Baloui

DAS BASIC-BUCH

zu

C 16
C 116
Plus/4

EIN DATA BECKER BUCH

Baloui

DAS BASIC-BUCH

zu

C 16
C 116
Plus/4

EIN DATA BECKER BUCH

ISBN 3-89011-204-8

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

Vorwort

Sehr geehrter Leser,

dieses Buch wendet sich vor allem an Einsteiger, die mit einem C16, C116 oder Plus/4 erste Schritte in die Welt der EDV wagen. Prinzipiell gibt es zwei verschiedene Methoden zum Einsatz eines C16/Plus4:

1. Sie verwenden ausschließlich fertige Programme, die im Fachhandel erhältlich sind (Textverarbeitung, Dateiverwaltung, Malprogramm etc.). In diesem Fall werden Sie dieses Buch kaum benötigen, da zum Einsatz dieser Programme die zugehörige Programmanleitung und das Handbuch Ihres Heimcomputers vollauf genügen.

2. Sie wählen den schwierigeren Weg, erlernen eine Programmiersprache wie zum Beispiel BASIC und schreiben eigene Programme. Dieser Weg hat mehrere Vorteile:

   - Sie verlieren die Scheu vor dem "schwarzen Kasten", als der ein Computer oftmals empfunden wird. Das Erlernen einer Programmiersprache bewirkt ein weitaus besseres Verständnis der rechnerinternen Abläufe als die Bedienung eines kommerziellen Programms. Wenn Sie ein erworbenes Programm benutzen, zum Beispiel eine Adressenverwaltung, stellen Sie womöglich fest, daß ein Druck auf die Taste F1 eine Adresse löscht oder daß die Adressen mit der Taste F2 alphabetisch sortiert werden können. Welche Vorgänge zum Löschen oder Sortieren von Adressen notwendig sind, werden Sie jedoch erst verstehen, wenn Sie selbst eine solche Adressenverwaltung erstellt haben.

   - Oftmals ist die Rede von sogenannten "Personal Computern", das heißt persönlichen Computern, die individuelle Ansprüche des jeweiligen Besitzers erfüllen sollen. Dieser Anspruch vom "persönlichen" Computer ist meist völlig übertrieben. In der Praxis kaufen Sie

eine Adressenverwaltung, mit der Sie private oder geschäftliche Daten verwalten wollen und stellen anschließend fest, daß das erworbene Programm für Sie persönlich leider ungeeignet ist, da es nicht auf Ihre individuellen Ansprüche und Bedürfnisse zugeschnitten ist. Der Unterschied zwischen einer erworbenen und einer selbst erstellten Adressenverwaltung entspricht in etwa jenem zwischen einem Anzug "von der Stange" und einem Maßanzug.

- Außer dem Kennenlernen des Computers und der Möglichkeit, wirklich individuelle Programme zu erstellen, spricht ein sehr gewichtiger Punkt für das Erlernen einer Programmiersprache: der Spaß am Programmieren. Es ist nicht übertrieben, wenn ich behaupte, daß Programmieren zur Sucht werden kann. Es ist eine ungeheuere Herausforderung, den Computer zu "beherrschen" und nach Tagen oder manchmal auch erst nach Wochen ein umfangreiches Programm zur einwandfreien Funktionsfähigkeit zu bringen.

In diesem Buch werde ich versuchen, vor allem diesen Spaß am Programmieren zu fördern. Es ist kein theoretisches Lehrbuch, sondern soweit wie möglich praxisorientiert. Sie werden die Programmiersprache BASIC erlernen, die es Ihnen erlaubt, alle Möglichkeiten Ihres C16 oder Plus/4 auszunutzen.

Den Schwerpunkt dieses Buches bildet ein "BASIC-Kurs". In diesem Kurs werden keineswegs alle BASIC-Befehle erläutert. Den Beginn des Kurses bildet eine Erläuterung der wichtigsten Grundlagen (Was ist ein Programm? Was passiert bei der Bearbeitung eines Programms?). Anschließend werden BASIC-Befehle behandelt, die in jedem Programm zur Eingabe, Ausgabe und Verarbeitung von Daten benötigt werden (PRINT, INPUT, LET etc.).

Nachdem die Voraussetzungen geschaffen wurden, werden die verschiedenen BASIC-Befehlsgruppen (Graphik, Musik, Dateiverwaltung etc.) anhand von Problemstellungen erörtert. So werden zum Beispiel die am häufigsten verwendeten Graphik-

befehle des C16, C116 bzw. Plus/4 an einem Malprogramm dargestellt, das Schritt für Schritt erarbeitet wird. Im Kapitel über Dateiverwaltung mit Cassette und Diskette werden die benötigten Befehle nach und nach bei der Entwicklung einer Adressenverwaltung eingeführt.

Nach Abschluß des BASIC-Kurses kennen Sie keineswegs alle Befehle, die Ihnen der C16, C116 oder Plus/4 zur Verfügung stellt. Sie wissen jedoch, wie Probleme analysiert, "Algorithmen" entwickelt und in Programme umgesetzt werden.

Auf dem Kurs aufbauend sollten Sie versuchen, eigene Problemstellungen mit Programmen zu lösen. Ab und zu werden Sie Befehle benötigen, die im Kurs nicht behandelt wurden. An den Kurs schließt sich eine Befehlsübersicht an, in der alle BASIC-Befehle Ihres Heimcomputers mit Kurzbeispielen vorgestellt werden. Ziel des Kurses ist es, Sie mit der grundlegenden Arbeitsweise und der Syntax der BASIC-Befehle so vertraut zu machen, daß Sie in der Lage sind, sich mit Hilfe der Übersicht zusätzlich benötigte Befehle eigenständig anzueignen.

Alle in diesem Buch entwickelten Programme (Malprogramm, Adressenverwaltung, Zinsberechnung etc.) sind gleichermaßen auf dem C16, dem C116 und dem Plus/4 lauffähig.

Eine Warnung möchte ich Ihnen noch mitgeben: Sollte meine Befürchtung richtig sein und die "Computerei" nach der Lektüre dieses Buches bei Ihnen zur Sucht ausarten, dann machen Sie mich bitte nicht für familiäre Probleme verantwortlich, wenn Sie mitten in der Nacht vor Ihrem C16, C116 oder Plus/4 sitzen, um noch "schnell" ein letztes Problem zu lösen.

Ludwigshafen, Juli 1986 Said Baloui

Inhaltsverzeichnis

1. Einleitung

Dieses Buch ist gleichermaßen für Besitzer eines C16, C116 und eines Plus/4 geeignet. Wie aus dem Titel hervorgeht, beschäftigt es sich mit der auf diesen Heimcomputern verwendeten Programmiersprache BASIC. Diese Sprache existiert in verschiedenen Versionen, vergleichbar den Dialekten einer natürlichen Sprache ("Bayerisch", "Kölsch" etc.). Der C16, C116 und Plus/4 verfügen über die gleiche Version dieser Sprache.

Der wichtigste Unterschied zwischen C16 und Plus4 besteht in der Speicherkapazität. Auf diese unterschiedliche Kapazität und ihre Auswirkungen auf Programmgröße und Datenmengen wird bei der Besprechung der Arrayvariablen und im Graphik-Kapitel eingegangen.

Bevor wir uns mit BASIC beschäftigen, werde ich zuerst allgemeine Grundlagen im Umgang mit C16, C116 oder Plus/4 erläutern, wie die Tastatur und den Editor. Sollten Sie der Meinung sein, daß Sie den Umgang damit bereits beherrschen, dürfen Sie dieses Kapitel ruhigen Gewissens überblättern.

Anschließend folgt der Hauptteil dieses Buches, der eigentliche BASIC-Kurs. Sie werden lernen, wie Sie beliebige Zeichen auf dem Bildschirm ausgeben können, was ein Programm ist, wie ein BASIC-Programm aufgebaut ist und welche Befehlsgruppen zur Lösung spezieller Probleme vorhanden sind.

Sollten Sie bereits mit BASIC ein wenig vertraut und an der praktischen Anwendung bestimmter Befehlsgruppen besonders interessiert sein, ist die folgende Kurzübersicht für Sie interessant.

- Im einführenden Teil wird nach der Behandlung von Tastatur und Editor vorwiegend auf den Umgang mit den Ein- und Ausgabebefehlen und auf die Verwendung von Variablen eingegangen.

- Im Graphik- und im Musikteil werden fast ausschließlich spezielle Graphik- beziehungsweise Musik-Befehle erläutert.

- Im Kapitel über Dateiverwaltung finden Sie außer der Besprechung der entsprechenden Datei-Befehle eine Erläuterung wichtiger String-Funktionen und reservierter Variablen.

Im Anschluß an den BASIC-Kurs werden die verschiedenen Befehlsgruppen dieser Sprache erläutert, die Rechenfunktionen, die Graphik- und Musikbefehle, Befehle zur Dateibehandlung mit Cassette und Diskette und so weiter.

In der Befehlsübersicht werden auch die verschiedenen "Toolkit-Funktionen" erläutert, die zum Programmieren zwar nicht "lebenswichtig", doch in vielen Fällen sehr hilfreich sind.

Alle Befehle werden auch im Handbuch zum C16, C116 bzw. Plus/4 beschrieben, so daß Sie sich vielleicht fragen werden, was eine erneute Aufzählung soll. Im Handbuch liegt der Schwerpunkt auf der theoretischen Erläuterung der Befehle, die Beispiele dazu fallen recht knapp aus.

In der Befehlsübersicht in diesem Buch wird umgekehrt vorgegangen. Die Erläuterungen beschränken sich auf Grundlegendes, umso ausführlicher sind die Beispiele. Schwierig zu verstehende Befehle wie zum Beispiel INSTR werden anhand kleiner Demoprogramme vorgestellt, inclusive aller Auswirkungen, die sich beim Programmlauf aus den Befehlen ergeben. Im Gegensatz zum Handbuch sind die Befehle nicht alphabetisch geordnet, sondern nach Funktionsgruppen zusammengefaßt (Graphik-Befehle, Datei-Befehle etc.).

Die Befehlsübersicht dient nur der Ergänzung. Schwerpunkt dieses Buches ist zweifellos der BASIC-Kurs, der die wichtigsten Befehle Ihres Rechners anhand praktischer Anwendungen erläutert. Als "BASIC-Einsteiger" sollten Sie den Kurs möglichst in der vorgestellten Reihenfolge durcharbeiten, da der Schwierigkeitsgrad allmählich ansteigt.

2. Tastatur und Editor

Im folgenden werde ich kurz auf die Tastatur und den eingebauten "Editor" des C16, C116 und Plus4 eingehen. Dieses Kapitel mag jenen Lesern eine Hilfe sein, die Probleme mit den entsprechenden Erläuterungen im Handbuch haben.

Die Beherrschung von Tastatur und Editor sind absolut notwendige Voraussetzungen zum Umgang und vor allem zur Programmierung Ihres C16, C116 beziehungsweise Plus/4.

2.1 Die Funktionstasten

Wenn Sie mit der Tastatur "herumspielen", werden Sie sicherlich manche Merkwürdigkeit erleben. Wenn Sie bereits Schreibmaschinenerfahrung besitzen, werden Sie problemlos mit der Buchstaben- und Zifferntastatur umgehen können.

Dennoch werden plötzlich unerklärliche Meldungen wie zum Beispiel "Syntax error", "Dload", "Dsave" etc. auf dem Bildschirm erscheinen. Ursache für diese seltsamen Meldungen sind häufig die sogenannten "Funktionstasten". Diese Funktionstasten sind mit F1, F2,...,F7 und HELP beschriftet. Wenn Sie eine dieser Funktionstasten betätigen, erscheint eine komplette Zeichenkette auf dem Bildschirm, ein "Befehl".

Grundsätzlich sollten Sie sich merken, daß Ihr C16, C116 oder Plus/4 nach dem Einschalten auf Befehle wartet, die Sie ihm geben. Diese Befehle sind Worte der Programmiersprache BASIC und werden mit der Tastatur eingegeben. Normalerweise wird ein Befehlswort Zeichen für Zeichen mit Hilfe der Buchstabentasten eingetippt. Eine Ausnahme stellen die erwähnten Funktionstasten dar. Manche BASIC-Befehle wie zum Beispiel RUN oder LIST werden sehr häufig eingegeben. Die Funktionstasten sind mit solchen häufig verwendeten Befehlsworten "belegt", das heißt eine Betätigung der jeweiligen Funktionstaste genügt und das komplette Befehlswort erscheint auf dem Bildschirm. Vor-

läufig werden wir uns jedoch auf die zeichenweise Eingabe von Befehlen mit den Buchstabentasten beschränken.

Außer den Funktionstasten, den Buchstabentasten, den Zifferntasten und den Sonderzeichentasten ("!", "$", "%", ...) sind mehrere Tasten mit Sonderfunktionen vorhanden, die ich im folgenden besprechen werde.

2.2 Die Cursortasten

Sowohl auf dem C16 und C116 als auch auf dem Plus/4 befinden sich jeweils vier Tasten, auf denen Pfeile abgebildet sind, die sogenannten "Cursortasten". Beim C16 und C116 finden Sie diese Cursortasten in der obersten Reihe der Tastatur, beim Plus/4 ist ein separater "Cursorblock" vorhanden.

Der "Cursor" ist eine Markierung auf dem Bildschirm, die Ihnen Ihre momentane Eingabeposition angibt. Wenn Sie ein beliebiges Zeichen eingeben, erscheint dieses exakt an jener Stelle, an der sich der blinkende Cursor befindet, und dieser wandert eine Position nach rechts.

Mit den vier Cursortasten können Sie sich beliebig auf dem Bildschirm umherbewegen, wobei Sie nur durch die Bildschirmgrenzen eingeschränkt werden.

Wenn Sie den Cursor in die unterste Bildschirmzeile bewegen und die Taste, die den Cursor eine Zeile nach unten bewegt, ein weiteres Mal betätigen, werden Sie feststellen, daß sich der komplette Bildschirm um eine Zeile nach oben verschiebt, wobei die oberste Bildschirmzeile verschwindet.

Um dieses Phänomen, das "Scrolling" genannt wird, zu verstehen, stellen Sie sich den Bildschirm am besten als eine Papierrolle vor, von der Sie immer nur einen kleinen Ausschnitt sehen. Wenn Sie diese Rolle beschreiben und am unteren Ende des Ausschnitts angelangt sind, wird die Rolle ein Stück weitergedreht, damit Sie weiterschreiben können.

2.3 Die SHIFT-Taste

Die Taste SHIFT existiert zweimal auf der Tastatur. Ihre Funktion ist wie bei der Schreibmaschine die Umschaltung des "Schreibmodus". Bei der Schreibmaschine wird durch die gleichzeitige Betätigung von SHIFT und einer Buchstabentaste ein Großbuchstabe ausgegeben, ohne SHIFT ein Kleinbuchstabe.

Nach dem Einschalten Ihres Rechners ist der "Großchreibmodus" jedoch bereits eingeschaltet. Auch ohne gleichzeitige Betätigung von SHIFT werden Großbuchstaben erzeugt. Der Grund besteht darin, daß der C16, C116 oder Plus4 zwei grundverschiedene Schreibarten besitzt, den "Klein/Großmodus" und den "Groß/Graphikmodus".

Nach dem Einschalten befindet sich Ihr Computer im "Groß/Graphikmodus". In diesem Modus werden werden bereits ohne gleichzeitige Betätigung von SHIFT Großbuchstaben erzeugt. Was bei gleichzeitiger Betätigung von SHIFT passiert, probieren Sie am besten selbst aus. Drücken Sie gleichzeitig SHIFT und die Taste A. Wie Sie sehen, erscheint das Zeichen für "Pik" auf dem Bildschirm. Bei gleichzeitiger Betätigung von SHIFT und Z erhalten Sie ein "Karo".

Auf allen Buchstabentasten sind jeweils zwei "Graphikzeichen" aufgemalt. Wie Sie sehen, erscheint bei Betätigung einer dieser Buchstabentasten zusammen mit SHIFT das jeweils rechte Graphikzeichen der betreffenden Taste.

Mit diesen Graphikzeichen und den Cursortasten können Sie nun zum Beispiel einen Rahmen aus waagrechten und senkrechten Linien auf den Bildschirm zeichnen. Fehler, die beim Zeichnen entstehen, können Sie korrigieren, indem Sie sich mit den Cursortasten zur betreffenden Position bewegen und das Zeichen einfach überschreiben.

2.4 Die SHIFT LOCK-Taste

Die Taste SHIFT LOCK rastet ein, wenn sie gedrückt wird. Bei eingerasteter SHIFT LOCK-Taste verhält sich der C16, C116 oder Plus/4 bei jedem Tastendruck so, als ob Sie gleichzeitig die Taste SHIFT betätigen würden. Diese Taste kann zum Beispiel beim Zeichnen von Graphiken verwendet werden. Sie ersparen sich auf diese Weise die ständige Betätigung von SHIFT. Wenn Sie diesen "geshifteten" Schreibmodus verlassen wollen, drücken Sie bitte ein weiteres Mal SHIFT LOCK.

2.5 INST/DEL und CLEAR/HOME

Die Tasten INST/DEL und CLEAR/HOME dienen zur Editierung von Eingaben und zur Cursorsteuerung. Wenn Sie die Taste INST/DEL betätigen, wird das Zeichen links vom Cursor gelöscht (DEL-Funktion) und der Cursor bewegt sich um eine Position nach links. Wenn Sie diese Taste gleichzeitig mit SHIFT betätigen, werden das Zeichen unter dem Cursor und alle Zeichen rechts davon um eine Position nach rechts verschoben. An der Cursorposition befindet sich nun eine Lücke und Sie können ein Zeichen einfügen (INST-Funktion).

Mit der Taste CLEAR/HOME können Sie den Cursor mit einem einzigen Tastendruck auf die linke obere Ecke des Bildschirms positionieren (HOME-Funktion), oder aber, wenn Sie zugleich SHIFT betätigen, den Bildschirm löschen. Nach dem Löschen des Bildschirms befindet sich der Cursor ebenfalls in der linken oberen Bildschirmecke.

2.6 Die COMMODORE-Taste (C=)

Neben der SHIFT-Taste auf der linken Seite der Tastatur befindet sich eine mit "C=" beschriftete Taste, die sogenannte "COMMODORE-Taste". Mit dieser Taste können Sie die zweite Hälfte der vorhandenen Graphikzeichen benutzen.

Erinnern Sie sich: Durch gleichzeitige Betätigung von SHIFT und einer Buchstabentaste erschien das rechte von den beiden Graphikzeichen, das auf der betreffenden Buchstabentaste abgebildet wird. Durch gleichzeitige Betätigung der COMMODORE-Taste und einer Buchstabentaste wird das linke der beiden Graphikzeichen auf dem Bildschirm dargestellt.

Mit SHIFT und der COMMODORE-Taste erreichen Sie auf diese Weise alle Graphikzeichen, über die Ihr C16, C116 beziehungsweise Plus/4 verfügt.

Die COMMODORE-Taste besitzt eine weitere, außerordentlich wichtige Funktion. Wie erwähnt, befindet sich der C16, C116 oder Plus/4 nach dem Einschalten im Groß/Graphikmodus, das heißt alle Buchstaben werden groß dargestellt, und die Betätigung einer Buchstabentaste gleichzeitig mit der Taste SHIFT führt zur Ausgabe von Graphikzeichen.

Betätigen Sie versuchsweise gleichzeitig die COMMODORE-Taste und die Taste SHIFT. Sie werden feststellen, daß alle Großbuchstaben, die sich auf dem Bildschirm befinden, nun in Kleinschreibweise dargestellt werden und aus Graphikzeichen Großbuchstaben werden.

Durch gleichzeitige Betätigung von C= und SHIFT haben Sie den Schreibmodus gewechselt und den Klein/Großmodus eingeschaltet. Wenn Sie nun weiterschreiben, werden ebenso wie bei der Schreibmaschine Buchstaben klein dargestellt, außer Sie betätigen gleichzeitig SHIFT, wodurch Sie Großbuchstaben erhalten. In diesem Modus können Sie die Tastatur daher ähnlich wie die einer Schreibmaschine verwenden und sowohl Klein- als auch Großbuchstaben darstellen.

Der Nachteil dieses Modus ist jedoch, daß Sie nur noch über die Hälfte der vorhandenen Graphikzeichen verfügen. Mit der COMMODORE-Taste können Sie immer noch die jeweils links auf den Buchstabentasten dargestellten Graphikzeichen erreichen. Die Taste SHIFT besitzt nun jedoch nicht mehr die Funktion, die rechts dargestellten Graphikzeichen zu erzeugen,

sondern führt - ebenso wie bei der Schreibmaschine - zur Darstellung von Großbuchstaben.

Ob Sie Ihre Programme im Groß/Graphikmodus oder im Klein/Großmodus schreiben, ist dem C16, C116 oder Plus/4 prinzipiell völlig egal. Es ist Ihnen überlassen, welchen Modus Sie bevorzugen, da alle BASIC-Befehle sowohl groß- (PRINT) als auch kleingeschrieben (print) von Ihrem Computer verstanden werden.

Eine weitere Funktion der COMMODORE-Taste ist die Farbsteuerung. Versuchen Sie bitte folgendes:

Betätigen Sie gleichzeitig die COMMODORE-Taste und die Zifferntaste 6. Sie werden feststellen, daß sich die Farbe des Cursors ändert. Auf einem Schwarzweißfernseher wird sich nur der Grauton ändern, mit einem Farbfernseher können Sie erkennen, daß der Cursor nun hellblau dargestellt wird.

Diese Farbe Hellblau entspricht exakt einer der drei Beschriftungen der Taste 6. Diese Taste ist an der Front mit "Grn" (Green = Grün) und darunter mit "L Blu" (Light Blue = Hellblau) beschriftet. Die Zifferntasten 1, 2, 3, ..., 8 tragen jeweils zwei Farbbeschriftungen. Wenn Sie eine dieser Tasten gleichzeitig mit der COMMODORE-Taste betätigen, erhält der Cursor die Farbe, die der unteren Beschriftung der Taste entspricht. Sie können daher zwischen acht verschiedenen Farben auswählen.

Diese Farbeinstellung bleibt erhalten, bis Sie auf die beschriebene Art und Weise eine andere Farbe anwählen. Alle eingegebenen Zeichen werden in der momentan eingestellten Farbe auf dem Bildschirm ausgegeben.

2.7 Die CONTROL-Taste

Die Taste CONTROL befindet sich ebenso wie SHIFT zweimal auf der Tastatur. Ebenso wie SHIFT und die COMMODORE-Taste ist auch CONTROL nur wirksam bei gleichzeitiger Betätigung einer anderen Taste. CONTROL besitzt drei Funktionen:

1. Farbsteuerung: Auf den Zifferntasten 1-8 befinden sich jeweils zwei Farbbeschriftungen. Die untere dieser Farben kann wie beschrieben mit der COMMODORE-Taste angewählt werden. Die obere der beiden Farben erhalten Sie, wenn Sie gleichzeitig die Taste CONTROL und eine dieser Zifferntasten drücken. Mit den Zifferntasten 1 bis 8 und den Tasten CONTROL und COMMODORE verfügen Sie daher über eine reichhaltige Farbpalette. Ihnen stehen insgesamt 16 Farben zur Verfügung.

2. Inversdarstellung ein/ausschalten: Die Zifferntasten 9 und 0 besitzen eine besondere Bedeutung. Ihre Front ist mit "RVS ON" (Revers On = Inversdarstellung einschalten) beziehungsweise "RVS OFF" (Revers Off = Inversdarstellung ausschalten) beschriftet. Mit dieser Inversdarstellung können Zeichen besonders hervorgehoben werden.

   Drücken Sie bitte gleichzeitig CONTROL und die Zifferntaste 9 und geben Sie anschließend mehrere Zeichen ein. Die eingegebenen Zeichen werden "revers" oder auch "invers" hervorgehoben. Die Zeichen werden in der Farbe des Untergrundes dargestellt und der Untergrund unter den eingegebenen Zeichen erhält die momentan eingestellte Zeichenfarbe.

   Mit CONTROL und der Taste 0 können Sie diese Reversdarstellung wieder ausschalten. Die bereits invers dargestellten Zeichen bleiben unverändert, neu eingegebene Zeichen werden jedoch wie gewohnt dargestellt.

3. Blinken ein/ausschalten: Sie sehen, Ihr Heimcomputer verfügt über eine enorme Vielfalt der Zeichendarstellung. Sie können Graphikzeichen verwenden, Zeichen in 16 verschiedenen Farben darstellen und außerdem Zeichen blinken lassen. Drücken Sie bitte die Tasten CONTROL und FLASH ON. Die Beschriftung FLASH ON befindet sich an der Front der Taste "<". Alle Zeichen, die Sie nun eingeben, werden blinkend dargestellt. Wenn Sie den Blinkmodus wieder verlassen wollen, drücken Sie gleichzeitig CONTROL und die benachbarte Taste FLASH OFF.

2.8 Die ESC-Taste

Die Taste ESC (Escape) besitzt eine unglaubliche Vielfalt von Funktionen. Wirksam wird diese Taste erst in Verbindung mit einer Buchstabentaste. Im Gegensatz zu SHIFT, COMMODORE und CONTROL werden jedoch nicht zwei Tasten gleichzeitig, sondern nacheinander betätigt.

Geben Sie bitte in einer beliebigen Bildschirmzeile mehrer Zeichen ein. Drücken Sie nun ESC, lassen Sie die Taste los und betätigen Sie die Buchstabentaste I. Die Zeile, in der Sie sich befinden (und alle Zeilen darunter) wird um eine Zeile nach unten geschoben und Sie erhalten eine Leerzeile an der momentanen Cursorposition. Wenn Sie mehrere Leerzeilen benötigen, wiederholen Sie den Vorgang mehrmals (ESC drücken, loslassen, I betätigen).

ESC besitzt eine Menge weitere "Editierfunktionen", die Ihnen die Eingabe von Befehlen oder das Malen mit den Graphikzeichen erleichtern. Im folgenden stelle ich eine kleine Tabelle vor, die all jene Funktionen enthält, die Sie "gefahrlos" ausprobieren können. Mehrere andere Funktionen von ESC dürften am Anfang zu einiger Verwirrung bei Ihnen führen und werden daher später besprochen, vor allem der mit der ESC-Taste mögliche Aufbau von Bildschirmfenstern, den "Windows".

ESC + A:	Einfügemodus einschalten
ESC + C:	Einfügemodus ausschalten
ESC + D:	Zeile löschen
ESC + I:	Zeile einfügen
ESC + J:	Cursor an Zeilenanfang setzen
ESC + K:	Cursor an Zeilenende setzen
ESC + P:	Zeile bis Cursorposition löschen
ESC + Q:	Zeile ab Cursorposition löschen

Sie werden beträchtliche Übung benötigen, um alle diese Funktionen ohne Nachschlagen benutzen zu können. Wenn Sie diese Editierfunktionen im Laufe der Zeit beherrschen, werden Sie festellen, daß sie eine erhebliche Arbeitserleichterung bei der Programmierung ermöglichen.

2.9 Die RETURN-Taste

Die RETURN-Taste ist wahrscheinlich die wichtigste Taste überhaupt. Ohne diese Taste könnten Sie Ihrem Computer nicht einen Befehl übermitteln, und zwar aus folgendem Grund:

Programmieren bedeutet, dem Computer eine Folge von Anweisungen zu geben. Diese Anweisungen werden in einer Sprache gegeben, die der Computer versteht, im Falle des C16, C116 oder Plus/4 die Sprache BASIC. Wenn Sie nun mit der Tastatur einen Befehl dieser Sprache eingeben, müssen Sie dem Rechner mitteilen, daß die Eingabe beendet ist, indem Sie die RETURN-Taste betätigen.

Jede (!) Eingabe wird durch Betätigung der RETURN-Taste abgeschlossen. Wenn Sie einen BASIC-Befehl eingeben, wird Ihr C16, C116 oder Plus/4 diesen Befehl erst dann ausführen, wenn Sie RETURN betätigen und ihm damit mitteilen, daß die Eingabe beendet ist.

Daß diese Behauptung richtig ist, erkennen Sie daran, daß bisher kaum Reaktionen auf Ihre Eingaben erfolgten. Sie sind zwar in der Lage, beliebige Zeichen in der vielfältigsten Art und Weise

auf dem Bildschirm darzustellen, zu weitergehenden Reaktionen konnten Sie Ihren Computer durch diese "Tipperei" bisher jedoch nicht veranlassen.

Geben Sie nun bitte folgendes ein:

```
ICH BESITZE EINEN C16/PLUS4
```

Der Cursor befindet sich nach dieser Eingabe unmittelbar hinter dem letzten Zeichen, der "4". Drücken Sie nun die Taste RETURN. Zum ersten Mal erleben Sie eine "eigenständige" Reaktion des Computers. Auf dem Bildschirm erscheint:

```
?SYNTAX ERROR

READY.
```

In dem Moment, in dem Sie die Taste RETURN betätigten, faßte der C16, C116 bzw. Plus/4 Ihre Eingabe als Befehl auf, den er ausführen sollte. Ein "SYNTAX ERROR" ist eine Fehlermeldung, die aussagt, daß Ihre Eingabe nicht verstanden wird, daß sie für den Computer schlicht unsinnig ist. Die Aussage selbst ist zwar zweifellos richtig, entspricht jedoch keinem der BASIC-Befehle, die ein C16, C116 oder Plus/4 versteht. Sie sprachen gewissermaßen in einer "Fremdsprache".

Das "READY" bedeutet, daß alle eingegebenen Befehle vom Computer bearbeitet wurden und Sie nun weitere Befehle eingeben können.

Sie sehen, ohne die RETURN-Taste ist es nicht möglich, Anweisungen zu erteilen, die der Computer ausführen soll. Nachdem Sie nun über Tastatur und "Editor", das heißt über die Möglichkeiten zur Eingabe und Korrektur von Befehlen Bescheid wissen, ist es an der Zeit, jene Sprache zu erlernen, die Ihr Rechner verstehen kann: BASIC.

3. BASIC-Kurs

3.1 Grundlagen

Zu Beginn dieses Kurses möchte ich Sie bitten, verschiedene kleine Programme abzutippen, die demonstrieren werden, daß sich bereits mit sehr kurzen Programmen interessante Effekte erzielen lassen.

3.1.1 Demoprogramme zum "Aufwärmen"

Geben Sie bitte folgende Befehle in einer beliebigen leeren Bildschirmzeile ein:

Bildschirmrahmen verändern

```
FOR I=100 TO 700:COLOR 4,4,I/100:NEXT
```

Drücken Sie bitte RETURN, wenn Sie diese Zeile komplett eingegeben haben.

Sie haben mehrere Befehle eingegeben, die durch Doppelpunkte voneinander getrennt werden müssen. Diese "Befehlskette" ist eine Folge von Anweisungen - also ein Programm -, die nach Betätigung von RETURN vom C16, C116 oder Plus/4 analysiert und alle (!) nacheinander ausgeführt werden.

Wenn Sie das Programm richtig eingegeben haben, wird sich die Farbe des Bildschirmrandes schrittweise verändern. Zuerst ist sie recht dunkel und wird dann immer heller werden.

Wenn Sie sich bei der Eingabe vertippt hatten, wird der C16, C116 oder Plus/4 die eingegebenen Befehle nicht verstehen, und auf dem Bildschirm erscheint die Fehlermeldung "SYNTAX ERROR". In diesem Fall korrigieren Sie die Eingabe bitte mit den erwähnten Editierfunktionen, das heißt mit den Cursortasten und DEL/INST zum Löschen beziehungsweise Einfügen eines

Zeichens. Nach beendeter Korrektur müssen Sie wiederum RETURN drücken, bevor der Computer die Befehle auszuführen versucht.

"Ball" über den Bildschirm bewegen

Das nächste Demoprogramm enthält bereits ein wesentliches Grundmerkmal aller Videospielprogramme, die Bewegung eines Objektes. Geben Sie folgende Zeile ebenfalls in einer freien Zeile ein:

```
FOR I=1 TO 38:CHAR 1,I,2,"O":CHAR 1,I,2," ":NEXT
```

Da in einer Bildschirmzeile maximal 40 Zeichen dargestellt werden können, die Eingabe jedoch 48 Zeichen umfasst, muß sie in der nächsten Zeile fortgesetzt werden. Drücken Sie bitte RETURN, wenn Sie die Befehle komplett eingegeben haben.

Wenn die Eingabe richtig war, wird sich das Zeichen "O" quer über den Bildschirm bewegen. Anschließend erhalten Sie die Meldung "READY", mit der Ihnen mitgeteilt wird, daß die Anweisungen ausgeführt worden sind, das heißt, daß das Programm nun beendet ist.

Geräusche erzeugen

Das dritte Demoprogramm besteht wiederum aus mehreren Befehlen, die durch Doppelpunkte getrennt in einer freien Zeile eingegeben werden müssen. Drücken Sie anschließend wieder RETURN, um den Computer zu veranlassen, das Programm auszuführen:

```
VOL 8:FOR I=100 TO 300:SOUND 1,I,5:NEXT
```

Es werden Geräusche mit immer höher werdender Frequenz erzeugt. Bedingung ist selbstverständlich, daß die Lautstärke am Fernseher nicht völlig auf Null gedreht ist.

Mit diesem Programm können Sie selbständig experimentieren, indem Sie die eingegebenen Befehle nach Ablauf des Programms ändern. Beschränken Sie die Änderungen bitte auf die beiden Zahlen 100 und 300, wobei Sie beachten sollten, daß die erste Zahl immer (!) kleiner ist als die zweite.

Wenn Sie die Zahlen ändern und die Eingabe anschließend mit RETURN beenden, wird das geänderte Programm ausgeführt werden. Die Unterschiede sind deutlich hörbar: Der erzeugte Ton besitzt eine andere "Start"- beziehungsweise "Endfrequenz". Diese Versuche können Sie beliebig oft durchführen.

3.1.2 Programmiersprachen, Programme und Algorithmen

Nach diesen Demoprogrammen, die Ihnen zeigen sollten, wozu Ihr C16/Plus4 mit wenigen Befehlen veranlasst werden kann, folgt nun ein wenig "trockene" Theorie, die jedoch unumgänglich ist.

Nur dann, wenn Sie über verschiedene Prinzipien der Programmierung Bescheid wissen, werden Sie später in der Lage sein, auch größere und komplexere Programme zu schreiben, die mehrere Bildschirmseiten lang sind.

Diese Grundlagen sind auch dann erforderlich, wenn Sie Ihren C16/Plus4 wirklich kennen- und interne Abläufe verstehen lernen wollen.

Zuerst will ich erläutern, was unter einem Programm zu verstehen ist. Den üblichen Ausgangspunkt bei der Programmierung bildet ein Problem, das mit Hilfe des Computers gelöst werden soll, zum Beispiel die Erstellung eines Malprogramms.

Den ersten Schritt bildet die Problemanalyse. Im Falle des Malprogramms wird geklärt, welche Funktionen enthalten sein sollen (Kreise zeichnen, Rechtecke zeichnen, mit verschiedenen Pinselstärken arbeiten usw.), ob farbig oder nur schwarzweiß gemalt werden kann, ob es möglich sein soll, eine erstellte Graphik

auszudrucken (und wenn ja, auf welchen Druckern) und so weiter. Auf diese Weise wird ein Anforderungskatalog erstellt.

Eine grobe Vorstellung von dem zu erstellenden Programm dürfte nach Erstellung dieses Anforderungskatalogs vorhanden sein. Diese Vorstellung wird nun verfeinert, zum Beispiel indem geklärt wird, wie sich das Programm optisch auf dem Bildschirm darstellen soll und wie der Benutzer einzelne Programmfunktionen benutzt (Beispiel: Wenn die Taste F1 gedrückt wird, soll die Graphik ausgedruckt werden, mit F2 soll die Graphik gelöscht werden usw.).

Wenn das Problem auf diese Weise immer weiter zergliedert wurde, so daß zum Schluß der gesamte Programmablauf und die einzelnen Programmteile feststehen, werden sogenannte "Algorithmen" entwickelt.

Die offizielle Definition eines Algorithmus lautet etwa: "Ein Algorithmus ist eine Vorschrift zur Lösung eines Problems mit einer endlichen Anzahl von Schritten".

Diese abstrakte Definition kann an einem Beispiel verdeutlicht werden. Wenn Ihr Problem lautet: "Wie berechne ich das Endkapital, das sich auf meinem Sparbuch bei einem Zinssatz von 4% aus einem bestimmten Ausgangskapital nach einem Jahr ergibt?", ergibt sich als Algorithmus folgende Formel:

Endkapital = Ausgangskapital + Ausgangskapital/100*4

Das heißt, Sie erhalten das Endkapital, indem Sie zum Ausgangskapital 4% des jeweiligen Betrages addieren.

Programmieren besteht zum größten Teil in der Entwicklung solcher Algorithmen für spezielle Probleme und deren Umsetzung in Befehle einer Programmiersprache. Übrigens haben Programme eine äußerst unangenehme Eigenschaft: Sie sind fehlerhaft! Dies gilt zumindest für umfangreichere Programme. Es ist fast unmöglich, Programme zu schreiben, die länger sind als eine Bildschirmseite und auf Anhieb einwandfrei funktionieren.

Dem Schreiben des Programms folgt als letztes die Phase der Fehlersuche und Programmkorrektur, die vor allem dann sehr langwierig ist, wenn das Programm schlecht geplant war (wenn Sie zum Beispiel im fertigen Malprogramm feststellen müssen, daß Sie verschiedene unbedingt nötige Funktionen wie das Malen von Rechtecken oder das Abspeichern und Laden von Cassette oder Diskette schlichtweg vergessen haben).

Sie kennen nun den theoretischen Ablauf beim Erstellen von Programmen. Dennoch wissen Sie noch nichts über die verschiedenen Programmiersprachen. Eine Programmiersprache ist notwendig, weil Computer prinzipiell "dumm" sind. Eigenständig unternehmen Sie nichts und in einer natürlichen Sprache kann man Ihnen keinerlei Anweisungen erteilen.

Beim C16/Plus4 wird zur Programmierung meist die Sprache BASIC verwendet, die beide Rechner verstehen und die serienmäßig eingebaut ist (dies ist keine Selbstverständlichkeit. Bei vielen Computern muß die Programmiersprache erst von Cassette oder Diskette geladen werden).

Mit dieser Sprache werden wir uns im folgenden beschäftigen. Zuvor schildere ich jedoch noch, was bei der Eingabe von Befehlen in dieser Sprache passiert.

Nach dem Einschalten Ihres C16 oder Plus4 wartet der sogenannte "BASIC-Interpreter" auf Ihre Eingaben. Diese Eingaben erfolgen mit Hilfe des ebenfalls fest eingebauten "Editors", der - wie wir sahen - komfortable Funktionen zur Eingabe und vor allem zur nachträglichen Korrektur von Eingaben zur Verfügung stellt.

Wenn die Taste RETURN betätigt wird, versucht der BASIC-Interpreter Ihre Eingabe zu "interpretieren", das heißt Ihre Anweisungen zu verstehen und auszuführen. Nun gibt es drei Möglichkeiten:

- Sie beherrschen die Sprache BASIC und die Befehlswörter dieser Sprache noch nicht und haben sich falsch ausgedrückt, was dazu führt, daß eine Fehlermeldung, wie zum Beispiel "SYNTAX ERROR", auf dem Bildschirm erscheint. Der BASIC-interpreter teilt Ihnen damit mit, daß er zumindest eine Ihrer Anweisungen nicht versteht. Wie Sie noch bemerken werden, sind die Regeln von BASIC äußerst streng. Ein Komma zuviel oder an der falschen Position führt bereits zu einem "SYNTAX ERROR".

- Die Syntax der Anweisungen stimmt zwar, der Programmablauf wurde jedoch nicht gründlich genug überdacht. In diesem Fall versteht der BASIC-Interpreter die gegebenen Anweisungen und führt Sie ohne Fehlermeldung aus, jedoch nicht mit dem gewünschten Ergebnis. Dieser Fall wird häufig vorkommen, wenn Sie mit BASIC bereits vertraut sind und beginnen, komplexe und entsprechend umfangreiche Programme zu schreiben.

- Der letzte Fall, der in der Praxis erst nach vielen Mühen und häufigen Programmkorrekturen vorkommt: Die Syntax der Anweisungen stimmt und der Programmablauf ist ebenfalls fehlerfrei, das heißt, Ihr Programm ist fertig und kann nun benutzt werden.

Falls Sie sich nun fragen, was Sie unter dem BASIC-Interpreter zu verstehen haben, der Anweisungen analysiert und ausführt: Der BASIC-Interpreter ist ebenfalls ein Programm, das jedoch fest in Ihrem Rechner eingebaut ist und nach dem Einschalten sofort tätig wird. Geschrieben ist dieses Programm nicht in BASIC, sondern in der sogenannten "Maschinensprache", die weitaus schwieriger zu beherrschen ist als BASIC und uns nicht weiter interessieren soll.

3.1.3 Grundlagen der BASIC-Programmierung

Im folgenden Kapitel werden Grundlagen der BASIC-Programmierung erläutert. Auch wenn Ihnen bereits bekannt ist, wie Programme gespeichert und geladen werden, und wie der PRINT-Befehl arbeitet, bitte ich Sie, dieses Kapitel dennoch zu lesen.

Das BASIC des C16/Plus4 besitzt einige "Tücken", wie zum Beispiel den "Einfüge-Modus" und vor allem die "Steuerzeichen". Das folgende Kapitel erläutert diese Eigenheiten, die den BASIC-Einsteiger immer wieder vor Probleme stellen.

3.1.3.1 Der erste BASIC-Befehl: PRINT

Der erste BASIC-Befehl, der nun erläutert wird, nennt sich PRINT und ist einer der wichtigsten und am häufigsten angewendeten Befehle. Wie alle BASIC-Befehle entstammt auch dieser der englischen Sprache und kann mit "Schreibe" oder "Gib aus" übersetzt werden.

PRINT gibt Zeichen oder Zeichenketten (sogenannte "Strings") auf dem Bildschirm aus. Bevor ich nun theoretisch erläutere, wie dieser Befehl genau angewendet wird, schlage ich vor, daß Sie es mit der folgenden Zeile selbst ausprobieren:

```
PRINT "DIES IST EIN TEST"
```

Geben Sie diese Zeile in einer leeren (!) Zeile ein und drücken Sie anschließend RETURN. In der Zeile darunter erscheint die Zeichenkette - der String - die Sie in Anführungszeichen angegeben haben. Darunter erscheint die Meldung "READY.", mit der Ihnen der BASIC-Interpreter mitteilt, daß Ihre Anweisung ausgeführt wurde.

```
PRINT "DIES IST EIN TEST"
DIES IST EIN TEST

READY.
```

Umgangssprachlich ausgedrückt lautete Ihre mit dem Befehl PRINT gegebene Anweisung 'Schreibe die Zeichen "DIES IST EIN TEST" auf den Bildschirm'. Das Leerzeichen zwischen dem PRINT-Befehl und den folgenden Anführungszeichen interessiert den BASIC-Interpreter übrigens nicht, sondern dient der besseren Übersichtlichkeit. Der BASIC-Interpreter versteht den Befehl auch dann, wenn Sie das Leerzeichen entfernen:

```
PRINT"DIES IST EIN TEST"
```

PRINT ist ein sogenannter "Ausgabebefehl". Mit PRINT können Sie auch mehrere Zeichenketten (Strings) hintereinander ausgeben, wenn diese durch je ein Semikolon voneinander getrennt werden:

```
PRINT "MEIN NAME IST";"XYZ"
MEIN NAME ISTXYZ

READY.
```

Wie Sie sehen, werden mehrere dieser "Strings", die mit Semikola voneinander getrennt sind, unmittelbar hintereinander ausgegeben. Wenn Sie ein Leerzeichen (auch "Space" genannt) als Trennung zwischen mehreren Strings wünschen, müssen Sie dies in der auszugebenden Zeichenkette angeben:

```
PRINT "MEIN NAME IST ";"XYZ"
MEIN NAME IST XYZ

READY.
```

Geben Sie bitte den geänderten PRINT-Befehl nicht komplett neu ein, sondern nutzen Sie den eingebauten Editor des C16, C116 bzw. Plus/4 zur Arbeitserleichterung. Bewegen Sie den Cursor auf das Anführungszeichen, drücken Sie die Taste INST

(gleichzeitige Betätigung von SHIFT notwendig), um die gewünschte Leerstelle einzufügen, und drücken Sie RETURN, um die Befehlseingabe abzuschließen.

Merken Sie sich bitte: Um eine Befehlseingabe mit RETURN abzuschliessen, muß sich der Cursor nicht notwendigerweise hinter dem Befehl befinden. Es genügt, wenn Sie sich irgendwo in der Befehlszeile befinden, an beliebiger Stelle!

Sie können mit PRINT mehrere Zeichenketten nicht nur unmittelbar hintereinander, sondern auch optisch gegliedert ("formatiert") auf dem Bildschirm ausgeben:

```
PRINT "DIES","IST","EIN","TEST"
DIES       IST       EIN       TEST

READY.
```

Der C16, C116 oder Plus/4 unterteilt den Bildschirm in vier horizontale "Zonen" mit je zehn Zeichen. Werden mehrere auszugebende Zeichenketten mit je einem Komma voneinander getrennt, wird jede dieser Zeichenketten in einer anderen Zone ausgegeben. Die Frage ist nun, was passiert, wenn mehr als vier Zeichenketten durch Kommata getrennt werden:

```
PRINT "DIES","IST","NOCH","EIN","TEST"
DIES       IST       NOCH      EIN
TEST

READY.
```

Die Zeichenketten werden der Reihe nach in den vier Bildschirmzonen ausgegeben, die nächste Ausgabe der fünften Zeichenkette beginnt wieder in Zone eins der nächsten Zeile.

Eine weitere Formatierungsmöglichkeit besteht in der Verwendung mehrerer PRINT-Befehle. Wie in den Demoprogrammen gezeigt wurde, können mehrere Befehle in einer Zeile eingegeben werden, wenn Sie mit Doppelpunkten voneinander getrennt werden. Wenn RETURN gedrückt wird, führt der BASIC-Inter-

preter anschließend alle (!) Befehle nacheinander aus. Geben Sie bitte folgende Zeile ein und schliessen Sie sie mit RETURN ab:

```
PRINT "DIES IST":PRINT "EIN TEST"
DIES IST
EIN TEST

READY.
```

Nach jedem PRINT-Befehl erfolgt ein sogenannter "Zeilenvorschub", das heißt der Cursor wird auf den Beginn der nächsten Zeile positioniert, analog dem Wagenrücklauf bei der Schreibmaschine. Erfolgt nun ein weiterer PRINT-Befehl, wird die Zeichenkette in dieser neuen Zeile ausgegeben. Da die Befehlszeile zwei PRINT-Befehle enthält, werden die angegebenen Zeichenketten untereinander ausgegeben.

Außer Zeichenketten können mit dem PRINT-Befehl auch Ergebnisse (!) von numerischen Ausdrücken ausgegeben werden. Mit Hilfe des PRINT-Befehls können Sie Ihren C16, C116 oder Plus/4 als eine Art Taschenrechner verwenden. Wenn Sie folgende Zeile eingeben:

```
PRINT 2*(10+5)
```

erhalten Sie als Ergebnis:

```
PRINT 2*(10+5)
 30

READY.
```

Beachten Sie bitte, daß der numerische Ausdruck nicht in Anführungszeichen gesetzt wird, da er ansonsten als unverändert auszugebende Zeichenkette aufgefasst wird, wie im folgenden Beispiel:

```
PRINT "2*(10+5)"
2*(10+5)

READY.
```

Wird der PRINT-Befehl zum Rechnen benutzt, können alle Formatierungsmöglichkeiten verwendet werden, die wir bei der Ausgabe von Zeichenketten kennenlernten.

Beispiel 1:
```
PRINT 2*(10+5),4/2,10*100
```
ergibt:
```
 30          2           1000
```

Beispiel 2:
```
PRINT 10-20;30+2-3;100*1.5
```
ergibt:
```
-10  29  150
```

Wie Sie sehen, besteht ein Unterschied zwischen der Formatierung mit dem Semikolon, je nachdem ob Zeichenketten oder die Ergebnisse numerischer Ausdrücke formatiert werden sollen: Im ersten Fall werden die Zeichenketten unmittelbar hintereinander ausgegeben, im zweiten Fall befinden sich kleine Leerräume zwischen den einzelnen Zahlen. Diese Leerräume dienen der besseren Unterschiedung. Wie sollten Sie ohne diese Leerräume erkennen können, ob "1234" eine (!) Zahl ist oder aber die beiden nacheinander ausgegebenen Zahlen "12" und "34" darstellt.

Den PRINT-Befehl wirklich zu beherrschen, ist dank seiner vielfältigen Möglichkeiten nicht ganz einfach. Bisher wurde gezeigt, wie Ausgaben formatiert werden können, und daß sowohl Zeichenketten als auch die Ergebnisse von Berechnungen von PRINT verarbeitet werden. Noch komplexer wird der Befehl dadurch, daß alle diese Möglichkeiten in einem (!) PRINT-Befehl gemischt werden können:

Beispiel 1:
```
PRINT "DIES","IST";"EIN","TEST"
```
ergibt:
```
DIES      ISTEIN    TEST
```

Beispiel 2:
```
PRINT "2 UND 3 IST";2+3
```
ergibt:
```
2 UND 3 IST 5
```

Beispiel 3:
```
PRINT "2*3=";2*3,"2/3=";2/3
```
ergibt:
```
2*3= 6    2/3= .666666667
```

Da ich der Ansicht bin, daß die einzelnen BASIC-Befehle nur durch ständiges Üben wirklich zu verstehen sind, empfehle ich Ihnen, vor dem Weiterlesen mit dem PRINT-Befehl zu experimentieren und sowohl Zeichenketten als auch numerische Ausdrücke auf verschiedenste Weise formatiert auszugeben.

3.1.3.2 Anführungszeichen-Modus und Einfüge-Modus

Wenn Sie meinen Rat befolgten und selbständig mit dem PRINT-Befehl experimentierten, werden Sie sicherlich gewisse Eigenheiten des Editors bemerkt haben.

Wenn Sie die Taste INST betätigen, um einen Leerraum in einem Wort zu schaffen, dann jedoch kein Zeichen eingeben, sondern eine Cursortaste betätigen, wird der Cursor nicht bewegt. Stattdessen erscheint ein invers dargestelltes Zeichen, ein sogenanntes "Steuerzeichen" auf dem Bildschirm.

Wenn Sie mehrmals nacheinander INST drücken, erscheinen bei Betätigung der Cursortasten entsprechend viele dieser Steuerzeichen. Um das auszuprobieren, drücken Sie bitte dreimal INST. Betätigen Sie anschließend mehrmals die Taste, die den Cursor eine Zeile nach unten bewegt.

Auf dem Bildschirm erscheinen genau drei Steuerzeichen, bevor der Cursor tatsächlich nach unten bewegt wird. Wenn INST ein- oder mehrmals gedrückt wird, befinden Sie sich im "Einfüge-Modus". In diesem Modus erwartet der BASIC-Interpreter von Ihnen die Eingabe von Zeichen, die eingefügt werden sollen, keine Cursorbewegungen.

Wie wir noch sehen werden, können Steuerzeichen beim Programmieren sehr nützlich verwendet werden, im Moment stören sie nur und sind nicht beabsichtigt. Wenn Sie versehentlich INST gedrückt hatten und nun den Cursor bewegen wollen, sollten Sie die Eingabe mit RETURN abschliessen. RETURN beendet nicht nur die Eingabe, sondern auch den Einfüge-Modus. Sie nehmen dadurch zwar einen "SYNTAX ERROR" in Kauf, wenn die Eingabe noch nicht vollständig war, vermeiden aber die unschönen Steuerzeichen.

Mit RETURN können Sie ebenfalls jederzeit den "Anführungszeichen-Modus" beenden. Dieser Modus wird eingeschaltet, wenn Sie ein Anführungszeichen eingeben. Jede nun folgende Cursorbewegung erzeugt ebenfalls ein Steuerzeichen. Der Grund: Nach Eingabe eines Anführungszeichens erwartet der BASIC-Interpreter die Eingabe einer Zeichenkette (siehe PRINT-Befehl) und keine Cursorbewegungen.

Versehentlich erzeugte Steuerzeichen können Sie wie jedes andere Zeichen mit der Taste DEL löschen.

3.1.3.3 Der Einsatz von Steuerzeichen

Ich deutete bereits an, daß Steuerzeichen sehr nützlich verwendet werden können. Mit Steuerzeichen kann der Cursor bewegt oder der Bildschirm gelöscht werden, und die Zeichenfarbe kann mit Hilfe von Steuerzeichen verändert werden.

Der bewußte Einsatz von Steuerzeichen erfolgt im PRINT-Befehl. Geben Sie bitte ein:

```
PRINT "
```

Drücken Sie noch nicht (!) RETURN, sondern betätigen Sie viermal die Taste CURSOR NACH UNTEN. Hinter dem Anführungszeichen erschienen vier Steuerzeichen. Geben Sie unmittelbar hinter diesen Steuerzeichen ein:

```
DIES IST EIN TEST".
```

Da inverse Graphikzeichen nur bei Commodore-Rechnern vorhanden sind und keinesfalls auf einer Schreibmaschine, ersetze ich die Steuerzeichen im folgenden durch das Zeichen "#". Ihre Eingabe sollte wie folgt aussehen:

```
PRINT "####DIES IST EIN TEST"
```

Schliessen Sie diese Eingabe mit RETURN ab. Die Zeichenkette "DIES IST EIN TEST" wird nicht unmittelbar unterhalb des PRINT-Befehls ausgegeben, sondern erst nach vier Leerzeilen. Diese vier Leerzeilen werden durch die vier Steuerzeichen bewirkt. Für jede Cursorbewegung existiert ein entsprechendes Steuerzeichen. Findet der BASIC-interpreter in einem mit PRINT auszugebenden String ein Steuerzeichen, führt er die entsprechende Cursorbewegung aus.

Mit Hilfe der Steuerzeichen können bei der Abarbeitung eines PRINT-Befehls Betätigungen der Cursortasten "simuliert" werden. Geben Sie folgendes ein:

```
PRINT "###DIES IST":PRINT "##EIN TEST"
```

Das Zeichen "#" soll das Steuerzeichen darstellen, das entsteht, wenn nach dem Anführungszeichen die Taste CURSOR NACH OBEN betätigt wird. Drücken Sie diese Taste drei- beziehungsweise zweimal, um entsprechend viele Steuerzeichen zu erzeugen. Die entsprechende Bildschirmausgabe:

```
EIN TEST
DIES IST
READY.
PRINT "###DIES IST":PRINT "##EIN TEST"
```

Wie Sie sehen, erscheint die erste Ausgabe ("DIES IST") drei Zeilen oberhalb der gewohnten Position (drei Steuerzeichen CURSOR NACH OBEN). Die zweite Ausgabe ("EIN TEST") würde normalerweise eine Zeile unterhalb der Zeichenkette "DIES IST" ausgegeben. Da der BASIC-Interpreter jedoch zuerst die beiden Steuerzeichen CURSOR NACH OBEN bearbeitet, wird die entsprechende Cursorbewegung ausgeführt und die zweite Zeichenkette oberhalb der ersten ausgegeben.

Wie dieses Besipiel zeigt, kann eine "Befehlszeile" mehrere durch Kommas getrennte Befehle enthalten. Wird die Eingabe mit RETURN beendet, werden alle (!) Befehle nacheinander bearbeitet.

Die Anzahl der Befehle ist durch die maximale Länge einer Befehlszeile beschränkt. Der BASIC-Interpreter kann Befehlszeilen bis zu einer Länge von 88 Zeichen verabeiten. Längere Befehlszeilen werden nicht akzeptiert und mit der Fehlermeldung "?STRING TOO LONG LONG ERROR" kommentiert.

Außer den vier Cursortasten kann auch die Betätigung der Tasten INST, DEL, RVS ON, RVS OFF und der Farbtasten simuliert werden. Wenn Sie eine Zeichenkette in der Farbe Orange ausgeben wollen, drücken Sie bitte nach dem ersten Anführungszeichen im PRINT-Befehl die entsprechende Farbtaste (C= und gleichzeitig die Zifferntaste 1), bevor Sie die Zeichenkette eingeben:

```
PRINT "#DIES IST EIN TEST"
```

3.1.3.4 Zeichen und Zeichencodes

Im Anhang Ihres Bedienungshandbuches finden Sie unter der Überschrift "ASC- und CHR$-Codes" eine Tabelle, in der jedem Zeichen eine Zahl zugeordnet wird. Um den Sinn dieser Tabelle zu verstehen, müssen wir etwas tiefer in die "Innereien" des C16, C116 bzw. Plus/4 einsteigen.

Wie jeder andere Computer auch kann der C16, C116 und Plus/4 nur mit Zahlen umgehen. Im Grunde genommen ist jeder Computer eine Art "aufgemöbelter Taschenrechner", für den der Umgang mit Zeichen und Buchstaben nicht möglich ist.

Die Zeichenverarbeitung geschieht auf einem Umweg. Wenn Sie dem BASIC-Interpreter den Befehl geben: PRINT "DIES IST EIN TEST", wird jeder Buchstabe analysiert und in eine Zahl umgewandelt, die dem betreffenden Zeichen zugeordnet ist (A=65, B=66, C=66,...). Mit diesen Zahlen arbeitet der BASIC-Interpreter. Wenn er seine Arbeit beendet hat, erfolgt die Umwandlung in umgekehrter Richtung. Jenes Zeichen wird auf dem Bildschirm ausgegeben, das der betreffenden Zahl zugeordnet ist.

Wie wir sahen, sind nicht nur Buchstaben und Ziffern Zeichen. Auch die Cursor- und die Farbtasten werden durch Zeichen dargestellt (die Steuerzeichen), denen ebenfalls je eine Zahl zugeordnet ist.

Mit dem Befehl PRINT "A" erteilen wir die Anweisung: "Gib das Zeichen A aus". Da wir dank der genannten Tabelle den Code kennen, der dem Buchstaben A zugeordnet ist (A=65), könnten wir genausogut sagen: "Gib das Zeichen aus, das dem Code 65 zugeordnet ist".

Genau diese Aufgabe besitzt der CHR$-Befehl. Dieser Befehl kann im PRINT-Befehl dazu benutzt werden, ein Zeichen auszugeben, das dem angegebenen Code entspricht. Der Code selbst muß in einer Klammer hinter dem CHR$-Befehl angegeben werden. Mit folgendem Befehl wird der Buchstabe A ausgegeben:

```
PRINT CHR$(65)
```

Weitere Beispiele:

1. Ausgabe von B (Code 66): `PRINT CHR$(66)`
2. Ausgabe von C (Code 67): `PRINT CHR$(67)`
3. Ausgabe von Z (Code 90): `PRINT CHR$(90)`
4. Ausgabe von % (Code 37): `PRINT CHR$(37)`
5. Ausgabe von 7 (Code 55): `PRINT CHR$(55)`

Wie das letzte Beispiel demonstriert, sind - so seltsam es Ihnen am Anfang auch erscheinen mag - allen (!) Zeichen, selbst den Ziffern, Zahlen zugeordnet. Geben Sie die Beispiele ein und überprüfen Sie die verwendeten Codes anhand der Tabelle in Ihrem Handbuch.

Wirklich interessant wird die CHR$-Funktion erst in Verbindung mit den Steuerzeichen. In der Tabelle werden Sie entdecken, daß fast allen Tasten der Tastatur ein Code zugeordnet ist, auch den Cursortasten und den Farbtasten. Die Betätigung einer dieser Tasten kann in einem PRINT-Befehl nicht nur mit Hilfe von Steuerzeichen, sondern auch mit der CHR$-Funktion simuliert werden.

Laut Tabelle ist der Taste HOME, die den Cursor in die linke obere Ecke des Bildschirms setzt, der Code 19 zugeordnet. Diese Tastenbetätigung läßt sich daher simulieren mit dem Befehl:

```
PRINT CHR$(19)
```

Das bereits bekannte Steuerzeichen für CURSOR UNTEN besitzt den Code 17 und kann daher mit PRINT CHR$(17) simuliert werden. Geben Sie folgende Zeilen ungefähr in der Bildschirmmitte ein und schliessen Sie sie mit RETURN ab:

```
PRINT CHR$(19);"DIES IST":PRINT CHR$(17);"EIN TEST"
```

Auf dem Bildschirm erscheint die Ausgabe:

```
DIES IST

      EIN TEST

READY.
```

Der Befehl CHR$(19) entspricht dem Steuerzeichen für die Taste HOME. Der Cursor wird daher auf die Ausgangsposition links oben gesetzt und anschließend die Zeichenkette "DIES IST" ausgegeben. Der Befehl CHR$(17), der der Taste CURSOR UNTEN entspricht, bewirkt die Leerzeile vor Ausgabe des Strings "EIN TEST".

Um den Bildschirm zu löschen und anschließend den String "C16/PLUS4" in der Farbe Purpur auszugeben, verwenden wir folgenden Befehl:

```
PRINT CHR$(147);CHR$(156);"C16/PLUS4"
```

Überprüfen Sie die verwendeten Codes anhand der Tabelle (CLR (CLEAR) = 147 und PUR (PURPUR) = 156).

Der Vorteil dieser Methode zur Benutzung von Steuerzeichen ist offensichtlich: Sie vermeiden in Ihren Programmen das Auftreten unverständlicher inverser Zeichen, die nur schwer erkennen lassen, um welches Steuerzeichen es sich handelt. Daß zum Beispiel der Code 147 der Taste CLEAR entspricht, läßt sich sicherlich leichter merken als das entsprechende Steuerzeichen.

3.1.3.5 Direkt-Modus und Programm-Modus

Bei den bisherigen Übungen gaben wir Befehle im sogenannten "Direkt-Modus" ein. Nach dem Beenden der Eingabe mit RETURN wurden die Befehle unverzüglich ausgeführt. Größere BASIC-Programme werden jedoch immer (!) im "Programm-Modus" geschrieben.

Der Unterschied zum Direktmodus: Mit RETURN wird im Direkt-Modus das Kommando zum Ausführen der Befehlszeile gegeben. Im Programm-Modus wird ebenfalls jede Befehlszeile mit RETURN abgeschlossen. Die eingegebenen Befehle werden jedoch nicht unmittelbar ausgeführt, sondern der BASIC-Interpreter merkt sich die Befehle zur späteren Ausführung vor.

Nachdem alle benötigten Befehlszeilen eingegeben wurden, wird die Abarbeitung des kompletten Programms mit dem Befehl RUN veranlaßt. Erst nach Eingabe von RUN werden die einzelnen Programmbefehle der Reihe nach analysiert und ausgeführt.

Die Frage ist nun, wie der BASIC-Interpreter zwischen Direkt- und Programm-Modus unterscheiden kann. Der Programm-Modus wird an den sogenannten "Zeilennummern" erkannt. Jede Befehlszeile eines BASIC-Programms erhält eine beliebige (!) Nummer zwischen null und 65280, die den eigentlichen Befehlen vorangestellt wird.

Beispiel:
```
10 PRINT "DIES IST EIN TEST"
```

Geben Sie diese Zeile bitte ein und drücken Sie anschließend RETURN. Der Cursor wird in die nächste Zeile springen, ansonsten erfolgt keinerlei Reaktion.

Sie haben nun zum ersten Mal ein echtes BASIC-Programm geschrieben. Dieses Programm wird erst ausgeführt, wenn Sie (im Direkt-Modus, das heißt ohne Zeilennummer) den Befehl RUN (= "Start") eingeben.

Geben Sie bitte den Befehl RUN ein und schliessen Sie diese Eingabe mit RETURN ab. Wie gewohnt erscheint auf dem Bildschirm die Ausgabe "DIES IST EIN TEST".

```
10 PRINT "DIES IST EIN TEST"
RUN
DIES IST EIN TEST

READY.
```

Der Programm-Modus besitzt unter anderem den großen Vorteil, daß erst durch ihn beliebig große Programme möglich werden. Wenn Sie mit dem Direkt-Modus intensiv experimentierten, werden Sie festgestellt haben, daß eingegebene Befehlszeilen nicht unbeschränkt lang sein dürfen (maximal 88 Zeichen).

Die Länge einer "Programmzeile" ist zwar ebenfalls auf 88 Zeichen beschränkt, dafür kann ein Programm aus beliebig vielen Programmzeilen bestehen. Geben Sie zur Demonstration folgende Programmzeilen ein (drücken Sie nach Eingabe einer Programmzeile immer RETURN!):

```
10 PRINT CHR$(147)
20 PRINT "DIES IST EIN TEST"
30 PRINT "GETESTET WIRD DER PROGRAMM-MODUS"
40 PRINT "DES C16/PLUS4"
```

"Starten" Sie das Programm mit RUN. Die einzelnen Programmzeilen werden nun nacheinander bearbeitet und die darin enthaltenen Befehle ausgeführt. Zuerst wird der Bildschirm gelöscht, da in Programmzeile 10 der Code des Zeichens CLEAR ausgegeben wird (siehe Tabelle im Handbuch).

Anschließend werden die Zeilen 20 bis 40 bearbeitet und die darin enthaltenen PRINT-Befehle ausgeführt. Auf dem gelöschten Bildschirm wird ausgegeben:

```
DIES IST EIN TEST
GETESTET WIRD DER PROGRAMM-MODUS
DES C16/PLUS4
```

Wenn Sie sich in einer der Programmzeilen vertippt haben, gibt es mehrere Korrekturmöglichkeiten:

1. Bewegen Sie den Cursor in die betreffende Zeile und an die Position des zu korrigierenden Fehlers. Nehmen Sie mit Hilfe der Editierfunktionen (INST, DEL etc.) die Korrektur vor. Wichtig: Drücken Sie nach erfolgter Korrektur unbedingt die Taste RETURN! Wie erläutert, merkt sich der BASIC-

Interpreter eine Programmzeile, wenn diese mit RETURN abgeschlossen wird, er "speichert" sie. Diese gespeicherte Programmzeile wird durch die korrigierte Zeile erst ersetzt, wenn die korrigierte Zeile ebenfalls mit RETURN abgeschlossen wird!

2. Geben Sie die fehlerhafte Zeile komplett neu ein. Angenommen, Zeile 10 war fehlerhaft. Sie gaben zum Beispiel ein: 10 PRINT CHR$(140). Geben Sie die komplette Zeile neu ein (10 PRINT CHR$(147)) und beenden Sie die Eingabe mit RETURN. Im gleichen Moment, in dem Sie RETURN drücken, ersetzt der BASIC-Interpreter die alte Zeile 10 durch die neu eingegebene.

Wie wichtig der Abschluß einer eingegebenen Programmzeile mit RETURN ist, möchte ich an einem Beispiel demonstrieren. Wenn Sie im Direkt-Modus (ohne Zeilennummer) den Befehl LIST eingeben, zeigt Ihnen der BASIC-Interpreter das komplette, von Ihnen eingegebene Programm auf dem Bildschirm. Geben Sie bitte ein:

```
LIST
```

und drücken Sie RETURN. Auf dem Bildschirm erscheinen alle vier eingegebenen Programmzeilen, das Programm wird "aufgelistet":

```
LIST

10 PRINT CHR$(147)
20 PRINT "DIES IST EIN TEST"
30 PRINT "GETESTET WIRD DER PROGRAMM-MODUS"
40 PRINT "DES C16/PLUS4"

READY.
```

Geben Sie nun bitte eine weitere Programmzeile ein, jedoch ohne (!) die Eingabe mit RETURN zu beenden:

```
35 PRINT "DIES IST AUCH EIN TEST"
```

Bewegen Sie den Cursor in die nächste Zeile und geben Sie wiederum LIST ein (inclusive RETURN!). Die neu eingegebene Zeile 35 taucht im ausgegebenen "Programm-Listing" nicht auf, sie wurde, wie man sagt, vom BASIC-Interpreter nicht "übernommen", das heißt nicht zur weiteren Verarbeitung gespeichert.

Wenn Sie Zeile 35 in Ihr Programm einfügen wollen, können Sie sie neu eingeben und die Eingabe mit RETURN abschliessen, oder aber Sie wählen den komfortableren Weg: Bewegen Sie den Cursor zu der Bildschirmzeile, in der Sie die neue Programmzeile eingaben, und betätigen Sie RETURN, um die Zeile nachträglich in das Programm zu übernehmen.

Grundsätzlich gilt: Der BASIC-Interpreter merkt sich eine Programmzeile, die sich auf dem Bildschirm befindet, wenn Sie sich mit dem Cursor in dieser Zeile befinden und die Taste RETURN betätigen.

Wenn Sie Zeile 35 entweder komplett neu eingegeben oder aber mit RETURN nachträglich in das Programm übernommen haben, "listen" Sie bitte das komplette Programm mit dem Befehl LIST und der Betätigung von RETURN. Auf dem Bildschirm sollte folgendes Programm ausgegeben werden:

```
LIST

10 PRINT CHR$(147)
20 PRINT "DIES IST EIN TEST"
30 PRINT "GETESTET WIRD DER PROGRAMM-MODUS"
35 PRINT "DIES IST AUCH EIN TEST"
40 PRINT "DES C16/PLUS4"

READY.
```

Wie Sie sehen, wurde Zeile 35 ordnungsgemäß zwischen Zeile 30 und Zeile 40 in das Programm eingefügt. Der BASIC-Interpreter stellt automatisch die korrekte Reihenfolge gemäß den Zeilennummmern her.

