Markt&Technik

Jürgen Hückstädt

Der Schneider CPC 6128

Alles über BASIC · CP/M Plus · Relative Dateiverwaltung
Ein leichtverständliches Nachschlagewerk mit vielen interessanten Anwendungsbeispielen.

Der Schneider CPC 6128

Jürgen Hückstädt

Der Schneider CPC 6128

Alles über BASIC
- CP/M Plus
- Relative Dateiverwaltung

Ein leichtverständliches Nachschlagewerk mit vielen interessanten Anwendungsbeispielen

Markt & Technik Verlag

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Hückstädt, Jürgen:
Der Schneider CPC 6128 : alles über BASIC, CP/M Plus, relative Dateiverwaltung ;
e. leichtverständl. Nachschlagewerk mit vielen interessanten Anwendungsbeispielen / Jürgen Hückstädt. —
Haar bei München : Markt-und-Technik-Verlag, 1985
ISBN 3-89090-192-1

15 14 13 12 11 10 9 8 7 6 5 4 3
89 88 87 86

ISBN 3-89090-192-1

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Jantsch, Günzburg
Printed in Germany

Vorwort

Das vorliegende Buch ist für alle diejenigen geschrieben, die sich über den Umgang mit dem Schneider CPC-6128-Computer informieren und damit arbeiten wollen. Dabei spielt es keine Rolle, ob Sie bereits ein erfahrener Programmierer, ein Einsteiger oder ein Umsteiger von einem anderen Computer sind.

Das Buch führt Sie durch alle wichtigen Lernphasen und hilft Ihnen dabei, diesen phantastischen Computer näher kennenzulernen. Während sich der Anfänger etwas intensiver mit den ersten Kapiteln befassen wird, sind diese für den fortgeschritteneren Programmierer, den vermutlich die weiter hinten behandelte Themen wie Dateiverwaltung, CP/M und Nutzung der zweiten Speicherbank mehr interessieren werden, ein nützliches und hilfreiches Nachschlagewerk.

Im einzelnen werden folgende Bereiche besprochen: Zunächst geht es um elementare Dingen wie die Zusammenstellung des Computersystems und dessen Ausbaumöglichkeiten. Es folgen einige wichtige Grundlageninformationen, bevor wir in Kapitel 2 Schritt für Schritt mit der BASIC-Programmiersprache Bekanntschaft schließen.

Im Anschluß an die Programmschleifen und Verzweigungen in Kapitel 3 erfahren wir in Kapitel 4 mehr über den Programmaufbau und befassen uns mit menügesteuerten Programmen. Kapitel 5 behandelt indizierte Variablen und gibt eine Einführung in die Verarbeitung von Feldern. Dabei handelt es sich um wichtige Grundlagen für das professionelle Arbeiten mit Dateien, mit denen wir uns in den Kapiteln 6 bis 8 näher beschäftigen. Davor erfahren wir noch in Kapitel 6 alles Wissenswerte über die Floppy und das Betriebssystem AMSDOS.

Ein Leckerbissen ganz besonderer Art ist Kapitel 7. Dort lernen wir verschiedene Befehlserweiterungen kennen, die auch eine relative Dateiverwaltung einschließen, für die der CPC 6128 eigentlich nicht vorgesehen ist.

Das folgende Kapitel 8 handelt von der zweiten Speicherbank, die den 6128 besonders auszeichnet. Wir lernen, wie wir sie zur Speicherung von Bildschirminhalten und als RAM-Disc (Ersatzfloppy) nutzen können.

Alle Graphik- und Soundfreunde kommen in Kapitel 9 auf ihre Kosten. Der CPC 6128 besitzt nämlich vorzügliche Eigenschaften zum Programmieren von vielfarbigen Graphiken und mehrstimmigen Melodien.

Das letzte Kapitel ist CP/M gewidmet, dem am weitesten verbreiteten Betriebssystem der Welt, für das es auch die meiste Software gibt. Wir befassen uns dort mit allen wichtigen Grundlagen, die zum Arbeiten mit CP/M nötig sind.

Viel Spaß beim Studium dieses Buches und Freude bei der Arbeit mit Ihrem CPC 6128 wünscht Ihnen

Der Autor

Inhalt

1 Der Schneider CPC 6128

Der CPC 6128 ist die neueste Entwicklung der AMSTRAD-Computer, die in Deutschland unter dem Namen Schneider angeboten werden. Der erste Schneider-Computer, der CPC 464, kam Ende 1984 auf den deutschen Markt, und der CPC 6128 ist bereits das zweite Nachfolgemodell, das in weniger als einem Jahr angeboten wird.

Schon der CPC 464 war ein ganz neuartiger Computer auf dem Home- und Personal-Computer-Sektor, der zu einem erstaunlich günstigen Preis angeboten wurde. Er besitzt ein umfangreiches BASIC, das auch viele Anweisungen für die Graphik- und Soundprogrammierung enthält. Darüber hinaus kann dieser Computer mit einem 3-Zoll-Diskettenlaufwerk erweitert werden, das erstmals für Computer dieser Preisklasse zur Verfügung steht. Da der CPC 464 auch unter CP/M 2.2 betrieben werden kann, steht dem Anwender die größte Softwarebibliothek der Welt zur Verfügung. Wenig später kam der CPC 664 auf den Markt, dessen BASIC noch um einige Befehle erweitert wurde und der ein eingebautes Diskettenlaufwerk enthält. Wir können ihn als ein Übergangmodell zum CPC 6128 betrachten.

Die wichtigste Neuerung am CPC 6128 sind die zwei RAM-Bänke, die jeweils 64k enthalten, so daß der Computer über insgesamt 128K RAM verfügt. Da das "Herz" des Computers, der Z80A-Mikroprozessor, immer nur 64K direkt adressieren kann, muß er bei Bedarf zwischen beiden Bänken hin- und herschalten. Auch die CP/M-Anwender werden sich über die 128K RAM freuen, da somit auch CP/M Plus auf dem CPC 6128 zur Verfügung steht. CP/M Plus ist eine CP/M-Version, die mit zwei oder mehreren Speicherbänken arbeitet. Sie bietet nicht nur mehr Speicherplatz für Anwenderprogramme, sondern enthält darüber hinaus auch einen erweiterten und komfortableren Befehlssatz als CP/M 2.2. Auch trägt der größere RAM-Bereich dazu bei, daß Diskettenoperationen unter CP/M Plus viel schneller ablaufen als unter CP/M 2.2, da weniger Diskettenzugriffe erforderlich sind.

1.1 Das CPC-BASIC

Der CPC 6128 besitzt ein sehr leistungsstarkes BASIC. Zunächst wollen wir uns aber Klarheit darüber verschaffen, was BASIC überhaupt ist und einen kurzen Blick in die geschichtliche Entwicklung von Computern werfen.

Viele von uns haben noch mechanische Rechenmaschinen oder den guten alten Rechenschieber in Erinnerung, die einzigen Hilfsmittel, die es früher gab, um Rechenvorgänge schneller, genauer und vor allem bequemer durchzuführen, als es im Kopf möglich war. Im Zuge der technologischen Entwicklung suchte man bald nach Möglichkeiten, Berechnungen auch auf dem elektronischen Wege durchführen zu können.

Ende der vierziger Jahre entstanden die ersten Computer, die sich aber noch völlig von den heutigen unterschieden, insbesondere was den inneren Aufbau, die Größe und die Leistungsfähigkeit betraf. Sie arbeiteten noch mit echten Relais, um Stromkreise zu schließen bzw. zu unterbrechen. Ein solches Relais konnte folglich nur eine logische Null-/Eins-Entscheidung übernehmen, die man auch als wahr/falsch oder ja/nein auf der niedrigsten Betriebsebene eines Computers interepretieren kann.

An dieser Stelle wird uns auch klar, daß ein Computer nur auf binärer Ebene arbeiten, d.h. nur zwischen den beiden genannten Zuständen unterscheiden kann. Bei der Abarbeitung eines Programms geschieht dies allerdings tausend- und millionenfach. Diese kleinste Speicher- und Informationseinheit, die nur zwei Zustände bzw. Werte annehmen kann, wird in der Computertechnik als Bit bezeichnet. In der binären Schreibweise bezeichnet man diese Werte als 0 und 1. Wenn wir nun ein zweites Bit dazunehmen, verdoppeln sich die Informationsmöglichkeiten auf insgesamt vier Werte, wie das folgende Beispiel zeigt:

0 = 00 1 = 01 2 = 10 3 = 11

Jetzt fügen wir ein weiteres Bit hinzu, wodurch sich die Informationsmöglichkeiten wiederum auf acht verdoppeln:

0 = 000 1 = 001 2 = 010 3 = 011
4 = 010 5 = 101 6 = 110 7 = 111

Wir sehen also, daß durch Zuschaltung eines jeden Bits oder Schaltkreises doppelt soviele Werte dargestellt werden können. Mathematisch ausgedrückt wachsen sie im Verhältnis zwei hoch der Anzahl der Bits.

Sie können sich sicher denken, wieviele Schaltkreise oder Relais ein Oldtimer-Computer besessen haben muß, um arithmetische Berechnungen mit 10 Stellen Genauigkeit auszuführen. Kein Wunder also, daß man früher einen haushohen Computer brauchte, um nur die einfachsten Berechnungen durchführen zu können. Darüber hinaus waren diese Monstren auch extrem störanfällig und hatten einen enormen Strombedarf.

Im Laufe der Zeit wurden die Relais durch Elektronenröhren und später durch Transistoren ersetzt, wodurch die Computer immer kleiner und leistungsfähiger wurden. Heute, im Zeitalter der Mikroelektronik, können in einem winzigen Chip Tausende solcher Schaltkreise untergebracht werden. Dabei werden meist mehrere Bits zu einer größeren Einheit zusammengefaßt. Viele Computer, wie auch der CPC 6128, speichern und verarbeiten die Informationen in Einheiten von 8 Bit, die man auch als Bytes bezeichnet; Ihr CPC 6128 ist also ein 8-Bit-Rechner. Ein Byte kann demgemäß 2*2*2*2*2*2*2*2 oder 2 hoch 8 bzw. 256 verschiedene Zustände annehmen bzw. Zahlenwerte darstellen.

Stellen Sie sich jetzt einmal vor, Sie müßten Ihren Computer mit Bits und Bytes füttern oder programmieren und sollten auf dieser untersten Maschinenebene die Zahlen 4.8 und 13.5 miteinander multiplizieren. Sicher würden Sie bald die Lust verlieren und das Ergebnis schneller im Kopf ausrechnen, ganz abgesehen davon, daß Sie fundierte Kenntnisse in der Maschinenprogrammierung benötigten und diese auch fehlerfrei anwenden müßten. Für spezielle Anwendungsgebiete wird diese Programmierart zwar noch gelegentlich eingesetzt, im vorliegenden Buch wird sie aber nur am Rande gestreift.

Schon sehr frühzeitig erkannte man diese Schwierigkeiten und suchte nach Wegen, den Computer auch einfacher programmieren zu können. Im Laufe der Zeit wurden deshalb sogenannte höhere Programmiersprachen entwickelt, bei denen oft eine einfache und leicht verständliche Anweisung genügt, um hochkomplizierte Vorgänge im Inneren des Computers auszulösen. Auf dem CPC 6128 können Sie Berechnungen, wie die oben genannte Multiplikation, genauso einfach wie auf einem Taschenrechner durchführen.

Eine dieser Programmiersprachen ist BASIC. Sie wurde in Dartmouth (USA) in den frühen sechziger Jahren entwickelt. Sie galt als leicht zu erlernende Sprache, um Anfängern das Programmieren auf Mikrocomputern beizubringen.

Anfangs war BASIC eine Programmiersprache mit relativ wenigen Anweisungen. Im Laufe der Jahre wurde es jedoch ständig weiterentwickelt und erweitert. Heute ist es die Standard-Sprache der meisten Personal- und Homecomputer, die unproblematisch auf den verschiedensten Gebieten angewandt werden kann. Allerdings gibt es heute eine Vielzahl von BASIC-Dialekten, die zwar alle auf dem urspünglichen BASIC aufbauen, sich aber dennoch teilweise stark unterscheiden. So haben die meisten Computer ihren eigenen BASIC-Dialekt, der nicht so ohne weiteres auch von einem anderen Computer verstanden werden kann.

Neben BASIC gibt es noch eine Vielzahl unterschiedlicher Programmiersprachen, die größtenteils auf bestimmte Anwendungsbereiche zugeschnitten sind. FORTRAN, der Vorgänger von BASIC, wurde beispielsweise für den technisch-wissenschaftlichen Bereich und COBOL wurde für die kaufmännische Anwendungen entwickelt. FORTH ist eine Sprache mit relativ wenigen Befehlen und wird für die maschinennahe Programmierung angewandt. Eine andere Sprache für strukturiertes Programmieren ist ALGOL, aus dem später PASCAL und in jüngster Zeit auch ADA hervorgegangen sind. LOGO ist dagegen eine reine Lernsprache, mit der besonders Kinder den Umgang mit dem Computer einüben können. Weitere Sprachen sind C, COMAL, LISP, PL/1, um nur noch einige Beispiele zu nennen.

BASIC ist sicher deshalb so verbreitet, weil es auch heute noch leicht zu erlernen ist und die Vorzüge anderer Sprachen mehr oder weniger vereinigt, was allerdings von Dialekt zu Dialekt verschieden ist. Bei vielen Computern, wie auch beim CPC 6128, ist BASIC direkt nach dem Einschalten verfügbar und muß nicht erst geladen werden. Darüber hinaus kann es sowohl als Interpreter- als auch als Compiler-Sprache Anwendung finden. Ein Interpreter liest den BASIC-Text, wie er eingegeben wurde, und arbeitet in direkt ab. Ein Compiler dagegen übersetzt ihn erst in Maschinensprache oder in einen maschinennahen Code.

Die meisten der oben genannten Programmiersprachen sind reine Compiler-Sprachen, deren Programme zwar schneller ablaufen, dafür aber nicht so leicht ausgetestet und korrigiert werden können. Nach jeder Änderung müssen sie erneut übersetzt oder compiliert werden.

Das BASIC des CPC 6128 ist eine reine Interpretersprache, bei der Programme nur eingetippt werden müssen und dann sofort ablauffähig sind. Dies ermöglicht ein äußerst bequemes Programmieren, bei dem Fehler schnell erkannt und behoben werden können. Es gibt eine Reihe von BASIC-Anweisungen, die als elementares BASIC zu bezeichnen sind und die mehr oder weniger auf allen BASIC-Computern laufen. Dazu gehören

z.B. einfache Zuordnungsanweisungen, die Berechnung arithmetischer Ausdrücke, einfache Stringverarbeitung sowie Verzweigungsanweisungen wie GOTO und GOSUB.

Wenn Sie Ihren CPC 6128 in BASIC programmieren, können Sie viele Aufgabenstellungen auf einfache Weise realisieren. Nachfolgend betrachten wir einige Anweisungen, die das CPC-BASIC besonders wertvoll machen und die nicht auf jedem anderen BASIC-Computer implementiert sind.

So enthält der CPC 6128 neben der gängigen FOR...NEXT-Anweisung auch WHILE...WEND, wodurch die Beendigung einer Schleife nicht mehr von einer vorgegebenen Anzahl von Durchläufen, sondern von der Erfüllung einer Bedingung abhängt. Als weiteres können Sie hinter RESTORE eine Zeilennummer setzen, so daß die Abarbeitung von DATA-Zeilen nicht immer mit der ersten Zeile beginnen muß.

Bei den Verzweigungsanweisungen ist IF...THEN...ELSE zu erwähnen, wobei besonders die Alternative ELSE zu beachten ist, was strukturiertes Programmieren erleichtert.

Mit PRINT USING können Sie auf einfache Weise Tabellen mit Zahlenwerten oder Strings ausgeben, wahlweise auch über den Drucker. Dadurch entfallen umständliche Formatierungsroutinen.

Nicht zu unterschätzen ist der Aufruf einer Fehlerbehandlungsroutine infolge von ON ERROR GOTO, die einen Programmabbruch umgeht und fast ebenso wie ein normales Unterprogramm behandelt wird.

Auch für die hochauflösende Graphik stehen Anweisungen zur Verfügung, mit denen Sie auf einfache Weise farbige Figuren zeichnen können, wobei MOVE, PLOT und DRAW die wichtigsten Anweisungen sind.

In der Soundprogrammierung ist es ein Kinderspiel, mehrstimmige Melodien zu programmieren. Durch zusätzliche Beeinflussung der Lautstärke- und Tonhüllkurve können Sie beinahe sämtliche Musikinstrumente simulieren und natürlich auch eine Menge anderer Geräusche erzeugen.

Nicht vergessen wollen wir auch die Programmierhilfen wie beispielsweise die automatische Zeilennumerierung (AUTO), das Löschen meherer Programmzeilen (DELETE) oder die Ablaufverfolgung (TRON/TROFF), mit der sämtliche Programmzeilen auf dem Bildschirm erscheinen, während sie abgearbeitet werden.

1.2 CP/M

CP/M ist keine Programmiersprache wie BASIC, sondern ein universelles Betriebssystem, das prinzipiell auf allen Computern lauffähig ist, die einen Z80- oder 8080-Mikroprozessor besitzen.

Mit CP/M alleine können Sie bestenfalls Maschinenprogramme erstellen. Allerdings werden verschiedene Programmiersprachen angeboten, die nur im Zusammenhang mit CP/M einzusetzen sind. Dabei handelt es sich z.B. um TURBO PASCAL, M-BASIC oder FORTH. Mit M-BASIC können Sie auf ähnliche Weise BASIC-Programme schreiben, wie mit dem CPC-BASIC, jedoch handelt es sich hierbei um einen anderen BASIC-Dialekt, weshalb die Programme nur bedingt austauschbar sind.

CP/M-Software läuft auf allen Computern, die das CP/M-Betriebssystem enthalten oder laden können. Hierzu gehört auch der CPC 6128. Aus diesem Grunde gibt es auch ein riesiges Softwareangebot für CP/M-Programme. Zu den bekanntesten CP/M-Programmen gehört das Textverarbeitungssystem WORDSTAR, das Datenbankprogramm dBASE II sowie das Tabellenkalkulationsprogramm MULTIPLAN. Diese Programme sind übrigens auch für den CPC 6128 erhältlich (Verlag Markt&Technik).

Wenn Sie andere CP/M-Programme auf Ihrem CPC 6128 laufen lassen wollen, müssen diese zunächst angepaßt werden. Obwohl CP/M-Software auf allen CP/M-Rechnern lauffähig ist, ist in der Regel eine Anpassung an den jeweiligen Rechner nötig. Meistens liegt der CP/M-Software ein Installationsprogramm oder zumindest eine Installationsanweisung mit Patchadressen bei, so daß die Anpassung normalerweise keine Schwierigkeiten bereitet.

Anders ist es allerdings, wenn Sie ein CP/M-Programm auf einem anderen Diskettenformat erwerben, das sie nicht direkt in Ihren CPC 6128 laden können. In diesem Fall ist es sinnvoll, das Programm von einem anderen Rechner über eine RS232 oder V24-Schnittstelle zu übertragen. Dieses Thema würde aber den Rahmen dieses Buches sprengen, weshalb wir hier nicht näher darauf eingehen wollen.

Die oben genannten drei CP/M-Programme sind jedenfalls fertig für den CPC 6128 installiert, so daß Sie sich nicht mehr um die Anpassung zu bemühen brauchen. Mit Sicherheit werden im Laufe der Zeit aber noch weitere CP/M-Programme für den CPC 6128 erhältlich sein.

Wie eingangs schon erwähnt, gibt es derzeit die CP/M-Versionen 2.2 und 3.0 (CP/M Plus). CP/M Plus benutzt die zweite Speicherbank des CPC 6128 und besitzt gegenüber CP/M 2.2 mehr Befehle, die ein komfortables Arbeiten ermöglichen. Die meisten CP/M-Programme, die unter CP/M 2.2 geschrieben wurden, laufen auch unter CP/M-Plus. Sollte es hier in Ausnahmefällen einmal Schwierigkeiten geben, können Sie noch auf CP/M 2.2 zurückgreifen, das sich mit auf der Systemdiskette befindet.

In Kapitel 10 werden wir uns noch intensiv mit CP/M beschäftigen.

1.3 Die Hardware - Zusammenstellung des Systems

Der CPC 6128 gehört zu den Personal-Computern, die Sie bereits in einer Ausbaustufe erhalten, in der Sie ihn direkt betreiben können. Beim Kauf des Gerätes erhalten Sie zwei Kartons, wobei sich in dem einen das Grundgerät mit Tastatur samt Floppylaufwerk und in dem anderen ein Schwarzweiß- oder Farbmonitor befindet.

Die Inbetriebnahme ist denkbar einfach: Sie müssen nur die drei Kabel verbinden, von denen zwei am Monitor und eines am Computer angebracht sind. Die Kabel dienen zur Stromversorgung des Computers, zur Übertragung der Video- und Tonsignale an den Monitor sowie zur Stromversorgung der Floppy.

Haben Sie die Kabel ordnungsgemäß in die zugehörigen Buchsen eingesteckt, brauchen Sie nur noch das Netzkabel des Monitors anzuschließen und dann den Monitor und die Floppy einzuschalten. Folgende Einschaltmeldung erscheint daraufhin auf dem Bildschirm:

```
Schneider 128K Microcomputer (v3)

(c)1985 Amstrad Consumer Electronics plc
        and Locomotive Software Ltd.

BASIC 1.1

Ready
```

Jetzt können Sie mit dem Programmieren in BASIC beginnen.

Ob Sie nun einen Schwarzweiß- oder einen Farbmonitor verwenden, die Aufstellung und Inbetriebnahme des Computers ist in beiden Fällen gleich. Sie sollten sich jedoch vor dem Kauf des Computers genau überlegen, welchen Monitor Sie wünschen.

Falls sie Ihren CPC 6128 überwiegend für Farbgraphik und Computerspiele einsetzen möchten, ist zweifellos der teurere Farbmonitor zu empfehlen. Es gibt aber auch Fälle, in denen der Schwarzweiß-Monitor günstiger ist.

Obwohl der Farbmonitor eine sehr gute Auflösung besitzt, werden 80 Zeichen pro Zeile (MODE 2) nicht so scharf wiedergegeben wie auf einem Schwarzweiß-Monitor. Wenn Sie nun auf Ihrem Computer viele Daten eintippen müssen oder Textverarbeitung betreiben, ist der Schwarzweiß-Monitor günstiger. Abgesehen vom schärferen Schriftbild ist er mit seiner grünen Einfärbung für die Augen wesentlich angenehmer als ein Farbmonitor.

Zusätzlich zur Grundausstattung können Sie sich noch einen Drucker zulegen. Der NLQ-401-Drucker von Schneider ist ein leistungsfähiger und preisgünstiger Matrixdrucker, mit dem Sie auch ein NLQ (Near Letter Quality)-Schriftbild erzeugen können.

Matrixdrucker besitzen einzelne Nadeln, die für jedes Zeichen ein Punktraster auf das Papier drucken. In der NLQ-Betriebsart wird jedes Zeichen nach einem bestimmten Schema zweimal überdruckt, wodurch ein sehr sauberes Schriftbild entsteht, das dem eines Typenraddruckers (s.u.) schon sehr nahe kommt.

Außer dem NLQ 401 können Sie aber jeden Drucker anschließen, der mit einer Centronics-Schnittstelle ausgerüstet ist. Dies ist bei den meisten handelsüblichen Druckern der Fall. Ein besonders schönes Schriftbild erzeugen die Typenraddrucker, bei denen keine Punktraster, sondern Typen für die einzelnen Zeichen, die sich auf einem Rad befinden, gedruckt werden. Ein Typenraddrucker druckt aber wesentlich langsamer als ein Matrixdrucker.

Darüber hinaus gibt es noch Tintenstrahldrucker, die ähnlich wie ein Matrixdrucker arbeiten. Das Zeichenraster wird hier aber nicht von Nadeln, sondern von feinen Farbdüsen erzeugt, die kleine Farbtupfer auf das Papier spritzen.

Die neueste Errungenschaft sind die Laserdrucker, die die Druckfarbe mit einem Laserstrahl auf das Papier übertragen. Diese Drucker arbeiten zwar extrem schnell, dafür ist ihr Anschaffungspreis aber auch sehr hoch.

Abschließend noch ein paar Worte zur Tastatur. Diese ist ähnlich aufgebaut wie die Tastatur einer Schreibmaschine. Wenn Sie die normalen Buchstabentasten drücken, erscheinen die Zeichen in Kleinschrift. Durch gleichzeitiges Drücken der SHIFT-Taste erhalten Sie Großbuchstaben. Die CAPS LOCK-Taste schaltet generell auf Großschrift um und ersetzt das ständige Drücken der SHIFT-Taste. Ein erneutes Drücken der CAPS LOCK-Taste schaltet die Tastatur wieder auf den normalen Groß-/Kleinschreibmodus zurück.

Eine wichtige Taste ist die RETURN-Taste, mit der Sie Anweisungen, die Sie dem Computer erteilen, abschließen müssen. Die Funktion der ENTER-Taste ist vollkommen mit derjenigen der RETURN-Taste identisch. Ist daher in diesem Buch von der RETURN-Taste die Rede, können Sie ebensogut auch die ENTER-Taste drücken.

Die ESC- und CTRL-Taste haben verschiedene Bedeutungen, auf die wir hier nicht weiter eingehen wollen. Sie werden an anderer Stelle im jeweiligen Zusammenhang erklärt.

Neben der Haupttastatur befindet sich ein Ziffernblock mit den Tasten f0 bis f9. Diese Tasten können als Funktionstasten definiert werden. Darunter befinden sich vier Tasten mit Pfeilen, die in verschiedene Richtungen zeigen und Cursorsteuertasten heißen. Wenn Sie diese Tasten drücken, bewegt sich der Cursor (das kleine helle Quadrat) in die jeweilige Richtung.

Die Tasten CLR, COPY und DEL dienen zur Bildschirmeditierung und werden an späterer Stelle erklärt.

1.4 Einige wichtige Grundlagen

1.4.1 Direkt- und Programm-Modus

Ihren Computer können Sie auf zwei Arten betreiben, nämlich im Direkt- und im Programm-Modus. Im Direktmodus erteilen Sie dem Computer Instruktionen, die er unmittelbar nach der Eingabe abarbeitet. Sie tippen lediglich die entsprechenden Anweisungen ein und drücken die RETURN-Taste.

Anders ist es im Programm-Modus. Hier müssen Sie zunächst Programmzeilen eingeben, die der Computer in seinem Arbeitsspeicher ablegt. Das Programm wird erst abgearbeitet, wenn der Computer den entsprechenden Befehl dazu erhält.

Der CPC 6128 besitzt auch einen Bildschirmeditor. Ein Editor ist ein internes Maschinenprogramm, das dazu dient, BASIC-Zeilen und sonstige Instruktionen über Tastatur und Bildschirm einzugeben bzw. zu korrigieren (editieren). Ist eine Zeile fertig eingegeben, wird sie durch Drücken der RETURN-Taste abgeschlossen. Im Direktmodus wird sie sogleich ausgeführt und im Programm-Modus als Programmzeile gespeichert.

Wie versteht nun der Computer, ob er im Direkt- oder im Programm-Modus arbeiten soll? Dies ist ganz einfach: Steht nämlich nach dem Drücken der RETURN-Taste am Zeilenanfang eine Zahl, so handelt es sich um eine Programmzeile, anderenfalls um eine Zeile, die im Direktmodus abgearbeitet wird. Übrigens können die meisten Anweisungen sowohl im Programm- als auch im Direktmodus erteilt werden.

Zum besseren Verständnis wollen wir ein paar Beispiele betrachten. Nach dem Einschalten des Computers sehen Sie ein helles Kästchen auf dem Bildschirm, das als Cursor bezeichnet wird. Der Cursor erscheint immer dann, wenn der Computer bereit ist, neue Anweisungen über die Tastatur entgegenzunehmen. Er zeigt auf die Stelle, an der das nächste Zeichen auf dem Bildschirm erscheint. Geben wir ein Zeichen ein, erscheint es an der Cursorposition und der Cursor selbst rückt um eine Stelle nach rechts, wo er auf das nächste Zeichen wartet.

Bevor wir nun dem Computer Anweisungen erteilen, geben Sie zunächst einmal

```
CLS
```

ein, wodurch der Bildschirm gelöscht wird. Sollte dies nicht funktionieren, drücken Sie solange die RETURN-Taste, bis der Cursor in einer freien Bildschirmzeile steht und wiederholen Sie dann die Anweisung. Sollte der Bildschirm danach immer noch nicht gelöscht sein, setzen Sie den Computer in den Ausgangszustand, indem Sie gleichzeitig die Tasten

CTRL, SHIFT und ESC

drücken. Damit wird auch der Bildschirm gelöscht.

Da wir zunächst im Direktmodus arbeiten wollen, geben Sie nun folgendes ein:

PRINT "Guten Tag, ich bin der CPC 6128" <RETURN>

Das Wort <RETURN> tippen Sie nicht in Buchstaben ein, sondern drücken stattdessen die RETURN-Taste. Diese Schreibweise wollen wir vorläufig noch beibehalten, um Sie darauf aufmerksam zu machen, daß die Zeile durch Drücken der RETURN-Taste abgeschlossen werden muß.

Haben Sie die Zeile korrekt eingegeben, wird die Meldung

```
Guten Tag, ich bin der CPC 6128
Ready
```

auf dem Bildschirm ausgegeben. Der Cursor ist wieder sichtbar und signalisiert, daß der Computer bereit ist, weitere Anweisungen entgegenzunehmen.

Wir haben hier bereits den ersten Befehl, PRINT, kennengelernt, der den Computer anweist, eine Zeichenkette auf dem Bildschirm auszugeben.

PRINT bezeichnet man auch als Schlüsselwort, das der Computer als Befehl interpretiert. Hinter PRINT steht dann in Anführungszeichen die Zeichenkette, die ausgegeben werden soll.

Geben Sie die gleiche Anweisung noch einmal ein, wobei Sie aber an den Zeilenanfang die Zahl 10 setzen:

10 PRINT "Guten Tag, ich bin der CPC 6128" <RETURN>

Nach Drücken der RETURN-Taste wird jetzt keine Meldung ausgegeben, sondern Sie sehen nur der Cursor. Wir haben nämlich die Anweisung, eine Meldung auszugeben, in eine Programmzeile mit der Nummer 10 gepackt. Diese Programmzeile legt der Computer im Speicher ab. Damit erstellten wir ein BASIC-Programm, das aus einer einzigen Zeile besteht und das wir jetzt ablaufen lassen, indem wir

```
RUN    <RETURN>
```

eingeben. Wiederum erscheint die Meldung

```
Guten Tag, ich bin der CPC 6128
Ready
```

Im allgemeinen besteht ein BASIC-Programm natürlich aus einer Vielzahl von Programmzeilen, die in aufsteigender Reihenfolge numeriert sind. Dabei sind Zeilennummern zwischen 1 und 65535 zulässig. Zwar gibt es kein BASIC-Programm, das 65535 Zeilen umfaßt, denn dazu wäre der Speicher viel zu klein. Wir können aber verschiedene Programmsegmente mit runden Zeilennummern (z.B. 1000, 2000, 3000 usw.) beginnen lassen, um die Übersichtlichkeit zu erhöhen.

Doch zunächst wollen wir ein kleineres Programm mit mehreren Zeilen schreiben. Geben Sie dazu als nächstes ein:

```
NEW    <RETURN>
```

Dieser Befehl löscht den BASIC-Arbeitsspeicher, so daß Sie jetzt ein neues Programm eingeben können, das aus drei Programmzeilen bestehen soll. Von nun an verzichten wir in unseren Anweisungen auf den Hinweis <RETURN>, denn wir wissen ja bereits, daß wir jede BASIC-Zeile sowohl im Direkt- als auch im Programm-Modus durch Drücken der RETRURN-Taste abschließen müssen.

```
10 PRINT "Guten Tag, ich bin der CPC 6128"
20 PRINT "und bringe Ihnen grosse Neuigkeiten,"
30 PRINT "denn mein BASIC ist einfach Spitze!!!"
```

Geben Sie jetzt wieder

```
RUN
```

ein. Darauf läuft das Programm ab und gibt folgende Meldung(en) aus:

```
Guten Tag, ich bin der CPC 6128
und bringe Ihnen grosse Neuigkeiten,
denn mein BASIC ist einfach Spitze!!!
```

An diesem Beispiel sehen wir ganz deutlich, daß der Computer die Programmzeilen der Reihe nach abarbeitet. Wir haben hier die Zeilennummern 10, 20 und 30 benutzt, was zwar üblich ist, aber keinesfalls so sein muß. Wir hätten ebenso die Nummern 1,2,3 oder 967, 22345, 61120 verwenden können und damit das gleiche Ergebnis erzielt. Versuchen Sie es doch einmal!

Die meisten Programmierer verwenden jedoch Zehnerschritte, da sie dann immer noch die Möglichkeit haben, zwischen zwei Zeilen bis zu neun weitere Zeilen einzufügen. Wie das funktioniert, werden wir gleich sehen.

1.4.2 Programmierhilfen

Betrachten wir nochmals unser kleines Programm, das wir soeben eingegeben und ausgeführt haben. Wurde es nicht durch NEW gelöscht, steht es immer noch im Speicher und wir können es beliebig oft ablaufen lassen, erweitern, ändern, auf Diskette abspeichern usw.

Ist Ihnen eigentlich bei unserem einzeiligen und dreizeiligen Programm etwas aufgefallen? Wenn ja, dann gehören Sie schon zu den fortgeschrittenen Programmierern.

Zeile 10 (unser erstes Programm) ist nämlich auch in dem zweiten Programm exakt enthalten. Daher wäre es nicht unbedingt notwendig gewesen, den Arbeitsspeicher mit NEW zu löschen und dann die gleiche Zeile erneut einzugeben. Wir hätten nur die Zeilen 20 und 30 anhängen müssen, indem wir sie eintippen und die RETURN-Taste drücken. Dabei spielt es keine Rolle, ob Sie nach Zeile 10 zuerst Zeile 30 und dann Zeile 20 eingeben oder umgekehrt. Der Editor setzt jede eingegebene Zeile an ihre richtige Stelle. Versuchen Sie es einmal!

Zum Glück müssen wir uns bei der Frage, welche Zeilen wir eingegeben haben, nicht auf unser Gedächtnis verlassen. Mit dem LIST-Befehl wird das Programm auf dem Bildschirm ausgeben. Geben Sie jetzt LIST ein und drücken Sie die RETURN-Taste:

```
LIST
10 PRINT "Guten Tag, ich bin der CPC 6128"
20 PRINT "und bringe Ihnen grosse Neuigkeiten,"
30 PRINT "denn mein BASIC ist einfach Spitze!!!"
Ready
```

Diese drei Zeilen können wir noch bequem auf dem Bildschirm darstellen. Was geschieht aber, wenn wir ein langes Programm listen wollen?

Für den LIST-Befehl gibt es einige Zusatzangaben, die dafür sorgen, daß nur ein Teil des Programms gelistet wird. Versuchen Sie folgendes:

```
LIST 10
10 PRINT "Guten Tag, ich bin der CPC 6128"
Ready
```

Steht hinter LIST eine Zeilennummer, so wird nur die betreffende Zeile ausgegeben. Im folgenden Beispiel werden alle Zeilen zwischen 10 und 20 gelistet:

```
LIST 10-20
10 PRINT "Guten Tag, ich bin der CPC 6128"
20 PRINT "und bringe Ihnen grosse Neuigkeiten,"
Ready
```

Die Anweisung

```
LIST -20
```

hätte in diesem Fall die gleiche Wirkung, denn sie listet alle Programmzeilen mit den Nummern bis einschließlich 20. Andererseits besteht auch die Möglichkeit, von einer Zeilennummer bis zum Programmende zu listen, wie im folgenden Beispiel:

```
LIST 20-
20 PRINT "und bringe Ihnen grosse Neuigkeiten,"
30 PRINT "denn mein BASIC ist einfach Spitze!!!"
Ready
```

Falls Sie einen Drucker besitzen, können Sie das Listing auch ausdrucken. Schalten Sie den Drucker ein und geben Sie folgende Anweisung ein:

```
LIST #8
```

Jetzt wollen wir unser Programm durch Hinzufügen der Zeilen 15 und 40 noch etwas erweitern. Die Reihenfolge wurde dabei absichtlich vertauscht.

```
40 PRINT "Das zeigt Ihnen dieses MARKT&TECHNIK-Buch."
15 PRINT "- der neueste Computer von SCHNEIDER -"
```

Wenn Sie jetzt LIST eingeben, sind die Programmzeilen in der richtigen Reihenfolge aufgeführt, in der sie auch abgearbeitet werden:

```
10 PRINT "Guten Tag, ich bin der CPC 6128"
15 PRINT "- der neueste Computer von SCHNEIDER -"
20 PRINT "und bringe Ihnen grosse Neuigkeiten,"
30 PRINT "denn mein BASIC ist einfach Spitze!!!"
40 PRINT "Das zeigt Ihnen dieses MARKT&TECHNIK-Buch."
```

Auf die Ausführung dieses Programms wollen wir hier verzichten, da wir ja wissen, in welcher Reihenfolge der Text erscheint. Das einzige, was am optischen Erscheinungsbild stört, ist die Zeile 15. Wir können die Zeilen aber mit dem RENUM-Befehl in Zehnerschritten neu durchnumerieren. Geben Sie jetzt

```
RENUM
```

ein und listen Sie das Programm nochmals:

```
10 PRINT "Guten Tag, ich bin der CPC 6128"
20 PRINT "- der neueste Computer von SCHNEIDER -
30 PRINT "und bringe Ihnen grosse Neuigkeiten,"
40 PRINT "denn mein BASIC ist einfach Spitze!!!"
50 PRINT "Das zeigt Ihnen dieses MARKT&TECHNIK-Buch."
```

RENUM ohne Zusatz numeriert das Programm also in Zehnerschritten, beginnend mit Zeilennummer 10, durch. Soll z.B. die erste Zeilennummer nicht 10, sondern 100 lauten, geben Sie

```
RENUM 100
```

ein. Wünschen Sie darüber hinaus eine Schrittweite von 20, lautet die Anweisung

```
RENUM 100,,20
```

Wenn wir eine neue Programmzeile mit einer Nummer eingeben, die bereits existiert, ersetzt die neue Zeile die alte. Geben Sie dagegen nur eine Zeilennummer ein und drücken RETURN, wird die betreffende Zeile im Programm gelöscht, falls sie bereits existiert.

Wenn wir z.B. in unserem neu durchnumerierten Programm die Zeilen 20 bis 40 löschen möchten, geben wir

```
20 <RETURN>
30 <RETURN>
40 <RETURN>
```

ein. mit der DELETE-Anweisung können wir uns die Arbeit etwas erleichtern, da sie es ermöglicht, mehrere Zeilen gleichzeitig zu löschen. Wenn Sie jetzt

```
DELETE 20-40
```

eingeben, werden die Zeilen 10 bis 40 gleichzeitig gelöscht.

Die Anweisung

```
DELETE 30-
```

löscht alle Zeilen von Nummer 30 bis Programmende und die Anweisung

```
DELETE -40
```

löscht alle Zeilen vom Programmanfang bis einschließlich Zeile 40.

Eine weitere Programmierhilfe ist die automatische Zeilennumerierung, die mit der AUTO-Anweisung eingeschaltet wird. Die Anweisung

```
AUTO 1000,10
```

schaltet die automatische Zeilennumerierung ein. Die Zeilennummern beginnen bei 1000 und werden in Zehnerschritten fortgezählt. Wenn Sie allerdings nur AUTO ohne Zusatz eingeben, erhält die erste Zeile die Nummer 10, die zweite die Nummer 20 usw. Nach Drücken der RETURN-Taste erscheint jeweils die nächste Zeilennummer. Der Cursor steht bereits an der richtigen Stelle, so daß Sie nur noch den Programmtext eingeben und wieder RETURN drücken müssen usw. Ist das Programm fertig eingegeben, drücken Sie einfach die ESC-Taste und die automatische Zeilennumerierung ist wieder abgeschaltet.

1.4.3 Fehlerkorrektur

Wahrscheinlich fragen Sie sich jetzt, was zu tun ist, wenn Sie beim Eintippen einen Fehler gemacht haben. Eine Möglichkeit besteht darin, die fehlerhafte Zeile neu einzugeben und zu überschreiben. Dies ist allerdings eine sehr umständliche Methode, besonders wenn es darum geht, nur ein Zeichen auszubessern.

Der CPC 6128 bietet zwei Möglichkeiten an, Fehler in Programmzeilen zu korrigieren. Die eine Möglichkeit arbeitet mit einem zweiten Cursor, dem Copy-Cursor. Diese Methode ist allerdings ziemlich kompliziert und nicht zu empfehlen, da sie auch leicht zu Fehlern führt. Aus diesem Grunde wollen wir uns nicht näher mit ihr befassen.

Wesentlich einfacher und sicherer arbeitet der EDIT-Befehl, mit dem wir uns die fehlerhafte Zeile auf den Bildschirm holen und dann leicht verbessern oder ergänzen können. Nehmen wir einmal an, Sie haben folgende Programmzeile falsch eingegeben:

```
10 PRINT "CONPUTER"
```

Um nun das N durch ein M zu ersetzen, geben Sie zunächst

```
EDIT 10
```

ein. Die Zeile erscheint jetzt auf dem Bildschirm, wobei der Cursor am Zeilenanfang steht. Mit Hilfe der Cursorsteuertasten setzen Sie den Cursor auf das fehlerhafte Zeichen N. Nun drücken Sie einmal die CLR-Taste, wodurch dieses Zeichen gelöscht wird und die restliche Zeile um ein Zeichen nach links aufrückt. Jetzt müssen Sie nur noch das N gegen ein M austauschen und die RETURN-Taste drücken, womit die Zeile richtig im Arbeitsspeicher abgelegt ist.

Wenn in Zeile 10 ein Zeichen fehlt wie z.B. bei

```
10 PRINT "COPUTER"
```

fahren Sie nach Ausführung des EDIT-Befehls mit dem Cursor auf das P, geben das M ein und drücken die RETURN-Taste.

Nehmen wir uns noch einmal die Zeile vor, die jetzt aber so aussieht:

```
10 PRINT "COMPUUUUTER"
```

Hier fahren wir mit dem Cusor auf das erste U und drücken dreimal die CLR-Taste. Eine weitere Möglichkeit besteht darin, den Cursor auf das letzte U zu setzen und dreimal die DEL-Taste zu drücken.

Fassen wir noch einmal zusammen: Die fehlerhafte Zeile ist zunächst mit dem EDIT-Befehl zu listen. CLR löscht das Zeichen, auf dem der Cursor steht, und DEL löscht das Zeichen links neben dem Cursor. In beiden Fällen rückt die Zeile hinter dem Cursor um ein Zeichen nach links. Soll ein Zeichen eingefügt werden, muß der Cursor auf das Zeichen, das später rechts von dem einzufügenden steht, gesetzt und anschließend das neue Zeichen eingegeben werden. Nach Beendigung der Korrektur drücken Sie die RETURN-Taste, wodurch die Zeile ins Programm übernommen wird.

1.4.4 Programme sichern und laden

Wenn Sie ein Programm auf Diskette abspeichern möchten, benötigen Sie zunächst einmal eine formatierte Diskette. Falls Sie diese nicht zur Hand haben, nehmen Sie eine fabrikneue Diskette und formatieren sie, wie in Kapitel 6 beschrieben.

Mit SAVE können Sie BASIC-Programme abspeichern und mit LOAD wieder laden. Dazu müssen Sie Ihrem Programm einen Namen geben, der nicht länger als acht Zeichen sein darf. Hier ein Beispiel:

```
SAVE "TEST"
```

legt ein BASIC-Programm unter dem Namen "TEST" auf Diskette ab. Wenn Sie das gleiche Programm wieder laden wollen, geben Sie

```
LOAD "TEST"
```

ein. Soll das Programm geladen und anschließend gleich ausgeführt werden, muß die Anweisung

```
RUN "TEST"
```

lauten.

Der folgende Befehl listet sämtliche Dateien, die sich auf einer Diskette befinden, auf dem Bildschirm:

```
CAT
```

Die Dateinamen erscheinen hierbei in alphabetischer Reihenfolge.

Mit den oben beschriebenen Anweisungen sollten Sie in der Lage sein, einfache Floppy-Operationen durchzuführen. In Kapitel 6 werden wir uns aber noch ausführlich mit der Floppy beschäftigen.

1.4.5 Bildschirmmodi

Den CPC 6128 können Sie mit einem 20-, 40- und 80-Zeichen-Bildschirm betreiben. Nach dem Einschalten befindet er sich immer im 40-Zeichen-Modus. Folgende Anweisungen schalten den Bildschirm um:

MODE 0 auf 20 Zeichen pro Zeile
MODE 1 auf 40 Zeichen pro Zeile
MODE 2 auf 80 Zeichen pro Zeile

In Kapitel 9 erfahren Sie mehr über die einzelnen Bildschirmmodi einschließlich der Farben, die mit ihnen darstellbar sind.

1.4.6 Funktionstasten

Neben der eigentlichen Schreibmaschinentastatur befindet sich noch ein Zifferntastenblock mit den Tasten f0 bis f9. Wenn Sie diese Tasten drükken, erscheinen normalerweise die angegebenen Ziffern. Es besteht aber auch die Möglichkeit, sie als Funktionstasten zu definieren.

Funktionen können hier beliebige Anweisungen oder BASIC-Schlüsselwörter wie LIST, LOAD oder SAVE sein.

Mit folgender Anweisung können Sie den Tasten eine neue Funktion geben:

KEY (Tastennummer), Anweisung(en)

Doch zunächst die entsprechenden Tastennummern:

Nummer	Taste
128	f0
129	f1
130	f2
131	f3
132	f4
133	f5
134	f6
135	f7
136	f8
137	f9
138	. (Punkttaste)

Angenommen, wir wollen durch Drücken der Taste f0 den LIST-Befehl ausführen. Dazu müssen wir zunächst mit

```
KEY 128,"LIST"+CHR$(13)
```

diese Taste neu definieren. Außer LIST enthält die Anweisung noch CHR$(13), was dem Drücken der RETURN-Taste entspricht. Ohne den Zusatz würde lediglich das Wort LIST auf dem Bildschirm erscheinen, worauf Sie die RETURN-Taste von Hand drücken müßten.

2 Die ersten Gehversuche

Im letzten Kapitel haben wir uns mit den einfachsten Grundbegriffen über den Aufbau von BASIC-Programmen beschäftigt. Sie wissen jetzt, was die Programmiersprache BASIC ist, können im Direkt- und im Programmodus einfache Meldungen ausgeben und haben einige nützliche Hilfsmittel zur Erstellung, Korrektur und Sicherung von Programmen kennengelernt. In diesem Kapitel erfahren Sie nähere Einzelheiten über Datentypen, Variablen und die wichtigsten Ein- und Ausgabeoperationen und lernen, damit umzugehen.

2.1 Datentypen

Der Computer kann eine Vielzahl von Daten verarbeiten, vorausgesetzt, er kennt alle dazu erforderlichen Informationen. Diese können entweder im BASIC-Programm selbst oder auf einem Datenträger (z.B. Kassette, Diskette) vorgegeben sein.

Der Begriff "Daten" ist sehr umfassend. Wenn Sie z.B. das Telefonbuch aufschlagen, finden Sie darin die Daten sämtlicher Fernsprechteilnehmer, wie Nach- und Vorname, Straße, Hausnummer und natürlich die Rufnummer. Darüberhinaus sind die Teilnehmer in alphabetischer Reihenfolge geordnet, damit Sie sie leicht herausfinden können.

Ein ähnliches Verzeichnis, wie es das Telefonbuch darstellt, können Sie durchaus auch in Ihrem Computer speichern und verarbeiten, selbst wenn es nur die Adressen und Telefonnummern Ihrer Freunde und Bekannten enthält. Eine solche Ansammlung verschiedener Daten, gleich welcher Art, bezeichnet man als Datei.

Wenn Sie Ihren Computer im Privatbereich nutzen wollen, können Sie sich z.B. ein Programm schreiben, das Ihr Haushaltsgeld verwaltet und die Einnahmen, Ausgaben usw. in einer Datei ablegt; oder ein Programm, das Ihre Schallplattensammlung oder die Autokosten auflistet. Im professionellen Bereich laufen heutzutage Buchhaltung und Lagerhaltung meist über die elektronische Datenverarbeitung mit teilweise riesigen Dateien, die auch als Datenbanken bezeichnet werden.

Auf die eigentliche Dateiverwaltung werden wir später noch zu sprechen kommen. Im Augenblick geht es nur darum, welche Art von Daten oder Datentypen Ihr Computer verarbeiten kann.

2.1.1 Strings

Bereits im letzten Kapitel haben wir mit Zeichenketten gearbeitet, die man in der Fachsprache als Strings bezeichnet. Die Anweisung

```
PRINT "Guten Tag, ich bin der CPC 6128"
```

gibt beispielsweise den String

```
Guten Tag, ich bin der CPC 6128
```

auf dem Bildschirm aus, was durch das Befehls- oder BASIC-Schlüsselwort PRINT ausgelöst wird. Hier einige weitere Beispiele für Strings:

```
"Hans Schmidt"
"Augustenstrasse 48"
"VIELE GRUESSE VOM COMPUTER"
"Pfeffer"
"Tomaten"
"123.456"
"Neue Computerbuecher sind Spitze!!!"
```

Strings sind in der Regel eine Aneinanderreihung von Buchstaben, Satzzeichen und Ziffern, wie Sie sie auf einer Schreibmaschine tippen. Damit der Computer erkennt, wo der String beginnt und wo er endet, ist er in Anführungszeichen eingeschlossen. Aus diesem Grund dürfen in einem String selbst keine Anführungszeichen auftreten. Soll ein Wort besonders hervorgehoben werden, verwendet man stattdessen meist ein Apostroph. Hier ein Beispiel:

Falsch:

"Sie sagte: "Mein Name ist Hase, ich weiss von nichts"."

Richtig:

"Sie sagte: 'Mein Name ist Hase, ich weiss von nichts.'"

Als wir im letzten Kapitel lernten, Programme abzuspeichern und zu laden, haben wir ebenfalls schon mit Strings gearbeitet. So ist beispielsweise in der Anweisung

LOAD "TEST"

der Programmname "TEST" ein String. Er wird von dem Schlüsselwort LOAD benötigt, um das richtige Programm mit dem Namen TEST von Diskette zu laden.

Aus programmiertechnischen Gründen verwendet man manchmal auch Strings, die keine Zeichen oder nur Leerzeichen (Blanks) enthalten. Z.B. ist

""

ein Nullstring, weil er keine Zeichen enthält und

" "

ein Leerstring mit 14 Blanks, der nur an den Anführungszeichen zu erkennen ist.

Jedes Zeichen, das in einem String dargestellt werden kann, besitzt einen entsprechenden Zahlencode, der im Computer gespeichert ist. Dieser Code ist genormt und heißt ASCII-Code. So entspricht z.B. der Buchstabe A dem ASCII-Code 65, der Buchstabe B dem ASCII-CODE 66 usw. (Siehe dazu auch ASCII-Code-Tabelle im Anhang!)

Entdeckt nun der Computer hinter dem Schlüsselwort PRINT ein Anführungszeichen, so weiß er, daß er einen String auf dem Bildschirm ausgeben soll. Ist der Wert in der nächsten Speicherzelle (Byte) eine 65, gibt er folglich den Buchstaben A aus.

Fast alle Computer arbeiten heute mit dem ASCII-Code. Dies hat den Vorteil, daß sie Daten untereinander austauschen können.

2.1.2 Fließkommazahlen

Fließkommazahlen sind neben den Strings ein Datentyp, mit dem der Computer echt "rechnen", d.h. mathematische Operationen durchführen kann. Fließkommazahlen werden deshalb vom Computer in einer anderen Form intern verwaltet als Strings.

Vielleicht fragen Sie sich, warum unter den Beispielen für Strings auch "123.456" aufgeführt ist. Zwar ist der Inhalt dieses Strings offensichtlich ein Zahlenwert, der Computer betrachtet ihn aber nicht als Zahl, da er in Anführungszeichen eingeschlossen ist. Der String enthält lediglich die äquivalenten ASCII-Zeichen für 1, 2, 3, 4, 5 und 6. In diesem Format ist der Computer nicht in der Lage, arithmetische Berechnungen durchzuführen.

Geben wir also 123.456 ohne Anführungszeichen ein, so wird dieser Wert als Zahl betrachtet und intern in einem speziellen Binärformat verwaltet. Zahlen im Sinne des Computers dürfen nur aus den Ziffern 0 bis 9 und eventuell einem Vorzeichen und einem Dezimalpunkt (kein Komma!) bestehen. Sehr große und sehr kleine Zahlen werden in Exponentialschreibweise angegeben, wobei der Wert durch ein großes E mit einem zweistelligen Zehnerexponent abgeschlossen wird. Hier einige Beispiele für zulässige Zahlenwerte:

```
1.5
652
-1546.98006
.044736524
1.54E+21
5.987543E-07
```

Unzulässig dagegen sind z.B.:

7,52+	richtig:	7.52
-65,765,879.8	richtig:	-65765879.8
34.652,87	richtig:	34652.87
3.43-	richtig:	-3.43

Außer Strings können mit PRINT auch Zahlen auf dem Bildschirm ausgegeben werden:

```
PRINT 45.789
 45.789
Ready
```

oder

```
PRINT -27
-27
Ready
```

Der CPC 6128 stellt Fließkommazahlen in einem möglichst optimalen Format dar. Deshalb werden besonders große oder kleine Werte, die nicht mit neun signifikanten Ziffern dargestellt werden können, in Exponentialschreibweise ausgegeben, wie z.B.

```
PRINT 1356789776655443
 1.35679E+15
Ready
```

oder

```
PRINT 0.0000012345
 1.2345E-06
Ready
```

Der CPC 6128 kann nur Zahlenwerte mit maximal 9 signifikanten Ziffern verarbeiten. Andere Zahlen werden auf insgesamt neun Ziffern gerundet, in Exponentialschreibweise nur auf sechs. Beachten Sie auch, daß der zulässige Wertebereich nur zwischen ca. 1E-39 und 1E+38 liegt. Kleinere Werte werden als 0 wiedergegeben, bei größeren erfolgt eine Überlauf-Fehlermeldung, wie im folgenden Beispiel:

```
PRINT 1E+90
Overflow
Syntax error
Ready
```

2.1.3 Integerzahlen

Neben den Fließkommazahlen gibt es noch die Integerzahlen. Sie dürfen nur ganzzahlige Werte im Bereich zwischen -32768 und +32767 enthalten.

Integerzahlen benötigen weniger Speicherplatz als Fließkommazahlen und eignen sich darüber hinaus auch für viele zeitkritische Rechenoperationen, sofern der zulässige Zahlenbereich nicht überschritten wird. Für uns wer-

den sie aber erst an späterer Stelle interessant, wenn wir uns näher mit Variablen und Variablennamen befassen.

2.2 Rechnen mit BASIC

In BASIC ist das Rechnen mit Zahlen sehr einfach. Die Eingabe erfolgt in ähnlicher Form wie in der Algebra, wie wir es aus der Schule kennen.

Mit PRINT können wir auch arithmetische Ausdrücke wie z.B. 4 + 9 oder 65 - 3 auswerten. Das Zeichen, das die betreffende Operation angibt, heißt Operator. In diesem Fall dient der Operator + zur Addition und der Operator / zur Division. Die Werte, mit der die Operation stattfinden soll, heißen Operanden. Das Ergebnis, das man durch Ausführung der Operation erhält, bezeichnet man als Wert der Operation.

Entdeckt nun der Computer hinter einer PRINT-Anweisung eine arithmetische Operation, so führt er sie aus und gibt deren Wert, also das Ergebnis, auf dem Bildschirm aus. Hier einige Beispiele:

```
PRINT 3 + 4
 7
Ready

PRINT 100 - 2
 98
Ready

PRINT -10/3
-3.33333333
Ready

PRINT 45 * 5
 225
Ready
```

Der Vollständigkeit halber wollen wir uns hier auch kurz mit den wissenschaftlichen Funktionen befassen. Beginnen wir mit der Exponentiation:

```
PRINT 3^2
 9
Ready
```

In diesem einfachen Fall wären wir auch mit einer Multiplikation von 3 * 3 zum gleichen Ergebnis gekommen. Wir sind aber nicht nur auf geradzahlige Exponenten beschränkt, wie das folgende Beispiel zeigt:

```
PRINT 45^1.63
 495.14956
Ready
```

Umgekehrt können wir mit SQR auch die Quadratwurzel ziehen:

```
PRINT SQR(2)
 1.41421356
Ready
```

Folgende Exponentiation führt zum gleichen Ergebnis, da die Quadratwurzel einem Exponenten von 0.5 entspricht:

```
PRINT 2^.5
 1.41421356
Ready
```

Hier noch ein kurzer Überblick über die trigonometrischen und logarithmischen Funktionen. X steht dabei jeweils für das Argument:

SIN(X)	Sinus
COS(X)	Cosinus
TAN(X)	Tangens
ATN(X)	Arcustangens
EXP(X)	Potenz zur Zahl e
LOG(X)	natürlicher Logarithmus
LOG10(X)	Zehnerlogarithmus

Zu bemerken ist noch, daß der Computer für trigonometrische Funktionen standardmäßig Argumente im Bogenmaß annimmt. Es besteht jedoch die Möglichkeit, ihn auf Gradmaß umzuschalten, was mit der Anweisung DEG geschieht. Jetzt müssen alle Winkel für trigonometrische Berechnungen im Gradmaß angegeben werden. Die Anweisung RAD schaltet wieder auf Bogenmaß zurück.

Als Beispiel wollen wir hier den Sinuswert für einen Winkel von 45 Grad berechnen, was folgendermaßen geschieht:

```
DEG
PRINT SIN(45)
 .707106781
Ready
```

Übrigens ist die Zahl Pi (3.14...), die oft für geometrische und trigonometrische Berechnungen benötigt wird, im CPC 6128 fest eingespeichert und kann unter dem Variablennamen PI aufgerufen und in Berechnungen mit einbezogen werden. Im folgenden Beispiel wollen wir den Umfang eines Kreises mit dem Durchmesser 5 bestimmen:

```
PRINT 5 * PI
 15.7079633
Ready
```

In BASIC können Sie auch arithmetische Ausdrücke berechnen, die mehrere Operationen erfordern, wie z.B.

```
PRINT 3 + 4 + 19 - 10
 16
Ready
```

und

```
PRINT 7 + 2*9 - 6/3
 23
Ready
```

Die Berechnungen werden von links nach rechts durchgeführt, wobei folgende Prioritäten zu beachten sind, die wir auch in der Algebra vorfinden:

+ und -	Addition und Subtraktion	(niedrigste Priorität)
* und /	Multiplikation und Division	
^	Potenzierung	(höchste Priorität)

Für unser letztes Beispiel bedeutet das, daß zur Zahl 7 das Produkt von 2*9 = 18 addiert wird. Von diesem Ergebnis wird dann der Quotient 6/3 = 2 subtrahiert; das führt zum Endergebnis von 23.

BASIC kann auch Klammerrechnungen ausführen, mit denen die Prioritäten umgangen werden können. Dabei wird jeweils der Klammerinhalt vorrangig ausgewertet.

Wir wollen uns nochmals das letzte Beispiel ansehen, jetzt aber etwas umgeformt:

```
PRINT (7+2) * (9-6) / 3
 9
Ready
```

Hier werden zunächst die beiden Klammerinhalte 9 bzw. 3 miteinander multipliziert und dann das Produkt durch 3 dividiert.

2.3 Weitere Möglichkeiten mit PRINT

Bisher haben wir mit PRINT einen String oder eine Zahl auf dem Bildschirm ausgegeben und schließlich die Auswertung eines arithmetischen Ausdrucks mit eingeschlossen. Darüber hinaus kann man mit PRINT noch einige nützliche Ausgabetechniken realisieren.

Bei den besprochnenen Beispielen gab jede PRINT-Anweisung immer nur einen Wert aus. Es besteht jedoch auch die Möglichkeit, den Inhalt mehrerer Strings in einer Bildschirmzeile darzustellen. Um dies zu demonstrieren, schreiben wir am besten ein kleines BASIC-Programm:

```
10 PRINT "zu"
20 PRINT "sam"
30 PRINT "men"
```

Nachdem Sie dieses Programm mit RUN gestartet haben, erscheinen die drei Strings in der uns bereits bekannten Form:

```
zu
sam
men
Ready
```

Jetzt ändern wir das Programm geringfügig ab, indem wir Zeile 10 und 20 mit einem Semikolon abschließen:

```
10 PRINT "zu";
20 PRINT "sam";
30 PRINT "men"
```

Nach Ausführung dieses Programms erhalten wir

```
zusammen
Ready
```

Wir sehen also, daß verschiedene Strings aus verschiedenen PRINT-Anweisungen aneinandergefügt werden, wenn sie jeweils mit einem Semikolon abschließen. Das gleiche Resultat können wir aber noch einfacher erhalten:

```
PRINT "zu";"sam";"men"
zusammen
Ready
```

Hier stehen die einzelnen Strings in einer PRINT-Anweisung, wobei sie jeweils durch Semikolons getrennt sind.

