm sprechen dann dasselbe Objekt (dieselbe Speicherstelle) an. Hierzu zwei Beispiele:

```
REM Beispiel 1
a=50:b=30
c=40:
CALL test
END

SUB test STATIC
PRINT a,b
END SUB
```

Dieses Programm gibt zweimal die Null auf dem Bildschirm aus – obwohl *a* und *b* im Hauptprogramm andere Werte erhalten haben.

```
REM Beispiel 2
a=50:b=30:c=40
CALL test
END

SUB test STATIC
SHARED a, b
PRINT a, b, c
END SUB
```

Das zweite Programm liefert hingegen die Zahlen 50, 30 und 0, da *a* und *b* im Haupt- und Unterprogramm nun dasselbe bedeuten. Die Variable *c* ist jedoch nicht als SHARED definiert und hat im Hauptprogramm zwar den Wert 40, im Unterprogramm jedoch den Wert 0.

6
Abschied

Auch das interessanteste Buch muß einmal ein Ende haben. Ich hoffe zumindest, daß es für Sie interessant war. Und vor allem hoffe ich, daß Sie so viele Anregungen für eigene Programme bekommen haben, daß Sie nicht mehr wissen, welche Sie zuerst in die Tat beziehungsweise in ein Programm umsetzen sollen.

Nun stehen noch einige Danksagungen aus. Zuerst bei Ihnen, daß Sie bis zum Schluß durchgehalten haben. Und dann vor allem bei Lisbeth, meiner Frau, die mich in der letzten Zeit nur noch zum Essen zu Gesicht bekommen hat und die neben ihrer eigenen Arbeit noch Zeit fand, mir beim Korrekturlesen zu helfen. Die letzte Danksagung gilt den Leuten, die den Amiga entwickelt haben, dafür, daß sie es fertiggebracht haben, ein solches Wunderwerk der Technik zu schaffen. Ohne sie gäbe es dieses Buch nicht.

Und nun wünsche ich Ihnen noch viel Freude mit dem Amiga und viele tolle Programmideen und Programme.

7
Übersicht der Basic-Befehle

Mini-Mathe	MENU-Funktion ABS EXP LOG SGN	
	DATE$ POS CSRLIN	P5.8-2
Biblio	LIBRARY	P5.9-1

Anhang 1
VEL

Vielleicht fragen Sie sich, was »VEL« bedeuten soll. Keine Sorge, es ist nicht der Name für ein neues Waschmittel. Es handelt sich vielmehr um ein Datei-Verwaltungsprogramm, und die drei Buchstaben VEL stehen für Verkauf-Einkauf-Lager. In diesem Programm wird alles verwendet, was Sie über Dateien erfahren haben und noch ein bißchen mehr.

Wegen der vielen Möglichkeiten, die das Programm bietet, ist der Programmieraufwand entsprechend hoch gewesen und das schlägt sich natürlich in der Programmlänge nieder. Deshalb folgt zunächst das komplette Programmlisting ohne Unterbrechung durch Erklärungen. Diese finden Sie im Anschluß an das Programm und in einigen wenigen Kommentaren (REMs) im Programm. Bedenken Sie bitte, daß eine Basic-Zeile bis zu 255 Zeichen enthalten darf und eine Druckzeile entsprechend kürzer ist. Auf die Breite dieses Buches passen jedoch weniger als 80 Buchstaben. Der Anfang von »Fortsetzungszeilen« ist deshalb auch in diesem Listing wieder mit einem Doppelstern "**" markiert. Der Inhalt solcher Zeilen gehört an das Ende der vorangehenden Zeile.

Weiterhin läuft das Programm nicht ohne besondere Vorkehrungen. Es benötigt unter anderem eine separate Datendiskette. Starten Sie es deshalb bitte nicht, ohne die Erklärungen im Anschluß an das Listing gelesen zu haben. Nun aber genug der Vorrede, und los gehts!

VEL – Das Programm

```
REM VEL
REM Dateiverwaltungsprogramm
REM by Horst-Rainer Henning
CLEAR ,40000&
ON ERROR GOTO fehler
DEFINT a-z
DIM gz(12)
```

```
CALL titel(datum$)
CALL titel(datum$)
GOSUB grundzahl
PRINT "bitte legen Sie die Daten-Diskette ein"
FOR i=1 TO 5000: NEXT i
OPEN "VEL-DATEN:test" FOR INPUT AS #1
CLOSE #2 'Bei fehlender Datei wird Fehlerroutine aktiv

start:
GOSUB datum
CALL titel(datum$)
PRINT "Bitte Drucker in Startposition":PRINT

seitenlaenge:
LOCATE 20,1:PRINT "Bitte Blattlaenge des Endlosformulares
** eingeben: 72"
LOCATE 20,50:INPUT lg
IF lg=0 THEN lg=72
IF lg<66 OR lg>72 THEN seitenlaenge
xl=lg-66
GOSUB oeffnen
GOSUB menuEin

ablauf:
ON MENU GOSUB menuKontrolle:MENU ON
ON BREAK GOSUB breakkontrolle:BREAK ON
CALL titel(datum$)
aus = 1

ablauf1:
WHILE aus
  SLEEP
WEND
CLOSE
MENU RESET
END
REM allgemeine Routinen
REM @@@@@@@@@@@@@@@@@@@@

ende:
aus=0
GOTO ablauf1

datum:
CALL titel(datum$)

datum1:
LOCATE 15,15
PRINT "Bitte geben Sie das heutige Datum ein > TTMMJJ"
LOCATE 15,58:LINE INPUT datum$
tag$=LEFT$(datum$,2)
```

```
tag=VAL(tag$)
IF tag>31 OR tag<1 THEN datum1
monat$=MID$(datum$,3,2)
monat=VAL(monat$)
IF monat>12 OR monat<1 THEN datum1
jahr$=MID$(datum$,5,2)
jahr=VAL(jahr$)
IF jahr>99 OR jahr<86 THEN datum1
katag=gz(monat)+tag
datum$=tag$+"."+monat$+"."+jahr$
datu=VAL(tag$+monat$)
RETURN
```

grundzahl:

```
FOR i=1 TO 12
  READ gz(i)
NEXT i
DATA 0,31,59,90,120,151,181,212,243,273,304,334
RETURN
```

datumrechnen:

```
njahr=jahr
IF liefter>365 THEN liefter=liefter-365:njahr=jahr+1
zeig3=0
FOR i = 1 TO 12
  dare=liefter-gz(i)
  IF dare<1 AND zeig3 = 0 THEN darech=gz(i-1) :da=i-
** 1:zeig3=1
NEXT
dat$=STR$(da)
ta=liefter-darech
tage$=STR$(ta)
jahre$=STR$(njahr)
liefdat$=tage$+"."+dat$+"."+jahre$
RETURN
SUB titel(zeit$) STATIC
CLS
IF z=0 THEN GOSUB init
PUT (0,0),tit
LOCATE 2,57:PRINT "MARKT&TECHNIK VERLAG"
LOCATE 3,57:PRINT "by Horst-R. Henning"
LOCATE 7,57:PRINT "Datum: "zeit$
LOCATE 10,1
GOTO zur
```

init:

```
x=20 : y=10
FOR i = 0 TO 7
  LINE (i,i)-(i+350,i)
```

```
   LINE -(i+350,i+50)
   LINE -(i,i+50)
   LINE -(i,i)
   LINE (x+i,y+i)-(x+10+i,y+30+i),3
   LINE -(x+30+i,y+i),3
   FOR j = 0 TO 30 STEP 15
        LINE (x+50+i,y+j+i)-(x+80+i,y+j+i),3
   NEXT j
   LINE (x+50+i,y+i)-(x+50+i,y+30+i),3
   LINE (x+100+i,y+i)-(x+100+i,y+30+i),3
   LINE -(x+130+i,y+30+i),3
NEXT i
LOCATE 2,20:PRINT "DATEIVERWALTUNGSPROGRAMM"
LOCATE 5,20:PRINT "Verkauf Einkauf Lager"
DIM tit(5158)
GET (0,0)-(355,55),tit
z=1
RETURN

zur:
END SUB

fehler:
IF ERR <>53 THEN CALL titel(datum$):PRINT "Programmabbruch
** wegen Fehlernummer: ";ERR : RESUME ende
RESUME anlegen

taste:
ta$=INKEY$
IF ta$<>"" THEN ta$=UCASE$(ta$) ELSE GOTO taste
RETURN

taste1:
ta$=INKEY$
IF ta$<CHR$(129) OR ta$>CHR$(131) THEN taste1 ELSE RETURN

funkt:
LOCATE 23,1:PRINT "F1=abbrechen F2=weiter F3=ausdrucken";
LOCATE 3,1
RETURN

funkt1:
LOCATE 23,1:PRINT "F1 = abbrechen F2 = speichern";
RETURN

funkt2:
LOCATE 23,1:PRINT "F1=abbrechen F2=weiter ";
RETURN

REM Datei Routinen
REM **************
```

anlegen:

```
CALL titel(datum$)
PRINT "Achtung!!! Saemtliche Daten der Diskette >VEL-DATEN<
** werden geloescht!"
PRINT :PRINT "F1 = abbrechen F2 = weiter "
GOSUB taste1
IF ta$=CHR$(129) THEN ende
OPEN "VEL-DATEN:test" FOR OUTPUT AS#1
  WRITE #1,"angelegt"
CLOSE #1
```

aktivieren:

```
CALL titel(datum$)
PRINT ,"Die Datendiskette wird aktiviert"
PRINT ,"Die Aktivierung dauert 15 Minuten"
PRINT ,"Sie koennen also eine Kaffepause machen"
GOSUB oeffnen
LSET Z1$=CHR$(0)
FOR n=1 TO 100
  PUT #1,n
  LOCATE 15,1:PRINT ,,"Kunden-Datensatz ";n
NEXT n
LSET Z2$=CHR$(0)
FOR n=1 TO 100
  PUT #2,n
  LOCATE 15,1:PRINT ,,"Lieferanten-Datensatz ";n
NEXT n
LSET Z3$=CHR$(0)
FOR n=1 TO 1000
  PUT #3,n
  LOCATE 15,1:PRINT ,,"Auftrags-Datensatz ";n
NEXT n
LSET Z4$=CHR$(0)
FOR n=1 TO 1000
  PUT #4,n
  LOCATE 15,1:PRINT ,,"Bestellungen-Datensatz ";n
NEXT n
LSET Z5$=CHR$(0)
FOR n=1 TO 1000
  PUT #5,n
  LOCATE 15,1:PRINT ,,"Artikel-Datensatz ";n
NEXT n
LSET Z6$=CHR$(0)
FOR n=1 TO 1000
  PUT #6,n
  LOCATE 15,1:PRINT ,,"Bestands-Datensatz ";n
NEXT n
LSET Z7$=CHR$(0)
```

```
FOR n=1 TO 1000
  PUT #7,n
  LOCATE 15,1:PRINT ,,"Angebots-Datensatz ";n
NEXT n
LSET Z8$=CHR$(0)
FOR n=1 TO 1000
  PUT #8,n
  LOCATE 15,1:PRINT ,,"Preislisten-Datensatz ";n
NEXT n
LSET Z9$=CHR$(0)
FOR n=1 TO 100
  PUT #9,n
  LOCATE 15,1:PRINT ,,"Liefer- u.Zahlungsbed. ";n
NEXT n
GOTO start
```

oeffnen:

```
CLOSE
OPEN"R",#1,"VEL-DATEN:Kunden",84
FIELD#1,1 AS Z1$,15 AS N1$,15 AS N2$,25 AS N3$,4 AS N4$,20
** AS N5$,2 AS B1$,2 AS B2$
OPEN"R",#2,"VEL-DATEN:Lieferanten",84
FIELD#2,1 AS Z2$,15 AS N6$,15 AS N7$,25 AS N8$,4 AS N9$,20
** AS N0$,2 AS B3$,2 AS B4$
OPEN"R",#3,"VEL-DATEN:Auftraege",17
FIELD#3,1 AS Z3$,2 AS D3$,2 AS V3$,2 AS M3$,2 AS A3$,4 AS
** P3$, 2 AS L3$,2 AS G3$
OPEN"R",#4,"VEL-DATEN:Bestellungen",17
FIELD#4,1 AS Z4$,2 AS D4$,2 AS E4$,2 AS M4$,2 AS A4$,4 AS
** P4$, 2 AS L4$,2 AS G4$
OPEN"R",#5,"VEL-DATEN:Artikel",40
FIELD#5,1 AS Z5$,2 AS L5$,37 AS a5$
OPEN"R",#6,"VEL-DATEN:Bestand",11
FIELD#6,1 AS Z6$,2 AS W1$,2 AS W2$,2 AS W3$,2 AS W4$,2 AS
** W5$
OPEN"R",#7,"VEL-DATEN:Angebote",23
FIELD#7,1 AS Z7$,2 AS E7$,2 AS M7$,2 AS R1$,4 AS S1$,2 AS
** R2$, 4 AS S2$,2 AS R3$,4 AS S3$
OPEN"R",#8,"VEL-DATEN:Preisliste",19
FIELD#8,1 AS Z8$,2 AS R4$,4 AS S4$,2 AS R5$,4 AS S5$,2 AS
** R6$, 4 AS S6$
OPEN"R",#9,"VEL-DATEN:ZuL.Bed",61
FIELD#9,1 AS Z9$,30 AS B9$,30 AS B8$
RETURN

REM Menue Routinen
REM ##############
```

menuEin:

```
MENU 1,0,1,"VERKAUF "
MENU 1,1,1,"Kunden neu"
MENU 1,2,1,"Auftrag neu"
MENU 1,3,1,"Preisliste"
MENU 2,0,1,"EINKAUF "
MENU 2,1,1,"Angebote neu"
MENU 2,2,1,"Lieferanten neu"
MENU 2,3,1,"Lieferbeding.neu"
MENU 2,4,1,"Zahlungsbed. neu"
MENU 3,0,1,"LAGER "
MENU 3,1,1,"Artikel neu"
MENU 3,2,1,"Wareneingang"
MENU 3,3,1,"Bestand "
MENU 4,0,1,"AUSKUNFT "
MENU 4,1,1,"Kunden suchen"
MENU 4,2,1,"Lieferanten suchen"
MENU 4,3,1,"Artikel-Liste"
MENU 4,4,1,"Angebote zeigen"
MENU 4,5,1,"Lieferbed. zeigen"
MENU 4,6,1,"Zahlungsb. zeigen"
MENU 4,7,1,"Bestand zeigen"
MENU 4,8,1,"Preisliste zeigen"
MENU 4,9,1,"alle Auftraege zeigen"
MENU 4,10,1,"offene Auftraege zeig"
MENU 4,11,1,"alle Bestellungen zei"
MENU 4,12,1,"offene Bestellg. zeig"
MENU 4,13,1,"Terminrueckstand zeig"
MENU 5,0,1,"beenden"
MENU 5,1,1,"E N D E"
RETURN
```

menuKontrolle:

```
menuTitel = MENU(0)
menuNummer= MENU(1)
ON menuTitel GOTO verkauf,einkauf,lager,auskunft,beenden
```

verkauf:

```
ON menuNummer GOSUB kundenNeu,auftragneu,preisliste
CALL titel(datum$)
RETURN
```

einkauf:

```
ON          menuNummer              GOSUB
** angebote,lieferantenNeu,liebedneu,zahlbedneu
CALL titel(datum$)
RETURN
```

```
lager:
ON menuNummer GOSUB artikel,wareneingang,bestand
CALL titel(datum$)
RETURN

auskunft:
ON                menuNummer          GOSUB
** kunden,lieferanten,artikelListe,angebotezei,
** liebedzei,zahlbedzei,bestandzei,preislistezei,auftragzei,
** auftragzeioffen,bestellungzei,bestellungzeioffen,
** rueckstand
CALL titel(datum$)
RETURN

beenden:
ON menuNummer GOSUB abschalten
CALL titel(datum$)
RETURN

kundenNeu:
zei=1:a=0
WHILE zei
  a=a+1
  GET#1,a
  IF Z1$=CHR$(0) THEN zei=0
WEND
satz=a

kundenNeu1:
CALL titel(datum$)
LOCATE 9,1: PRINT "NEUEN KUNDEN ANLEGEN"
GOSUB adresse
GOSUB funkt1
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
LSET Z1$=CHR$(1)
LSET N1$=nam$
LSET N2$=abt$
LSET N3$=ste$
LSET N4$=plz$
LSET N5$=ort$
LSET B1$=lie$
LSET B2$=zah$
PUT #1,satz
satz=satz+1
GOTO kundenNeu1

adresse:
LOCATE 13,1: PRINT "Nummer: ";satz
LOCATE 15,1: LINE INPUT "Firmenname: ";nam$
```

```
LOCATE 16,1: INPUT "Abteilung: ";abt$
LOCATE 17,1: INPUT "Strasse: ";ste$
LOCATE 18,1: LINE INPUT "Postleitzahl: ";plz$
LOCATE 19,1: INPUT "Ort: ";ort$
LOCATE 20,1: INPUT "Lieferbeding. ";lie$
LOCATE 21,1: INPUT "Zahlungsbedin.";zah$
RETURN
```

