```
**********************************
*                                *
*                                *
*                                *
**********************************
*                                *
*                                *
*                                *
*  B A S I C  3.5                *
*                                *
*  L E X I K O N                 *
*                                *
**********************************
```

```
****************
*  EINLEITUNG  *
****************
```

Die vielen Beispiele und Übungen dieses Handbuches haben Ihnen bereits einen guten Eindruck von der Computerprogrammierung in BASIC vermittelt. Das folgende Lexikon beschreibt den vollständigen Wortschatz und die Regeln (SYNTAX) der Programmiersprache BASIC 3.5. Jedes Sprachelement wird durch kurze Beispiele erläutert, die Sie durch eigenes Experimentieren fortführen und vertiefen sollten. Sie brauchen nicht zu befürchten, daß der COMMODORE 116 durch irgendeine falsche Eingabe beschädigt werden könnte. Nur ständige Übung führt zur Meisterschaft!

Im Lexikon finden Sie die Formatdefinitionen, zusammenfassende Erklärungen sowie Beispiele zu allen Sprachelementen von BASIC 3.5. Es kann jedoch keinen Lehrgang ersetzen; hier muß vielmehr auf einschlägige Lehrbücher verwiesen werden.

Bei den BASIC-Befehlen haben wir zwischen Kommandos und Anweisungen zu unterscheiden, wobei die Grenze zwischen beiden Bereichen nicht immer klar definiert ist. Kommandos werden hauptsächlich im DIREKT-Modus angewendet, während Anweisungen meist in ein Programm eingebaut sind. Sofern die Kommandos mit einer Zeilennummer versehen werden, sind sie (bis auf wenige Ausnahmen) auch im PROGRAMM-Modus einsetzbar. Umgekehrt lassen sich viele Anweisungen auch im DIREKT-Modus verwenden (also ohne vorangehende Zeilennummer). Innerhalb der getrennten Kapitel sind die Kommandos bzw. Anweisungen alphabetisch aufgeführt.

Das BASIC 3.5 LEXIKON ist folgendermaßen aufgebaut:

---

* KOMMANDOS: Befehle, die hauptsächlich dem Umgang mit Programmen dienen, z.B. für das Laden, Starten, Editieren, Abspeichern, Löschen und für die Diskettenverwaltung.

* ANWEISUNGEN: Befehle, die hauptsächlich innerhalb von numerierten Programmzeilen eingesetzt werden.

* FUNKTIONEN: String-, Numerische und Druck-Funktionen.

* VARIABLEN UND OPERATOREN: Die unterschiedlichen Variablen-Typen, zugelassene Variablen-Namen, arithmetische und logische Operatoren.

```
**********************************
*  KOMMANDO- UND ANWEISUNGSFORMAT  *
**********************************
```

Die in diesem Kapitel aufgeführten Befehle werden anhand von festen Formatregeln beschrieben, um sie so eindeutig wie möglich darzustellen. In den meisten Fällen sind noch eine Reihe von Beispielen mit den entsprechenden Befehlen angefügt. Das folgende Beispiel zeigt einige der Formatregeln, wie sie in Kommandos und Anweisungen verwendet werden:

```
BEISPIEL:  DLOAD"Programm-Name" [,D0,U8]
             !          !        !  !
             !          !        !  Zusätzlicher Parameter
             !          !       Zusätzlicher Parameter
             !         Parameter
            Schlüsselwort (KEYWORD)
```

Die Teile eines Befehls, die exakt nach Vorlage einzutippen sind, werden großgeschrieben und unterstrichen gedruckt. Eingaben, die von Fall zu Fall variieren, sind durch Hinweistexte normaler Groß-/Kleinschrift repräsentiert. Enthält die Formatmaske Trennzeichen (z.B. Komma oder Semikolon) oder Anführungszeichen (" "; meist bei Programm- oder Datei-Namen), dann sind sie fester Bestandteil des Formats und müssen mit eingegeben werden.

Allgemein ist zu beachten, daß alle Zeichen OHNE SHIFT-Taste einzugeben sind. Das gilt auch dann, wenn Sie den Computer in den Schreibmaschinen-Modus geschaltet haben und anstelle der in der Formatmaske stehenden Großbuchstaben kleine Buchstaben auf dem Bildschirm erscheinen. Geshiftete Buchstaben oder Graphikzeichen sind prinzipiell nur innerhalb von Anführungszeichen erlaubt! (einzige Ausnahme: Abkürzungen von Schlüsselwörtern).

SCHLÜSSELWÖRTER (auch reservierte Worte genannt) erscheinen in Großbuchstaben und unterstrichen. DIESE WÖRTER MÜSSEN SO EINGEGEBEN WERDEN, WIE SIE VORGEDRUCKT SIND, allerdings können viele dieser Schlüsselworte mit ihrer Abkürzung (siehe hierzu Anhang: ABKÜRZUNGEN) eingegeben werden.

Die Schlüsselwörter sind Hauptbestandteile der Programmiersprache BASIC, wie sie der COMMODORE 116 versteht. Sie sind der wichtigste Teil eines Befehls, weil sie dem Computer mitteilen, welche Aktion er als nächstes auszuführen hat. Diese Wörter sind reserviert und dürfen daher nicht als Bestandteil eines Variablennamens auftreten!

PARAMETER: Werden in Groß-/Kleinbuchstaben geschrieben. Parameter stellen den Teil eines Befehls dar, den Sie selbst festlegen müssen. Sie vervollständigen ein Schlüsselwort, indem Sie diesem wesentliche Zusatzinformationen liefern. Beispielsweise wird dem Computer per Schlüsselwort mitgeteilt, ein Programm zu laden, während Parameter 1 spezifiziert, welches Programm, und Parameter 2, aus welchem Disketten-Laufwerk (bei Verwendung eines Doppellaufwerks) das Programm zu laden ist. Zu den Parametern zählen Filenamen, Variable, Zeilennummern, usw. Das Zeichen # in einem Parameter steht immer für 'Nummer'.

ECKIGE KLAMMERN: Enthalten Zusatz-Parameter, deren Angabe nicht zwingend, sondern in gewissen Standard-Situationen entbehrlich ist.

SPITZE KLAMMERN: Enthalten eine Liste von Zusatz-Parametern, wovon einer zwingend gewählt werden muß.

AUSLASSUNGSZEICHEN (...): Eine Folge von drei Punkten bedeutet, daß eine Parameterliste um weitere, gleichartige Parameter verlängert werden kann (die jeweils durch Kommata getrennt werden).

TRENNSTRICH (|) : Werden die Teile einer Liste durch Trennstriche anstelle von Kommata getrennt, so handelt es sich um eine Auswahlliste alternativer Eingabemöglichkeiten.

VARIABLE: An dieser Stelle muß eine gültige BASIC-VARIABLE wie z.B. X, A$ oder T% eingetippt werden. An vielen Stellen ist jedoch auch eine ganze Formel (arithmetischer oder String-Ausdruck) zugelassen, die zur Sicherheit in Klammern eingeschlossen werden kann.

TERM: Dies ist jeder in BASIC gültige Formelterm wie z.B. A+B+2 oder .5*(X+3).

ANFÜHRUNGSZEICHEN, RUNDE KLAMMERN, KOMMA, SEMIKOLON, BINDESTRICH: Diese Zeichen sind fester Bestandteil eines Befehls und müssen stets eingegeben werden.

```
*****************************
*  KOMMANDOS IN BASIC 3.5   *
*****************************
```

AUTO
****
AUTO [Zeilenabstand]

Schaltet die automatische Zeilennumerierung ein, wodurch die Programmeingabe wesentlich erleichtert wird. Die nächste Zeilennummer wird nach jedem <Return> automatisch vorgegeben, und der CURSOR wartet in der richtigen Schreibposition. Der Klammerwert (Parameter) bestimmt, in welchem Zeilennummernabstand die nächste Zeile automatisch ausgegeben wird. Das Kommando AUTO ohne Parameter schaltet den Modus wieder aus; der RUN-Befehl bewirkt das gleiche. Der AUTO-Befehl kann nur im DIREKT-Modus eingegeben werden.

BEISPIELE:

AUTO 10	Numeriert Zeilen automatisch im Abstand von 10.
AUTO 50	Numeriert Zeilen automatisch im Abstand von 50.
AUTO	Schaltet automatische Zeilennumerierung AUS.

BACKUP
******
BACKUP DLaufwerk-# TO DLaufwerk-# [<,|ON> UGeräte-#]

Mit diesem Befehl werden bei einem Doppel-Laufwerk alle Files der einen Diskette auf die andere Diskette kopiert. Dabei kann die 'empfangende' Diskette noch unformatiert sein, da der BACKUP-Befehl alle Informationen der 'sendenden' Diskette - einschließlich Format - übernimmt. Wichtige Original-Disketten sollen stets mit diesem Befehl gesichert werden (Sicherung gegen Beschädigung oder Verlust!).

Das BACKUP-Kommando formatiert (headert) auch die Diskette, d.h. alle eventuell auf dieser Diskette vorhandenen Daten werden unwiederbringlich gelöscht - daher VORSICHT!. Siehe auch 'COPY'.

WICHTIG: Das BACKUP-Kommando kann nur bei einer Doppel-Floppy verwendet werden.

BEISPIELE:

```
BACKUP D0 TO D1              Kopiert die gesamte Diskette in Drive 0
                             auf die Diskette in Laufwerk 1 (Drive 1)

BACKUP D0 TO D1, U9          Kopiert die gesamte Diskette in Drive 0
BACKUP D0 TO D1 ON U9        auf die Diskette in Drive 1, Gerät 9
( BACKUP D0 TO D1,ON U9 ... wird ebenfalls akzeptiert. )
```

COLLECT
****
COLLECT [DLaufwerk-#] [<,|ON> UGeräte-#]

Mit diesem Befehl wird die Diskette 'bereinigt', d.h. nicht korrekt geschlossene Files werden geschlossen und damit auch das Inhaltsverzeichnis auf den richtigen, freien Speicherplatz korrigiert.

BEISPIEL:

```
COLLECT D0
```

CONT (=CONTINUE)
****

Ein gestopptes Programm - egal ob per Taste <Run/Stop>, durch eine STOP- oder END-Anweisung in einem Programm - kann mit dem Befehl CONT wieder gestartet werden. Die Programmfortsetzung erfolgt genau hinter der Abbruchstelle.

Das Kommando CONT funktioniert nicht, wenn nach dem Stopvorgang Programmzeilen geändert, hinzugefügt oder entfernt wurden. Hielt das Programm wegen eines Programmfehlers (ERROR) an, oder Sie veranlaßten während des Stopvorgangs eine ERROR-Meldung, so ist ein Wiederstart (mit CONT) nicht möglich. In diesen Fällen erscheint die Fehlermeldung: CAN'T CONTINUE ERROR.

COPY
****
COPY [DLw-#,]"Quelle"TO[DLw-#,]"Ziel"[<,|ON> UGeräte-#]

Mit dem Befehl COPY kann ein bestimmtes File von einem Laufwerk (Quell-File) auf die Diskette im anderen Laufwerk (nur in einer Doppelfloppy möglich, Zieldiskette muß schon formatiert sein) unter dem gleichen Filenamen, oder im selben Laufwerk unter anderem Filenamen kopiert werden.

BEISPIELE:

COPY D0,"ABEND" TO D1,"NACHT"	Kopiert File "ABEND" von Laufwerk 0 auf Diskette in Drive 1 mit Umbenennung in "NACHT".
COPY D0,"ABEND" TO D1,"ABEND"	Kopiert File "ABEND" von Laufwerk 0 auf Diskette in Drive 1.
COPY D0 TO D1	Kopiert alle Files von Laufwerk 0 auf Diskette in Drive 1.
COPY "KATZEN" TO "HUNDE"	Kopiert File "KATZEN" auf demselben Laufwerk mit Umbenennung in File "HUNDE".
COPY D0,"DATEN*" TO D1,"*"	Kopiert von Laufwerk 0 alle Files, deren Namen mit "DATEN" anfangen, unter selbem Namen nach Laufwerk 1.
COPY D0,"?" TO D1,"*"	Kopiert von Laufwerk 0 alle Files, deren Namen genau ein Zeichen lang sind, genauso nach Laufwerk 1.

DELETE
******
DELETE [Erste Zeilen-#] [- [Letzte Zeilen-#]]

Dieses Kommando löscht BASIC-Programmzeilen (von - bis). Es ist nur im DIREKT-Modus anwendbar.

BEISPIELE:

DELETE 75	Löscht Programmzeile 75.
DELETE 10-50	Löscht die Zeilen 10 bis einschließlich Zeile 50.
DELETE -50	Löscht alle Zeilen vom Programmanfang bis einschließlich Zeile 50.
DELETE 75-	Löscht alle Zeilen des Programms ab Zeile 75 bis zum Programmende.

DIRECTORY
****
DIRECTORY [DLaufwerk-#] [<,|ON> UGeräte-#] [,"Filename"]

Um das Inhaltsverzeichnis einer Diskette auf den Bildschirm des COMMODORE 116 zu bringen, dient das Kommando DIRECTORY. Der Durchlauf (= Scrolling) des gelisteten Inhaltsverzeichnisses kann mit der Tastenkombination <Control> & <S> angehalten und mit jeder beliebigen Taste wieder gestartet werden. Mit der COMMODORE-Taste <C=> wird der Durchlauf verlangsamt. Mit dem Befehl DIRECTORY läßt sich keine Hardcopy (= Papierausdruck des Bildschirminhalts) erstellen. Um dies zu erreichen, muß das DIRECTORY in den Speicher geladen (eventuell vorhandene Programme werden dadurch überschrieben - und unbrauchbar, daher VORSICHT!) und wie ein Programm auf den Drucker gelistet werden.

BEISPIELE:

DIRECTORY	Listet alle Files der Diskette.
DIRECTORY D1, U9, "WALD"	Listet das File "WALD" der Diskette im Laufwerk 1 mit der Gerätenummer 9.

DIRECTORY D0, "PGM ?.VERS" Das sog. Jokerzeichen '?' steht stellvertretend für alle Zeichen an dieser Textstelle: Die Files "PGM 1.VERS", "PGM 2.VERS", "PGM 3.VERS" passen dazu und werden daher gelistet.

DIRECTORY D1, "PGM.*" Das Jokerzeichen "*" steht stellvertretend für alle möglichen Fortsetzungen: Dazu passen z.B. "PGM.", "PGM.ALT", "PGM.N-V1".

WICHTIG: Um das Inhaltsverzeichnis der Diskette im Laufwerk 0 des Geräts 8 auszudrucken, benutzen Sie die Kommandofolge:

```
LOAD "$0",8
OPEN 4,4: CMD 4: LIST
PRINT#4: CLOSE 4
```

DLOAD
*****
DLOAD "Filename" [,DLaufwerk-#] [<, |ON> UGeräte-#]

Mit diesem Befehl wird ein Programm bestimmten Namens in den vorhandenen Speicherbereich geladen. (Kassettenprogramme werden mit dem Befehl LOAD geladen). Im Gegensatz zum LOAD-Befehl von Kassette muß der Programmname angegeben werden, wobei die Jokerzeichen "?" und "*" verwendet werden dürfen.