Ihr CPC 6128 bietet außerdem die Möglichkeit der festen Tabellierung, wobei die Werte im Abstand von jeweils 13 Zeichen beginnen. Zu diesem Zweck müssen sie durch Kommas getrennt sein:

```
PRINT "aus","ein","an","der"
aus           ein           an            der
Ready
```

Ein Tabulator mit variablen Abständen steht mit der TAB-Anweisung zur Verfügung:

```
PRINT TAB(4)"abc";TAB(20)"def"
   abc              def
Ready
```

Der Klammerwert innerhalb der TAB-Anweisung gibt die jeweilge Position in der Zeile an, beginnend mit Position 1 am linken Rand. Im vorliegenden Beispiel beginnt der erste String bei Position 4 und der zweite bei Position 20, Sämtliche PRINT-Anweisungen gelten auch für Zahlen vom Datentyp Fließkomma oder Integer bzw. für Werte von berechneten Ausdrücken, wie wir sie bereits kennengelernt haben. Bei positiven Werten wird statt des Vorzeichens immer ein Leerzeichen freigelassen:

```
PRINT 5,-4,3
 5            -4            3
Ready
```

Steht zwischen den Zahlen allerdings ein Semikolon, so bleibt ein zusätzliches Leerzeichen frei, um die Übersichtlichkeit zu erhöhen:

```
PRINT 5;-4;3
 5 -4  3
Ready
```

Abschließend noch ein Programm, das für verschiedene Schüler den Namen zusammen mit der Punktzahl und der Note einer Klassenarbeit ausgibt:

```
10 PRINT "Name","Punkte","Note"
20 PRINT
30 PRINT "Hans",45,3
40 PRINT "Nicole",40,4
50 PRINT "Stefan",50,2.7
60 PRINT "Christian",38,4.3
70 PRINT "Ursula",60,2
```

Nach Eingabe von RUN erscheint folgende Liste auf dem Bildschirm:

```
Name          Punkte        Note

Hans          45            3
Nicole        40            4
Stefan        50            2.7
Christian     38            4.3
Ursula        60            2
Ready
```

2.4 Variablen

Wir wissen bereits, daß der Computer einen RAM-Speicher besitzt, d.h. einen flüchtigen Speicher, dessen Inhalt sich ständig ändern kann und beim Ausschalten des Computers verlorengeht. Ein Großteil des RAM-Speichers dient zur Ablage von BASIC-Programmen und allen dazugehörigen Informationen.

Weiter haben wir bereits einfache Programme kennengelernt, die in numerierte BASIC-Zeilen aufgeteilt sind. Unsere Beispiele im Direkt- und im Programmodus behandelten bisher ausschließlich die Ausgabe von Strings auf dem Bildschirm und die Berechnung arithmetischer Ausdrücke. Die Werte, die zu berechnen bzw. auszugeben waren, standen unmittelbar hinter der PRINT-Anweisung.

Auch kennen wir bereits die verschiedenen Datentypen, nämlich Strings für Zeichenketten, Fließkommazahlen und Integerzahlen. Intern verwaltet der Computer jeden dieser Datentypen auf unterschiedliche Weise. So werden Strings in ASCII-Zeichen, Fließkommazahlen in einem Binärformat und Integerzahlen in einer kurzen Binärschreibweise abgelegt. Zu diesem Zweck muß der Computer für jeden einzelnen Wert einen Speicherbereich reservieren.

Betrachten wir nochmals unser allererstes Beispiel, das wir diesmal allerdings im Programmodus schreiben wollen:

```
10 PRINT "Guten Tag, ich bin der CPC 6128"
```

Wenn Sie diese Programmzeile eintippen und die RETURN-Taste drücken, ordnet der Computer dieser Zeile einen bestimmten Speicherplatz zu, in dem auch die auszugebende Meldung enthalten ist. Sofern wir den Text nicht ändern, erscheint jedesmal bei Ausführung von Zeile 10 dieselbe Meldung. Wenn wir den Text nun anstelle von einem Mal zehnmal hintereinander ausgeben wollen, brauchen wir ein Programm, das zehn Zeilen mit jeweils der gleichen PRINT-Anweisung enthält. Dabei könnten die Zeilen z.B. von 10 bis 100 in Zehnerschritten durchnumeriert werden - eine wahrhaftig umständliche Angelegenheit!

BASIC hat jedoch eine Einrichtung, bei der wir den String nur einmal definieren müssen und ihn dann mehrfach hintereinander mit einer wesentlich kürzeren Anweisung ausgeben können. Dazu ordnen wir ihn einer Variablen zu, die später nach Belieben aufgerufen werden kann:

```
10 LET a$ = "Guten Tag, ich bin der CPC 6128"
20 PRINT a$
30 PRINT a$
40 PRINT a$
```

Wenn Sie dieses Programm ausführen, erscheint dreimal die gleiche Meldung auf dem Bildschirm. In Zeile 10 haben wir den Text in einer Stringvariablen mit dem Namen a$ abgelegt, was mit der Zuordnungsanweisung LET geschieht. Der Computer führt ein internes Verzeichnis, in dem genau angegeben ist, an welcher Stelle sich der Inhalt von a$ im Speicher befindet und wieviele Zeichen der String umfaßt. Jedesmal, wenn nun bei der Abarbeitung des Programms a$ auftaucht, sucht der Computer diese Variable im Verzeichnis und bearbeitet ihren Inhalt gemäß der entsprechenden Anweisung. In unserem Beispiel wird sie insgesamt dreimal durch PRINT aufgerufen und auf dem Bildschirm ausgegeben.

Ähnlich, wie bei der Zuweisung eines Strings zu einer Variablen kann auch bei numerischen Ausdrücken verfahren werden, wie das folgende Beispiel zeigt:

```
10 LET a = 3
20 LET b = 5
30 LET c = a * b
40 PRINT c
```

Nach Ausführung dieses Programms erscheint als Ergebnis die Zahl 15 auf dem Bildschirm. Wie es dazu kommt, wollen wir nun einmal näher untersuchen.

In Zeile 10 erhält die Fließkommavariable a den Wert 3 und in Zeile 20 die Fließkommavariable b den Wert 5. Beide Werte werden im internen Zahlenformat im Speicher abgelegt und können mit Hilfe der Variablennamen a bzw. b wieder aufgefunden werden. Dies geschieht in Zeile 30. Hier entsteht eine neue Variable c, nachdem die Inhalte der Variablen a und b miteinander multipliziert wurden. Variable c enthält nun das Produkt aus a und b und hat ebenfalls einen festen Speicherbereich. Schließlich wird ihr Inhalt in Zeile 40 aufgerufen und auf dem Bildschirm ausgegeben.

In diesem Beispiel haben wir die Zuordnung eines Wertes zu einer Variablen mit LET durchgeführt. Die LET-Anweisung stammt noch aus der Anfangszeit von BASIC und ist bei modernen BASIC-Dialekten nicht mehr erforderlich. Auch wir wollen sie in diesem Buch nicht weiter verwenden und lassen sie einfach weg. Dann sieht unser Beispielprogramm folgendermaßen aus:

```
10 a = 5
20 b = 3
30 c = a * b
40 PRINT c
```

Hier nun noch etwas zu den Variablennamen. Wir kennen bereits die drei Datentypen String, Fließkomma und Integer, von denen jeder einen bestimmten Variablennamentyp besitzt. Wir wissen, daß Variablennamen für Strings immer mit einem $-Zeichen enden, das bei den Fließkommavariablen fehlt. Dies ist auch genau das Unterscheidungsmerkmal der beiden Typen. Namen für Integervariablen enden mit einem %-Zeichen. Hier nochmals eine Zusammenfassung:

Variablentyp	Merkmal
String	$
Fließkomma	keines
Integer	%

Jeder Variablenname des gleichen Typs darf in einem Programm nur einmal vorkommen. Eine erneute Zuordnung mit einem anderen Wert ist jedoch möglich, wobei der alte Wert verlorengeht. Hier ein Beispiel:

```
10 a = 2
20 a = 100
30 a = 3 * 5
40 PRINT a
```

In Zeile 10 erhält die Variable a den Wert 2 und in Zeile 20 den Wert 100, wobei der Wert 2 aus der Zuordnung in Zeile 10 verlorengeht. Das gleiche geschieht nochmals in Zeile 30, wo die Variable a das Produkt aus 3 und 5, also 15 erhält.

Anders ist es, wenn der gleiche Variablenname in den verschiedenen Typen erscheint, wie z.B.:

```
10 a = 6.78
20 a% = 2
30 a$ = "Hallo!"
40 PRINT a
50 PRINT a%
60 PRINT a$
```

Nach Ausführung des Programms erscheint:

```
 6.78
 2
Hallo!
Ready
```

Wir sehen also, daß der gleiche Variablenname unter den verschiedenen Typenbezeichnungen existieren kann.

Variablennamen dürfen im CPC-BASIC, im Gegensatz zu anderen BASIC-Versionen, bis zu 40 Zeichen umfassen. Allerdings muß das erste Zeichen auf jeden Fall aus einem Buchstaben zwischen a und z bestehen. Alle anderen Zeichen sind nicht zwingend und können entweder ebenfalls aus Buchstaben oder aus den Ziffern 0 bis 9 bestehen. Hier ein paar Beispiele zulässiger Variablennamen:

ab	zip1$	cocacola	spacelab7%
type7ok$	bingbeng%	r$	artikelnummer

Beachten Sie bitte, daß die Variablennamen nicht durch Leerzeichen unterbrochen sind oder sonstige Füllzeichen, wie z.B. &,(,),+,- enthalten.

Sie sollten den Variablen möglichst sinnvolle Namen zuteilen, was gleichzeitig der Übersichtlichkeit des Programms dient. Für ein Lagerverwaltungsprogramm könnten sie z.B. so aussehen:

auftragnummer%	artikel$	einzelpreis	gesamtpreis
mehrwertsteuer	wareneingang		

Beachten Sie, daß mit der Länge der Variablennamen auch die Verarbeitungsgeschwindigkeit des Computers abnimmt, da dieser zum Auffinden der Namen mehr Zeichen vergleichen muß als bei kurzen Namen. Darüber hinaus benötigt das Programm auch mehr Speicherplatz. Bei speicher- oder zeitkritischen Programmen sollten Sie sich deshalb auf kurze Variablennamen mit einem oder zwei Zeichen Länge beschränken.

Es ist keinesfalls zwingend, daß Variablennamen nur aus Kleinbuchstaben bestehen müssen. Der BASIC-Interpreter behandelt eine Variable "Artikelnummer" ebenso wie "artikelnummer" oder "ARTIKELNUMMER" und macht keinen Unterschied zwischen Klein- und Großbuchstaben. In diesem Buch wollen wir Variablennamen aber immer klein schreiben, um sie von den BASIC-Schlüsselwörtern abzuheben. Diese werden nämlich, wenn sie in Kleinschrift eingegeben werden, automatisch in Großbuchstaben umgewandelt, sofern sie korrekt geschrieben wurden. Erscheinen sie später im Listing klein, liegt ein Eingabefehler vor, den Sie somit sofort erkennen können.

Auf Ihren CPC 6128 sind jedoch einige Variablennamen für bestimmte Zwecke reserviert und dürfen nicht anderweitig verwendet werden:

PI	enthält die Zahl Pi = 3.14159...
EOF	zur Kennzeichnung des Dateiendes
ERR, ERL	zur Fehlerbehandlung
DERR	zur Abfrage von Fehlern bei Diskettenoperationen

Als weiteres sind auch Variablennamen unzulässig, die aus BASIC-Schlüsselwörtern bestehen, wie z.B:

```
DATA    BORDER    EDIT
```

Achten Sie darauf, daß Daten- und Variablentyp übereinstimmen, d.h. daß Sie einer Stringvariablen keinen numerischen Wert zuweisen und umgekehrt. Wäre dies der Fall, wie im folgenden Beispiel

```
a$ = 5.25
```

entsteht die Fehlermeldung

```
Type mismatch
```

Die richtige Eingabe wäre gewesen:

```
a = 5.25
```

Im CPC-BASIC gibt es die Anweisungen DEFINT, DEFSTR und DEFREAL, die Variablen eine andere Typenkennzeichnung geben. Wir wollen hier aber nicht näher darauf eingehen, da dies nur Verwirrung stiften würde.

2.5 Die INPUT-Anweisung

Wir wissen bereits, daß wir Variablen bestimmte Werte zuordnen können. Bisher geschah dies durch Zuordnungsanweisungen mit oder ohne LET.

Oft ist es jedoch erforderlich, während des Programmablaufs Werte anzufordern, die über die Tastatur eingegeben und dann einer Variablen zugeordnet werden. Dies geschieht mit der INPUT-Anweisung. Probieren Sie einmal das folgende kleine Programm aus:

```
10 INPUT a
20 PRINT a
```

Wenn Sie dieses Programm mit RUN starten, erscheint auf dem Bildschirm ein Fragezeichen und dahinter der blinkende Cursor. Geben Sie jetzt eine beliebige Fließkommazahl ein und drücken Sie die RETURN-Taste wie im folgenden Beispiel:

```
? 145     <RETURN>
 145
Ready
```

Hierbei fordert die INPUT-Anweisung in Zeile 10 eine Fließkommazahl an, die, nachdem sie eingegeben und durch RETURN abgeschlossen wurde, der Fließkommavariablen a zugeordnet wird. Zur Kontrolle steht die PRINT-Anweisung in Zeile 20, die die betreffende Zahl wieder auf dem Bildschirm ausgeben muß.

INPUT ist nicht nur auf Fließkommawerte beschränkt, sondern funktioniert mit jedem Datentyp gleichermaßen. Hier einige weitere Beispiele:

```
INPUT b%
INPUT xy$
INPUT a, b$, c, dy
```

Betrachten wir einmal das letzte Beispiel genau, bei dem insgesamt vier Werte für verschiedene Variablen angefordert werden:

```
10 INPUT a, b$, c, dy
20 PRINT a
30 PRINT b$
40 PRINT c
50 PRINT dy

RUN
? 2,prima,25,3.14
 2
prima
 25
 3.14
ready
```

Hier werden nacheinander Werte für die Fließkommavariable a, die Stringvariable b$ sowie die Fließkommavariablen c und dy angefordert. Bei der Eingabe müssen sie durch Kommas getrennt werden, damit der BASIC-Interpreter erkennt, wo sie beginnen und enden.

Strings müssen hier nicht unbedingt in Anführungszeichen eingegeben werden. Wenn Sie allerdings statt einem reinen Zahlenwert unzulässige Zeichen,wie z.B. Buchstaben eingeben, gibt der Computer

```
?Redo from start
?
```

aus. Der erfaßte Wert wird ignoriert und das untere Fragezeichen fordert Sie erneut zur Eingabe auf.

Wenn nun während des Progtrammablaufs plötzlich ein Fragezeichen und der Cursor auftauchen, weiß der Benutzer zwar, daß er einen Wert eingeben soll, er weiß jedoch häufig nicht, um welchen Wert es sich dabei handelt. BASIC sieht deshalb die Möglichkeit vor, daß vor der Anforderung zunächst ein Hinweis auf dem Bildschirm erscheint:

```
10 INPUT "Bitte beliebige Zahl eingeben"; z
20 PRINT "Sie haben soeben die Zahl"; z; " eingegeben"

RUN
Bitte beliebige Zahl eingeben? 100
Sie haben soeben die Zahl 100 eingegeben
Ready
```

Beachten Sie die PRINT-Anweisung in Zeile 20, in der das Programm insgesamt drei Werte ausgibt. Zunächst handelt es sich dabei um den String "Sie haben soeben die Zahl", dann um die eingegebene Zahl z und schließlich den zweiten String "eingegeben".

Zum Schluß wollen wir noch ein kleines Programm betrachten, das das Volumen eines Quaders berechnet. Dies geschieht nach der Formel

Volumen = Länge * Breite * Höhe

Für das BASIC-Programm wählen wir hier folgende Variablennamen:

v für Volumen
l für Länge
b für Breite
h für Höhe

Hier nun das komplette Programm:

```
10 REM Volumenberechnung
20 INPUT "Laenge"; l
30 INPUT "Breite"; b
40 INPUT "Hoehe"; h
50 v = l * b * h
60 PRINT "Das Volumen betraegt"; v
```

Dieses Programm enthält die REM-Anweisung. REM steht für englisch "Remark" oder Kommentar. Entdeckt der Computer eine REM-Anweisung, so übergeht er sie und ignoriert alle nachfolgenden Zeichen. REM-Zeilen dienen in erster Linie der Übersichtlichkeit von Programmen und können jeden beliebigen Text enthalten. In unserem Fall zeigt die REM-Anweisung in Zeile 10 an, daß wir es mit einem Programm zur Volomenberechnung zu tun haben. Ein Programm kann aber beliebig viele REM-Anweisungen an beliebigen Stellen enthalten.

In den Zeilen 20 bis 40 werden nacheinander Länge, Breite und Höhe eingegeben und den Variablen l, b und h zugeordnet. Zeile 50 berechnet dann das Volumen v, indem es diese drei Werte miteinander multipliziert. Zeile 60 gibt schließlich das Ergebnis mit einem entsprechenden Hinweis aus. Hier ein Probelauf:

```
Laenge? 20
Breite? 10
Hoehe? 6
Das Volumen betraegt 1200
Ready
```

Achten Sie darauf, daß ein String, der mit Hilfe der INPUT-Anweisung anfordert wird, keine Kommas oder Anführungszeichen enthalten darf. Jedoch gibt es eine spezielle Form der INPUT-Anweisung, die jedes eingegebene Zeichen übernimmt und sich deshalb für die Eingabe von Textzeilen in einem Textverarbeitungsprogramm besonders gut eignet. Dabei handelt es sich um die LINE INPUT-Anweisung, die nur Strings, aber keine numerischen Werte übernehmen kann. Auf dem Bildschirm erscheint nur der Cursor, aber kein Fragezeichen. Hier ein Beispiel:

```
10 LINE INPUT a$
20 PRINT a$
```

Nach dem Starten des Programms geben Sie jetzt einige Kommas oder Anführungszeichen ein und drücken die RETURN-Taste. Die PRINT-Anweisung in Zeile 20 läßt die Zeichen dann so auf dem Bildschirm erscheinen, wie Sie sie eingegeben haben.

3 Programmschleifen und Verzweigungen

In den vorangehenden Kapiteln haben wir Programme kennengelernt, die alle von der ersten bis zur letzten Zeile abgearbeitet wurden und dann endeten. Ein Computer hätte jedoch nur einen kleinen Einsatzbereich, wenn er lediglich Programme dieser Art verarbeiten könnte.

Außer zu Übungszwecken gibt es aber kaum ein Programm, das sich mit den bisher vorgestellten Anweisungen begnügt. Vielmehr werden viele Anweisungen wiederholt abgearbeitet oder nur dann ausgeführt, wenn eine bestimmte Bedingung erfüllt oder nicht erfüllt ist. Mit derartigen Anweisungen wollen wir uns in diesem Kapitel beschäftigen.

Bei wiederholt ausgeführten Anweisungen spricht man auch von Programmschleifen, die aber nicht nur aus einer, sondern aus einer Vielzahl von Einzelanweisungen bestehen können. Häufig enthalten diese Schleifen eine Abbruchbedingung, d.h. sie werden solange wiederholt ausgeführt, bis die Bedingung erfüllt ist. Durch diese Programmstruktur ist der Computer in der Lage, regelrechte Entscheidungen zu treffen.

Im folgenden wollen wir uns nun mit den verschiedenen Wiederholungs- und Verzweigungsanweisungen befassen, die das CPC-BASIC zur Verfügung stellt.

3.1 FOR...NEXT

Die wohl einfachste und bekannteste Wiederholungsanweisung ist die FOR...NEXT-Anweisung. Wir wollen sie anhand eines kleinen Beispiels einmal näher betrachten.

Angenommen, Sie wollen die Zahlen 1 bis 10 nacheinander auf dem Bildschirm ausgeben. Dazu gibt es zwei Möglichkeiten, von denen die erste so aussieht:

```
10 PRINT 1;
20 PRINT 2;
30 PRINT 3;
40 PRINT 4;
50 PRINT 5;
60 PRINT 6;
70 PRINT 7;
80 PRINT 8;
90 PRINT 9;
100 PRINT 10;
```

Jetzt starten wir dieses Programm mit RUN und erhalten

```
1  2  3  4  5  6  7  8  9  10
Ready
```

In diesem Fall mußten wir zehnmal die PRINT-Anweisung ausführen, um alle Zahlen zwischen 1 und 10 auszugeben. Nun wollen wir das Programm so umschreiben, daß wir hinter PRINT nicht eine Zahl, sondern eine Variable i ausgeben. Damit dieses Programm nicht allzuviel Platz einnimmt, wollen wir immer zwei durch einen Doppelpunkt getrennte Anweisungen in eine Programmzeile setzen. Theoretisch können es soviele Anweisungen sein, wie in eine Zeile passen. Eine Programmzeile darf im BASIC 7.0 maximal 255 Textzeichen umfassen, was beim 80-Zeichen-Modus (MODE 2) etwas mehr als drei und beim 40-Zeichen-Modus (MODE 1) etwas mehr als sechs Zeilen auf dem Bildschirm entspricht.

```
10 i = 1 : PRINT i;
20 i = 2 : PRINT i;
30 i = 3 : PRINT i;
40 i = 4 : PRINT i;
50 i = 5 : PRINT i;
60 i = 6 : PRINT i;
70 i = 7 : PRINT i;
80 i = 8 : PRINT i;
90 i = 9 : PRINT i;
100 i = 10 : PRINT i;
```

In diesem Beispiel haben wir jeweils die Werte 1 bis 10 der Variablen i zugeordnet, aber immer noch müssen wir für jeden Wert die Anweisungen schreiben.

Um dies zu erleichtern, verwenden wir jetzt die FOR...NEXT-Anweisung. Dann sieht unser Programm so aus:

```
10 FOR i = 1 TO 10
20 PRINT i;
30 NEXT i
```

i ist hier die Laufvariable, die von 1 bis 10 durchzählt. Alle Anweisungen, die zwischen FOR und NEXT stehen, werden so häufig ausgeführt, wie es in der FOR-Anweisung angegeben ist. Erinnern wir uns nochmals daran, daß hinter der PRINT-Anweisung ein Semikolon steht, damit die einzelnen Zahlenwerte nebeneinander ausgegeben werden. Fehlt das Semikolon, erscheinen die Zahlenwerte untereinander, jeweils in einer neuen Zeile.

Damit sind die Möglichkeiten der FOR...NEXT-Anweisung aber noch lange nicht erschöpft. Mit dem Zusatz STEP können wir auch rückwärts oder in größeren Schritten zählen; fehlt er dagegen, werden Einerschritte angenommen. Betrachten wir dazu folgendes Beispiel:

```
10 FOR i = 10 TO 1 STEP -1
20 PRINT i;
30 NEXT i
```

Nach RUN folgt

```
 10  9  8  7  6  5  4  3  2  1
Ready
```

STEP -1 gibt an, daß in Einerschritten zwischen 10 und 1 rückwärts gezählt werden soll. STEP kann aber auch einen größeren positiven oder negativen Wert einnehmen, wie im folgenden Beispiel:

```
10 FOR i = 0 TO 50 STEP 10
20 PRINT i;
30 NEXT i
```

Nach RUN folgt

```
 0  10  20  30  40  50
Ready
```

Hinter NEXT kann der Laufvariablenname i auch weggelassen werden, was u.a. eine geringfügige Erhöhung der Abarbeitungsgeschwindigkeit zur Folge hat.

Die Schrittweite muß nicht unbedingt ganzzahlig sein, wie das folgende Beispiel zeigt:

```
10 FOR i = .2 TO -.6 STEP -.2
20 PRINT i;
30 NEXT
```

Nach RUN folgt

```
 .2  0 -.2 -.4 -.6
Ready
```

Die allgemeine Form der FOR...NEXT-Anweisung lautet:

```
FOR Variable = Anfangswert TO Endwert STEP Schrittweite
(Verschiedene Anweisungen)
NEXT Variable oder nur NEXT
```

In der FOR-NEXT-Anweisung können sowohl Anfangs- und Endwert als auch die Schrittweite aus Variablen, Ausdrücken oder Konstanten bestehen. Ein Beispiel:

```
10 a = 20 : b = 10  : c = -.5
20 FOR i = a TO b STEP c
30 PRINT c * 3 + 5
40 NEXT
```

a steht hier für den Anfangswert, b für den Endwert und c für die Schrittweite.

Es besteht auch die Möglichkeit, mehrere FOR...NEXT-Schleifen zu verschachteln. Dabei ist jedoch darauf zu achten, daß die inneren Schleifen in sich abgeschlossen sind. Hier ein Beispiel:

```
10 FOR i = 1 TO 10
20 FOR j = 0 TO 20 STEP 2
30 FOR k = 7 TO 50
40 .... (Anweisungen)
50 NEXT k
60 NEXT j
70 NEXT i
```

Beachten Sie, daß in diesem Beispiel die innere Schleife die Laufvariable k, die mittlere j und die äußere i hat. Falsch dagegen ist es, die Reihenfolge von NEXT zu vertauschen, wie im folgenden Beispiel (hier nur Zeilen 50 bis 70):

```
50 NEXT i
60 NEXT k  (f a l s c h !)
70 NEXT j
```

Bevor wir die FOR...NEXT-Anweisung verlassen, noch ein praktisches Beispiel zur Zinsberechnung:

```
10 REM Zinsberechnung
20 INPUT "Kapital"; k
30 INPUT "Zinssatz (Prozent)"; zp
40 INPUT "Laufzeit (Jahre)"; l
50 z = zp/ 100
60 b = k
70 FOR i = 1 TO l
80 b = b + b * z
90 NEXT
100 PRINT "Der Betrag ist"; b
```

In Zeile 20 wird per INPUT-Anweisung das Kapital eingegeben und der Variablen k zugeordnet. Das gleiche geschieht in Zeile 30 mit dem Zinssatz in Prozent (zp) und in Zeile 40 mit der Laufzeit in Jahren (l). Zeile 50 rechnet den Zinssatz in Prozent (zp) in einen Dezimalbruch (z) um. In Zeile 60 steht die Variable b für den jeweiligen Betrag, der zunächst mit dem

Kapital k gleichgesetzt wird. Die Zeilen 70 bis 90 bilden dann eine Schleife mit einer FOR...NEXT-Anweisung. Für jedes Jahr der Laufzeit l wird hier der jeweilige Betrag berechnet. Zeile 100 gibt schließlich das Endergebnis auf dem Bildschirm aus. Beachten Sie bei dieser PRINT-Anweisung, daß sie zunächst den String "Der Betrag ist" und dann den Inhalt der Fließkommavariablen b ausgibt.

Wir lassen das Programm jetzt einmal laufen, wobei das Kapital von 1000000 über 10 Jahre mit einem Zinssatz von 12% verzinst werden soll:

```
Kapitel? 1000000
Zinssatz (Prozent)? 12
Laufzeit (Jahre)? 10
Der Betrag ist 3105848.21
Ready
```

Das Ergebnis wäre sicherlich übersichtlicher und besser zu lesen, wenn es so geschrieben würde: 3 105 848.21. Im weiteren Verlauf dieses Buches werden wir noch die PRINT USING-Anweisung kennenlernen, die eine formatierte Ausgabe ermöglicht.

Ändern wir dieses Programm leicht ab, so können wir es zur Berechnung von Annuitäten verwenden. Der einzige Unterschied besteht darin, daß zu Beginn eines jeden Jahres ein neuer Betrag hinzukommt. Deshalb erhöht sich der Wert von b nicht nur um die Zinsen, sondern auch um die jeweils neue Einlage.

Diese Einlage ist in der Variablen e abgelegt. Der Betrag b wird dann in zwei Stufen berechnet, indem zunächst die jährliche Einlage hinzugezählt und schließlich aus der Summe die Zinsen des letzten Jahres berechnet und hinzuaddiert werden:

```
10 REM Annuitaeten
20 INPUT "Anfangsbetrag"; b
30 INPUT "Zinssatz (Prozent)"; zp
40 INPUT "Einlage pro Jahr"; e
50 INPUT "Laufzeit (Jahre)"; l
60 z = zp/ 100
70 FOR i = 1 TO l
80 b = b + e
90 b = b + b * z
100 NEXT
110 PRINT "Der Betrag ist"; b
```

Nachdem das Programm mit RUN gestartet wurde, testen wir es mit folgenden Werten:

```
Anfangsbetrag? 1000000
Zinssatz (Prozent)? 15
Einlage pro Jahr? 10000
Laufzeit (Jahre)? 10
Der Betrag ist 4279050.5
```

Bitte beachten Sie, daß im Anfangsbetrag die erste Jahreseinlage nicht enthalten ist, weshalb sich der Betrag zunächst aus dem Anfangsbetrag und der ersten Jahreseinlage zusammensetzt.

3.2 WHILE...WEND

Die FOR...NEXT-Anweisung teilt dem Computer mit, wie oft er die zwischen FOR und NEXT liegenden Anweisungen ausführen soll. Dies ist in der Laufvariablen hinter FOR angegeben. Es gibt jedoch Fälle, in denen es von vornherein nicht bekannt ist, wie oft eine Schleife auszuführen ist. Dies geschieht dann solange, bis eine bestimmte Bedingung erfüllt ist. Das CPC-BASIC bietet dafür die WHILE...WEND-Anweisung.

Ähnlich wie bei FOR...NEXT umfaßt die Schleife hier sämtliche zwischen WHILE und WEND liegenden Anweisungen; dabei ist jedoch keine vorgegebene Anzahl von Durchläufen festgelegt.

Um uns dies zu verdeutlichen, wollen wir nochmals alle Zahlen zwischen 1 und 10 ausgeben, diesmal aber nicht mit der FOR...NEXT-, sondern mit der WHILE...WEND-Anweisung:

```
10 i = 1
20 WHILE i <= 10
30 PRINT i;
40 i = i + 1
50 WEND
```

Nach Eingabe von RUN erfolgt

```
 1  2  3  4  5  6  7  8  9  10
Ready
```

Als Zählvariable verwenden wir hier wieder i. In Zeile 10 erhält sie den Anfangswert 1. Die WHILE...WEND-Schleife umfaßt die Zeilen 20 bis 50, wobei Zeile 30 den jeweiligen Wert von i auf dem Bildschirm ausgibt. Zeile 40 erhöht den Wert von i jeweils um 1, d.h. bei jedem Schleifendurchlauf wird i fortgezählt.

Dieser Vorgang wiederholt sich nun so oft, bis die Abbruchbedingung der WHILE-Anweisung erfüllt ist. Die Schleife wird nämlich nur solange ausgeführt, wie i kleiner oder gleich 10 ist, anderenfalls wird die Wiederholung abgebrochen und mit der nächsten Anweisung fortgefahren.

Die Zeichen "<" und "=" sind Vergleichssymbole, die wir bereits aus der Algebra kennen. Dabei wird der jeweilige Wert von i mit der Zahl 10 verglichen und zwar solange, wie er kleiner oder gleich 10 ist.

Insgesamt gibt es in BASIC sechs Vergleichssymbole, die abgefragt werden können. Sie dienen zum Vergleichen von konstanten Werten, Variablen und Ausdrücken. Hier eine Zusammenfassung der Symbole:

Vergleichssymbol	Bedeutung
=	gleich
<	kleiner als
<=	kleiner gleich
>	größer als
>=	größer gleich
<>	ungleich

Auf keinen Fall darf aber das Gleichheitszeichen als Vergleichssymbol mit demjenigen in einer Zuordnungsanweisung verwechselt werden, da es sich hier um völlig verschiedene Anwendungen handelt. In einer Zuordnung sagt

```
y = z
```

aus, daß der Wert der Fließkommavariablen y gleich demjenigen der Variablen z gesetzt wird. Schreiben wir dagegen

```
WHILE y = z
```

so liegt eine Bedingung vor, die erst erfüllt ist, wenn y den gleichen Wert wie z erreicht hat.

Grundsätzlich sind die Vergleichssymbole auch für Strings anwendbar, wobei allerdings die alphabetische Reihenfolge geprüft wird.

Wie wir bereits wissen, setzen sich Strings aus sogenannten ASCII-Zeichen zusammen, wobei jedem Buchstaben intern ein bestimmter Zahlenwert zugeordnet ist. So entspricht der Buchstabe A dem Code 65, B dem Code 66 usw. und Z dem Code 90. Kleinbuchstaben liegen analog im Bereich 97 bis 122 und haben somit größere ASCII-Werte als die Großbuchstaben. Dies können wir anhand der folgenden beiden Anweisungen überprüfen:

```
PRINT CHR$(65)
 A
Ready

PRINT ASC("A")
 65
Ready
```

Die CHR$-Anweisung erzeugt für einen vorgegebenen ASCII-Code das entsprechende Zeichen; in unserem Beispiel erzeugt die Zahl 65 den Buchstaben A. ASC ist hiervon die Umkehrfunktion, die in unserem Fall den dem Buchstaben A entsprechenden ASCII-Code erzeugt.

Normalerweise sind die CHR$- und ASC-Anweisungen nur für den fortgeschrittenen Programmierer interessant, der sie meist für ganz spezielle Anwendungsbereiche benötigt. Für uns sollen sie im Augenblick ausschließlich zum Verständnis dienen, warum der Computer auch Strings mit den Vergleichssymbolen in alphabetischer Reihenfolge vergleichen kann. Gegenüber den Zahlenvergleichen haben die Symbole hier eine etwas andere Bedeutung, wie die nachfolgende Aufstellung zeigt:

Vergleichssymbol	Bedeutung
=	gleich
<	kleiner in alphabetischer Reihenfolge
<=	kleiner oder gleich in alphab. Reihenfolge
>	größer in alphabetischer Reihenfolge
>=	größer oder gleich in alphab. Reihenfolge
<>	ungleich

Hier noch einige Beispiele zum Stringvergleich:

HANS < MARTIN
Unger > Meyer
SCHUH < SCHUHLADEN
dreimal > drei mal
beispiel > BEISPIEL

Beachten Sie bitte, daß beim dritten Beispiel das Wort SCHUH kleiner als SCHUHLADEN ist. Letzteres enthält zwar vorne das Wort SCHUH, besteht aber noch aus zusätzlichen Zeichen.

Auch das Wort "dreimal" ist größer als "drei mal", da "drei mal" ein Leerzeichen (ASCII-Code 32) enthält, das kleiner ist als die Buchstaben des Alphabets.

Auch ist "beispiel" größer als "BEISPIEL", da Kleinbuchstaben höheren ASCII-Codes entsprechen als Großbuchstaben (s.o.).

3.3 READ und DATA

Wir kennen bereits die INPUT-Anweisung, die während des Programmablaufs Daten über die Tastatur vom Bediener anfordert. Es gibt aber noch eine weitere Möglichkeit, Daten ins Programm zu übernehmen. Solche Daten sind in DATA-Anweisungen abgelegt und werden bei Bedarf zur weiteren Verarbeitung mit der READ-Anweisung gelesen.

Hier kommen in erster Linie konstante Datenwerte in Frage und nicht variable, wie bei der INPUT-Anweisung. Es ist nämlich häufig der Fall, daß ein Programm mit festen Tabellen arbeiten muß, die dann aus DATA-

Zeilen eingelesen werden können. Anderenfalls müßte sie der Bediener jeweils von Hand eingeben oder von einem externen Datenträger (Diskette oder Band) einladen.

Die letzte Möglichkeit empfiehlt sich nur bei extrem umfangreichen Datendateien, die, wären sie in DATA-Zeilen abgelegt, zuviel Speicherplatz beanspruchen würden. Auch eine sich ständig ändernde Datei (z.B. Adressendatei) legt man besser auf einem Datenträger ab. Halten sich dagegen die Datenmengen in kleinerem Rahmen und bleiben sie bei jedem Programmablauf gleich, ist es oft sinnvoll, sie in DATA-Anweisungen abzulegen, die ein fester Bestandteils des BASIC-Programmtextes sind.

Bereits in Kapitel 2 haben wir mit Hilfe der PRINT-Anweisung eine Tabelle auf dem Bildschirm ausgegeben. Diese Tabelle, die den Namen, die Punktzahl und die Noten von Klassenarbeiten für verschiedene Schüler enthält, wollen wir hier zu Demonstrationszwecken mit READ- und DATA-Anweisungen ausgeben. Dazu dient das folgende Programm:

```
10 PRINT "Name","Punkte","Note"
20 PRINT
30 READ na$ : IF na$  = "ende" THEN END
40 READ p, n
50 PRINT na$, p, n
60 GOTO 30
100 DATA Hans, 35, 3
110 DATA Nicole, 40, 4
130 DATA Stefan, 50, 2.7
140 DATA Christian, 38, 4.3
150 DATA Ursula, 60, 2
160 DATA ende
```

Zu Beginn erscheint wieder die Kopfzeile und eine Leerzeile (Zeilen 10 und 20). Die Zeilen 30 bis 60 bilden eine Schleife, die für jeden Schüler durchlaufen wird, während die eigentliche "Datei" erst in Zeile 100 beginnt. Hier ist für jeden einzelnen Schüler eine DATA-Zeile aufgeführt.

In einer DATA-Anweisung können eine oder mehrere Daten stehen. Sind mehrere vorhanden, wie in unserem Fall, müssen sie durch Kommas getrennt sein.

Beachten Sie bitte auch, daß Strings in DATA-Zeilen nicht in Anführungszeichen eingeschlossen sein brauchen. Es ist allerdings kein Fehler, dies zu tun. Enthält ein String ein Komma, so ist dies jedoch zwingend notwendig,

da Kommas ansonsten als Trennzeichen zwischen den verschiedenen Daten dienen. Das folgende Beispiel verdeutlicht dies:

```
100 DATA "Bis gleich, auf Wiedersehen", Tschuess, "Ade"
```

Diese DATA-Zeile enthält drei verschiedene Stringdaten. Wäre "Bis gleich, auf Wiedersehen" nicht in Anführungszeichen eingeschlossen, würde BASIC annehmen, daß hier zwei Strings, also "Bis gleich" und "auf Wiedersehen" vorliegen. "Tschuess" dagegen gilt als ein Wert und braucht nicht in Anführungszeichen zu stehen. Das gleiche gilt für "Ade", wo die Anführungszeichen keine Wirkung haben und durchaus weggelassen werden könnten.

Doch nun zurück zu unserem Programm. In der Schleife liest die READ-Anweisung in Zeile 30 zunächst den ersten Datenwert, der hier ein Name (String) ist, aus der Datentabelle und ordnet ihn der Stringvariablen na$ zu. Anschließend folgt die Abfrage nach dem Label "ende". Ein Label ist in der Computertechnik eine Kennzeichnung, die zu bestimmten Abfragen dient. In unserem Fall haben wir den String "ende" an das Tabellenende gesetzt. Wird nun dieses Wort (Label) als Name eingelesen, weiß der Computer, daß die Tabelle abgearbeitet ist und beendet das Programm. Statt "ende" könnten wir auch jedes andere beliebige Wort als Abbruchkriterium (Label) vereinbaren.

Enthält nun na$ einen anderen Wert als "ende", ist also das Tabellenende noch nicht erreicht, wird in einer weiteren READ-Anweisung noch die Punktzahl und die Note des Schülers eingelesen. Genauso wie bei DATA können auch hinter READ mehrere durch Komma getrennte Variablennamen stehen. Zeile 50 gibt dann sämtliche Werte auf dem Bildschirm aus, worauf das Programm in Zeile 60 an den Schleifenanfang zurückspringt.

In diesem Programm haben wir noch zwei weitere Anweisungen kennengelernt, mit denen wir uns ebenfalls etwas näher beschäftigen werden. IF fragt, ähnlich wie WHILE, eine Bedingung ab, in unserem Fall, ob das Dateiende bereits erreicht ist. Die Anweisung GOTO führt einen unbedingten Sprung aus, der hier in Zeile 30 zum Schleifenanfang führt.

Nun noch ein paar Worte zum Aufbau von Programmen mit DATA-Zeilen. Findet BASIC nach dem Programmstart durch RUN erstmals eine READ-Anweisung vor, so liest es den ersten Wert in der ersten DATA-Zeile. Wird READ ein zweites Mal vorgefunden, so ist der zweite Datenwert an der Reihe usw. BASIC verfügt nämlich über einen internen Zeiger und weiß deshalb, welche DATA-Anweisung als nächste an der Reihe ist. Es spielt deshalb keine Rolle, an welcher Stelle die DATA-Zeilen im Programm ste-

hen. Entscheidend ist nur ihre Reihenfolge im Programmtext. Im vorliegenden Beispiel hätten wir sie genauso gut auch an den Programmanfang setzen oder zwischen die anderen BASIC-Zeilen verteilen können, was allerdings von der Übersichtlichkeit her nicht zu empfehlen ist.

Beim Programmstart durch RUN wird der interne DATA-Zeiger automatisch auf den ersten Datenwert gesetzt. Manchmal ist es jedoch erforderlich, ihn während des Programmablaufs wieder an den Anfang zurückzusetzen. Hierzu dient die Anweisung

RESTORE

Das CPC-BASIC bietet darüber hinaus noch die Möglichkeit, den Zeiger auf jede beliebige DATA-Zeile zu setzen, wodurch die Datenwerte erst ab der angegebenen Zeile gelesen werden. Dies wollen wir anhand unseres Programms einmal ausprobieren. Dazu setzen wir folgende Zeile an den Programmanfang:

5 RESTORE 130

Wenn Sie jetzt das Programm starten, werden als erstes die in Zeile 130 abgelegten Daten ausgegeben.

Wird die READ-Anweisung öfter aufgerufen als Datenwerte vorhanden sind, läuft also der interne DATA-Zeiger über den letzten Datenwert hinaus, erscheint die Fehlermeldung

DATA exhausted in (Zeilennummer)

Diese Meldung führt gleichzeitig zum Programmabbruch.

3.4 IF...THEN...ELSE

Die IF-Anweisung haben wir bereits bei der Schleifenprogrammierung kennengelernt, wo sie zur Bestimmung des Abbruchkriteriums diente. Sie hatte dort die allgemeine Form

IF (Bedingung) THEN ...

Aber auch jede andere Abfrage kann durch IF vorgenommen werden. Stößt nämlich der Computer auf eine IF-Anweisung, prüft er, ob die Bedingung wahr oder falsch ist. Ist die Bedingung wahr, werden die hinter THEN stehenden Anweisungen ausgeführt, anderenfalls werden sie übersprungen. Betrachten wir diesen Vorgang anhand eines einfachen Beispiels, bei dem die Zahl 25 erraten werden soll:

```
5 REM Zahlenraten
10 INPUT "Bitte Zahl eingeben"; z
20 IF z = 25 THEN PRINT "RICHTIG!" : GOTO 40
30 PRINT "FALSCH!"
40 PRINT "Weiterraten - bitte Leertaste druecken"
50 PRINT "Wenn andere Taste - Ende"
60 a$=INKEY$ : IF a$ = "" THEN 60
70 IF a$=" " THEN 10
80 END
```

In Zeile 10 ist eine beliebige Zahl z einzugeben. In Zeile 20 prüft dann die IF-Anweisung, ob es sich hierbei um die richtige Zahl 25 handelt. Ist dies der Fall, werden die hinter THEN stehenden Anweisungen ausgeführt, d.h. es erscheint das Wort "RICHTIG!" auf dem Bildschirm. Damit das Programm nun nicht mit Zeile 30 fortfährt, steht dahinter noch die Sprunganweisung GOTO 40, d.h. die Programmausführung wird in Zeile 40 fortgesetzt.

Ist die Zahl z eine andere als 25, werden die hinter der IF...THEN-Anweisung stehenden Anweisungen nicht ausgeführt und das Programm fährt mit Zeile 30 fort.

In beiden Fällen wird der Programmteil ab Zeile 40 ausgeführt, wo zunächst zwei Strings zur Information auf dem Bildschirm ausgegeben werden. In Zeile 60 stehen die Anweisungen

```
60 a$=INKEY$ : IF a$="" THEN 60
```

INKEY$ fragt die Tastatur ab. Wurde eine Taste gedrückt, wird deren ASCII-Code in den String a$ übernommen. Wenn keine Taste gedrückt wurde, enthält a$ einen Nullstring "", der mit der dahinterstehenden IF-Anweisung abgefragt wird. In diesem Fall hängt das Programm in einer Warteschleife; es springt wieder an den Anfang von Zeile 60 zurück und fragt erneut die Tastatur ab. Wenn nun eine Taste gedrückt wurde, fragt eine weitere IF-Anweisung in Zeile 70, ob es sich hierbei um die Leertaste handelt, die durch einen Leerstring von einem Zeichen Länge gekennzeichnet ist. Ist die Bedingung erfüllt, findet ein Sprung zur INPUT-Anweisung in Zeile 10 statt, wo eine neue Zahl einzugeben ist. Wurde eine andere Taste gedrückt, ist also die Bedingung nicht erfüllt, wird in Zeile 80 die END-Anweisung ausgeführt.

In diesem Fall könnte man die END-Anweisung auch fortlassen, da der Programmtext ohnehin an dieser Stelle zu Ende ist. END ist überall dort erforderlich, wo die Programmausführung innerhalb des Textes beendet wird, weil sie sonst mit den unteren Zeilen fortfahren würde.

Unser Zahlenrateprogramm können wir dank des leistungsstarken CPC-BASIC auch noch etwas kompakter schreiben, wie das folgende Listing zeigt:

```
5 REM Zahlenraten
10 INPUT "Bitte Zahl eingeben"; z
20 IF z = 25 THEN PRINT "RICHTIG!": ELSE PRINT "FALSCH!"
30 PRINT "Weiterraten - bitte Leertaste druecken"
40 PRINT "Wenn andere Taste - Ende"
50 a$=INKEY$ : IF a$="" THEN 50
60 IF a$=" " THEN 10 : ELSE END
```

Hier werden die beiden Möglichkeiten einer richtigen bzw. falschen Zahleneingabe in einer Zeile abgehandelt. Ist die Bedingung erfüllt, wird wie zuvor "RICHTIG!" ausgegeben; ist sie nicht erfüllt, werden stattdessen die hinter ELSE stehenden Anweisungen ausgeführt. Dasselbe wiederholt sich in Zeile 60 mit der Abfrage der Leertaste.

Wichtig! Die Anweisungen IF...THEN und IF...THEN...ELSE müssen einschließlich der Bedingung und sämtlicher in diesem Zusammenhang auszuführender Anweisungen in einer Programmzeile stehen. Steht hinter THEN bzw. ELSE eine Zeilennummer, so findet ein Sprung in die betreffende Zeile statt. Man könnte hierfür auch THEN GOTO (Zeile) bzw. ELSE GOTO (Zeile) schreiben.

3.5 Die Booleschen Operatoren

Haben wir z.B. die Bedingung

```
x = y
```

sagt diese aus, daß der Wert von x gleich dem Wert von y ist. Darüber hinaus ist es wünschenswert, auch Bedingungen in zusammengesetzten Sätzen zu schreiben, wie z.B. "Der Wert von u ist gleich dem Wert von v und der Wert von x ist kleiner als der Wert von y". Durch Einsetzen der entsprechenden logischen Operatoren AND, OR und NOT (deutsch: und, oder, nicht), können Bedingungen gemäß dem oben zusammengesetzten Satz geschrieben werden:

```
u = v AND x < y
```

Bevor wir uns nun mit den einzelnen logischen Operatoren befassen, wollen wir uns darüber klar werden, wie der Computer reagiert, wenn eine Bedingung wahr oder falsch ist. In einigen Programmiersprachen gibt es dafür einen speziellen Datentyp, der nur die beiden Werte TRUE (wahr) und FALSE (falsch) annehmen kann. Ein solcher Datentyp existiert in BASIC nicht; stattdessen findet der Datentyp Integer Verwendung. 0 steht hier für falsch und -1 für wahr. Bedingungen wie a = b sind Ausdrücke, die den Wert -1 ergeben, wenn die Bedingung wahr ist und 0, wenn sie falsch ist:

```
PRINT 2 = 2
-1
Ready

PRINT 12 > 20
 0
Ready

PRINT "Hans" < "Martin"
-1
Ready

PRINT "Fritz" = "Fritzchen"
 0
Ready
```

Ergibt eine durch IF geprüfte Bedingung den Wert -1, dann werden die Anweisungen im THEN-Teil ausgeführt; entsteht dagegen der Wert 0, dann sind es die Anweisungen im ELSE-Teil.

```
IF -1 THEN PRINT "wahr" ELSE PRINT "falsch"
wahr
Ready

IF 0 THEN PRINT "wahr" ELSE PRINT "falsch"
falsch
Ready
```

Die IF-Anweisung akzeptiert in der Regel auch andere Werte (außer 0), um "wahr" zu erzeugen, wie im folgenden Beispiel:

```
IF 3 THEN PRINT "wahr" : ELSE PRINT "falsch"
wahr
Ready
```

BASIC kennt somit nur einen Wert, nämlich den Wert 0, der als "falsch" gedeutet wird, alle anderen Werte bedeuten "wahr". Obwohl alle Werte, die ungleich 0 sind, den Zustand "wahr" darstellen, wollen wir hier trotzdem den Wert -1 weiterverwenden, da er auch von den Booleschen Operatoren erzeugt wird. Im folgenden werden wir die Operatoren einzeln besprechen.

3.5.1 NOT

Ist einer Bedingung NOT vorangestellt, wird sie verneint, genauso, wie ein Satz, der das Wort "nicht" enthält, verneint wird.

```
NOT 7 < 5
```

entspricht somit der Aussage, daß 7 nicht kleiner als 5 ist. Um dies zu erreichen, muß NOT von -1 nach 0 und von 0 nach -1 umgeändert werden:

```
PRINT NOT -1; NOT 0
 0 -1
Ready

PRINT NOT 7 < 5
-1
Ready
```

3.5.2 AND

Betrachten wir zunächst die Bedingung:

```
2 < 3 AND 10 > 12
```

In Worten ausgedrückt bedeutet dies: "2 ist kleiner als 3 und 10 ist größer als 12". Ein zusammengesetzter Satz mit AND (und) ist nur wahr, wenn beide Teilsätze wahr sind. AND erzeugt den Wert -1 also nur dann, wenn beide betrachteten Werte wahr, also auch -1 sind.

```
PRINT 0 AND 0; 0 AND -1; -1 AND 0; -1 AND -1
 0  0  0 -1
Ready

PRINT 2 < 3 AND 10 > 12
 0
Ready
```

3.5.3 OR

Zunächst sehen wir uns wieder die Bedingung an:

```
2 < 3 OR 10 > 12
```

Sie sagt aus, daß 2 kleiner als 3, oder 10 größer als 12 ist oder beides zutrifft. Durch OR entsteht der Wert -1 (wahr), wenn einer oder beide Werte -1 (wahr) sind.

```
PRINT 0 OR 0; 0 OR -1; -1 OR 0; -1 OR -1
 0 -1 -1 -1
Ready

PRINT 2 < 3 OR 10 > 12
-1
Ready
```

3.5.4 XOR

Die Bedingung

```
2 < 3 XOR 10 > 12
```

sagt aus, daß 2 kleiner als 3 und 10 größer als 12 ist, aber nicht beides zutrifft. Nur wenn beide Werte gleich sind, unterscheiden sich OR und XOR, wobei OR den Wert -1 und XOR den Wert 0 erzeugt:

```
PRINT 0 XOR 0; 0 XOR -1; -1 XOR 0; -1 XOR -1
 0 -1 -1  0
Ready
```

Abschließend wollen wir uns noch einen praktischen Anwendungsfall für die Booleschen Operatoren ansehen. In der Geodäsie ist es üblich, zum Zwecke weiterer Fernbeobachtungen eine Strecke von 50 m genau einzumessen. Dazu wird die Strecke mehrfach gemessen, um sicherzustellen, daß sich der Abstand der Endpunkte innerhalb der zulässigen Toleranzgrenze von 0,03 m bewegt. Messungen, die über diese Toleranzgrenze hinausgehen, werden nicht weiter verwertet und scheiden aus.

```
10 REM Toleranzmessung einer Messtrecke
20 PRINT "Laenge", "angenommen"
30 PRINT
40 READ l
50 IF l = 9999 THEN END
60 PRINT l,
70 IF l>=49.97 AND l<=50.03 THEN PRINT "ja": ELSE PRINT "nein"
80 GOTO 40
90 DATA 50.01, 50.07, 49.97, 49.99, 49.93, 9999
```

Nach Eingabe von RUN erscheint folgende Auswertung:

```
Laenge        angenommen

 50.01        ja
 50.07        nein
 49.97        ja
 49.99        ja
 49.93        nein
Ready
```

Die einzelnen Meßwerte l werden in der Schleife (Zeile 40 bis 80) aus den DATA-Zeilen eingelesen, getestet und ausgegeben. Ist ihre Länge innerhalb der zulässigen Toleranzgrenze, wird zusätzlich das Wort "ja" ausgegeben, anderenfalls das Wort "nein". Das Tabellenende ist hier wieder mit einem Label gekennzeichnet, denn wenn das Programm den Wert 9999 erkennt, bricht es ab.

4 Programmaufbau und Fehlerbehandlung

In diesem Kapitel wollen wir uns mit den wesentlichen Elementen zum Aufbau von BASIC-Programmen befassen und in diesem Zusammenhang weitere wichtige Anweisungen kennenlernen. Zwar wissen wir bereits, daß wir Programmzeilen beliebig aneinanderreihen können, um sie dann nacheinander abzuarbeiten, wobei wir zweifellos ein ablauffähiges Programm erhalten. Es bleibt jedoch die Frage offen, ob das Programm optimal aufgebaut ist, insbesondere was den Programmieraufand, die Rechenzeit und die Übersichtlichkeit betrifft. Wir werden uns deshalb nunmehr mit Funktionen und Unterprogrammen beschäftigen, aus denen sich gut strukturierte Programme zusammensetzen. Ein auf diese Weise sinnvoll durchgeführter Programmaufbau ist der beste Weg, um komplizierte Aufgabenstellungen in übersichtliche Teile zu untergliedern.

Je umfangreicher ein Programm wird, desto größer ist auch die Wahrscheinlichkeit, daß Programmierfehler auftreten. Aus diesem Grunde befassen wir uns am Schluß des Kapitels mit Möglichkeiten, Fehler aufzuspüren und zu beheben.

4.1 Eingebaute Funktionen

Der BASIC-Interpreter enthält einige eingebaute Funktionen, die uns das Programmieren erleichtern. Sie dienen zur Definition von Zahlen- oder Stringoperationen und können als separate Programmteile betrachtet werden, die nicht gesondert definiert werden müssen.

Nun gibt es im CPC-BASIC einfachere und kompliziertere Funktionen. Die wissenschaftlichen Funktionen, die wir schon in Kapitel 2 kennengelernt haben, gehören zu den komplizierteren. Zweifellos könnten Sie z.B. die Sinusfunktion auch selbst programmieren, indem Sie ein Programm schreiben, das bei jedem Aufruf eine Potenzreihe durchrechnet. Dazu brauchten Sie allerdings Kenntnisse in der Potenzreihenentwicklung und müßten darüber hinaus beachten, daß diese Reihe aureichend schnell konvergiert, um bei einer minimalen Anzahl von Schleifendurchläufen die größtmögliche Rechengenauigkeit zu erhalten.

Diese und ähnliche Aufgaben nimmt Ihnen nun der BASIC-Interpreter ab, indem er eine interne Maschinenroutine aufruft, welche die Berechnungen auch noch um ein Vielfaches schneller ausführt, als es mit einem BASIC-Programm möglich wäre.

4.1.1 ABS

Die ABS-Funktion bestimmt den absoluten Wert einer Zahl, d.h. sie führt eine Operation aus, bei der negative Zahlen ein positives Vorzeichen erhalten, während es bei positiven Zahlen gleich bleibt. Mit anderen Worten: Es spielt keine Rolle, ob der Ausgangswert positiv oder negativ ist, das Ergebnis ist auf jeden Fall positiv.

Betrachten wir dies an einem Beispiel: Der Wert -3 wird nach Ausführung der ABS-Funktion zu +3 oder einfach 3, während der Wert 100 gleich bleibt. In BASIC sieht das folgendermaßen aus:

```
PRINT ABS(-3)
 3
Ready

PRINT ABS(100)
 100
Ready
```

Hinter ABS muß das Argument, wie bei allen Funktionen, in Klammern eingeschlossen sein. Dabei muß es nicht unbedingt ein konstanter Zahlenwert sein, sondern kann auch aus einer Variablen oder einem arithmetischen Ausdruck bestehen wie in den folgenden Beispielen:

```
a = -4 : PRINT ABS(a)
 4
Ready

PRINT (ABS 100*(5-2))
 300
Ready
```

Ein typischer Anwendungsfall für die ABS-Funktion ist die Bestimmung von Abweichungen in der Toleranzmessung. Betrachten wir nochmals unser Beispiel aus Kapitel 3, in dem die Meßwerte einer Meßtrecke ausgewertet wurden. Wir wollen es hier etwas erweitern, indem wir nicht nur angeben, ob die Messung verwertbar ist oder nicht, sondern zusätzlich noch den absoluten Fehler in Metern. Dabei interessiert es bei solchen Messungen nicht, ob der Sollwert nach oben oder nach unten abweicht, d.h. ob die Differenz positiv oder negativ ist, denn man verwertet in solchen Fällen immer den Absolutwert.

```
5  MODE 2
10 REM Toleranzmessung einer Messtrecke
20 PRINT "Laenge","Fehler",,"angenommen"
30 PRINT
40 READ l
50 IF l = 9999 THEN END
60 PRINT l, ABS(50-l),
70 IF l>=49.97 AND l<=50.03 THEN PRINT "ja": ELSE PRINT "nein"
80 GOTO40
90 DATA 50.01, 50.07, 49.97, 49.99, 49.93, 9999
```

Beachten Sie Zeile 60, in der außer der gemessenen Länge auch noch der absolute Fehler ausgegeben wird. Nach Eingabe von RUN erscheint die Auswertung:

```
Laenge          Fehler              angenommen

 50.01          1.00000E-02         ja
 50.07          7.00000E-02         nein
 49.97          3.00000E-02         ja
 49.99          1.00000E-02         ja
 49.93          7.00000E-02         nein
Ready
```

4.1.2 INT, FIX und ROUND

Viele Zahlen bestehen aus einem ganzzahligen und einem gebrochenen Teil. So setzt sich z.B. die Zahl 123.456 aus 123 als ganzzahligem und .456 als gebrochenem Teil zusammen.

Die INT-Funktion trennt nun den gebrochenen Teil ab und gibt den gannzahligen wieder. Dabei handelt es sich strenggenommen um den nächstkleineren ganzzahligen Wert, was besonders bei negativen Zahlen zu Irrtümern führen kann. Hier ein paar Beispiele:

```
PRINT INT(123.456)
 123
Ready

PRINT INT(2.2)
 2
Ready

PRINT INT(3.5)
 3
Ready

PRINT INT(-3.5)
-4
Ready

PRINT INT(-5)
-5
Ready
```

Beachten Sie bitte besonders die letzten beiden Beispiele. Von -3.5 ist die nächstkleinere ganze Zahl -4 und nicht -3. -5 hat keinen gebrochenen Teil und wird deshalb nach Ausführung der INT-Funktion genauso wiedergegeben.

Anders ist es bei der Funktion FIX, die bei positiven und negativen Zahlen den gebrochenen Teil entfernt:

```
PRINT FIX(-45.765)
-45
Ready

PRINT FIX(3.78)
 3
Ready
```

Das CPC-BASIC beinhaltet außerdem die Funktion ROUND, die Zahlenwerte auf eine bestimmte Anzahl Nachkommastellen auf- bzw. abrundet. Die allgemeine Syntax für ROUND lautet:

```
ROUND (Zahlenwert,Nachkommastellen)
```

Hier einige Beispiele:

```
PRINT ROUND (123.456,2)
 123.46
Ready

PRINT ROUND (1444.432,0)
 1444
Ready

PRINT ROUND (-12.3578,3)
-12.358
Ready
```

4.1.3 SGN

SGN ist eine Funktion, die das Vorzeichen ihres Arguments bestimmt oder angibt, ob es Null ist. Ist das Argument positiv, erzeugt SGN den Wert 1, ist es negativ, den Wert -1, und bei Null entsteht das Ergebnis 0. Hier einige Beispiele:

```
PRINT SGN(5.23)
 1
Ready

PRINT SGN(-78)
-1
Ready

PRINT SGN(0)
 0
Ready
```

4.1.4 SQR

Die SQR-Funktion erzeugt die Quadratwurzel ihres Arguments. Wird das Ergebnis mit sich selbst multipliziert (quadriert), entsteht wieder der Wert des Arguments:

```
PRINT SQR(49)
 7
Ready

PRINT SQR(2)
 1.41421356
Ready

PRINT SQR(-3)
Improper argument
Ready
```

Im letzten Fall entstand die Fehlermeldung "Improper argument", die angibt, daß ein unzulässiges Argument vorliegt. Aus -3 kann nämlich keine direkte Quadratwurzel gezogen werden.