Es ist daher jederzeit möglich, zwischen zwei Programmzeilen weitere Zeilen einzufügen, wenn für diese hinzukommenden Zeilen dazwischenliegende Zeilennummern verwendet werden.

Nun wird auch der Grund sichtbar, warum das Demoprogramm nicht in "Einerschritten" (1 PRINT CHR$(147) 2 PRINT ... 3 PRINT ... usw.), sondern in "Zehnerschritten" durchnumeriert wurde: Um jederzeit übersehene Befehle in das Programm nachträglich einfügen zu können.

Ich empfehle Ihnen, Ihre Programme immer in Zehnerschritten durchzunumerieren. Die erste Programmzeile sollte im übrigen die Nummer 100 erhalten. In der Praxis kommt es häufig vor, daß vor dem Programmbeginn noch eine ganze Reihe von Befehlen eingefügt werden müssen. Erhielt die erste Programmzeile die Nummer 100, so können Sie problemlos jederzeit 100 Programmzeilen vor dieser ersten Zeile einfügen (0-99).

Außer vergessenen Zeilen taucht in der Praxis zusätzlich das Problem überflüssiger Programmzeilen auf, die entfernt, "gelöscht", werden müssen.

Mit dem bisher Gelernten sollten Sie bereits selbst in der Lage sein, eine Programmzeile zu löschen. Wie erwähnt, wird bei der Eingabe einer Programmzeile eine bereits existierende Zeile mit der gleichen Zeilennummer durch die neue Programmzeile ersetzt. Es müßte daher möglich sein, eine Programmzeile zu löschen, indem eine neue Zeile eingegeben wird, der die gleiche Zeilennummer gegeben wird, die jedoch keinen einzigen Befehl enthält.

Dies ist tatsächlich möglich, wie Sie sehr leicht feststellen können. Geben Sie folgende Zeilennummer ein:

```
20
```

und drücken Sie RETURN, um die Eingabe abzuschliessen. Der BASIC-Interpreter ersetzt die alte Zeile 20 (PRINT "DIES IST EIN TEST") durch die neue Zeile 20, das heißt durch "nichts". Den Beweis liefert der Befehl LIST, der Ihnen folgendes Programm zeigt:

```
LIST

10 PRINT CHR$(147)
30 PRINT "GETESTET WIRD DER PROGRAMM-MODUS"
35 PRINT "DIES IST AUCH EIN TEST"
40 PRINT "DES C16/PLUS4"

READY.
```

Die Zeile 20 wurde gelöscht. Den endgültigen Beweis liefert ein Start des Programms mit dem Befehl RUN (plus RETURN):

```
GETESTET WIRD DER PROGRAMM-MODUS
DIES IST AUCH EIN TEST
DES C16/PLUS4

READY.
```

Jede Programmzeile kann gelöscht werden, indem die Zeilennummer eingegeben und anschließend RETURN betätigt wird.

Wenn Sie ein völlig neues Programm eingeben wollen, ist es angebracht, den Rechnerspeicher zu "säubern" und alle (!) Programmzeilen zu löschen.

Mit der gezeigten Methode ist das Löschen vieler Programmzeilen leider etwas aufwendig. Am einfachsten lässt sich das komplette Programm mit dem im Direkt-Modus gegebenen

Befehl NEW löschen. Geben Sie bitte NEW ein und "listen" Sie das Programm anschließend.

```
NEW

READY.
LIST

READY.
```

Nach der Eingabe von LIST erscheint nicht eine Programmzeile auf dem Bildschirm. Das Programm wurde komplett gelöscht.

Übrigens: Mit der STOP-Taste kann das Listen eines Programms jederzeit unterbrochen werden. Dies ist vor allem dann nützlich, wenn ein längeres Programm auf dem Bildschirm "durchrollt". Weiterhin kann dem LIST-Befehl angegeben werden, daß nur ein bestimmter Programmabschnitt gelistet werden soll. Wie vielfältig der LIST-Befehl ist, zeigt Ihnen die Befehlsübersicht am Ende dieses Buches.

3.1.3.6 Kommentieren von Programmen (REM)

Programme gewinnen sehr an Übersichtlichkeit und Verständlichkeit, wenn sie "kommentiert" werden. Der Befehl REM wird benutzt, um Kommentare in einem Programm unterzubringen.

```
10Ø X=2:REM ZUWEISUNG AN X
11Ø Y=5:REM ZUWEISUNG AN Y
12Ø PRINT X:REM AUSGABE VON X
13Ø PRINT Y:REM AUSGABE VON Y
```

Löschen Sie den Speicher mit NEW, geben Sie dieses Programm ein und starten Sie es mit RUN. Sie werden sehen, daß der REM-Befehl und die folgenden Kommentare keinerlei Auswirkungen auf den Programmablauf besitzen.

Wenn der BASIC-Interpreter auf den Befehl REM trifft, wird die Bearbeitung der aktuellen Programmzeile abgebrochen und die nächste Zeile bearbeitet.

Hinter dem Befehl REM können daher beliebige (!) Kommentare in eine Programmzeile integriert werden. Diese Technik zur Kommentierung von Programmen werde ich in den meisten folgenden Demoprogrammen anwenden.

3.1.3.7 Speichern und Laden von Programmen

Programme würden nicht allzu viel wert sein, wenn man sie nicht "speichern" und wieder "laden" könnte. Ein Programm befindet sich im "Arbeitsspeicher" Ihres C16, C116 beziehungsweise Plus/4. Wenn Sie den Rechner ausschalten, geht der Inhalt des Arbeitsspeichers, und damit das mühsam erstellte Programm, verloren.

Nun ist es sicher nicht sehr sinnvoll zum Beispiel ein Malprogramm mit 50 oder 100 Programmzeilen jedesmal, wenn es benutzt werden soll, komplett neu einzutippen. Programme können daher auf Cassette oder Diskette dauerhaft gespeichert und wieder geladen werden.

Zum Abspeichern wird der Befehl SAVE oder DSAVE verwendet, je nachdem, ob das Programm auf Cassette oder Diskette gespeichert werden soll.

Speichern auf Cassette: `SAVE"(NAME)`
Speichern auf Diskette: `DSAVE"(NAME)`

Die Angabe NAME muß von Ihnen durch einen beliebigen (maximal 16 Zeichen langen) Programmnamen ersetzt werden. Beim Abspeichern auf Cassette kann der Programmname entfallen. In diesem Buch werde ich jedoch immer (!) Programmnamen verwenden. Auf diese Weise fällt Ihnen die Umstellung leichter, wenn Sie zwar momentan mit einem Cassettenrecorder arbeiten, sich jedoch später eventuell ein Diskettenlaufwerk zulegen.

Geben Sie bitte das zur Übung verwendete Programm ein.

```
10 PRINT CHR$(147)
30 PRINT "GETESTET WIRD DER PROGRAMM-MODUS"
35 PRINT "DIES IST AUCH EIN TEST"
40 PRINT "DES C16/PLUS4"
```

Um dieses Programm auf Cassette unter dem Namen "DEMO" zu speichern, geben Sie ein: SAVE"DEMO und drücken Sie RETURN. Der BASIC-Interpreter fordert Sie auf, die Tasten PLAY und RECORD an der Datasette zu drücken. Anschließend wird das Programm auf der eingelegten Cassette aufgezeichnet.

Mit dem Befehl DSAVE"DEMO können Sie das Programm unter dem Namen "DEMO" auf Diskette speichern. Achten Sie bitte darauf, daß eine Diskette eingelegt ist und sich kein Schreibschutz darauf befindet. Die eingelegte Diskette muß "formatiert" sein. Was darunter zu verstehen ist, wird im Kapitel über die Dateiverwaltung mit der Floppy erläutert.

Wenn Sie momentan noch nicht in der Lage sind, eine neue Diskette zu formatieren, ist die einfachste Lösung, die Commodore-Demodiskette zu verwenden, nachdem der Schreibschutz entfernt wurde.

Schalten Sie Ihren Rechner nun bitte aus und anschließend wieder ein. Geben Sie LIST ein. Wie Sie sehen, ging das Programm durch das Ausschalten des Rechners verloren. Mit dem Befehl LOAD beziehungsweise DLOAD kann es wieder von Cassette oder Diskette geladen werden.

Laden von Cassette:	`LOAD"(NAME)`
Laden von Diskette:	`DLOAD"(NAME)`

Da das Programm unter dem Namen "DEMO" gespeichert wurde, muß der gleiche Name angegeben werden, um dieses Programm zu laden.

Um das Programm von Cassette zu laden, spulen Sie das Band zuerst zu jener Position zurück, ab der das Programm gespeichert wurde (Zählwerk beachten!). Geben Sie anschließend ein: LOAD"DEMO. Sie werden aufgefordert, die Taste "PLAY" zu drücken. Nach kurzer Zeit ist das Programm geladen und kann wie gewohnt "gelistet" und weiterbearbeitet werden.

Mit dem Befehl DLOAD"DEMO wird das Programm mit dem Namen "DEMO" von Diskette in den Rechnerspeicher geladen und kann ebenfalls anschließend "gelistet" werden.

Achten Sie bitte bei Verwendung eines Cassettenrecorders darauf, daß Sie Programme an verschiedenen Bandpositionen (oder besser: auf verschiedenen Cassetten) speichern, um ein Überschreiben zu vermeiden.

Bei Verwendung eines Diskettenlaufwerks ist für Sie der Befehl DIRECTORY sehr interessant, der das Inhaltsverzeichnis der eingelegten Diskette auf dem Bildschirm zeigt. Nach Eingabe von DIRECTORY wird das komplette Inhaltsverzeichnis "gelistet".

Da Ihnen in diesem Kapitel eine ganze Menge Stoff zum Programm-Modus geboten wurde, der nicht leicht zu "verdauen" ist, hier eine kurze Zusammenfassung.

Zusammenfassung

1. Programmzeilen und im Direkt-Modus eingegebene Befehlszeilen dürfen eine maximale Länge von 88 Zeichen besitzen.

2. Im "Programm-Modus" erhält jede "Programmzeile" eine beliebige (!) vorangestellte "Zeilennummer" zwischen null und 65280.

3. Eine Programmzeile wird in das komplette Programm aufgenommen (übernommen), wenn sich der Cursor in der Programmzeile befindet und Sie RETURN betätigen.

4. Die in den Programmzeilen enthaltenen BASIC-Befehle werden erst bearbeitet, wenn das Programm mit RUN "gestartet" wird.

5. Durch den Befehl LIST wird das komplette, "gespeicherte" Programm (oder ein Teil des Programms) auf dem Bildschirm sichtbar.

6. Programmzeilen können geändert werden, indem die Zeile selbst mit den Editierfunktionen geändert und anschließend RETURN betätigt wird, oder aber, indem die komplette Zeile neu eingegeben und mit RETURN übernommen wird. In beiden Fällen wird bei gleicher Zeilennummer die alte Programmzeile durch die neue Zeile ersetzt.

7. Zwischen zwei Programmzeilen kann jederzeit eine neue Zeile eingefügt werden, wenn eine dazwischenliegende Zeilennummer verwendet wird. Den BASIC-Interpreter interessiert nicht die optische Reihenfolge der Programmzeilen auf dem Bildschirm, sondern er "sortiert" die Zeilen nach der Größe der Zeilennummern.

8. Eine Programmzeile wird gelöscht, wenn die Zeilennummer eingegeben und diese "Leerzeile" mit RETURN übernommen wird.

9. Mit dem im Direkt-Modus einzugebendem Befehl NEW wird ein Programm vollständig gelöscht. NEW sollte vor der Erstellung eines neuen Programms immer (!) eingegeben werden, um "Überreste" eines noch vorhandenen Programms zu beseitigen.

10. Der Befehl REM dient der Kommentierung von Programmzeilen. Nach diesem Befehl können beliebige Kommentare folgen, ohne den Programmablauf zu beeinflussen.

11. Programme auf Cassette speichern: `SAVE"(NAME)`
    Programme auf Diskette speichern: `DSAVE"(NAME)`
    Programme von Cassette laden: `LOAD"(NAME)`
    Programme von Diskette laden: `DLOAD"(NAME)`

Ein Tip zum Abschluß: Wenn Ihre Programme nicht so funktionieren, wie sie sollten, "listen" Sie das Programm mit dem LIST-Befehl. Oftmals werden Sie feststellen, daß Programmzeilen fehlen oder aber Änderungen nicht übernommen wurden. Der Grund ist immer der gleiche: Nach Eingabe der Programmzeile oder der Änderung vergaßen Sie, die Änderung mit RETURN zu bestätigen. Denken Sie daran, daß RETURN die wichtigste Taste Ihres C16/Plus4 ist!

3.1.4 Rechnen mit dem C16, C116, Plus/4

Daß der C16, C116 bzw. Plus/4 die Grundrechenarten beherrscht, sahen wir bereits. Außer den vier Grundrechenarten stellt er eine Vielzahl weiterer arithmetischer Funktionen zur Verfügung, trigonometrische Funktionen, wie Sinus oder Tangens, die Qadratwurzelfunktion und so weiter.

3.1.4.1 Die Prioritätsebenen

Bei allen Rechenoperationen gelten die üblichen Prioritäten, das heißt Multiplikationen oder Divisionen besitzen Vorrang vor der Addition und der Subtraktion, so daß der Ausdruck 2+3*3 dem Ausdruck 2+(3*5) entspricht (die Klammerung soll die Reihenfolge der Rechenoperationen angeben. Vor der Multiplikation oder Division besitzen wiederum Potenzierungen Vorrang (im Ausdruck 2*3↑4 wird zuerst der Teil 3↑4 berechnet und anschließend die drei addiert). Die höchste Priorität besitzen Klammern. Mit Klammern können Sie die Reihenfolge von Berechnungen nach Belieben festlegen. Ausdrücke in Klammern werden immer zuerst berechnet.

Prioritätsebenen

1. Klammer
2. Potenzierung
3. Multiplikation/Division
4. Addition/Subtraktion

3.1.4.2 Zahlenbereiche

Beim Rechnen mit dem C16, C116 oder Plus/4 ist es wichtig zu wissen, welche Zahlen verarbeitet werden können. Der C16, C116 bzw. Plus/4 kann Zahlen mit bis zu neun Vorkommastellen direkt verarbeiten, das heißt zwischen zwischen -999999999 und +999999999 verarbeiten. Überschreitet ein Ergebnis diesen Bereich, wird die sogenannte "wissenschaftliche Notation" verwendet. Ein Beispiel:

```
PRINT 1000000000*20
2E+09
```

Der Ausdruck "2E+09" wird "2 mal 10 hoch 9" gesprochen. Der Exponent neun gibt die Anzahl der Nullen an, die der Zahl zwei folgen. Daher entspricht "2E+09" ausgeschrieben der Zahl "2000000000", das heißt einer zwei mit neun folgenden Nullen.

Bei der Eingabe einer Berechnung ist es äußerst wichtig, daß Sie zur Trennung von Vor- und Nachkommastellen den "Dezimalpunkt" und nicht das Komma verwenden:

Falsch:
```
PRINT 2*4,6
 8             6
```

Richtig:
```
PRINT 2*4.6
 9.2
```

Bei Verwendung eines Kommas werden zwei Zahlen ausgegeben, da ein Komma zur "Formatierung" im PRINT-Befehl verwendet wird. Das Ergebnis des Ausdrucks "2*4" wird in der ersten Bildschirmzone ausgegeben, die folgende Zahl 6 wird in der

zweiten Zone ausgegeben. Der BASIC-Interpreter erkennt nicht, daß "4,6" eine (!) Zahl mit Vor- und Nachkommastellen ist.

Alle folgenden Beispiele können ebenfalls im Direktmodus sofort ausgeführt werden. Auch die Programmform ist möglich, indem den Befehlen eine Zeilennummer vorangestellt und das Programm nach Eingabe der Programmzeile mit RUN gestartet wird.

In den Beispielen wird der PRINT-Befehl zur Demonstration verschiedener Funktionen verwendet. Erinnern Sie sich: PRINT ohne Anführungszeichen liefert Ihnen das Ergebnis eines beliebigen numerischen Ausdrucks. In dem numerischen Ausdruck können alle Grundrechenarten und die verschiedenen Funktionen des C16, C116 und Plus/4 verwendet werden.

3.1.4.3 Die Rechenfunktionen

Die verschiedenen Rechenfunktionen des C16, C116 und Plus/4 werden im folgenden kurz dargestellt. Einige davon werden wir in den folgenden Kapiteln benötigen, wo die betreffenden Funktionen nochmals ausführlich dargestellt werden.

Die Betragsfunktion (ABS)

Die Betragsfunktion ABS berechnet den Betrag einer Zahl, der immer positives Vorzeichen besitzt. Das heißt, negative Zahlen erhalten positives Vorzeichen und positive Zahlen bleiben unverändert.

```
ABS (ZAHL)
```

Beispiele:	`PRINT SQR(5)`	ergibt 5
	`PRINT SQR(-5)`	ergibt 5

Die Vorzeichenfunktion (SGN)

Die Vorzeichenfunktion "Signum" gibt das Vorzeichen einer Zahl an. Das Ergebnis ist immer null (Zahl null), minus eins (negative Zahl) oder plus eins (positive Zahl).

```
SGN(ZAHL)
```

Beispiele:

```
PRINT SGN(-21)      ergibt -1
PRINT SGN(Ø)        ergibt 0
PRINT SGN(123)      ergibt 1
```

Die Integerfunktion (INT)

Die INT-Funktion wird sehr häufig benötigt. INT(ZAHL) berechnet den ganzzahligen Anteil einer beliebigen Zahl, wobei eventuelle Nachkommastellen entfernt werden.

Beispiele:

```
PRINT INT(2Ø.34)      ergibt 20
PRINT INT(1,357623)   ergibt 1
PRINT INT(9/4)        ergibt 2
```

Die Quadratwurzelfunktion (SQR)

Der Begriff "Wurzel ziehen" dürfte Ihnen noch aus der Schule bekannt sein. Wie bereits jeder vernünftige Taschenrechner, so beherrscht auch Ihr C16, C116 oder Plus/4 diese oft benötigte Funktion. Die Syntax des zugehörigen Befehls lautet:

```
SQR (ZAHL)
```

Beispiele:

```
PRINT SQR(4)        ergibt 2
PRINT SQR(25)       ergibt 5
```

Die Exponentialfunktion (EXP)

Die Exponentialfunktion potenziert die Euler'sche Zahl e (2,71828...).

```
EXP(ZAHL)
```

Beispiele:

```
PRINT EXP(1)        ergibt 2.71828183
PRINT EXP(2)        ergibt 7.3890561
```

Die Logarithmusfunktion (LOG)

Die Logarithmusfunktion ist gewissermaßen die Umkehrung der Exponentialfunktion und berechnet den natürlichen Logarithmus einer Zahl zur Basis e.

```
LOG (ZAHL)
```

Beispiele:

```
PRINT LOG(2.71828183) ergibt 1
PRINT LOG(7.3890561) ergibt 2
```

Numerische Ausdrücke können aus beliebigen Kombinationen der genannten Funktionen bestehen. Die Länge eines solchen Ausdrucks und die Anzahl der "Klammerebenen" ist - fast - unbegrenzt.

```
PRINT ((2*3)↑4*LOG(3)+EXP(1.2))/4
 356.78Ø411
```

Wie die folgenden Beispiele zeigen, kann anstelle einer Zahl jeder der beschriebenen Funktionen ein beliebiger numerischer Ausdruck angegegeben werden, in dem sogar selbst wiederum Funktionen enthalten sein dürfen:

1.
```
PRINT INT(2+3.4)
 5
```

2.
```
PRINT SQR(3*2+3)
 9
```

3.
```
PRINT ABS(-1Ø/4+INT(3.231))
 Ø.5
```

3.1.5 Variablen

Der C16, C116 bzw. Plus/4 verfügt über verschiedene Typen von Variablen. Variablen sind "Stellvertreter" für Zahlen oder Zeichenketten, ohne die kein größeres Programm denkbar ist.

3.1.5.1 Fließkommavariablen

Weitere Funktionen werden für unseren BASIC-Kurs vorläufig nicht benötigt. Mit den erworbenen Kenntnissen sind Sie bereits in der Lage, einfache mathematische Programme zu schreiben.

Zum Beispiel könnten wir ein Programm erstellen, das uns Wertpapiererträge berechnet. Ausgehend von einem bestimmten Kapital, einer gegebenen Laufzeit und dem Zinssatz können wir berechnen, welches Endkapital sich ergibt. Folgende "Parameter" (Voraussetzungen) seien gegeben:

Kapital: 2300 DM
Laufzeit: 1 Jahr
Zinssatz: 3.5%

Zur Berechnung verwenden wir folgendes Programm (löschen Sie zuerst den Speicher mit NEW):

```
100 PRINT "KAPITAL: 2300 DM"
110 PRINT "LAUFZEIT: 1 JAHR"
120 PRINT "ZINSSATZ: 3.5%"
130 PRINT
140 PRINT "ZINSEN:";2300/100*3.5
150 PRINT "GESAMTKAPITAL:";2300+2300/100*3.5
```

Nach dem Starten des Programms mit RUN erhalten wir als Ergebnis:

```
RUN
KAPITAL: 2300 DM
LAUFZEIT: 1 JAHR
ZINSSATZ: 3.5%

ZINSEN: 80.5
GESAMTKAPITAL: 2380.5

READY.
```

Die Programmzeilen 100-120 geben zur Erinnerung die "Ausgangsparameter" auf dem Bildschirm aus. Der PRINT-Befehl in Zeile 130 bewirkt einen "Zeilenvorschub", das heißt er fügt eine Leerzeile zwischen den bisherigen Ausgaben und den folgenden Ergebnissen ein.

In den Zeilen 140 und 150 werden die sich ergebenden Zinsen und das Gesamtkapital (Ausgangskapital + Zinsen) ausgegeben. Wenn Sie diese Ausgabe optisch verbessern wollen, ändern Sie die Zeilen wie folgt ab und starten Sie das Programm erneut mit RUN.

```
140 PRINT "ZINSEN:";2300/100*3.5;"DM"
150 PRINT "GESAMTKAPITAL:";2300+2300/100*3.5;"DM"
```

Die geänderten Programmzeilen geben nach der Ausgabe der Ergebnisse hinter beiden Zahlen zusätzlich die Zeichenkette "DM" aus. Wie Sie erneut sehen, kann in einem PRINT-Befehl die Ausgabe von Zeichenketten (in Anführungszeichen) und Ergebnissen von Berechnungen (ohne Anführungszeichen) beliebig gemischt werden.

Dieses Programm funktioniert zwar einwandfrei, es bietet jedoch kaum eine Erleichterung. Die Berechnung kann ebensogut mit einem Taschenrechner durchgeführt werden. Der Grund liegt im zu starren Einsatzgebiet des Programms. Es ist in der Lage, eine bestimmte Aufgabe zu lösen, die Berechnung von Zinsen und Endkapital bei einem einzigen Typ von Wertpapier.

Die Aufgabe von Programmen besteht jedoch meist darin, nicht nur eine bestimmte Aufgabe, sondern einen Aufgaben-Typ (!) zu lösen. In unserem Fall ist das Programm erst sinnvoll, wenn es uns die Berechnung beliebiger Wertpapiere ermöglicht, mit verschiedenen Zinssätzen, verschiedener Laufzeit und unterschiedlichem Anlagekapital.

Dieses Problem kann nur mit Hilfe von "Variablen" gelöst werden. Meine Lehrer bezeichneten Variablen als "Statthalter", ein treffender Ausdruck, wie ich finde. In der Tat ist eine Variable ein Statthalter oder Stellvertreter für eine Zahl, die noch nicht feststeht.

Zur Berechnung der Zinsen wurde der Ausdruck verwendet:

```
140 PRINT "ZINSEN:";2300/100*3.5;"DM"
```

Ersetzen Sie diesen Ausdruck bitte durch:

```
140 PRINT "ZINSEN:";2300/100*X
```

Ersetzen Sie bitte zusätzlich den Ausdruck:

```
150 PRINT "GESAMTKAPITAL:";2300+2300/100*3.5;"DM"
```

durch:

```
150 PRINT "GESAMTKAPITAL:";2300+2300/100*X;"DM"
```

Der zuvor festgelegte Prozentsatz (3.5%) wird in beiden Programmzeilen durch eine Variable mit der Bezeichnung X ersetzt. Geben Sie nun bitte eine neue Programmzeile ein:

```
10 X=3.5
```

"Listen" Sie das Programm. Wenn Ihre Änderungen korrekt sind, werden folgende Zeilen ausgegeben:

```
LIST

10 X=3.5:REM X WIRD DER WERT 3.5 ZUGEWIESEN
100 PRINT "KAPITAL: 2300 DM"
110 PRINT "LAUFZEIT: 1 JAHR"
120 PRINT "ZINSSATZ:";X;"%"
130 PRINT
140 PRINT "ZINSEN:";2300/100*X;"DM"
150 PRINT "GESAMTKAPITAL:";2300+2300/100*X;"DM"

READY.
```

Starten Sie das Programm mit RUN. Die ausgegebenen Ergebnisse sollten exakt der vorigen Programmversion entsprechen. Ist dies nicht der Fall, "listen" Sie das Programm erneut und korrigieren Sie eventuelle Tippfehler.

Wenn Ihre Schulzeit bereits länger zurückliegt, werden Sie sich sicherlich fragen, wieso der Ausdruck "2300/100*X" das gleiche Ergebnis ergibt wie "2300/100*3.5". X ist eine Variable, ein Statthalter für einen Wert. Sinnvoll wird die Verwendung von Variablen erst durch "Wertzuweisungen". Eine solche Zuweisung

erfolgt in Zeile 10. Der Ausdruck "X=3.5" weist der Variablen X den Wert 3.5 zu. Da der BASIC-Interpreter über verschiedene Typen von Variablen verfügt, sollten Sie sich merken, daß es sich bei den momentan verwendeten Variablen um sogenannte "Fließkommavariablen" handelt.

Immer dann, wenn der BASIC-Interpreter nach einer solchen Zuweisung auf die Variable X trifft, wird diese Variable durch den Wert 3.5 ersetzt, der Ihr zugewiesen wurde. X "steht" für den Wert 3.5.

Dieses Prinzip der Wertzuweisung an eine Variable und der anschließenden Verwendung dieser Variablen können Sie sehr leicht mit im Direkt-Modus gegebenen Befehlen ausprobieren:

```
X=1Ø:PRINT X
 1Ø

READY.
```

Wie erläutert, können (sowohl im Direkt- als auch im Programm-Modus) in einer Zeile mehrere Befehle eingegeben werden, wenn sie mit Doppelpunkten voneinander getrennt werden.

Der erste Befehl weist der Variablen X den Wert 10 zu. Der zweite Befehl weist den BASIC-Interpreter, den Wert (!) der Variablen X auszugeben, eben 10. Beachten Sie, daß PRINT ohne Anführungszeichen einen Wert (!) ausgibt, keine Zeichenkette (im Unterschied zu PRINT "X").

"Offiziell" befindet sich vor der eigentlichen Zuweisung das Wort LET (LET X=10). Da das Wort LET jedoch im Grunde überflüssig ist, wird es in der Praxis meist weggelassen. Im gesamten Buch werde ich LET daher niemals gebrauchen. Ich denke, die Kurzform der Zuweisung ohne LET (X=10) ist verständlicher als mit diesem Befehl (LET X=10).

Wertzuweisungen an eine Variable können auch Ergebnisse von Berechnungen sein, wie folgende Beispiele demonstrieren:

1.
```
X=2+3:PRINT X
 5
```

2.
```
X=1Ø*2:PRINT X
 2Ø
```

3.
```
X=5*2/3:PRINT X
 3.33333333
```

Diese Beispiele zeigen zugleich, daß jede neue Zuweisung an eine Variable den alten Wert der Variablen "überschreibt". Nach Eingabe von Beispiel 1 erhielt X den Wert fünf. Dieser Wert wurde durch die im Beispiel 2 vorgenommene neue Zuweisung überschrieben, X erhielt den neuen Wert 20. Durch das dritte Beispiel wurde auch dieser Wert überschrieben und X der Wert 3.33333333 zugewiesen.

Eine Zuweisung an eine Variable bleibt nur solange erhalten, bis eine neue Zuweisung zur gleichen Variablen erfolgt.

In der Schule verwendeten Sie oftmals mehrere Variablen gleichzeitig, zum Beispiel X, Y und Z. Auch der C16, C116 oder Plus/4 gestattet die gleichzeitige Verwendung mehrerer Variablen, die anhand ihrer Bezeichnung, des "Variablennamens", voneinander unterschieden werden.

Ein Variablenname kann theoretisch aus beliebig vielen Buchstaben bestehen. Außer Buchstaben dürfen auch Zahlen zur Bezeichnung verwendet werden. Das erste Zeichen eines Variablennamens muß (!) jedoch ein Buchstabe sein.

Variablennamen:	Korrekt	Falsch
	X	1
	Y	5
	XA	2X
	ZINS	1ZINS
	XAB1	1XAB

Meine Behauptung, zur Bezeichnung von Variablen seien beliebig viele Zeichen zulässig, muß leider eingeschränkt werden. Den BASIC-Interpreter des C16, C116 und Plus/4 interessieren nur die beiden ersten Zeichen eines Variablennamens. Daher sind die Variablen ZINS und ZINSEN für ihn identisch.

Daß diese eingeschränkte Unterscheidung sehr problematisch ist, zeigt folgendes Beispiel:

```
ZINS=5:ZINSEN=20:PRINT ZINS,ZINSEN
 20          20
```

Der Variablen ZINS wird der Wert 5 und der Variablen ZINSEN der Wert 20 zugewiesen. Mit PRINT werden anschließend die Werte beider Variablen ausgegeben.

Der ausgegebene Wert ist in beiden Fällen der gleiche, da der BASIC-Interpreter ZINS und ZINSEN wie eine (!) Variable behandelt (die beiden ersten Zeichen des Variablennamens sind identisch). Dieser Variablen wird der Wert fünf zugewiesen und dieser Wert anschließend durch die erneute Zuweisung des Wertes 20 an die gleiche Variable überschrieben.

Ein Gegenbeispiel:

```
ZINS=5:BETRAG=20:PRINT ZINS,BETRAG
 5           20
```

ZINS und BETRAG sind zwei verschiedene Variablen. Durch die Zuweisung des Wertes 20 an die Variable BETRAG bleibt der Wert 5, der der Variablen ZINS zugewiesen wurde, unverändert.

Achten Sie bitte in Ihren Programmen auf diese "Falle" des BASIC-Interpreters. Sie könnten nun einwenden, daß sich dieses Problem leicht umgehen ließe, wenn - wie in der Schule meist üblich - nur ein Buchstabe zur Bezeichnung einer Variablen verwendet wird (A, B,...,X, Y, Z).

Die Verwendung längerer Variablennamen bietet jedoch den Vorteil der besseren Übersichtlichkeit von Programmen. Die Programmzeile:

```
140 PRINT "ZINSEN:";2300/100*X;"DM"
```

ist sicherlich schlechter zu verstehen als:

```
140 PRINT "ZINSEN:";2300/100*ZINS;"DM"
```

Die zweite Variante ist auf Anhieb verständlich, da ein Rätseln über den Sinn der Variablen X entfällt. Die Variable ist "selbsterklärend". Sie wissen sofort, daß ZINS stellvertretend für einen bestimmten Zinssatz ist.

Bestimmte Variablenbezeichnungen sind leider verboten, die sogenannten "reservierten" Bezeichnungen. Bei diesen reservierten Bezeichnungen handelt es sich um alle BASIC-Befehlsworte (wie PRINT, RUN, DO, IF etc.) und einige Variablen mit festgelegten Bezeichnungen, die für spezielle Zwecke verwendet werden (ST, TI, TI$, DS, DS$, ER, EL, ERR$).

Reservierte Bezeichnungen dürfen keinesfalls in einem Variablennamen enthalten sein, wie folgendes Beispiel zeigt.

```
DIFF=10

?SYNTAX ERROR
READY.
```

Die Variablenbezeichnung DIFF ist unzulässig, da das BASIC-Befehlswort IF enthalten ist.

Im diesem Buch werde ich ausschließlich Variablennamen verwenden, deren Funktion bereits durch ihre Bezeichnung deutlich wird.

Um dieses Prinzip sofort zu verwirklichen, sollte unser "Zinsberechnungsprogramm" ein weiteres Mal geändert werden. Ändern Sie die Zeilen 10, 140 und 150.

```
10 ZINS=3.5:REM ZUWEISUNG DES WERTE 3.5 AN ZINS
120 PRINT "ZINSSATZ:";ZINS;"%"
140 PRINT "ZINSEN:";2300/100*ZINS;"DM"
150 PRINT "GESAMTKAPITAL:";2300+2300/100*ZINS;"DM"
```

Fassen Sie diese wiederholten Änderungen bitte nicht als Schikane auf, sondern sehen Sie darin ein willkommene Gelegenheit, den Umgang mit den Editiertasten zu üben.

Wenn Sie das Programm nach erfolgter Korrektur erneut mit RUN starten, werden selbstverständlich die gleichen Zahlen wie zuvor ausgegeben, da der Programmablauf (!) unverändert bleibt.

In diesen Erläuterungen zum Umgang mit Variablen wurde bisher wenig über ihre Vorteile gesagt. Wenn Sie sich die aktuelle Version des "Zinsprogramms" vor Augen halten, werden die Vorteile sichtbar.

```
10 ZINS=3.5:REM ZUWEISUNG
100 PRINT "KAPITAL: 2300 DM"
110 PRINT "LAUFZEIT: 1 JAHR"
120 PRINT "ZINSSATZ:";ZINS;"%"
130 PRINT:REM BEWIRKT EINE LEERZEILE
140 PRINT "ZINSEN:";2300/100*ZINS;"DM"
150 PRINT "GESAMTKAPITAL:";2300+2300/100*ZINS;"DM"
```

Die Variable ZINS steht stellvertretend für den Zinssatz, der in die beiden Formeln eingesetzt werden soll. Unser Programm ist nun bereits erheblich flexibler, wenn Sie bedenken, daß die Änderung von Zeile zehn - der Wertzuweisung an die Variable ZINS - ausreicht, um die Berechnung für ein Wertpapier mit völlig anderem Zinssatz durchzuführen.

Wenn Sie Zeile zehn zum Beispiel in "10 ZINS=7.2" ändern, erhalten Sie für ein Wertpapier mit einem Zinssatz von 7.2% folgende Ergebnisse:

```
ZINSEN: 165.6 DM
GESAMTKAPITAL: 2465.6 DM
```

Beide Berechnungen können Sie durch Ändern von Zeile zehn und dem Starten des Programms mit RUN beliebig oft für die verschiedensten Zinssätze durchspielen. Mit diesem Programm demonstriert der C16, C116 oder Plus/4 zum ersten Mal seine Überlegenheit gegenüber einem einfachen Taschenrechner.

Außer der Arbeitserleichterung (die Änderung einer Zahl genügt, um zwei (!) Berechnungen zu beeinflussen) ergibt sich ein weiterer Vorteil:

Ein Programm, das einmal geschrieben wurde und völlig fehlerfrei ist, ermöglicht immer wieder die korrekte Lösung des jeweiligen Aufgaben-Typs. Werden die Berechnungen dagegen dutzende Male mit verschiedenen Zinssätzen und einem Taschenrechner ausgeführt, ist die Wahrscheinlichkeit recht hoch, daß die beiden Formeln ab und zu falsch eingetippt werden und die Ergebnisse entsprechend unkorrekt sind.

Variablen sollten in Programmen immer dann anstelle von "Konstanten" (festen Zahlen) eingesetzt werden, wenn diese Zahlen sich ändern können. In Zweifelsfällen sollte der Verwendung von Variablen der Vorzug gegeben werden. Welche Arbeitserleichterung Variablen ermöglichen, verdeutlichen zwei Programmbeispiele, in denen zu verschiedenen Produkten 14% Mehrwertsteuer addiert werden sollen.

Version 1:

```
100 PRINT 1000+100/100*14
110 PRINT 13.45+13.45/100*14
120 PRINT 1241.4567+1241.4567/100*14
130 PRINT 0.56+0.56/100*14
```

Angenommen, der Mehrwertsteuersatz erhöht sich von 14% auf 15%. Um die Berechnungen der neuen Lage anzupassen, müssen Sie jede einzelne Programmzeile ändern. Im der zweiten Version dieses Programms genügt die Änderung einer Programmzeile.

Version 2:

```
10 MWST=14
100 PRINT 1000+100/100*MWST
110 PRINT 13.45+13.45/100*MWST
120 PRINT 1241.4567+1241.4567/100*MWST
130 PRINT 0.56+0.56/100*MWST
```

Dieses Programm ist einer Änderung des Mehrwertsteuersatzes jederzeit leicht anzupassen, indem der Variablen MWST in Zeile zehn der neue Wert zugewiesen wird.

Das Zinsberechnungsprogramm kann noch flexibler gestaltet werden, wenn auch für das eingesetzte Kapital eine Variable KAPITAL verwendet wird.

```
10 ZINS=3.5
20 KAPITAL=2300
100 PRINT "KAPITAL: 2300 DM"
110 PRINT "LAUFZEIT: 1 JAHR"
120 PRINT "ZINSSATZ:";ZINS;"%"
130 PRINT
140 PRINT "ZINSEN:";KAPITAL/100*ZINS;"DM"
150 PRINT "GESAMTKAPITAL:";KAPITAL+KAPITAL/100*ZINS;"DM"
```

Der Variablen "KAPITAL" wird in Zeile 20 der Wert 2300 zugewiesen, der bisher für das angelegte Kapital verwendet wurde. In den Zeilen 140 und 150 wird der Wert 2300 durch diese Variable ersetzt.

Das Programm erlaubt nun die einfache Änderung von Zinssatz und Kapitaleinsatz durch Änderung der Zuweisungen in Zeile zehn beziehungsweise 20.

3.1.5.2 Der Eingabebefehl INPUT

Die Befehle INPUT und GET bilden eine Art "Gegenstück" zu dem Ausgabebefehl PRINT. Beide sind sogenannte "Eingabebefehle". Die allgemeine Syntax des INPUT-Befehls, das "Befehlsformat", lautet:

```
INPUT ("STRING";) VARIABLE
```

Ebenso wie bei allen folgenden Befehlserläuterungen sind Angaben in Klammern "optional", das heißt, derartige Angaben können (!) vorgenommen werden, sind jedoch nicht verpflichtend.

INPUT kann ebenso wie GET nur (!) im Programm-Modus verwendet werden. Wenn Sie INPUT im Direkt-Modus verwenden, zum Beispiel den Befehl:

```
INPUT X
```

eingeben, gibt der BASIC-Interpreter die Fehlermeldung "?ILLEGAL DIRECT ERROR" auf dem Bildschirm aus, die besagt, daß INPUT nicht im Direkt-Modus verwendet werden darf. Sie sehen, nicht jeder Befehl ist in beiden Modi anwendbar.

Um die Wirkungsweise von INPUT zu erforschen, löschen Sie bitte das Zinsprogramm durch Eingabe von NEW und ersetzen Sie es durch das folgende Beispielprogramm.

```
100 INPUT ZINS:REM EINGABE UND ZUWEISUNG AN ZINS
110 PRINT ZINS:REM AUSGABE DES WERTES VON ZINS
```

Im Beispiel wird INPUT ohne die optionale Angabe eines Strings verwendet. Wenn Sie das Programm mit RUN starten, erhalten Sie eine "Eingabeaufforderung" auf dem Bildschirm:

```
RUN
?
```

Das Fragezeichen, hinter dem der Cursor blinkt, fordert Sie zu einer Eingabe auf. Geben Sie nun bitte eine beliebige Zahl ein, zum Beispiel 23, und drücken Sie zum Abschluß der Eingabe RETURN. Die eingegebene Zahl 23 wird in der nächsten Bildschirmzeile ausgegeben:

```
RUN
? 23
 23

READY.
```

Der INPUT-Befehl ist nicht einfach zu verstehen, da der bei der Eingabe blinkende Cursor vortäuscht, der Programmablauf sei bereits beendet (Sie haben sicherlich bemerkt, daß der Cursor während eines Programmablaufs verschwindet).

Der Programmablauf ist jedoch nicht beendet, sondern nur vorübergehend unterbrochen. INPUT hat folgende Wirkung: Wenn der BASIC-Interpreter in einem Programm auf diesen Befehl trifft, wird das Programm "angehalten" und erst dann fortgesetzt, wenn der Benutzer eine Eingabe vorgenommen und mit der RETURN-Taste abgeschlossen hat. Die Eingabe des Benutzers wird der Variablen zugewiesen, die im INPUT-Befehl angegeben wurde.

Im dem Demoprogramm hieß diese Variable ZINS. Wenn Sie die Zahl 23 eingeben und RETURN betätigen, wird diese eingegebene Zahl der Variablen ZINS zugewiesen, ZINS erhält den Wert 23.

Der folgende PRINT-Befehl in Zeile 110 gibt den Wert von ZINS auf dem Bildschirm aus, im Beispiel daher den Wert 23.

Sie sehen, Programme werden erst mit Hilfe des INPUT-Befehls wirklich flexibel. Dieser Befehl erlaubt dem Benutzer des Programms eine echte Einflußnahme auf das laufende Programm. Ohne INPUT ist zum Beispiel ein Programm zur Verwaltung von Adressen kaum vorstellbar, denn wie sollte der Benutzer sonst in der Lage sein, Namen, Straßen und Wohnorte einzugeben?

Um mit dieser Möglichkeit, während eines Programmlaufs einer Variablen Werte über die Tastatur zuzuweisen, vertraut zu werden, empfehle ich Ihnen, das Programm mehrmals mit RUN neu zu starten und jeweils andere Zahlen einzugeben, wenn die Eingabeaufforderung erscheint.

Sie werden sehen, daß immer die von Ihnen eingegebene Zahl auf dem Bildschirm erscheint, also tatsächlich die jeweils eingegebene Zahl der Variablen ZINS zugewiesen wurde.

Übrigens: Wir sahen bei der Eingabe von Befehlszeilen, daß der BASIC-Interpreter nur Zeilen bis zu einer Länge von 88 Zeichen bearbeiten kann. Diese Beschränkung gilt auch für den INPUT-Befehl. Die Eingaben des Benutzers dürfen keinesfalls länger sein als 88 Zeichen.

So wie er bisher angewendet wurde, hat der INPUT-Befehl einen Nachteil: Auf dem Bildschirm erscheint ein Fragezeichen und der Benutzer soll eine Zahl eingeben. In einem Programm, in dem mehrere Zahlen einzugeben sind, sollte der Benutzer jedoch auch darauf hingewiesen werden, um welche Zahl es sich jeweils handelt, ob zum Beispiel der Zinssatz, die Laufzeit oder das Kapital einzugeben ist.

Bei der Erläuterung der "Syntax" des INPUT-Befehls wies ich darauf hin, daß optional ein String, also eine Folge von Zeichen in Anführungszeichen, angegeben werden kann. Angegebene Zeichen werden vor dem Fragezeichen auf dem Bildschirm ausgegeben, ähnlich wie bei dem PRINT-Befehl. Beachten Sie jedoch, daß in diesem Fall hinter den zweiten Anführungszeichen

und dem Variablennamen ein Semikolon eingefügt werden muß, wie im folgenden Beispiel:

```
100 INPUT "BITTE ZINSSATZ EINGEBEN";ZINS
110 PRINT ZINS
```

Ändern Sie bitte die Programmzeile 100 gemäß dem abgebildeten Programm und starten Sie das Programm mit RUN. Wieder werden Sie zur Eingabe einer Zahl aufgefordert, diesmal wird Ihnen jedoch mitgeteilt, welche Zahl einzugeben ist:

```
RUN
BITTE ZINSSATZ EINGEBEN? 16
 16

READY.
```

Die eingegebene Zahl (im Beispiel 16) wird wie zuvor der Variablen ZINS zugewiesen und der Wert dieser Variablen durch den folgenden PRINT-Befehl ausgegeben.

Dank der möglichen Angabe eines Strings kann der INPUT-Befehl sehr flexibel als eine Art "kombinierter Ein-/Ausgabebefehl" verwendet werden, zur Ausgabe einer beliebigen Zeichenkette, die dem Benutzer exakt mitteilt, welche Eingabe das Programm von ihm erwartet.

Dank INPUT kann das Zinsprogramm noch flexibler gestaltet werden. Um Wertpapiere mit anderem Zinssatz oder aber einen geänderten Kapitaleinsatz zu berücksichtigen, war bisher die Änderung von Programmzeilen erforderlich. Die Zuweisungen in den Zeilen 10 beziehungsweise 20 mußten geändert werden.

INPUT bietet uns die Möglichkeit, das Programm so zu ändern, daß die Werte für den Zinssatz und das eingesetzte Kapital vom Benutzer nach dem Start des Programms eingegeben werden.

```
10 INPUT "ZINSSATZ";ZINS
20 INPUT "KAPITALEINSATZ";KAPITAL
30 PRINT
100 PRINT "LAUFZEIT: 1 JAHR"
110 PRINT "ZINSSATZ:";ZINS
100 PRINT "KAPITAL:";KAPITAL
130 PRINT
140 PRINT "ZINSEN:";KAPITAL/100*ZINS;"DM"
150 PRINT "GESAMTKAPITAL:";KAPITAL+KAPITAL/100*ZINS;"DM"
```

Geben Sie dieses Programm, das sich doch erheblich von der alten Version unterscheidet, am besten komplett neu ein, das heißt, löschen Sie den Programmspeicher komplett durch Eingabe von NEW.

"Listen" Sie das Programm anschließend und korrigieren Sie eventuelle Tippfehler, bevor Sie es mit RUN starten. Nach dem Start verlangt das Programm von Ihnen die Eingabe des Zinssatzes. Nach Beenden der Eingabe mit RETURN werden Sie aufgefordert, den Kapitaleinsatz einzugeben.

Diese beiden Werte und die Laufzeit (ein Jahr) werden zur Kontrolle mit PRINT ausgegeben, bevor in den Zeilen 140 und 150 die Zinsen und das Endkapital berechnet und ebenfalls ausgegeben werden.

```
RUN
ZINSSATZ? 3.4
KAPITALEINSATZ? 2000

LAUFZEIT: 1 JAHR
ZINSSATZ: 3.4
KAPITAL: 2000

ZINSEN: 68 DM
GESAMTKAPITAL: 2068 DM

READY.
```

Wenn Sie wie im Beispiel als Zinssatz 3.4% und als eingesetztes Kapital 2000 DM eingeben, erhalten Sie als Ergebnis 68 DM an Zinsen und 2068 DM als Gesamtkapital nach einem Jahr.

Sie können das Programm anschließend sofort wieder mit RUN starten und sich die entsprechenden Zahlen für Wertpapiere mit anderem Zinssatz und bei anderem Kapitaleinsatz ausgeben lassen. Sie besitzen nun das erste wirklich nützliche und flexible Programm. Die im Programm verwendeten Werte sind nicht festgelegt, sondern werden erst während des Programmablaufs von Ihnen eingegeben.

Der entscheidende Unterschied zur ersten Version des Zinsprogramms besteht darin, daß diese nur eine spezielle Aufgabe lösen konnte, das vorliegende Programm jedoch für die verschiedensten Aufgaben des gleichen Typs eingesetzt werden kann.

3.1.5.3 Integervariablen

Der C16, C116 bzw. Plus/4 besitzt zwei verschiedene Variablentypen, denen Zahlen zugewiesen werden können, die "Fließkommavariablen" und die "Integervariablen". Bisher wurden von uns ausschließlich Fließkommavariablen verwendet.

Diesem Variablentyp können beliebige Zahlen zugewiesen werden, positive, negative, ganze und gebrochene Zahlen.

Im Gegensatz zu den Fließkommavariablen besitzen Integervariablen einen erheblich eingeschränkteren "Wertebereich". Dieser Variablentyp kann nur ganze Zahlen im Bereich zwischen -32768 und +32767 aufnehmen.

Integervariablen werden durch das dem eigentlichen Variablennamen folgende Zeichen "%" gekennzeichnet, zum Beispiel A% oder XY%. Aufgrund des geringeren Wertebereiches ist eine Zuweisung wie zum Beispiel X%=100000 nicht möglich. Die Zahl 100000 ist zu groß, um einer Integervariablen zugewiesen zu werden.

Nun werden Sie sich fragen, wozu dieser Variablentyp überhaupt benötigt wird, da er doch offensichtlich keinerlei Vorteile gegenüber den Fließkommavariablen besitzt.

Der Unterschied wird erst bei der Besprechung der sogenannten "Variablenarrays" sichtbar werden. Diese Arrays bestehen aus einer Vielzahl einzelner Variablen und benötigen daher immens viel von dem kostbaren Speicherplatz im Rechner. Erst wenn Arrays benötigt werden, zeigen Integervariablen ihre Vorteile, nämlich den geringen Platzbedarf. Ein Array, das aus Integervariablen besteht, benötigt erheblich weniger Platz als das gleiche Array aus Fließkommavariablen.

Ich empfehle Ihnen, sich sofort mit diesen verschiedenen Variablentypen "anzufreunden". Schreiben Sie mehrere kleine Programme und verwenden Sie einmal Fließkomma- und einmal Integervariablen. Denken Sie daran, daß bei einer Integervariablen dem Variablennamen immer ein Prozentzeichen folgt.

3.1.5.4 Mehrere Eingaben mit einem INPUT-Befehl

INPUT bietet die Möglichkeit, mit einem Befehl mehrere Eingaben des Benutzers verschiedenen Variablen zuzuweisen. Die angegebenen Variablen müssen durch Kommata voneinander getrennt werden. Mehrfachzuweissungen sind wiederum mit oder ohne Kommentare möglich.

```
INPUT (VARIABLE1),(VARIABLE2),...
```

oder

```
INPUT "(STRING)";(VARIABLE1),(VARIABLE2),...
```

Eine Mehrfachzuweisung ist nur dann möglich, wenn der Benutzer wie im folgenden Beispiel seine Eingaben ebenfalls mit Kommata voneinander trennt.

```
100 INPUT "DREI ZAHLEN EINGEBEN";X,Y,Z
110 PRINT X,Y,Z

RUN
DREI ZAHLEN EINGEBEN? 3,612,311.23
3       612       311.23

READY.
```

Achten Sie bitte darauf, daß sich die Anzahl der Eingaben des Benutzers und der angegebenen Variablen entsprechen. Soll der Benutzer drei durch Kommas getrennte Zahlen eingeben, ist im INPUT-Befehl die Angabe von ebenfalls drei durch Kommas getrennten numerischen Variablen erforderlich.

Numerische Variablen können auch Ergebnisse aufnehmen, das heißt, Ausdrücke wie Z=2*X+Y sind zulässig. Dieser Ausdruck weist der Variablen Z das Ergebnis einer Berechnung zu, in der Variablen und Konstanten gemischt vorkommen.

```
100 INPUT "ZAHL 1";Z1:REM EINGABE AN Z1 ZUWEISEN
110 INPUT "ZAHL 2";Z2:REM EINGABE AN Z2 ZUWEISEN
120 SU=Z1+Z2:REM ADDIEREN U.ZUWEISUNG AN SU
130 DI=Z1-Z2:REM SUBTRAKTION U.ZUWEISUNG AN DI
140 PR=Z1*Z2:REM MULTIPLIKATION U.ZUWEISUNG AN PR
150 PRINT:REM LEERZEILE
160 PRINT SU,DI,PR:REM VARIABLENINHALTE AUSGEBEN

RUN
ZAHL1? 20
ZAHL2? 5

25      15      100

READY.
```

Dieses Demoprogramm fordert Sie zur Eingabe zweier Zahlen auf, die den Variablen Z1 und Z2 zugewiesen werden. In den Zeilen 120-140 werden diese Variablen (das heißt die Werte, die ihnen zugewiesen werden) addiert, subtrahiert und multipliziert.

Die Ergebnisse der Berechnungen werden jeweils einer weiteren Variablen zugewiesen (SU, DI, PR). In Zeile 160 werden die Inhalte dieser Variablen (also die Ergebnisse der Berechnungen) mit PRINT ausgegeben.

In Programmen werden Variablen bei arithmetischen Operationen häufig nicht nur mit Konstanten (Z=X*5) oder anderen Variablen (Z=X*Y) verknüpft, sondern mit sich selbst (Z=2*Z)!

Dieser Vorgang ist nicht leicht zu verstehen. Im Beispiel Z=2*Z wird der Variablen Z der Wert zugewiesen, der sich aus der Multiplikation des alten (!) Wertes von Z mit zwei ergibt.

Wird in einer Zuweisung ein Ausdruck verwendet, in dem die "Zuweisungs-Variable" selbst enthalten ist, verwendet der BASIC-Interpreter bei der Berechnung des Ausdrucks immer den momentanen Wert der Variablen!

```
100 X=1:REM WERT VON X: 1
110 PRINT X
120 X=X+1:REM WERT VON X: 2
130 PRINT X
140 X=X+1:REM WERT VON X: 3
150 PRINT X

RUN
1
2
3

READY.
```

Dieses Beispiel verdeutlicht den Mechanismus: In Zeile 100 wird der Variablen X der Wert 1 zugewiesen. Dieser Wert wird in Zeile 110 ausgegeben.

In Zeile 120 wird X der momentane Wert von X (1) plus eins zugewiesen, also der Wert zwei und anschließend ebenfalls ausgegeben.

In Zeile 140 erfolgt wiederum eine Zuweisung, in der X der momentane Wert dieser Variablen, nun also eine 2, zugewiesen wird und zu diesem Wert 1 addiert wird. Der ausgegebene Wert beträgt daher 3.

Dieses Programm können Sie beliebig erweitern. Jede neue Zuweisung erhöht den Wert von X um 1.

3.1.5.5 Stringvariablen

Einer der wesentlichsten Unterschiede zwischen einem Taschenrechner und einem Computer besteht darin, daß Taschenrechner nur mit Zahlen umgehen können, Computer dagegen mit beliebigen Zeichen (über den Umweg der "Codierung" von Zeichen entsprechend der erläuterten Tabelle).

Wie wir sahen, erleichtern numerische Variablen den Umgang mit Zahlen ganz erheblich. Außer den numerischen Variablen existieren sogenannte "Stringvariablen", die den Umgang mit Zeichenketten vereinfachen.

Numerische Variablen sind Statthalter für Zahlen, die ihnen zugewiesen werden. Einer Stringvariablen wird eine Zeichenkette - ein "String" - zugewiesen. Stringvariablen werden von numerischen Variablen durch das Zeichen "$" (Dollarzeichen) hinter dem Variablennamen unterschieden. Ansonsten gelten für den Variablennamen die gleichen Regeln wie für numerische Variablen.

Geben Sie ein:

```
X$="DIES IST EIN TEST":PRINT X$
```

Dieser im Direkt-Modus gegebene Befehl weist der Stringvariablen X$ die Zeichenkette "DIES IST EIN TEST" zu. Der folgende PRINT-Befehl gibt den Inhalt der Zeichenkette auf dem Bildschirm aus.

Denken Sie beim Umgang mit Stringvariablen immer daran, daß die jeweilige Variable stellvertretend für die zugewiesene Zeichenkette steht, daß Sie also mit der Variablen ebenso wie mit der Zeichenkette selbst umgehen können.

```
X$="DIES IST":Y$="EIN TEST":PRINT X$,Y$
```

Um die Wirkungsweise dieses Befehls zu verstehen, ersetzen Sie bitte in Gedanken die im PRINT-Befehl verwendeten Stringvariablen X$ und Y$ durch die jeweiligen Zeichenketten, die den Variablen zugewiesen wurden. Sie sehen, daß der Befehl ohne Verwendung von Variablen wie folgt geschrieben werden kann:

```
PRINT "DIES IST","EIN TEST"
```

Der Befehl INPUT wurde bisher dazu verwendet, den Benutzer zur Eingabe einer Zahl aufzufordern, die anschließend im Programm weiterverarbeitet werden sollte.

Mit INPUT kann der Benutzer auch zur Eingabe einer Zeichenkette aufgefordert werden. Die eingegebene Zeichenkette weist der INPUT-Befehl der angegebenen Stringvariablen zu, mit der im weiteren Programmablauf gearbeitet werden kann. Geben Sie bitte versuchsweise die folgenden Programmbeispiele ein und starten Sie die Programme mit RUN. Löschen Sie den Speicher jeweils mit NEW, bevor Sie ein neues Programm eingeben.

Beispiel 1:

```
100 INPUT "ZEICHENKETTE EINGEBEN";X$
110 PRINT X$

RUN
? DIES IST EIN C16/PLUS4
DIES IST EIN C16/PLUS4

READY.
```

Beispiel 2:

```
100 INPUT "VORNAME";VOR$
110 INPUT "NACHNAME";NACH$
120 PRINT
130 PRINT "IHR VORNAME LAUTET ";VOR$
140 PRINT "IHR NACHNAME LAUTET ";NACH$
150 PRINT "IHR VOLLSTAENDIGER NAME LAUTET ";VOR$;" ";NACH$

RUN
VORNAME? OSKAR
NACHNAME? MAIER

IHR VORNAME LAUTET OSKAR
IHR NACHNAME LAUTET MAIER
IHR VOLLSTAENDIGER NAME LAUTET OSKAR MAIER

READY.
```

In beiden Demoprogrammen wird das Programm bei Abarbeitung eines INPUT-Befehls wie gewohnt unterbrochen und Sie werden zur Eingabe aufgefordert. Geben Sie die in den Beispielen verwendeten Zeichenketten ein und beenden Sie die Eingabe mit RETURN.

Im ersten Programm wird die Zeichenkette der Variablen X$ zugewiesen und der Inhalt dieser Variablen mit PRINT ausgegeben.

Im zweiten Beispiel geben Sie bitte Ihren Vornamen ein, beenden Sie die Eingabe und geben Sie auf die folgende Aufforderung hin Ihren Nachnamen ein.

Wenn diese Eingabe ebenfalls mit RETURN beendet wurde, werden die beiden Stringvariablen VN$ und NN$ (VN$ enthält den eingegebenen Vornamen, NN$ den eingegebenen Nachnamen) vom Programm weiterverarbeitet.

Beim Umgang mit Stringvariablen sind mehrere Besonderheiten zu berücksichtigen.

"Addition" von Zeichenketten

Zeichenketten können "addiert" werden. Diese "Addition" verkettet zwei Strings miteinander, wie das folgende Beispiel zeigt.

```
100 INPUT "VORNAME";VN$
110 INPUT "NACHNAME";NN$
120 NAME$=VN$+NN$:REM NAME=VORNAME+NACHNAME
130 PRINT NAME$

RUN
VORNAME? OSKAR
NACHNAME? MAIER
OSKARMAIER
```

Der Stringvariablen NAME$ wird in Zeile 120 eine Zeichenkette zugewiesen, die aus dem Inhalt der Variablen VN$, der Zeichenkette "OSKAR", und dem Inhalt der Variablen NN$, der Zeichenkette "MAIER", besteht.

Beliebige Zeichenketten können mit dem Operator "+" verknüpft werden, sowohl direkt angegebene als auch in Variablen enthaltene Zeichenketten.

Die unschöne Ausgabe des Namens im obigen Beispiel lässt sich korrigieren, wenn an den Vornamen zuerst ein Leerzeichen (" ") und anschließend der Nachname "angehängt" wird.

```
100 INPUT "VORNAME";VN$
110 INPUT "NACHNAME";NN$
120 NAME$=VN$+" "+NN$:REM NAME=VORNAME+" "+NACHNAME
130 PRINT NAME$

RUN
VORNAME? OSKAR
NACHNAME? MAIER
OSKAR MAIER
```

Maximale Länge von Stringvariablen

Eine unangenehme Eigenschaft von Stringvariablen ist die begrenzte Länge. Stringvariablen können zwar beliebige Zeichen aufnehmen, jedoch nicht beliebig lange Zeichenketten. Geben Sie (nach Eingabe von NEW) das folgende Programm ein und starten Sie es.

```
100 X$="12345678901234567890123456789012345678901"
110 Y$=Y$+X$:REM LAENGE 40 ZEICHEN
120 Y$=Y$+X$:REM LAENGE 80 ZEICHEN
130 Y$=Y$+X$:REM LAENGE 120 ZEICHEN
140 Y$=Y$+X$:REM LAENGE 160 ZEICHEN
150 Y$=Y$+X$:REM LAENGE 200 ZEICHEN
160 Y$=Y$+X$:REM LAENGE 240 ZEICHEN
170 Y$=Y$+X$:REM LAENGE 280 ZEICHEN (?)
```

Der Ablauf dieses Programms: In Zeile 100 wird X$ eine 40 Zeichen lange Zeichenkette zugewiesen. In der folgenden Zeile wird der Stringvariablen Y$ der alte Inhalt dieser Variablen plus dem Inhalt von X$ zugewiesen.

Da bisher keinerlei Zuweisung zur Variablen Y$ erfolgte, ist diese Variable "leer", sie enthält kein einziges Zeichen. Die Länge dieser Stringvariablen beträgt daher null Zeichen.

Diese leere Variable wird nun mit X$ verkettet, die 40 Zeichen dieser Stringvariablen werden angehängt. Die neue Länge von Y$ ergibt sich daher zu 40 Zeichen.

In den folgenden Zeilen wiederholt sich dieser Vorgang. Die in X$ enthaltene Zeichenkette wird an die momentan in Y$ enthaltene Zeichenkette angehängt.

Jede Verkettung erhöht die Länge von Y$ um weitere 40 Zeichen. Y$ besitzt daher zuerst die Länge null und in den folgenden Zeilen die Längen 40, 80, 120, 160, 200, 240 und 280 Zeichen.

Wenn Sie dieses Programm mit RUN starten, gibt der BASIC-Interpreter einen "STRING TOO LONG ERROR IN 170" auf dem Bildschirm aus. Er will Ihnen damit mitteilen, daß in der Programmzeile Nummer 170 die maximale Stringlänge überschritten wurde.

Die maximale Stringlänge muß daher irgendwo zwischen 240 und 280 zu finden sein. Um Ihnen ein langwieriges Experimentieren zu ersparen: Stringvariablen können sowohl beim C16 und C116 als auch beim Plus4 maximal 255 Zeichen enthalten.

Stringvariablen sind außerordentlich wichtig und werden in Programmen sehr häufig verwendet. Wie in der Praxis üblich, werde ich im folgenden die Begriffe "Stringvariable" und "String" verwenden. "Stringvariable" kennzeichnet eine Variable, die Zeichenketten enthalten kann, mit dem Ausdruck "String" ist immer eine Zeichenkette gemeint, die entweder direkt in einem Programm steht (100 PRINT "DIES IST EIN TEST") oder aber den Inhalt (!) einer Stringvariablen kennzeichnet (100 X$="DIES IST EIN TEST").

INPUT mit String- und numerischen Variablen

Der INPUT-Befehl wurde von uns sowohl zur Eingabe von Zahlen als auch beliebiger Zeichenketten verwendet. Die Einhaltung des erforderlichen Variablentyps (Stringvariable / numerische Variable) muß unbedingt beachtet werden, wie die folgenden Beispiele zeigen.

Beispiel 1:

```
100 INPUT X
RUN
? OSKAR
?REDO FROM START
?
```

Beispiel 2:

```
1ØØ INPUT X$
11Ø INPUT Y$
RUN
? OSKAR
? 1ØØ.34
```

Im ersten Beispiel führt die Eingabe von "OSKAR" zu der Fehlermeldung "?REDO FROM START" (= REDO FROM START = WIEDERHOLE DIE EINGABE). Erneut erscheint ein Fragezeichen, das uns zur Wiederholung der Eingabe auffordert. Der Grund: In Zeile 100, dem INPUT-Befehl, wurde die Variable X angegeben. Die Eingabe soll nach Beenden mit RETURN X zugewiesen werden. Da X jedoch eine numerische Variable ist, kann dieser Variablen nur eine Zahl, nicht jedoch eine Zeichenkette zugewiesen werden. Daher die Aufforderung zur Wiederholung der Eingabe.

Im zweiten Beispiel werden die Stringvariablen X$ und Y$ benutzt. Geben Sie bitte wie im Beispiel gezeigt auf die erste Eingabeaufforderung "OSKAR" und als zweite Eingabe "100.34" ein. Beide Eingaben werden wiederspruchslos akzeptiert und den beiden Stringvariablen zugewiesen.

Daraus folgt: Numerische Variablen können ausschließlich Zahlen zugewiesen werden, Stringvariablen dagegen beliebige Zeichenketten. Eine Zuweisung wie "100.34" an eine Stringvariable weist dieser die Zeichenkette (!) "100.34" zu, die nicht als Zahl interpretiert wird! Soll die Zahl (!) 100.34 zur weiteren Verarbeitung eingegeben werden, muß im INPUT-Befehl eine numerische Variable angegeben werden.

Außer dem Typ der erwarteten Eingabe ist bei Verwendung des INPUT-Befehls eine weitere Besonderheit zu berücksichtigen. INPUT gestattet theoretisch die Zuweisung einer beliebigen Zeichenkette an eine Stringvariable. Leider ist dieser Befehl nicht fehlerfrei, wie das folgende Beispiel zeigt.

```
100 INPUT X$
RUN
? MAIER,OSKAR
?EXTRA IGNORED
```

Die Eingabe der Zeichenkette "MAIER,OSKAR" führt zur Fehlermeldung "?EXTRA IGNORED" und zum Programmabbruch. INPUT akzeptiert bei der Eingabe von Zeichenketten leider nicht alle Zeichen, über die der C16, C116 oder Plus/4 verfügt. Die Eingabe eines Kommas, Doppelpunktes oder eines Anführungszeichens führt zu Fehlern.

Stringfunktionen

Wirklich komfortabel wird der Umgang mit Zeichenkette und Stringvariablen erst durch die verschiedenen "Stringfunktionen", die der C16, C116 bzw. Plus/4 zur Verfügung stellt.

Die wichtigsten Stringfunktionen will ich im folgenden kurz darstellen. Einige dieser Funktionen werden so häufig benötigt, daß sie uns das gesamte Buch hindurch "verfolgen" werden.

Die Stringfunktionen benötigen oftmals mehrere Angaben, und zwar sowohl die Angabe von Zahlen als auch von Zeichenketten. Wenn Zahlen anzugeben sind, können immer auch numerische Variablen oder numerische Ausdrücke verwendet werden. Ebenso kann die Angabe einer Zeichenkette jederzeit durch eine Stringvariable oder einen Ausdruck, der einen String ergibt, ersetzt werden. Alle Stringfunktionen liefern als Ergebnis wiederum eine Zeichenkette. Daher können Stringfunktionen selbst in anderen Stringfunktionen verwendet werden (analog den numerischen Ausdrücken und Funktionen).

LEFT$ und RIGHT$

Mit LEFT$ und RIGHT$ kann das rechte oder das linke Ende einer Zeichenkette isoliert werden. Das genaue Format der beiden Befehle lautet:

```
LEFT$(STRING,LÄNGE) und RIGHT$(STRING,LÄNGE)
```

Die Funktion LEFT$ schneidet den linken Teil einer Zeichenkette ab. "Länge" gibt an, wieviele Zeichen abzutrennen sind. Zum Beispiel ergibt LEFT$("MAIER",2) die Zeichenkette "MA", das heißt die beiden ersten Zeichen des angegebenen Strings.

Beispiele:

1.
```
PRINT LEFT$("C16 UND PLUS",3)
C16
```

2.
```
PRINT LEFT$("C16 UND PLUS4",7)
C16 UND
```

RIGHT$ liefert im Gegensatz zu LEFT$ den rechten Teil einer Zeichenkette. RIGHT$("MAIER",2) ergibt daher "ER", die beiden letzten Zeichen des angegebenen Strings.

Beispiele:

1.
```
PRINT RIGHT$("C16 UND PLUS4",5)
PLUS4
```

2.
```
PRINT RIGHT$("C16 UND PLUS4",8)
UND PLUS4
```

MID$

Die Funktion MID$ ist schwieriger zu handhaben als LEFT$ und RIGHT$, dafür jedoch auch vielseitiger. MID$ trennt einen beliebigen Teil aus einer Zeichenkette heraus. Sie müssen angeben, ab dem wievielten Zeichen dieses "Herausschneiden" beginnen soll und wie viele Zeichen abzutrennen sind.

```
MID$(STRING,POSITION,LÄNGE)
```

Beispiele:

1.
```
PRINT MID$("C16 UND PLUS4",5,3)
UND
```

2.
```
PRINT MID$("C16 UND PLUS4",3,10)
6 UND PLUS
```

Sie sehen, alle drei Stringfunktionen liefern als Ergebnis eine Zeichenkette, einen bestimmten Teil des angegebenen Strings. Die resultierende Zeichenkette kann - wie jede Zeichenkette - einer beliebigen Stringvariablen zugewiesen werden, wie das folgende Programm demonstriert.

```
100 X$=RIGHT$("C16 UND PLUS4",5):REM LETZTE 5 ZEICHEN
110 Y$=LEFT$("C16 UND PLUS4",3):REM ERSTE DREI ZEICHEN
120 Z$=MID$("C16 UND PLUS4",5,3):REM DREI ZEICHEN AB DEM FUENFTEN
ZEICHEN
130 PRINT X$
140 PRINT Y$
150 PRINT Z$

RUN
PLUS4
C16
UND
```

Im Umgang mit Strings sind zwei weitere Funktionen sehr nützlich, LEN und INSTR. INSTR wird im Kapitel über Dateiverwaltung ausführlich erläutert.