auftragneu:

```
CALL titel(datum$)
GOSUB funkt2
LOCATE 15,1:PRINT "Bitte Funktionstaste druecken"
GOSUB taste1
IF ta$=CHR$(129) THEN bestandsmeldung
CALL titel(datum$)
PRINT "A U F T R A G A N L E G E N",datum$
zei=1:a=0
WHILE zei
  a=a+1
  GET#3,a
  IF Z3$=CHR$(0) THEN zei=0
WEND
satz=a
PRINT "Auftragsnummer: ",a
10 INPUT "Kunde ";firma
GET#1,firma
PRINT N1$;N2$
PRINT N4$;N5$
INPUT "Artikel-Nummer ";artik
GET#5,artik
PRINT a5$
INPUT "Menge in Stueck: ";bedarf
GET#8,artik
IF bedarf>=CVI(R5$)THEN
  IF bedarf>=CVI(R6$)THEN
       preis!=CVS(S6$)
  ELSE
       preis!=CVS(S5$)
  END IF
ELSE
  IF bedarf<CVI(R4$) THEN bedarf=CVI(R4$)
  preis!=CVS(S4$)
END IF
GET#5,artik
wbz=VAL(L5$)
INPUT "Liefertermin in Wochen";liefter
liefter=(liefter*7)+katag
GET#6,artik
```

```
IF liefter<(katag+(wbz*7)) AND (CVI(W1$)-CVI(W4$))<bedarf
** THEN liefter=katag+(wbz*7)
gelie=0
PRINT "Kalendertag= ";liefter
GOSUB funkt1
GOSUB taste1
IF ta$=CHR$(129) THEN bestandsmeldung
GOSUB auftragspei
GOTO auftragneu
```

auftragspei:

```
GET#3,satz
LSET Z3$=CHR$(1)
LSET D3$=MKI$(datu)
LSET V3$=MKI$(firma)
LSET M3$=MKI$(bedarf)
LSET A3$=MKI$(artik)
LSET P3$=MKS$(preis!)
LSET L3$=MKI$(liefter)
LSET G3$=MKI$(gelie)
PUT #3,satz
GET#4,artik
vormerke=bedarf+CVI(W4$)
LSET W4$=MKI$(vormerke)
PUT #6,artik
GOSUB auftragsbestdruck
RETURN
```

preisliste:

```
CALL titel(datum$)
PRINT "P R E I S L I S T E A E N D E R N O D E R A N L E
** G E N ":PRINT
INPUT "Artikel-Nummer ";satz
GET#8,satz
IF Z8$=CHR$(0) THEN bm1=0:lp1!=0:bm2=0:lp2!=0:bm3=0:lp3!=0:
**  GOSUB prlispei:GET#8,satz
LOCATE 15,1:PRINT "Bestellmenge 1: ";CVI(R4$)
LOCATE 15,24:INPUT bm1
IF bm1 =0 THEN bm1=CVI(R4$)
LOCATE 16,1:PRINT "Listenpreis 1: ";CVS(S4$)
LOCATE 16,24:INPUT lp1!
IF lp1!=0 THEN lp1!=CVS(S4$)
LOCATE 17,1:PRINT "Bestellmenge 2: ";CVI(R5$)
LOCATE 17,24:INPUT bm2
IF bm2 =0 THEN bm2=CVI(R5$)
LOCATE 18,1:PRINT "Listenpreis 2: ";CVS(S5$)
LOCATE 18,24:INPUT lp2!
IF lp2!=0 THEN lp2!=CVS(S5$)
LOCATE 19,1:PRINT "Bestellmenge 3:  ";CVI(R6$)
```

```
LOCATE 19,24:INPUT bm3
IF bm3 =0 THEN bm3=CVI(R6$)
LOCATE 20,1:PRINT "Listenpreis 3: ";CVS(S6$)
LOCATE 20,24:INPUT lp3!
IF lp3!=0 THEN lp3!=CVS(S6$)
GOSUB funkt1
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
GOSUB prlispei
GOTO preisliste
```

prlispei:

```
LSET Z8$=CHR$(1)
LSET R4$=MKI$(bm1)
LSET S4$=MKS$(lp1!)
LSET R5$=MKI$(bm2)
LSET S5$=MKS$(lp2!)
LSET R6$=MKI$(bm3)
LSET S6$=MKS$(lp3!)
PUT #8,satz
RETURN
RETURN
```

auftragsabwicklung:

```
zeig=1:a=1
GET#3,a
WHILE zeig 'prueft ob Auftraege zur Lieferung faellig
  IF Z3$=CHR$(1) AND katag >=CVI(L3$) THEN
       artik=CVI(A3$) 'Artikel-Nummer
       bedarf=CVI(M3$) 'Bestell-Menge
       GET#6,artik
  IF CVI(W1$)>=bedarf THEN 'ist Bestell-Menge auf Lager?
            satz=a
            GOSUB rechnung
            GET#3,a 'Auftrags Datei
            LSET Z3$=CHR$(2) 'Zeiger CHR$(2)=erledigt
            LSET G3$=MKI$(datu) 'Lieferdatum
            PUT #3,a
       END IF
  END IF
  a=a+1 'naechster Auftragssatz
  GET#3,a
  IF Z3$=CHR$(0) THEN zeig=0 'letzter Auftragssatz?
WEND
RETURN
```

rechnung:

```
REM rechnung
  istmenge=CVI(W1$)-bedarf
```

```
   vormerke=CVI(W4$)-bedarf
   IF vormerke<0 THEN vormerke=0
   LSET W1$=MKI$(istmenge)
   LSET W4$=MKI$(vormerke)
   PUT #6,artik
   firma=CVI(V3$)
   preis!=CVS(P3$)
GOSUB rechnungsdruck
RETURN
```

angebote:

```
CALL titel(datum$)
PRINT "A N G E B O T E A E N D E R N O D E R A N L E
** G E N ":PRINT
INPUT "Angebots-Nummer ";satz
GET#7,satz
IF Z7$=CHR$(0) THEN lfn=0:mmg=0:bm1=0:lp1!=0:bm2=0:lp2!=0:
** bm3=0:lp3!=0:GOSUB angebotspei:GET#7,satz
LOCATE 13,1:PRINT "Lieferanten-Nummer ";CVI(E7$)
LOCATE 13,24:INPUT lfn
IF lfn =0 THEN lfn=CVI(E7$)
LOCATE 14,1:PRINT "Mindestmenge: ";CVI(M7$)
LOCATE 14,24:INPUT mmg
IF mmg =0 THEN mmg=CVI(M7$)
LOCATE 15,1:PRINT "Bestellmenge 1: ";CVI(R1$)
LOCATE 15,24:INPUT bm1
IF bm1 =0 THEN bm1=CVI(R1$)
LOCATE 16,1:PRINT "Staffelpreis 1 : ";CVS(S1$)
LOCATE 16,24:INPUT lp1!
IF lp1!=0 THEN lp1!=CVS(S1$)
LOCATE 17,1:PRINT "Bestellmenge 2: ";CVI(R2$)
LOCATE 17,24:INPUT bm2
IF bm2 =0 THEN bm2=CVI(R2$)
LOCATE 18,1:PRINT "Listenpreis 2: ";CVS(S2$)
LOCATE 18,24:INPUT lp2!
IF lp2!=0 THEN lp2!=CVS(S2$)
LOCATE 19,1:PRINT "Bestellmenge 3: ";CVI(R3$)
LOCATE 19,24:INPUT bm3
IF bm3 =0 THEN bm3=CVI(R3$)
LOCATE 20,1:PRINT "Listenpreis 3: ";CVS(S3$)
LOCATE 20,24:INPUT lp3!
IF lp3!=0 THEN lp3!=CVS(S3$)
GOSUB funkt1
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
GOSUB angebotspei
GOTO angebote
```

```
angebotspei:
LSET Z7$=CHR$(1)
LSET E7$=MKI$(lfn)
LSET M7$=MKI$(mmg)
LSET R1$=MKI$(bm1)
LSET S1$=MKS$(lp1!)
LSET R2$=MKI$(bm2)
LSET S2$=MKS$(lp2!)
LSET R3$=MKI$(bm3)
LSET S3$=MKS$(lp3!)
PUT #7,satz
RETURN

bestandsmeldung:
CALL titel(datum$)
PRINT "Rechner prueft ob Bestellungen ausgeloest werden
** muessen."
GOSUB bestellungsabwicklung
CALL titel(datum$)
PRINT "Rechner prueft ob Auftraege ausgeliefert werden
** koennen."
GOSUB auftragsabwicklung
RETURN

bestellungsabwicklung:
zeig=1:a5=1
GET#6,a5
WHILE zeig
   IF (CVI(W1$)+CVI(W5$)-CVI(W4$))<CVI(W2$) THEN
        bedarf=(CVI(W3$)+CVI(W4$))-(CVI(W1$)+CVI(W5$))
        artik=a5
        GOSUB bestellung
   END IF
   a5=a5+1
   GET#6,a5
   IF Z6$=CHR$(0) THEN zeig=0
WEND
RETURN

bestellung:
zei=1:a=0
WHILE zei
   a=a+1
   GET#4,a
   IF Z4$=CHR$(0) THEN zei=0
WEND
satz=a
GET#7,artik
firma=CVI(E7$)
```

```
IF bedarf>=CVI(R2$)THEN
   IF bedarf>=CVI(R3$)THEN
        preis!=CVS(S3$)
   ELSE
        preis!=CVS(S2$)
   END IF
ELSE
   IF bedarf<CVI(M7$) THEN bedarf=CVI(M7$)
   preis!=CVS(S1$)
END IF
GET#5,artik
wbz=VAL(L5$)
liefter=katag+((wbz-1)*7)
gelie=0
LSET Z4$=CHR$(1)
LSET D4$=MKI$(datu)
LSET E4$=MKI$(firma)
LSET M4$=MKI$(bedarf)
LSET A4$=MKI$(artik)
LSET P4$=MKS$(preis!)
LSET L4$=MKI$(liefter)
LSET G4$=MKI$(gelie)
PUT #4,satz
bestellt=bedarf+CVI(W5$)
LSET W5$=MKI$(bestellt)
PUT #6,artik
GOSUB bestelldruck
RETURN
```

lieferantenNeu:

```
zei=1:a=0
WHILE zei
   a=a+1
   GET#2,a
   IF Z2$=CHR$(0) THEN zei=0
WEND
satz=a
```

lieferantenNeu1:

```
CALL titel(datum$)
LOCATE 9,1: PRINT "NEUEN LIEFERANTEN ANLEGEN"
GOSUB adresse
GOSUB funkt1
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
LSET Z2$=CHR$(1)
LSET N6$=nam$
LSET N7$=abt$
LSET N8$=ste$
```

```
LSET N9$=plz$
LSET N0$=ort$
LSET B3$=lie$
LSET B4$=zah$
PUT #2,satz
satz=satz+1
GOTO lieferantenNeu1
```

liebedneu:

```
ueber$="NEUE LIEFERBEDINGUNGEN ANLEGEN":zei=1:a=0
```

liebedneu2:

```
WHILE zei
   a=a+1
   GET#9,a
   IF Z9$=CHR$(0) THEN zei=0
WEND
satz=a
```

liebedneu1:

```
lg1$=STRING$(30,168):lg2$=lg1$
te1$="1.Zeile Bedingungen: "
te2$="2.Zeile Bedingungen: ":GOSUB maske1
IF ta$=CHR$(129) THEN RETURN
LSET Z9$=CHR$(1)
LSET B9$=eeg$
LSET B8$=zeg$
PUT #9,satz
satz=satz+1
GOTO liebedneu1
```

zahlbedneu:

```
ueber$="NEUE ZAHLUNGSBEDINGUNGEN ANLEGEN":zei=1:a=49
GOTO liebedneu2
```

artikel:

```
zei=1:a=0
WHILE zei
   a=a+1
   GET#5,a
   IF Z5$=CHR$(0) THEN zei=0
WEND
satz=a
```

artikel1:

```
ueber$="NEUEN ARTIKEL ANLEGEN"
lg1$=STRING$(2,168):lg2$=STRING$(37,168)
te1$="Wiederbesch.(Wochen): "
te2$="Artikelbezeichnung: ":GOSUB maske1
IF ta$=CHR$(129) THEN RETURN
```

```
LSET Z5$=CHR$(1)
LSET L5$=eeg$
LSET a5$=zeg$
PUT #5,satz
satz=satz+1
GOTO artikel1
```

maske1:

```
CALL titel(datum$)
LOCATE 9,10: PRINT ueber$
LOCATE 9,60: PRINT "Datum: ";datum$
LOCATE 11,1: PRINT "Nummer: ..... ";satz
LOCATE 15,1: PRINT te1$
LOCATE 16,24: PRINT lg1$
LOCATE 15,24: LINE INPUT eeg$
LOCATE 18,1: PRINT te2$
LOCATE 19,24: PRINT lg2$
LOCATE 18,24: LINE INPUT zeg$
GOSUB funkt1
GOTO taste1
RETURN
```

bestand:

```
CALL titel(datum$)
PRINT "B E S T A N D A E N D E R N O D E R A N L E G
** E N ":PRINT
INPUT "Artikel-Nummer ";satz
GET#6,satz
IF Z6$=CHR$(0) THEN tbe=0:ube=0:obe=0:vbe=0:bbe=0:
** GOSUB bestspei: GET#6,satz
LOCATE 15,1:PRINT "tatsaechlicher Bestand: ";CVI(W1$)
LOCATE 15,24:INPUT tbe
IF tbe =0 THEN tbe=CVI(W1$)
LOCATE 16,1:PRINT "unterer Bestand: ";CVI(W2$)
LOCATE 16,24:INPUT ube
IF ube =0 THEN ube=CVI(W2$)
LOCATE 17,1:PRINT "oberer Bestand: ";CVI(W3$)
LOCATE 17,24:INPUT obe
IF obe =0 THEN obe=CVI(W3$)
LOCATE 18,1:PRINT "Vormerke:  ";CVI(W4$)
LOCATE 18,24:INPUT vbe
IF vbe =0 THEN vbe=CVI(W4$)
LOCATE 19,1:PRINT "bestellter Bestand: ";CVI(W5$)
LOCATE 19,24:INPUT bbe
IF bbe =0 THEN bbe=CVI(W5$)
GOSUB funkt1
GOSUB taste1
```

```
IF ta$=CHR$(129) THEN bestandsmeldung
GOSUB bestspei
GOTO bestand
```

bestspei:

```
LSET Z6$=CHR$(1)
LSET W1$=MKI$(tbe)
LSET W2$=MKI$(ube)
LSET W3$=MKI$(obe)
LSET W4$=MKI$(vbe)
LSET W5$=MKI$(bbe)
PUT #6,satz
RETURN
```

wareneingang:

```
CALL titel(datum$)
INPUT "bitte Bestell-Nummer eingeben: ";eingang
PRINT
INPUT "bitte Liefermenge in Stueck: ";menge
GET #4,eingang
LSET Z4$=CHR$(2)
LSET G4$=MKI$(datu)
PUT #4,eingang
satz=CVI(A4$)
GET #6,satz
bestellt=CVI(W5$)-menge
IF bestellt<0 THEN bestellt=0
menge=CVI(W1$)+menge
LSET W1$=MKI$(menge)
LSET W5$=MKI$(bestellt)
PUT #6,satz
GOTO bestandsmeldung
```