Als Anwendungsbeispiel für die SQR-Funktion nachfolgend ein Programm, das die Diagonale eines Rechtecks berechnet. Hierzu müssen Länge und Breite quadriert, die Quadrate addiert und aus der Summe die Quadratwurzel gezogen werden (Satz von Pythagoras).

```
10 REM Diagonale in Rechteck
20 INPUT "Laenge"; a
30 INPUT "Breite"; b
40 c = SQR(a^2 + b^2)
50 PRINT "Die Diagonale betraegt"; c
```

Hier ein Probelauf:

```
Laenge? 5
Breite? 2
Die Diagonale betraegt 5.38516481
Ready
```

Beachten Sie bitte, daß es von der Rechenzeit her günstiger ist, die beiden Exponentiationen in Zeile 40 durch Multiplikationen zu ersetzen:

```
40 c = SQR(a*a + b*b)
```

In unserem Beispiel dürfte sich der Zeitvorteil durch Multiplikation noch weniger bemerkbar machen; anders ist es jedoch, wenn Sie große Datenmengen nach dieser Formel berechnen.

4.2 Wissenschaftliche Funktionen

Der Vollständigkeit halber wollen wir hier die wissenschaftlichen Funktionen nochmals betrachten, obwohl sie in Kapitel 2 bereits vorgestellt wurden. Falls Sie diese Funktionen nicht benötigen, können Sie diesen Abschnitt übergehen. Sie sind jedoch für alle Leser wichtig, die mit ihrem CPC 6128 naturwissenschaftlich-technische Berechnungen durchführen möchten.

BASIC bietet insgesamt sieben vorprogrammierte wissenschaftliche Funktionen, von denen X jeweils das Argument darstellt:

SIN(X)	Sinus
COS(X)	Cosinus
TAN(X)	Tangens
ATN(X)	Arcustangens
EXP(X)	Potenz zur Zahl e (2.71...)
LOG(X)	natürlicher Logarithmus
LOG10(X)	Zehnerlogarithmus

Beachten Sie, daß die Winkel für sämtliche trigonometrischen Funktionen im Bogenmaß angegeben werden müssen, es sei denn, Sie verwenden vorher die DEG-Anweisung.

Die folgende Tabelle enthält zwar nicht alle wissenschaftlichen Funktionen, zumindest aber die wichtigsten, aus denen alle anderen leicht hergeleitet werden können. Hier einige Beispiele:

Funktion	Programmierung in BASIC
SECANS(X)	1/COS(X)
COSECANS(X)	1/SIN(X)
COTANGENS(X)	1/TAN(X)
ARCUSSINUS(X)	ATN(X/SQR(1-X*X))
ARCUSCOSINUS(X)	-ATN(X/SQR(1-X*X)) + PI/2
ARCUSCOTANGENS(X)	ATN(X) + PI/2
ARCUSSECANS(X)	ATN(X/SQR(X*X-1))
ARCUSCOSECANS(X)	ATN(X/SQR(X*X-1)) + (SGN(X)-1)*PI/2
SINUS HYPERBOLICUS(X)	(EXP(X)-EXP(-X))/2
COSINUS HYPERBOLIKUS(X)	(EXP(X)+EXP(-X))/2
TANGENS HYPERBOLIKUS(X)	EXP(-X)/(EXP(X)+EXP(-X))*2+1
COTANGENS HYP.(X)	EXP(-X)/(EXP(X)-EXP(-X))*2+1
SECANS HYP.(X)	2/(EXP(X)+EXP(-X))
COSECANS HYP.(X)	2/(EXP(X)-EXP(-X))
ARCUSSINUS HYP.(X)	LOG(X+SQR(X*X+1))
ARCUSCOSINUS HYP.(X)	LOG(X+SQR(X*X-1))
ARCUSTANGENS HYP.(X)	LOG((1+X)/(1-X))/2
ARCUSCOTANGENS HYP.(X)	LOG((X+1)/(X-1))/2
ARCUSSECANS HYP.(X)	LOG((SQR(1-X*X)+1)/X)
ARCUSCOSECANS HYP.(X)	LOG((SQR(1+X*X)+1)/X)

4.3 Zufallszahlen

Über die Funktion RND(X), die Zufallszahlen generiert, werden sich sicher die Leser freuen, die selbst Spiele programmieren möchten. Jedes Spiel wäre nämlich langweilig, wenn sein Ablauf vorhersehbar wäre und der Spieler würde bald die Lust verlieren.

Diese Zufallszahlen sind in Wirklichkeit nur Pseudo-Zufallszahlen, denn der Computer kann keine "zufällige" Entscheidung treffen. Sie sind durchaus errechnet und werden durch einen internen Taktzähler bestimmt, der u.a. auch die interene Uhr steuert.

Die von BASIC erzeugten Zufallszahlen liegen alle im Bereich zwischen 0 und .999999999. Wenn Sie die Anweisung

```
PRINT RND(X)
```

wiederholt ausführen, erhalten Sie jedesmal eine andere Zahl in diesem Bereich. Da es keine negativen Zufallszahlen gibt, läßt sich leicht mit Hilfe der INT-Funktion ein Würfelspiel simulieren, das ganzzahlige Zufallswerte zwischen 1 und 6 erzeugt:

```
10 a$=INKEY$ : IF a$ = "" THEN 10
20 z = RND(x)
30 PRINT INT(6*z)+1
40 GOTO 10
```

Ist dieses Programm gestartet, hängt es in einer Schleife. Zunächst wartet die INKEY$-Anweisung darauf, daß Sie eine beliebige Taste drücken. Zeile 20 ordnet dann der Variablen z eine Zufallszahl zwischen 0 und .999999999 zu. Indem dieser Zahlenbereich mit 6 multipliziert wird, entstehen Zufallszahlen zwischen 0 und 5.99999999. Die INT-Funktion eliminiert nun den gebrochenen Teil, so daß ganze Zahlen zwischen 0 und 5 übrigbleiben. Zum Schluß wird noch 1 hinzuaddiert, so daß wir die gewünschten Zahlen zwischen 1 und 6 erhalten. Schließlich springt Zeile 40 wieder an den Programmanfang zurück.

Durch geringe Abänderung des Programms können wir Zufallszahlen in jedem beliebigen Bereich erzeugen. Wünschen wir z.B. Roulettzahlen zwischen 1 und 37, so lautet Zeile 30

```
30 PRINT INT(37*z)+1
```

Kennen Sie sich erst einmal in der Programmierung von Zufallszahlen aus, sind Ihrer Phantasie keine Grenzen gesetzt, eigene Spiele zu schreiben. Zur Abrundung dieser Beschreibung hier noch ein Zahlenratespiel, bei dem eine Zahl zwischen 1 und 1000 zu erraten ist. Bei jedem Durchlauf bestimmt der Computer eine Zufallszahl in diesem Bereich. Daraufhin werden Sie aufgefordert, diese Zahl zu erraten und einzugeben.

Durch Hinweise erfahren Sie dann, ob diese Zahl, falls sie nicht stimmte, zu groß oder zu klein war und können dann solange einen weiteren Versuch starten, bis Sie die richtige Zahl erraten haben.

```
10 REM Zahlenratespiel
20 r = INT(1000*RND(x))+1
30 INPUT "Bitte Zahl 1 - 1000 eingeben"; z
40 IF z=r THEN PRINT "Richtig! Herzlichen Glueckwunsch!":END
50 IF z>r THEN PRINT "Zu gross!":ELSE PRINT "Zu klein!"
60 GOTO 30
```

Zeile 20 erzeugt zunächst eine Zufallszahl zwischen 1 und 1000. In Zeile 30 werden Sie aufgefordert, eine Zahl in diesem Bereich einzugeben. Haben Sie richtig geraten, erscheint die Meldung "Richtig! Herzlichen Glueckwunsch!" und das Programm endet. Anderenfalls entsteht die Meldung "Zu gross!" bzw. "Zu klein" und Sie werden aufgefordert, eine weitere Zahl einzugeben.

4.4 Frei definierte Funktionen

Neben den fest eingebauten Funktionen bietet BASIC die Möglichkeit, Funktionen selbst zu definieren und bei Bedarf aufzurufen und auszuführen. Dies geschieht mit der DEF FN-Anweisung, die vor dem ersten Aufruf definiert weden muß. Aus diesem Grunde stehen DEF FN-Anweisungen meistens am Programmanfang.

Eine Funktion frei zu definieren ist besonders dann sinnvoll, wenn sie häufig im Programm aufgerufen wird. Durch die Definition spart man nicht nur an Programmieraufwand, sondern auch an Speicherplatz. Kommt die hinter DEF FN stehende Funktion jedoch nur ein- oder zweimal vor, ist es meist sinnvoller, sie an der betreffenden Programmstelle als normale Anweisung auszuführen.

Einen Nachteil hat die DEF FN-Anweisung trotzdem, denn Sie darf nur eine Variable verwalten, die die Aufrufanweisung FN als Argument enthalten kann. Soll z.B. in einem Programm häufig die gleiche Formel mit einem variablen Parameter (Argument) durchgerechnet werden, kann dieser hinter FN aufgeführt werden. Anders ist es bei der Volumenberechnung eines Quaders, in der Länge, Breite und Höhe in Variablen abgelegt sind. Zwar können wir auch diese Berechnung mit Hilfe einer Funktionsdefinition durchführen, müssen aber die Variablenwerte gesondert eingeben. Wie dies funktioniert, zeigt das folgende Beispiel:

```
10 DEF FN v(x) = l * b * h
20 INPUT "Laenge"; l
30 INPUT "Breite"; b
40 INPUT "Hoehe"; h
50 PRINT "Das Volumen betraegt"; FN v(x)
60 GOTO 20
```

In Zeile 10 wird die Funktion v definiert. v ist hier kein Variablen-, sondern der Funktionsname. Das x in Klammern ist in diesem Fall ein Blindargument, da wir die drei Variablen l, b und h gesondert eingeben.

Die Definition lautet nun, daß die Variablen l, b und h miteinander zu multiplizieren sind, wenn die Funktion in Form von FN v(x) aufgerufen wird. Dies geschieht in Zeile 50, nachdem den Variablen mit Hilfe von INPUT-Anweisungen Werte zugeordnet wurden.

Je nach Größe der Funktion benötigt die FN-Anweisung weit weniger Platz als die komplette Funktion.

Im folgenden Beispiel benötigen wir zur Quadratwurzelberechnung nur eine Variable x, die wir als Argument von DEF FN und FN verwenden wollen. Wichtig ist nur, daß sie in der Definition der Funktion enthalten ist:

```
10 DEF FN v(X) = SQR(x)
20 x = 25
30 PRINT FN v(x)
40 PRINT FN v(49)
50 a = 64
60 PRINT FN v(a)
```

In diesem Fall ist das Argument x auch Bestandteil der SQR-Funktion. Wird jetzt der Variablen x während des Programmablaufs ein Wert zugeordnet, wie hier in Zeile 20, dient dieser Wert bei Aufruf der Funktion als

Argument (Zeile 30). Dabei kann das Argument auch aus einer Konstanten (Zeile 40) oder einer anderen Variablen bestehen (Zeile 60). Nach Ablauf des Programms erscheinen folgende Ergebnisse:

```
 5
 7
 8
Ready
```

4.5 Unterprogramme

Frei definierte Funktionen können immer nur eine Programmzeile umfassen. Dies ist in vielen Fällen sinnvoll und völlig ausreichend. Nicht selten kommt es aber vor, daß ganze Programmteile, die aus einer Vielzahl von Zeilen bestehen, während des Programmablaufs wiederholt ausgeführt werden müssen.

Solche Programmteile können nun als Unterprogramm definiert und an jeder beliebigen Stelle des Hauptprogramms aufgerufen werden. Kein Wunder also, daß Unterprogramme noch mehr als Funktionen ein zeit- und platzsparendes Programmieren unterstützen.

Unterprogramme sind also komplett in sich abgeschlossene BASIC-Programme, die alle Befehle und Anweisungen enthalten dürfen. Sie müssen allerdings mit einer RETURN-Anweisung abschließen, um nach ihrer Abarbeitung wieder in das Hauptprogramm zurückzuspringen. Sinnvollerweise setzt man sie meist an das Programmende. Das Hauptprogramm muß dann allerdings mit END abschließen, um nicht in das Unterprogramm hineinzulaufen.

Jeder Aufruf erfolgt mit einer GOSUB-Anweisung, welcher die Zeilennummer folgt, mit der das Unterprogramm beginnt. Nach dem Rücksprung wird mit der Anweisung fortgefahren, die der GOSUB-Anweisung folgt. Hier ein einfaches Beispiel:

```
10 REM Hauptprogramm
20 GOSUB 100
30 GOSUB 200
40 GOSUB 300
50 GOSUB 200
60 END
100 REM erstes Unterprogramm
110 PRINT "Hier ist das erste Unterprogramm"
120 RETURN
200 REM zweites Unterprogramm
210 PRINT "Hier ist das zweite Unterprogramm"
220 RETURN
300 REM drittes Unterprogramm
310 PRINT "Hier ist das dritte Unterprogramm"
320 RETURN
```

Nach Eingabe von RUN folgt:

```
Hier ist das erste Unterprogramm
Hier ist das zweite Unterprogramm
Hier ist das dritte Unterprogramm
Hier ist das zweite Unterprogramm
Ready
```

Zeile 20 ruft das erste Unterprogramm auf, das mit Zeile 100 beginnt. Nach dessen Abarbeitung fährt das Programm mit Zeile 30 fort, wo das zweite Unterprogramm aufgerufen wird. Dasselbe wiederholt sich in Zeile 40 mit dem dritten und in Zeile 50 nochmals mit dem zweiten Unterprogramm. Damit das Programm anschließend abbricht, steht in Zeile 60 die END-Anweisung. Würde sie fehlen, würde das Hauptprogramm ins erste Unterprogramm geraten. Da es in diesem Fall nicht durch GOSUB aufgerufen wurde, erscheint dann die Fehlermeldung:

```
Unexpected RETURN in 120
Ready
```

4.6 Menütechnik

Vielleicht haben Sie schon einmal mit professionellen Programmen gearbeitet, die Sie fertig gekauft haben. Solche Programme zeichnen sich oft durch eine besondere Bedienerfreundlichkeit aus. Sie starten das Programm, und auf dem Bildschirm erscheinen Anweisungen, die Ihnen mitteilen, wie Sie das Programm nutzen können.

Diese Anweisungen werden in der Computertechnik auch Menü genannt. Sie sind so abgefaßt, daß auch derjenige damit arbeiten kann, der sonst über keine weiteren Programmierkenntnisse verfügt. Durch Drücken bestimmter Tasten werden einzelne Menüpunkte aufgerufen, die wiederum in verschiedene Teilprogramme oder untergeordnete Menüs verzweigen. Nach Abarbeitung der jeweiligen Teilroutinen gelangt man meistens wieder in das Hauptmenü zurück.

Auch wir wollen hier ein menügesteuertes Programm betrachten, das Volumenberechnungen für verschiedene Körper durchführt. Hier zunächst einmal das Programmlisting:

```
10 REM Diverse Volumenberechnungen
20 :
30 CLS
40 PRINT "       VOLUMENBERECHNUNG"
50 PRINT
60 PRINT "Quader                1"
70 PRINT "Zylinder              2"
80 PRINT "Kegel                 3"
90 PRINT "Pyramide              4"
100 PRINT "Kugel                5"
110 PRINT
120 PRINT "E n d e              e"
130 PRINT
140 PRINT "Bitte waehlen Sie"
150 PRINT
160 m$=INKEY$
170 IF m$="e" THEN END
180 m=VAL(m$)
190 IF m<1 OR m>5 THEN 160
200 ON m GOSUB 250, 400, 540, 680, 820
210 GOTO 10
220 :
```

```
230 :
240 :
250 REM Unterprogramm fuer Quader
260 PRINT "Quader"
270 INPUT "Laenge"; l
280 INPUT "Breite"; b
290 INPUT "Hoehe"; h
300 PRINT
310 v=l*b*h
320 PRINT "Das Volumen betraegt"; v
330 PRINT
340 PRINT "Weiter, beliebige Taste druecken"
350 a$=INKEY$:IF a$="" THEN 350
360 RETURN
370 :
380 :
390 :
400 REM Unterprogramm fuer Zylinder
410 PRINT "Zylinder"
420 INPUT "Durchmesser"; d
430 INPUT "Hoehe"; h
440 v = d*d*PI*h/4
450 PRINT
460 PRINT "Das Volumen betraegt"; v
470 PRINT
480 PRINT "Weiter, beliebige Taste druecken"
490 a$=INKEY$:IF a$="" THEN 490
500 RETURN
510 :
520 :
530 :
540 REM Unterprogramm fuer Kegel
550 PRINT "Kegel"
560 INPUT "Durchmesser"; d
570 INPUT "Hoehe"; h
580 v = d*d*h*PI/12
590 PRINT
600 PRINT "Das Volumen betraegt"; v
610 PRINT
620 PRINT "Weiter, beliebige Taste druecken"
630 a$=INKEY$:IF a$="" THEN 630
640 RETURN
650 :
```

```
660 :
670 :
680 REM Unterprogramm fuer Pyramide
690 PRINT "Pyramide"
700 INPUT "Grundseite"; l
710 INPUT "Hoehe"; h
720 v=l*l*h/3
730 PRINT
740 PRINT "Das Volumen betraegt"; v
750 PRINT
760 PRINT "Weiter, beliebige Taste druecken"
770 a$=INKEY$:IF a$="" THEN 770
780 RETURN
790 :
800 :
810 :
820 REM Unterprogramm fuer Kugel
830 PRINT "Kugel"
840 INPUT "Durchmesser"; d
850 v=d*d*d/8*PI*4/3
860 PRINT
870 PRINT "Das Volumen betraegt"; v
880 PRINT
890 PRINT "Weiter, beliebige Taste druecken"
900 a$=INKEY$:IF a$="" THEN 900
910 RETURN
```

Nachdem Sie dieses Programm gestartet haben, erscheint zunächst einmal das Menü auf dem Bildschirm:

```
        VOLUMENBERECHNUNG

Quader                  1
Zylinder                2
Kegel                   3
Pyramide                4
Kugel                   5

E n d e                 e

Bitte waehlen Sie
```

Sie haben nun die Möglichkeit, das Volumen wahlweise für Quader, Zylinder, Kegel, Pyramide oder Kugel zu berechnen. Diese Angaben, zusammen mit der dazugehörigen Ziffer, nennt man Menüpunkte. Drücken Sie nun eine 1, gelangen Sie in das Teilprogramm, welches das Volumen eines Quaders berechnet. Wenn Sie die 2 drücken, können Sie das Volumen eines Zylinders berechnen usw. Möchten Sie das Programm abbrechen, drücken Sie die Taste e.

Uns interessiert hier in erster Linie der Aufbau dieses menügesteuerten Programms:

Zunächst geben die Zeilen 10 bis 150 das Menü auf dem Bildschirm aus. aus. Davor jedoch löscht Zeile 30 den Bildschirm.

Zeile 160 wartet jetzt mit INKEY$ auf einen Tastendruck. Dabei wird der Stringvariablen m$ ein Wert zugeordnet. Die IF-Anweisung in Zeile 170 prüft dann, ob Sie ein "e" eingegeben haben, wenn Sie das Programm verlassen möchten. In diesem Fall wird dann die Anweisung END ausgeführt.

Die eigentlichen Menüpunkte 1 bis 5 werden zunächst ebenfalls in der Stringvariablen m$ gespeichert, weil die INKEY$-Anweisung nur Strings annehmen kann. Für uns ist es aber wesentlich bequemer, wenn wir die Zahlenwerte in Fließkommaschreibweise zur Verfügung haben. Die Umformumg nimmt Zeile

```
180 m = VAL(m$)
```

vor. Enthält das Argument der VAL-Anweisung Ziffernwerte, die auch für numerische Werte zulässig sind, formt sie diese ins numerische Fließkommaformat um, so daß wir mit ihnen mathematische Operationen ausführen können. In unserem Fall liegen nun also die eingegebenen Ziffern im Fließkommaformat vor und sind der Variablen m zugeordnet.

Jedes gute Menü führt auch eine Prüfung durch, ob die eingegebenen Werte zulässig sind. In unserem Programm übernimmt dies Zeile 190:

```
190 IF m<1 OR m>5 THEN 160
```

Die IF-Anweisung fragt, ob der Variablenwert M kleiner als 1 oder größer als 5 ist, denn wir können hier ja nur die Menüpunkte 1 bis 5 verarbeiten. Ist eine dieser beiden Bedingungen erfüllt (Boolesche Operation OR),

springt das Programm wieder zurück in Zeile 160 zur INKEY$-Anweisung und erwartet einen neuen Tastendruck. Dieser Vorgang ist von außen her nicht sichtbar. Drücken Sie nämlich eine falsche Taste, geschieht scheinbar gar nichts, denn das Programm hängt in einer Warteschleife.

Sämtliche Teilprogramme, die die einzelnen Volumenberechnungen durchführen, sind hier als Unterprogramme definiert und schließen deshalb jeweils mit einer RETURN-Anweisung ab. Ihr eigentlicher Aufbau braucht uns nicht weiter zu interessieren, denn solche Programme haben wir in ähnlicher Form schon kennengelernt.

Was uns aber interessiert ist die Art und Weise, wie sie aufgerufen werden. Wir wissen bereits, daß die GOSUB-Anweisung ein Unterprogramm aufruft und dieses nach der Ausführung wieder an die Aufrufstelle zurückkehrt. In Zeile 200 finden wir GOSUB in einer anderen Form vor:

```
200 ON m GOSUB 250, 400, 540, 680, 820
```

Nun wird uns auch klar, warum wir die Zahlenwerte ins numerische Format umgewandelt haben. Diese Anweisung fragt nämlich den Wert der Fließkommavariablen m ab und entscheidet dann aufgrund des Wertes, welches Unterprogramm aufgerufen wird:

Wert von m	Unterprogramm ab Zeile
1	250
2	400
3	540
4	680
5	820

Somit entspricht die erste Zahl hinter ON m GOSUB der Zeile, die aufgerufen wird, wenn m den Wert 1 erhält, die zweite, wenn m den Wert 2 erhält usw. Dasselbe hätten wir auch mit folgenden 5 IF-Anweisungen erreicht:

```
IF m = 1 THEN GOSUB 250
IF m = 2 THEN GOSUB 400
IF m = 3 THEN GOSUB 540
IF m = 4 THEN GOSUB 680
IF m = 5 THEN GOSUB 820
```

Wir sehen jedoch, daß die ON...GOSUB-Anweisung bequemer zu programmieren ist und weniger Speicherplatz beansprucht.

Nach Abarbeitung der einzelnen Unterprogramme springt die GOTO-Anweisung in Zeile 210 wieder an den Programmanfang und gibt das Menü erneut aus.

In diesem Zusammenhang sei noch erwähnt, daß es für GOTO die ähnliche Anweisung ON...GOTO gibt. Hier ein Beispiel:

```
100 ON a GOTO 100, 200, 300
```

Ist der Inhalt der Variablen a gleich 1, findet ein Sprung in Zeile 100 statt, ist er gleich 2, wird in Zeile 200 gesprungen usw.

4.7 Fehlersuche und Fehlerbehandlung

In diesem Abschnitt wollen wir uns mit der Fehlersuche und der Fehlerbehandlung beschäftigen. Jeder Programmierer, selbst der erfahrenste, bringt kaum ein Programm zustande, das auf Anhieb fehlerfrei läuft. Dabei nimmt die Wahrscheinlichkeit, daß Fehler auftreten, bei längeren und komplizierteren Programmen zu und es wird immer schwieriger, die Fehler zu lokalisieren.

Viele Fehler entstehen aus Unachtsamkeit bei der Programmierung. So wird z.B. ein Variablenname oder Schlüsselwort falsch geschrieben oder eine Zahl falsch eingegeben. Wie beim legendären zerstreuten Professor hat schon mancher A gesagt, B geschrieben, C gedacht und in Wirklichkeit war D gemeint. Viele waren nahe daran, ihre Nerven zu verlieren, wenn ein Programm nicht so ablief, wie es ablaufen sollte.

Wir können aber lernen, auf angemessene Weise mit den Fehlern umzugehen oder besser noch, sie von vornherein zu vermeiden. So gibt es verschiedene Arten von Fehlern, die wir im folgenden kurz betrachten wollen.

Da sind einmal die Leichtsinnsfehler, die durch mangelnde Konzentration oder Unachtsamkeit entstehen. Subtilere und deshalb gewöhnlich schwerwiegendere Fehler sind solche, die auf mangelndem Verständnis für die Arbeitsweise von bestimmten Anweisungen beruhen, die dem Computer erteilt werden. Oft wird verkannt, daß Situationen auftreten können, die es

gilt, in den Griff zu bekommen. Irren ist menschlich, und ein Programmierer ist ein Mensch, der Computer dagegen nicht.

Als weiteres gibt es Fehler, die nicht aufgrund eines menschlichen Irrtums entstehen, sondern die bei der Programmerstellung bewußt eingeplant werden. Komfortablere Anwenderprogramme enthalten z.B. oft Anweisungen, die feststellen sollen, ob ein Peripheriegerät, wie beispielsweise der Drukker, eingeschaltet ist. Ist dies nicht der Fall, gibt der Computer eine Fehlermeldung aus, die abgefragt wird, um gegebenenfalls entsprechende Maßnahmen zu ergreifen. Bei nicht eingeschaltetem Drucker könnte dies so aussehen, daß der Bediener aufgefordert wird, das Gerät einzuschalten.

Derartige Maßnahmen werden jedoch noch drastischer praktiziert: Viele Softwareanbieter möchten nämlich nach Möglichkeit verhindern, daß ihre Programme auf Diskette unzulässigerweise vervielfältigt werden. Dazu bringen sie auf der Diskette extra Fehler an. Wird nun versucht, dieses Programm zu kopieren und "normal" zu starten, ist es nicht lauffähig, da bestimmte eingebaute "Fehler", die zur einwandfreien Arbeitsweise des Programms unbedingt erforderlich sind, nicht vorhanden sind. Im Extremfall kann in einer solchen Situation die Diskette sogar zerstört werden.

Eine Grundvoraussetzung zur Vermeidung von Fehlern ist zunächst einmal eine klare Zielsetzung von dem, was ein Programm alles ausführen soll. Viele Programmierer verwenden zur Planung spezielle Fluß- oder Ablaufdiagramme, die jede einzelne Anweisung enthalten. Wir wollen hier wegen unserer relativ einfachen Beispiele auf derartige Diagramme verzichten, es aber dennoch nicht versäumen, ein Konzept aufzustellen. Betrachten wir unter diesem Aspekt nochmals das Programm, das für fünf verschiedene Körper das Volumen berechnet. Das Konzept dazu könnte folgendermaßen aussehen:

1. Zielsetzung: Was für ein Programm will ich schreiben? Zu welchem Zweck möchte ich es einsetzen?

   Antwort: Ich wünsche ein Programm, das das Volumen von Quadern, Zylindern, Kegeln, Pyramiden und Kugeln berechnet.

2. Aufbau: Aus welchen Teilelementen soll sich das Programm zusammensetzen?

Antwort: Eingabemenü
Unterprogramm für Quader
Unterprogramm für Zylinder
Unterprogramm für Kegel
Unterprogramm für Pyramide
Unterprogramm für Kugel

3. Detaillierte Ausarbeitung für jedes Teilelement:

Eingabemenü: Bildschirm löschen
Menüpunkte ausgeben
Menüpunkte abfragen
Gewähltes Unterprogramm ausführen
Zurück ins Eingabemenü

Unterprogramm für Quader:
Meldung "Quader" ausgeben
Eingabewerte lesen (Länge, Breite, Höhe)
Volumen berechnen
Volumen ausgeben
Warteschleife auf Tastendruck
Rückkehr ins Hauptprogramm

Auf die Ausarbeitung der anderen Unterprogramme wollen wir hier verzichten, da sie ähnlich aufgebaut sind.

Nun können wir uns an den Computer setzen, geeignete Anweisungen wählen und schließlich das Programm eingeben. Bevor wir es ablaufen lassen, speichern wir es sicherheitshalber auf Diskette oder Kassette ab.

Wir haben das Programm zwar jetzt im Computer, aber mit hoher Wahrscheinlichkeit läuft es noch nicht fehlerfrei ab. Dank unseres strukturierten Aufbaus sind wir aber in der Lage festzustellen, in welchem Teilprogramm ein Fehler auftritt.

Häufig haben wir die Schreibregeln für die BASIC-Programmiersprache verletzt. In einem solchen Fall erscheint die Fehlermeldung

```
Syntax error in (Zeilennummer)
Ready
```

und das Programm bricht ab. Nun können wir die fehlerhafte Zeile listen und korrigieren.

Manchmal ist es sinnvoll, das Programm an bestimmten Stellen anzuhalten, um den Inhalt einiger Variablen zu überprüfen. Gibt z.B. das Unterprogramm für Quader ein falsches Ergebnis aus, fügen wir folgende Zeile ein, die wir später wieder löschen:

```
315 STOP
```

Das Programm hält jetzt jedesmal in Zeile 315 an und meldet sich mit

```
Break in 315
Ready
```

Nun können wir prüfen, ob die Variablen l, b, h und v die richtigen Werte oder Zwischenergebnisse enthalten. Dies geschieht ganz normal mit der PRINT-Anweisung im Direktmodus. Wir können uns aber Tipparbeit sparen, indem wir statt PRINT nur ein Fragezeichen setzen, z.B.

```
? v
 5.6
Ready
```

Wurde ein Programm mit BREAK unterbrochen, kann es mit

```
CONT
```

fortgesetzt werden. In unserem Fall würde es dann mit Zeile 320 fortfahren. CONT funktioniert allerdings nicht, wenn bei der Unterbrechung Änderungen im Programm vorgenommen wurden. Dann nämlich erscheint die Fehlermeldung

```
Cannot CONTinue
Ready
```

4.7.1 Ablaufverfolgung

Eine zusätzliche Hilfe zur Auffindung von Fehlern ist die Ablaufverfolgung. Ist sie eingeschaltet, erscheinen beim Programmablauf die Nummern der gerade abgearbeiteten Zeilen auf dem Bildschirm. Die Ablaufverfolgung ist eine große Hilfe, wenn festgestellt werden soll, ob Programmverzweigungen richtig funktionieren. Jeder Fehler kann dann anhand der ausgegebenen Zeilennummern sofort festgestellt werden.

Die Ablaufverfolgung wird mit

TRON

eingeschaltet, worauf das Programm normal mit RUN zu starten ist.

TROFF

schaltet sie wieder ab. TRON und TROFF können aber auch in Programme eingebaut und somit im Programmodus betrieben werden.

4.7.2 Programmierte Fehlerbehandlung

In diesem Abschnitt behandeln wir nun diejenigen Fehler, die bewußt mit eingeplant werden. Das CPC-BASIC stellt uns Anweisungen zur Verfügung, mit denen wir Fehler auffangen und bearbeiten können. Dies geschieht zunächst einmal mit der ON ERROR GOTO-Anweisung, der eine Zeilennummer folgt. Tritt ein Fehler auf, springt das Programm in diese Zeile und führt dann die Fehlerbehandlung durch. Anschließend kehrt RESUME wieder an die Stelle zurück, an der die Fehlermeldung auftrat.

Für den Einsatz der ON ERROR GOTO-Anweisung gibt es viele Anwendungsbereiche. So kann das Programm z.B. feststellen, ob während einer Berechnung Ergebnisse auftreten, die größer als der zulässige Wertebereich von ca. 1E+38 sind, oder ob die Quadratwurzel aus einem negativen Wert gezogen werden soll.

Im Normalfall bricht das Programm bei Auftreten eines Fehlers ab und gibt eine entsprechende Fehlermeldung aus. Bei früheren BASIC-Versionen war es oft ärgerlich, wenn Programme mit langer Laufzeit aufgrund von Fehlern plötzlich abbrachen und nach deren Behebung neu gestartet werden mußten.

Um zu demonstrieren, wie die ON ERROR GOTO-Anweisung arbeitet, wollen wir nun ein kleines Programm betrachten. Wir wissen bereits aus einem früheren Beispiel, daß das Argument der SQR-Funktion nicht negativ sein darf, da sonst die Fehlermeldung "Improper argument" auftritt und das Programm abbricht. Das folgende Programm berechnet ebenfalls die Quadratwurzel, bricht aber im Fehlerfall nicht ab.

```
10 ON ERROR GOTO 1000
20 INPUT "Argument"; a
30 PRINT SQR(a)
40 GOTO 20
50 :
1000 IF ERR=5 THEN PRINT "Bitte nur positives Argument"
1010 RESUME NEXT
```

Die ON ERROR GOTO-Anweisung in Zeile 10 bestimmt, daß das Programm im Falle eines Fehlers, ganz gleich welcher Art, in Zeile 1000 verzweigen soll. Zeile 20 fordert das Argument an, dessen Quadratwurzel im Normalfall dann in Zeile 30 ausgegeben wird.

Bei der Eingabe eines negativen Argumentes springt das Programm jedoch in Zeile 1000. Dort wird die reservierte Variable ERR abgefragt, ob sie den Wert 5 enthält, der gleichbedeutend mit "Improper argument" ist. Liegt dieser Fehler vor, erscheint die Meldung "Bitte nur positives Argument".

Zeile 1010 enthält nun die RESUME-Anweisung, die die Fehlerbehandlung abschließt und ins aufrufende Programm zurückkehrt. Dies geschieht ähnlich wie der Rücksprung aus einem Unterprogramm, jedoch sind hierbei noch einige Feinheiten zu beachten. In unserem Programm steht nämlich nicht nur RESUME, sondern

```
RESUME NEXT
```

Dadurch fährt das Programm erst mit derjenigen BASIC-Anweisung fort, die direkt der Anweisung folgt, die den Fehler hervorgerufen hat. In unserem Fall ist dies die GOTO-Anweisung in Zeile 40, die wiederum in Zeile 20 springt, wo ein neues Argument angefordert wird. Es gibt aber noch zwei weitere Varianten der RESUME-Anweisung. Schließt z.B. eine Fehlerbehandlungsroutine mit

```
RESUME 500
```

ab, so springt das Programm anschließend in Zeile 500 und setzt dort seine Ausführung fort. Erscheint dagegen nur

```
RESUME
```

ohne Zusatz, so wird die fehlererzeugende Anweisung nach Abarbeitung der Fehlerroutine nochmals ausgeführt. Dies ist jedoch nur dann sinnvoll, wenn die Ursache des Fehlers behoben ist und bei der erneuten Abarbeitung kein Fehler mehr auftreten kann.

5 Listenverarbeitung

In den vorangegangenen Kapiteln haben wir nur mit relativ kleinen Datenmengen gearbeitet, die wir über die INPUT-Anweisung oder über DATA-Zeilen ins Programm eingelesen haben. Zu Übungszwecken mag dies für Sie vielleicht ganz interessant gewesen sein, zum Aufbau einer professionellen Dateiverwaltung gehört aber noch einiges mehr.

Programme, die große Datenmengen verarbeiten, nennt man Datenverarbeitungs- oder Datenerfassungsprogramme. Sie arbeiten meist mit einem Massenspeicher, der in vielen Fällen aus einem oder mehreren Floppylaufwerken, oft aber auch aus einer Festplatte besteht, die weitaus mehr Daten als eine Diskette aufnehmen kann. Darüber hinaus besitzt der CPC 6128 eine Speicherbank mit zusätzlichen 64K, die als RAM-Floppy genutzt werden kann. Doch hierüber später mehr.

Um ein "echtes" Datenerfassungsprogramm zu schreiben, reicht aber unser bisheriges Wissen noch nicht aus; dazu benötigen wir weitere Kenntnisse und Erfahrungen, die in diesem Kapitel behandelt werden. Den Massenspeicher wollen wir vorläufig allerdings nicht berücksichtigen, sondern uns statt dessen zur Eingabe von Daten weiterhin mit DATA-Zeilen begnügen.

Auf den folgenden Seiten finden sie alles, was Sie zu einer Listen- oder Tabellenverarbeitung benötigen. Zunächst lernen wir verschiedene Stringverarbeitungstechniken und die formatierte Ausgabe kennen. Anschließend befassen wir uns mit Feldern, die eine Vielzahl von Datenelementen enthalten können und die einen wichtigen Platz in der Datenverarbeitung einnehmen. Nicht zu kurz kommt dabei auch das Auffinden und Sortieren von Daten. die in Feldern abgelegt sind, wodurch die Datenverarbeitung erst richtig interessant wird.

5.1 Stringmanipulationen

In den vorherigen Kapiteln haben wir bereits wiederholt mit Strings gearbeitet. Wir wissen, daß sie einen gesonderten Datentyp darstellen und daß Stringvariablen Zeichenketten enthalten, die in der Regel aus ASCII-Zeichen bestehen.

Allerdings ist uns noch wenig über Anweisungen bekannt, die Stringmanipulationen durchführen, d.h. Strings aneinanderreihen, trennen oder in sonstiger Weise beeinflussen.

Gegenüber der reinen Zahlenverarbeitung bringt die Stringverarbeitung eine gewisse Problematik mit sich. Wir kennen bereits die drei Datentypen Fließkomma, Integer und String. Fließkomma- und Intergerzahlen benötigen immer die gleiche Menge an Speicherplatz. Ändert sich ihr Variablenwert, so wird der alte Wert einfach von dem neuen überschrieben. Aus diesem Grund können numerische Variablen während des Programmablaufs immer an der gleichen Stelle im Speicher plaziert sein.

Bei Strings ist dies anders, da sie verschiedene Längen annehmen können. Ein String im CPC-BASIC kann beispielsweise zwischen 0 und 255 Zeichen enthalten.

Dies wäre unerheblich, wenn am Anfang des Programms die jeweiligen Stringvariablen eine feste Länge erhielten, die dann, wie bei den numerischen Variablen, während des gesamten Programmablaufs konstant bleibt.

Wie wir jedoch gleich sehen werden, können die Längen der Stringvariablenwerte sich ständig verändern. Betrachten wir dazu ein einfaches Beispiel:

```
10 INPUT "Bitte ein nettes Wort eingeben"; a$
20 PRINT a$
30 PRINT "Dankeschoen!"
40 GOTO 10
```

Dieses Programm hängt in einer Schleife und fordert Sie jedesmal erneut auf, ein nettes Wort einzugeben. Dies geschieht mit der INPUT-Anweisung in Zeile 10, wo der eingegebene String der Variablen a$ zugeordnet wird. Lassen wir jetzt das Programm einmal ablaufen:

```
Bitte ein nettes Wort eingeben? Hi!
Hi!
Dankeschoen
Bitte ein nettes Wort eingeben? Guten Morgen!
Guten Morgen!
Dankeschoen
Bitte ein nettes Wort eingeben? Hallo - wie geht's?
Hallo - wie geht's?
Dankeschoen
usw.
```

Im ersten Durchgang wurde das Wort "Hi!" eingegeben, das insgesammt drei Zeichen umfaßt, im zweiten das Wort "Guten Morgen!" mit 13 Zeichen und schließlich "Hallo - wie geht's?" mit 19 Zeichen. Die Länge des Variableninhalts von a$ ändert sich also ständig.

Um dieses Problem in den Griff zu bekommen, haben sich die Erfinder des BASIC-Interpreters etwas einfallen lassen. Im Speicher befindet sich eine sogenannte Variablentabelle, in der alle Variablen, ganz gleich welchen Typs, eingetragen sind. Da numerische Variablen immer den gleichen Speicherplatz benötigen, sind ihre Werte gleich mit in der Variablentabelle enthalten. Für Stringvariablen ist in der Tabelle lediglich ein Zeiger enthalten, der die Länge und die Anfangsadresse des Strings im Speicher enthält. Die Werte von Stringvariablen füllen den noch freien Speicher von oben nach unten, d.h. von der höchsten zur niedrigsten, nicht anderweitig belegten, freien Adresse, auf.

Wird nun eine neue Stringvariable angelegt, so füllt ihr Wert immer mehr den Speicher nach unten hin auf. Das gleiche geschieht auch mit der Variablen a$ in unserem Programm, wenn sie einen neuen Wert erhält. Dabei wird der Zeiger, auch Descriptor genannt, auf den neuen Wert gesetzt. Der alte Wert von a$ wird also nicht überschrieben und bleibt als "Stringmüll" zurück.

Es ist daher einleuchtend, daß Programme, die viele Strings erzeugen oder umändern, den Speicherplatz immer mehr nach unten hin auffüllen und gleichzeitig eine Menge "Stringmüll" nicht mehr benötigter Strings zurücklassen. Irgendwann ist dann der Zeitpunkt gekommen, daß es so nicht mehr weitergehen kann.

Vielleicht haben Sie schon einmal mit Programmen gearbeitet, die eine umfangreiche Stringverarbeitung durchführten. Plötzlich unterbrach der Computer für einige Sekunden seine Abeit und Sie glaubten bereits, daß irgend etwas nicht stimmt.

Es besteht aber absolut kein Grund zur Sorge, denn in dieser Pause führt der Computer eine Stringmüllbeseitigung (garbage collect) durch. Dabei werden alle Strings, die noch gültig, also in der Variablentabelle aufgeführt sind, nach oben hin neu geordnet und aneinandergefügt, damit wieder genügend Platz für neue Strings frei wird.

Je nach Anzahl der sich im Speicher befindlichen Strings ist die Stringmüllbeseitigung eine Frage der Zeit. Sie wird umso seltener auftreten, je größer der BASIC-Speicher ist und umso weniger Platz das Programm einschließlich der Variablentabelle benötigt.

Sie können die Stringmüllbeseitigung auch künstlich mit der FRE-Anweisung durchführen. Wenn Sie nämlich

```
PRINT FRE(0)
```

eingeben, sagt Ihnen der Computer, wieviele Bytes im Programmspeicher noch frei sind. Dazu muß er nämlich sämtliche im Speicher abgelegten Strings erst einmal neu ordnen. Wenn Sie die FRE-Anweisung direkt nach dem Einschalten oder Rücksetzen des Computers (gleichzeitiges Drücken der CTRL-, SHIFT- und ESC-Taste) ausführen, erhalten Sie

```
PRINT FRE(0)
 43533
Ready
```

Diese Zahl gibt genau die Größe des Arbeitsspeichers wieder, der unter BASIC zur Verfügung steht. Dabei sind allerdings die zusätzlichen 64K RAM nicht berücksichtigt, die nur mit Spezialanweisungen genutzt werden können.

5.1.1 Stringverkettung

Das Pluszeichen dient in BASIC nicht nur zur Addition von Zahlenwerten oder als Vorzeichen, sondern auch zur Verkettung von Strings. Somit ist es möglich, aus Einzelstrings einen Gesamtstring zu erzeugen, der allerdings nicht mehr als 255 Zeichen enthalten darf. Hier ein Beispiel:

```
10 a$ = "SCHNEIDER"
20 b$ = "CPC"
30 c$ = "6128"
40 d$ = "COMPUTER"
50 g$ = a$ + b$ + c$ + d$
60 PRINT g$
```

Nach Eingabe von RUN erscheint

```
SCHNEIDERCPC6128COMPUTER
Ready
```

Hier wurde aus den vier Einzelstrings a$, b$, c$ und d$ der Gesamtstring g$ gebildet. In der vorliegenden Form sollten wir ihn aber nicht belassen und deshalb Leerzeichen zwischen die einzelnen Wörter setzen. Aus diesem Grunde schreiben wir Zeile 50 etwas um:

```
50 g$ = a$ + " " + b$ + " " + c$ + " " + d$
```

Jetzt erscheint g$ in übersichtlicher Form:

```
SCHNEIDER CPC 6128 COMPUTER
Ready
```

5.1.2 LEN

Mit LEN können Sie die Länge eines Strings bestimmen. Als Beispiel wollen wir nochmals unser letztes Programm betrachten. Nachdem Sie es ausgeführt haben, also sämtliche Variablen angelegt sind, geben Sie folgendes ein (statt PRINT setzen wir hier ein Fragezeichen):

```
? LEN(g$)
 27
Ready

? LEN(a$)
 9
Ready
```

Wir haben soeben die Längen der Strings g$ und a$ bestimmt. Der Gesamtstring g$ ist 27 Zeichen, der String a$ 9 Zeichen lang.

Die LEN-Anweisung sollten Sie immer dann verwenden, wenn es darum geht, die Länge von Strings zu prüfen. Dies kann besonders bei Programmen mit vielen Stringoperationen der Fall sein.

Beim Versuch, einen String mit mehr als 255 Zeichen Länge zu erzeugen, entsteht die Fehlermeldung

String too long in (Zeilennummer)

5.1.3 LEFT$ und RIGHT$

Betrachten wir folgendes Beispiel:

```
A$ = "ABCDEFGHIJKLMN"
Ready

? LEFT$(A$,5)
ABCDE
Ready

? RIGHT$(A$,3)
LMN
Ready
```

Damit erkären sich die Anweisungen LEFT$ und RIGHT$ eigentlich schon von selbst. So erzeugt LEFT$(a$,X) einen neuen String, der aus den ersten X Zeichen des Strings a$ besteht. Bei RIGHT$(a$,X) entsteht ebenfalls ein neuer String, der aber die rechten X-Zeichen des Strings a$ enthält.

5.1.4 MID$

MID$(a$,X,Y) erzeugt einen String aus a$, der bei der Position X beginnt und Y Zeichen lang ist. Auch hierzu ein Beispiel:

```
a$ = "ABCDEFGHIJKLMN"
Ready

? MID$(a$,8,7)
HIJKLMN
Ready
```

Hier wird aus dem String A$ ein neuer gebildet, der mit dem 8. Zeichen beginnt und 7 Zeichen lang ist.

MID$ kann aber auch in umgekehrter Weise Verwendung finden, indem wir in einen gegebenen String einen neuen einsetzen. Hier ein Beispiel.

```
a$ = "**********************"
Ready

MID$(a$,8,6) = "HALLO!"
Ready
? a$
*******HALLO!*******
Ready
```

In diesem Fall wird der String "HALLO!" in den String a$ eingesetzt und erscheint dort ab dem 8. Zeichen.

Bei allen Stringoperationen ist es egal, ob die Strings als Konstante oder Variable vorliegen. Im letzen Beispiel hätten wir auch den String "HALLO!" der Variablen c$ zuordnen und den Inhalt von a$ ausschreiben können:

```
MID$("**********************",8,6) = c$
```

5.1.5 INSTR

Mit dieser Anweisung ist es möglich, einen String in einem anderen zu suchen, und dessen Position anzugeben, z.B.:

```
a$ = "MARKT & TECHNIK - VERLAG"
Ready

b$ = "TECHNIK"
Ready

? INSTR(a$,b$)
 9
Ready
```

Hier wird untersucht, ob der String b$ im String a$ enthalten ist. Dies ist der Fall, denn das Wort "TECHNIK" beginnt ab Position 9 des Strings a$. Ist der gesuchte String dagegen nicht enthalten, erscheint das Ergebnis 0.

5.1.6 STRING$ und SPACE$

STRING$ erzeugt auf einfache Weise Strings, in denen nur ein Zeichen wiederholt aneinandergereiht ist. Versuchen Sie einmal folgendes:

```
PRINT STRING$(20,"A")
AAAAAAAAAAAAAAAAAAAA
Ready
```

Es entsteht ein String aus 20 Zeichen "A".

SPACE$ erzeugt einen Leerstring mit vorgegebener Länge, wie z.B.:

```
a$ = SPACE$(10)
```

Die hier erzeugte Stringvariable a$ enthält 10 Leerzeichen.

5.1.7 STR$ und VAL

STR$ wandelt einen numerischen Wert in einen String um:

```
a = 1000
Ready

s$ = STR$(a)
Ready

? s$
 1000
Ready

? LEN(s$)
 5
READY.
```

Hier haben wir den numerischen Wert 1000, der in der Variablen a abgelegt ist, in einen String mit dem Namen s$ umgewandelt. Nach außen hin macht es keinen Unterschied, ob wir die Zahl 1000 als String oder als numerische Variable auf dem Bildschirm ausgeben.

Mit Hilfe von LEN stellen wir fest, daß s$ 5 Zeichen lang ist, obwohl die Zahl 1000 nur 4 Ziffern besitzt. Dies hat durchaus seine Richtigkeit, denn die STR$-Anweisung reserviert immer eine Stelle für das Vorzeichen. Ist das Vorzeichen negativ, erscheint vorne ein Minuszeichen, ist es dagegen positiv, erscheint ein Leerzeichen (Blank).

VAL ist die Umkehrfunktion von STR$ und formt einen als String eingegebenen Zahlenwert ins Fließkommaformat um. Hierzu ein Beispiel:

```
10 INPUT "1.Zahl"; a$
20 INPUT "2.Zahl"; b$
30 a = VAL(a$)
40 b = VAL(b$)
50 PRINT a * b
```

Ein Probelauf zeigt folgendes Ergebnis:

```
1. Zahl? 5
2. Zahl? 4
 20
Ready
```

In diesem Programm haben wir die beiden Operanden in Form von Strings eingegeben. Um mit ihnen mathematische Operationen ausführen zu können, müssen sie zunächst einmal ins Fließkommaformat umgewandelt werden. Dies geschieht in Zeile 30 und 40, so daß Zeile 50 schließlich die Multiplikation ausführen kann.

5.1.8 CHR$ und ASC

Sämtlichen Zeichen, einschließlich den unsichtbaren Steuerzeichen, ist intern ein äquivalenter Zifferncode zugeordnet, den man auch ASCII-Code nennt. Wir wollen jetzt einmal versuchen, diesen Code dem Computer zu entlocken:

```
? ASC ("A")
 65
Ready

? ASC ("B")
 66
Ready

? ASC ("Z")
 90
Ready
```

Wir haben dazu die ASC-Anweisung benutzt, die zu jedem Buchstaben den entsprechenden ASCII-Code erzeugt. Ohne diese Prozedur nun für alle 26 Buchstaben durchführen zu müssen, können wir uns leicht denken, daß der Buchstabe C den Code 67, D den Code 68 und Y den Code 89 besitzt.

Diese Codes wollen wir nun dazu verwenden, das Alphabet in etwas ungewöhnlicher Form auf dem Bildchirm auszugeben:

```
10 REM Ausgabe des Alphabets
20 FOR i = 65 TO 90
30 PRINT CHR$(i);
40 NEXT
```

Nach RUN erscheint

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ready
```

Dieses Programm enthält die CHR$-Anweisung, die aus den ASCII-Codes die entsprechenden lesbaren Zeichen erzeugt.

Wir wissen bereits, daß Anführungszeichen einen String abgrenzen, aber sonst nicht in ihm enthalten sein dürfen. Es gibt jedoch einen Trick, dies doch zu erreichen, denn das Anführungszeichen entspricht dem ASCII-Code 34:

```
10 a$ = "Sie sagte "+CHR$(34)+"Guten Morgen!"+CHR$(34)
20 PRINT a$
```

Nach der Ausführung erscheint:

```
Sie sagte "Guten Morgen!"
Ready
```

Sie sehen also, daß Sie auch "unzulässige" Zeichen mit CHR$ erzeugen können.

Abschließend noch ein String, der Sie verblüffen wird.

```
10 a$ = "Erste Zeile"+CHR$(13)+CHR$(10)+"Zweite Zeile"
20 PRINT a$
```

Nach der Ausführung erscheint

```
Erste Zeile
Zweite Zeile
Ready
```

Dieser String wird also in zwei Zeilen untereinander ausgegeben. Dies liegt an den ASCII-Codes 13 und 10, die einen Wagenrücklauf (Carriage Return) und einen Zeilenvorschub (Line Feed) zur Folge haben. Diese beiden Zeichen haben die gleiche Wirkung wie die RETURN-Taste.

5.1.9 Digitaluhr

Das nachfolgende Programm simuliert eine Digitaluhr mit Hilfe von Stringanweisungen. Nach dem Starten werden Sie aufgefordert, die Uhr zu stellen, indem Sie nacheinander Stunden, Minuten und Sekunden eingeben.

```
10 REM Digitaluhr
20 MODE 1
30 INPUT "Stunden";h
40 INPUT "Minuten";m
50 INPUT "Sekunden";s
60 CLS
70 z$="Es ist jetzt 00 Uhr 00 Min. 00 Sek."
80 EVERY 50 GOSUB 100
90 GOTO 90
100 s=s+1
110 IF s=60 THEN m=m+1:s=0
120 IF m=60 THEN h=h+1:m=0
```

```
130 IF h=24 THEN h=0
140 LOCATE 1,12
150 MID$ (z$,14,3) = RIGHT$("0"+MID$(STR$(h),2,3),2)
160 MID$ (z$,21,3) = RIGHT$("0"+MID$(STR$(m),2,3),2)
170 MID$ (z$,29,3) = RIGHT$("0"+MID$(STR$(s),2,3),2)
180 PRINT z$
190 RETURN
```

Strenggenommen lernen wir hier nicht nur Stringmanipulationen kennen, sondern erfahren auch, wie der Computer als Zeitzähler eingesetzt werden kann. Das Kernstück des ganzen Programms sind die Zeilen 80 und 90:

```
80 EVERY 50 GOSUB 100
90 GOTO 90
```

Diejenigen Leser, die sich schon einmal mit Maschinenprogrammierung befaßt haben, kennen sicher den Begriff "Interrupt". Immer wenn ein äußeres Ereignis auftritt, z.B. der Drucker Daten anfordert, wird das eigentliche Programm im Computer unterbrochen und eine spezielle Interruptroutine angesprungen, die in diesem Fall den Drucker bedient. Ist das Interruptprogramm abgearbeitet, setzt der Computer seine Arbeit an der Stelle fort, an welcher der Interrupt auftrat.

Man kann sich eine Interruptroutine auch als Unterprogramm vorstellen, das nicht nur an den Stellen, die besonders mit GOSUB gekennzeichnet sind, aufgerufen wird, sondern jederzeit, wenn Sie es wünschen oder ein externes Gerät den Befehl dazu gibt. So wird z.B. auch die Tastatur Ihres Computers per Interrupt abgefragt, indem 300mal pro Sekunde eine spezielle Routine angesprungen wird, um festzustellen, ob Sie eine Taste gedrückt haben und wenn ja, welche.

Mit dem CPC 6128 können Sie Interruptroutinen nicht nur in Maschinensprache, sondern auch in BASIC programmieren. Genau dies haben wir bei unserem Uhrenprogramm getan.

Die EVERY-Anweisung sorgt nämlich dafür, daß mit Hilfe eines Timers alle 1/50-tel Sekunde ein interner Zähler um eins erhöht wird. EVERY 50 GOSUB 100 bedeutet demnach, daß das Programm einmal pro Sekunde (50 x 1/50-tel Sek.) unterbrochen wird und das Unterprogramm ab Zeile 100 anspringt. Nach Abarbeitung des Unterprogramms setzt das eigentliche Programm seine Ausführung an der Stelle fort, an der es unterbrochen wurde.

Nachdem die EVERY-Anweisung erteilt wurde, hängt unser Uhrenprogramm in Zeile 90 in einer Endlosschleife, die nur durch zweimaliges Drücken der ESC-Taste unterbrochen werden kann. Trotzdem wird das Unterprogramm ab Zeile 100 einmal pro Sekunde per Interrupt aufgerufen.

Dieses Interruptprogramm steuert die eigentliche Bildschirmanzeige. Zunächst werden die Sekunden um eins erhöht (Variable s). Bei 60 Sekunden beginnt der Sekundenzähler wieder von Null an zu zählen, wobei der Minutenzähler (Variable m) um eins erhöht wird. Ähnliches spielt sich auch zwischen dem Minuten- und dem Stundenzähler (Variable h) ab.

Schließlich bringen die Zeilen 150 bis 170 Stunden, Minuten und Sekunden ins richtige Format und setzen sie gleichzeitig per MID$-Anweisung in den String z$ ein.

Der Ausgabestring z$ soll aber nicht an beliebiger, sondern immer wieder in der Bildschirmmitte erscheinen. Dazu dient die LOCATE-Anweisung, die folgendes allgemeines Format hat:

LOCATE Spalte, Zeile

Die Spalten- und Zeilenzählung beginnt jeweils in der linken oberen Bildschirmecke mit dem Wert 1. Je nach Bildschirmmodus (MODE 0,1 oder 2) können 20, 40 oder 80 Spalten, jedoch immer nur 25 Zeilen angesteuert werden. In unserem Programm bedeutet demnach

LOCATE 1,12

daß die Zeitanzeige in Spalte 1 von Zeile 12 beginnt.

5.1.10 Laufschrift

Mit Hilfe der verschiedenen Stringoperationen können wir eine Laufschrift auf dem Bildschirm erzeugen. Hier ein Beispiel:

```
10 REM Laufschrift
20 MODE 1
30 i=0
40 a$="  Neue Computerbuecher sind ganz toll!  "
50 a$=a$+a$
60 i=i+1:IF i=41 THEN i=1
70 LOCATE 1,12
80 PRINT MID$(a$,i,40)
90 FOR j=1 TO 200:NEXT
100 GOTO 60
```

Das Programm funktioniert denkbar einfach. Der als Laufschrift auszugebende String steht in Zeile 40 und hat eine Länge von 40 Zeichen. Er ist also genauso lang wie eine Bildschirmzeile im 40 Zeichen-Modus (MODE 1).

Zeile 50 reiht den auszugebenden String zweimal aneinander, so daß er jetzt eine Gesamtlänge von 80 Zeichen hat. Der ganze Trick besteht darin, 40 Zeichen des Strings in einer Schleife wiederholt auszugeben. Bei jedem Durchgang verschiebt sich das erste auszugebende Zeichen im String um eine Stelle nach rechts, was durch die MID$-Anweisung in Zeile 80 gesteuert wird. Gleichzeitig erfolgt die Ausgabe infolge der LOCATE-Anweisung immer an der gleichen Stelle, wodurch der Eindruck einer Laufschrift entsteht.

5.2 Formatierte Ausgabe

Die bisher betrachteten PRINT-Anweisungen konnten zwar Werte auf dem Bildschirm ausgeben, hinterließen aber nicht immer das beste Erscheinungsbild. Zwar konnten wir das Aussehen etwas verändern, indem wir hinter PRINT ein Komma oder Semikolon setzten, die Übersichtlichkeit war deshalb aber noch lange nicht gegeben.

Gerade wer Listen und Tabellen mit dem Computer ausgeben möchte, weiß es zu schätzen, wenn in Zahlenkolonnen die Dezimalpunkte untereinander stehen. Mit der ROUND-Anweisung und den bisher behandelten Stringoperationen könnten wir sicher eine solche Formatierung erreichen, wenn auch auf etwas umständliche Weise. Viel einfacher geht es aber mit der PRINT USING-Anweisung, mit der wir uns gleich nachfolgend befassen werden.

Da umfangreiche Listen und Tabellen selten nur auf dem Bildschirm ausgegeben werden, wollen wir uns in diesem Zusammenhang auch mit der Druckerausgabe beschäftigen. Dabei spielt es keine Rolle, welchen Drucker Sie benutzen. Die meisten Drucker sind zwar für die formatierte Listenausgabe ausgerüstet, benötigen jedoch dazu diverse Steuerzeichen, die sich von Fabrikat zu Fabrikat unterscheiden. Die hier vorgestellten Formatierungsanweisungen sind aber so aufgestellt, daß sie gleichermaßen für Bildschirm und Drucker Gültigkeit haben.

Text auf den Drucker auszugeben ist genauso einfach wie auf dem Bildschirm. Achten Sie aber unbedingt darauf, daß Ihr Drucker auch eingeschaltet ist. Anderenfalls hängt der Computer nämlich in einer Warteschleife, die Sie meist nur durch Abschalten oder durch einen totalen Reset (gleichzeitiges Drücken der CTRL-, SHIFT und ESC-Taste) wieder verlassen können. Ihr Programm geht in einem solchen Fall natürlich verloren.

Leider gibt es von BASIC aus keine direkte Abfrage, ob der Drucker eingeschaltet ist. In Kapitel 7 wird aber eine Befehlserweiterung vorgestellt, die dies ermöglicht, ohne daß der Computer in der Warteschleife hängen bleibt.

Wenn Sie einen String a$ auf dem Bildschirm ausgeben, schreiben Sie

```
PRINT a$
```

Möchten Sie den gleichen String jedoch ausdrucken, müssen Sie

```
PRINT#8,a$
```

schreiben. Indem Sie an die PRINT-Anweisung ein Rautenzeichen (#), die Streamnummer 8 (für Drucker) und ein Komma anhängen, beziehen sich sämtliche aufgeführten Variablen, Konstanten oder arithmetischen Ausdrücke dieser Anweisung auf den Drucker.

Für PRINT USING, das wir gleich kennenlernen werden, schreiben Sie entsprechend

```
PRINT#8,USING
```

Somit steht Ihnen nichts mehr im Wege, formatierte Listen auch auf dem Drucker auszugeben.

5.2.1 PRINT USING

Mit PRINT USING können wir Zahlen und Strings formatiert ausgeben. Dabei sind die auszugebenen Stellen in einem Formatstring mit Rautenzeichen (#) vorzugeben. Hier einige Beispiele:

```
PRINT USING "###.##"; 3.187
  3.19
Ready

a = -3.14159 : PRINT USING "####.####"; a
  -3.1420
Ready
```

Manchmal ist es aber notwendig, ein Vorzeichen auszugeben. Es muß dann im Formatstring mit angegeben sein und kann wahlweise vorne oder hinten stehen:

```
PRINT USING "+###.##"; 6.23
+6.23
Ready

PRINT USING "###.##-"; -17.3
 17.30-
Ready
```

Mit PRINT USING können auch Zahlenwerte in Exponentialschreibweise ausgegeben werden. Zu diesem Zweck setzt man hinter das Rautenzeichen vier Pfeile:

