IN0120 1A

M20

BASIC 8000

unter PCOS (1.3)

olivetti L1

© Copyright 1983 by Olivetti

Druck Nr. 1701/5/9.83

Das Handbuch dient der Information, sein Inhalt ist ohne ausdrückliche schriftliche Vereinbarung nicht Vertragsgegenstand. Technische Änderungen behalten wir uns vor. Die angegebenen Daten sind lediglich Nominalwerte.

BASIC 8000

unter PCOS (1.3)

BASIC-HANDBUCH M 20

1.) DIE PROGRAMMIERSPRACHE BASIC

1.1 DEFINITION BASIC

1.2 ANWAHL "BASIC" IN PCOS

1.3 ZULÄSSIGER ZEICHENSATZ IN BASIC

1.3.1 Eingabelänge und Abschlußtasten
1.3.2 Die Umschalttasten
1.3.3 Die Groß- und Kleinschreibung
1.3.4 Spezielle Kontrollzeichen
1.3.5 Bildschirmcursor-Steuertasten
1.3.6 Programmierbare Tasten

2.) ELEMENTE DER SPRACHE BASIC

2.1 AUFBAU EINES BASIC-PROGRAMMES

2.2 BEFEHLE

2.3 ANWEISUNGEN

2.4 FUNKTIONEN

2.5 DATEN

2.5.1 Numerische Daten

2.5.1.1 Der Typ "Integer"
2.5.1.2 Der Typ "Single Precision" (einfache Genauigkeit)
2.5.1.3 Der Typ "Double Precision" (doppelte Genauigkeit)
2.5.1.4 Typenumwandlung

2.5.2 Strings
2.5.3 Konstanten und Variablen

2.5.3.1 Konstanten
2.5.3.2 Variablen
2.5.3.3 Arrays

2.6 OPERATOREN

2.6.1 Mathematische Operatoren
2.6.2 Vergleichsoperatoren
2.6.3 Logische Operatoren
2.6.4 Stringoperatoren

2.7 AUSDRÜCKE

2.7.1 Numerische Ausdrücke
2.7.2 Stringausdrücke
2.7.3 Vergleichsausdrücke und logische Ausdrücke

2.8 DATEN IN EINEM PROGRAMM

2.9 ZUSAMMENFASSUNG DER DEFINITIONEN

3.) ARBEITSWEISE DES INTERPRETERS

3.1 EINGABE VON BEFEHLEN

3.1.1 Übergang in den Command Mode
3.1.2 Eingaben in den Command Mode
3.1.3 Verlassen des Command Mode

3.2 DIREKT-MODE

3.3 ERSTELLEN EINES PROGRAMMS

3.3.1 Vorbereiten des Arbeitsspeichers
3.3.2 Eingeben von Programmzeilen
3.3.3 Möglichkeiten nach dem Ende der Eingaben

3.4 KORREKTUR BESTEHENDER PROGRAMME

3.4.1 Einfügen zusätzlicher Programmzeilen
3.4.2 Ersetzen von Programmzeilen
3.4.3 Löschen von Programmzeilen
3.4.4 Ändern bestehender Programmzeilen (Edit-Mode)

3.5 AUSFÜHREN VON PROGRAMMEN

3.5.1 Programmstart
3.5.2 Unterbrechen der Programmausführung
3.5.3 Testen und Korrektur von Programmen (Debug)
3.5.4 Ende einer Programmausführung
3.5.5 Verbinden mehrerer Programme

4.) DIE VERWENDUNG DER SPRACHELEMENTE

4.1 BILDUNG VON VARIABLENNAMEN; FORMATBESCHREIBUNG; SYNTAX; FILENAMEN

4.1.1 Bildung von Variablennamen
4.1.2 Formatbeschreibung und Syntax von Befehlen, Anweisungen und Funktionen
4.1.3 Bildung von Filenamen

4.2 GEGLIEDERTE ZUSAMMENSTELLUNG DER BEFEHLE, ANWEISUNGEN UND FUNKTIONEN

4.2.2 Liste der für Variablennamen gesperrten Sprachelemente

4.3 FUNKTIONELLER ZUSAMMENHANG DER BEFEHLE, ANWEISUNGEN UND FUNKTIONEN

4.3.1 Erstellen von Programmen
4.3.2 Vereinbarungsteil
4.3.3 Zuweisungen
4.3.4 Stringverarbeitung

4.3.4.1 Stringverkettung (String-Addition)
4.3.4.2 Teilstringbildung (Substrings) und Ersetzungen in Strings
4.3.4.3 Konvertierungen

4.3.5 Ablaufsteuerung
4.3.6 Schleifen
4.3.7 Unterprogramme und selbstdefinierte Funktionen

4.3.7.1 Unterprogramme (Subroutine)
4.3.7.2 Selbstdefinierte Funktionen

4.3.8 Inputs über Tastatur
4.3.9 Outputs

4.3.9.1 Bildschirm (alphanumerisches Arbeiten)
4.3.9.2 Drucker

4.3.10 Externe Datenfiles

4.3.10.1 Organisationsmöglichkeiten
4.3.10.2 Befehle, Anweisungen und Funktionen

4.3.11 Spezielle und reservierte Variablen
4.3.12 Fehlerbehandlungsroutine

4.3.14 Graphisches Arbeiten

4.3.17 Deutscher Zeichensatz (ISO-Code-Tabelle)
4.3.18 Bildschirmorganisation und Windows

4.3.18.1 Zeichendarstellung und Bildschirmorganisation
4.3.18.2 Arbeiten mit Windows
4.3.18.3 Verwendung von Farbe

5.) ALPHABETISCHE BESCHREIBUNG DER BASIC-BEFEHLE, -ANWEISUNGEN UND -FUNKTIONEN

7.) MELDUNGEN DES INTERPRETERS; FEHLER-CODES

7.1 Liste der Fehler-Codes (Basic-Fehler)
7.2 Erläuterung der Fehler-Codes (Basic-Fehler)
7.3 PCOS-Fehler

8.) ANHANG

I ISO-Codes
II Steuerzeichen für Drucker

IV PCOS-Befehle (Überblick)

9.) REGISTER

9.1 Alphabetisches Verzeichnis der Befehle, Anweisungen und Funktionen

1. DIE PROGRAMMIERSPRACHE BASIC

Personal-Computer werden nur durch ihre Programmierbarkeit in einer höheren Programmiersprache universell einsetzbar. Dabei hat BASIC wegen seiner leichten Erlernbarkeit und großen Flexibilität im kommerziellen und technischen Bereich für Personal-Computer besondere Bedeutung erlangt.