kunden:

```
CALL titel(datum$)
LOCATE 9,1:PRINT "SUCHEN NACH KUNDEN":PRINT
INPUT "bitte die ersten 4 Buchstaben des Namens: ";such$
PRINT :PRINT "NUMMER ANSCHRIFT":PRINT
zei=1:a=0
WHILE zei
  a=a+1
  GET#1,a
  IF Z1$=CHR$(0) THEN zei=0
  IF LEFT$(N1$,4)=such$ THEN
  PRINT USING"###";a;
  PRINT " ";
  PRINT N1$;N2$,N3$
  PRINT N4$;" ";N5$;" Lieferbeding: ";B1$" Zahlungsbeding:
** ";B2$
```

```
  END IF
WEND
PRINT "Ende der Eintragungen"
GOSUB funkt2
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
GOTO kunden
```

lieferanten:

```
CALL titel(datum$)
LOCATE 9,1:PRINT "SUCHEN NACH LIEFERANTEN":PRINT
INPUT "bitte die ersten 4 Buchstaben des Namens: ";such$
PRINT :PRINT "NUMMER ANSCHRIFT":PRINT
zei=1:a=0
WHILE zei
  a=a+1
  GET#2,a
  IF Z2$=CHR$(0) THEN zei=0
  IF LEFT$(N6$,4)=such$ THEN
  PRINT USING"###";a;
  PRINT " ";
  PRINT N6$;N7$,N8$
  PRINT N9$;" ";N0$;" Lieferbeding: ";B3$;"
** Zahlungsbeding:";B4$
  END IF
WEND
PRINT "Ende der Eintragungen"
GOSUB funkt2
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
GOTO lieferanten
```

artikelListe:

```
a=0
```

artikelListe1:

```
a1=a+15
GOSUB arlizei
```

artikelListe2:

```
GOSUB taste1
IF ta$=CHR$(130) AND a>=a1 THEN artikelListe1
IF ta$=CHR$(129) THEN RETURN
IF ta$=CHR$(131) THEN a2=a:GOSUB arlidruck:a=a2
IF a<a1 THEN PRINT "Ende der Eintragungen"
GOTO artikelListe2
```

arlizei:

```
CLS
PRINT "A R T I K E L L I S T E ";datum$
```

```
GOSUB funkt
zei=1
WHILE zei
  a=a+1
  IF a>=a1 THEN zei=0
  GET#5,a
  IF Z5$=CHR$(0) THEN
       zei=0
  ELSE
       PRINT USING "***#";a;
       PRINT " ";L5$;" ";a5$
  END IF
WEND
RETURN
```

angebotezei:

```
CLS
PRINT ,"A N G E B O T E ";datum$
INPUT "Angebots-Nummer: ";a
PRINT
GET#7,a
IF Z7$=CHR$(0) THEN PRINT "Angebot nicht vorhanden":GOTO
** angebotezei1
GET#5,a
PRINT "Artikel: ";a5$
lief=CVI(E7$)
GET#2,lief
PRINT "Lieferanten-Nummer: ";lief
PRINT "Lieferant: ";N6$,N0$
PRINT "Mindestmenge: ";CVI(M7$)
PRINT "Menge 1: ";CVI(R1$);" Preis 1: ";CVS(S1$):PRINT
PRINT "Menge 2: ";CVI(R2$);" Preis 2: ";CVS(S2$):PRINT
PRINT "Menge 3: ";CVI(R3$);" Preis 3: ";CVS(S3$)
```

angebotezei1:

```
GOSUB funkt2
GOSUB taste1
IF ta$=CHR$(129) THEN RETURN
GOTO angebotezei
```

zahlbedzei:

```
a=49:tex$="Z A H L U N G S B E D I N G U N G E N"
GOTO liebedzei1
```

liebedzei:

```
a=0:tex$="L I E F E R B E D I N G U N G E N "
```

liebedzei1:

```
a1=a+15:CLS
GOSUB funkt2
```

```
LOCATE 1,1
PRINT tex$,datum$
PRINT
zei=1
WHILE zei
   a=a+1
   IF a>a1 THEN zei=0
   GET#9,a
   IF Z9$=CHR$(0) THEN
        zei=0
   ELSE
        PRINT USING "***#";a;
        PRINT " ";B9$;" ";B8$
   END IF
WEND
```

liebedzei2:

```
GOSUB taste1
IF ta$=CHR$(130) AND a>=a1 THEN liebedzei1
IF ta$=CHR$(129) THEN RETURN
IF a<a1 THEN PRINT "Ende der Eintragungen"
GOTO liebedzei2
```

bestandzei:

```
a=0
```

bestandzei1:

```
a1=a+15
CLS
GOSUB funkt2
LOCATE 1,1
PRINT "B E S T A N D ","Datum: ";datum$:PRINT
PRINT " Artikel ist unterer oberer Vor- Bestell-"
PRINT " Nummer Bestand Bestand Bestand merke Bestand":
** PRINT
zei=1
WHILE zei
   a=a+1
   IF a>a1 THEN zei=0
   GET#6,a
   IF Z6$=CHR$(0) THEN
        zei=0
   ELSE
        PRINT USING "##########";a,CVI(W1$),CVI(W2$),CVI(W3$),
        **  CVI(W4$),CVI(W5$)
   END IF
WEND
```

```
bestandzei2:
GOSUB taste1
IF ta$=CHR$(130) AND a>=a1 THEN bestandzei1
IF ta$=CHR$(129) THEN RETURN
IF a<a1 THEN PRINT "Ende der Eintragungen"
GOTO bestandzei2

preislistezei:
a=0

preislistezei1:
a1=a+15
CLS
PRINT "P R E I S L I S T E ","Datum: ";datum$
GOSUB funkt
tex2$= " Artikel Bestell Staffel Bestell Staffel Bestell
** Staffel"
tex3$= " Nummer Menge 1 Preis 1 Menge 2 Preis 2 Menge 3
** Preis 3"
PRINT tex2$
PRINT tex3$
PRINT
zei=1
WHILE zei
  a=a+1
  IF a>a1 THEN zei=0
  GET#8,a
  IF Z8$=CHR$(0) THEN
       zei=0
  ELSE
       PRINT USING "######.##";a,CVI(R4$) ,CVS(S4$),CVI(R5$),
       ** CVS(S5$),CVI(R6$),CVS(S6$)
  END IF
WEND

preislistezei2:
GOSUB taste1
IF ta$=CHR$(130) AND a>=a1 THEN preislistezei1
IF ta$=CHR$(129) THEN RETURN
IF ta$=CHR$(131) THEN a2=a:GOSUB preislidruck:a=a2
IF a<a1 THEN PRINT "Ende der Eintragungen"
GOTO preislistezei2
REM ##############################################
REM ENDE ERSTER TEIL

REM ##############################################
REM ##############################################
REM ANFANG ZWEITER TEIL
REM ##############################################
```

```
auftragzei:
zeiger=0
text1$="A L L E A U F T R A E G E "
GOTO auftragzei3

auftragzeioffen:
zeiger=1
text1$="A U F T R A E G E N O C H N I C H T G E L I E F
** E R T "

auftragzei3:
a=0

auftragzei1:
GOSUB leiste
WHILE zei
  a=a+1
  IF a>a1 THEN zei=0
  GET#3,a
  IF Z3$=CHR$(1) AND zeiger=1 THEN GOSUB auftragzeile
  IF Z3$<>CHR$(0) AND zeiger=0 THEN GOSUB auftragzeile
  IF Z3$=CHR$(0) THEN zei=0
WEND

auftragzei2:
GOSUB taste1
IF ta$=CHR$(130) AND a>=a1 THEN auftragzei1
IF ta$=CHR$(129) THEN RETURN
IF a<a1 THEN PRINT "Ende der Eintragungen"
GOTO auftragzei2

auftragzeile:
PRINT USING "########";a,CVI(D3$),CVI(V3$),CVI(M3$),CVI(A3$);
PRINT USING "######.##";CVS(P3$);
liefter =CVI(L3$):GOSUB datumrechnen
PRINT " ";liefdat$;
PRINT USING "##########";CVI(G3$)
RETURN

leiste:
a1=a+15
CLS
GOSUB funkt2
LOCATE 1,1
PRINT text1$;datum$:PRINT
PRINT " Nummer Schreib Firma Menge Artikel Preis
** Liefer tatsaechl"
PRINT " Datum Stueck Nummer DM
```

```
** Termin Termin":PRINT
zei=1
RETURN

rueckstand:
zeiger=2
text1$="B E S T E L L U N G E N MIT TERMINRUECKSTAND "
GOTO bestellungzei3

bestellungzei:
zeiger=0
text1$="A L L E B E S T E L L U N G E N "
GOTO bestellungzei3
bestellungzeioffen:
zeiger=1
text1$="B E S T E L L U N G E N  N O C H N I C H T G E
L I
** E F E R T "

bestellungzei3:
a=0:

bestellungzei1:
GOSUB leiste
WHILE zei
  a=a+1
  IF a>a1 THEN zei=0
  GET#4,a
  IF zeiger=2 AND Z4$=CHR$(1) AND katag>CVI(L4$) THEN
** GOSUB bestellzeile
  IF zeiger=1 AND Z4$=CHR$(1) THEN GOSUB bestellzeile
  IF zeiger=0 AND Z4$<>CHR$(0) THEN GOSUB bestellzeile
  IF Z4$=CHR$(0) THEN zei=0
WEND

bestellungzei2:
GOSUB taste1
IF ta$=CHR$(130) AND a>=a1 THEN bestellungzei1
IF ta$=CHR$(129) THEN RETURN
IF a<a1 THEN PRINT "Ende der Eintragungen"
GOTO bestellungzei2

bestellzeile:
PRINT USING "########";a,CVI(D4$),CVI(E4$),CVI(M4$),CVI(A4$);
PRINT USING "######.##";CVS(P4$);
liefter =CVI(L4$):GOSUB datumrechnen
PRINT " ";liefdat$;
PRINT USING "#########";CVI(G4$)
RETURN
```

```
abschalten:
aus=0
RETURN
REM Drucker Routinen
REM ################

rechnungsdruck:
CALL titel(datum$)
laenge$=STRING$(29,42)
satz=a
GET#1,firma
GOSUB briefkopf
GOSUB briefkopf2
LPRINT "R E C H N U N G N U M M E R ";satz
gpreis!=preis!*bedarf
mwst!=gpreis!*.14
rebetr!=gpreis!+mwst!
GOSUB artikeldruck
LPRINT TAB(20)" Gesamt-Preis 14% MwSt Rechnungs-Betrag"
LPRINT TAB(20)"DM ";:LPRINT USING
** "########.##";gpreis!,mwst!, rebetr!
GET#9,VAL(B2$)
LPRINT
LPRINT
LPRINT "Zahlung erbitten wir: ";B9$
LPRINT " ";B8$
LPRINT
LPRINT "Auf unser Konto: 000 0000"
LPRINT "ABC-Bank BLZ 00000000000"
FOR i = 1 TO 12+xl
  LPRINT
NEXT i
RETURN

auftragsbestdruck:
CALL titel(datum$)
laenge$=STRING$(38,42)
GET#1,firma
GOSUB briefkopf
GOSUB briefkopf2
LPRINT "A U F T R A G S B E S T A E T I G U N G Nummer
** ";satz
GOSUB artikeldruck
LPRINT"Liefertermin: ";liefdat$
LPRINT:LPRINT
GET#9,VAL(B1$)
LPRINT "Lieferung: ";B9$
```

```
LPRINT " ";B8$
GET#9,VAL(B2$)
GOTO bestelldruck1
```

bestelldruck:

```
CALL titel(datum$)
laenge$=STRING$(19,42)
GET#2,firma
GOSUB briefkopf
GOSUB briefkopf1
LPRINT "B E S T E L L U N G Nummer ";satz
GOSUB artikeldruck
LPRINT"Liefertermin: ";liefdat$
LPRINT:LPRINT
GET#9,VAL(B3$)
LPRINT "Lieferung: ";B9$
LPRINT " ";B8$
GET#9,VAL(B4$)
REM 6 Zeilen
```

bestelldruck1:

```
LPRINT
LPRINT "Zahlung: ";B9$
LPRINT " ";B8$
FOR i = 1 TO 14+xl
  LPRINT
NEXT i
REM 18+xl Zeilen
RETURN
```

artikeldruck:

```
LPRINT laenge$
LPRINT:LPRINT:LPRINT
GET#5,artik
LPRINT "Menge Nummer Bezeichnung Stueck-Preis":LPRINT
LPRINT bedarf;" ";artik;" ";a5$
LPRINT TAB(52)"DM ";:LPRINT USING "####.##";preis!
FOR i=1 TO 4
  LPRINT
NEXT i
GOSUB datumrechnen
REM 11 Zeilen
RETURN
```

briefkopf:

```
GOSUB briefkopf3
GOTO briefkopf4
```

```
briefkopf3:
FOR i=1 TO 4
  LPRINT
NEXT i
LPRINT "T E S T F I R M A Grosshandel fuer H A U S H A L
** T S W A R E N"
LPRINT STRING$(60,45)
LPRINT TAB(50)"Nirgendwo 22"
LPRINT TAB(50)"8500 Nuernberg
LPRINT TAB(50)"Tel. 0911/000 000"
LPRINT
LPRINT TAB(50)"Datum: "datum$
LPRINT
RETURN

briefkopf4:
LPRINT "TESTFIRMA*Nirgendwo 22*8500 Nuernberg"
LPRINT STRING$(37,45)
LPRINT
LPRINT "Firma"
REM 16 Zeilen
RETURN

briefkopf1:
LPRINT N6$
LPRINT N7$
LPRINT N8$
LPRINT
LPRINT N9$;" ";N0$
FOR i=1 TO 10
  LPRINT
NEXT i
REM 15 Zeilen
RETURN

briefkopf2:
LPRINT N1$
LPRINT N2$
LPRINT N3$
LPRINT
LPRINT N4$;" ";N5$
FOR i=1 TO 10
  LPRINT
NEXT i
RETURN

arlidruck:
a=0:seite=1
arlidruck1:
GOSUB briefkopf3
```

```
LPRINT:LPRINT
LPRINT "A R T I K E L L I S T E Seite ";seite
LPRINT:LPRINT
listende=0
lilg=15
zei=1
WHILE zei
  a=a+1
  lilg=lilg+1
  GET#5,a
  IF Z5$=CHR$(0) THEN
       zei=0
       listende=1
  ELSE
       LPRINT USING "***#";a;
       LPRINT " ";L5$;a5$
  END IF
  IF lilg=45 THEN zei=0
WEND
  FOR i = 1 TO (66-lilg)+xl
       LPRINT
  NEXT i
IF listende THEN RETURN
seite=seite+1
GOTO arlidruck1
```

preislidruck:

```
a=0:seite=1
```

preislidruck1:

```
GOSUB briefkopf3
LPRINT:LPRINT
LPRINT "P R E I S L I S T E Seite ";seite
LPRINT:LPRINT
LPRINT tex2$
LPRINT tex3$
LPRINT
listende=0
lilg=20
zei=1
WHILE zei
  a=a+1
  lilg=lilg+1
  GET#8,a
  IF Z8$=CHR$(0) THEN
       zei=0
       listende=1
  ELSE
       LPRINT USING "#########";a;
```

```
        LPRINT USING "######.##";CVI(R4$),CVS(S4$),CVI(R5$),
        ** CVS(S5$),CVI(R6$),CVS(S6$)
   END IF
   IF lilg=45 THEN zei=0
WEND
   FOR i = 1 TO (66-lilg)+xl
        LPRINT
   NEXT i
IF listende THEN RETURN
seite=seite+1
GOTO preislidruck1

breakkontrolle:
RETURN
```

Erläuterungen zum Programm

Fehlersuche

Nachdem Sie nun, hoffentlich ohne gravierende Fehler, das Programm eingegeben haben, hier zunächst einige Tips, mit denen Sie kleinere Fehler relativ schnell beheben können. Zuerst legen Sie bitte eine neue Diskette mit dem Namen VEL-DATEN an, wie am Anfang des nächsten Abschnitts beschrieben. Setzen Sie dann in der Subroutine *ablauf* die BREAK-Anweisung außer Kraft, indem Sie ein Hochkomma davorsetzen. Wenn Sie es nicht mehr wissen, das Hochkomma befindet sich links von der RETURN-Taste und hat die gleiche Wirkung wie die REM-Anweisung. Wenn Sie die ON-BREAK-Anweisung nicht außer Kraft setzen, haben Sie keine Möglichkeit, das Programm abzubrechen, wenn es – zum Beispiel wegen eines winzigen Tippfehlers – etwas anderes unternimmt als Sie wollen.