```
PRINT USING "+#.##^^^^"; 78.98
+7.90E+01
Ready
```

Darüberhinaus können auch Strings mit Hilfe von PRINT USING ausgegeben werden, wie in den folgenden Beispielen:

```
PRINT USING "\        \"; "ABC"
ABC
Ready

PRINT USING "\        \"; "Artikel-Nummer"
Artikel-Nu
Ready
```

Bei Strings besteht der Formatstring nicht aus Rautenzeichen, sondern aus Leerzeichen, wobei das erste und das letzte Zeichen durch einen Schrägstrich gekennzeichnet ist. Enthält der auszugebende String mehr Zeichen als der Formatstring, werden die überhängenden Zeichen abgeschnitten, wie im zweiten Beispiel.

Abschließend wollen wir noch ein kleines Programm betrachten, das eine Preisliste mit Artikelnummer, Artikelbezeichnung und Preis ausgeben soll. Die Artikelnummer wird hierin als String behandelt:

```
20 REM PREISLISTE
30 PRINT "Nr.   Artikel      Preis"
40 PRINT
50 READ n$
60 IF n$ = "9999" THEN END
70 READ at$, p
80 PRINT USING "\    \";n$;
90 PRINT "  ";
100 PRINT USING "\          \";at$;
110 PRINT "  ";
120 PRINT USING "####.##";p
130 GOTO 50
140 DATA 1581,Tisch,125
```

```
150 DATA 2659,Stuhl,89.5
160 DATA 0435,Sessel,239.5
170 DATA 5437,Regal,134.95
180 DATA 3436,Lampe,43.7
190 DATA 9999
```

Nach Eingabe von RUN erscheint die Preisliste folgendermaßen auf dem Bildschirm:

```
Nr.        Artikel        Preis

1581       Tisch        125.00
2659       Stuhl         89.50
0435       Sessel       239.50
5437       Regal        134.95
3436       Lampe         43.70
Ready
```

Der Programmaufbau ist uns aus früheren ähnlichen Beispielen schon bekannt. Neu sind hier ledigleich die PRINT USING-Anweisungen zwischen Zeile 80 und 120. Indem sie jeweils mit einem Semikolon abschließen, lassen sich mehrere formatierte Ausgabewerte in eine Reihe setzen. Diese Möglichkeit kennen wir bereits von der normalen PRINT-Anweisung her.

Die einzelnen Daten sind hier wieder in DATA-Zeilen vorgegeben, wobei der letzte Wert 9999 die Abbruchbedingung in Zeile 60 darstellt.

Wollen Sie umfangreiche Listen ausdrucken, empfiehlt es sich, vorher einen Ausdruckplan anzufertigen. Dieser ist ein gitterartiges Schema, dessen Reihen den Zeilen und dessen Spalten den Druckpositionen in der Zeile entsprechen. Die verschiedenen Titel, Spaltenüberschriften und Einzelposten werden in den Ausdruckplan genauso eingetragen, wie sie später auf dem Bildschirm oder auf dem Papier erscheinen. Dabei ist es sinnvoll, die einzelnen Positionen durch ihren entsprechenden Formatstring auf dem Ausdruckplan zu kennzeichnen.

Für Ausdruckpläne verwenden Sie spezielle Formulare, die es im Fachhandel zu kaufen gibt oder aber einfaches, kariertes Papier. Achten Sie auch darauf, daß Sie pro Zeile nicht mehr Positionen angeben, als Ihr Drucker ausgeben kann. Die meisten Drucker können 80 Zeichen, einige Sonderausführungen 132 oder mehr Zeichen pro Zeile ausdrucken, was unter Umständen auch von der eingestellten Schriftbreite abhängt.

5.3 Windows

Der CPC 6128 bietet die Möglichkeit, auf dem Bildschirm bis zu acht Textfenster (Windows) einzurichten. Ist dies einmal geschehen, finden sämtlche Bildschirmoperationen in diesem Textfenster statt, das wie ein eignener Bildschirm zu betrachten ist.

Auf dem Bildschirm können verschiedene Windows mit unterschiedlicher Größe erscheinen. Viele professionelle Programme wenden die Window-Technik an, um im oberen oder unteren Teil des Bildschirms verschiedene Anweisungen zu geben, die der Bediener ständig vor Augen hat, solange er mit dem Programm arbeitet.

Auf ein einmal definiertes Fenster beziehen sich alle Anweisungen, die in irgendeiner Weise Einfluß auf den Bildschirm haben, wie z.B. INPUT, PRINT oder INKEY$. Solche Anweisungen, zu denen auch das Löschen des Bildschirms (CLS) oder die Cursorbewegungen gehören, beeinflussen nur das Window selbst, nicht aber den restlichen Bildschirmbereich. Hinter all diesen Anweisungen muß jeweils ein Rautenzeichen und die Nummer des Windows stehen.

Fenster werden mit der WINDOW-Anweisung erzeugt, die folgendermaßen definiert werden muß:

```
WINDOW # (Nummer), linke Spalte, rechte Spalte,
                   oberste Zeile, unterste Zeile
```

Direkt hinter WINDOW steht ein Rautenzeichen mit der entsprechenden Nummer. Ansonsten dürfen die Spalten- und Zeilenangaben sich nur innerhalb der jeweiligen Bildschirmgrenze bewegen. Der 20-Zeichen-Bildschirm (MODE 0) umfaßt die Spalten 1 bis 20, der 40-Zeichen-Bildschirm (MODE 1) diejenigen von 1 bis 40 und der 80-Zeichen-Bildschirm (MODE 2) die Spalten von 1 bis 80. Die Zeilennummern bewegen sich bei allen drei Modi zwischen 1 und 25.

Als Anwendungsbeispiel wollen wir ein Textfenster erzeugen, das ringsherum einen Rand von vier Zeichen auf dem Bildschirm freiläßt. Im Fenster selbst geben wir in einer Endlosschleife den Satz "****Dies ist das Textfenster****" aus, auf dem Rand erscheint das Wort "RAND":

```
10 MODE 1
20 BORDER 3
30 CLS
40 LOCATE 18,2: PRINT "Rand"
50 WINDOW#1,3,36,5,21
60 PRINT#1,"****Dies ist das Textfenster****"
70 GOTO 60
```

Zeile 10 stellt den 40-Zeichen-Bildschirm ein, worauf Zeile 20 mit der BORDER-Anweisung eine rote Rahmenfarbe setzt. Zeile 40 gibt das Wort "Rand" aus, worauf Zeile 50 das Fenster definiert. Die Zeilen 60 und 70 geben schließlich den Satz innerhalb des Fensters in einer Endlosschleife aus.

5.4 Felder

Allen Variablen, die wir bisher betrachtet haben, konnte immer nur ein Wert gleichzeitig zugeordnet werden. Dies galt gleichermaßen für Fließkomma-, Integer- und Stringvariablen. Jetzt lernen wir Variablen kennen, die sich auf ganze Listen oder Wertebereiche gleichzeitig beziehen. Solche Variablen heißen indizierte Variablen oder auch Felder und können vom Datentyp Fließkoma, Integer oder String sein.

In Feldern lassen sich große Datenmengen ablegen und verarbeiten. Betrachten wir zunächst nochmals unser Beispiel, das die Zahlen 1 bis 10 auf dem Bildschirm ausgibt:

```
10 FOR i = 1 TO 10
20 PRINT i;
30 NEXT
```

Wenn wir dieses Programm ausführen, wird in jedem Schleifendurchgang der jeweilige Inhalt der Laufvariablen i ausgegeben, der sich beim nächsten Durchgang schon wieder ändert. i ist hier also variabel und behält nie einen fest zugeordneten Wert.

Es kann nun erforderlich sein, daß sämtliche Werte, die i bei der Abarbeitung dieser Schleife annimmt, ständig abrufbereit gespeichert sind. Dazu legen wir ein eindimensionales Feld mit einem einzigen Variablennamen an.

Diesem Feld wollen wir den Namen a geben und es soll zehn Kästchen oder Elemente besitzen, in denen wir unsere Werte ablegen. Das erste Kästchen erhält somit die Zahl 1, das zweite die Zahl 2 usw. und das zehnte die Zahl 10.

Feldvariablennamen sind prinzipiell genauso aufgebaut wie diejenigen "normaler Variablen". Sie unterscheiden sich aber durch einen angefügten Index, der in Klammern stehen muß. Hier nun ein Beispiel, in dem wir die Zahlen 1 bis 10 der Feldvariablen a zuordnen:

```
10 DIM a(10)
20 a(1) = 1
30 a(2) = 2
40 a(3) = 3
50 a(4) = 4
60 a(5) = 5
70 a(6) = 6
80 a(7) = 7
90 a(8) = 8
100 a(9) = 9
110 a(10) = 10
```

Feldvariablen werden nicht automatisch angelegt, sondern benötigen eine besondere Anweisung, die ihnen genügend Speicherplatz reserviert. Dies geschieht in unserem Beispiel in Zeile 10 mit der DIM-Anweisung. Mit DIM a(10) wird BASIC aufgefordert, eine Feldvariable a mit Platz für zehn Elemente anzulegen. Diesen Vorgang nennt man Dimensionierung.

Die restlichen Zeilen ordnen den verschiedenen Elementen der Feldvariablen die entsprechenden Werte zu. Nach Ausführung des Programms machen wir im Direktmodus einige Stichproben, um festzustellen, ob die Werte auch richtig zugeordnet wurden:

```
PRINT a(3)
 3
Ready

PRINT a(8)
 8
Ready
```

```
PRINT a(1)
 1
Ready
```

Wir sehen also, daß die Elemente die richtigen ihnen zugeordneten Werte enthalten, die wir jederzeit natürlich auch im Programmodus abrufen können. Normalerweise brauchen wir den Elementen aber nicht durch Einzelanweisungen Werte zuordnen, sondern können dies auch in einer Schleife tun.

Das folgende Programm enthält zwei Schleifen, von denen die erste den Feldelementen Werte zuordnet, die von der zweiten wieder aufgerufen und ausgegeben werden.

```
10 DIM a (10)
20 FOR i = 1 TO 10
30 a(i) = i
40 NEXT
50 :
60 FOR i = 1 TO 10
70 PRINT a(i);
80 NEXT
```

Nach Eingabe von RUN erfolgt

```
 1  2  3  4  5  6  7  8  9  10
Ready
```

Wir können Zeile 30 auch so abändern, daß jedes Element den fünffachen Wert seines Indexes erhält:

```
30 a(i) = 5 * i
```

Wenn wir das Programm jetzt ausführen erhalten wir:

```
 5  10  15  20  25  30  35  40  45  50
Ready
```

Strenggenommen haben wir mit der DIM-Anweiung nicht Platz für 10, sondern für 11 Elemente geschaffen, denn bei jeder Dimensionierung entsteht immer auch ein Feld mit dem Index 0. In unserem Fall hat das Element A(0) aber keine Bedeutung.

Wenn Sie den Versuch unternehmen, mehr Elemente zu belegen, als durch die DIM-Anweisung vorgegeben sind, entsteht die Fehlermeldung

Subscript out of range in (Zeilennummer)

Dimensionieren Sie also die Felder immer ausreichend groß, daß Sie alle Daten, die Sie benötigen, darin unterbringen können. Am besten führen Sie dies gleich am Programmanfang durch. Achten Sie aber darauf, daß jedes Feld nur einmal dimensioniert werden darf. Wird nämlich der Versuch unternommen, dies ein zweites Mal zu tun, erscheint die Fehlermeldung

Array already dimensioned in (Zeilennummer)

und das Programm bricht ab. Da der Speicher begrenzt ist, können Sie ein Feld auch nicht beliebig groß dimensionieren.

Nun wollen wir noch zwei praxisbezogene Beispiele betrachten. Zunächst nehmen wir uns dafür wieder das Programm vor, das eine Preisliste ausgibt. Wir ändern es aber so um, daß sämtliche Daten nicht nur ausgegeben, sondern darüber hinaus auch in Feldern abgelegt werden:

```
10 REM Preisliste
15 DIM n$(20), at$(20), p(20)
20 i = 1
30 PRINT "Nr.   Artikel      Preis"
40 PRINT
50 READ n$(i)
60 IF i>20 OR n$(i)  = "9999" THEN END
70 READ at$(i), p(i)
80 PRINT USING "\    \";n$(i);
90 PRINT "  ";
100 PRINT USING"\          \"; at$(i);
110 PRINT "  ";
120 PRINT USING "####.##";p(i)
125 i = i+1
130 GOTO 50
140 DATA 1581,Tisch,125
150 DATA 2659,Stuhl,89.5
160 DATA 0435,Sessel,239.5
170 DATA 5437,Regal,134.95
180 DATA 3436,Lampe,43.7
190 DATA 9999
```

Statt einfacher Variablen verwenden wir hier die Feldvariablen n$(i) für die Artikelnummer, at$(i) für den Artikel und p(i) für den Preis. Zeile 15 enthält eine DIM-Anweisung, die alle drei Felder gleichzeitig dimensioniert. Für jedes Feld wurden hier absichtlich 20 Elemente vorgesehen, um die Preisliste noch erweitern zu können. Zeile 17 setzt den Zähler i auf 1, der bei jedem Artikel jeweils um eins erhöht wird.

Das nächste Programm wertet Temperaturmessungen aus und bestimmt den höchsten und den niedrigsten Temperaturwert.

```
10 REM Temperaturauswertung
20 DIM t(20)
30 FOR i = 1 TO 20
40 READ t(i)
50 NEXT
60 l = t(1)
70 h = t(1)
80 FOR i = 2 TO 20
90 IF t(i) > h THEN h = t(i)
100 IF t(i) < l THEN l =t(i)
110 NEXT
120 PRINT "Die tiefste Temperatur war"; l
130 PRINT "Die hoechste Temperatur war"; h
140 DATA 12.4, 13.7, 10.3, 18.8, 15.0
150 DATA 14.2, 15.5, 17.4, 16.6, 14.9
170 DATA 11.4, 13.5, 12.8, 11.9, 10.2
180 DATA 13.6, 14.8, 17.1, 13.6, 15.0
```

Nach Eingabe von RUN erscheint:

```
Die tiefste Temperatur war 10.2
Die hoechste Temperatur war 18.8
Ready
```

Zunächst werden die verschiedenen Temperaturwerte eingelesen und im Feld t(i) abgelegt. Die niedrigste (l) und die höchste Temperatur (h) werden dann mit dem ersten Wert t(1) vorbelegt. In der FOR...NEXT-Schleife werden sämtliche Werte vom zweiten Wert an gelesen und geprüft, ob sie größer als der bisher höchste oder kleiner als der bisher tiefste Wert sind. Wenn ja, werden sie zum neuen höchsten bzw. tiefsten Wert.

Felder besitzen die wichtige Eigenschaft des Direktzugriffs. Dies bedeutet, daß man einzelne Elemente nach Belieben beschreiben oder lesen kann,

ohne die davorliegenden Elemente erst einlesen zu müssen. Wir können diese Eigenschaft bei allen hier aufgeführten Beispielen beobachten, die Datenwerte aus DATA-Zeilen einlesen. Ist der Einlesevorgang nämlich erst einmal beendet, kann auf jedes Element frei zugegriffen werden.

Neben den eindimensionalen gibt es auch mehrdimensionale Felder. Sie sind durch mehrere, durch Kommas getrennte Indices gekennzeichnet. Hier ein Beispiel, das ein zweidimensionales Feld dimensioniert:

```
10 DIM p(25,10)
20 FOR j = 1 TO 25
30 FOR i = 1 TO 10
40 p(j,i) = j*i
50 NEXT i
60 NEXT j
```

Zeile 10 dimensioniert das Feld p mit 25 mal 10 Elementen, denen in Zeile 40 Werte aus dem Produkt von i und j zugeordnet werden.

5.5 Suchen und Sortieren

Felder eignen sich besonders für Such- und Sortiervorgänge. Stellen Sie sich einmal vor, Sie suchen einen Namen aus einer unsortierten Adressenliste heraus. Dabei gehen Sie die Liste von oben nach unten durch und vergleichen jeden einzelnen Namen mit dem, den Sie suchen. Dies wiederholen Sie solang, bis Sie entweder Ihren Namen gefunden haben oder die Liste zu Ende ist.

Auch der Computer kann ein Feld nach diesem Schema durchsuchen, was man als sequentielles (aufeinanderfolgendes) Suchen bezeichnet. Hier ein kleines Unterprogramm, das die sequentielle Suche durchführt und das man an jedes andere Programm anfügen kann. Dabei kennzeichnet n$(i) das zu durchsuchende Feld und na$ den Namen, der gesucht werden soll:

```
1000 REM Sequentielles Suchen
1010 i = 1
1020 IF n$(i) = na$ OR n$(i) = "Ende" THEN RETURN
1030 i = i + 1
1040 GOTO 1020
```

Bei diesem Programm wird die Suche so lange fortgeführt, bis entweder der gesuchte Name gefunden oder das Tabellenende erreicht ist. Dies erkennt das Programm an der Markierung "Ende", die im ersten nicht mehr benutzten Feldelement vorhanden sein muß.

Darüber hinaus gibt es noch das binäre Suchen, das sich aber nur für sortierte Felder eignet; es ist mit dem Suchen eines Stichwortes in einem Lexikon vergleichbar. Zunächst schlagen Sie dabei das Lexikon in der Mitte auf, wodurch es in zwei Hälften geteilt wird. Nun wird das erste Wort auf der ersten Seite der zweiten Hälfte betrachtet. Erscheint Ihr Stichwort davor, so ist es in der ersten Hälfte zu suchen, anderenfalls befindet es sich in der zweiten Hälfte.

Diesen Vorgang wiederholen Sie nun für die Hälfte des Lexikons, in der das gesuchte Wort steht. Sie wird wieder in der Mitte aufgeschlagen, und es wird erneut bestimmt, welche der beiden Hälften das Wort enthält. Diesen Vorgang wiederholen Sie so häufig, bis nur noch eine Seite übrigbleibt.

Binäres Suchen in einem Lexikon mag eine etwas umständliche Methode darstellen, in Computerprogrammen ist es aber dem sequentiellen Suchen weit überlegen, besonders bei großen Dateien. So benötigt ein Verzeichnis mit 1000 Einträgen bei sequentieller Suche durchschnittlich 500, maximal jedoch 1000 Vergleiche, um einen Eintrag zu finden, während es bei der binären Suche nie mehr als 10 sind.

Auch für das binäre Suchen wollen wir ein kleines Unterprogramm betrachten:

```
1000 REM Binaeres Suchen
1010 l = 1 : mi = 1 : h = fr -1
1020 WHILE l<=h AND n$(mi)<>na$
1030 mi = INT((l+h)/2)
1040 IF na$ < n$(mi) THEN h = mi-1
1050 IF na$ > n$(mi) THEN l = mi+1
1060 WEND
1070 i=mi
1080 RETURN
```

Hierin ist wieder n$(i) das Feld mit den Namen und na$ der gesuchte Name. fr ist das erste freie Element, das nicht belegt ist. l und h grenzen den Bereich der Elemente ein, in denen sich der gesuchte Name befindet. mi zeigt jeweils auf die Mitte des zu halbierenden Bereichs. i gibt die

Nummer des Elements, welches den gesuchten Namen enthält, an das aufrufende Hauptprogramm zurück.

Nun wollen wir uns noch zwei Sortierverfahren für Strings zuwenden, die aber in leicht abgewandelter Form auch für numerische Werte einzusetzen sind. Bei Strings verwenden wir ein Stringfeld und bei numerischen Werten ein Feld für Fließkommazahlen.

Zunächst befassen wir uns mit dem Sortieren durch Einfügen. Während des ganzen Sortiervorganges wird das Feld in zwei Teile unterteilt, nämlich in einen bereits sortierten und in einen unsortierten Teil. Aus dem unsortierten Teil wird immer das erste Element herausgenommen, mit den Elementen im sortierten Teil verglichen und dort an der richtigen Stelle eingefügt. Hier zunächst einmal das Programmlisting:

```
10 REM Sortieren durch Einfuegen
20 :
30 :
40 :
50 REM Daten einlesen
60 DIM n$(20)
70 i=1
80 :
90 READ n$(i)
100 IF n$(i)="Ende" THEN k=i-1:GOTO 160
110 i=i+1
120 GOTO 90
130 :
140 :
150 :
160 REM Daten unsortiert ausgeben
170 PRINT "Namen unsortiert"
180 PRINT
190 FOR i=1 TO k
200 PRINT n$(i)
210 NEXT
220 PRINT
230 PRINT
240 :
250 :
260 :
270 REM Daten sortieren
280 FOR i=2 TO k
```

```
290 n$(0)=n$(i)
300 j=i-1
310 WHILE n$(j)>n$(j+1)
320 a$=n$(j+1)
330 n$(j+1) = n$(j)
340 n$(j)=a$
350 j=j-1
360 WEND
370 NEXT
380 :
390 :
400 :
410 REM Daten sortiert ausgeben
420 PRINT "Namen sortiert"
430 PRINT
440 FOR i=1 TO k
450 PRINT n$(i)
460 NEXT
470 END
480 :
490 :
500 :
510 REM Datenwerte
520 DATA Hans,Inge,Fritz,Rainer,Sigmund,Anton,Susanne
530 DATA Elisabeth,Christa,Franz,Maria,Monika,Ende
```

Zunächst werden sämtliche Namen, die hier sortiert werden sollen, aus den DATA-Zeilen am Programmende eingelesen und den Elementen des Feldes n$(i) zugeordnet. Zeile 60 dimensioniert das Feld für 20 Einträge. Wir hätten es aber ebensogut mit einem anderen Wert dimensionieren können, um Platz für mehr oder weniger Einträge zu schaffen.

Am Ende der DATA-Zeilen steht wieder die Markierung "Ende", die das Ende der Tabelle kennzeichnet. Wir können nun auch leicht die Anzahl der Werte bestimmen, die die Tabelle enthält. Im Programm ist diese der Variablen k zugeordnet. Ab Zeile 160 werden dann sämtliche Namen in unsortierter Reihenfolge ausgegeben.

Das eigentliche Sortieren des Feldes n$(i) beginnt in Zeile 270. k zeigt auf das letzte Element des Feldes, was besagt, daß alle Elemente von n$(1) bis n$(k) zu sortieren sind. Das ansonsten nicht verwendete Element n$(0) dient hier als Markierung, damit der Tabellenanfang nicht unterlaufen wird.

Bei jedem Sortierschritt wird das Feld n$ in einen bereits sortierten und einen noch unsortierten Teil aufgeteilt. Dabei zeigt die Variable i jeweils auf den ersten Wert des nicht-sortierten Teilfeldes, d.h. die Elemente n$(1) bis n$(I-1) sind jeweils sortiert, während die Elemente n$(I) bis n$(K) noch nicht sortiert sind.

Der Sortiervorgang beginnt mit dem zweiten Element (FOR...NEXT-Schleife), da eine Tabelle mit einem Element immer sortiert ist. In den Zeilen 310 bis 360 (WHILE...WEND-Schleife) wird jeweils das einzufügende Element solange mit den Elementen des sortierten Teils verglichen, bis es an seiner richtigen Stelle steht. Ist es noch größer, als das nächstfolgende Element, wird es mit ihm vertauscht (Zeile 320 bis 340). Damit der Tauschvorgang zwischen den Werten von n$(j) und n$(j+1) stattfinden kann, wird die Hilfsvariable a$ eingeführt. Dieser Vorgang wird nun für jedes Element der Tabelle wiederholt. Dabei nimmt bei jedem Durchgang (FOR...NEXT-Schleife) die Anzahl der sortierten Elemente um eins zu, während die nicht-sortierten Elemente um eins abnehmen. Ist die Tabelle fertig sortiert, werden die Elemente nochmals in ihrer richtigen Reihenfolge ausgegeben (Zeile 410 bis 470).

Das Sortierverfahren durch Einfügen ist nur für kurze Listen geeignet, da bei längeren Listen die Sortierzeit ins Unermeßliche steigt. Das nachfolgend aufgeführte Verfahren nach Shell arbeitet wesentlich schneller, da es nicht zwei nebeneinanderliegende Elemente vergleicht und gegebenenfalls vertauscht, sondern zwei Elemente, die weit auseinanderliegen, wobei viele dazwischenliegende Elemente übersprungen werden.

Das Shell-Verfahren ist dem Verfahren durch Einfügen zwar ähnlich, unterscheidet sich aber dadurch, daß anstelle der Elemente n$(j) und n$(j+1) die Elemente n$(j) und n$(j+g) verglichen und (falls nötig) vertauscht werden. Dabei kann g viel größer als 1 sein. Das Shell-Verfahren beinhaltet also das Einfügverfahren in abgeänderter Form, wobei die Sortiervorgänge für verschiedene Werte von g durchgeführt werden.

Hier nun das gleiche Programm noch einmal mit der Sortierroutine nach Shell:

```
10 REM Sortieren nach Shell
20 :
30 :
40 :
50 REM Daten einlesen
60 DIM n$(20)
```

```
70 i=1
80 :
90 READ n$(i)
100 IF n$(i)="Ende" THEN k=i-1:GOTO 160
110 i=i+1
120 GOTO 90
130 :
140 :
150 :
160 REM Daten unsortiert ausgeben
170 PRINT "Namen unsortiert"
180 PRINT
190 FOR i=1 TO k
200 PRINT n$(i)
210 NEXT
220 PRINT
230 PRINT
240 :
250 :
260 :
270 REM Daten sortieren
280 g=INT(k/2)
290 WHILE g>0
300 FOR i=g+1 TO k
310 j=i-g
320 WHILE j>0
330 IF n$(j)<=n$(j+g) THEN j=0:GOTO 390
340 a$=n$(j+g)
350 n$(j+g)=n$(j)
360 n$(j)=a$
370 j=j-g
380 :
390 WEND
400 NEXT
410 g=INT(g/2)
420 WEND
430 :
440 :
450 :
460 REM Daten sortiert ausgeben
470 PRINT "Namen sortiert"
480 PRINT
490 FOR i=1 TO k
```

```
500 PRINT n$(i)
510 NEXT
520 END
530 :
540 :
550 :
560 REM Datenwerte
570 DATA Hans,Inge,Fritz,Rainer,Sigmund,Anton,Susanne
580 DATA Elisabeth,Christa,Franz,Maria,Monika,Ende
```

Zu Beginn des Programms wird g auf die halbe Größe der zu sortierenden Liste gesetzt. Danach wird g wiederholt durch 2 geteilt und mit jedem dadurch erhaltenen Wert das abgeänderte Verfahren durch Einfügen durchgeführt. Dies geschieht solange, bis g gleich eins und damit der Sortiervorgang beendet ist.

Versuche haben ergeben, daß das Shell-Verfahren große Datenmengen etwa zehnmal schneller als das Einfügeverfahren sortiert. Bei kleinen Listen mag das Einfügeverfahren geringfügig schneller arbeiten, da es weniger Anweisungen benötigt. Insgesamt betrachtet ist jedoch das Shell-Verfahren gegenüber dem Verfahren durch Einfügen erheblich vorteilhafter!

6 Die Floppy

In diesem Kapitel wollen wir uns intensiver mit der Floppy beschäftigen. Obwohl sie fest in Ihren CPC 6128 eingebaut ist, ist die Floppy eigentlich ein Peripheriegerät.

Unter Peripherie versteht man in der Computertechnik alle Geräte, die an die Zentraleinheit angeschlossen werden. Dazu gehört auch der Drucker, das Modem oder ein selbstgebasteltes Interface mit Relais, das morgens um sechs Ihre Kaffemaschine einschaltet, damit Sie gleich nach dem Aufstehen heißen Kaffee vorfinden.

Als Pripheriegerät benötigt die Floppy Ihres CPC 6128 eine eigenene Stromversorgung. Dazu dient das Kabel, das hinten aus Ihrem Computer heraushängt. Vergessen Sie deshalb nicht, dieses Kabel in die Buchse 12V DC am Monitor zu stecken, da sonst die Floppy nicht funktioniert.

Die anderen beiden Verbindungskabel hängen am Monitor und müssen am Computer eingesteckt werden. Sie dienen zur Stromversorgung des Computers und zur Übertragung der Video- und Tonsignale an den Monitor.

Sowohl der Computer als auch die Floppy als Peripheriegerät haben eigene Betriebssysteme, die beide Hand in Hand miteinander arbeiten. Diejenigen Leser, die schon mit dem Vorgängermodell CPC 464 gearbeitet haben, wissen, daß man hier die Floppy einschließlich Betriebssystem separat anschließen mußte.

Nachfolgend befassen wir uns mit der Floppy und ihrem Betriebssystem, das den Namen AMSDOS (AMSTRAD Disc-Operating-System) trägt.

6.1 Laufwerk und Disketten

Das 3-Zoll-Floppylaufwerk des CPC 6128 gehört zu den kleinsten und kompaktesten Laufwerken, die es heute gibt. Dabei ist es genauso leistungsfähig wie seine größeren Brüder mit 8 und 5.25 Zoll.

Wenn Sie eine 3-Zoll-Diskette neben eine 5-Zoll-Diskette legen, fällt Ihnen nicht nur der Größenunterschied auf, sondern auch die Schutzhülle. Die eigentliche Diskette besteht nämlich aus einer empfindlichen Magnetscheibe, die gegen mechanische Verletzungen unbedingt geschützt werden muß.

Die Hülle der 5.25-Zoll-Disketten ist sehr biegsam und besitzt darüber hinaus zwei Öffnungen für den Schreib-/Lesekopf und das Indexloch. Es verwundert daher nicht, daß der Umgang mit solchen Disketten besonderer Sorgfalt bedarf. Auch sollten die Disketten nur zum Gebrauch aus ihrer Papierhülle, die die Öffnungen schützt, herausgenommen werden.

Die für die Schneider-Floppies verwendeten 3-Zoll-Disketten befinden sich in einem stabilen Plastikgehäuse, das bei weitem nicht so empfindlich ist wie die Hülle der 5.25-Zoll-Disketten. Zusätzlich werden die Öffnungen für den Schreib-/Lesekopf und das Indexloch mit einer Metallklappe verschlossen, die sich beim Einschieben der Diskette in das Laufwerk automatisch öffnet. An der Seite des Gehäuses befindet sich nämlich eine Führungsschiene mit einem weißen Schieber. Wenn Sie diesen Schieber (vorsichtig!) betätigen, spannen Sie eine Feder und öffnen gleichzeitig die Klappe, wobei die Magnetscheibe sichtbar wird. Beim Loslassen des Schiebers schnappt die Klappe aufgrund der Federwirkung wieder zu.

Die 3-Zoll-Disketten sind beidseitig beschreibbar; Sie können aber immer nur eine Seite, nämlich Seite A oder Seite B gleichzeitig nutzen. Das Laufwerk greift jeweils auf die Seite zu, die beim Einschieben oben auf der Diskette sichtbar ist.

Das Einlegen der Diskette in das Laufwerk ist denkbar einfach. Sie schieben die Diskette so weit hinein, bis sie einrastet. Dies sollte aber nur dann geschehen, wenn sowohl der Computer, als auch die Floppy (Verbindungskabel s.o.) eingeschaltet sind, da es ansonsten unter ungünstigen Umständen zu Datenverlusten kommen kann.

Das Herausnehmen der Diskette ist genauso einfach. Sie drücken lediglich den Knopf rechts unter der Öffnung und die Diskette springt heraus.

Sie können Ihre Disketten vor versehentlichem Schreiben schützen, wenn Sie links oben auf jeder Seite das kleine, mit einer roten Klappe verschlossene Loch öffnen. Dies erreichen Sie, indem Sie mit einem spitzen Stift den seitlich angebrachten roten Schieber betätigen.

Ungeachtet des Schreibschutzes sollten Sie von allen wichtigen Programmen und Dateien mindestens eine Sicherheitskopie anfertigen, die sich nach Möglichkeit auf einer anderen Diskette befinden sollte. Trotz der hohen Aufzeichnungssicherheit ist es dennoch nicht völlig auszuschließen, daß Schreib- oder Lesefehler auftreten. Doch wahrscheinlicher als rein technische Fehler ist das "menschliche Versagen", wodurch unbeabsichtigt eine Diskette neu formatiert oder wichtige Dateien gelöscht oder überschrieben werden. Auch in einem solchen Fall kann man dann auf die Sicherheitskopie zurückgreifen.

Ihren CPC 6128 können Sie übrigens auch mit zwei Laufwerken betreiben. Es sind nämlich Laufwerke in einem extra Gehäuse erhältlich, die mit einem besonderen Anschlußkabel mit dem CPC 6128 verbunden werden. Dieses Zweitlaufwerk wird dann als Laufwerk B bezeichnet, während das eingebaute Laufwerk immer Laufwerk A ist.

Einige Fremdfirmen bieten 5.25-Zoll-Laufwerke für Schneider-Computer an, die als Zweitlaufwerke betrieben werden können. Sie werden ebenfalls durch das AMSDOS-Betriebssystem gesteuert, so daß sich sämtliche Anweisungen für 3-Zoll-Laufwerke auch auf die 5.25-Zoll-Laufwerke beziehen.

Es stellt sich allerdings die Frage, welchen Nutzen diese Laufwerke bringen. Der Einsatz von 5.25-Zoll-Disketten alleine bringt keinen Vorteil, wie wir bereits gesehen haben. Solche Laufwerke sind nur dann sinnvoll, wenn Sie Daten mit einem anderen Computer austauschen möchten, dessen Floppy ein IBM-kompatibles Diskettenaufzeichnungsformat verwendet. Neben dem IBM PC gibt es noch eine Reihe anderer Computer, die mit diesem Format arbeiten oder es zumindest lesen können. Die Floppy des CPC 6128 arbeitet normalerweise zwar nicht im IBM-Format, sie kann aber Disketten in diesem Format lesen und beschreiben.

6.2 Arbeiten mit AMSDOS

In diesem Abschnitt wollen wir die Anweisungen und Dienstprogramme kennenlernen, die Sie zum Arbeiten mit AMSDOS benötigen. Doch zuvor noch ein wichtiger Hinweis: Das hier vorgestellte Dienstprogramm DISCKIT3 läuft nur unter CP/M. Da die Diskettenorganisation sowohl unter AMSDOS als auch unter CP/M identisch ist, können wir mit diesem Dienstprogramm auch Disketten bearbeiten, die unter AMSDOS beschrieben wurden.

6.2.1 Das Dienstprogramm DISCKIT3

Dieses Dienstprogramm benötigen wir zum Formatieren, Kopieren (Backup) und Verifizieren von Disketten. Obwohl es nur unter CP/M arbeitet, ist es das wichtigste Dienstprogramm, das wir brauchen, wenn wir in BASIC bzw. AMSDOS arbeiten.

Bevor Sie dieses Programm laden, achten Sie bitte darauf, daß sich kein wichtiges BASIC- oder Maschinenprogramm im Speicher befindet. Wir müssen nämlich zunächst CP/M laden, um mit DISCKIT3 arbeiten zu können. Legen Sie dazu Seite 1 der beiden Systemdisketten ins Laufwerk A (eingebautes Laufwerk) und geben Sie dann

```
|CPM <RETURN>
```

ein. Nach einigen Sekunden erscheint die Einschaltmeldung und das Bereitschaftszeichen (Prompt)

```
CP/M Plus Amstrad Consumer Electronics plc

A>
```

auf dem Bildschirm. Geben Sie jetzt

```
DISCKIT3 <RETURN>
```

ein, worauf DISCKIT3 von der Diskette geladen und gestartet wird.

DISCKIT3 fragt zunächst einmal ab, wieviele Laufwerke angeschlossen sind. Haben Sie nur das eingebaute Laufwerk A in Betrieb, erscheint

One drive found

Ist dagegen ein zusätzliches Laufwerk B angeschlossen, lautet die Meldung

Two drives found

Im unteren Bildschirm erscheint nun folgendes Auswahlmenü:

Copy	7	(Diskette kopieren)
Format	4	(Diskette formatieren)
Verify	1	(Diskette verifizieren)
Exit from program	0	(DISCKIT3 verlassen)

Zur Auswahl des gewünschten Menüpunktes müssen Sie die entsprechende Funktionstaste im Zifferntastenblock drücken.

Bevor wir überhaupt etwas auf eine Diskette schreiben können, müssen wir sie zunächst einmal formatieren. Hierbei werden auf die Diskette Spuren und Sektoren aufgetragen, in denen später Daten oder Programme abgelegt werden können.

Die Disketten erhalten dabei 40 Spuren (Spur 0 bis 39), von denen jede in 9 Sektoren zu je 512 Bytes unterteilt ist. Die gesamte für den Anwender verfügbare Diskettenkapazität beträgt demnach

40 x 9 x 512 = 184320 Bytes bzw. 180 K zu je 1024 Bytes

Nachdem Sie DISCKIT3 geladen haben, drücken Sie die Taste f4 zum formatieren. Nehmen Sie jetzt die Systemdiskette aus dem Laufwerk und legen Sie stattdessen die zu formatierende Diskette ein. Achten Sie unbedingt darauf, daß Sie eine fabrikneue Diskette einlegen oder eine solche, auf dessen Inhalt Sie unbedingt verzichten können. Durch die Formatierung gehen etwa vorhandene Daten auf der Diskette unwiderruflich verloren!

Sie sehen nun ein weiteres Menü auf dem Bildschirm, in dem Sie zur Art der Formatierung gefragt werden:

System format	9	(Systemformat)
Data format	6	(Datenformat)
Vendor format	3	(Vendorformat)
Exit menu	.	(Menü verlassen)

Falls Sie auf der zu formatierenden Diskette nur unter AMSDOS arbeiten möchten, wählen Sie jetzt das Datenformat (Taste f6) aus. Soll Ihre Diskette CP/M-Dateien erhalten oder sowohl CP/M- als auch AMSDOS-Dateien, müssen Sie Taste f9 (Systemformat) drücken. Dabei werden die beiden äußersten Spuren der Diskette (0 und 1) mit dem CP/M-Betriebssystem belegt und stehen deshalb für den Anwender nicht zur Verfügung. Entsprechend verringert sich auch die verfügbare Diskettenkapazität geringfügig auf 171 k.

Das Vendorformat ist mit dem Systemformat identisch, mit der Ausnahme, daß die Systemspuren für CP/M zwar reserviert, aber nicht beschrieben werden. Dieses Format ist nur für den Verkauf von CP/M-Software interessant, wobei zwar die Software, nicht aber das CP/M-Betriebssystem mit verkauft wird.

Nachdem Sie die gewünschte Formatierungsart ausgewählt haben, erscheint folgende Meldung auf dem Bildschirm:

```
Y       Format as Data

        Any other key to exit menu
```

Drücken Sie jetzt Y, falls Sie eine Datendiskette formatieren möchten. Dies ist eine letzte Sicherheitsabfrage vor dem Formatieren. Falls Sie ein anderes Format gewählt haben, erscheint eine entsprechend ähnliche Meldung. Wenn Sie statt Y eine andere Taste drücken, findet keine Formatierung statt und der Computer kehrt ins Menü zurück.

Falls zwei Laufwerke angeschlossen sind, erfolgt vor dem Formatieren noch zusätzlich die Abfrage, ob die zu formatierende Diskette in Laufwerk A oder B liegt.

Kommen wir jetzt zum Kopieren einer Diskette (Taste f7). Dabei muß die Diskette, auf die kopiert werden soll (Zieldiskette), schon formatiert sein.

Sie brauchen sich beim Kopieren nicht darum kümmern, ob die zu kopierende Diskette (Quelldiskette) im System-, Daten- oder Vendorformat formatiert ist, da dies der Computer automatisch erkennt. Falls Sie nur mit einem Laufwerk kopieren, erfolgt zunächst die Sicherheitsabfrage

```
Y     Copy

          Any other Key to exit menu
```

Wenn Sie diese mit Y (Ja) beantworten, kann der Kopiervorgang beginnen. Sie sollten zuvor jedoch Ihre Systemdiskette aus dem Laufwerk nehmen und statt dessen die Diskette einlegen, die kopiert werden soll. Nun erscheinen abwechselnd die Meldungen

```
Insert disc to WRITE          (Zieldiskette einlegen)
Press any key to continue     (Weiter - beliebige Taste drücken)
```

und

```
Insert disc to READ           (Quelldiskette einlegen)
Press any Key to continue     (Weiter - beliebige Taste drücken)
```

Wenn Sie die Anweisungen befolgen und den Diskettenwechsel immer richtig vornehmen, erscheint nach einigen Durchgängen die Meldung

```
Copy completed                (Kopiervorgang beendet)
Remove disc                   (Diskette herausnehmen)
Press any key to continue     (Weiter - beliebige Taste drücken)
```

Damit ist der Kopiervorgang beendet.

Falls sie mit zwei Laufwerken kopieren, müssen Sie Quell- und Zieldiskette nicht ständig wechseln. Zunächst erscheint das Zusatzmenü:

```
Write to A:        9      (auf Laufwerk A kopieren)

Write to B:        6      (auf Laufwerk B kopieren)

Exit menu          3      (Menü verlassen)
```

Sie legen jetzt Quell- und Zieldiskette in das richtige Laufwerk ein und drücken die entsprechende Taste. Es ist allerdings bequemer und schneller, von Laufwerk A auf Laufwerk B zu kopieren.

Der letzte Menüpunkt von DISCKIT3 ist Verify (Taste f1). Hierbei wird der Zustand der zu überprüfenden Diskette untersucht und alle Fehler angezeigt.

Verify arbeitet ebenfalls wahlweise mit einem oder zwei Laufwerken. Beachten Sie alle Anweisungen, die auf dem Bildschirm erscheinen, um den Test durchzuführen.

Um von CP/M wieder zurück zu BASIC und AMSDOS zu gelangen, gibt es drei Möglichkeiten, bei denen es allerdings keine Rettung für den Speicherinhalt gibt:

1. Computer aus- und wieder einschalten
2. CTRL-, SHIFT- und ESC-Taste gleichzeitig drücken (Reset)
3. CP/M-Programm AMSDOS aufrufen und ausführen

6.2.2 Dateitypen

Sämtliche Dateien, die sich auf einer Diskette befinden, sind in dem Disketten-Inhaltsverzeichnis (Directory) eingetragen. Legen Sie einmal eine Seite der Systemdiskette oder eine andere Diskette mit Dateien ins Laufwerk A ein und geben Sie dann

```
|DIR
```

ein. Auf dem Bildschirm erscheinen sämtliche Dateinamen, die im Directory eingetragen sind. Es gibt jedoch noch eine andere Anweisung, die etwas detailliertere Aufschlüsse über die abgelegten Dateien gibt:

```
CAT
```

Die CAT-Anweisung listet ebenfalls sämtliche Dateinamen auf dem Bildschirm, jedoch zusätzlich alphabetisch sortiert. Darüber hinaus steht hinter jedem Namen, wieviel Speicherplatz in kByte (1024 Bytes) die betreffende Datei auf der Diskette einnimmt.

Selbst wenn eine Datei nur 3 Bytes umfaßt, wird trotzdem 1K auf der Diskette reserviert. Dieser Umstand hängt mit der internen Diskettenverwaltung unter AMSDOS und CP/M zusammen.

Das Directory kann maximal 64 Einträge aufnehmen. Dabei ist allerdings zu beachten, daß Dateien, die mehr als 16 K umfassen, für jede angefangenen 16 K einen weiteren Directoryeintrag (Extent) benötigen. Beim Auflisten des Directories erscheint der Name aber nur einmal.

Dateinamen dürfen maximal acht Zeichen umfassen, an die sich zusätzlich noch ein Punkt und eine Extension von drei Zeichen anschließen können. Hier einige Beispiele für generell zulässige Dateinamen:

TEST.BAS
DATEI.TXT
WS.COM
TITEL.
ERWEIT.BIN
LAGER.BAK
BIBLTHEK.DAT

In der Praxis haben jedoch einige Extensionen eine bestimmte Bedeutung. Wenn Sie z.B. ein BASIC-Programm von Diskette laden, prüft das Betriebssystem des Computers, ob die Extension .BAS vorhanden ist. Hier nun die für bestimmte Zwecke reservierten Extensionen:

.BAS	BASIC-Programm
.BIN	Binärformat, meist Maschinenprogramm
.BAK	Backup-Datei
.COM	Command-Datei (CP/M)
.SUB	Stapelverarbeitungsdatei (CP/M)
.	undefiniert, meist reine Datendateien

Ist eine Datei bereits unter einem bestimmten Namen auf Diskette abgelegt und wird eine neue Datei unter gleichem Namen abgespeichert, wird die alte Datei nicht überschrieben, sondern stattdessen als Backup-Datei (Extension .BAK) beibehalten. Die ursprüngliche Extension spielt dabei keine Rolle. Backup-Dateien dienen in erster Linie als Sicherheitskopien und stellen einen bedingten Schutz gegen versehentliches Löschen dar.

Erst wenn ein drittes Mal eine Datei unter gleichem Namen abgelegt wird, wird die erste Datei gelöscht, während die zweite zur Backup-Datei wird. Hier ein Beispiel: Angenommen, Sie entwickeln ein BASIC-Programm unter dem Namen "TEST", das Sie anschließend durchtesten, da es noch nicht fehlerfrei läuft. Dieses Programm speichern Sie jetzt mit

```
SAVE "TEST"
```

auf Diskette ab. Wenn Sie sich jetzt das Directory anschauen, ist das Programm dort unter dem Dateinamen "TEST.BAS" aufgeführt, da jedes BASIC-Programm automatisch die Extension .BAS erhält. Jetzt verbessern Sie Ihr BASIC-Programm und speichern es unter dem gleichen Namen erneut ab. Wenn Sie danach einen Blick ins Directory werfen, finden Sie folgende Einträge vor:

```
TEST.BAS
TEST.BAK
```

Das zuletzt abgespeicherte Programm erscheint jetzt unter dem Namen TEST.BAS, während das ursprüngliche Programm den Namen TEST.BAK erhält.

An einer Backup-Datei allein ist nicht immer zu erkennen, welche Extension ursprünglich vorhanden war. So könnte TEST.BAK durchaus auch eine Backup-Kopie von TEST.BIN, TEST.COM oder TEST.TXT sein. In unserem Beispiel wissen wir aber, daß es sich hierbei um eine .BAS-Datei handelt.

Sollte es einmal nötig sein, eine Backup-Datei wieder zu laden, so ist es sinnvoll, ihr die ursprüngliche Extension wiederzugeben. Dazu muß der Dateiname aber erst umbenannt werden, was mit der REN-Anweisung (s.u.) geschieht.

6.2.3 Programme laden und sichern

Zunächst betrachten wir einmal normale BASIC-Programme. Wir wissen bereits aus Kapitel 1, daß wir ein solches Programm mit SAVE abspeichern und mit LOAD wieder laden. Nun gibt es aber insgesamt drei Möglichkeiten, ein BASIC-Programm auf Diskette abzulegen, wie das folgende Beispiel zeigt:

```
SAVE "TEST"
SAVE "TEST",P
SAVE "TEST",A
```

In allen drei Fällen wird das Programm unter dem Dateinamen TEST.BAS im Directory eingetragen. Dennoch handelt es sich hierbei um verschiedene Aufzeichnungsverfahren.

Im ersten Fall wird das Programm so, wie es im Speicher steht, auf Diskette geschrieben, also als interner BASIC-Code. Das gleiche geschieht auch im zweiten Fall, wobei allerdings der Zusatz P angibt, daß das Programm beim Laden listgeschützt ist. Mit dem Zusatz A wird das Programm nicht im internen BASIC-Format, sondern als reine ASCII-Datei auf Diskette geschrieben.

Als ASCII-Datei kann man ein Programmlisting leicht in eine Textdatei zur Textverarbeitung übernehmen. So wurden die meisten Programmbeispiele, die in diesem Buch aufgeführt sind, zunächst als ASCII-Datei auf Diskette geschrieben und später mit dem übrigen Text in einem Textverarbeitungsprogramm vermischt.

Wenn Sie das Programm nun wieder laden, geben Sie

```
LOAD "TEST"
```

ein. Dieser Befehl lädt ohne weiteres das normal abgespeicherte BASIC-Programm, wie auch das als ASCII-Datei abgelegte Programm. Die Dateien enthalten nämlich, mit Ausnahme der ASCII-Dateien, einen Vorspann (Header), an dem das Betriebssystem erkennt, um welchen Dateityp es sich handelt.

Versuchen wir dagegen ein geschütztes Programm mit LOAD zu laden, führt der Computer nach dem Laden automatisch die Anweisung NEW aus, wodurch das Programm im Speicher zerstört wird. Dadurch ist weder ein Auflisten noch ein Ausführen mit RUN möglich.

Geschützte Programme lassen sich nur mit RUN "Dateiname" laden und gleichzeitig starten (Autostart). Somit lädt die Anweisung

```
RUN "TEST"
```

das BASIC-Programm "TEST.BAS" und startet es dann. Dabei spielt es keine Rolle, ob es sich um ein normal abgespeichertes, ein geschütztes oder ein als ASCII-Datei abgelegtes Programm handelt.

Von BASIC aus können wir auch Maschinenprogramme oder sonstige Speicherinhalte als Binärdatei auf Diskette ablegen und wieder laden. Einzige Voraussetzung ist, daß sich der betreffende Speicherbereich im RAM befindet, also keine besondere Speicherbankumschaltung (Bank switching) vorgenommen werden muß. Ein typisches Beispiel hierfür ist die Abspeicherung des Bildschirm-RAMs. Angenommen, Sie haben sich eine Bildschirm-Graphik erstellt und wollen diese jetzt komplett auf Diskette ablegen und wieder laden. Dabei handelt es sich um insgesamt 16K RAM-Speicher (16384 Bytes) von der Adresse 49152 an aufwärts. Die Datei soll den Namen BILD erhalten.

```
SAVE "BILD",B,49142,16384
```

bzw.

```
SAVE "BILD",B,&C000,&4000
```

schreibt Ihnen das gesamte Video-RAM auf Diskette. Da hinter dem Dateinamen der Zusatz B steht, wird automatisch eine Binärdatei mit der Extension .BIN erzeugt. Im Directory steht somit der Name BILD.BIN.

Hinter B folgt die Anfangsadresse und die Anzahl der Bytes, die von dieser Adresse an aufwärts abgespeichert werden sollen. Die zweite SAVE-Anweisung hat die gleiche Wirkung, nur wurden hier die Adressen in Hexadezimalzahlen angegeben, denen in BASIC ein "&" voranstehen muß. In beiden Fällen werden ab der Adresse 49152 (&C000) 16384 (&4000) Bytes auf Diskette geschrieben.

Zum Laden der Binärdatei geben Sie lediglich

```
LOAD "BILD"
```

ein und Ihre Graphik erscheint sofort wieder auf dem Bildschirm ab der Adresse 49152 (&C000). Falls Sie Ihr Kunstwerk jedoch in einem anderen Speicherbereich laden möchten, geben Sie noch zusätzlich die Ladeadresse an:

```
LOAD "BILD",&8000
```

lädt die Datei nicht ab Adresse &C000, sondern ab Adresse &8000.

Abschließend wollen wir uns noch kurz damit befassen, wie BASIC-Programme aneineindergehängt oder vermischt werden können.

```
CHAIN MERGE "Dateiname"
```

lädt das unter "Dateiname" angegebene Programm von Diskette und mischt es mit dem im Speicher befindlichen Programm entsprechend der jeweiligen Zeilennummern. Besitzt das neu hinzugeladene Programm höhere Zeilennummern als das im Speicher befindliche, wird es hinten angehängt.

Infolge eines Fehlers im Betriebssystem kann es bei einigen Geräten vorkommen, daß diese Anweisung nicht einwandfrei funktioniert. Die nachgeladene Datei ist nämlich dann zu Ende, wenn das Zeichen 1AH (Textende) eingelesen wird. Dieses Zeichen gilt allerdings nur für ASCII-Dateien, kann aber durchaus auch im internen BASIC-Code vorkommen, z.B. wenn eine Zeilennummer 26 (1AH) vorhanden ist.

Ob Ihr Rechner von dem Fehler betroffen ist, können Sie am besten feststellen, indem Sie ein BASIC-Programm nachladen, das die Zeilennummer 26 enthält und darüber hinaus auch Zeilen mit höheren Nummern. Bricht die Übertragung vor Zeile 26 ab, liegt ein Fehler im Betriebssystem vor. Sie können ihn nur so umgehen, indem Sie das nachzuladende Programm als ASCII-Datei auf Diskette schreiben.

6.2.4 Dateien löschen

Eine Datei können Sie von BASIC bzw. AMSDOS aus nur in zwei Schritten löschen. Hier ein Beispiel:

```
a$ = "TEST.BAS"
|ERA, @a$
```

Zunächst müssen Sie den Namen der zu löschenden Datei samt Extension in einem String ablegen. Dann erst können Sie die eigentliche Löschanweisung |ERA ausrufen.

Manchmal ist es sinnvoll, im Dateinamen sogenannte Wildcards "*" zu verwenden. Angenommen, Sie möchten alle Backup-Dateien auf einer Diskette löschen, da sie zu viel Platz wegnehmen. In diesem Fall brauchen Sie nicht jede Datei einzeln zu löschen, sondern geben lediglich

```
a$ = "*.BAK"
```

als Dateiname an.

Mit dem folgenden Dateinamen löschen Sie sämtliche Dateien, die sich auf der Diskette befinden:

```
a$ = "*.*"
```

Aus Sicherheitsgründen fragt AMSDOS noch einmal nach, ob Sie wirklich sämtliche Dateien löschen möchten, denn ein Fehler kann hier schwerwiegende Folgen haben.

6.2.5 Dateien umbenennen

Mit |REN können Sie Diskettendateien einen neuen Namen geben. Nehmen wir einmal an, Sie wollen die Datei TEST.BAS in PROBE.BAS umbenennen. Dazu sind folgende Anweisungen nötig:

```
a$ = "TEST.BAS"        (alter Name)
b$ = "PROBE.BAS"       (neuer Name)
|REN, @b$, @a$
```

Die Extension muß hier bei jedem Dateinamen unbedingt mit angegeben sein. In der Anweisung steht zuerst der neue und dann der alte Dateiname als Stringvariable mit vorangestelltem Klammeraffen.

6.2.6 Floppy und Kassette

Der CPC 6128 ist zwar herstellungsmäßig nicht mit einem Kassettenrecorder, dafür aber mit der wesentlich schnelleren Floppy ausgerüstet. Sie haben jedoch die Möglichkeit, einen Kassettenrecorder an der Rückseite Ihres CPC 6128 anzuschließen, um z.B. käufliche Software zu laden, die es nur auf Kassette gibt. Mit folgenden Anweisungen können Sie die Ein-/Ausgabekanäle zwischen Floppy und Kassette umschalten:

```
|TAPE
```

schaltet Ein- und Ausgabe auf Kassette. Die Anweisungen SAVE, LOAD, OPENIN, OPENOUT, CLOSEIN, CLOSEOUT, PRINT#9, WRITE#9, INPUT#9, LINE INPUT#9 und CAT beziehen sich jetzt nicht mehr auf die Floppy, sondern auf das Kassettenlaufwerk.

```
|DISC
```

lenkt die Ein-/Ausgabe wieder zurück auf die Floppy.

Darüber hinaus können Sie die Floppy als Eingabegerät und den Kassettenrecorder als Ausgabegerät bestimmen oder umgekehrt. Die entsprechenden Anweisungen lauten:

|TAPE.IN	Kassettenrecorder als Eingabegerät
|DISC.IN	Floppy als Eingabegerät
|TAPE.OUT	Kassettenrecorder als Ausgabegerät
|DISC.OUT	Floppy als Ausgabegerät

6.3 Sequentielle Dateien

Bisher hatten wir uns mit dem Abspeichern und Laden von Programmen sowie einigen wichtigen Hilfsmitteln und Dienstprogrammen beschäftigt. Mit dem Wissen aus den vergangenen Kapiteln können wir uns jetzt daran machen, eine professionelle Dateiverwaltung auf Diskette aufzubauen.

Nun ist Dateiverwaltung nicht gleich Dateiverwaltung. Die Überschrift dieses Absatzes enthält den Zusatz "sequentiell". Dies bedeutet, daß einzelne Begriffe oder Datensätze nacheinander auf Diskette geschrieben und ebenso wieder gelesen werden.

Wenn Sie z.B. eine Adressendatei mit 200 Namen und Adressen anlegen, wird zuerst der erste Name, dann der zweite und schließlich der 200. Name auf Diskette geschrieben. Wenn Sie nun die Namen alphabetisch sortiert haben und Sie suchen die Adresse von Franz Zeppelin, ist anzunehmen, daß dieser Name erst ziemlich gegen Ende der Datei, d.h. möglicherweise erst im 199. Eintrag aufzufinden ist.

Um nun den 199. Eintrag von Diskette zu lesen, müssen zunächst die 198 davor liegenden Einträge ebenfalls eingelesen werden. Es ist - jedenfalls mit den normalen AMSDOS-Anweisungen - nicht möglich, direkt auf diesen Eintrag zuzugreifen. Im nächsten Kapitel ist jedoch ein umfangreiches Assemblerprogramm abgedruckt, mit dessen Hilfe eine relative Datei mit Direktzugriff leicht aufzubauen ist.

Zunächst wollen wir aber noch ein wenig bei den Möglichkeiten verweilen, die uns AMSDOS von Haus aus mit auf den Weg gibt. So hat eine sequentielle Dateiverwaltung durchaus ihre Berechtigung. Ist nämlich die Anzahl der Datensätze relativ klein, so daß sie alle in den Arbeitsspeicher hineinpassen, liest man eben sämtliche 200 Datensätze in den Speicher und sucht sich dann mit Hilfe der Feldverarbeitung (s. Kapitel 5) den richtigen Namen heraus.

Auf diese Weise kann man jeden einzelnen Datensatz bearbeiten und schließlich die gesamte Datei wieder auf Diskette schreiben.

Bevor wir uns mit einem ausführlichen Anwendungsbeispiel zur sequentiellen Adressenverwaltung befassen, hier noch zwei kleine Programme, an denen Sie das Prinzip der sequentiellen Dateiverwaltung kennenlernen.

Das folgende Programm liest einige Begriffe aus DATA-Zeilen und schreibt sie auf Diskette:

```
10 REM Daten auf Diskette schreiben
15 OPENOUT "DUMMY" : MEMORY HIMEM - 1 : CLOSEOUT
20 OPENOUT "TEST"
30 READ a$
40 IF a$="Ende" THEN 70
50 PRINT#9,a$
60 GOTO 30
70 CLOSEOUT
80 DATA Wurst, Kaese, Eier, Fleisch, Gemuese
90 DATA Obst, Hering, Mehl, Rosinen, Ende
```

Zeile 15 öffnet zunächst eine "DUMMY"-Datei zur Ausgabe und schließt sie sogleich wieder. Diese Anweisungen sollten am Anfang eines jeden Programms stehen, das Daten auf Diskette schreibt bzw. von ihr liest. Hierdurch wird nämlich ein 2k großer Ein-/Ausgabepuffer im oberen Speicherbereich angelegt, den AMSDOS unbedingt benötigt. Ohne diese Maßnahme würde das Programm zwar auch laufen, jedoch umso langsamer, je mehr Strings bereits im Arbeitsspeicher abgelegt sind. Bei jedem Lesen bzw. Schreiben eines Datensatzes müßte zunächst eine Stringmüllbeseitigung (garbage collect) duchgeführt werden, die den Ablauf des Programms erheblich verlangsamen würde.

Zwischen dem Öffnen und dem Schließen der "DUMMY"-Datei steht noch die Anweisung

```
MEMORY HIMEM -1
```

Mit MEMORY wird die Speicherobergrenze des BASIC-Speichers festgelegt. Die reservierte Variable HIMEM zeigt auf die erste Speicherzelle des Ein-/Ausgabepuffers, der ja aus den eben genannten Gründen reserviert werden soll. HIMEM-1 ist dann die Speicherobergrenze für BASIC, d.h ein Byte unter der Anfangsadresse des Puffers.

Zeile 20 öffnet nun mit der Anweisung

```
OPENOUT "TEST"
```

die sequentielle Datei "TEST" zum Schreiben. Da wir den Ein-/Ausgabepuffer bereits angelegt haben, können alle folgenden Diskettenoperationen auf ihn zugreifen.

Die folgenden Anweisung sind uns größtenteils schon aus früheren Kapiteln bekannt. Hier werden die verschiedenen Begriffe aus den DATA-Zeilen gelesen und auf die Markierung "Ende" hin untersucht. Zeile 50 gibt die Werte diesmal nicht auf dem Bildschirm, sondern mit

```
PRINT#9,a$
```

auf die Floppy aus, auf die sich immer die Streamnummer 9 bezieht. Sind alle Werte gelesen und auf Diskette geschrieben, wird die Ausgabedatei in Zeile 70 mit

```
CLOSEOUT
```

wieder geschlossen.

Hier nun das Programm, das die Daten aus der Datei TEST wieder von der Floppy einliest und sowohl auf dem Bildschirm als auch auf dem Drucker ausgibt. Sollten Sie keinen Drucker angeschlossen haben, lassen Sie Zeile 50 einfach aus, da das Programm sonst in einer Warteschleife hängt.