LEN(STRING) ist eine numerische Funktion, das heißt LEN liefert als Ergebnis eine Zahl, und zwar die Länge einer Zeichenette. So ergibt zum Beispiel PRINT LEN("MAIER") die Zahl 5 und X$="OTTO":PRINT LEN(X$) die Zahl 4. Wie jede Zahl kann selbstverständlich auch das Ergebnis von LEN einer numerischen Variablen zugewiesen werden.

Zusammenfassung

1. Variablennamen bestehen aus beliebigen "alphanumerischen" Zeichen (Buchstaben und Ziffern), wobei das erste Zeichen ein Buchstabe sein muß. Signifikant sind nur die beiden ersten Zeichens des Variablennamens (ZINS und ZINSEN kennzeichnen die gleiche Variable). In einem Variablennamen dürfen keine reservierten Wörter enthalten sein. Stringvariablen sind durch das Dollarzeichen hinter den Variablennamen gekennzeichnet (X$).

2. Numerische Variablen sind Statthalter für Zahlen, Stringvariablen Stellvertreter für Zeichenketten. Soll einer numerischen Variablen eine Zeichenkette zugewiesen werden, die nicht als Zahl aufgefasst werden kann, wird die Fehlermeldung "?EXTRA IGNORED" ausgegeben werden.

3. Der C16/Plus4 besitzt zwei verschiedene Arten numerischer Variablen, die meist verwendeten Fließkommavariablen und die Integervariablen. Während Fließkommavariablen praktisch jede Zahl verarbeiten können, besitzen Integervariablen einen sehr eingeschränkten Wertebereich. Dieser Variablentyp kann nur ganze Zahlen im Bereich -32768 bis +32767 verarbeiten. Integervariablen unterscheiden sich von Fließkommavariablen durch das Prozentzeichen hinter dem Variablennamen (X%).

4. Mit numerischen Variablen kann ebenso wie mit dem zugewiesenen Wert selbst gerechnet werden. Ergebnisse können wiederum numerischen Variablen zugewiesen werden (Y=2*X).

5. Jede neue Zuweisung an eine Variable überschreibt den bisher gültigen Variableninhalt.

6. Variablen können Werte entweder direkt im Programm zugewiesen werden, oder aber - mit Hilfe des Befehls INPUT - von "außen", durch den Benutzer. INPUT unterbricht den Programmablauf und wartet auf eine Eingabe des Benutzers. Eingaben dürfen nicht länger als 88 Zeichen sein. Nach Beenden der Eingabe mit RETURN wird diese der angegebenen Variablen zugewiesen und die Programmbearbeitung fortgesetzt.

7. INPUT kann auf verschiedene Weise angewendet werden:
   - einfache Zuweisung: INPUT (VARIABLE)
   - Zuweisung plus Ausgabe eines Kommentars: INPUT "(KOMMENTAR)";(VARIABLE)
   - Mehrfachzuweisungen: Mit einem INPUT-Befehl können mehrere Benutzereingaben einer entsprechenden Anzahl an Variablen zugewiesen werden. Sowohl die verschiedenen Eingaben als auch die aufgeführten Variablen müssen mit Kommas voneinander getrennt werden (INPUT (VARIABLE1), (VARIABLE2),... oder INPUT "(KOMMENTAR)";(VARIABLE1), (VARIABLE2),...).

8. Bei Eingabe der Zeichen ":", "," und des Anführungszeichens arbeitet INPUT fehlerhaft. Die Eingabe dieser Zeichen sollte daher unbedingt vermieden werden.

3.1.6 Steuerung des Programmablaufs

Alle vorgestellten Programm besaßen eine gemeinsame Eigenschaft: Es handelte sich um sogenannte "sequentielle" Programme. "Sequentielles" Programm bedeutet, daß ein Programmbefehl nach dem anderen bearbeitet und ausgeführt wird, zuerst alle Befehle einer Programmzeile, danach die Befehle der nächsten Programmzeile und so weiter.

Zum Üben können Sie eine Vielzahl solcher sequentiell ablaufender Programme erstellen. Sie werden jedoch oftmals feststellen, daß bestimmte Probleme andere Arten des Programmablaufs erfordern.

Das folgende Kapitel zeigt Ihnen, welche Möglichkeiten zur Beeinflussung des Programmablaufs vorhanden sind. Die vorgestellten Befehle (IF...THEN...:ELSE, GOTO, FOR...TO...NEXT usw.) werden in jedem größeren BASIC-Programm ständig benötigt. Sie stellen daher eine Grundvoraussetzung zur Behandlung der Musik-, Graphik- und Dateibefehle des C16, C116 und Plus/4 dar, die anhand umfangreicherer Programmbeispiele erläutert werden.

Da diese Befehle derart wichtig sind, bitte ich Sie - auch wenn Ihnen das Folgende ein wenig trocken und theoretisch erscheinen mag - dieses Kapitel keinesfalls zu überblättern, sondern alle (!) Demoprogramme sorgfältig durchzuarbeiten und eigene Programmbeispiele mit den besprochenen "Steuerbefehlen" zu entwickeln.

3.1.6.1 Die Verzweigung (IF)

Mit einem sequentiellen Program ist es nicht möglich, das folgende Problem zu lösen: Sie schreiben ein Programm zur Berechnung der Mehrwertsteuer für Zeitschriften und KFZ-Teile. Der Mehrwertsteuersatz ist jedoch von Produkt zu Produkt unterschiedlich. Auf Zeitschriften wird ein Mehrwertsteuersatz von 7%, auf KFZ-Zubehörteile jedoch ein Mehrwertsteuersatz von 14% angewandt.

Optimal wäre folgende Lösung: Der Benutzer gibt den Preis des Produktes ein und anschließend die Produktart. Handelt es sich um eine Zeitschrift, berechnet das Programm 7%, ansonsten 14% Mehrwertsteuer.

Das Problem wird mit Hilfe des IF-Befehls gelöst, der sogenannten "Verzweigung". Wenn eine angegebene Bedingung erfüllt ist, führt der IF-Befehl einen Programmbefehl aus, ist die Bedingung nicht erfüllt, wird ein anderer Programmbefehl ausgeführt.

```
IF (BEDINGUNG) THEN (BEFEHL1):ELSE (BEFEHL2)
```

Anstatt Ihnen langwierige theoretische Erläuterungen zu diesem Befehl zu geben, stelle ich ein Programm vor, das seine Wirkungsweise anschaulich demonstriert. Löschen Sie den Speicher mit NEW, bevor Sie dieses Programm eingeben und starten.

```
100 INPUT "ZAHL EINGEBEN";X
110 IF X=10 THEN PRINT "X GLEICH 10":ELSE PRINT "X UNGLEICH 10"
```

Starten Sie das Programm bitte mehrmals. Geben Sie jeweils unterschiedliche Zahlen ein. Immer dann, wenn Sie die Zahl zehn eingeben, wird "X GLEICH 10", ansonsten "X UNGLEICH 10" auf dem Bildschirm ausgegeben.

```
RUN
? 3.4
X UNGLEICH 10

READY.

RUN
? 10
X GLEICH 10

READY.
```

```
RUN
? 11
X UNGLEICH 10

READY.
```

Der Programmablauf: Die eingegebene Zahl wird der numerischen Variablen X zugewiesen. Die im IF-Befehl angegebene Bedingung lautet X=10. Wenn diese Bedingung erfüllt ist, das heißt, wenn X den Wert zehn besitzt, wird der dem Befehl THEN folgende PRINT-Befehl ausgeführt. Besitzt X jedoch einen Wert ungleich zehn, wird stattdessen der dem Befehl ELSE folgende PRINT-Befehl ausgeführt.

Diese Funktionsweise des IF-Befehls wird noch deutlicher, wenn der wie üblich englische Befehl ins Deutsche übersetzt wird.

> WENN X=10 IST, DANN SCHREIBE "X GLEICH 10", ANSONSTEN SCHREIBE "X UNGLEICH 10".

Vergleichsoperatoren

Die angegebene Bedingung kann ein beliebiger numerischer oder alpahanumerischer Ausdruck sein, in dem sowohl Konstanten als auch Variablen gemischt verwendet werden können. Außer dem Operator "=" können folgende "Vergleichsoperatoren" verwendet werden.

">"	(größer)
"<"	(kleiner)
">="	(größer oder gleich)
"<="	(kleiner oder gleich)
"<>"	(ungleich)

Um den IF-Befehl sicher in den Griff zu bekommen, benötigen Sie Beispielprogramme. Geben Sie die folgenden Demoprogramme bitte ein und starten Sie sie. In allen Programmen fordert Sie der INPUT-Befehl zur Eingabe auf. Verwenden Sie

die in den Beispielen benutzten Eingaben und versuchen Sie, den Programmablauf zu ergründen, bevor Sie weiterlesen.

Beispiel 1:

```
100 INPUT "ZAHL";X
110 IF X>10 THEN PRINT "X GROESSER 10":ELSE PRINT "X
KLEINER/GLEICH 10"

RUN
ZAHL? 9
X KLEINER/GLEICH 10

READY.

RUN
ZAHL? 11
X GROESSER 10

READY.

RUN
ZAHL? 10
X KLEINER/GLEICH 10

READY.
```

Beispiel 2:

```
100 INPUT "ZAHL";X
110 IF 2*X<100 THEN PRINT "BEFEHL1":ELSE PRINT "BEFEHL2"

RUN
ZAHL? 20
BEFEHL1

READY.
```

```
RUN
ZAHL? 5Ø
BEFEHL2

READY.

RUN
ZAHL?
53.34
BEFEHL2

READY.
```

Beispiel 3:

```
1ØØ INPUT "ZEICHENKETTE";X$
11Ø IF X$="MAIER" THEN PRINT "BEFEHL1":ELSE PRINT "BEFEHL2"

RUN
ZEICHENKETTE? MAIER
BEFEHL1

READY.

RUN
ZEICHENKETTE? MAYER
BEFEHL2

READY.

RUN
ZEICHENKETTE? MUELLER
BEFEHL2

READY.
```

Beispiel 4:

```
100 INPUT "ZEICHENKETTE";X$
110 IF X$>"MAIER" THEN PRINT "GROESSER MAIER":ELSE PRINT
"KLEINER/GLEICH MAIER"

RUN
ANTON
KLEINER/GLEICH MAIER

READY.

RUN
WILLI
GROESSER MAIER

READY.

RUN
MEIER
GROESSER MAIER

READY.
```

Die beiden ersten Beispiele sind problemlos zu verstehen. Zwei Zahlen werden miteinander verglichen. Die vom Benutzer eingegebene Zahl X oder auch 2*X wird mit einer fest im Programm angegebenen Konstanten (zehn beziehungsweise 100) verglichen. In Beispiel 1 lautet die Bedingung umgangssprachlich "IST X GROESSER ALS 10?", in Beispiel 2 "IST 2*X KLEINER ALS 100?". Ist die jeweilige Bedingung erfüllt, wird der dem THEN folgende Befehl ausgeführt, wenn sie nicht erfüllt ist, wird der dem ELSE folgende Befehl ausgeführt. Man spricht auch vom "THEN-Zweig" beziehungsweise "ELSE"-Zweig", wobei - abhängig von der Bedingung - jeweils nur einer der beiden Zweige bearbeitet wird.

Das dritte Beispiel ist ebenfalls noch verständlich. Der Inhalt der eingegebenen Zeichenkette, die sich in der Stringvariablen X$ befindet, wird mit der Zeichenkette "MAIER" verglichen. Sind

beide Zeichenketten identisch (wurde also "MAIER" eingegeben), wird der THEN-Zweig ausgeführt, ansonsten der IF-Zweig. Die Bedingung 'X$="MAIER"' ist nicht erfüllt, wenn sich die Zeichenketten in einem oder mehreren Zeichen unterscheiden.

Das vierte Beispiel stellt Sie vermutlich vor Probleme. Wie kann verglichen werden, ob eine Zeichenkette größer ist als eine andere Zeichenkette? Nun, sehr einfach: Die Operatoren größer und kleiner beziehen sich bei der Anwendung auf Zeichenketten auf das Alphabet. Der Vergleich findet alphabetisch statt.

Da zum Beispiel "WILLI" alphabetisch gesehen zweifellos größer ist als "MAIER", ist die Bedingung erfüllt. Die erste Zeichenkette ist größer als die zweite und der THEN-Zweig wird ausgeführt.

Das Mehrwertsteuerproblem kann mit diesem Befehl höchst elegant gelöst werden.

```
100 INPUT "BETRAG";BE
110 INPUT "ZEITSCHRIFTEN ('J', WENN JA)";ZE$
120 IF ZE$="J" THEN MW=BE/100*7:ELSE MW=BE/100*14
130 PRINT MW

RUN
BETRAG? 300
ZEITSCHRIFTEN? ('J', WENN JA)? J
 21

READY.

RUN
BETRAG? 300
ZEITSCHRIFTEN? ('J', WENN JA)? N
 42

READY.
```

Der Programmablauf: Der INPUT-Befehl in Zeile 100 fordert den Benutzer zur Eingabe eines beliebigen Betrags auf, der der numerischen Variablen BE zugewiesen wird. In Zeile 20 folgt ein INPUT-Befehl mit Kommentar. Der Benutzer wird aufgefordert, das Zeichen "J" (für "JA") einzugeben, wenn es sich um eine Zeitschrift handelt. Wenn es sich nicht um eine Zeitschrift handelt, geben Sie bitte ein beliebiges anderes Zeichen ein, zum Beispiel "N" als Abkürzung für "NEIN".

In Zeile 120 erfolgt die Berechnung der Mehrwertsteuer, wobei entweder 7% oder 14% vom eingegebenen Betrag BE berechnet werden. Wenn die Bedingung 'ZE$="J"' erfüllt ist, das heißt, wenn der Benutzer auf die Frage "ZEITSCHRIFTEN? ('J', WENN JA)?" mit "J" antwortet, wird der THEN-Zweig ausgeführt und vom eingegebenen Betrag BE 7% Mehrwertsteuer berechnet.

Ist die Bedingung nicht erfüllt, wurde also ein beliebiges (oder mehrere) anderes Zeichen eingegeben, werden 14% berechnet.

Das Ergebnis der ausgeführten Berechnung wird der Variablen MW (= "Mehrwertsteuer") zugewiesen. In Zeile 130 wird der Inhalt dieser Variablen ausgegeben.

Ein wichtiger Hinweis: Wie besprochen wird der THEN-Zweig bearbeitet, wenn die angegebene Bedingung erfüllt, ansonsten der ELSE-Zweig. Jeder dieser Zweige darf beliebig viele durch Kommas voneinander getrennte Befehle enthalten, wie das folgende Beispiel zeigt.

```
100 INPUT "ZAHL";Z$
110 IF Z=1 THEN PRINT "DIE EINGEGEBENE ZAHL":PRINT "HAT DEN
WERT 1"

RUN
ZAHL? 1
DIE EINGEGEBENE ZAHL
HAT DEN WERT 1

READY.
```

Logische Operatoren

Außer Vergleichs- können auch "logische" Operatoren im IF-Befehl verwendet werden, die so genannt werden, weil sie den Gesetzen der formalen Logik entsprechen. Logische Operatoren wie "UND", "ODER" und "NICHT" sind Ihnen sicher bekannt, wie folgende Beispiele zeigen.

1. "Wenn es nicht heiß ist, dann schwitze ich NICHT".
2. "Wenn es heiß UND die Luftfeuchtigkeit hoch ist, dann schwitze ich".
3. "Wenn es kalt ODER aber die Heizung defekt ist, dann friere ich".

Bedingungen, die mit logischen Operatoren arbeiten, können beliebig "verknüpft" sein.

1. "Wenn es heiß ist UND die Luftfeuchtigkeit hoch ist UND der Ventilator defekt ist, dann schwitze ich".
2. "Wenn es heiß UND die Luftfeuchtigkeit hoch ist, ODER ich in einer Sauna bin, schwitze ich.

BASIC erlaubt wie jede natürliche Sprache die Verwendung einfacher oder mehrerer verknüpfter logischer Operatoren. Logische Operatoren werden im Verzweigungsbefehl IF angewendet, wenn die Ausführungen bestimmter Befehle von mehreren (!) Bedingungen abhängt.

Ein praktisches Beispiel: Der Mehrwertsteuersatz von 7% wird außer auf Zeitschriften auch auf Bücher angewendet. Um diese Tatsache zu berücksichtigen, muß das Mehrwertsteuerprogramm entsprechend geändert werden. Die Berechnung erfolgt mit dem Wert 7%, wenn es sich bei dem Produkt um eine Zeitschrift oder um ein Buch handelt, ansonsten werden 14% verwendet. Die Ausführung der dem IF folgenden Befehle ist von zwei mit ODER verknüften Bedingungen abhängig.

```
100 INPUT "BETRAG";BE
110 INPUT "ZEITSCHRIFTEN ('J', WENN JA)";ZE$
120 INPUT "BUCH ('J', WENN JA)";BU$
130 IF ZE$="J" OR BU$="J" THEN MW=BE/100*7:ELSE MW=BE/100*14
140 PRINT MW
```

Das geänderte Progrmm fragt den Benutzer zuerst, ob es sich um eine Zeitschrift handelt. Die Antwort wird ZE$ zugewiesen. Anschließend wird die Antwort auf die Frage "BUCH ('J', WENN JA)" der Variablen BU$ zugewiesen.

Der folgende IF-Befehl enthält zwei Bedingungen: 'IF ZE$="J" OR BU$="J"...'. Die beiden Bedingungen sind mit OR verknüpft, der BASIC-Entsprechung des Wortes ODER. Wenn eine der beiden Bedingungen erfüllt ist, wird der THEN-Zweig ausgeführt. Ist keine von beiden Bedingungen erfüllt, wird der ELSE-Zweig ausgeführt.

Soll der THEN-Zweig nur dann ausgeführt werden, wenn zwei Bedingungen zugleich (!) erfüllt sind, kommt das BASIC-Wort AND zum Einsatz, das Äquivalent zu UND.

```
100 INPUT "ZAHL1";Z1
110 INPUT "ZAHL2";Z2
120 IF Z1<10 AND Z2<10 THEN PRINT "KLEINER 10":ELSE PRINT
"NICHT KLEINER 10"

RUN
ZAHL1? 16
ZAHL2? 5
NICHT KLEINER 10

READY.

RUN
ZAHL1? 9
ZAHL2? 7
KLEINER 10

READY.
```

Dieses Beispiel demonstriert die Anwendung von AND. Der THEN-Zweig wird nur dann ausgeführt, wenn beide (!) Bedingungen erfüllt sind, das heißt wenn sowohl ZAHL1 als auch ZAHL2 kleiner als zehn ist. Ist dies nicht der Fall, wird der ELSE-Zweig ausgeführt.

Die dem IF-Befehl folgende Bedingung kann aus beliebig vielen "Teilbedingungen" bestehen, die mit logischen Operatoren verknüpft sind. In jeder Teilbedingung können beliebige Vergleichsoperatoren verwendet werden. Alle Operatoren können in Verbindung mit numerischen und Stringausdrücken angewandt werden, wodurch außerordentlich komplexe Ausdrücke entstehen können.

```
100 INPUT "ZAHL1";Z1
110 INPUT "ZAHL2";Z2
120 INPUT "ZEICHENKETTE";Z$
130 IF Z1<10 AND Z2<10 OR Z$="TEST" THEN PRINT
"BEFEHL1":ELSE PRINT "BEFEHL2"

RUN
ZAHL1? 9
ZAHL2? 8
ZEICHENKETTE? XYZ
BEFEHL1

READY.

RUN
ZAHL1? 9
ZAHL2? 12
ZEICHENKETTE? XYZ
BEFEHL2

READY.
```

```
RUN
ZAHL1? 9
ZAHL2? 12
ZEICHENKETTE? TEST
BEFEHL2

READY.
```

Die Bedingung kann wie folgt "übersetzt" werden: 'Wenn Z1 und Z2 beide kleiner als zehn sind, oder aber Z$ die Zeichenkette "TEST" enthält, schreibe "BEFEHL1", ansonsten "BEFEHL2"'.

Die gesamte Bedingung ist erfüllt, wenn

1. die beiden ersten Teilbedingungen gleichzeitig erfüllt sind, unabhängig davon, ob die dritte Teilbedingung erfüllt oder nicht erfüllt ist.

2. die zweite Teilbedingung erfüllt ist, unabhängig davon, ob eine der beiden ersten Bedingungen (oder gar beide) ebenfalls erfüllt sind.

IF-Befehle mit einer Vielzahl von Unterbedingungen sind leider teilweise schwer zu verstehen. Das Verständnis wird durch die Verwendung von Klammern erleichtert. Zeile 130 könnte unter Verwendung von Klammern wie folgt formuliert werden.

```
130 IF (Z1<10 AND Z2<10) OR Z$="TEST" THEN PRINT
"BEFEHL1":ELSE PRINT "BEFEHL2"
```

Wie man sieht, können die beiden ersten Bedingungen zu einer (!) Unterbedingung zusammengefasst werden. Die Gesamtbedingung ist nun leichter zu verstehen. Wenn die beiden Bedingungen Z1<10 und Z2<10 erfüllt sind, ist die erste Unterbedingung (Z1<10 AND Z2<10) erfüllt. Wenn diese oder die zweite Unterbedingung IF Z$="TEST" erfüllt ist, ist die Gesamtbedingung (Z1<10 AND Z2<10) OR Z$="TEST" erfüllt.

Wie bei allen Befehlen bin ich auch beim IF-Befehl der Ansicht, daß ein volles Verständnis nur durch die Anwendung möglich ist und rate Ihnen daher dringend, eigene Beispielprogramme zu schreiben, in denen Sie alle genannten Möglichkeiten zur Verwendung dieses Befehls ausnutzen.

Eine Ergänzung zum Abschluß: Der IF-Befehl existiert in zwei unterschiedlichen Versionen. Der "vollständigen" Version (IF (BEDINGUNGEN) THEN (BEFEHL1):ELSE (BEFEHL2)) und einer einfacheren Version (IF (BEDINGUNGEN) THEN (BEFEHL)).

In dieser einfacheren Version existiert kein ELSE-Zweig, so daß sich folgende Möglichkeiten ergeben.

1. Die Bedingung(en) ist(sind) erfüllt. In diesem Fall wird der THEN-Zweig ausgeführt.

2. Die Bedingung(en) ist(sind) nicht erfüllt und der THEN-Zweig wird nicht ausgeführt. Da kein ELSE-Zweig vorhanden ist, wird in diesem zweiten Fall kein (!) Befehl ausgeführt.

```
100 INPUT "ZAHL1";Z1
110 INPUT "ZAHL2";Z2
120 INPUT "ADDIEREN ('J', WENN JA)";AD$
130 IF AD$="J" THEN PRINT Z1+Z2
```

In diesem Beispiel wird die Summe von Z1 und Z2 nur dann berechnet und ausgegeben, wenn die Frage 'ADDIEREN ("J", WENN JA)' mit "J" beantwortet wurde.

```
RUN
ZAHL1? 5
ZAHL2? 10
ADDIEREN? J
15

READY.
```

```
RUN
ZAHL1? 5
ZAHL2? 10
ADDIEREN? N

READY.
```

IF-Befehle können "verschachtelt" werden. Der THEN-Zweig besteht bei der Schachtelung wiederum aus einem IF-Befehl, der nur dann ausgeführt wird, wenn die Bedingung im vorhergehenden IF-Befehl erfüllt ist.

```
100 INPUT "ZAHL1";Z1
110 INPUT "ZAHL2";Z2
120 IF Z1<10 THEN IF Z2<10 THEN PRINT "KLEINER 10":ELSE PRINT "NICHT
K.10"
```

Die Verschachtelung mehrerer IF-Konstruktionen kann durch die Verknüpfung der jeweiligen Bedingungen mit AND ersetzt werden (IF Z1<10 AND Z2<10 THEN PRINT "KLEINER 10":ELSE PRINT "NICHT K.10"). Ich empfehle Ihnen, diese Lösung anzuwenden, da die entstehenden Ausdrücke leichter zu verstehen sind als geschachtelte IF-Befehle.

Zusammenfassung

1. Mit dem Befehl IF (BEDINGUNG) THEN (BEFEHL) lassen sich "Verzweigungen" realisieren. Der THEN-Zweig (BEFEHL1:BEFEHL2:...) wird nur dann ausgeführt, wenn die Bedingung erfüllt ist. Bei der zweiten Version IF (BEDINGUNG) THEN (BEFEHL1:BEFEHL2:...):ELSE (BEFEHL1:BEFEHL2:...) wird immer (!) ein Befehl ausgeführt. Ist die Bedingung erfüllt, werden die dem THEN-Zweig folgenden Befehle ausgeführt, ist sie nicht erfüllt, die dem ELSE-Zweig folgenden.

2. Zur Formulierung der Bedingung können Vergleichs- ("=", "<", "<", "<=", ">=", "<>") und logische Operatoren ("AND", "OR", "NOT") verwendet werden. Bedingungen sind beliebige numerische oder Stringausdrücke, die Konstanten und Variablen enthalten dürfen.

3. Die Bedingung kann aus beliebig vielen Teilbedingungen zusammengesetzt werden, die mit je einem logischen Operator verknüpft sind.

4. IF-Befehle können beliebig ineinander verschachtelt werden (IF BEDINGUNG1 THEN IF BEDINGUNG1 THEN IF BEDINGUNG3 THEN ...). Die Verschachtelung kann durch die Verknüpfung der Bedingungen mit AND ersetzt werden.

3.1.6.2 Der Sprungbefehl GOTO

Es dürften nur wenige BASIC-Programme existieren, die ohne den Befehl GOTO auskommen. Dieser Befehl ist schuld daran, daß BASIC einen teilweise sehr schlechten Ruf als Programmiersprache besitzt. Der Grund ist, daß sich mit GOTO äußerst unübersichtliche Programme schreiben lassen, die der Programmierer nach kurzer Zeit selbst nicht mehr versteht, sogenannte "Spaghetti-Programme". In unserem BASIC-Kurs werden Sie jedoch lernen, wie sich übersichtliche und "strukturierte" Programme trotz der Verwendung von GOTO erstellen lassen.

Stellen Sie sich das folgende Problem vor: Sie wollen die Zahlen eins bis 100 auf dem Bildschirm mit PRINT-Befehlen ausgeben. Mit dem bisher Gelernten bietet sich folgende Möglichkeit an:

```
100 PRINT "1"
110 PRINT "2"
120 PRINT "3"
...
...
...
970 PRINT "98"
980 PRINT "99"
990 PRINT "100"
```

Wahrscheinlich werden Sie nicht allzu begeistert sein, zur Lösung eines derart kleinen Problems ein Programm mit 100 (!) Programmzeilen schreiben zu müssen.

Sie stimmen mir sicher zu, wenn ich behaupte, daß die nachstehende Programmversion die gleiche Funktion besitzt und ebenfalls die Zahlen eins bis 100 ausgibt.

```
100 X=1:PRINT X:REM X=1
110 X=X+1:PRINT X:REM X=2
120 X=X+1:PRINT X:REM X=3
...
...
...
970 X=X+1:PRINT X:REM X=98
980 X=X+1:PRINT X:REM X=99
990 X=X+1:PRINT X:REM X=100
```

Auch dieses Programm besteht aus 100 einzelnen Zeilen. In jeder Zeile wird der Wert der Variablen X um eins erhöht und der jeweilige Wert ausgegeben. Dieses Programm ist noch umständlicher als die erste Version. Es deutet jedoch die Richtung an, die zu einer eleganten Lösung des Problems führt.

Es sollte möglich sein, eine Programmzeile, die jeweils X erhöht und den neuen Wert dieser Variablen ausgibt, immer wieder (!) ausführen zu lassen. Wir benötigen einen Befehl, der den BASIC-Interpreter anweist, nicht die nächste Zeile zu bearbeiten, sondern den Programmablauf mit der gerade bearbeiteten Zeile fortzusetzen.

Dieser benötigte Befehl ist der "Sprungbefehl" GOTO. Dem Wort GOTO folgt immer eine Zeilennummer, zum Beispiel GOTO 130. Der BASIC-Interpreter setzt die Programmbearbeitung mit der angegebenen Programmzeile fort, es erfolgt ein "Sprung" zu dieser Zeile. Geben Sie bitte das folgende Programm ein und starten Sie es.

```
100 X=X+1:PRINT X
110 GOTO 100

RUN
1
2
3
4
...
...
...
```

Nach dem Starten mit RUN wird dieses Programm die Zahlenreihe 1, 2, 3, 4, ... ausgeben. Diese Ausgabe wird niemals beendet, außer Sie unterbrechen das Programm, indem Sie die Taste STOP drücken.

Das Programm befindet sich in einer "Endlosschleife". In Zeile 100 wird der jeweilige Wert von X um eins erhöht und der sich ergebende Wert ausgegeben. In Zeile 110 erfolgt ein Sprung zu Zeile 100 (GOTO 100), mit der die Programmbearbeitung fortgesetzt wird. Der neue X-Wert wird um eins erhöht und ausgegeben. Anschließend erfolgt wieder ein Sprung zu Zeile 100, die nächste Erhöhung und Ausgabe findet statt, und so weiter.

Endlosschleifen sind fast immer sinnlos. In der Praxis soll eine (oder mehrere) Programmzeile mehrmals wiederholt werden, anschließend soll das Programm mit der Bearbeitung des restlichen Programms beginnen. Eine Programmschleife ist daher meist mit einer "Abbruchbedingung" verbunden, die angibt, wann die Schleife verlassen werden soll.

Die Abbruchbedingung lautet in unserem Fall: "Kehre nur dann zu Zeile 100 zurück, wenn die Zahl 100 noch nicht ausgegeben wurde". Der GOTO-Befehl soll laut dieser Bedingung nur ausgeführt werden, wenn X noch nicht gleich 100 ist.

```
100 X=X+1:PRINT X
110 IF X<>100 THEN GOTO 100
120 PRINT "ALLE ZAHLEN ZWISCHEN 1 U.100 WURDEN AUSGEGEBEN"
```

Dieses Programm gibt alle Zahlen von eins bis 100 aus und ist anschließend beendet.

```
RUN
1
2
3
...
...
98
99
100
ALLE ZAHLEN ZWISCHEN 1 U.100 WURDEN AUSGEGEBEN
```

Mit Hilfe des IF-Befehls wurde aus dem "unbedingten" Sprung mit GOTO ein "bedingter" Sprung, der nur dann ausgeführt wird, wenn die Bedingung IF X<>100 erfüllt ist. Wie wir wissen, wird der THEN-Zweig - das heißt der Sprungbefehl - nicht ausgeführt, wenn die angegebene Bedingung nicht erfüllt ist.

Sie sehen, durch geschickte Kombinationen der Befehle IF und GOTO zusammen mit einer geeigneten Bedingung lassen sich kontrollierte "Programmschleifen" erstellen, die nicht endlos

wiederholt, sondern nach einer bestimmten Anzahl von Wiederholungen abgebrochen werden.

Unser früheres Zinsprogramm besaß einen erheblichen Nachteil: Es war nicht möglich, Zinsen für einen beliebigen Zeitraum zu berechnen, sondern immer nur für ein Jahr. Um die Berechnung auf beliebige Zeiträume auszudehnen, benötigen wir eine Programmschleife. Sollen zum Beispiel die sich in fünf Jahren ergebenden Zinsen ermittelt werden, muß die Berechnung fünfmal wiederholt werden.

```
100 INPUT "ZINSSATZ";ZINS
110 INPUT "KAPITALEINSATZ";KAPITAL
120 INPUT "LAUFZEIT (IN JAHREN);ZEIT
130 PRINT
140 :
150 X=1:REM STARTWERT FUER SCHLEIFENZAEHLER
160 ZSUMME=ZSUMME+KAPITAL/100*ZINS;"DM"
170 X=X+1
180 IF X<=ZEIT THEN GOTO 160
190 :
200 PRINT "GESAMTKAPITAL:";KAPITAL+ZSUMME
```

Sollten Sie sich an den Zeilen 140 und 190 stören: Zeilen, die nur aus der Zeilennummer und einem nachfolgenden Doppelpunkt bestehen, interessieren den BASIC-Interpreter ebensowenig wie der Kommentarbefehl REM. Diese Zeilen können daher hervorragend zur optischen Gliederung eines Programms eingesetzt werden. Im Demoprogramm wird auf diese Weise der Programmteil zur Zinsberechnung hervorgehoben.

Sie sollten diese Technik in umfangreicheren Programmen möglichst oft zur optischen Trennung der einzelnen Programmteile einsetzen. Optisch gegliederte Programme lassen sich erheblich leichter verstehen als Programme, in denen in sich abgeschlossene Programmteile unmittelbar aufeinander folgen.

Doch nun zum Programmablauf: In den Zeilen 10-30 wird der Programmablauf jeweils unterbrochen, bis vom Benutzer die Eingaben des Zinssatzes (ZINS), des Kapitaleinsatzes

(KAPITAL) und nun auch der Laufzeit (ZEIT) vorgenommen werden.

Die Zinsberechnung erfolgt in den Zeilen 150-180. Die Schleife, in der die Zinsen pro Jahr aufsummiert werden, soll nach einer bestimmten Anzahl an "Schleifendurchgängen" verlassen werden, abhängig von der angegebenen Laufzeit. Wir benötigen daher einen "Zähler", der uns angibt, wie oft die Berechnung bereits durchgeführt wurde.

Als Zähler wird die Variable X verwendet, der in Zeile 150 der "Startwert" 1 zugewiesen wird. In der folgenden Zeile 160 werden die Zinsen für ein Jahr berechnet und zur bisherigen Zinssumme (ZSUMME) addiert.

In Zeile 170 wird der Wert von X, unserem "Schleifenzähler", um eins erhöht, da die Schleife einmal bearbeitet wurde. In Zeile 180 wird der momentane Wert von X mit der Laufzeit ZEIT verglichen. Wenn X kleiner oder gleich ("<=") der Laufzeit ist, muß die Zinsberechnung für mindestens ein weiteres Jahr stattfinden. Es erfolgt ein Sprung zu Zeile 160, die Zinsen für ein weiteres Jahr werden berechnet und zur bisherigen Zinssumme addiert.

Die Schleife wird verlassen, wenn die Bedingung IF X<=ZEIT nicht erfüllt ist, das heißt, wenn X entweder ebenso groß oder größer (dieser Fall wird nie eintreten!) als ZEIT ist, die Zinsen somit bereits für die gesamte Laufzeit des betreffenden Wertpapiers berechnet wurden.

Zum Abschluß des Programms wird das Gesamtkapital ausgegeben, das sich aus dem Anfangskapital und den errechneten Gesamtzinsen ergibt.

Merken Sie sich bitte das grundlegende Prinzip zum Aufbau von Schleifen mit GOTO.

1. Vor Beginn der Schleife erhält der Schleifenzähler, eine numerische Variable, den Startwert 1.

2. Es folgen beliebig viele Befehle innerhalb der Schleife, die bearbeitet werden sollen.

3. Am "Schleifenende" wird der Schleifenzähler um eins erhöht.

4. Der aktuelle Zählerwert wird mit der gewünschten Anzahl an Schleifendurchgängen verglichen. Ist der Zählerwert kleiner oder gleich der gewünschten Anzahl an Durchgängen, erfolgt ein Sprung zum Schleifenanfang (IF ZAEHLER<=ANZAHL THEN GOTO ...). Ist diese Bedingung nicht erfüllt, wird der GOTO-Befehl nicht ausgeführt. Der BASIC-Interpreter setzt die Programmbearbeitung sequentiell mit der nächsten Programmzeile fort, die Schleife ist beendet.

Einen kleinen Haken besitzt das Zinsprogramm noch: In der Realität werden nicht nur Zinsen, sondern Zinzeszinsen berechnet. Das heißt, nach einem Jahr erhöht sich das Anfangskapital um den jeweiligen Zinsbetrag und im folgenden Jahr wird der Zinsbetrag ausgehend vom neuen Kapital berechnet. Die benötigte Änderung zur Zinzeszinsberechnung sollten Sie nun selbst vornehmen können. Versuchen Sie es und vergleichen Sie Ihre Lösung anschließend mit dem nachfolgenden Programm.

```
100 INPUT "ZINSSATZ";ZINS
110 INPUT "KAPITALEINSATZ";KAPITAL
120 INPUT "LAUFZEIT (IN JAHREN);ZEIT
130 PRINT
140 :
150 X=1:REM STARTWERT FUER SCHLEIFENZAEHLER
160 KAPITAL=KAPITAL+KAPITAL/100*ZINS;"DM"
170 X=X+1
180 IF X<=ZEIT THEN GOTO 160
190 :
200 PRINT "GESAMTKAPITAL:";KAPITAL
```

Ball bewegen

Mit IF und GOTO sind wir nun in der Lage, das zu Beginn des Kurses vorgestellte Demoprogramm zu erstellen, das einen Ball über den Bildschirm bewegt. Als Ball verwenden wir das Graphikzeichen mit dem Code 113 (Ausgabe mit PRINT CHR$(113)). Geben Sie bitte das folgende Programm ein.

```
100 X=1:REM STARTWERT FUER SCHLEIFENZAEHLER
110 :
120 PRINT CHR$(113);
130 X=X+1
140 IF X<=40 THEN GOTO 120
```

Zuerst wird in Zeile 100 dem Schleifenzähler X der Startwert 1 zugewiesen.

Die eigentliche Programmschleife umfasst die Zeilen 120-140. Das Graphikzeichen mit dem Code 113 wird ausgegeben, das einen Ball symbolisiert. Den Abschluß dieses PRINT-Befehls muß (!) ein Semikolon bilden. Das Semikolon gewährleistet, daß die nächste Ausgabe mit PRINT unmittelbar neben der vorigen Ausgabe stattfindet.

Der Schleifenzähler X wird nun erhöht und mit dem Wert 40 verglichen. Mit 40, da die Schleife genau 40-mal bearbeitet werden soll. Der Ball soll 40mal nebeneinander ausgegeben werden, entsprechend der Spaltenanzahl des Bildschirms.

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

Da X nach dem ersten Schleifendurchgang den Wert 2 besitzt, ist die Bedingung IF X<=40 erfüllt und es erfolgt ein Sprung zum Schleifenanfang (THEN GOTO 120). Der Ball wird erneut ausgegeben, und zwar in der gleichen Bildschirmzeile, jedoch der nächsten Spalte dieser Zeile (dank dem Formatierungszeichen ";").

Wenn die Schleife 40-mal durchlaufen und der Ball somit nacheinander in jeder Spalte ausgegeben wurde, enthält X den Wert 41. Die Bedingung IF X<=40 ist nicht mehr erfüllt und die Schleife wird verlassen (der GOTO-Befehl wird nicht ausgeführt).

Dieses Programm ist noch nicht ganz perfekt. Der Ball wird zwar 40-mal nebeneinander ausgegeben, der jeweils zuvor gemalte Ball sollte jedoch gelöscht werden.

Nun kommen die Steuerzeichen zur Bewegung des Cursors im (!) Programm zum Einsatz. Nach Ausführung des Befehls PRINT CHR$(113); befindet sich der Cursor (für uns unsichtbar) unmittelbar hinter dem ausgegebenen Graphikzeichen, an jener Position, an der der nächste PRINT-Befehl einen weiteren Ball ausgibt. Bevor ein weiterer PRINT-Befehl erfolgt, sollte der Cursor eine Spalte zurückbewegt werden und der an dieser Position zuletzt ausgegebene Ball gelöscht werden.

Zurückbewegt wird der Cursor mit dem Befehl PRINT CHR$(157); (Code für CURSOR LINKS). Wichtig ist wiederum das Semikolon als Abschluß des PRINT-Befehls, um zu verhindern, daß die nächste PRINT-Ausgabe in der darunterliegenden Zeile erfolgt. Der Ball, der sich an der Cursorposition befindet, wird durch Überschreiben mit einem Leerzeichen gelöscht (PRINT " ";). Beide PRINT-Befehle können

wie üblich zusammengefasst werden (PRINT CHR$(157);" ";). Fügen Sie bitte diesen zusätzlichen PRINT-Befehl mit der Zeilennummer 129 in das Programm ein.

```
100 X=1:REM STARTWERT FUER SCHLEIFENZAEHLER
110 :
120 PRINT CHR$(113);:REM BALL ZEICHNEN
129 PRINT CHR$(157);" ";:REM ALTEN BALL LOESCHEN
130 X=X+1:REM SCHLEIFENZAEHLER ERHOEHEN
140 IF X<=40 THEN GOTO 120:REM SCHLEIFE WIEDERHOLEN?
```

Dieses Programm zeichnet einen Ball, löscht den Ball wieder, zeichnet unmittelbar benachbart erneut einen Ball und so weiter. Wenn Sie das Programm starten, werden Sie sehen, daß tatsächlich der Eindruck eines über den Bildschirm "flitzenden" Balls erzeugt wird.

Allerdings wird der gerade ausgegebe Ball viel zu schnell wieder gelöscht, um länger sichtbar zu sein. Um das Programm zu perfektionieren, benötigen wie eine Art Verzögerung zwischen der Ausgabe und dem Löschen des Balls.

Eine solche Verzögerung kann mit einer "Verzögerungsschleife" erreicht werden, einer Schleife, die nur aus Schleifenanfang und Schleifenende besteht, jedoch keinerlei (!) zu bearbeitende Befehle innerhalb der Schleife selbst enthält.

```
100 X=1:REM STARTWERT F.SCHLEIFENZAEHLER
110 :
120 X=X+1:REM SCHLEIFENANFANG
130 IF X<=10 THEN GOTO 110:REM SCHLEIFE WIEDERHOLEN?
```

Im vorgestellten Demoprogramm bilden die Zeilen 120 und 130 eine Verzögerungsschleife, die durchlaufen wird, bis X den Wert 10 enthält. Sichtbare Aktionen finden nicht statt. Wie groß die erreichte Verzögerung ist, hängt von der Anzahl der Schleifendurchgänge ab. Wird die Überprüfung IF X<=10 ersetzt durch IF X<=20, ist die Verzögerung doppelt so groß wie zuvor.

Eine solche Verzögerungsschleife bauen wir nun in das Ballprogramm ein. Als Schleifenzähler darf jedoch keinesfalls die Variable X verwendet werden. X dient bereits als Zähler für die "äußere" Schleife und würde durch die Zuweisungen in der "inneren" Schleife (der Verzögerungsschleife) überschrieben und damit als Zähler unbrauchbar werden.

```
100 X=1:REM STARTWERT FUER SCHLEIFENZAEHLER
110 :
120 PRINT CHR$(113);:REM BALL ZEICHNEN
121 :
122 Y=1:REM ZAEHLER INNERE SCHLEIFE
123 Y=Y+1:ZAEHLER INNERE SCHL.ERHOEHEN
124 IF Y<=10 THEN 123:REM INNERE SCHL.WIEDERHOLEN?
125 :
129 PRINT CHR$(157);" ";:REM ALTEN BALL LOESCHEN
130 X=X+1:REM SCHLEIFENZAEHLER ERHOEHEN
140 IF X<=40 THEN GOTO 120:REM SCHLEIFE WIEDERHOLEN?
```

Fügen Sie bitte die Zeilen 121-125 in das Programm ein. Die Zeilen 121 und 125 dienen der optischen Gliederung und zur Hervorhebung der inneren Schleife. Dieses Programm erfüllt zwei Funktionen. Zum einen ist die Ballbewegung nun gut sichtbar, und zum anderen haben Sie gelernt, wie mehrere Schleifen ineinander geschachtelt werden können. Denken Sie immer daran: Bei ineinandergeschachtelten Schleifen müssen unbedingt verschiedene (!) Zählervariablen verwendet werden.

Zusammenfassung

1. Der Befehl GOTO (ZEILENNUMMER) weist den BASIC-Interpreter an, die Programmausführung mit der angegebenen Zeile fortzusetzen. GOTO ist ein "unbedingter" Sprungbefehl.

2. GOTO kann zur Bildung von "Programmschleifen" eingesetzt werden. Durch den gleichzeitigen Einsatz von IF können Schleifen mit "Abbruchbedingungen" formuliert werden, im Gegensatz zu "Endlosschleifen", die immer weiter durchlaufen würden.

3. Folgende Schritte werden zur Schleifenbildung mit GOTO benötigt:
   - Einer Zählvariablen den Startwert 1 zuweisen.
   - Befehle, die in der Schleife bearbeitet werden sollen.
   - Zählvariable um eins erhöhen.
   - Abfrage, ob die Zählvariable kleiner oder gleich der gewünschten Anzahl an Schleifendurchläufen ist. Wenn ja, Sprung mit GOTO zum Schleifenanfang ("bedingter" Sprung).

3.1.6.3 Die Bildung von Programmschleifen

Wir sahen, daß mit den Befehlen GOTO und IF Programmschleifen gebildet werden können, und daß wir exakt kontrollieren können, wie oft die in der Schleife enthaltenen Befehle ausgeführt werden sollen.

Schleifen werden in Programmen so häufig verwendet, daß jede vernünftige BASIC-Version eigene Befehle zur Bildung von Schleifenstrukturen zur Verfügung stellt. Das BASIC des C16, C116 und Plus/4 besitzt sogar mehrere derartiger "Schleifenbefehle".

Schleifen mit FOR...TO...NEXT

Die sogenannte "FOR-NEXT-Schleife" ist die am häufigsten eingesetzte Schleifenstruktur. Der FOR...TO...NEXT-Befehl (im folgenden verwende ich der Einfachheit halber den Ausdruck FOR-Befehl) arbeitet ebenso wie mit GOTO und IF erstellte Schleifen, nimmt uns jedoch einiges an Arbeit ab. Eine mit dem FOR-Befehl erstellte Schleife sieht wie folgt aus:

```
FOR (SCHLEIFENVARIABLE)=(STARTWERT) TO (ENDWERT)
BEFEHL 1
BEFEHL 2
...
...
...
BEFEHL N
NEXT (SCHLEIFENVARIABLE)
```

Beispiel:

```
100 FOR X=1 TO 40
110 PRINT CHR$(113);:REM BALL ZEICHNEN
120 FOR Y=1 TO 10:NEXT Y
130 PRINT CHR$(157);" ";:REM BALL LOESCHEN
140 NEXT X
```

Das Beispielprogramm entspricht dem Demoprogramm "Ball bewegen". Die Funktionen beider Programme sind exakt gleich. Der Unterschied besteht darin, daß zur Schleifenkonstruktion (äußere beziehungsweise innere Schleife) der FOR-Befehl eingesetzt wurde, wodurch das Programm erheblich kürzer wird.

Der Befehl FOR (SCHLEIFENVARIABLE)=(STARTWERT) TO (ENDWERT) ist der sogenannte "Schleifenanfang". Der Teil FOR (SCHLEIFENVARIABLE)=(STARTWERT) entspricht der auch zuvor von uns vorgenommenen Zuweisung eines Startwertes an eine Zählvariable, zum Beispiel X=1 oder Y=1.

Das "Schleifenende" bildet der Befehl NEXT (SCHLEIFENVARIABLE). Dieser Befehl erhöht den Wert der Schleifenvariablen um eins (analog X=X+1) und vergleicht diesen Wert anschließend mit dem angegebenen Endwert. Ist der Wert der Schleifenvariablen kleiner oder gleich dem Endwert, erfolgt ein Sprung zum Schleifenanfang (analog IF X<=40 THEN GOTO 120).

Der FOR-Befehl stellt daher im Grunde nichts anderes als eine Abkürzung für die von uns verwendeten Zuweisungs-, Vergleichs- und Sprungbefehle dar. Diese Befehle werden automatisch ausgeführt, ohne daß wir uns darum kümmern müssen.

Ein weiteres Beispiel:

```
100 FOR X=1 TO 100:PRINT X:NEXT X
110 PRINT "ALLE ZAHLEN ZWISCHEN 1 U.100 WURDEN AUSGEGEBEN"

RUN
1
2
3
...
...
...
98
99
100
ALLE ZAHLEN ZWISCHEN 1 U.100 WURDEN AUSGEGEBEN
```

Dieses aus nur zwei Zeilen bestehende Programm ersetzt das Programm:

```
100 X=X+1:PRINT X
110 IF X<>100 THEN GOTO 100
120 PRINT "ALLE ZAHLEN ZWISCHEN 1 U.100 WURDEN AUSGEGEBEN"
```

Programmschleifen sind erheblich besser zu verstehen, wenn Schleifenanfang, Schleifenbefehle und Schleifenende optisch gegliedert werden. Eine solche Gliederung ist mit Hilfe des Zeichens ":" möglich. Doppelpunkte können in Programmen beliebig zur Gliederung eingesetzt werden (jedoch nicht innerhalb (!) eines Befehls; nur am Zeilenanfang oder zwischen zwei Befehlen); sie werden vom BASIC-Interpreter einfach überlesen und ignoriert. Sowohl der Doppelpunkt als auch eventuell folgende Leerzeichen werden überlesen.

Wie wertvoll eine solche Gliederung sein kann, beweist das folgende Programm mit zwei ineinandergeschachtelten Schleifen.

```
100 FOR X=1 TO 3
110 : FOR Y=1 TO 3
120 :   PRINT "AEUSSERE SCHLEIFE: DURCHGANG NR.";X
130 :   PRINT "INNERE SCHLEIFE: DURCHGANG NR.";Y
140 : NEXT Y
150 : PRINT:REM LEERZEILE
160 NEXT X

RUN
AEUSSERE SCHLEIFE: DURCHGANG NR. 1
INNERE SCHLEIFE: DURCHGANG NR. 1
AEUSSERE SCHLEIFE: DURCHGANG NR. 1
INNERE SCHLEIFE: DURCHGANG NR. 2
AEUSSERE SCHLEIFE: DURCHGANG NR. 1
INNERE SCHLEIFE: DURCHGANG NR. 3

AEUSSERE SCHLEIFE: DURCHGANG NR. 2
INNERE SCHLEIFE: DURCHGANG NR. 1
AEUSSERE SCHLEIFE: DURCHGANG NR. 2
INNERE SCHLEIFE: DURCHGANG NR. 2
AEUSSERE SCHLEIFE: DURCHGANG NR. 2
INNERE SCHLEIFE: DURCHGANG NR. 3

AEUSSERE SCHLEIFE: DURCHGANG NR. 3
INNERE SCHLEIFE: DURCHGANG NR. 1
AEUSSERE SCHLEIFE: DURCHGANG NR. 3
INNERE SCHLEIFE: DURCHGANG NR. 2
AEUSSERE SCHLEIFE: DURCHGANG NR. 3
INNERE SCHLEIFE: DURCHGANG NR. 3
```

Dieses Programm demonstriert sehr anschaulich die Verschachtelung mehrerer Schleifen. Für die äußere Schleife wird die Schleifenvariable X, für die innere Schleife die Zählvariable Y verwendet.

Bei jedem Durchgang der äußeren Schleife werden alle in Ihr enthaltenen Befehle abgearbeitet. Wie Sie sehen, können diese Befehle wiederum aus Programmschleifen bestehen. Die innere Schleife wird komplett (!) bearbeitet (drei Druchgänge), bevor der zweite Durchlauf der äußeren Schleife erfolgt.

```
100 FOR X=1 TO 3
110 : FOR Y=1 TO 3
120 :   PRINT "AEUSSERE SCHLEIFE: DURCHGANG NR.";X
130 :   PRINT "INNERE SCHLEIFE: DURCHGANG NR.";Y
140 : NEXT Y
150 : PRINT:REM LEERZEILE
160 NEXT X
```

Ineinandergeschachtelte Schleifen erlauben uns, einen Vorgang, der aus mehreren Wiederholungen besteht, selbst wiederum mehrmals zu wiederholen. Ein Beispiel für einen solchen Vorgang ist der Ball, der sich über den Bildschirm bewegt. Eine Schleife wurde benötigt, um den Ball 40-mal je eine Spalte weiterzubewegen. Mit einer zusätzlichen äußeren Schleife kann der gesamte Vorgang (Ball über den Bildschirm bewegen) beliebig oft wiederholt werden.

```
100 FOR Z=1 TO 10:REM ANFANG AEUSSERE SCHLEIFE
110 : FOR X=1 TO 40:REM ANFANG INNERE SCHLEIFE
120 :   PRINT CHR$(113);:REM BALL ZEICHNEN
130 :   FOR Y=1 TO 10:NEXT Y:REM VERZOEGERUNGSSCHLEIFE
140 :   PRINT CHR$(157);" ";:REM BALL LOESCHEN
150 : NEXT X:REM ENDE INNERE SCHLEIFE
160 NEXT Z:REM ENDE AUESSERE SCHLEIFE
```

Mit dem erweiterten Programm wird der Vorgang der Ballbewegung zehnmal wiederholt. Dank der optischen Gliederung ist die Verschachtelung von äusserer und innerer Schleife gut verständlich.

Der Schleifenbefehl FOR ist für die BASIC-Programmierung ebenso wichtig wie die Verzweigung mit IF. Entwickeln Sie eigene Beispielprogramme und experimentieren Sie solange, bis Sie sich sicher fühlen. Wenn dies der Fall ist, können wir den

Schwierigkeitsgrad erhöhen, indem ich Ihnen das "erweiterte" Format des FOR-Befehls vorstelle.

Die STEP-Anweisung

Der FOR-Befehl wird meist in der vorgestellten Form verwendet. Für spezielle Probleme wird jedoch ab und zu die erweiterte Fassung FOR (SCHLEIFENVARIABLE) = (STARTWERT) TO (ENDWERT) STEP (SCHRITTWEITE) benötigt. Wie wir sahen, erhöht jeder NEXT-Befehl die Schleifenvariable um eins, ebenso wie X=X+1. Angenommen, wir wollen jede zweite Zahl zwischen eins und 100 ausgeben. Das Problem ist am einfachsten zu lösen, wenn wir die Zählvariable nicht um eins, sondern jeweils um zwei erhöhen.

```
100 X=1:REM STARTWERT FUER ZAEHLVARIABLE
110 PRINT X
120 X=X+2:REM UM ZWEI (!) ERHOEHEN
130 IF X<=100 THEN GOTO 110

RUN
1
3
5
...
...
...
95
97
99
```

Der Zusatz STEP (SCHRITTWEITE) gestattet die Angabe einer beliebigen "Schrittweite", eines (ganzzahligen!) Wertes, um den die Zählvariable beim Erreichen des Schleifenendes erhöht wird. Das obige Programm kann daher auch unter Verwendung des erweiterten FOR-Befehls erstellt werden.

```
100 FOR X=1 TO 100 STEP 2:REM SCHRITTWEITE 2 !
110 : PRINT X
120 NEXT X
```

Auch negative Schrittweiten können angegeben werden, zum Beispiel -1 (Erniedrigung der Schleifenvariablen um eins nach jedem Durchlauf).

Jedoch Vorsicht: Eine Schleife wird verlassen, wenn die Zählvariable den angegebenen Endwert überschritten hat. Im Programm:

```
100 FOR X=1 TO 100 STEP -2:REM SCHRITTWEITE -2
110 : PRINT X
120 NEXT X
```

besitzt X im ersten Durchgang den Wert eins, im zweiten Durchgang den Wert -1 (Erniedrigung um zwei), im dritten den Wert -3, dann den Wert -5 und so weiter. Der angegebene Endwert 100 wird nie erreicht, das Programm befindet sich in einer Endlosschleife.

Bei der Verwendung negativer Schrittweiten muß daher ein niedrigerer Start- als Endwert angegeben werden. Wird diese Regel berücksichtigt, können negative Schrittweiten sinnvoll eingesetzt werden, zum Beispiel um unser Programm "rückwärts" zählen zu lassen.

```
100 FOR X=100 TO 1 STEP -2:REM SCHRITTWEITE -2
110 : PRINT X
120 NEXT X
```

```
RUN
100
98
96
...
...
...
6
4
2
```

Schleifen mit dem DO-Befehl

Der FOR-Befehl wird zur Bildung von Schleifen verwendet, wenn bekannt ist, wie oft die Schleife durchlaufen werden soll. Oftmals ist dieser Wert unbekannt und die Schleife soll durchlaufen werden, bis eine bestimmte Bedingung erfüllt oder nicht mehr erfüllt ist. In diesem Fall werden Schleifen mit dem DO-Befehl gebildet.

Ein Beispiel: Sie interessiert, nach wie vielen Jahren aus einem Anfangskapital von 1000 DM bei einem Zinssatz von 5% ein Endkapital von mindestens 10000 DM entsteht (unter Berücksichtigung der Zinseszinsen).

Die Anzahl der Schleifendurchgänge ist in diesem Fall unbekannt. Das Programm soll so lange die Zinsen für ein weiteres Jahr berechnen, bis der gewünschte Betrag 10000 DM erreicht oder überschritten wird.

```
100 KA=1000:REM KAPITAL
110 ZI=5:REM ZINSSATZ IN PROZENT
120 JA=0:REM STARTWERT FUER ANZAHL AN JAHREN
130 :
140 DO UNTIL KA>=10000
150 : KA=KA+KA/100*ZI:REM KAPITAL BERECHNEN
160 : JA=JA+1:REM JAHRESZAEHLER ERHOEHEN
```

```
170 LOOP
180 :
190 PRINT "NACH";JA;"JAHREN BETREAGT DAS KAPITAL";KA;"DM"

RUN
NACH 48 JAHREN BETRAEGT DAS KAPITAL 10401.2697 DM
```

Diese Schleife besitzt folgendes allgemeines Format:

```
DO UNTIL (BEDINGUNG)
  BEFEHL 1
  BEFEHL 2
  ...
  ...
  BEFEHL N
LOOP
```

DO UNTIL (BEDINGUNG) bildet den Schleifenanfang, LOOP das Schleifenende. Die Befehle innerhalb der Schleife werden solange ausgeführt, bis die angegebene Bedingung erfüllt ist. Eine sehr ähnliche Konstruktion wird durch Ersetzen des Befehls UNTIL durch den Befehl WHILE gebildet:

```
DO WHILE (BEDINGUNG)
  BEFEHL 1
  BEFEHL 2
  ...
  ...
  BEFEHL N
LOOP
```

Mit DO WHILE wird eine Schleife gebildet, die ausgeführt ist, solange die angegebene Bedingung erfüllt ist. Das Demoprogramm kann daher auch mit DO WHILE erstellt werden.

```
100 KA=1000:REM KAPITAL
110 ZI=5:REM ZINSSATZ IN PROZENT
120 JA=0:REM STARTWERT FUER ANZAHL AN JAHREN
130 :
140 DO WHILE KA<10000
150 : KA=KA+KA/100*ZI:REM KAPITAL BERECHNEN
160 : JA=JA+1:REM JAHRESZAEHLER ERHOEHEN
170 LOOP
180 :
190 PRINT "NACH";JA;"JAHREN BETREAGT DAS KAPITAL";KA;"DM"
```

Zum besseren Verständnis: Der Befehl DO UNTIL (BEDINGUNG) kann übersetzt werden mit "Führe aus, bis die angegebene Bedingung erfüllt ist". DO WHILE wird übersetzt mit "Führe aus, solange die angegebene Bedingung erfüllt ist".

Die Befehle DO UNTIL KA>=10000 und DO WHILE KA<10000 sind daher logisch äquivalent ("Führe aus, bis KA größer oder gleich 10000 ist" und "Führe aus, solange KA kleiner 10000 ist").

Die korrekte Anwendung dieser beiden Versionen des DO-Befehls erfordert zugegebenermaßen eine gehörige Portion an logischem Verständnis. Ich versichere Ihnen jedoch, daß dieses ein gerade zwangsläufiges Ergebnis intensiven Programmierens ist.

Ein weiteres Beispiel: Professionelle Programme verwenden sogenannte "Eingabeprüfungen". Das heißt, Eingaben des Benutzers werden geprüft, bevor eventuell unsinnige Eingaben Berechnungen verfälschen können. Angenommen, der Benutzer soll auf eine Frage wie "Wollen Sie die Adresse löschen (J/N) ?" nur mit "J" oder "N" antworten. Nach erfolgter Eingabe prüft das Programm, ob "J" beziehungsweise "N" eingegeben wurde. Wenn nicht, wird die Eingabeaufforderung wiederholt. Wir besitzen inzwischen mehrere Möglichkeiten zur Lösung dieses Problems. Die "unschönste" Lösung verwendet den Befehl GOTO.

```
100 INPUT "WOLLEN SIE DIE ADRESSE LOESCHEN (J/N)";X$
110 IF X$<>"J" AND X$<>"N" THEN GOTO 100
```

In dieser Version wird das eingegebene Zeichen (X$) mit "J" beziehungsweise "N" verglichen. Entspricht es keinem der beiden Zeichen (ist es sowohl ungleich "J" als auch ungleich "N"), wird die Eingabezeile erneut angesprungen.

Diese Lösung ist unschön, da jeder Sprungbefehl die Übersichtlichkeit eines Programms vermindert. Stellen Sie sich ein Programm mit hunderten von GOTO-Befehlen vor. Um den Programmablauf nachzuvollziehen, müssen Sie andauernd nach den angegebenen Zeilennummern Ausschau halten.

Weitaus besser verständlich ist das Programm, wenn eine DO UNTIL oder DO WHILE-Konstruktion angewendet wird.

Mit DO UNTIL:

```
100 X$="":REM EINGABEVARIABLE LOESCHEN
110 DO UNTIL X$="J" OR X$="N"
120 : INPUT "WOLLEN SIE DIE ADRESSE LOESCHEN (J/N)";X$
130 LOOP
```

Äquivalente Version mit DO WHILE:

```
100 X$="":REM EINGABEVARIABLE LOESCHEN
110 DO WHILE X$<>"J" AND X$<>"N"
120 : INPUT "WOLLEN SIE DIE ADRESSE LOESCHEN (J/N)";X$
130 LOOP
```

Die Bedingungen DO UNTIL X$="J" OR X$="N" ('Führe aus, bis X$ entweder gleich "J" oder gleich "N" ist') und DO WHILE X$<>"J" AND X$<>"N" ('Führe aus, solange X$ ungleich "J" und ungleich "N" ist') sind wiederum logisch äquivalent. Das Beispiel zeigt auch, daß ebenso wie beim IF-Befehl die Gesamtbedingung aus mehrenen Teilbedingungen bestehen kann, die mit logischen Operatoren verknüpft werden.

Wenn Sie nun glauben, der DO-Befehl sei reichlich kompliziert, so muß ich Sie leider frustrieren. Tatsächlich ist dieser Befehl noch erheblich komplexer als Sie annehmen, da sich die Schleifenbedingung UNTIL (BEDINGUNG) beziehungsweise

WHILE (BEDINGUNG) auch am Ende der Schleife befinden kann.

```
DO                          DO
  BEFEHL 1                    BEFEHL 1
  BEFEHL 2                    BEFEHL 2
  ...                         ...
  ...                         ...
  BEFEHL N                    BEFEHL N
LOOP UNTIL (BEDINGUNG)      LOOP WHILE (BEDINGUNG)
```

Wie Sie sehen, besteht eine Eigenschaft des Programmierens darin, daß immer (!) mehrere Möglichkeiten zur Lösung bestehen. Gute Programme zeichnen sich dadurch aus, daß der eleganteste, schnellste, kürzeste und übersichtlichste "Algorithmus" verwendet wird. Wie Sie noch feststellen werden, widersprechen sich diese Anforderungen zum Teil gegenseitig. Worauf Sie in Ihren Programmen am meisten Wert legen, ist selbstverständlich Ihre eigene Sache. Spätestens bei der Erstellung umfangreicher und komplexer Programme sollte die Übersichtlichkeit und Verständlichkeit eines Programms im Vordergrund stehen.

Zusammenfassung

1. Schleifen können wie beschrieben "per Hand" mit IF und GOTO programmiert werden, oder mit einer FOR- beziehungsweise DO-Schleife.

2. Die FOR-Schleife wird verwendet, wenn die Anzahl der Schleifendurchgänge bekannt ist. Das Format:

```
FOR (VARIABLE)=(STARTWERT) TO (ENDWERT) STEP (SCHRITTWEITE)
: (BEFEHL 1)
: (BEFEHL 2)
: ...
: ...
: (BEFEHL N)
NEXT (VARIABLE)
```

Als Schrittweite können positive und negative ganze Zahlen verwendet werden.

3. DO wird verwendet, wenn die Anzahl der benötigten Schleifendurchgänge unbekannt ist und die Schleife ausgeführt werden soll, bis eine bestimmte Bedingung erfüllt oder aber nicht mehr erfüllt ist. Ebenso wie beim IF-Befehl kann die Gesamtbedingung aus mehreren mit logischen Operatoren verknüpften Teilbedingungen bestehen.

4. DO UNTIL (BEDINGUNG): Schleifenbefehle werden ausgeführt, bis die angegebene Bedingung erfüllt ist.

```
DO UNTIL (BEDINGUNG)
  (BEFEHL 1)
  (BEFEHL 2)
  ...
  ...
  (BEFEHL N)
LOOP
```

5. DO WHILE (BEDINGUNG): Schleifenbefehle werden ausgeführt, solange die angegebene Bedingung erfüllt ist.

```
DO WHILE (BEDINGUNG)
  (BEFEHL 1)
  (BEFEHL 2)
  ...
  ...
  (BEFEHL N)
LOOP
```

6. Die Bedingung kann am auch am Ende einer DO-Schleife abgefragt werden:

```
DO
  ...
  ...
LOOP UNTIL/WHILE (BEDINGUNG)
```

3.1.6.4 Unterprogramme (GOSUB/RETURN)

Unterprogramme stellen ähnlich dem GOTO-Befehl Abweichungen vom sequentiellen Programmablauf dar. Der Sinn von Unterprogrammen: Umfangreiche Programme bestehen aus den verschiedensten Programmteilen. Eine Adressenverwaltung besteht zum Beispiel aus den grundlegenden Teilen "Eingeben", "Suchen", "Ändern" und "Löschen" von Adressen.

Oftmals bestehen verschiedene Programmteile aus teilweise identischen Befehlsfolgen. Soll zum Beispiel eine Adresse geändert werden, ist die übliche Vorgehensweise in einem Programm, die alte Adresse komplett zu löschen und die geänderte Adresse neu einzutragen.

Der Programmteil "Ändern" besteht daher im Grunde aus einer Kombination der bereits vorhandenen Teile "Eingeben" und "Löschen". Nun ist es sicherlich umständlich, bereits im Programm vorhandene Befehlsfolgen in anderen Programmteilen nochmals einzugeben. Geschickter wäre es, wenn vorhandene Teile von verschiedenen Stellen im Programm aus verwendet, "aufgerufen" werden könnten.

Derartige "wiederverwendbare" Programmteile heißen "Unterprogramme". Die Anwendung von Unterprogrammen soll an einem kleinen Beispiel erläutert werden.

```
100 GOSUB 150:REM UNTERPROGRAMM-AUFRUF
110 GOSUB 150:REM UNTERPROGRAMM-AUFRUF
120 GOSUB 150:REM UNTERPROGRAMM-AUFRUF
130 END
140 :
150 REM UNTERPROGRAMM
160 PRINT "DIES IST EIN UNTERPROGRAMM"
170 RETURN

RUN
DIES IST EIN UNTERPROGRAMM
DIES IST EIN UNTERPROGRAMM
DIES IST EIN UNTERPROGRAMM
```

Nach dem Starten dieses Programms geschieht Seltsames: Die Zeichenkette "DIES IST EIN UNTERPROGRAMM" wird dreimal ausgegeben, obwohl Sie nur einmal im Programm vorhanden ist und keinerlei Schleife für eine Wiederholung sorgt.

Die Zeilen 150-170 bilden ein Unterprogramm. Der Befehl GOSUB 150 "ruft das Unterprogramm auf". Wenn dieser Befehl bearbeitet wird, erhält der BASIC-Interpreter viel zu tun.

- Der BASIC-Interpreter merkt sich die Nummer der aktuellen Zeile.
- Es erfolgt ein Sprung zur im GOSUB-Befehl ("gehe zum Unterprogramm") angegebenen Zeile, analog dem Befehl GOTO (ZEILENNUMMER).

Der Unterschied zwischen GOTO und GOSUB besteht darin, daß GOSUB zur Speicherung der aktuellen Zeilennummer führt. Nach einem Sprung mit GOTO weiß der Interpreter nicht mehr, von wo aus der Sprung erfolgte.

Nach erfolgtem Sprung wird das Unterprogramm abgearbeitet, das heißt, wie üblich werden die einzelnen darin aufgeführten Befehle ausgeführt.

Den Abschluß eines Unterprogramms muß immer der Befehl RETURN bilden ("kehre vom Unterprogramm zurück"). Der BASIC-Interpreter kehrt zu jener Programmzeile zurück, von der aus der Sprung erfolgte und arbeitet den nächsten Befehl hinter dem GOSUB-Befehl ab (beziehungsweise die nächste Programmzeile, wenn nach GOSUB kein weiterer Befehl folgt).