Anschließend starten Sie das Programm. Es werden als erstes, wenn kein Fehler passiert, die benötigten Dateien angelegt. Wenn der Amiga währenddessen andere Disketten anfordert, folgen Sie den Anweisungen. Nach etwa einer Viertelstunde sind die Dateien angelegt. Wählen Sie nun aus dem Pull-down-Menü *Beenden* den Befehl *ENDE* aus. Basic meldet sich dann mit einem freundlichen *ok* zurück. Wie eben für die ON-BREAK-Anweisung beschrieben, setzen Sie nun die ON-ERROR-Anweisung, ganz am Anfang des Programmes, außer Kraft. Im Falle eines Fehlers erhalten Sie sonst nämlich nur die Fehlernummer angezeigt. Abgesehen davon, daß es doch recht umständlich ist, bei jedem Fehler im Basic-Handbuch nachschlagen zu müssen, wissen Sie dann auch nicht, an welcher Stelle des Programms der Fehler aufgetreten ist. Amiga-Basic hat eine so hervorragende Fehlerbehandlung, daß wir sie bei der Fehlersuche auch nutzen wollen.

Die nun folgende Fehlersuche basiert darauf, daß Sie den Amiga, beziehungsweise das Programm »VEL« arbeiten lassen. Dabei schreiben Sie sich am besten zu jedem Programmpunkt auf, was Sie eingegeben haben und rufen dann das oder die zugehörigen Auskunftsbilder auf. Wenn der Interpreter eine Fehlermeldung ausgibt, beheben Sie den

Fehler und starten Sie wieder von vorne. Auf diese Art und Weise gehen Sie jede Programmfunktion durch. In den Auskunftsbildern können Sie auch gleich überprüfen, ob die eingetragenen Daten dort erscheinen, wo sie hingehören. Wenn nicht, müssen Sie sich den zugehörigen Programmpunkt genau ansehen. Die anschließenden Erklärungen helfen Ihnen sicher dabei. Falsche Eingaben in die Dateien rufen nämlich nur selten eine Fehlermeldung hervor.

Wenn Sie alle Funktionen durchgeprüft haben, nehmen Sie sich die Drucker-Routinen vor. Eventuelle Eingabefehler in die Dateien können Sie gut an den ausgedruckten Listen erkennen. Wenn auch diese zur Zufriedenheit funktionieren, haben Sie es geschafft. Setzen Sie die beiden Befehle, die Sie vorher außer Kraft gesetzt haben, nun wieder in Betrieb, indem Sie jeweils das Hochkomma am Anfang der Zeile löschen.

Grundsätzliches zum Programm VEL

Kommen wir nun zurück zum eigentlichen Programm. Aus den Beispielen im Kapitel Dateien haben Sie erfahren, daß es für größere Programme meist nicht genügt, eine einzige Datei zu verwenden. Den gewünschten Erfolg bringt nur eine Kombination unterschiedlicher Dateien. Nicht zuletzt deshalb ist VEL ein **Datei**-Verwaltungsprogramm. Es arbeitet mit insgesamt zehn Dateien und macht nichts anderes, als diese zehn Dateien beziehungsweise die Daten darin zu verwalten. Bei den zehn Dateien handelt es sich um eine sequentielle und neun Direktzugriffsdateien. Daß ein Programm überhaupt mit so vielen Dateien arbeiten kann, liegt daran, daß nicht alle Dateien immer geöffnet sein müssen und daß es für die Direktzugriffsdateien immer nur eine Zugriffsart gibt. Mit Ausnahme der sequentiellen Datei sind während des Programmablaufes sämtliche Dateien geöffnet! Das hat allerdings auch einen kleinen Nachteil. Sind im Programm Fehler versteckt, können Daten verlorengehen. Aber die Fehlersuche sollten Sie ja schon hinter sich haben.

Erläuterungen zu den einzelnen Routinen von VEL

Nun aber zu den Programmteilen und Subroutinen von VEL im einzelnen. Zu jedem Teil und fast jeder Subroutine finden Sie auf dieser und den folgenden Seiten eine manchmal knappe, manchmal aber auch recht ausführliche Erläuterung. Wenn Sie in dem Programm nach einer bestimmten Routine suchen, hilft Ihnen die Grobeinteilung, die durch Kommentare und mit Sternchen gefüllte Zeilen hervorgehoben wird. Die einzelnen Programmblöcke nennen sich wie folgt: *allgemeine Routinen, Datei-Routinen, Menü-Routinen* und *Drucker-Routinen.*

Am Anfang des Programms wird zunächst mit CLEAR reichlich Platz für das Programm und seine Daten geschaffen. Nach den üblichen Vorbereitungen, wie DIM und DEF, wird die eine sequentielle Datei, die das Programm benötigt, zum Lesen geöffnet. Ist diese nicht vorhanden, wird in die Routine zur Fehlerbehandlung verzweigt (siehe unten).

start: Hier wird der Anwender aufgefordert, den Drucker in Startposition zu bringen. Gleichzeitig wird die Seitenlänge erfragt und gespeichert. Voreingestellt ist eine Seitenlänge von 72 Zeilen, entsprechend dem deutschen Standard. Wählbar ist ein Bereich von 66 bis 72 Zeilen. Bei anderen Eingaben wird zu *seitenlaenge* zurückverzweigt. Mit Hilfe der LOCATE-Anweisung liegt der Cursor beim INPUT

genau auf der ersten Ziffer von 72. Die Variable *xl* ist der Differenzwert zwischen der wirklichen Seitenlänge von 66 Zeilen. Sie kann also einen Wert von 0 bis 6 annehmen. Die Variable wird in den Druckroutinen benötigt und sorgt für einen gleichmäßig langen Druck der Blätter. Wenn Sie eine andere Papierlänge verwenden, als hier vorgesehen wurde, müssen Sie das Programm entsprechend abändern.

ablauf: Diese Routine gehört wieder zum Kernprogramm. Die MENU- und BREAK-Anweisungen werden hier aktiviert. Solange der Wert von *aus* wahr ist, bleibt das Programm innerhalb der WHILE-Schleife. Es wird nur durch die vorher aktivierten MENU- und BREAK-Anweisungen unterbrochen. Wird der Wert von *aus* unwahr (gleich Null), so verläßt das Programm die Schleife und kommt zur Routine *ende*.

datum: Den ersten Teil dieses Programmteils kennen Sie schon aus dem Programm »Terminkalender«. Hier ist aber die korrekte Eingabe des Datums extrem wichtig. Sämtliche wichtigen Funktionen des Programms stützen sich auf das eingegebene Datum! Gerade dann, wenn Sie das Programm ernsthaft einsetzen wollen, sollten Sie die möglichen Werte für die Jahresangabe deshalb weiter einschränken. Ändern Sie also die »Reichweite« der Anweisung *IF jahr>99 OR jahr<86 THEN datum1* jeweils nach Ihren Bedürfnissen. Die letzten drei Zeilen dieser Routine legen das übergebene Datum beziehungsweise Teile davon in verschiedene Variablen. Diese drei Variablen werden in fast allen anderen Programm-Routinen benötigt.

grundzahl: Dieser kurze Abschnitt enthält die Werte für die Berechnung des Kalendertages (ktag), während *datumrechnen* die *grundzahl* benötigt, um ein lesbares Datum zu erzeugen. Der Kalendertag ist nicht nur als handlicher Wert zur Identifizierung eines Datums sehr praktisch, sondern kann auch gut für Datumsarithmetik benutzt werden. Die Differenz (in Tagen oder Wochen) zwischen zwei Kalendertagen läßt sich nämlich viel einfacher berechnen als die zwischen zwei Datumsangaben (zum Beispiel 1.4.1987 minus 12.1.1987).

titel: Diese Subroutine kam in ähnlicher Form schon im Programm Terminkalender vor. Hier wird ihr allerdings noch das aktuelle Datum übergeben, das sie mit am Bildschirm ausgibt.

fehler: Zu dieser Routine wird beim Auftreten eines Fehlers verzweigt. Wenn die vom Interpreter gemeldete Fehlernummer nicht gleich 53 ist, wird das Programm beendet und die Nummer des Fehlers angezeigt. Hat die Fehlermeldung den Code 53 (*File not found;* oder auf deutsch *Datei nicht gefunden*), wird zu *anlegen* verzweigt. Der Fehler wird also mißbraucht, um festzustellen, ob die Datei angelegt werden muß oder schon vorhanden ist.

taste1: Wartet auf eine Tastatur-Eingabe. Erst wenn diese zwischen CHR$(129) und CHR$(131) liegt, wird zurückgesprungen. 129, 130 und 131 sind die Zeichen-Codes für die Funktionstasten F1, F2 und F3. (Dies wurde im Kapitel TIPS und TRICKS erläutert.) Diese drei Funktionstasten werden innerhalb der einzelnen Routinen zum Abbrechen, Speichern, Weitermachen und Ausdrucken eingesetzt.

anlegen: Dem Anwender wird zuerst die Möglichkeit gegeben, den Vorgang abzubrechen. Falls er sich dafür entscheidet, das Programm fortfahren zu lassen, wird zuerst die sequentielle Datei »test« angelegt. Ihr Inhalt ist dabei zunächst unwichtig. Wichtig ist nur, daß sie vorhanden ist. Sie zeigt nämlich später dem Programm an, daß alle nachfolgenden Initialisierungs-Operationen (vor allem das Anlegen der Dateien für den direkten Zugriff) bereits erledigt sind. Dafür wird deshalb eine sequentielle Datei benutzt, weil eine Direktzugriffs-Datei beim Öffnen einfach angelegt wird, falls sie noch nicht vorhanden ist – was natürlich keine Fehlermeldung ergibt. Die Fehlermeldung wird aber benötigt, um das automatische Anlegen der Dateien programmieren zu können.

aktivieren: Diese Routine ist praktisch ein Unterpunkt von *anlegen.* Hier werden alle Datensätze der neun Direktzugriffs-Dateien mit einer Startmarkierung CHR$(0) versehen, die besagt, daß diese Datensätze leer sind. Da dieser Vorgang geraume Zeit in Anspruch nimmt, wird sein Fortschreiten auf dem Bildschirm angezeigt.

oeffnen: In dieser Routine werden die neun Dateien für Ein- und Ausgabe im direkten Zugriff geöffnet und die Datensätze mit der FIELD-Anweisung in einzelne Felder zerlegt.

menuEin: VEL arbeitet mit Pull-down-Menüs. Wer sich bei der täglichen Arbeit mit professioneller Software laufend durch einen Wust von Unter- und Hauptmenüs durchkämpfen muß, weiß diese Möglichkeit von Amiga-Basic zu schätzen. Mit 10 verschiedenen Menüs mit 19 Menü-Punkten lassen sich bis zu 190 Programm-Funktionen recht übersichtlich anordnen! VEL begnügt sich aber mit fünf Menü-Titeln und insgesamt 24 Menü-Punkten.

menuKontrolle: Diese Routine wird über die ON-MENU-Anweisung angesprungen und verzweigt zu den Routinen, die den einzelnen Menü-Titeln entsprechen: *verkauf, einkauf, lager, auskunft* und *beenden.* Diese folgen nun auch in genau dieser Reihenfolge. Zwischen ihnen befinden sich allerdings noch die zugehörigen Unterroutinen.

kundenNeu: Diese Routine schreibt neue Kunden in die Datei »Kunden«. In einer WHILE-Schleife wird zunächst nach dem ersten freien Datensatz in der Datei gesucht. Freie Datensätze sind mit CHR$(0) am Anfang markiert (siehe oben). Die einzelnen Daten des Kunden werden dann mit INPUT erfragt und in den entsprechenden Variablen abgelegt. Bevor die Daten in die Datei übernommen werden, erhält der Benutzer noch einmal die Möglichkeit, den Vorgang abzubrechen. Anschließend werden die Daten abgespeichert und der Datensatz wird mit CHR$(1) als belegt markiert. Das Programm springt zur weiteren Eingabe, nach *kundenNeu1.* Eine Änderung der Kunden-Datensätze ist im Programm nicht vorgesehen. Sollten Sie dies wünschen, können Sie diesen Programmpunkt bestimmt auch selbst erstellen.

auftragneu: Das Programm sucht zunächst wieder, wie eben beschrieben, einen freien Datensatz und verwendet dessen Datensatznummer dann als Auftragsnummer. Der Anwender muß die Kundennummer, die Artikelnummer und die bestellte Menge eingeben. Aus der Kunden-Datei (#1) wird dann die Kundenanschrift und aus der Artikel-Datei (#5), die Artikel-Bezeichnung geholt. In einer IF-Anweisung wird aus den Daten der Preisliste (#8) der Preis ermittelt, der der bestellten Menge entspricht. Wird die Mindestmenge unterschritten, wird diese als neue Mengenangabe verwendet. Aus der

Artikel-Datei (#5) wird die Wiederbeschaffungszeit entnommen und mit der eingegebenen Lieferzeit verglichen. Ist kein Artikel auf Lager und die vorgegebene Lieferzeit niedriger, wird die Wiederbeschaffungszeit als Lieferzeit verwendet. Der Artikel-Bedarf wird in der Bestands-Datei (#6) dann vorgemerkt. Schließlich wird der Auftrag in der Auftrags-Datei (#3) eingetragen. Das hört sich jetzt vielleicht recht kompliziert an, ist aber nichts weiter als die programmtechnische Umsetzung der üblichen Vorgänge, die sich ohne Verwendung eines Computers bei einer Bestellung abspielen. Am Ende dieses Abschnitts finden Sie für diese Routine ein Ablaufschema, das den Vorgang sicher verständlicher macht.

preisliste: Mit dieser Routine kann man entweder neue Preisinformationen anlegen oder vorhandene ändern. Um dies zu ermöglichen, wird anhand der eingegebenen Artikel-Nummer zunächst geprüft, ob der entsprechende Datensatz schon belegt ist. Ist das erste Byte noch gleich CHR$(0), werden alle Felder des Datensatzes mit Null belegt und abgespeichert. Dann wird weiterverfahren wie bei der Änderung eines vorhandenen Eintrags.