```
10 REM Daten von Diskette lesen und ausdrucken
15 OPENOUT "DUMMY" : MEMORY HIMEM-1 : CLOSEOUT
20 OPENIN "TEST"
30 INPUT#9,a$
40 PRINT a$
50 PRINT#8,a$
60 IF EOF=0 THEN 30
70 CLOSEIN
```

Zunächst legt Zeile 15 wiederum den Ein-/Ausgabepuffer für die Floppy an. In Zeile 20 öffnet

```
OPENIN "TEST"
```

die Datei TEST zum Lesen. Zeile 30 liest mit

```
INPUT#9,a$
```

den ersten Datensatz ein und ordnet ihn der Stringvariablen a$ zu. Zeile 40 gibt ihn dann auf dem Bildschirm und Zeile 50 auf dem Drucker (Streamnummer 8) aus. Sehr wichtig ist Zeile 60, denn die reservierte Variable EOF (end of file) enthält den Wert Null, wenn die Datei noch nicht zu

Ende gelesen ist. In diesem Fall springt das Programm wieder zurück in Zeile 30, um den nächsten Datenwert von der Floppy einzulesen.

Sind dagegen alle Datenwerte gelesen, erhält EOF den Wert -1 und der Computer führt die Anweisung

```
CLOSEIN
```

aus, um die Eingabedatei wieder zu schließen.

Achten Sie darauf, daß sowohl Eingabe- wie auch Ausgabedateien wieder geschlossen werden müssen, wenn alle Daten geschrieben bzw. gelesen sind. Gleichzeitig dürfen höchstens immer nur eine Eingabe- und eine Ausgabedatei geöffnet sein.

6.3.1 Adressenverwaltungsprogramm (sequentiell)

Als ausführliches Beispiel für die sequentielle Dateiverwaltung auf Diskette wollen wir hier ein komplettes Adressenverwaltungsprogramm betrachten. Dieses Programm ist modular aufgebaut und menügesteuert. Es kann in leicht abgeänderter Form auch für jede andere Art von Dateiverwaltung eingesetzt werden.

Nach dem Laden und Starten erscheint zunächst das Menü, das folgendermaßen aussieht:

```
        Adressenverwaltung
        ==================

Adressen schreiben/fortschreiben                1

Adressen aendern/loeschen                       2

Adressen sortieren                              3

Adressen laden                                  4

Adressen abspeichern                            5

Adressen ausdrucken                             6

Adressen auf Etiketten ausdrucken               7


E n d e                                         e


Bitte waehlen Sie
```

Wie Sie sehen, bietet dieses Programm verschiedene Möglichkeiten zur Datenbehandlung. Menüpunkt 1 dient zum Schreiben und Fortschreiben der Adressendatei. Schreiben bedeutet, daß Sie eine Datei, die vorher leer war, von Anfang an mit Daten versehen. Mit Fortschreiben ist gemeint, daß eine Datei, die bereits angelegt wurde, nun erweitert werden soll. Die Datei kann vorher bereits auf Diskette abgespeichert gewesen sein oder gerade geschrieben und sortiert oder ausgedruckt werden usw.

Wenn Sie nun Menüpunkt 1 auswählen, werden Sie aufgefordert, verschiedene Daten einzugeben:

Adressen schreiben/fortschreiben

12

Anrede (0-5)? 1

Name ? Meier

Vorname ? Franz

Strasse ? Bergstr. 47

PLZ/Ort ? 8000 Muenchen 80

Naechste Eingabe - RETURN-Taste druecken

Zurueck ins Menue - Leertaste druecken

Dies ist ein komplettes Beispiel, wie die Daten eingegeben werden müssen. Direkt unter der Überschrift befindet sich eine Zahl, die angibt, den wievielten Datensatz wir gerade bearbeiten. In diesem Fall ist es der zwölfte. Eine bisher leere Datei beginnt immer mit dem 1. Datensatz, während eine fortgeschriebene, die z.B. von der Diskette eingelesen wurde, mit dem nächstfolgenden Datensatz fortfährt. Lesen Sie z.B. eine Datei mit 58 Datensätzen von der Diskette ein, erscheint beim Fortschreiben oben die Nummer 59. Die Datensätze werden zunächst in der Reihenfolge ihrer Eingabe abgelegt, ungeachtet der Tatsache, ob sie sich in alphabetischer Reihenfolge befinden oder nicht.

Bevor Sie nun die eigentlichen Daten eingeben, erscheint unter der fortlaufenden Nummer zunächst nur die Aufforderung, den Code für die Anrede einzugeben, während der Rest des Bildschirms noch leer bleibt. Bei der Anrede können Sie unter sechs Möglichkeiten auswählen:

Code	Anrede
0	keine
1	Herrn
2	Frau
3	Fraeulein
4	Herrn und Frau
5	Firma

Diese Codes sollten Sie sich vorher merken oder notieren, da sie einfacher einzugeben sind, als wenn Sie jedesmal die Anrede ausschreiben. Nach dem Drücken der RETURN-Taste erscheint die Aufforderung zur Eingabe des (Nach-)Namens. Sie geben ihn ein und drücken wiederum die RETURN-Taste. Das gleiche wiederholt sich dann für die Strasse und den Ort mit Postleitzahl. Ist ein Datensatz komplett eingegeben, erhalten Sie unten im Bildschirm den Hinweis, entweder die RETURN- oder die Leertaste zu drücken. Letztere führt Sie wieder zurück ins Menü, während Sie mit der RETURN-Taste gleich zum nächsten Datensatz kommen. Der gleiche Vorgang wiederholt sich dann von vorne, allerdings mit einer um eins erhöhten Datensatznummer.

Achten Sie darauf, daß die einzelnen Eingabewerte eine bestimmte Anzahl von Zeichen nicht überschreiten dürfen:

Name	15 Zeichen
Vorname	15 Zeichen
Straße	20 Zeichen
PLZ/Ort	21 Zeichen

Kommen wir jetzt zum 2. Menüpunkt, der zum Ändern und Löschen von Adressen dient. Sie werden zunächst aufgefordert, den betreffenden Namen als Suchindex einzugeben. Daraufhin sucht das Programm, ob dieser Name in der Datei vorhanden ist. Wenn nein, erscheint ein Hinweis "Nicht gefunden", und Sie können den nächsten Namen suchen oder ins Menü zurückkehren. Wurde der Name dagegen gefunden, erscheint der komplette Datensatz auf dem Bildschirm, ähnlich wie unter Menüpunkt 1, wobei der Cursor eine Zeile unter dem Anredecode steht. Wenn Sie keine Änderung vornehmen, drücken Sie einfach die RETURN-Taste, anderenfalls schreiben Sie den neuen Datenwert unmittelbar unter den alten und drücken ebenfalls die RETURN-Taste. Nach Beendigung des Korrekturvorgangs können Sie wie in Punkt 1 entweder fortfahren (RETURN-Taste) oder ins Menü zurückkehren.

Soll ein Datensatz gelöscht werden, wählen Sie ebenfalls Menüpunkt 2 und überschreiben den Anredecode mit dem Buchstaben l. Dann drücken Sie die RETURN-Taste, worauf der betreffende Datensatz aus der Datei entfernt wird. Durch den Löschvorgang rücken alle Datensätze mit höherer Nummer um eins nach unten, wodurch ihre Gesamtzahl um eins abnimmt.

Menüpunkt 3 sortiert die Datensätze in alphabetischer Reihenfolge nach dem Shell-Sortierverfahren, das wir bereits kennengelernt haben. Die Nummern der Datensätze ändern sich dadurch entsprechend.

Wir kommen nun zu den Floppy-Operationen (Menüpunkt 4 und 5). In beiden Fällen werden Sie aufgefordert, den Dateinamen einzugeben, unter dem die Adressen auf Diskette abgelegt sind bzw. abgelegt werden sollen.

Die letzten beiden Menüpunkte dienen zum Ausdrucken der Datei. Punkt 6 liefert einen normalen Listenausdruck, während Sie mit Punkt 7 Etiketten beschriften können. Dabei wird davon ausgegangen, daß die Etiketten genau 9 Druckzeilen auseinander liegen. Sollten Sie Etiketten anderer Größe verwenden, können Sie das Teilprogramm ab Zeile 3840 durch Hinzufügen oder Entfernen von PRINT#8-Anweisungen leicht anpassen.

Nachfolgend das komplette Programmlisting:

```
1000 REM Adressenverwaltungsprogramm (sequentiell)
1010 REM ==================================
1020 REM
1030 REM
1040 OPENOUT "dummy":MEMORY HIMEM-1:CLOSEOUT
1050 ma=200:REM max. Anzahl der Datensaetze
1060 MODE 1
1070 k=0:REM Datensaetze im Speicher
1080 ln=15:REM Zeichen Laenge Namen
1090 lv=15:REM Zeichen Laenge Vornamen
1100 ls=20:REM Zeichen Laenge Strasse
1110 lo=21:REM Zeichen Laenge PLZ/Ort
1120 la=1:REM Zeichen Laenge Anredecode
1130 cl$=CHR$(8):REM Cursor nach links
1140 cr$=CHR$(9):REM Cursor nach rechts
1150 cd$=CHR$(10):REM Cursor nach unten
1160 cu$=CHR$(11):REM Cursor nach oben
1170 n$="                                        "
1180 DIM n$(ma)
1190 DIM ar$(5)
1200 ar$(0)=""
1210 ar$(1)="Herrn"
1220 ar$(2)="Frau"
1230 ar$(3)="Fraeulein"
1240 ar$(4)="Herrn und Frau"
1250 ar$(5)="Firma"
1260 :
1270 :
1280 :
```

```
1290 REM Menue
1300 CLS
1310 PRINT TAB(10);"Adressenverwaltung"
1320 PRINT TAB(10);"==================="
1330 PRINT
1340 PRINT
1350 PRINT "Adressen schreiben/fortschreiben      1"
1360 PRINT
1370 PRINT "Adressen aendern/loeschen             2"
1380 PRINT
1390 PRINT "Adressen sortieren                    3"
1400 PRINT
1410 PRINT "Adressen laden                        4"
1420 PRINT
1430 PRINT "Adressen abspeichern                  5"
1440 PRINT
1450 PRINT "Adressen ausdrucken                   6"
1460 PRINT
1470 PRINT "Adressen auf Etiketten ausdrucken     7"
1480 PRINT
1490 PRINT
1500 PRINT "E n d e                               e"
1510 PRINT
1520 PRINT
1530 PRINT "Bitte waehlen Sie"
1540 m$=INKEY$:IF m$=""THEN 1540
1550 IF m$="e"THEN END
1560 m=VAL(m$)
1570 IF m<1 OR m>7 THEN 1540
1580 ON m GOSUB 1630,2030,2720,3020,3280,3530,3840
1590 GOTO 1290
1600 :
1610 :
1620 :
1630 REM Adressen schreiben/fortschreiben
1640 CLS
1650 PRINT
1660 PRINT
1670 PRINT "    Adressen schreiben/fortschreiben    "
1680 PRINT
1690 PRINT
1700 k=k+1
1710 PRINT k
```

```
1720 PRINT
1730 IF k>ma THEN PRINT "Datei voll":GOTO 1900
1740 PRINT "anrede (0-5)";:INPUT a$
1750 a=VAL(a$)
1760 IF a<0 OR a>5 OR LEN(a$)<>1 THEN PRINT cu$;:GOTO 1740
1770 PRINT
1780 PRINT "Name         ";:INPUT na$
1790 PRINT
1800 na$=LEFT$(na$+n$,ln)
1810 PRINT "Vorname      ";:INPUT nv$
1820 PRINT
1830 nv$=LEFT$(nv$+n$,lv)
1840 PRINT "Strasse      ";:INPUT ns$
1850 PRINT
1860 ns$=LEFT$(ns$+n$,ls)
1870 PRINT "PLZ, Ort     ";:INPUT no$
1880 no$=LEFT$(no$+n$,lo)
1890 n$(k)=na$+nv$+ns$+no$+RIGHT$(STR$(a),1)
1900 PRINT
1910 PRINT
1920 PRINT "Naechste Eingabe - Enter-Taste druecken"
1930 PRINT
1940 PRINT"Zurueck ins Menue - Leertaste druecken"
1950 m$=INKEY$:IF m$=""THEN 1950
1960 IF m$=CHR$(13)THEN 1630
1970 IF m$=" "THEN 1990
1980 GOTO 1950
1990 RETURN
2000 :
2010 :
2020 :
2030 REM Adressen aendern/loeschen
2040 CLS
2050 PRINT
2060 PRINT
2070 PRINT "       Adressen aendern/loeschen"
2080 PRINT
2090 PRINT
2100 PRINT "Name         ";:INPUT na$
2110 na$=LEFT$(na$+n$,ln)
2120 i=1
2130 WHILE i<=k
2140 IF LEFT$(n$(i),ln)=na$ THEN 2200
```

```
2150 i=i+1
2160 WEND
2170 PRINT
2180 PRINT "nicht gefunden"
2190 GOTO 2590
2200 na$=LEFT$(n$(i),ln)
2210 nv$=MID$(n$(i),ln+1,lv)
2220 ns$=MID$(n$(i),ln+lv+1,ls)
2230 no$=MID$(n$(i),ln+lv+ls+1,lo)
2240 a$=RIGHT$(n$(i),1)
2250 PRINT i
2260 PRINT
2270 PRINT "Anrede (0-5)  ";a$
2280 PRINT
2290 PRINT "Name          ";na$
2300 PRINT
2310 PRINT "Vorname       ";nv$
2320 PRINT
2330 PRINT "Strasse       ";ns$
2340 PRINT
2350 PRINT "PLZ, Ort      ";no$
2360 FOR q=1 TO 8:PRINT cu$;:NEXT q
2370 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2380 IF ww$<>"" THEN a$=ww$
2390 a=VAL(a$)
2400 IF a$="l"THEN GOSUB 4120:GOTO 2590
2410 IF a<0 OR a>5 OR LEN(a$)<>1 THEN PRINT cu$;:GOTO 2370
2420 PRINT
2430 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2440 IF ww$<>"" THEN na$=ww$
2450 PRINT
2460 na$=LEFT$(na$+n$,ln)
2470 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2480 IF ww$<>"" THEN nv$=ww$
2490 PRINT
2500 nv$=LEFT$(nv$+n$,lv)
2510 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2520 IF ww$<>"" THEN ns$=ww$
2530 PRINT
2540 ns$=LEFT$(ns$+n$,ls)
2550 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2560 IF ww$<>"" THEN no$=ww$
2570 no$=LEFT$(no$+n$,lo)
```

```
2580 n$(i)=na$+nv$+ns$+no$+RIGHT$(STR$(a),1)
2590 PRINT
2600 PRINT
2610 PRINT "Naechste Eingabe - Enter-Taste druecken"
2620 PRINT
2630 PRINT "Zurueck ins Menue - Leertaste druecken"
2640 m$=INKEY$:IF m$="" THEN 2640
2650 IF m$=CHR$(13) THEN 2030
2660 IF m$=" " THEN 2680
2670 GOTO 2640
2680 RETURN
2690 :
2700 :
2710 :
2720 REM Adressen sortieren
2730 CLS
2740 PRINT
2750 PRINT
2760 PRINT "          Adressen sortieren"
2770 PRINT
2780 PRINT
2790 g=INT(k/2)
2800 WHILE g>0
2810 FOR i=g+1 TO k
2820 j=i-g
2830 WHILE j>0
2840 IF n$(j)<=n$(j+g) THEN j=0:GOTO 2890
2850 a$=n$(j+g)
2860 n$(j+g)=n$(j)
2870 n$(j)=a$
2880 j=j-g
2890 WEND
2900 NEXT
2910 g=INT(g/2)
2920 WEND
2930 PRINT
2940 PRINT
2950 PRINT "          Zurueck ins Menue"
2960 PRINT "       beliebige Taste druecken"
2970 m$=INKEY$:IF m$="" THEN 2970
2980 RETURN
2990 :
3000 :
```

```
3010 :
3020 REM Adressendatei laden
3030 CLS
3040 PRINT
3050 PRINT
3060 PRINT "          Adressen laden"
3070 PRINT
3080 PRINT
3090 INPUT "Dateiname";dn$
3100 PRINT
3110 OPENIN dn$
3120 k=0
3130 IF k=ma THEN PRINT "Datei voll":GOTO 3180
3140 k=k+1
3150 PRINT k;cu$
3160 INPUT#9,n$(k)
3170 IF EOF=0 THEN 3130
3180 CLOSEIN
3190 PRINT
3200 PRINT
3210 PRINT "          Zurueck ins Menue"
3220 PRINT "       beliebige Taste druecken"
3230 m$=INKEY$:IF m$="" THEN 3230
3240 RETURN
3250 :
3260 :
3270 :
3280 REM Adressendatei abspeichern
3290 CLS
3300 PRINT
3310 PRINT
3320 PRINT "       Adressen abspeichern"
3330 PRINT
3340 PRINT
3350 IF k=0 THEN PRINT "Datei leer":PRINT:GOTO 3440
3360 INPUT "Dateiname";dn$
3370 PRINT
3380 OPENOUT dn$
3390 FOR i=1 TO k
3400 PRINT i;cu$
3410 PRINT#9,n$(i)
3420 NEXT i
3430 CLOSEOUT
```

```
3440 PRINT
3450 PRINT
3460 PRINT "          Zurueck ins Menue"
3470 PRINT "       beliebige Taste druecken"
3480 m$=INKEY$:IF m$="" THEN 3480
3490 RETURN
3500 :
3510 :
3520 :
3530 REM Adressendatei ausdrucken
3540 CLS
3550 PRINT
3560 PRINT
3570 PRINT "       Adressen ausdrucken"
3580 PRINT
3590 PRINT
3600 FOR i=1 TO k
3610 PRINT#8,LEFT$(n$(i),ln);
3620 PRINT#8," ";
3630 PRINT#8,MID$(n$(i),ln+1,lv);
3640 PRINT#8," ";
3650 PRINT#8,MID$(n$(i),ln+lv+1,ls);
3660 PRINT#8," ";
3670 PRINT#8,MID$(n$(i),ln+lv+ls+1,lo);
3680 PRINT#8," ";
3690 PRINT#8,RIGHT$(n$(i),1)
3700 IF i-INT(i/64)*64<>0 THEN 3740
3710 FOR j=1 TO 8
3720 PRINT#8
3730 NEXT j
3740 NEXT i
3750 PRINT
3760 PRINT
3770 PRINT "          Zurueck ins Menue"
3780 PRINT"       beliebige Taste druecken"
3790 m$=INKEY$:IF m$="" THEN 3790
3800 RETURN
3810 :
3820 :
3830 :
3840 REM Adressen auf Etiketten drucken
3850 CLS
3860 PRINT
```

```
3870 PRINT
3880 PRINT " Adressen auf Etiketten ausdrucken"
3890 PRINT
3900 PRINT
3910 FOR i=1 TO k
3920 PRINT#8
3930 PRINT#8
3940 a=VAL(RIGHT$(n$(i),1))
3950 PRINT#8,ar$(a)
3960 PRINT#8,LEFT$(n$(i),ln+lv)
3970 PRINT#8,MID$(n$(i),ln+lv+1,ls)
3980 PRINT#8
3990 PRINT#8,MID$(n$(i),ln+lv+ls+1,lo)
4000 PRINT#8
4010 PRINT#8
4020 NEXT i
4030 PRINT
4040 PRINT
4050 PRINT "          Zurueck ins Menue"
4060 PRINT "       beliebige Taste druecken"
4070 m$=INKEY$:IF m$="" THEN 4070
4080 RETURN
4090 :
4100 :
4110 :
4120 REM Datensatz loeschen
4130 FOR j=i+1 TO k
4140 n$(j-1)=n$(j)
4150 NEXT j
4160 k=k-1
4170 FOR q=1 TO 8:PRINT cd$;:NEXT q
4180 PRINT "geloescht"
4190 PRINT cu$;cu$;
4200 RETURN
```

Wir wollen uns nun einige wichtige Einzelheiten zum Programmaufbau ansehen. Zunächst wird in Zeile 1040 der Ein-/Ausgabepuffer für die Floppy reserviert. Über die Größe der Datensätze geben die folgenden Zeilen Auskunft, so daß Sie sie leicht Ihren eigenen Wünschen anpassen können.

Zunächst gibt die Variable ma die maximale Anzahl der Datensätze an. Sie ist hier mit 200 vorbelegt, kann aber noch um einiges erhöht werden, falls

dies nötig sein sollte. Die einzige Begrenzung ist dabei in der Größe des BASIC-Speichers gegeben.

Die Variable k gibt an, wieviele Datensätze sich gerade im Speicher befinden. Sie kann an jeder Stelle des Programms abgefragt werden, sofern Sie noch zusätzliche Kontrollmöglichkeiten einbauen. Die folgenden Variablen ln, lv, ls und lo geben die Längen für die einzelnen Datenwerte an und können bei Bedarf ebenfalls verändert werden.

Sämtliche Datenwerte eines Datensatzes bilden zusammen einen String, der genau 72 Zeichen lang und selbst wieder ein Element des Feldes n$(i) ist. Zeile 1180 führt die Dimensionierung für dieses eindimensionale Feld mit ma Elementen durch.

In jedem Datensatz haben die einzelnen Werte ihren festen Platz, wie die folgende Aufstellung zeigt:

Name	15 Zeichen von	Position	1 bis 15
Vorname	15 Zeichen von	Position	16 bis 29
Straße	20 Zeichen von	Position	31 bis 50
PLZ/Ort	21 Zeichen von	Position	51 bis 71
Anredecode	1 Zeichen	Position	72

Die einzelnen Werte können mit Hilfe der Stringoperationen leicht aus dem Gesamtstring eliminiert bzw. wieder zu diesem zusammengesetzt werden. Werden bei der Eingabe die zulässigen Längen überschritten, werden alle zusätzlichen Zeichen abgeschnitten. Werden dagegen weniger Zeichen eingegeben, werden sie mit Leerzeichen aufgefüllt, so daß ihre vorgeschriebene Länge in jedem Fall konstant bleibt.

Die einzelnen Anreden werden ebenfalls in einem Feld ar$(i) entsprechend ihrem Code abgelegt. Wirklich ausgegeben werden sie aber nur beim Etikettenausdruck.

Das Menü ist ähnlich aufgebaut wie dasjenige, das wir bereits im Programm zur Volumenberechnung verschiedener Körper kennengelernt haben. Die einzelnen Menüpunkte werden mit Hilfe der INKEY$-Anweisung in Zeile 1540 als Strings gelesen und dann in eine numerische Variable umgeformt. Eine Ausnahme bildet lediglich der Punkt e, der das Programm beendet. Schließlich ruft die ON...GOSUB-Anweisung in Zeile 1580 die entsprechenden Routinen auf.

Das Unterprogramm ab Zeile 1630 dient zur Eingabe und Aufbereitung der Eingabewerte, die gleichzeitig ihre vorgeschriebene Länge erhalten (s.o.). In Zeile 1890 werden sie schließlich zum Gesamtstring n$(k) zusammengesetzt.

Etwas komplizierter ist dagegen das Teilprogramm ab Zeile 2030, das die Adressen löscht und ändert. Zunächst wird der gesuchte Name angefordert und sämtliche Datensätze nach ihm durchsucht. Wird der Datensatz mit dem betreffenden Namen gefunden, erhält die Variable i die entsprechende Datensatznummer, wird er dagegen nicht gefunden, erscheint der Hinweis "nicht gefunden" auf dem Bildschirm.

Daraufhin werden sämtliche Werte eines Datensatzes an derselben Stelle wie bei ihrer Eingabe unter Menüpunkt 1 auf dem Bildschirm ausgegeben. Anschließend wird der Cursor eine Zeile unter den Anredecode gesetzt. Sie haben jetzt die Möglichkeit, den jeweiligen Datenwert unverändert zu übernehmen, indem Sie einfach die RETURN-Taste drücken (CHR$(13)). Der Cursor geht dann weiter zum nächsten Datenwert. Ansonsten müssen Sie den betreffenden Datenwert komplett neu eingeben und ebenfalls die RETURN-Taste drücken.

Zum Löschen eines Datensatzes wird der Anredecode außer auf die sonst zulässigen Werte von 0 bis 5 auch auf den Buchstaben l abgefragt (Zeile 2400). Daraufhin wird das Unterprogramm in Zeile 4120 aufgerufen, das den Löschvorgang vornimmt und alle Datensätze mit höherer Nummer um eine Stelle nach unten aufrückt. Die derzeitige Gesamtanzahl der Datensätze im Speicher wird dabei um eins reduziert (Variable k).

Alphabetisch sortiert werden die Datensätze im Unterprogramm ab Zeile 2720, wobei das schnelle Shell-Sortierverfahren zur Anwendung kommt, das wir bereits kennengelernt haben.

In den nächsten beiden Unterprogrammen werden die Datensätze von Diskette gelesen bzw. auf sie geschrieben.

Zu den Druckerroutinen selbst (Zeile 3530 und 3840) gibt es nicht mehr viel zu sagen. Lediglich beim Listenausdruck findet eine Prüfung statt, ob bereits 64 Zeilen hintereinander ausgedruckt wurden. Ist dies der Fall, wird durch 8 Leerzeilen ein Seitenvorschub erzeugt.

7 Befehlserweiterungen

Dieses Kapitel behandelt keine normalen Anweisungen, die Sie im Handbuch finden. Stattdessen lernen wir hier einige wichtige Befehlserweiterungen kennen, die Ihren CPC 6128 noch wertvoller und für professionelle Zwecke einsatzfähiger machen.

Im einzelnen handelt es sich hierbei um eine komfortable Fehlerabfrage der Floppy, bei der Sie die Fehlermeldungen wahlweise ausgeben können oder nicht. Als weiteres befassen wir uns mit dem Drucker. Sie können jetzt eine Abfrage in Ihre Programme einbauen, die Ihnen mitteilt, ob der Drucker eingeschaltet ist oder nicht, ohne daß der Computer in einer Warteschleife hängt, die Sie nicht mehr verlassen können. Auch können Sie jetzt den Drucker parallel zur Bildschirmausgabe schalten, so daß sämtliche Bildschirmausgaben auch auf dem Drucker erscheinen. Ein besonderer Anwendungsfall hierfür ist das Auflisten des Directories auf dem Drucker, was selbstverständlich auch mit dem CAT-Befehl durchgeführt werden kann.

Die wichtigste Erweiterung, nach der sich schon viele CPC-Anwender gesehnt haben, ist jedoch die relative Dateiverwaltung. Sie können jetzt per Direktzugriff (Random Access) Datensätze von BASIC aus in eine Datei schreiben und auch wieder daraus lesen, ohne sich um den Rest der Datei kümmern zu müssen.

Verständlicherweise können solche Erweiterungen nicht von BASIC aus realisiert werden, sondern benötigen ein umfangreiches Maschinenprogramm. Trotzdem ist es nicht notwendig, daß Sie Kenntnisse in der Z80-Assembler- oder Maschinenprogrammierung besitzen. In den folgenden Abschnitten 7.1. bis 7.3 werden zunächst die einzelnen Erweiterungen erläutert. Im Anschluß daran finden Sie in Abschnitt 7.4 einen BASIC-Lader für das Maschinenprogramm. Für alle diejenigen Leser, die sich für den Aufbau

des Programms näher interessieren, ist in Abschnitt 7.5 ein kommentiertes Assemblerlisting abgedruckt. Am Schluß des Kapitels lernen Sie ein Programm zur relativen Adressenverwaltung kennen, das die hier behandelten Befehlserweiterungen im praktischen Einsatz zeigt.

Bevor Sie nun die einzelnen Befehlserweiterungen ausprobieren, sollten Sie den in Absatz 7.4 abgedruckten BASIC-Lader in Ihren Computer eintippen und auf Diskette abspeichern. Wegen des größeren Umfangs werden die DATA-Zeilen in vier Blöcke (0 bis 3) unterteilt. Zum leichteren Auffinden von Tippfehlern wird von jedem Block eine Prüfsumme gebildet und abgefragt. Bei einer falschen Prüfsumme erscheint ein Hinweis, in welchem Block der Fehler zu suchen ist.

Das Maschinenprogramm belegt den oberen Speicher ab A000H. Wenn Sie die Erweiterungen in Ihre eigenen Programme einbinden möchten, gibt es drei Möglichkeiten: Entweder, Sie laden den BASIC-Lader, führen ihn aus, indem Sie ihn gleichzeitig initialisieren und laden daraufhin Ihr eigenes Programm; oder Sie hängen den Lader mit Hilfe von CHAIN-MERGE an Ihre Programme an und rufen ihn dann als Unterprogramm auf. Wahrscheinlich müssen Sie dazu die Programmzeilen des Laders mit RENUM neu durchnumerieren, so daß er höhere Zeilennummern erhält als Ihr BASIC-Programm. Achten Sie aber darauf, daß Sie in Zeile 1100 NEW durch RETURN ersetzen. Auch darf der Lader nur einmal ausgeführt bzw. das Maschinenprogramm nur einmal initialisiert werden, und zwar am Anfang Ihres Anwenderprogramms, da bei einer weiteren Initialisierung der Computer abstürzt. Dies können Sie verhindern, indem Ihr Anwenderprogramm gleich zu Beginn anhand von zwei oder drei Speicherinhalten überprüft, ob das Maschinenprogramm bereits geladen ist. In diesem Fall ist dann der entsprechende Unterprogrammaufruf zu überspringen.

Die letzte und bequemste Möglichkeit besteht darin, den Lader auszuführen und als Binärdatei auf Diskette zu schreiben. Dies geschieht mit

```
SAVE "ERWBIN",B,&A000,&54A
```

Sie können dann in Ihr Anwenderprogramm eine LOAD-Anweisung zum Laden des Maschinenprogramms einsetzen. Zuvor muß jedoch die Speicherobergrenze auf 9FFFH herabgesetzt werden. Nicht zu vergessen ist die Initialisierung des Maschinenprogramms durch CALL &A000 und die Abfrage, ob es sich bereits im Speicher befindet (s.o.). In dem am Schluß des Kapitels aufgeführten Adressenverwaltungsprogramm wird diese Methode praktiziert.

7.1 Floppy-Fehlerabfrage

Wir kennen bereits die Fehlerbehandlung, bei der das Programm bei Auftreten eines Fehlers in eine speziell hierfür vorgesehende Routine verzweigt. Dies geschieht mit Hilfe der ON ERROR GOTO-Anweisung und der Abfrage der Fehlernummer in ERR.

Auch für die Floppy enthält der CPC 6128 eine Fehlerabfrage, die allerdings nicht für alle Anwendungsbereiche zufriedenstellend arbeitet. Nachfolgend wollen wie sie kurz betrachten.

Es gibt zwei Floppy-Fehlermeldungen, die vom BASIC-Betriebssystem generiert werden und mit der Variablen ERR abzufragen sind:

ERR	Fehlermeldung	
31	File not found	(Datei nicht gefunden)
32	Broken in	(Abbruch in)

Darüber hinaus gibt es noch eine Reihe von AMSDOS-Fehlermeldungen, die mit der reservierten Variablen DERR abgefragt werden können (genaue Aufstellung siehe Anhang).

Dies alles wäre eine wunderbare Einrichtung, gäbe es nicht zwei unangenehme Begleiterscheinungen, die unter normalen Umständen nicht zu umgehen sind. Selbst wenn infolge der Anweisungen ON ERROR GOTO und RESUME eine Fehlerbehandlungsroutine ausgeführt wird, erscheint in jedem Fall die Fehlermeldung auf dem Bildschirm. Außerdem wird in manchen Fällen das Programm unterbrochen, allerdings nicht endgültig abgebrochen. Dies ist immer dann der Fall, wenn der Disc-Controller eine Fehlermeldung ausgibt. Hier ein Beispiel, wenn keine Diskette im Laufwerk liegt:

```
Drive A: disc missing

Retry, Ignore or Cancel?
```

Hier unterbricht das Programm und fordert Sie auf, entweder einen neuen Versuch zu unternehmen (Retry), die Meldung zu ignorieren (Ignore) oder zu annulieren (Cancel). Nur wenn Sie hier C eingeben, setzt das Programm seine Ausführung fort und verläßt die Fehlerbehandlungsroutine.

Da eine solche Fehlerbehandlung in vielen professionellen Programmen nicht erwünscht ist, wurde hier die Befehlserweiterung |DISCERROR geschaffen, die diesem Mißstand Abhilfe schafft. Sowohl die Fehlernummer, die normalerweise in DERR steht, als auch die Fehlermeldung als solche werden in Variablen abgelegt und erscheinen nur noch auf Wunsch und nicht mehr automatisch auf dem Bildschirm. Darüber hinaus unterbricht das Programm nicht mehr, wenn der Disc-Controller einen Fehler feststellt (s.o.), sondern legt automatisch das Zeichen C im Tastaturpuffer ab, das Sie normalerweise von Hand eingeben müßten.

Nun liegt es im Ermessen des Programmierers, welche Fehlerbehandlung er durchführt. Dies bringt andererseits eine größere Verantwortung mit sich, da die Fehlerbehandlung durch AMSDOS hier total ausgeschaltet ist und von sich aus keine Fehlermeldungen mehr ausgibt. Allerdings muß das BASIC-Programm eine Routine enthalten, die infolge von ON ERROR GOTO aufgerufen und mit RESUME NEXT wieder verlassen wird. Die normale Floppy-Fehlerabfrage mit DERR funktioniert jedoch nicht mehr und sollte deshalb nicht verwendet werden. Die Wirkungsweise unserer Fehlerbehandlung sehen Sie am besten an folgendem Beispiel. Probieren Sie es einmal aus, vergessen Sie aber nicht, vorher die Befehlserweiterung zu laden und zu initialisieren (s.o.).

```
10 ON ERROR GOTO 1000
20 e$="":e%=0
30 INPUT "Dateiname"; a$
40 OPENIN a$
50 |DISCERROR,@e%,@e$
60 PRINT "Fehlernummer"; e%
70 PRINT "Fehlermeldung"; e$
80 CLOSEIN
90 ON ERROR GOTO 0
100 END
1000 REM Fehlerbehandlungsroutine
1010 RESUME NEXT
```

Die ON ERROR GOTO-Anweisung in Zeile 10 sorgt dafür, daß im Falle eines Fehlers die Fehlerbehandlungsroutine ab Zeile 1000 aufgerufen wird. Diese muß unbedingt vorhanden sein, selbst wenn sie, wie in diesem Fall, nur RESUME NEXT enthält, was einen sofortigen Rücksprung in das aufrufende Programm zur Folge hat. Hier wird dann mit der Anweisung fortgefahren, die der fehlererzeugenden Anweisung unmittelbar folgt.

In Zeile 20 werden die Variablen e$ und e% angelegt. Es spielt keine Rolle, welchen Wert sie hier erhalten, sie müssen lediglich für die Fehlerabfrage im Variablenspeicher vorhanden sein. Dabei enthält e% später die Fehlernummer (anstelle von DERR) und e$ den Text der Fehlermeldung. Statt e% und e$ können Sie auch andere Variablennamen verwenden. Wichtig ist nur, daß die Fehlernummer in einer Integer-Variablen und die Meldung in einer Stringvariablen abgelegt ist.

Achten Sie bei dieser wie auch bei allen anderen Befehlserweiterungen unbedingt darauf, daß diese Variablen richtig angelegt und geschrieben werden, da die Fehlerabfrage anderenfalls nicht richtig arbeitet und darüber hinaus auch keine gesonderte Fehlermeldung erzeugt. Dies erkennen Sie am besten daran, wenn Sie das Programm mit einem Dateinamen ausführen, der auf der eingelegten Diskette vorhanden ist. Enthält die Variable e$ nicht den Text "Ok", haben Sie vielleicht statt e% nur e geschrieben, oder in der eigentlichen Fehlerabfrageroutine in Zeile 50 nur eine statt zwei Variablen angegeben oder den Klammeraffen vergessen.

Die eigentliche Fehlerbehandlung geschieht durch Öffnen einer Datei zum Lesen (OPENIN) in Zeile 40. In Zeile 50 fragt dann |DISCERROR ab, ob dies ordnungsgemäß geschehen ist. Wenn ja, erhält e% den wert 0 oder 22 und e$ die Meldung "Ok". Wurde bei OPENIN ein Fehler festgestellt, existiert z.B. der Dateiname nicht oder liegt keine Diskette im Laufwerk, erhält e% die entsprechende Fehlernummer und e$ den dazugehörigen Text, der sonst auf dem Bildschirm erscheinen würde.

Es liegt nun bei Ihnen, Fehlernummer und Text auszugeben oder es zu unterlassen. Zur Demonstration geben wir hier beides auf dem Bildschirm aus (Zeile 60 und 70), schließen die Eingabedatei wieder und schalten den Sprung in die Fehlerbehandlungsroutine mit ON ERROR GOTO 0 wieder ab. Diese Maßnahme ist besonders zu empfehlen, da ansonsten sämtliche anderen Fehler unter Umständen nicht angezeigt werden. Haben Sie sich z.B. bei der Eingabe Ihres Programms vertippt, würde ein "Syntax error" nicht erkannt werden und Sie würden sich wundern, warum Ihr Programm nicht richtig läuft.

7.2 Drucker

Für den Drucker enthält unser Maschinenprogramm drei Anweisungen, die Sie sicher zu schätzen wissen. Vielleicht haben Sie den Wunsch, das Directory auf den Bildschirm auszugeben und den Drucker parallel zur Bildschirmausgabe zu schalten. Mit

```
|PRON
```

schalten Sie den Drucker zu und mit

```
|PROFF
```

wieder ab. Diese beiden Anweisungen haben die gleiche Wirkung wie CTRL-P unter CP/M. Allerdings findet hier keine gesonderte Prüfung statt, ob der Drucker eingeschaltet ist. Sie können lediglich die Erweiterung

```
p%=0 : |PRBUSY,@p%
```

in Ihre Programme einbauen. Ist der Drucker eingeschaltet, enthält die Integer-Variable p% den Wert Null, anderenfalls den Wert 255. Sie dürfen aber auch hier nicht vergessen, die Variable p% vorher anzulegen. Hier ein kleines Programmbeispiel:

```
10 p%=0 : |PRBUSY,@p%
20 IF p%=0 THEN 60
30 PRINT "Bitte Drucker einschalten"
40 PRINT "und dann beliebige Taste druecken"
50 a$=INKEY$:IF a$="" THEN 50 : ELSE 10
60 PRINT#8,"Hurra, ich bin eingeschaltet!"
```

Zeile 10 legt zunächst die Variable p% an und führt die Erweiterung |PRBUSY aus. Enthält p% jetzt den Wert Null, meldet sich Ihr Drucker in Zeile 60 mit "Hurra, ich bin eingeschaltet!" Falls Sie das Einschalten vergessen haben, werden Sie in Zeile 30 bis 60 gebeten, dies nachzuholen und dann eine beliebige Taste zu drücken.

7.3 Relative Dateiverwaltung

In diesem Abschnitt lernen wir die bedeutendste Befehlserweiterung kennen, nämlich das Arbeiten mit relativen Dateien. Leider sieht AMSDOS diese Möglichkeit nicht vor, selbst nicht über Sprungvektoren, die im Handbuch stehen.

Es gibt aber dennoch einen brauchbaren Weg, relative Dateien zu erzeugen. Beim Studium der Floppy-ROM-Routinen fällt nämlich auf, daß die Floppy AMSDOS-Dateien in sogenannten Records, das sind Aufzeichnungseinheiten zu je 128 Bytes, auf Diskette schreibt und sie auch wieder davon liest. Dies geschieht unabhängig von der Größe der physikalischen Sektoren, die 512 Bytes umfassen.

Glücklicherweise gibt es AMSDOS-Routinen, die - ähnlich wie unter CP/M - einen bestimmten Record einer existierenden Datei lesen und beschreiben können, ohne die davorliegenden Records erst einlesen zu müssen. Dies ist die Voraussetzung, die für die Realisierung des Direktzugriffs gegeben sein muß.

Eine relative Datei muß zunächst einmal als sequentielle Datei angelegt werden. Dies geschieht am besten dadurch, daß wir mit SPACE$ Leerstrings erzeugen, die exakt der gewünschten Datensatzlänge entsprechen. Um die ralative Datei später wahlweise auch sequentiell lesen zu können, fügen wir an jeden Leerstring noch ein Carriage Return- (CHR$(13)) und ein Line Feed-Zeichen (CHR$(10)) an. Allerdings ist das zum Arbeiten mit der relativen Datei nicht unbedingt erforderlich und soll hier nur zur Demonstration dienen. Immerhin werden für jeden Datensatz zwei Bytes mehr benötigt.

Jetzt schreiben wir die gleiche Anzahl von Leerstrings (einschließlich CR und LF) in die sequentielle Datei, die die relative Datei an Datensätzen erhalten soll. Nun können wir mit der Befehlserweiterung

```
|RECWRITE
```

jeden einzelnen Datensatz beschreiben und ihn mit

```
|RECREAD
```

wieder lesen.

Übrigens muß die Datensatzlänge keinesfalls mit der Recordlänge von 128 Bytes übereinstimmen, sondern kann zwischen 1 und 255 Zeichen umfassen, selbst wenn ein Datensatz über zwei oder drei Records verteilt ist. Die Befehlserweiterung ist deshalb so aufgebaut, daß ein String mit gegebener Länge ab dem n-ten Zeichen der Datei geschrieben bzw. gelesen wird, wobei das erste Zeichen in der Datei die Nummer null hat. Das Maschinenprogramm errechnet hieraus automatisch den zugehörigen Record und die Position in dem Record, ab der gelesen oder geschrieben werden soll.

Bei jedem Schreib- und Lesevorgang werden immer drei hintereinanderliegende Records in einen Puffer gelesen bzw. von dort aus wieder auf Diskette geschrieben. Die Anzahl der zu lesenden bzw. zu schreibenden Bytes ist in der Stringlänge gegeben.

Diesen Vorgang wollen wir uns anhand eines Beispiels verdeutlichen. Angenommen, wir haben eine Datensatzlänge von 200 Bytes, einschließlich CR und LF und möchten den 38. Datensatz lesen und ihn in der Stringvariablen a$ ablegen. In diesem Fall müssen wir zunächst mit

```
a$ = SPACE$(198)
```

einen Leerstring mit 198 Zeichen (200 abzüglich CR und LF) erzeugen. Als nächstes müssen drei aufeinanderfolgende Records in den Puffer gelesen werden, beginnend mit dem, der das

```
38 x 200 = 7600. Zeichen
```

der Datei enthält. Nun ist

```
7600/128 = 59 Rest 48
```

weshalb wir die Records 59, 60 und 61 einlesen müssen. Da die Bytezählung der Records nicht bei Byte 1, sondern bei Byte 0 beginnt, greifen wir hier auf Record 59, Byte 47 zu. Der 38. Datensatz ist somit in

```
Record 59, Byte 47 bis 127  ( 80 Bytes)  und
Record 60, Byte  0 bis 117  (118 Bytes) (ohne CR, LF)
```

aufgeteilt. Auf den dritten Record 61 brauchen wir in diesem Fall nicht zugreifen. Er wird aber jeweils mitgelesen und geschrieben. Schließlich wird der Datensatz aus dem Recordpuffer in den vorher angelegten Leerstring zur weiteren Bearbeitung übertragen.

Beim Schreiben werden zunächst die drei betreffenden Records von Diskette gelesen. Dann wird der Inhalt des Datensatzes in den Recordpuffer an der richtigen Stelle abgelegt, worauf die drei Records wieder auf Diskette geschrieben werden.

Die Größe der relativen Datei ist lediglich durch die Aufzeichnungskapazität der Diskette beschränkt, die im Datenformat bei 180K liegt.

Beachten Sie, daß sowohl vor Schreib- als auch Lesezugriffen in relativen Dateien immer die OPENIN-Anweisung ausgeführt werden muß, aber niemals OPENOUT, da dies eine neue Datei anlegt und schlimme Folgen haben kann.

Ebenfalls schlimme Folgen hat ein Schreibzugriff auf eine höhere Datensatznummer, als für die die Datei ausgelegt ist. Deshalb sollten Sie eine Sicherheitsabfrage in Ihre Programme einbauen, die so etwas verhindert. Darüber hinaus sollten beim Anlegen der Datei immer noch zwei zusätzliche Records reserviert werden, indem mindestens 2 x 128 Bytes hinten angefügt werden. Dies bewahrt Sie vor unangenehmen Überraschungen, wenn Sie auf einen der letzten Datensätze zugreifen.

Hier nun drei kurze Programme, die das Arbeiten mit relativen Dateien demonstrieren. Ein ausführliches Beispiel stellt dann die relative Adressenverwaltung am Schluß des Kapitels dar.

Zunächst ein Programm, das eine relative Datei als sequentielle Datei anlegt:

```
10 REM Anlegen einer relativen Datei
20 OPENOUT "dummy":MEMORY HIMEM-1:CLOSEOUT
30 OPENOUT "reldatei"
40 FOR i=1 TO 300
50 PRINT#9,RIGHT$("   "+STR$(i),3)+". Datensatz"
60 NEXT i
70 PRINT #9,SPACE$(128);SPACE$(128)
80 CLOSEOUT
```

In Zeile 20 wird zunächst wieder ein Ein-/Ausgabepuffer reserviert. Zeile 30 öffnet die sequentielle Datei "reldatei". Anschließend werden in einer FOR...NEXT-Schleife 300 Datensätze mit folgendem Inhalt auf Diskette geschrieben:

1. Datensatz
2. Datensatz
.
.
.
300. Datensatz

Jeder Datensatz ist 14 Zeichen lang. Beim sequentiellen Schreiben auf Diskette wird jeweils noch ein CR und LF angefügt, so daß wir beim relativen Lesen und Schreiben von einer Gesamt-Datensatzlänge von 16 Zeichen ausgehen müssen, obwohl wir nur 14 benötigen. Zeile 70 schreibt schließlich sicherheitshalber noch zwei Leerstrings mit je 128 Bytes auf Diskette.

Jetzt wollen wir die Datei wieder lesen, allerdings nicht sequentiell, sondern per Direktzugriff. Dabei beginnen wir mit dem 300. Datensatz und hören mit dem 1. auf:

```
10 REM Relative Datei lesen
20 OPENIN "reldatei"
30 FOR i=300 TO 1 STEP -1
40 a$=SPACE$(14)
50 z=(i-1)*16
60 |RECREAD,@z,@a$
70 PRINT a$
80 NEXT i
90 CLOSEIN
```

Zeile 20 öffnet die Datei "reldatei", diesmal aber zum relativen Lesen. Zeile 40 definiert den String a$, in dem die Datensätze abgelegt werden sollen. Die absolute Position z des Datensatzes in der Datei wird in Zeile 50 errechnet. Zeile 60 liest den Datensatz aufgrund von z und a$. ein.

Als letztes Beispiel hier noch ein Programm, daß Ihnen erlaubt, einen bestimmten Datensatz neu zu schreiben. Beachten Sie, daß Sie die relative Datei hier mit OPENIN zum Schreiben öffnen müssen:

```
10 REM Datensatz schreiben
20 OPENIN "reldatei"
30 INPUT "Datensatz-Nr";i
40 IF i<1 OR I>300 THEN 30
50 a$=SPACE$(14)
60 z=(i-1)*16
70 |RECREAD,@z,@a$
80 PRINT a$
90 INPUT "Neuer Datensatz"; a$
100 a$=LEFT$(a$+SPACE$(14),14)
110 |RECWRITE,@z,@a$
120 CLOSEIN
```

Das Programm fragt, welchen Datensatz Sie neu schreiben möchten. Es folgt eine Abfrage, ob Sie eine gültige Datensatznummer eingegeben haben. Wenn ja, wird der betreffende Datensatz zunächst einmal gelesen, damit Sie wissen, was in ihm steht. Dann geben Sie ihn neu ein. Zeile 100 formt ihn in die richtige Länge von 14 Zeichen um, worauf er in Zeile 110 auf Diskette geschrieben wird.

Probieren Sie dieses Programm einmal aus und schreiben Sie in einen der 300 Datensätze etwas anderes hinein. Schließlich rufen Sie das Programm erneut auf und vergewissern sich, ob der Datensatz auch richtig geschrieben wurde.

7.4 BASIC-Lader

```
1000 REM Befehlserweiterung fuer CPC-Rechner
1010 REM ==============================
1020 :
1030 MEMORY &A000-1
1040 RESTORE
1050 a=&A000:b=&A0FF:bl=0:z=29615:GOSUB 1120
1060 a=&A100:b=&A1FF:bl=1:z=33348:GOSUB 1120
1070 a=&A200:b=&A2FF:bl=2:z=27503:GOSUB 1120
1080 a=&A300:b=&A378:bl=3:z=19412:GOSUB 1120
1090 CALL &A000
1100 NEW
1110 :
1120 ges=0
```

```
1130 FOR i=a TO b
1140 READ c
1150 POKE i,c
1160 ges=ges+c
1170 NEXT
1180 IF ges<>z THEN PRINT "Fehler in Block";bl:END
1190 RETURN
1200 :
1210 REM Block 0
1220 DATA &01,&0C,&A0,&21,&48,&A0,&CD,&D1
1230 DATA &BC,&C3,&7E,&A0,&20,&A0,&C3,&A4
1240 DATA &A1,&C3,&C8,&A2,&C3,&06,&A3,&C3
1250 DATA &E5,&A1,&C3,&EF,&A1,&C3,&F5,&A1
1260 DATA &44,&49,&53,&43,&45,&52,&52,&4F
1270 DATA &D2,&52,&45,&43,&52,&45,&41,&C4
1280 DATA &52,&45,&43,&57,&52,&49,&54,&C5
1290 DATA &50,&52,&4F,&CE,&50,&52,&4F,&46
1300 DATA &C6,&50,&52,&42,&55,&53,&D9,&00
1310 DATA &00,&00,&00,&00,&00,&00,&00,&00
1320 DATA &00,&00,&00,&00,&DF,&58,&A0,&C9
1330 DATA &4C,&CD,&07,&DF,&5F,&A0,&C9,&55
1340 DATA &CD,&07,&DF,&66,&A0,&C9,&58,&CD
1350 DATA &07,&DF,&6D,&A0,&C9,&61,&CD,&07
1360 DATA &DF,&74,&A0,&C9,&6A,&CD,&07,&DF
1370 DATA &7B,&A0,&C9,&6D,&CD,&07,&3A,&5A
1380 DATA &BB,&32,&4C,&A0,&2A,&5B,&BB,&22
1390 DATA &4D,&A0,&3E,&C3,&32,&5A,&BB,&21
1400 DATA &ED,&A0,&22,&5B,&BB,&3A,&06,&BB
1410 DATA &32,&4F,&A0,&2A,&07,&BB,&22,&50
1420 DATA &A0,&3E,&C3,&32,&06,&BB,&21,&14
1430 DATA &A1,&22,&07,&BB,&3E,&C3,&32,&77
1440 DATA &BC,&32,&80,&BC,&32,&83,&BC,&32
1450 DATA &8C,&BC,&32,&95,&BC,&32,&98,&BC
1460 DATA &21,&68,&A1,&22,&78,&BC,&21,&71
1470 DATA &A1,&22,&81,&BC,&21,&7A,&A1,&22
1480 DATA &84,&BC,&21,&83,&A1,&22,&8D,&BC
1490 DATA &21,&8C,&A1,&22,&96,&BC,&21,&95
1500 DATA &A1,&22,&99,&BC,&3E,&00,&32,&52
1510 DATA &A0,&32,&53,&A0,&C9,&F5,&3A,&E1
1520 DATA &A1,&B7,&CA,&FF,&A0,&F1,&2A,&E3
1530 DATA &A1,&77,&23,&22,&E3,&A1,&C9,&3A
1540 :
1550 REM Block 1
```

```
1560 DATA &52,&A0,&B7,&CA,&10,&A1,&F1,&F5
1570 DATA &CD,&2E,&BD,&38,&FB,&CD,&2B,&BD
1580 DATA &F1,&C3,&4C,&A0,&F5,&3A,&E1,&A1
1590 DATA &B7,&28,&04,&F1,&3E,&43,&C9,&F1
1600 DATA &C3,&4F,&A0,&F5,&E5,&3E,&01,&32
1610 DATA &E1,&A1,&21,&F9,&A4,&22,&E3,&A1
1620 DATA &E1,&F1,&C9,&F5,&C5,&D5,&E5,&32
1630 DATA &E2,&A1,&3E,&00,&32,&E1,&A1,&38
1640 DATA &0D,&E1,&D1,&C1,&F1,&E5,&F5,&E1
1650 DATA &2E,&BB,&E5,&F1,&E1,&C9,&21,&9E
1660 DATA &A1,&11,&F9,&A4,&01,&06,&00,&ED
1670 DATA &B0,&2A,&E3,&A1,&01,&06,&00,&09
1680 DATA &22,&E3,&A1,&E1,&D1,&C1,&F1,&C9
1690 DATA &CD,&23,&A1,&CD,&54,&A0,&C3,&33
1700 DATA &A1,&CD,&23,&A1,&CD,&5B,&A0,&C3
1710 DATA &33,&A1,&CD,&23,&A1,&CD,&62,&A0
1720 DATA &C3,&33,&A1,&CD,&23,&A1,&CD,&69
1730 DATA &A0,&C3,&33,&A1,&CD,&23,&A1,&CD
1740 DATA &70,&A0,&C3,&33,&A1,&CD,&23,&A1
1750 DATA &CD,&77,&A0,&C3,&33,&A1,&0D,&0A
1760 DATA &4F,&6B,&0D,&0A,&FE,&02,&C0,&DD
1770 DATA &6E,&00,&DD,&66,&01,&2B,&7E,&FE
1780 DATA &02,&C0,&23,&E5,&2A,&E3,&A1,&01
1790 DATA &F9,&A4,&37,&3F,&ED,&42,&2B,&2B
1800 DATA &2B,&2B,&7D,&E1,&77,&01,&FB,&A4
1810 DATA &23,&71,&23,&70,&DD,&6E,&02,&DD
1820 DATA &66,&03,&2B,&7E,&FE,&01,&C0,&23
1830 DATA &3A,&E2,&A1,&77,&23,&3E,&00,&77
1840 DATA &C9,&00,&00,&00,&00,&CD,&2E,&BD
1850 DATA &D8,&3E,&01,&32,&52,&A0,&C9,&3E
1860 DATA &00,&32,&52,&A0,&C9,&FE,&01,&C0
1870 DATA &DD,&6E,&00,&DD,&66,&01,&2B,&7E
1880 :
1890 REM Block 2
1900 DATA &FE,&01,&C0,&23,&CD,&2E,&BD,&30
1910 DATA &08,&3E,&FF,&32,&53,&A0,&C3,&16
1920 DATA &A2,&3E,&00,&32,&53,&A0,&3A,&53
1930 DATA &A0,&77,&23,&3E,&00,&77,&C9,&00
1940 DATA &00,&00,&00,&00,&00,&00,&00,&00
1950 DATA &00,&00,&C3,&C8,&A2,&C3,&06,&A3
1960 DATA &FE,&02,&C2,&AD,&A2,&DD,&6E,&02
1970 DATA &DD,&66,&03,&2B,&7E,&FE,&04,&C2
1980 DATA &AD,&A2,&23,&11,&1F,&A2,&01,&05
```

```
1990 DATA &00,&ED,&B0,&DD,&6E,&00,&DD,&66
2000 DATA &01,&2B,&7E,&FE,&02,&C2,&AD,&A2
2010 DATA &23,&11,&27,&A2,&01,&03,&00,&ED
2020 DATA &B0,&3A,&23,&A2,&FE,&81,&30,&0D
2030 DATA &21,&00,&00,&22,&24,&A2,&3E,&00
2040 DATA &32,&26,&A2,&37,&C9,&FE,&98,&D2
2050 DATA &AD,&A2,&21,&22,&A2,&CB,&FE,&3A
2060 DATA &23,&A2,&FE,&97,&28,&0F,&21,&23
2070 DATA &A2,&34,&2B,&CB,&3E,&2B,&CB,&1E
2080 DATA &2B,&CB,&1E,&18,&EA,&2A,&21,&A2
2090 DATA &22,&24,&A2,&3A,&20,&A2,&CB,&3F
2100 DATA &32,&26,&A2,&3A,&27,&A2,&FE,&00
2110 DATA &CA,&AD,&A2,&37,&C9,&37,&3F,&C9
2120 DATA &2A,&24,&A2,&23,&22,&24,&A2,&11
2130 DATA &29,&00,&2A,&7D,&BE,&19,&3A,&24
2140 DATA &A2,&77,&23,&3A,&25,&A2,&77,&C9
2150 DATA &CD,&30,&A2,&D0,&CD,&D5,&A2,&CD
2160 DATA &F1,&A2,&ED,&B0,&C9,&CD,&B7,&A2
2170 DATA &21,&79,&A3,&CD,&53,&A3,&CD,&B0
2180 DATA &A2,&21,&F9,&A3,&CD,&53,&A3,&CD
2190 DATA &B0,&A2,&21,&79,&A4,&CD,&53,&A3
2200 DATA &C9,&3A,&26,&A2,&5F,&16,&00,&21
2210 DATA &79,&A3,&19,&ED,&5B,&28,&A2,&3A
2215 :
2216 REM Block 3
2220 DATA &27,&A2,&4F,&06,&00,&C9,&CD,&30
2230 DATA &A2,&D0,&CD,&B7,&A2,&21,&79,&A3
2240 DATA &CD,&53,&A3,&CD,&F1,&A2,&EB,&ED
2250 DATA &B0,&CD,&B7,&A2,&21,&79,&A3,&CD
2260 DATA &5A,&A3,&CD,&B0,&A2,&21,&F9,&A3
2270 DATA &CD,&53,&A3,&CD,&F1,&A2,&EB,&ED
2280 DATA &B0,&CD,&B7,&A2,&21,&F9,&A3,&CD
2290 DATA &5A,&A3,&CD,&B0,&A2,&21,&79,&A4
2300 DATA &CD,&53,&A3,&CD,&F1,&A2,&EB,&ED
2310 DATA &B0,&CD,&B7,&A2,&21,&79,&A4,&CD
2320 DATA &5A,&A3,&C9,&DF,&57,&A3,&C9,&92
2330 DATA &D3,&07,&DF,&5E,&A3,&C9,&61,&A3
2340 DATA &07,&E5,&D5,&C5,&E5,&11,&08,&00
2350 DATA &CD,&98,&CA,&CD,&10,&D4,&D2,&A9
2360 DATA &D3,&EB,&E3,&CD,&F3,&D9,&C3,&A6
2370 DATA &D3
```

7.5 Assemblerlisting

```
                ;CPC-BEFEHLSERWEITERUNGEN
                ;========================
                ;
                ;
                          ORG  0A000H
                          LOAD 0A000H
                ;
                ;FIRMWARE-SPRUNGVEKTOREN
                ;----------------------
                ;
                KLLOGEXT:   EQU  0BCD1H
                TXTOUT:     EQU  0BB5AH
                KMWAIT:     EQU  0BB06H
                OPENIN:     EQU  0BC77H
                CHARIN:     EQU  0BC80H
                DIRIN:      EQU  0BC83H
                OPENOUT:    EQU  0BC8CH
                CHAROUT:    EQU  0BC95H
                DIROUT:     EQU  0BC98H
                MCPRINT:    EQU  0BD2BH
                PBUSY:      EQU  0BD2EH
                ;
                ;
                ;SPRUNGADRESSEN IN FLOPPY-ROM
                ;---------------------------
                ;
                DOPENIN:    EQU  0CD4CH
                DCHARIN:    EQU  0CD55H
                DDIRIN:     EQU  0CD58H
                DOPENOUT:   EQU  0CD61H
                DCHAROUT:   EQU  0CD6AH
                DDIROUT:    EQU  0CD6DH
                ;
                ;INITIALISIERUNG
                ;--------------
                ;
A000 010CA0     START:      LD   BC,RSX       ;BEFEHLS-
A003 2148A0                 LD   HL,BYTE      ;ERWEITERUNGEN
A006 CDD1BC                 CALL KLLOGEXT     ;EINBINDEN
A009 C37EA0                 JP   WEITER
```

```
41                      ;
42 A00C 20A0            RSX:        DW   TABELLE
43 A00E C3A4A1                      JP   DISCERROR
44 A011 C3C8A2                      JP   RECREAD
45 A014 C306A3                      JP   RECWRITE
46 A017 C3E5A1                      JP   PRON
47 A01A C3EFA1                      JP   PROFF
48 A01D C3F5A1                      JP   PRBUSY
49                      ;
50 A020 44495343        TABELLE:    DB   'DISCERRO','R'+80H
50 A024 4552524F
50 A028 D2
51 A029 52454352                    DB   'RECREA','D'+80H
51 A02D 4541C4
52 A030 52454357                    DB   'RECWRIT','E'+80H
52 A034 524954C5
53 A038 50524FCE                    DB   'PRO','N'+80H
54 A03C 50524F46                    DB   'PROF','F'+80H
54 A040 C6
55 A041 50524255                    DB   'PRBUS','Y'+80H
55 A045 53D9
56 A047 00                          DB   0
57                      ;
58 A048 00000000        BYTE:       DB   0,0,0,0
59                      ;
60 A04C 000000          TXTOUT1:    DB   0,0,0
61 A04F 000000          KMWAIT1:    DB   0,0,0
62                      ;
63 A052 00              PRINTFLAG:  DB   0
64 A053 00              BUSYFLAG:   DB   0
65                      ;
66 A054 DF              OPENIN1:    RST  18H
67 A055 58A0                        DW   OPENIN2
68 A057 C9                          RET
69 A058 4CCD            OPENIN2:    DW   DOPENIN
70 A05A 07                          DB   7
71                      ;
72 A05B DF              CHARIN1:    RST  18H
73 A05C 5FA0                        DW   CHARIN2
74 A05E C9                          RET
75 A05F 55CD            CHARIN2:    DW   DCHARIN
76 A061 07                          DB   7
77                      ;
```