BEISPIELE:

DLOAD "AUTOBUS" Sucht auf der Diskette (im Inhaltsverzeichnis) den Filenamen "AUTOBUS" und lädt dieses Programm in den Speicher.

DLOAD (A$) Lädt ein Programm, dessen Name in der Variablen 'A$' gespeichert ist. Ist A$ leer, kommt eine ERROR-Meldung.

Der DLOAD-Befehl kann auch aus einem BASIC-Programm heraus verwendet werden, um ein Programm zu finden und zu starten (RUN). Dieser Vorgang wird Verketten (= chaining) genannt. Das nachgeladene Programm muß kürzer als das vorhergehende sein, dann können die bisher verwendeten Variablen weiterverarbeitet werden.

DSAVE
*****
DSAVE "Filename" [,DLaufwerk-#] [<,|ON> UGeräte-#]

Programme werden mit diesem Befehl auf Diskette abgespeichert. (Bei Kassettenbetrieb lautet der Befehl SAVE). Es muß ein Programmname vergeben werden. Ein 'Klammeraffe' (bei DIN-Tastatur ein §) vor dem Namen bestimmt, daß ein vorhandenes Programm gleichen Namens zu überschreiben ist.

BEISPIELE:

DSAVE "§FREITAG"	Das Programm "FREITAG" auf Diskette überschreibend abspeichern.
DSAVE (A$)	Das Programm, dessen Name in der Variablen 'A$' abgelegt ist, auf Diskette speichern.
DSAVE "PGM 3",D0,U9	Speichert das Programm "PGM 3" auf die Diskette im Laufwerk 0 des Geräts 9.

HEADER
******
HEADER "Diskettenname", DLaufwerk-# [,IID] [<, ON> UGeräte-#]

Bevor eine neue Diskette erstmals in einer Floppy eingesetzt werden kann, muß sie mit dem Befehl HEADER formatiert werden. Auch gebrauchte Disketten, die neu benützt werden sollen, lassen sich auf diese Weise 'erneuern' - mit dem Befehl HEADER. Der HEADER-Befehl veranlaßt, daß die Diskette in Spuren und Sektoren eingeteilt wird und daß dafür auch ein Verzeichnis (das DIRECTORY) auf der Diskette angelegt wird.

Als Diskettenname kommt jede Kombination bis zu einer Maximallänge von 16 Zeichen in Frage. Die Identität (= ID) besteht aus zwei Zeichen. Jede Diskette soll möglichst eine andere ID bekommen. Und Vorsicht mit dem HEADER-Befehl - er bewirkt, daß alle Daten auf der Diskette unwiederbringlich gelöscht werden!

Wenn Sie keine ID vergeben, erfolgt der HEADER-Vorgang wesentlich schneller - es bleibt dann bei der bisherigen ID. Verständlich, daß diese Methode nur bei gebrauchten Disketten funktionieren kann, da der schnelle HEADER-Vorgang lediglich das Inhaltsverzeichnis löscht und nicht die gesamte Diskette formatiert.

BEISPIELE:
HEADER "TEST DISK", I23, D0
HEADER "UEBUNGEN", I1A, D1, U8

HELP
****
HELP

Der HELP-Befehl wird nach Programmfehlern aufgerufen. Wenn Sie HELP eintippen (und mit Taste <Return> abschicken), dann wird die fehlerhafte Programmzeile gelistet, wobei der Teil mit dem Fehler blinkend dargestellt wird (s.a. HELP-Taste).

KEY
***
KEY [Funktionstasten-#, Zugeordneter String]

Der COMMODORE 116 verfügt über acht Funktionstasten: vier Tasten ohne und vier Tasten mit der Taste <Shift> zu bedienen. Unabhängig zu der durch den Einschaltvorgang vorgenommenen Standardbelegung, kann jede der acht Funktionstasten mit einem anderen String belegt werden.

Geben Sie KEY ohne Parameter ein, dann wird die momentane Belegung am Bildschirm gelistet. Der einer Funktionstaste zugeordnete String wird bei Betätigung dieser Taste am Bildschirm ausgedruckt. Die Gesamtlänge aller acht Strings zusammen darf 128 Zeichen nicht überschreiten. Jeder der acht Tasten können sowohl Einzel- wie auch Serienbefehle zugeordnet werden; wie zum Beispiel:

```
KEY 7, "GRAPHIC 0" + CHR$(13) + "LIST" + CHR$(13)
```

Dieses Belegungsbeispiel veranlaßt - sofern Funktionstaste 7 gedrückt und im DIREKT-Modus abgeschickt wird - die Umschaltung in den TEXT-Modus und ein anschließendes Programmlisting auf den Bildschirm. Mit dem String CHR$(13) wird ein RETURN-Befehl ausgelöst. Wenn Sie Anführungszeichen in dem String brauchen, dann schreiben Sie diese mit dem String CHR$(34).

Auch per Programm lassen sich die Funktionstasten umschreiben. Ein Beispiel:

```
10 KEY 2, "TEST VON" + CHR$(34): KEY 3, "NEIN"
```

oder

```
10 FOR I=1 TO 8: KEY I, CHR$(I+132): NEXT
```

Diese Programmzeile definiert die Funktionstasten, wie sie beim COMMODORE 64 und VC 20 belegt sind.

Um die Funktionstasten wieder mit ihren Standardwerten zu belegen, muß die Reset-Taste des COMMODORE 116 gedrückt (oder natürlich die geänderten Tasten durch KEY ... neu belegt) werden.

LIST
****
LIST [Erste Pgm-Zeile] [- [Letzte Pgm-Zeile]]

Mit dem LIST-Befehl kann ein BASIC-Programm Zeile für Zeile ausgedruckt werden, das vorher mit dem LOAD-Befehl in den Speicher des COMMODORE 116 geladen wurde. Wird der LIST-Befehl ohne zusätzliche Parameterangaben (ohne nachfolgende Zahlen) benützt, dann erscheint das gesamte Listing auf dem Bildschirm. Verlangsamt wird per COMMODORE-Taste <C=>, angehalten mit Taste <Control> & Taste <S>, wieder ausgelöst mit irgendeiner Taste und gestoppt mit der Taste <Run/Stop>.

Folgt dem LIST-Befehl nur eine Programmzeilennummer, so wird diese Zeile (sofern es sie im Speicher gibt) angezeigt. Werden mit dem LIST-Befehl zwei aufsteigende Zahlen, mit einem Bindestrich (Minussymbol '-') getrennt, geschrieben, dann wird das Programm von der ersten Zeilennummer bis einschließlich der zweiten Nummer gelistet. LIST mit einer Zahl und einem Bindestrich veranlaßt das Listing ab dieser Zeilennummer bis zum Programmende, und LIST mit Bindestrich und Zahl bewirkt ein Programmlisting vom Programmanfang bis zu dieser Zeilennummer. Beim richtigen Einsatz dieser Variationsmöglichkeiten läßt sich jeder Programmteil listen bzw. zur Modifikation auf den Bildschirm bringen.

BEISPIELE:

LIST	Zeigt das gesamte Programm.
LIST 100-	Zeigt von Zeile 100 bis zum Programmende alles.
LIST 10	Zeigt nur die Programmzeile 10.
LIST -100	Zeigt alles vom Programmanfang bis einschließlich Zeile 100.
LIST 10-200	Zeigt alle Programmzeilen ab Zeile 10 bis inclusive Programmzeile 200.

LOAD
****
LOAD "File Name" [,Geräte-#] [,Speicheradresse-Flag]

Mit dem Befehl LOAD werden Programme, die entweder auf Kassette oder Diskette abgespeichert sind, in den Arbeitsspeicher des Computers geholt. LOAD allein und die Taste <Return> veranlaßt, daß der Bildschirm des COMMODORE 116 blank wird. Wird nun an der (angeschlossenen) Datassette die Taste <PLAY> gedrückt, startet der COMMODORE 116 und sucht ein Programm auf der Kassette. Das erste gefundene Programm wird mit FOUND "File Name" gemeldet (der Rekorder hält an). Ein Druck auf die COMMODORE-Taste <C=> veranlaßt den Beginn des Ladevorgangs; die Betätigung der LEER-Taste läßt den COMMODORE 116 nach dem nächsten Programm auf der Kassette Ausschau halten. Ist ein Programm erst einmal im Speicher des COMMODORE 116, so kann es in der Folge mit RUN (und Taste <Return>) gestartet, mit LIST gelistet oder abgeändert werden.

Dem Befehl LOAD kann auch ein Programmname (stets zwischen Anführungszeichen " ") folgen, und nach dem Programmnamen ein Komma (',' - außerhalb der Anführungszeichen) und eine Zahl (oder Zahlenvariable). Diese Zahl gibt die Gerätenummer an, d.h. sie bestimmt, ob von Diskette oder Kassette geladen wird. Fehlt diese Zahl, so interpretiert der COMMODORE 116 dies als Gerätenummer 1 (= Datassette). Das Diskettenlaufwerk hat normalerweise die Gerätenummer 8.

BEISPIELE:

LOAD	Lädt das nächste auf Kassette befindliche Programm in den Speicher des COMMODORE 116.
LOAD "BASIS"	Sucht die Kassette nach dem Programm "BASIS" durch und lädt es, sofern gefunden.
LOAD A$	Sucht nach einem Kassettenprogramm, dessen Name in der Variablen A$ abgelegt ist.
LOAD "MAUERN",8	Es wird das Programm "MAUERN" auf Diskette gesucht und sofern vorhanden auch in den Speicher geladen.

Den LOAD-Befehl kann man auch in BASIC-Programme einbinden und auf diese Weise das nächste Programm von Kassette nachladen und starten. Dieser Vorgang wird als Verketten (Chaining) bezeichnet. Das zweite Programm muß in der Regel kürzer als das erste sein.

Das Speicheradresse-FLAG bestimmt, an welche Speicheradresse das Programm zu laden ist. FLAG = 0 informiert den COMMODORE 116, das Programm an den Anfang des BASIC-Speicherbereichs zu laden. FLAG = 1 lädt dagegen das Programm auf jeden Fall an die Adresse, die es beim Abspeichern innehatte. Standard (d.h. wenn keine Angabe erfolgt) ist FLAG = 0. Der Wert 1 wird im allgemeinen nur in Verbindung mit zu ladenden Maschinenprogrammen benützt.

NEW
***
NEW

Der BASIC-Befehl NEW löscht das gegenwärtige Programm im Speicher und setzt gleichzeitig alle verwendeten Variablen wieder auf Null. Wenn das Programm vorher nicht abgespeichert wurde, ist es bis zur Wiedereingabe verloren. Daher ist der Befehl NEW mit VORSICHT anzuwenden.

Auch der Befehl NEW kann in BASIC Programme eingebaut werden. Erreicht der COMMODORE 116 die Programmzeile mit dem NEW-Befehl, wird das gesamte Programm gelöscht und der Programmablauf abgebrochen. Das ist natürlich höchstens dann sinnvoll, wenn ein Programm nach vollständigem Ablauf gezielt beendet werden soll.

RENAME
******
RENAME "Alt" TO "Neu" [,DLaufwerk-#] [<,|ON> UGeräte-#]

Dieser RENAME Befehl wird eingesetzt, um einen Filenamen auf der Diskette umzubenennen.

BEISPIEL:

RENAME "VORTEIL" TO "SCHULDEN",D0 Tauscht auf der Diskette den Filenamen "VORTEIL" in den Namen "SCHULDEN" um.

RENUMBER
****
RENUMBER [Neue Startz-# [,Zeilenabstand [,Alte Startz-#]]]

Die neue Startzeilennummer bildet nach dem RENUMBER-Vorgang die erste Programmzeile. Erfolgt keine Angabe, so nimmt das System als Standard ('Default') Zeilennummer 10 an.

Zeilenabstand ist der numerische Abstand zwischen zwei Programmzeilen. Standard ist ebenfalls 10.

Die alte Startzeilennummer legt den Beginn des Renumber-Prozesses fest. Somit lassen sich auf diese Weise auch Teile des Programms neu durchnumerieren. Fehlt dieser Parameter, wird auf die erste Zeile Bezug genommen.

Der RENUMBER-Befehl kann nur im DIREKT-Modus eingesetzt werden.

BEISPIELE:

RENUMBER 20, 20, 1 Der RENUMBER-Vorgang startet bei der bisherigen Zeile 1, die zur Zeile 20 wird. Die Zeilennummern steigen ab da jeweils um 20.

RENUMBER , , 65 Beginnend ab Zeile 65, die nun Zeile 10 wird, steigen die Zeilennummern mit 10.

RUN
***
RUN [Zeilen-#]

Nachdem ein Programm in den Speicher des COMMODORE 116 eingetippt oder eingeladen worden ist, kann es durch den RUN-Befehl gestartet werden. RUN löscht vor Programmbeginn alle Variablen. Steht der Befehl RUN allein, so beginnt der Computer bei der niedrigsten Zeilennummer mit der Programmausführung. Wird der Befehl RUN mit einer Zahl verbunden, so stellt dieser Parameter die Zeilennummer dar, bei der der Programmablauf startet. Der Befehl RUN kann auch in ein BASIC-Programm eingebunden werden.

BEISPIELE:

RUN	Startet den Programmablauf bei der ersten Zeile.
RUN 100	Startet das Programm ab der Zeilenzahl 100.

SAVE
****
SAVE [" File Name" [,Geräte-# [,EOT-FLAG]]]

Mit dem Befehl SAVE wird ein gegenwärtig im Speicher befindliches Programm auf Kassette oder Diskette abgespeichert. Geben Sie nur SAVE und <Return> ein, ist der COMMODORE 116 auf Kassettenspeicherung programmiert. Für den Computer besteht keine Möglichkeit, festzustellen, ob sich an dieser Stelle der Kassette bereits ein Programm befindet. Daher empfehlen sich genaue Aufzeichnungen über die Bespielung der Kassetten.

Wird mit dem SAVE-Befehl auch der zwischen Anführungszeichen stehende Filename oder eine String-Variable eingegeben, dann speichert der COMMODORE 116 dieses Programm unter diesem Namen ab. In Zukunft ist dieses Programm dann leichter wiederzufinden und einzusetzen.

Soll auch die Geräte-Nummer eingegeben werden, dann ist diese nach den Anführungszeichen mit einem Komma und der Zahl oder einer Zahlenvariablen miteinzugeben. Gerät 1 ist die Datassette, und Nummer 8 aktiviert die Diskettenstation.

Nach der Zahl 1 für Kassette kann nach einem weiteren Komma noch eine Zahl folgen: das EOT-FLAG (End Of Tape = Ende des Bandes). Ist diese zweite Zahl eine Eins ('1'), so wird vom COMMODORE 116 am Ende des abgespeicherten Programms eine Markierung gesetzt, die im Falle eines bis zu dieser Marke erfolglosen Versuchs eines Einladevorgangs einen Abbruch mit der Meldung FILE NOT FOUND ERROR bewirkt.

Bei Speicherung auf Diskette kann die Laufwerksnummer mit nachfolgendem Doppelpunkt direkt vor den Filenamen geschrieben werden. Wird vor die Laufwerksnummer noch ein 'Klmmeraffe' (bei DIN-Tastatur Paragraph) geschrieben, wird ein schon bestehendes Programmfile überschrieben.