Das Demoprogramm arbeitet daher wie folgt: In Zeile 100 erfolgt ein Sprung zu dem ab Zeile 150 beginnenden Unterprogramm. Der darin enthaltene PRINT-Befehl wird ausgeführt. Der RETURN-Befehl bewirkt die Rückkehr zu Zeile 100. Da hinter dem GOSUB-Befehl kein Befehl folgt, wird die nächste Programmzeile, Zeile 110, bearbeitet.

In dieser und der folgenden Zeile 120 wiederholt sich der Vorgang. Das Unterprogramm wird aufgerufen, der PRINT-Befehl ausgeführt und zu jener Zeile zurückgekehrt, an der der Unterprogrammaufruf erfolgte.

Unterprogramme besitzen mehrere Funktionen:

1. Sie sparen Schreibarbeit. Immer wieder benötigte Programmteile müssen nur einmal eingetippt werden.
2. Sie sparen kostbaren Speicherplatz.
3. Programme werden durch intensive Nutzung von Unterprogrammen übersichtlicher und klarer.

Zum letzten Punkt gestatten Sie mir bitte einige Bemerkungen. Unterprogramme können (!) zu übersichtlichen Programmen führen, müssen jedoch nicht. Voraussetzung ist, daß nur "funktionell abgeschlossene" Programmteile als Unterprogramme geschrieben werden. Sie sollten keinesfalls aus zwei identischen in einem Programm vorkommenden Programmteilen sofort ein Unterprogramm machen.

Ein Unterprogramm sollte eine "Aufgabe" besitzen. Beispiele dafür sind die erwähnten Programmteile zum "Eingeben" beziehungsweise "Löschen" einer Adresse. Diese Programmteile sind Paradebeispiele für Unterprogramme. Ein weiteres Beispiel ist eine "Sortierroutine", die alle eingegebenen Adressen alphabe-

tisch sortiert und von verschiedenen Programmteilen benutzt wird.

Das ideal gegliederte Programm (leider in der Praxis kaum zu verwirklichen) besteht aus einem "Hauptprogramm", das nur aus GOSUB-Befehlen besteht, und zahlreichen in sich abgeschlossenen Unterprogrammen mit klar umrissenen Aufgabengebieten.

```
---HAUPTPROGRAMM---
GOSUB ...:REM LOESCHEN
GOSUB ...:REM EINGEBEN
GOSUB ...:REM SUCHEN
GOSUB ...:REM AENDERN
...
...
...

---UNTERPROGRAMME---

-UNTERPROGRAMM LOESCHEN-
BEFEHL1
BEFEHL2
...
BEFEHLX
RETURN

-UNTERPROGRAMM EINGEBEN-
BEFEHL1
BEFEHL2
...
BEFEHLX
RETURN

-UNTERPROGRAMM SUCHEN-
...
...
...
```

Ein derart in einzelne "Module" gegliedertes Programm ist problemlos zu verstehen.

Unterprogramme lassen sich ebenso wie Schleifen verschachteln, das heißt ein Unterprogramm kann ein weiteres Unterprogramm aufrufen.

```
100 REM HAUPTPROGRAMM
110 GOSUB 140:REM UNTERPROGRAMM EBENE1
120 END
130 :
140 REM UNTERPROGRAMM EBENE 1
150 PRINT "EBENE 1"
160 GOSUB 190
170 RETURN
180 :
190 REM UNTERPROGRAMM EBENE 2
200 PRINT "EBENE 2"
210 RETURN
```

Der Programmablauf: Das Hauptprogramm ruft das ab Zeile 140 beginnende Unterprogramm der "ersten Ebene" auf. Der PRINT-Befehl wird ausgeführt, anschließend ruft dieses Unterprogramm ein weiteres, ab Zeile 190 beginnendes Unterprogramm auf. Der RETURN-Befehl in Zeile 210 führt zur Rückkehr in das Unterprogramm der "letzten Ebene", das heißt die Programmausführung wird mit Zeile 170 fortgesetzt, dem nächsten Befehl, der dem letzten Unterprogrammaufruf folgt. Der RETURN-Befehl in Zeile 170 bewirkt die Rückkehr auf die "nächsthöhere Ebene", das heißt in das Hauptprogramm, dessen Ausführung mit Zeile 120 fortgesetzt wird.

Wenn Unterprogramme von Ihnen unter anderem zur besseren Gliederung eingesetzt werden sollen, empfehle ich Ihnen dringend, wie im letzten Beispiel nicht mit Kommentarzeilen zu sparen. Optimal sind mehrere Kommentarzeilen vor jedem Unterprogramm, die dessen Funktion exakt erläutern.

Außerdem sollte sich hinter jedem GOSUB-Befehl ebenfalls ein Kommentar befinden, der auf die Funktion des aufgerufenen Unterprogramms hinweist (GOSUB 500:REM SORTIEREN ist sicherlich aussagekräftiger als GOSUB 500).

3.1.7 Arrays

Sie kennen nun alle wichtigen Befehle zur Kontrolle des Programmablaufs und sind bereits seit längerem in der Lage, mit numerischen und Stringvariablen umzugehen. Die in diesem Abschnitt behandelten "Arrays" (auch "Variablenfeld" oder "Matrix" genannt) besitzen sehr viel Ähnlichkeit mit den bisher bekannten Variablen. Ein Array besteht aus mehreren numerischen oder Stringvariablen und unterscheidet sich von diesen durch den Variablennamen.

Die verwendeten Variablennamen waren eine Art "Konstante", das heißt es handelte sich um starr festgelegte Bezeichnungen wie X, X$ oder KA. Die Bezeichung einer Arrayvariablen (oder auch "Feldvariable") besteht aus dem Variablennamen und einem in Klammern angegebenen "Index". Ein Array besteht aus mehreren "einfachen" Variablen, die durch den Index unterschieden werden. Beispiel: X(5) bezeichnet eine andere Variable als X(7) oder X(0). Der kleinste Index ist immer 0, der größte Index wird mit dem DIM-Befehl festgelegt (DIM (VARIABLE(GRÖSSTER INDEX)).

Beispiel 1:

DIM X(20) legt ein Array aus 21 numerischen Variablen an ("X(0)", "X(1)", "X(2)",...,"X(20)").

Beispiel 2:

DIM X$(74) legt ein Array aus 75 Stringvariablen an ("X$(0)", "X$(1)", "X$(2)",...,"X$(74)").

Die einzelnen Variablen eines Arrays aus numerischen oder Stringvariablen können wie gewohnt verwendet werden.

```
100 DIM X(20):REM ARRAY MIT 21 NUMERISCHEN VARIABLEN
110 X(0)=10:REM ZUWEISUNG
120 X(1)=20:REM ZUWEISUNG
130 X(2)=X(0)+X(1):REM X(0) UND X(1) ADDIEREN
140 PRINT X(2):REM SUMME AUSGEBEN
```

Sie stellen sich nun sicherlich die Frage, welchen Sinn Arrayvariablen besitzen, die sich doch anscheinend nur durch einen recht umständlichen Variablennamen von einfachen Variablen unterscheiden. Um diese Frage zu beantworten, müssen Sie wissen, daß der Index eine Zahl ist und - wie uns seit langem bekannt ist - Zahlen durch numerische Variablen ersetzt werden können. Zum "Ansprechen" einer bestimmten Variable kann daher wiederum eine Variable verwendet werden, wie im folgenden Beispiel.

```
100 DIM X(20):REM ARRAY MIT 21 NUMERISCHEN VARIABLEN
110 Z=0:REM ZUWEISUNG AN EINFACHE VARIABLE
120 X(Z)=10:REM ZUWEISUNG AN ARRAYVARIABLE
130 Z=Z+1:REM ZUWEISUNG AN EINFACHE VARIABLE
140 X(Z)=20:REM ZUWEISUNG
150 Z=Z+1:REM ZUWEISUNG AN EINFACHE VARIABLE
160 X(2)=X(0)+X(1):REM X(0) UND X(1) ADDIEREN
170 PRINT X(2):REM SUMME AUSGEBEN
```

Dieses Programm weist ebenfalls den Variablen X(0) und X(1) einen Wert zu, addiert beide Variablen und weist die Summe der Variablen X(2) zu, um diese Summe anschließend auszugeben.

Zur Identifikation der Variablen X(0), X(1) und X(2) wird die einfache numerische Variable Z verwendet. Z wird in Zeile 110 der Wert 0 zugewiesen. Der Ausdruck X(Z)=10 in Zeile 120 entspricht daher dem Ausdruck X(0)=10. Die Zuweisung erfolgt nicht direkt, sondern "durch die Hintertür".

Wenn Sie der Ansicht sind, daß dieses Programm mit Sicherheit umständlicher und weniger durchschaubar ist als die erste Version, muß ich Ihnen recht geben. Es sollte lediglich den prinzipiellen Umgang mit Arrayvariablen demonstrieren. Effektiv eingesetzt werden Arrays meist in Verbindung mit Schleifenstrukturen.

```
100 DIM X(20):REM ARRAY MIT 21 NUMERISCHEN VARIABLEN
110 FOR I=0 TO 20
120 : X(I)=10
130 NEXT I
```

Dieses Programm legt ein Array aus 21 numerischen Variablen an (X(0) bis X(20)) und weist allen 21 Variablen den Wert zehn zu. Um die einzelnen Variablen anzusprechen, wird die Schleifenvariable I als Indexzahl verwendet. Wie wir wissen, wird I bei jedem Schleifendurchgang um eins erhöht. I besitzt beim ersten Schleifendurchgang den Startwert 0, die Zuweisung des Wertes 10 erfolgt daher an die Variable X(0).

Durch den Befehl NEXT I wird I um eins erhöht und besitzt beim zweiten Durchgang den Wert 1. Die nächste Zuweisung erfolgt daher zur Variablen X(1). Beim letzten Schleifendurchgang besitzt I den Endwert 20 und der Variablen X(20) wird der Wert 10 zugewiesen. Die Verwendung einfacher Variablen würde 21 einzelne Zuweisungen erfordern statt einer kurzen Schleife mit einem Zuweisungsbefehl.

Ein praktisches Beispiel: Sie schreiben eine Mini-Dateiverwaltung. Der Benutzer gibt zehn Adressen ein, die anschließend wieder ausgegeben werden sollen. Mit einfachen Variablen wird das Problem wie folgt gelöst:

```
100 INPUT "ADRESSE";A$
110 INPUT "ADRESSE";B$
120 INPUT "ADRESSE";C$
130 INPUT "ADRESSE";D$
140 INPUT "ADRESSE";E$
150 INPUT "ADRESSE";F$
160 INPUT "ADRESSE";G$
170 INPUT "ADRESSE";H$
180 INPUT "ADRESSE";I$
190 INPUT "ADRESSE";J$
200 :
210 PRINT A$
220 PRINT B$
230 PRINT C$
240 PRINT D$
250 PRINT E$
260 PRINT F$
270 PRINT G$
280 PRINT H$
290 PRINT I$
300 PRINT J$
```

Das abgebildete Programm-Listing ist das Gegenteil dessen, was ich unter einem kurzen und eleganten Programm verstehe. Mit Hilfe eines Arrays aus zehn Stringvariablen wird das Programm erheblich kürzer.

```
100 DIM A$(9):REM ARRAY A$(0) BIS A$(9) ANLEGEN
110 :
120 FOR X=0 TO 9
130 : INPUT "ADRESSE";A$(X)
140 NEXT X
150 :<<
160 FOR X=0 TO 9
170 : PRINT A$(X)
180 NEXT X
```

Die zehn Eingaben von Adressen und Zuweisungen erfolgen in einer Schleife, wobei die von 0 bis 9 "laufende" Schleifenvariable X als Indexzahl für die angesprochenen Arrayvariablen A$(0) bis A$(9) verwendet wird.

Analog verläuft die Ausgabe. Die Inhalte der Variablen A$(0) bis A$(9) werden in einer Schleife ausgegeben. Die Schleifenvariable X wird wiederum als fortlaufende Indexzahl verwendet.

Dieses Beispiel demonstriert zum ersten Mal den effektiven Einsatz von Arrayvariablen. In einer echten Adressenverwaltung werden hunderte von Adressen in Stringvariablen gespeichert. Jede vom Benutzer neu eingegebene Adresse erfordert eine neue Zuweisung. In einer "Eingabeschleife" verwendete Arrayvariablen ersetzen in diesem Fall hunderte von Programmzeilen mit Zuweisungsbefehlen.

3.1.7.1 Arrays und der verfügbare Speicherplatz

Eine Frage bleibt offen: Wozu wird der DIM-Befehl benötigt, der bei einfachen Variablen überflüssig ist?

Arrays müssen vor der ersten Benutzung "angelegt" werden. Bei diesem Vorgang geben Sie an, wie viele Arrayvariablen Sie maximal benötigen und der BASIC-Interpreter reserviert Platz im Rechnerspeicher zur Ablage der Variablen. Dieser Vorgang ist außerordentlich wichtig, da er nicht rückgängig gemacht werden kann!

Um mit dem DIM-Befehl richtig umzugehen, müssen Sie wissen, wieviel Platz eine Arrayvariable benötigt. Der Platzbedarf ist vom Array-Typ abhängig. Numerische Arrayvariablen benötigen fünf "Byte" pro Variable, Stringvariablen drei "Byte".

Ein "Byte" ist die Grundeinheit, in der der Speicherplatz Ihres Rechners gemessen wird. Der C16 und C116 verfügen über etwa 12000 Byte, der Plus/4 über etwa 60000 Byte, die zur Speicherung von Programmen und Daten im Rechners genutzt werden können.

Wenn man den Platzbedarf für das Programm selbst nicht berücksichtigt, kann daher auf dem Plus/4 ein numerisches Array mit etwa 12000 Variablen angelegt werden. "So viele Variablen benötige ich niemals", denken Sie nun sicherlich. Das eigentliche Problem stellen jedoch vor allem Stringvariablen dar. Beim Anlegen eines Arrays aus Stringvariablen werden pro Variable drei Byte an Speicherplatz benötigt, so daß theoretisch ein Array mit bis zu 20000 Stringvariablen auf dem Plus/4 angelegt werden kann.

Erfolgt nun jedoch eine Zuweisung an eine dieser Stringvariablen, wird zusätzlicher Speicherplatz benötigt. Die Zeichen, die der Variablen zugewiesen werden, müssen offensichtlich irgendwo im Speicher untergebracht werden. Jedes in einer Stringvariablen enthaltene Zeichen benötigt ein zusätzliches Byte. Die Zuweisung A$(3)="HANS MAIER" belegt daher weitere zehn Byte.

Aus Gründen, deren Erläuterung zu weit führen dürfte, benötigt der BASIC-Interpreter weitere zwei Byte bei jeder Zuweisung an eine Stringvariable, unabhängig von der Länge der zugewiesenenen Zeichenkette.

Halten wir fest: Beim Anlegen eines numerischen Arrays mit dem DIM-Befehl werden fünf Byte pro Variable benötigt, beim Anlegen eines Stringarrays drei Byte pro Stringvariable. Wird einer Stringvariablen eine Zeichenkette zugewiesen, belegt diese Zeichenkette pro Zeichen ein weiteres Byte. Weitere zwei Byte

Speicherplatz werden unabhängig von der Länge der Zeichenkette bei jeder Zuweisung an eine Stringvariable belegt.

Mit diesen Kenntnissen können wir nun berechnen, wie viele Adressen sich mit einem Adressenprogramm auf dem C16, C116 beziehungsweise Plus/4 verwalten lassen. Gehen wir davon aus, daß jede Adresse in einer Variablen des Stringarrays A$(...) untergebracht wird. Wenn eine Adresse (Name, Vorname, Strasse, Postleitzahl, Wohnort etc.) im Durchschnitt aus 70 Zeichen besteht, errechnet sich der pro Adresse benötigte Speicherplatz wie folgt:

1. Drei Byte beim Anlegen des Stringarrays.
2. Durchschnittlich 70 Byte pro Zuweisung plus 2 weitere "Verwaltungs-Bytes" des BASIC-Interpreters.

Pro Adresse benötigen wir circa 75 Byte. Mit einem C16 oder können daher maximal 160 (12000/75=160), mit einem Plus/4 800 (60000/75=800) Adressen verwaltet werden. Wahrscheinlich sind 160 Adressen in den meisten Fällen völlig ausreichend. Die Rechnung besitzt jedoch mehrere Haken: In umfangreicheren Programmen werden fast immer mehrere Arrays benötigt und eine Vielzahl einfacher Variablen, die ebenfalls Speicherplatz benötigen.

Einfache Variablen müssen bei Überschlagsrechnungen nicht unbedingt berücksichtigt werden. Eine Vielzahl von numerischen oder Stringarrays mit jeweils 20, 30 oder mehr Variablen können sich jedoch schnell als "Speicherplatzfresser" erweisen.

Das Programm, das die Adressen verwaltet, muß in einer Überschlagsrechnung unbedingt berücksichtigt werden. Auch das Programm selbst wird wie die Inhalte von Variablen im Rechnerspeicher untergebracht und belegt Speicherplatz. Wieviel Platz das Programm benötigt, hängt von der Länge ab. Man kann jedoch davon ausgehen, daß es kaum möglich ist, ein vernünftiges Programm zur Adressenverwaltung zu schreiben, das weniger als etwa 5000 Byte benötigt. Werden alle diese Faktoren berücksichtigt, schrumpft der scheinbar so große Speicherplatz sehr schnell (vor allem auf dem Plus/4).

Folgender Fall tritt in der Praxis häufig auf: Auf dem Bildschirm erscheint während eines Programmlaufs die Fehlermeldung "OUT OF MEMORY ERROR IN (ZEILENNUMMER)". Diese Fehlermeldung sagt aus, daß in der betreffenden Programmzeile eine Operation ausgeführt werden sollte, die mehr Speicherplatz benötigt, als momentan noch verfügbar ist, das heißt der Speicher ist "voll".

Tritt diese Fehlermeldung auf, müssen Sie dafür sorgen, daß vom Programm und den Variablen weniger Platz benötigt wird. Am Platzbedarf des Programms läßt sich normalerweise wenig ändern, jedoch am Platzbedarf der verwalteten Daten. Der übliche Ausweg besteht in der Verkleinerung der angelegten Arrays, das heißt in den DIM-Befehlen werden kleinere maximale Indexzahlen angegeben.

Derartige Änderungen sind meistens im Laufe der Entwicklung eines Programms nötig, wenn dieses immer größer und umfangreicher wird. Den gesamten Platzbedarf vor der Erstellung eines Programms exakt zu berechnen, ist leider nicht möglich, da nicht bekannt ist, wie umfangreich das Programm letztendlich wird.

Ein Tip: Mit der FRE-Funktion kann abgeschätzt werden, wieviel Speicherplatz momentan zur Verfügung steht. Die Syntax lautet: FRE(0). Wenn Sie wissen wollen, wieviel Byte ein Programm und die darin verwendeten Variablen benötigen, gehen Sie wie folgt vor:

1. Starten Sie das Programm mit RUN.

2. "Spielen" Sie alle Programmfunktionen durch (in einer Adressenverwaltung zum Beispiel "Eingeben", "Suchen", "Löschen" und "Ändern" von Adressen), damit möglichst jede Programmzeile abgearbeitet und alle darin enthaltenen Zuweisungen ausgeführt werden.

3. Unterbrechen Sie den Programmablauf mit der STOP-Taste.

4. Geben Sie ein: PRINT FRE(0).

5. Die ausgegebene Zahl gibt den freien Speicherplatz in Bytes an.

6. Geben Sie ein: NEW, um sowohl das Programm als auch die Inhalte aller Variablen zu löschen.

7. Geben Sie erneut PRINT FRE(0) ein und subtrahieren Sie vom ausgegebenen Wert den mit dem ersten FRE-Befehl ausgegebenen Wert. Sie erhalten die Anzahl der Bytes, die Programm plus Daten belegen.

3.1.7.2 Integervariablen und Integerarrays

Numerische Variablen sind Ihnen inzwischen ein Begriff. Es existieren jedoch zwei verschiedene Typen numerischer Variablen. Den bisher kennengelernten Typ bezeichnet man als "Fließkommavariable" beziehungsweise als "Fließkommaarray". Dieser Typ wird dadurch gekennzeichnet, daß Fließkommavariablen außer ganzen Zahlen auch Kommazahlen zugewiesen werden können, im Gegensatz zu den sogenannten "Integervariablen" oder "Integerarrays".

"Integervariablen" sind mit Fließkommavariablen eng verwandt. Im Unterschied zu diesen werden sie durch das Zeichen "%" hinter dem eigentlichen Variablennamen gekennzeichnet (zum Beispiel X% oder Z1%). Dem Integertyp können ebenfalls Zahlen zugewiesen werden, jedoch nur ganze Zahlen im Bereich zwischen -32767 und +32767. Eine Zuweisung wie zum Beispiel X%=3.56 ist daher nicht möglich.

Integervariablen besitzen Fließkommavariablen gegenüber keinerlei Vorteile, im Unterschied zu Integerarrays. Wir sahen, daß es weder auf dem C16 noch auf dem Plus4 sinnvoll ist, verschwenderisch mit dem verfügbaren Speicherplatz umzu-

gehen, und daß vor allem große numerische oder Stringarrays enorm viel Platz verbrauchen.

Fließkommaarrays benötigen fünf Byte pro Variable. Der Platzbedarf von Integerarrays ist mit zwei Byte pro Variable deutlich geringer. Es ist daher zu empfehlen, Fließkommaarrays (X(100)) wo immer möglich durch Integerarrays (X%(100)) zu ersetzen. Voraussetzung ist natürlich, daß den Arrayvariablen nur ganze Zahlen im angegebenen Bereich zugewiesen werden müssen.

Zusammenfassung

1. Arrays bestehen aus mehreren Variablen des gleichen Typs, die über eine Indexzahl angesprochen werden. Die Vorteile von Arrays bestehen vor allem in der Verwendung von Variablen als Indexzahl.

2. Arrays müssen vor der ersten Benutzung "dimensioniert" werden, das heißt mit dem Befehl DIM wird angegeben, wie viele Variablen das Array enthalten soll.

3. Fließkommaarrays (X(...)) belegen pro angelegter Variable fünf, Integerarrays (X%(...)) drei, und Stringarrays (X$(...)) ebenfalls drei Byte an Speicherplatz. Zuweisungen an eine Stringvariable belegen zusätzlich zwei Byte plus für jedes Zeichen der Zeichenkette ein weiteres Byte.

3.2 Graphik

Endlich ist es soweit: Sie beherrschen alle grundlegenden BASIC-Befehle. In den folgenden Kapiteln stehen nicht mehr Befehle, sondern Programme im Vordergrund. Im Graphik-Kapitel wird ein Malprogramm entwickelt. Die Anforderungen und der Aufbau des Programms werden behandelt, bei der anschließenden Umsetzung des Ablaufs in ein Programm erlernen Sie die zur Umsetzung benötigten Graphik-Befehle "nebenbei".

3.2.1 Die Graphik-Modi des C16, C116, Plus/4

Der C16, C116 bzw. Plus/4 besitzt eine Vielzahl spezieller Graphik-Befehle. Sie wollen nun sicherlich wissen, was unter dem Begriff "Graphik" zu verstehen ist. Prinzipiell muß zwischen dem "Text-Modus" und dem "Graphik-Modus" unterschieden werden. Im Text-Modus können Sie mit Ausgabebefehlen wie PRINT Zeichen auf dem Bildschirm ausgeben. Jedes dieser Zeichen besteht aus einzelnen Punkten (je acht Zeichen in horizontaler und vertikaler Richtung, insgesamt 64 Punkte pro Zeichen). Einzelne Punkte können im Text-Modus nicht beeinflußt werden.

Im Graphik-Modus arbeiten Sie ausschließlich (!) mit einzelnen Punkten, die Sie an beliebigen Positionen setzen oder löschen können. In diesem Modus kann zum Beispiel ein Programm entwickelt werden, das eine beliebige mathematische Funktion aus einzelnen Punkten zeichnet. Im Text-Modus ist das Zeichnen einer Funktion aufgrund der geringen Auflösung (25 Zeilen mit je 40 Zeichen) kaum möglich. Im Graphik-Modus können dank der weitaus höheren Auflösung Funktionen äußerst fein dargestellt werden.

Man unterscheidet zwei verschiedene Graphik-Modi:

1. Die hochauflösende Graphik, auch "Hi-Res-Modus" genannt (Hi-Resolution = hohe Auflösung): In diesem Modus ist der Bildschirm in 320*200 Punkte aufgeteilt (320 Punkte in horizontaler, 200 Punkte in vertikaler Richtung). Im Gegensatz zum Text-Modus, in dem jedem Zeichen eine individuelle Farbe gegeben werden kann, können zwar Hintergrund- und Punktfarbe eingestellt werden, zwei unmittelbar benachbarte Punkte können jedoch nicht in verschiedenen Farben dargestellt werden.

2. Die mehrfarbige Graphik, auch "Multicolor-Modus" genannt: In diesem Modus steht zwar in horizontaler Richtung nur die halbe Auflösung zur Verfügung (160 Punkte), das heißt die Punkte sind doppelt so breit, dafür verfügen Sie jedoch über vier Farben: Ebenso wie in allen anderen Modi kann die Farbe des Hintergrundes beliebig eingestellt werden; im Unterschied zum Hi-Res-Modus kann jeder einzelne Punkt in einer von drei Punktfarben dargestellt werden.

Bei der Behandlung der Graphik-Befehle werde ich mich vorwiegend mit dem Hi-Res-Modus beschäftigen. Am Ende dieses Kapitels werden die speziellen Farbeigenschaften des Multicolor-Modus erläutert und erklärt, wie Programme, die für den Hi-Res-Modus entwickelt wurden, zur Zusammenarbeit mit dem Multicolor-Modus abgeändert werden.

Ein wichtiger Hinweis für alle C16-Besitzer: Beim Einschalten der Graphik (Hi-Res- und Multicolor-Modus) gehen Ihnen etwa zehn Kilobyte an verfügbarem Benutzerspeicher verloren. Es verbleiben noch circa zwei Kilobyte für ein Graphikprogramm und benötigte Daten. Diese zwei Kilobyte sind leider nur für sehr kleine Programme ausreichend. Sollten Sie des öfteren die Fehlermeldung "OUT OF MEMORY ERROR IN (ZEILENNUMMER)" erhalten, müssen Sie versuchen, das Programm zu kürzen, selbst wenn dies nur auf Kosten ausführlicher Kommentare möglich ist.

Um Ihnen einen Vorgeschmack auf das zu entwickelnde Malprogramm zu geben, empfehle ich Ihnen, das folgende Programmlisting einzugeben und zu starten. Es zeichnet Ihnen einen kleine Demographik im Hi-Res-Modus auf den Bildschirm.

```
100 GRAPHIC 1,1
110 FOR I=10 TO 130 STEP 3
120 : BOX 1,I*2,I,I*2+50,I+50
130 NEXT
140 GETKEY A$:GRAPHIC 0
```

Wenn die Zeichnung beendet ist, drücken Sie bitte eine beliebige Taste. Sie kehren anschließend in den normalen Text-Modus zurück.

3.2.1.1 Anforderungen an das Malprogramm

Sinn eines Malprogrammes ist es, den Computer in eine Art "Leinwand" zu verwandeln. Auf dieser Leinwand soll wie mit einem Pinsel gezeichnet werden können. Ein gutes Malprogramm bietet außer dem Zeichnen mit einem "computerisierten Pinsel" komfortable Funktionen wie zum Beispiel:

- Zeichnen einer Linie
- Zeichnen von Rechtecken

Diese Komfort-Funktionen sollen mit möglichst wenig Aufwand zu bedienen sein. So soll es genügen, die beiden Endpunkte einer Linie zu markieren, um von dem Programm die Linie zeichnen zu lassen. Unser Programm soll selbstverständlich alle genannten Funktionen ebenfalls bieten. Alle Funktionen sollen mit einem einzigen Tastendruck angewählt werden.

Zur Anwahl verschiedener Programmfunktionen werden häufig die erwähnten Funktionstasten verwendet. Einigen wir uns darauf (ich gebe zu: beim Lesen dieses Buches haben Sie sehr wenig Mitspracherecht), die Funktionstasten wie folgt zu verwenden:

- Mit F1 wird die Funktion "Linie ziehen" angewählt.
- Mit F2 wird "Rechteck malen" angewählt.

Zum "Freihandzeichnen" verwenden wir am besten die Cursortasten. CURSOR OBEN zeichnet einen Punkt oberhalb der aktuellen Position, CURSOR UNTEN unterhalb dieser Position und so weiter.

Eine Kleinigkeit fehlt noch: Graphiken sollen nicht nur gezeichnet, sondern auch gelöscht werden können. Wir benötigen eine Art "Radiergummi", mit der sich beliebige Teile einer Zeichnung wieder löschen lassen.

Unterscheiden wir daher zwischen einem "Mal-Modus" und einem "Lösch-Modus". Weiterhin benötigen wir einen "Bewegungs-Modus", in dem wir den "abgehobenen" Pinsel über das Zeichenbrett bewegen können, ohne daß gezeichnet oder gelöscht wird. Zwischen den drei Modi soll mit der Taste RETURN umgeschaltet werden. Jede Betätigung von RETURN schaltet den jeweils nächsten Modus ein.

Die Anforderungen stehen nun in etwa fest. Das Programm wird wie folgt aufgebaut: Den ersten Programmteil bildet das Einschalten der Graphik. Der nächste Programmteil fragt - wie, wissen wir noch nicht - die Tastatur ab und kontrolliert, welche Tasten betätigt werden. Je nach Taste wird zu einem der Unterprogramme verzweigt:

- Pinsel bewegen
- Zeichen-Modus umschalten
- Linie ziehen
- Rechteck malen

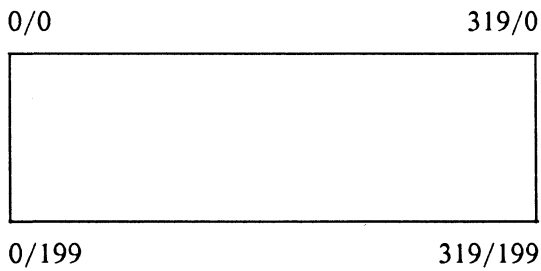
Nach Rückkehr aus dem betreffenden Unterprogramm wird erneut auf ein über die Tastatur gegebenes Kommando gewartet. Das Programm läuft in einer Schleife (Befehlsabfrage, Befehlsausführung, Befehlsabfrage usw.), die erst dann verlassen wird, wenn das Kommando "Programm beenden" gewählt wird.

3.2.1.2 Koordinaten, Farben und Farbquellen

Bevor ich Ihnen die ersten Graphikbefehle nenne, muß ich mehrere grundlegende Begriffe erläutern, die immer wieder auftauchen werden.

Koordinatensystem

Im Graphik-Modus besteht der Bildschirm nicht mehr aus Zeichen, sondern aus einzelnen Punkten, die getrennt gesetzt oder gelöscht werden können. Man spricht von einem "Koordinatensystem" mit der Auflösung von 320 Punkten in "X-Richtung" und 200 Punkten in "Y-Richtung" (Voraussetzung: Hi-Res-Modus). Der "Ursprung" dieses Koordinatensystems, der Punkt 0/0 (X=0 und Y=0) befindet sich in der linken oberen Bildschirmecke.



Jeder Punkt in diesem Koordinatensystem kann gesetzt oder gelöscht werden. Wichtig ist in diesem Zusammenhang, daß das Löschen eines Punktes nur bedeutet, daß dieser in der Farbe des Bildschirm-Hintergrundes gesetzt wird. Wenn die Farbe des Punktes und des Hintergrundes identisch sind, hebt sich der Punkt nicht vom Hintergrund ab, er ist für den Beobachter nicht existent. Ein Punkt kann daher "gelöscht" werden, indem er erneut gesetzt wird, jedoch in der Hintergrund-Farbe!

Mit dem Befehl COLOR kann die Farbe gewählt werden, in der ein gesetzter Punkt dargestellt wird, oder auch die Farbe des Bildschirm-Hintergrundes und des Bildschirm-Rahmens eingestellt werden. COLOR besitzt folgendes Format:

```
COLOR (BEREICH),(FARBE),(HELLIGKEIT)
```

Mit BEREICH kann angegeben werden, ob die Farbe eines Punktes, des Bildschirm-Hintergrundes oder des Bildschirm-Rahmens verändert werden soll.

Bereichsangabe im COLOR-Befehl

0 = Bildschirm-Hintergrund
1 = Buchstaben- bzw. Punktfarbe Nr.1
2 = Buchstaben- bzw. Punktfarbe Nr.2
3 = Buchstaben- bzw. Punktfarbe Nr.3
4 = Bildschirm-Rand

Im Hi-Res-Modus wird uns nur die Punktfarbe Nummer eins interessieren. Die Farben Nummer zwei und drei werden vorwiegend im Multicolor-Modus angewendet.

Farbangabe im COLOR-Befehl

1 = Schwarz	9 = Orange
2 = Weiß	10 = Braun
3 = Rot	11 = Gelb-Grün
4 = Zyan	12 = Rosa
5 = Purpur	13 = Blau-Grün
6 = Grün	14 = Hellblau
7 = Blau	15 = Dunkelblau
8 = Gelb	16 = Hellgrün

Helligkeitsangabe im COLOR-Befehl

0 = äußerst dunkel
1 = sehr dunkel
2 = dunkel
...
...
...
7 = äußerst hell

Im Zusammenhang mit dem Bereich ist der Begriff "Farbquelle" wichtig. Mit dem COLOR-Befehl können für verschiedene Bereiche (Punkte, Hintergrund, Rahmen) Farben eingestellt werden. Jeder dieser Bereiche ist eine Farbquelle. Wird ein Punkt gesetzt, muß die Farbquelle angegeben werden.

Ein Beispiel: Als Punktfarbe wurde ein helles Gelb angegeben (COLOR 1,8,7). Wird beim Setzen eines Punktes als Farbquelle die Punktfarbe eins angegeben, wird der Punkt in hellem Gelb dargestellt. Wird hingegen beim Setzen als Farbquelle der Bildschirm-Hintergrund angegeben, wird der Punkt in der Farbe des Hintergrundes gemalt, das heißt er ist unsichtbar, da er sich nicht vom Hintergrund abhebt. Sie sehen, es besteht ein enger Zusammenhang zwischen Bereich und Farbquelle. Jeder Bereich kann als Farbquelle beim Setzen eines Punktes dienen, der in der Farbe des angegebenen Bereichs gemalt wird.

Der COLOR-Befehl kann im Text- und im Graphik-Modus verwendet werden. Zum Beispiel können Sie mit dem Befehl COLOR 1,5,4 im Text-Modus den Zeichen, im Graphik-Modus den Punkten, die Farbe Purpur (in mittlerer Helligkeit dargestellt) geben. Der COLOR-Befehl kann alternativ zu den Farbtasten verwendet werden, ist jedoch weit flexibler als diese (Einstellung der Helligkeit, der Farben von Hintergrund und Rahmen).

3.2.1.3 Graphik ein-/ausschalten

Die hochauflösende Graphik wird mit dem Befehl GRAPHIC eingeschaltet. Die genaue Syntax dieses Befehls lautet:

```
GRAPHIC (MODUS),(LÖSCHEN)
```

MODUS:
- 0 = Text-Modus
- 1 = Hochauflösende Graphik
- 2 = Hochauflösende Graphik + Text
- 3 = Multicolor-Graphik
- 4 = Multicolor-Graphik + Text

LÖSCHEN:
- 0 = Bildschirm wird nicht gelöscht
- 1 = Bildschirm wird gelöscht

Beispiele:

GRAPHIC 0	schaltet den Text-Modus ein, ohne den Bildschirm zu löschen.
GRAPHIC 0,1	schaltet den Text-Modus ein und löscht den Bildschirm.
GRAPHIC 1,1	schaltet die Hi-Res-Graphik ein und löscht den Bildschirm.
GRAPHIC 2,1	schaltet die Hi-Res-Graphik ein, löscht den Bildschirm und reserviert die unteren fünf Bildschirmzeilen für die Textdarstellung.

Die Modi drei und vier reservieren auf dem Graphikbildschirm die untersten fünf Zeilen für Texte. Mit diesen Modi können Graphik- und Text-Modus gemischt werden.

Vorteile sollen diese Modi vor allem bei der Programmentwicklung bieten. Wenn die Graphik eingeschaltet ist und Sie LIST eingeben, erfolgt keinerlei Reaktion. Der Grund: Texte wie zum Beispiel Programmlistings werden auf dem "Graphikbildschirm" nicht dargestellt (Ausnahme: mit dem noch zu besprechenden Befehl CHAR). Am besten stellen Sie sich den Text- und den Graphikbildschirm als zwei voneinander unabhängige Bildschirme vor. LIST und PRINT geben Zeichen auf dem Textbildschirm aus, die auf dem Graphikbildschirm nicht sichtbar werden.

Im gemischten Text-/Graphik-Modus können Listings in den untersten Bildschirmzeilen dargestellt werden. Um dies auszuprobieren, bewegen Sie den Cursor bitte in die unterste Bildschirmzeile und geben Sie im Direkt-Modus den Befehl GRAPHIC 2,1 ein.

Mit diesem Befehl wurde die Hi-Res-Graphik eingeschaltet, der Graphikbildschirm gelöscht, und die fünf letzten Bildschirmzeilen für den normalen Text-Modus reserviert.

Nach Eingabe des Befehls wird der größte Teil des Bildschirms gelöscht werden. In den unteren Zeilen wird wie gewohnt Text dargestellt (GRAPHIC 2,1; READY.). In der untersten Zeile blinkt der Cursor.

Bewegen Sie den Cursor bitte mehrmals nach oben. Wenn der Cursor aus dem Textbildschirm heraus- und in den Graphikbildschirm bewegt wird, ist er nicht mehr sichtbar, da auf dem Graphikbildschirm kein Textzeichen dargestellt werden kann.

Mit der Taste CURSOR UNTEN können Sie den Cursor wieder in den für den Text-Modus reservierten Bildschirmteil bewegen und sichtbar machen. Wenn Sie den Befehl GRAPHIC 0 eingeben, wird der Graphik-Modus aus- und der Text-Modus wieder eingeschaltet.

Wie erläutert, soll der gemischte Text-/Graphik-Modus Vorteile bei der Programmentwicklung bieten, um bei eingeschalteter Graphik das Programm listen und korrigieren zu können. In der Praxis sind fünf Bildschirmzeilen jedoch viel zu wenig, um eine Übersicht über ein Programm zu gewinnen.

Im folgenden arbeiten wir daher im reinen Graphik-Modus, wobei der Bildschirm beim Einschalten der Graphik gelöscht werden soll (GRAPHIC 1,1). Vor der Programmerstellung muß noch ein Problem gelöst werden: Wie können Sie das zu entwickelnde Programm ändern und erweitern, wenn es nicht auf dem Bildschirm sichtbar ist?

Eine Möglichkeit besteht darin, vor jeder Änderung den Befehl GRAPHIC 0 einzugeben, um den Text-Modus einzuschalten. Da dieser Befehl jedoch mangels Möglichkeiten zur Textdarstellung "unsichtbar" eingetippt werden muß (mit entsprechend vielen Tippfehlehlern), ist diese Lösung sicherlich nicht sehr elegant.

Im folgenden soll eine Betätigung der Funktionstaste F4 ausreichen, um den Text-Modus einzuschalten. Geben Sie bitte folgenden Befehl ein:

```
KEY 4,"GRAPHIC Ø"+CHR$(13)
```

Dieser Befehl "belegt" die Taste F4 mit der Zeichenkette "GRAPHIC 0" und dem Code der Taste RETURN zur Bestätigung der Eingabe. Da diese neue Belegung von Ihrem C16, C166 oder Plus/4 nach dem Ausschalten "vergessen" wird, geben Sie diesen Befehl bitte nach dem nächsten Einschalten erneut ein.

Wenn Sie die folgenden Graphikprogramme aufgrund von Tippfehlern korrigieren müssen und gerade der Graphik-Modus eingeschaltet ist, genügt eine Betätigung von F4, um in den Text-Modus zurückzukehren, in dem Sie das Programm problemlos überprüfen und ändern können.

Da wir im Hi-Res-Modus arbeiten wollen und der Bildschirm beim Einschalten der Graphik, gelöscht werden soll, lautet der erste Befehl in unserem Malprogramm:

```
100 GRAPHIC 1:REM HI-RES-GRAPHIK EIN
```

Geben Sie diese Programmzeile bitte noch nicht ein, da sie für sich allein sinnlos ist. Eine Eingabe ist erst sinnvoll, wenn das Programm weiterentwickelt ist.

Übrigens: Wie bereits in diesem Buch erwähnt wurde, gehen beim Einschalten der Graphik zehn Kilobyte an freiem Speicher für Daten und Programme verloren. Nach dem Ausschalten der Graphik mit GRAPHIC 0 wird dieser Speicherplatz nicht zurückgewonnen, er bleibt reserviert!

Mit dem Befehl GRAPHIC CLR wird diese Reservierung rückgängig gemacht uns Sie können wieder den vollen Speicherplatz Ihres C16, C116 oder Plus/4 zur Ablage von Daten und Programmen benutzen.

Nach dem Einschalten der Hi-Res-Graphik ist es sinnvoll, die Farben für Bildschirm-Hintergrund, Bildschirm-Rahmen und die Punktfarbe voreinzustellen. Die nächsten drei Befehle unseres Malprogramms lauten:

```
110 COLOR 1,2,6:REM PUNKTE: HELLES WEISS
120 COLOR 0,1,0:REM HINTERGRUND: DUNKLES SCHWARZ
130 COLOR 4,1,0:REM RAHMEN: DUNKLES SCHWARZ
```

3.2.1.4 Format der Graphikbefehle

Wie Sie bereits am Beispiel des COLOR-Befehls sahen, sind die Graphikbefehle leider sehr kompliziert, da eine Vielzahl sogenannter "Parameter" angegeben werden müssen. Der Großteil dieser Angaben ist immer gleich und soll daher am Beispiel des DRAW-Befehls besprochen werden.

Mit DRAW können einzelne Punkte gesetzt, gelöscht oder auch ganze Linien gezogen beziehungsweise gelöscht werden. Sie sehen, DRAW besitzt eine Vielzahl von Möglichkeiten. Dementsprechend komplex ist leider auch die Verwendung dieses Befehls:

```
DRAW (FARBQUELLE),(X1),(Y1) TO (X2),(Y2)
```

1. FARBQUELLE: 0 bis 3. Die Zahl entspricht einem der Bereiche 0 bis 3 (Hintergrund, Punktfarben 1 bis 3). Gezeichnet wird in jener Farbe, die der angegebene Bereich besitzt. Im Hi-Res-Modus werden üblicherweise nur die Farbquellen 0 und 1 verwendet (0 = Hintergrund-Farbe; 1 = Punktfarbe 1). Die Angabe von 0 löscht einen Punkt, da der Punkt in der Hintergrundfarbe gezeichnet wird und sich von diesem nicht abhebt. Sichtbar werden gesetzte Punkte unter Angabe der Bereichsfarbe 1 (Punktfarbe 1). Im Multicolor-Modus kann zwischen drei verschiedenen Punktfarben gewählt werden (Punktfarben 2 bis 3).

2. X1: Erste X-Koordinate (0-299)

3. Y1: Erste Y-Koordinate (0-199)

4. X2: Zweite X-Koordinate (0-299)

5. Y2: Zweite Y-Koordinate (0-199)

Soll ein Punkt (!) gesetzt oder gelöscht werden, entfällt die Angabe der zweiten X- und Y-Koordinate. Der Befehl besitzt in diesem Fall das abgekürzte Format:

```
DRAW (FARBQUELLE),(X1),(Y1)
```

Die Angabe der Parameter X2 und Y2 ist nur beim Zeichnen oder Löschen von Linien nötig, da in diesem Fall zwei (!) Punkte (die beiden Endpunkte der Linie) angegeben werden müssen.

Beispiele:

1. DRAW 1,200,100 setzt einen Punkt in die Bildschirmmitte (X=200; Y=100), der in der im COLOR-Befehl angegebenen ersten Punktfarbe gezeichnet wird.

2. DRAW 0,200,100 löscht einen Punkt, der sich an der Position 200/100 befindet, indem er in der Hintergrundfarbe gezeichnet wird.

3. DRAW 1,0,0 TO 319,199 zeichnet eine quer über den Bildschirm verlaufende Linie in der Punktfarbe 1.

4. DRAW 0,0,0 TO 319,199 löscht die unter 3. beschriebene Linie wieder.

Demoprogramm

Das folgende Demoprogramm verwendet alle bisher verwendeten Befehle, um Punkte und Linien in hochauflösender Graphik zu zeichnen.

```
100 GRAPHIC 1,1:REM HI-RES EIN UND LOESCHEN
110 COLOR 0,1,4:REM UNTERGRUND: SCHWARZ MITTLERER HELLIGKEIT
120 COLOR 1,8,6:REM PUNKTFARBE 1: HELLES GELB
130 :
140 DRAW 1,200,150:REM PUNKT ZEICHNEN
150 DRAW 1,0,0 TO 319,199:REM LINIE ZEICHNEN
160 FOR I=1 TO 1000:NEXT I:REM WARTESCHLEIFE
170 DRAW 0,200,150:REM PUNKT LOESCHEN
180 DRAW 0,0,0 TO 319,199:REM LINIE LOESCHEN
190 :
200 GRAPHIC 0:REM GRAPHIC AUS, TEXT-MODUS EIN
```

Übrigens: Der DRAW-Befehl ist vielseitiger, als Sie denken. Mit DRAW lassen sich nicht nur Rechtecke, sondern Vielecke zeichnen. Das allgemeine Format von DRAW lautet:

```
DRAW (FARBQUELLE),(X1),(Y1) TO (X2),(Y2) TO (X2),(Y2) TO ...
```

Im DRAW-Befehl können beliebig viele Punkte angegeben werden, die miteinander verbunden sollen. Ein Dreieck kann mit dem Befehl gezeichnet werden:

```
DRAW 1,10,10 TO 200,50 TO 150,150
```

Ebenso wie beim DRAW-Befehl muß auch bei den Befehlen zum Zeichnen von Rechtecken oder Kreisen zuerst die Farbquelle angegeben werden, wobei im Hi-Res-Modus meist nur die Werte 0 (Löschen) und 1 (Setzen in Punktfarbe 1) verwendet werden.

Welche weiteren Parameter angegeben werden, hängt vom jeweiligen Befehl ab. Je komplexer eine zu zeichnende Figur ist, desto mehr Parameter sind zur Definition der Figur nötig.

Das allgemeine Prinzip der Graphik-Befehle ist jedoch immer gleich. Nach der Angabe der unbedingt benötigten Parameter (DRAW: Farbquelle und Koordinaten eines Punktes) können (!) zusätzliche Parameter angegeben werden, es ist jedoch nicht unbedingt notwendig.

Ein Beispiel: Der BOX-Befehl zeichnet ein Rechteck. Außer der Farbquelle wird die Angabe der Koordinaten der linken oberen und der rechten unteren Ecke benötigt. Diese Parameter müssen (!) angegeben werden. Zusätzlich kann (!) ein Drehwinkel (in Grad) angegeben werden, um das Rechteck auf dem Bildschirm gedreht darzustellen. Ebenfalls "optional" (optional = mögliche, jedoch nicht zwingend notwendige Angabe) kann anschließend eine 0 oder eine 1 angegeben werden (0 = leeres Rechteck; 1 = in der gewählten Farbe ausgemaltes Rechteck).

```
BOX (FARBQUELLE),(X1),(Y1),(X2),(Y2),(DREHWINKEL),(FUELLEN)
```

Um optionale von notwendigen Parametern zu unterscheiden, werde ich im folgenden optionale Angaben in eckige Klammern setzen.

```
DRAW [(FARBQUELLE)],(X1),(Y1) [TO (X2),(Y2)]
BOX [(FARBQU.)],(X1),(Y1),(X2),(Y2)[,(DREHWINKEL),(FUELLEN)]
```

Optionale Parameter

Wie Sie an der Syntax der Befehle DRAW und BOX sehen, ist die Angabe der Farbquelle ebenfalls optional. Wenn diese Angabe entfällt, nimmt der BASIC-Interpreter für die Farbquelle den sogenannten "Standardwert" 1 zum Zeichnen, das heißt die Punktfarbe 1.

Soll ein optionaler Parameter entfallen, muß der Parameter durch ein Komma ersetzt werden, wenn ansonsten die Reihenfolge der Angaben durcheinandergerät.

Richtig: `RAW ,200,100`
Falsch: `DRAW 200,100`

Anhand des Kommas erkennt der Interpreter, daß für die Farbquelle der Standardwert 1 benutzt werden soll. Das Komma ist zur Angabe der Parameter-Reihenfolge notwendig. Im Befehl DRAW 200,100 nimmt der Interpreter an, daß der erste Parameter 200 die Farbquelle angibt und erkennt nicht, daß diese Angabe durch den Standardwert ersetzt werden soll.

Ein weiteres Beispiel: Mit dem BOX-Befehl soll ein gefülltes Rechteck gezeichnet werden. Das Rechteck soll nicht gedreht werden, die Angabe eines Drehwinkels entfällt daher.

```
Richtig:   BOX ,10,10,200,100,,1
Falsch:    BOX ,10,10,200,100,1
```

In der falschen Version dieses Befehls entfällt die Angabe des Drehwinkels völlig. Unmittelbar nach den Koordinaten der rechten unteren Ecke wird eine 1 angegeben, um das Rechteck zu füllen. Der Interpreter mißversteht diese Angabe jedoch. Die 1 wird als Drehwinkel (Ein Grad) interpretiert.

In der korrekten Version wird der Drehwinkel durch ein Komma ersetzt. Der BASIC-Interpreter versteht das Komma als Aufforderung, den Drehwinkel durch den Standardwert (null Grad) zu ersetzen und interpretiert die folgende Angabe korrekt als letzten Parameter (Rechteck füllen: Ja (1) oder nein (0)).

Da Sie nun den BOX-Befehl kennen, sollten Sie auch in der Lage sein, das zu Beginn des Graphikteils vorgestellte Demoprogramm zu verstehen.

```
100 GRAPHIC 1,1
110 FOR I=10 TO 130 STEP 3
120 : BOX 1,I*2,I,I*2+50,I+50
130 NEXT
140 GETKEY A$:GRAPHIC 0
```

In Zeile 100 wird die Hi-Res-Graphik eingeschaltet und der Bildschirm gelöscht. Anschließend werden in einer Schleife 40 Rechtecke gemalt. Länge und Breite der Rechtecke betragen je

50 Punkte in X- und in Y-Richtung. Die Schleifenvariable I wird verwendet, um die Positionen der beiden Eckpunkte zu bestimmen (linke obere und rechte untere Ecke). Wenn alle Rechtecke gezeichnet wurden, wird in Zeile 140 auf das Drücken einer beliebigen Taste gewartet (GETKEY lernen wir in Kürze kennen), bevor die Graphik aus- und der Text-Modus eingeschaltet wird.

3.2.1.5 Cursorbewegung im Graphik-Modus

Nachdem Sie nun die prinzipielle Arbeitsweise der Graphik-Befehle kennen, wird es Zeit, daß wir uns dem eigentlichen Malprogramm zuwenden.

Zuvor sollte jedoch ein weiterer Punkt theoretisch behandelt werden: Die Cursorbewegung im Graphik-Modus. Im Gegensatz zum Text-Modus wird im Graphik-Modus leider nicht mehr automatisch ein 8*8 Punkte großer Cursor dargestellt, um die Eingabeposition anzugeben. Im Graphik-Modus ist zwar ein "Graphik-Cursor" vorhanden, der einen einzigen Punkt groß ist und an beliebige Positionen gesetzt werden kann, dieser Graphik-Cursor ist jedoch leider unsichtbar, er wird nicht auf dem Bildschirm dargestellt. Um ihn sichtbar zu machen, müssen wir an der aktuellen Position unseres Graphik-Cursors mit dem DRAW-Befehl einen Punkt setzen.

Cursorbewegungen sollen diesen Punkt nach rechts, links, oben oder unten verschieben, was bedeutet, daß der Punkt an der alten Position gelöscht wird, bevor an der neuen Position erneut ein Punkt gesetzt wird. In einem Fall darf der alte Punkt jedoch keinesfalls gelöscht werden, und zwar dann, wenn er sich unabhängig vom Cursor auf dem Bildschirm befindet. Ein Beispiel dafür ist eine Linie, die wir überschreiten. Der Punkt, an dem wir die Linie überqueren, soll auch nach der Überquerung weiterhin vorhanden sein. Wir benötigen daher folgenden Programmablauf:

1. Nach Betätigung einer der Cursortasten wird der Graphik-Cursor auf die neue Position gesetzt.

2. Wir merken uns die Farbquelle, die an der neuen Position gültig ist (kann mit der Funktion RDOT erfragt werden), das heißt die Farbe, die der Punkt an der neuen Cursorposition besitzt.

3. Wir setzen einen Punkt an der neuen Cursorposition, um den Graphik-Cursor darzustellen. Als Farbquelle wird die Punktfarbe angegeben, damit sich der Punkt vom Hintergrund abhebt.

4. Wird der Cursor erneut bewegt, wird an der momentanen Position ein Punkt gesetzt, wobei jene Farbquelle angegeben wird, die wir uns merkten (0 oder 1). Resultat: Der Punkt erhält exakt jene Farbe, die er besaß, bevor der Cursor zu dieser Position bewegt wurde. Befand sich zuvor an dieser Position ein sichtbarer (in der Punktfarbe gemalter) Punkt, wird er erneut in der Punktfarbe gesetzt werden. War zuvor kein Punkt sichtbar (Farbquelle 0 (Hintergrund)), wird der Punkt wie zuvor in der Hintergrund-Farbe gesetzt und ist damit ebensowenig sichtbar wie zuvor.

Ich gebe zu, dieser Ablauf ist sehr komplex, jedoch leider unumgänglich, um einen sich bewegenden Cursor auf dem Graphik-Bildschirm zu simulieren. Der Ablauf wird weiter verkompliziert, da im "Mal-Modus" immer (!) ein Punkt gesetzt und im "Lösch-Modus" ebenfalls immer (!) ein Punkt gelöscht werden soll, wenn der Cursor bewegt wird. Denken Sie bitte bei der Unterscheidung der verschiedenen Modi immer daran, daß 320*200 Punkte tatsächlich existieren! Ob die Punkte sichtbar sind oder nicht, hängt von ihrer Farbe ob (Punktfarbe = sichtbar; Hintergrund-Farbe = unsichtbar). "Löschen eines Punktes" ist eigentlich ein falscher Ausdruck. Der Punkt bleibt weiterhin vorhanden, ist jedoch nicht mehr sichtbar, da er in der Hintergrund-Farbe dargestellt wird.

3.2.2 Erstellung des Malprogramms

Alles nötige Grundwissen zur Erstellung des Malprogramms besitzen Sie nun. Das Programm wird im folgenden Schritt für Schritt erstellt.

Im Anschluß an das Programm folgen zwei Kapitel, die die Unterschiede zwischen dem Hi-Res-Modus und dem Multicolor-Modus, und das Speichern beziehungsweise Laden einer Graphik erläutern.

Das Wissen, das in diesen Kapitel verwendet wird, können Sie unmittelbar am Malprogramm umsetzen, zum Beispiel, um es so zu ändern, daß es im Multicolor-Modus arbeitet.

3.2.2.1 Programminitialisierung

Unter "Programminitialisierung" versteht man die Vorbereitungen, die nach dem Starten eines Programms getroffen werden, um den korrekten Ablauf zu gewährleisten. In unserem Fall ist folgende Vorbereitung nötig.

1. Hi-Res-Modus einschalten und Bildschirm löschen
2. Gewünschte Farben einstellen
3. Ausgangs-Modus festlegen (Cursor bewegen)
4. Cursor auf Bildschirmmitte positionieren

Geben Sie den folgenden Programmteil bitte noch nicht ein!

```
100 GRAPHIC 1,0:REM HI-RES EIN, LOESCHEN
110 COLOR 1,2,6:REM PUNKTE: HELLES WEISS
120 COLOR 0,1,0:REM HINTERGRUND: DUNKLES SCHWARZ
130 COLOR 4,1,0:REM RAHMEN: DUNKLES SCHWARZ
140 :
150 MODE=2:REM BEWEGEN
160 X=160:Y=100:REM AUSGANGSKOORDINATEN FUER CURSOR
170 LOCATE X,Y:REM GRAPHIK-CURSOR POSITIONIEREN
180 FQ=RDOT(2):REM FARQUELLE MERKEN
```

In Zeile 150 wird der Arbeits-Modus festgelegt. Wir verwenden die Variable "MODE" und legen fest:

MODE=0: Lösch-Modus
MODE=1: Mal-Modus
MODE=2: Bewegungsmodus

In Zeile 160 werden die Ausgangskoordinaten des Graphik-Cursors den Variablen X und Y zugewiesen. Anschließend wird der Cursor mit dem Befehl LOCATE auf diese Position gesetzt (LOCATE (X-KOORDINATE),(Y-KOORDINATE)). In diesem Moment ist er leider noch unsichtbar, da an den betreffenden Koordinaten kein Punkt in der Punktfarbe gesetzt wurde (nach dem Löschen des Graphik-Bildschirms werden alle Punkte in der Hintergrund-Farbe dargestellt).

Um nach einer Cursorbewegung an der alten Position den vorhergehenden Zustand wiederherstellen zu können (Punkt gesetzt (Hintergrund-Farbe) oder gelöscht(Punktfarbe)), merken wir uns den Zustand an der Ausgangsposition 160/100 in der Variablen FQ (FQ=Farbquelle). Da der Punkt an der aktuellen Cursorposition nach dem Programmstart die Farbe des Hintergrundes besitzt, weisen wir der Variablen FQ den Wert 0 zu (Farbquelle = Hintergrund).

3.2.2.2 Bewegen des Zeichenstiftes

Nach dem Einschalten der Graphik folgt ein Programmteil, der die Cursortasten abfragt und je nach gedrückter Taste den Graphik-Cursor bewegt. Wir benötigen nun einen Befehl, mit dem wir überprüfen können, ob zum Beispiel die Taste CURSOR RECHTS gedrückt wurde.

Dieser Befehl heißt GETKEY (STRINGVARIABLE). GETKEY ähnelt ein wenig dem INPUT-Befehl. INPUT wartet auf eine Eingabe des Benutzers und die Bestätigung der Eingabe mit der RETURN-Taste. Die Eingabe wird anschließend der angegebenen Variable zugewiesen.

GETKEY wartet auf den Druck einer (!) Taste und weist das eingegebene Zeichen der angegebenen Stringvariablen (!) zu. Die Eingabe ist beendet, sobald eine beliebige Taste betätigt wurde, ein Abschluß mit RETURN ist überflüssig. Mit einem kleinen Programm kann die Funktionsweise von GETKEY erprobt werden.

```
100 GETKEY A$:REM AUF TASTE WARTEN
110 IF A$="X" THEN PRINT "X":ELSE PRINT "ANDERE TASTE"
```

Dieses Programm wartet auf die Betätigung einer Taste. Das eingegebene Zeichen wird der Variablen A$ zugewiesen und anschließend mit dem Zeichen "X" verglichen. Wurde "X" eingegeben, ist die Bedingung IF A$="X" erfüllt, ansonsten wurde eine andere Taste betätigt.

Die Cursortasten können wir mit dem Befehl GETKEY abfragen, indem wir das eingegebene Zeichen mit den Codes dieser vier Tasten vergleichen (17, 29, 145, 157).

```
200 REM TASTENABFRAGE UND VERZWEIGUNG
210 GETKEY A$:REM AUF TASTE WARTEN
220 IF MODE=2 THEN DRAW FQ,X,Y:ELSE DRAW MODE,X,Y
230 IF A$=CHR$(17)  THEN Y=Y+1:REM CURSOR UNTEN
240 IF A$=CHR$(29)  THEN X=X+1:REM CURSOR RECHTS
250 IF A$=CHR$(145) THEN Y=Y-1:REM CURSOR OBEN
260 IF A$=CHR$(157) THEN X=X-1:REM CURSOR LINKS
270 LOCATE X,Y:REM CURSOR POSITIONIEREN
280 FQ=RDOT(2):REM FARBQUELLE MERKEN
290 DRAW 1,X,Y:REM CURSOR-PUNKT SETZEN
300 GOTO 210:REM ZUR TASTENABFRAGE
```

In Zeile 210 wartet GETKEY auf eine Taste. Das eingegebene Zeichen wird der Variablen A$ zugewiesen.

Zeile 220 behandelt den Punkt an der momentanen Cursor-Position. Im Bewegungs-Modus (MODE gleich 2) soll dieser Punkt im gleichen Zustand verlassen werden, in dem er vorgefunden wurde, bevor der Cursor an diese Position gesetzt wurde. Im

Bewegungs-Modus wird daher ein Punkt in der zuvor gültigen Bereichsfarbe gesetzt, die durch FQ gekennzeichnet ist.

Im Lösch-Modus wird der Punkt in der Hintergrund-Farbe gesetzt, im Mal-Modus in der Zeichenfarbe. Wie erläutert, soll MODE in einem späteren Programmteil den Wert 0 erhalten, wenn der Lösch-Modus eingeschaltet wird, und den Wert 1, wenn der Mal-Modus eingeschaltet wird. Der Befehl DRAW MODE,X,Y löscht daher im Lösch-Modus den Punkt (er entspricht dem Befehl DRAW 0,X,Y und setzt daher den Punkt in der Hintergrund-Farbe) und setzt ihn im Mal-Modus (Zeichnen in der eingestellten Punktfarbe (DRAW 1,X,Y)).

Der Befehl CHR$ wandelt einen Code in das zugehörige Zeichen um. X$=CHR$(64) weist daher der Variablen X$ das Zeichen zu, das dem Code 64 entspricht, also das Zeichen "A". Entsprechend vergleicht der Befehl IF A$=CHR$(17) in Zeile 230 das eingegebene Zeichen mit dem Zeichen, das dem Code 17 zugeordnet ist, dem Zeichen für CURSOR NACH UNTEN (siehe Tabelle).

Wenn der Benutzer diese Taste betätigt, wird X - die X-Koordinate unseres Cursors - um eins erhöht. Entsprechendes gilt für die folgenden Zeilen, in denen A$ mit den restlichen Cursortasten verglichen und entsprechende Veränderungen der X- und Y-Koordinaten ausgeführt werden.

Nachdem die Koordinaten je nach gedrückter Cursortaste geändert wurden, wird der Cursor mit dem Befehl LOCATE X,Y an die neue Position gesetzt, ist jedoch noch unsichtbar.

Bevor an der neuen Position ein Punkt gezeichnet wird (Zeile 290: DRAW 1,X,Y), wird Zeile 280 ausgeführt, die einen bisher unbekannten Befehl enthält: RDOT(...).

RDOT besitzt drei Funktionen:

1. RDOT(1) liefert die aktuelle X-Koordinate des Graphik-Cursors.
2. RDOT(2) liefert die Y-Koordinate.
3. RDOT(3) gibt an, welche Farbe der Punkt an der aktuellen Cursorposition besitzt, das heißt welche Farbquelle beim Setzen verwendet wurde (0=Hintergrundfarbe oder 1=Punktfarbe).

Durch die Zuweisung FQ=RDOT(2) wird die Farbquelle des aktuellen Punktes der Variablen FQ zugewiesen. Im Bewegungs-Modus wird bei einer weiteren Cursorbewegung ein Punkt mit dieser Farbquelle gesetzt (siehe Zeile 220), um den alten Zustand wiederherzustellen.

Nachdem nun mit DRAW 1,X,Y ein sichtbarer (1=Punktfarbe) Punkt gesetzt wird, um den Cursor darzustellen, erfolgt ein Sprung zu Zeile 210, wo erneut auf die Betätigung einer Cursortaste gewartet wird.

Dieses Programm ist lauffähig. Sie können es abtippen und mit RUN starten. Betätigen Sie anschließend die Cursortasten. Der durch einen Punkt dargestellte Graphik-Cursor wird sich in der gewünschten Richtung bewegen. Um das Programm abzubrechen, drücken Sie bitte die STOP-Taste und geben Sie ein: GRAPHIC 0 oder betätigen Sie F4, wenn Sie diese Funktionstaste wie erläutert belegt haben.

3.2.2.3 Umschalten der Zeichen-Modi

In diesem Programm ist der zu Beginn mit dem Wert 2 (Bewegen) initialisierte Modus fest eingestellt und kann nicht geändert werden. Wie erläutert, soll es jedoch möglich sein, mit der RETURN-Taste zwischen den drei Modi "Bewegen", "Löschen" und "Malen" hin- und herzuschalten.

Unsere Tastenabfrage muß daher um die Abfrage der RETURN-Taste (Code 13) erweitert werden:

```
211 IF A$=CHR$(13) THEN GOSUB 400:GOTO 210:REM RETURN-TASTE?
```

Wenn RETURN betätigt wurde, werden im THEN-Zweig zwei Befehl ausgeführt: Der Aufruf eines ab Zeile 400 beginnenden Unterprogramms und - nach Rückkehr aus diesem Unterprogramm - der Sprung zu Zeile 210, wo erneut auf eine Taste gewartet wird.

```
400 REM MODUS UMSCHALTEN
410 MODE=MODE+1:REM NAECHSTER MODUS
420 IF MODE>2 THEN MODE=0
430 RETURN
```

Erweitern Sie das Programm bitte um diese Zeilen und starten Sie es erneut. Jede Betätigung einer Cursortaste bewegt immer noch den Cursor. Drücken Sie nun bitte RETURN. Der Wert von MODE wird um eins erhöht und beträgt nun 3. Da dieser Modus nicht existiert (nur 0 bis 2) wird in Zeile 420 der Variablen MODE der Wert 0 zugewiesen, wenn der Maximalwert zwei überschritten wurde.

Der Modus 0 bedeutet für unser Programm "Löschen". In Zeile 220 wird nun bei jeder Cursorbewegung der Befehl DRAW 0,X,Y ausgeführt, da MODE den Wert 0 besitzt. Dieser Befehl setzt Punkte in der Farbe des Hintergrundes, das heißt er löscht Punkte, über die der Cursor bewegt wird. Da jedoch noch keinerlei gesetzte Punkte existieren, werden Sie keine Veränderung gegenüber dem Bewegungs-Modus bemerken.

Drücken Sie nun erneut RETURN. MODE wird wieder erhöht und erhält dadurch den Wert 1. Das Programm befindet sich nun im Mal-Modus und der Befehl DRAW MODE,X,Y in Zeile 220 entspricht dem Befehl DRAW 1,X,Y, da MODE gleich 1 ist. Bei der Bewegung des Cursors werden nun Punkte in der eingestellten Punktfarbe 1 gesetzt und dadurch sichtbar. Der Cursor wirkt nun als Pinsel, mit dem bei jeder Bewegung Punkte gesetzt werden.

Malen Sie bitte ein beliebige Zeichnung und betätigen Sie anschließend erneut RETURN. Sie sind wieder im Bewegungs-Modus und können Ihre Zeichnung mit dem Cursor überqueren, ohne Spuren zu hinterlassen.

Drücken Sie bitte noch einmal RETURN, um den Lösch-Modus einzuschalten. Jede Bewegung des Cursors über die Zeichnung hinweg radiert diese an der betreffenden Position aus.

Eine Kleinigkeit sollte noch geändert werden: Der Benutzer sollte darüber informiert werden, in welchem Modus er sich befindet, um Fehlbedienungen zu vermeiden, wie zum Beispiel das versehentliche Ausradieren einer Graphik, die man eigentlich nur im Bewegungs-Modus überqueren wollte.

Schön wäre es, wenn der jeweilige Modus mit einem PRINT-Befehl ausgegeben werden könnte. PRINT ist jedoch ein nur im Text-Modus anwendbarer Befehl. Der C16, C116 bzw. Plus/4 besitzt jedoch einen vollwertigen Ersatz, den Befehl CHAR, um beliebige Texte auf dem Graphik- und auf dem Text-Bildschirm (!) auszugeben.

```
CHAR [(FARBZONE)],X,Y,(STRING),[(INVERS?)]
```

1. Farbzone: Sinnvoll ist meist nur die Angabe der Farbzone 1, der Punkt- beziehungsweise Zeichenfarbe. Da 1 der Standardparameter für die Farbzone ist, kann diese Angabe durch ein Komma ersetzt werden.

2. X und Y: X-Koordinate (0-39) beziehungsweise Y-Koordinate (0-24), ab der der String ausgegeben werden soll. Die Angabe bezieht sich auf die Spalten- und Zeilenorientierung des Text-Bildschirms. Beispiel: Mit CHAR 1,0,24,"DIES IST EIN TEST" wird der String "DIES IST EIN TEST" am unteren Bildschirmrand ausgegeben, unabhängig davon, ob der Text- oder der Graphik-Modus eingeschaltet ist! Beachten Sie bitte diese Art der

Angabe, die sich auch bei eingeschaltetem Graphik-Modus an der Bildschirm-Unterteilung des Text-Modus orientiert.