```
 78 A062 DF       DIRIN1:     RST  18H
 79 A063 66A0                 DW   DIRIN2
 80 A065 C9                   RET
 81 A066 58CD     DIRIN2:     DW   DDIRIN
 82 A068 07                   DB   7
 83               ;
 84 A069 DF       OPENOUT1:   RST  18H
 85 A06A 6DA0                 DW   OPENOUT2
 86 A06C C9                   RET
 87 A06D 61CD     OPENOUT2:   DW   DOPENOUT
 88 A06F 07                   DB   7
 89               ;
 90 A070 DF       CHAROUT1:   RST  18H
 91 A071 74A0                 DW   CHAROUT2
 92 A073 C9                   RET
 93 A074 6ACD     CHAROUT2:   DW   DCHAROUT
 94 A076 07                   DB   7
 95               ;
 96 A077 DF       DIROUT1:    RST  18H
 97 A078 7BA0                 DW   DIROUT2
 98 A07A C9                   RET
 99 A07B 6DCD     DIROUT2:    DW   DDIROUT
100 A07D 07                   DB   7
101              ;
102 A07E 3A5ABB  WEITER:     LD   A,(TXTOUT)
103 A081 324CA0              LD   (TXTOUT1),A
104 A084 2A5BBB              LD   HL,(TXTOUT+1)
105 A087 224DA0              LD   (TXTOUT1+1),HL
106 A08A 3EC3                LD   A,0C3H
107 A08C 325ABB              LD   (TXTOUT),A
108 A08F 21EDA0              LD   HL,TXTOUT0
109 A092 225BBB              LD   (TXTOUT+1),HL
110              ;
111 A095 3A06BB              LD   A,(KMWAIT)
112 A098 324FA0              LD   (KMWAIT1),A
113 A09B 2A07BB              LD   HL,(KMWAIT+1)
114 A09E 2250A0              LD   (KMWAIT1+1),HL
115 A0A1 3EC3                LD   A,0C3H
116 A0A3 3206BB              LD   (KMWAIT),A
117 A0A6 2114A1              LD   HL,KMWAIT0
118 A0A9 2207BB              LD   (KMWAIT+1),HL
119              ;
120 A0AC 3EC3                LD   A,0C3H
```

```
121 A0AE 3277BC             LD   (OPENIN),A
122 A0B1 3280BC             LD   (CHARIN),A
123 A0B4 3283BC             LD   (DIRIN),A
124 A0B7 328CBC             LD   (OPENOUT),A
125 A0BA 3295BC             LD   (CHAROUT),A
126 A0BD 3298BC             LD   (DIROUT),A
127 A0C0 2168A1             LD   HL,OPENIN0
128 A0C3 2278BC             LD   (OPENIN+1),HL
129 A0C6 2171A1             LD   HL,CHARIN0
130 A0C9 2281BC             LD   (CHARIN+1),HL
131 A0CC 217AA1             LD   HL,DIRIN0
132 A0CF 2284BC             LD   (DIRIN+1),HL
133 A0D2 2183A1             LD   HL,OPENOUT0
134 A0D5 228DBC             LD   (OPENOUT+1),HL
135 A0D8 218CA1             LD   HL,CHAROUT0
136 A0DB 2296BC             LD   (CHAROUT+1),HL
137 A0DE 2195A1             LD   HL,DIROUT0
138 A0E1 2299BC             LD   (DIROUT+1),HL
139 A0E4 3E00               LD   A,0
140 A0E6 3252A0             LD   (PRINTFLAG),A
141 A0E9 3253A0             LD   (BUSYFLAG),A
142 A0EC C9                 RET
143              ;
144              ;
145              ;PATCH FUER TXT OUTPUT
146 A0ED F5      TXTOUT0:   PUSH AF
147 A0EE 3AE1A1             LD   A,(FLAG)          ;FLAG=0?
148 A0F1 B7                 OR   A
149 A0F2 CAFFA0             JP   Z,TXT
150 A0F5 F1                 POP  AF                ;NEIN IN
151 A0F6 2AE3A1             LD   HL,(ERRPOINTER) ;ERRORPUFFER
152 A0F9 77                 LD   (HL),A            ;ABLEGEN
153 A0FA 23                 INC  HL
154 A0FB 22E3A1             LD   (ERRPOINTER),HL
155 A0FE C9                 RET
156 A0FF 3A52A0  TXT:       LD   A,(PRINTFLAG)     ;PRINTFLAG
157 A102 B7                 OR   A                 ;=0?
158 A103 CA10A1             JP   Z,SCREEN
159 A106 F1                 POP  AF                ;NEIN AUF
160 A107 F5                 PUSH AF                ;DRUCKER
161 A108 CD2EBD  BUSY:      CALL PBUSY             ;AUSGEBEN
162 A10B 38FB               JR   C,BUSY
163 A10D CD2BBD             CALL MCPRINT
```

```
164 A110 F1       SCREEN:     POP  AF
165 A111 C34CA0               JP   TXTOUT1
166               ;
167               ;PATCH FUER KM WAIT CHAR
168 A114 F5       KMWAIT0:    PUSH AF
169 A115 3AE1A1               LD   A,(FLAG)  ;FLAG=0?
170 A118 B7                   OR   A
171 A119 2804                 JR   Z,KMW
172 A11B F1                   POP  AF       ;NEIN,'CANCEL'
173 A11C 3E43                 LD   A,'C'    ;FUER FEHLERMELDUNG
174 A11E C9                   RET           ;BEREITSTELLEN
175 A11F F1       KMW:        POP  AF
176 A120 C34FA0               JP   KMWAIT1
177               ;
178 A123 F5       INIT:       PUSH AF
179 A124 E5                   PUSH HL
180 A125 3E01                 LD   A,1              ;FLAG=1
181 A127 32E1A1               LD   (FLAG),A
182 A12A 21F9A4               LD   HL,ERRBUFFER     ;ERRORPUFFER
183 A12D 22E3A1               LD   (ERRPOINTER),HL ;INITIALIS.
184 A130 E1                   POP  HL
185 A131 F1                   POP  AF
186 A132 C9                   RET
187               ;
188 A133 F5       FINISH:     PUSH AF
189 A134 C5                   PUSH BC
190 A135 D5                   PUSH DE
191 A136 E5                   PUSH HL
192 A137 32E2A1               LD   (ERRORNR),A
193 A13A 3E00                 LD   A,0              ;FLAG=0
194 A13C 32E1A1               LD   (FLAG),A
195 A13F 380D                 JR   C,ENDE        ;KEIN FEHLER,OK
196 A141 E1                   POP  HL
197 A142 D1                   POP  DE
198 A143 C1                   POP  BC
199 A144 F1                   POP  AF
200 A145 E5                   PUSH HL
201 A146 F5                   PUSH AF
202 A147 E1                   POP  HL
203 A148 2EBB                 LD   L,0BBH        ;PROZESSORFLAG
204 A14A E5                   PUSH HL
205 A14B F1                   POP  AF
206 A14C E1                   POP  HL
```

```
207 A14D C9                     RET
208 A14E 219EA1   ENDE:         LD   HL,OK          ;KEIN FEHLER
209 A151 11F9A4                 LD   DE,ERRBUFFER   ;OK BEREITST.
210 A154 010600                 LD   BC,6
211 A157 EDB0                   LDIR
212 A159 2AE3A1                 LD   HL,(ERRPOINTER)
213 A15C 010600                 LD   BC,6
214 A15F 09                     ADD  HL,BC
215 A160 22E3A1                 LD   (ERRPOINTER),HL
216 A163 E1                     POP  HL
217 A164 D1                     POP  DE
218 A165 C1                     POP  BC
219 A166 F1                     POP  AF
220 A167 C9                     RET
221                 ;
222                 ;PATCHES FUER DISCROUTINEN
223 A168 CD23A1   OPENIN0:      CALL INIT           ;DISC IN OPEN
224 A16B CD54A0                 CALL OPENIN1
225 A16E C333A1                 JP   FINISH
226                 ;
227 A171 CD23A1   CHARIN0:      CALL INIT           ;DISC IN CHAR
228 A174 CD5BA0                 CALL CHARIN1
229 A177 C333A1                 JP   FINISH
230                 ;
231 A17A CD23A1   DIRIN0:       CALL INIT           ;DISC IN DIRECT
232 A17D CD62A0                 CALL DIRIN1
233 A180 C333A1                 JP   FINISH
234                 ;
235 A183 CD23A1   OPENOUT0:     CALL INIT           ;DISC OUT OPEN
236 A186 CD69A0                 CALL OPENOUT1
237 A189 C333A1                 JP   FINISH
238                 ;
239 A18C CD23A1   CHAROUT0:     CALL INIT           ;DISC OUT CHAR
240 A18F CD70A0                 CALL CHAROUT1
241 A192 C333A1                 JP   FINISH
242                 ;
243 A195 CD23A1   DIROUT0:      CALL INIT           ;DISC OUT DIRECT
244 A198 CD77A0                 CALL DIROUT1
245 A19B C333A1                 JP   FINISH
246                 ;
247 A19E 0D0A4F6B OK:           DB   0DH,0AH,'Ok',0DH,0
247 A1A2 0D0A
248                 ;
```

```
249                ;
250                ;DISCERROR-ROUTINE
251                ;----------------
252                ;
253 A1A4 FE02      DISCERROR:  CP   2
254 A1A6 C0                   RET  NZ
255 A1A7 DD6E00               LD   L,(IX+0)   ;STRINGDESCR.HOLEN
256 A1AA DD6601               LD   H,(IX+1)
257 A1AD 2B                   DEC  HL
258 A1AE 7E                   LD   A,(HL)
259 A1AF FE02                 CP   2                ; STRING?
260 A1B1 C0                   RET  NZ
261 A1B2 23                   INC  HL
262 A1B3 E5                   PUSH HL
263 A1B4 2AE3A1               LD   HL,(ERRPOINTER) ;PUFFERZEIGER
264 A1B7 01F9A4               LD   BC,ERRBUFFER   ;-PUFFERANFANG
265 A1BA 37                   SCF                 ;=STRINGLAENGE
266 A1BB 3F                   CCF
267 A1BC ED42                 SBC  HL,BC
268 A1BE 2B                   DEC  HL           ;CR/LF AUSLASSEN
269 A1BF 2B                   DEC  HL            ;(-4 ZEICHEN)
270 A1C0 2B                   DEC  HL
271 A1C1 2B                   DEC  HL
272 A1C2 7D                   LD   A,L
273 A1C3 E1                   POP  HL
274 A1C4 77                   LD   (HL),A
275 A1C5 01FBA4               LD   BC,ERRBUFFER+2 ;STRINGDESCR.
276 A1C8 23                   INC  HL           ;AUF ERORPUFFER
277 A1C9 71                   LD   (HL),C        ;SETZEN
278 A1CA 23                   INC  HL
279 A1CB 70                   LD   (HL),B
280 A1CC DD6E02               LD   L,(IX+2)      ;ERRORNR.
281 A1CF DD6603               LD   H,(IX+3)      ;HOLEN UND
282 A1D2 2B                   DEC  HL            ;UEBERTRAGEN
283 A1D3 7E                   LD   A,(HL)
284 A1D4 FE01                 CP   1             ;INTEGER ?
285 A1D6 C0                   RET  NZ
286 A1D7 23                   INC  HL
287 A1D8 3AE2A1               LD   A,(ERRORNR)
288 A1DB 77                   LD   (HL),A
289 A1DC 23                   INC  HL
290 A1DD 3E00                 LD   A,0
291 A1DF 77                   LD   (HL),A
```

```
292 A1E0 C9                 RET
293              ;
294 A1E1 00      FLAG:      DB   0
295 A1E2 00      ERRORNR:   DB   0
296 A1E3 0000    ERRPOINTER: DB  0,0
297              ;
298              ;
299              ;DRUCKER ZUSCHALTEN
300              ;------------------
301              ;
302 A1E5 CD2EBD  PRON:      CALL PBUSY
303 A1E8 D8                 RET  C
304 A1E9 3E01               LD   A,1
305 A1EB 3252A0             LD   (PRINTFLAG),A
306 A1EE C9                 RET
307              ;
308              ;
309              ;DRUCKER ABSCHALTEN
310              ;------------------
311              ;
312 A1EF 3E00    PROFF:     LD   A,0
313 A1F1 3252A0             LD   (PRINTFLAG),A
314 A1F4 C9                 RET
315              ;
316              ;
317              ;DRUCKER EINGESCHALTET?
318              ;----------------------
319              ;
320 A1F5 FE01    PRBUSY:    CP   1
321 A1F7 C0                 RET  NZ
322 A1F8 DD6E00             LD   L,(IX+0)
323 A1FB DD6601             LD   H,(IX+1)
324 A1FE 2B                 DEC  HL
325 A1FF 7E                 LD   A,(HL)
326 A200 FE01               CP   1                ;INTEGER?
327 A202 C0                 RET  NZ
328 A203 23                 INC  HL
329 A204 CD2EBD             CALL PBUSY
330 A207 3008               JR   NC,BEREIT
331 A209 3EFF               LD   A,0FFH
332 A20B 3253A0             LD   (BUSYFLAG),A
333 A20E C316A2             JP   VAR
334 A211 3E00    BEREIT:    LD   A,0
```

```
335 A213 3253A0              LD   (BUSYFLAG),A
336 A216 3A53A0   VAR:       LD   A,(BUSYFLAG)
337 A219 77                  LD   (HL),A
338 A21A 23                  INC  HL
339 A21B 3E00                LD   A,0
340 A21D 77                  LD   (HL),A
341 A21E C9                  RET
342               ;
343               ;
344               DREAD:     EQU  0D392H
345               DBASIS:    EQU  0BE7DH
346               ;
347 A21F 00000000 BYTENR:    DB   0,0,0,0,0
347 A223 00
348 A224 0000     RECNR:     DB   0,0
349 A226 00       RECBYTE:   DB   0
350 A227 000000   STRING:    DB   0,0,0
351               ;
352 A22A C3C8A2   RECREAD0:  JP   RECREAD
353 A22D C306A3   RECWRITE0: JP   RECWRITE
354               ;
355               ;PARAMETER HOLEN
356 A230 FE02     PARAM:     CP   2          ;2 PARAMETER?
357 A232 C2ADA2              JP   NZ,ERR     ;NEIN,FERTIG
358 A235 DD6E02              LD   L,(IX+2)   ;FORTLAUFENDE
359 A238 DD6603              LD   H,(IX+3)   ;BYTENUMMER HOLEN
360 A23B 2B                  DEC  HL         ;UND IN PUFFER
361 A23C 7E                  LD   A,(HL)     ;ABLEGEN
362 A23D FE04                CP   4          ;REAL?
363 A23F C2ADA2              JP   NZ,ERR
364 A242 23                  INC  HL
365 A243 111FA2              LD   DE,BYTENR
366 A246 010500              LD   BC,5
367 A249 EDB0                LDIR
368 A24B DD6E00              LD   L,(IX+0)  ;STRINGDESCRIPTOR
369 A24E DD6601              LD   H,(IX+1)  ;HOLEN U.SPEICHERN
370 A251 2B                  DEC  HL
371 A252 7E                  LD   A,(HL)
372 A253 FE02                CP   2                ;STRING?
373 A255 C2ADA2              JP   NZ,ERR
374 A258 23                  INC  HL
375 A259 1127A2              LD   DE,STRING
376 A25C 010300              LD   BC,3
```

```
377 A25F EDB0              LDIR
378 A261 3A23A2            LD   A,(BYTENR+4)   ;UMRECHNUNG
379 A264 FE81              CP   081H           ;BYTENR. IN
380 A266 300D              JR   NC,NORMAL      ;RECORDNR.
381 A268 210000            LD   HL,0           ;UND RESTBYTES
382 A26B 2224A2            LD   (RECNR),HL
383 A26E 3E00              LD   A,0
384 A270 3226A2            LD   (RECBYTE),A
385 A273 37                SCF
386 A274 C9                RET
387 A275 FE98     NORMAL:  CP   098H
388 A277 D2ADA2            JP   NC,ERR
389 A27A 2122A2            LD   HL,BYTENR+3
390 A27D CBFE              SET  7,(HL)
391 A27F 3A23A2   SCHLEIFE: LD   A,(BYTENR+4)
392 A282 FE97              CP   97H
393 A284 280F              JR   Z,UEBER
394 A286 2123A2            LD   HL,BYTENR+4
395 A289 34                INC  (HL)
396 A28A 2B                DEC  HL
397 A28B CB3E              SRL  (HL)
398 A28D 2B                DEC  HL
399 A28E CB1E              RR   (HL)
400 A290 2B                DEC  HL
401 A291 CB1E              RR   (HL)
402 A293 18EA              JR   SCHLEIFE
403 A295 2A21A2   UEBER:   LD   HL,(BYTENR+2)
404 A298 2224A2            LD   (RECNR),HL
405 A29B 3A20A2            LD   A,(BYTENR+1)
406 A29E CB3F              SRL  A
407 A2A0 3226A2            LD   (RECBYTE),A
408 A2A3 3A27A2            LD   A,(STRING)
409 A2A6 FE00              CP   0
410 A2A8 CAADA2            JP   Z,ERR
411 A2AB 37                SCF
412 A2AC C9                RET
413 A2AD 37       ERR:     SCF                 ;WENN FEHLER
414 A2AE 3F                CCF                 ;CARRY LOESCHEN
415 A2AF C9                RET
416               ;
417 A2B0 2A24A2   NEXT:    LD   HL,(RECNR)     ;REC-NR.+1
418 A2B3 23                INC  HL
419 A2B4 2224A2            LD   (RECNR),HL
```

```
420 A2B7 112900   OFFSET:     LD   DE,29H           ;29H+ BASIS
421 A2BA 2A7DBE               LD   HL,(DBASIS)      ;(A700H)
422 A2BD 19                   ADD  HL,DE            ;=DOS-REC.-
423 A2BE 3A24A2               LD   A,(RECNR)        ;REGISTER
424 A2C1 77                   LD   (HL),A
425 A2C2 23                   INC  HL
426 A2C3 3A25A2               LD   A,(RECNR+1)
427 A2C6 77                   LD   (HL),A
428 A2C7 C9                   RET
429              ;
430              ;
431              ;DATENSATZ LESEN
432              ;---------------
433              ;
434 A2C8 CD30A2  RECREAD:     CALL PARAM      ;PARAM. HOLEN
435 A2CB D0                   RET  NC
436 A2CC CDD5A2               CALL LESEN      ;3 RECORDS LESEN
437 A2CF CDF1A2               CALL TRANSFER   ;UEBERTRAG VORBER.
438 A2D2 EDB0                 LDIR            ;IN STRING UEBER-
439 A2D4 C9                   RET             ;TRAGEN
440 A2D5 CDB7A2  LESEN:       CALL OFFSET     ;3 RECORDS IN
441 A2D8 2179A3               LD   HL,RECBUF0 ;RECORDPUFFER
442 A2DB CD53A3               CALL RECREAD1   ;LESEN
443 A2DE CDB0A2               CALL NEXT
444 A2E1 21F9A3               LD   HL,RECBUF1
445 A2E4 CD53A3               CALL RECREAD1
446 A2E7 CDB0A2               CALL NEXT
447 A2EA 2179A4               LD   HL,RECBUF2
448 A2ED CD53A3               CALL RECREAD1
449 A2F0 C9                   RET
450              ;
451 A2F1 3A26A2  TRANSFER:    LD   A,(RECBYTE) ;UEBERTRAG
452 A2F4 5F                   LD   E,A         ;IN/AUS STRING
453 A2F5 1600                 LD   D,0         ;VORBEREITEN
454 A2F7 2179A3               LD   HL,RECBUF0
455 A2FA 19                   ADD  HL,DE
456 A2FB ED5B28A2             LD   DE,(STRING+1)
457 A2FF 3A27A2               LD   A,(STRING)
458 A302 4F                   LD   C,A
459 A303 0600                 LD   B,0
460 A305 C9                   RET
461              ;
462              ;
```

```
463                ;DATENSATZ SCHREIBEN
464                ;------------------
465                ;3* HINTEREINANDER RECORD
466                ;LESEN, AKTUALISIEREN
467                ;UND WIEDER SCHREIBEN
468                ;
469 A306 CD30A2     RECWRITE:   CALL PARAM
470 A309 D0                     RET  NC
471 A30A CDB7A2                CALL OFFSET            ;1.RECORD
472 A30D 2179A3                LD   HL,RECBUF0
473 A310 CD53A3                CALL RECREAD1
474 A313 CDF1A2                CALL TRANSFER
475 A316 EB                    EX   DE,HL
476 A317 EDB0                  LDIR
477 A319 CDB7A2                CALL OFFSET
478 A31C 2179A3                LD   HL,RECBUF0
479 A31F CD5AA3                CALL RECWRITE1
480 A322 CDB0A2                CALL NEXT              ;2.RECORD
481 A325 21F9A3                LD   HL,RECBUF1
482 A328 CD53A3                CALL RECREAD1
483 A32B CDF1A2                CALL TRANSFER
484 A32E EB                    EX   DE,HL
485 A32F EDB0                  LDIR
486 A331 CDB7A2                CALL OFFSET
487 A334 21F9A3                LD   HL,RECBUF1
488 A337 CD5AA3                CALL RECWRITE1
489 A33A CDB0A2                CALL NEXT              ;3.RECORD
490 A33D 2179A4                LD   HL,RECBUF2
491 A340 CD53A3                CALL RECREAD1
492 A343 CDF1A2                CALL TRANSFER
493 A346 EB                    EX   DE,HL
494 A347 EDB0                  LDIR
495 A349 CDB7A2                CALL OFFSET
496 A34C 2179A4                LD   HL,RECBUF2
497 A34F CD5AA3                CALL RECWRITE1
498 A352 C9                    RET
499                ;
500 A353 DF       RECREAD1:    RST  18H           ;RECORD LESEN
501 A354 57A3                  DW   RECREAD2
502 A356 C9                    RET
503 A357 92D3     RECREAD2:    DW   DREAD
504 A359 07                    DB   7
505                ;
```

```
506 A35A DF       RECWRITE1:  RST  18H          ;RECORD SCHREIBEN
507 A35B 5EA3                 DW   RECWRITE2
508 A35D C9                   RET
509 A35E 61A3     RECWRITE2:  DW   DWRITE
510 A360 07                   DB   7
511 A361 E5       DWRITE:     PUSH HL
512 A362 D5                   PUSH DE
513 A363 C5                   PUSH BC
514 A364 E5                   PUSH HL
515 A365 110800               LD   DE,8H
516 A368 CD98CA               CALL 0CA98H
517 A36B CD10D4               CALL 0D410H
518 A36E D2A9D3               JP   NC,0D3A9H
519 A371 EB                   EX   DE,HL
520 A372 E3                   EX   (SP),HL
521 A373 CDF3D9               CALL 0D9F3H
522 A376 C3A6D3               JP   0D3A6H
523               ;
524               RECBUF0:    DS   80H              ;PUFFER 0
525               RECBUF1:    DS   80H              ;PUFFER 1
526               RECBUF2:    DS   80H              ;PUFFER 2
527               ERRBUFFER:  DS   50H          ;FUER FEHLERMELD.
528 A549 00                   DB   0
529                           END
```

7.6 Adressenverwaltungsprogramm (relativ)

Das in Kapitel 6 besprochene sequentielle Adressenverwaltungsprogramm wurde nachfolgend für die relative Dateiverwaltung umgeschrieben. Dabei befinden sich die Datensätze nicht mehr im Speicher, sondern in einer relativen Datei auf Diskette.

Jedesmal, wenn ein Datensatz bearbeitet werden soll, wird er entweder neu eingegeben oder einzeln aus der relativen Datei gelesen und nach der Bearbeitung wieder auf Diskette geschrieben. Doch woher weiß nun das Programm, in welchem Eintrag sich welche Adresse befindet?

Wird die erste eingegebene Adresse als 1. Datensatz, die zweite als 2. Datensatz usw. abgelegt, ist die Sache noch relativ einfach, denn wir müssen dann nur die betreffende Datensatznummer aufrufen. Schwieriger wird es, wenn wir die Adressen alphabetisch sortiert oder einzelne Adressen aus der Datei entfernt haben.

Des Rätsels Lösung heißt indexsequentielle Dateiverwaltung. Indexsequentiell bedeutet, daß von jedem Datensatz die Nummer und ein spezieller Index zusätzlich in einer sequentiellen Datei abgelegt wird, die zusammen mit der relativen Datei auf Diskette geschrieben wird. Dieser Index dient dann als Suchkriterium zum Auffinden eines Datensatzes.

Für unser Adressenverwaltungsprogramm wurde der Name als Index gewählt. Bevor nun Datensätze angelegt oder bearbeitet werden können, wird zunächst die Indexdatei in den Arbeitsspeicher des Computers geladen. Sie bleibt dort solange, wie wir mit der relativen Datei arbeiten und wird im Anschluß daran wieder auf Diskette geschrieben.

Wenn wir nun einen Datensatz anlegen, "weiß" die Indexdatei bereits, welche Einträge schon belegt und welche noch frei sind. Sie sucht den nächstverfügbaren Eintrag heraus und legt dort den Datensatz ab. Gleichzeitig erhält auch die Indexdatei einen neuen Eintrag, der den Nachnamen als Index und die dazugehörige Datensatznummer enthält.

Suchen wir nun umgekehrt den Datensatz zum Namen Müller, wird zunächst die Indexdatei durchsucht, um festzustellen, ob der Datensatz in der Datei enthalten ist. Ist dies der Fall, steht gleichzeitig die Nummer des fortlaufenden Eintrags zur Verfügung, die der Datensatz in der Datei belegt, so daß er leicht aufgerufen werden kann.

Bei der sequentiellen Dateiverwaltung benötigten wir keine zusätzliche Indexdatei, da ja alle Datensätze zu jeder Zeit vollständig im Speicher abgelegt waren und wir sie nur nach dem richtigen Namen durchsuchen mußten. Sicher wäre es auch möglich, analog zur sequentiellen Dateiverwaltung, sämtliche Records von Diskette einzulesen, um einen bestimmten Namen ausfindig zu machen. Dies ist jedoch eine sehr mühselige und zeitraubende Angelegenheit, da das Einlesen von Diskette viel mehr Zeit in Anspruch nimmt, als ein Feld im Arbeitsspeicher zu durchsuchen.

Die Indexdatei benötigt außerdem viel weniger Speicherplatz als die gesamte Datei, was zusätzlich Speicherplatzproblemen entgegenwirkt. Wenn wir nun unsere Adressendatei alphabetisch sortieren, geschieht dies letztlich nur mit der Indexdatei, wobei die Reihenfolge, in der die Datensätze in der relativen Datei angelegt sind, beibehalten wird. Ähnliches passiert auch beim Löschen eines Datensatzes, wobei lediglich sein Eintrag in der Indexdatei gelöscht wird. Der Eintrag in der relativen Datei dagegen bleibt unverändert und wird nur als nicht belegt gekennzeichnet.

Bevor wir uns nun detailliert mit unserem Programm befassen, werfen wir noch einen Blick auf das Menü, das auch hier nach dem Starten des Programms auf dem Bildschirm erscheint:

```
        Adressenverwaltung
        ==================

Adressen schreiben/fortschreiben                1

Adressen aendern/loeschen                       2

Adressen sortieren                              3

Relative Datei oeffnen                          4

Relative Datei anlegen                          5

Adressen ausdrucken                             6

Adressen auf Etiketten ausdrucken               7


E n d e                                         e

Bitte waehlen Sie
```

Vielleicht ist es Ihnen aufgefallen, daß die meisten Menüpunkte mit denen im sequentiellen Adressenverwaltungsprogramm identisch sind und daß sich nur die Punkte 4 und 5 unterscheiden.

Wir wissen bereits, daß wir eine relative Datei anlegen müssen, bevor wir darin Eintragungen vornehmen können. Genau dies bewirkt der Menüpunkt 5. Zusätzlich wird hier noch die Indexdatei erzeugt und auf Diskette geschrieben. Wenn Sie sich jetzt das Disketten-Inhaltsverzeichnis ansehen, enthält es eine relative und eine sequentielle Datei, die beide neu angelegt sind, aber noch keine Einträge haben. Damit es keine Verwechslungen mit den Dateinamen gibt, erhält die relative Datei den von Ihnen gewählten Namen und die sequentielle Indexdatei den gleichen Namen mit der Extension ".IND". Wenn Sie nun z.B. eine Datei unter dem Namen "KUNDEN" anlegen, bekommt die relative Datei den Namen "KUNDEN. ", während er für die Indexdatei "KUNDEN.IND" lautet.

Bevor Sie nun Datensätze anlegen oder bearbeiten können, müssen Sie auf jeden Fall Menüpunkt 4 anwählen, der die relative Datei öffnet. Öffnen ist hier nicht nur unbedingt im Sinne der OPENIN- bzw. OPENOUT-Anweisung zu verstehen, sondern bedeutet vielmehr eine Initialisierung der Adressendatei. Dabei wird die Indexdatei in den Speicher geladen und gleichzeitig festgestellt, welche Einträge belegt und welche noch frei sind.

Erst jetzt können Sie daran gehen, Adressen einzugeben oder anderweitig zu bearbeiten. Die restlichen Menüpunkte haben für den Bediener die gleiche Funktion wie bei der sequentiellen Adressenverwaltung. Lediglich bei Beendigung des Programms muß hier unbedingt die E-Taste gedrückt werden, damit die Indexdatei auf Diskette geschrieben werden kann. Wenn Sie dies vergessen und den Computer einfach ausschalten oder das Programm beispielsweise mit der ESC-Taste abbrechen, kann es unter Umständen zu erheblichen Fehlern beim späteren Wiedereinlesen der Datensätze kommen.

Nachfolgend das Programmlisting:

```
1000 REM Adressenverwaltungsprogramm (relativ)
1010 REM ================================
1020 REM
1030 REM
1040 IF PEEK(&A000)=1 AND PEEK(&A001)=12 THEN 1090
1050 MEMORY &9FFF
1060 LOAD "erwbin",&A000
1070 CALL &A000
```

```
1080 OPENOUT "dummy" :MEMORY HIMEM-1:CLOSEOUT
1090 ma=200:REM max.Anzahl der Datensaetze
1100 MODE 1
1110 k=0:REM Datensaetze im Speicher
1120 ln=15:REM Zeichen Laenge Namen
1130 lv=15:REM Zeichen Laenge Vornamen
1140 ls=20:REM Zeichen Laenge Strasse
1150 lo=21:REM Zeichen Laenge PLZ/Ort
1160 la=1:REM Zeichen Laenge Anredecode
1170 fl=0:REM Flag ob Datei im Speicher
1180 rl=74:REM Recordlaenge+2 (cr/lf)
1190 cl$=CHR$(8):REM Cursor nach links
1200 cr$=CHR$(9):REM Cursor nach rechts
1210 cd$=CHR$(10):REM Cursor nach unten
1220 cu$=CHR$(11):REM Cursor nach oben
1230 ds%=0:ds$="":REM fuer Fehlerabfrage
1240 pr%=0:REM fuer Druckerabfrage
1250 l=ln+lv+ls+lo+la+1
1260 g$="**************************"
1270 n$="                                "
1280 DIM n$(ma)
1290 DIM ar$(5)
1300 ar$(0)=""
1310 ar$(1)="Herrn"
1320 ar$(2)="Frau"
1330 ar$(3)="Fraeulein"
1340 ar$(4)="Herrn und Frau"
1350 ar$(5)="Firma"
1360 :
1370 :
1380 :
1390 REM Menue
1400 CLS
1410 PRINT TAB(10);"Adressenverwaltung"
1420 PRINT TAB(10);"=================="
1430 PRINT
1440 PRINT
1450 PRINT "Adressen schreiben/fortschreiben      1"
1460 PRINT
1470 PRINT "Adressen aendern/loeschen             2"
1480 PRINT
1490 PRINT "Adressen sortieren                    3"
1500 PRINT
```

```
1510 PRINT "Relative Datei oeffnen                    4"
1520 PRINT
1530 PRINT "Relative Datei anlegen                    5"
1540 PRINT
1550 PRINT "Adressen ausdrucken                       6"
1560 PRINT
1570 PRINT "Adressen auf Etiketten ausdrucken         7"
1580 PRINT
1590 PRINT
1600 PRINT "E n d e                                   e"
1610 PRINT
1620 PRINT
1630 PRINT "Bitte waehlen Sie"
1640 m$=INKEY$:IF m$="" THEN 1640
1650 IF m$="e" THEN 5220
1660 m=VAL(m$)
1670 IF m<1 OR m>7 THEN 1640
1680 ON m GOSUB 1730 ,2260,3120,3420,3750,4210,4590
1690 GOTO 1390
1700 :
1710 :
1720 :
1730 REM Adressen schreiben/fortschreiben
1740 CLS
1750 PRINT
1760 PRINT
1770 PRINT "    Adressen schreiben/fortschreiben    "
1780 PRINT
1790 PRINT
1800 IF dn$="" THEN PRINT "Bitte Datei oeffnen":GOTO 2160
1810 ON ERROR GOTO 5340
1820 OPENIN dn$:|DISCERROR,@ds%,@ds$:CLOSEIN
1830 ON ERROR GOTO 0
1840 IF ds$<>"Ok" THEN PRINT ds$:GOTO 2160
1850 k=k+1
1860 PRINT k
1870 PRINT
1880 IF k>ma THEN PRINT "Datei voll":GOTO 2130
1890 PRINT "Anrede (0-5)";:INPUT a$
1900 a=VAL(a$)
1910 IF a<0 OR a>5 OR LEN(a$)<>1 THEN PRINT cu$;:GOTO 1890
1920 PRINT
1930 PRINT "Name        ";:INPUT na$
```

```
1940 PRINT
1950 na$=LEFT$(na$+n$,ln)
1960 PRINT "Vorname      ";:INPUT nv$
1970 PRINT
1980 nv$=LEFT$(nv$+n$,lv)
1990 PRINT "Strasse      ";:INPUT ns$
2000 PRINT
2010 ns$=LEFT$(ns$+n$,ls)
2020 PRINT "PLZ, Ort     ";:INPUT no$
2030 no$=LEFT$(no$+n$,lo)
2040 r$=na$+nv$+ns$+no$+RIGHT$(STR$(a),1)+CHR$(13)+CHR$(10)
2050 d=VAL(MID$(n$(k),ln+1,10))
2060 po=rl*d
2070 OPENIN dn$
2080 |RECWRITE,@po,@r$
2090 CLOSEIN
2100 IF ds$<>"Ok" THEN PRINT ds$:GOTO 2160
2110 fl=1
2120 n$(k)=na$+STR$(d)
2130 PRINT
2140 PRINT
2150 PRINT "Naechste Eingabe - Enter-Taste druecken"
2160 PRINT
2170 PRINT "Zurueck ins Menue - Leertaste druecken"
2180 m$=INKEY$:IF m$="" THEN 2180
2190 IF m$=CHR$(13) AND dn$<>"" THEN 1730
2200 IF m$=" " THEN CLOSEIN:GOTO 2220
2210 GOTO 2180
2220 RETURN
2230 :
2240 :
2250 :
2260 REM Adressen aendern/loeschen
2270 CLS
2280 PRINT
2290 PRINT
2300 PRINT "        Adressen aendern/loeschen"
2310 PRINT
2320 PRINT
2330 IF dn$="" THEN PRINT "Bitte Datei oeffnen":GOTO 3020
2340 ON ERROR GOTO 5340
2350 OPENIN dn$:|DISCERROR,@ds%,@ds$:CLOSEIN
2360 ON ERROR GOTO 0
```

```
2370 IF ds$<>"Ok"THEN PRINT ds$:GOTO 3020
2380 PRINT "Name         ";:INPUT na$
2390 na$=LEFT$(na$+n$,ln)
2400 i=1
2410 WHILE i<=k
2420 IF LEFT$(n$(i),ln)=na$ THEN 2480
2430 i=i+1
2440 WEND
2450 PRINT
2460 PRINT "nicht gefunden"
2470 GOTO 2990
2480 d=VAL(MID$(n$(i),ln+1,10))
2490 r$=SPACE$(rl-2)
2500 po=rl*d
2510 OPENIN dn$
2520 |RECREAD,@po,@r$
2530 CLOSEIN
2540 na$=LEFT$(r$,ln)
2550 nv$=MID$(r$,ln+1,lv)
2560 ns$=MID$(r$,ln+lv+1,ls)
2570 no$=MID$(r$,ln+lv+ls+1,lo)
2580 a$=RIGHT$(r$,1)
2590 PRINT i
2600 PRINT
2610 PRINT "Anrede (0-5)  ";a$
2620 PRINT
2630 PRINT "Name          ";na$
2640 PRINT
2650 PRINT "Vorname       ";nv$
2660 PRINT
2670 PRINT "Strasse       ";ns$
2680 PRINT
2690 PRINT "PLZ, Ort      ";no$
2700 FOR q=1 TO 8:PRINT cu$;:NEXT q
2710 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2720 IF ww$<>""THEN a$=ww$
2730 a=VAL(a$)
2740 IF a$="l" THEN GOSUB 4940:GOTO 2990
2750 IF a<0 OR a>5 OR LEN(a$)<>1 THEN PRINT cu$;:GOTO 2710
2760 PRINT
2770 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2780 IF ww$<>"" THEN na$=ww$
2790 PRINT
```

```
2800 na$=LEFT$(na$+n$,ln)
2810 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2820 IF ww$<>"" THEN nv$=ww$
2830 PRINT
2840 nv$=LEFT$(nv$+n$,lv)
2850 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2860 IF ww$<>"" THEN ns$=ww$
2870 PRINT
2880 ns$=LEFT$(ns$+n$,ls)
2890 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2900 IF ww$<>"" THEN no$=ww$
2910 no$=LEFT$(no$+n$,lo)
2920 r$=na$+nv$+ns$+no$+RIGHT$(STR$(a),1)+CHR$(13)+CHR$(10)
2930 d=VAL(MID$(n$(i),ln+1,10))
2940 OPENIN dn$
2950 po=rl*d
2960 |RECWRITE,@po,@r$
2970 CLOSEIN
2980 n$(i)=na$+STR$(d)
2990 PRINT
3000 PRINT
3010 PRINT "Naechste Eingabe - Enter-Taste druecken"
3020 PRINT
3030 PRINT "Zurueck ins Menue - Leertaste druecken"
3040 m$=INKEY$:IF m$="" THEN 3040
3050 IF m$=CHR$(13) AND dn$<>"" THEN 2260
3060 IF m$=" "THEN 3080
3070 GOTO 3040
3080 RETURN
3090 :
3100 :
3110 :
3120 REM Adressen sortieren
3130 CLS
3140 PRINT
3150 PRINT
3160 PRINT "          Adressen sortieren          "
3170 PRINT
3180 PRINT
3190 g=INT(k/2)
3200 WHILE g>0
3210 FOR i=g+1 TO k
3220 j=i-g
```

```
3230 WHILE j>0
3240 IF n$(j)<=n$(j+g) THEN j=0:GOTO 3290
3250 a$=n$(j+g)
3260 n$(j+g)=n$(j)
3270 n$(j)=a$
3280 j=j-g
3290 WEND
3300 NEXT
3310 g=INT(g/2)
3320 WEND
3330 PRINT
3340 PRINT
3350 PRINT "          Zurueck ins Menue"
3360 PRINT "       beliebige Taste druecken"
3370 m$=INKEY$:IF m$=""THEN 3370
3380 RETURN
3390 :
3400 :
3410 :
3420 REM Relative Datei oeffnen
3430 CLS
3440 PRINT
3450 PRINT
3460 PRINT "      Relative Datei oeffnen       "
3470 PRINT
3480 PRINT
3490 INPUT "Dateiname";dn$
3500 PRINT
3510 i=0
3520 ON ERROR GOTO 5340
3530 OPENIN dn$+".ind"
3540 |DISCERROR,@ds%,@ds$
3550 ON ERROR GOTO 0
3560 IF ds$<>"Ok"THEN PRINT ds$:GOTO 3650
3570 fl=0
3580 INPUT#9,k
3590 PRINT k
3600 IF k>0 THEN fl=1
3610 IF i=ma THEN PRINT "Datei voll":GOTO 3650
3620 i=i+1
3630 INPUT#9,n$(i)
3640 IF EOF=0 THEN 3610
3650 CLOSEIN
```

```
3660 PRINT
3670 PRINT
3680 PRINT "          Zurueck ins Menue"
3690 PRINT "       beliebige Taste druecken"
3700 m$=INKEY$:IF m$="" THEN 3700
3710 RETURN
3720 :
3730 :
3740 :
3750 REM Relative Datei anlegen
3760 CLS
3770 PRINT
3780 PRINT
3790 PRINT "       Relative Datei anlegen          "
3800 PRINT
3810 PRINT
3820 INPUT "Dateiname";dn$
3830 PRINT
3840 ON ERROR GOTO 5340
3850 OPENIN dn$:|DISCERROR,@ds%,@ds$:CLOSEIN
3860 ON ERROR GOTO 0
3870 IF ds$="Ok"THEN PRINT "bestehende ";:ELSE 3940
3880 PRINT "Datei ueberschreiben ? (j/n)"
3890 PRINT
3900 m$=INKEY$:IF m$="" THEN 3900
3910 IF m$="j" THEN 3940
3920 IF m$="n" THEN 4120
3930 GOTO 3900
3940 r$=SPACE$(rl-2)+CHR$(13)+CHR$(10)
3950 OPENOUT dn$
3960 FOR i=0 TO ma
3970 PRINT i;cu$
3980 PRINT#9,r$;
3990 NEXT i
4000 PRINT#9,SPACE$(128);SPACE$(128);
4010 CLOSEOUT
4020 OPENOUT dn$+".ind"
4030 PRINT#9,0
4040 FOR i=1 TO ma
4050 PRINT i;cu$
4060 n$(i)=LEFT$(g$,ln)+STR$(i)
4070 PRINT#9,n$(i)
4080 NEXT i
```

```
4090 CLOSEOUT
4100 fl=1
4110 k=0
4120 PRINT
4130 PRINT
4140 PRINT "          Zurueck ins Menue"
4150 PRINT "       beliebige Taste druecken"
4160 m$=INKEY$:IF m$="" THEN 4160
4170 RETURN
4180 :
4190 :
4200 :
4210 REM Adressendatei ausdrucken
4220 GOSUB 5090
4230 CLS
4240 PRINT
4250 PRINT
4260 PRINT "       Adressen ausdrucken            "
4270 PRINT
4280 PRINT
4290 OPENIN dn$
4300 FOR i=1 TO k
4310 d=VAL(MID$(n$(i),ln+1,10))
4320 po=rl*d
4330 r$=SPACE$(rl-2)
4340 |RECREAD,@po,@r$
4350 PRINT#8,LEFT$(r$,ln);
4360 PRINT#8," ";
4370 PRINT#8,MID$(r$,ln+1,lv);
4380 PRINT#8," ";
4390 PRINT#8,MID$(r$,ln+lv+1,ls);
4400 PRINT#8," ";
4410 PRINT#8,MID$(r$,ln+lv+ls+1,lo);
4420 PRINT#8," ";
4430 PRINT#8,RIGHT$(r$,1)
4440 IF i-INT(i/64)*64<>0 THEN 4480
4450 FOR j=1 TO 8
4460 PRINT#8
4470 NEXT j
4480 NEXT i
4490 CLOSEIN
4500 PRINT
4510 PRINT
```

```
4520 PRINT "          Zurueck ins Menue"
4530 PRINT "       beliebige Taste druecken"
4540 m$=INKEY$:IF m$="" THEN 4540
4550 RETURN
4560 :
4570 :
4580 :
4590 REM Adressen auf Etiketten drucken
4600 GOSUB 5090
4610 CLS
4620 PRINT
4630 PRINT
4640 PRINT " Adressen auf Etiketten ausdrucken"
4650 PRINT
4660 PRINT
4670 OPENIN dn$
4680 FOR i=1 TO k
4690 d=VAL(MID$(n$(i),ln+1,10))
4700 r$=SPACE$(rl-2)
4710 po=rl*d
4720 |RECREAD,@po,@r$
4730 PRINT#8
4740 PRINT#8
4750 a=VAL(RIGHT$(r$,1))
4760 PRINT#8,ar$(a)
4770 PRINT#8,LEFT$(r$,ln+lv)
4780 PRINT#8,MID$(r$,ln+lv+1,ls)
4790 PRINT#8
4800 PRINT#8,MID$(r$,ln+lv+ls+1,lo)
4810 PRINT#8
4820 PRINT#8
4830 NEXT i
4840 CLOSEIN
4850 PRINT
4860 PRINT
4870 PRINT "          Zurueck ins Menue"
4880 PRINT "       beliebige Taste druecken"
4890 m$=INKEY$:IF m$=""THEN 4890
4900 RETURN
4910 :
4920 :
4930 :
4940 REM Datensatz loeschen
```

```
4950 z$=LEFT$(g$,ln)+MID$(n$(i),ln+1,10)
4960 FOR j=i+1 TO k
4970 n$(j-1)=n$(j)
4980 NEXT j
4990 n$(k)=z$
5000 k=k-1
5010 FOR q=1 TO 8:PRINT cd$;: NEXT q
5020 PRINT "geloescht"
5030 PRINT cu$;cu$;
5040 IF k=0 THEN fl=0
5050 RETURN
5060 :
5070 :
5080 :
5090 REM Abfrage, ob Drucker eingeschaltet
5100 |PRBUSY,@pr%
5110 IF pr%=0 THEN RETURN
5120 CLS
5130 PRINT
5140 PRINT
5150 PRINT "      Bitte Drucker einschalten"
5160 PRINT "    und beliebige Taste druecken"
5170 m$=INKEY$:IF m$="" THEN 5170
5180 GOTO 5100
5190 :
5200 :
5210 :
5220 REM Programm beenden
5230 IF fl=0 THEN 5300
5240 OPENOUT dn$+".ind"
5250 PRINT#9,k
5260 FOR i=1 TO ma
5270 PRINT#9,n$(i)
5280 NEXT
5290 CLOSEOUT
5300 END
5310 :
5320 :
5330 :
5340 RESUME NEXT
```

Nachfolgend geben wir noch einige Anmerkungen zum Programmlisting, wobei wir uns nur auf diejenigen Punkte konzentrieren wollen, die sich von der sequentiellen Adressenverwaltung unterscheiden.

Zu Beginn des Programms fragt Zeile 1040 die ersten beiden Speicherzellen des Maschinenprogramms für die Befehlserweiterung ab. Ist sie noch nicht geladen, wird die Speicherobergrenze auf 9FFFH herabgesetzt, die Erweiterung geladen und initialisiert und schließlich der Ein-/Ausgabepuffer angelegt.

In Zeile 1180 legt die Variable rl die Länge der Datensätze fest, wobei bereits zwei zusätzliche Zeichen für CR und LF enthalten sind. Die maximale Anzahl der Datensätze wurde hier ebenfalls mit 200 angesetzt (Variable ma) und kann bei Bedarf noch erhöht werden.

Kommen wir nun zum Unterprogramm, das eine relative Datei anlegt (Zeile 3750). Zunächst wird geprüft, ob bereits eine Datei unter dem gleichen Namen dn$ angelegt ist. Dies geschieht durch kurz aufeinanderfolgendes Öffnen und Schließen der Datei und anschließender Fehlerabfrage. Lautet die Fehlermeldung "Ok", existiert eine solche Datei und der Bediener wird gefragt, ob diese Datei überschrieben werden soll. Anderenfalls entsteht die Fehlermeldung "(Filename) not found", worauf die Datei für ma Einträge mit der Länge rl angelegt wird (Zeile 3950 ff). Anschließend folgt ab Zeile 4020 die Anlage der sequentiellen Indexdatei, ebenfalls mit ma Elementen. Diese Indexdatei wird außerdem im Feld n$(i) angelegt, in das sie auch später zur Bearbeitung der Datei eingelesen wird und wo sie während der gesamten Bearbeitungsdauer verbleibt.

Da die Indexdatei noch leer ist, wird als erster Wert die Zahl Null hineingeschrieben, gefolgt von ma Elementen, die aus 15 Sternchen und der fortlaufenden Datensatznummer bestehen. Hier die ersten Elemente der Indexdatei:

```
 0
*************** 1
*************** 2
*************** 3
usw.
```

Die Sternchen kennzeichnen einen leeren oder als gelöscht gekennzeichneten Eintrag. Wird der Eintrag später belegt, erscheint oben statt der Null die Anzahl der belegten Einträge, die ansonsten in der Variable k abgelegt ist und statt der Sternchen erscheint der betreffende Name als Suchindex. Da

Namen nur in Strings gespeichert werden können, wandeln wir auch die entsprechenden Eintragsnummern mit der STR$-Anweisung in Strings um, damit wir Name und Nummer in einen String packen können. Wird die Nummer später benötigt, wird sie aus dem String herausgelöst und mit der VAL-Anweisung wieder in einen numerischen Wert umgewandelt.

Kommen wir nun zum "Öffnen" einer relativen Datei ab Zeile 3420. Hier wird lediglich die Indexdatei eingelesen, wobei vorausgesetzt wird, daß, wenn sich eine Indexdatei auf der Diskette befindet, auch die zugehörige relative Datei vorhanden sein muß. Wird die Indexdatei nicht gefunden, erscheint die AMSDOS-Fehlermeldung "(Filename) not found".

Der erste aus der Indexdatei eingelesene Wert gibt die Anzahl der bereits belegten Einträge wieder und wird sogleich der Variablen k zugeordnet. Die restlichen Elemente werden im Feld n$(i) abgelegt.

Im Unterprogramm "Adressen schreiben/fortschreiben" ab Zeile 1730 wird dann mit dem k+1ten Datensatz fortgefahren. Ist k gleich Null, also die Datei leer, wird zur Eingabe des ersten Datensatzes aufgefordert. Nach beendeter Eingabe wird in Zeile 2070 der Datensatz in die relative Datei geschrieben und anschließend in Zeile 2120 die Indexdatei aktualisiert, indem der Name darin eingetragen wird.

Im Menüpunkt "Aendern/Loeschen" wird der gesuchte Name zunächst in der Indexdatei gesucht und dann mit Hilfe der darin enthaltenen Nummer aus der relativen Datei gelesen. Nach Durchführung der Änderung erhält der Datensatz wieder den gleichen Eintrag in der Datei.

Zum Löschen wird das Unterprogramm ab Zeile 4940 aufgerufen. Dabei wird der Name in der Indexdatei mit Sternchen überschrieben, wobei allerdings die fortlaufende Nummer des Eintrags in der relativen Datei erhalten bleibt. Ist der zu löschende Datensatz an der Stelle i in der Indexdatei eingetragen, werden alle Elemente von i+1 bis k um eine Stelle nach unten aufgerückt und das Element des gelöschten Datensatzes an der Stelle k eingefügt. Da sich durch das Löschen die Anzahl der Datensätze um eins reduziert, erniedrigt sich abschließend der Wert von k ebenfalls um eins. Der gelöschte Eintrag steht nun wieder für einen neuen Datensatz zur Verfügung.

Zum Sortieren der Indexeinträge dient das Unterprogramm ab Zeile 3120, wobei wieder das Sortierverfahren nach Shell zur Anwendung kommt. Meistens verschiebt sich erst durch das Sortieren - aber auch durch das Löschen - die Reihenfolge der Einträge im Verhältnis zu derjenigen im Indexverzeichnis.

Zu den Routinen, die die Adressen auf dem Drucker listen bzw. Etiketten ausdrucken, gibt es nur soviel zu sagen, daß dies in der Reihenfolge der Elemente in der Indexdatei geschieht. Dabei wird der jeweilige Datensatz eingelesen und anschließend ausgedruckt.

Wird bei Abschluß der Bearbeitung im Menü die Taste E gedrückt, verzweigt das Programm nach Zeile 5220, wo die Indexdatei in ihrem aktualisierten Zustand wieder auf die Diskette geschrieben wird und daraufhin das Programm endet.

8 Die zweite Speicherbank

Gegenüber seinen Vorgängermodellen besitzt der CPC 6128 eine zweite Speicherbank mit zusätzlichen 64K RAM. Wir wissen bereits, daß RAM ein flüchtiger Schreib-/Lesespeicher ist, in dem auch Ihre BASIC-Programme oder der Bildschirminhalt abgelegt werden.

Da das "Herz" Ihres Computers, der Z80-Mikroprozessor, nur 64K Speicher adressieren kann, bilden diese 64K eine Speicherbank. Es gibt jedoch Mittel und Wege, daß der Prozessor auch auf mehrere Bänke zugreifen kann. Dazu muß er zwischen den einzelnen Bänken hin- und herschalten, was man in der Fachsprache als Bankswitching oder Banking bezeichnet.

Nun führt Ihr CPC 6128 auch ohne Nutzung dieser zweiten Speicherbank bereits ständig Bankswitchings durch, ohne daß Sie hiervon etwas bemerken. Dies geschieht bei der Abarbeitung eines BASIC-Programms genauso wie bei jeder Bildschirmausgabe oder der Ansteuerung der Floppy. Während die "normalen" 64K RAM überwiegend als BASIC-Speicher und Video-RAM genutzt werden, erfolgt jeder Zugriff auf das BASIC-, Betriebssystem- oder Floppy-ROM ebenfalls durch Bankswitching.

Sämtliche BASIC-Anweisungen und Programme, mit denen wir uns bisher beschäftigt haben, greifen nicht auf die zweite Speicherbank zu, sondern nutzen nur die erste Bank. Bis hierher unterscheidet sich der CPC 6128 kaum von seinem Vorgängermodell CPC 464 und noch weniger vom CPC 664. Sie werden sich jetzt zu Recht die Frage stellen, warum Sie sich dann überhaupt einen CPC 6128 angeschafft und dafür einige hundert Mark mehr bezahlt haben.

Über das eingebaute BASIC können Sie zwar nicht auf die zweite Speicherbank zugreifen, aber auf der mitgelieferten Systemdiskette (Seite 1) befindet sich das Programm BANKMAN.BAS. Es enthält einen sogenannten Bankmanager, den Sie, ähnlich wie die Befehlserweiterungen in Kapitel 7, in Ihre Programme einbinden können.

Dieses Bankmanager-Programm stellt Ihnen einige zusätzliche Befehle zur Verfügung, mit denen Sie auf die zweite Speicherbank Ihres CPC 6128 zugreifen können. Mit diesen Befehlserweiterungen wollen wir uns im weiteren Verlauf dieses Kapitels näher beschäftigen. Zuvor müssen Sie jedoch den Bankmanager erst einmal von Diskette (Seite 1) laden und initialisieren, was mit

```
RUN "BANKMAN"
```

geschieht. Nun können Sie alle Befehlserweiterungen nutzen, die Ihnen der Bankmanager bietet.

8.1 Bildschirmspeicher

In Kapitel 6 haben wir ein Beispiel betrachtet, das den Inhalt des gesamten Video-RAMs auf Diskette speichert und wieder lädt, was für die dauerhafte Aufbewahrung von Graphiken durchaus sinnvoll ist.

Häufig besteht aber der Wunsch, Text- oder Graphikbildschirme nur kurzfristig zwischenzuspeichern, um sie bei Bedarf schnell wieder ins eigentliche Video-RAM kopieren zu können. Da ein Bildschirminhalt 16K Speicher benötigt, bietet sich hierfür nichts besseres als die zusätzliche Speicherbank an. Mit ihren 64K kann sie sogar vier Bildschirminhalte gleichzeitig speichern.

Der Bankmanager enthält zwei Anweisungen, mit denen Sie fünf verschiedene Bildschirmspeicher adressieren können. Dabei ist der Bildschirmspeicher 1 das normale Video-RAM zwischen C000H und FFFFH in der ersten Speicherbank. Die Bildschirmspeicher 2 bis 5 belegen jeweils 16K in der zweiten Speicherbank.

Die Anweisung

|SCREENSWAP, Bildschirm-Nr., Bildschirm-Nr.

vertauscht den Inhalt zweier Bildschirmspeicher. Somit tauscht z.B.

|SCREENSWAP, 1, 3

den Inhalt des normalen Video-RAMs (Speicher 1) mit dem von Bildschirmspeicher 3 in der zweiten Speicherbank aus.

Darüber hinaus gibt es noch die Anweisung

|SCREENCOPY, zu Bildschirm-Nr., von Bildschirm-Nr.

die die Inhalte der einzelnen Bildschirmspeicher kopiert. Beispielsweise überträgt

|SCREENCOPY, 1, 5

den Inhalt des Bildschirmspeichers 5 (zweite Speicherbank) in das Video-RAM.

Mit den hier gezeigten Anweisungen können leicht verschiedene Bildschirminhalte kreiert, zwischengespeichert und wieder hervorgeholt werden. Probieren Sie es einmal aus.

8.2 RAM-Disc

Außer zur Speicherung von Bildschirminhalten können die zusätzlichen 64K auch als RAM-Disc genutzt werden. Dies bedeutet, daß Sie die zweite Speicherbank im gewissen Sinne als Floppy-Ersatz nutzen können, indem Sie Daten nicht auf Diskette, sondern statt dessen in die zweite Speicherbank schreiben. Natürlich handelt es sich hierbei nur um eine vorübergehende Ablage, da ja die Daten beim Ausschalten des Computers verloren gehen. Dafür können Sie aber auf einzelne Datensätze wesentlich schneller zugreifen, als wenn sie auf Diskette gespeichert wären.

Der Direktzugriff kann mit der RAM-Disc sehr leicht realisiert werden, sofern die Speicherbank in einzelne Datensätze konstanter Länge eingeteilt ist. Die Anweisung

|BANKOPEN, Datensatzlänge

legt die Länge der einzelnen Datensätze fest.

|BANKWRITE, @ Kontrollcode, String, Datensatz-Nr.

schreibt einen Datensatz aus einem String in die RAM-Disc und

|BANKREAD, @ Kontrollcode, String, Datensatz-Nr.

liest einen Datensatz aus der RAM-Disc und ordnet seinen Inhalt einer Stringvariablen zu.

|BANKFIND, @ Kontrollcode, String, von Datensatz-Nr.,
bis Datensatz-Nr.

durchsucht die Datensätze zwischen den angegebenen Nummern nach einem vorgegebenen String.

Verlief die jeweilige Operation fehlerfrei, enthält der Kontrollcode die Datensatznummer, auf die zugegriffen wurde. Im Fehlerfall enthält er jedoch negative Werte, die folgende Bedeutung haben:

-1 Datensatznummer überschreitet 64K-Grenze
-2 Fehler im Bankswitching (sollte niemals auftreten)
-3 gesuchter String wurde nicht gefunden (|BANKFIND)

Hier noch einige Anwendungsbeispiele:

|BANKOPEN, 30

legt eine Datensatzlänge von 30 Zeichen fest.

|BANKWRITE, @e%, a$, 25

schreibt den Inhalt von a$ in den 25. Datensatz. Bei erfolgreicher Operation enthält die Variable e% die Datensatznummer 25, sonst eine der oben angegebenen Fehlernummern.

```
|BANKREAD, @e%, b$, 100
```

liest den Inhalt des 100. Datensatzes und legt ihn in der Stringvariablen b$ ab. Verlief die Operation erfolgreich, enthält e% die Datensatznummer 100, anderenfalls eine der obigen negativen Fehlernummern.

```
|BANKFIND, @e%, "CPC 6128", 10, 50
```

durchsucht die Datensätze 10 bis 50 nach dem String "CPC 6128". Wurde der Ausdruck gefunden, erscheint in e% die Nummer des Datensatzes, in dem er auftrat. Wurde er dagegen nicht gefunden, erscheint die Zahl -3.

Beachten Sie, daß der Kontrollcode in Form einer Integervariablen angegeben sein muß (hier e%). Diese Variable muß jedoch - am besten gleich am Programmanfang - definiert werden, z.B. in Form von e%=0.

Mit der RAM-Disc läßt sich leicht eine relative Dateiverwaltung realisieren, ähnlich wie wir sie in Kapitel 7 kennengelernt haben. Allerdings erfolgt der Direktzugriff hier nicht auf die Diskette, sondern statt dessen auf die RAM-Disc.

Nachfolgend ein kleines Programmbeispiel zur Demonstration, das 300 Datensätze auf die RAM-Disc schreibt und anschließend in umgekehrter Reihenfolge wieder liest. Denken Sie aber daran, vor Ausführung des Programms den Bankmanager zu laden und zu initialisieren (s.o.).