BEISPIELE:

SAVE	Speichert das Programm ohne Namen auf die Kassette ab.
SAVE "PAPIER"	Speichert das Programm unter dem Namen "PAPIER" auf Kassette ab.
SAVE A$	Speichert das Programm unter dem Variablen Namen aus A$ auf Kassette ab.
SAVE "EIGENTUM",8	Speichert das Programm unter dem Namen "EIGENTUM" auf Diskette ab.
SAVE "1:TAGEBUCH",8	Speichert das Programm unter dem Namen "TAGEBUCH" auf Laufwerk 1 der Floppy ab.
SAVE "§0:TAGEBUCH",8	Überschreibt das Programm "TAGEBUCH" auf Laufwerk 0 der Floppy.
SAVE "SPIELE",1,1	Speichert das Programm "SPIELE" mit dem EOT-FLAG auf Kassette ab.

SCRATCH
****
SCRATCH "File Name" [,DLaufwerk-#] [<,|ON> UGeräte-#]

Um ein File auf der Diskette zu löschen, bedient man sich des SCRATCH-Befehls. Als zusätzliche Sicherheit kommt nach der Eingabe die Frage: Are you sure? (=sind Sie sicher?). Wird diese Frage mit 'y' für 'yes' (=ja) beantwortet, beginnt erst der Löschvorgang. Mit 'n' für 'no' (=nein) wird der Vorgang abgebrochen. Der Befehl ist u.a. sehr nützlich, um wieder mehr Speicherplatz auf der Diskette zu schaffen, indem unerwünschte Files gelöscht werden.

BEISPIEL:

```
SCRATCH "VERSION 2", D1      Löscht das File "VERSION 2" auf der
                             Diskette in Laufwerk 1.
```

VERIFY
******
VERIFY ["File Name" [,Geräte-# [,Speicheradressen-FLAG]]]

Der byteweise Vergleich eines auf Diskette gespeicherten Programms mit dem im Speicher des COMMODORE 116 befindlichen Programm erfolgt mit dem Befehl VERIFY. Soeben abgespeicherte Programme werden am besten sofort auf einwandfreie Speicherung damit überprüft.

Der Befehl VERIFY bewährt sich auch im Einsatz der Kassettenstation, weil damit exakt das Ende des gleichen (oder auch eines anderen - nur wird dies dann entsprechend gemeldet) Programms auf der Kassette gefunden werden kann. Das nächste Programm kann jetzt ab dieser Stelle ohne die Gefahr des Überschreibens abgespeichert werden.

Lautet der Befehl VERIFY ohne Parameterzusatz, dann wird auf Kassette einfach das nächste kommende Programm mit dem Programm im Speicher auf Gleichheit - ohne Beachtung des Programmnamens - geprüft. Wird der VERIFY-Befehl mit einem zwischen Anführungszeichen geschriebenen Programmnamen oder einem Variablennamen gestartet, dann wird ausschließlich dieses Programm gesucht und, wenn vorhanden, mit dem Programm im Speicher verglichen. Die erste Zahl nach dem Programmnamen bestimmt das Gerät des Datenträgers (1= Kassette, 8= Diskette). Das Speicheradressen-FLAG hat die gleiche Bedeutung wie im LOAD-Befehl.

BEISPIEL:

VERIFY	Überprüft das nächste Programm auf Kassette auf Gleichheit mit dem im Speicher.
VERIFY "WIRKLICHKEIT"	Sucht das Programm "WIRKLICHKEIT" auf der Kassette und prüft es wenn gefunden gegen das im Speicher befindliche Programm.
VERIFY "EDEN",8,1	Sucht das Programm "EDEN" auf der Diskette im Gerät 8 (ab gespeicherter Adresse) und prüft es.

```
*******************************
*  ANWEISUNGEN IN BASIC 3.5   *
*******************************
```

BOX
***
BOX [Farbzonen-#], a1, b1, a2, b2, [,[Drehwinkel] [,Farbe]]

Farbzonen-# ..... Farbzone (0-3); Standard ist 1 (Vordergrund-Farbe)
a1, b1 .... Eck-Koordinaten (skaliert - links, oben)
a2, b2 .... Eck-Koordinaten (skaliert - rechts, unten); Standard ist PC (=Pixel Cursor)
Drehwinkel ..... Drehung im Uhrzeigersinn (in Grad); Standard: 0 Grad
Farbe .... Füllt Umrisse mit Farbe (0=AUS, 1=EIN); Standard: 0

Mit Hilfe der Anweisung BOX lassen sich Rechtecke jeder Größe auf den Bildschirm zeichnen. Um den Standardwert zu verwenden, genügt es, ein Komma zu schreiben - ohne Wert. Der Drehpunkt befindet sich in der Mitte des Rechtecks. Der Pixel-Cursor (PC) befindet sich (unsichtbar) in der Koordinate a2, b2, nachdem die BOX-Anweisung ausgeführt wurde.

BEISPIELE:

BOX 1, 10, 10, 60, 60	Zeichnet die Umrisse eines Rechtecks.
BOX , 10, 10, 60, 60, 45, 1	Zeichnet ein gedrehtes, eingefärbtes Rechteck (Rhombus).
BOX , 30, 90, , 45, 1	Zeichnet ein eingefärbtes, gedrehtes Vieleck (Polygon).

CHAR
****
CHAR [Farbzonen-#],x,y [,[String] [,Reverse-FLAG]]

Farbzonen-# ..... Farbzone (0-3)
x ..... Buchstaben/Zeichen-Spalte (0-39)
y ..... Buchstaben/Zeichen-Zeile (0-24)
String ..... Zu druckender Text
Reverse ..... Reverse-Flag (0=AUS, 1=EIN)

Textzeilen (Alphanumerische Strings) können mit dem Befehl CHAR in jedem Grafik-Modus an der vorbestimmtem Position angezeigt werden. Die Buchstaben werden aus dem Zeichengenerator des COMMODORE 116 gelesen. Einzugeben sind lediglich die Anfangskoordinaten x und y sowie die darzustellende Textzeile. Die Farbzonennummer und das Reverse-FLAG sind keine zwingenden Zusatzinformationen.

Läuft eine Textzeile über den rechten Rand hinaus, dann wird sie in der nächsten Zeile fortgesetzt. Ist der TEXT-Modus eingeschaltet, dann verhält sich der mit der CHAR-Anweisung gedruckte String wie ein mit dem PRINT-Befehl gedruckter String, einschließlich Revers-, CURSOR-, Blinken- (Ein/Aus) Darstellungen, etc. Diese Kontrollfunktionen innerhalb des Strings funktionieren nicht, wenn für die Darstellung von Text im Grafik-Modus die Anweisung CHAR verwendet wird.

WICHTIG: Wenn im MEHRFARBEN-Modus ein Buchstabe/Zeichen im Mehrfarbenbereich 2 dargestellt werden soll, so ist die Farbzonen-# auf 0 und das Reverse-FLAG auf 1 zu setzen. Darstellung im Mehrfarbenbereich 1 erfordert Farbzonen-# und Reverse-FLAG auf 0.

CIRCLE
******
CIRCLE [cs], [a,b], xr, [yr], [sa], [ea], [Winkel], [inc]

```
cs ..............  Farbzone (0-3)
a, b ............  Mittelpunkt-Koordinaten (skaliert)
                    Standard ist die Position des PC (=Pixel-Cursor)
xr ..............  Radius in X-Achse (skaliert)
yr ..............  Radius in Y-Achse (als Standard wird xr übernommen)
sa ..............  Winkel (in Grad) am Kreisbogen-Anfang (Standard = 0)
ea ..............  Winkel (in Grad) am Kreisbogen-Ende (Standard = 360)
Winkel ..........  Drehung im Uhrzeigersinn (in Grad); Standard = 0 Gd.
inc .............  Winkel zwischen zwei Segmenten; Standard = 2 Grad
```

Mit der Anweisung CIRCLE lassen sich Kreise, Ellipsen, Kreissegmente, Dreiecke oder auch regelmäßige Vielecke zeichnen. Nach der Ausführung befindet sich der Pixel-Cursor auf dem Kreisumfang am Ende des Kreisbogens. Jede Drehung erfolgt um den Mittelpunkt. Kreisbögen werden durch regelmäßige Vielecke approximiert und vom Anfangspunkt im Uhrzeigersinn zum Endpunkt gezeichnet. Mit dem Winkel zwischen zwei Segmenten steuern Sie die Glätte der Rundung; je kleiner der Winkel, desto runder wird der Umfangsbogen. (Formel für die Anzahl der Ecken im Vollkreis: 360/inc).

BEISPIELE:

CIRCLE , 160,100,65,10	Zeichnet eine Ellipse
CIRCLE , 160,100,65,50	Zeichnet einen Kreis
CIRCLE , 60,40,20,18,,,,45	Zeichnet ein Achteck
CIRCLE , 260,40,20,,,,,90	Zeichnet eine Raute
CIRCLE , 60,140,20,18,,,,120	Zeichnet ein Dreieck

CLOSE
*****
CLOSE [File-#]

Mit der Anweisung CLOSE wird jedes geöffnete File abgeschlossen. Ein File wird gezielt über die der Anweisung folgende Zahl (= File-#) angesprochen.

BEISPIEL:

CLOSE 2 Das logische File 2 wird geschlossen.

CLR
***
CLR

Mit der Anweisung CLR werden zwar alle Variablen auf Null gesetzt, das im Speicher befindliche Programm bleibt hingegen erhalten. Die CLR-Anweisung ist automatisch im RUN- oder NEW-Befehl enthalten; aber auch mit jedem EDITIER-Vorgang (z.B. Programmzeilen ändern oder ergänzen) wird automatisch der CLR-Befehl ausgeführt.

CMD
***
CMD File-#

Ausgaben, die normalerweise zum Bildschirm gesendet werden (wie z.B. per PRINT-Anweisungen oder LIST-Befehl, jedoch keine POKE-Befehle auf den Bildschirm), werden mit dem Befehl CMD zu einem anderen Gerät umgeleitet. Dies können ein Drucker oder Daten-Files auf Kassette oder Diskette sein. Dieses Gerät bzw. File muß vorher jedoch geöffnet werden. Dann folgt der CMD-Befehl mit der Zahl oder numerischen Variablen, die die File-# repräsentiert. Wichtig: Vor dem CLOSE-Befehl auf das angegebene File immer einen PRINT#-Befehl mit gleicher File-# geben, sonst kann es zu merkwürdigen Bildschirmreaktionen kommen.

BEISPIELE:

OPEN 1,4	Öffnet Gerät #4, also den Drucker.
CMD 1	Die gesamte Ausgabe geht zum Drucker.
LIST	Das Listing erfolgt jetzt nicht am Bildschirm, sondern am Drucker - einschließlich READY.
PRINT #1	Schaltet die Ausgaben zurück zum Bildschirm.
CLOSE 1	Schließt das logische File 1.

COLOR

*****

COLOR Farbzonen-#, Farb-#, [, Helligkeitswert]

Mit der COLOR-Anweisung wird eine der fünf Farbzonen bestimmt:

Farbzonen-#	Bezeichnung
0	Bildschirm-Hintergrund
1	Vordergrund (Buchstaben, Zeichen)
2	Mehrfarben 1
3	Mehrfarben 2
4	Bildschirmrand

Die zweite Zahl in der COLOR-Anweisung selektiert die Hintergrundfarbe des gewählten Bildschirmbereichs. Diese Zahl ist mit den Farbtasten der Tastatur identisch.

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

Jede Farbe ist auch noch in ihrer Helligkeit (LUMINANZ) veränderbar. Im Anhang an die Farbnummer kann diese LUMINANZ von Null ('0'= dunkel) bis sieben ('7'= hell) variieren. Der Standardwert für die Helligkeit ist 7. Die acht Helligkeitsstufen gelten für alle Farben, mit Ausnahme von Schwarz.

DATA
****
DATA Liste von Elementen, die durch Kommata getrennt sind

Der DATA-Anweisung folgt eine Liste von Elementen, die mit der READ-Anweisung vom Programm gelesen werden. Diese Posten können Zahlen oder Strings sein und müssen durch Kommata getrennt sein. Strings benötigen keine Anführungszeichen, außer es gehören dazu: Leerstellen, Doppelpunkt oder Komma. Steht zwischen zwei Kommata kein Element, so liest der Computer entweder eine Null oder einen Leerstring, je nach Variablentyp (s. READ), dafür ein. Siehe auch die RESTORE-Anweisung; mit ihr setzt der COMMODORE 116 den DATA-Zeiger auf das erste Element der (angegebenen) DATA-Zeile.

BEISPIEL:

```
DATA 100, 200, FRED, "WILMA:", , 3, 14, ABC123
```

DEF FN (=DEFiniere FuNktion)
******
DEF FN Name der Variablen = Formel

Mit der Anweisung DEF FN läßt sich eine komplexe Funktion mit einem kurzen Namen (Variable) definieren. Handelt es sich um eine komplexe und lange Formel, die oft aufgerufen wird, so kann mit Hilfe der DEF FN Anweisung in einem Programm viel Speicherplatz gespart werden.

Der Name, den Sie der Funktion geben, beginnt stets mit FN, gefolgt von irgend einem Namen. Bei der Definition muß die Funktion mit der Anweisung DEF, gefolgt vom zugeteilen Namen, angegeben werden. Dann folgt die in Klammern eingeschlossene numerische Variable (in unserem Beispiel 'X'). Nach dem Gleichheitszeichen folgt die zugewiesene Formel. Diese Formel kann jederzeit aufgerufen werden, wenn die Variable X durch eine Zahl ersetzt wird (siehe Beispiel Zeile 20):

BEISPIEL:

```
10 DEF FN A(X) = 12*(34.75-X/.3)+X
```

Mit der nachstehenden Programmzeile wird für die Variable X an allen Stellen der Wert 7 eingesetzt.

```
20 PRINT FN A(7)
```

Nach RUN und <Return> wird die Zahl 144 ausgegeben.

DIM

***

DIM Variable (Anzahl [, Anzahl ...]) [, ...]

Bevor ein Feld mit Variablen benutzt werden kann, muß das Programm die DIM-Anweisung durchlaufen, um den gewünschten Bereich einzurichten. Bei eindimensionalen Feldern mit weniger als 11 Elementen kann jedoch auf diese Anweisung verzichtet werden.

Der DIM-Anweisung folgt der Name (jeder zulässige Variablenname) für die Matrix. Danach in Klammern die Anzahl (auch Variablen zulässig) der einzurichtenden Elemente je Dimension. Von Matrix spricht man, wenn ein Feld mit mehreren Dimensionen errichtet wird. Ein Feld ist in seiner Dimension nur durch den benötigten Speicherplatz begrenzt. Um den Gesamtspeicherbedarf eines Feldes zu errechnen, multiplizieren Sie die Elementanzahlen jeder Dimension (plus 1, da die erste Nummer immer 0 ist) mit dem Speicherbedarf eines einzelnen Elements (Real 5 Bytes, Integer 2, Strings 3-258).

WICHTIG: Ganzzahlige (Integer-) Felder nehmen nur 2/5 des Speichers in Anspruch, den Fließkomma-Felder benötigen.