Bei der Eingabe werden dann die alten Feldwerte vorgeschlagen und können durch Betätigung von RETURN unverändert übernommen werden. Zum Schluß werden die Daten dann in der Preisliste (Datei #8) abgespeichert.

auftragsabwicklung: Diese Routine kann nicht über einen Menü-Punkt angewählt werden. Sie gehört zu einer Reihe von Subroutinen, die das Programm automatisch aufruft. Ihr genauer Ablauf wird in den Programmlaufplänen am Ende dieses Abschnitts dargestellt. Die einzelnen Zeilen sind im Listing kommentiert.

rechnung: Diese Routine ist eine Unterroutine von *auftragsabwicklung*. In der Bestands-Datei werden die Ist-Menge und die Vormerkungen um die bestellte Menge reduziert. Für den Rechnungs-Druck wird der Preis des Artikels und die Kunden-Anschrift aus der Auftrags-Datei gelesen.

angebote: Diese Routine entspricht vom Aufbau und Programmablauf her der Routine *preisliste*.

bestandsmeldung: Dieser Programmpunkt dient als Verteiler in die beiden Routinen *bestellungsabwicklung* und *auftragsabwicklung* und ist deshalb in zwei der am Ende dieses Abschnitts folgenden Programmlaufplänen zu finden.

bestellungsabwicklung: Da sämtliche Bestellungen automatisch vom Programm geschrieben werden sollen, wird hier überprüft, ob Bestellungen nötig sind. In einer WHILE-Schleife wird die gesamte Bestands-Datei daraufhin überprüft, ob der Bestand niedriger ist als die Mindestlagermenge. Wenn ja, wird eine Bestellung ausgelöst. Die Bestellmenge richtet sich nach der maximalen Lagermenge. Es werden so lange Bestellungen angefordert, bis in der Bestands-Datei ein unbelegter Datensatz (mit CHR(0) markiert) gefunden wird.

bestellung: Zuerst wird in einer WHILE-Schleife der nächste freie Datensatz in der Bestell-Datei (#4) ermittelt. In zwei in sich geschachtelten IF-Anweisungen wird aus den Daten der Angebote (#7) der Preis für die Bestellung ermittelt, der der Menge entspricht, welche von *bestellungsabwicklung* angefordert wurde. Wird dabei die Mindestbe-

stellmenge unterschritten, wird diese als zu bestellende Menge verwendet. Aus der Artikel-Datei (#5) wird dann die Wiederbeschaffungszeit geholt, um eine Woche für die Durchgangszeiten reduziert und zum Kalendertag hinzugerechnet. Die Bestellmenge wird in der Bestands-Datei (#6) festgehalten. Schließlich wird die Bestellung in die Bestell-Datei (#4) eingetragen.

lieferantenNeu: Dieser Programmabschnitt ist genauso wie *kundenNeu* aufgebaut und kann dort nachgelesen werden.

liebedneu, zahlbedneu und **artikel** sind Routinen, in denen längere Texte eingegeben werden. Alle drei verwenden die Eingabemaske *maske1*. Unterhalb der Eingabezeile werden mit Hilfe der STRING$-Funktion Punkte in der maximalen Textlänge auf dem Bildschirm angezeigt. (Hierzu wird aber nicht der normale Punkt CHR$(46), sondern zwei nebeneinander liegende Punkte CHR$(168) verwendet.) Bei der Eingabe erkennt der Anwender so stets, wie lang der Text werden darf. Bei zu langen Eingaben wird nur der vordere Teil des Textes, bis zum letzten Punkt, gespeichert.

bestand: Der Aufbau dieses Programmteils entspricht weitgehend dem von *angebot* und *preisliste*. Anders als in diesen, springt das Programm beim Betätigen der Funktionstaste1 aber nicht zurück, sondern verzweigt zu *bestandsmeldung*. Damit werden in diesem Programmabschnitt weitere Aktionen ausgelöst, die Sie aus dem Programmlaufplan im Anschluß an diesen Abschnitt ersehen können.

wareneingang: Diese Routine gehört zu den wichtigsten des ganzen Programms. Sie finden deshalb auch einen entsprechenden Programmlaufplan im Anschluß an diesen Abschnitt. Hier wird zunächst die Bestellnummer und die gelieferte Menge vom Benutzer erfragt. In der Bestell-Datei (#4) wird der entsprechende Datensatz dann mit CHR$(2) an erster Stelle markiert, um diese Bestellung als geliefert zu markieren, und das Lieferdatum gespeichert. Außerdem wird die Artikelnummer des gelieferten Artikels festgestellt und mit dieser dann der entsprechende Datensatz aus der Artikel-Datei geholt. In diesem Datensatz wird der Bestand nun um die gelieferte Menge erhöht und die noch benötigte Menge um die gelieferte Menge reduziert.

kunden: Diese Routine dient dazu, bestimmte Kunden in der entsprechenden Datei zu finden und Informationen über sie auf den Bildschirm zu bringen. Die ersten vier Buchstaben aus dem Namen des gesuchten Kunden werden dazu zunächst der Textvariablen such$ zugeordnet. In einer WHILE-Schleife werden dann Datensätze gesucht, in denen die ersten vier Buchstaben des Namens-Feldes mit such$ übereinstimmen. Bei Übereinstimmung erfolgt die Ausgabe der kompletten Anschrift auf dem Bildschirm. Das Programm sucht dann weiter, ob nicht noch andere Kunden mit den gleichen Anfangsbuchstaben zu finden sind.

Stellvertretend für die anderen Auskunftsbilder wird das der Bestellungen erklärt, da der Programmaufbau bei allen ähnlich ausgelegt ist. artikelListe: und preislistezei: können zusätzlich eine Druck-Routine aufrufen.

rueckstand, bestellungzei und **bestellungzeioffen** sind alle recht ähnlich und listen bestimmte Datensätze aus der Bestellungsdatei auf dem Bildschirm auf. (Ähnliche Routinen gibt es natürlich auch für die anderen Dateien.) Die drei Routinen nutzen deshalb auch

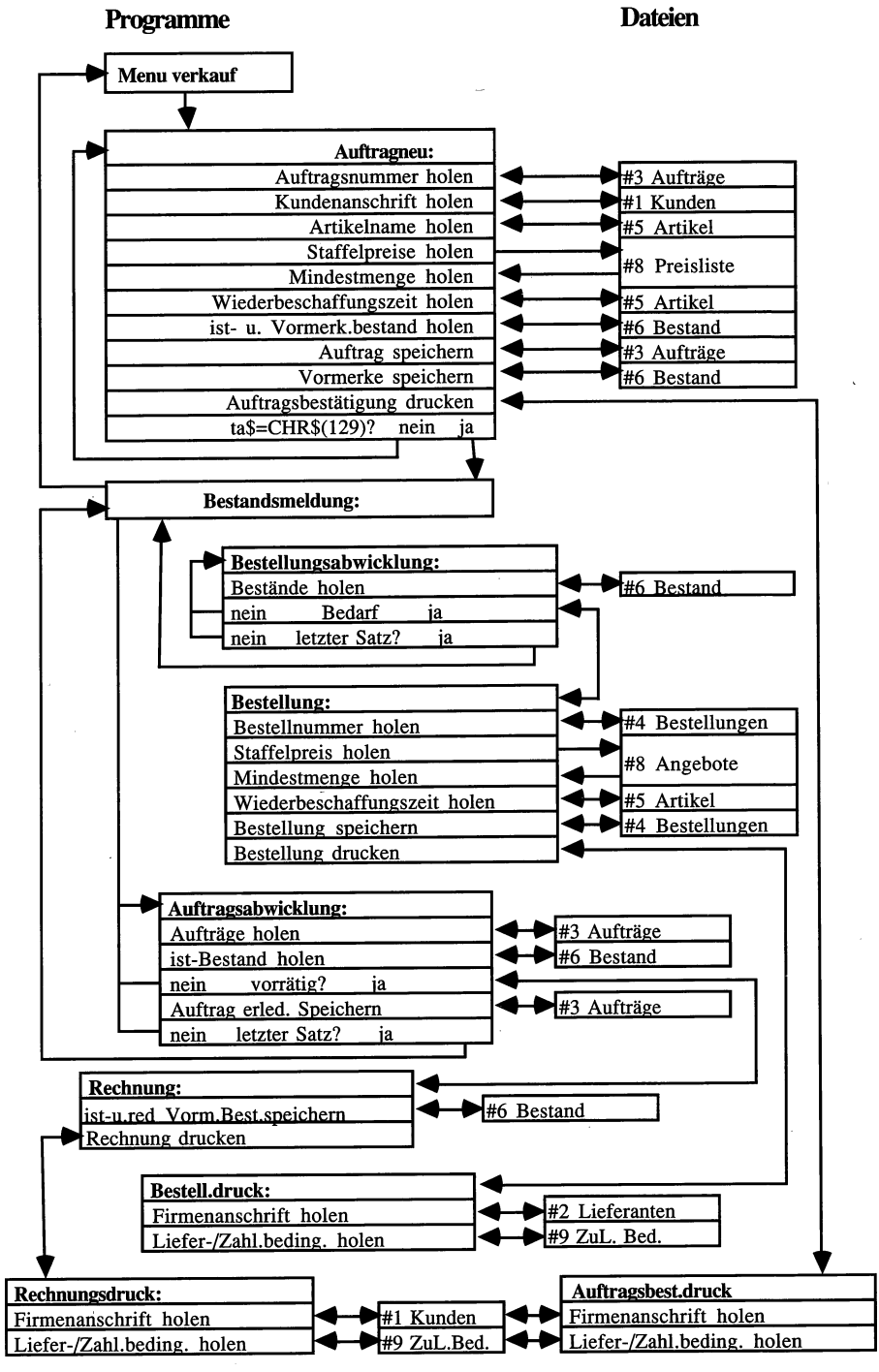
große Programmabschnitte gemeinsam, wobei diesen Programmabschnitten in bestimmten Variablen mitgeteilt wird, nach welchen Datensätzen gerade gesucht werden soll (alle Bestellungen, noch offene Bestellungen oder überfällige Bestellungen). Eine WHILE-Schleife dient wieder dazu, die gesamte Bestell-Datei nach entsprechenden Datensätzen zu durchsuchen. Jedesmal, wenn einer gefunden wurde, wird er auf dem Bildschirm ausgegeben. Dabei zählt das Programm in der Variablen *a* mit, wieviele Datensätze bereits ausgegeben wurden und macht nach 15 eine Pause. Danach hat der Benutzer Gelegenheit, sich die Liste am Bildschirm zu betrachten. Weitere Datensätze werden erst dann ausgegeben, wenn er eine bestimmte Taste drückt.

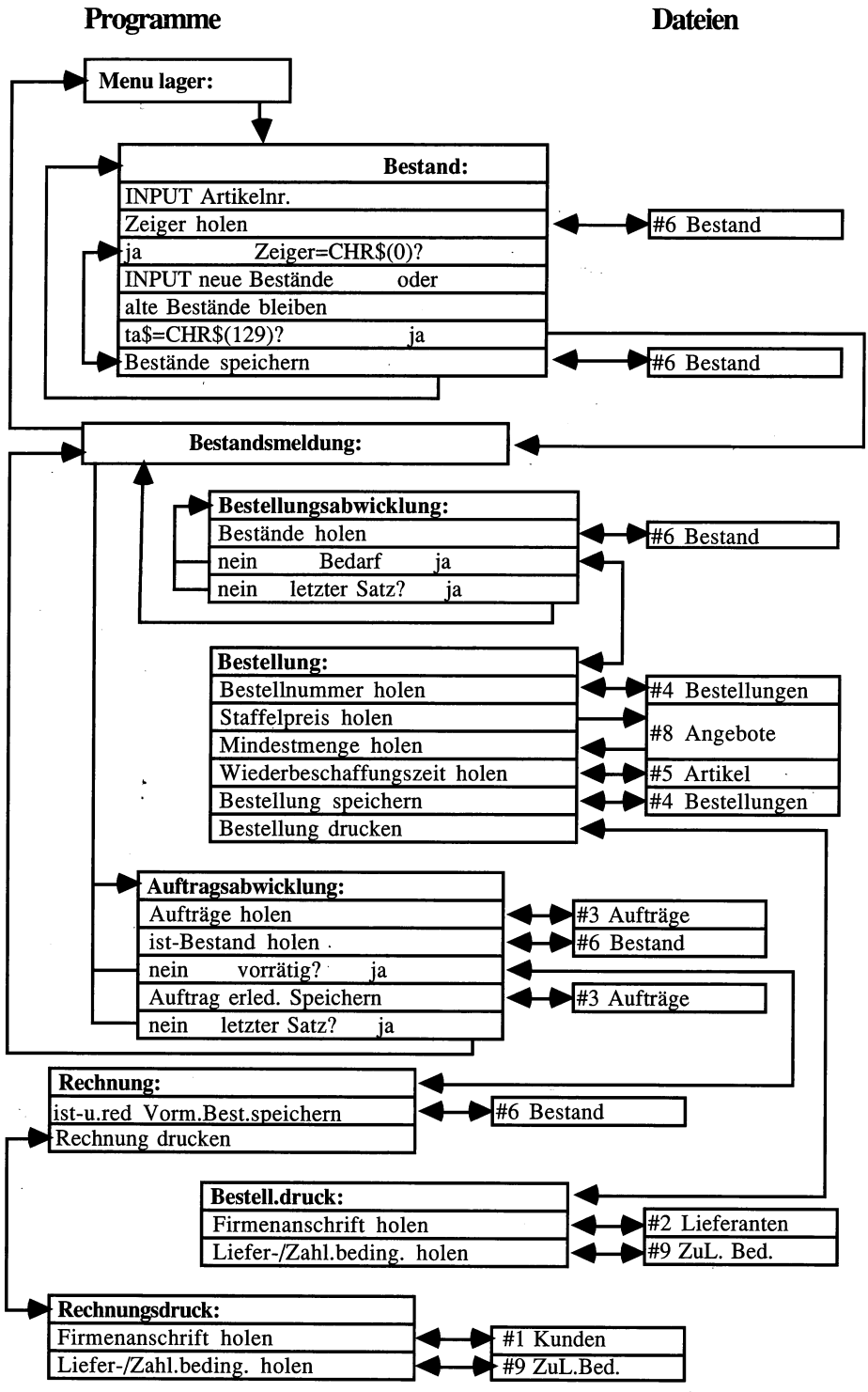
Nun folgen die einzelnen Drucker-Routinen. Diese suchen sich ihre Daten ebenfalls aus den entsprechenden Dateien, oder es werden ihnen die Variablen von dem Programm mitgegeben, welches den Druck angefordert hat.

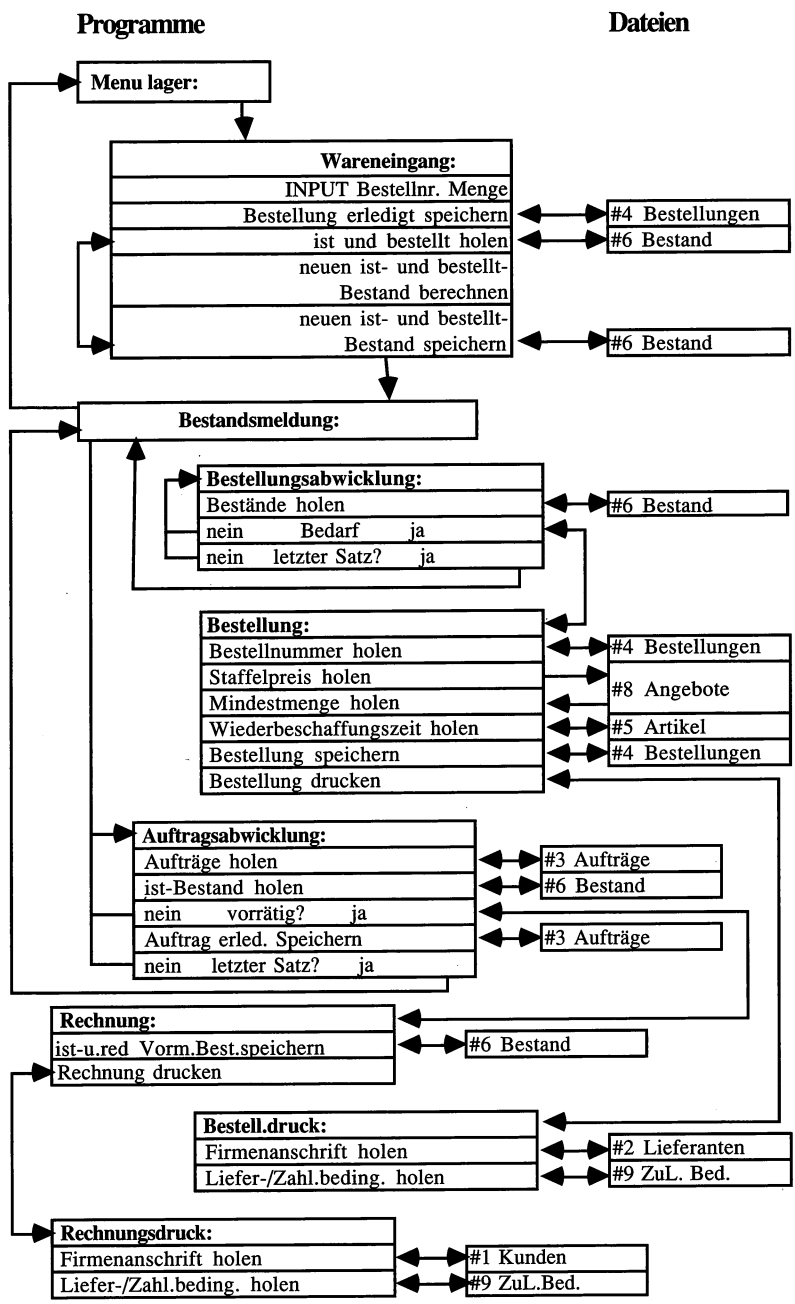
briefkopf: Diese Unterroutine kommt für alle gedruckten Dokumente zur Anwendung. Ändern Sie sie bitte nach Ihren eigenen Vorstellungen ab – falls sie nicht »TEST-FIRMA« heißen. Natürlich können Sie auch die Blattaufteilung verändern. Achten Sie dabei darauf, daß die Anzahl der LPRINT-Anweisungen gleichbleibt, da folgende Druckausgaben sonst nicht mehr an der richtigen Stelle innerhalb der Seite beginnen.