1.1 DEFINITION BASIC

BASIC ist der Name der Programmiersprache, die aus den Anfangsbuchstaben von Beginners' All Purpose Symbolic Instruction Code gebildet wurde. Dieses Handbuch bezieht sich auf die Sprachform von BASIC, die für den Olivetti L1 M 20 Personal Computer realisiert wurde.

Die Bezeichnung lautet:

L1.M20 BASIC-8000 Rev.5.20

Alle Beschreibungen beziehen sich auf diese von Olivetti verwendete Sprachversion.
Programme der Programmiersprache werden interpretierend ausgeführt. Dadurch wird große Flexibilität in den Sprachmöglichkeiten und bei der Erstellung und beim Testen von Programmen erreicht.

Bei Übernahme von Programmen anderer Sprachformen von BASIC auf M 20 muß geprüft werden, ob die zulässigen Regeln für das BASIC des L1.M 20 eingehalten sind.

1.2 ANWAHL "BASIC" IN PCOS

Nach dem Laden des Betriebssystems befindet sich das System in der PCOS-Ebene.

Um BASIC-Programme erstellen oder ausführen zu können, muß der BASIC-Interpreter aktiviert werden.
Dazu stehen folgende Möglichkeiten zur Verfügung:

1.) Eingabe des PCOS-Befehles **ba**

2.) Vorhandensein eines Programmes INIT.BAS

3.) Vorwahl während der Ausführung der Selbstüberprüfung durch Eingabe von **b**

Jede dieser Möglichkeiten bewirkt aus der PCOS-Ebene die Aktivierung und die Übergabe der Kontrolle an den BASIC-Interpreter. Am Bildschirm wird die jeweils geltende Fassung des aktuellen Betriebssystems angezeigt.
Der dem Anwender verbleibende Speicherplatz hängt ab von

- der Speicherausrüstung und Hardwarekonfiguration des M 20
- dem aktuellen Betriebssystem
- Der Anzahl von PCOS-Befehlen, die mit dem Befehl **pl** in den Speicher geladen wurden.

Zu 1.) Der Systembefehl **ba** (basic) bewirkt den Übergang in BASIC. Wird **ba** Programmname angegeben, so wird das entsprechende Programm gesucht, in den Arbeitsspeicher geladen und die Ausführung des Programms gestartet.

Zu 2.) Der M 20 sieht einen Selbststart vor. Enthält eine Diskette das Programm INIT.BAS, so wird nach dem Laden des Systems in die BASIC-Ebene verzweigt und mit der Ausführung des Programmes INIT.BAS begonnen.

Zu 3.) Nach dem Einschalten des Systems oder nach Drücken von RESET wird vom System ein Selbsttest durchgeführt. Wird während der Ausführung dieses Tests **b** eingegeben, so wird ebenfalls sofort der BASIC-Interpreter aktiviert. Ist ein Programm INIT.BAS vorhanden, so wird es in diesem Fall nicht ausgeführt.

Der Betriebszustand BASIC kann verlassen werden durch:

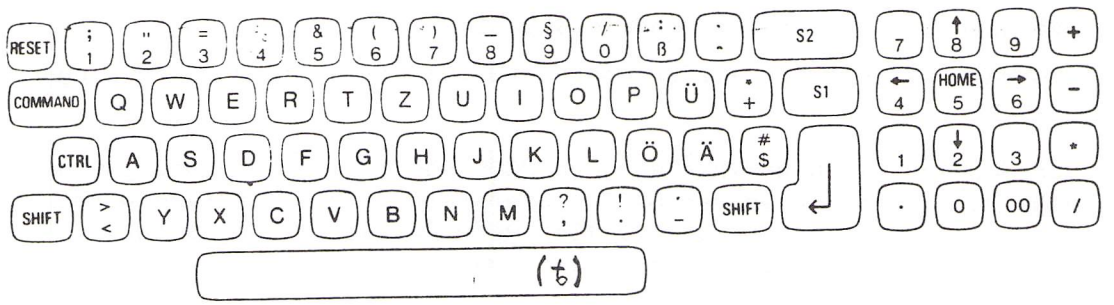
- Eingabe des Befehles **SYSTEM** CTRL
- Drücken der Tasten ~~shift~~ (Umschalttaste für Großschreibung) in Verbindung mit RESET. Dies bewirkt ein Neuladen des Systems.

1.3 ZULÄSSIGER ZEICHENSATZ IN BASIC

Die Tastatur des M 20 sieht einige Tasten vor, die in BASIC eine besondere Bedeutung haben. Ebenso hat die Programmiersprache BASIC Vorschriften über die Zulässigkeit von bestimmten Zeichen.

Der nachfolgende Abschnitt faßt diese Regeln über die Verwendung der Tastatur zusammen und erklärt die Schreibweise, die für bestimmte Tastenkombinationen in diesem Handbuch gewählt wurde.

Deutsche Tastatur M 20



1.3.1 Eingabelänge und Abschlußtasten

Alle Eingaben werden im aktiven Window angezeigt. Übersteigt die Anzahl der eingegebenen Zeichen die Länge der Zeile im Window, so wird in der nächsten Zeile fortgesetzt.

Die maximale Länge einer Eingabezeile beträgt 255 Zeichen.
Soll die Eingabe früher abgeschlossen werden, was im allgemeinen der Fall ist, so sieht das System 3 Abschlußtasten vor.

Diese Tasten sind ↵, S1, S2.

Alle drei Tasten erzeugen das Abschlußzeichen:

CR (Carriage-Return, ISO-Code 13).
Es wird in der Folge generell für das Drücken einer der drei Abschlußtasten die Bezeichnung CR geschrieben.

Anmerkung:

Auf Systemebene und beim Arbeiten mit dem Interpreter haben alle drei Abschlußtasten die gleiche Wirkung.
In Anwenderprogrammen kann durch Aufruf des PCOS-Befehles **lt** geprüft werden, welche der Abschlußtasten gedrückt wurde.

Wird die volle Zeilenlänge eingegeben, so wird nach dem 256. Zeichen automatisch die Eingabe unterbrochen.
Das gilt sowohl bei der Durchführung von Eingaben im Command-Mode des Interpreters als auch bei der Ausführung von Programmen.

1.3.2 Die Umschalttasten

Am linken Rand der Tastatur sind drei Umschalttasten vorgesehen.
Sie ermöglichen die volle Verwendung aller 255 möglichen Textzeichen, die jedoch nicht alle druckbar sind und teilweise eine spezielle Bedeutung haben.

Die Umschalttaste in der untersten Tastaturreihe links (SHIFT) - sie ist wie bei Schreibmaschinen auch auf der rechten Seite der untersten Tastenreihe vorhanden - bewirkt den Wechsel zwischen Groß- und Kleinschreibung bzw. den Wechsel zu dem Zeichen, das bei doppelt beschrifteten Tasten oben angegeben ist.