BEISPIEL:

```
10 DIM A$(40), B7(15), CC%(4,4,4)
          !       !          !
          !       !          125 (= 5*5*5) Elemente, 250 Bytes
          !       16 Elemente, 80 Bytes
          41 Elemente
```

Mit einer DIM-Anweisung lassen sich auch mehrere Felder dimensionieren, indem die Felder mit Komma getrennt werden. Wird die DIM-Anweisung vom Programm her mehr als einmal durchlaufen, wird das Programm mit der Fehlermeldung REDIM'D ARRAY ERROR abgebrochen. Plazieren Sie daher die DIM-Anweisung stets am Programmanfang.

DO/LOOP/WHILE/UNTIL/EXIT
****

DO [UNTIL Boolesches Arg./WHILE Boolesches Arg.]
[Anweisung(en) EXIT]
LOOP [UNTIL Boolesches Arg./ WHILE Boolesches Arg.]
(Ein Beispiel eines Booleschen Arguments ist A=1 oder H=>57)

Damit werden die Anweisungen zwischen der DO-Anweisung und der LOOP-Anweisung durchgeführt. Werden weder die DO- noch die LOOP-Anweisung von einem nachfolgenden WHILE- oder UNTIL-Parameter abgeschlossen, so erfolgt die Ausführung der dazwischenliegenden Anweisungen unbegrenzt oft. Ist in die DO-Schleife eine EXIT-Anweisung eingebaut, so geht die weitere Programmdurchführung auf die erste der LOOP-Anweisung folgende Anweisung über. DO-Schleifen können nach den FOR-NEXT-Schleifenregeln verschachtelt werden.

Wird UNTIL verwendet, wird die Schleife solange ausgeführt, bis (nach Booleschen Regeln) die Anweisung 'wahr' wird. WHILE stellt praktisch das Gegenteil von UNTIL dar: das Programm durchläuft die Schleife, solange die Anweisung 'wahr' ist.

BEISPIEL:

```
DO UNTIL X=0 OR X=1
     :
        LOOP

DO WHILE A$="": GET A$: LOOP
```

DRAW
****
DRAW [Farbzonen-#], [a1, b1,] [TO a2, b2] , ...

Mit dieser Anweisung lassen sich einzelne Punkte, Linien und Formen zeichnen. Es können die vier Farbzonen-Nummern ebenso angesprochen werden, wie die Anfangs- (a1, b1) und Endpunkte (a2, b2).

BEISPIELE:

Ein PUNKT:	DRAW 1, 100, 50	Kein Endpunkt spezifiziert: bewirkt, daß die Werte a1, b1 für a2, b2 gelten und dadurch ein Punkt entsteht.
Eine LINIE:	DRAW , 10,10, TO 100,60	
	DRAW , TO 25,30	
Ein DREIECK:	DRAW , 10,10 TO 10,60 TO 100,60 TO 10,10	

END
***
END

Erreicht ein laufendes Programm in einer Programmzeile eine END-Anweisung, so stoppt das Programm unverzüglich. Mit dem CONT-Befehl läuft das Programm ab der Anweisung weiter, die der END-Anweisung folgt.

FOR...TO...STEP
****
FOR Variable = Startwert TO Endwert [STEP Schrittweite]

Zusammen mit der NEXT-Anweisung veranlaßt die FOR-TO-Anweisung, daß ein bestimmter Programmabschnitt mehrfach durchlaufen wird. Auf diese Weise lassen sich z.B. Programmpausen unterschiedlichster Länge programmieren, Zählvorgänge durchführen oder bestimmte Arbeiten (z.B. Druckvorgänge) n-fach ausführen.

Nach jedem FOR-NEXT-Durchlauf wird die Schleifenvariable um den Wert 'Schrittweite' hinauf- oder heruntergezählt, wobei der Start- und der Endwert die Grenzwerte der Schleifenvariablen bilden.

Die FOR-Anweisung verläuft nach folgender Logik:
Zuerst wird die Schleifenvariable auf den Startwert gesetzt. Erreicht das Programm eine NEXT-Anweisung, wird die Schrittweite (Standard = 1) zum momentanen Wert der Schleifenvariablen hinzugezählt und danach das Ergebnis mit dem Endwert verglichen.

Ist das Ergebnis kleiner oder gleich (also nicht größer), wird der unmittelbar der FOR-Anweisung folgende Programmteil ausgeführt. Ist ein über dem Endwert liegender Wert erreicht, kommt der Programmteil zur Ausführung, der direkt auf die NEXT-Anweisung folgt. Siehe auch die NEXT-Anweisung. Bei negativer Schrittweite gilt sinngemäß das gleiche.

BEISPIEL:

```
10 FOR L = 1 TO 20
20 PRINT L
30 NEXT L
40 PRINT "BLACK JACK! L = " L
```

Dieses kleine Programm schreibt die Zahlen von Eins bis Zwanzig auf den Bildschirm und schließt mit der Meldung: BLACK JACK! L = 21.

Der FOR-TO-Anweisung kann die Anweisung STEP und die Schrittweite in Form eines arithmetischen Ausdrucks folgen. In diesem Fall wird der der Anweisung STEP folgender Wert statt der Standard-Schrittweite von Eins bei jedem Durchlauf zur Schleifenvariable hinzugezählt.

Schleifen können verschachtelt werden. Verschachtelte Schleifen müssen sorgfältig angelegt werden, und zwar so, daß die äußere Schleife erst dann weiterzählt, wenn die innere ganz durchlaufen ist.

BEISPIEL VERSCHACHTELTER SCHLEIFEN:

```
10 FOR L = 1 TO 100
20 FOR A = 5 TO 11 STEP 2      Diese FOR ... NEXT Schleife ist inner-
30 NEXT A                      halb der äußeren (größeren)
                               verschachtelt.
40 NEXT L
```

GET
***
GET Variable (meist String-Variable) [, ...]

Mit der GET-Anweisung ist es möglich, einzelne Zeichen über die Tastatur in den Computer zu bekommen. Erreicht ein Programm während der Ausführung eine GET-Anweisung, wird die Tastatur abgefragt. Wurde keine Taste gedrückt, so nimmt die GET-Anweisung dies als Null-Zeichen (Leer-String) an. Das Programm läuft weiter, ohne auf eine Tasteneingabe zu warten. Es besteht bei dieser (Eingabe-) Anweisung keine Notwendigkeit, die Taste <Return> zu drücken - vielmehr ist es mit der GET-Anweisung sogar möglich, die Taste <Return> als eigene Eingabe auszuwerten.

Der GET-Anweisung folgt stets ein Variablenname, meist eine Stringvariable. Wird eine numerische Variable benützt und die Eingabe ist keine Zahl, dann stoppt das Programm mit einer Fehlermeldung. Auch kann die GET-Anweisung in einer Schleife eingesetzt werden, wobei ständig nach einer Leereingabe (d.h. es erfolgt kein Tastendruck) abgefragt wird. Solange keine Tastatureingabe erfolgt, verweilt das Programm in dieser Schleife. Für diese Eingabetechnik bietet sich auch die GETKEY-Anweisung an. GET und GETKEY können nur im Programm-Modus angewendet werden !

BEISPIEL:

```
10 GET A$: IF A$ <> "A" THEN 10
```

Diese-GET Schleife wird solange durchlaufen, bis die Taste <A> gedrückt wird - dann erst läuft das Programm weiter.

GETKEY
******
GETKEY Variable (meist String-Variable) [, ...]

Die GETKEY-Anweisung ist der GET-Anweisung sehr ähnlich. Im Gegensatz zur GET-Anweisung wird bei der GETKEY-Anweisung keine Schleife benötigt, um das Programm an dieser Eingabestelle anzuhalten. GETKEY wartet mit dem Programmablauf, bis eine Taste gedrückt wird. Damit lassen sich auf einfache Weise Einzelabfragen der Tastatur programmieren. Diese Anweisung ist nur in einem Programm anwendbar.

BEISPIEL:

```
10 GETKEY A$
```

Bei dieser Programmzeile wartet das Programm, bis eine Taste gedrückt wird. Mit jedem Tastendruck läuft das Programm weiter.

GET#
****
GET# File-Nummer, Variable [, ...]

Die GET#-Anweisung wird benutzt, um über einen zuvor mit OPEN geöffneten Kanal ein Zeichen von einem bestimmten Gerät einzulesen. Es wird auf das Zeichen gewartet, ein Leerstring ist in Wirklichkeit CHR$(0). Ansonsten funktioniert die GET#-Anweisung wie die GET-Anweisung, ist also auch nur im Programm-Modus einsetzbar.

BEISPIEL:

```
100 GET#1, A$
```

GOSUB
*****
GOSUB Zeilen-#

Im Gegensatz zur GOTO-Anweisung merkt sich der COMMODORE 116 die Stelle, an der er die GOSUB-Anweisung erhielt. Erreicht das Programm in der Folge eine Programmzeile mit der Anweisung RETURN, springt das Programm automatisch zurück auf den Befehl, der der GOSUB-Anweisung folgt. Der von der GOSUB-Anweisung abgedeckte Bereich wird Subroutine genannt.

Eine Subroutine ist u.a. dann nützlich, wenn sie von verschiedenen Teilen des Programms aus angesprungen wird. Anstatt einen gleichen Programmteil an den unterschiedlichen Stellen immer wieder zu programmieren, erstellen Sie eine Subroutine und springen sie mit der GOSUB-Anweisung von den verschiedenen Stellen aus an. Siehe auch die RETURN-Anweisung.

BEISPIEL:

```
20 GOSUB 800
    :
    :
800 PRINT "HIER SIND WIR": RETURN
```

bedeutet, daß das Programm zu der bei Zeile 800 beginnenden Subroutine springen muß und dort weiter arbeitet ... bis zum RETURN.

GOTO oder GO TO
**** *****
GOTO Zeilen-#

Nach der Ausführung einer GOTO-Anweisung fährt das Programm am Anfang der angegebenen Zeile fort. Im DIREKT Modus angewandt, ermöglicht die Anweisung GOTO Zeilen-# einen Programmstart ab der eingegebenen Zeilenzahl - ohne dadurch auch sämtliche Variable (wie dies bei RUN der Fall ist) zu löschen.

BEISPIEL:

```
10 PRINT "UEBUNG MACHT DEN MEISTER"
20 GOTO 10
```

Die GOTO-Anweisung in Zeile 20 bewirkt einen ständigen Programmlauf - bis die Taste <Run/Stop> gedrückt wird.

GRAPHIC
****

GRAPHIC Modus [, Clear-Parameter]
GRAPHIC CLR

Der COMMODORE 116 kann in einen der fünf nachstehenden Grafik-Modi versetzt werden:

Modus	Grafik-Modus
0	Text
1	Hochauflösende Grafik
2	Hochauflösende Grafik und Text
3	Mehrfarben-Grafik
4	Mehrfarben-Grafik und Text

Clear-Parameter	Modus
0	Bildschirm wird n i c h t gelöscht
1	Bildschirm wird gelöscht

Bei der Ausführung einer der GRAPHIC-Anweisungen 1 bis 4 wird ein 10 KByte 'bit-mapped' Bereich (= HI-RES-Bereich) abgetrennt, und der Anfang des BASIC-Speicherbereichs über diesen Bereich verlegt. Dieser Hi-Res-Bereich bleibt auch gesperrt, wenn mit der Anweisung GRAPHIC 0 in den Text-Modus zurückgeschaltet wird. Enthält die GRAPHIC-Anweisung im zweiten Parameter eine Eins (z.B. GRAPHIC 0,1), dann wird mit der Umschaltung gleichzeitig der Bildschirm gelöscht.

Mit der Anweisung GRAPHIC CLR wird jedoch der Hi-Res Bereich-(10 KByte = 10240 Byte) wieder als BASIC-Speicherbereich für BASIC-Text und Variable freigegeben.

BEISPIELE:

GRAPHIC 3,1	Wählt den Hi-Res-Grafik-Modus und löscht den Bildschirm.
GRAPHIC CLR	Löscht den GRAPHIC-Bereich und gibt den gesamten Speicherbereich wieder für BASIC frei.

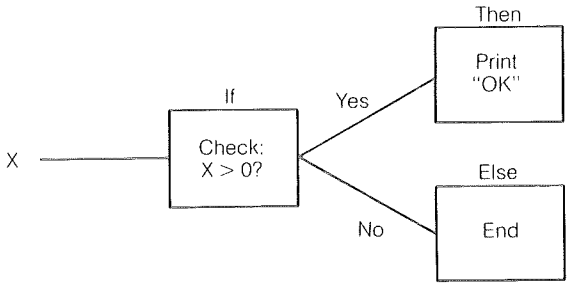
IF...THEN [..:ELSE]
** **** ****

IF Bedingung THEN wenn/dann-Klausel [:ELSE sonst-Klausel]

Mit der IF...THEN-Anweisung ist der COMMODORE 116 in der Lage, abhängig von einer bestimmten Situation zu entscheiden, welche Aktionen auszuführen sind. Wird die Bedingung erfüllt (d.h. ist sie im Booleschen Sinn 'wahr'), dann wird das Programm entsprechend der THEN-Anweisung weiter ausgeführt. Ist die Bedingung nicht erfüllt ('falsch'), dann wird das Programm in der nächsten Zeile fortgesetzt - es sei denn, daß auch noch ein ELSE-Teil besteht.

Die zu erfüllende Bedingung kann eine Variable oder eine Formel sein, wobei der Zustand 'wahr' eintritt, wenn die Bedingung einen Wert ungleich Null ergibt, und der Zustand 'falsch', wenn die Abfrage Null ergibt. In den meisten Fällen enthält diese Bedingung jedoch logische Operatoren (wie =, <, >, <=, =>, <>, AND, OR, NOT).

Wenn ein ELSE-Teil existiert, muß er in derselben Zeile wie die IF...THEN-Anweisung stehen. Der nach ELSE stehende Befehl wird dann ausgeführt, wenn die Bedingung nicht zutraf.



BEISPIEL:

```
50 IF X>0 THEN PRINT "OK": ELSE PRINT "FEHLER"
```

Mit dieser Anweisung wird der Wert X überprüft. Ist X>0, dann wird der THEN-Teil und nicht der ELSE-Teil ausgeführt. Ist X<=0, dann wird der ELSE-Teil ausgeführt und nicht der THEN-Teil.

INPUT
*****
INPUT ["Kommentar";] Variable [, ...]

Mit der INPUT-Anweisung können in einem Programm eine oder mehrere Variablenwerte an den COMMODORE 116 übergeben werden. Dazu hält das Programm an, gibt ein Fragezeichen ('?') auf den Bildschirm aus und wartet, bis eine Eingabe erfolgt, die mit der Taste <Return> abgeschlossen werden muß. Danach läuft das Programm wieder weiter.

Dem Wort INPUT folgt der Variablenname oder eine durch Kommata getrennte Variablenliste. Vor der Variablen kann ein Kommentar zwischen Anführungszeichen (" ") stehen, der sich auf die INPUT-Anweisung bezieht. Wird mit einem Kommentar gearbeitet, so ist dieser nach dem zweiten Anführungszeichen von der Variablen mit einem Semikolon (';') zu trennen.

Wenn Sie mehrere Werte einlesen wollen, müssen sie bei der Eingabe durch Kommata getrennt werden. Diese Anweisung ist nur im Programm-Modus zulässig!