Die beiden Listen **arlidruck** und **preislidruck** sind so aufgebaut, daß sie nach 25 beziehungsweise 30 Datensätzen eine neue Seite anfangen. Die Variable *seite* enthält dabei die Nummer der aktuellen Seite, die oben auf jedem neuen Blatt ausgedruckt wird.

breakkontrolle: Bei einem BREAK springt das Programm an diese Stelle. Da hier aber nur RETURN für den Rücksprung steht, passiert dabei überhaupt nichts. Im Klartext bedeutet dies zum Beispiel, daß das Programm nicht von außen unterbrochen werden kann.


Programme
Dateien
Menu verkauf
Auftragneu:
Auftragsnummer holen
Kundenanschrift holen
Artikelname holen
Staffelpreise holen
Mindestmenge holen
Wiederbeschaffungszeit holen
ist- u. Vormerk.bestand holen
Auftrag speichern
Vormerke speichern
Auftragsbestätigung drucken
ta$=CHR$(129)? nein ja
#3 Aufträge
#1 Kunden
#5 Artikel
#8 Preisliste
#5 Artikel
#6 Bestand
#3 Aufträge
#6 Bestand
Bestandsmeldung:
Bestellungsabwicklung:
Bestände holen
nein Bedarf ja
nein letzter Satz? ja
#6 Bestand
Bestellung:
Bestellnummer holen
Staffelpreis holen
Mindestmenge holen
Wiederbeschaffungszeit holen
Bestellung speichern
Bestellung drucken
#4 Bestellungen
#8 Angebote
#5 Artikel
#4 Bestellungen
Auftragsabwicklung:
Aufträge holen
ist-Bestand holen
nein vorrätig? ja
Auftrag erled. Speichern
nein letzter Satz? ja
#3 Aufträge
#6 Bestand
#3 Aufträge
Rechnung:
ist-u.red Vorm.Best.speichern
Rechnung drucken
#6 Bestand
Bestell.druck:
Firmenanschrift holen
Liefer-/Zahl.beding. holen
#2 Lieferanten
#9 ZuL. Bed.
Rechnungsdruck:
Firmenanschrift holen
Liefer-/Zahl.beding. holen
#1 Kunden
#9 ZuL.Bed.
Auftragsbest.druck
Firmenanschrift holen
Liefer-/Zahl.beding. holen


Programme
Dateien
Menu lager:
Bestand:
INPUT Artikelnr.
Zeiger holen
#6 Bestand
ja Zeiger=CHR$(0)?
INPUT neue Bestände oder
alte Bestände bleiben
ta$=CHR$(129)? ja
Bestände speichern
#6 Bestand
Bestandsmeldung:
Bestellungsabwicklung:
Bestände holen
#6 Bestand
nein Bedarf ja
nein letzter Satz? ja
Bestellung:
Bestellnummer holen
#4 Bestellungen
Staffelpreis holen
#8 Angebote
Mindestmenge holen
Wiederbeschaffungszeit holen
#5 Artikel
Bestellung speichern
#4 Bestellungen
Bestellung drucken
Auftragsabwicklung:
Aufträge holen
#3 Aufträge
ist-Bestand holen
#6 Bestand
nein vorrätig? ja
Auftrag erled. Speichern
#3 Aufträge
nein letzter Satz? ja
Rechnung:
ist-u.red Vorm.Best.speichern
#6 Bestand
Rechnung drucken
Bestell.druck:
Firmenanschrift holen
#2 Lieferanten
Liefer-/Zahl.beding. holen
#9 ZuL. Bed.
Rechnungsdruck:
Firmenanschrift holen
#1 Kunden
Liefer-/Zahl.beding. holen
#9 ZuL.Bed.


Programme
Dateien
Menu lager:
Wareneingang:
INPUT Bestellnr. Menge
Bestellung erledigt speichern
ist und bestellt holen
neuen ist- und bestellt-Bestand berechnen
neuen ist- und bestellt-Bestand speichern
#4 Bestellungen
#6 Bestand
#6 Bestand
Bestandsmeldung:
Bestellungsabwicklung:
Bestände holen
nein Bedarf ja
nein letzter Satz? ja
#6 Bestand
Bestellung:
Bestellnummer holen
Staffelpreis holen
Mindestmenge holen
Wiederbeschaffungszeit holen
Bestellung speichern
Bestellung drucken
#4 Bestellungen
#8 Angebote
#5 Artikel
#4 Bestellungen
Auftragsabwicklung:
Aufträge holen
ist-Bestand holen
nein vorrätig? ja
Auftrag erled. Speichern
nein letzter Satz? ja
#3 Aufträge
#6 Bestand
#3 Aufträge
Rechnung:
ist-u.red Vorm.Best.speichern
Rechnung drucken
#6 Bestand
Bestell.druck:
Firmenanschrift holen
Liefer-/Zahl.beding. holen
#2 Lieferanten
#9 ZuL. Bed.
Rechnungsdruck:
Firmenanschrift holen
Liefer-/Zahl.beding. holen
#1 Kunden
#9 ZuL.Bed.

VEL-Handbuch

Vorbereitungen für die Inbetriebnahme

Das Programm VEL erwartet, sämtliche Dateien, mit denen es arbeitet, auf einer eigenen Diskette vorzufinden. Bevor Sie sich näher mit dem Programm beschäftigen, müssen Sie deshalb diese Daten-Diskette anlegen. Formatieren Sie dazu (auf der Workbench) eine neue oder nicht mehr benötigte Diskette mit dem Befehl *Initialize* aus dem *Workbench*-Menü. Klicken Sie dann das Icon der frisch initialisierten Diskette an und geben Sie ihr (mit *Rename* aus dem *Workbench*-Menü) den Namen »VEL-DATEN«. (Achten Sie darauf, daß Sie nicht aus Versehen vor, nach oder in dem Namen eine Leerstelle eingeben; er muß wirklich exakt »VEL-DATEN« lauten.) Kopieren Sie dann den Basic-Interpreter und das Programm »VEL« auf diese Diskette. Sie brauchen dann nur noch diese Diskette, um mit dem Programm zu arbeiten (und natürlich die Kickstart- und Workbench-Diskette zu *Start*).

Wenn Sie sich so eine Arbeitsdiskette geschaffen haben, stehen Sie als nächstes vor der Frage, wieviele Datensätze Sie maximal in jeder der neun Direktzugriffsdateien benötigen. Eine Wahl, die Sie einmal getroffen haben, können Sie nicht wieder zurücknehmen. Es sei denn, es macht Ihnen Freude, alle Daten nochmals einzugeben (soweit dann noch greifbar). Bevor Sie diese Entscheidung aber treffen können, müssen Sie zunächst einmal wissen, welche Dateien mit welchen Feldern VEL verwendet. Die folgende Aufstellung listet deshalb alle Dateien mit ihren Feldern und dem jeweils dafür benötigten Speicherbedarf auf.

Datei #1: Kunden		**100 Sätze zu 84 = 8400 Byte**	
Kundennummer = Satznummer			
Feld	**Bedeutung**	**Typ**	**Größe**
Z1$	Zeiger	Zeichen	1 Byte
N1$	Kundenname	Zeichenkette	15 Byte
N2$	Kundenname	Zeichenkette	15 Byte
N3$	Straße	Zeichenkette	25 Byte
N4$	Postleitzahl	Zeichenkette	4 Byte
N5$	Ort	Zeichenkette	20 Byte
B1$	Nr.Lieferbedingungen	Zeichenkette	2 Byte
B2$	Nr.Zahlungsbedingungen	Zeichenkette	2 Byte

Datei #2 Lieferanten		**100 Sätze zu 84 = 8400 Byte**	
Lieferantennum. = Satznummer			
Feld	**Bedeutung**	**Typ**	**Größe**
Z2$	Zeiger	Zeichen	1 Byte
N6$	Lieferantenname	Zeichenkette	15 Byte
N7$	Lieferantenname	Zeichenkette	15 Byte
N8$	Straße	Zeichenkette	25 Byte
N9$	Postleitzahl	Zeichenkette	4 Byte
N0$	Ort	Zeichenkette	20 Byte
B3$	Nr.Lieferbedingungen	Zeichenkette	2 Byte
B4$	Nr.Zahlungsbedingungen	Zeichenkette	2 Byte

Datei #3 Auftraege		**1000 Sätze zu 17 =17000 Byte**	
Auftragsnummer = Satznummer			
Feld	**Bedeutung**	**Typ**	**Größe**
Z3$	Zeiger	Zeichen	1 Byte
D3$	Datum	kurze Ganzzahl	2 Byte
V3$	Kundennr.	kurze Ganzzahl	2 Byte
M3$	Menge	kurze Ganzzahl	2 Byte
A3$	Artikelnummer	kurze Ganzzahl	2 Byte
P3$	Preis	einfache Genauigkeit	4 Byte
L3$	Liefertermin	kurze Ganzzahl	2 Byte
G3$	geliefert	kurze Ganzzahl	2 Byte

Datei #4 Bestellungen		**1000 Sätze zu 17 =17000 Byte**	
Bestellnummer = Satznummer			
Feld	**Bedeutung**	**Typ**	**Größe**
Z4$	Zeige	Zeichen	1 Byte
D4$	Datum	kurze Ganzzahl	2 Byte
E4$	Lieferantennr.	kurze Ganzzahl	2 Byte
M4$	Menge	kurze Ganzzahl	2 Byte
A4$	Artikelnummer	kurze Ganzzahl	2 Byte

P4$	Preis	einfache Genauigkeit	4 Byte
L4$	Liefertermin	kurze Ganzzahl	2 Byte
G4$	geliefert	kurze Ganzzahl	2 Byte

Datei #5 Artikel *		**1000 Sätze zu 40 = 40000 Byte**	
Artikelnummer = Satznummer			
Feld	**Bedeutung**	**Typ**	**Größe**
Z5$	Zeiger	Zeichen	1 Byte
A5$	Artikelname	Zeichenkette	37 Byte

Datei #6 Bestand * **Bestandsnummer = Satznummer**		**1000 Sätze zu 11 = 11000 Byte**	
Feld	**Bedeutung**	**Typ**	**Größe**
Z6$	Zeiger	Zeichen	1 Byte
W1$	Ist-Bestand	kurze Ganzzahl	2 Byte
W2$	Untergrenze	kurze Ganzzahl	2 Byte
W3$	Obergrenze	kurze Ganzzahl	2 Byte
W4$	Vormerkungen	kurze Ganzzahl	2 Byte
W5$	bestellt	kurze Ganzzahl	2 Byte

Datei #7 Angebote *		**1000 Sätze zu 23 =23000 Byte**	
Angebotsnummer =		**Satznummer**	
Z7$	Zeiger	Zeichen	1 Byte
E7$	Lieferantennr.	kurze Ganzzahl	2 Byte
M7$	Mindestmenge	kurze Ganzzahl	2 Byte
R1$	Menge 1	kurze Ganzzahl	2 Byte
S1$	Preis 1	doppelte Genauigk.	4 Byte
R2$	Menge 2	kurze Ganzzahl	2 Byte
S2$	Preis 2	doppelte Genauigk.	4 Byte
R3$	Menge 3	kurze Ganzzahl	2 Byte
S3$	Preis 3	doppelte Genauigk.	4 Byte

Datei #8 Preisliste *		**1000 Sätze zu 19 = 19000 Byte**	
Listennummer = Satznummer			
Z8$	Zeiger	Zeichen	1 Byte
R4$	Menge 1	kurze Ganzzahl	2 Byte
S4$	Preis 1	doppelte Genauigk.	4 Byte
R5$	Menge 2	kurze Ganzzahl	2 Byte
S5$	Preis 2	doppelte Genauigk.	4 Byte
R6$	Menge 3	kurze Ganzzahl	2 Byte
S6$	Preis 3	doppelte Genauigk.	4 Byte

Datei #9 ZuL.Bed		**100 Sätze zu 61 = 6100 Byte**	
Lieferbedingungsnummer =		**Satznummer (1-49)**	
Zahlungsbedingungsnummer =		**Satznummer (50-99)**	
Feld	**Bedeutung**	**Typ**	**Größe**
Z9$	Zeiger	Zeichen	1 Byte
B9$	1Zeile	Zeichenkette	30 Byte
B8$	2.Zeile	Zeichenkette	30 Byte

Zählen Sie die für die einzelnen Dateien benötigten Datensätze und deren jeweiligen Platzbedarf zusammen, und Sie kommen auf einen Gesamtplatzbedarf von 149 000 Byte für VEL. Wenn man berücksichtigt, daß auf einer frisch initialisierten Diskette etwa 896 000 Byte frei sind und das Betriebssystem davon (für Icons und so weiter) etwa 30 000 Byte benötigt, bleiben noch 719 000 Byte übrig. Sie können also noch einiges mehr an Datensätzen unterbringen, als im abgedruckten Programm vorgesehen ist. Bitte beachten Sie dabei aber, daß die Dateien, die in der Übersicht mit einem Sternchen versehen sind, zusammengehören und die gleiche Anzahl von Datensätzen haben müssen. Wenn Sie die Anzahl der Datensätze erhöhen, müssen Sie in der Routine *aktivieren* die obere Grenze der FOR-Schleifen ebenfalls erhöhen. Die Zeit für die Aktivierung erhöht sich ebenfalls entsprechend. Da dies aber nur ein einmaliger Vorgang ist, kann man den Zeitaufwand verkraften.

Bevor Sie das Programm für ernsthafte Aufgaben einsetzen, sollten Sie unbedingt einen größeren Testlauf durchführen und dabei vor allem penibel überprüfen, ob die Warenbuchungen ordnungsgemäß vonstatten gehen. Starten Sie dann das Programm. Nun beginnt die Markierung der einzelnen Datensätze. Diese Prozedur dauert (bei unveränderter Anzahl von Datensätzen in den einzelnen Dateien) etwa 15 Minuten.

Dann meldet sich das Programm wieder und verlangt von Ihnen die Eingabe des Tagesdatums. Bitte geben Sie hier das richtige Datum ein, da sich sämtliche Berechnungen der Lieferzeiten und so weiter auf diese Eingabe stützen. Verlangt werden jeweils zweistellige Eingaben für Tag, Monat und Jahr (TTMMJJ). (Gegen irrtümlich falsche Ein-

gaben ist das Programm weitestgehend gesichert.) Anschließend werden Sie aufgefordert, das erste Blatt Papier im Drucker in Startposition zu bringen, und Sie müssen die Papierlänge eingeben. Arbeiten Sie mit dem normalen Endlospapier, brauchen Sie nur RETURN zu drücken (dessen Länge von 72 Zeilen ist voreingestellt). Nach einem kurzen Moment leert sich dann der Bildschirm und das Programm harrt auf Ihre Aktionen.

Gehen Sie nun mit der Maus auf die Fensterleiste und schauen Sie sich die einzelnen Menüpunkte in Ruhe an. Es gibt fünf verschiedene Menüs, die den einzelnen Aufgabengebieten des Programms entsprechen. Das erste Aufgabengebiet ist der Verkauf. Hier können Sie neue Kunden anlegen, Aufträge eingeben und die Preisliste erstellen oder ändern. Der zweite Programmbereich stellt Funktionen für den Einkauf bereit. Hier werden Angebote verwaltet, Lieferanten gespeichert und Liefer- und Zahlungsbedingungen festgelegt. Die dritte Rubrik faßt die Funktionen zur Lagerhaltung zusammen. Diese Funktionen sind zuständig für die Erstellung der Artikel-Datei, die Verbuchung von Wareneingängen und die Bestandspflege. Das nächste Menü enthält Funktionen für die Ausgabe von Listen auf dem Bildschirm oder dem Drucker. Sie können sich hiermit zum Beispiel Kundenadressen, Bestellungen und Lagerbestände zeigen lassen. Das letzte Menü *beenden* schließlich beinhaltet nur einen Menüpunkt, mit dem Sie das Programm jederzeit ordnungsgemäß verlassen können. (Dies ist – abgesehen von einem Fehler – auch die einzige Möglichkeit, das Programm zu verlassen, da eine Unterbrechung des Programms von außen durch die ON-BREAK-Anweisung ausgeschlossen wurde.)