3. Invers: Mit dieser optionalen Angabe kann ein String - durch Angabe einer 1 - invers ausgegeben werden. Beispiel: CHAR 1,0,0,"DIES IST EIN TEST",1 gibt den String "DIES IST EIN TEST" invers dargestellt am linken oberen Bildschirmrand aus.

Mit dem CHAR-Befehl können Strings an beliebigen Bildschirm-Positionen ausgegeben werden. CHAR kann wie erwähnt auch im reinen Text-Modus als vollwertiger Ersatz für den PRINT-Befehl verwendet werden. Wird als String ein sogenannter "Leerstring" (A$="") angegeben, erfolgt keinerlei Zeichenausgabe, der Cursor wird jedoch auf die angegebene Position gesetzt (CHAR 1,10,10,"" positioniert den Cursor auf Spalte 10 von Zeile 10).

Mit Hilfe des CHAR-Befehls ist es möglich, dem Benutzer bei jeder Umschaltung des Modus mitzuteilen, welcher Modus eingestellt wurde.

```
400 REM MODUS UMSCHALTEN
410 MODE=MODE+1:REM NAECHSTER MODUS
420 IF MODE>2 THEN MODE=0
421 IF MODE=0 THEN CHAR 1,0,24,"LOESCH-MODUS    "
422 IF MODE=1 THEN CHAR 1,0,24,"MAL-MODUS       "
423 IF MODE=2 THEN CHAR 1,0,24,"BEWEGUNGS-MODUS"
430 RETURN
```

3.2.2.4 Linien ziehen und Rechtecke malen

Alle folgenden Programmteile werden mit Hilfe der Funktionstasten gesteuert. Die Abfrage der Funktionstasten ist jedoch problematisch, da diese nicht mit einem einzelnen Zeichen, sondern mit Zeichenketten wie zum Beispiel "RUN" oder "LIST" belegt sind, GET jedoch die Eingabe eines Zeichens erwartet.

Die Lösung besteht in einer Änderung der "Funktionstasten-Belegung". Mit dem Befehl KEY (N),(STRING) kann eine beliebige Funktionstaste mit einer angegebenen Zeichenkette oder auch einem einzelnen Zeichen belegt werden. Da die Buchstabentasten in diesem Programm nicht benötigt werden, belegen wir die Funktionstasten wie folgt:

```
10 KEY 1,"A":REM F1 BELEGEN
20 KEY 2,"B":REM F2 BELEGEN
```

Übrigens: Um die Funktionstaste HELP zu belegen, ist folgender Befehl nötig: KEY 8,(STRING). Die Funktionstaste HELP ist für den C16, C116 und Plus/4 die achte Funktionstaste und wird daher mit KEY 8 angesprochen (KEY HELP,"D" führt zu einer Fehlermeldung).

Entsprechend muß die Tastenabfrage erweitert werden.

```
212 IF A$="A" OR A$="B" THEN GOSUB 500:GOTO 210 :REM F-TASTE?
```

Wenn A$ einem der Zeichen "A", "B" oder "C" entspricht, wird das Unterprogramm "Funktionstasten behandeln" aufgerufen, das ab Zeile 500 beginnt. Nach der Rückkehr aus diesem Unterprogramm wird erneut zur Tastenabfrage gesprungen.

```
50 XA=-1:REM KEIN PUNKT MARKIERT

500 REM FUNKTIONSTASTEN BEHANDELN
510 IF XA=-1 THEN XA=X:YA=Y:RETURN
520 IF A$="A" THEN DRAW 1,XA,YA TO X,Y
530 IF A$="B" THEN BOX 1,XA,YA,X,Y
540 XA=-1:REM PUNKT 1 NICHT MARKIERT
550 RETURN
```

Um Linien zu ziehen oder Rechtecke zu malen, müssen jeweils zwei Punkte angegeben werden: Die beiden Endpunkte der Linie beziehungsweise die linke obere und die rechte untere Ecke des Rechtecks.

Diese doppelte Angabe wird wie folgt gelöst: Der Benutzer drückt eine der beiden Funktionstasten F1 oder F2. Das Programm merkt sich die Koordinaten der aktuellen Cursorposition (Punkt eins). Der Benutzer bewegt den Cursor danach zu einem beliebigen weiteren Punkt. Drückt er anschließend die gleiche Funktionstaste ein weiteres Mal, ist auch der zweite benötigte Punkt festgelegt, der durch die neue Cursorposition definiert wird.

Der Programmablauf: Am Programmanfang wird eine weitere Zeile (Zeile 50) eingefügt, die nach dem Programmstart der Variablen XA den Wert -1 zuweist. Gehen wir davon aus, daß der Benutzer eine Linie ziehen will und F1 betätigt. In Zeile 500 wird geprüft, ob XA den Wert -1 besitzt. Bei der ersten Betätigung von F1 ist dies der Fall (siehe Zeile 50). Daher wird die aktuelle Cursorposition gemerkt, die Inhalte der Variablen X und Y werden den Variablen XA und YA zugewiesen. Das Unterprogramm wird anschließend verlassen (RETURN).

Wird die Funktionstaste F1 ein weiteres Mal betätigt (nachdem der Benutzer den Cursor an den Endpunkt der gewünschten Linie bewegt hat), ist die Bedingung IF XA=-1 nicht erfüllt, der THEN-Zweig wird nicht ausgeführt, sondern die nächste Zeile des Unterprogramms bearbeitet.

In Zeile 520 wird geprüft, ob A$ (der String, dem mit GETKEY die gedrückte Taste zugewiesen wurde) das Zeichen "A" enthält. Da der Benutzer F1 betätigte und diese Taste mit dem Zeichen "A" belegt wurde, ist die Bedingung IF A$="A" erfüllt und der folgende THEN-Zweig wird ausgeführt.

Der THEN-Zweig enthält den Befehl: DRAW 1,XA,XY TO X,Y. Dieser Befehl zeichnet eine Linie in der eingestellten Punktfarbe zwischen den beiden Punkten XA/YA und X/Y. Die Endpunkte der Linie sind durch den zuerst markierten und in XA beziehungsweise YA festgehaltenen Punkt eins und die aktuelle Cursorposition X/Y festgelegt.

Die Linie wird gezeichnet und die nächste Zeile übergangen, da die Bedingung IF A$="B" nicht erfüllt ist. In Zeile 540 wird der Variablen XA der Wert -1 zugewiesen. Der Grund: Der Wert -1 ist für das Programm das Kennzeichen, an dem es erkennt, ob eine Funktionstaste zum ersten oder bereits zum zweiten Mal betätigt wird, ob der erste Punkt markiert wird, oder aber bereits der zweite Punkt und die Linie gezeichnet werden soll.

Besitzt XA den Wert -1, geht das Programm davon aus, daß der erste Punkt noch nicht markiert wurde und diese Aktion nun stattfinden soll (siehe Zeile 510). Wurde eine Linie gezeichnet, erhält XA wieder den Anfangswert -1, da für eine weitere Linie der erste Punkt noch nicht markiert wurde.

Der Wert -1 wird als sogenanntes "Flag" benutzt, als Kennzeichen, das angibt, welcher von mehreren Zuständen vorliegt (Punkt eins bereits markiert oder noch nicht markiert). Die Zahl -1 wurde verwendet, da diese X-Koordinate niemals versehentlich auftreten kann (mögliche X-Koordinaten: 0-319).

Doch nun zu Zeile 530, dem Zeichnen eines Rechtecks. Der Ablauf dieser Funktion unterscheidet sich prinzipiell nicht vom Ziehen einer Linie. Wenn zum ersten Mal F2 gedrückt wird, merkt sich das Programm die Koordinaten von Punkt eins, der linken oberen Ecke des Rechtecks.

Wird F2 zum zweiten Mal betätigt, stehen auch die Koordinaten des zweiten Punktes fest, der rechten unteren Ecke des Rechtecks. Sie entsprechen den aktuellen Kursorkoordinaten, das heißt den momentanen Werten der Variablen X und Y.

Das Rechteck wird unter Angabe dieser beiden Punkte gezeichnet und - nachdem XA auf den Ausgangswert -1 gesetzt wurde (kein Punkt für ein weiteres Rechteck markiert) - das Unterprogramm verlassen.

Unser einfaches Malprogramm ist nun vollständig. Wenn Sie die noch fehlenden Zeilen eingegeben haben, sollten Sie das komplette Programm mit dem Befehl DSAVE"MALEN auf

Diskette beziehungsweise mit SAVE"MALEN auf Cassette speichern.

3.2.2.5 Listing des Malprogramms

```
10 KEY 1,"A":REM F1 BELEGEN
20 KEY 2,"B":REM F2 BELEGEN
30 :
50 XA=-1:REM KEIN PUNKT MARKIERT
60 :
100 GRAPHIC 1,1:REM HI-RES EIN, LOESCHEN
110 COLOR 1,2,6:REM PUNKTE: HELLES WEISS
120 COLOR 0,1,0:REM HINTERGRUND: DUNKLES SCHWARZ
130 COLOR 4,1,0:REM RAHMEN: DUNKLES SCHWARZ
140 :
150 MODE=2:REM BEWEGEN
160 X=160:Y=100:REM AUSGANGSKOORDINATEN FUER CURSOR
170 LOCATE X,Y:REM GRAPHIK-CURSOR POSITIONIEREN
180 FQ=RDOT(2):REM FARQUELLE MERKEN
190 :
200 REM TASTENABFRAGE UND VERZWEIGUNG
210 GETKEY A$:REM AUF TASTE WARTEN
211 IF A$=CHR$(13) THEN GOSUB 400:GOTO 210:REM RETURN-TASTE?
212 IF A$="A" OR A$="B" THEN GOSUB 500:GOTO 210 :REM F-TASTE?
220 IF MODE=2 THEN DRAW FQ,X,Y:ELSE DRAW MODE,X,Y
230 IF A$=CHR$(17)  THEN Y=Y+1:REM CURSOR UNTEN
240 IF A$=CHR$(29)  THEN X=X+1:REM CURSOR RECHTS
250 IF A$=CHR$(145) THEN Y=Y-1:REM CURSOR OBEN
260 IF A$=CHR$(157) THEN X=X-1:REM CURSOR LINKS
270 LOCATE X,Y:REM CURSOR POSITIONIEREN
280 FQ=RDOT(2):REM FARBQUELLE MERKEN
290 DRAW 1,X,Y:REM CURSOR-PUNKT SETZEN
300 GOTO 210:REM ZUR TASTENABFRAGE
310 :
400 REM MODUS UMSCHALTEN
410 MODE=MODE+1:REM NAECHSTER MODUS
420 IF MODE>2 THEN MODE=0
421 IF MODE=0 THEN CHAR 1,0,24,"LOESCH-MODUS   "
422 IF MODE=1 THEN CHAR 1,0,24,"MAL-MODUS      "
```

```
423 IF MODE=2 THEN CHAR 1,0,24,"BEWEGUNGS-MODUS"
430 RETURN
440 :
500 REM FUNKTIONSTASTEN BEHANDELN
510 IF XA=-1 THEN XA=X:YA=Y:RETURN
520 IF A$="A" THEN DRAW 1,XA,YA TO X,Y
530 IF A$="B" THEN BOX 1,XA,YA,X,Y
540 XA=-1:REM PUNKT 1 NICHT MARKIERT
550 RETURN
```

3.2.3 Kreise zeichnen und Flächen ausfüllen

Unser Malprogramm ist zwar fertiggestellt, das Graphik-Kapitel jedoch noch nicht beendet. Außer DRAW und BOX gibt es noch weitere Befehle zum Zeichnen von Figuren. Einer der komplexesten Befehle ist CIRCLE. Mit CIRCLE können Kreise gemalt werden.

```
CIRCLE (FARBQUELLE), (MITTELPUNKT: X), (MITTELPUNKT: Y), (BREITE),
[(HÖHE), (ANFANG), (ENDE), (DREHWINKEL)]
```

Alle Parameter in eckigen Klammern sind optional und müssen nicht unbedingt angegeben werden. Prinzipiell wird ein Kreis durch den Mittelpunkt und den Radius definiert. Der Mittelpunkt wird durch seine X- und Y-Koordinate festgelegt (MITTELPUNKT: X und MITTELPUNKT: Y), der Parameter BREITE entspricht dem Kreisradius. Diese Angaben reichen aus, um mit einem kleinen Demoprogramm einen Kreis zu zeichnen.

```
100 GRAPHIC 1,1:REM GRAPHIK EIN
110 CIRCLE 1,160,100,50:REM KREIS ZEICHNEN
120 GETKEY A$:REM AUF TASTE WARTEN
130 GRAPHIC 0:REM GRAPHIC AUS
```

Dieses Programm zeichnet einen Kreis, dessen Mittelpunkt sich in der Bildschirmmitte befindet (160/100), und der den Radius 50 besitzt.

Durch gleichzeitige Angabe von BREITE und HÖHE können beliebige Ellipsen gezeichnet werden. Eine Ellipse entsteht immer dann, wenn BREITE und HÖHE nicht identisch sind.

```
110 CIRCLE 1,160,100,50,80:REM ELLIPSE ZEICHNEN
```

Die Angaben ANFANG und ENDE werden zum Zeichnen beliebiger Kreisausschnitte verwendet. ANFANG kennzeichnet den Anfang und ENDE das Ende des zu zeichnenden Kreisausschnitts. Die Maßeinheit ist jeweils Grad. Mit der Angabe von 90 als Anfang und 360 als Ende eines Ausschnitts wird ein Dreiviertelkreis gezeichnet.

```
110 CIRCLE 1,160,100,50,,90,360:REM KREISAUSSCHNITT
```

Bitte beachten Sie, daß der Parameter HÖHE nicht angegeben und daher durch ein Komma ersetzt wird. Durch eine von BREITE differierende Angabe kann ein Ellipsenausschnitt gezeichnet werden.

```
110 CIRCLE 1,160,100,50,80,90,360:REM ELLIPSENAUSSCHNITT
```

Der Ellipsenausschnitt kann durch Angabe des DREHWINKELS gedreht werden, zum Beispiel um 50 Grad.

```
110 CIRCLE 1,160,100,50,80,90,360,50:REM ELLIPSENAUSSCHNITT GEDREHT
```

Selbstverständlich ist auch eine Drehung des Kreisauschnitts möglich.

```
110 CIRCLE 1,160,100,50,,90,360,50:REM KREISAUSSCHNITT GEDREHT
```

Auch ein Vollkreis kann um beliebige Winkel gedreht werden. Allerdings werden Sie keinen Unterschied zwischen gedrehten und nicht gedrehten Vollkreisen festellen.

Wie Sie sehen, ist der CIRCLE-Befehl außerordentlich flexibel. So flexibel, daß sehr viel Übung benötigt wird, um alle Möglichkeiten dieses Befehls auszunutzen.

Ein weiterer, jedoch recht einfacher Befehl ist PAINT. Mit PAINT können beliebige Flächen mit einer Farbe gefüllt werden.

```
PAINT (FARBQUELLE),(X),(Y)
```

Die Koordinaten X und Y geben einen beliebigen Punkt innerhalb der zu füllenden Fläche an. Achtung! Die Fläche muß völlig geschlossen sein, sonst wird mit PAINT der komplette Bildschirm "angemalt".

```
100 GRAPHIC 1,1:REM GRAPHIK EIN
110 CIRCLE 1,160,100,50:REM KREIS ZEICHNEN
112 PAINT 1,160,100:REM MIT PUNKTFARBE FUELLEN
114 PAINT 0,160,100:REM MIT HINTERGRUND-F.FUELLEN
120 GETKEY A$:REM AUF TASTE WARTEN
130 GRAPHIC 0:REM GRAPHIC AUS
```

Dieses Programm malt einen Kreis mit dem Mittelpunkt 160/100 und dem Radius 50. Der Kreis wird anschließend mit der Farbe von Farbquelle 1 (Punktfarbe 1) ausgefüllt. Als innerhalb des Kreises liegender Punkt wird einfach der Kreismittelpunkt angegeben. Zeile 114 löscht den Kreis, indem er wiederum gefüllt wird, jedoch mit der Farbe von Farbquelle 1, der aktuellen Hintergrund-Farbe.

3.2.4 Multicolor-Graphik

Im Multicolor-Modus stehen drei verschiedene Punktfarben zur Verfügung. Der Nachteil besteht in der geringeren Auflösung, die nur noch 160 Punkte in der X-Richtung beträgt.

Mit dem Befehl COLOR 3,1 wird die Multicolor-Graphik eingeschaltet und der Bildschirm gelöscht. Mit COLOR 4,1 wird analog dem Hi-Res-Befehl COLOR 2,1 der untere Bildschirmbereich (Zeile 20 bis 24) für den Text-Modus reserviert.

Alle Befehle des Hi-Res-Modus können auch im Multicolor-Modus verwendet werden (DRAW, BOX usw.). Der Unterschied besteht - außer in der geringeren Auflösung - darin, daß nun auch die Punktfarben zwei und drei sinnvoll anzuwenden sind.

Das folgende Beispiel zeichnet eine Graphik im Multicolor-Modus. Es malt drei Kreise in unterschiedlichen Farben und unterschiedlichen Helligkeiten auf den Bildschirm.

```
100 GRAPHIC 3,1:REM MULTICOLOR EIN
110 COLOR 0,1,0:REM HINTERGRUND: DUNKELSCHWARZ
120 COLOR 4,1,0:REM RAHMEN: DUNKELSCHWARZ
130 COLOR 1,9,6:REM PUNKTFARBE 1: HELLORANGE
140 COLOR 2,6,4:REM PUNKTFARBE 2: MITTELGRUEN
150 COLOR 3,8,2:REM PUNKTFARBE 3: DUNKELGELB
160 :
170 CIRCLE 1,60,60,30:REM KREIS IN FARBE 1
180 CIRCLE 2,90,90,30:REM KREIS IN FARBE 2
190 CIRCLE 3,120,120,30:REM KREIS IN FARBE 3
200 :
210 GETKEY A$:REM AUF TASTE WARTEN
220 GRAPHIC 0:REM MULTICOLOR AUS
```

Außer der Farbenvielfalt und der geringeren Auflösung besteht ein weiterer Unterschied zum Hi-Res-Modus. Alle mit der Punktfarbe 3 gezeichneten Objekte ändern Ihre Farbe, wenn diese Punktfarbe mit dem COLOR-Befehl geändert wird.

Diese spezielle Eigenschaft von Punktfarbe 3 können Sie mit den folgenden Programmzeilen testen, die in das obige Demoprogramm einzufügen sind.

```
192 FOR X=1 TO 7
193 COLOR 3,X,X:REM FARBE 3 AENDERN
194 FOR Y=1 TO 1000:NEXT Y:REM WARTESCHLEIFE
195 NEXT X
```

Punktfarbe 3 wird schrittweise von dunklem Schwarz hin zu hellem Blau verändert. Jeder COLOR-Befehl ändert sofort die Farbe des mit Punktfarbe drei gezeichneten Kreises.

3.2.5 SHAPES, die Graphik-Objekte des C16, C116, Plus/4

Die Befehle DRAW, CIRCLE, BOX und PAINT werden vorwiegend in "statischen" Graphikprogrammen eingesetzt. Die Positionen der gezeichneten Figuren können nachträglich nur schwer verändert werden.

Speziell Spielprogramme zeichnen sich jedoch durch "dynamische", das heißt durch bewegte Graphiken aus. Bewegung kam in unserem Malprogramm durch die variierbare Cursorposition zustande. Wir sahen jedoch, mit welchen Schwierigkeiten es bereits verbunden ist, einen einfachen Punkt durch eine Graphik zu steuern, ohne "Spuren" zu hinterlassen.

In einem Spielprogramm werden jedoch meist nicht nur Punkte, sondern komplexe Graphiken (Schiffe, Autos etc.) über den Bildschirm bewegt. Der C16, C116 bzw. Plus/4 besitzt zwei Befehle, die eigens zur Bewegung derartig komplexer Figuren vorhanden sind, die Befehle SSHAPE und GSHAPE. Mit diesen Befehlen können beliebige Graphik-Objekte, die sogenannten "SHAPES", in einer Variablen gespeichert oder auch - analog einem Foto - "aufgenommen" und an beliebigen Bildschirmpositionen "wiedergegeben" werden.

3.2.5.1 Figuren mit SSHAPE speichern

Mit dem Befehl SSHAPE kann ein beliebiger Ausschnitt des Graphik-Bildschirms - ein SHAPE - einer Stringvariablen zugewiesen werden. Die genaue Funktionsweise dieses Befehls ist sehr komplex, für uns jedoch unwichtig. In der Praxis interessiert nur, daß nach Anwendung dieses Befehls die angegebene Stringvariable eine Zeichenkette enthält, die in einem speziellen Format exakt den angegebenen Bildschirmausschnitt wiederspiegelt.

SSHAPE kann nur in Verbindung mit dem Befehl GSHAPE sinnvoll verwendet werden. GSHAPE überträgt den Inhalt (den Bildschirmausschnitt) einer angegebenen Variablen auf den Graphik-Bildschirm, an eine beliebige anzugebende Position.

Das Format beider Befehle ist leider außerordentlich kompliziert und wird im folgenden ausführlich erläutert.

```
SSHAPE (VARIABLE),(X1),(Y2)[,(X2),(Y2)]
```

1. Variable: Name der Stringvariablen, die den zu speichernden Ausschnitt enthalten soll.

2. X1 und Y1: Koordinaten der linken oberen Ecke der jeweiligen Figur (abgespeichert wird immer ein rechteckiger Ausschnitt).

3. X2 und Y2: Koordinaten der rechten unteren Ecke der jeweiligen Figur. Diese Angabe ist optional. Entfällt die Angabe, setzt der BASIC-Interpreter stattdessen die aktuelle Cursorposition ein. Befindet sich der Cursor nicht an der gewünschten rechten unteren Ecke des Ausschnitts, müssen deren Koordinaten unbedingt angegeben werden!

Beispiele:

1. SSHAPE A$,150,100,200,150 speichert in der Variablen "A$" einen Bildschirmausschnitt, dessen (im Hi-Res-Modus) linke obere Ecke in der Mitte des Bildschirms liegt, und dessen Länge ebenso wie seine Breite jeweils 50 Punkte in X- beziehungsweise Y-Richtung beträgt.

2. SSHAPE XY$,75,100,125,150 speichert den entsprechenden Ausschnitt im Multicolor-Modus in der Variablen "A$". Ausgehend vom Multicolor-Modus mit der geringeren X-Auflösung (150 Punkte) speichert dieser Befehl ebenfalls einen Ausschnitt, dessen

linke obere Ecke in der Bildschirmmitte liegt und der 50 Punkte breit und lang ist.

Da die Länge einer Stringvariablen begrenzt ist (maximal 255 Zeichen), ist die Frage interessant, wie groß ein Ausschnitt höchstens sein kann. Mit den folgenden Formeln kann die bei einem bestimmten Ausschnitt benötigte Stringlänge berechnet werden.

```
LÄNGE=INT((ABS(X1-X2)+1)/FAKTOR+Ø.99)*(ABS(Y1-Y2)+1)+4
```

X1/Y1 und X2/Y2 sind die Koordinaten der angegebenen Eckpunkte. FAKTOR ist ein Wert, der vom jeweiligen Graphik-Modus abhängt.

Hi-Res-Modus: FAKTOR=8
Multicolor-Modus: FAKTOR=4

Die Funktion ABS wurde noch nicht erläutert. Sie ergibt den "Absolutbetrag" der in Klammern angegebenen Zahl. Beispiel: ABS(-30) ergibt 30 und ABS(30) ergibt ebenfalls 30. ABS wird vorwiegend dann angewendet, wenn nur die Differenz zweier Zahlen interessiert, jedoch nicht, welche der beiden Zahlen größer und welche kleiner ist.

Sollten Sie diese Berechnungen häufiger benötigen (wenn Sie nicht sicher sind, ob ein abzuspeichernder Bereich die maximale Stringlänge überschreitet oder nicht), empfiehlt es sich, diese Formel in ein kleines Programm umzusetzen.

```
1ØØ INPUT "ECKE OBEN LINKS (X,Y)";X1,Y1
11Ø INPUT "ECKE UNTEN RECHTS (X,Y)";X2,Y2
12Ø INPUT "HI-RES- (H) ODER MULTICOLOR-MODUS (M)";MO$
13Ø IF MO$="H" THEN FA=8:ELSE FA=4
14Ø SL=INT((ABS(X1-X2)+1)/FA+Ø.99)*(ABS(Y1-Y2)+1)+4
15Ø PRINT SL
```

```
RUN
OBEN LINKS? Ø,Ø
UNTEN RECHTS? 7,7
MODUS? H
12
```

Die Programmbedienung: Nach dem Starten werden Sie nach den beiden Eckkordinaten gefragt. Geben Sie bitte jeweils die X- und die Y-Koordinate an und trennen Sie beide Koordinaten durch ein Komma.

Auf die Frage "MODUS?" antworten Sie mit "H", wenn Sie im Hi-Res-Modus, und mit "M", wenn Sie im Multicolor-Modus arbeiten wollen. Das Programm gibt nach Beantwortung aller Fragen die benötigte Stringlänge aus.

Wenn Sie dieses Programm öfter benötigen, empfiehlt sich das Speichern auf Cassette oder Diskette, um ein wiederholtes Eintippen zu vermeiden.

Wenn ein Bildschirmausschnitt zu groß ist, um mit dem SSHAPE-Befehl in ein Variable übertragen zu werden, erfahren Sie dies auch ohne das vorgestellte Programm. Ein SSHAPE-Befehl, der aufgrund der Ausschnittgröße nicht durchführbar ist, erzeugt die Fehlermeldung "ILLEGAL QUANTITY ERROR IN (ZEILENNUMMER)".

Um Ihnen einen ungefähren Eindruck der speicherbaren Ausschnittsgröße zu geben: Im Hi-Res-Modus kann ein rechteckiger Ausschnitt maximal 41*41 Punkte groß sein, was einem jeweils 50 Punkte breiten und langen Rechteck entspricht.

3.2.5.2 Gespeicherte Figuren mit GSHAPE wiedergeben

Mit dem Befehl GSHAPE kann eine gespeicherte Figur an einer beliebigen Bildschirmposition wiedergegeben werden.

```
GSHAPE (VARIABLE) [,(X),(Y),(MODUS)]
```

1. VARIABLE: Die Stringvariable, der der gewünschte Bildschirmausschnitt zugewiesen wird.

2. X und Y: Optionale Angabe der linken oberen Ecke des Rechtecks zur Bestimmung der Wiedergabeposition. Entfällt diese Angabe, verwendet der Interpreter stattdessen die aktuelle Cursorposition als Position dieser Ecke.

3. MODUS: Diese optionale Angabe besitzt eine Vielzahl von Bedeutungen.
   - MODUS=0 (Standardwert): Die Fläche wird wie "aufgenommen" wiedergegeben.
   - MODUS=1: Die Fläche wird invertiert wiedergegeben, das heißt wo zuvor im gespeicherten SHAPE ein Punkt gesetzt war, wird kein Punkt gesetzt und umgekehrt.
   - MODUS=2: Die Fläche wird mit dem Untergrund an der Wiedergabeposition ODER-verknüpft. Das bedeutet: Die auf dem Untergrund vorhandenen Punkte werden nicht überschrieben, sondern bleiben erhalten. Die gesetzten Punkte des aufgenommenen SHAPES werden zusätzlich gesetzt. Am besten vergleichen Sie diese ODER-Verknüpfung von SHAPE und Untergrund mit zwei verschiedenen Dias, die übereinandergelegt werden.
   - MODUS=3: UND-Verknüpfung von SHAPE und Untergrund an der Wiedergabeposition. Punkte werden nur an jenen Positionen gesetzt, an denen sowohl im SHAPE (in der Variablen enthalten) als auch auf dem Untergrund ein Punkt gesetzt ist.
   - MODUS=4: EXCLUSIV-ODER-Verknüpfung von SHAPE und Untergrund. Punkte werden nur dort gesetzt, wo entweder das SHAPE oder aber der Untergrund einen gesetzten Punkt aufweisen.

Das folgende Beispiel demonstriert den gemeinsamen Einsatz der Befehle SSHAPE und GSHAPE.

```
100 GRAPHIC 1,1:REM HI-RES EIN
110 BOX 1,0,0,40,40:REM RECHTECK MIT ECKEN 0/0 U.40/40
120 SSHAPE A$,0,0,40,40:REM RECHTECK 0/0 BIS 40/40 NACH A$
130 SCNCLR:REM BILDSCHIRM LOESCHEN
140 GSHAPE A$,150,100:REM RECHTECK AB 150/100 (OBEN LINKS)
150 GETKEY A$:REM AUF TASTE WARTEN
160 GRAPHIC 0:REM HI-RES AUS U.TEXT-MODUS EIN
170 PRINT LEN(A$):REM LAENGE DES STRINGS A$ AUSGEBEN
```

Der Programmablauf: In Zeile 100 wird die hochauflösende Graphik eingeschaltet. Zeile 110 zeichnet ein Rechteck, dessen linke obere Ecke sich bei 0/0 und dessen rechte untere Ecke sich an der Koordinate 40/40 befindet. Breite und Länge des Rechtecks betragen daher jeweils 41 Punkte (0-40) und entsprechen der maximalen Ausschnittsgröße, die einer Stringvariablen zugewiesen werden kann.

In Zeile 120 wird ein Ausschnitt, der exakt das gezeichnete Rechteck umfaßt (identische Koordinaten) der Variablen A$ zugewiesen. Der in Zeile 130 folgende Befehl SCNCLR ist neu für uns. Dieser Befehl löscht den Bildschirm und zwar unabhängig vom jeweiligen Modus. Mit SCNCLR kann sowohl im Text- als auch im Hi-Res- und im Multicolor-Modus der Bildschirm gelöscht werden.

Nachdem der Bildschirm gelöscht wurde, wird der gespeicherte Ausschnitt an einer völlig anderen Position mit GSHAPE wiedergegeben. Die Positionsangabe 150/100 kennzeichnet die Position der linken oberen Ecke des wiederzugebenden Ausschnitts.

Da die Angabe MODUS entfällt, verwendet der Interpreter den Standardwert 0, das heißt der Ausschnitt wird exakt so wiedergegeben, wie er (in A$) gespeichert wurde.

Nachdem Sie eine beliebige Taste betätigt haben (Zeile 150), wird der Text-Modus eingeschaltet und mit dem Befehl PRINT LEN(A$) die Länge der Zeichenkette ausgegeben, die der Variablen A$ zugewiesen wurde. Diese Länge beträgt 250 Zeichen. Sie sehen, mit dem gespeicherten Ausschnitt bewegen wir uns am Rande der Aufnahmekapazität von Stringvariablen (255 Zeichen).

Nun interessiert Sie sicherlich, welche speziellen Effekte sich mit der Angabe des MODUS erzielen lassen. Geben Sie hierzu bitte das folgende Demoprogramm ein. Dieses Demoprogram weist einen Ausschnitt (mit BOX gezeichnet) einer Variablen zu und gibt diesen Ausschnitt mit den möglichen Angaben für den MODUS an unterschiedlichen Bildschirmpositionen wieder.

```
100 GRAPHIC 1,1:REM HI-RES EIN
110 DRAW 1,0,100 TO 319,100:REM LINIE QUER UEBER BILDSCHIRM
120 BOX 1,0,0,40,40:REM RECHTECK MALEN
130 SSHAPE A$,0,0,40,40:REM AUSSCHNITT SPEICHERN
140 :
150 GSHAPE A$,10,80,0:REM STANDARDWIEDERGABE
160 GSHAPE A$,70,80,1:REM INVERSWIEDERGABE
170 GSHAPE A$,130,80,2:REM ODER-VERKNUEPFUNG
180 GSHAPE A$,190,80,3:REM UND-VERKNUEPFUNG
190 GSHAPE A$,250,80,4:REM EXCLUSIC-ODER-VERKN.
200 GETKEY B$:REM AUF TASTE WARTEN
210 GSHAPE A$,130,80,4:REM EXOR-VERKN.MIT BEREICH 3
220 :
230 GETKEY B$:REM AUF TASTE WARTEN
230 GRAPHIC 0:REM TEXT-MODUS EINSCHALTEN
```

Ich finde, daß dieses Programm die Wirkungsweise der MODUS-Angabe und der resultierenden Verknüpfung von aufgenommenem Objekt und Untergrund an der Wiedergabeposition äußerst anschaulich demonstriert.

Das Programm speichert ebenfalls ein mit BOX gezeichnetes Rechteck in der Variablen A$. Außer dem Rechteck wird mit DRAW eine quer über den Bildschirm verlaufende Linie gezeichnet (ab 0/0 bis 319/100).

Die folgenden sechs GSHAPE-Befehle zeichnen das aufgenommene Objekt an verschiedenen Bildschirmpositionen. Jede Position ist so gewählt, daß sich das Objekt mit der vorhandenen Linie - dem Untergrund, auf den das Objekt "gelegt" wird - überschneidet. Folgende Variationen ergeben sich:

1. GSHAPE mit MODUS=0 führt dazu, daß die Linie von dem darübergelegten Rechteck "überschrieben" wird. An der Position, an der der rechteckige Ausschnitt, das "SHAPE" wiedergegeben wird, befindet sich anschließend nur (!) der mit BOX gezeichnete Rahmen. Die zuvor vorhandene Linie ist verschwunden. Mit MODUS=0 wird somit der alte Untergrund komplett überschrieben.

2. MODUS=1, die "Invers-Wiedergabe" überschreibt ebenfalls den alten Untergrund. Der Rahmen wird jedoch invers wiedergegeben. Der Original-Rahmen war leer, der mit GSHAPE wiedergegebene Rahmen ist mit gesetzten Punkten gefüllt.

3. Die ODER-Verknüpfung von SHAPE und Untergrund (MODUS=2) legt das SHAPE - den Rahmen - über den Untergrund, ohne daß dieser komplett ausgelöscht wird. Im Leerraum des Rahmens befindet sich immer noch die bereits zuvor vorhandene Linie. Das Ergebnis weist überall dort gesetzte Punkte auf, wo entweder das SHAPE (der Rahmen) oder aber der Untergrund (die Linie) gesetzte Punkte aufweisen (oder auch beide zugleich).

4. Der mit MODUS=3 (der UND-Verknüpfung) wiedergegebene Ausschnitt ist beinahe leer. Die einzigen gesetzten Punkte befinden sich an den Kreuzungspunkten von Rahmen-SHAPE und der zuvor vorhandenen Linie. Wie erläutert, bewirkt die UND-Verknüpfung, daß Punkte ausschließlich an Positionen gesetzt werden, an denen sowohl das

SHAPE als auch der Untergrund gesetzte Punkte aufweisen.

5. Der fünfte GSHAPE-Befehl (die EXCLUSIV-ODER-Verknüpfung von SHAPE und Untergrund) weist scheinbar keinerlei Unterschiede zur ODER-Verknüpfung auf. Wenn Sie genau hinschauen, stellen Sie jedoch fest, daß an den beiden Kreuzungspunkten von Rahmen und Linie keine Punkte gesetzt sind. Die EXCLUSIV-ODER-Verknüpfung (kurz: EOR-Verknüpfung oder auch EXOR-Verknüpfung) ist die schwierigste "logische Verknüpfung". Bei der EXOR-Verknüpfung werden nur dort Punkte gesetzt, wo entweder das SHAPE oder aber der Untergrund gesetzte Punkte aufweisen. Im Unterschied zur ODER-Verknüpfung werden an Positionen, an denen sowohl SHAPE als auch Untergrund gesetzte Punkte aufweisen, keine Punkte gesetzt!

Betätigen Sie bitte eine beliebige Taste. Anschließend wird in Zeile 210 das SHAPE - der Rahmen - mit dem dritten Bereich - der Kombination aus Rahmen und Linie durch EXOR verknüpft. Der Rahmen verschwindet, die ursprünglich vorhandene Linie ist jedoch unverändert vorhanden, das heißt der Ausgangszustand ist wiederhergestellt. Wenn Sie genau hinsehen, stellen Sie jedoch eine Ausnahme fest: An den Kreuzungspunkten zwischen Rahmen und Linie wurde der Rahmen ebenfalls ausgelöscht, die beiden Linienpunkte jedoch nicht wiederhergestellt.

Der Ablauf: Der Rahmen wurde mit der entstandenen Kombination aus Rahmen und Linie verknüpft, wobei der SHAPE-Rahmen deckungsgleich über den in der Kombination vorhandenen Rahmen gelegt wurde. Alle Rahmenpunkte sind sowohl im SHAPE als auch im Untergrund gesetzt. Nach den Regeln der EXOR-Verknüpfung werden an diesen Positionen daher keine Punkte gesetzt und der Rahmen gelöscht. Leider betrifft dieses Löschen auch zwei Punkte der Linie, die mit zwei

Rahmenpunkten identisch sind. Daher stammen die beiden anschließend in der Linie vorhandenen Lücken.

3.2.5.3 Bewegung von SHAPES

Im Malprogramm tauchte das Problem auf, den Cursor (einen einzelnen Graphik-Punkt) zu bewegen, ohne Graphiken zu beeinflussen, die überquert werden. Die Lösung dieses Problems war mit einigem Aufwand verbunden.

Mit SHAPES kann die Bewegung eines Objektes über (!) einen Untergrund hinweg stark vereinfacht werden.

1. Das zu bewegende Objekt wird aufgenommen, das heißt mit SSHAPE einer Stringvariablen zugewiesen, zum Beispiel A$.

2. Der Untergrund an jener Position, an der das SHAPE wiedergegeben werden soll, wird aufgenommen und der Variablen OLD$ zugewiesen.

3. Das SHAPE A$ wird mit dem Untergrund ODER-verknüpft. An der gewünschten Position überlagern sich daher SHAPE und Untergrund.

4. Bevor das SHAPE weiterbewegt wird, muß an der aktuellen Position der Ausgangszustand wiederhergestellt werden. Der alte Untergrund wurde der Variablen OLD$ zugewiesen. Der Inhalt dieser Variablen wird mit GSHAPE und im Standard-Modus wiedergegeben, das heißt der alte Untergrund überschreibt den jeweiligen Ausschnitt und ist damit wiederhergestellt.

3.2.5.4 Automatische SHAPE-Bewegung

Das folgende Demoprogramm zeichnet ein schmales Rechteck (linke obere Ecke 0/99; rechte untere Ecke 319/101). Das SHAPE stellt ein "Fadenkreuz" dar, das sich - immer entlang dem Rechteck - quer über den Bildschirm bewegt. Der jeweilige Untergrund - das Rechteck - bleibt trotz der Bewegung des Fadenkreuzes völlig unbeeinflußt, da er nach jeder Bewegung auf die beschriebene Art und Weise wiederhergestellt wird.

```
100 GRAPHIC 1,1:REM HI-RES EIN
110 BOX 1,0,99,319,101:REM RECHTECK MALEN
120 DRAW 1,0,0 TO 4,4:REM FADENKREUZ LINIE1
130 DRAW 1,0,4 TO 4,0:REM FADENKREUZ LINIE2
140 SSHAPE A$,0,0,4,4:REM FADENKREUZ AUFNEHMEN
150 :
160 FOR I=0 TO 319
170 : SSHAPE OLD$,I,98,I+4,102:REM UNTERGRUND AUFNEHMEN
180 : GSHAPE A$,I,98,2:REM FADENKREUZ WIEDERGEBEN
190 : FOR J=1 TO 30:NEXT J:REM KURZ WARTEN
200 : GSHAPE OLD$,I,98:REM UNTERGRUND WIEDERHERSTELLEN
210 NEXT I
220 :
230 GRAPHIC 0:REM TEXT-MODUS EIN
```

Der Ablauf: Nachdem der Hi-Res-Modus eingeschaltet wurde, wird ein schmales Rechteck vom linken zum rechten Rand des Bildschirms gezeichnet. Anschließend werden die beiden Linien eines Fadenkreuzes gezeichnet, die jeweils eine Länge von fünf Punkten besitzen (die beiden Diagonalen in einem Rechteck mit der linken oberen Ecke 0/0 und der rechten unteren Ecke 4/4). Der rechteckige Ausschnitt, in dem sich das Fadenkreuz befindet, wird mit dem SSHAPE-Befehl aufgenommen und der Variablen A$ zugewiesen.

Die folgende Schleife (Zeile 160-210) ist für die Bewegung des SHAPES verantwortlich. Die Schleifenvariable I wird als Zähler für die X-Koordinate der linken oberen SHAPE-Ecke verwendet. I läuft vom Startwert 0 bis zum Endwert 319, dem größten möglichen X-Wert.

In der Schleife werden jeweils drei Schritte ausgeführt:

1. Vor der Wiedergabe des SHAPES und der Beeinflussung des betreffenden Bildschirmausschnitts wird dieser aufgenommen (Zeile 170). Sein Inhalt wird in der Variablen OLD$ gespeichert.

2. Das SHAPE A$ wird mit dem betreffenden Ausschnitt ODER-verknüpft (Zeile 180). Dadurch wird das Fadenkreuz über den Untergrund gelegt, es entsteht eine Kombination aus SHAPE und Untergrund.

3. Eine Warteschleife sorgt dafür, daß dieser Zustand kurzzeitig erhalten bleibt und das Fadenkreuz auch tatsächlich sichtbar ist, bevor es wieder gelöscht wird. Das Löschen erfolgt, indem der Auschnitt (Kombination Fadenkreuz + Untergrund) durch das SHAPE OLD$ (den ursprünglichen Inhalt des Ausschnitts) überschrieben wird.

3.2.5.5 SHAPE-Steuerung mit Tastatur und Joystick

Mit dem gewonnen Wissen sollte es möglich sein, ein Fadenkreuz mit den Cursortasten über den Bildschirm zu bewegen, ähnlich der Bewegung eines Punktes im Malprogramm.

Der benötigte Programmablauf: Vor der eigentlichen Schleife zur Bewegung des Fadenkreuzes muß ein "Initialisierungsteil" durchlaufen werden, in dem

- die Graphik eingeschaltet wird
- Rechteck und Fadenkreuz gezeichnet werden
- das Fadenkreuz einer Variablen zugewiesen wird
- die Ausgangskoordinaten des Fadenkreuzes festgelegt werden (Bildschirmmitte)
- der aktuelle an diesen Ausgangskoordinaten vorhandene Untergrund einer weiteren Variablen zugewiesen wird

Der Ablauf der "Hauptschleife":

1. Auf die Betätigung einer Cursortaste warten.
2. Den alten Untergrund wiederherstellen.
3. Die Fadenkreuz-Koordinate je nach Richtungstaste ändern.
4. Den Untergrund des neuen Ausschnitts "retten", das heißt mit SSHAPE einer Variablen zuweisen.
5. Das Fadenkreuz an der neuen Position wiedergeben (ODER-Verknüpfung mit dem Untergrund).

```
100 GRAPHIC 1,1:REM HI-RES EIN
110 BOX 1,0,99,319,101:REM RECHTECK MALEN
120 CIRCLE 1,160,100,50:REM KREIS MALEN
130 DRAW 1,0,0 TO 4,4:REM FADENKREUZ LINIE1
140 DRAW 1,0,4 TO 4,0:REM FADENKREUZ LINIE2
150 SSHAPE A$,0,0,4,4:REM FADENKREUZ AUFNEHMEN
160 X=158:Y=98:REM AUSGANGS-KOORDINATEN FESTLEGEN
170 SSHAPE OLD$,X,Y,X+4,Y+4:REM UNTERGRUND AUFNEHMEN
180 :
190 GETKEY CT$:REM AUF CURSORTASTE WARTEN
200 GSHAPE OLD$,X,Y:REM UNTERGRUND AN AKTUELLER POSITION
WIEDERHERSTELLEN
210 IF CT$=CHR$(17) THEN Y=Y+1:REM CURSOR UNTEN
220 IF CT$=CHR$(29) THEN X=X+1:REM CURSOR RECHTS
230 IF CT$=CHR$(145) THEN Y=Y-1:REM CURSOR OBEN
240 IF CT$=CHR$(157) THEN X=X-1:REM CURSOR LINKS
250 SSHAPE OLD$,X,Y,X+4,Y+4:REM UNTERGRUND AN NEUER POSITION AUFNEHMEN
260 GSHAPE A$,X,Y,2:REM FADENKREUZ AN NEUER POSITION WIEDERGEBEN
270 GOTO 190:REM ZUM SCHLEIFENANFANG
```

In den Zeilen 100-170 wird die erwähnte Initialisierung vorgenommen. Da der Bildschirm nicht völlig leer sein sollte, werden ein Rechteck und ein Kreis gezeichnet (Zeile 110-120). In den Zeilen 130-150 werden die beiden Linien des Fadenkreuzes gezogen und das Rechteck aufgenommen, in dem das Fadenkreuz enthalten ist. Zeile 110 legt die Ausgangs-Koordinaten für die linke obere Ecke des Fadenkreuzes fest, und in Zeile 170 wird der Ausschnitt, der durch diese Koordinaten festgelegt ist, mit SSHAPE in die Variable OLD$ übertragen.

In der folgenden Hauptschleife wird auf die Betätigung einer der vier Cursortasten gewartet und das jeweilige Zeichen der Variablen CT$ zugewiesen. Bevor das Fadenkreuz bewegt wird, muß an der aktuellen Position der alte Untergrund wiederhergestellt werden, den die Variable OLD$ enthält (Zeile 200).

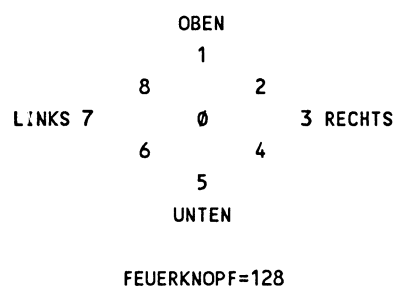
Anschließend werden die Koordinaten X und Y je nach gedrückter Taste verändert (Zeile 210-240). Bevor das Fadenkreuz an der neu festgelegten Position wiedergegeben wird (Zeile 260), wird der momentane Inhalt dieses Ausschnitts in die Variable OLD$ übertragen.

Nachdem das Fadenkreuz diesem Ausschnitt überlagert wurde, ist die Schleife beendet und das Programm kehrt zum Schleifenanfang zurück (Zeile 270).

Sollten Sie einen Joystick besitzen, ist es möglich, die Tastatur-Steuerung des Fadenkreuzes durch die Steuerung mit dem Joystick zu ersetzten. Der Befehl JOY gibt Auskunft über die Richtung, in die der Steuerknüppel bewegt wird.

JOY(N) (N=Nummer des Joystick-Ports (0 oder 1))

Die Abfrage des Joysticks mit der Funktion JOY(N) ergibt einen der folgenden Werte:



Kombinationen aus gedrücktem Feuerknopf und Steuerknüppel werden anhand der Addition der jeweiligen Zahlen erkannt (Feuerknopf + UNTEN = 128 + 5 = 133).

Wie Sie sehen können mit dieser Funktion - im Gegensatz zu den vier Cursortasten - auch diagonale Bewegungsrichtungen abgefragt werden, die jedoch bei der folgenden Programmänderung nicht berücksichtigt werden.

```
190 GSHAPE OLD$,X,Y:REM UNTERGRUND AN AKTUELLER POSITION
WIEDERHERSTELLEN
200 IF JOY(1)=5 THEN Y=Y+1:REM JOYSTICK UNTEN
210 IF JOY(1)=3 THEN Y=Y+1:REM JOYSTICK RECHTS
220 IF JOY(1)=1 THEN Y=Y+1:REM JOYSTICK OBEN
230 IF JOY(1)=7 THEN Y=Y+1:REM JOYSTICK LINKS
240 SSHAPE OLD$,X,Y,X+4,Y+4:REM UNTERGRUND AN NEUER POSITION AUFNEHMEN
250 GSHAPE A$,X,Y,2:REM FADENKREUZ AN NEUER POSITION WIEDERGEBEN
260 GOTO 190:REM ZUM SCHLEIFENANFANG
```

Die benötigten Änderungen sind minimal. Der Befehl GETKEY CT$ entfällt völlig und die vier Vergleiche werden nicht mit der Variablen CT$, sondern mit den vier Joystick-Werten für UNTEN, RECHTS, OBEN und LINKS durchgeführt. Das Programm setzt voraus, daß an Port Nummer 1 ein Joystick angeschlossen ist (JOY(1)).

3.2.6 Graphiken speichern und laden

Ohne eine Erläuterung der Möglichkeiten zum Speichern und Laden einer Graphik wäre dieses Kapitel unvollständig. Leider können mit SAVE oder DSAVE zwar Programme gespeichert werden, nicht jedoch Graphiken.

Zu diesem Zweck muß der eingebaute "Monitor" verwendet werden. Der Monitor ist ein Programm, das üblicherweise nicht zur BASIC-Programmierung des C16, C116 oder Plus/4 verwendet wird. Sie benötigen dieses Programm nur zur Programmierung in der bereits erwähnten Maschinensprache.

Mit folgenden Schritten wird eine Graphik gespeichert:

```
MONITOR
S "FILENAME",GERÄTENUMMER,1800,4000
X
```

Der erste Befehl startet das Monitorprogramm, der zweite Befehl speichert den Graphikbildschirm auf Cassette oder Diskette, und der dritte Befehl beendet das Monitorprogramm.

Außer einem von Ihnen gewählten FILENAMEN für die Graphik muß die GERÄTEADRESSE angegeben werden. Die Datasette besitzt die Geräteadresse 1, die Floppy die Adresse 8. Um zum Beispiel eine Graphik unter dem Namen "GRAPHIK" zu speichern, geben Sie bitte folgendes ein:

```
Datasette:   S"GRAPHIK",1,1800,4000
Floppy:      S"GRAPHIK",8,1800,4000
```

Vor dem Laden einer Graphik müssen Sie den gewünschten Graphik-Modus (Hi-Res oder Multicolor) ein- und wieder ausschalten.

```
GRAPHIC 1,1:REM HI-RES EIN
GRAPHIC 0:REM TEXT-MODUS
```

Diese Vorgehensweise mag Ihnen seltsam erscheinen, sie ist jedoch begründet. Wie erwähnt, benötigt der Graphikbildschirm einen recht großen Speicherplatz. Durch das Einschalten der Graphik wird dieser Platz reserviert. Erst anschließend kann die Graphik selbst geladen werden.

Der Befehl GRAPHIC 0 schaltet die Graphik zwar wieder aus, der einmal reservierte Platz bleibt jedoch vorhanden. Der Text-Modus wird eingeschaltet, damit Sie die folgenden Befehle zum Laden der Graphik eingeben können:

```
MONITOR
L"FILENAME",GERÄTENUMMER
X
```

Um zum Beispiel die unter dem Namen "GRAPHIK" gespeicherte Graphik zu laden, geben Sie bitte ein:

```
Datasette:   L"GRAPHIK",1
Floppy:      L"GRAPHIK",8
```

Die Graphik befindet sich nun im reservierten Speicherplatz und wird sichtbar, sobald der gewünschte Graphik-Modus erneut eingeschaltet wird.

Übrigens: Wenn Sie eine Graphik laden und mit dem erstellten Malprogramm weiterbearbeiten wollen, müssen Sie in diesem eine Zeile ändern. Zeile 100 lautet: 100 GRAPHIC 1,1. Nach dem Starten des Programms mit RUN wird durch diesen Befehl der Graphikbildschirm gelöscht und dadurch die soeben geladene Graphik zerstört.

Ändern Sie daher bitte diesen Befehl in: 100 GRAPHIC 1, wenn Sie bereits erstellte Graphiken laden und weiterbearbeiten wollen.

3.3 Musik

Der C16, C116 bzw. Plus/4 besitzt zwei "Tongeneratoren", mit denen Töne und Rauschen erzeugt werden können. Bevor ich Ihnen näheres über die Tongeneratoren erzähle, bitte ich Sie, das folgende kleine Demoprogramm einzugeben.

```
100 VOL 8
110 SOUND 1,200,300
120 FOR I=1 TO 1000:NEXT I
130 SOUND 2,400,300
```

Dieses Programm schaltet nacheinander beide Tongeneratoren ein. Zuerst hören Sie die eine Stimme, nach kurzer Zeit ertönt ein Duett, die zweite Stimme fällt ein.

Alle zur Steuerung der Tongeneratoren benötigten Befehle finden Sie in dem Demoprogramm. Mit den Befehlen VOL und SOUND lassen sich die musikalischen Fähigkeiten des C16, C116 oder Plus/4 voll ausschöpfen.

```
VOL (LAUTSTÄRKE)
```

LAUTSTÄRKE: 0-8 (0=minimale, 8=maximale Lautstärke)

Mit dem Befehl VOL kann die erzeugte Lautstärke in neun Stufen variiert werden (0-8). VOL 0, die geringste mögliche Lautstärke, entspricht dem Ausschalten der Tongeneratoren.

Vor dem ersten zu erzeugenden Ton oder Geräusch muß mit einem VOL-Befehl die Lautstärke bestimmt werden. Üblicherweise kann der Befehl VOL 8 verwendet werden. Sollte Ihnen die Lautstärke zu hoch sein, können Sie sie mit dem Lautstärkeregler Ihres Fernsehers oder Monitors immer noch stufenlos niedriger regeln.

Nachdem mit VOL die Lautstärke eingestellt wurde, können mit dem SOUND-Befehl Töne oder Geräusche erzeugt werden.

```
SOUND (STIMME),(TONHÖHE),(TONDAUER)
```

Ihnen stehen drei Stimmen zur Verfügung, wobei jedoch immer nur zwei Stimmen gleichzeitig (!) angesprochen werden können.

Stimme 1: Erzeugt Töne
Stimme 2: Erzeugt Töne
Stimme 3: Erzeugt Geräusche

Töne sind reine Sinusschwingungen (mehr oder weniger rein, abhängig von der Qualität der Tongeneratoren) und Geräusche stellen eine Mischung verschiedener Schwingungen dar.

Geräusche lassen sich am besten in Spielen einsetzen, um Explosionen, Fallgeräusche und ähnliches akustisch zu untermalen.

In "seriösen" Programmen wird man meist mit Stimme eins oder zwei (oder mit beiden Stimmen zugleich) Töne erzeugen.

Zur Tonhöhe ist zu sagen, daß Sie leider nicht direkt Noten angeben können. Die Tonhöhe ist eine beliebige Zahl zwischen 0 und 1023. Zwischen der angegebenen Tonhöhe und der zugehörigen Frequenz besteht jedoch eine direkte Beziehung, die durch folgende Formeln wiedergegeben wird:

TONHÖHE =1024-111840.45/FREQUENZ
FREQUENZ=1024-111840.45/TONHÖHE

Zusammen mit der nachfolgenden Tabelle, die die Frequenzen verschiedener Töne wiedergibt, kann problemlos berechnet werden, welche Tonhöhe angegeben werden muß, um eine bestimmte Note zu spielen.

Note	Frequenz
A	110
H	123,5
C	130,8
D	146,8
E	164,7
F	174,5
G	195,9

Diese Tabelle liefert Ihnen die zu jeder Note einer Oktave zugehörige Frequenz. Eine Oktave höher entspricht einer Frequenzverdoppelung, eine Oktave niedriger einer Halbierung der Frequenz. Alle weiteren Oktaven lassen sich daher aus der Tabelle berechnen.

Mit diesen Kenntnissen ist es möglich, ein Programm zu schreiben, daß uns zu jeder Note sowohl die zugehörige Frequenz als auch den Tonhöhenwert angibt.

```
100 DIM N$(7):DIM FR(7):REM ARRYS DIMENSIONIEREN
110 N$(1)="A":FR(1)=110
120 N$(2)="H":FR(2)=123.5
130 N$(3)="C":FR(3)=130.8
140 N$(4)="D":FR(4)=146.8
150 N$(5)="E":FR(5)=164.7
160 N$(6)="F":FR(6)=174.5
170 N$(7)="G":FR(7)=195.9
180 :
190 REM BENUTZEREINGABEN
200 INPUT "NOTE (A-G)";N$
210 INPUT "OKTAVE";OK
220 :
230 REM TABELLE DURCHSUCHEN
240 FOR I=1 TO 7
250 : IF N$=N$(I) THEN FR=FR(I):GOTO 290
260 NEXT I
270 :
280 REM BERECHNUNG
290 IF OK=1 THEN 340:REM 1.OKTAVE?
300 FOR I=1 TO OK-1
310 : FR=FR*2:REM FREQUENZ VERDOPPELN
320 NEXT I
330 :
340 TH=1024-111840.45/FR:REM TONHOEHE BERECHNEN
350 PRINT "FREQUENZ";FR
360 PRINT "TONHOEHE";TH
```

Dieses Programm demonstriert zum ersten Mal den Umgang mit Tabellen. Tabellen werden häufig in Programmen aller Art benötigt und fast immer in Form von Arrays im Speicher gehalten. In der ersten Programmzeile werden die Arrays N$(...) und FR(...) in der benötigten Größe dimensioniert.

In diesen Arrays wird die Tabelle anschließend abgelegt. Je eine Note wird einem Element des Arrays N$(...) zugewiesen und die zugehörige Frequenz dem Element des Arrays FR(...) mit der gleichen Indexzahl.

Die Benutzereingaben NOTE und OKTAVE werden den Variablen N$ und OK zugewiesen. Beachten Sie bitte sowohl bei den Benutzereingaben als auch bei der Tabelle den verwendeten Variablen-Typ. N$ und N$(...) sind Stringvariablen beziehungsweise Stringarrays, denen Zeichenketten zugewiesen werden. OK und FR(...) sind Fließkommavariablen beziehungsweise Fließkommaarrays, denen Zahlen zugewiesen werden.

FR(...) darf keinesfalls durch ein Integerarray ersetzt werden (FR%(...)), da die zugewiesenen Zahlen nicht unbedingt ganzzahlig sind!

Nachdem die Eingaben des Benutzers den Variablen N$ und OK zugewiesen wurden, wird das Stringarray N$(...) nach der eingegebenen Note durchsucht. In einer Schleife wird jedes Element dieses Arrays mit der eingegebenen und N$ zugewiesenen Note verglichen. Wird Übereinstimmung festgestellt, steht auch die zur jeweiligen Note zugehörige Frequenz fest.

Ein Beispiel: Im dritten Schleifendurchgang wird Übereinstimmung zwischen der vom Benutzer eingegebenen Note "C" und dem Arrayelement N$(3) festgestellt, das ebenfalls die Note "C" enthält.

Im diesem dritten Schleifendurchgang besitzt die Schleifenvariable I den Wert 3. Der bei Übereinstimmung ausgeführte Befehl FR=FR(I) entspricht daher dem Befehl FR=FR(3) und weist das dritte Element des Arrays FR(...) der Variablen FR zu. Durch den folgenden GOTO-Befehl wird die Schleife bei Übereinstimmung unverzüglich verlassen, da die gesuchte Frequenz nun feststeht und der Variablen FR zugewiesen wurde.

Allgemein: Jeder Note des Arrays N$(...) entspricht eine Variable mit der zugehörigen Frequenz im Array FR(...), die die gleiche Indexzahl besitzt. Wird beim Durchsuchen des "Noten-Arrays" die gesuchte Note gefunden, kann daher über die Indexzahl (die mitlaufende Schleifenvariable I) unverzüglich auf die zugehörige Frequenz im "Frequenz-Array" zugegriffen werden.

Dieser Ablauf ist für Sie, der Sie noch wenig Erfahrung im Umgamg mit Arrays und Schleifen besitzen, zweifellos nicht leicht zu verstehen. Sie sollten jedoch unbedingt versuchen, diesen Programmteil zu verstehen, da das dargestellte Prinzip zum Durchsuchen von Tabellen in Array-Form sehr häufig benötigt wird.

Die Ermittlung der Frequenz ist noch nicht beendet. Wurde zum Beispiel die Note "A" eingegeben, beträgt die zugehörige Frequenz in der ersten Oktave 110, in der zweiten 220, in der dritten Oktave 440 und so weiter (Frequenzverdoppelung pro höhere Oktave). Die Ermittlung der Frequenz ist nur dann beendet, wenn als Oktave eine 1 eingegeben wurde. In diesem Fall wird der folgende Programmteil mit einem Sprungbefehl übergangen (Zeile 290).

Wurde auf die Frage "OKTAVE?" eine Zahl größer als 1 eingegeben, muß die Frequenz FR entsprechend oft verdoppelt werden. Diese Verdoppelung erfolgt in einer Schleife, die vom Startwert 1 bis zum Endwert OK-1 läuft. Beispiel: Wurde die dritte Oktave angegeben, muß die Frequenz zweimal verdoppelt werden, was zwei Schleifendurchgängen entspricht. Da OK den Wert 3 enthält, läuft die Schleife vom Startwert 1 bis zum Endwert 3 (OK-1 = 3-1 = 2) und es wird wie gewünscht zweimal ausgeführt.

Der letzte Schritt besteht darin, anhand der beschriebenen Formeln aus der Frequenz die im SOUND-Befehl anzugebende Tonhöhe zu gewinnen (Zeile 340) und zum Schluß beide Werte auszugeben. Nachfolgend sehen Sie mehrere Beispiele für mögliche Programmläufe.

```
RUN
NOTE? A
OKTAVE? 1
FREQUENZ 110
TONHOEHE 7.26863635

READY.
```

```
RUN
NOTE? D
OKTAVE? 3
FREQUENZ 587.2
TONHOEHE 833.536Ø18

READY.
```

Die von dem Programm ausgegebenen Werte können Sie sehr leicht anhand der Tabelle im Anhang Ihres Commodore-Handbuches überprüfen, die eine ausführliche Tabelle der Noten und den zugehörigen Frequenzen und Tonhöhen enthält.

Die Angabe der Tondauer im SOUND-Befehl wurde noch nicht erläutert. Sie können beliebige Zahlen zwischen 0 und 65535 angeben. Je kleiner die Zahl ist, desto geringer ist die Tondauer. Mit der Angabe 0 kann ein Ton ausgeschaltet werden.

Geben Sie versuchsweise im Direkt-Modus ein:

```
VOL 8
SOUND 1,11Ø,1ØØØØ

READY.
```

Nach Eingabe beider Befehle werden Sie den Ton A (erste Oktave) hören. Die angegebene Tondauer 10000 entspricht etwa 200 Sekunden (Die Tonlänge in Sekunden erhalten Sie, wenn Sie Tondauer durch 50 teilen). Sie können den Ton jederzeit vor Ablauf dieser Zeit mit dem Befehl SOUND 1,110,0 ausschalten.

Auf dem Umrechnungsprogramm aufbauend, können wir sehr leicht ein Programm entwickeln, daß es uns erlaubt, die Commodore-Tastatur in eine Art Klavier-Tastatur zu verwandeln. Die Funktionsweise ist allerdings sehr vereinfacht: Durch Betätigen einer der Tasten A, B, C, D, E, F oder G soll der entsprechende Ton gespielt werden. Mit der Taste CURSOR OBEN kann jederzeit eine höhere, mit CURSOR UNTEN eine niedrigere Oktave angewählt werden.

```
100 DIM N$(7):DIM FR(7):REM ARRAYS DIMENSIONIEREN
110 N$(1)="A":FR(1)=110
120 N$(2)="H":FR(2)=123.5
130 N$(3)="C":FR(3)=130.8
140 N$(4)="D":FR(4)=146.8
150 N$(5)="E":FR(5)=164.7
160 N$(6)="F":FR(6)=174.5
170 N$(7)="G":FR(7)=195.9
180 :
190 VOL 8:OK=1:REM VOLLE LAUTSTAERKE UND AUSGANGSOKTAVE 1
200 :
210 REM TASTATUR ABFRAGEN
220 GETKEY N$
230 IF N$<>CHR$(17) AND N$<>CHR$(145) THEN GOTO 290
240 IF N$=CHR$(17) AND OK>1 THEN OK=OK-1
250 IF N$=CHR$(145) AND OK>4 THEN OK=OK+1
260 GOTO 220
270 :
280 REM TABELLE DURCHSUCHEN
290 FOR I=1 TO 7
300 : IF N$=N$(I) THEN FR=FR(I):GOTO 340
310 NEXT I
320 :
330 REM BERECHNUNG
340 IF OK=1 THEN 390:REM 1.OKTAVE?
350 FOR I=1 TO OK-1
360 : FR=FR*2:REM FREQUENZ VERDOPPELN
370 NEXT I
380 :
390 TH=1024-111840.45/FR:REM TONHOEHE BERECHNEN
400 SOUND 1,TH,10
410 GOTO 220
```

Der Programmablauf: In Zeile 190 wird die volle Lautstärke eingeschaltet und die Oktave eins festgelegt. In Zeile 220 wird auf eine Taste gewartet und das jeweilige Zeichen der Variablen N$ zugewiesen. Entspricht die Taste weder den Zeichen für CURSOR UNTEN (Code 17) noch CURSOR OBEN (Code 145), wird sofort Zeile 290 angesprungen.

Ab Zeile 290 folgt der bekannte Vergleich der gedrückten Taste mit den im Array N$(...) enthaltenen Noten-Buchstaben. Fällt ein Vergleich positiv aus, wird die zugehörige Frequenz der Variablen FR zugewiesen.

Anschließend wird die entsprechende Tonhöhe TH berechnet und es folgt die erste wirkliche Neuerung, der Befehl SOUND 1,TH,10. Mit diesem Befehl wird Stimme 1 angewiesen, einen Ton in der errechneten Tonhöhe TH etwa 0.2 Sekunden lang zu spielen (10/50=0.2). Der folgende Befehl bewirkt den Rücksprung zur "Eingabe-Routine".

Wurde eine der beiden Cursortasten betätigt, wird die Oktave OK entsprechend erniedrigt beziehungsweise erhöht. Erniedrigt wird die Oktave jedoch nur, wenn sie noch nicht den niedrigsten Wert 1 erreicht hat (240...AND OK>1...), erhöht wird sie nur, wenn die größte Oktave 4 noch nicht erreicht ist (250...AND OK<4...).

Bedienungshinweise

1. Die Töne A, B, C, D, E, F, G werden durch Betätigen der entsprechenden Buchstabentasten gespielt.
2. Mit der Taste CURSOR UNTEN wird die Oktave verringert, mit CURSOR OBEN erhöht.

3.4 Dateiverwaltung

3.4.1 Grundlagen der Dateiverwaltung

Oftmals sollen vom Benutzer eingegebene oder berechnete Daten dauerhaft auf einem "Massenspeicher" (Cassette oder Diskette) festgehalten werden.

Die Befehle DSAVE, SAVE, DLOAD und LOAD erlauben zwar die Speicherung und das Laden von Programmen, jedoch nicht von Daten. Daten werden meist in Arrays (String-, Fließkomma- oder Integerarray) im Speicher gehalten und gehen verloren, sobald der Computer ausgeschaltet wird.

Nun ist es jedoch ein Unding, zum Beispiel in einer Adressenverwaltung mühsam 50 oder 60 Adressen einzugeben, nur um diese nach dem Ausschalten sofort wieder zu verlieren. Adressenverwaltungen oder andere Programme, die mit großen Datenmengen umgehen, können nur sinnvoll eingesetzt werden, wenn wir wissen, wie die Inhalte von Variablen ebenso wie Programme selbst gespeichert und wieder geladen werden.

Der C16, C116 oder Plus/4 verwendet zum Datenaustausch mit "Peripheriegeräten" sogenannte "logische Dateien". Um Daten an ein Peripheriegerät wie die Floppy oder die Datasette zu übergeben, muß zuerst eine logische Datei "geöffnet" werden. In eine offene Datei können Daten geschrieben oder daraus gelesen werden. Nach Abschluß des Schreib- beziehungsweise Lesevorgangs wird die Datei "geschlossen" und der Datenaustausch ist beendet.

Peripheriegeräte sind die Floppy, die Datasette, der Drucker, der Bildschirm und die Tastatur. Wie Sie wissen, können Daten zwischen dem Rechner und Bildschirm beziehungsweise Tastatur - zum Beispiel mit PRINT oder INPUT - ausgetauscht werden, ohne eine Datei zu öffnen und zu schliessen.

Bildschirm und Tastatur sind sogenannte "Standardgeräte". Ohne weitere Angabe spricht ein PRINT-Befehl immer das Standardausgabegerät Bildschirm an. Der BASIC-Interpreter sorgt automatisch für das Öffnen und Schließen der benötigten logischen Datei, ohne daß sich der Benutzer um diesen Vorgang kümmern muß.

Ebenso wie der Bildschirm das Standardausgabegerät ist, ist die Tastatur das Standardeingabegerät. Ein Eingabebefehl wie INPUT oder GET bezieht sich ohne weitere Angabe eines Gerätes immer auf die Tastatur.

Sollen Daten mit einem anderen als einem Standardgerät ausgetauscht werden, müssen wir uns um alle "Formalitäten" kümmern, die sonst der BASIC-Interpreter übernimmt. Der erste Schritt besteht im Öffnen einer logischen Datei.