BEISPIEL :

```
10 INPUT "Werte=";A,B,C
RUN
Werte =?  3,4,5.5
```

Werden weniger Werte eingegeben, als Variable vorhanden sind, so werden die fehlenden Werte durch zwei Fragezeichen nachgefordert.

BEISPIEL:

```
10 INPUT "WIE HEISSEN SIE";A$
20 INPUT "UND IHRE LIEBLINGSFARBE";B$
30 INPUT "WIE SCHNELL FLIEGT EINE SCHWALBE";A
```

INPUT#
******
INPUT# File-Nummer, Variable [, ...]

Die Anweisung INPUT# funktioniert wie die INPUT-Anweisung, nur daß die Daten von einem vorher geöffneten (OPEN) File oder Gerät kommen. Ein Kommentar ist in diesem Fall nicht gestattet. Auch diese Anweisung ist nur innerhalb eines Programms zulässig.

BEISPIEL:

```
50 INPUT#2, A$, C, D$          Hierbei müssen die Werte auch im
                               File durch Kommata getrennt sein!
```

LET
***
LET Variable = Formel

Die LET-Anweisung wird benutzt, um einer Variablen einen Wert zuzuweisen. Das Wort LET ist jedoch optional, d.h., es kann auch weggelassen werden. Bei jeder Zuweisung ist sie automatisch (auch ungeschrieben) implementiert. Dabei befindet sich die empfangende Variable (die z.B. das Ergebis einer Berechnung aufnehmen soll) stets links vom Gleichheitszeichen ('=') und der zuzuweisende Wert (die Zahl oder die Formel) immer rechts davon.

BEISPIEL:

```
10 LET A = 5
20 B = 6
30 C = A < B+3
40 D$ = "HALLO"
```

LOCATE
******
LOCATE x-Koordinate, y-Koordinate

Mit der Anweisung LOCATE kann der PC (= Pixel-Cursor) an jede Stelle des Bildschirms plaziert werden. Der PC stellt die momentane Position des für die nächste Zeichnung notwendigen Startpunktes dar. Im Gegensatz zum regulären CURSOR ist der PC nicht als blinkender Punkt sichtbar - aber er ist mit der LOCATE-Anweisung dennoch bewegbar. Zum Beispiel:

LOCATE 160, 100

bringt den PC in die Mitte des Hi-Res-Bildschirmteils. Solange Sie nicht mit der Zeichnung beginnen, werden Sie nichts sehen. Mit Hilfe der RDOT (0)-Funktion können Sie jederzeit herausfinden, auf welcher x-Koordinate, und mit RDOT (1), auf welcher y-Koordinate sich der PC befindet. Mit der Anweisung PRINT RDOT (2) erfahren Sie auch, welcher Farbzonen-Nummer sich der PC gerade bedient. (Sie erinnern sich, daß in allen Zeichen-Anweisungen mit Farbwahl ein entsprechender Farbzonenwert zwischen 0 und 3 auszuwählen ist - entsprechend Vordergrund, Hintergrund, Mehrfarben 1, Mehrfarben 2).

MONITOR
****
MONITOR

Um in den eingebauten Maschinensprachenmonitor zu gelangen, bedienen Sie sich der Anweisung MONITOR. Der Monitor dient der Entwicklung, Fehlerbehebung (Debugging) und Ausführung von in Maschinensprache geschriebenen Programmen. Es empfiehlt sich das Studium weiterführender Literatur.

Um zurück in den BASIC-Modus zu gelangen, tippen Sie ein 'X' ein und drücken die Taste <Return>. Siehe auch im Anhang.

NEXT
****
NEXT [Variable,...,Variable]

Die NEXT-Anweisung wird in Verbindung mit der FOR-Anweisung gebraucht. Erreicht der Computer im Programmablauf eine NEXT-Anweisung, springt er zur zugehörigen FOR-Anweisung zurück und prüft die Schleifen-Variable. (Für mehr Detailinformationen siehe Anweisung FOR). Ist der Schleifendurchlauf beendet, wird das Programm mit der nächsten Programmanweisung hinter der NEXT-Anweisung fortgesetzt. Auf das Wort NEXT folgen meist eine oder mehrere Variablen, die durch Kommata getrennt sind.

Folgt kein Variablenname der Anweisung NEXT, so wird an dieser Stelle die zuletzt im Programm begonnene Schleife geschlosssen. Sind jedoch mehrere Variablen angegeben, so müssen sie entsprechend der Verschachtelung aufgeführt werden.

BEISPIEL:

```
10 FOR L = 1 TO 10: NEXT
20 FOR L = 1 TO 10: NEXT L
30 FOR L = 1 TO 10: FOR M = 1 TO 10: NEXT M, L
```

ON
**
ON Formel <GOTO/GOSUB> Zeilen-# 1 [, Zeilen-# 2,...]

Mit der ON-Anweisung lassen sich GOTO- oder GOSUB-Anweisungen in eine spezielle Version der IF-Anweisung verwandeln. Dem Wort ON folgt zuerst eine Formel, dann entweder die GOTO- oder die GOSUB-Anweisung und dann die durch Kommata getrennten Zeilennummern. Ergibt das numerische Ergebnis der Formel eine Eins, dann wird die erste der Zeilennummern angesprungen. Ist das Resultat eine Zwei, dann die zweite Zeilennummer, usw.

Ist das Ergebnis Null oder größer als die Anzahl der angegebenen Zeilennummern, dann wird die nächste auf die ON-Anweisung folgende Anweisung ausgeführt. Ergibt sich eine negative Zahl, stoppt das Programm mit der Meldung ILLEGAL QUANTITY ERROR.

BEISPIEL:

```
10 INPUT X: IF X<0 THEN 10  Wenn X=1, dann verzweigt die Anweisung ON
                            zur ersten Zeilennummer der Liste (50).
20 ON X GOTO 50, 30, 30, 70 Wenn X=2, dann verzweigt die Anweisung ON
                            zur zweiten Zeilennummer der Liste (30).
25 PRINT "DURCHGEFALLEN": GOTO 10
30 PRINT "ZU HOCH": GOTO 10
50 PRINT "ZU NIEDRIG": GOTO 10
70 END
```

OPEN
****
OPEN File-#,Geräte-# [,Sekundär-Adresse] ["File-Name,Typ,Modus"]

Mit der OPEN-Anweisung ist der COMMODORE 116 in der Lage, mit Geräten wie der Datassette, der Diskettenstation, einem Drucker oder auch dem Bildschirm, Daten auszutauschen. Dem Wort OPEN folgt eine logische File-Nummer, auf die sich in der Folge alle BASIC-Anweisungen beziehen müssen. Diese Zahl kann zwischen 1 und 255 liegen.

Die folgende zweite Zahl ist die Gerätenummer (Primär-Adresse). Beim COMMODORE 116 sind:

```
0   Tastatur
1   Datassette          (Kassettenaufzeichnung)
3   Bildschirm
4   Drucker              (Normal-Konfiguration)
8   Diskettenlaufwerk    (Normal-Konfiguration)
```

Es empfiehlt sich, die Gerätenummer auch als File-Nummer zu verwenden - es merkt sich leichter.

Der zweiten Zahl kann noch eine dritte folgen: die Sekundäradresse. Bei Verwendung der Datassette bewirkt Sekundäradresse 0: Daten lesen, Sekundäradresse 1: Daten schreiben, und Sekundäradresse 2: Daten schreiben mit EOT-Markierung (End Of Tape = Bandende) am Ende. Bei Verwendung der Diskette steht die Sekundäradresse mit der Kanalnummer in Zusammenhang. Bei Druckern wird per Sekundäradresse der Druckmodus bestimmt. Mehr Informationen über die Sekundäradresse bringen die einzelnen Gerätehandbücher.

In Anschluß an die dritte Zahl kann noch ein String folgen, der sowohl ein Diskettenkommando, oder auch der Name des Files auf Kassette oder Diskette sein kann. Direkt vor dem Filenamen kann die Laufwerksnummer, gefolgt von einem Doppelpunkt, stehen. Davor noch zeigt ein 'Klammeraffe' (auf DIN-Tastatur ein §) an, daß ein evtl. schon existierendes File gleichen Namens zu überschreiben ist. Typ und Modus beziehen sich ausschließlich auf die Diskette. File-Typen bzw. Modi sind:

```
PRG = Programm-File
SEQ = Sequentielles File
REL = Relatives File
USR = User-File

READ   = File zum Lesen eröffnen
WRITE  = File zum Schreiben eröffnen
APPEND = File zum Schreiben, und zwar hinter den
         letzten bereits bestehenden Datensatz, eröffnen
```

BEISPIELE:

10 OPEN 3,3	Bildschirm als Gerät öffnen
10 OPEN 1,0	Tastatur als Gerät öffnen
20 OPEN 1,1,0,"HINAUF"	Kassette als Gerät öffnen und lesend nach dem File mit Namen "HINAUF" suchen
OPEN 4,4	Öffnet einen Kanal zum Drucker
OPEN 15,8,15	Öffnet den Befehlskanal zur Diskette
5 OPEN 8,8,12,"§0:TESTFILE,S,W"	Eröffnet ein sequentielles File zum Überschreiben auf Diskette

Siehe auch die Anweisungen: CLOSE, CMD, GET#, INPUT# und PRINT#, sowie die System-Variablen ST, DS und DS$.

PAINT
*****
PAINT [Farbregister-#] [,[a,b] [,Modus]]

Farbregister-# .. (0-3); Standard = 1 = Vordergrund
a,b ..... Koordinaten, skaliert (Standard = PC Position)
Modus ..... 0 = gleiche Farbe wie gewähltes Farbregister
1 = irgend eine Nicht-Hintergrundfarbe

Mit der PAINT-Anweisung lassen sich (geschlossene!) Flächen farbig gestalten. Gefärbt wird diejenige Fläche, in der die angegebenen Koordinaten liegen. Als Grenzen sind Bereiche gleicher Farbe (oder irgend eine Nicht-Hintergrundfarbe, in Abhängigkeit vom gewählten Modus) anzusehen. Am Ende des PAINT-Vorgangs befindet sich der PC wieder im Ausgangspunkt.

WICHTIG: Befindet sich der Startpunkt bereits in der Farbe, die Sie auswählten (oder in einer Nicht-Hintergrundfarbe, wenn Modus 1 gewählt wurde), dann merken Sie keine Veränderung.

BEISPIEL:

```
10 CIRCLE , 160,100,65,50      Zeichnet den Umriß eines Kreises.
20 PAINT , 160,100             Füllt den Kreis mit Farbe.
```

POKE
****
POKE Zieladresse, Wert

Mit der POKE-Anweisung kann jeder Wert im RAM-Speicherbereich des COMMODORE 116 und viele der COMMODORE 116-Input/Output-Register geändert werden. Es wird also ein Byte neu gesetzt. Der POKE-Anweisung folgen stets zwei Zahlen (oder Formeln).

Die erste Zahl bezieht sich auf eine Adresse im Speicher des Computers und kann einen Wert von 0 bis 65535 annehmen. Der Wert der zweiten Zahl liegt im Bereich von 0 bis 255 und kann an die zuvor spezifizierte Adresse geschrieben werden; der bisher dort gespeicherte Wert wird dadurch überschrieben.

BEISPIEL:

```
10 POKE 28000,8          Setzt Inhalt der Adresse 28000 auf 8.
20 POKE 28*1000,27       Setzt Inhalt der Adresse 28000 auf 27.
```

WICHTIG: Die Anweisung PEEK, das Gegenstück zu POKE, finden Sie bei den FUNKTIONEN erklärt.

PRINT
*****
PRINT String/Variable/ [,...]

Die PRINT-Anweisung ist die wichtigste Ausgabe-Anweisung in BASIC. Obwohl die PRINT-Anweisung eine der ersten BASIC-Anweisungen ist, die gelernt wird, bleiben noch viele zusätzliche Anwendungsmöglichkeiten offen. Nach dem Wort PRINT kann stehen:

Texte innerhalb von Anführungszeichen	("Text Zeilen")
Variablen-Namen	(A, B, A$, X$)
Formeln	(SIN(23)-ABS(3*X))
Satzzeichen	(; ,)

Die Texte innerhalb der Anführungszeichen werden wortwörtlich wiedergegeben, d.h. sie werden genau so ausgedruckt, wie sie da stehen. Variablennamen werden mit ihrem Wert (gleichgültig, ob Zahlenwert oder String) ausgedruckt. Auch bei den Formeln wird nur das Ergebnis ausgegeben. Mit Hilfe der Satzzeichen kann das Format der Ausgabe am Bildschirm angepaßt werden.

Das Komma teilt den Bildschirm für den Ausdruck der Daten in vier Bereiche von je 10 Spalten Breite, während bei Verwendung des Semikolons die Daten ohne Zwischenraum ausgedruckt werden. Beide Satzzeichen können das Ende einer Anweisung bilden, was zur Folge hat, daß die nächste PRINT-Anweisung mit oder ohne Abstand (abhängig von Komma oder Semikolon) zur vorhergegangenen direkt anschließend druckt.

BEISPIELE:

```
                                        Ausdruck - Ergebnis
-------------------------------------------------------------------------
10 PRINT "HALLO"                        HALLO
20 A$="IHR ALLE":PRINT"HALLO,"A$        HALLO,IHR ALLE
30 A=4:B=2:PRINT A+B                     6      (Leerstelle für Vorzeichen)
50 J=41:PRINT J;:PRINT J-1               41  40 (Leerstelle für Vorzeichen)
60 C=A+B:D=A-B:PRINT A;B;C,D             4  2  6   2
```

Siehe auch die FUNKTIONEN: POS(), SPC(), und TAB()

PRINT#
******
PRINT# File-#, String/Variable/ [, ...]

Zwischen der Anweisung PRINT# und der vorher beschriebenen Anweisung PRINT besteht ein kleiner Unterschied. Auf das Wort PRINT# folgt stets eine Zahl, die mit dem vorher geöffneten Gerät oder File (s. OPEN) korrespondiert. Dieser Zahl folgt ein Komma und dann die Liste der zu druckenden Strings bzw. Variablen. In dieser Liste bewirkt das Semikolon das gleiche, wie oben bei PRINT beschrieben. VORSICHT! Nicht alle Geräte arbeiten mit den Funktionen Komma (wirkt wie 10 Leerzeichen) und TAB! Den jeweiligen Gerätehandbüchern ist zu entnehmen, wie diese auf Steuerzeichen reagieren, die mit dem PRINT#-Befehl (z.B. in Form eines Strings) gesendet werden.

BEISPIELE:

```
100 PRINT#1, "HALLO IHR ALLE! ",A$,B$,
110 PRINT#1, A ; "," ; B ; "," ; C
```

Ausgabe von Kommata zwischen Zahlenwerten, damit diese von einer INPUT#-Anweisung gelesen werden können.

PRINT USING

****

PRINT USING "Formatvorgabe"; String/Variable [,...]

Die Anweisung PRINT USING ermöglicht es, das Format von Strings oder Zahlenreihen vorzugeben, die auf den Bildschirm, den Drucker oder ein anderes Gerät ausgegeben werden sollen.

Das gewünschte Format steht zwischen Anführungszeichen und wird Formatvorgabe genannt. Daran anschließend folgt ein Semikolon und die Liste der auszudruckenden Strings oder Variablen. Statt der Variablen können ebenso die aktuellen Werte stehen, die ausgegeben werden sollen.