Darüber liegt die blaue Umschalttaste. Sie wird in der Folge als Taste CONTROL bezeichnet. Die Taste CONTROL bewirkt die Erzeugung von Steuerzeichen. Im Handbuch wird die Verwendung einer Taste gemeinsam mit CONTROL durch Negativdarstellung dieser Taste dargestellt.

H bedeutet, daß die Taste H gemeinsam mit CONTROL gedrückt werden muß.

Die gelbe Umschalttaste wird mit COMMAND bezeichnet. Sie wird überwiegend zur freien Belegung von Tasten (z.B. innerhalb von Programmen) verwendet, so daß diese Taste gemeinsam mit COMMAND die Wirkung einer Funktionstaste hat.

1.3.3. Groß- und Kleinschreibung

Mit der Tastatur des M 20 können sowohl Groß- als auch Kleinbuchstaben erzeugt werden. Um für die Eingabe von Programmen die Tastaturhandhabung zu erleichtern, werden Kleinbuchstaben in den sinnvollen Fällen automatisch in Großbuchstaben umgewandelt.

Diese Verwandlung erfolgt in der BASIC-Ebene für:

Alle BASIC-Schlüsselwörter
Alle Variablennamen

Sie erfolgt nicht für Stringkonstanten (d.h. in Anführungszeichen eingeschlossene Strings), also auch nicht für Filenamen, die in Form von Stringkonstanten definiert sind.

Um bei der Eingabe von Folgen von Großbuchstaben nicht ständig die Umschalttaste (shift-Taste) drücken zu müssen, sieht das System eine Feststellfunktion vor.

Mit Eingabe von

COMMAND - (gleichzeitig) (- unten rechts verwenden)

kann das Umschalten auf Großbuchstaben erreicht werden.
Die Großschreibung gilt so lange, bis erneut COMMAND- gedrückt wird.

1.3.4 Spezielle Kontrollzeichen

Wesentliche Steuerfunktionen werden über Kontrollzeichen erreicht. Sie werden durch Drücken der blauen Taste (Taste CONTROL) gemeinsam mit einer anderen Taste erzeugt und in diesem Handbuch durch **Negativdarstellung** dieses Zeichens dargestellt.

C Unterbricht die Ausführung des Programmes oder von unterbrechbaren Befehlen, z.B. **LIST** (siehe Kapitel 3).

G Erlaubt die verdeckte Eingabe von Zeichen. Wird **G** gedrückt, so werden nachfolgend eingegebene Zeichen nicht am Bildschirm angezeigt, so lange, bis entweder erneut **G** oder CR gedrückt wird.
Ob **G** aktiviert ist, kann aus der Form des Cursors ersehen werden.

H Löscht das jeweils zuletzt eingegebene Zeichen und der Cursor wird um eine Position zurückgesetzt.
H ist nicht möglich, während G aktiviert ist.

S Unterbricht die Ausführung eines Programmes oder Befehles. Das Drücken einer beliebigen anderen Taste bewirkt die Fortsetzung der Ausführung.

Neben diesen Kontrolltasten gelten im Edit-Mode weitere Kontrolltasten als Edit-Subbefehle. Sie sind in Kapitel 3.4.4 bzw. beim Befehl **EDIT** beschrieben.

1.3.5 Bildschirmcursor-Steuertasten

Im numerischen Teil der Tastatur (rechts) sind die Tasten mit den Beschriftungen ←,→,↑,↓, und HOME zu finden. Werden die betreffenden Tasten und SHIFT gedrückt, kann damit der Bildschirmcursor gesteuert werden. Dies gilt aber nur für BASIC-Programme, die dies vorsehen. Die Bildschirmcursor-Steuerung ist im Zusammenhang der Tastatur-Inputs in BASIC zu programmieren.

Die Bedeutung der Cursor-Steuertasten ist dann wie folgt:

→ : Cursor um eine Position nach rechts
← : Cursor um eine Position nach links
↑ : Vorbereitung zum Einfügen genau eines Zeichens an der Cursor-Position
↓ : Vorbereitung zum Löschen von **n** Zeichen ab der Cursor-Position
HOME : Löschen des gesamten angezeigten Eingabefeldes

1.3.6 Programmierbare Tasten

Mit den Umschalttasten (shift, CONTROL, COMMAND) und den Tasten des alphanumerischen Teiles der Tastatur können alle in einem Byte speicherbaren binären Werte erzeugt werden, es ist jedoch nicht allen Tasten standardmäßig eine Bedeutung zugeordnet.
PCOS sieht jedoch über den PCOS-Befehl **pk** diese Möglichkeit vor.
Jeder Taste kann ein beliebiger String als Bedeutung zugeordnet werden.
Jedes Drücken der entsprechenden Taste oder Tastenkombination (z.B. COMMAND + Taste) erzeugt die Zeichenfolge, die mit **pk** zugeordnet wurde.

Üblicherweise bleibt diese Tastenbelegung so lange aufrecht, bis das System neu geladen wird. Es kann jedoch über **ps** eine vom Anwender gewählte Tastenbelegung im Systemfile gespeichert werden, sodaß bei jedem Laden des Systems diese Tastenbelegung wieder verfügbar ist.

2. ELEMENTE DER SPRACHE BASIC

Bei der Erstellung eines Programmes wird vom Ersteller ein Ablauf festgelegt. Dieser Ablauf wird in Abhängigkeit von den aktuellen Daten bei jeder Ausführung des Programmes vom M 2Ø ausgeführt.
Es werden im folgenden Kapitel alle Sprachelemente beschrieben, die
- zur Eingabe und zum Testen eines Programmes benötigt werden
- die ein BASIC - Programm bilden können.

2.1 AUFBAU EINES BASIC-PROGRAMMES

Ein BASIC-Programm besteht aus einer Folge von Anweisungen. Sie bilden Programmzeilen, die mit einer Zeilennummer versehen sind. Enthält eine Programmzeile mehrere Anweisungen, so sind die Anweisungen durch : (Doppelpunkt) getrennt. Eine Programmzeile kann maximal 255 Zeichen lang sein.

Eine Programmzeile hat daher folgendes Format:

Zeilennr. Anweisung [:Anweisung] ...

Dabei bedeutet:

Zeilennummer:	ganze positive Zahl im Bereich von Ø bis 65529
Anweisung:	Eine Folge von Sprachelementen, die nach den Vorschriften in Abschnitt 2.3 aufgebaut sind.