3.4.1.1 Öffnen einer logischen Datei

Das Öffnen einer Datei ist ein recht komplexer Vorgang. Der zugehörige Befehl ist entsprechend umfangreich.

```
OPEN (FILENUMMER),(GERÄTEADRESSE) [,(SEKUNDÄRADRESSE,
"FILENAME,FILETYP,MODUS")]
```

1. FILENUMMER: Die sogenannte "logische Filenummer" (abgekürzt: LFN) oder auch einfach "Filenummer" ist eine Angabe, die den späteren Bezug auf die Datei vereinfachen soll. "File" ist der englische Ausdruck für "Datei". Der zu öffnenden Datei wird eine beliebige Nummer zwischen 1 und 255 gegeben. Sollen später Daten in eine Datei geschrieben oder daraus gelesen werden, muß dem BASIC-Interpreter mitgeteilt werden, welche Datei gemeint ist, da mehrere Dateien gleichzeitig geöffnet werden können. Diese Angabe der angesprochenen Datei geschieht mit der jeweiligen Filenummer.

2. GERÄTEADRESSE: Da ein Datenaustausch zwischen dem C16, C116 oder Plus/4 und den verschiedensten Geräten möglich ist, muß angegeben werden, zu welchem Gerät Daten gesendet beziehungsweise von welchem Gerät Daten gelesen werden sollen. Jedes Gerät besitzt eine eigene "Gerätenummer", die im OPEN-Befehl anzugeben ist.

   Datasette: 1
   Floppy: 8
   Drucker: 4
   Bildschirm: 3
   Tastatur: 0

3. SEKUNDÄRADRESSE: Die "Sekundäradresse" oder auch kurz "SA" ist eine Zahl zwischen 0 und 15, die jedoch im Gegensatz zur Filenummer nicht beliebig

ist. Die Bedeutung der Sekundäradresse ist unterschiedlich, je nachdem, welches Gerät angesprochen wird.

Datasette (SA: 0 bis 2): Die Sekundäradresse 0 teilt der Datasette mit, daß Daten gelesen werden sollen. Mit der Sekundäradresse 1 wird mitgeteilt, daß Daten auf das Band geschrieben werden. Sekundäradresse 2 schreibt Daten mit einer sogenannten "EOT-Markierung" auf Band, die noch beschrieben wird.

Floppy (SA: 2 bis 14): Bei Verwendung der Floppy kann die Sekundäradresse zwischen 2 und 14 beliebig gewählt werden. Eine besondere Bedeutung besitzt die Sekundäradresse 15. Sie wird benötigt, um der Floppy spezielle Befehle zu übermitteln, zum Beispiel "Lösche die Datei XYZ" oder "Kopiere die Datei XYZ".

Drucker: Die Bedeutung der Sekundäradresse beim Datenaustausch mit einem Drucker ist von Drucker zu Drucker unterschiedlich und müssen Sie im jeweiligen Druckerhandbuch nachlesen. Ein Ausdruck von Daten ist jedoch meist ohne Angabe einer Sekundäradresse möglich, die - wie Sie an der Syntax erkennen - eine optionale Angabe ist.

Bildschirm und Tastatur: Beim Datenaustausch mit Bildschirm oder Tastatur entfällt die Angabe einer Sekundäradresse.

4. FILENAME: Der "Filename" oder "Dateiname" ist ebenso wie die Sekundäradresse eine optionale Angabe, die nicht immer benötigt wird. Unbedingt notwendig ist diese Angabe beim Austausch von Daten mit einem Diskettenlaufwerk. Einer Disketten-Datei muß (!) ein Name gegeben werden, ebenso wie es nötig ist, einem auf Diskette zu speichernden Programm einen Namen zu geben. Ein

Filename darf aus maximal 16 Zeichen bestehen. Die Zeichen "?", "*" und das Anführungszeichen sind nicht zulässig. Beim Datenaustausch mit der Datasette können Sie einen Filenamen angeben, Sie müssen jedoch nicht.

5. FILETYP (S, P, U, R): Die Datasette kann nur sogenannte "sequentielle Dateien" verwalten, im Gegensatz zur Floppy, die verschiedene Dateitypen verwaltet, "sequentielle Dateien", "relative Dateien", "Programm-Dateien" und "User-Dateien". Aufgrund dieser Vielfalt muß der Floppy beim Datenaustausch angegeben werden, welcher Dateityp geöffnet werden soll. In diesem Buch interessieren uns nur sequentielle Dateien. Das Kennzeichen für diesen Dateityp ist der Buchstabe "S".

6. MODUS (R, W, A): Auch diese Angabe interessiert nur bei Verwendung der Floppy. Der Datasette wird mit der Sekundäradresse mitgeteilt, ob Daten geschrieben oder aber gelesen werden sollen. Diese Angabe erfolgt bei der Floppy mit einem Buchstaben, der den Modus kennzeichnet. Der Buchstabe "R" steht für "Read" (=Lesen), "W" für "Write" (=Schreiben) und "A" für "Append" (=Anhängen).

Beispiele:

1. OPEN 1,1,0 öffnet eine Datei zum Lesen von Daten von der Datasette mit der logischen Filenummer 1 (Geräteadresse 1: Datasette; Sekundäradresse 0: Lesen von Daten)
2. OPEN 1,1,1 öffnet eine Datei zum Schreiben von Daten auf Cassette.
3. OPEN 2,8,3,"ADRESSEN,S,R" öffnet eine Datei mit der LFN 2 und der SA 3 zum Lesen von Daten von Diskette. Die Datei, aus der Daten gelesen werden sollen, ist eine sequentielle Datei und besitzt den Namen "ADRESSEN".

4. OPEN 2,8,2,"ADRESSEN,S,W" öffnet eine sequentielle Disketten-Datei mit dem Namen "ADRESSEN", der LFN 2 und der SA 2 zum Schreiben von Daten.
5. OPEN 4,4 öffnet eine Drucker-Datei. Drucker-Dateien können verständlicherweise nur zu Schreibzwecken geöffnet werden, wodurch die explizite Angabe des Modus entfällt.

3.4.1.2 Schreiben und Lesen von Daten

Nachdem eine logische Datei mit OPEN geöffnet wurde, können Daten mit den Ein-/Ausgabebefehlen in die Datei geschrieben oder daraus gelesen werden. Die Syntax dieser Befehle unterscheidet sich durch die Angabe der logischen Filenummer jedoch leicht von der gewohnten Syntax der Befehle PRINT, INPUT und GET.

```
PRINT#(LFN),(STRING)
INPUT#(LFN),(VARIABLE)
GET#(LFN),(VARIABLE)
```

Abgesehen von dieser zusätzlichen Angabe kann mit diesen Befehlen wie gewohnt umgegangen werden. PRINT# dient zum Schreiben von Daten und INPUT# beziehungsweise GET# zum Lesen aus einer Datei. Daten können niemals aus einer geöffneten Datei abwechselnd gelesen und geschrieben werden. Wurde die Datei zum Schreiben geöffnet, dürfen Daten ausschließlich mit PRINT# in die Datei geschrieben werden, wurde sie zum Lesen geöffnet, dürfen Daten ausschließlich mit INPUT# oder GET# gelesen werden.

3.4.1.3 Schließen einer logischen Datei

Nach der Ausführung der gewünschten "Schreib-" beziehungsweise "Lesezugriffe" muß die logische Datei mit dem Befehl CLOSE unter Angabe der Filenummer geschlossen werden.

```
CLOSE (LFN)
```

Beispiele:

1. CLOSE 2 schließt eine unter der Filenummer 2 geöffnete Datei.
2. CLOSE 1 schließt eine unter der Filenummer 1 geöffnete Datei.

3.4.1.4 Datenaustausch mit Floppy und Datasette

Theoretisch kennen Sie nun alle Befehle, die zum Datenaustausch mit Peripheriegeräten benötigt werden. In der Praxis tauchen jedoch bei der Arbeit mit Dateien häufig Schwierigkeiten auf. Der Umgang mit Dateien wird daher im folgenden anhand zahlreicher Demoprogramme erläutert. Das erste dieser Demoprogramme schreibt Daten in eine Datei auf Diskette oder Cassette.

```
100 INPUT "CASSETTE(C) ODER DISKETTE(D)";GE$
110 IF GE$="C" THEN OPEN 2,1,1:ELSE OPEN 2,8,2,"TEST,S,W"
120 PRINT#2,"DIES IST":REM STRING UNMITTELBAR SCHREIBEN
130 A$="EIN TEST"
140 PRINT#2,A$:REM STRINGVARIABLE SCHREIBEN
150 PRINT#2,1,2,3,4,5:REM ZAHLEN UNMITTELBAR SCHREIBEN
160 A=6:B=7:C=8:D=9:E=10
170 PRINT#2,A,B,C,D,E:REM NUMERISCHE VARIABLEN SCHREIBEN
180 CLOSE 2:REM DATEI SCHLIESSEN
```

Das Programm fragt den Benutzer, ob die Daten auf Cassette oder Diskette geschrieben werden sollen. Lautet die Antwort "C" für "Cassette", wird der THEN-Zweig des folgenden IF-Befehls ausgeführt (Zeile 110). In diesem Zweig befindet sich ein

OPEN-Befehl, der eine Datasetten-Datei (Gerätenummer 1) mit der logischen Filenummer 2 zum Schreiben (Sekundäradresse 1) öffnet.

Soll hingegen eine Disketten-Datei geöffnet werden, wird der ELSE-Zweig ausgeführt. Eine Disketten-Datei (Gerätenummer 8) mit der logischen Filnenummer 2 und der Sekundäradresse 2 wird unter dem Namen "TEST" zum Schreiben (Modus W) geöffnet. Der angegebene Typ "S" kennzeichnet eine sequentielle Datei, den einzigen Dateityp, den wir behandeln wollen.

Nach dem Öffnen der Datei werden Daten in den verschiedensten Varianten in die Datei geschrieben. In Zeile 120 wird analog dem PRINT-Befehl PRINT "DIES IST" mit PRINT#2,"DIES IST" der String "DIES IST" unter Angabe der Filenummer in die geöffnete Datei geschrieben.

Ebenso wie mit PRINT die Ausgabe von Variablen möglich ist, wird in Zeile 140 die zuvor definierte Stringvariable A$ mit dem Inhalt "EIN TEST" in die Datei geschrieben.

In den Zeilen 150 bis 170 werden die Zahlen 1 bis 5 einmal direkt (Zeile 150) und einmal als Inhalte von Variablen in die Datei geschrieben. Die - in diesem Fall auf die Datei - auszugebenden Zahlen werden wie bereits bekannt mit Kommas "formatiert".

Dieses Demoprogramm zeigt Ihnen alle Möglichkeiten, die Sie beim Schreiben von Daten in eine Datei besitzen. Es können sowohl Zeichenketten als auch Zahlen auf eine Datei ausgegeben werden, die Angaben können unmittelbar oder in Form von Variablen erfolgen. Auch die Formatierung mehrerer Zahlen oder Zeichenketten mit dem Komma oder Semikolon kann wie gewohnt verwendet werden, jedoch mit einem anderen Ziel: Bei der Ausgabe auf dem Bildschirm dienen die Formatierungszeichen vorwiegend der optischen Gliederung von Tabellen. Bei der Ausgabe auf eine Datei ist ein Begriff wie "optische Gliederung" sinnlos. Der Vorteil liegt darin, daß mehrere Daten

mit einem Befehl ausgegeben werden können und Sie Tipparbeit einsparen.

Nach dem Starten dieses Programms wird die Datei entweder sofort auf der eingelegten (!) Diskette gespeichert oder aber - beim Speichern auf Cassette - erscheint die Meldung "PRESS PLAY AND RECORD ON TAPE". Diese Meldung fordert Sie auf, gleichzeitig die Tasten RECORD und PLAY an Ihrer Datasette zu betätigen. Die Daten werden gespeichert, wenn Sie der Aufforderung nachgekommen sind.

Die Daten befinden sich nun in einer Datei auf Diskette oder Cassette und sollen wieder eingelesen werden. Diese Aufgabe übernimmt das folgende Programm.

```
100 INPUT "CASSETTE(D) ODER DISKETTE(D)";GE$
110 IF GE$="C" THEN OPEN 2,1,0:ELSE OPEN 2,8,2,"TEST,S,R"
120 INPUT#2,A$:REM STRING EINLESEN
130 PRINT A$:REM GELESENEN STRING AUF BILDSCHIRM
140 INPUT#2,B$:REM STRING EINLESEN
150 PRINT B$:REM GELESENEN STRING AUF BILDSCHIRM
160 INPUT#2,A,B,C,D,E:REM ZAHLEN EINLESEN
170 PRINT A,B,C,D,E:REM GELESENE ZAHLEN AUF BILDSCHIRM
180 INPUT#2,F,G,H,I,J:REM ZAHLEN EINLESEN
190 PRINT F,G,H,I,J:REM GELESENE ZAHLEN AUF BILDSCHIRM
200 CLOSE 2:REM DATEI SCHLIESSEN
```

In diesem Demoprogramm sind fast alle zum Lesen von Daten vorhandenen Möglichkeiten vorhanden. Je nach Benutzereingabe wird eine Cassetten- oder eine Disketten-Datei geöffnet. Bei der Cassetten-Datei wird als logische Filenummer eine 2, als Gerätenummer 1 und - da Daten gelesen werden - als Sekundäradresse eine 0 angegeben. Beim Öffnen der Disketten-Datei wird die Filenummer 2, die Geräteadresse 8 und die Sekundäradresse 2 verwendet. Denken Sie daran, daß die Filenummer im Bereich 0 bis 255 beliebig gewählt werden kann. Statt zwei kann eine beliebige andere Filenummer verwendet werden, wenn in den folgenden INPUT-Befehlen Bezug auf die neue Filenummer genommen wird.

Äußerst wichtig ist beim Öffnen der Disketten-Datei die Angabe des Datei-Namens. Es muß der Name jener Datei angegeben werden, in der sich die zu lesenden Daten befinden, das heißt der beim Schreiben verwendete Name "TEST". Der Dateityp lautet "S", da es sich wie erwähnt um eine sequentielle Datei handelt. Als Modus wird der Buchstabe "R" angegeben, da anschließend Daten aus der Datei gelesen werden.

Mit dem Befehl INPUT#2,A$ wird die erste in die Datei geschriebene Zeichenkette "DIES IST EIN TEST" in die Variable A$ eingelesen. Nun fragen Sie sich sicherlich, wie der INPUT-Befehl bei Anwendung auf eine Datei (INPUT#) funktioniert. Wie wir wissen, wird eine von INPUT angeforderte Tastatureingabe erst angenommen, wenn wir die RETURN-Taste betätigen. Datasette und Floppy beherbegen jedoch keine "kleinen Männchen", die einen solchen Tastendruck vornehmen. Woran erkennt der C16, C116 oder Plus/4 das Ende der Eingabe, das heißt das Ende der eingelesenen Zeichenkette?

Zeichenketten und Zahlen werden mit PRINT#-Befehlen auf ein Peripheriegerät ausgegeben. Nach jedem PRINT#-Befehl "sendet" der C16, C116 bzw. Plus/4 automatisch das Zeichen CHR$(13), das der RETURN-Taste entspricht. Dieses Zeichen bildet den Abschluß jedes PRINT#-Befehls und wird auf Diskette beziehungsweise Band automatisch an die mit PRINT# übermittelte Zeichenkette "angehängt".

Wird nun mit INPUT# eine Zahl oder eine Zeichenkette aus einer Datei gelesen, wird Zeichen für Zeichen gelesen, bis das Zeichen CHR$(13) eingelesen wird. Der Interpreter erkennt an dieser "Endemarke", daß die Zeichenkette oder Zahl komplett eingelesen wurde und weist sie der im INPUT#-Befehl angegebenen Variablen zu.

Die Zeilen 120 bis 140 lesen die beiden Zeichenketten "DIES IST" und "EIN TEST" ein und geben Sie anschließend mit PRINT auf dem Bildschirm aus. Das Ende dieser Zeichenketten wird wie erläutert anhand der beim Schreiben automatisch angehängten Endmarke CHR$(13) erkannt.

Problematischer ist der Ablauf der Zeilen 160 bis 190, die die Zahlen 1 bis 10 einlesen und auf dem Bildschirm ausgeben. Es wäre möglich gewesen, diese Zahlen ebenso wie die Zeichenketten mit einzelnen PRINT#-Befehlen in die Datei zu schreiben und mit einzelnen INPUT#-Befehlen daraus zu lesen.

```
PRINT#2,1
PRINT#2,2
...
...
PRINT#2,A
PRINT#2,B
...
...
PRINT#2,E
```

und

```
INPUT#2,A
INPUT#2,B
...
...
INPUT#2,J
```

Der Unterschied zur gewählten Lösung besteht im Schreibaufwand. Mit einem PRINT#-Befehl wurden fünf Zahlen gleichzeitig in die Datei geschrieben und mit einem INPUT#-Befehl fünf Zahlen gleichzeitig gelesen. Sowohl beim Schreiben als auch beim Lesen wurden die einzelnen Zahlen beziehungsweise Variablen mit Kommas voneinander getrennt.

Das Trennzeichen ist nun jedoch ein Komma und kein CHR$(13). Der Befehl PRINT#2,1,2,3,4,5 schreibt fünf Zahlen in eine Datei. Hinter den vier ersten Zahlen befindet sich ein Komma. Dieses Komma wird anstelle eines CHR$(13) als Trennzeichen auf Band oder Diskette geschrieben. Hinter der fünften Zahl befindet sich kein Komma. Dieser Zahl folgt daher wie üblich das Standard-Trennzeichen CHR$(13).

Nun ergibt sich ein Problem. Angenommen, Sie wollen die fünf Zahlen mit fünf einzelnen INPUT#-Befehlen einlesen (INPUT#,A:INPUT#,B:...:INPUT#,E). Der Standard-INPUT#-Befehl erwartet ein CHR$(13) als Endemarke. Hinter der ersten Zahl befindet sich jedoch stattdessen ein Komma. Der Interpreter wartet daher vergeblich auf die Endemarke CHR$(13) und die Zahl eins wird nicht korrekt eingelesen, ebensowenig wie die folgenden Zahlen.

Merken Sie sich daher: Zahlen und Zeichenketten müssen unbedingt im gleichen Format gelesen werden, wie sie gespeichert wurden. Wurden beim Speichern Kommata als Trennzeichen angegeben, müssen diese Kommata auch beim Lesen angegeben werden.

Nun ist auch verständlich, warum die Zahlen mit dem Befehl INPUT#2,A,B,C,D,E korrekt gelesen werden. Hinter den ersten vier angegebenen Variablen befindet sich als Trennzeichen jeweils ein Komma. Genau dieses Komma erwartet der INPUT#-Befehl beim Einlesen dieser Zahlen, die ja auch tatsächlich mit diesem Trennzeichen gespeichert wurden. Hinter der fünften im INPUT#-Befehl angegebenen Variablen E befindet sich kein Komma und der Interpreter erwartet beim Einlesen das Standard-Trennzeichen CHR$(13), das hinter dieser Zahl vorhanden ist (PRINT#2,1,2,3,4,5).

Sollte Ihnen die Wirkungsweise der Trennzeichen noch nicht völlig klar sein, vergleichen Sie den INPUT#-Befehl am besten mit dem INPUT-Befehl.

Wir sahen bereits früher, daß mit einem INPUT-Befehl mehrere Eingaben des Benutzers verschiedenen Variablen zugewiesen werden konnten (INPUT A$,B$,C$,...), wenn der Benutzer seine Eingaben ebenfalls mit Kommas voneinander trennte (?DIES,IST,EIN,...). Auch bei der Tastatureingabe mußte daher vom Benutzer das im INPUT-Befehl angegebene Trennzeichen angegeben werden, entweder ein Komma oder aber die Standard-Endmarke RETURN (=CHR$(13)).

Wenn Sie sich Schwierigkeiten mit gemischt verwendeten Trennzeichen ersparen wollen, empfehle ich Ihnen, pro zu speichernde Zeichenkette oder Zahl einen PRINT#-Befehl zu verwenden und Zahlen beziehungsweise Zeichenketten ebenfalls einzeln mit je einem INPUT#-Befehl wieder einzulesen, das heißt immer die automatisch nach jedem PRINT#-Befehl gesandte Endemarke CHR$(13) zu verwenden.

Mit dieser Methode ersparen Sie sich zwar keine Tipparbeit, sie ist jedoch weit sicherer als die gemischte Verwendung von Trennzeichen.

Zeile 200 wurde noch nicht besprochen. Sie ist jedoch problemlos und enthält als einzigen Befehl CLOSE 2, mit dem die geöffnete Datei nach Abschluß der Lesezugriffe unter Angabe der Filenummer geschlossen wird.

3.4.1.5 Lesen mit GET#

Sie werden sich fragen, welche Bedeutung der Befehl GET# beim Lesen von Daten aus logischen Dateien besitzt. GET# liest ein einzelnes Zeichen aus einer geöffneten Datei und wird verwendet, wenn die gespeicherten Zeichen eines der Sonderzeichen Komma, Semikolon oder Anführungszeichen enthalten.

Diese Sonderzeichen können mit INPUT# leider nicht gelesen werden beziehungsweise führen zu Fehlern beim Lesen von Daten (analog der Tastatureingabe mit INPUT). Vor allem beim Komma ist diese Problematik verständlich. Das Komma ist ein Trennzeichen zwischen mehreren Daten und wird daher beim Lesen aus logischen Dateien nicht wie ein beliebiges sonstiges Zeichen behandelt.

Oftmals kann auf die Verwendung dieser Sonderzeichen verzichtet werden. Ist ihre Verwendung jedoch unumgänglich (zum Beispiel in einer Textverarbeitung), müssen die Daten statt mit INPUT# mit GET# gelesen werden. Wie wir bereits von der Tastatur her wissen, liest GET ein Zeichen ein, ohne daß eine Endemarke wie das Drücken der RETURN-Taste benötigt wird.

Entsprechendes gilt für GET# und das Lesen aus logischen Dateien. GET# liest ein Zeichen aus einer geöffneten Datei. Trennzeichen wie CHR$(13) oder das Komma werden wie jedes andere Zeichen auch behandelt. Das folgende Demoprogramm demonstriert den Einsatz von GET#.

```
100 OPEN 2,8,2,"TEST1,S,W":REM DISK-DATEI OEFFNEN
110 PRINT#2,"MAIER, ANTON"
120 PRINT#2,"MUELLER, HEINRICH"
130 CLOSE 2:REM DATEI SCHLIESSEN
140 :
150 OPEN 2,8,2,"TEST1,S,R":REM DISK-DATEI OEFFNEN
160 GOSUB 230:REM STRING 1 EINLESEN
170 PRINT A$:REM STRING 2 AUSGEBEN
180 GOSUB 230:REM STRING 2 EINLESEN
190 PRINT A$:REM STRING 2 AUSGEBEN
200 END:REM PROGRAMMENDE
210 :
220 REM STRING MIT GET# LESEN
230 A$="":REM A$ LOESCHEN
240 Z$="":REM Z$ LOESCHEN
250 DO UNTIL Z$=CHR$(13)
260 : GET#2,Z$:REM ZEICHEN LESEN
270 : A$=A$+Z$:REM AN Z$ ANHAENGEN
280 LOOP
290 RETURN
```

Dieses Programm schreibt die zwei Zeichenketten "MAIER, ANTON" und "MUELLER, HEINRICH" in eine Disketten-Datei. Pro Zeichenkette wird ein PRINT#-Befehl verwendet, daher wird jede Zeichenkette mit CHR$(13) beendet.

Nachdem die zum Schreiben (Modus "W") geöffnete Datei "TEST1" geschlossen wurde, wird Sie erneut - jedoch für Lesezugriffe (Modus R) - geöffnet. Die beiden Strings werden durch zweimaligen Aufruf (Zeile 160 und 180) gelesen und auf dem Bildschirm ausgegeben.

Zeile 200 enthält einen für uns neuen Befehl, END. Trifft der BASIC-Interpreter auf den Befehl END, wird der Programmablauf beendet, auch wenn weitere Programmzeilen folgen. Dieser Befehl ist im vorliegenden Programm sehr nützlich, um zu verhindern, daß nach dem Durchlaufen des Hauptprogramms und erfolgtem Lesen beider Zeichenketten der BASIC-Interpreter mit einer dritten Bearbeitung des Unterprogramms (das ja dem Hauptprogramm folgt) fortfährt. Eine solche dritte Bearbeitung führt zur Fehlermeldung "RETURN WITHOUT GOSUB", da der Interpreter in der letzten Programmzeile auf den Befehl RETURN trifft, jedoch ohne daß ein dritter Aufruf des Unterprogramms mit dem Befehl GOSUB 230 erfolgte.

Doch nun zum eigentlich interessanten Programmteil, dem Einlesen einer Zeichenkette mit GET#. Ziel des Unterprogramms ist es, eine Zeichenkette einzulesen, deren Endemarke das Standard-Kennzeichen CHR$(13) bildet. Erst wenn dieses Zeichen erkannt wird, soll der Lesevorgang beendet werden. Alle übrigen Zeichen - also auch Kommata, Semikola und Anführungszeichen - sollen unterschiedslos gelesen werden.

Das Prinzip: Zeichen für Zeichen des zu lesenden Strings werden mit GET# gelesen und der Variablen Z$ zugewiesen. Jedes gelesene Zeichen wird an den String A$ angehängt (Stringaddition). Der Lesevorgang wird beendet, wenn das gelesene Zeichen mit CHR$(13) identisch und daher die Standard-Endemarke erreicht ist.

In den Zeilen 230 und 240 werden eventuelle alte Inhalte der Stringvariablen A$ und Z$ gelöscht. Anschließend folgt die erste praktische Nutzung der Schleifenkonstruktion mit DO UNTIL ... LOOP.

Die Befehle innerhalb der Schleife (Zeile 260 und 270, das Lesen eines Zeichens und Anhängen an den "Summenstring" A$) werden solange ausgeführt, bis die Bedingung UNTIL Z$=CHR$(13) erfüllt ist, bis das gelesene Zeichen mit der Endemarke CHR$(13) identisch ist.

Dieses Unterprogramm können Sie jederzeit in Ihren eigenen Programmen verwenden, wenn Sie vor dem Problem stehen, Zeichenketten mit den erwähnten Sonderzeichen zu lesen. Beachten Sie bitte, daß die Routine ausschließlich Zeichenketten mit der Endemarke CHR$(13) korrekt liest. Sie dürfen daher keinesfalls mehrere durch Kommata getrennte Zeichenketten mit einem PRINT#-Befehl speichern.

Datasetten-Besitzer warten sicherlich bereits sehnsüchtig auf die zum Öffnen einer Datasetten-Datei angepassten OPEN-Befehle.

```
100 OPEN 2,1,1:REM DATASETTEN-DATEI OEFFNEN
150 OPEN 2,1,0:REM DATASETTEN-DATEI OEFFNEN
```

Mehrere Angaben entfallen (Filenamens, Filetyps, Modus). Die Geräteadresse 8 wird durch 1 ersetzt und der Modus (Schreiben/Lesen) wird mit der Sekundäradresse (0=Lesen; 1=Schreiben) übermittelt.

3.4.1.6 Sequentielle Dateien

Der Ausdruck "Sequentielle Datei" fiel in diesem Kapitel sehr oft, ohne daß eine nähere Erläuterung dieses Begriffs stattfand. Eine sequentielle Datei ist eine Datei, in der Daten nahtlos aneinander gereiht sind und auf die nur in der Reihenfolge der Speicherung zugegriffen werden kann.

Die Demoprogramme veranschaulichen diese Arbeitsweise. Alle Daten werden exakt in jener Reihenfolge gelesen, in der sie gespeichert wurden. Es ist nicht möglich, direkt auf eine Zeichenkette oder Zahl mitten in der Datei zuzugreifen. Sequentielle Dateien entsprechen einem Band, auf dem die Daten hintereinander aufgereiht sind.

GERHARD/OTTO/WILLI/STEFAN/WERNER

Soll die dritte in diesem Beispiel verwendete Zeichenkette ("WILLI") gelesen werden, so müssen drei INPUT#-Befehle verwendet werden. Der erste Befehl liest die Zeichenkette

"GERHARD", der zweite liest "OTTO", und erst der dritte INPUT#-Befehl liest den gewünschten String "WILLI" ein.

Da Speichern und Lesen von Daten in der gleichen Reihenfolge stattfinden, sind die Programmteile zum Schreiben und Lesen von der Struktur her praktisch identisch.

Schreiben von Daten

```
OPEN ....:REM DATEI ZUM SCHREIBEN OEFFNEN
PRINT#(LFN),DATUM 1
PRINT#(LFN),DATUM 2
PRINT#(LFN),DATUM 3
...
...
...
PRINT#(LFN),DATUM N
CLOSE (LFN):REM DATEI SCHLIESSEN
```

Lesen von Daten

```
OPEN ....:REM DATEI ZUM LESEN OEFFNEN
INPUT#(LFN),VARIABLE 1
INPUT#(LFN),VARIABLE 2
INPUT#(LFN),VARIABLE 3
...
...
...
INPUT#(LFN),VARIABLE N
CLOSE (LFN):REM DATEI SCHLIESSEN
```

Mit ihrem C16, C116 oder Plus/4 ist - jedoch nur mit einem Diskettenlaufwerk - auch ein direkter Zugriff auf Daten möglich, wenn "relative" Dateien verwendet werden. Der Umgang mit relativen Dateien ist jedoch sehr anspruchsvoll. Wenn Sie den Umgang mit sequentiellen Dateien perfekt beherrschen und über eine Floppy verfügen, finden Sie im zugehörigen Floppy-Handbuch die zur Verwaltung relativer Dateien benötigten Befehle.

3.4.1.7 Speichern und Lesen von Arrays

In Dateiverwaltungen werden meist umfangreiche Datenmengen mit Hilfe von Arrays verwaltet. Zum Speichern und Einlesen eines großen Arrays werden jedoch keineswegs entsprechend viele PRINT#- und INPUT#-Befehle benötigt. Das folgende Demoprogramm gestattet dem Benutzer die Eingabe - fast - beliebig vieler Adressen, speichert die Adressen auf Band oder Diskette und liest sie wieder ein.

```
100 DIM AD$(50):REM ARRAY DIMENSIONIEREN
110 DO UNTIL AD$(IZ)="ENDE"
120 : INPUT "ADRESSE";AD$(IZ)
130 : IZ=IZ+1:REM INDEXZAHL ERHOEHEN
140 LOOP
150 IZ=IZ-1:REM ZAEHLER KORRIGIEREN
160 INPUT "CASSETTE(C) ODER DISKETTE(D)";GE$
170 :
180 REM SPEICHERN
190 IF GE$="C" THEN OPEN 2,1,1:ELSE OPEN 2,8,2,"TEST,S,W"
200 FOR I=0 TO IZ
210 : PRINT#2,AD$(I)
220 NEXT I
230 CLOSE 2
240 :
250 REM LESEN
260 IF GE$="C" THEN OPEN 2,1,0:ELSE OPEN 2,8,2,"TEST,S,R"
270 FOR I=0 TO IZ
280 : INPUT#2,AD$(I)
290 : PRINT AD$(I)
300 NEXT I
310 CLOSE 2
```

Der Programmablauf: Nachdem das Array A$(50) dimensioniert wurde, nimmt das Programm vom Benutzer Adresse nach Adresse entgegen und speichert sie in den Arrayvariablen A$(0), A$(1), A$(2) und so weiter. Nach jeder Eingabe wird die Indexzahl IZ erhöht. Die Schleife wird verlassen, wenn der Benutzer

die Zeichenkette "ENDE" eingibt und die Bedingung UNTIL AD(IZ)="ENDE" erfüllt ist.

Wurden drei Adressen eingegeben und lautete die vierte Eingabe "ENDE", besitzt die Variable IZ nach dem Verlassen der Schleife den Wert 3. Der Wert dieser Variablen wird um eins vermindert, da diese Variable die Indexzahl der letzten zu speichernden Adresse angeben soll, das heißt der letzten Eingabe vor "ENDE".

Beispiel für drei Eingaben

AD$(0): OTTO
AD$(1): MAIER
AD$(2): WILLI
AD$(3): ENDE

IZ besitzt in diesem Beispiel zuerst den Wert 3, nach der Verminderung um eins den Wert 2 und "zeigt" damit auf den letzten zu speichernden String "WILLI".

Die Arrayvariablen werden in einer Schleife gespeichert. Der Startwert ist 0 (PRINT#2,AD$(0)), der Endwert IZ (PRINT#2,AD$(IZ), im Beispiel: PRINT#2,AD$(2)).

Gelesen werden die Daten ebenfalls mit einer Schleife und identischen Start- und Endwerten. Die Reihenfolge des Lesens in die Arrayvariablen A$(0) bis A$(2) erfolgt in der beim Speichern verwendeten Reihenfolge.

3.4.1.8 Die Statusvariable ST

Oftmals stellt sich das Problem, daß aus einer Datei Daten gelesen werden sollen, die Anzahl der darin enthaltenen Daten jedoch nicht bekannt ist. Eine FOR-Schleife mit festem Endwert kann in diesen Fällen nicht verwendet werden.

Der C16, C116 bzw. Plus/4 verfügt jedoch über eine "Statusvariable" mit dem Namen ST, die Informationen über das Ergebnis der letzten Ein- beziehungsweise Ausgabeoperation übergibt. Ist die letzte Operation geglückt, besitzt diese Variable den Wert 0.

Interessant für uns ist der Wert 64. Diesen Wert nimmt ST an, wenn das Ende einer Datei erreicht wurde. Eine Datei mit unbekannter Größe kann daher in ein Array gelesen werden, indem nach jedem Lesevorgang ST mit dem Wert 64 verglichen wird. Das folgende Programm ersetzt den Teil "Lesen der Datei" und liest solange Daten ein, bis ST den Wert 64 besitzt, das heißt bis das Ende der Datei erreicht wird.

```
270 I=0:REM ANFANGSWERT FUER INDEXZAHL
280 DO UNTIL ST=64
290 : INPUT#2,AD$(I)
300 : PRINT AD$(I)
310 : I=I+1:REM INDEXZAHL ERHOEHEN
320 LOOP
330 CLOSE 2
```

Wenn Sie eine "unbekannte" Datei lesen wollen, dürfen Sie ausschließlich Stringvariablen im INPUT#-Befehl angeben, keinesfalls numerische Variablen wie INPUT#2,A.

In einer unbekannten Datei können sowohl Strings als auch Zahlen enthalten sein. Einer numerischen Variablen können jedoch keine Zeichenketten zugewiesen werden. Der Befehl INPUT#2,A führt daher zur Ausgabe der Fehlermeldung "FILE DATA ERROR IN (ZEILENNUMMER)", wenn eine Zeichenkette wie "MAIER" gelesen wird, die der Interpreter der numerischen Variablen A nicht zuweisen kann.

3.4.1.9 Speichern und Lesen von Leerstrings

Anhand der erwähnten problematischen Sonderzeichen konnten Sie bereits erkennen, daß die Dateiverwaltung mit dem C16, C116 oder Plus/4 nicht ohne Tücken ist, wenn nicht genau bekannt ist, wie der Interpreter Daten auf Cassette oder Diskette speichert und wieder liest.

Leerstrings, das heißt Strings ohne Inhalt ("" oder A$=""), die kein Zeichen enthalten, bewirken ebenfalls eine gewisse "Verwirrung" bei Ihrem Computer. Nicht beim Speichern, jedoch beim Lesen von Daten. Beim Speichern eines Leerstrings werden verständlicherweise keine Zeichen in die Datei geschrieben. Soll ein solcher Leerstring mit INPUT# gelesen werden, liest der BASIC-Interpreter die nächste mit einer Endmarke versehene Zeichenkette ein, das heißt jene Zeichenkette, die dem Leerstring folgt (!) und die Reihenfolge der Daten gerät durcheinander.

Sie sollten daher immer vermeiden, Leerstrings zu speichern. Die einfachste Methode ist das Ersetzen von Leerstrings durch ein Sonderzeichen, zum Beispiel durch "*". Beim Lesen von Daten wird dieser Vorgang umgekehrt. Wird eine Zeichenkette gelesen, die nur aus dem Zeichen "*" besteht, steht fest, daß es sich um einen beim Speichern "korrigierten" Leerstring handelt und das Zeichen "*" durch einen Leerstring ersetzt.

```
100 DIM AD$(50):REM ARRAY DIMENSIONIEREN
110 DO UNTIL AD$(IZ)="ENDE"
120 : INPUT "ADRESSE";AD$(IZ)
130 : IZ=IZ+1:REM INDEXZAHL ERHOEHEN
140 LOOP
150 IZ=IZ-1:REM ZAEHLER KORRIGIEREN
160 INPUT "CASSETTE(C) ODER DISKETTE(D)";GE$
170 :
180 REM SPEICHERN
190 IF GE$="C" THEN OPEN 2,1,1:ELSE OPEN 2,8,2,"TEST,S,W"
200 FOR I=0 TO IZ
205 : IF AD$(I)="" THEN PRINT#2,"*":GOTO 220
210 : PRINT#2,AD$(I)
```

```
220 NEXT I
230 CLOSE 2
240 :
250 REM LESEN
260 IF GE$="C" THEN OPEN 2,1,0:ELSE OPEN 2,8,2,"TEST,S,R"
270 FOR I=0 TO IZ
280 : INPUT#2,AD$(I)
285 : IF AD$(I)="*" THEN AD$(I)=""
290 : PRINT AD$(I)
300 NEXT I
310 CLOSE 2
```

Mit den beiden zusätzlichen Zeilen 205 und 285 wird unser Demoprogramm entsprechend erweitert. Zeile 205 (IF AD$(I)="" THEN PRINT#2,"*":GOTO 220) prüft vor dem Speichern einer Zeichenkette, ob es sich um einen Leerstring handelt. Wenn ja, wird stattdessen der String "*" gespeichert und der folgende PRINT#-Befehl übersprungen.

Beim Lesen wird umgekehrt vorgegangen. Zeile 285 (IF AD$(I)="*" THEN AD$(I)="") prüft, ob der String "*" gelesen wurde und ersetzt ihn gegebenenfalls durch einen Leerstring.

3.4.1.10 Spezielle Floppy-Befehle

Den folgenden Abschnitt können Sie überblättern, wenn Sie mit einer Datasette arbeiten. Er ist ausschließlich für Besitzer eines Diskettenlaufwerks von Interesse.

Sowohl der C16 und C116 als auch der Plus/4 besitzen eine Reihe von Befehlen, die den Umgang mit der Floppy erheblich erleichtern, die Befehle DIRECTORY, HEADER, SCRATCH, RENAME, COPY und COLLECT.

Diese Befehle werden im folgenden grob erläutert. Eine ausführliche Erläuterung halte ich in diesem BASIC-Kurs, dessen Schwerpunkt auf der Programmierung liegt, für unangebracht und Platzverschwendung. Ausführlichere Erläuterungen finden Sie sowohl im Handbuch zum C16, C116 bzw. Plus/4 als auch

im Floppy-Handbuch. In der folgenden Kurzerläuterung beziehe ich mich auf die Verwendung eines Diskettenlaufwerks, da wohl kaum ein Leser mehrere Laufwerke besitzen wird.

Der wichtigste dieser Befehle ist HEADER. Neue Disketten müssen vor der ersten Benutzung speziell vorbereitet, "formatiert" werden. Formatiert wird eine Diskette mit dem Befehl HEADER.

```
HEADER "(DISKETTEN-NAME)" [,I(ID)],DØ
```

1. DISKETTEN-NAME: Beliebiger Name mit maximal 16 Zeichen Länge.
2. ID: Zusatzname aus maximal zwei Zeichen. Jede Diskette sollte eine andere "ID" erhalten.

Beispiele:

1. HEADER "MAIER",I01 gibt der Diskette den Namen "MAIER" und die ID "01".
2. HEADER "MUELLER",IAB gibt der Diskette den Namen "MUELLER" und die ID "AB".

Achtung: Wird versehentlich eine Diskette formatiert, auf der bereits Programme oder Dateien vorhanden sind, werden alle Daten unwiderruflich gelöscht. Der HEADER-Befehl sollte mit größter Vorsicht verwendet wird.

Die Ausführung dieses Befehls dauert relativ lange. Soll HEADER eingesetzt werden, um bewußt (!) Daten auf einer bereits formatierten Diskette zu löschen, entfällt die Angabe einer ID. Obwohl alle Daten gelöscht werden, nimmt die Ausführung des Befehls erheblich weniger Zeit in Anspruch.

Wenn Sie bereits mit mehreren Disketten arbeiten, wird es Ihnen immer schwerer fallen, die Übersicht über vorhandene Programme und Dateien zu behalten. Der Befehl DIRECTORY liefert ein Inhaltsverzeichnis aller auf der eingelegten Diskette

vorhandenen Programme und Dateien. Der DIRECTORY-Befehl benötigt keine weiteren Angaben.

Mit SCRATCH können überflüssige Programme oder Dateien gelöscht werden, um Platz auf der Diskette zu schaffen.

```
SCRATCH "(FILENAME)
```

Filename: Der Name der zu löschenden Datei beziehungsweise des Programms. SCRATCH "TEST löscht die Datei beziehungsweise das Programm "TEST".

RENAME erlaubt die nachträgliche Umbenennung einer Datei oder eines Programms.

```
RENAME "(FILENAME ALT)" TO "(FILENAME NEU)
```

FILENAME ALT: Bisheriger Filename
FILENAME NEU: Gewünschter Filename

RENAME "TEST" TO "DEMO" gibt der Datei "TEST" den neuen Namen "DEMO".

Mit COPY kann eine Kopie eines wichtigen Programms (oder Datei) erstellt werden.

```
COPY "FILENAME ORIGINAL" TO "FILENAME KOPIE"
```

Der Kopie muß ein anderer Name gegeben werden, da es nicht möglich ist, mehrere "Files" mit identischen Namen auf einer Diskette zu verwalten.

COPY "TEST" TO "DEMO" kopiert die Datei "TEST" und gibt der Kopie den Namen "DEMO".

COLLECT "bereinigt" eine Diskette. Dateien, die nach Schreibvorgängen nicht ordnungsgemäß geschlossen wurden, können nicht mehr gelesen werden. COLLECT löscht solche unbrauch-

baren Dateien. Der Befehl COLLECT benötigt ebenso wie DIRECTORY keine weiteren Parameter.

Alle Befehle lassen sich sowohl im Direkt-Modus als auch - wie das folgende Programm zeigt - im Programm-Modus einsetzen.

```
100 HEADER "TESTDISK",I01,D0:REM FORMATIEREN
110 DIRECTORY:REM INHALTSVERZEICHNIS ZEIGEN
120 OPEN 2,8,2,"TEST,S,W":PRINT#2,"ABC":CLOSE 2:REM DATEI ERSTELLEN
130 DIRECTORY:REM INHALTSVERZEICHNIS ZEIGEN
140 RENAME "TEST" TO "DEMO":REM DATEI UMBENENNEN
150 DIRECTORY:REM INHALTSVERZEICHNIS ZEIGEN
160 COPY "DEMO" TO "DEMO1":REM DATEI KOPIEREN
170 DIRECTORY:REM INHALTSVERZEICHNIS ZEIGEN
180 SCRATCH "DEMO:REM ORIGINALDATEI LOESCHEN
190 DIRECTORY:REM INHALTSVERZEICHNIS ZEIGEN
200 COLLECT:REM DISKETTE BEREINIGEN
210 DIRECTORY:REM INHALTSVERZEICHNIS ZEIGEN
```

Dieses Programm formatiert die eingelegte Diskette. Verwenden Sie daher unbedingt eine leere Diskette!

3.4.2 Erstellung einer Adressenverwaltung

Im Graphik-Kapitel wurde ein Malprogramm erstellt. In diesem Kapitel über Dateiverwaltung wird eine Adressenverwaltung entworfen, die sowohl mit Cassette als auch mit Diskette lauffähig ist. Die benötigten Grundlagen kennen Sie bereits. Dennoch sollten Sie die folgenden Seiten keinesfalls überblättern.

Die Adressenverwaltung dient nicht nur der Übung und dem Einsatz der Datei-Befehle in einem umfangreicheren Programm. Sie werden vor allem von der Programmplanung profitieren, da Dateiverwaltungen ohne gründliche Planung und Vorüberlegungen zu den verwendeten "Datenstrukturen" kaum denkbar sind.

Zusätzlich bietet dieses Kapitel "fortgeschrittene Stringhandhabung". Sie werden mehrere Befehle kennenlernen, die den Umgang mit Stringvariablen geradezu zum Vergnügen machen.

3.4.2.1 Anforderungen an die Adressenverwaltung

Wie Sie noch sehen werden, sind die Ansprüche an eine gute Adressenverwaltung recht hoch und das zu entwickelnde Programm wird entsprechend komplex. Die Anforderungen an Sie sind erheblich höher als es bei der Entwicklung des Malprogramms der Fall war, doch ich denke, daß Sie durch den bisherigen Kursteil in die Lage versetzt wurden, mit diesen Anforderungen Schritt zu halten.

Eine Adressenverwaltung soll eine Vielzahl verschiedener Dinge beherrschen:

1. Eingabe von Adressen durch den Benutzer
2. Ändern bereits eingegebener Adressen
3. Löschen von Adressen
4. Suchen bestimmter Adressen
5. Sortieren von Adressen
6. Ausdrucken der Adressen
7. Speichern und Laden der Adresssen

Sie sehen, die gestellten Anforderungen sind äußerst vielfältig. Das zu entwickelnde Programm wird die verwendeten Prinzipien aufzeigen. Ich behaupte jedoch keinesfalls, daß das Programm Vergleichen mit käuflichen Programmen standhalten kann. Außer dem erstellten Grundgerüst ein wirklich professionelles Programm zu entwickeln, überlasse ich Ihnen.

Bedient werden soll das Programm mit Hilfe der Zifferntasten. Jeder Zifferntaste soll einer der acht Funktionen zugeordnet sein:

1: Adressen eingeben
2: Adressen ändern
3: Adressen löschen
4: Adressen sortieren
5: Adressen ausdrucken
6: Adressen speichern
7: Adressen laden
8: Adressen suchen
9: Adressen ausgeben

Zusätzlich zu den bereits erwähnten Funktionen wird die Funktion "Adressen ausgeben" in das Programm aufgenommen, da wahrscheinlich nicht jeder von Ihnen einen Drucker besitzt. Diese Funktion soll die Ausgabe aller in der Datei enthaltenen Adressen auf dem Bildschirm gestatten.

3.4.2.2 Die Datenstrukturen

Es gibt hunderte verschiedener Datenstrukturen, unter denen die Art und Weise verstanden wird, wie Daten miteinander verknüpft werden. Wenn der Begriff Datenstruktur weitgefaßt wird, fällt darunter nicht nur der Umgang mit im Speicher gehaltenen Adressen, sondern auch die Methode der Speicherung auf und des Ladens von einem Massenspeicher.

Im folgenden verwenden wir die einfachsten und am wenigsten problematischen Datenstrukturen, die ich kenne. Die vom Benutzer eingegebenen Adressen werden im Speicher "gesammelt", und zwar in dem Stringarray AD$(...). Jede Arrayvariable soll eine eingegebene Adresse enthalten. Es handelt sich um eine sogenannte "Listenstruktur". Das Array AD$(...) bildet eine ungeordnete Liste von Datensätzen. Diese Listenstruktur wurde immer dann von uns verwendet, wenn wir eingegebene Daten der Reihe nach in Arrayvariablen mit steigender Indexzahl speicherten.

Zum Speichern und Laden der Datei wird ebenfalls eine bekannte Struktur verwendet, nämlich die sequentielle Datei. Alle Arrayvariablen, die Adressen enthalten, werden der Reihe nach in eine Datei geschrieben.

Diese Strukturen sind im Grunde nicht neu für uns. Bei der Programmplanung ist es jedoch wichtig, sich vor der Programmierung darüber im klaren zu sein, wie die verschiedenen Funktionen mit den verwendeten Datenstrukturen zu realisieren sind.

1. Eintragen: Eine Zählvariable Z soll im Programm Auskunft über die Anzahl der vorhandenen Adressen geben. Beim Eintragen eines neuen Datensatzes wird Z um eins erhöht und deutet damit auf das erste nicht belegte "Element" der Liste, das heißt auf die erste freie Arrayvariable, die der letzten Adresse in der Liste folgt. Die neue Adresse wird dieser Variablen zugewiesen.

2. Ändern: Das Ändern von Adressen ist problemlos. Die Arrayvariable, die die zu ändernde Adresse enthält, wird einfach überschrieben.

3. Löschen: Wenn ein Datensatz gelöscht wird, muß die Zählvariable Z um eins vermindert werden. Um "Lücken" im Stringarray AD$(...) zu vermeiden, werden alle Adressen, die dem zu löschenden Element folgen (das heißt alle Arrayvariablen mit einer höheren Indexzahl) aufgerückt.

4. Suchen: Das Suchen von Datensätzen ist unproblemtisch, da keinerlei Eingriffe in die Liste erfolgen. Das Array AD$(...) wird Variable für Variable mit den "Suchkriterien" verglichen. Nachdem eine entsprechende Adresse gefunden wird, erhält der Benutzer die Möglichkeit, weiterzusuchen (eventuell enthält die Liste mehrere Adressen, die einem Suchkriterium wie zum Beispiel "MAIER" entsprechen).

5. Sortieren: Die Liste wird durch Vertauschen von Strings alphabetisch sortiert. Verwendet wird ein noch zu beschreibender "Sortier-Algorithmus".

6. Ausdrucken: Nach Anwahl dieser Funktion werden alle Adressen gedruckt, indem eine Drucker-Datei geöffnet und der Reihe nach alle Stringvariablen in diese Datei geschrieben werden.

7. Ausgeben: Die Ausgabe auf dem Bildschirm werden vor allem Leser begrüßen, die nicht über einen Drucker verfügen. Das Prinzip entspricht der Ausgabe auf die Drucker-Datei (siehe "Ausdrucken"), mit einem Unterschied: Da eine größere Datei nicht komplett auf dem Bildschirm ausgegeben werden kann, wird die jeweils nächste Adresse erst ausgegeben, wenn der Benutzer eine Taste betätigt, um ein "Durchscrollen" auf dem Bildschirm zu vermeiden.

3.4.2.3 Der Programmablauf

Aufgrund der vielfältigen Programmfunktionen sollten dem Benutzer Hinweise zur Programmbedienung gegeben werden. Dies erfolgt mit Hilfe eines "Auswahl-Menüs". Auf den Initialisierungsteil folgt ein Programmteil, der dem Benutzer auf dem Bildschirm Informationen zur Programmbedienung gibt und auf ein Kommando wartet.

Je nach gegebenem Kommando wird zu einem der Unterprogramme verzweigt (Eingeben, Ändern,...). Nach Ausführung der gewünschten Funktion werden erneut die Informationen ausgegeben und auf ein Kommando gewartet. Das Programm läuft in einer Schleife, in der sich die Abläufe "Infos ausgeben", "Auf Kommando warten" und "Kommando ausführen" ständig wiederholen.

3.4.2.4 Initialisierung

Im Initialisierungsteil soll das Array AD$(...) dimensioniert werden, das die Adressen aufnehmen wird. Zusätzlich wird der Benutzer gefragt, ob er mit Cassette oder Diskette arbeitet, da das Programm diese Informationen beim späteren Öffnen der Datei benötigt.

```
100 REM INITIALISIERUNG
110 DIM AD$(99):REM MAX.100 ADRESSEN
120 :
130 DO UNTIL GE$="C" OR GE$="D"
140 : INPUT "CASSETTE(C) ODER DISKETTE(D)";GE$
150 LOOP
```

In Zeile 110 wird ein Stringarray mit 100 Arrayvariablen angelegt (AD$(0) bis AD$(99)). Die Dateiverwaltung wird daher bis zu 100 Adressen bearbeiten können.

Die von Zeile 130 bis Zeile 150 reichende Schleife wird zur Abfrage des Massenspeichers benutzt. Um fehlerhafte Eingaben des Benutzers "abzufangen", wird die Schleife erst verlassen, wenn eine sinnvolle Eingabe vorgenommen wurde, das heißt, wenn der Benutzer die Frage "CASSETTE(C) ODER DISKETTE(D)" entweder mit "C" oder mit "D" beantwortet. Findet eine andere - unzulässige - Eingabe statt, wird die Frage wiederholt.

Infos ausgeben

```
190 REM INFOS AUSGEBEN
200 SCNCLR:REM BILSCHIRM LOESCHEN
210 PRINT TAB(10)"ADRESSENVERWALTUNG"
220 PRINT TAB(10)"-----------------"
230 PRINT
240 PRINT TAB(15)"1: EINGEBEN"
250 PRINT TAB(15)"2: AENDERN"
260 PRINT TAB(15)"3: LOESCHEN"
270 PRINT TAB(15)"4: SORTIEREN"
```

```
280 PRINT TAB(15)"5: DRUCKEN"
290 PRINT TAB(15)"6: SPEICHERN"
300 PRINT TAB(15)"7: LADEN"
310 PRINT TAB(15)"8: SUCHEN"
320 PRINT TAB(15)"9: AUSGEBEN"
330 PRINT
340 PRINT TAB(10)"GEWUENSCHTE FUNKTION ?"
```

Ungewohnt an der Ausgabe der Informationen mit PRINT ist der TAB-Befehl, der folgendes Format besitzt:

PRINT TAB(SPALTE),(ZEICHENKETTE)

TAB ist ein Zusatz zum PRINT-Befehl, der es erlaubt, eine gewünschte Spalte (0-39) anzugeben, ab der die betreffende Zahl oder Zeichenkette ausgegeben wird. Diese Formatierung wäre ebenso unter Einsatz des CHAR-Befehls möglich gewesen, mit dem bekanntlich Spalte und Zeile (!) bestimmt werden können.

3.4.2.5 Kommandoeingabe

```
390 REM AUF TASTE WARTEN
400 A$="":REM A$ LOESCHEN
410 DO UNTIL VAL(A$)>0 AND VAL(A$)<10
420 : GETKEY A$
430 LOOP
```

Der Kommandoeingabeteil mag auf den ersten Blick unnötig kompliziert vorkommen. Anstelle eines einfachen GETKEY-Befehls wird eine Schleife zur Eingabe verwendet. Sinn dieser Schleife ist eine "Eingabeprüfung". Jede Betätigung einer der Zifferntasten eins bis neun ist ein zulässiges Kommando. Der GETKEY-Befehl soll solange wiederholt werden, bis eine dieser Tasten gedrückt wird. Jede andere Taste wird ignoriert.

Der Befehl VAL(STRING) liefert den numerischen Wert einer Zeichenkette. VAL("30") ergibt die Zahl (!) 30 und A$="123.7":PRINT VAL(A$) gibt die Zahl (!) 123.7 aus. Der VAL-Befehl wird für sogenannte "Typ-Umwandlungen" benötigt, um eine Zeichenkette in eine Zahl zu wandeln.

Der Befehl GETKEY A$ wartet auf eine Taste und weist das betreffende Zeichen (!) der Stringvariablen A$ zu. Mit dem Befehl DO UNTIL VAL(A$)>0 AND VAL(A$)<10 wird der numerische Wert (!) dieses Zeichens geprüft. Ist der Wert kleiner als 1 oder größer als 9, ist die Bedingung nicht erfüllt, es wurde keine der Zifferntasten 1 bis 9 gedrückt. In diesem Fall wird weiter auf die Betätigung einer korrekten Taste gewartet, der nächste Schleifendurchgang beginnt.

Eine Frage ist noch offen: Warum wird vor dem Eintritt in die Schleife die Variable A$ gelöscht? Bedenken Sie, daß das Programm in einer Schleife läuft. Angenommen, Sie wählen die Funktion "Ausgabe auf Bildschirm", indem Sie die Taste 9 drücken. A$ enthält das Zeichen "9" und das entsprechende Unterprogramm wird aufgerufen. Nach der Rückkehr aus diesem Unterprogramm werden die Infos erneut ausgegeben und die Kommandoschleife ausgeführt.

Da A$ jedoch immer noch das Zeichen "9" enthält, ist die Bedingung UNTIL VAL(A$)>0 AND VAL (A$)<10 sofort erfüllt und die Schleife wird verlassen, noch bevor Sie ein neues Kommando geben. Das Unterprogramm "Ausgabe" wird erneut aufgerufen. Dieser Vorgang wiederholt sich endlos, wenn die Variable A$ nicht vor Eintritt in die Kommandoschleife gelöscht wird.

3.4.2.6 Verzweigung

```
450 REM VERZWEIGUNG
460 K=VAL(A$):REM K=NUM.WERT VON A$
470 ON K GOSUB 500,1000,1500,2000,2500,3000,3500,4000,4500
480 GOTO 200:REM ANFANG HAUPTSCHLEIFE
```

Auch in diesem Programmteil entdecken Sie einen neuen Befehl.

```
ON (NUMERISCHE VARIABLE) GOSUB (ZEILE1),(ZEILE2),..., (ZEILEN).
```

Die herkömmliche Lösung zur Verzweigung zu verschiedenen Unterprogrammen aufgrund der gedrückten Zifferntaste besteht in einer Reihe von IF-Befehlen.

```
IF A$="1" THEN GOSUB 500
IF A$="2" THEN GOSUB 1000
IF A$="3" THEN GOSUB 1500
...
...
...
IF A$="9" THEN GOSUB 4500
```

Der Befehl ON...GOSUB ersetzt alle acht Programmzeilen. Seine Funktionsweise: Nach dem Wort ON wird eine Zahl, numerische Variable oder ein Ausdruck, der eine Zahl liefert, angegeben. Dem Wort GOSUB folgen die Zeilennummern, an denen die aufzurufenden Unterprogramme beginnen.

Besitzt die angegebene numerische Variable den Wert 1, wird das erste angegebene Unterprogramm aufgerufen (GOSUB 500), besitzt sie den Wert 2, wird das zweite Unterprogramm aufgerufen (GOSUB 1000) und so weiter. Allgemein: Abhängig von der angegebenen Zahl N wird das N-te angegebene Unterprogramm aufgerufen.

Im Beispiel ergibt der Ausdruck K=VAL(A$) je nach gedrückter Zifferntaste ("1", "2",...,"9") eine der Zahlen 1 bis 9. Je nach Kommando wird eines der angegebenen Unterprogramme aufgerufen.

Zeile 480 stellt das Ende der Hauptschleife dar. Nach der Rückkehr aus dem aufgerufenen Unterprogramm und Ausführung der gewünschten Funktion wird zum Schleifenanfang gesprungen.

Im folgenden werden die einzelnen Unterprogramme vorgestellt und zwar in möglichst sinnvoller Reihenfolge, so daß Sie bereits nach Eingabe der ersten Unterprogramme mit der Adressenverwaltung arbeiten können.

3.4.2.7 Eingabe von Adressen

```
500 REM EINGABE
510 SCNCLR
520 INPUT "NAME";N$
530 INPUT "VORNAME";V$
540 INPUT "STRASSE";S$
550 INPUT "WOHNORT";W$
560 INPUT "TELEFON";T$
570 PRINT "KORREKTUR (J=JA) ?"
580 GETKEY A$
590 IF A$="J" THEN GOTO 520:REM WIEDERHOLEN
600 :
610 AD$(Z)=N$+"/"+V$+"/"+S$+"/"+W$+"/"+T$
620 Z=Z+1:REM INDEXZAHL ERHOEHEN
630 RETURN
```

Der erste Teil der Eingabe-Routine ist recht einfach. Der Bildschirm wird gelöscht und der Benutzer nach den verschiedenen Teilen der einzugebenden Adresse gefragt. Berücksichtigt werden muß der Fall, daß Fehler in Teilen der Adresse nach (!) dem Beenden der Eingabe mit RETURN entdeckt werden. Eine falsch eingegebene Adresse darf nicht rücksichtslos eingetragen werden. Dem Benutzer sollte eine Gelegenheit zur Korrektur gegeben werden.

Nach der vollständigen Eingabe der Adresse wird der Benutzer daher gefragt, ob Korrekturen nötig sind. Antwortet er mit "J" für "JA", wird die komplette Eingabe der Adresse wiederholt (Zeile 570-590).

Gehen wir davon aus, daß eine Adresse eingegeben und eventuelle Korrekturen vorgenommen wurden. Die Adresse besteht aus fünf verschiedenen Strings und soll in einer Stringvariablen untergebracht werden.

Zeile 610 verknüpft die fünf Strings miteinander, sie werden "addiert". Die einzelnen Angaben werden mit dem Zeichen "/" voneinander getrennt, um bei einer späteren Ausgabe einer Adresse Name, Vorname und die restlichen drei "Felder" voneinander unterscheiden zu können. Eine Ausgabe wie:

```
MAIERGEORGIDASTR.3MUENCHEN0621.1234
```

ist sicherlich verwirrender als:

```
MAIER/GEORG/IDASTR.3/MUENCHEN/0621.1234.
```

Die resultierende Zeichenkette - die komplette Adresse inclusive Trennzeichen - wird der Variablen AD$(Z) zugewiesen. Die Variable Z besitzt nach dem Starten des Programms mit RUN wie alle numerischen Variablen den Wert 0. Die erste Adresse wird daher der Arrayvariablen AD$(0) zugewiesen. Nach erfolgter Zuweisung wird die Indexzahl Z um eins erhöht (Zeile 620).

Eine weitere Adresse (Z=1) wird daher der Variablen AD$(1) zugewiesen, die dritte Adresse (Z=2) der Variablen AD$(2) und so weiter.

3.4.2.8 Ausgabe aller Adressen auf dem Bildschirm

Es dürfte sinnvoll sein, als nächstes den Programmteil "Ausgabe" zu erstellen. Mit diesem Programmteil kann das Programm bereits erprobt werden. Sie können Adressen eingeben und sich die eingegebenen Daten auf dem Bildschirm anschauen.

```
4500 REM AUSGABE AUF BILDSCHIRM
4510 SCNCLR
4520 FOR I=0 TO Z-1
4530 : PRINT AD$(I):REM ADRESSE AUSGEBEN
4540 : GETKEY A$:REM AUF TASTE WARTEN
4550 NEXT I
4560 RETURN
```

Dieser Programmteil ist erfreulich kurz. Die Ausgabe erfolgt in einer Schleife mit dem Startwert 0 (Ausgabe von AD$(0)) und dem Endwert Z-1 (Ausgabe von AD$(Z-1)).

Erinnern Sie sich, daß die Indexzahl Z immer um 1 größer ist als die Indexzahl der letzten eingetragenen Adresse.

In der Schleife wird jeweils eine Adresse ausgegeben und anschließend auf das Drücken einer beliebigen Taste gewartet. Alle Tasten des C16/Plus4 besitzen Dauerfunktion, das heißt eine wiederholte Betätigung wird erzielt, wenn Sie eine Taste längere Zeit betätigen. Sie können eine große Datei problemlos auf dem Bildschirm durchlaufen lassen, wenn Sie eine beliebige Taste konstant betätigen. Wenn Sie einen interessanten Ausschnitt der Datei genauer betrachten wollen, lassen Sie die Taste los und das Programm wird geduldig auf Sie warten.

3.4.2.9 Ausdrucken der kompletten Datei

Wie bereits besprochen wurde, sind die Funktionen "Ausgabe auf Bildschirm" und "Ausdrucken" nahezu identisch. Es bietet sich daher an, als nächstes die Druck-Routine zu entwerfen. Auch diese Routine besteht im Grunde nur aus einer Schleife, in

der alle Arrayvariablen nacheinander ausgegeben werden (jedoch ohne daß auf einen Tastendruck des Benutzers gewartet wird).

Der Unterschied zur Ausgabe auf dem Bildschirm besteht in der Verwendung einer logischen Datei, die zum Datenaustausch mit dem Drucker geöffnet und nach Ausgabe aller Datensätze wieder geschlossen wird.

```
2500 REM DRUCKEN
2510 OPEN 2,4:REM DRUCKER-DATEI OEFFNEN
2520 FOR I=0 TO Z-1
2530 : PRINT#2,AD$(I)
2540 NEXT I
2550 CLOSE 2
2560 RETURN
```

Beachten Sie die beim Öffnen der Datei benötigten Angaben. Als logische Filenummer wird 1 verwendet (ebenso gut können Sie eine beliebige andere Filenummer verwenden, wenn Sie anschließend Zeile 120 (Bezugnahme auf Filenummer) entsprechend ändern), und als Geräteadresse 4, die Gerätenummer des Druckers. Die Angabe eines Dateinamens und des Modus entfällt bei Drucker-Dateien.

3.4.2.10 Speichern und Laden der Datei

Wenn Sie alle vorgestellten Programmteile eingeben, können Sie bereits ein wenig mit der Adressenverwaltung "spielen". Außerordentlich bedauerlich ist es jedoch, daß nach dem Ausschalten des Rechners alle Daten neu eingegeben werden müssen.

Daher ist der nächste Schritt die Erstellung der Unterprogramme zum Speichern und Laden der Datei.

```
300 REM SPEICHERN
3010 IF GE$="C" THEN OPEN 2,1,1:ELSE SCRATCH"ADRESSEN":OPEN
2,8,2,"ADRESSEN,S,W"
3020 FOR I=0 TO Z-1
3030 : PRINT#2,AD$(I)
```

```
3040 NEXT I
3050 CLOSE 2
3060 RETURN
```

Die Routine zum Speichern der Datei kennen Sie bereits aus vorhergenden Programmbeispielen. Der einzige Unterschied besteht in dem SCRATCH-Befehl, den Zeile 110 enthält. Wie erläutert, ist es nicht möglich, mehrere Dateien mit identischen Namen auf einer Diskette zu speichern. Bevor die neue Datei "ADRESSEN" gespeichert wird, löscht SCRATCH eine eventuell bereits unter diesem Namen vorhandene Datei.

```
3500 REM LADEN
3510 IF GE$="C" THEN OPEN 2,1,0:ELSE OPEN 2,8,2,"ADRESSEN,S,R"
3520 Z=0
3530 DO UNTIL ST=64
3540 : INPUT#2,AD$(Z)
3550 : Z=Z+1
3560 LOOP
3570 CLOSE 2
3580 RETURN
```

Auch die Leseroutine ist bereits bekannt. In einer Schleife wird Adresse für Adresse gelesen und der Zähler Z nach jedem Lesevorgang erhöht. Das Einlesen ist beendet, wenn die Statusvariable ST den Wert 64 besitzt und das Dateiende erreicht wurde.

3.4.2.11 Suchen bestimmter Adressen

Die Suchroutine gehört in jeder Dateiverwaltung zu den wichtigsten Programmteilen, da sie auch von anderen Teilen benötigt wird (bevor eine Adresse gelöscht oder geändert werden kann, muß die betreffende Adresse gesucht werden).

Der C16, C116 bzw. Plus/4 besitzt einen Befehl, mit dem sich ohne größeren Aufwand äußerst komfortable Suchroutinen erstellen lassen, den Befehl INSTR.

```
INSTR(STRING,STRINGTEIL)
```

STRING und STRINGTEIL sind jeweils Zeichenketten oder Stringvariablen, die Zeichenketten enthalten. Der Befehl INSTR gibt die Position an, ab der STRINGTEIL in der Zeichenkette STRING enthalten ist.

Beispiele:

1. PRINT INSTR("MAIER","A") gibt die Zahl 2 auf dem Bildschirm aus, da "A" ab dem zweiten Zeichen in "MAIER" enthalten ist.

2. A$="MAIER":B$="A":PRINT INSTR(A$,B$) entspricht Beispiel 1, mit dem Unterschied, daß die direkte Angabe von Zeichenketten durch Stringvariablen ersetzt wird.

3. PRINT INSTR("MAIER","N") ergibt 0, da "N" in "MAIER" nicht enthalten ist.

4. PRINT INSTR("GERD MAIER","MAIER") ergibt 6, da "MAIER" ab dem sechsten Zeichen in "GERD MAIER" enthalten ist.

Sie sehen, INSTR ist eine außerordentlich komfortable Funktion. In unserem Programm wird sie wie folgt eingesetzt.

1. Der Benutzer gibt zur Suche nach einer bestimmten Adresse eine beliebige Zeichenkette ein, zum Beispiel "MAIER" oder "WAGNERSTR." oder "MANNHEIM".

2. Die Suchroutine überprüft mit der Funktion INSTR, ob das "Suchkriterium" in einer Adresse enthalten ist.

Da die Suchroutine auch zum Löschen und Ändern von Adressen benötigt wird, schreiben wir sie als eigenständiges Unterprogramm, das von verschiedenen Programmteilen aufgerufen werden kann.

```
5000 REM SUCHEN VON ADRESSEN
5010 REM UEBERGEBEN WIRD DIE VARIABLE 'I'
5020 REM U.DAS SUCHKRITERIUM 'S$'
5030 REM 'AD$(...)' WIRD AB 'AD$(I)'
5040 REM BIS 'AD$(Z)' DURCHSUCHT
5050 :
5060 DO WHILE I<Z
5070 : IF INSTR(AD$(I),S$)<>0 THEN PRINT "  ";AD$(I):EXIT
5080 : I=I+1
5090 LOOP
5100 RETURN
5110 :
5120 REM WAR DIE SUCHE VERGEBLICH,
5130 REM ENTHAELT 'I' NACH RUECKKEHR
5140 REM DEN GLEICHEN WERT WIE 'Z', SONST
5150 REM DIE INDEXZAHL DER GEFUNDENEN ADRESSE
```

Der Suchroutine werden vom aufrufenden Programmteil zwei Parameter "übergeben", die Variable I und das Suchkriterium S$. Der Wert von I bestimmt den Verlauf der Suche.