BEISPIELE:

```
5 X=32: Y=100.23: A$="COMPUTER 116"
10 PRINT USING "$##.##";13.25,X,Y
20 PRINT USING "###>#";"CBM",A$
```

Geben Sie nun RUN ein und drücken die Taste <Return>, so wird entsprechend den Angaben in Zeile 10 gedruckt:

```
$13.25    $32.00    $*****
```

Die Sterne (*****) werden anstelle des Y-Wertes gedruckt, weil der Wert Y fünf Dezimalstellen umfaßt und dies mit der Formatanweisung von vier Dezimalstellen "$##.##" nicht in Einklang gebracht werden kann.

Nach den Angaben in Zeile 20 wird gedruckt:

```
  CBM      COMMODORE 116
```

Bevor der String 'CBM' gedruckt wird, werden drei Leerstellen, wie im Formatstring vorgegeben "###>#", gedruckt.

Welche Zeichen beeinflussen was?

Zeichen	Variable	String
Nummern-Raute (#)	X	X
Plus (+)	X	
Minus (-)	X	
Dezimalpunkt (.)	X	
Komma (,)	X	
Dollarzeichen ($)	X	
Vier Exponentzeichen (↑↑↑↑)	X	
Gleichheitszeichen (=)		X
Größer-als-Zeichen (>)		X

Die Nummern-Raute (#) reserviert je Einzelzeichen einen Platz im Ausgabefeld. Enthält die Datenliste mehr Zeichen, als Rauten (#) im Formatstring vorgegeben sind, dann passiert folgendes:

Statt mit Zahlen wird das vorgegebene Datenfeld mit Sternen (*) gefüllt (siehe auch Zeile 10 im obigen Beispiel).

BEISPIEL:

```
10 PRINT USING "####";A
```

Für die nachstehenden Werte von A erfolgt der Ausdruck:

```
A = 12.34          12
A = 567.89        568
A = 123456       ****
```

Bei einem STRING werden nur so viele Stellen gedruckt, wie Rautenzeichen (#) dafür vorgegeben sind; der Rest wird rechts abgeschnitten.

Ein Plus(+)- oder ein Minus(-)-Zeichen kann sowohl an erster wie an letzter Stelle des Formatfeldes stehen - jedoch nicht an beiden Stellen. Steht das Pluszeichen im Formatfeld, so wird es bei positiven, das Minuszeichen bei negativen Zahlen gesetzt.

Steht im Formatfeld ein Minuszeichen, so wird bei positiven Werten ein Leerzeichen, bei negativen das Minuszeichen ausgegeben.

Geben Sie im Formatstring weder ein Plus- noch ein Minuszeichen für einen Zahlenwert an, so wird unmittelbar vor der ersten Stelle ein Minuszeichen oder ein Dollarzeichen gedruckt, wenn die Zahl negativ ist. Ist die Zahl positiv, wird kein Vorzeichen gedruckt - dies bedeutet, daß Ihnen bei positiven Zahlen eine Stelle mehr zur Verfügung steht. Existieren mehr Stellen, als mit den Symbolen '#' bzw. '+' oder ' ' vorgegeben, dann tritt ein 'OVERFLOW' (= Kapazitätsüberschreitung) ein, und das Druckfeld wird mit Sternen (*) gefüllt.

Der Dezimalpunkt (.) im Formatfeld bestimmt die Position des Dezimalpunkts im auszudruckenden Zahlenfeld. Pro Formatstring darf nur ein Dezimalpunkt fixiert werden. Wird im Formatstring kein Dezimalpunkt festgelegt, wird der ausgedruckte Zahlenwert auf die nächstgelegene Zahl gerundet und ohne Dezimalstellen ausgedruckt.

Achten Sie bei der Festlegung des Dezimalpunkts auf die Anzahl der zu druckenden Stellen (einschließlich Minuszeichen, wenn es sich um einen negativen Wert handelt), ob sie sich mit der Anzahl der Rauten (#) vor dem Dezimalpunkt decken. Bei zu vielen Stellen tritt wieder ein OVERFLOW ein, und das Druckfeld wird mit Sternchen (*) gefüllt.

Mit einem Komma im Formatfeld läßt sich ein Komma auch im numerischen Feld plazieren, wobei sich die Stelle im Formatstring mit der Stelle im Ausdruck deckt. Es werden nur Kommata innerhalb einer Zahl gedruckt. Unbenützte Kommata vor der ersten Stelle erscheinen als Füllzeichen. Vor dem ersten Komma im Formatfeld muß mindestens eine Raute (#) stehen.

Werden Kommata in einem Feld definiert und die Zahl ist negativ, dann wird als erstes Zeichen ein Minus (-) gedruckt, auch wenn diese Position durch ein Komma im Formatfeld besetzt wurde.

BEISPIELE:

Formatfeld	Zahlenwert	Ausdruck	Erklärung
##.#+	-.01	0.0-	Die erste Null wird eingefügt, die Eins wird abgerundet.
##.#-	1	1.0	Die Nachkomma-Null wird angefügt.
####	-100.5	-101	Gerundet ohne Kommastellen.
####	-1000	****	OVERFLOW, weil vier Stellen und das Minuszeichen nicht mehr in das Druckfeld passen.
###.	10	10.	Der Dezimalpunkt wird angefügt.
#$##	1	$1	Vorangestelltes Dollarzeichen.

Ein Dollarzeichen ($) im Formatfeld bestimmt, daß auch ein Dollarzeichen mit der Zahl gedruckt wird. Soll sich das Dollarzeichen der wechselnden Stellenzahl anpassen (stets vor der Zahl stehen), dann muß vor dem Dollarzeichen noch eine Raute (#) plaziert werden. Fehlt diese davorstehende Raute (#), dann wird das Dollarzeichen entsprechend seiner Stelle im Formatfeld gedruckt.

Positionieren Sie Kommata und/oder ein Plus- oder ein Minuszeichen in einem Formatstring, in dem auch ein Dollarzeichen enthalten ist, so wird vom Programm ein Komma oder ein Plus/Minuszeichen vor das Dollarzeichen gedruckt.

Die vier Exponentenpfeile (↑↑↑↑) symbolisieren, daß die Zahl im wissenschaftlichen Format (E-Format) ausgedruckt wird. Zusätzlich zu den ↑↑↑↑ ist mit Rauten (#) die Stellenzahl (Feldbreite) zu definieren. Die ↑↑↑↑ können sowohl vor wie auch nach den Rauten im Feldstring plaziert werden. Eine Eingabe von mehr als einem Pfeil (↑) und weniger als vier Pfeile führt zu einem SYNTAX ERROR. Werden mehr als vier Pfeile fixiert, so gelten nur die ersten vier. Weitere Pfeile werden ignoriert.

Mit dem Gleichheitszeichen (=) wird ein String innerhalb des Formatfeldes zentriert. Die Breite des Feldes geben Sie mit der Anzahl der Zeichen # und = im Feldstring an. Enthält der String weniger Zeichen als im Feld angegeben, so wird er im Feld zentriert. Enthält der String mehr Zeichen, als in das definierte Feld passen, werden die überzähligen rechts abgeschnitten und das Feld vollständig ausgefüllt.

Das 'Größer-als' - Zeichen (>) dient dazu, den String im Feld rechtsbündig zu justieren. Die Breite des Feldes geben Sie mit der Anzahl der Zeichen # und = im Feldstring an. Enthält der String weniger Zeichen als im Feld angegeben, so wird er im Feld rechtsbündig justiert. Enthält der String mehr Zeichen, als in das definierte Feld passen, werden die überzähligen rechts abgeschnitten und das Feld vollständig ausgefüllt.

PUDEF
*****
PUDEF "1 bis 4 Zeichen"

Mit der Anweisung PUDEF lassen sich bis zu vier Zeichen in der PRINT USING-Anweisung neu definieren. Sie können Leerstellen, Kommata, Dezimalpunkte und Dollarzeichen durch andere Zeichen ersetzen, indem Sie das neue Zeichen an eine der vier möglichen Stellen in der PUDEF-Anweisung plazieren.

Position 1 Repräsentiert die Leerstelle. Standard ist ein 'BLANK' (' ' = Leerstelle). Sie schreiben an diese erste Stelle jenen Buchstaben, der sich nach Durchführung der Anweisung an Stelle der Leerzeichen im String befinden soll.

Position 2 Steht für die Kommata. Standard ist ein Komma (,).

Position 3 Wird vom Dezimalpunkt eingenommen.

Position 4 Gilt für das Dollarzeichen.

BEISPIELE:

10 PUDEF "*" Anstelle der Leerstellen werden Sterne (*) gedruckt.

20 PUDEF " &" Anstelle der Kommata werden 'kaufm. und' (&) gedruckt.

30 PUDEF " .," Anstelle der Kommata werden Dezimalpunkte und anstelle der Dezimalpunkte Kommata gedruckt.

40 PUDEF "£., " Anstelle des Dollarzeichens wird das Englische Pfund, anstelle der Kommata werden Dezimalpunkte und anstelle der Dezimalpunkte Kommata gedruckt.

Achten Sie auf genaue Eingabe, weil der COMMODORE 116 sonst unerwünschte Zeichen anstelle der Standardwerte ausdruckt.

READ
****
READ Variablen-Liste

Die Anweisung READ wird benutzt, um in DATA-Zeilen gespeicherte Information in Variable zu übertragen. Die Variablenliste der READ-Anweisung kann sowohl Strings als auch numerische Variablen enthalten. Zu beachten ist jedoch, daß kein String eingelesen wird, wenn die READ-Anweisung einen numerischen Wert erwartet; dies führt zu einer Fehlermeldung.

BEISPIEL:

```
READ A$, G$, Y
```

REM
***
REM Bemerkung

Mit Hilfe der Anweisung REM (von REMark = Bemerkung) können an jeder Stelle des Programmlistings Anmerkungen notiert werden. Dadurch lassen sich Teile des Programms erklären, Informationen über den Programmierer festhalten, usw. REM-Anweisungen haben keinen Einfluß auf den Programmablauf, mit der Ausnahme, daß das Programm länger und dadurch langsamer wird. Dem Wort REM kann jede Art Text folgen, aber auch Grafikzeichen (bzw. Großbuchstaben), die jedoch, sofern sie nicht in Anführungsstriche eingeschlossen sind, beim LIST-Befehl als Schlüsselworte ausgegeben werden. Die REM-Anweisung muß die letzte Anweisung einer Zeile sein: nach REM wird kein Befehl mehr ausgeführt, auch nicht nach einem Doppelpunkt!

BEISPIEL:

```
10 NEXT X: REM SCHLEIFEN-VARIABLE X ZAEHLT DURCHLAEUFE
```

RESTORE
****
RESTORE [Zeilen-Nummer]

Mit Hilfe der Anweisung RESTORE ist es möglich, DATA-Zeilen mehrfach zu lesen. Beim Abarbeiten dieser Anweisung wird der DATA-Zeiger wieder auf die erste Position der ersten DATA-Zeile gesetzt. Folgt der RESTORE-Anweisung eine Zeilennummer, dann wird der DATA-Zeiger auf diese Zeile gesetzt.

BEISPIEL:

```
125 RESTORE 200
```

RESUME
******
RESUME [Zeilen-Nummer / NEXT]

Um nach einer Fehleranalyse mit der Anweisung TRAP (siehe dort) den Programmablauf fortzusetzen, wird die Anweisung RESUME gegeben. Folgen der Anweisung keine Parameter, dann fährt das Programm mit dem Befehl fort, welcher den Fehler verursachte. Mit RESUME NEXT setzt das Programm mit dem Befehl hinter demjenigen, der den Fehler erzeugte, fort. RESUME mit einer Zeilen-Nummer funktioniert wie eine GOTO-Anweisung und veranlaßt im Fehlerfall die Programmfortführung ab der angegebenen Zeile.

RETURN
******
RETURN

Die RETURN-Anweisung wird immer in Verbindung mit der GOSUB-Anweisung benützt. Erreicht der Programmablauf eine RETURN-Anweisung, verzweigt das Programm zu dem der zugehörigen GOSUB-Anweisung folgenden Befehl. Existiert keine zugehörige GOSUB-Anweisung, dann kommt die Fehlermeldung RETURN WITHOUT GOSUB ERROR, und das Programm wird gestoppt.

SCALE
*****
SCALE <1/0>

Mit Hilfe der Anweisung SCALE kann im Mehrfarben-Modus und im Hi-Res-Modus die Skalierung der 'Bit Maps' geändert werde.

SCALE 1 schaltet die Skalierung ein. Statt der Koordinatenwerte in nachstehender Tabelle stehen dann in beiden Achsen (X und Y) je die Werte von 0 bis 1023 zur Verfügung. Die Normalwerte lauten hingegen:

Mehrfarben-Modus	X = 0 bis 159	Y = 0 bis 199
Hochauflösende-Grafik-Modus (Hi-Res)	0 bis 319	0 bis 199
Geteilter Bildschirm (beide Modi)	wie oben	0 bis 159

Mit SCALE 0 wird die höhere Skalierung wieder abgeschaltet.

SCNCLR
******
SCNCLR

Mit der Anweisung SCNCLR wird der Bildschirm, unabhängig vom Modus, gelöscht.

SOUND
*****
SOUND (Stimme, Notenwert, Klangdauer)

Die Anweisung SOUND erzeugt Töne bzw. Geräusche mit einer von zwei Stimmen, wobei der Bereich der Notenwerte von 0 bis 1023 reicht und die Klangdauer zwischen 0 und 65535 Fünfzigstel einer Sekunde (= 1311 Sekunden = 21.8 Minuten) erreichen kann.

Stimme #	Ergibt
1	Einzelton (1. Stimme)
2	Einzelton (2. Stimme)
3	Geräusch (2. Stimme)

Wird ein SOUND für die Stimme N aufgerufen, und dieselbe Stimme N erzeugt gerade noch einen Ton, dann wartet das BASIC-Programm mit dem Programmablauf, bis der gegenwärtige Ton ausgeklungen ist. Einen Spezialfall stellt SOUND mit der Klangdauer 0 dar. Damit wird nämlich der gegenwärtig erzeugte Ton abrupt abgebrochen, unabhängig von der eingegebenen Klangdauer. Die Zuordnung der einzelnen Notenwerte zu den tatsächlichen Tonhöhen finden Sie im Anhang dieses Handbuchs.