Die Zeilennummern bestimmen die Reihenfolge, in der die Programmzeilen im Arbeitsspeicher abgestellt werden. Dieser Programmaufbau legt auch den normalen Ablauf der Verarbeitung fest. Die Reihenfolge der Verarbeitung kann durch Sprachelemente verändert werden.
Innerhalb einer Programmzeile erfolgt die Ausführung der Anweisungen von links nach rechts.
Die Ablaufsteuerung kann sowohl durch Anweisungen im Programm erfolgen als auch während der Ausführung durch Eingabe von Befehlen im Direkt-Mode (vgl. Kapitel 3) erreicht werden.
Das ordnungsgemäße Ende eines Programmes kann festgelegt werden durch:
- die Ausführung der Anweisung **END**,
- die Verzweigung in ein anderes Programm.

2.2 BEFEHLE

Befehle sind BASIC-Elemente, die den Zweck haben, eine unmittelbare Aktion des Computers zu bewirken. Befehle arbeiten mit Programmen oder Datenfiles, aber nicht mit Daten.
Befehle werden daher üblicherweise verwendet für:
- die Eingabe von Programmen,
- das Ändern, Speichern, Dokumentieren,
- einzelne Operationen mit Datenbeständen

Beispiel:
Wenn ein Programm eingegeben wurde, soll zur Kontrolle und Dokumentation ein Protokoll am Drucker erstellt werden.
Die Eingabe von

LLIST

bewirkt das Drucken aller im Arbeitsspeicher vorhandenen Programmzeilen in der Reihenfolge der aufsteigenden Zeilennummern.

Anmerkung: Einige Befehle können auch sinnvoll innerhalb von Programmen verwendet werden.

2.3 ANWEISUNGEN

Ein Programm ist die Festlegung eines Ablaufes. Welche Aktion bei der Ausführung eines Programmes durchgeführt wird, ist durch Anweisungen bestimmt.
Eine Anweisung besteht aus

- einem oder mehreren Schlüsselwörtern, die den Typ der Anweisung bestimmen;
- den Parametern der Anweisung. Sie bestimmen, mit welchen Daten die Anweisung auszuführen ist.

a) Die Anweisungen werden verwendet als Bestandteil von Programmzeilen.
Beispiel:

10 **PRINT** A	ist eine Anweisung.
10	ist die Zeilennummer der Programmzeile
PRINT	ist das Schlüsselwort. Die Anweisung bewirkt eine Ausgabe am Bildschirm.
A	ist der Datenteil der Anweisung. Er bestimmt, daß der Wert der Variablen A am Bildschirm anzuzeigen ist.

Anweisungen müssen den syntaktischen Regeln entsprechend eingegeben werden (vgl. Kapitel 4.1.2).
Da sie nicht unmittelbar ausgeführt werden, erfolgt keine unmittelbare Überprüfung auf ihr gültiges Format. Diese Überprüfung erfolgt erst bei der Ausführung.

b) Zur direkten Eingabe
Beispiel:
Ein Programm soll auf seine Korrektheit geprüft werden. Dann soll der Wert von Daten an einer bestimmten Stelle geprüft werden.
Das Programm wird ausgeführt und an der gewünschten Stelle durch Drücken der blauen Taste in Verbindung mit der C-Taste (C) (ohne Eingabe-Abschlußtaste) angehalten. Es soll der Wert der Variablen A (siehe Abschnitt 2.5) überprüft werden.

Die Eingabe

PRINT A	bewirkt das gewünschte Ergebnis.
PRINT A	ist eine Eingabe, die unmittelbar ausgeführt wird, aber nicht der Definition des Befehls aus 2.2 entspricht.

Die Unterscheidung zwischen diesen beiden Fällen wird in Kapitel 3 erläutert.

2.4 FUNKTIONEN

Eine Funktion ist eine Berechnungsvorschrift, die in Abhängigkeit von Parametern (Argumenten) genau einen Wert als Ergebnis liefern.
Viele Funktionen sind in BASIC standardmäßig enthalten.
Diese Funktionen heißen Standardfunktionen. Sie sind zu ihrer Identifikation mit einem reservierten Namen versehen.
Zusätzlich hat der Anwender die Möglichkeit, in einem Programm selbst Funktionen zu vereinbaren und an beliebigen nachfolgenden Stellen im Programm zu verwenden (siehe Abschnitt 4.3.7).
Eine Funktion kann immer nur innerhalb von Anweisungen aufgerufen werden.
Sie kann nicht allein stehen. Es muß in einer Anweisung festgelegt werden, was mit dem Ergebnis der Funktion gemacht werden soll.

Bei den von Anwendern definierten Funktionen ist die Definition der Funktion ("der Vorschrift") eine Anweisung.
Der Aufruf der Funktion (mit ihrem Namen und aktuellen Parameter) ist Bestandteil jener Anweisungen, in denen der Funktionsname aufgerufen wird.

2.5 DATEN

Die Anweisungen bilden die Vorschrift, nach denen die Daten in einem Programm behandelt werden.
Die Daten sind in unterschiedliche Typen nach verschiedenen Gesichtspunkten gegliedert. Ihre interne Darstellung unterliegt bestimmten Vorschriften, die in den nachfolgenden Abschnitten erläutert werden.

Daten können in einem Programm konstant, (während der Ausführung nicht veränderbar) oder variabel (während der Ausführung veränderbar) sein.
Daten kommen daher vor als:

- Konstanten
- Variablen.

Eine wesentliche Unterscheidung ist die Art der Daten. Daten können "Zahlen" sein, mit denen im Programm arithmetische Operationen ausgeführt werden (oder ohne Umwandlung ausführbar wären).
Solche Daten sind

Numerische Daten.

Im Unterschied dazu sind beliebige Zeichenfolgen als Daten zulässig.
Die Bedeutung solcher Zeichenfolgen wird nur durch die Interpretation des Anwenders bestimmt.
(Beispiel:
Die Zeichenfolge "12345" wird vom Anwender als Zahl interpretiert.
Vom Programm wird sie jedoch nur als Folge von 5 Zeichen behandelt.)
Solche in einem Programm vorkommende Zeichenfolgen heißen:

Strings.

2.5.1 Numerische Daten

Numerische Daten sind Daten, mit denen ohne Umwandlung arithmetische Operationen (z.B. Addition) ausgeführt werden können.
Solche Daten werden im Speicher des Rechners in spezieller Form gespeichert. Um eine möglichst optimale Verwendung des Speicherplatzes zu gewährleisten und unnötige Rechenzeiten zu vermeiden, werden numerische Daten entsprechend ihrem zulässigen Wertebereich in unterschiedliche Gruppen eingeteilt:

- Integer (ganzzahlig)
- Single Precision (einfache Genauigkeit)
- Double Precision (doppelte Genauigkeit)

Numerische Daten können auf zwei Arten dargestellt werden:

- Festkommaformat
- Gleitkommaformat
  (floating-point-Format)

Die Bezeichnungen sind irreführend, da am M 20 das Dezimalkomma stets durch einen Dezimalpunkt ersetzt wird.