Kontrolle und Eingabe der Stammdaten

Wenn Sie eine Weile mit dem Programm gearbeitet haben, sollten Sie zunächst einmal die verschiedenen Listen anschauen, die die Befehle im Menü AUSKUNFT produzieren. Anhand dieser Listen können Sie überprüfen, ob Ihre Eingaben ordnungsgemäß abgespeichert wurden. Stellen Sie hier eine Differenz zwischen einer Liste und Ihren Eingaben fest, liegt wahrscheinlich ein Programmfehler vor. Sie sollten ihn beheben und erst dann weitermachen. Der Inhalt der Daten-Diskette bleibt davon, außer bei schwerwiegenden Fehlern in den Datei-Routinen, unberührt. Die ersten beiden Menüpunkte im Menü *AUSKUNFT* dienen der Suche nach Kunden und Lieferanten. Sie brauchen nur die ersten 4 Buchstaben des Namens einzugeben (beachten Sie aber bitte die Klein- beziehungsweise Großschreibung der Buchstaben) und Sie erhalten alle Kunden oder Lieferanten, deren Namen mit diesen Buchstaben beginnen, komplett mit Anschrift und weiteren Informationen aufgelistet.

Der dritte Befehl, *Artikel-Liste,* zeigt Ihnen die Aufstellung der bislang erfaßten Artikel. Ausgegeben werden jeweils 15 Datensätze. Mit der Funktionstaste F2 können Sie weiterblättern, bis das Ende der Liste erreicht ist. Wenn Sie die Taste F3 betätigen, erhalten Sie die Artikelliste auch schriftlich. Ein Muster dieser Liste finden Sie am Ende dieses Kapitels.

Mit dem vierten Befehl, *Angebote zeigen,* können Sie die gespeicherten Angebote abrufen. Sie werden dazu aufgefordert, eine Angebotsnummer einzugeben und bekommen dann dieses Angebot, welches Grundlage für die Bestellungen ist, gezeigt. Wenn Sie F2 wählen, können Sie sich das nächste Angebot zeigen lassen. In dem Programm VEL sind alle Wahl-Funktionen über die Funktionstasten einzugeben.

Die nächsten beiden Befehle, *Lieferbed. zeigen* und *Zahlungsbed. zeigen,* listen die Liefer- und Zahlungsbedingungen mit den jeweiligen Nummern auf. Diese Nummern müssen Sie verwenden, wenn Sie die entsprechende Bedingung bei einer Lieferung oder Bestellung verwenden wollen.

Der folgende Befehl, *Bestand zeigen,* ist für eine Überprüfung der Arbeitsweise des Programms besonders wichtig. Er gibt einen Überblick über den Lagerbestand aus, also wieviel im Augenblick von jedem Artikel verfügbar ist.

Der Befehl *Preisliste zeigen* listet die Preise auf, die Sie (Ihre Firma) für die verschiedenen Artikel verlangen. Diese Preise können nach Mengen gestaffelt sein. Auch hier haben Sie die Möglichkeit, die Liste mit F3 ausdrucken zu lassen. So haben Sie für sich und Ihre Kundschaft immer die aktuellen Preise griffbereit.

Die folgenden beiden Befehle liefern Ihnen einen Überblick über den aktuellen Auftragsbestand. *Alle Auftraege zeig* listet sämtliche gespeicherten Aufträge auf, *offene Auftraeg zeig* nur die, die noch nicht erfüllt wurden.

Bei den Bestellungen haben Sie sogar drei Listen zur Auswahl. Sie können sich entweder alle Bestellungen zeigen lassen, oder die noch nicht gelieferten, oder nur die, die bereits überfällig sind (und angemahnt werden sollten).

Nachdem Sie nun wissen, welche Informationen Sie abrufen können, wenden Sie sich der Eingabe zu. Zunächst müssen Sie die »allgemeinen« Daten hierfür vorgeben.

Wählen Sie dazu bitte *Lieferbeding.neu* aus dem Menü *EINKAUF*. Die erste Nummer wird Ihnen gezeigt und Sie müssen die entsprechende Lieferbedingung dazu eingeben. Um ein besseres Druckbild zu erzielen, werden diese in zwei Zeilen eingegeben. Sie können den Text dabei jeweils soweit eingeben, wie die gepunktete Linie darunter reicht. (Text, der über diese Linie hinausgeht, wird abgeschnitten.) Wenn Sie einen Eingabefehler gemacht haben, den Sie mit der BACKSPACE-Taste nicht mehr beheben können, haben Sie die Möglichkeit, die Eingabe mit F1 abzubrechen. Erst wenn Sie F2 drücken, wird der Text gespeichert und Sie können den nächsten Datensatz eingeben. Verlassen Sie diesen Programmteil (wenn Sie alle Lieferbedingungen eingegeben haben), indem Sie zweimal RETURN und dann F1 betätigen. Anschließend können Sie auf dieselbe Weise die Zahlungsbedingungen eingeben, indem Sie den Befehl *Zahlungsbed.neu* wählen.

Der nächste Schritt ist das Abspeichern Ihrer Kundenliste. (Anschließend gehen Sie bitte genauso bei der Eingabe der Lieferanten vor.) Wählen Sie dazu den Befehl *Kunden neu* aus dem *VERKAUF*-Menü. Das Programm vergibt daraufhin eine neue Kundennummer und zeigt sie Ihnen. Sie müssen nun die vollständige Anschrift und die Schlüsselzahlen für die Liefer- und Zahlungsbedingungen, die dieser Kunde erhält, eingeben. Auch hier gilt wieder, daß die Speicherung erst erfolgt, wenn Sie F2 drücken, und daß jederzeit ein Abbruch der Funktion mit der Taste F1 möglich ist. Eine Änderung der Anschrift eines Kunden ist im Programm nicht vorgesehen. In einem solchen Fall können Sie sich aber damit behelfen, den Kunden als neuen Kunden unter einer neuen Kundennummer abzuspeichern.

Dann müssen Sie die Artikel, die Sie vertreiben, in die Artikel-Datei eintragen. Rufen Sie dazu den Befehl *Artikel neu* aus dem *LAGER*-Menü auf. Das Programm vergibt nun eine neue Artikelnummer (indem es einen freien Datensatz sucht) und zeigt sie an. Zuerst müssen Sie nun die Wiederbeschaffungszeit dieses Artikels in Wochen angeben. (Die Wiederbeschaffungszeit ist die Lieferzeit Ihres Lieferanten für diesen Artikel zuzüglich der Durchlaufzeiten für Transport, Einlagerung und so weiter.) Bei der Eingabe der Artikelbezeichnung halten Sie sich bitte an die gepunktete Linie unterhalb der Eingabezeile. Auch hier erfolgt wieder die Speicherung erst nach Tastendruck von F2.

Als nächsten Schritt rufen Sie nun bitte *Angebote neu* im Menü *EINKAUF* auf. Sie können in der daraufhin erscheinenden Maske entweder Änderungen an bereits eingegangenen Angeboten vornehmen oder neue Angebote eingeben. Zunächst werden Sie dazu aufgefordert, die Angebotsnummer einzugeben. Verwenden Sie hier bitte die gleiche Nummer, die auch der betreffende Artikel erhalten hat. (Bei der automatischen Bestellabwicklung wird jeder Artikelnummer das Angebot mit der identischen Angebotsnummer zugeordnet.) Geben Sie dann die Nummer des Lieferanten an, von dem Sie die Ware beziehen wollen. Ist eine Mindestabnahmemenge vorgeschrieben, müssen Sie diese im nächsten Schritt angeben. Bei der Änderung eines Angebots zeigt VEL Ihnen bei jeder Eingabe den alten Wert des entsprechenden Feldes. Diesen können Sie durch Betätigung von RETURN übernehmen, ohne ihn neu eintippen zu müssen. Sie haben nun die Möglichkeit, drei verschiedene Staffelpreise mit den zugehörigen Mengen einzugeben. Wenn eine Bestellung geschrieben wird, sucht sich das Programm automatisch den zur bestellten Menge passenden Preis aus. Die Speicherung beziehungsweise weitere Eingabe von Angeboten und der Abbruch des Befehls geschieht bei *Angebote neu* genauso wie bei den anderen Eingabefunktionen.

Jetzt fehlen nur noch die Preise, die Sie Ihren Kunden berechnen wollen. Dazu wählen Sie bitte *Preisliste* aus dem Menü *VERKAUF*. Bis auf die Eingabe des Lieferanten und der Mindestmenge gehen Sie genauso vor wie bei *Angebote neu*. Auch Ihre Verkaufspreise können Sie nach Mengen gestaffelt eingeben. Weiterhin muß die hier eingetragene Artikelnummer der vom Rechner vergebenen Nummer in der Artikel-Datei entsprechen.

Bei allen Eingaben, die Sie bislang gemacht haben, ist nichts »Wesentliches« passiert. Sie haben das Programm erst mit einem gewissen Grundstock von Daten ausgestattet. Ab jetzt wird das anders. Alle nun folgenden Eingaben führen sofort zu Aktionen. Sobald diese einmal angelaufen sind, ist kein Eingreifen in diesen Prozeß möglich. Aber keine Sorge; ein kleines Hintertürchen bleibt Ihnen noch. Davon aber später mehr.

Arbeiten mit VEL

Zunächst müssen Sie die Artikel in der Lager-Datei mit den richtigen Werten versorgen. Legen Sie also bitte für jeden Artikel den minimalen und den maximalen Lagerbestand fest und wählen Sie aus dem Menü *LAGER* den Unterpunkt *Bestand* an. Die Eingabe läuft genauso ab wie beim Menüpunkt *Preisliste*. Wenn ein Artikel im Augenblick nicht auf Lager ist, geben Sie bitte trotzdem den minimalen und den maximalen Bestand ein. Bei Feldern, in die Sie nichts eingeben wollen, brauchen Sie wie üblich nur die RETURN-Taste zu drücken. Wenn Sie alle Bestände eingegeben haben, drücken Sie bitte

die Taste F1 (für »abbrechen«). Jetzt prüft das Programm, ob Lieferungen möglich oder Bestellungen nötig sind. Da Sie eventuell durch die Eingabe des minimalen Bestandes einen Bedarf erzeugt haben, fängt der Drucker nun sofort an, Bestellungen auszudrucken. Wie Sie aus den einzelnen Menüs ersehen können, gibt es keinen expliziten Befehl für das Drucken von Bestellungen. Diese werden immer automatisch nach Bedarf erstellt. Je nach Bestellmenge werden die korrekten Artikel-Preise aus den Angeboten entnommen und dem Lieferanten ein Liefertermin vorgeschrieben. (Der Liefertermin ist gleich der Wiederbeschaffungszeit minus eine Woche plus das Tagesdatum.) Für jeden Artikel wird eine separate Bestellung vorgenommen. Ein Zusammenfassen mehrerer Artikel, die bei demselben Lieferanten geordert werden, auf einer Bestellung ist nicht vorgesehen. (Hier können Sie Ihre eigenen Programmierkünste spielen lassen.)

Da in Ihrer Firma hoffentlich manchmal auch etwas verkauft wird, werden Sie nun erfahren, wie Sie Aufträge eingeben können. Rufen Sie dazu bitte den Befehl *Auftrag neu* aus dem Menü *VERKAUF* auf. Nachdem Sie die Taste F2 betätigt haben, gelangen Sie in die entsprechende Eingabemaske. Die Auftragsnummer wird wieder vom Programm vergeben und auf dem Bildschirm gezeigt. Wenn Sie die Kundennummer angegeben haben, holt das Programm die komplette Kundenanschrift aus der entsprechenden Datei und zeigt sie Ihnen. Genauso erhalten Sie nach der Eingabe der Artikelnummer im Klartext den Artikelnamen darunter. Jetzt geben Sie bitte noch die gewünschte Menge ein und Sie können dann die Berechnung des Preises dem Programm überlassen. Letzte Eingabe ist der vom Kunden gewünschte Liefertermin. Ist die Ware verfügbar (auf Lager), wird der Termin akzeptiert. Liegt der Artikel nicht auf Lager, wird als Liefertermin die entsprechende Wiederbeschaffungszeit dem Tagesdatum hinzugerechnet. Der Drucker fängt nun wieder an zu arbeiten und gibt die Auftragsbestätigung aus. Sie können nun den nächsten Auftrag eingeben. Wenn Sie diesen Programmteil verlassen, prüft der Rechner, ob Aufträge ausgeführt werden können. Hat er einen gefunden, wird umgehend die Rechnung gedruckt. Das Drucken von Lieferscheinen ist noch nicht realisiert worden.

Nun fehlt nur noch eine Eingabe-Routine, der Wareneingang. Diesen Punkt finden Sie im *LAGER*-Menü. Wenn Sie also eine Lieferung erhalten haben, wählen Sie bitte den Befehl *Wareneingang* im Menü *LAGER*, und geben Sie die Bestellnummer und die gelieferte Menge ein. Achten Sie darauf, daß Teillieferungen im Programm nicht vorgesehen sind und deshalb mit der Eingabe des Wareneinganges die komplette Bestellung als ausgeliefert gilt. Eventuell könnten Sie deshalb vorsorglich den Text *Teillieferungen werden nicht akzeptiert* in das Bestellformular mit aufnehmen. Beim Befehl *Wareneingang* ist nur die Registrierung eines Wareneingangs vorgesehen. Bei mehreren Lieferungen müssen Sie ihn deshalb mehrmals neu anwählen. Nach der Buchung der Warenmenge prüft das Programm, ob nun eventuell Aufträge ausgeliefert werden können. Ist dies der Fall, erfolgt sofort der Druck der entsprechenden Rechnungen.

Nachdem alle Menübefehle besprochen sind, bleibt nur noch die Frage offen, wie wir eventuelle Unstimmigkeiten korrigieren können. Diese Korrekturen können Sie nur in einer Datei vornehmen, der Bestands-Datei. Wählen Sie dazu den Befehl *Bestand* aus dem Menü *LAGER*. In der daraufhin auftauchenden Bildschirmmaske können Sie Korrekturen am tatsächlichen Bestand, am Maximum und Minimun, bei den Vormerkungen und beim bestellten Bestand vornehmen.

Anhang 2
Überblick über die Beispielprogramme

In den vorangegangenen Kapiteln sind eine Vielzahl von Beispielprogrammen vorgestellt worden. Damit Sie diese nicht alle eintippen müssen, liegt diesem Buch eine 3,5-Zoll-Diskette im Amiga-Format bei. Sie enthält alle abgedruckten Beispielprogramme als Amiga-Basic-Programme, die Sie (fast) sofort starten können, ohne sich lange mit der Eingabe und der Suche nach Tippfehlern aufhalten zu müssen. Vorher sind nur einige wenige »Handgriffe« nötig.

Starten Sie Ihren Amiga und legen Sie, sobald die Workbench erscheint, die beiliegende Diskette in ein Laufwerk. Öffnen Sie das daraufhin auftauchende Disketten-Piktogramm dann mit einem Doppelklick und betrachten Sie sich seinen Inhalt. Die Diskette enthält unter anderem sechs Schubladen (Directories), die den fünf Kapiteln dieses Buches und dem Anhang 1 entsprechen. Die Schublade »ScreenNeu« enthält Versionen der Beispielprogramme, die an die Workbench 1.2 angepaßt sind. Schieben Sie nun die »Extras«- beziehungsweise »ExtrasD«-Diskette, die Ihrem Amiga beim Kauf beilag, in ein Laufwerk, und legen Sie das Piktogramm »AmigaBasic« von dieser Diskette auf die Diskette mit den Programmen.

Wenn die Diskettenlaufwerke Ihres Amiga zur Ruhe gekommen sind, können Sie mit den Programmen arbeiten. Öffnen Sie dazu einfach mit einem Doppelklick eine der Schubladen und suchen Sie in dem Fenster, das daraufhin auftaucht, nach einem Piktogramm mit dem Namen des gewünschten Programms. Machen Sie auf dieses Piktogramm erneut einen Doppelklick und das entsprechende Programm wird gestartet. Wenn Sie es anhalten wollen, brauchen Sie nur den Befehl *Stop* aus dem Menü *Run* zu wählen. Wollen Sie das Listing des Programms am Bildschirm betrachten, müssen Sie *Show List* aus dem Menü *Windows* wählen.