```
10 a$=SPACE$(14)
20 e%=0
30 |BANKOPEN,14
40 FOR i=0 TO 300
50 c$=RIGHT$("   "+STR$(i),3)+". Datensatz"
60 |BANKWRITE,@e%,c$,i
70 NEXT i
80 :
90 FOR i=300 TO 0 STEP -1
100 |BANKREAD,@e%,a$,i
110 PRINT a$
120 NEXT i
```

Zunächst wird in Zeile 10 ein Leerstring mit 14 Zeichen Länge angelegt, der später zum Lesen der Datensätze aus der RAM-Disc benötigt wird. Anschließend wird die Integervariable e% als Kontrollcode definiert. Zeile 30 legt eine Datensatzlänge von 30 Zeichen fest, worauf in der nachfolgenden FOR...NEXT-Schleife 300 Datensätze mit einer konstanten Länge von 14 Zeichen in die RAM-Disc geschrieben werden.

In der zweiten FOR...NEXT-Schleife ab Zeile 90 werden nun sämtliche Datensätze in umgekehrter Reihenfolge wieder gelesen und jeweils in der Stringvariablen a$ abgelegt, die wir in Zeile 10 definiert haben. Zeile 110 gibt dann jeweils den String a$ aus, so daß bei einwandfreier Funktion des Programms folgendes auf dem Bildschirm erscheinen muß:

300. Datensatz
299. Datensatz
298. Datensatz
.
.
.
1. Datensatz
0. Datensatz

Achten Sie darauf, daß die Stringlängen zum Schreiben und Lesen unbedingt mit der in |BANKOPEN angegebenen Datensatzlänge übereinstimmen, da es sonst zu unliebsamen Überraschungen kommen kann.

In unserem Beispiel beträgt die Datensatzlänge exakt 14 Zeichen, die der String c$ in Zeile 50 erhält, bevor er in die RAM-Disc geschrieben wird. Die gleiche Länge muß auch der String a$ zum Lesen erhalten, indem er z.B. mit 14 Leerzeichen angelegt wird.

8.3 Adressenverwaltungsprogramm (relativ mit RAM-Disc)

Nachfolgend finden Sie das gleiche Adressenverwaltungsprogramm wie in Kapitel 7, das hier jedoch nicht die Diskette, sondern die RAM-Disc als relativen Datenspeicher benutzt.

Das Anlegen der relativen Datei geschieht dabei genauso, wie es in Kapitel 7 beschrieben wurde, d.h., es wird zunächst die vorgegebene Anzahl von Datensätzen in Form von Leerstrings auf Diskette geschrieben und anschließend die Indexdatei angelegt.

Wenn Sie den Menüpunkt "Relative Datei öffnen" aufrufen, werden sowohl die eigentliche Datendatei als auch die Indexdatei sequentiell von Diskette gelesen. Die Datensätze, die sich in der Datendatei befinden, werden dabei in der zweiten Speicherbank abgelegt, so daß sie dort mit Hilfe der Direktzugriffsbefehle des Bankmanagers bearbeitet werden können.

Haben Sie die Datei fertig geschrieben bzw. bearbeitet, drücken Sie auch hier die E-Taste zur Beendigung des Programms. Dabei wird aber nicht nur die Indexdatei, sondern auch die Datendatei wieder auf Diskette geschrieben, da sie sonst durch Abschalten des Computers verlorenginge. Vergessen Sie nicht, vor Ausführung des Programms den Bankmanager zu laden und zu initialisieren! Nachfolgend das vollständige Programmlisting:

```
1000 REM Adressenverwaltungsprogramm (relativ mit RAM-Disc)
1010 REM=============================================
1020 REM
1030 REM
1040 OPENOUT "dummy" :MEMORY HIMEM-1:CLOSEOUT
1050 ma=200:REM max.Anzahl der Datensaetze
1060 MODE 1
1070 k=0:REM Datensaetze im Speicher
1080 ln=15:REM Zeichen Laenge Namen
1090 lv=15:REM Zeichen Laenge Vornamen
1100 ls=20:REM Zeichen Laenge Strasse
1110 lo=21:REM Zeichen Laenge PLZ/Ort
1120 la=1:REM Zeichen Laenge Anredecode
1130 fl=0:REM Flag ob Datei im Speicher
1140 rl=74:REM Recordlaenge+2 (cr/lf)
1150 cl$=CHR$(8):REM Cursor nach links
1160 cr$=CHR$(9):REM Cursor nach rechts
1170 cd$=CHR$(10):REM Cursor nach unten
```

```
1180 cu$=CHR$(11):REM Cursor nach oben
1190 e%=0:REM fuer Fehlerabfrage RAM-Disc
1200 l=ln+lv+ls+lo+la+1
1210 g$="************************"
1220 n$="                                "
1230 DIM n$(ma)
1240 DIM ar$(5)
1250 ar$(0)=""
1260 ar$(1)="Herrn"
1270 ar$(2)="Frau"
1280 ar$(3)="Fraeulein"
1290 ar$(4)="Herrn und Frau"
1300 ar$(5)="Firma"
1310 :
1320 :
1330 :
1340 REM Menue
1350 CLS
1360 PRINT TAB(10);"Adressenverwaltung"
1370 PRINT TAB(10);"=================="
1380 PRINT
1390 PRINT
1400 PRINT "Adressen schreiben/fortschreiben      1"
1410 PRINT
1420 PRINT "Adressen aendern/loeschen             2"
1430 PRINT
1440 PRINT "Adressen sortieren                    3"
1450 PRINT
1460 PRINT "Relative Datei oeffnen                4"
1470 PRINT
1480 PRINT "Relative Datei anlegen                5"
1490 PRINT
1500 PRINT "Adressen ausdrucken                   6"
1510 PRINT
1520 PRINT "Adressen auf Etiketten ausdrucken     7"
1530 PRINT
1540 PRINT
1550 PRINT "E n d e                               e"
1560 PRINT
1570 PRINT
1580 PRINT "Bitte waehlen Sie"
1590 m$=INKEY$:IF m$="" THEN 1590
1600 IF m$="e" THEN 4740
```

```
1610 m=VAL(m$)
1620 IF m<1 OR m>7 THEN 1590
1630 ON m GOSUB 1680 ,2140,2920,3220,3570,3920,4270
1640 GOTO 1340
1650 :
1660 :
1670 :
1680 REM Adressen schreiben/fortschreiben
1690 CLS
1700 PRINT
1710 PRINT
1720 PRINT "    Adressen schreiben/fortschreiben    "
1730 PRINT
1740 PRINT
1750 IF dn$="" THEN PRINT "Bitte Datei oeffnen":GOTO 2040
1760 k=k+1
1770 PRINT k
1780 PRINT
1790 IF k>ma THEN PRINT "Datei voll":GOTO 2010
1800 PRINT "Anrede (0-5)";:INPUT a$
1810 a=VAL(a$)
1820 IF a<0 OR a>5 OR LEN(a$)<>1 THEN PRINT cu$;:GOTO 1800
1830 PRINT
1840 PRINT "Name          ";:INPUT na$
1850 PRINT
1860 na$=LEFT$(na$+n$,ln)
1870 PRINT "Vorname      ";:INPUT nv$
1880 PRINT
1890 nv$=LEFT$(nv$+n$,lv)
1900 PRINT "Strasse        ";tINPUT ns$
1910 PRINT
1920 ns$=LEFT$(ns$+n$,ls)
1930 PRINT "PLZ, Ort      ";:INPUT no$
1940 no$=LEFT$(no$+n$,lo)
1950 r$=na$+nv$+ns$+no$+RIGHT$(STR$(a),1)+CHR$(13)+CHR$(10)
1960 d=VAL(MID$(n$(k),ln+1,10))
1970 |BANKOPEN,rl
1980 |BANKWRITE,@e%,r$,d
1990 fl=1
2000 n$(k)=na$+STR$(d)
2010 PRINT
2020 PRINT
2030 PRINT "Naechste Eingabe - Enter-Taste druecken"
```

```
2040 PRINT
2050 PRINT "Zurueck ins Menue - Leertaste druecken"
2060 m$=INKEY$:IF m$="" THEN 2060
2070 IF m$=CHR$(13) AND dn$<>"" THEN 1680
2080 IF m$=" " THEN CLOSEIN:GOTO 2100
2090 GOTO 2060
2100 RETURN
2110 :
2120 :
2130 :
2140 REM Adressen aendern/loeschen
2150 CLS
2160 PRINT
2170 PRINT
2180 PRINT "       Adressen aendern/loeschen"
2190 PRINT
2200 PRINT
2210 IF dn$="" THEN PRINT "Bitte Datei oeffnen":GOTO 2820
2220 PRINT "Name        ";:INPUT na$
2230 na$=LEFT$(na$+n$,ln)
2240 i=1
2250 WHILE i<=k
2260 IF LEFT$(n$(i),ln)=na$ THEN 2320
2270 i=i+1
2280 WEND
2290 PRINT
2300 PRINT "nicht gefunden"
2310 GOTO 2790
2320 d=VAL(MID$(n$(i),ln+1,10))
2330 r$=SPACE$(rl-2)
2340 |BANKOPEN,rl
2350 |BANKREAD,@e%,r$,d
2360 na$=LEFT$(r$,ln)
2370 nv$=MID$(r$,ln+1,lv)
2380 ns$=MID$(r$,ln+lv+1,ls)
2390 no$=MID$(r$,ln+lv+ls+1,lo)
2400 a$=RIGHT$(r$,1)
2410 PRINT i
2420 PRINT
2430 PRINT "Anrede (0-5)  ";a$
2440 PRINT
2450 PRINT "Name          ";na$
2460 PRINT
```

```
2470 PRINT "Vorname       ";nv$
2480 PRINT
2490 PRINT "Strasse       ";ns$
2500 PRINT
2510 PRINT "PLZ, Ort      ";no$
2520 FOR q=1 TO 8:PRINT cu$;:NEXT q
2530 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2540 IF ww$<>""THEN a$=ww$
2550 a=VAL(a$)
2560 IF a$="l" THEN GOSUB 4590:GOTO 2790
2570 IF a<0 OR a>5 OR LEN(a$)<>1 THEN PRINT cu$;:GOTO 2530
2580 PRINT
2590 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2600 IF ww$<>"" THEN na$=ww$
2610 PRINT
2620 na$=LEFT$(na$+n$,ln)
2630 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2640 IF ww$<>"" THEN nv$=ww$
2650 PRINT
2660 nv$=LEFT$(nv$+n$,lv)
2670 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2680 IF ww$<>"" THEN ns$=ww$
2690 PRINT
2700 ns$=LEFT$(ns$+n$,ls)
2710 FOR q=1 TO 12:PRINT cr$;:NEXT q:INPUT ww$
2720 IF ww$<>"" THEN no$=ww$
2730 no$=LEFT$(no$+n$,lo)
2740 r$=na$+nv$+ns$+no$+RIGHT$(STR$(a),1)+CHR$(13)+CHR$(10)
2750 d=VAL(MID$(n$(i),ln+1,10))
2760 |BANKOPEN,rl
2770 |BANKWRITE,@e%,r$,d
2780 n$(i)=na$+STR$(d)
2790 PRINT
2800 PRINT
2810 PRINT "Naechste Eingabe - Enter-Taste druecken"
2820 PRINT
2830 PRINT "Zurueck ins Menue - Leertaste druecken"
2840 m$=INKEY$:IF m$="" THEN 2840
2850 IF m$=CHR$(13) AND dn$<>"" THEN 2140
2860 IF m$=" "THEN 2880
2870 GOTO 2840
2880 RETURN
2890 :
```

```
2900 :
2910 :
2920 REM Adressen sortieren
2930 CLS
2940 PRINT
2950 PRINT
2960 PRINT "          Adressen sortieren       "
2970 PRINT
2980 PRINT
2990 g=INT(k/2)
3000 WHILE g>0
3010 FOR i=g+1 TO k
3020 j=i-g
3030 WHILE j>0
3040 IF n$(j)<=n$(j+g) THEN j=0:GOTO 3090
3050 a$=n$(j+g)
3060 n$(j+g)=n$(j)
3070 n$(j)=a$
3080 j=j-g
3090 WEND
3100 NEXT
3110 g=INT(g/2)
3120 WEND
3130 PRINT
3140 PRINT
3150 PRINT "          Zurueck ins Menue"
3160 PRINT "       beliebige Taste druecken"
3170 m$=INKEY$:IF m$=""THEN 3170
3180 RETURN
3190 :
3200 :
3210 :
3220 REM Relative Datei oeffnen
3230 CLS
3240 PRINT
3250 PRINT
3260 PRINT "      Relative Datei oeffnen       "
3270 PRINT
3280 PRINT
3290 INPUT "Dateiname";dn$
3300 PRINT
3310 i=0
3320 OPENIN dn$+".ind"
```

```
3330 fl=0
3340 INPUT#9,k
3350 PRINT k
3360 IF k>0 THEN fl=1
3370 IF i=ma THEN PRINT "Datei voll":GOTO 3410
3380 i=i+1
3390 INPUT#9,n$(i)
3400 IF EOF=0 THEN 3370
3410 CLOSEIN
3420 OPENIN dn$
3430 |BANKOPEN,rl
3440 FOR i=0 TO ma
3450 INPUT#9,r$
3460 |BANKWRITE,@e%,r$,i
3470 NEXT i
3480 PRINT
3490 PRINT
3500 PRINT "          Zurueck ins Menue"
3510 PRINT "       beliebige Taste druecken"
3520 m$=INKEY$:IF m$="" THEN 3520
3530 RETURN
3540 :
3550 :
3560 :
3570 REM Relative Datei anlegen
3580 CLS
3590 PRINT
3600 PRINT
3610 PRINT "       Relative Datei anlegen          "
3620 PRINT
3630 PRINT
3640 INPUT "Dateiname";dn$
3650 PRINT
3660 r$=SPACE$(rl-2)+CHR$(13)+CHR$(10)
3670 OPENOUT dn$
3680 FOR i=0 TO ma
3690 PRINT i;cu$
3700 PRINT#9,r$;
3710 NEXT i
3720 CLOSEOUT
3730 OPENOUT dn$+".ind"
3740 PRINT#9,0
3750 FOR i=1 TO ma
```

```
3760 PRINT i;cu$
3770 n$(i)=LEFT$(g$,ln)+STR$(i)
3780 PRINT#9,n$(i)
3790 NEXT i
3800 CLOSEOUT
3810 fl=1
3820 k=0
3830 PRINT
3840 PRINT
3850 PRINT "          Zurueck ins Menue"
3860 PRINT "       beliebige Taste druecken"
3870 m$=INKEY$:IF m$="" THEN 3870
3880 RETURN
3890 :
3900 :
3910 :
3920 REM Adressendatei ausdrucken
3930 CLS
3940 PRINT
3950 PRINT
3960 PRINT "       Adressen ausdrucken          "
3970 PRINT
3980 PRINT
3990 |BANKOPEN,rl
4000 FOR i=1 TO k
4010 d=VAL(MID$(n$(i),ln+1,10))
4020 r$=SPACE$(rl-2)
4030 |BANKREAD,@e%,r$,d
4040 PRINT#8,LEFT$(r$,ln);
4050 PRINT#8," ";
4060 PRINT#8,MID$(r$,ln+1,lv);
4070 PRINT#8," ";
4080 PRINT#8,MID$(r$,ln+lv+1,ls);
4090 PRINT#8," ";
4100 PRINT#8,MID$(r$,ln+lv+ls+1,lo);
4110 PRINT#8," ";
4120 PRINT#8,RIGHT$(r$,1)
4130 IF i-INT(i/64)*64<>0 THEN 4170
4140 FOR j=1 TO 8
4150 PRINT#8
4160 NEXT j
4170 NEXT i
4180 PRINT
```

```
4190 PRINT
4200 PRINT "          Zurueck ins Menue"
4210 PRINT "      beliebige Taste druecken"
4220 m$=INKEY$:IF m$="" THEN 4220
4230 RETURN
4240 :
4250 :
4260 :
4270 REM Adressen auf Etiketten drucken
4280 CLS
4290 PRINT
4300 PRINT
4310 PRINT " Adressen auf Etiketten ausdrucken"
4320 PRINT
4330 PRINT
4340 |BANKOPEN,rl
4350 FOR i=1 TO k
4360 d=VAL(MID$(n$(i),ln+1,10))
4370 r$=SPACE$(rl-2)
4380 |BANKREAD,@e%,r$,d
4390 PRINT#8
4400 PRINT#8
4410 a=VAL(RIGHT$(r$,1))
4420 PRINT#8,ar$(a)
4430 PRINT#8,LEFT$(r$,ln+lv)
4440 PRINT#8,MID$(r$,ln+lv+1,ls)
4450 PRINT#8
4460 PRINT#8,MID$(r$,ln+lv+ls+1,lo)
4470 PRINT#8
4480 PRINT#8
4490 NEXT i
4500 PRINT
4510 PRINT
4520 PRINT "          Zurueck ins Menue"
4530 PRINT "      beliebige Taste druecken"
4540 m$=INKEY$:IF m$=""THEN 4540
4550 RETURN
4560 :
4570 :
4580 :
4590 REM Datensatz loeschen
4600 z$=LEFT$(g$,ln)+MID$(n$(i),ln+1,10)
4610 FOR j=i+1 TO k
```

```
4620 n$(j-1)=n$(j)
4630 NEXT j
4640 n$(k)=z$
4650 k=k-1
4660 FOR q=1 TO 8:PRINT cd$;: NEXT q
4670 PRINT "geloescht"
4680 PRINT cu$;cu$;
4690 IF k=0 THEN fl=0
4700 RETURN
4710 :
4720 :
4730 :
4740 REM Programm beenden
4750 IF fl=0 THEN 4820
4760 OPENOUT dn$+".ind"
4770 PRINT#9,k
4780 FOR i=1 TO ma
4790 PRINT#9,n$(i)
4800 NEXT
4810 CLOSEOUT
4820 OPENOUT dn$
4830 r$=SPACE$(rl-2)
4840 |BANKOPEN,rl
4850 FOR i=0 TO ma
4860 |BANKREAD,@e%,r$,i
4870 PRINT#9,r$
4880 NEXT i
4890 CLOSEOUT
4900 END
```

9 Farbe, Graphik und Sound

Mit dem CPC 6128 stehen Ihnen viele Möglichkeiten offen, Ihre kreativen Fähigkeiten einzusetzen. Graphik und Sound sind ein umfangreiches Gebiet, wobei Ihrer Phantasie keine Grenzen gesetzt sind.

In diesem Kapitel befassen wir uns mit den Grundlagen der Farbgraphik und der Tonerzeugung. Das CPC-BASIC enthält viele Anweisungen, mit denen Sie Ihre Kunstwerke auf einfache Weise programmieren können.

Zunächst werden wir uns mit den verschiedenen Farben beschäftigen, die Ihr CPC 6128 erzeugen kann. Dann lernen Sie anhand des deutschen Zeichensatzes, selbstdefinierte Sonderzeichen zu generieren. Es folgt eine Einführung in die hochauflösende Graphik, der sich die Erzeugung von Tönen und Melodien anschließt.

9.1 Bildschirmmodi und Farben

Aus den früheren Kapiteln ist Ihnen bereits bekannt, daß Sie auf Ihrem CPC 6128 drei verschiedene Bildschirmmodi einstellen können und zwar

MODE 0 mit 20 Zeichen pro Zeile,
MODE 1 mit 40 Zeichen pro Zeile und
MODE 2 mit 80 Zeichen pro Zeile

In allen drei Modi ist jedoch die Größe des Video-RAMs mit 16384 Bytes (16K) konstant. Lediglich die Aufteilung in Bildschirmpunkte (Pixels) und die dazugehörigen Farbspeicher unterscheiden sich.

Nun setzt sich ein Bildschirmzeichen aus 8x8=64 Punkten zusammen, die man in der Fachsprache als Pixel bezeichnet. Im MODE 0 können Sie somit 20x8=160, in MODE 1 40x8=320 und in MODE 3 80x8=640 Pixel in der Horizontalen darstellen. Dabei fällt allerdings die Darstellungsbreite der Pixel in MODE 0 am breitesten und in MODE 2 am schmalsten aus.

Zur Darstellung eines Pixels wird jeweils ein Bit im Video-RAM benötigt. Ist das Bit gesetzt, erscheint der Bildschirmpunkt hell, bei nicht gesetztem Bit dagegen dunkel. Diese hell/dunkel-Bezeichnung trifft strenggenommen aber nur auf eine reine Schwarweiß-Darstellung zu.

Da Ihr CPC 6128 aber auch für Farbgraphik geeignet ist, könnte man schwarz und weiß ebenfalls als Farbe bezeichnen, was auch tatsächlich der Fall ist (s. Farbentabelle).

Im MODE 2 besteht der Bildschirm aus insgesamt 640x200=128000 Bildpunkten bzw. Bits, die im Video-RAM benötigt werden. Da 8 Bits ein Byte ergeben, benötigen wir in dieser Darstellungsart 128000/8=16000 Bytes, was knapp 16K (16384 Bytes), also dem gesamten Video-RAM entspricht. Deshalb können wir im MODE 2 nur zwei Farben darstellen, wobei natürlich auch andere Farben außer Schwarz und Weiß in Frage kommen.

Betrachten wir jetzt MODE 1 mit 320x200=64000 Pixel, die den gesamten Bildschirm füllen. Dies entspricht exakt dem halben Speicher (8K), der für MODE 2 benötigt wird. Was geschieht aber nun mit den restlichen 8K?

Die Antwort ist einfach: Sie haben jetzt für jedes Pixel nicht mehr nur ein, sondern zwei Bits. Diese beiden Bits können aber nicht nur zwei Zuständen, wie z.B. hell und dunkel, sondern vier verschiedene Zustände annehmen. Deshalb können Sie im MODE 1 auch vier Farben gleichzeitig verwenden.

Sie sehen also, daß die zweiten 8K keinesfalls ungenutzt sind, wenn Sie im MODE 1 arbeiten. Die größere Farbenvielfalt wird allerdings durch eine geringere Auflösung erkauft.

Das gleiche gilt analog auch für MODE 0 mit einer Bildschirmauflösung von 160x200=32000 Punkten, die 4K entsprechen. Für jeden Bildschirmpunkt stehen hier sogar 4 Bits zur Verfügung, die 16 verschiedene Zustände (Farben) annehmen können. Deshalb können Sie im MODE 0 auch mit der größten Farbenpracht arbeiten, was allerdings die geringste Auflösung zur Folge hat.

Der CPC 6128 kann insgesamt 27 verschiedene Farben erzeugen, die in der nachfolgenden Tabelle aufgeführt sind:

Nr.	Farbe	Nr.	Farbe	Nr.	Farbe
0	Schwarz	9	Grün	18	Hellgrün
1	Blau	10	Blaugrün	19	Seegrün
2	Hellblau	11	Himmelblau	20	Helles Blaugrün
3	Rot	12	Gelb	21	Limonengrün
4	Magenta	13	Weiß	22	Pastellgrün
5	Hellviolett	14	Pastellblau	23	Pastellblaugrün
6	Hellrot	15	Orange	24	Hellgelb
7	Purpur	16	Rosa	25	Pastellgelb
8	Helles Magenta	17	Pastellmagenta	26	Leuchtendweiß

Bei der Farbgraphik können Sie dem Rahmen, dem Hintergrund und dem Vordergrund (Schrift) verschiedene Farben zuordnen. Am einfachsten ist es immer noch bei der Rahmenfarbe, bei der Sie hinter die Anweisung BORDER lediglich die gewünschte Farbnummer setzen. Wünschen Sie beispielsweise eine rote Rahmenfarbe (Farbe Nr. 3), so geben Sie ein:

```
BORDER 3
```

Etwas schwieriger wird es allerdings bei der Hintergrundfarbe (PAPER) und der Vordergrund- oder Schriftfarbe (PEN). Hier genügt die Angabe der gewünschten Farbe nicht.

Wir wissen bereits, daß wir im MODE 0 sechszehn, im MODE 1 vier und im MODE 2 zwei Farben gleichzeitig verwenden können (Rahmenfarbe ausgenommen). Ihr CPC 6128 kann jedoch 27 verschiedene Farben erzeugen und es stellt sich nun die Frage, woher wir wissen, welche Farben jeweils zugelassen sind?

Diese Frage ist leicht zu beantworten, denn es sind 16, 4 bzw. 2 Farben Ihrer Wahl. Stellen Sie sich dazu einmal folgenden bildlichen Vergleich vor:

In jedem Bildschirmmodus besitzen Sie eine zulässige Anzahl Farbtöpfe. Demnach dürfen Sie im Modus 0 sechzehn, im Modus 1 vier und im Modus 2 zwei verschiedene Farbtöpfe benutzen. Jeden dieser Farbtöpfe dürfen Sie aber mit einer beliebigen Farbe füllen, die Sie aus den 27 verfügbaren Farben frei wählen können.

Diese "Farbtöpfe" heißen Farbregister und werden mit der INK-Anweisung "gefüllt". Der erste Parameter gibt die Nummer des Registers und der zweite Parameter die Farbnummer an.

```
INK 3,18
```

füllt beispielsweise das Farbregister Nr. 3 mit der Farbe Nr. 18 (hellgrün).

Im Modus 0 stehen Ihnen somit die Farbregister 0 bis 15, im Modus 1 die Register 0 bis 3 und im Modus 2 die Register 0 und 1 zur Verfügung.

Die folgenden beiden Anweisungen setzen jeweils die Hintergrund- und Schriftfarbe. Der angegebene Parameter bezieht sich aber nicht auf die Farbe als solche, sondern immer auf das Farbregister, dem wir ja mit der INK-Anweisung bereits eine Farbe zugeordnet haben. Mit

```
PAPER 2
```

erscheint beispielsweise der Hintergrund in der Farbe, die sich in Register Nr. 2 befindet. Durch

```
PEN 0
```

erhält die Schrift die Farbe des Registers Nr. 0.

Nach dem Einschalten hat Ihr Computer eine blaue Rahmen- und Hintergrundfarbe (Nr. 1) sowie eine hellgelbe Schriftfarbe (Nr. 24). Dabei ist die Hintergrundfarbe dem Register 0 und die Schriftfarbe dem Register 1 zugeordnet. Diese Grundeinstellung entspricht den Einzelanweisungen

```
INK 0,1
INK 1,24
```

und

```
PAPER 0
PEN 1
```

Wenn Sie jetzt

```
PEN 0 : PAPER 1 : CLS  <RETURN>
```

eingeben, vertauschen Sie Hintergrund- und Schriftfarbe. CLS löscht den Bildschirm. Er wird dabei gelb und sämtliche Schrift erscheint fortan in blau. Übrigens sollten Sie diese drei Anweisungen in eine Zeile schreiben, da Sie sonst nicht mehr sehen, was Sie eingeben, weil Hintergrund und Schrift die gleiche Farbe annehmen.

Hier nun eine Tabelle mit den vorbelegten Farbnummern für die jeweiligen Farbregister in den drei Bildschirmmodi, die Sie ohne Ausführung einer INK-Anweisung erhalten:

Farbregister-Nummer		Farbe Modus 0	Farbe Modus 1	Farbe Modus 2	
0		1	1	1	(Hintergrund)
1		24	24	24	(Schrift)
2		20	20	(1)	
3		6	6	(24)	
4		26	(1)	(1)	
5		0	(24)	(24)	
6		2	(20)	(1)	
7		8	(6)	(24)	
8		10	(1)	(1)	
9		12	(24)	(24)	
10		14	(20)	(1)	
11		16	(6)	(24)	
12		18	(1)	(1)	
13		22	(24)	(24)	
14	Blinken	1 u.24	(20)	(1)	
15	Blinken	16 u.11	(6)	(24)	

Falls Sie die hier angegebenen Werte ändern wollen, müssen Sie dies mit der INK-Anweisung tun. Die in Klammern eingeschlossenen Werte im Modus 1 und 2 wiederholen sich periodisch in Vierer- bzw. Zweierschritten. Da im Modus 2 nur zwei Farbregister zugelassen sind, enthält das fiktive Register 2 die gleiche Nummer wie Register 0 usw.

Indem die INK-Anweisung nicht nur einen, sondern zwei Farbparameter erhält, kann ein ständiges Wechseln zwischen zwei Farben erreicht werden (Blinkeffekt). Dabei können Sie sowohl die Hintergrund- als auch die Schriftfarbe blinken lassen. Hier ein Beispiel, das die Schriftfarbe Ihres Computers rot-gelb blinken läßt.

```
INK 1,6,24
```

1 entspricht der Farbregister-Nummer, 6 der Farbe Hellrot und 24 der Farbe Hellgelb, die sich beide ständig abwechseln.

Selbstverständlich können Sie auch die Rahmenfarbe blinken lassen. So läßt z.B.

```
BORDER 3,24
```

den Rahmen abwechselnd rot und hellgelb erscheinen.

9.2 Selbstdefinierte Zeichen

Bisher haben wir nur Zeichen auf dem Bildschirm ausgegeben, deren Punktraster von 8x8 Pixels im Character-ROM fest vorgegeben war. Es besteht aber auch die Möglichkeit, selbstdefinierte Zeichen auf einfache Weise auf dem CPC 6128 zu erzeugen.

Ein häufiger Anwendungsfall ist die Definition deutscher Sonderzeichen, die der CPC 6128 in seinem Standard-Zeichensatz nicht enthält. Damit wollen wir uns jetzt näher befassen.

Wir wissen bereits, daß sämtlichen Zeichen ein ASCII-Code zugeordnet ist (s. Anhang). Da der ASCII-Zeichensatz eine amerikanische Norm ist, enthält er keine deutschen Sonderzeichen. Man verwendet in Deutschland aber einen abgewandelten ASCII-Code, der einige Zeichen des Orginalcodes durch deutsche Sonderzeichen ersetzt. Die folgende Tabelle macht dies deutlich:

Zeichen	ASCII-Code	
§	64	(normal @)
Z	90	bleibt
Ä	91	(normal [ )
Ö	92	(normal \ )
Ü	93	(normal])
z	122	bleibt
ä	123	(normal {)
ö	124	(normal |)
ü	125	(normal })
ß	126	(normal ~)

Wir sehen also, daß sich der eingedeutschte ASCII-Zeichensatz direkt an die Buchstaben des Alphabets anschließt und damit einige amerikanische Zeichen ersetzt. Sie müssen deshalb u.a. auf eckige und geschweifte Klammern verzichten und das Directory mit

öDIR

laden, da der senkrechte Strich dem Buchstaben ö zum Opfer fällt.

Wenn Sie das folgende Programm ausführen, erhalten Sie nicht nur deutsche Sonderzeichen, sondern auch eine geänderte Tastaturbelegung, die derjenigen einer deutschen Schreibmaschine ähnelt. Übrigens verwendet auch der Schneider-Drucker NLQ 401 und die meisten anderen Drucker die gleichen ASCII-Codes für deutsche Sonderzeichen, sofern der deutsche Zeichensatz eingeschaltet ist.

```
100 REM Deutsche Sonderzeichen
110 SYMBOL AFTER 63
120 SYMBOL  64,&3E,&60,&7C,&66,&3E,&06,&06,&7C
130 SYMBOL  91,&66,&18,&3C,&66,&7E,&66,&66,&00
140 SYMBOL  92,&66,&3C,&66,&66,&66,&66,&3C,&00
150 SYMBOL  93,&66,&00,&66,&66,&66,&66,&3C,&00
160 SYMBOL 123,&66,&00,&3C,&06,&3E,&66,&3E,&00
170 SYMBOL 124,&66,&00,&3C,&66,&66,&66,&3C,&00
180 SYMBOL 125,&66,&00,&00,&66,&66,&66,&3E,&00
190 SYMBOL 126,&3C,&66,&66,&7C,&66,&66,&7C,&60
200 :
210 REM Tastatur umbelegen
220 KEY DEF 71,1,121,89,25
```

```
230 KEY DEF 43,1,122,90,26
240 KEY DEF 22,1,64
250 KEY DEF 24,1,94,126
260 KEY DEF 26,1,125,93
270 KEY DEF 17,1,124,92
280 KEY DEF 19,1,123,91
```

Die Anweisung in Zeile 110

```
SYMBOL AFTER 63
```

lagert sämtliche Zeichenraster, die sich auf einen höheren ASCII-Code als 63 beziehen, aus dem Charakter-ROM ins obere BASIC-RAM aus, so daß wir sie neu gestalten können.

Mit der SYMBOL-Anweisung erhält nun jedes gewünschte Zeichen ein neues Aussehen. Die erste Zeichenänderung in Zeile 120 wollen wir uns einmal genauer ansehen.

Der erste Parameter hinter SYMBOL gibt den ASCII-Code des Zeichens an. Hier ist es der Code 64, der ursprünglich dem Klammeraffen entspricht und den wir jetzt durch das Paragraphenzeichen ersetzen wollen.

Die restlichen acht Werte entsprechen dem Raster des neuen Zeichens. Sie sind in hexadezimaler Schreibweise angegeben, was aber nicht unbedingt sein muß. Hier nun das Raster des Paragraphen-Zeichens, wobei ein Sternchen (*) einem gesetzten Rasterpunkt und ein normaler Punkt einem nicht gesetzten Rasterpunkt entspricht:

```
..*****.    &3E
.**.....    &60
.*****..    &7C
.**..**.    &66
..*****.    &3E
.....**.    &06
.....**.    &06
.*****..    &7C
```

In diesem Raster ist das Paragraphenzeichen ganz deutlich zu erkennen. Wir setzen nun für jeden Punkt eine Null und für jedes Sternchen eine Eins und erhalten dadurch für jede Punktreihe eine Binärzahl, die wir dann in eine Dezimal- oder Hexadezimalzahl umformen können. Als Beispiel betrachten wir einmal die oberste Reihe:

..*****. entspricht 00111110 binär oder

```
  0 * 128
+ 0 *  64
+ 1 *  32
+ 1 *  16
+ 1 *   8
+ 1 *   4
+ 1 *   2
+ 0 *   1
```

Nach Addition dieser Faktoren erhalten wir die Zahl 62 oder &3E in hexadezimaler Schreibweise.

Nach dem Entwurf der Sonderzeichen (am besten auf kariertem Papier) wiederholen wir diesen Vorgang für jede einzelne Rasterreihe und erhalten somit die weiteren acht Parameter für die SYMBOL-Anweisung.

Die schönsten deutschen Sonderzeichen wären nur sehr umständlich zu handhaben, wenn wir nicht auch die Tastatur entsprechend umstellen würden. Mit der Anweisung

```
KEY DEF  Tastennummer,
         Wiederholung,
         ASCII-Zeichen ohne SHIFT,
         ASCII-Zeichen mit SHIFT,
         ASCII-Zeichen mit CTRL
```

können wir jede Taste auf der Tastatur neu belegen. Im Anhang befindet sich eine Skizze, aus der Sie für jede Taste den zugehörigen Tastaturcode entnehmen können. Unser Programm belegt folgende Tasten um:

```
Y und Z vertauschen
ü, Ü    auf Klammeraffen-Taste  ( @ u. | )
ö, Ö    auf Taste für eckige Klammer auf  ( [ )
ä, Ä    auf Taste für eckige Klammer zu   ( ] )
ß       auf geshiftete Pfeiltaste (anstelle von £-Zeichen)
§       auf Schrägstrich-Taste ( \ ) rechts unten
```

Wenn Sie ständig mit der deutschen Tastatur arbeiten wollen, können Sie sich für die geänderten Tasten Aufkleber herstellen.

Abschließend wollen wir uns als Beispiel die KEY DEF-Anweisung in Zeile 220 einmal genauer ansehen. Sie legt den Buchstaben Y auf die Z-Taste und lautet folgendermaßen:

```
KEY DEF 71,1,121,89,25
```

Aus der Tastaturcodetabelle im Anhang entnehmen wir, daß die Z-Taste die Nummer 71 besitzt. Somit ist 71 auch der erste Parameter der Anweisung.

Die 1 gibt an, daß die Taste eine Autorepeat-Funktion erhalten soll, wie alle anderen Tasten auch. Wollen wir Autorepeat abschalten, setzen wir für diesen Parameter eine Null.

Die folgenden drei Ziffern geben die gewünschten ASCII-Codes an, die wir durch Drücken dieser Taste erzeugen wollen. Somit entsprechen

```
121  y  (ohne SHIFT)
 89  Y  (mit SHIFT)
 25  CTRL-Y
```

Hätten wir den letzten Parameter weggelassen, wäre durch Drücken von CTRL-Z wieder das ursprüngliche Zeichen entstanden. Wir wollen hier aber die entsprechenden CTRL-Zeichen gleich mit verlegen.

9.3 Hochauflösende Graphik

Mit der hochauflösenden Graphik können Sie beliebige Formen und Figuren auf den Bildschirm zeichnen und somit Ihr künstlerisches Talent voll zum Ausdruck bringen. Alles, womit wir uns bisher befaßt haben, spielte sich in fertigen oder selbstdefinierten Zeichen ab, d.h., wir konnten nur Zeichen, die sich aus einer 8x8-Pixel-Matrix zusammensetzen, darstellen.

Die hochauflösende Graphik erlaubt das Ansteuern jedes einzelnen Punktes auf dem Bildschirm. Bevor wir uns jedoch näher damit befassen, hier noch einige wichtige Grundlagen, um Verwechslungen mit der Textdarstellung zu vermeiden.

Zu Beginn dieses Kapitels haben wir uns mit den drei Bildschirmmodi MODE 0, 1 und 2 befaßt. Dabei stellten wir fest, daß im Modus 0 20 Zeichen, im Modus 1 40 Zeichen und im Modus 2 80 Zeichen pro Zeile dargestellt werden können, wodurch der Bildschirm eine horizontale Auflösung von 160, 320 und 640 Pixel erhält.

Diese Feststellung ist für die Textdarstsellung richtig, für die hochauflösende Graphik müssen wir sie allerdings noch ergänzen.

Strenggenommen werden im Modus 0 und 1 nicht nur 160 bzw. 320, sondern genauso wie im Modus 2, 640 Pixel in der Horizontalen dargestellt. Die Auflösung ist aber geringer, wenn in den Modi 0 und 1 vier bzw. zwei Pixel zu einem supergroßen Pixel zusammengefaßt werden, das als ein Punkt auf dem Bildschirm erscheint.

In der hochauflösenden Graphik beziehen sich alle Anweisungen, unabhängig vom jeweiligen Modus, immer auf 640 Punkte (0 bis 639) in der Horizontalen und 400 Punkte (0 bis 399) in der Vertikalen, wobei der Koordinatenursprung in der linken unteren Ecke liegt. Deshalb spielt es im Prinzip keine Rolle, ob Sie eine Figur im Modus 0, 1 oder 2 zeichnen, denn sie erhält immer die gleiche Form. Lediglich die Möglichkeit der gleichzeitig darstellbaren Farben ist verschieden.

Probieren Sie alle nachfolgenden Beispiele ruhig in allen drei Modi aus. Dabei werden Sie feststellen, daß Sie im Modus 0 dicke und im Modus 2 dünnere Linien mit wesentlich höherer Auflösung vorfinden.

Nicht nur für die Textdarstellung gibt es einen Cursor, sondern auch für die hochauflösende Graphik; dieser wird als Graphikcursor bezeichnet. Wir können den Graphikcursor an jede beliebige Stelle auf dem Bildschirm setzen - ähnlich wie den Textcursor mit der LOCATE-Anweisung.

Zur Positionierung des Graphikcursors gibt es zwei Möglichkeiten:

PLOT X-Pos., Y-Pos., (Farbregister)

zeichnet einen Punkt an der angegebenen Position. Dabei kann wahlweise ein Farbregister vorgegeben werden, so daß der Punkt in der Farbe erscheint, die diesem Farbregister zugeordnet ist.

MOVE X-Pos., Y-Pos.

setzt den Cursor ebenfalls an die angegebene Stelle, zeichnet allerdings keinen Punkt.

Jetzt wollen wir den Graphikcursor in die Bildschirmmitte setzen und dort einen Punkt zeichnen. Bei 640 Pixel in der Horizontalen und 400 in der Vertikalen hat dieser Punkt die Position 320,200. Wenn Sie jetzt

PLOT 320,200

eingeben, erscheint im Modus 0 ein großer, im Modus 1 ein mittelgroßer und im Modus 2 ein kleiner Punkt in der Bildschirmmitte.

Kommen wir jetzt zur DRAW-Anweisung, mit der wir Linien zwischen zwei angegebenen Punkten zeichnen können. Ausgangspunkt ist dabei immer die augenblickliche Cursorposition. Hier die allgemeine Form von DRAW:

DRAW X-Pos., Y-Pos., (Farbregister)

Die angegebene X/Y-Position entspricht dem Zielpunkt, zu dem die Linie gezeichnet werden soll. Am besten betrachten wir dazu ein kleines Beispiel, das ein Rechteck auf den Bildschirm zeichnet:

```
10 REM Rechteck
20 CLS
30 MOVE 100,100
40 DRAW 540,100
50 DRAW 540,300
60 DRAW 100,300
70 DRAW 100,100
80 a$=INKEY$
90 IF a$="" THEN 80
```

Nach dem Löschen des Bildschirms in Zeile 20 wird der Graphikcursor in Zeile 30 auf die Ausgangsposition 100/100 gesetzt, von der aus das Rechteck gezeichnet werden soll. Von dieser Stelle aus zeichnet Zeile 40 eine Linie zum Punkt 540/100, also die untere waagerechte Linie des Rechtecks. Der Graphikcursor steht dann ebenfalls auf 540/100, dem Punkt, der gleichzeitig die Ausgangsposition für die nächste Linie ist, die senkrecht nach oben gezeichnet wird. Der gleiche Vorgang wiederholt sich für die restlichen zwei Seiten, bis der Cursor wieder an seine ursprüngliche Position 100/100 zurückkehrt.

In Zeile 80 und 90 hängt das Programm in einer Warteschleife, die Sie nur durch Drücken einer beliebigen Taste verlassen können. Diese Warteschleife sollte immer am Ende Ihrer Graphikprogramme stehen, da sonst das Programm sofort beendet wäre und die Ready-Meldung samt Cursor Ihre Graphik zerstören könnte.

Wahlweise können Sie an die DRAW-Anweisung noch einen dritten Parameter anhängen, der eine gewünschte Farbregister-Nummer enthält. Angenommen, wir wollen jetzt die erste Seite in Hellrot (Register 3), die zweite in Hellgrün (Register 9), die dritte in Hellgelb (Register 1) und die vierte in leuchtendweiß (Register 4) zeichnen. Damit die Linien nebst Farben gut zu erkennen sind, wählen wir dazu den Bildschirmmodus 0, der ja ohnehin die größte Farbenvielfalt bietet. Andererseits können wir hier auf eine besonders hohe Auflösung verzichten. Die einzelnen Farbregister enthalten die Farben, die ihnen standardmäßig für den Modus 0 zugeordnet wurden.

```
10 REM Rechteck
15 MODE 0
20 CLS
30 MOVE 100,100
40 DRAW 540,100,3
50 DRAW 540,300,9
60 DRAW 100,300,1
70 DRAW 100,100,4
80 a$=INKEY$
90 IF a$="" THEN 80
```

Außer DRAW gibt es noch die Anweisung DRAWR, die die Zielpunkte nicht absolut, sondern relativ auf den Ausgangspunkt bezogen ansteuert. Im folgenden Beispiel zeichnen wir dasselbe Rechteck noch einmal, dieses Mal aber mit relativen Koordinaten.

```
10 REM Rechteck
15 MODE 0
20 CLS
30 MOVE 100,100
40 DRAWR 440,0,3
50 DRAWR 0,200,9
60 DRAWR -440,0,1
70 DRAWR 0,-200,4
80 a$=INKEY$
90 IF a$="" THEN 80
```

In unserem Programm zeichnet z.B. Zeile 40 eine Linie vom Ausgangspunkt 100/100 zu dem Punkt, der um 440 Punkte in X-Richtung und 0 Punkte in Y-Richtung versetzt ist.

Mit Hilfe der DRAW- und DRAWR-Anweisung können wir beliebige Figuren zeichnen, sofern uns jeweils die Endpunkte bzw. der relative Versatz bekannt ist.

Auf Ihrem CPC 6128 können Sie auch Kreise zeichnen, indem Sie einzelne Kreispunkte errechnen und diese miteinander verbinden. Hier ein Programm, das einen Kreis in 360 Schritten zeichnet:

```
10 REM Kreis
20 r=180
30 DEG
40 CLS
50 ORIGIN 320,200
60 MOVE r,0
70 FOR phi=0 TO 360
80 DRAW r*COS(phi),r*SIN(phi)
90 NEXT
100 A$=INKEY$
110 IF a$="" THEN 100
```

Zunächst legt Zeile 20 den Radius 180 fest, wobei Sie auch einen anderen Wert wählen können. Infolge von DEG in Zeile 30 müssen sämtliche Winkel nicht mehr im Bogenmaß, sondern im Gradmaß angegeben werden.

Zeile 50 legt mit der ORIGIN-Anweisung den Koordinatenursprung, der bisher in der linken unteren Ecke lag, in die Bildschirmmitte (Position 320/200), was für das Zeichnen von Kreisen wesentlich günstiger ist. Bevor nun der Kreis gezeichnet wird, setzt Zeile 60 den Graphikcursor auf den Ausgangspunkt. Schließlich zeichnet die FOR...NEXT-Schleife den Kreis in 360 Segmenten, wobei die einzelnen Kreispunkte in der DRAW-Anweisung nach den Formeln

```
x = r * cos(phi)     und
y = r * sin(phi)
```

berechnet werden.

Ändern wir dieses Programm geringfügig ab, können wir auch eine Ellipse zeichnen:

```
10 REM Ellipse
20 a=180 : b=100
30 CLS
40 DEG
50 ORIGIN 320,100
60 MOVE a,0
70 FOR phi=0 TO 360
80 DRAW a*COS(phi),b*SIN(phi)
90 NEXT
100 a$=INKEY$
110 if a$="" THEN 100
```

Statt des Radius werden in diesem Beispiel die beiden Ellipsenachsen a und b vorgegeben, die wir hier mit 180 bzw. 100 annehmen. Die jeweiligen Ellipsenpunkte errechnen sich nach den Formeln

```
x = a * cos(phi)
y = b * sin(phi)
```

9.4 Sound

Zum Abschluß des Kapitels wollen wir uns noch mit der Sounderzeugung beschäftigen. Dieses Thema ist nicht ganz einfach zu verstehen, da es für die Tonerzeugung sehr viele Variationsmöglichkeiten gibt. So spielen neben der eigentlichen Tonhöhe und -dauer auch Schwankungen bei der Lautstärke und Tonhöhe eine wichtige Rolle.

Jeder Ton hat seine ganz individuellen Eigenschaften, wobei es keine Rolle spielt, ob es sich um die menschliche Stimme, ein Musikinstrument oder das Rauschen des Meeres handelt. In Ihren CPC 6128 ist jedoch ein Sound-Chip eingebaut, der viele Tonarten auf erstaunlich gute Weise simulieren kann.

Gerade wegen der Vielfalt der möglichen Töne und Geräusche sollten Sie selbst mit der SOUND-Anweisung und den dazugehörigen Zusatzanweisungen experimentieren, wobei Ihnen dieser Abschnitt das nötige Rüstzeug auf den Weg gibt.

Gelingt es Ihnen erst einmal, Töne nach Ihren eigenen Vorstellungen zu erzeugen, können Sie leicht ganze Lieder mit bis zu drei Stimmen auf Ihrem CPC 6128 spielen.

Die wichtigste Anweisung, um überhaupt einen Ton aus Ihrem Computer herauszulocken, ist die SOUND-Anweisung, die bis zu sieben Parameter enthalten kann, wie die folgende Aufstellung zeigt:

```
SOUND   Kanalstatus,
        Tonperiode,
        Dauer,
        Lautstärke,
        Lautstärke-Hüllkurve,
        Ton-Hüllkurve,
        Geräusch-Periode
```

Zu Beginn benötigen Sie jedoch nur die ersten drei Parameter. Da der CPC 6128 drei Soundkanäle besitzt, können Sie bis zu drei Stimmen gleichzeitig erzeugen. Der Kanalstatus legt nun fest, welche dieser Kanäle angesprochen wird. Darüber hinaus regelt er auch, welche Stimmen gleichzeitig erklingen sollen. Doch hierzu später mehr.

Die Soundkanäle werden nicht mit der Frequenz, sondern der Tonperiode programmiert, die allerdings im umgekehrten Verhältnis zur Frequenz steht. Wundern Sie sich deshalb nicht, wenn hohe Töne eine kleine und tiefe Töne eine große Periode besitzen.

Die Tonperiode berechnet sich nach folgender Formel:

Periode = 125000 / Frequenz (Hz)

Der CPC 6128 umfaßt insgesamt acht Oktaven, deren Tonperioden Sie der folgenden Tabelle entnehmen können:

Note	Oktave -3 Periode	Oktave -2 Periode	Oktave -1 Periode	Oktave 0 Periode
C	3822	1911	956	478
C#	3608	1804	902	451
D	3405	1703	851	426
D#	3214	1607	804	402
E	3034	1517	758	379
F	2863	1432	716	358
F#	2703	1351	676	338
G	2551	1276	638	319
G#	2408	1204	602	301
A	2273	1136	568	284
A#	2145	1073	536	268
B	2025	1012	506	253

Note	Oktave 1 Periode	Oktave 2 Periode	Oktave 3 Periode	Oktave 4 Periode
C	239	119	60	30
C#	225	113	56	28
D	213	106	53	27
D#	201	100	50	25
E	190	95	47	24
F	179	89	45	22
F#	169	84	42	21
G	159	80	40	20
G#	150	75	38	19
A	142	71	36	18
A#	134	67	34	17
B	127	63	32	16

Beachten Sie, daß die in der Tabelle angegebene Note B in Deutschland der Note H entspricht.

Der dritte Parameter steht für die Dauer des Tons in Einheiten von 1/100-stel Sekunden.

Wir wollen jetzt einmal über den Soundkanal 1 den Kammerton A (Oktave 0) eine Sekunde lang ertönen lassen:

```
SOUND 1,284,100
```

Für eine andere Note suchen Sie sich die entsprechende Tonperiode aus der Tabelle heraus. Da Melodien in ganzen, halben, Viertelnoten usw. gespielt werden, empfiehlt es sich, die Dauer für eine ganze Note fetzulegen.

Nachfolgend ein kleines Programm, das die ersten Noten des Liedes "Oh du lieber Augustin" spielt. Als ganze Note nehmen wir hier den Wert 80 an, als halbe Note 40 usw.

```
100 SOUND 1,319,40
110 SOUND 1,284,20
120 SOUND 1,319,20
130 SOUND 1,358,20
140 SOUND 1,379,20
150 SOUND 1,478,20
160 SOUND 1,478,40
```

Mit dem vierten Parameter können wir die Lautstärke wählen. Sie kann Werte zwischen 0 und 15 annehmen. Der Standardwert beträgt 12.

Der fünfte Parameter gibt die Lautstärke-Hüllkurve an, die wir mit der ENV-Anweisung beliebig definieren können. Bei umfangreichen Melodien können Sie bis zu 16 verschiedene Lautstärke-Hüllkurven festlegen, deren Nummer in der SOUND-Anweisung als fünfter Parameter aufgerufen wird. Selbstverständlich muß die Hüllkurve vor der ersten SOUND-Anweisung, die sie verwenden soll, definiert werden.

Mit dieser Lautstärke-Hüllkurve kann die Lautstärke innerhalb eines Tons beliebig variieren. Dabei können bis zu fünf verschiedene Abschnitte vorgegeben werden. Hier die allgemeine Definition der ENV-Anweisung:

```
ENV N, P1,Q1,R1, P2,Q2,R2,...., P5,Q5,R5
```

wobei

N = Lautstärke-Hüllkurvennummer (0 bis 15)
P1 = Schrittanzahl (0 bis 127)
Q1 = Schrittgröße (-128 bis +127)
R1 = Pausenlänge in 1/100-stel Sekunden (0 bis 127)

Für P2 bis P5, Q2 bis Q5 und R2 bis R5 gilt dasselbe wie für P1, Q1 und R1.

Im folgenden Beispiel betrachten wir noch einmal die gleiche Melodie wie oben, definieren aber die Lautstärke-Hüllkurve 1, so daß sie aus drei Abschnitten besteht. In diesem Fall müssen wir den vierten Parameter für die Lautstärke auslassen, da wir stattdessen die Hüllkurve benutzen:

```
90 ENV 1, 3,5,3, 6,0,10, 3,-5,5
100 SOUND 1,319,40,,1
110 SOUND 1,284,20,,1
120 SOUND 1,319,20,,1
130 SOUND 1,358,20,,1
140 SOUND 1,379,20,,1
150 SOUND 1,478,20,,1
160 SOUND 1,478,40,,1
```

Sicher ist Ihnen aufgefallen, daß in dieser Melodie die letzten beiden Töne nicht mehr ineinanderlaufen, wie es im ersten Beispiel der Fall war. Trotz ihrer gleichen Tonhöhe kann man sie jetzt deutlich unterscheiden.

Auch für die Tonhöhe können wir eine Hüllkurve definieren, deren Nummer an sechster Stelle in der SOUND-Anweisung aufgeführt ist. Durch die gezielte Steuerung der Tonperiode bzw. Frequenz kann ein gewisser Jauleffekt entstehen. Diese Hüllkurve wird durch die ENT-Anweisung festgelegt und hat einen ähnlichen Aufbau wie die ENV-Anweisung. Hier ihre allgemeine Form:

```
ENT S, T1,V1,W1, T2,V2,W2,....., T5,V5,W5
```

wobei

S = Ton-Hüllkurvennummer (0 bis 15)
T1 = Schrittanzahl (0 bis 239)
V1 = Schrittgröße (-128 bis +127)
W1 = Pausenlänge in 1/100-stel Sekunden (0 bis 255)

Für T2 bis T5, V2 bis V5 und W2 bis W5 gilt dasselbe wie für T1, V1 und W1.

Jetzt wollen wir das Lied "Oh du lieber Augustin" ein letztes Mal spielen und zwar mit einer definierten Tonhüllkurve in Zeile 80, die dann jeweils als sechster Parameter in der SOUND-Anweisung aufzuführen ist:

```
80 ENT 1, 6,0,4, 6,-4,4
90 ENV 1, 3,5,3, 6,0,10, 3,-5,5
100 SOUND 1,319,40,,1,1
110 SOUND 1,284,20,,1,1
120 SOUND 1,319,20,,1,1
130 SOUND 1,358,20,,1,1
140 SOUND 1,379,20,,1,1
150 SOUND 1,478,20,,1,1
160 SOUND 1,478,40,,1,1
```

Abschließend kommen wir nochmals auf den Kanalstatus zurück. Er enthält nicht nur die Nummer des Soundkanals, sondern steuert auch das gleichzeitige Erklingen mehrerer Stimmen. Diesen Effekt nennt man "Rendezvous".

Hier zunächst eine Tabelle, die Sie zur Berechnung des Kanalstatus benötigen:

Ziffer	Bit	Bedeutung
1	0	Kanal 1
2	1	Kanal 2
4	2	Kanal 3
8	3	Rendezvous mit Kanal 1
16	4	Rendezvous mit Kanal 2
32	5	Rendezvous mit Kanal 3

Der Kanalstatus ist also ein internes Register, das ein Byte belegt und in dem jedes Bit eine bestimmte Bedeutung hat. Wollen wir z.B. eine Melodie zweistimmig über Kanal 1 und 2 spielen, müssen die SOUND-Anweisungen für beide Stimmen folgendermaßen aussehen:

Spielt Kanal 1 alleine, erhält der Kanalstatus eine 1. Soll dagegen gleichzeitig Stimme 2 gespielt werden (Rendezvous), so ist zu diesem Wert 16 zu addieren, was 17 als Kanalstatus für Kanal 1 ergibt.

Umgekehrt führt Kanal 2 (Ziffer 2) ebenfalls ein Rendezvous mit Kanal 1 (Ziffer 8), was einen Kanalstatus von 10 ergibt. Im folgenden Beispiel wollen wir beide Kanäle gleichzeitig erklingen lassen:

```
10 SOUND 17,478,50
20 SOUND 10,379,100
```

Da Kanal 2 eine längere Tondauer erzeugt als Kanal 1, erklingen zunächst beide Kanäle, worauf dann nur noch Kanal 2 zu hören ist.

10 CP/M

In den vorangehenden Kapiteln kamen wir schon wiederholt auf CP/M zu sprechen, da es einfach zum CPC 6128 dazugehört. Selbst wenn wir mit BASIC und AMSDOS arbeiten, kommen wir nicht ohne CP/M aus, weil einige Dienstprogramme nur unter CP/M laufen. Dies gilt besonders für das Programm DISCKIT3, das wir zum Formatieren und Kopieren von Disketten benötigen.

Darüber hinaus ist die interne Diskettenverwaltung unter CP/M und AMSDOS fast identisch. Alles was in Kapitel 6 bereits über CP/M und Dateitypen gesagt wurde, wollen wir deshalb an dieser Stelle nicht noch einmal wiederholen. Auch das eben genannte Dienstprogramm DISCKIT3 wurde dort ausführlich beschrieben, und es spielt keine Rolle, ob Sie mit ihm eine Diskette formatieren, auf der Sie später BASIC- oder CP/M-Programme ablegen. Sogar auf den mitgelieferten Systemdisketten befinden sich Programme für beide Betriebsarten auf ein und derselben Seite.

10.1 Was ist CP/M?

Anfang bis Mitte der siebziger Jahre steckte die Mikrocomputertechnik noch in den Kinderschuhen. Der 8080-Microprozessor, der Vorgänger des Z80, war damals die neueste Errungenschaft. Auch waren Computer noch sehr teuer, obwohl sie für unsere heutigen Begriffe nur über einen kleinen RAM-Speicher verfügten.

Schon bald entstand der Wunsch, für alle Computer, die einen 8080- und später einen Z80-Mikroprozessor besaßen, ein einheitliches Betriebssystem zu entwickeln, das sich mit wenig Mühe an die jeweilige Hardware anpassen ließ.

Im Jahre 1975 kam die erste CP/M-Version 1.4 auf den Markt, fand aber zunächst wenig Beachtung. Erst nach und nach konnte sie sich durchsetzen. Damals arbeitete CP/M nur mit IBM-kompatiblen 8-Zoll-Diskettenlaufwerken und war nur schwer an andere Systeme anzupassen.

Erst als um 1979/80 CP/M 2.2 herauskam, wurde dies anders. CP/M wurde erweitert und konnte jetzt an die verschiedensten Disketten-Aufzeichnungsformate angepaßt werden. Dies hatte zur Folge, daß CP/M weit verbreitet und eine große Menge an Software für dieses System geschrieben wurde, die ja - von geringen Anpassungen einmal abgesehen - auf den verschiedensten CP/M-Rechnern läuft. Es ist daher auch nicht verwunderlich, daß CP/M-Programme die größte Softwarebibliothek der Welt darstellen.

Zu dieser Version, die eigentlich bis heute als Standard CP/M gilt, kamen noch einige neue Versionen hinzu, die sich mehr oder weniger durchsetzten. Darunter befand sich auch die Version CP/M-86 für die mittlerweile aufgekommenen 16-Bit-Rechner wie beispielsweise der IBM PC. Sie konnte sich aber nicht gegen MSDOS durchsetzen. Aus diesem Grunde blieb CP/M ein Betriebssystem für 8-Bit-Rechner und wird heute vorwiegend auf kleineren Personal- oder Home-Computern eingesetzt, sofern sie dazu ausgerüstet sind.

10.2 CP/M 2.2 und CP/M Plus

Für den CPC 6128 stehen zwei CP/M-Versionen zur Verfügung, nämlich CP/M 2.2 und CP/M Plus. CP/M Plus ist eine weiterentwickelte Version von CP/M 2.2 für 8-Bit-Rechner und wird als gebankte oder als ungebankte Version angeboten. Dies bedeutet, daß CP/M Plus sowohl auf Rechnern mit nur einer Speicherbank (ungebankt), als auch auf Rechnern mit zwei oder mehreren Speicherbänken (gebankt) betrieben werden kann.

Da der CPC 6128 zwei Speicherbänke zu je 64K besitzt, bietet sich hierfür CP/M Plus in der gebankten Version geradezu an. Darüber hinaus enthält CP/M Plus mehr Befehle und Systemprogramme, so daß man mit dieser Version erheblich komfortabler arbeiten kann. Außerdem sind die meisten Programme, die unter CP/M 2.2 erstellt wurden, auch unter CP/M Plus lauffähig, selbst wenn gelegentlich geringe Anpassungen notwendig sind. Sollte wider Erwarten ein Programm nicht unter CP/M Plus laufen, haben Sie die Möglichkeit, auch CP/M 2.2 von der Systemdiskette zu laden.

Da der CPC 6128 für den Betrieb von CP/M Plus ausgelegt ist, werden wir uns im vorliegenden Kapitel schwerpunktmäßig mit dieser Version befassen und CP/M 2.2 nur am Rande betrachten. Doch zuvor wollen wir einen Blick auf die besonderen Eigenschaften werfen, die dieses System gegenüber CP/M 2.2 bietet.

Zum besseren Verständnis müssen wir uns zunächst mit dem Aufbau von CP/M und der internen Speicherverwaltung etwas näher befassen.

Jedes CP/M-System ist unabhängig von seiner Version in drei Hauptteile aufgeteilt: in den Command Control Prozessor (CCP), der die Ein-/Ausgabe über Tastatur und Bildschirm steuert, in das Basic Disc Operating System (BDOS), das für Diskettenoperationen zuständig ist und in das Basic Input/Output System (BIOS), eine Schnittstelle zwischen den geräteunabhängigen Teilen CCP und BDOS und der jeweiligen Hardware. Aus diesem Grunde sind auf allen Rechnern, die mit der gleichen CP/M-Version arbeiten, CCP und BDOS immer identisch, während das BIOS geräteabhängig ist und jeweils auf Rechner, Diskettenformat, Drucker etc. angepaßt werden muß. Beim CPC 6128 ist dies bereits werksseitig geschehen, so daß Sie sich hierum nicht mehr zu kümmern brauchen.

Außer diesen Systemteilen wird auch noch Speicherplatz für die auszuführenden Programme benötigt, der als Transient Program Area (TPA) bezeichnet wird. Während unter CP/M 2.2 Betriebssystem und TPA in nur einer Bank lagen, sind sie unter CP/M Plus in der gebankten Version auf zwei Bänke verteilt, nämlich auf die Systembank (Bank 0) und die TPA-Bank (Bank 1).

Nun kann der Programmspeicher (TPA) aber nicht den gesamten Speicherplatz in der Bank einnehmen, da ein geringer Teil hiervon noch für das residente BDOS und BIOS benötigt wird. Beim CPC 6128 bleiben aber immer noch 61K TPA übrig.

Der Hauptteil des BDOS und BIOS liegt jedoch in der anderen Speicherbank (Bank 0), obwohl er dort bei weitem nicht den gesamten Platz in Anspruch nimmt. In dieser Speicherbank ist nämlich außerdem noch der CCP abgelegt, so daß er bei einem Warmstart jederzeit in die TPA übertragen werden kann. Bei CP/M 2.2 mußte der CCP dagegen jedesmal von Diskette nachgeladen werden.

Der größte Teil von Bank 0 dient jedoch als Puffer für Diskettenoperationen. Je mehr Speicherplatz hier zur Verfügung steht (eventuell auch noch in zusätzlichen Bänken ab Bank 2), desto seltener muß auf die Diskette zugegriffen werden. Dadurch verringern sich natürlich die Zugriffszeiten.

Wurden Dateien unter CP/M 2.2 noch im Recordformat (128 Bytes) vom BIOS gelesen und geschrieben, so sind es unter CP/M-Plus physikalische Diskettensektoren, die meist wesentlich größer sind als 128 Bytes und im Puffer abgelegt werden. Die Aufteilung in Records übernimmt hier das BDOS.

Neben dem schnelleren Diskettenzugriff und mehr TPA-Speicher hat CP/M Plus aber noch weitere Vorteile. Es enthält ein HELP-Programm, das Sie aufrufen können, wenn Sie Unterstützung brauchen und das Ihnen häufig das Suchen im Handbuch erspart. Auch können Sie in sämtliche Dateien das Datum mit genauer Uhrzeit schreiben. Wenn Sie es wünschen, können Sie außerdem ein nur Ihnen bekanntes Passwort eingeben und somit verhindern, daß Unbefugte Zugang zu Ihren Dateien haben.

CP/M Plus enthält viele Dienstprogramme, die sich teilweise von denen unter CP/M 2.2 unterscheiden und die ein angenehmes Arbeiten ermöglichen. Im weiteren Verlauf dieses Kapitels werden wir noch näher darauf eingehen.

10.3 Der CP/M-Start

Zunächst müssen wir CP/M Plus bzw. CP/M 2.2 von der Systemdiskette laden und initialisieren. Lagen Sie dazu Seite 1 (CP/M 2.2 Seite 4) Ihrer Systemdiskette ins Laufwerk und geben Sie dann

```
|CPM  <RETURN>
```

ein. Nach einigen Sekunden erscheint folgende Einschaltmeldung auf dem Bildschirm (CP/M Plus):

```
CP/M Plus Amstrad Consumer Electronics plc
v 1.0, 61K TPA, 1 disc drive

A>
```

Das A mit der spitzen Klammer ist das Anforderungszeichen (Prompt) für Laufwerk A. In der rechten unteren Ecke steht noch "Drive is A:". Wie wir bereits wissen, ist Laufwerk A das eingebaute Laufwerk, während ein zusätzlich angeschlossenes Laufwerk die Bezeichnung B trägt. Diese Meldung besagt außerdem, daß Laufwerk A das Bezugslaufwerk ist, auch dann, wenn Sie ein zweites Laufwerk angeschlossen haben.

Bei zwei Laufwerken können Sie auch Laufwerk B als Bezugslaufwerk festlegen. Dies geschieht mit

A>B: <RETURN>

Vom Bezugslaufwerk sollten Sie immer das CP/M-Betriebssystem und andere Dienstprogramme laden. Ein zweites Laufwerk erleichtert jedoch die Arbeit mit Datendateien erheblich.

Nehmen wir einmal das bekannte Textverarbeitungsprogramm WORDSTAR als Beispiel. Dieses Programm wird nicht etwa nur einmal geladen und anschließend gleich ausgeführt; vielmehr besteht WORDSTAR aus mehreren Dateien, die während der Textbearbeitung abwechselnd in die TPA geladen werden. Andererseits müssen Sie auch Ihre Textdateien auf Diskette ablegen und zwar einschließlich einer Backup-Datei (Extension .BAK) und einer Zwischendatei (Extension .$$$). Dabei ist die BACKUP-Datei eine Sicherungsdatei, die die gleiche Bedeutung hat wie unter AMSDOS. Die Zwischendatei dagegen wird hier von WORDSTAR während der Textbearbeitung angelegt und im Anschluß daran wieder gelöscht.

Arbeiten Sie nur mit einem Laufwerk, müssen Sie Ihre Texte auf die gleiche Diskette schreiben, auf der sich auch die WORDSTAR-Programmdateien befinden, die aber schon rund zwei Drittel der Diskettenkapazität beanspruchen. Für Ihre eigenen Dateien bleiben somit nur noch rund 60K übrig. Beachten Sie aber, daß Sie hiervon nur etwa ein Drittel echt nutzen können, da die Backup- und Zwischendatei ungefähr den gleichen Speicherplatz beanspruchen, wie die eigentliche Textdatei. Für sie verbleiben dann nur noch ca. 20K, was etwa 5 vollbeschriebenen DIN A4-Seiten entspricht.

Da WORDSTAR theoretisch beliebig lange Textdateien bearbeiten kann, von denen immer nur ein kleiner Teil in die TPA geladen wird, ist es wesentlich bequemer, ein zweites Laufwerk B mit anzuschließen. Dann nämlich kann die WORDSTAR-Diskette in Laufwerk A verbleiben und die Diskette mit Textdateien in Laufwerk B.

Ähnliche überlegungen gelten natürlich auch für andere CP/M-Programme, wie z.B. das Datenbankprogramm dBASE II oder das Tabellenkalkulationsprogramm MULTIPLAN. Diese Programme sind übrigens, genauso wie WORDSTAR, bei Markt&Technik erhältlich und zwar fertig angepaßt an den CPC 6128.

CP/M-Programme werden in der Regel geladen und automatisch ausgeführt, was man als Autostart bezeichnet. Diese Programme müssen als Command-Datei gekennzeichnet sein und die Extension .COM erhalten. Hier einige Beispiele:

```
SID.COM
PIP.COM
WS.COM
```

Mit dem letzten Namen wird beispielsweise WORDSTAR aufgerufen. Legen Sie nach dem Initialisieren von CP/M die WORDSTAR-Diskette ins Bezugslaufwerk und geben Sie ein:

```
A>WS  <RETURN>
```

Dabei dürfen Sie keinesfalls die Extension .COM angegeben. An ihr erkennt CP/M lediglich, daß es sich um eine Command-Datei mit Autostart handelt. Das schließt allerdings nicht aus, daß WORDSTAR oder ein anderes Programm Programmteile nachlädt, die mit einer anderen Extension auf der Diskette abgelegt sind. WORDSTAR benötigt mindestens noch die Dateien WSMSGS.OVR und WSOVLY1.OVR, um überhaupt lauffähig zu sein.

Genauso wie WORDSTAR können Sie auch andere .COM-Dateien laden und ausführen. Auch hier geben Sie nur den Namen (ohne .COM) ein und drücken die RETURN-Taste. In Zukunft setzen wir dies als bekannt voraus und verzichten auf den Hinweis <RETURN>. So lädt und startet z.B.