Zahlen in Festkommaform bestehen aus einem Vorzeichen, den Vorkommastellen, dem Dezimalpunkt und den Nachkommastellen.

Beispiele:

3.14 -68.2 -.523 1.0001

Zahlen in Gleitkommaform bestehen aus einer Mantisse incl. Vorzeichen, der Basis 10 und einer Hochzahl incl. Vorzeichen (Exponent). Der Wert der Zahl ergibt sich aus der Multiplikation von Mantisse mit 10 hoch Exponent.

Beispiele:

$-6.3*10^{2} = -630$
Mantisse: -6.3 Exponent: 2

$5.29*10^{4} = 52900$
$2.3*10^{-2} = .023$ (da 10 hoch -2 = 1/(10 hoch 2) = 1/100 = .001)
$-68.94*10^{-3} = -.06894$
$3.14*10^{0} = 3.14$

Ist der Exponent n positiv, so wird in der Mantisse der Dezimalpunkt um n Stellen nach rechts versetzt, ist n negativ, so wird der Dezimalpunkt in der Mantisse um n Stellen nach links versetzt.

Es ist besonders zu beachten, daß somit für die gleiche Festkommazahl mehrere Gleitkommadarstellungen möglich sind.

Beispiel:

$6.34 = .634*10^{+1} = 634.0*10^{-2} = .00634*10^{+3}$ etc.

2.5.1.1 Der Typ "Integer"

Numerische Daten vom Typ Integer sind immer ganzzahlig und haben einen Wertebereich von

$$-32768 \leq \text{Integer} \leq 32767$$

Ihre interne Darstellung erfolgt in zwei Bytes. Zur Unterscheidung von anderen Typen werden Integer-Daten mit dem Zeichen **%** am Ende gekennzeichnet.

Beispiel: **-12%**
VARIABLE%
sind numerische Daten vom Typ Integer.

Anmerkung:
Integer-Daten sparen sowohl Speicherplatz als auch Rechenzeit.
Ein Integer-Wert belegt genau zwei Bytes.

2.5.1.2 Der Typ "Single Precision"

Ein numerischer Wert in Single Precision ("einfacher Genauigkeit") wird bestimmt durch:

- maximal 7 signifikante Ziffern
- einen Exponenten, der mit **E** angegeben wird
- Konstanten oder Variablen, die am Ende mit **!** (Rufzeichen) gekennzeichnet sind.

Der Bestandteil **E** steht für "*10 hoch" und kann auch bei der Eingabe bzw. Ausgabe eines Single-Precision-Wertes verwendet werden.

Der Wertebereich für numerische Daten in einfacher Genauigkeit ist:

$$-3.40282E38 \leq X \leq 3.40282E38$$

Achtung: Eingaben, die diese Grenzen überschreiten, führen zum Overflow. Es wird aber - nach Fehlermeldung - mit dem niedrigsten bzw. höchsten intern darstellbaren Wert sofort weitergearbeitet.

Numerische Daten, für die kein eigener Typ angegeben wird, werden als Single-Precision interpretiert.
Ein Single-Precision-Wert belegt genau 4 Bytes.

2.5.1.3 Der Typ Double Precision ("doppelte Genauigkeit")

Ein numerischer Wert hat doppelte Genauigkeit, wenn:

- er mehr als 7 signifikante Ziffern hat oder
- der Exponent mit **D** dargestellt wird oder
- am Ende ein **#**-Zeichen als Kennzeichen verwendet wird.

Der Bestandteil **D** steht für "*10 hoch" und kann auch bei der Eingabe bzw. Ausgabe eines num. Wertes verwendet werden. Es sind maximal 16 signifikante Ziffern möglich.

Der Wertebereich für numerische Daten in doppelter Genauigkeit ist:

$$-1.79769313486231D308 \leq X \leq 1.79769313486231D308$$

Es sind zwar Eingaben im Intervall

$$-3.5953862697246D308 \leq X \leq 3.5953862697246D308$$

möglich; intern wird jedoch mit den obigen Werten weitergearbeitet; der Rest ist also Signifikanzverlust.
Ein Double-Precision-Wert belegt genau 8 Bytes.

Für Overflows gilt die Anmerkung am Ende von 2.5.1.2

2.5.1.4 Typenumwandlung

Falls erforderlich, werden von BASIC Typenumwandlungen automatisch durchgeführt. Dabei bestehen folgende Möglichkeiten:

1. Falls eine numerische Konstante eines Types einer numerischen Variablen eines anderen Types zugewiesen wird, erfolgt die Zuweisung entsprechend dem Typ der Variablen.
   Beispiel:
   ```
   10 A%=23.42
   20 PRINT A%
   RUN
    23
   ```
2. Bei der Ausführung von Rechenoperationen mit Werten unterschiedlichen Typs erfolgt eine Umwandlung in den Typ der höchsten vorkommenden Genauigkeit. Das Ergebnis hat ebenfalls diese Genauigkeit.
   Beispiel:
   1)
   ```
   10 D# =6# /7%
   20 PRINT D#
   RUN
    .8571428571428571
   ```
   2)
   ```
   10 D=6# /7%
   20 PRINT D
   RUN
    .857143
   ```

   Im Beispiel 1) wurde die Division in doppelter Genauigkeit ausgeführt und der doppelt genauen Variablen D# zugewiesen.
   Im Beispiel 2) wurde die Division ebenfalls in doppelter Genauigkeit ausgeführt, die Zuweisung an die einfach genaue Variable D hat jedoch eine neuerliche Typenumwandlung in einfache Genauigkeit bewirkt.
3. Logische Operatoren (siehe Abschnitt 2.6.3) erfordern Werte vom Typ Integer. Bei der Umwandlung wird nur der ganzzahlige Anteil des Wertes berücksichtigt.
4. Wird eine Gleitkommazahl (Single- oder Double-Precision) in einen Integer-Wert umgewandelt, so wird der Wert auf Ganzzahligkeit kaufmännisch gerundet.
   ```
   10 C%=55.88
   20 PRINT C%
   RUN
    56
   ```

5. Wird einer doppelt genauen Variablen ein Wert in einfacher Genauigkeit zugewiesen, so sind nur die ersten sieben Ziffern signifikant. Die nachfolgenden Stellen sind nicht definiert. Die Differenz zwischen dem ursprünglichen Wert in einfacher Genauigkeit und dem Inhalt der doppelt genauen Variablen ist in jedem Fall kleiner als 6.3E-8 mal dem ursprünglichen Wert.
   Beispiel:

```
10 A=2.04
20 B# =A
30 PRINT A;B#
RUN
 2.04   2.039999961853027
```

6. Werden doppelt genaue Werte außerhalb des für einfach genaue Werte zulässigen Intervalls (vgl. Kapitel 2.5.1.2) in einfach genaue Werte verwandelt, sind zwei Fälle zu unterscheiden:

   a) Liegt der zu konvertierende doppelt genaue Wert außerhalb des Intervalls $-1.80925139D75 \leq x \leq 1.80925139D75$
   wird die Meldung "Overflow" gegeben. Es wird jedoch mit den verschiedensten völlig falschen Werten weitergerechnet!!!
   b) Liegt der zu konvertierende doppelt genaue Wert zwar unterhalb des Intervalls zu a), aber über dem Intervall $-3.40282D38 \leq x \leq 3.40282D38$,
   erfolgt keine Overflow-Meldung. Intern wird mit dem niedrigsten bzw. höchsten in einfacher Genauigkeit intern darstellbaren Wert (vgl. Intervall) weitergerechnet!