BEISPIEL:

```
SOUND 2, 800, 3000      Spielt Stimme 2 mit einem Notenwert 800 eine
                        Minute lang.
```

SSHAPE/GSHAPE

****

SSHAPE- und GSHAPE-Anweisungen werden eingesetzt, um Rechteckflächen, die im Mehrfarben- oder Hi-Res-Modus dargestellt sind, als BASIC-Stringvariable abzuspeichern bzw. wieder zu laden. Der Speicherbefehl lautet:

```
SSHAPE String[-Variable], a1,b1 [,a2,b2]
```

```
String[-Variable] .... Variable, in die der String abgelegt wird
a1, b1 ............... Eckkoordinate (skaliert)
a2, b2 ............... a1, b1 gegenüberliegende Eck-
                       koordinate (Standard = PC)
```

Da BASIC die Länge einer Stringvariablen mit 255 Zeichen begrenzt, ist die Größe des abzuspeichernden Bereichs begrenzt. Die pro Fläche notwendige Variablen-Länge läßt sich mit nachstehenden (unskalierten) Formeln berechnen:

```
L(MFM)=INT((ABS(a1-a2)+1)/4+.99)*(ABS(b1-b2)+1)+4
L(HIR)=INT((ABS(a1-a2)+1)/8+.99)*(ABS(b1-b2)+1)+4
MFM = MehrFarbenModus   HIR = Hochauflösende Grafik
```

Die Fläche wird Zeile für Zeile abgespeichert. Die letzten vier Bytes des Strings enthalten jeweils die Spalten- und Zeilenlänge abzüglich Eins (d.h. ABS(a1-a2)) im Low/High-Byte-Format (niederwertiges Byte zuerst, dann das höherwertige Byte (wie beim Programmieren in Maschinensprache). Befinden Sie sich im skalierten Modus, dann muß die Länge in der X-Achse durch 3.2 und in der Y-Achse durch 5.12 dividiert werden.

GSHAPE String[-Variable] [, [a,b] [,Modus]]

Mit der Anweisung GSHAPE läßt sich eine abgespeicherte Fläche wieder an jede Stelle des Bildschirms laden.

String[-Variable] ... Enthält wiederzugebenden String

a,b ..... Linke, obere Ecke, ab der gezeichnet werden soll (Standard = PC), skaliert

Modus ..... Wiedergabe-Modus:
0: Wiedergabe wie aufgenommen (Standard)
1: Wiedergabe in REVERSE-Modus
2: Wiedergabe mit Fläche ODER-verknüpfen
3: Wiedergabe mit Fläche UND-verknüpfen
4: Wiedergabe mit Fläche Exclusiv-ODER-verknüpfen

BEISPIELE:

SSHAPE "SCHIFF", 0, 0 — Speichert Bildschirmbereich ab der linken oberen Ecke bis zur CURSOR Position unter dem Namen "SCHIFF".

GSHAPE "SCHIFF",,,1 — Gibt die abgespeicherte Fläche namens "SCHIFF" mit umgekehrten Hinter- und Vordergrund-Farben ab der CURSOR-Position wieder.

STOP
****
STOP

Die Anweisung STOP hält den Programmablauf an dieser Stelle mit der Meldung BREAK IN LINE # an, wobei die Line # die Programmzeile ist, in der die STOP-Anweisung steht. Mit der Anweisung CONT läßt sich das Programm ab dem der Anweisung STOP folgenden Befehl wieder starten. Die Anweisung STOP wird hauptsächlich bei der Fehlersuche in BASIC--Programmen eingesetzt.

SYS
***
SYS Speicher-Adresse

Dem Wort SYS folgt als Ganzzahl eine Dezimalzahl oder eine Variable im Bereich von 0 bis 65535. Mit der SYS-Anweisung wird ein Maschinensprache-Programm ab der angegeben Speicheradresse gestartet. Die Anweisung ist ähnlich der USR-Anweisung, mit dem Unterschied, daß kein Parameter übergeben wird. Das Maschinenprogramm muß mit einem RTS-Befehl ($60) enden.

TRAP
****
TRAP [Zeilen-#]

Mit der Anweisung TRAP läßt sich innerhalb eines Programms auf alle Fehlersituationen (einschließlich der Taste <Run/Stop> und mit Ausnahme der Fehlermeldung UNDEF'D STATEMENT ERROR) gezielt reagieren. Im Falle eines aufgetretenen Fehlers wird das Fehler-FLAG gesetzt, und die Programmweiterführung wird zu der in der TRAP Anweisung gegebenen Zeilennummer umgeleitet. Die Zeilennummer, in welcher der Fehler auftrat, ist in der Variablen EL gespeichert. Die eigentliche Fehlermeldung enthält die System Variable ER. Mit der String-Funktion ERR$(ER) ist jede Fehlermeldung zur korrespondierenden Fehlersituation ER auslesbar.

WICHTIG: Ein Fehler in einer TRAP-Programmroutine kann nicht ausgelesen (ge'trappt') werden. Mit der Anweisung RESUME kann der Programmablauf wieder aufgenommen werden. Ohne Angabe der Zeilennummer wird die Fehlerverfolgung wieder ausgeschaltet.

TRON
****
TRON

Die Anweisung TRON dient der Fehlersuche. Ab dieser Anweisung wird Abarbeitung des Programms protokolliert. Dabei wird jede Anweisung ausgeführt und die Zeilennummer dieser Anweisung ausgegeben.

TROFF
*****
TROFF

Mit der Anweisung TROFF wird die Fehlersuche (TRON) wieder abgeschaltet.

VOL
***
VOL Lautstärke-Pegel

Mit der Anweisung VOL und einem Parameter zwischen 0 und 8 wird die Lautstärke der im COMMODORE 116 erzeugten Töne und Geräusche eingestellt. Maximale Lautstärke wird mit 8 erreicht, mit 0 wird abgeschaltet. Mit der Anweisung VOL werden beide Stimmen gesteuert.

WAIT
****
WAIT Speicher Adresse, Wert 1 [, Wert 2]

Mit der Anweisung WAIT kann ein Programm angehalten werden, und zwar solange, bis der Speicherinhalt einer vorgegebenen Speicheradresse einen bestimmten Wert erreicht hat, bzw. sich in einer speziellen Art verändert hat. Der Speicheradreßbereich liegt zwischen 0 und 65535. Die Werte 1 und 2 können von 0 bis 255 reichen.

Der Inhalt der Speicherstelle wird zuerst mit dem Wert 2 (falls vorgegeben) Exclusiv-ODER-verknüpft und anschließend mit Wert 1 logisch UND-verknüpft. Ergibt dies ein Resultat von Null, wird der Inhalt der Speicherstelle erneut überprüft. Ist das Ergebnis nicht Null, setzt das Programm bei der nächsten Anweisung fort.

VORSICHT! Nimmt der Inhalt der angesprochenen Speicheradresse nicht den erwarteten Wert an, dann kann der Rechner nur durch einen RESET wieder zum Leben erweckt werden. Gehen Sie deshalb sorgfältig mit diesem Befehl um! Verwenden Sie ihn nur zur Abfrage von Ein-/Ausgabe-Registern, Tastaturabfragen und ähnlichem.

****
* WEITERE INFORMATIONEN ZU DEN GRAFIK-ANWEISUNGEN *
****

Es gibt ein paar Hinweise, die für alle Anweisungen gültig sind, die sich mit Hochauflösender Grafik beschäftigen. Zuerst Hinweise zum Pixel-Cursor (PC). Dieser PC ist dem regulären CURSOR im Text-Modus ähnlich; er bestimmt die nächste Position, an der ein weiteres Pixel gesetzt wird. Im Gegensatz zum normalen CURSOR ist der PC nicht sichtbar. In allen Zeichen-Anweisungen wird der PC verwendet. Zusätzlich läßt sich der PC mit der LOCATE-Anweisung überallhin positionieren - ohne dazwischen etwas zu zeichnen.

Überall, wo (X,Y)-Koordinaten in einer Zeichen-Anweisung vorkommen, können stattdessen auch RELATIVE Koordinaten verwendet werden. Relative Koordinaten bauen auf der momentanen Position des PC auf. Um die relativen Koordinaten zu verwenden, genügt es, vor die eigentlichen Koordinaten ein Plus- oder Minuszeichen zu setzen. Danach bewegt ein Pluszeichen vor der X-Koordinate den PC nach rechts. Ein Minus vor der X-Koordinate bewegt den PC nach links. Ganz ähnlich bewegt ein Minuszeichen vor der Y-Koordinate den PC nach oben und ein Plus nach unten.

Ein Beispiel:

LOCATE +100,-25	Bewegt den PC 100 Pixel nach rechts und 25 nach oben.
DRAW,+10,+10TO100,100	Zeichnet eine Linie 10 Pixel rechts und 10 Pixel unter der gegenwärtigen PC-Position zu dem Punkt mit den absoluten Werten 100, 100.

Auf gleiche Weise lassen sich Abstände und Winkelmaße RELATIV zur gegenwärtigen PC-Position festlegen, indem die beiden Parameter durch ein Semikolon (;) getrennt werden.

Ein Beispiel:

LOCATE 50;45	Bewegt den PC aus seiner momentanen Position um 50 Punkte und unter einem Winkel von 45 Grad weg.

```
**************
* FUNKTIONEN *
**************

*************************
* NUMERISCHE FUNKTIONEN *
*************************
```

Numerische Funktionen werden so bezeichnet, da sie numerische Zahlenwerte ergeben. Der Bereich, den sie abdecken, reicht von mathematischen Rechenfunktionen bis zur Bestimmung einer Bildschirmposition. Numerische Funktionen folgen nachstehenden Regeln:

FUNKTION (Argument),

wobei das Argument je nachdem ein numerischer Wert, eine Variable oder ein String (Zeichenkette) sein kann. Es können mehrere Argumente, auch unterschiedlichen Typs, erforderlich sein.

ABS (X) (Absoluter Wert)

Die Funktion ABS gibt den absoluten Wert einer Zahl an; d.h. ohne Berücksichtigung eines Vorzeichens (+ oder -). Das Ergebnis ist stets Positiv bzw. Null.

ASC (X$)

Mit der Funktion ASC wird der ASCII-Kode (Zahlenwert, CBM-Version, siehe Anhang) des ersten Zeichens im String X$ ausgegeben.

ATN (X) (Arcustangens)

Ergibt den Winkel (in Bogenmaß, siehe SIN), dessen Tangens gleich X ist.

COS (X) (Cosinus)

Ergibt den Cosinus des Winkels X. Der Winkel ist im Bogenmaß einzugeben (siehe SIN).

DEC (H$) (H$ für 'Hexadezimal-String')

Liefert den Dezimal-Wert eines Hexadezimal-Strings (0 < Hexadezimal-String < FFFF , zugelassene Zeichen sind 0-9 und A-F).

BEISPIEL:

N = DEC ("F4") ergibt den Wert 244 .

EXP (X) (Exponential- oder e-Funktion)

Ergibt die X-te Potenz der mathematischen Konstanten e (e=2.71828183).

FNxx (X)

Berechnet den Wert der Benutzer-definierten Funktion FNxx, die in einer DEF FNxx-Anweisung festgelegt wurde.

INSTR (String 1, String 2 [, Start-Position])

Bringt die Position, ab der String 2 in String 1 enthalten ist. Die Suche beginnt ab der Start-Position. Standardmäßig befindet sich die Start-Position am Beginn des ersten Strings. Wird keine Deckungsgleichheit gefunden, ist der Wert 0.

BEISPIEL:

PRINT INSTR ("DIE KATZE IM SACK", "KATZE")

bringt als Ergebnis 5, weil der String (2) KATZE sich ab der fünften Stelle mit dem entsprechenden Teil von String (1) deckt.

INT (X)

Ergibt den ganzzahligen Anteil von X. Das Ergebnis ist immer kleiner oder gleich X. Das bedeutet, daß bei positiven Zahlen alle Stellen nach dem Dezimalpunkt abgeschnitten werden, daß aber negative Zahlen dem Betrag nach größer werden (z.B. INT(-4.5)= -5).

Wird die INT-Funktion zur Rundung eingesetzt, dann lautet die Anweisung INT(X + .5).

BEISPIEL:

```
X = INT(X*100+.5)/100          Damit wird auf den nächst höheren
                               1/100-Wert (z.B. Pfennig) gerundet.
```

JOY (n)

Für den Joystick 1 ist n = 1
Für den Joystick 2 ist n = 2

Jeder Wert größer 127 bedeutet, daß auch der Feuerknopf gedrückt ist. Die einzelnen Richtungen ergeben sich aus nachstehender Matrix:

```
                              RAUF
FEUER = 128 +                  1
*****                    8           2
              LINKS  7     0                 3  RECHTS
                         6           4
                               5
                              RUNTER
```

BEISPIEL:

Die Abfrage von JOY(2) liefert 135, wenn am Joystick #2 gleichzeitig der Feuerknopf gedrückt und der Hebel nach links gehalten wird.

LOG (X)

Ergibt den natürlichen Logarithmus (zur Basis e) vom Wert X. Zur Umwandlung in den Briggsschen Logarithmus (dekadischer Logarithmus, zur Basis 10) wird durch LOG(10) dividiert.

PEEK (X)

Diese Funktion ergibt den Inhalt der Speicheradresse X, wobei die Adresse X zwischen 0 und 65535 liegen muß. Der Inhalt kann Werte von 0 bis 255 annehmen. PEEK wird oft im Zusammenhang mit der Anweisung POKE eingesetzt.

RCLR (N)

Gibt die der Farbquelle N (0 < N < 4) zugeordnete, gegenwärtige Farbzone an. (0= Bildschirm-Hintergrund, 1= Vordergrund, 2= Mehrfarben 1, 3= Mehrfarben 2, 4= Bildschirm-Rand).

RDOT (N)

Gibt die momentanen Koordinaten des PC (=Pixel Cursor) an.

N = 0 für x-Position
N = 1 für y-Position
N = 2 für Farbquellen-#

RGR (X)

Liefert gegenwärtigen Grafik-Modus, wobei X ein 'Dummy' (= beliebiger Füllwert) ist.

RLUM (N)

Gibt die der Farbzone N zugewiesene Farbintensität an.

RND (X)

Mit dieser Funktion erhalten Sie eine Zufallszahl zwischen 0 und 1. Dies ist u.a. hilfreich bei der Programmierung von Spielen, um

Würfelwerte oder ähnliche Zufallswerte zu simulieren, oder auch wichtig für statistische Anwendungen. Die erste Zufallszahl sollte mit der Formel RND(-TI) erzeugt werden, damit bei jedem Start stets eine andere Zahl erzeugt wird. Danach kann die Zahl in der Variablen X eine Eins oder eine andere positive Zahl sein (X repräsentiert die Kernzahl, auf der die Zufallszahl aufbaut). Ist X = 0, dann wird RND von der eingebauten Uhr mit jedem Aufruf von RND auf eine Kernzahl, die sich durch die Uhr ergibt, zurückgesetzt. Ein negativer Wert für X bedeutet bei jedem RND-Aufruf eine Rückstellung der Kernzahl. Auch bei wiederholtem Aufruf der RND-Anweisung wird, bei unverändertem negativen Argument, immer die gleiche Folge von Zufallszahlen gebildet werden. Ein positiver Wert für X ergibt neue Zufallszahlen, die auf der vorangegangenen Kernzahl basieren.

Als Beispiel simulieren wir ein Würfelspiel mit der Formel INT(RND(1)*6+1). Als erstes werden damit die Zufallszahlen des COMMODORE 116, die zwischen 0 und 1 liegen, mit 6 multipliziert. Daraus entstehen Zufallszahlen, die größer als 0 und kleiner als 6 sind (0<n<6). Damit Zufallszahlen im Bereich 1<n<7 entstehen, addieren wir 1, und mit der INT-Funktion werden alle Dezimalstellen abgetrennt. Somit erhalten wir Ganzzahlen zwischen 1 und 6. Zwei Würfel lassen sich simulieren, wenn zwei der durch obige Formel erhaltenen Zahlen zusammengezählt werden.