In einer Schleife wird das Array ab AD$(I) durchsucht. Der übergebene Parameter I bestimmt somit den Start der Suche, also das erste zu überprüfende Arrayelement. Gesucht wird, indem mit der INSTR-Funktion geprüft wird, ob das Suchkriterium S$ in der untersuchten Arrayvariable AD$(I) enthalten ist. Wenn ja, ist die Suche beendet. Die komplette Adresse (AD$(I)) wird in diesem Fall mit PRINT auf dem Bildschirm ausgegeben. Warum zwei Leerzeichen (" ") vor der eigentlichen Adresse ausgegeben werden, erfahren Sie im Programmteil "Ändern".

Anschließend wird der Befehl EXIT ausgeführt (Zeile 5020). Mit EXIT kann eine DO-Schleife vorzeitig verlassen werden, auch wenn die am Schleifenanfang angegebene Bedingung nicht erfüllt ist (WHILE I<Z). Die Suchschleife soll verlassen werden,

da die gesuchte Adresse gefunden wurde und eine weitere Suche daher überflüssig ist. Beachten Sie bitte, daß I nach Rückkehr aus dem Unterprogramm mit der Indexzahl der gefundenen Adresse identisch ist, diese Zahl als Parameter an das aufrufende Programm zurückübergeben wird.

Wenn das Suchkriterium im geprüften String nicht enthalten ist, wird die Suche fortgesetzt. Die Variable I wird um eins erhöht und die Prüfung findet im nächsten Schleifendurchgang mit der im Array folgenden Adresse statt.

Wie wir sehen, wird die Suchschleife in zwei Fällen verlassen.

1. Wenn eine Adresse gefunden wurde, die das Suchkriterium enthält (Vorzeitiges Verlassen der Schleife mit EXIT). In diesem Fall enthält I die Indexzahl des gefundenen Arrayelementes.

2. Wenn das komplette Array durchsucht wurde (übliches Verlassen, wenn die Schleifenbedingung WHILE I<Z nicht mehr erfüllt und daher die Arraygrenze erreicht ist). Der Wert von I entspricht in diesem Fall dem Wert von Z.

Wie diese Suchroutine und vor allem die hin- und zurückübergebenen Parameter effektiv eingesetzt wird, wird in Verbindung mit dem Programmteil "Suchen von Adressen" deutlich.

```
4000 REM SUCHEN
4010 SCNCLR
4020 INPUT "SUCHKRITERIUM";S$
4030 I=0:REM SUCHBEGINN AB 'AD$(0)'
4040 A$="":REM 'A$' LOESCHEN
4050 :
4060 DO UNTIL A$="N"
4070 : GOSUB 5060:REM AUFRUF SUCHROUTINE
4080 : IF I=Z THEN EXIT:REM ARRAYENDE?
4090 : I=I+1:REM SUCHE AB NAECHSTEM ELEMENT
4100 : PRINT "WEITERSUCHEN (N=NEIN) ?"
```

```
4110 : GETKEY A$
4120 LOOP
4130 RETURN
```

Der Ablauf: In einem vorbereitenden Teil wird der Bildschirm gelöscht, das Suchkriterium S$ vom Benutzer entgegengenommen und den Variablen I und A$ Ausgangswerte zugewiesen.

Anschließend beginnt eine Schleife, in der zuerst die Suchroutine aufgerufen wird. Diese Schleife wird sofort mit EXIT verlassen, wenn I nach Rückkehr aus der Suchroutine den gleichen Wert wie Z besitzt (Zeile 4080). Der Grund: Wir wissen, daß die Variable Z das Arrayende kennzeichnet. Da das Arrayende erreicht wurde, wenn I=Z gilt, ist die Weitersuche sinnlos und der Programmteil "Suchen" kann verlassen werden.

Wenn die Bedingung I=Z nach Rückkehr aus der Suchroutine nicht erfüllt ist, wurde eine Adresse gefunden, in der das Suchkriterium enthalten ist (siehe Zeile 5060). I enthält in diesem Fall die Indexzahl der betreffenden Arrayvariablen.

Wenn der Benutzer die folgende Frage "WEITERSUCHEN (N=NEIN) ?" (Zeile 4100) nicht mit "N" beantwortet (Zeile 4110), das heißt, wenn die Schleifenbedingung UNTIL A$="N" nicht erfüllt ist, erfolgt ein weiterer Schleifenduchgang, in dem erneut die Suchroutine aufgerufen wird (Zeile 4070). Da jedoch zuvor die Variable I um eins erhöht wurde (Zeile 4090), beginnt die neue Suche unmittelbar nach (!) der zuvor ausgegebenen Adresse, da die Indexzahl I nun auf das nächste Arrayelement weist.

Der Verlauf der Suche ist so komplex, weil ein Unterprogramm aufgerufen wird, dessen genauer Ablauf von übergebenen Parametern abhängig ist, der Suchroutine. In umfangreichen Programmen wird diese Methode, Unterprogramme zu verwenden, denen Parameter (I, S$) übergeben werden und die an das aufrufende Programm Parameter zurückübergeben (I), häufig angewendet.

Es ist außerordentlich wichtig, daß in solchen Unterprogrammen alle übergebenen Parameter mit REM-Zeilen ausführlich dokumentiert werden.

Der Vorteil dieser Methode ist eine Suchroutine, die so allgemein verwendbar ist, daß sie auch von den Programmteilen "Ändern" und "Löschen" verwendet werden kann.

3.4.2.12 Ändern von Datensätzen

```
1000 REM AENDERN
1010 SCNCLR
1020 INPUT "SUCHKRITERIUM";S$
1030 I=0:REM INDEXZAHL AUF STARTWERT
1040 A$="":REM 'A$' LOESCHEN
1050 :
1060 DO UNTIL A$="N"
1070 : GOSUB 5060:REM SUCHROUTINE
1080 : IF I=Z THEN EXIT:REM ARRAY DURCHSUCHT?
1090 : PRINT CHR$(145);:CURSOR OBEN
1100 : INPUT AD$(I):REM GEAENDERTE ADRESSE
1110 : I=I+1:REM INDEXZAHL ERHOEHEN
1120 : PRINT "WEITERSUCHEN (N=NEIN) ?"
1130 : GETKEY A$
1140 LOOP
1150 RETURN
```

Der Ablauf des Programmteils "Ändern" entspricht prinzipiell dem Programmteil "Suchen". Kern ist eine Schleife, die verlassen wird, wenn der Benutzer nicht mehr weitersuchen will (Zeile 1060), oder aber, wenn das Array komplett durchsucht wurde (Zeile 1080).

Wenn die Schleife nach dem Aufruf der Suchroutine (Zeile 1070) nicht sofort verlassen wird (Zeile 1080), wurde ein Datensatz gefunden und von der Suchroutine in folgender Form auf dem Bildschirm ausgegeben (Zeile 5070):

MAIER/OTTO/WAGNERSTR.3/MANNHEIM/0621.1234

Wie wir wissen, wird der Cursor nach einer Ausgabe mit PRINT ohne folgendes Komma oder Semikolon auf den Beginn folgenden Zeile gesetzt. Er befindet sich daher - während des Programmlaufs für uns unsichtbar - in der Zeile unterhalb der ausgegebenen Adresse.

Mit dem Befehl PRINT CHR$(145) in Zeile 1090 wird er eine Zeile nach oben bewegt (145=Code der Taste CURSOR OBEN). Nach Ausführung dieses Befehls befindet sich der Cursor auf der ersten Spalte jener Zeile, in der die Adresse ausgegeben wurde.

Wie wir wissen, erzeugt der folgende INPUT-Befehl (Zeile 1100) an dieser Position ein Fragezeichen und ein nachfolgendes Leerzeichen, bevor auf die Eingabe des Benutzers gewartet wird.

```
? MAIER/OTTO/WAGNERSTR.3/MANNHEIM/0621.1234
```

Zum Ändern der Adresse müssen wir diese nicht neu eingeben, die alte Adresse wird vorgegeben. Es genügt, diese Vorgabe zu bearbeiten und die Eingabe mit RETURN zu beenden.

```
? MAIER/OTTO/MOZARTSTR.71/MANNHEIM/0621.4591
```

Der INPUT-Befehl weist die Eingabe der Variablen AD$(I) zu. Da I die Indexzahl jener Stringvariablen ist, die die gefundene Adresse enthält, wird diese Variable durch die geänderte Adresse überschrieben.

Ebenso wie beim Suchen von Adressen wird der Benutzer gefragt, ob die Suche fortgesetzt werden soll. Wenn ja, kann eine weitere Adresse, die dem Suchkriterium entspricht, auf die gleiche Weise geändert werden.

3.4.2.13 Löschen von Adressen

Wie zu erwarten war, verläuft der Ablauf beim Löschen analog der Suche oder Änderung von Adressen. Der prinzipiell gleiche Ablauf wird verwendet, mit dem Unterschied, daß eine gefundene Adresse nicht mit einer geänderten Adresse überschrieben, sondern vollständig entfernt wird.

```
1500 REM LOESCHEN
1510 SCNCLR
1520 INPUT "SUCHKRITERIUM";S$
1530 I=0:REM INDEXZAHL AUF STARTWERT
1540 A$="":REM 'A$' LOESCHEN
1550 :
1560 DO UNTIL A$="N"
1570 : GOSUB 5060:REM SUCHROUTINE
1580 : IF I=Z THEN EXIT:REM ARRAY DURCHSUCHT?
1590 : FOR J=I TO Z-2
1600 :   AD$(J)=AD$(J+1)
1610 : NEXT J
1620 : Z=Z-1
1630 : PRINT "ADRESSE GELOESCHT"
1640 : PRINT "WEITERSUCHEN (N=NEIN) ?"
1650 : GETKEY A$
1660 LOOP
1670 RETURN
```

Die einfachste Methode zum Löschen einer Arrayvariablen - wenn Lücken im Array vermieden werden sollen - besteht darin, alle nachfolgenden Variablen nachzurücken, das heißt, ihren Index um eins zu vermindern. Genau dies ist die Aufgabe der FOR-Schleife (Zeile 1590-1610). Alle der zu löschenden Variablen AD$(I) folgenden Variablen werden der jeweils "darunterliegenden" Variablen zugewiesen. Ein Schema soll diesen Ablauf verdeutlichen.

Löschen der Adresse "MAIER/OTTO/..." (Indexzahl 2):

Vor dem Löschen (Z=6)	Nach dem Löschen (Z=5)
A$(0)="HANSEN/STEFAN..."	A$(0)="HANSEN/STEFAN..."
A$(1)="KLINGE/BERND..."	A$(1)="KLINGE/BERND..."
A$(2)="MAIER/OTTO..."	A$(2)="MUELLER/MICHAEL..."
A$(3)="MUELLER/MICHAEL..."	A$(3)="MAIER/GERD..."
A$(4)="MAIER/GERD..."	A$(4)="MAYER/WILLI..."
A$(5)="MAYER/WILLI..."	

Nach Ablauf aller Schleifendurchgänge sind die Arrayvariablen korrigiert, der Zähler Z muß jedoch noch - entsprechend der verminderten Adressenanzahl - um eins verringert werden (Zeile 1620).

Die Erhöhung von I vor dem erneuten Aufruf der Suchroutine entfällt. Zweck der Erhöhung ist es, die Suche ab jenem Arrayelement fortzusetzen, das dem gefundenen Element folgt. Nach beendetem Aufrücken aller Variablen besitzt das folgende Element (A$(2)="MUELLER/MICHAEL...") jedoch den gleichen Index wie das zuvor gefundene - und inzwischen gelöschte - Element (A$(2)="MAIER/OTTO..."). Die Suche soll daher mit jener Indexzahl fortgesetzt werden, mit der sie zuvor endete.

3.4.2.14 Sortieren der Adressen

Das Sortieren der Adressen ist die letzte noch fehlende Programmfunktion. Zum Glück ist das alphabetische Sortieren von Daten problemlos möglich, da Zeichenkettenvergleiche wie IF "MAIER"<"MUELLER" oder IF A$>=B$ vom C16, C116 oder Plus/4 automatisch unter alphabetischen Gesichtspunkten vorgenommen werden.

Sortierroutinen bestehen fast immer aus nur wenigen Programmzeilen, die es jedoch "in sich haben" und meist nur sehr schwer zu durchschauen sind. Eine Sortierroutine kann niemals einfach "runtergeschrieben" werden, aufgrund der Komplexität muß sie sorgfältig geplant werden. Da sich Sortierroutinen hervorragend

zur Demonstration des prinzipiellen Schemas der Programmentwicklung eignen (Lösungsvorschrift entwickeln und in Befehle einer Programmiersprache umsetzen), nutze ich diese Gelegenheit, um die Routine "gemeinsam" mit Ihnen zu entwickeln.

3.4.2.14.1 Sortieralgorithmus

Sortieralgorithmen, also Vorschriften zur Sortierung, existieren in vielen Versionen (Bubble-Sort, Quicksort, Shell-Sort etc.).

Sortierroutinen sollen Elemente üblicherweise in aufsteigender Ordnung sortieren, das größte Element soll nach beendeter Sortierung an letzter, das zweitgrößte an vorletzter Stelle im Array sein und so weiter.

Unsortiertes Array	Sortiertes Array
A$(0)="MAYER/WILLI..."	A$(0)="HANSEN/STEFAN..."
A$(1)="HANSEN/STEFAN..."	A$(1)="KLINGE/BERND..."
A$(2)="MUELLER/MICHAEL..."	A$(2)="MAYER/WILLI..."
A$(3)="MEIER/GERD..."	A$(3)="MEIER/GERD..."
A$(4)="KLINGE/BERND..."	A$(4)="MUELLER/MICHAEL..."

Es gibt die verschiedensten Möglichkeiten, diese Reihenfolge zu erreichen. Der von mir verwendete Sortieralgorithmus, "Straight Select", beruht auf folgender Idee:

1. Zuerst wird das gesamte Array aus N Elementen nach dem größten Element ("MUELLER/MICHAEL...") durchsucht. Das gefundene größte Element wird mit dem momentan letzten Element ("KLINGE/BERND...") des Arrays vertauscht und befindet sich damit an seiner korrekten Position, am Arrayende (Indexzahl N).

2. An vorletzter Position soll sich das zweitgrößte Element befinden. Da das größte Element mit dem Index N nicht mehr berücksichtigt werden muß,

wird das Array bis zu Element N-1 durchsucht, bis zum momentan vorletzten Element. Das in diesem Bereich gefundene größte Element wird mit dem Element N-1 vertauscht und befindet sich an seiner korrekten Position, an vorletzter Stelle im Array.

3. Da sich nun die beiden größten Elemente an ihrer korrekten Position befinden, wird das Array nun bis zu Element N-2 durchsucht und das gefundene größte Element mit dem drittletzten Element N-2 vertauscht.

Dieser Ablauf wiederholt sich so lange, bis das Array komplett sortiert ist. Das folgende Schema verdeutlich die einzelnen "Sortierstufen".

Unsortiert	Stufe 1	Stufe 2	Stufe 3	Sortiert
A$(0)="C"	A$(0)="C"	A$(0)="C"	A$(0)="A"	[A$(0)="A"]
A$(1)="E"	A$(1)="D"	A$(1)="B"	[A$(1)="B"]	A$(1)="B"
A$(2)="A"	A$(2)="A"	[A$(2)="A"]	A$(2)="C"	A$(2)="C"
A$(3)="B"	[A$(3)="B"]	A$(3)="D"	A$(3)="D"	A$(3)="D"
[A$(4)="D"]	A$(4)="E"	A$(4)="E"	A$(4)="E"	A$(4)="E"

Die eckigen Klammern sollen andeuten, bis wohin das Array in der jeweiligen Sortierstufe durchsucht wird. Das unsortierte Array wird bis zum letzten Element A$(4) durchsucht. Das größte gefundene Element AD$(1) wird mit AD$(4) vertauscht.

In Stufe 1 wird das Array bis zum (N-1)-ten Element durchsucht, bis A$(3). Das in diesem Bereich größte Element A$(1) wird mit dem letzten Element des Bereichs - A$(3) - vertauscht und die beiden größten Arrayelemente sind korrekt eingeordnet (letzte und vorletzte Position).

In Stufe 2 wird das Array bis zu A$(2) durchsucht. Das größte gefundene Element A$(0) wird mit dem letzten Bereichselement A$(2) vertauscht.

Stufe 3 durchsucht den Bereich A$(0) bis A$(1). Eine Vertauschung findet nicht statt, da sich das größere der beiden Elemente (A$(1)) bereits am Ende dieses Bereichs befindet. Die Sortierung ist beendet.

3.4.2.14.2 Programmierung des Algorithmus

Nachdem der Algorithmus feststeht, folgt die "Codierung", die Umsetzung des Algorithmus in ein BASIC-Programm. Das Schema zeigt bereits, daß zwei Schleifen benötigt werden. Eine äußere Schleife, die den zu durchsuchenden Bereich angibt und eine innere Schleife, die diesen Bereich nach dem größten enthaltenen Element durchsucht.

```
FOR I=... TO ...
  FOR J=... TO ...
    ...
    ...
    ...
  NEXT J
NEXT I
```

Die äußere Schleife muß einen Zeiger auf das Ende des zu durchsuchenden Bereichs zur Verfügung stellen, der nach jedem Schleifendurchgang um eins vermindert wird (das gesamte Array wird zuerst bis zum Element N, dann bis zum Element N-1, Element N-2 und so weiter durchsucht).

In unserer Dateiverwaltung zeigt Z auf das Arrayende, genauer: auf das letzte Element plus 1! Die äußere Schleife besitzt daher folgende Merkmale:

```
FOR I=Z-1 TO 1 STEP -1
```

Die Schleifenvariable I wird als Zeiger verwendet. Sie zeigt im ersten Durchgang auf das letzte Element (AD$(Z-1)) und wird in jedem folgenden Durchgang um eins vermindert.

Die innere Schleife soll den Bereich, dessen Ende durch den momentanen Wert der Variable I angezeigt wird, komplett durchsuchen.

```
FOR J=Ø TO I
```

Die Suche beginnt beim ersten Arrayelement AD$(0) und endet beim letzten Element des aktuellen Bereichs, bei AD$(I):

```
2ØØØ REM SORTIEREN
2Ø1Ø FOR I=Z-1 TO 1 STEP -1
2Ø2Ø : K$=""
2Ø3Ø : FOR J=Ø TO I
2Ø4Ø :   IF AD$(J)>K$ THEN K$=AD$(J):K=J
2Ø5Ø : NEXT J
2Ø6Ø : AD$(K)=AD$(I):AD$(I)=K$
2Ø7Ø NEXT I
2Ø8Ø RETURN
```

Zu Beginn der äußeren Schleife wird K$ gelöscht (Zeile 2020). Diese Variable soll nach beendetem Durchlauf der inneren Schleife das größte Bereichselement enthalten. Die innere Schleife durchsucht das Array bis zum Element AD$(I).

K$ wird in der inneren Schleife mit dem gerade untersuchten Element verglichen. Ist dieses Element größer als K$, wird K$ der Inhalt des jeweiligen Strings und der Variablen K die Indexzahl des untersuchten Elements zugewiesen.

K$ enthält daher während des Duchlaufs der inneren Schleife immer das bisher größte gefundene Element und K die zugehörige Indexzahl, die Position, an der sich das Element befindet.

AD$(I) enthält das letzte Element des untersuchten Bereichs. Nach beendeter innerer Schleife steht fest, daß AD$(K) (K: Index des größten Bereichelements) das größte Element des untersuchten Bereichs enthält. Beide Elemente werden miteinander vertauscht.

Anschließend erfolgt der nächste Durchgang der äußeren Schleife. I wird vermindert und der nächste Bereich durchsucht.

Seien Sie bitte nicht frustriert, wenn Sie diesen Programmteil nicht auf Anhieb verstehen, das tue ich auch nicht. Sortierroutinen zu verstehen, erfordert ein intensives "Hineindenken" in die Schleifenabläufe. Am besten verfolgen Sie die einzelnen Schleifendurchgänge mit einem Blatt Papier, auf dem Sie die Werte der Schleifenvariablen und den "Arrayzustand" Durchgang für Durchgang nachvollziehen.

Unsere kleine Dateiverwaltung ist vollendet. Nachfolgend finden Sie das komplette Programmlisting.

Wir sind zugleich am Ende des BASIC-Kurses angelangt. Sicherlich haben Sie bemerkt, daß der Schwierigkeitsgrad im Verlauf des Kurses allmählich anstieg und mit der Sortierroutine einen "Höhepunkt" erreichte.

Wenn Sie diesen Kurs nachvollziehen wollen, sollten Sie sich vor allem um die Dateiverwaltung kümmern. Sie soll vor allem demonstrieren, wie größere Programme entwickelt werden, nämlich "modular".

Am Anfang stehen Vorüberlegungen zum Funktionsumfang. Erst wenn die Datenstrukturen festgelegt sind, beginnt die Erstellung der einzelnen "Module". Diese Module sollten möglichst unabhängig voneinander sein. Je unabhängiger die einzelnen Programmteile sind, umso leichter lassen sie sich im nachhinein ändern, ohne daß die Änderungen andere Programmteile beeinflussen.

Wichtig ist das Steuerungsmodul, das das Gesamtprogramm kontrolliert. Wenn ein Programm modular aufgebaut ist, genügt eine winzige Änderung des Steuerungsmoduls und ein weiteres Programmodul, um neue Funktionen in ein bestehendes Programm zu integrieren.

Der gesamte BASIC-Kurs sollte Ihnen auch zeigen, daß Programmieren nicht einfach aus der Aneinanderreihung bestimmter Befehle besteht. Die Befehle selbst sind eigentlich unwichtig. Ich empfehle Ihnen, BASIC keinesfalls stur Befehl für Befehl zu erlernen.

Gehen Sie vor wie in diesem Kurs. Stellen Sie sich eine Aufgabe, planen Sie das Programm uns beginnen Sie mit der Erstellung der Module. Wenn ein spezielles Problem auftritt, können Sie immer noch in der Befehlsübersicht nachschauen, welcher neue BASIC-Befehl sich zur Lösung am besten eignet.

3.4.2.15 Programmlisting

```
100 REM INITIALISIERUNG
110 DIM AD$(99):REM MAX.100 ADRESSEN
120 :
130 DO UNTIL GE$="C" OR GE$="D"
140 : INPUT "CASSETTE(C) ODER DISKETTE(D)";GE$
150 LOOP
160 :
190 REM INFOS AUSGEBEN
200 SCNCLR:REM BILSCHIRM LOESCHEN
210 PRINT TAB(10)"ADRESSENVERWALTUNG"
220 PRINT TAB(10)"-----------------"
230 PRINT
240 PRINT TAB(15)"1: EINGEBEN"
250 PRINT TAB(15)"2: AENDERN"
260 PRINT TAB(15)"3: LOESCHEN"
270 PRINT TAB(15)"4: SORTIEREN"
280 PRINT TAB(15)"5: DRUCKEN"
290 PRINT TAB(15)"6: SPEICHERN"
300 PRINT TAB(15)"7: LADEN"
310 PRINT TAB(15)"8: SUCHEN"
320 PRINT TAB(15)"9: AUSGEBEN"
330 PRINT
340 PRINT TAB(10)"GEWUENSCHTE FUNKTION ?"
350 :
```

```
390 REM AUF TASTE WARTEN
400 A$="":REM A$ LOESCHEN
410 DO UNTIL VAL(A$)>0 AND VAL(A$)<10
420 : GETKEY A$
430 LOOP
440 :
450 REM VERZWEIGUNG
460 K=VAL(A$):REM K=NUM.WERT VON A$
470 ON K GOSUB 500,1000,1500,2000,2500,3000,3500,4000,4500
480 GOTO 200:REM ANFANG HAUPTSCHLEIFE
490 :
495 :
500 REM EINGABE
510 SCNCLR
520 INPUT "NAME";N$
530 INPUT "VORNAME";V$
540 INPUT "STRASSE";S$
550 INPUT "WOHNORT";W$
560 INPUT "TELEFON";T$
570 PRINT "KORREKTUR (J=JA) ?"
580 GETKEY A$
590 IF A$="J" THEN GOTO 520:REM WIEDERHOLEN
600 :
610 AD$(Z)=N$+"/"+V$+"/"+S$+"/"+W$+"/"+T$
620 Z=Z+1:REM INDEXZAHL ERHOEHEN
630 RETURN
640 :
650 :
1000 REM AENDERN
1010 SCNCLR
1020 INPUT "SUCHKRITERIUM";S$
1030 I=0:REM INDEXZAHL AUF STARTWERT
1040 A$="":REM 'A$' LOESCHEN
1050 :
1060 DO UNTIL A$="N"
1070 : GOSUB 5060:REM SUCHROUTINE
1080 : IF I=Z THEN EXIT:REM ARRAY DURCHSUCHT?
1090 : PRINT CHR$(145);:CURSOR OBEN
1100 : INPUT AD$(I):REM GEAENDERTE ADRESSE
1110 : I=I+1:REM INDEXZAHL ERHOEHEN
```

```
1120 : PRINT "WEITERSUCHEN (N=NEIN) ?"
1130 : GETKEY A$
1140 LOOP
1150 RETURN
1160 :
1170 :
1500 REM LOESCHEN
1510 SCNCLR
1520 INPUT "SUCHKRITERIUM";S$
1530 I=0:REM INDEXZAHL AUF STARTWERT
1540 A$="":REM 'A$' LOESCHEN
1550 :
1560 DO UNTIL A$="N"
1570 : GOSUB 5060:REM SUCHROUTINE
1580 : IF I=Z THEN EXIT:REM ARRAY DURCHSUCHT?
1590 : FOR J=I TO Z-2
1600 :   AD$(J)=AD$(J+1)
1610 : NEXT J
1620 : Z=Z-1
1630 : PRINT "ADRESSE GELOESCHT"
1640 : PRINT "WEITERSUCHEN (N=NEIN) ?"
1650 : GETKEY A$
1660 LOOP
1670 RETURN
1680 :
1690 :
2000 REM SORTIEREN
2010 FOR I=Z-1 TO 1 STEP -1
2020 : K$=""
2030 : FOR J=0 TO I
2040 :   IF AD$(J)>K$ THEN K$=AD$(J):K=J
2050 : NEXT J
2060 : AD$(K)=AD$(I):AD$(I)=K$
2070 NEXT I
2080 RETURN
2090 :
2100 :
2500 REM DRUCKEN
2510 OPEN 2,4:REM DRUCKER-DATEI OEFFNEN
2520 FOR I=0 TO Z-1
```

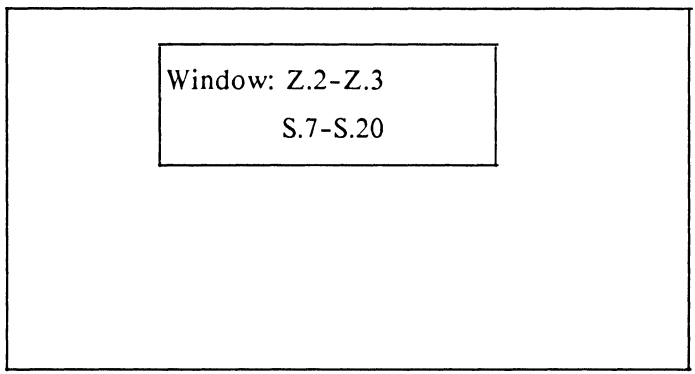
```
2530 : PRINT#2,AD$(I)
2540 NEXT I
2550 CLOSE 2
2560 RETURN
2570 :
2580 :
3000 REM SPEICHERN
3010 IF GE$="C" THEN OPEN 2,1,1:ELSE SCRATCH"ADRESSEN":OPEN
2,8,2,"ADRESSEN,S,W"
3020 FOR I=0 TO Z-1
3030 : PRINT#2,AD$(I)
3040 NEXT I
3050 CLOSE 2
3060 RETURN
3070 :
3080 :
3500 REM LADEN
3510 IF GE$="C" THEN OPEN 2,1,0:ELSE OPEN 2,8,2,"ADRESSEN,S,R"
3520 Z=0
3530 DO UNTIL ST=64
3540 : INPUT#2,AD$(Z)
3550 : Z=Z+1
3560 LOOP
3570 CLOSE 2
3580 RETURN
3590 :
3600 :
4000 REM SUCHEN
4010 SCNCLR
4020 INPUT "SUCHKRITERIUM";S$
4030 I=0:REM SUCHBEGINN AB 'AD$(0)'
4040 A$="":REM 'A$' LOESCHEN
4050 :
4060 DO UNTIL A$="N"
4070 : GOSUB 5060:REM AUFRUF SUCHROUTINE
4080 : IF I=Z THEN EXIT:REM ARRAYENDE?
4090 : I=I+1:REM SUCHE AB NAECHSTEM ELEMENT
4100 : PRINT "WEITERSUCHEN (N=NEIN) ?"
4110 : GETKEY A$
4120 LOOP
```

```
4130 RETURN
4140 :
4150 :
4500 REM AUSGABE AUF BILDSCHIRM
4510 SCNCLR
4520 FOR I=0 TO Z-1
4530 : PRINT AD$(I):REM ADRESSE AUSGEBEN
4540 : GETKEY A$:REM AUF TASTE WARTEN
4550 NEXT I
4560 RETURN
4570 :
4580 :
5000 REM SUCHEN VON ADRESSEN
5010 REM UEBERGEBEN WIRD DIE VARIABLE 'I'
5020 REM U.DAS SUCHKRITERIUM 'S$'
5030 REM 'AD$(...)' WIRD AB 'AD$(I)'
5040 REM BIS 'AD$(Z)' DURCHSUCHT
5050 :
5060 DO WHILE I<Z
5070 : IF INSTR(AD$(I),S$)<>0 THEN PRINT "  ";AD$(I):EXIT
5080 : I=I+1
5090 LOOP
5100 RETURN
5110 :
5120 REM WAR DIE SUCHE VERGEBLICH,
5130 REM ENTHAELT 'I' NACH RUECKKEHR
5140 REM DEN GLEICHEN WERT WIE 'Z', SONST
5150 REM DIE INDEXZAHL DER GEFUNDENEN ADRESSE
```

3.5 "Windowing" mit dem C16, C116, Plus/4

Sollte Ihnen der Begriff "Window" bisher unbekannt sein: Ein Window ist ein "Fenster", ein Bildschirmausschnitt. Der C16/Plus4 bietet Ihnen die Möglichkeit, den Bildschirm beliebig zu verkleinern, eine Art "Bildschirm im Bildschirm" zu definieren.

Sie können zum Beispiel angeben, daß der Bildschirm nur noch von Zeile zwei bis Zeile drei reichen soll, daß der linke Bildschirm-Rand bei Spalte 7 der rechte Rand bei Spalte 24 sein soll.



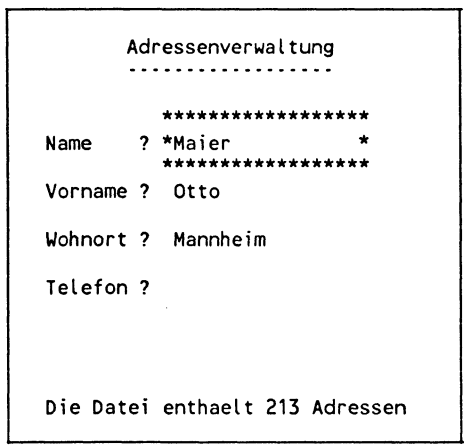
Alle Ausgabebefehle beziehen sich von nun an auf den neu definierten Bildschirm, das Window. SCNCLR löscht nicht mehr den kompletten Bildschirm, sondern nur den Inhalt des Windows. Cursorbewegungen sind ebenfalls nur innerhalb des Windows möglich.

3.5.1 Einsatzgebiete von Windows

Windows werden vorwiegend in "Eingaberoutinen" genutzt. Ein Beispiel: Ihr Programm will vom Benutzer wissen, welche Telefonnummer Herr Maier besitzt. Die Eingabe wird mit dem INPUT-Befehl vorgenommen.

Vielleicht haben Sie bereits bemerkt, daß INPUT nicht unbedingt ein "professioneller" Eingabebefehl ist. Während der Eingabe kann der Benutzer alle Editiertasten wie gewohnt verwenden. Er kann mit CLEAR den Bildschirm löschen, oder beliebige Bildschirminhalte mit den Editiertasten INST/DEL zerstören.

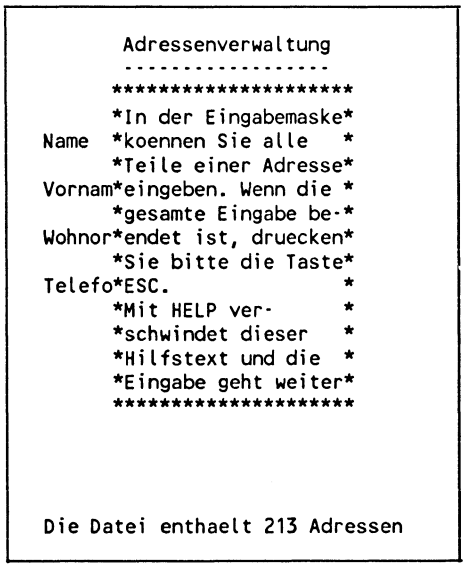
Stellen Sie sich nun ein Programm vor, daß mit einem "liebevollen" Bildschirmaufbau, vielleicht sogar einer sogenannten "Eingabemaske" arbeitet.



Bei der letzten Eingabe "Telefon ?" drückt der Benutzer versehentlich die Taste CLEAR und zerstört damit unbeabsichtigt den gesamten Bildschirmaufbau. Schade, nicht wahr? Und vor allem ist es unangenehm, daß Sie keine Möglichkeit besitzen, derartige Mißgeschicke zu verhindern, oder?

Nun, eine Möglichkeit bieten die erwähnten "Windows". Die Maske schreiben Sie mit PRINT-Befehlen auf den Bildschirm, anschließend definieren Sie hinter dem "Maskenfeld" "Name ?" ein Window (im Bild durch das Zeichen "*" dargestellt), bevor Ihr Programm mit INPUT eine Eingabe vom Benutzer anfordert. Alle Editierfunktionen beschränken sich nun auf das definierte Window. CLEAR löscht den Inhalt den kleinen Windows, der Rest des Bildschirms bleibt unangetastet.

Weitere Einsatzgebiete sind zum Beispiel die Ausgaben von "Hilfstexten" in einem Window, die dem Benutzer Auskunft über die Programmbedienung geben.



Diesen Hilfstext "über" den momentanen Bildschirminhalt zu legen, ist selbstverständlich auch ohne Windows möglich, jedoch umständlicher. Sie benötigen mehrere CHAR-Angaben, in denen gezielt anzugeben ist, wo die jeweilige Zeichenkette auszugeben ist.

Mit Windows wird das Problem einfacher gelöst. Sie definieren ein Window (mit "*" gekennzeichnet) und geben den Hilfstexte einfach mit PRINT-Anweisungen aus. Da die Bildschirmgrenzen neu definiert wurden, beziehen sich alle PRINT-Anweisungen auf das Window und lassen den restlichen Bildschirm unberührt.

3.5.2 Windows im Direkt-Modus

Genug der theoretischen Erläuterungen, es wird Zeit zu einem kleinen Experiment. Löschen Sie bitte den Bildschirm und bewegen Sie den Cursor zur fünften Spalte in der Zeile drei. Drücken Sie nun nacheinander - nicht zugleich! - die Tasten ESC und T. Bewegen Sie den Cursor in Spalte 20 von Zeile 10 und drücken Sie nacheinander die Tasten ESC und B.

Sie haben nun ein Window definiert, dessen linke obere durch Spalte 5 und Zeile 3, und dessen untere linke Ecke durch Spalte 20 und Zeile 10 gekennzeichnet ist.

Wenn Sie den Cursor bewegen, Zeichen eingeben und editieren, werden Sie festellen, daß sich alle Eingaben und Editierfunktionen auf den markierten Bereich beziehen, daß es Ihnen nicht möglich ist, den Bildschirm ausßerhalb des Windows zu beeinflußen.

Drücken Sie nun bitte nacheinander die Tasten ESC und N. Der Bildschirm wird gelöscht und das Window aufgehoben. Ihnen steht wieder der gesamte Bildschirm zur Verfügung.

ESC-Funktionen zur Arbeit mit Windows

- ESC+T markiert die linke obere Windowecke an der aktuellen Cursorposition.

- ESC+B markiert die rechte untere Windowecke an der aktuellen Cursorposition.

- ESC+N hebt das Window auf und löscht den Bildschirm.

3.5.3 Windows im Programm-Modus

Im Direkt-Modus werden Windows kaum benötigt. Nützlich sind sie vor allem bei der Anwendung in Programmen, zum Beispiel zur Verbesserung einer Eingabe mit INPUT. In Programmen können Sie zwar den Cursor nicht mit der Tastatur an die gewünschten Windowecken steuern und diese mit ESC+T beziehungsweise ESC+B festlegen.

Dennoch gibt es eine Möglichkeit, Windows auch in Programmen zu definieren. Das erste Problem, den Cursor an die gewünschte Position zu setzen, wird mit dem CHAR-Befehl

gelöst. CHAR mit Angabe eines Leerstrings gibt zwar keinerlei Zeichen aus, setzt jedoch den Cursor an die angegebene Position.

CHAR 1,15,14,"" setzt den Cursor auf Spalte 15 von Zeile 14.
CHAR 1,20,22,"" setzt den Cursor auf Spalte 20 von Zeile 22.

Das zweite Problem, das "Markieren" der Position mit ESC+T und ESC+B wird mit der CHR$-Funktion gelöst. Mit CHR$ können wir beliebige Zeichen ausgeben, indem wir den Code des Zeichens angeben. Der Taste ESC ist (siehe Tabelle) der Code 27 zugeordnet.

Der Befehl PRINT CHR$(27);"T" entspricht daher dem Drücken der Tasten ESC und T. ESC und B kann mit PRINT CHR$(27);"B" "simuliert" werden.

Das folgende Programm enthält einen INPUT-Befehl, der zur Eingabe auffordert. Die Eingabe findet in einem zuvor definierten Window statt, der Benutzer kann den Bildschirm außerhalb dieser "Eingabezone" nicht beeinflusssen.

```
100 T$=CHR$(27)+"T":REM OBERE LINKE ECKE FESTLEGEN
110 B$=CHR$(27)+"B":REM UNTERE RECHTE ECKE FESTLEGEN
120 SCNCLR:REM BILDSCHIRM LOESCHEN
130 PRINT TAB(10)"ADRESSENVERWALTUNG"
140 PRINT TAB(10)"------------------"
150 CHAR 1,0,5,"NAME      ":GOSUB 1000:N$=A$
160 CHAR 1,0,6,"NAME      ":GOSUB 1000:N$=A$
170 CHAR 1,0,7,"NAME      ":GOSUB 1000:N$=A$
180 CHAR 1,0,8,"NAME      ":GOSUB 1000:N$=A$
190 :
200 PRINT N$:PRINT V$:PRINT W$:PRINT T$
210 END:REM PROGRAMMENDE
220 :
230 :
1000 REM WINDOW-INPUT
1010 PRINT T$;"          ";B$:REM LAENGE 10 ZEICHEN
1020 INPUT A$:REM EINGABE IM WINDOW
1030 CHAR 1,0,0,T$:REM NORMALE OBERE ECKE
```

```
1040 CHAR 1,39,24,B$:REM NORMALE UNTERE ECKE
1050 RETURN
```

Der Programmablauf: Um ein wenig Tipparbeit zu sparen, werden am Programmanfang die Strings T$ und B$ definiert, die die Zeichen ESC und "T" beziehungsweise "B" enthalten. Zur Definition einer Window-Ecke kann daher im folgenden Programm PRINT CHR$(27);"T" durch PRINT T$ und PRINT CHR$(27);"B" durch PRINT B$ ersetzt werden.

Anschließend wird der Bildschirm gelöscht und der "Programmtitel" ausgegeben (Zeilen 120-140). Die folgenden vier Befehle fordern jeweils zu einer Eingabe auf. Zuerst wird ein Kommentar ausgegeben "NAME" etc.), danach das Eingabeunterprogramm aufgerufen. Dieses Unterprogramm übergibt die Eingabe des Benutzers an A$. Die Eingabe wird nach der Rückkehr aus dem Unterprogramm einer der Variablen N$, V$, W$ oder T$ zugewiesen ("NAME", "VORNAME", "WOHNORT" und "TELEFON").

Für uns interessant ist vor allem das ab Zeile 1000 beginnende Unterprogramm. Dieses Unterprogramm definiert zuerst das "Eingabefenster" (Zeile 1010). An der aktuellen Cursorposition wird die linke obere Ecke markiert (PRINT T$) und zehn Zeichen weiter (" ") die rechte untere Window-Ecke. Das Window hat daher eine Breite von zehn Zeichen und eine Länge von nur einer Zeile, eben der gewünschten Eingabezeile.

Der folgende INPUT-Befehl (Zeile 1020) fordert den Benutzer zur Eingabe auf. Der Benutzer kann eine Eingabe mit einer Länge von maximal zehn Zeichen vornehmen (die Eingabelänge kann jederzeit durch Erhöhung oder Verminderung der Anzahl an Leerzeichen zwischen dem Window-Anfang und dem Window-Ende geändert werden).

Löscht er mit CLEAR den Bildschirm, betrifft dies nur das Window, nicht den restlichen Bildschirm. Ebensowenig ist es dem Benutzer möglich, mit den Cursortasten das Window zu verlassen und den Bildschirmaufbau zu zerstören. Er ist im definierten Window - der Eingabezone - "eingesperrt".

Nachdem seine Eingabe A$ zugewiesen wurde, wird die Window-Markierung rückgängig gemacht, der normal große Bildschirm als Window definiert (Zeile 1030-1040) und zum aufrufenden Hauptprogramm zurückgekehrt. Das Aufheben des Windows ist zwar auch mit PRINT CHR$(27);"N" möglich, hierbei wird jedoch unerwünschterweise der Bildschirm gelöscht.

Wenn Sie dieses Programm starten, stellen Sie eine unerwünschte Eigenschaft der Eingabe fest. Wenn bei der Eingabe das Ende des Windows erreicht wird, löscht der BASIC-Interpreter automatisch den gesamten Window-Inhalt.

Der Grund für dieses Verhalten: Am Ende der Eingabezone wird wie üblich der Bildschirm um eine Zeile nach oben gescrollt, und die unterste Bildschirmzeile ist leer. Die Bildschirmgröße wurde jedoch neu festgelegt, die Windowlänge beträgt eine einzige Zeile. Wird das Window um eine Zeile nach oben gescrollt, verschwindet die gescrollte Zeile daher völlig!

Abhilfe schafft eine weitere "Escape-Funktion", ESC+M. Diese Funktion schaltet das Scrollen des Bildschirms ab. Mit ESC+L wird das Scrolling wieder eingeschaltet. Fügen Sie daher bitte die folgenden Zeilen in das Programm ein:

```
90 PRINT CHR$(27);"M":REM SCROLLING AUS
195 PRINT CHR$(27);"L";REM SCROLLING EIN
```

Am Programmanfang wird das Scrolling ausgeschaltet. Wieder eingeschaltet wird es erst, nachdem alle Eingaben vorgenommen wurde.

Diese Eingaberoutine können Sie selbstverständlich in beliebigen eigenen Programmen einsetzen. Sie umgeht einige Schwächen, die der INPUT-Befehl besitzt.

3.6 Fehlerbehandlung mit TRAP und RESUME

Der C16, C116 bzw. Plus/4 besitzt äußerst komfortable Befehle zur "Fehlerbehandlung". Im Laufe der Zeit werden Sie sich sicherlich mit Hunderten von Fehlern in Ihren Programmen abgeben müssen. Die Behebung logischer Fehler im Programmablauf müssen Sie selbst übernehmen, es gibt jedoch eine Menge möglicher Fehler, um die sich Ihr Rechner selbst kümmern kann.

Ein gutes Beispiel für einen solchen Fehler ist die Meldung "DIVISION BY ZERO ERROR". Dieser Fehler tritt immer dann auf, wenn Ihr C16, C116 oder Plus/4 Umögliches leisten soll, zum Beispiel wenn Sie verlangen, daß er eine Zahl durch null teilt (oder können Sie mir sagen, was 30/0 ergibt?).

```
100 INPUT "ZAHL";Z
110 PRINT 30/Z
RUN
ZAHL? 0
DIVISION BY ZERO ERROR IN 110
```

Eine Möglichkeit, diesen Fehler "abzufangen", besteht darin, zu prüfen, ob die Zahl null eingegeben wurde und die Eingabe gegebenenfalls zu wiederholen.

Die elegantere Lösung sieht so aus:

```
100 TRAP 1000
110 INPUT "ZAHL";Z
120 PRINT 30/Z
130 :
1000 RESUME 110
RUN
ZAHL? 0
ZAHL? 10
 3

READY.
```

Wenn Sie dieses Programm starten, stellen Sie fest, daß die Eingabe solange wiederholt wird, bis eine Zahl ungleich null eingegeben wird, ohne daß diese Zahl vom Programm selbst überprüft wird.

Der Ablauf: Der Befehl TRAP (der immer am Programmanfang stehen sollte!) weist den Interpreter an, zur "Fehlerbehandlungsroutine" nach Zeile 1000 zu verzweigen, wenn während des Programmlaufs ein Fehler auftritt.

Sobald ein Fehler wie "DIVISION BY ZERO" auftritt, verzweigt der Interpreter zu Zeile 1000. Im folgenden Programmteil wird der Fehler "behandelt".

Die Anweisung RESUME 110 wirkt ähnlich wie ein GOTO 110. Das Programm wird mit Zeile 110 fortgesetzt und die Eingabe wiederholt.

Es bestehen drei Möglichkeiten, aus einer Fehlerroutine wieder in das Hauptprogramm zu verzweigen:

1. RESUME setzt das Programm mit jenem Befehl fort, der den Fehler verursachte. Ist der Fehler konstant vorhanden, ergibt sich daraus eine Endlosschleife.

2. RESUME NEXT setzt das Programm mit jenem Befehl fort, der dem Befehl folgt, der den Fehler verursachte.

3. RESUME (ZEILENNUMMER) setzt das Programm mit der angegebenen Zeilennummer fort.

Wie Sie sehen, sind die Möglichkeiten zur Fehlerbehandlung, die der C16/Plus4 bietet, sehr komfortabel. Zusätzlich zu TRAP und RESUME stehen Ihnen die Variablen ER, ER und ERR$(ER) zur Verfügung, die Auskunft geben über die Art des Fehlers und den Ort, an dem er auftrat.

1. ER: Nummer des aufgetretenen Fehlers.

2. EL: Nummer der Programmzeile, in der der Fehler auftrat.

3. ERR$(EL): Fehlermeldung im Klartext.

Auskunft über die Bedeutung der Fehlernummer gibt die Tabelle im Handbuch zum C16, C116 oder Plus/4.

Einen Nachteil besitzt die Fehlerbehandlung: Diskettenfehler werden von TRAP nicht erkannt. Dafür stehen Ihnen jedoch die reservierten Variablen DS und DS$ zur Verfügung, die Meldungen über aufgetretene Diskettenfehler enthalten.

1. DS: Nummer des Fehlers bei Diskettenoperationen (Standardwert 0).

2. DS$: Fehlermeldung im Klartext bei Diskettenoperationen.

Ich empfehle Ihnen zur Behandlung von Diskettenfehlern meine eigene Methode: Fragen Sie vor allem nach dem Öffnen einer Datei den Wert von DS ab. Besitzt DS den Wert 0, wurde die Datei problemlos geöffnet. Enthält DS einen anderen Wert, sollten Sie zur Fehlerroutine verzweigen, die möglichst alle Fehler behandelt, die auftreten können.

Das folgende Demoprogramm zeigt, wie eine Fehlerbehandlungsroutine erstellt werden könnte.

```
100 TRAP 1000:REM FEHLERROUTINE
110 OPEN 2,8,2,"XYZ,S,W"
120 IF DS<>0 THEN GOTO 1100
130 PRINT#2,"DIES IST"
140 PRINT#2,"EIN TEST"
150 CLOSE 2
160 END
170 :
180 :
```

```
1000 REM FEHLERBEHANDLUNG
1010 PRINT ERR$(ER):REM FEHLERMELDUNG
1020 PRINT "WIEDERHOLEN (J=JA) ?"
1030 GETKEY A$
1040 IF A$="J" THEN RESUME:ELSE RESUME NEXT
1050 :
1100 REM DISK-FEHLER
1110 PRINT DS$:REM FEHLERMELDUNG
1120 PRINT "WIEDERHOLEN (J=JA) ?"
1130 GETKEY A$
1140 IF A$="J" THEN CLOSE 2:GOTO 110
```

Tritt ein "rechnerinterner" Fehler auf, der von TRAP erkannt wird, verzweigt dieses Programm zur Fehlerbehandlungsroutine, die ab Zeile 1000 beginnt. In dieser Routine wird dem Benutzer die Fehlermeldung im Klartext ausgegeben und er wird gefragt, ob der Vorgang wiederholt werden soll.

Anhand der Fehlermeldung kann der Benutzer versuchen, den Fehler zu beheben (zum Beispiel das nächste Mal als Zahl, durch die geteilt wird, keine null eingeben). Antwortet er auf die Frage "WIEDERHOLEN (J=JA) ?" mit "J", wird der Vorgang wiederholt, wünscht er dagegen keine Wiederholung, wird das Hauptprogramm mit dem auf die Fehlerursache folgenden Befehl fortgesetzt.

Die Behandlung von Diskettenfehlern verläuft ähnlich, Sie als Programmierer müssen einen aufgetretenen Fehler jedoch selbst durch Prüfung der Variablen DS erkennen.

Die Disketten-Fehlermeldung wird im Klartext ausgegeben (zum Beispiel "WRITE PROTECT ON" oder "FILE NOT FOUND") und - wenn vom Benutzer gewünscht - der gesamte Vorgang einschließlich des Öffnens der Datei wiederholt. Zuvor muß jedoch die bereits geöffnete Datei geschlossen werden, da der Versuch, eine offene Datei erneut zu öffnen, ebenfalls zu einem Fehler führt.

Wünscht der Benutzer keine Wiederholung, endet das Programm. Anstelle dieser einfachsten Lösung dürfen Sie sich gern eine eleganter aussuchen. Keine Lösung ist es jedoch, mit den folgenden Schreibbefehlen einfach fortzufahren, wenn bereits beim Öffnen der Datei ein Fehler auftrat, da diese dann mit Sicherheit ebenfalls mißlingen.

Noch eine Bemerkung zur Fehlerbehandlung mit TRAP: Sie sollten dafür sorgen, daß in der Fehlerroutine selbst keinesfalls ein Fehler auftreten kann. Fehler in der Fehlerbehandlungsroutine können auch mit TRAP nicht mehr abgefangen werden.

3.7 Fehlersuche mit TRON, TROFF, STOP und CONT

Wir sind nun am Ende des BASIC-Kurses angelangt. Wenn Sie mit der Erstellung eigener Programme beginnen, werden Sie schnell feststellen, daß die Fehlersuche oft mehr Zeit in Anspruch nimmt als die eigentliche Programmerstellung.

Der C16, C116 oder Plus/4 bietet Ihnen jedoch auch Hilfsmittel zum Testen Ihrer Programme an. Mit den Befehlen TRON und TROFF können Sie den Programmablauf detailliert verfolgen.

Wenn Sie feststellen, daß ein bestimmter Programmteil fehlerhaft funktioniert, fügen Sie am Anfang dieses Programmteils den Befehl TRON ("Trace on") ein. Sobald der Programmteil erreicht wir, gibt der C16, C116 bzw. Plus/4 ständig die Nummer der momentan bearbeiteten Programmzeile auf dem Bildschirm aus. Anhand dieser Ausgaben können Sie verfolgen, ob eine Schleife zu oft oder nicht oft genug durchlaufen wird, oder ob zum Beispiel ein "bedingter" Sprung mit GOTO in einem IF-Befehl nicht ausgeführt nicht, weil die Bedingung niemals erfüllt ist.

Sobald Sie vom "Trace"-Modus genug haben, unterbrechen Sie das Programm mit STOP und geben TROFF ("Trace off") ein, um diesen Modus wieder abzuschalten.

```
100 TRON
110 GOTO 200
120 :
200 GOTO 110
RUN
[110] [200] [110] [200] [110] [200] [110] [200]
[110] [200] [110] [200] [110] [200] [110] [200]
...
```

Die Endlosschleife des Demoprogramms wird im Trace-Modus schnell erkannt.

Mit dem Befehl (nicht der Taste!) STOP haben Sie die Möglichkeit, ein Programm an einer gewünschten Stelle vorübergehend zu beenden, und anschließend mit CONT ab der Abbruchstelle fortzusetzen.

```
100 X=10
110 IF X=10 THEN Y=2*X
120 STOP
130 X=X+2
140 STOP
150 X=X+Y
RUN
BREAK IN 120
PRINT Y
 0
CONT

READY.
```

Dieses Demoprogramm zeigt den Einsatz von STOP und CONT. Gehen wir davon aus, daß Y am Programmende den Wert 30 besitzen soll, Sie jedoch leider feststellen, daß Y am Programmende einen völlig anderen Wert besitzt. Unmittelbar hinter der von Ihnen zuerst vermuteten Fehlerquelle, der Zeile 110, fügen Sie den Befehl STOP ein.

Sobald der BASIC-Interpreter auf diesen Befehl trifft, wird das Programm unterbrochen und die Meldung "BREAK IN 120" ausgegeben.

Im Direkt-Modus lassen Sie sich mit PRINT Y den Wert von Y ausgeben und stellen fest, daß Y nach Bearbeitung von Zeile 110 wie beabsichtigt den Wert 20 besitzt.

Nun wissen Sie bereits, daß das Programm bis einschließlich Zeile 110 einwandfrei funktioniert. Sie geben im Direkt-Modus den Befehl CONT ein, und das Programm wird fortgesetzt, als hätte die Unterbrechung niemals stattgefunden.

Der nächste "Haltepunkt" ist in Zeile 140. Das Programm wird erneut unterbrochen und Sie können wiederum "per Hand" die Inhalte der verschiedenen Variablen im Direkt-Modus überprüfen.

Mit STOP und CONT kann man sich in einem Programm "vortasten", bis eine Fehlerquelle eindeutig lokalisiert ist.

Achtung: Eine Fortsetzung des Programms mit CONT ist nicht mehr möglich, wenn nach der Unterbrechung Programmzeilen geändert wurden oder ein Fehler auftrat!

4. Befehlsübersicht

Dieses Kapitel enthält eine Übersicht der Befehle des C16, C116 und Plus/4. Im Gegensatz zum BASIC-Kurs ist die Übersicht weniger "locker", sondern erläutert systematisch das Befehls-Format und die jeweilige Funktion.

Die Erläuterungen sind nur kurz, da alle Befehle bereits im Handbuch zum C16, C116 bzw. Plus/4 ausführlich beschrieben sind. Alternativ zu diesem Handbuch liegt das Schwergewicht dieser Übersicht in den Beispielen, die wiederum im Handbuch nur spärlich vorhanden sind.

Ich hoffe Ihr Einverständnis zu haben, wenn ich auf die Darstellung der arithmetischen Funktionen verzichte. Die arithmetischen Funktionen sind ausführlich im Handbuch beschrieben, so daß ich eine Wiederholung des dort gesagten Ihnen und mir selbst ersparen will.

Wie bereits öfter in diesem Buch wiederholt, gilt für alle Befehle: Nur durch ständiges Üben werden Sie die Befehle "im Schlaf" beherrschen und wirklich optimal einsetzen können.

4.1 Befehlsformat

Wie wir bereits sahen, besitzt der C16, C116 und Plus/4 viele Befehle mit "optionalen" Parametern, das heißt mit Parametern, die angegeben werden können! Optionale Parameter werden in eckigen Klammern dargestellt (BOX [FARBZONE], X1,Y1, X2,Y2 [,WINKEL]). Werden optionale Parameter nicht angegeben, nimmt der C16, C166 oder Plus/4 "Standardwerte".

Soll ein optionaler Parameter nicht angegeben, ein folgender Parameter jedoch angegeben werden, kann der optionale Parameter durch ein Komma ersetzt werden. Auch in diesem Fall verwendet der BASIC-Interpreter den Standardwert des Parameters.

Beispiel:
```
CIRCLE 1,100,80,50,70,,,30
```

zeichnet eine Ellipse mit dem Mittelpunkt 100/80, der Breite 50 und der Höhe 70. Die Ellipse wird um 30 Grad gedreht. Die optionalen Parameter ANFANG und ENDE (nötig zum Zeichnen von Ausschnitten) werden durch die Standardwerte ersetzt.

4.2 Toolkit-Befehle

Toolkit-Funktionen sind zur Programmerstellung nützlich. Fast alle unter dieser Rubrik vorgestellten Befehle können nur im Direkt-Modus eingegeben werden. Ausnahmen werden beschrieben.

AUTO

Format:
```
AUTO [ZEILENABSTAND]
```

Funktion: AUTO schaltet die automatische Zeilennumerierung ein, das heißt nach der Eingabe einer Programmzeile wird Ihnen die nächste Zeilennummer vorgegeben. Die erste Zeilennummer müssen Sie selbst eingeben, alle weiteren Zeilennummern werden - in jeweiligen Zeilenabstand - vorgegeben. Wird die Angabe des Zeilenabstands weggelassen, nimmt der BASIC-Interpreter den Standardwert 10.

Der AUTO-Modus wird durch die erneute Eingabe von AUTO ausgeschaltet und vorübergehend verlassen, wenn nach Vorgabe der Zeilennummer ohne Befehlseingabe RETURN gedrückt wird.

Beispiel:

```
AUTO 10
100 PRINT"HALLO":REM ERSTE ZEILENNUMER 'PER HAND'
110
```

CONT

Format: `CONT`

Funktion: Setzt ein mit der STOP-Taste unterbrochenes Programm ab der Abbruchstelle fort. CONT kann nicht ausgeführt werden, wenn nach dem Abbruch Programmänderungen erfolgten oder - durch im Direkt-Modus gegebene Befehle - Fehlermeldungen erzeugt wurden.

Beispiel:

```
RUN
BREAK IN 100                    (Abbruch mit STOP)
CONT                            (Fortsetzung ab Zeile 100)
```

DELETE

Format: `DELETE [ERSTE ZEILENNUMMER-LETZTE ZEILENNUMMER]`

Funktion: Löscht eine einzige Programmzeile oder einen Bereich von Programmzeilen.

Beispiel:

`DELETE 100`	löscht Zeile 100.
`DELETE 100-1000`	löscht die Zeilen 100 bis 1000.
`DELETE 100-`	löscht Zeile 100 und alle folgenden Zeilen.
`DELETE -1000`	löscht Zeile 1000 und alle vorhergenden Zeilen.

HELP

Format: `HELP`

Funktion: Nach "Absturz" eines Programms und Eingabe von HELP wird die für den Fehler verantwortliche Programmzeile gelistet und der Befehlsteil blinkend markiert, der den Fehler verursachte.

Beispiel:

```
100 PRINT "DIES IST"
110 PRINT 'EIN TEST":REM KEIN DOPPELTES ANFUEHRUNGSZEICHEN

RUN
DIES IST

?SYNTAX ERROR IN 110
READY.
HELP

110 PRINT 'EIN TEST"

READY.
```

KEY

Format: `KEY [NR.DER FUNKTIONSTASTE,"ZEICHENKETTE"]`

Funktion: Mit KEY können die acht Funktionstasten mit beliebigen Zeichenketten belegt werden. Ein Druck auf eine Funktionstaste genügt, um die gesamte Zeichenkette "abzurufen". Wenn Sie die mit HELP beschriftete Funktionstaste belegen wollen, beachten Sie bitte, daß diese die Nummer 8 besitzt.

KEY ohne weitere Angaben zeigt die momentane Belegung aller Funktionstasten.

Alle acht Zeichenketten dürfen zusammen die maximale Länge von 128 Zeichen nicht überschreiten. Zuweisungen können direkt oder über Stringvariablen und Stringfunktionen erfolgen (KEY 1,A$ weist der Funktionstaste F1 die Zeichenkette zu, die "A$" enthält).

Der KEY-Befehl kann im Direkt- und im Programm-Modus verwendet werden!

Beispiele:

```
KEY 1,"DIES IST"
KEY 2,"EIN TEST"
(TASTE F1 DRUECKEN)
DIES IST
(TASTE F2 DRUECKEN)
EIN TEST

KEY
KEY 1,"DIES IST"
KEY 2,"EIN TEST"
...
...
...
KEY 8,...
```

LIST

Format: `LIST [ERSTE ZEILENNUMMER-LETZTE ZEILENNUMMER]`

Funktion: Gibt ein BASIC-Programm - oder Programmteile - in der Reihenfolge der Zeilennummern auf dem Bildschirm aus. Das "Listing" kann mit der COMMODORE-Taste verlangsamt oder mit der Tastenkombination CTRL+S bis zum Drücken einer beliebigen Taste angehalten werden.

Beispiele:

`LIST`	listet das komplette Programm.
`LIST 10Ø`	listet Zeile 100.
`LIST 10Ø-1ØØØ`	listet die Zeilen 100 bis 1000.
`LIST 1ØØØ-`	listet Zeile 1000 und alle folgenden Zeilen.
`LIST -1ØØØ`	Zeile 1000 und alle vorhergehenden Zeilen.

NEW

Format: `NEW`

Funktion: NEW löscht ein im Speicher vorhandenes Programm und die Inhalte aller Variablen. Die Eingabe von NEW ist nur dann sinnvoll, wenn ein völlig neues Programm eingegeben werden soll.

Beispiel:

```
LIST
10Ø PRINT "DIES IST"
11Ø PRINT "EIN TEST"

READY.
NEW

READY.
LIST

READY.
```

RENUMBER

Format: `RENUMBER [NEUE STARTZEILE,ZEILENABSTAND,ALTE STARTZEILE]`

Funktion: RENUMBER numeriert ein Programm oder Teile davon neu durch und ist sinnvoll, wenn zwischen Programmzeilen neue Zeilen eingefügt werden sollen, jedoch aufgrund der Numerierung kein "Platz" vorhanden ist.

RENUMBER ohne weitere Angaben numeriert das komplette Programm im Zehnerabstand durch, die erste Zeile erhält die Zeilennumer zehn. Sowohl die neue erste Zeilennummer als auch der Numerierungsabstand können mit den Parametern NEUE STARTZEILE und ZEILENABSTAND beliebig festgelegt werden.

Mit ALTE STARTZEILE kann festgelegt werden, mit welcher Zeilennummer die Neunumerierung beginnen soll.

RENUMBER korrigiert alle angegebenen Zeilennummer (GOSUB 1000, GOTO 1000) gemäß der neuen Numerierung.

Beispiel:

Folgendes Programm sei gegeben:

```
100 PRINT X
110 X=X+1
120 IF X<1000 THEN GOTO 100
```

a) Numerierung mit Standardwerten

```
RENUMBER
LIST
10 PRINT X
20 X=X+1
30 IF X<1000 THEN GOTO 10

READY.
```

b) Numerierung mit Startzeile 100 in Zwanzigerabständen

```
RENUMBER 100,20
LIST
100 PRINT X
120 X=X+1
140 IF X<1000 THEN GOTO 100

READY.
```

c) Die Zeilen 120- in Fünferschritten mit Startzeile 1000

```
RENUMBER 1000,50,120
LIST
100 PRINT X
1000 X=X+1
1005 IF X<1000 THEN GOTO 100

READY.
```

4.3 Starten, Speichern und Laden von Programmen

Die drei beschriebenen Befehle RUN, LOAD und SAVE können sowohl im Programm- als auch im Direkt-Modus verwendet werden. Interessant ist vorwiegend die Verwendung von LOAD in Programmen. Mit ihm können Programme "nachgeladen" werden. Das im Speicher vorhandene Programm wird durch das nachgeladene Programm ersetzt und dieses nach Beenden des Ladevorgangs automatisch gestartet. Voraussetzung: Das nachzuladende Programm muß kürzer sein als das "Ladeprogramm".

RUN

Format: `RUN [ZEILENNUMMER]`

Funktion: RUN startet ein Programm, beginnend mit der ersten oder einer angegebenen Programmzeile. Alle Variablen werden beim Start mit RUN gelöscht, das heißt, alle numerischen Variablen besitzen den Wert 0 und alle Stringvariablen enthalten keinerlei Inhalt.

Beispiel:

```
10Ø PRINT "DIES IST"
11Ø PRINT "EIN TEST"

RUN                              (Start mit der ersten Programmzeile)

DIES IST
EIN TEST

READY.

RUN 11Ø                          (Programmstart ab Zeile 110)
EIN TEST

READY.
```

SAVE

Format: `SAVE ["FILENAME",GERÄTENUMMER,FLAG]`

Funktion: SAVE speichert ein BASIC-Programm, das sich im Speicher befindet, unter dem eingegebenen Filenamen (maximal 16 Zeichen) auf Cassette (Gerätenummer 1) oder Diskette ab (Gerätenummer 8).

Bei Verwendung der Datasette kann die Angabe eines Filenamens entfallen. Entfällt die Angabe der Geräteadresse, wird automatisch von der Datasette ausgegangen.

Wird als Flag eine 1 angegeben, wird ein späterer Datasetten-Ladevorgang mit der Meldung "FILE NOT FOUND ERROR" abgebrochen. Sinnvoll ist diese Angabe bei dem letzten (!) Programm, das auf einer Cassette gespeichert wird, um eine vergebliche Weitersuche bis zum Bandende zu vermeiden, wenn sich das gesuchte Programm nicht auf der Cassette befindet.

Beispiele:

`SAVE`	speichert ein Programm ohne Namen auf Cassette.
`SAVE "TEST",1`	speichert ein Programm unter dem Namen "TEST" auf Cassette.
`SAVE "TEST",8`	speichert ein Programm unter dem Namen "TEST" auf Diskette.

LOAD

Format: `LOAD ["FILENAME",GERÄTEADRESSE,FLAG]`

Funktion: Lädt ein BASIC-Programm von Cassette oder Diskette. Entfällt beim Laden von Cassette die Angabe des Namens, wird das erste gefundene Programm geladen.

Mit Angabe der Flag-Nummer 1 wird beim Laden von Diskette erreicht, daß ein Maschinenspracheprogramm an jene Adresse im Speicher geladen wird, an der es sich beim Speichern befand.

Beispiele:

`LOAD`	lädt das erste Programm von Cassette.
`LOAD "TEST",1`	lädt das Programm "TEST" von Cassette.
`LOAD "TEST",8`	lädt das Programm "TEST" von Diskette.

VERIFY

Format: `VERIFY [FILENAME,GERÄTEADRESSE,FLAG]`

Funktion: VERIFY vergleicht ein auf Cassette oder Diskette gespeichertes Programm mit dem Original, das sich im Speicher befindet. Wird nach VERIFY keine Fehlermeldung ausgegeben, wurde das Programm fehlerfrei gespeichert. VERIFY ohne Angabe des Filenamens vergleicht das erste, VERIFY mit Angabe eines Namens das angegebene Programm mit dem Programm im Speicher.

Die Angabe Flag (eins) wird nur benötigt, wenn ein gespeichertes Maschinenspracheprogramm mit dem Original im Speicher verglichen wird.

Beispiele:

`VERIFY`	vergleicht das erste Programm von Cassette.
`VERIFY "TEST",1`	vergleicht das Programm "TEST" von Cassette.
`VERIFY "TEST",8`	vergleicht das Programm "TEST" von Diskette.

4.4 Disketten-Befehle

In diesem Abschnitt werden Befehle und Variablen vorgestellt, die nur zusammen mit einem Diskettenluaufwerk eingesetzt werden können. BACKUP dürfte nur für die wenigsten Besitzer eines C16/Plus4 interessant sein, da dieses Kommando nur mit zwei Laufwerken eingesetzt werden kann. Um Mißverständnisse zu vermeiden, werde ich im folgenden bei der Beschreibung des Befehls-Format voraussetzen, daß ein Laufwerk verwendet wird. Ich gehe davon aus, daß wohl kaum ein Leser mehrere Diskettenlaufwerke besitzt. Wenn doch, lesen Sie bitte im Handbuch zu Ihrem Rechner das komplette Befehls-Format nach.

BACKUP

Format: `BACKUP D(NUMMER 1) TO D(NUMMER 2) [ON(GERÄTENUMMER)]`

Funktion: BACKUP kopiert alle Daten von Laufwerk 1 auf die in Laufwerk 2 eingelegte Diskette, die zuvor formatiert wird.

Beispiel:

`BACKUP DØ TO D1`	kopiert von Laufwerk 1 auf Laufwerk 2.
`BACKUP DØ TO D1 ON U9`	kopiert von Laufwerk 1 auf Laufwerk 2, wenn dieses die Gerätenummer 9 besitzt.

COLLECT

Format: `COLLECT`

Funktion: COLLECT schließt alle eventuell offenen Files und korrigiert das Inhaltsverzeichnis entsprechend. Sinnvoll ist dieser Befehl, wenn mehrere Dateien die Länge null besitzen und mit einem Stern markiert sind. Diese Dateien können nicht mehr bearbeitet werden. Nicht geschlossene Dateien werden vorwiegend beim falschen Umgang mit den Dateibefehlen erzeugt (OPEN, CLOSE etc.). COLLECT löscht derart fehlerhafte Dateien.