Anmerkungen:

- Werden numerische Konstanten bei der Eingabe eines Programmes am Ende mit **%** definiert, so wird dieses Kennzeichen beim Programm-Listing weggelassen.
- Ein Underflow existiert nicht; für extrem nahe an Ø liegende Daten wird genau Ø angesetzt und weitergearbeitet.

2.5.2 Strings

Strings sind eine beliebige Folge von Zeichen.
Die Anzahl der Zeichen, die eine Zeichenfolge bilden, heißt Länge des Strings. Die maximale Länge von Strings ist 255 Zeichen.
Ein String, der aus keinem Zeichen besteht, heißt "Leerstring". Der Leerstring hat die Länge Ø. Der String, der nach einmaliger Betätigung der Leertaste entsteht, heißt Blank. Das Blank ist ein Zeichen wie alle anderen auch. Es hat den ISO-Code 32. Gelegentlich wird es in diesem Handbuch durch das Zeichen ♭ (anstelle der Leertaste) dargestellt.

Die Speicherung von Strings erfolgt entsprechend ihrer Länge. Ein String braucht daher umsomehr Speicherplatz, je mehr Zeichen er enthält. Bei der Ausführung von Programmen wird der Speicherplatz für Strings entsprechend ihrer aktuellen Länge dynamisch verwaltet. (Der Rechner belegt intern nur soviel Platz, wie für den String benötigt wird.)

2.5.3 Konstanten und Variablen

Sowohl numerische Werte als auch Strings können in Programmen sowohl als Konstanten als auch als Variablen vorkommen.

2.5.3.1 Konstanten

Numerische Konstanten werden in Programmen mit ihrem Wert dargestellt. Sie haben ohne explizite Angabe eines Typkennzeichens einfache Genauigkeit, wenn die Konstante weniger als 7 Ziffern lang ist oder der Exponent mit **E** angegeben wurde. Sie haben doppelte Genauigkeit, wenn sie mehr als 7 Ziffern haben oder der Exponent mit **D** bezeichnet wurde oder das Typkennzeichen **#** angehängt ist. Sie sind vom Typ Integer, wenn das Typkennzeichen **%** angehängt ist.

Numerische Konstanten können auch oktal oder hexadezimal angegeben werden.
Hexadezimale Konstanten sind mit dem Präfix **&H** anzugeben.
Beispiele für hexadezimale Konstanten sind:

&H76	(Dezimal 118)
&H3F2	(Dezimal 1010)

Oktale Konstanten werden mit dem Präfix **&O** angegeben.
Beispiele für oktale Konstanten sind:

&O347	(Dezimal 231)
&O12	(Dezimal 10)

Die Typumwandlung von Konstanten in die internen Datenformate wird automatisch durchgeführt.

Stringkonstanten sind Zeichenfolgen, die im Programm durch Anführungszeichen am Anfang und am Ende des Strings gekennzeichnet werden.
Beispiel:

Numerische Konstanten sind:	Stringkonstanten sind:
12!	"15.10.1982"
157.25	"MAX"
-3.15**#**	"Basic-Zeile"
10E57	"125.-"
3.141592765	
1**D**-15	

2.5.3.2 Variablen

Daten, die im Ablauf des Programms veränderbar sind, heißen variable Daten. Sie werden in Variablen gespeichert.
Eine Variable ist ein Speicherplatz, der durch einen Namen identifiziert wird.

Eine Variable wird charakterisiert durch:

- ihren Namen
- ihren Typ
- ihren aktuellen Inhalt.

Variablennamen:
Der Name einer Variablen ist eine Folge von maximal 40 Zeichen, wobei das erste Zeichen ein Buchstabe sein muß.

Folgende Zeichen sind als eventuelle weitere Zeichen (bis zu 38) zulässig:
- Buchstaben (A bis Z) und Ziffern (Ø bis 9) (Kleinbuchstaben können zwar eingegeben werden. Sie werden vom M 2Ø automatisch als Großbuchstaben interpretiert und bei Listings in solche verwandelt.)
- Punkt (.)
- Das Typkennzeichen **!**, **%**, **#** , **$** kann an der letzten Stelle, gegebenenfalls auch als 4Ø. Zeichen, eingesetzt werden.

Unzulässig ist die Verwendung von reservierten Wörtern als Variablennamen

Reservierte Wörter sind:
- die Schlüsselwörter der Befehle und Anweisungen (hier: stets **fett** gedruckt)
- die Namen der Standardfunktionen (hier: stets **fett** gedruckt)
- **DATE$**, **TIME$** als spezielle Variablen zur Steuerung und Abfrage des Datums und der Uhr
- **ERR** und **ERL** für Fehlerroutinen als reservierte Variablen
- **NAME$** als vom System in einem Assembler-Makro benötigte Variable (vgl. **DRAW**)
- Ebenfalls vom System benötigt und deshalb gesperrt sind die Variablennamen **USR**, **PEEK** und **POKE**
- Variablennamen dürfen nicht mit der Buchstabenfolge **FN** beginnen, da diese für Aufrufe von selbstdefinierten Funktionen reserviert ist.
- Wird mit indexsequentiellen Files unter Zuhilfenahme von ISAM-Assembler-Programmroutinen gearbeitet, sind im betreffenden BASIC-Programm die Arrays ISAM$(2Ø) und ISAM%(2Ø) zu dimensionieren und nur für bestimmte Arbeiten im Zusammenhang mit der Verwaltung der indexsequentiellen Files zu verwenden.
- Wird mit der "normierten Programmierung" der DEUTSCHEN OLIVETTI gearbeitet, sind die Anfangsbuchstaben O,P und Q für Variablennamen reserviert, da die in den Modulen verwendeten Variablen stets mit O,P oder Q beginnen. (Es ist standardmäßig ein **DEFINT** O und ein **DEFSNG** P und ein **DEFDBL** Q erfolgt; von dieser Vereinbarung abweichende gesperrte Variablennamen sind durch Typkennzeichen gekennzeichnet.)
- Eine Zusammenstellung aller gesperrten Variablennamen findet sich im Kapitel 4.2.2.