BEISPIEL:

```
100 X=INT(RND(1)*6)+INT(RND(1)*6)+2      Simuliert zwei Würfel
100 X=INT(RND(1)*1000)+1                 Erzeugt Zahlen von 1-1000
100 X=INT(RND(1)*150)+100                Erzeugt Zahlen von 100-249
```

SGN (X)

Mit dieser Funktion läßt sich das Vorzeichen von X ermitteln. Bei positivem X ergibt sich +1, bei Null 0 und bei negativem X -1.

SIN (X)

Dies ist die trigonometrische Funktion Sinus. Das Ergebnis ist der Sinus des Winkels X, wobei X in Bogenmaß einzugeben ist. Ist X im Gradmaß angegeben, so ist die Umrechnung SIN(X* π /180) zu verwenden.

SQR (X)

Mit der Funktion SQR (Square-Root) wird die Quadratwurzel von X gezogen, wenn X gleich (=) oder größer (>) Null (0) ist. Bei negativen Zahlen kommt die Fehlermeldung ILLEGAL QUANTITY ERROR.

TAN (X)

Der Tangens des Winkels X wird mit dieser Funktion berechnet. Der X-Wert wird im Bogenmaß eingegeben (siehe SIN).

USR (X)

Wird diese Funktion angesprochen, dann springt das Programm in ein Maschinenprogramm, dessen Startadresse in den Speicherstellen 1281 und 1282 steht. Diese Adresse muß zuvor in LO-/HI-Byte-Schreibweise (d.h. Adresse = LO+256*HI, wie in Maschinensprache üblich) mit POKE in 1281 (für LO) und 1282 (für HI) plaziert werden. Der Parameter X wird dem 'Floating Point Akkumulator' = 'FPA' übergeben. Das mit USR aufgerufene Unterprogramm (in Assembler) wertet den Inhalt des 'FPA' aus und gibt einen neuen Wert zurück.

VAL (X$)

Diese Funktion wandelt den String X$ in eine Zahl um. Sie ist besonders wichtig, wenn es um die Umkehr der Funktion STR$ (siehe String-Funktionen) geht. Der String selbst wird Zeichen für Zeichen, von links beginnend, nach rechts - bis zur vorgegebenen Gesamtstellenzahl - nach Zahlen abgefragt. Findet der COMMODORE 116 ein unzulässiges Zeichen, wird der String bis zu diesem unzulässigen Zeichen in eine Zahl konvertiert.

BEISPIEL:

```
10 X = VAL("123.456")          Ergibt  X=123.456
10 X = VAL("3E03")             Ergibt  X=3000
10 X = VAL("12A13B")           Ergibt  X=12
10 X = VAL("RIUO17*")          Ergibt  X=0
10 X = VAL("-1.23.23.23")      Ergibt  X=-1.23
```

```
************************
*  STRING-FUNKTIONEN   *
************************
```

String-Funktionen liefern immer einen String (Zeichenkette) als Ergebnis. Als Argumente können Zahlen oder Strings auftreten.

CHR$ (X)

Die Funktion CHR$ (= Umkehrfunktion zu ASC(X$)) bildet einen String der Länge eins (d.h. ein Zeichen), dessen ASCII-Kode (CBM-Version, siehe Anhang) gleich X ist.

ERR$ (N)

Zeigt den String an, der den Fehlerzustand N (siehe TRAP) beschreibt.

HEX$ (N)

Diese Funktion liefert den vier Zeichen langen Hexadezimalwert einer ganzen Zahl N (0 < N < 65535).

LEFT$ (X$,X)

Ergibt den String, der aus den ersten X Zeichen des Strings X$ - von links gezählt - besteht.

LEN (X$)

Die Funktion LEN gibt die Gesamtlänge (einschließlich Leerstellen und Steuerzeichen) des Strings X$ als Zahl aus, ist also eigentlich eine Numerische Funktion.

MID$ (X$,S,X)

Diese Funktion ergibt einen String, der ab dem S-ten Zeichen eine Anzahl von X Zeichen aus dem String X$ enthält. MID$ kann sowohl auf der linken Seite einer Variablen Zuweisung (MID$ = ...) stehen, als auch als Funktion wirken.

Für das erste Zeichen gilt S=1. Bei fehlender Längenangabe X besteht der Teilstring aus allen Zeichen ab dem S-ten Zeichen. X bzw. S sind auf einen Wertbereich von 0 bis 255 beschränkt.

```
BEISPIEL:
10 A$ = "ZUM LETZTEN MAL":
20 PRINT A$
30 MID$(A$,5,4)=" ERS"
40 PRINT A$                          Ergibt 'ZUM  ERSTEN MAL'.
```

RIGHT$ (X$,X)

Ergibt X Zeichen des Strings X$ - von rechts gezählt.

STR$ (X)

Damit wird ein String gebildet, der der Zahl von X entspricht.

```
BEISPIEL:

X = 99

A$ = STR$ (X)

PRINT A$

RUN

 99
```

```
*************************
*  SONSTIGE FUNKTIONEN  *
*************************
```

FRE (X)

Die Funktion FRE ergibt die Anzahl freier Bytes im Speicher des COMMODORE 116. Der Wert X ist ein beliebiger 'Dummy'.

POS (X)

Teilt die derzeitige-CURSOR Position mit (0 bis 79 in einer logischen Bildschirmzeile). Da der COMMODORE 116 einen 40-Zeichen-Bildschirm besitzt, beziehen sich die Zahlen 40 bis 79 auf die jeweils zweite Bildschirmzeile. Der Wert X ist ein beliebiger 'Dummy'.

SPC (X)

Diese Funktion wird in Verbindung mit der PRINT-Anweisung verwendet, um X Stellen zu überspringen. Dabei kann X den Wert von 0 bis 255 annehmen.

TAB (X)

Auch diese Funktion wird mit PRINT-Anweisungen verwendet. Das nächste Zeichen wird in Spalte X gedruckt. Der Wert X kann zwischen 0 und 255 liegen. Ist auf Druckern nicht anwendbar.

π (= PI)

Das PI-Symbol hat innerhalb einer Gleichung den fest gespeicherten Wert von 3.14159265.

```
*******************************
*  VARIABLEN UND OPERATOREN   *
*******************************

***************
*  VARIABLEN  *
***************
```

Der COMMODORE 116 kennt drei Variablen-Arten in BASIC. Es sind dies: Normale Gleitkomma-Variablen, Ganzzahl-Variablen und String-Variablen (alphanumerische Zeichenketten).

Die Gleitkomma-Variablen können zunächst Werte zwischen -999999999 und +999999999 annehmen - mit neun Stellen Genauigkeit. Ist eine Zahl größer, als sich mit neun Stellen ausdrücken läßt, dann erfolgt die Zahlendarstellung in wissenschaftlicher Notation. Die Gleitkomma-Variable setzt sich dann aus Mantisse, Buchstabe E (für Exponent) und dem eigentlichen Zehner-Exponenten zusammen. Für Exponenten gilt der Bereich von -39 bis +38. Die Zahl 12345678901 wird dann z.B. so dargestellt: 1.23456789E+10.

Ganzzahl-Variablen (Integer) können für ganze Zahlen im Bereich von +32767 bis -32768 (ohne Komma oder Dezimalpunkt!) verwendet werden. Integer sind Zahlen wie 5, 10 oder auch -100. Integer-Zahlen verbrauchen in Feldern wesentlich weniger Speicherplatz als Gleitkomma-Variablen.

String-Variablen werden für Textstücke aus Buchstaben, Zahlen und anderen Zeichen benötigt. Ein typischer Inhalt einer String-Variablen ist z.B. "COMMODORE 116".

```
**********************
*  VARIABLEN-NAMEN   *
**********************
```

Variablen-Namen bestehen aus einem Buchstaben oder einem Buchstaben, gefolgt von einer Zahl, oder aus zwei Buchstaben. Variablen-Namen können auch länger sein, aber zur Erkennung für den Computer zählen nur die ersten zwei Zeichen. In einem Namen darf kein reservierter Name (wie ST oder TI) und kein BASIC-Schlüsselwort (wie ON oder IF) enthalten sein!

Die Integer-Variable (Ganzzahl-Variable) wird durch ein zusätzliches Prozentzeichen ('%') im Anhang an den Variablen-Namen gekennzeichnet. String- oder Text-Variable führen nach dem Namen das Dollarzeichen ('$').

BEISPIELE:

Gleitpunkt-Variablen-Namen: A, A5, BZ

Integer-Variablen-Namen: A%, A5%, BZ%

String-Variablen-Namen: A$, A5$, BZ$

```
**********************
*  MATRIZEN/FELDER   *
**********************
```

Eine Matrix, auch Feld genannt, besteht aus einer Anzahl Variablen, die alle den gleichen Namen tragen, aber zur Festlegung ihrer Position im Matrixfeld eine zusätzliche Indexzahl haben. Die Größe der Felder wird mit der DIM-Anweisung festgelegt, und die Matrix selbst kann für Fließkomma-, Ganzzahl- oder String-Variable festgelegt werden. Dem Matrix-Variablen-Namen folgen - in Klammern - die Indexzahl(en). Der niedrigste benutzbare Indexwert ist immer 0.

BEISPIELE:

Solche (eindimensionale) Felder werden auch 'Vektoren' genannt. Eine Matrix kann jedoch mehr als eine Dimension haben. Ein zweidimensionales Feld besteht aus Zeilen und Spalten, wobei die erste Zahl nach der Variablen (in Klammern) die Zeile und die zweite Zahl (in Klammern) die Spalte definiert - wie beim Koordinatengitter einer Landkarte. Analog kann man sich bei dreidimensionalen Matrizen ein räumliches Gitter vorstellen, während bei höheren Dimensionen die grafische Anschauung versagt.

BEISPIELE:

```
A(7,2),BZ%(2,2,4),Z$(3,2),W7(0,5)
```

```
**********************************
*  RESERVIERTE VARIABLEN-NAMEN   *
**********************************
```

Der COMMODORE 116 kennt sieben Variablen-Namen, die reserviert sind, d.h. die für andere, als die vorgesehene Verwendung, nicht benützt werden können. Es sind dies die Variablen: DS, DS$, ER, EL, ST, TI und TI$. Außerdem können Schlüsselworte, wie TO und IF, nicht als freie Variablen-Namen eingesetzt werden. Es ist auch nicht zulässig, daß Variable gebildet werden, die Schlüsselworte enthalten, wie z.B. SRUN, RNEW oder XLOAD.

ST ist die Status-Variable für alle Ein- und Ausgabe-Operationen, mit Ausnahme normaler Bildschirm- bzw. Tastatur-Operationen. Der in ST stehende Wert ist vom Ergebnis der letzten I/O-Operation abhängig. Detailinformationen entnehmen Sie dem Programmierhandbuch. Eins jedoch vorab: Ist der Statuswert =0, dann verlief der I/O-Vorgang fehlerfrei.

TI und TI$ sind Variablen, die mit der im COMMODORE 116 eingebauten Echtzeituhr direkt gekoppelt sind. Die Systemuhr wird alle 60-tel Sekunden neu gesetzt. Beim Einschalten des COMMODORE 116 startet die Uhr bei 0. Mit jeder Neuzuweisung an die Variable TI$ verändert sich ihr Wert. Über die Variable TI kann Echtzeit ausgelesen werden.

TI$ liest den Wert der 24-Stunden Uhr als String. Die ersten zwei Ziffern stellen die Stunden, die 3. und 4. Ziffer die Minuten und die 5. und 6. Ziffer die Sekunden dar. Mit der Zuweisung TI$="hhmmss" kann jederzeit die Uhrzeit aktualisiert werden (hh=Stunden, mm=Minuten, ss=Sekunden). Die 6-stellige Angabe ist obligatorisch.

BEISPIEL:

TI$ = "142030" setzt die Uhr auf 14 Uhr 20 Minuten und 30 Sekunden.

Mit Zugriff auf die Variable DS kann der Befehlskanal des Disketten-Laufwerks gelesen und die Fehleranzeige des Laufwerks rückgesetzt werden (z.B. nach einer Fehlermeldung). Als Textstring wird die Variable DS$ abgefragt. Diese Statusabfragen sollten nach jeder Disketten-Operation wie z.B. DLOAD oder DSAVE erfolgen.

ER, EL sind Variablen, die in Fehlersuchroutinen (ERROR-TRAPPING) Verwendung finden. Mit ER kann der letzte Fehler seit dem Programmstart ausgelesen werden. EL meldet die Zeile, in der der Fehler auftrat. Mit ERR$ kann vom Programm aus eine der BASIC-Fehlermeldungen ausgedruckt werden. Mit PRINT ERR$(ER) wird die letzte Fehlermeldung ausgedruckt.

```
**********************
*  BASIC-OPERATOREN  *
**********************
```

Die Arithmetischen Operatoren schließen folgende Zeichen ein:

\+ Addition

\- Subtraktion

* Multiplikation

/ Division

↑ Potenzieren (Exponent)

Enthält eine Zeile mehrere Operatoren, dann tritt eine Prioritätenfolge in Kraft, d.h. bestimmte Operationen werden vor anderen durchgeführt:

1. Potenzieren
2. Multiplikation und Division
3. Addition und Subtraktion

Treffen Operationen gleicher Priorität aufeinander, dann erfolgt der Ablauf der Operation von links nach rechts. Wird eine anderslautende Ablaufordnung gewünscht, dann kann dies im COMMODORE 116 BASIC durch Klammern erreicht werden. Operationen in Klammern werden vorrangig ausgeführt. Dabei ist darauf zu achten, daß alle geöffneten Klammern wieder geschlossen werden, sonst kommt es bei laufendem Programm zum Abbruch mit der Fehlermeldung SYNTAX ERROR.

Zum Vergleich der Werte zweier Operanden, etwa zum Test auf Gleichheit oder Ungleichheit, stehen VERGLEICHSOPERATOREN zur Verfügung. Das Ergebnis ist -1 für 'wahr' und 0 für 'falsch'. Arithmetische Operatoren haben stets Vorrang vor Relationalen (Vergleichs-) Operatoren. Es gibt folgende Vergleichsoperatoren:

=	Gleich
<	Kleiner als
>	Größer als
<= oder =<	Kleiner oder gleich
>= oder =>	Größer oder gleich
<> oder ><	Ungleich

Abschließend gibt es drei LOGISCHE OPERATOREN, mit Priorität hinter den Arithmetischen wie den Relationalen Operatoren. Es sind dies:

AND	Logisches UND, aber auch bitweise Verknüpfung
OR	Logisches ODER, aber auch bitweise Verknüpfung
NOT	Logisches NICHT, aber auch bitweise Inversion + 1

Sie werden meist in Erweiterung von IF...THEN-Anweisungen genützt. Werden sie in Verbindung mit Arithmetischen Operatoren benützt, dann rangieren sie noch hinter + und -.

BEISPIELE:

IF A=B AND C=D THEN 100	Erfordert beide Teilbedingungen A=B, C=D.
IF A=B OR C=D THEN 100	Erfordert mindestens eine Teilbedingung.
A=5:B=4:PRINT A=B	Druckt einen Wert von 0 aus.
A=5:B=4:PRINT A>B	Druckt einen Wert von -1 aus.
PRINT 123 AND 15:PRINT 5 OR 7	Druckt Werte von 11 und 7 aus.