```
A>SID
```

das Debugger-Dienstprogramm SID. Selbstverständlich müssen Sie hier nicht das Anforderungszeichen A> eingegben, denn es erscheint automatisch auf dem Bildschirm, wenn CP/M einen Warmstart durchgeführt hat und zur Aufnahme neuer Befehle bereit ist.

10.4 Steuerzeichen mit CTRL-Taste

Unter CP/M-Plus gelten folgende Steuerzeichen, die durch gleichzeitiges Drücken der CTRL- und der angegebenen Buchstabentaste erzeugt werden:

CTRL-A	Cursor um ein Zeichen nach links
CTRL-B	Setzt Cursor an den Zeilenanfang. Wenn er sich dort bereits befindet, springt er zum Zeilenende
CTRL-C	Beendet Programmausführung
CTRL-E	Physikalischer Carriage Return, ohne Übergabe der Kommandozeile an das System
CTRL-F	Cursor um ein Zeichen nach rechts
CTRL-G	Löscht das Zeichen unter dem Cursor
CTRL-H	Löscht das Zeichen links vom Cursor und setzt den Cursor um eine Stelle nach links (Backspace)
CTRL-I	Cursor springt zum nächsten Tabulatorstop
CTRL-J	Setzt Cursor an den Anfang der Kommandozeile und übergibt das Kommando an das System
CTRL-K	Löscht sämtliche Zeichen von Cursorposition bis Zeilenende
CTRL-M	Carriage Return
CTRL-P	Schaltet Drucker an und ab
CTRL-Q	Nimmt Bildschirm-Scrollen nach CTRL-S wieder auf
CTRL-R	Alle Zeichen links vom Cursor werden in die nächste Zeile kopiert und im Kommandopuffer abgelegt
CTRL-S	Unterbindet Bildschirm-Scrollen
CTRL-U	Kopiert alle Zeichen links vom Cursor in den Kommandopuffer
CTRL-W	Wiederholt letzte Kommandozeile, wenn die neue Kommandozeile kein Zeichen enthält, anderenfalls wird Cursor ans Zeilenende gesetzt
CTRL-X	Löscht alle Zeichen links vom Cursor
CTRL-Z	Textende

10.5 CP/M Plus-Befehle und -Dienstprogramme

Dieser Abschnitt stellt sämtliche CP/M-Befehle und -Dienstprogramme in alphabetischer Reihenfolge vor, die standardmäßig für jedes CP/M Plus-System zur Verfügung stehen. Einige Befehle sind dabei bereits im CCP enthalten, für andere muß ein spezielles Programm von der Systemdiskette geladen werden.

10.5.1 DATE

DATE dient zum Einstellen und Abfragen des Datums und der Uhrzeit. Die interne Uhr läuft natürlich nur so lange richtig mit, wie der Computer eingeschaltet bleibt. Deshalb müssen Sie nach jedem Einschalten Datum und Uhrzeit neu einstellen.

```
DATE mm/tt/jj hh:mm:ss
```

stellt die interne Uhr. Hier ein Beispiel:

```
DATE 08/18/85 16:11:15
```

Diese Eingabe gibt als Datum den 16. August 1985 und als Uhrzeit 16 Uhr, 11 Min. und 15 Sek. an. Geben Sie dagegen nur DATE ein, fragen Sie die interne Uhr ab, und Datum und Uhrzeit erscheinen auf dem Bildschirm.

10.5.2 DIR

Mit diesem Befehl können Sie das Directory oder Teile davon auflisten.

```
DIR
```

listet alle Dateinamen der Diskette im Bezugslaufwerk,

```
DIR B:
```

alle Dateinamen der Diskette in Laufwerk B. Mit

```
DIR S???????.*
```

erscheinen alle Dateinamen, die mit S beginnen und

```
DIR * COM
```

listet alle .COM-Dateien.

Aus diesen Beispielen erkennen wir, daß ein Sternchen (*) für sämtliche Dateinamen (ohne Extension) bzw. alle Extensionen gilt. Wird Name oder Extension nur teilweise angegeben, müssen die nichtgenannten Zeichen durch Fragezeichen ersetzt werden. Dabei muß der Dateiname aus insgesamt acht Zeichen bestehen und im Anschluß an einen weiteren Punkt die Extension mit drei Zeichen folgen. Die Extension muß aber auf jeden Fall angegeben werden. Das gesamte Directory im Bezugslaufwerk könnten wir theoretisch auch so listen:

```
DIR *.*
```

oder

```
DIR ????????.???
```

Durch Drücken von CTRL-P wird zusätzlich der Drucker ein- und wieder abgeschaltet. Achten Sie unbedingt darauf, daß in diesem Fall ein Drucker angeschlossen und eingeschaltet ist, da der Computer sonst in einer Warteschlange hängt, aus der er nicht mehr herauskommt. Indem Sie nacheinander

```
CTRL-P
DIR *.COM
CTRL-P
```

eingeben, schalten Sie den Drucker ein, listen dann sämtliche COM-Dateien auf Bildschirm und Drucker und schalten anschließend den Drucker wieder ab.

Auf diese Weise können Sie nicht nur das Directory ausdrucken, sondern auch sämtliche Bildschirmausgaben gleichzeitig über den Drucker ausgeben. Dies funktioniert allerdings nur, wenn der Cursor hinter dem Anforderungszeichen A> steht, d.h. CP/M-Befehle entgegennehmen kann.

10.5.3 DUMP

DUMP listet eine beliebige Datei in hexadezimaler Schreibweise auf. Selbstverständlich können Sie auch hier den Drucker zuschalten (s.o.). Die Anweisung

```
DUMP PIP.COM
```

listet beispielweise das Programm PIP.COM. Auch hier muß unbedingt die Extension angegeben werden.

10.5.4 ED

ED ist ein einfaches Texteditorprogramm, mit dem man Textdateien erstellen und ändern kann. Allerdings ist die Anwendung dieses Programms sehr umständlich gegenüber komfortablen Textverarbeitungsprogrammen wie z.B. WORDSTAR. ED sollte daher nur dann eingesetzt werden, wenn es keine andere Möglichkeit gibt.

10.5.5 ERA

Dieser Befehl dient zum Löschen von Dateien. Hier einige Beispiele:

```
ERA TEXT.LIB
```

löscht die Datei TEXT.LIB, und die Eingabe

```
ERA *.BAK
```

löscht sämtliche Backup-Dateien (um eventuell mehr Platz für andere Dateien auf der Diskette zu schaffen) und

```
ERA *.*
```

löscht sämtliche Dateien auf der Diskette. Da dieser Befehl bei falscher Anwendung fatale Folgen haben kann, erscheint vor Ausführung des Befehls zunächst die folgende Sicherheitsabfrage auf dem Bildschirm:

ALL (Y/N)?

Soll die Löschung erfolgen, geben Sie Y ein, anderenfalls brechen Sie mit N das Kommando ab.

10.5.6 HELP

Dieses Programm können Sie aufrufen, wenn Sie Informationen über CP/M Plus benötigen. Sie ersparen sich dadurch das Nachschlagen im Handbuch.

Wenn Sie nur

HELP

eingeben, erscheint eine Auflistung verschiedener Stichworte auf dem Bildschirm, die Sie einzeln aufrufen können. Wenn Sie jetzt das gewünschte Stichwort eintippen und die RETURN-Taste drücken, werden die entsprechenden Informationen ausgegeben.

Sie können auch die HELP-Funktion zusammen mit einem gewünschten Stichwort aufrufen. Wenn Sie z.B.

HELP DIR

eingeben, erhalten Sie sämtliche Informationen über den DIR-Befehl.

10.5.7 INITDIR

Mit diesem Befehl können Sie im Directory zusätzliche Informationen ablegen. So ist es beispielsweise möglich, bestimmte Dateien mit einem Passwort zu versehen, um sie vor unberechtigtem Zugriff zu schützen oder um Dateien für den automatischen Eintrag des Datums und der Uhrzeit vorzusehen.

10.5.8 LINK

LINK dient zum Verbinden (link) oder Verschieben (relocate) von Dateien im Microsoft-REL-Format. Dieses Format wird von verschiedenen Assemblern und Compilern zur Erzeugung des Objektcodes (ausführbares Maschinenprogramm) verwendet.

10.5.9 PIP

PIP ist ein universelles Kopierprogramm zur Übertragung von Dateien von Diskette auf ein Peripheriegerät. Es kann auch zum Ausdrucken von Dateien oder zum Kopieren von Dateien auf ein anderes Diskettenlaufwerk benutzt werden.

Die Funktionsweise dieses Programms betrachten wir am besten anhand einiger Beispiele. Die Anweisung

```
PIP B:NEUDATEI.COM=A.ALTDATEI.COM
```

kopiert die Datei ALTDATEI.COM, die sich in Laufwerk A befindet, auf Laufwerk B, wo sie unter dem Namen NEUDATEI.COM abgelegt wird. Geben Sie

```
PIP B:=A:*.*
```

ein, werden sämtliche Dateien von Laufwerk A nach Laufwerk B kopiert.

PIP kann auch mehrere Dateien verbinden und daraus eine neue Datei erzeugen:

```
PIP A:NEUDATEI.TXT=B:DATEI1.TXT,DATEI2.TEXT,DATEI3.TXT
```

Hier werden die Dateien DATEI1.TXT, DATEI2.TXT und DATEI3.TXT, die sich in Laufwerk B befinden, zusammengesetzt und unter dem Namen NEUDATEI.TXT in Laufwerk A abgelegt.

Sollen Dateien über ein Peripheriegerät ein- oder ausgegeben werden, so sind folgende Bezeichnungen zu verwenden:

CON:	Ausgabe auf Bildschirm
LST:	Ausgabe auf Drucker
PUN:	Ausgabe auf serielles Interface (z.B. RS232-Schnittstelle)
RDR:	Eingabe über serielles Interface (z.B. RS232-Schnittstelle)

Hier zwei häufige Anwendungsfälle. Die Anweisung

```
PIP CON:=TEST.TXT
```

gibt die Datei TEST.TXT auf dem Bildschirm aus. Mit

```
PIP LST:=TEST.TXT
```

wird die gleiche Datei über den Drucker ausgegeben. In beiden Fällen sollten aber nur ASCII-Dateien ausgegeben werden, da es sonst zu sinnlosen Zeichen und unkontrollierbaren Bildschirmreaktionen kommt.

In allen hier gezeigten Beispielen wird zunächst die Datei PIP geladen und dann der entsprechende Kopiervorgang eingeleitet. Wir haben aber auch die Möglichkeit, PIP nur einmal zu laden und dann beliebig oft anzuwenden. Die Angabe

```
PIP  (ohne Zusatz)
```

lädt das Kopierprogramm PIP in die TPA und meldet sich mit einem Sternchen. Jetzt müssen Sie nur noch die entsprechende Anweisung geben, wobei PIP jeweils entfällt. Dann kopiert beispielsweise

```
*B:=A:*.*
```

sämtliche Dateien von Laufwerk A nach Laufwerk B oder

```
*LST:=TEST.TXT
```

gibt die Datei TEST.TXT über den Drucker aus. Nach Beendigung des jeweiligen Kopiervorgangs erscheint wieder das Sternchen (*) und signalisiert, daß PIP bereit ist, weitere Anweisungen entgegenzunehmen.

Um PIP zu verlassen, drücken Sie einfach die RETURN-Taste. Der Computer führt dann einen Warmstart aus und das Anforderungszeichen A> erscheint wieder auf dem Bildschirm.

10.5.10 REN

REN dient zum Umbenennen von Dateinamen und ist sehr einfach zu handhaben. Hier zwei Beispiele:

```
REN TEST.TXT=TEST.BAK
REN NEU.DAT=ALT.DAT
```

Das erste Beispiel benennt eine Backup-Datei wieder in eine Textdatei um. Dieser Anwendungsfall ist dann gegeben, wenn man auf eine Sicherungskopie (Extension .BAK) wieder zugreifen muß. Im zweiten Fall erhält die Datei ALT.DAT den neuen Namen NEU.DAT.

Achten Sie darauf, daß immer zuerst der neue und dann der alte Name angegeben werden muß.

10.5.11 RMAC

Hierbei handelt es sich um einen 8080-Makroassembler, mit dem Sie eigene Maschinenprogramme schreiben können. Er erzeugt einen verschiebbaren Objektcode im REL-Format, der dann mit Hilfe des LINK-Programms (s.o.) an die richtige Stelle im Speicher verschoben wird.

10.5.12 SAVE

SAVE dient zum erneuten Abspeichern von CP/M-Programmen, die mit dem Debugger SID geändert wurden. Nach dem Aufruf des Programms werden Sie zur Eingabe einiger Werte aufgefordert. Hier ein Beispiel:

```
SAVE Ver 3.0
File (or RETURN to exit)?TEST.COM
From? 100
To? 800
```

Zunächst muß der Name mit Extension eingegeben werden, unter dem die Datei auf Diskette geschrieben werden soll. In unserem Fall lautet er TEST.COM. Die beiden nächsten Eingabewerte stellen die Anfangs- und Endadresse dar, die hier nicht in dezimaler, sondern in hexadezimaler Schreibweise anzugeben sind. In diesem Beispiel wird also die Datei TEST.COM von 100H (Beginn aller COM-Dateien) bis 800H auf Diskette geschrieben.

10.5.13 SET

Mit diesem Programm können Directory und Dateien mit bestimmten Angaben versehen werden. So kann das Directory ein Label erhalten, d.h. eine spezielle Kennzeichnung für die jeweilige Diskette. Außerdem können Sie Directory-Label und Dateien noch ein Passwort zuordnen.

SET steht in engem Zusammenhang mit INITDIR.COM (s.o.).

10.5.14 SHOW

Mit SHOW erhalten Sie disketten- bzw. dateispezifische Angaben aufgezeigt:

SHOW (ohne Zusatz)

gibt den verfügbaren Diskettenspeicher aller Laufwerke an.

SHOW B:

hat die gleiche Funktion, nur für Laufwerk B.

10.5.15 SID

SID ist ein symbolischer 8080-Debugger, der zum Auflisten und Ändern von CP/M-Dateien dient und mit dem Sie außerdem Programme im Einzelschrittmodus testen (trace) können. Der Aufruf von

SID TEST.COM

lädt z.B. zunächst den Debugger SID und dann das Programm PIP.COM, das untersucht werden soll. Nach Beendigung der Bearbeitung kann die Datei mit Hilfe von SAVE (s.o.) wieder auf Diskette geschrieben werden.

10.5.16 SUBMIT

Mit diesem Befehl können Sie eine beliebige Anzahl von CP/M-Befehlen aneinanderreihen, die dann durch einen einzigen Aufruf ausgeführt werden. Diesen Vorgang nennt man Stapelverarbeitung (batch processing).

So ist es z.B. möglich, durch einen Aufruf mehrere Programme auszuführen, das Directory zu listen, .BAK-Dateien zu löschen und sogar Parameter zu übergeben.

Zunächst müssen Sie dazu jedoch eine Befehlsdatei mit der Extension .SUB anlegen, die die verschiedenen Befehle enthält und ausführt. Im folgenden Beispiel wollen wir eine SUBMIT-Datei unter dem Namen TEST.SUB erzeugen. Dabei sollen sämtliche Backup-Dateien gelöscht, das Directory ausgegeben und dann das Programm SUPER.COM ausgeführt werden. Um dies zu ermöglichen, müssen die einzelnen Anweisungen mit Hilfe eines Texteditors geschrieben und in der Datei TEST.SUB abgelegt werden. Dies können wir aber auch in BASIC und AMSDOS tun, wie das folgende Beispiel zeigt:

```
10 OPENOUT "TEST.SUB"
20 PRINT#9, "ERA *.BAK"
30 PRINT#9, "DIR"
40 PRINT#9, "SUPER"
50 CLOSEOUT
```

Wenn Sie jetzt

```
SUBMIT TEST
```

eingeben, werden alle Anweisungen der Reihe nach abgearbeitet.

10.5.17 TYPE

Mit dem TYPE-Befehl können Sie ASCII-Dateien auf dem Bildschirm ausgeben. Geben Sie dann CTRL-P (s. 10.5.1) ein, erfolgt die Ausgabe auch auf dem Drucker. Beispielsweise listet

```
TYPE PROBE.TXT
```

den Inhalt der Datei PROBE.TXT.

Anhang

Verzeichnis der BASIC-Schlüsselwörter

ABS	Bestimmt Absolutwert einer Zahl
AFTER	Ruft in bestimmten Zeitabständen Unterprogramm auf
AND	Logische UND-Verknüpfung
ASC	Erzeugt ASCII-Wert eines Zeichens
ATN	Arcustangens-Funktion
AUTO	Automatische Zeilennumerierung
BIN$	Erzeugt Binärzahl
BORDER	Setzt Farbe für Bildschirmrand
BREAK	siehe ON BREAK CONT, ON BREAK GOSUB, ON BREAK STOP
CALL	Ruft Maschinen-Unterprogramm auf
CAT	Listet Disketten-Inhaltsverzeichnis
CHAIN	Lädt neues Programm von Diskette und ersetzt altes Programm im Speicher
CHAIN MERGE	Lädt neues Programm von Diskette und vermischt es mit dem im Speicher befindlichen Programm
CHR$	Erzeugt Zeichen aus ASCII-Code
CINT	Erzeugt gerundeten ganzzahligen Wert einer Zahl
CLEAR	Löscht alle Variablen
CLEAR INPUT	Löscht Tastaturpuffer
CLG	Füllt Graphikbildschirm mit Hintergrundfarbe

CLOSEIN	Schließt Eingabedatei
CLOSEOUT	Schließt Ausgabedatei
CLS	Löscht Bildschirm
CONT	Programmfortführung nach Unterbrechung
COPYCHR$	Kopiert Zeichen auf Bildschirm
COS	Cosinus-Funktion
CREAL	Wandelt Zahl in Fließkommawert um
CURSOR	Schaltet Cursor bei INKEY$ an/ab
DATA	Speichert Datenwerte für READ-Anweisung
DEC$	Erzeugt Dezimalstring einer Zahl
DEF FN	Definiert Funktion
DEFINT	Ändert Variablentyp in Integer um
DEFREAL	Ändert Variablentyp in Fließkomma um
DEFSTR	Ändert Variablentyp in String um
DEG	Winkeleingaben in Gradmaß
DELETE	Löscht Zeilennummern
DERR	Abfrage von Diskettenfehlern
DI	Verhindert Unterbrechungen außer BREAK
DIM	Dimensioniert Felder
DRAW	Zeichnet Linien
DRAWR	Zeichnet Linien mit relativen Koordinaten
EDIT	Editiert Programmzeile
EI	Läßt Unterbrechungen wieder zu nach DI
ELSE	siehe IF
END	Ende eines BASIC-Programms
ENT	Definiert Tonhüllkurve
ENV	Definiert Lautstärke-Hüllkurve
EOF	Fragt auf Dateiende ab
ERASE	Löscht Datei auf Diskette
ERL	Enthält Nummer der Zeile, in der Fehler auftrat
ERR	Enthält Fehlernummer
ERROR	Führt Behandlung des Fehlers mit angegebener Nummer aus
EVERY	Ruft Unterprogramm in bestimmten Zeitabständen auf

EXP	Errechnet Potenz zur Zahl e
FILL	Füllt Graphiken farbig aus
FIX	Bestimmt ganzzahligen Wert einer Zahl
FN	Ruft durch DEF FN definierte Funktion auf
FOR...NEXT..STEP	Erzeugt Zählschleife, evt. mit Schrittweite
FRAME	Erzeugt weiche Bildschirmausgabe
FRE	Bestimmt freien BASIC-Speicher
GOSUB	Ruft Unterprogramm auf
GOTO	Springt in angegebene Programmzeile
GRAPHICS PAPER	Dient zum Zeichnen unsichtbarer Linien
GRAPHICS PEN	Setzt Farbe zum Zeichnen
HEX$	Formt Dezimalzahl in Hexadezimalstring um
HIMEM	Ermittelt Speicherobergrenze für BASIC
IF...THEN...ELSE	Fragt Bedingung ab und verzweigt gegebenenfalls
INK	Ordnet Farbe einem Farbregister zu
INKEY	Dient zur Abfrage einzelner Tasten
INKEY$	Fragt von der Tastatur eingegebenes Zeichen ab
INP	Ermittelt Eingabewert eines I/O-Ports
INPUT	Holt Eingabewert vom Bildschirm
INSTR	Gibt Position eines Strings in einem anderen an
INT	Ermittelt den nächstniedrigeren ganzzahligen Wert einer Zahl
JOY	Fragt Joystickposition ab
KEY	Definiert Funktionstaste
KEY DEF	Definiert Taste
LEFT$	Gibt linken Teil eines Strings wieder
LEN	Gibt Länge eines Strings wieder
LET	Zuordnungsanweisung
LINE INPUT	Wie INPUT, jedoch auch mit beliebigen Zeichen
LIST	Listet BASIC-Programm
LOAD	Lädt Programm von Diskette oder Kassette
LOCATE	Positioniert Cursor
LOG	Berechnet natürlichen Logarithmus
LOG10	Berechnet Zehnerlogarithmus
LOWER$	Erzeugt Ausgabe in Kleinbuchstaben

MASK	Setzt Maske bei Graphik
MAX	Bestimmt den Maximalwert einer Zahlenreihe
MEMORY	Setzt Speicherobergrenze für BASIC
MERGE	siehe CHAIN MERGE
MID$	Gibt Teil eines Strings wieder oder setzt einen String in einen anderen ein
MIN	Bestimmt den Minimalwert einer Zahlenreihe
MOD	Gibt Rest bei Division an
MODE	Setzt Bildschirmmodus auf 20, 40 oder 80 Zeichen
MOVE	Setzt Graphikcursor
MOVER	Setzt Graphikcursor relativ zur Ausgangsposition
NEW	Löscht BASIC-Programm im Speicher
NEXT	siehe FOR...NEXT
NOT	Logische NICHT-Verknüpfung
ON BREAK CONT	Verhindert Programmunterbrechung durch ESC-Taste
ON BREAK GOSUB	Aufruf eines Unterprogramms bei Drücken der ESC-Taste
ON BREAK STOP	Läßt Programmunterbrechung nach ON BREAK CONT wieder zu
ON ERROR GOTO	Verzweigt bei Fehler in Fehlerbehandlungsroutine
ON...GOSUB	Aufruf eines Unterprogramms bei Auftreten eines Zahlenwertes
ON...GOTO	Verzweigung bei Auftreten eines Zahlenwertes
ON SQ...GOSUB	Ruft Unterprogramm auf, falls Platz in Tonwarteschlange
OPENIN	Öffnet Eingabedatei
OPENOUT	Öffnet Ausgabedatei
OR	Logische ODER-Verknüpfung
ORIGIN	Setzt Koordinatenursprung bei Graphik
OUT	Gibt Wert in I/O-Port aus
PAPER	Ordnet Hintergrundfarbe einem Farbregister zu
PEEK	Liest Inhalt einer Speicherzelle im RAM
PEN	Ordnet Vordergrundfarbe einem Farbregister zu

PI	Enthält 3.14159...
PLOT	Zeichnet Graphikpunkt
PLOTR	Zeichnet Graphikpunkt mit relativem Versatz
POKE	Schreibt Wert in Speicherzelle (RAM)
POS	Gibt horizontale Cursorposition an
PRINT	Gibt Daten auf Bildschirm aus
PRINT USING	Formatierte Ausgabe von Zahlen und Strings
RAD	Eingabe von Winkeln im Bogenmaß
RANDOMIZE	Setzt Keimzahl für Zufallszahlengenerator
READ	Liest DATA-Werte und legt sie in Variablen ab
RELEASE	Hebt Wartezustand bei Tönen auf
REM	Beginn einer Kommentarzeile
REMAIN	Gibt verbleibende Restzeit des Weckers an
RENUM	Neunumerierung des BASIC-Programms
RESTORE	Setzt DATA-Zeiger auf Zeilennummer
RESUME	Beendigung einer Fehlerbehandlungsroutine
RETURN	Beendigung eines Unterprogramms
RIGHT$	Gibt rechten Teil eines Strings wieder
RND	Erzeugt Zufallszahl
ROUND	Rundet Zahlenwert
RUN	Startet Programmablauf
SAVE	Speichert Programm auf Diskette oder Kassette
SGN	Bestimmt Vorzeichen einer Zahl
SIN	Sinusfunktion
SOUND	Erzeugt Toneffekte
SPACE$	Erzeugt String mit Leerzeichen
SPC	Dient zur Formatierung des Bildschirms
SPEED INK	Setzt Länge der Farbdauer bei Farbwechsel
SPEED KEY	Setzt Geschwindigkeit bei Tastenwiederholfunktion
SPEED WRITE	Setzt Schreibgeschwindigkeit auf Kassette
SQ	Gibt Anzahl der freien Plätze in Tonwarteschlange an
SQR	Berechnet Quadratwurzel
STEP	siehe FOR...NEXT...STEP

STOP	Unterbricht BASIC-Programm
STR$	Wandelt Zahl in String um
STRING$	Erzeugt String mit gleichlautenden Zeichen
SWAP	siehe WINDOW SWAP
SYMBOL	Definition eigener Zeichen
SYMBOL AFTER	Setzt untere Grenze (ASCII-Code) zur Zeichendefinition
TAB	Setzt Tabulator auf Bildschirm
TAG	Ermöglicht PRINT auf Graphikcursorposition
TAGOFF	Abschalten von TAG
TAN	Tangens-Funktion
TEST	Gibt Zeichenfarbe an angegebener Graphikcursorposition an
TESTR	Wie TEST, nur mit relativen Koordinaten
THEN	siehe IF...THEN...ELSE
TIME	Gibt Zeit in 1/300 Sek. nach Einschalten des Computers an
TRON	Schaltet Ablaufverfolgung ein
TROFF	Schaltet Ablaufverfolgung aus
UNT	Gibt ganzzahligen Wert eines 16 Bit-Ausdrucks
UPPER$	Erzeugt Ausgabe in Großbuchstaben
VAL	Wandelt Ziffernstring in numerischen Wert um
VPOS	Gibt vertikale Cursorposition an
WAIT	Wartet, bis Wert über I/O-Port eingegeben
WHILE...WEND	Programmschleife
WIDTH	Setzt Anzahl Zeichen pro Zeile für Drucker
WINDOW	Definiert Bildschirmfenster
WINDOW SWAP	Tauscht Inhalt zweier Bildschirmfenster aus
WRITE	Wie PRINT, jedoch mit beliebigen Zeichen
XOR	Logische EXKLUSIV-ODER-Verknüpfung
XPOS	Gibt X-Koordinate des Graphikcursors an
YPOS	Gibt Y-Koordinate des Graphikcursors an
ZONE	Ändert Breite der Druckzonen bei PRINT

ASCII-Code-Tabelle

0 NUL	16 DLE	32 SP	48 0
1 SOH	17 DC1	33 !	49 1
2 STX	18 DC2	34 "	50 2
3 ETX	19 DC3	35 #	51 3
4 EOT	20 DC4	36 $	52 4
5 ENQ	21 NAK	37 %	53 5
6 ACK	22 SYN	38 &	54 6
7 BEL	23 ETB	39 '	55 7
8 BS	24 CAN	40 (	56 8
9 HT	25 EM	41)	57 9
10 LF	26 SUB	42 *	58 :
11 VT	27 ESC	43 +	59 ;
12 FF	28 FS	44 ,	60 <
13 CR	29 GS	45 -	61 =
14 SO	30 RS	46 .	62 >
15 SI	31 US	47 /	63 ?

ASCII-Code-Tabelle (Fortsetzung)

64 @	80 P	96 '	112 p
65 A	81 Q	97 a	113 q
66 B	82 R	98 b	114 r
67 C	83 S	99 c	115 s
68 D	84 T	100 d	116 t
69 E	85 U	101 e	117 u
70 F	86 V	102 f	118 v
71 G	87 W	103 g	119 w
72 H	88 X	104 h	120 x
73 I	89 Y	105 i	121 y
74 J	90 Z	106 j	122 z
75 K	91 [	107 k	123 {
76 L	92 \	108 l	124 |
77 M	93]	109 m	125 }
78 N	94 ^	110 n	126 ~
79 O	95 _	111 o	127 ■

BASIC-Fehlermeldungen

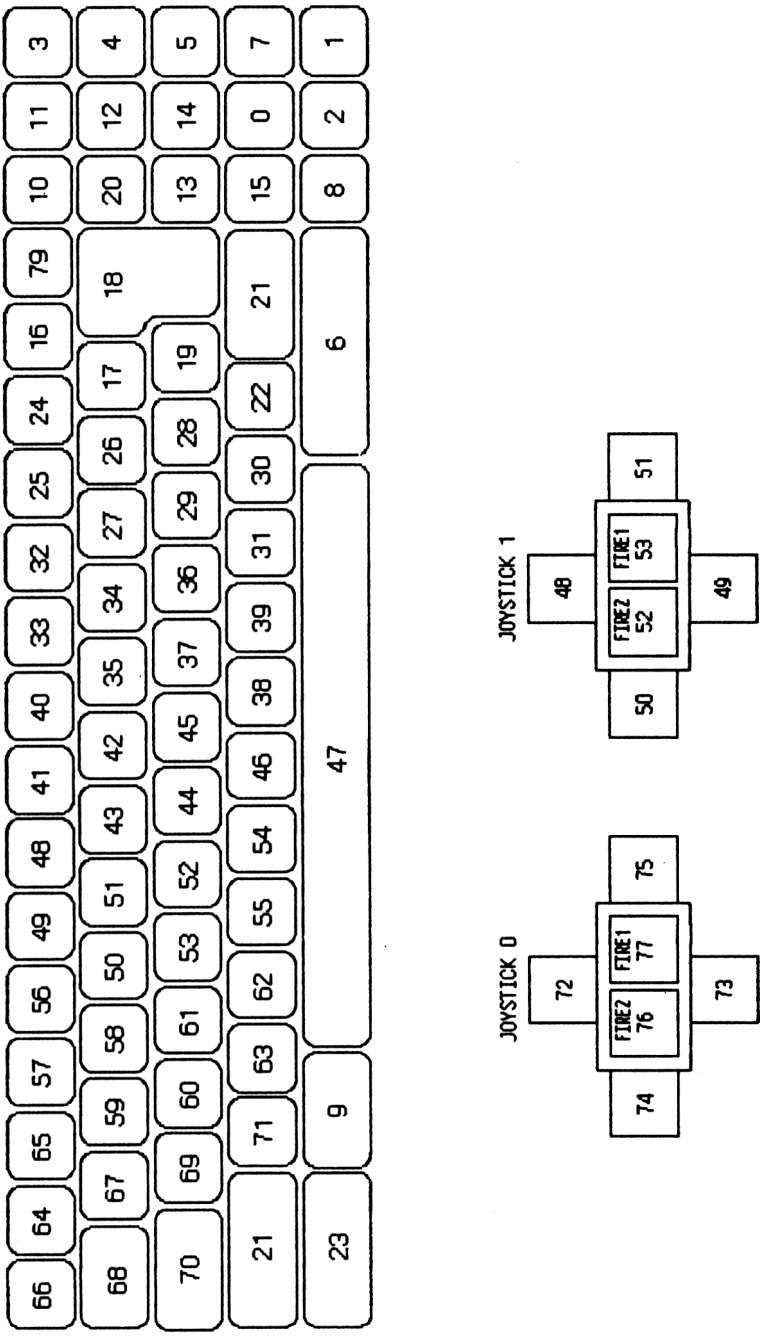
1	Unexpected NEXT	kein FOR für NEXT gefunden
2	Syntax error	Schreibfehler im Programmtext
3	Unexpected RETURN	RETURN ohne GOSUB-Aufruf
4	DATA exhausted	DATA-Tabelle bei READ schon abgearbeitet
5	Improper argument	ungültiges Argument
6	Overflow	zulässiger Zahlenbereich überschritten
7	Memory full	BASIC-Speicher voll
8	Line does not exist	Programmzeile nicht vorhanden
9	Subscript out of range	Aufruf eines nicht definierten Elementes in einem Feld
10	Array already dimensioned	Es wurde versucht, ein Feld zum zweiten Mal zu dimensionieren
11	Division by zero	Division durch Null
12	Invalid direct command	Unzulässige Anweisung im Direktmodus
13	Type mismatch	Falscher Variablentyp
14	String space full	Stringspeicher voll
15	String too long	String enthält mehr als 255 Zeichen
16	String expression too complex	String mit zu komplexer Zusammensetzung
17	Cannot CONTinue	CONT-Anweisung hier nicht zulässig
18	Unknown user function	Unbekannte Anwenderfunktion
19	RESUME missing	RESUME fehlt am Ende einer Fehlerbehandlungsroutine
20	Unexpected RESUME	RESUME ohne Aufruf einer Fehlerbehandlungsroutine
21	Direct command found	Anweisung für Direktmodus im Programm
22	Operand missing	Fehlender Operand
23	Line too long	Eingegebene Programmzeile zu lang

24	EOF met	Dateiende überschritten
25	File type error	Falscher Dateityp
26	NEXT missing	NEXT fehlt in FOR...NEXT-Anweisung
27	File already open	Es wurde versucht, eine Datei zum zweiten Mal zu öffnen
28	Unknown command	Unbekannte Anweisung
29	WEND missing	WEND fehlt in WHILE...WEND Anweisung
30	Unexpected WEND	WEND ohne WHILE angetroffen
31	File not open	Datei nicht geöffnet
32	Broken in	Programmabbruch in Zeile ...

Floppy-Fehlermeldungen (AMSDOS) in DERR

0 oder 22	ESC-Taste gedrückt
142	Ein-/Ausgabestream nicht in Ordnung
143	Dateiende erreicht (Hard End)
144	Anweisung oder Dateiname falsch
145	Datei existiert bereits
146	Datei existiert nicht
147	Directory voll
148	Diskette voll
149	Diskettenwechsel bei geöffneter Datei
150	Versuch, in eine schreibgeschützte Datei zu schreiben
151	Dateiende erreicht (Soft End)

Tastaturcodes



Index

Weitere Fachbücher aus unserem Verlagsprogramm

COMMODORE

Das Commodore 128-Handbuch

Juli 1985, 383 Seiten

In diesem Buch finden Sie einen Querschnitt durch alle wichtigen Funktions- und Anwendungsbereiche des Commodore 128. Sie werden mit dem C64/C128-Modus und der Benutzung von CP/M 3.0 vertraut gemacht, erfahren alles über die Grafik- und Soundmöglichkeiten des C128, lernen die Techniken der Speicherverwaltung und das Banking kennen und werden in die Programmierung mit Assemblersprache sowie die Grafikprogrammierung des 80-Zeichen-Bildschirms eingeführt. Ein umfassendes Handbuch, das Sie immer griffbereit haben sollten!

Best.-Nr. MT 90195, ISBN 3-89090-195-9
(sFr. 47,80/öS 405,60) **DM 52,–**

BASIC 7.0 auf dem Commodore 128

Juli 1985, 239 Seiten

Ganz gleich, ob Sie bereits über Programmierkenntnisse verfügen oder nicht, dieses Buch wird Ihnen helfen, den größtmöglichen Nutzen aus dem leistungsstarken BASIC 7.0 des Commodore 128PC zu ziehen. Sie eignen sich bei der Durcharbeitung dieses Buches alle notwendigen Kenntnisse an, um immer anspruchsvollere Aufgabenstellungen zu bewältigen: Listenverarbeitung, indexsequentielle Dateiverwaltung, Grafikdarstellungen und Sounderzeugung. Ein unentbehrliches Lehrbuch, das sich auch für den geübten Anwender als Nachschlagewerk eignet.

Best.-Nr. MT 808, ISBN 3-89090-170-0
(sFr. 47,80/öS 405,60) **DM 52,–**

WordStar 3.0 mit MailMerge für den Commodore 128 PC

November 1985, 435 Seiten

WordStar ist ein umfangreiches und leistungsfähiges Textverarbeitungsprogramm und damit sicherlich zu Recht das meistverkaufte Programm seiner Art. Doch bedeutet dies nicht unbedingt, daß es auch einfach zu bedienen ist. Hier setzt dieses Buch an: Es macht in vorbildlicher Weise mit allen Möglichkeiten von WordStar und MailMerge vertraut und ist damit eine ideale Ergänzung zum Handbuch. Es versammelt alle wichtigen Informationen für den effektiven Einsatz dieser Programme auf dem Commodore 128 PC.

Best.-Nr. MT 780, ISBN 3-89090-181-6
(sFr. 45,10/öS 382,20) **DM 49,–**

dBASE II für den Commodore 128 PC

November 1985, 280 Seiten

Das vorliegende Buch gibt nach einer kurzen Einführung in den Komplex »Datenbanken« eine Anleitung für den praktischen Umgang mit dBASE II. Schon nach Beherrschung weniger Befehle ist der Anwender in der Lage, Dateien zu erstellen, mit Informationen zu laden und auszuwerten. Dabei hilft ihm ein integrierter Reportgenerator, der im Dialog mit dem Benutzer Berichte gestaltet und in Tabellenform ausdruckt.

Best.-Nr. MT 838, ISBN 3-89090-189-1
(sFr. 45,10/öS 382,20) **DM 49,–**

Multiplan für den Commodore 128 PC

November 1985, 226 Seiten

MULTIPLAN wurde ursprünglich für das 16-Bit-Betriebssystem MS-DOS entwickelt. Inzwischen ist aber auch die in diesem Buch beschriebene CP/M-Version für den Commodore 128 PC auf dem Markt, die den vollen Leistungsumfang der 16-Bit-Version enthält.

Das vorliegende Buch soll eine praktische Einführung in den Umgang mit MULTIPLAN auf dem Commodore 128 PC geben. Anhand von praxisnahen Beispielen werden alle Befehle und Funktionen in der Reihenfolge beschrieben, die der Arbeit in der Praxis entspricht. Bereits nach Abschluß des ersten Kapitels werden Sie in der Lage sein, eigene kleine MULTIPLAN-Anwendungen zu realisieren.

Best.-Nr. MT 836, ISBN 3-89090-187-5
(sFr. 45,10/öS 382,20) **DM 49,–**

Die Floppy 1571

1. Quartal 1986, ca. 400 Seiten

Dieses Buch soll es sowohl dem Einsteiger als auch dem fortgeschrittenen Programmierer ermöglichen, die vielfältigen Möglichkeiten dieses neuen Gerätes voll auszuschöpfen. Sämtliche Betriebsarten und Diskettenformate werden ausführlich erläutert. Anhand vieler Beispiele werden Sie in die Dateiverwaltung mit dieser Floppy eingeführt. Der Benutzer lernt die zahlreichen Systembefehle kennen und erfährt zugleich wichtige Grundlagen für das Arbeiten mit dem Betriebssystem CP/M.

Best.-Nr. MT 90185, ISBN 3-89090-185-9
(sFr. 47,80/öS 405,60) **DM 52,–**

C 64 Fischertechnik Messen, Steuern, Regeln

1. Quartal 1986, ca. 200 Seiten

Ziel dieses Buches ist es, jedem Besitzer eines Commodore 64/VC20 eine neue Welt zu erschließen: die Welt der Roboter, der computergesteuerten Fertigungsstraßen. Alles, was Sie benötigen, ist einer der beiden genannten Computer und der Fischertechnik Computing Baukasten mit dazugehörigem Interface.

Best.-Nr. MT 90194, ISBN 3-89090-194-8
(sFr. 27,60/öS 233,20) **DM 29,90**

Mini-CAD mit Hi-Eddi-Plus

Dezember 1985, 234 Seiten inkl. Diskette

Neben den »Standardbefehlen« zum Setzen und Löschen von Punkten, dem Zeichnen von Linien, Kreisen und Rechtecken sowie dem Ausfüllen unregelmäßiger Flächen und dem Verschieben und Duplizieren von Bildschirmbereichen bietet Hi-Eddi eine Reihe von Besonderheiten, die dieses Programm von anderen Grafikprogrammen abhebt: bis zu sieben Grafikbildschirme stehen gleichzeitig zur Verfügung; es besteht die Möglichkeit, Text in die Grafik einzufügen, die Bildschirme zu verknüpfen oder in schneller Folge durchzuschalten.

Best.-Nr. MT 736, ISBN 3-89090-136-0
(sFr. 44,20/öS 374,40) **DM 48,–**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

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

Weitere Fachbücher aus unserem Verlagsprogramm

BASIC-Programmierung PC-10/PC-20
Oktober 1985, ca. 500 Seiten
Ein amerikanisch-lockerer BASIC-Kurs von dem kalifornischen Professor Lien. Durch seine Systematik ideal als Kursunterlage für PC-10/20 und Kompatible. Mit Einführung in das PC-10-System und Tastendarstellung im Text.
Best.-Nr. PW 559, ISBN 3-921803-66-7
(sFr. 54,30/öS 460,20) **DM 59,—**

C64 - Wunderland der Grafik
Juli 1985, 236 Seiten inklusive Beispieldiskette
Dieses Buch zeigt eine Vielzahl sehr interessanter Lösungen, um die grafischen Möglichkeiten des Commodore 64 optimal zu nutzen. Als Krönung enthält es ein zuschaltbares Assemblerprogramm, das umfangreiche grafische und einige neue BASIC-Befehle anbietet. Im zweiten Teil des Buches wird eine Möglichkeit gezeigt, wie man bis zu 70 verschiedene Farben erzeugen kann. Viele Beispielprogramme begleiten die Reise durch das Wunderland der Grafik.
Best.-Nr. MT 756, ISBN 3-89090-130-1
(sFr. 45,10/öS 382,20) **DM 49,—**

Das C64-Profihandbuch
Juli 1985, 410 Seiten
Ein Buch, das alle wichtigen Informationen für professionelle Anwendungen mit dem C64 enthält. Mit allgemeinen Algorithmen, die auch auf andere Rechner übertragbar sind, und vielen Utilities, getrennt nach BASIC- und Maschinenprogrammen. Besonders nützlich: erweiterte PEEK- und POKE-Funktionen.
Best.-Nr. MT 749, ISBN 3-89090-110-7
(sFr. 47,80/öS 405,60) **DM 52,—**

Programmieren unter CP/M mit dem C64
Juni 1985, 290 Seiten
Wenn Sie wissen wollen, wie das Betriebssystem CP/M 2.2 auf dem C64 implementiert ist, außerdem einiges über Turbo-Pascal, Nevada-Fortran, MBASIC-80 erfahren wollen, dann ist dieses Buch genau richtig für Sie! Mit Schaltplänen zur eigenen Fertigung des CP/M-Moduls. Für eingefleischte C64-Profis.
Best.-Nr. MT 751, ISBN 3-89090-091-7
(sFr. 47,80/öS 405,60) **DM 52,—**

C64 - Programmieren in Maschinensprache
August 1985, 327 Seiten inklusive Beispieldiskette
In diesem Buch finden Sie über 100 Beispiele zur Assembler-Programmierung mit viel Kommentar und Hintergrundinformationen: das Schreiben von Maschinenprogrammen · Rechnen und Texten mit vorhandenen Routinen · Bedienung von Drucker und Floppy · wie man BASIC- und Maschinenprogramme verknüpft · Erstellen von eigenen Befehlen in Modulform. Für Profis!
Best.-Nr. MT 830, ISBN 3-89090-168-9
(sFr. 47,80/öS 405,60) **DM 52,—**

Einführungskurs: Commodore 64
Mai 1984, 276 Seiten
Die Programmiersprache BASIC · Einsatzgebiete des Commodore 64-BASIC: Grafik, Musik, Dateiverwaltung · mit vielen Beispielprogrammen, häufig benötigten Tabellen und nützlichen Tips · für Einsteiger und Fortgeschrittene.
Best.-Nr. MT 685, ISBN 3-89090-017-8
(sFr. 35,—/öS 296,40) **DM 38,—**

Commodore 64 - leicht verständlich
Juni 1984, 154 Seiten
Informationen für den Computer-Neuling · Installation und Inbetriebnahme · Programmieren in BASIC · Grafik und Töne · Auswahl von Hardware und Zubehör · Software für Ihren Computer · die ideale Einführung in das Arbeiten mit Ihrem Commodore 64.
Best.-Nr. MT 700, ISBN 3-89090-022-4
(sFr. 27,50/öS 232,40) **DM 29,80**

Das Commodore 64-LOGO-Arbeitsbuch
September 1984, 225 Seiten
Kinder lernen auf dem Commodore 64 mit der Schildkröte als Lehrer: Bilder malen · Grafikeffekte erzeugen · Wörter verarbeiten · Prozeduren und Variablen · Umgang mit Begriffen wie: Längenmaß, Winkel, Dreieck, Quadrat.
Best.-Nr. MT 720, ISBN 3-89090-063-1
(sFr. 31,30/öS 265,20) **DM 34,—**

Der sensible Commodore 64
Januar 1985, 130 Seiten
Eine Softwaresammlung zu den technologischen Neuerscheinungen im Commodore 64 · für Erstbenutzer wie für Experten ein Buch zur optimalen Softwarenutzung.
Best.-Nr. PW 727, ISBN 3-921803-45-4
(sFr. 27,50/öS 232,40) **DM 29,80**

35 ausgesuchte Spiele für Ihren Commodore 64
September 1984, 141 Seiten
Programmieren Sie selbst 35 faszinierende Spiele · geschrieben in Commodore 64-BASIC · mit Farbe, Grafiken und Ton · Vorschläge zur Programmabwandlung · für kreative Computerfans, die Ihre Programmierkenntnisse vertiefen wollen!
Best.-Nr. MT 774, ISBN 3-89090-064-X
(sFr. 23,—/öS 193,40) **DM 24,80**

BASIC mit dem Commodore 64
April 1984, 320 Seiten
Ein BASIC-Lehrbuch für den jugendlichen Anfänger · übersichtlich gegliederte Lernprogramme · Alles über INPUT-GOTO · Let-Befehle · Editorfunktionen · POKE-Befehle für die Grafik · geeignet auch als Leitfaden für Lehrer und Eltern.
Best.-Nr. MT 657, ISBN 3-922120-91-1
(sFr. 44,20/öS 374,40) **DM 48,—**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt&Technik-Bücher bei Ihrem Buchhändler

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

Weitere Fachbücher aus unserem Verlagsprogramm

Das große Spielebuch - Commodore 64

Februar 1984, 141 Seiten

46 Spielprogramme · Wissenswertes über Programmiertechnik · praxisnahe Hinweise zur Grafikherstellung · alles über Joystick- und Paddleansteuerung · das Spielebuch mit Lerneffekt.

Best.-Nr. MT 603, ISBN 3-922120-63-6
(sFr. 27,50/öS 232,40) **DM 29,80**
Best.-Nr. MT 604 (Beispiele auf Diskette)
(sFr. 38,—/öS 342,—) **DM 38,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Spiele für den Commodore 64

November 1984, 196 Seiten

Bewährte alte und raffinierte neue Spiele für Ihren Commodore 64 · klar und übersichtlich gegliederte Programme im Commodore-BASIC · Sie lernen: wie man Unterprogramme einsetzt · eine Tabelle aufbauen und verarbeiten · Programme testen · mit vielen Programmiertricks · für Anfänger.

Best.-Nr. MT 90074, ISBN 3-89090-074-7
(sFr. 23,—/öS 193,40) **DM 24,80**
Best.-Nr. MT 795 (Beispiele auf Diskette)
(sFr. 38,—/öS 342,—) **DM 38,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Grafik & Musik auf dem Commodore 64

Oktober 1984, 336 Seiten

68 gut strukturierte und kommentierte Beispielprogramme zur Erzeugung von Sprites und Klangeffekten · Sprite-Tricks · Zeichengrafik · hochauflösende Grafik · Musik nach Noten · spezielle Klangeffekte · Ton und Grafik · für fortgeschrittene Anfänger, die alle Möglichkeiten des C64 ausnutzen wollen.

Best.-Nr. MT 743, ISBN 3-89090-033-X
(sFr. 35,—/öS 296,40) **DM 38,—**

Commodore 64 Listings - Band 1: Spiele

Oktober 1984, 199 Seiten

Mit ausführlicher Dokumentation · Spielanleitung · Variablen für die Änderung der Spiele · vollständige Listings für: Bürger Joe · Nibbler · Zingel Zangel · Universe · Würfelpoker · Maze-Mission · der magische Kreis · Todeskommando Atlantik · Enterprise.

Best.-Nr. MT 748, ISBN 3-89090-068-2
(sFr. 23,—/öS 193,40) **DM 24,80**
Best.-Nr. MT 804 (Beispiele auf Diskette)
(sFr. 38,—/öS 342,—) **DM 38,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Commodore 64 Listings
Band 2: Dateiverwaltung · Schule · Hobby

Oktober 1984, 179 Seiten

Ein Buch mit Programmen für die ganze Familie · DATAVE - Eine Dateiverwaltung · mathematische Funktionen · Konjugation und Deklination in Latein · Regressionsanalyse · Bundesligatabelle.

Best.-Nr. MT 766, ISBN 3-89090-071-2
(sFr. 23,—/öS 193,40) **DM 24,80**

Commodore 64 - Multiplan

1984, 230 Seiten

Multiplan jetzt auch für den Commodore 64 · der volle Leistungsumfang der 16-Bit-Version · Einführung in die Arbeitsweise von Tabellenkalkulationsprogrammen · praxisnahe Beispiele · Beschreibung aller Befehle und Funktionen. Ein Buch nicht nur für Anfänger.

Best.-Nr. MT 655, ISBN 3-922120-89-X
(sFr. 44,20/öS 374,40) **DM 48,—**

Das Commodore 64-Buch, Bd. 1: Ein Leitfaden für Erstanwender

Mai 1984, 270 Seiten

Der Commodore 64 und seine Handhabung · Einführung in die Grafik · Balkendiagramme · Einführung in die Spritetechnik · BASIC-Erweiterungen in Assembler · Ein Leitfaden für Erstanwender, die sich bereits BASIC-Kenntnisse angeeignet haben. Alle Beispiele auf Diskette erhältlich!

Best.-Nr. MT 591, ISBN 3-922120-61-X
(sFr. 44,20/öS 374,40) **DM 48,—**
Best.-Nr. MT 592 (Beispiele auf Diskette)
(sFr. 58,—/öS 522,—) **DM 58,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Das Commodore 64-Buch, Bd. 2: BASIC-Spiele

Mai 1984, 181 Seiten

Spiele nicht nur zum Abtippen · Programmlisting · Programmbeschreibung · Variablenübersicht · Programme nach Anleitung frei ergänzbar · das ideale Buch, um Programmieren spielend zu lernen · für Anfänger.

Best.-Nr. MT 593, ISBN 3-922120-68-7
(sFr. 35,—/öS 296,40) **DM 38,—**
Best.-Nr. MT 594 (Beispiele auf Diskette)
(sFr. 58,—/öS 522,—) **DM 58,—***

*** inkl. MwSt. Unverbindliche Preisempfehlung.**

Die angegebenen Preise sind Ladenpreise

Sie erhalten Markt & Technik-Bücher bei Ihrem Buchhändler

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

Der Schneider CPC 6128

JÜRGEN HÜCKSTÄDT
Jahrgang 1949, ist Diplomingenieur für Wasserbau. Im Rahmen seiner langjährigen Tätigkeit in einem Ingenieurbüro kam er schon frühzeitig mit Personal Computern in Berührung. Er entwickelte verschiedene Softwarepakete für wasserwirtschaftliche und geodätische Anwendungen und wurde schließlich mit der Leitung der EDV-Abteilung beauftragt. Seit einigen Jahren leitet er Seminare für BASIC- und Assemblerprogrammierung und ist darüber hinaus auch als Übersetzer, Lektor und Autor von EDV-Fachliteratur tätig.

Der CPC 6128 ist ein ausgereifter Personal Computer für den privaten und professionellen Bereich, der ebenso wie seine Vorgängermodelle CPC 464/664 auf dem besten Weg ist, einen bedeutenden Marktanteil zu erreichen.
Dieses Buch ist für jeden CPC 6128-Besitzer eine wertvolle Hilfe, die vielfachen Möglichkeiten dieses bisher einmaligen Computers kennenzulernen und anzuwenden. Dabei spielt es keine Rolle, ob Sie bereits über Programmiererfahrung verfügen oder nicht. Der Computerneuling wird Schritt für Schritt in den Umgang mit dem Computer und die BASIC-Programmierung eingeführt, bis er alle notwendigen Kenntnisse besitzt, die mancher Profi bereits mitbringt. Aber an dieser Stelle wird das Programmieren mit dem CPC 6128 erst interessant, nämlich dann, wenn es darum geht, eine eigene Dateiverwaltung aufzubauen oder Grafik und Sound zu programmieren. Als ganz besonderen Leckerbissen zeigt Ihnen dieses Buch, daß es doch möglich ist, auch mit Schneider-Computern eine echte relative Dateiverwaltung aufzubauen, die Ihnen das Tor zur professionellen Datenverarbeitung öffnet.
Weiterhin erfahren Sie alles über CP/M Plus auf dem CPC 6128, dem universellen Betriebssystem für die größte Softwarebibliothek der Welt. Dadurch stehen Ihnen zusätzliche ungeahnte Möglichkeiten zur Verfügung.
Ein unentbehrliches Lehrbuch und Nachschlagewerk für jeden, der mit dem CPC 6128 arbeitet.

ISB N 3-89090-192-1

4 001057 901926

Markt&Technik
Verlag Aktiengesellschaft

DM 46,-
sFr. 42,30
öS 358,80

Hückstädt

Der Schneider CPC 6128

M&T

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/