Die Typvereinbarung von Variablen:
Variablen können auf folgende Arten in ihrem Typ festgelegt werden:
1. Durch Angabe des Typkennzeichens am Ende und als Bestandteil des Namens
   % legt eine Variable als Integer-Variable fest
   ! legt eine Variable in einfacher Genauigkeit fest
   # legt eine Variable in doppelter Genauigkeit fest
   $ legt eine Stringvariable fest

2. Implizit werden numerische Variablen in einfacher Genauigkeit festgelegt, falls keine Typvereinbarung erfolgte.

3. Durch Verwendung der Anweisungen **DEFINT**, **DEFSNG**, **DEFDBL**, **DEFSTR**.
   Es kann durch diese Anweisungen für Gruppen von Anfangsbuchstaben von Variablennamen ein bestimmter Typ festgelegt werden.
   Beispiel:
   10 **DEFINT** I-N
   bewirkt, daß alle Variablen, deren Namen den Anfangsbuchstaben I bis N haben, als Integer-Variable vereinbart sind.

Die Vereinbarung von Variablen

Variablen werden durch ihr erstes Auftreten im Programm vereinbart. Dabei wird numerischen Variablen der Wert Ø, Stringvariablen der Leerstring zugewiesen.
Bei der Verbindung mehrerer Programme können die Variablen des vorangehenden Programms mit ihren aktuellen Werten in das nachfolgende Programm übernommen werden.

2.5.3.3 Arrays

Arrays sind eine Form von Variablen, bei der einem Namen mehrere Werte zugeordnet sind. Zur Identifikation jedes einzelnen Wertes müssen zusätzlich zum Namen noch ein oder mehrere Indices angegeben werden. Die Indices werden in Klammern eingeschlossen und durch Komma getrennt. Für die Namen von Arrays gelten dieselben Vorschriften wie in Abschnitt 2.5.3.2 beschrieben. Arrays können in allen zulässigen Variablentypen gebildet werden. Die Dimension und die Anzahl der maximalen Werte der Indices jeder Dimension wird für jeden Array in einer **DIM**-Anweisung festgelegt.

Beispiele für zulässige Variablennamen:

A!	Num. Variable in einfacher Genauigkeit
PI#	Num. Variable in doppelter Genauigkeit
MAXIMUM%	Integervariable
BEZEICHNUNG$	Stringvariable
KUNDEN.NAME$	Stringvariable
KUNDEN.ADRESSE$	Stringvariable
MONAT%(3)	Bezeichnet das dritte Element des Arrays MONAT%(), in dem jedes Array-Element einen Integer-Wert annehmen kann
KOEFFIZIENTENMATRIX(ZEILE%,SPALTE%)	Bezeichnet das Element in Zeile 'ZEILE%' und Spalte 'SPALTE%' des zweidimensionalen Arrays KOEFFIZIENTENMATRIX(), in dem jedes Array-Element einen einfach genauen Wert annehmen kann

2.6 OPERATOREN

Operatoren werden zur Verknüpfung von Daten verwendet. Sie werden nach den Ergebnissen, die sie liefern, klassifiziert.

2.6.1 Arithmetische Operatoren

Mit arithmetischen Operatoren können numerische Daten verknüpft und dadurch Berechnungsvorschriften festgelegt werden.

Arithmetische Operatoren werden mit unterschiedlicher Priorität ausgeführt, wenn sie gemeinsam verwendet werden.

Es stehen folgende arithmetische Operatoren in absteigender Priorität zur Verfügung:

Operator	Operation	Beispiel
^	Potenzierung	X ^ Y
-	Negation	-X
*, /	Multiplikation, Division	X * Y, X / Y
\	Integerdivision	X \ Y (Das Zeichen \ (backslash) wird im deutschen Zeichensatz durch den Großbuchstaben Ö dargestellt.)
MOD	Rest einer Integerdivision (Modulo)	X MOD Y
+, -	Addition, Subtraktion	X+Y, X-Y

Regeln für arithmetische Operatoren

- Potenzierung:

B ^ EXPO — Die Basis B wird zur Potenz EXPO erhoben.

Sonderfälle:

Basis B	Exponent EXPO	Ergebnis B ^ EXPO
Ø	Ø	1
Ø	< Ø	"Division by zero"
Ø	> Ø	Ø
< Ø	≠INT(EXPO)	"Illegal function call"
≠ Ø	Ø	1

- Multiplikation und Addition:

A*B	A wird mit B multipliziert
A+B	B wird zu A addiert

Regeln:
Es gilt das kommutative Gesetz:
A*B = B*A
A+B = B+A
Das transitive Gesetz gilt nicht uneingeschränkt. Durch die Reihenfolge der Operationen können minimale Abweichungen entstehen:
A*(B*C) ≈ (A*B)*C
A+(B+C) ≈ (A+B)+C

Es entstehen nämlich gelegentlich Signifikanz-Unterschiede bei der internen Abspeicherung von Zwischenergebnissen.

- Division und Subtraktion:

A/B	A wird durch B dividiert
A-B	B wird von A subtrahiert

Die Division durch Null führt zur Fehlermeldung "Division by zero".

Erlaubte Vorzeichenkombinationen sind zum Beispiel:
-B+(-A)+C-(-Z)
A+-B
A*-B
A++B

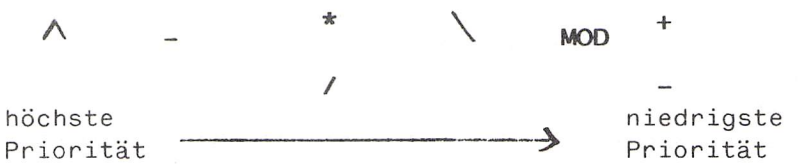
- Integer-Division
  Die Operation ist durch das Sprachelement \ (backslash) gekennzeichnet und liefert den ganzzahligen Teil einer Division. (Das Zeichen \ wird an der deutschen Tastatur des M 20 durch den Großbuchstaben Ö dargestellt.) **Die Operanden werden auf Ganzzahligkeit kaufmännisch gerundet** und müssen im Bereich von $-32768 \leq X \leq 32767$ sein.
  Das Ergebnis wird auf seinen ganzzahligen Teil reduziert.

  Beispiele:
  10\4 liefert den Wert 2
  25.68\6.99 liefert den Wert 3, denn 26 dividiert durch 7 ist 3 Rest 5
  20.51\7.49 liefert den Wert 3, denn 21 dividiert durch 7 ist 3 Rest 0.

- Modulo-Berechnung
  Die Operation ist durch das Sprachelement **MOD** gekennzeichnet und liefert den Divisionsrest einer ganzzahligen Division. Die Operanden müssen ebenfalls im Intervall $-32768 \leq x \leq 32767$ liegen.