Die »Gliederung« der beiliegenden Diskette entspricht weitgehend der Gliederung dieses Buches. Die im Kapitel 2 besprochenen und abgedruckten Programme finden Sie zum Beispiel in der Schublade »2.Kapitel«. Enthält ein Kapitel sehr viele Programme, so sind diese teilweise noch einmal in »Unter-Schubladen« gegliedert. Auch diese enthalten manchmal wieder »Unter-Unter-Schubladen«. Diese Schachtelung von Schubladen kann auf den ersten Blick etwas verwirrend wirken. Die folgende Liste soll Ihnen deshalb ein wenig dabei helfen, sich auf der Diskette zurechtzufinden. Sie enthält nach Schubladen geordnet alle Dateien (Piktogramme), die sich auf der Diskette befinden. (Die Liste ist übrigens mit Hilfe des AmigaDOS-Befehls *DIR* entstanden.) Die Schubladen selbst tauchen ebenfalls auf. Sie sind an einem (*Dir*) hinter dem Namen zu erkennen. Unter dem Namen jeder Schublade folgen dann, etwas nach rechts eingerückt, die Namen der Dateien und Schubladen, die in dieser Schublade enthalten sind.

1.Kapitel (dir)
  Dreieck
  Dreiecksfl
  freilassen
  langsamer
  oder
  P1.6.3-1
  Pfeil
  Plus
  Postleit
  Quader
  Textverzweigung
  und
  Variablen
2.Kapitel (dir)
  PAINTAMIGA (dir)
    WP (dir)
      Blitz
      pinh
      pinl
      pinr
    SP (dir)
      pins1
      pins2
      pins3
      pins?
    PAINT-AMIGA
    SP.info
  Unterprogramme
    Farbtafel
    Farbtafel1
    Abloesung
    ausfuehren
    B.Schirm
    B.Schirm2
    Dame
    Farbpalette
    Gewitter
    Hasen
    Kelch
    Mauer
    P2.1-1
    PAINTAMIGA
    Regenbogen
    Tunnel
    Wald
    zweiSCREENS
3.Kapitel (dir)
  REZEPTE (dir)
  KFZDATEI (dir)
    A-LT-333
    AMI-GA-444
    KFZ-Datei
    N-EU-111
    name
    TE-ST-222
  AMIGA-K
  AMIGA-L
  Artikel
  Farbe
  Fehler
  FeldSpeicher
  iffLesenSpeich
  KFZDATEI.info
  Koch-Datei
  reinraus
  sort
  StringSpeich
  SW-Hardc
  Teile
  Termin
  VarSpeicher
  Video-Datei
  Zeichen
4.Kapitel (dir)
  Geraeusch (dir)
    Abklang
    Alarmstufe1
    Feuerwehr
    Hochspannung
    Hupe
    Laser
    Masch.Gewehr
    Motor
    Ufo-landet
  Sprache (dir)
    deutsch
    deutsch2
    englisch
    englisch2

Rechenma
Sprache
Rollen (dir)
Fehlstart
Hochhaus
Muster
Roll-Test
Softball
Vorspann
Sprites (dir)
Abschuss
abw
ang
Bewegung
Bummelbahn
Farbenpracht
Kunstflug
lkw
lok1
lok2
Sprite-Test
Steuerknueppel
wag
SOUND (dir)
Abendlied
Dreiecksw
Pulsw
Rauschen
Saegezahn
schwaebische
Sinusw
Tonleiter
Tonleiter2
Volksweise
5.Kapitel (dir)
ABC
Adventure
aufloesen
Biblio
Bob-Farben
famu
Fatamorgana
Fluesse
Get-Mani
graphics.bmap
invert.
Kreisber
Lotto
Mini
Obj.Dat.
RamDisk
Sprite-Far
sprtest
Staedtetesttext
Wuerfel
screenNeu (dir)
WP (dir)
Blitz
H
v
SP (dir)
pins1
pins2
pins3
pins?
baum
Farbenp-V1+2
Farbpal-V1+2
flugz
Gewit-Test
Gewitt-V1+2
INFORMATION_bitte_lesen
Kunstfl-V1+2
PAINT-Test
PAINT-V1+2
Regen-V1+2
Schirm2-V1+2
SCREENzeig
spr32
test32
Anhang (dir)
VEL

Stichwortverzeichnis

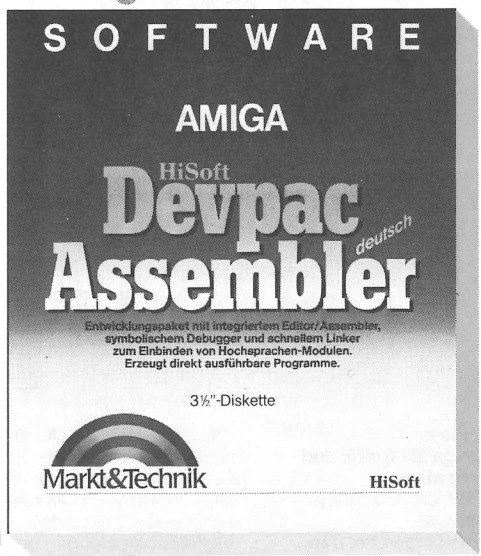
Amiga-Software

Devpac Assembler

Ein komplettes Assembler-Entwicklungspaket für Amiga-Programmierer!
Mit diesem Entwicklungspaket erhalten Sie eine Reihe von Programmen, die Sie bei der Erstellung von 68000er-Assembler-Programmen und deren Umgang tatkräftig unterstützen. Enthalten sind:

- GenAmiga, ein Makro-Assembler mit integriertem Bildschirmeditor.

Durch die Kombination von Assembler und Editor wird eine problemlose Eingabe und ein schnelles Assemblieren (1000 Zeilen in 6 Sekunden bei Verwendung einer RAM-Disk) gewährleistet. Der Editor unterstützt alle gängigen Standardfunktionen und arbeitet unter Intuition – es muß also nicht auf Fenstertechnik, Mausbedienung und Menüs verzichtet werden.
Der Zwei-Paß-Assembler erfüllt den Motorola-Standard. Mit hoher Geschwindigkeit können sowohl linkbare als auch sofort ausführbare Binärdateien generiert werden. Bei Auftritt eines Fehlers erfolgt automatisch ein Rücksprung in den Editor. Zeitraubendes Nachladen und Speichern entfällt.



Bestell-Nr. 51656

DM 148,–*

(sFr 134,–*/öS 1690,–*)

* Unverbindliche Preisempfehlung

- MonAmiga, ein symbolischer Debugger und Disassembler.

Mit MonAmiga können Sie Ihre Programme im Speicher inklusive aller Labels überprüfen und müssen sich nicht mit sechsstelligen Hex-Zahlen auseinandersetzen. MonAmiga bietet auch die Möglichkeit, einzelne Befehle schrittweise nacheinander auszuführen. Durch Programmfehler entstehende Exceptions werden vom Debugger so abgefangen, daß es selten zu den berüchtigten »Guru«-Meldungen kommt.
Devpac – Ihre professionelle Komplettlösung!

Hardware-Anforderungen:
Amiga 500, 1000 oder 2000 mit mindestens 512 Kbyte Arbeitsspeicher, Monitor und mindestens einem Diskettenlaufwerk.

Software-Anforderung:
Kickstart Version 1.1 oder höher

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

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

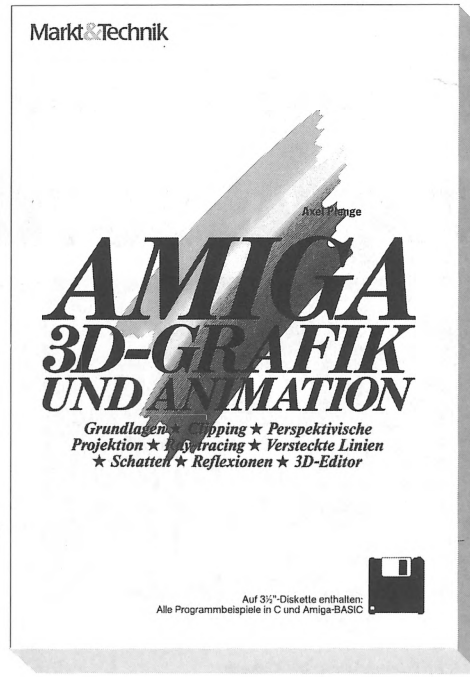
801322

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

Bücher zum Amiga

M. Breuer
DELUXE Grafik mit dem Amiga
1987, 370 Seiten
Eine behutsame Einführung in die grundlegenden Konzepte des jeweiligen Programms. Führt anhand kleiner überschaubarer Beispiele die wichtigsten Programmbefehle vor. Ein Nachschlageteil zu jedem Programm listet alle Befehle und ihre Bedeutung auf. Den Abschluß der Beschreibung jedes Programms bildet eine Sammlung von Tips und Tricks.
- Das deutsche Handbuch für den kreativen Grafikkünstler mit der DELUXE-Serie.

Best.-Nr. 90412
ISBN 3-89090-412-2
DM 49,–



A. Plenge
Amiga 3D-Grafik und Animation
1988, ca. 350 Seiten, inkl. Disk.
Angefangen bei den einfachsten Problemstellungen lernen Sie, professionelle 3D-Grafiken auf Ihrem Commodore Amiga zu planen, zu programmieren und darzustellen. Sämtliche theoretischen Grundlagen werden Ihnen anschaulich und leichtverständlich vermittelt. Auch scheinbar komplizierte Grafiken wie Schattenbildung, Reflexion, durchsichtige Gegenstände, Vielfachspiegelungen oder »raytracing« werden verständlich dargestellt.
- Eine ausführliche und leichtverständliche Anleitung für die Erstellung von dreidimensionalen Grafiken.

Best.-Nr. 90526
ISBN 3-89090-526-9
DM 69,–

M. Kohlen
Grafik auf dem Amiga
1987, 337 Seiten
Dieses Buch enthält eine ausführliche Beschreibung der Grafik-Hard- und -Software, deren Funktionsweise und führt in die Grundzüge der Grafikprogrammierung ein. In den folgenden Kapiteln werden diese Kenntnisse dann in praktischen Beispielen umgesetzt. Außerdem bietet das Buch einen Überblick über die vorhandenen Soft- und Hardware-Erweiterungen für den Amiga.
- Eine Pflichtlektüre für jeden, der sich für die phantastische Grafik des Amiga interessiert.

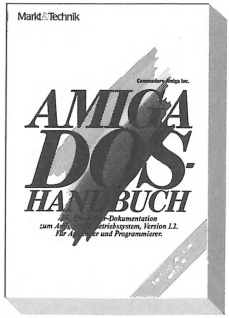
Best.-Nr. 90236
ISBN 3-89090-236-7
DM 49,–

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

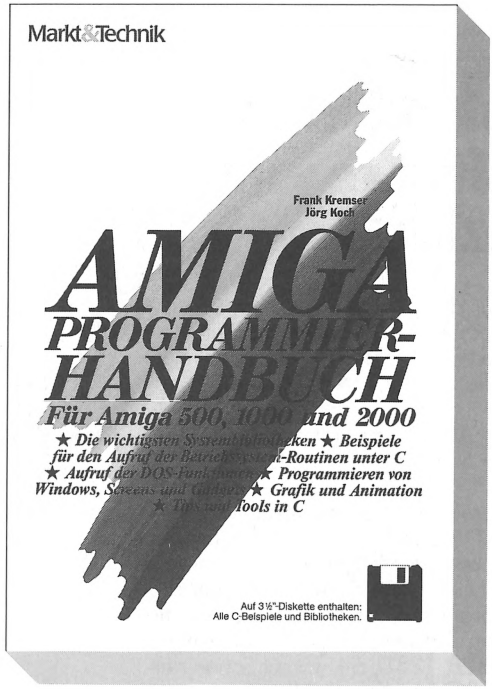
712300

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

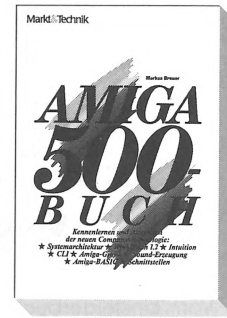
Bücher zum Amiga



Commodore-Amiga Inc.
Das Amiga-DOS-Handbuch für Amiga 500, 1000 und 2000
1988, 342 Seiten
Die Pflichtlektüre für jeden Commodore-Amiga-Anwender und Programmierer: eine Entwickler-Dokumentation zum Amiga-DOS-Betriebssystem, Version 1.2. Programmierung, interne Datenstruktur und Diskettenhandling. Mit diesem Buch lernen Sie das mächtige Amiga DOS schnell und sicher zu beherrschen. Alle Möglichkeiten des Systems, bis hin zum »Multi-Tasking« werden ausführlich und anschaulich beschrieben.
Best.-Nr. 90465
ISBN 3-89090-465-3
DM 59,–



Kremser/Koch
Amiga Programmierhandbuch
1987, 387 Seiten,
inkl. Diskette
Eine tolle Einführung in die »Interna« des Amiga: Die wichtigsten Systembibliotheken, die das Betriebssystem zur Verfügung stellt, werden anhand vieler Beispiele erklärt. Aus dem Inhalt: Aufruf der Betriebssystem-Routinen unter C, Aufruf der DOS-Funktionen, Programmieren von Windows, Screens und Gadgets, Grafik und Animation, Tips und Tools in C.
Best.-Nr. 90491
ISBN 3-89090-491-2
DM 69,–



M. Breuer
Das Amiga-500-Handbuch
1987, 489 Seiten
Eine ausführliche Einführung in die Bedienung des Amiga 500. Kennenlernen und Anwenden der neuen Computer-Technologie: Systemarchitektur, Workbench 1.2, Intuition, CLI, Amiga-Grafik, Sound-Erzeugung, Amiga-BASIC und Schnittstellen. Neben dem Handbuchteil mit vielen Bildschirmfotos und Übersichtstabellen, die Ihnen beim täglichen Einsatz helfen; schnell und reibungslos zu arbeiten, enthält das Buch eine ausführliche Beschreibung des Amiga 500 und seines Zubehörs.
Best.-Nr. 90522
ISBN 3-89090-522-6
DM 49,–

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

712328

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

Horst-Rainer Henning

Programmieren mit AMIGA-BASIC

Der Autor:

Persönliches: Geboren 1943 in Nürnberg, verheiratet, ein Sohn.
Berufliches: Nach einigen Semestern Maschinenbaustudium Tätigkeit in einem technischen Büro. Bis heute technischer Einkäufer in einer Maschinenfabrik.
Computer: Seit 1983 Hobby-Programmierer auf einem C64. Programmierte den 6510 zunächst in BASIC, später in Assembler. 1986 kaufte er einen der ersten Amiga-Rechner und ist seitdem überzeugter Anhänger dieses Computertyps.

Eine moderne Hardware-Architektur, erstaunliche Grafikfähigkeiten, superschnelle Prozessoren und eine freundliche grafische Benutzeroberfläche sind die herausragenden Merkmale der neuen Commodore-Amiga-Modelle. Auch das mitgelieferte Amiga-BASIC paßt zu einem Computer der Superlative. Es ist schnell und vielseitig, fast jeder Anwenderwunsch läßt sich damit realisieren.
Was können Sie von diesem Buch erwarten? Es wird Sie in die Programmierung des Amiga-BASIC einführen. Sie brauchen keine Angst zu haben, daß Sie mit theoretischem Wissen erschlagen werden. Die praktische Anwendung steht im Vordergrund. Etwa 100 Programme und viele Beispiele zeigen Ihnen den Einsatz von über 200 Amiga-BASIC-Befehlen. Die Listing-Auswahl ist vielfältig, größere Anwendungen sind ein Malprogramm und eine leistungsfähige Dateiverwaltung.

Aus dem Inhalt:

- Programmieren von Fenstern und komfortablen Menüs
- Die Befehle zur Sprach- und Musikausgabe
- Sequentielle Dateiverwaltung
- Dateien mit direktem Zugriff
- Animation (bewegte Grafiken wie Sprites)
- Tastaturabfrage
- Spieleprogrammierung
- Zahlreiche Tips und Tricks aus der Praxis

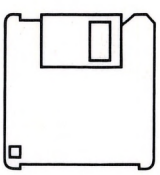
Dem Buch liegt eine 3½″-Diskette mit über 100 Programmbeispielen bei.

Hard- und Software-Anforderungen:

- Amiga 500, 1000 oder 2000 mit 512 Kbyte Arbeitsspeicher
- gegebenenfalls ein grafikfähiger Matrixdrucker und ein Joystick
- Amiga-BASIC von Microsoft

ISB N 3-89090-434-3

Markt&Technik



4 001057 904347

DM 59,-
sFr 54,30
öS 460,20