COPY

Format: `COPY "NAME ORIGINAL" TO "NAME KOPIE"`

Mit COPY kann auf der eingelegten Diskette eine Kopie einer beliebigen Datei erzeugt werden. Da ein Dateiname nur einmal auf einer Diskette verwendet werden kann, muß der Kopie ein anderer Name gegeben werden. NAME ORIGINAL ist der

Name der Original-Datei, NAME KOPIE der Name der zu erstellenden Kopie.

Beispiel:

```
DIRECTORY

10   "ADRESSEN"       PRG
20   "DATEI"          SEQ
600 BLOCKS FREE

READY.
COPY "ADRESSEN" TO "ADR.KOPIE"
DIRECTORY

10   "ADRESSEN"       PRG
20   "DATEI"          SEQ
10   "ADR.KOPIE"      PRG
590 BLOCKS FREE

READY.
```

DIRECTORY

Format: `DIRECTORY [FILENAME]`

Funktion: Listet alle oder bestimmte Dateien und Programme auf, die sich auf der eingelegten Diskette befinden.

Beispiel:

```
DIRECTORY

10   "ADRESSEN"       PRG
20   "DATEI"          SEQ
600 BLOCKS FREE

READY.
```

```
DIRECTORY "DA*"

20  "DATEI"          SEQ
600 BLOCKS FREE

READY.
```

DLOAD

Format: `DLOAD "FILENAME`

Funktion: Lädt das angegebene Programm von Diskette in den Rechnerspeicher. DLOAD ist als einfache Alternative zum LOAD-Befehl anzusehen.

Beispiel:

`DLOAD "TESTPROG` — lädt das Programm "TESTPROG" von Diskette.

DSAVE

Format: `DSAVE "FILENAME`

Funktion: Speichert ein Programm unter dem angegebenen Namen auf Diskette.

Beispiel:

`DSAVE "TESTPROG` — speichert das Programm im Rechnerspeicher unter dem Namen "TESTPROG" auf Diskette.

HEADER

Format: `HEADER "DISKNAME" [,"ID"]`

Funktion: Jede neue Diskette muß vor der ersten Benutzung mit dem Befehl HEADER "formatiert" oder "geheadert" werden. Die Diskette erhält den angegebenen Namen DISKNAME und ein spezielles Kennzeichen aus den beiden Buchstaben, die als ID angegeben werden.

HEADER löscht alle eventuell auf der eingelegten Diskette vorhandenen Daten. Jede Diskette sollte eine andere ID erhalten. HEADER ohne Angabe der ID kann benutzt werden, um - schneller als mit Angabe einer ID - alle Informationen auf einer Diskette zu löschen.

Beispiel:

`HEADER "TESTDISK",Ø1`	formatiert eine Diskette, die den Namen "TESTDISK" und die ID "01" erhält.
`HEADER "TESTDISK"`	löscht alle Daten auf der eingelegten Diskette und gibt ihr den neuen Namen "TESTDISK".

RENAME

Format: `RENAME "NAME ALT" TO "NAME NEU"`

Funktion: Mit RENAME kann eine Datei oder ein Programm umbenannt werden. NAME ALT ist der momentane Dateiname, NAME NEU der gewünschte Name.

Beispiel:

`RENAME "TEST" TO "DEMO"`	gibt der Datei "TEST" den neuen Namen "DEMO".

SCRATCH

Format: `SCRATCH "FILENAME`

Funktion: SCRATCH löscht die angegebene Datei, nachdem die Frage "ARE YOU SURE?" mit "Y" beantwortet wurde.

Beispiel:

```
DIRECTORY

10   "ADRESSEN"        PRG
20   "DATEI"           SEQ
600 BLOCKS FREE

READY.
SCRATCH "ADRESSEN
ARE YOU SURE? Y

READY.
DIRECTORY "DA*"

20   "DATEI"           SEQ
590 BLOCKS FREE

READY.
```

4.5 Graphik-Befehle

Bei der Arbeit im Graphik-Modus sind verschiedene Begriffe zum Verständnis der folgenden Befehle nötig.

1. Graphik-Modus: Modus, in dem die Farbe jedes einzelnen Punktes des Bildschirms individuell festgelegt werden kann.

2. Farbzone oder Bereich: Diese Angabe wird in nahezu jedem Graphik-Befehl benötigt. Sie legt die Farbe fest, die Punkte oder Figuren erhalten. Mit dem Befehl COLOR

können den verschiedenen Bereichen (Rand, Hintergrund, Zeichen) Farben zugewiesen werden. Beim Zeichnen von Punkten oder Figuren wird mit der Angabe Farbzone der "Farbtopf" bestimmt, den der BASIC-Interpreter verwendet. Ist die Angabe der Farbzone optional, wird der Standardwert 1 (Zeichenfarbe 1) verwendet. Beispiel: DRAW ,100,200 entspricht dem Befehl DRAW 1,100,200.

3. Graphik-Cursor: Ebenso wie im Text-Modus steht auch im Graphik-Modus ein Cursor zur Verfügung, der die aktuelle Bildschirmposition anzeigt. Der Graphik-Cursor ist jedoch nicht sichtbar und nur einen einzigen Punkt groß. Mit dem Befehl LOCATE kann der Graphik-Cursor an beliebige Positionen gesetzt werden. In allen Graphik-Befehlen, die die optionale Angabe einer oder mehrerer Koordinaten verlangen, wird eine entfallende Angabe durch die aktuelle Position des Graphik-Cursors gesetzt. Beispiel: Der Graphik-Cursor wird mit LOCATE an Position 100/50 gesetzt. Der Befehl BOX 1,10,30 zeichnet ein Rechteck mit der linken oberen Ecke 10/30. Da die optionale Angabe der linken unteren Ecke im Beispiel entfällt, wird die Position des Graphik-Cursors verwendet und der Befehl besitzt die gleiche Wirkung wie BOX 1,10,30,100,50.

BOX

Format: `BOX [FARBZONE],X1,Y1,X2,Y2,[,[WINKEL],FARBE]]`

Funktion: BOX zeichnet ein Rechteck in der Farbe der angegebenen Farbzone. X1/Y1 markiert die Koordinaten der linken oberen, X2/Y2 der rechten unteren Ecke. Mit der Angabe WINKEL kann das Rechteck um einen beliebigen Winkel gedreht werden. Wird für den Wert FARBE eine 1 angegeben, wird das Rechteck gefüllt.

Beispiel:

`BOX 1,50,70,200,100`	zeichnet ein Rechteck.
`BOX 0,50,70,200,100`	löscht ein Rechteck (zeichnet es in der Farbe des Hintergrunedes).
`BOX 1,50,70,200,100,30,1`	zeichnet und füllt ein um 30 Grad gedrehtes Rechteck.

CHAR

Format: `CHAR [FARBZONE],X,Y [,[STRING] [,FLAG]]`

Funktion: Mit CHAR kann eine Zeichenkette sowohl im Text- als auch im Graphik-Modus an der Position X/Y ausgegeben werden. Die X/Y-Koordinaten entsprechen immer der Bildschirmunterteilung des Text-Modus (X: 0-39 / Y:0-24). Die Zeichenkette wird in der Farbe der angegebenen Farbzone ausgegeben. Wird für FLAG der Wert 1 verwendet, wird die Zeichenkette invers ausgegeben. Im Text-Modus sollte immer die Farbzone eins angegeben werden.

Beispiele:

`CHAR 1,5,10,"TEST"`	gibt den String "TEST" ab Spalte 5 von Zeile 10 aus.
`CHAR 1,10,12,A$,1`	gibt den Inhalt von A$ ab Spalte 10 von Zeile 12 invers aus.

CIRCLE

Format: `CIRCLE [FARBZONE], [X,Y], BREITE, [HÖHE], [WINKEL1], [WINKEL2], [WINKEL], [SCHRITTWEITE]`

Funktion: CIRCLE malt Kreis, Ellipsen oder Kreisaussschnitte. X/Y ist der Kreismittelpunkt. BREITE ist der Radius des Kreises in X- und HÖHE der Kreisradius in Y-Richtung. Sind beide Radien gleich, entsteht ein Kreis, ansonsten eine Ellipse.

Mit WINKEL1 und WINKEL2 kann ein Kreisausschnitt definiert werden, der gezeichnet werden soll. Mit der Angabe WINKEL kann eine Ellipse oder ein Ausschnitt gedreht werden. SCHRITTWEITE entscheidet über die Genauigkeit der Zeichnung. Je kleiner dieser Wert ist, desto exakter ist die Zeichnung (desto langsamer erfolgt jedoch der Zeichenvorgang).

Beispiel:

`CIRCLE 1,100,50,30`	malt einen Kreis mit dem Mittelpunkt 100/50 und dem Radius 30.
`CIRCLE 1,100,50,30,70,,10`	malt eine um 10 Grad gedrehte Ellipse mit dem Mittelpunkt 100/50, dem X-Radius 30 und dem Y-Radius 70.
`CIRCLE 1,100,50,30,,20,40`	malt einen Ausschnitt des Kreises mit dem Mittelpunkt 100/50 und dem Radius 30. Der Ausschnitt beginnt bei 20 und endet bei 40 Grad.

COLOR

Format: `COLOR FARBZONE,FARBE [,HELLIGKEIT]`

Funktion: Mit COLOR können die Farben von Bildschirm-Rand, Bildschirm-Hintergrund und drei Zeichenfarben festgelegt werden, wovon jedoch nur die Zeichenfarbe 1 im Text- oder Hi-Res-Modus verwendet werden kann. Die Zeichenfarben 2 und 3 können sinnvoll im Multicolor-Modus eingesetzt werden. Jede Farbe kann in unterschiedlichen Helligkeitsstufen wiedergegeben werden. Im Graphik-Modus werden durch Angabe der Farbzone 1 (oder 2 bzw. 3 im Multicolor-Modus) Punkte sichtbar (gesetzt), da sie eine andere Farbe als der Hintergrund erhalten. Mit der Angabe 0 werden Punkte unsichtbar (gelöscht), da sie die gleiche Farbe erhalten, die der Hintergrund besitzt und sich von diesem nicht abheben.

Farbzone	Bereich
0	Hintergrund
1	Zeichenfarbe 1
2	Zeichenfarbe 2
3	Zeichenfarbe 3
4	Rand

Farb-Nr.	Farbe	Farb-Nr.	Farbe
1	Schwarz	9	Orange
2	Weiß	10	Braun
3	Rot	11	Gelb-Grün
4	Zyan	12	Rosa
5	Purpur	13	Blau-Grün
6	Grün	14	Hellblau
7	Blau	15	Dunkelblau
8	Gelb	16	Hellgrün

Helligkeit:

0	Sehr dunkel
1	Dunkel
...	
...	
6	Hell
7	Sehr hell

Beispiel:

```
100 GRAPHIC 1,1:REM HI-RES EIN
110 COLOR 0,9,1:REM HINTERGRUND: DUNKLES ORANGE
120 COLOR 1,2,6:REM ZEICHENFARBE1: HELLES WEISS
130 BOX 1,10,10,100,100:REM RECHTECK ZEICHNEN
140 BOX 0,10,10,100,100:REM WIEDER 'LOESCHEN'
```

DRAW

Format: `DRAW [FARBZONE],[X1,Y1] [TO X2,Y2] [TO X3,Y3]...`

Funktion: DRAW zeichnet einen Punkt, eine Linie, ein Dreieck oder ein sonstiges geschlossenes oder offenes Vieleck. In der Reihenfolge der Koordinatenangabe werden alle angegebenen Punkte miteinander verbunden.

Beispiel:

`DRAW 1,100,150`	zeichnet einen Punkt an den Koordinaten 100/50.
`DRAW 1,10,50 TO 70,80`	zeichnet eine Linie vom Punkt 10/50 zum Punkt 70/80.
`DRAW 1,10,50 TO 70,80 TO 100,100`	zieht eine Linie vom Punkt 10/50 zum Punkt 70/80 und von dort aus weiter zum Punkt 100/100.

GRAPHIC

Format: `GRAPHIC MODUS [,1]`

Funktion: GRAPHIC schaltet den Text- oder aber einen von vier verschiedenen Graphik-Modi ein, wobei wahlweise der Bildschirm gelöscht wird (wenn der optionale Parameter 1 angegeben wird).

Modus	Text-/Graphik-Modus
0	Text-Modus
1	Hi-Res-Graphik (320*200 Punkte, Farbe 1)
2	Hi-Res-Graphik + Text-Modus
3	Multicolor-Modus (160*200 Punkte, Farben 1-3)
4	Multicolor-Modus + Text-Modus

In den "gemischten" Modi (2 und 4) wird zwar die Graphik eingeschaltet, die untersten vier Bildschirmzeilen bleiben jedoch weiterhin für den Text-Modus reserviert.

Im Hi-Res-Modus steht eine Auflösung von 320 Punkten in X-Richtung und 200 Punkten in Y-Richtung zur Verfügung. In diesem Modus können die Zeichenfarben 2 und 3 nicht sinnvoll eingesetzt werden (die Farben "verwaschen").

Im Multicolor-Modus beträgt die Auflösung in X-Richtung nur noch 160 Punkte, dafür kann mit allen drei Zeichenfarben zugleich gearbeitet werden.

Nach Einschalten eines der Graphik-Modi stehen zehn Kilobyte weniger an Speicherplatz zur Verfügung als zuvor, die für den "Graphik-Bildschirm" benötigt werden.

Beispiel:

```
GRAPHIC Ø
GRAPHIC 4,1
```

schaltet den Text-Modus ein.
schaltet den Multicolor-Modus ein, löscht den Bildschirm und reserviert die fünf untersten Zeilen für die Verwendung im Text-Modus.

GRAPHIC CLR

Format: `GRAPHIC CLR`

Funktion: Gibt nach dem Ausschalten eines der Graphik-Modi den benötigten Speicherplatz von zehn Kilobyte wieder für Programme und Daten frei.

Beispiel:

```
1ØØ GRAPHIC Ø:REM TEXT-MODUS
11Ø GRAPHIC CLR:REM SPEICHERPLATZ FREIGEBEN
```

LOCATE

Format: `LOCATE X,Y`

Funktion: Setzt den Graphik-Cursor auf die angegebene Position.

Beispiel:

`LOCATE 320,100` setzt den Graphik-Cursor im Hi-Res-Modus auf die Bildschirmmitte.

PAINT

Format: `PAINT [FARBZONE] [,[X,Y] [,1]]`

Funktion: PAINT füllt geschlossene (!) Flächen mit der Farbe der angegebenen Farbzone. Die Koordinaten X/Y müssen (!) einen Punkt innerhalb der Fläche kennzeichnen. Mit Angabe der Farbzone 0 wird die gefüllte Fläche wieder gelöscht (Füllen in der Farbe des Hintergrunds). Die Angabe des optionalen Parameters 1 führt dazu, daß der Füllvorgang beendet wird, wenn beim "Ausmalen" Punkte mit einer anderen als der Hintergrundfarbe erreicht werden.

Beispiel:

```
100 GRAPHIC 1,1:REM HI-RES EIN
110 BOX 1,50,50,200,100:REM RECHTECK
120 PAINT 1,100,70:REM RECHTECK AUSMALEN
130 PAINT 0,100,70:REM VORGANG UMKEHREN
```

SCALE

Format: `SCALE [Ø/1]`

Funktion: SCALE schaltet die "Skalierung" ein (SCALE 1) beziehungsweise aus (SCALE 0). Mit eingeschalteter Skalierung wird der Graphik-Bildschirm sowohl im Hi-Res-Modus als auch im Multicolor-Modus in ein Koordinatensystem unterteilt, das in X-Richtung und in Y-Richtung von 0 bis 1024 reicht.

Um Ihnen von vornherein falsche Hoffnungen zu nehmen: Die Koordinatenangaben ändern sich bei eingeschalteter Skalierung, die Auflösung selbst wird jedoch nicht höher.

Beispiel:

```
1ØØ GRAPHIC 1,1:REM HI-RES
11Ø SCALE 1:REM SKALIERUNG EIN
12Ø CIRCLE 1,512,512,1ØØ:REM KREIS IN DER BILDSCHIRM-
13Ø REM MITTE ZEICHNEN
14Ø SCALE Ø:REM SKALIERUNG AUS
15Ø CIRCLE 1,16Ø,1ØØ,2Ø:REM WEITEREN KREIS IN
16Ø REM BILDSCHIRMMITTE ZEICHNEN
```

SCNCLR

Format: `SCNCLR`

Funktion: SCNCLR löscht den Bildschirm sowohl im Text- als auch in einem der Graphik-Modi.

SSHAPE

Format: `SSHAPE STRINGVAR.,X1,Y1 [,X2,Y2]`

Funktion: SSHAPE speichert einen rechteckigen (!) Ausschnitt des Graphik-Bildschirms in der angegebenen Stringvariablen. X1/Y1 kennzeichnet die linke obere, X2/Y2 die rechte untere Ecke des Ausschnitts.

Angaben zur maximalen Bereichsgröße finden Sie in diesem Buch im Graphik-Teil des BASIC-Kurses.

Beispiel:

```
100 GRAPHIC 1,1:REM HI-RES
110 CIRCLE 1,50,50,10:REM KREIS ZEICHNEN
120 SSHAPE KR$,40,40,60,60:REM KREIS SPEICHERN
```

GSHAPE

Format: `GSHAPE STRINGVAR. [,[X,Y] [,MODUS]]`

Funktion: GSHAPE gibt einen mit SSHAPE "aufgenommenen" Bildschirmausschnitt wieder. Die anzugebenden Koordinaten X/Y kennzeichnen die linke obere Ecke der Wiedergabeposition. Vier verschiedene Wiedergabe-Modi sind vorhanden, deren genaue Funktion im Graphik-Kapitel dieses Buches erläutert wird.

Modus	Funktion
0	Überschreibt Untergrund
1	Überschreibt Untergrund + inverse Wiedergabe
2	ODER-Verknüpfung mit Untergrund
3	UND-Verknüpfung mit Untergrund
4	EOR-Verknüpfung mit Untergrund

Beispiel (Fortsetzung zu SSHAPE-Beispiel):

```
130 FOR I=1 TO 100
140 : GSHAPE KR$,I,100,0:REM KREIS-WIEDERGABE
150 : GSHAPE KR$,I,100,4:REM KREIS LÖSCHEN
160 NEXT I
170 REM BEWEGT EINEN KREIS UEBER DEN BILDSCHIRM
```

4.6 Graphik-Funktionen

RCLR (N)

Funktion: Gibt die Farbnummer an, die der Bereich N (0-4) besitzt.

Beispiel:

```
100 COLOR 1,2,6:REM ZEICHENFARBE1: HELLES WEISS
110 PRINT RCLR(1)

RUN
 2                                  (2=Weiß)

READY.
```

RDOT (N)

Funktion: Gibt die Koordinaten des Graphik-Cursors an. N=0 ergibt die X-Koordinate, N=1 die Y-Koordinate und N=2 die Farbzonen-Nummer, die für den Punkt an der Cursorposition gilt.

Beispiel:

```
100 GRAPHIC 1,1:REM HI-RES
110 LOCATE 160,100:REM CURSOR SETZEN
120 DRAW 1,160,100:REM PUNKT SETZEN
130 PRINT RDOT(0):REM X-KOORDINATE
140 PRINT RDOT(1):REM Y-KOORDINATE
150 PRINT RDOT(2):REM FARBZONE

RUN
 160                          (x)
 100                          (y)
 1                            (Farbzone)

READY.
```

RGR (X)

Funktion: Gibt den gegenwärtigen Graphik-Modus an. X ist ein beliebiger "Füllwert", zum Beispiel 1.

Beispiel:

```
100 GRAPHIC 1,1:REM HI-RES
110 PRINT RGR(1)
120 GRAPHIC 0:REM TEXT-MODUS
130 PRINT RGR(1)

RUN
 1                            (Hi-Res-Modus)
 0                            (Text-Modus)

READY.
```

RLUM (N)

Funktion: Gibt die Farbhelligkeit des Bereichs N an.

Beispiel:

```
100 COLOR 1,2,6:REM HELLES WEISS
110 PRINT RLUM(1)

RUN
 6                                (Helligkeit)

READY.
```

4.7 Musik-Befehle

SOUND

Format: `SOUND STIMME,TONHÖHE,TONDAUER`

Funktion: Mit SOUND können Töne oder Geräusche erzeugt werden. Der C16/Plus4 besitzt drei "Stimmen", von denen jeweils zwei gleichzeitig spielen können. Den Stimmen sind Zahlen zugeordnet, mit denen sie im SOUND-Befehl gezielt angesprochen werden.

Stimme 1: Sinuston
Stimme 2: Sinuston
Stimme 3: Geräusche

Je höher die angegebene TONHÖHE (zwischen 0 und 1023) ist, desto höher ist die Frequenz eines erzeugten Tones. Im "Musik-Kapitel" dieses Buches wird beschrieben, wie die Frequenz einer angegebenen Tonhöhe und aus der Frequenz wiederum die zugehörige Note errechnet wird.

Der Wert TONDAUER muß im Bereich zwischen 0 und 65535 liegen. Tondauer/60 ergibt die Dauer in Sekunden.

VOL

Format: `VOL N`

Funktion: Mit VOL wird die Lautstärke von Tönen und Geräuschen bestimmt, die mit SOUND erzeugt werden. N ist eine Zahl zwischen 0 und 8 (0 = minimale, 8 = maximale Lautstärke).

Beispiel:

```
100 REM ZWEI SINUSTÖNE JEWEILS
110 REM 10 SEKUNDEN SPIELEN
120 VOL 8:REM MAX.LAUTSTAERKE
130 SOUND 1,200,600:REM STIMME 1
140 SOUND 2,400,600:REM STIMME 2
150 VOL 0
```

4.8 Ein-/Ausgabebefehle für Bildschirm und Tastatur

Ein Ausgabebefehl fehlt in der folgenden Übersicht, der Befehl CHAR. Dieser Befehl wurde bereits im Abschnitt über die Graphik-Befehle besprochen.

GET

Format: `GET STRINGVARIABLE`

Funktion: GET fragt die Tastatur ab und weist eine Taste, die gedrückt wird, während der GET-Befehl bearbeitet wird, der angegebenen Variablen zu. GET wartet nicht darauf, daß eine Taste gedrückt wird. Wurde keine Taste gedrückt, übergibt GET einen Leerstring (A$="").

Beispiel:

```
100 REM WARTESCHLEIFE
110 GET A$
120 IF A$="" THEN GOTO 110:REM KEINE TASTE?
130 PRINT A$:REM TASTE AUSGEBEN
```

GETKEY

Format: `GETKEY STRINGVARIABLE`

Funktion: GETKEY wartet auf einen Tastendruck (eine Warteschleife wie bei GET entfällt) und weist die gedrückte Taste der angegebenen Variablen zu.

Beispiel:

```
100 GETKEY A$:REM AUF TASTE WARTEN
110 PRINT A$:REM TASTE AUSGEBEN
```

INPUT

Format: `INPUT ["KOMMENTAR";] VAR1 [,VAR2] [,VAR2]...`

Funktion: INPUT fordert den Benutzer zur Eingabe einer Zahl (numerische Variable) oder einer Zeichenkette (Stringvariable) auf. Die Eingabe muß vom Benutzer mit RETURN beendet werden. Vor der Eingabeaufforderung, dem Fragezeichen, kann ein Kommentar an den Benuzter ausgegeben werden.

Mit einem INPUT-Befehl können mehrere Eingaben - die der Benutzer mit Kommata trennt - mehreren angegebenen - und ebenfalls mit Kommata getrennten - Variablen zugewiesen werden.

INPUT verarbeitet nur Eingaben bis zu einer maximalen Länge von 88 Zeichen. Kommata und Doppelpunkte dürfen in einer (!) Eingabe nicht verwendet werden (das Komma dient der Trennung mehrerer Eingaben).

Beispiel:

```
10Ø INPUT A:REM ZAHL OHNE KOMMENTAR
11Ø INPUT "ZAHL1, ZAHL2";B,C:REM MEHRERE ZAHLEN
12Ø INPUT "ZEICHENKETTE";A$:REM ZEICHENKETTE MIT KOMMENTAR
13Ø PRINT A,B,C,A$

RUN
? 1Ø
ZAHL1, ZAHL2? 2Ø,3Ø
ZEICHENKETTE? OTTO MAIER
1Ø    2Ø    3Ø    OTTO MAIER

READY.
```

PRINT

Format: `PRINT STRING1/ZAHL1 [;,] STRING2/ZAHL2 [;,]...`

Funktion: Mit PRINT können beliebige Zahlen oder Zeichenketten auf dem Bildschirm ausgegeben werden. Mehrere Zeichenketten/Zahlen werden mit einem PRINT-Befehl unmittelbar hintereinander ausgegeben, wenn sie im PRINT-Befehl mit dem Semikolon getrennt werden. Eine formatierte Ausgabe in verschiedenen Bildschirmzonen (die Bildschirmbreite wird in vier Zonen mit je zehn Zeichen unterteilt) erhält man mit dem Komma als Trennzeichen. PRINT ohne weitere Angaben bewirkt einen Zeilenvorschub.

Anstatt einer Zahl kann ein beliebiger numerischer Ausdruck verwendet werden. Der PRINT-Befehl gibt das Ergebnis des Ausdrucks aus.

Beispiel:

```
10Ø A=1Ø:B=2Ø
11Ø PRINT "DIES IST EIN TEST"
12Ø PRINT A*B,A+B

RUN
DIES IST EIN TEST
2ØØ     3Ø

READY.
```

PRINT USING

Format: `PRINT USING "FORMAT"; STRING1/ZAHL1 [;,]...`

Funktion: PRINT USING gestattet die Ausgabe von Zahlen oder Zeichenketten in einem zuvor festgelegten Format.

Beispiele:

1. Mit dem Zeichen "#" wird die Ausgabelänge festgelegt.

```
10Ø PRINT USING "####";"C16/PLUS4:REM 4 ZEICHEN
RUN
C16/
```

2. Der Punkt trennt Vor- und Nachkommastellen. Nachkommastellen werden - wenn nicht vorhanden - mit Nullen aufgefüllt.

```
10Ø PRINT USING "##.##";12.1
RUN
12.Ø1
```

3. An erster oder letzter Stelle des "Formatstrings" kann ein Plus- oder Minuszeichen eingegeben werden. Plus bedeutet, daß bei der Ausgabe von Zahlen das

Vorzeichen mit ausgegeben wird, Minus, daß bei negativen Zahlen das Vorzeichen und bei positiven Zahlen ein Leerzeichen ausgegeben wird.

```
100 PRINT USING "+##.##";12.1
110 PRINT USING "+##.##";-12.1
120 PRINT USING "-##.##";12.1
130 PRINT USING "-##.##";-12.1
RUN
+12.01
-12.01
 12.01
-12.01
```

Der PRINT USING-Befehl bietet eine Unzahl weiterer Formatierungsmöglichkeiten, die Sie im Handbuch nachlesen sollten, da sie den Rahmen dieser Kurzübersicht überschreiten.

4.9 Ausgabe-Funktionen

Alle folgenden Funktionen werden in Verbindung mit dem PRINT-Befehl verwendet.

POS (X)

Funktion: Liefert die aktuelle Cursorspalte. X ist ein beliebiger Wert.

Beispiel:

```
100 CHAR 1,22,10,"":REM CURSOR AUF SPALTE 22
110 PRINT POS(1)
RUN
22

READY.
```

SPC (X)

Funktion: Die nächsten X Zeichen (X = 0 bis 255) werden bei der Ausgabe mit PRINT übersprungen.

Beispiel:

```
100 PRINT "DIES IST";
110 PRINT SPC(5) "EIN TEST"
RUN
DIES IST     EIN TEST

READY.
```

TAB (X)

Funktion: Die nächste Ausgabe erfolgt ab Spalte X (X = 0 bis 255).

Beispiel:

```
100 PRINT TAB(5) "DIES IST";
110 PRINT TAB(20) "EIN TEST"
RUN
     DIES IST       EIN TEST

READY.
```

4.10 Datei-Befehle

Die im folgenden dargestellten Datei-Befehle beziehen sich auf den Umgang mit logischen Dateien. Was eine logische Datei ist und wie sie zur Datenverwaltung eingesetzt wird, finden Sie im diesem Buch im Kapitel über Dateibehandlung.

Format: `OPEN FILENR, GERÄTENR [,SEKUNDÄRADRESSE] ["FILENAME, TYP, MODUS"]`

Funktion: Öffnet eine logische Datei für nachfolgende Schreib- oder Lesezugriffe. Die logische Filenummer ist eine beliebige Zahl zwischen 1 und 255, auf die bei allen späteren Schreib- oder Lesebefehle Bezug genommen wird.

Gerätenummer:

0 Tastatur
1 Datasette
2 RS-232-Schnittstelle
3 Bildschirm
4 Drucker
8 Diskettenlaufwerk

Die Sekundäradresse wird vorwiegend in Zusammenhang mit Datasette, Floppy oder Drucker benötigt.

Sekundäradresse:

Floppy:	Beliebige Zahl zwischen zwei und 14
Datasette:	0=Daten lesen
	1=Daten schreiben
	2=Daten schreiben mit EOT-Markierung

FILENAME: Ein beliebiger Name mit maximal 16 Zeichen. Die Angabe des Filenamens kann bei Verwendung der Datasette entfallen.

TYP: Bei Verwendung der Floppy muß der Dateityp angegeben werden. Bei der Datasette ist dies nicht notwendig, da die Datasette im Gegensatz zur Floppy nur eine Dateiart verwalten kann.

Typ:

P=Programm-Datei
S=Sequentielle Datei
R=Relative Datei
U=User-Datei

MODUS: Ob eine Datei zum Lesen oder Schreiben geöffnet wird, wird bei der Datasette anhand der Sekundäradresse angegeben. Bei Verwendung der Floppy besitzt die frei - im Bereich zwei bis 15 - wählbare Sekundäradresse diese Bedeutung nicht. Die Zugriffsart muß explizit angegeben werden.

Modus:

R=Lesen von Daten
W=Schreiben von Daten
A="Anhängen" von Daten an das Ende einer Datei

Beispiel:

```
100 OPEN 1,1,0,"TEST":REM DATASETTEN-DATEI MIT FILENUMMER
110 REM 1 UNTER DEM NAMEN 'TEST' ZUM LESEN OEFFNEN

100 OPEN 1,1,1,"TEST":REM DATASETTEN-DATEI MIT FILENUMMER
110 REM 1 UNTER DEM NAMEN 'TEST' ZUM SCHREIBEN OEFFNEN

100 OPEN 1,8,2,"TEST,S,R":REM FLOPPY-DATEI MIT FILENUMMER1
110 REM UNTER DEM NAMEN 'TEST' ZUM LESEN OEFFNEN

100 OPEN 1,8,2,"TEST,S,W":REM FLOPPY-DATEI MIT FILENUMMER1
110 REM UNTER DEM NAMEN 'TEST' ZUM SCHREIBEN OEFFNEN
```

In allen Beispielen werden die in diesem Buch ausführlich beschriebenen sequentiellen Dateien verwendet.

CLOSE

Format: `CLOSE FILENUMMER`

Funktion: Schließt eine logische Datei nach Ausführung der gewünschten Schreib-/Lesevorgänge. Die angegebene logische Filenummer entspricht der beim Öffnen der Datei verwendeten Filenummer.

Beispiel:

```
100 OPEN 1,8,2,"TEST,S,W":REM FLOPPY-DATEI MIT FILENUMMER1
110 REM UNTER DEM NAMEN 'TEST' ZUM SCHREIBEN OEFFNEN
120 CLOSE 2:REM DATEI MIT FILENUMMER 2 SCHLIESSEN
```

GET#/PRINT#/INPUT#

Format:
```
GET#FILENUMMER,VARIABLE
PRINT#FILENUMMER,VARIABLE
INPUT#FILENUMMER,VARIABLE
```

Funktion: Diese Befehle entsprechen den Befehlen GET#, PRINT# und INPUT#. Der einzige Unterschied besteht in der Angabe jener logischen Filenummer, die beim Öffnen der Datei verwendet wurde.

Beispiel:

```
100 OPEN 1,8,2,"TEST,S,W":REM FLOPPY-DATEI MIT FILENUMMER1
110 REM UNTER DEM NAMEN 'TEST' ZUM SCHREIBEN OEFFNEN
120 PRINT#1,"DIES IST EIN TEST":REM SCHREIBEN
130 CLOSE 2:REM DATEI SCHLIESSEN
140 :
150 OPEN 1,8,2,"TEST,S,R":REM FLOPPY-DATEI MIT FILENUMMER1
160 REM UNTER DEM NAMEN 'TEST' ZUM LESEN OEFFNEN
170 INPUT#2,A$:REM LESEN
180 PRINT A$:REM AUSGEBEN
190 CLOSE 2:REM DATEI SCHLIESSEN
```

CMD

Format:
```
CMD FILENUMMER
```

Funktion: Der Befehl CMD leitet alle Ausgaben, die normalerweise auf den Bildschirm erfolgen, auf eine geöffnete logische Datei um. Das heißt, statt mit PRINT# kann eine Ausgabe auf

Drucker oder Diskette auch mit PRINT erfolgen, wenn zuvor die Ausgabe auf die Drucker-Datei umgelenkt wurde.

Nach dem Beenden der Ausgabe und vor dem Schließen der Datei sollte immer ein PRINT#-Befehl ohne weitere Angaben erfolgen!

Das zur Erläuterung der Befehle GET#, PRINT# und INPUT# vorgestellte Demoprogramm lautet bei Verwendung von CMD:

```
100 OPEN 1,8,2,"TEST,S,W":REM FLOPPY-DATEI MIT FILENUMMER1
110 REM UNTER DEM NAMEN 'TEST' ZUM SCHREIBEN OEFFNEN
120 CMD 1:REM AUSGABE AUF LOG.DATEI UMLENKEN
130 PRINT "DIES IST EIN TEST":REM SCHREIBEN
140 PRINT#1:REM ZEILENVORSCHUB
150 CLOSE 2:REM DATEI SCHLIESSEN
160 :
170 OPEN 1,8,2,"TEST,S,R":REM FLOPPY-DATEI MIT FILENUMMER1
180 REM UNTER DEM NAMEN 'TEST' ZUM LESEN OEFFNEN
190 INPUT#2,A$:REM LESEN
200 PRINT A$:REM AUSGEBEN
210 CLOSE 2:REM DATEI SCHLIESSEN
```

Wirklich sinnvoll ist der CMD-Befehl zur Umleitung der Ausgabe eines Programmlistings auf einen angeschlossenen Drucker.

```
100 OPEN 1,4:REM DRUCKER-DATEI OEFFNEN
110 CMD 1:REM AUSGABE AUF DRUCKER
120 LIST:REM PROGRAMMLISTING AUSGEBEN
130 PRINT#1:REM ZEILENVORSCHUB
140 CLOSE 1:REM DATEI SCHLIESSEN
```

Wenn Sie diese Zeilen im Direkt-Modus eingeben, können Sie außerdem das Inhaltsverzeichnis einer Diskette ausdrucken, wenn Sie dieses zuvor mit dem Befehl LOAD"$",8 in den Rechnerspeicher laden.

4.11 Reservierte Datei-Variablen

DS und DS$

Funktion: Mit den Variablen DS und DS$ kann überprüft werden, ob bei Diskettenzugriffen Fehler auftraten. Trat kein Fehler auf, enthält DS den Wert 0, ansonsten eine Fehlernummer, die in Ihrem Floppy-Handbuch erläutert ist.

DS$ enthält die Fehlermeldung im "Klartext".

Beispiel:

```
100 OPEN 2,8,2,"TEST,S,W":REM DATEI OEFFNEN
110 IF DS<>0 THEN CLOSE 2:PRINT DS$:GETKEY A$:GOTO 100
120 REM WENN BEI OEFFNEN FEHLER AUFTRATEN:
130 REM - DATEI SCHLIESSEN
140 REM - FEHLERMELDUNG AUSGEBEN
150 REM - AUF TASTE WARTEN
160 REM - OEFFNEN WIEDERHOLEN
```

Wenn beim Öffnen der Datei ein Fehler auftrat, schließt dieses Demoprogramm die Datei wieder, gibt dem Benutzer die Fehlermeldung DS$ aus, und wartet auf eine Taste, bevor das Öffnen wiederholt wird.

Der Benutzer erhält dadurch Gelegenheit, die Fehlerursache (Schreibschutz, Diskette nicht eingelegt etc.) zu beseitigen.

ST

Funktion: Die Statusvariable ST wird in der Praxis vorwiegend dazu benutzt, das Ende einer Datei zu erkennen, die gelesen wird. Im Normalfall besitzt ST den Wert 0, am Dateiende den Wert 64.

Beispiel:

```
100 DIM A$(100):REM ARRAY DIMENSIONIEREN
110 OPEN 2,8,2,"TEST,S,R":REM DATEI OEFFNEN
120 DO UNTIL ST=64
130 : INPUT#2,A$(I)
140 : I=I+1
150 LOOP
```

4.12 Kontrolle des Programmablaufs

DO

Format:

```
DO [UNTIL/WHILE BEDINGUNG(EN)]
  BEFEHL1
  BEFEHL2
  [IF BEDINGUNG THEN EXIT]
  BEFEHLN
LOOP [UNTIL/WHILE BEDINGUNG(EN)]
```

Funktion: Diese äußerst komplexe Programmstruktur führt die zwischen DO und LOOP aufgeführten Befehle solange aus, bis (UNTIL) eine oder mehrere Bedingungen erfüllt sind, oder aber solange (WHILE) die Bedingung(en) erfüllt ist (sind).

WHILE und UNTIL können sowohl am Schleifenanfang (hinter DO) als auch am Schleifenende (hinter LOOP) verwendet werden. Der Unterschied: Wird der jeweilige Befehl am Schleifenanfang eingesetzt, wird die Bedingung geprüft, bevor (!) die Schleifenbefehle zum ersten Mal ausgeführt werden.

Wird WHILE beziehungsweise UNTIL am Schleifenende verwendet, werden die Befehle innerhalb der Schleife auf jeden Fall mindestens einmal ausgeführt, da die Prüfung der Bedingung erst am Schleifenende erfolgt.

Diese Schleifenstruktur wird noch komplexer durch die Möglichkeit, die Schleife vorzeitig mit dem Befehl EXIT zu verlassen, der üblicherweise zusammen mit dem IF-Befehl verwendet wird.

Beispiele:

```
100 X=10
110 DO UNTIL X>=10:REM PRUEFUNG AM SCHLEIFENANFANG
120 : PRINT X
130 : X=X+1
140 LOOP
RUN

READY.

100 X=10
110 DO
120 : PRINT X
130 : X=X+1
140 LOOP UNTIL X>=10:REM PRUEFUNG AM SCHLEIFENENDE
RUN
 10

READY.

100 X=10
110 DO UNTIL X=100
120 : PRINT X
130 : X=X+1
140 : IF X=12 THEN EXIT:REM VORZEITIG VERLASSEN
150 LOOP
RUN
 10
 11
 12

READY.
```

FOR

Format: `FOR VAR=STARTWERT TO ENDWERT [STEP SCHRITTWEITE]`

Funktion: DO-Schleifen werden eingesetzt, wenn vor Eintritt in die Schleife nicht bekannt ist, wie oft diese durchlaufen werden sollen. FOR-Schleifen setzen voraus, daß exakt feststeht, wie oft eine Schleife durchlaufen werden soll. Angegeben wird eine beliebige numerische Variable, der ein Anfangswert gegeben wird (VAR=STARTWERT). Zusätzlich wird ein Endwert festgelegt (ENDWERT).

Nach jedem Schleifendurchgang wird der Wert der angegebenen Variablen um eins erhöht und mit dem Endwert verglichen. Ist der Endwert überschritten, erfolgt erneut ein Schleifendurchgang, ansonsten wird die Schleife verlassen.

Wenn gewünscht, kann eine ganzzahlige "Schrittweite" angegeben werden, die angibt, um welchen Betrag die Schleifenvariable nach jedem Durchgang erhöht (positive Schrittweite) oder erniedrigt (negative Schrittweite) wird.

Beispiele:

```
100 FOR I=1 TO 3
110 : PRINT I
120 NEXT I
RUN
 1
 2
 3

READY.

100 FOR I=5 TO 1 STEP -2
110 : PRINT I
120 NEXT I
```

```
RUN
 5
 3
 1

READY.
```

GOSUB/RETURN

Format: GOSUB ZEILENNUMMER
und RETURN

Funktion: GOSUB setzt die Programmbearbeitung mit der angegebenen Programmzeile fort. Die folgenden Befehle werden bearbeitet, bis ein RETURN-Befehl vorliegt. RETURN setzt die weitere Bearbeitung an jener Stelle im Programm fort, von der aus der "Unterprogrammaufruf" erfolgte.

Beispiel:

```
100 GOSUB 1000:REM UNTERPROGRAMM AUFRUFEN
110 GOSUB 1000:REM UNTERPROGRAMM AUFRUFEN
120 END:REM PROGRAMM BEENDEN
130 :
1000 PRINT "DIES IST EIN TEST"
1010 RETURN:REM RUECKKEHR AUS UNTERPROGRAMM

RUN
DIES IST EIN TEST
DIES IST EIN TEST

READY.
```

GOTO

Format: `GOTO ZEILENNUMMER`

Funktion: GOTO ermöglicht die Unterbrechung eines linearen Programmablaufs. GOTO setzt die Programmbearbeitung mit der angegebenen Programmzeile fort.

Beispiel:

```
10Ø PRINT "DIES"
11Ø GOTO 14Ø:REM SPRUNG ZU ZEILE 14Ø
12Ø PRINT "IST"
13Ø PRINT "EIN"
14Ø PRINT "TEST"
RUN
DIES
TEST

READY.
```

ON

Format: `ON ZAHL GOTO/GOSUB ZEILENNR.1 [,ZEILENNR.2]...`

Funktion: ON wird oft als Ersatz für mehrere einander folgende GOTO- oder GOSUB-Befehle verwendet. Abhängig von der angegebenen Zahl oder dem Wert des angegebenen Ausdrucks wird zu einer der aufgeführten Programmzeilen (GOTO...) oder Unterprogramme (GOSUB...) verzweigt.

Ist der Wert des Ausdrucks 1, wird zur ersten angegebenen Zeilen verzweigt, ist der Wert 2, wird zur zweiten Zeilennummer verzweigt und so weiter.

Beispiele:

```
100 I=3
110 ON I GOTO 200,300,400
200 PRINT "ZEILE 200"
300 PRINT "ZEILE 300"
400 PRINT "ZEILE 400"
RUN
ZEILE 400

READY.

100 FOR I=1 TO 2
110 : ON I GOSUB 200,300
120 NEXT I
130 END
140 :
200 PRINT "UNTERPROGRAMM 1"
210 RETURN
220 :
300 PRINT "UNTERPROGRAMM 2"
310 RETURN
RUN
UNTERPROGRAMM 1
UNTERPROGRAMM 2

READY.
```

IF

Format: `IF BEDINGUNG(EN) THEN BEFEHL(E) [:ELSE BEFEHL(E)]`

Funktion: Führt einen oder mehrere im THEN-Zweig aufgeführte Befehle nur aus, wenn die angegebene(n) Bedingung(en) erfüllt ist (sind). Optional kann ein ELSE-Zweig angegeben werden, dessen Befehle nur dann ausgeführt werden, wenn die angegebene(n) Bedingung(en) nicht erfüllt ist (sind).

Bedingungen können beliebige Kombinationen arithmetischer Ausdrücke oder Stringausdrücke enthalten. Mehrere Bedingungen können mit beliebigen logischen Operatoren verknüpft werden.

IF-Befehle können ineinander verschachtelt werden.

Beispiel:

```
100 INPUT "ZAHL";Z
110 IF Z<10 THEN PRINT "<10":ELSE PRINT ">=10"
RUN
ZAHL? 9
<10

READY.
```

END

Format: `END`

Funktion: END beendet den Programmablauf. Ein Programm ist beendet, wenn entweder die letzte Programmzeile ausgeführt wurde oder der Befehl END bearbeitet wird. END ist vor allem nützlich, um eine ungewollte Bearbeitung eines Unterprogramms zu verhindern, das sich am Programmende befindet.

Beispiel:

```
100 GOSUB 1000:REM UNTERPROGRAMM AUFRUFEN
110 GOSUB 1000:REM UNTERPROGRAMM AUFRUFEN
120 END:REM PROGRAMM BEENDEN
130 :
1000 PRINT "DIES IST EIN TEST"
1010 RETURN:REM RUECKKEHR AUS UNTERPROGRAMM
```

REM

Format: `REM KOMMENTAR`

Funktion: REM dient der Kommentierung von Programmen. Alle dem Befehl REM folgenden Zeichen werden überlesen und besitzen keinerlei Einfluß auf den Programmablauf.

Beispiele zu diesem Befehl erübrigen sich, da er in diesem Buch ständig angewandt wurde.

WAIT

Format: `WAIT SPEICHERSTELLE,WERT1 [,WERT2]`

Funktion: WAIT hält den Programmablauf an, bis der Inhalt einer angegebenen Speicherstelle einen bestimmten Wert annimmt.

Der Inhalt der Speicherstelle wird zuerst - wenn angegeben - mit dem optionalen "Wert2" EOR-verknüpft. Anschließend folgt eine UND-Verknüpfung mit "Wert1". Solange das Ergebnis dieser Verknüpfungen null ergibt, wird die Abfrage der Speicherstelle wiederholt.

Beispiel:

```
100 REM AUF EINE TASTE WARTEN
110 REM SPEICHERSTELLE 198 BESITZT
120 REM DEN INHALT 64, WENN KEINE
130 REM TASTE GEDRUECKT IST
140 :
150 WAIT 198,191,64:REM AUF TASTE WARTEN
160 :
170 REM WENN INHALT VON 198 GLEICH 64:
180 REM 64=%01000000 U. 191=%10111111
190 REM %01000000(ODER)$01000000=%00000000
200 REM %00000000(UND) $10111111=%00000000
210 REM ERGEBNIS NULL => WEITER PRUEFEN
```

4.13 Fehlersuche und Fehlerbehandlung

STOP/CONT

Format: `STOP`
und `CONT`

Funktion: STOP unterbricht das laufende Programm, das anschließend mit CONT ab der Abbruchstelle fortgesetzt werden kann (wenn inzwischen weder das Programm geändert noch eine Fehlermeldung erzeugt wurde).

STOP und CONT sind hilfreich beim Austesten eines Programms, zum Beispiel um an kritischen Programmstellen im Direkt-Modus die Inhalte verschiedener Variablen zu überprüfen, bevor das Programm fortgesetzt wird.

Der Befehl STOP wird im Programm eingefügt, CONT im Direkt-Modus eingegeben.

Beispiel:

```
10Ø X=Ø
11Ø X=X+1
12Ø STOP:REM UNTERBRECHUNG
13Ø X=X+1
RUN
BREAK IN 12Ø                    (Meldung der Unterbrechung)
PRINT X                         (Prüfung des Inhaltes von "X")
 1

READY.
CONT                            (Fortsetzung des Programms)
READY.
PRINT X
 2

READY.
```

TRON/TROFF

Format: TRON
und TROFF

Funktion: Einschalten (TRON) beziehungsweise Abschalten (TROFF) des "Trace"-Modus. Wird ein Programm im Trace-Modus abgearbeitet, werden auf dem Bildschirm die Nummern der gerade bearbeiteten Programmzeilen ausgegeben. Im Trace-Modus kann daher der Ablauf eines Programms problemlos verfolgt werden (wann wird welches Unterprogramm aufgerufen oder ein GOTO-Befehl ausgeführt, der zu einem anderen Programmteil verzweigt?).

Obwohl beide Befehle sowohl im Programm- als auch im Direkt-Modus gegeben werden können, wird TRON üblicherweise an jener Stelle in das Programm eingefügt, ab der Sie den weiteren Ablauf kontrollieren wollen. Bis zur Ausführung des Befehls TRON läuft das Programm wie gewohnt ab. Mit TROFF kann der Trace-Modus jederzeit wieder abgeschaltet werden.

Beispiel:

```
100 TRON:REM TRACE-MODUS EIN
110 FOR I=1 TO 5
120 : PRINT I
130 NEXT I
140 TROFF:REM TRACE-MODUS AUS
RUN
[100] [110] [120]  1
[130] [120]  2
[130] [120]  3
[130] [120]  4
[130] [120]  5
[130] [140]
READY.
```

TRAP/RESUME und EL/ER/ERR$(ER)

Format: `TRAP ZEILENNUMMER`
und `RESUME [ZEILENNUMMER/NEXT]`

Funktion: TRAP verzweigt zur angegebenen Zeilennummer, wenn während eines Programmlaufs ein Fehler auftritt, üblicherweise zu einer sogenannten "Fehlerbehandlungsroutine". Die Variable EL enthält die Zeilennummer, in der der Fehler auftrat, die Variable ER die Nummer des Fehlers (siehe Tabelle im Handbuch). Die Stringvariable ERR$(ER) enthält die Fehlermeldung im Klartext.

Fehler, die in der Fehlerbehandlungsroutine selbst auftreten, das heißt in jenem Programmteil, zu dem mit TRAP beim Auftreten eines Fehlers verzweigt wird, können nicht "abgefangen" werden.

Mit dem Befehl RESUME kann der Programmablauf auf drei verschiedene Arten wiederaufgenommen werden.

1. RESUME setzt das Programm mit jenem Befehl fort, der den Fehler verursachte.

2. RESUME NEXT setzt das Programm mit dem Befehl fort, der jenem folgt, der den Fehler verursachte.

3. RESUME ZEILENNUMMER setzt das Programm mit der angegebenen Zeilennummer fort.

Beispiel:

```
100 TRAP 200:REM FEHLERBEHANDLUNG AB 200
110 INPUT "ZAHL";Z
120 PRINT 10/Z
130 END
140 :
```

```
200 REM FEHLERROUTINE
210 PRINT ERR$(ER):REM FEHLERMELDUNG AUSGEBEN
220 PRINT "WIEDERHOLEN (J=JA) ?"
230 GETKEY A$
240 IF A$="J" THEN RESUME 110:ELSE RESUME NEXT
```

Dieses Demoprogramm erwartet von Ihnen die Eingabe einer Zahl, anschließend wird 10 durch die eingegebene Zahl dividiert. Wenn Sie 0 eingeben, führt die Division 10/0 zur Fehlermeldung "DIVISION BY ZERO", die von Zeile 210 ausgegeben wird.

Wenn Sie auf die Frage "WIEDERHOLEN (J=JA) ?" mit "J" antworten, bewirkt RESUME 110 die Fortsetzung des Programms mit Zeile 110 und die Eingabe der Zahl wird wiederholt. Ansonsten wird das Programm mit dem nächsten Befehl (END) fortgesetzt.

4.14 Behandlung von Zeichenketten

Die folgenden Stringfunktionen ergeben immer eine Zeichenkette, die wiederum beliebigen Stringvariablen zugewiesen werden kann.

CHR$(X)

Funktion: Liefert das dem Code X zugeordnete Zeichen.

Beispiel:

```
100 PRINT CHR$(147):REM BILDSCHIRM LOESCHEN
110 PRINT CHR$(65):REM A
120 PRINT CHR$(66):REM B
RUN
A
B
```

HEX$(N)

Funktion: Wandelt eine ganze Zahl zwischen 0 und 65535 in die Hexadezimalform um und übergibt die jeweilige Hexadezimalzahl als String.

Beispiel:

```
100 PRINT HEX$(1024)
RUN
0400
```

LEFT$(X$,X)

Funktion: Ergibt eine Zeichenkette, die aus den X letzten Zeichen des Strings X$ besteht.

Beispiel:

```
100 A$="DIES IST EIN TEST"
110 PRINT LEFT$(A$,8)
RUN
EIN TEST
```

RIGHT$(X$,X)

Funktion: Ergibt eine Zeichenkette, die aus den ersten X Zeichen des Strings X$ besteht.

Beispiel:

```
100 A$="DIES IST EIN TEST"
110 PRINT RIGHT$(A$,8)
RUN
DIES IST
```

MID$(X$,S,X)

Funktion: Ergibt eine Zeichenkette, die aus den ersten X Zeichen des Strings X$ ab Zeichen Stelle S besteht.

Beispiel:

```
100 A$="DIES IST EIN TEST"
110 PRINT MID$(A$,6,7):REM ZEICHEN 6 BIS 12
RUN
IST EIN
```

STR$(X)

Funktion: Wandelt die Zahl X in eine Zeichenkette um.

Beispiel:

```
100 A$=STR$(100)
110 PRINT A$
RUN
 100
```

Die folgenden numerischen Funktionen benutzen als "Argumente" Zeichenketten. Alle Funktionen ergeben als Ergebnis jedoch eine Zahl, die zum Beispiel einer numerischen Variablen zugewiesen werden kann.

ASC(X$)

Funktion: Liefert den Code, der dem ersten Zeichen von X$ zugeordnet ist.

Beispiel:

```
100 PRINT ASC("A")
110 PRINT ASC("B")
RUN
 65
 66
```

DEC(H$)

Funktion: Übergibt den hexadezimalen Wert einer Zeichenkette.

Beispiel:

```
100 PRINT DEC("0400")
RUN
 1024
```

VAL(X$)

Funktion: Übergibt den numerischen Wert einer Zeichenkette.

Beispiel:

```
100 A$="12"
110 B$="TEST"
120 C$="2TEST"
130 PRINT VAL(A$),VAL(B$),VAL(C$)
RUN
 12     0      2
```

INSTR(STRING1,STRING2 [,START])

Funktion: INSTR prüft, ob ein String in einem anderen enthalten ist, zum Beispiel "BACH" in "FISCHBACHWEG". INSTR übergibt die Nummer des Zeichens, ab dem "String2" in "String1" enthalten ist, beziehungsweise den Wert 0, wenn keine Übereinstimmung festgestellt wird.

Mit dem Wert "Start" kann angegeben werden, ab welcher Position "String1" nach "String2" durchsucht werden soll.

Beispiel:

```
100 A$="FISCHBACHWEG"
110 B$="CH"
120 PRINT INSTR(A$,B)
130 PRINT INSTR(A$,B$,6)
140 PRINT INSTR(A$,B$,10)
RUN
 4
 8
 0
```

4.15 Variablen

LET

Format: `LET VARIABLE=FORMEL`

Funktion: Wertzuweisung an eine Variable. Mit LET wird einer numerischen Variablen ein Zahlenwert zugewiesen, einer Stringvariablen eine Zeichenkette. "Formel" ist ein beliebiger Ausdruck, der jedoch vom gleichen Typ sein muß wie die angegebene Variable.

Zum Beispiel ist die Zuweisung LET A="OTTO" nicht möglich, da der numerischen Variablen A keine Zeichenkette zugewiesen werden kann. Ebensowenig ist die Zuweisung LET A$=10

möglich (jedoch: LET A$=STR$(10), da STR$ eine Zeichenkette ergibt).

Das Wort LET kann weggelassen werden.

Beispiel:

```
LET A=1Ø oder A=1Ø
```

DIM

Format: `DIM VARIABLE (ZAHL [,ZAHL]...),(VARIABLE...)`

Funktion: DIM reserviert Speicherplatz für die benötigte Anzahl an Arrayvariablen (oder auch "Feldvariablen"). Ein Array kann ein- oder mehrdimensional sein. Auf die einzelnen Arrayvariablen wird über ihre Indexzahlen zugegriffen. Der erste Index ist immer 0, der letzte Index gleich der bei der Dimensionierung angegebenen Zahl.

Da große Arrays die verfügbare Speicherkapazität schnell überschreiten können, ist der benötigte Platzbedarf außerordentlich wichtig.

Integerarray (DIM X%(...)): 2 Byte pro Variable.
Fließkommaarray (DIM X(...)): 5 Byte pro Variable.
Stringarray (DIM X$(...)): 3 Byte pro Variable.

Bei Stringarrays kommt hinzu, daß außer diesem bereits bei der Dimensionierung benötigten Platz zusätzlich Platz benötigt wird, wenn eine Zuweisung erfolgt, und zwar zwei Byte plus ein Byte für jedes Zeichen der zugewiesenen Zeichenkette.

DIM A$(10): Eindimensionales Stringarray.

A$(0)
A$(1)
...
...
...
A$(9)
A$(10)

DIM X%(10,10): Zweidimensionales Integerarray.

X%(0,0)	X%(0,1)	...	X%(0,9)	X%(0,10)
X%(1,0)	X%(1,1)	...	X%(1,9)	X%(1,10)
		...		
		...		
X%(9,0)	X%(9,1)	...	X%(9,9)	X%(9,10)
X%(10,0)	X%(10,1)	...	X%(10,9)	X%(10,10)

Beispiel:

```
100 DIM X%(10,10)
110 FOR I=0 TO 10
120 : FOR J=0 TO 10
130 : X%(I,J)=J
140 : PRINT X%(I,J)
150 : NEXT J
160 NEXT I
RUN
 0 1 2 3 4 5 6 7 8 9 10          (X%(0,0) bis (X%(0,10))
 0 1 2 3 4 5 6 7 8 9 10          (X%(1,0) bis (X%(1,10))
...
...
 0 1 2 3 4 5 6 7 8 9 10          (X%(9,0) bis (X%(9,10))
 0 1 2 3 4 5 6 7 8 9 10          (X%(10,0) bis (X%(10,10))
```

4.16 Data-Zeilen

READ/DATA/RESTORE

Format: `READ VAR1,VAR2...`
und `DATA ELEMENT1,ELEMENT2...`
und `RESTORE ZEILENNUMMER`

Funktion: Zuweisung einer Liste von numerischen oder Stringelementen an Variablen. Wenn Sie den Umgang mit Datasetten- oder Disketten-Dateien vermeiden wollen, bieten Ihnen diese Befehle eine Alternative zur dauerhaften Datenspeicherung.

Beliebige Daten können in Form sogenannter "Data-Zeilen" im Programm abgelegt werden. Mit dem READ-Befehl lassen sich die Elemente beliebigen Variablen zuweisen.

Beispiel:

```
100 DIM A$(5)
110 FOR I=1 TO 5
120 : READ A$(I)
130 : PRINT A$(I)
140 NEXT I
150 :
200 REM DATA-ZEILEN
210 DATA "OTTO","WILLI","GERD"
220 DATA "MANFRED","STEFAN"
```

Dieses Programm enthält eine kleine Adressenliste in den Data-Zeilen am Programmende. Jede Data-Zeile beginnt mit dem Wort DATA, dem eine Liste von Elementen folgt. Zahlen können unmittelbar eingetippt werden, Zeichenketten werden wie üblich in Anführungszeichen eingegeben.

Eine Data-Zeile kann beliebig viele Elemente enthalten. Die Elemente müssen voneinander durch Kommata getrennt werden.

Nach dem Starten des Programms wird in einer Schleife fünfmal der Befehl READ A$(I) bearbeitet. Jeder READ-Befehl veranlaßt den BASIC-Interpreter, nach dem nächsten "Data-Element" zu suchen.

Der erste READ-Befehl liest das erste Data-Element "OTTO" ein und weist es der Stringvariablen A$(1) zu. Der zweite READ-Befehl liest das zweite Element "WILLI" ein, das der Variablen A$(2) zugewiesen wird, und so weiter.

Der Interpreter merkt sich die Programmstelle, an der das letzte Element gefunden wurde und durchsucht das Programm beim nächsten READ-Befehl ab der Position des zuletzt eingelesenen Elements.

In der Praxis tritt nun häufig das Problem auf, daß bereits gelesene Datas erneut gelesen werden sollen. Mit dem Befehl RESTORE ZEILENNUMMER kann angegeben werden, ab welcher Zeilennummer die nächste Suche beginnen soll.

Beispiel:

```
100 DIM A$(5)
110 FOR I=1 TO 5
120 : READ A$(I)
130 : PRINT A$(I)
140 NEXT I
150 RESTORE 220:REM AB ZEILE 220 LESEN
160 FOR I=1 TO 3
170 : READ A$
180 : PRINT A$
190 NEXT I
200 :
210 REM DATA-ZEILEN
220 DATA "OTTO","WILLI","GERD"
230 DATA "MANFRED","STEFAN"
```

4.17 Verbindung zur Maschinensprache

SYS

Format: `SYS STARTADRESSE`

Funktion: Ruft ein Maschinenprogramm auf, das ab der angegebenen Adresse im Speicher liegt. Der Aufruf entspricht dem Befehl JSR STARTADRESSE, also einem Unterprogrammaufruf.

POKE

Format: `POKE ADRESSE,WERT`

Funktion: Schreibt einen Ein-Byte-Wert (0 bis 255) in die angegebene Speicherstelle.

PEEK

Format: `PEEK (ADRESSE)`

Funktion: Liest einen Ein-Byte-Wert aus der angegebenen Speicherstelle.

USR

Format: `USR(WERT)`

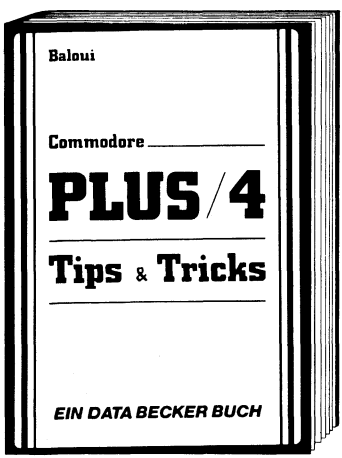
Funktion: Übergibt einem Maschinenspracheprogramm einen Wert im Fließkomma-Format und ruft dieses zugleich auf. Der USR-Vektor ($0501/$0502) muß vor dem Aufruf mit USR auf die Startadresse der Maschinenroutine "verbogen" werden.

Die Übergabe erfolgt im Fließkomma-Akkumulator 1 (FAC 1). Das Maschinenprogramm kann mit diesem Wert beliebige Berechnungen vornehmen. Das Maschinenprogramm muß mit RTS enden, und das Ergebnis der mit dem Wert vorgenommenen

Berechnungen muß ebenfalls im Fließkomma-Format in FAC 1 übergeben werden.

Bücher zu C16, C116 und Plus/4

Plus/4 Tips & Tricks ist eine Sammlung von Anregungen, Ideen und fertigen Lösungen zur Programmierung und Anwendung Ihres Plus/4.
Alles, was Sie brauchen, um eigene Programme zu schreiben.



Aus dem Inhalt:

- Hinweise zur Eingabe der Programme
- Verwendung der Datasette
- Anwenderprogramme
  Spiele
  Graphik
  Text- und Dateiverarbeitung
  Kfz-Überwachung
  Mathematik
  Hardcopy
  Merge
  Shapeeditor
- Die wichtigsten Tips & Tricks
- Von BASIC bis ASSEMBLER
- Die wichtigsten Zeropage-Adressen
- Routinen des Betriebssystems und des BASIC-Interpreters
- Tips & Tricks für Fortgeschrittene

Baloui
Plus/4 Tips & Tricks
222 Seiten, DM 29,–
ISBN 3-89011-203-X

Löffelmann · Plenge

DAS GRAFIK-BUCH

zu

C 16
C 116
Plus/4

EIN DATA BECKER BUCH

Bücher zu C16, C116 und Plus/4

Ein Team von Grafikspezialisten deckt wirklich alle Geheimnisse des C16, C116 und Plus/4 auf. Von den grundlegenden Grafikbefehlen bis hin zu nützlichen Utilities enthält dieses Buch alles, was Sie schon immer über Computergrafik wissen wollten.

Aus dem Inhalt:

- theoretische Grundlagen
- Linienberechnung in Assembler
- Einführung in die Hardware
- hochauflösende Grafik
- komfortabler Charactereditor
- Computer Aided Design
- Statistik
- Funktionsplotter
- Hardcopyroutinen
- Posterhardcopy auf allen Star-Druckern
- Shape-Editor

Plenge, Löffelmann
Grafikbuch zu C16, C116 und Plus/4
232 Seiten, DM 29,–
ISBN 3-89011-205-6

Vüllers

C 16
C 116
Plus/4

Maschinensprache

EIN DATA BECKER BUCH

Bücher zu C16, C116 und Plus/4

Viele Computerbenutzer, die bislang die Hemmschwelle zur Maschinensprache nicht überwunden haben, erhalten hier die perfekte Einführung in den Befehlssatz des 7501-Mikroprozessors. Neben leichtverständlicher Beschreibung aller 7501-Befehle werden viele Betriebssystemroutinen unter die Lupe genommen.

Aus dem Inhalt:

- Von BASIC zu Maschinensprache
- Der Maschinensprachemonitor
- Der 7501-Prozessor und sein Befehlssatz
- Ein- und Ausgabe von Zeichen in Maschinensprache
- Einbinden von Maschinenprogrammen in BASIC-Programme
- Der BASIC-Lader
- Routinen des Betriebssystems richtig genutzt
- Die Vektoren des Betriebssystems
- Interruptprogrammierung
- Beispielprogramme mit trickreichen Utilities
- Zeropage-, Befehlscode- und Umrechnungstabellen

Vüllers
C16, C116, Plus/4 Maschinensprache
ca. 300 Seiten, DM 29,–
ISBN 3-89011-206-4

Baloui

C16
Tips & Tricks

Eine Fundgrube für den
Commodore 16 Anwender

EIN DATA BECKER BUCH

Bücher zu C16, C116 und Plus/4

C16 Tips und Tricks bietet eine hochinteressante Sammlung von Anregungen, Ideen und fertigen Lösungen zur Programmierung und Anwendung Ihres C16/116. Eine wahre Fundgrube für jeden, der auf dem Commodore 16 eigene Programme schreiben will!

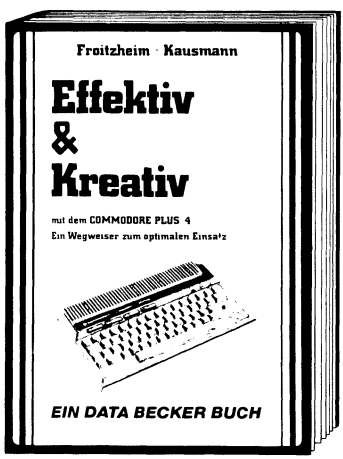
Aus dem Inhalt:

- Hinweise zur Eingabe der Programme
- Alle Programme für Kassetten- und Diskettenbetrieb
- Anwenderprogramme aus den Bereichen Unterhaltung, Grafik, Text- und Dateiverarbeitung, Mathematik
- Viele Utilities, wie Hardcopy auf Drucker, REM-Killer, Mergen von einzelnen Programmteilen, Shape-Editor
- Etikettendruck
- Lottozahlen
- Datumsberechnung
- Die wichtigsten Tips & Tricks
- Von BASIC zu Maschinensprache
- Wichtige Zeropageadressen
- Betriebssystemroutinen
- Routinen des BASIC-Interpreters

Baloui
C16 Tips & Tricks
202 Seiten, DM 29,–
ISBN 3-89011-168-8

Bücher zu C16, C116 und Plus/4

Effektiv und kreativ mit dem Plus/4 und seiner eingebauten Software zu arbeiten ermöglicht dieses Buch. Im ersten Teil werden zahlreiche Ideen, Anwendungen und praktische Nutzungsmöglichkeiten der integrierten Software beschrieben, der zweite Teil enthält viele interessante Programmiertips und Tricks und eine Reihe kompletter Anwendungsprogramme.



Aus dem Inhalt:

- Abspeichern ganzer Grafikseiten
- Menügesteuerte Disk-Hilfe
- Verbesserte Eingaberoutinen
- Listschutz
- Erstellung von Balkengrafiken
- Funktionsplotter
- Komfortable Dateiverwaltung
- Programm zur Entscheidungshilfe und vieles mehr

Froitzheim, Kausmann
Effektiv und kreativ mit dem Plus/4
244 Seiten, DM 49,–
ISBN 3-89011-073-8

DAS STEHT DRIN:

Die BASIC-Programmierung auf Ihrem C16, C116, Plus/4 wird mit diesem Buch zum Kinderspiel! Der Autor führt detailliert in die BASIC-Programmierung ein und gibt eine Vielzahl von Tips und Tricks für den BASIC-Newcomer. Eine gut gegliederte Befehlsübersicht des C16-, C116-, Plus/4-BASIC erleichtert die Arbeit am Computer. Werden auch Sie zum BASIC-Profi!

Aus dem Inhalt:

- Tastatur und Editor
- BASIC-Kurs
- Demo-Programme
- Rechnen mit dem C16, C116, Plus/4
- Variablen
- Steuerung des Programmablaufs
- Arrays
- Graphik-Programmierung
- Malprogramm
- Multicolor-Graphik
- Shapes
- Graphiken speichern und laden
- Musik-Programmierung
- Datei- und Adressenverwaltung
- Window-Technik
- BASIC-Befehlsübersicht

UND GESCHRIEBEN HAT DIESES BUCH:

Said Baloui ist Student und hat eine langjährige Erfahrung mit Commodore-Computern. Er programmiert professionelle Dateiverwaltungen und hat sich als Autor von zahlreichen DATA BECKER Büchern einen Namen gemacht.

ISBN 3-89011-204-8