  Beispiele:
  11 **MOD** 4 liefert den Wert 3 (11\4 hat das ganzzahlige Ergebnis 2 mit Divisionsrest 3)
  -25.68 **MOD** 6.99 liefert den Wert -5 (-26\7 hat das ganzzahlige Ergebnis -3 mit Rest -5)

- Prioritätsregeln:
  Die numerischen Operatoren haben in nachstehender Reihenfolge Priorität:



Die Operatoren * und / sowie + und - haben jeweils die gleiche Priorität. Operatoren gleicher Priorität werden von links nach rechts verarbeitet, es sei denn, daß durch Klammern etwas anderes definiert wird; Klammern durchbrechen die Prioritätsregeln.

Beispiele: -3^2 liefert -9; bzw. (-3)^2 liefert 9
8/4*2 liefert 4; bzw. 8/(4*2) liefert 1

- Overflow und Division durch Null

  Bei Division durch Ø oder einer Potenzierung von Ø mit negativem Exponten wird die Meldung "Division by zero" ausgegeben. Als Ergebnis wird die größtmögliche darstellbare Zahl mit richtigem Vorzeichen gesetzt und die Ausführung **fortgesetzt.**
  Falls ein Overflow auftritt, wird ebenfalls die größtmögliche darstellbare Zahl mit richtigem Vorzeichen als Ergebnis angenommen. Es wird die Meldung "Overflow" angezeigt und die Ausführung **fortgesetzt.**

2.6.2 Vergleichsoperatoren

Vergleichsoperatoren erlauben den Vergleich von 2 numerischen Werten oder 2 Strings.
Das Ergebnis kann "wahr" oder "falsch" sein. Vergleiche werden zur Steuerung des Programmablaufes verwendet.

Operator	Prüfung	Beispiel
=	Gleichheit	X = Y
<>	Ungleichheit	X<>Y
<	kleiner als	X < Y
>	größer als	X > Y
<=	kleiner oder gleich	X <=Y
>=	größer oder gleich	X >=Y

Der Vergleich von numerischen Werten erfolgt entsprechend ihrem Wert.
Beim Vergleich von Strings wird zeichenweise von links beginnend der Vergleich durchgeführt.
2 Strings sind gleich, wenn sie in allen ihren Zeichen gleich sind.
String 1 ist kleiner als String 2, wenn das erste verschiedene Zeichen in String 1 einen niedrigeren ASCII-Code hat als das entsprechende Zeichen in String 2.
Sind zwei Strings bis zum Ende des einen Strings gleich, ein String aber länger, so ist der längere String der größere.

2.6.3 Logische Operatoren

Logische Operatoren erlauben die Verknüpfung von mehreren Vergleichen oder die Verknüpfung auf Bit-Ebene mit Boole'schen Operationen. Sie liefern eine Bitfolge als Ergebnis, wo jedes Bit entweder den Wert "wahr" oder "falsch" darstellt.
Logische Operatoren haben unterschiedliche Prioritäten. Die nachfolgende Liste führt die logischen Operatoren in absteigender Priorität auf.

Logische Operatoren in absteigender Priorität, dargestellt in Wahrheitstafeln (1 = wahr; Ø = falsch)

NOT

X	NOT X
1	Ø
Ø	1

NEGATION (KOMPLEMENT)

AND

X	Y	X AND Y
1	1	1
1	Ø	Ø
Ø	1	Ø
Ø	Ø	Ø

KONJUNKTION (logisches Produkt)

OR

X	Y	X OR Y
1	1	1
1	Ø	1
Ø	1	1
Ø	Ø	Ø

DISJUNKTION, IMPLIZITES ODER (logische Summe)

XOR

X	Y	X XOR Y
1	1	Ø
1	Ø	1
Ø	1	1
Ø	Ø	Ø

EXPLIZITES ODER (symmetrische Differenz)

IMP

X	Y	X IMP Y
1	1	1
1	Ø	Ø
Ø	1	1
Ø	Ø	1

IMPLIKATION (SUBJUNKTION)

EQV

X	Y	X EQV Y
1	1	1
1	Ø	Ø
Ø	1	Ø
Ø	Ø	1

ÄQUIVALENZ (BIJUNKTION)

- Ausführung von Vergleichen

Für die Ausführung von Entscheidungen können mehrere Vergleiche erforderlich sein.
Die Verknüpfungen können folgendermaßen anschaulich gemacht werden:

AND	SOWOHL Bedingung 1 ALS AUCH Bedingung 2
OR	ENTWEDER Bedingung 1 ODER Bedingung 2 (oder beide)
XOR	GENAU Bedingung 1 ODER GENAU Bedingung 2 (also nicht beide)
IMP	WENN Bedingung 1 DANN AUCH Bedingung 2 (od. nur Bed. 2)
EQV	Bedingung 1 MUSS SEIN WIE Bedingung 2

Beispiele:	Der Ausdruck
A > B **AND** A > C	ist genau dann wahr, wenn sowohl A größer als B ist als auch A größer als C ist.
A > B **OR** A > C	ist dann wahr, wenn entweder nur A größer als B ist oder nur A größer als C ist oder wenn sowohl A größer als B und A größer als C ist.
A > B **XOR** A > C	ist genau dann wahr, wenn nur A größer als B ist oder aber wenn nur A größer als C ist. Er ist falsch, wenn beide Bedingungen erfüllt oder nicht erfüllt sind.
A > B **IMP** C > D	Falls A größer als B ist, muß auch C größer als D sein. Die Abfrage ist aber auch dann wahr, wenn nur C größer als D ist.
A > B **EQV** C > D	ist genau dann wahr, wenn entweder A größer als B ist und C größer als D ist, oder wenn A <= B und C <= D ist.

- Ausführung von logischen Operationen

Die Operanden von logischen Operatoren müssen numerisch sein und werden stets unter kaufmännischer Rundung in den Typ Integer umgewandelt.
Das Bitmuster der zwei Bytes langen internen Darstellung des Integerwertes der beiden Operanden wird bitweise entsprechend der Regeln für die Operation verknüpft. Das Ergebnis ist der dem Bitmuster entsprechende numerische Wert.

Beispiele:

63 **AND** 16

$63_{(10)}$ entspricht: $0000000000111111_{(2)}$
$16_{(10)}$ entspricht: $0000000000010000_{(2)}$

63 **AND** 16 : $0000000000010000_{(2)}$ daher 63 **AND** 16 liefert $16_{(10)}$

4 **OR** 2

$4_{(10)}$: $0000000000000100_{(2)}$
$2_{(10)}$: $0000000000000010_{(2)}$

4 **OR** 2: 0000000000000110 daher 4 **OR** 2 liefert $6_{(10)}$

-1 **OR** -2

$-1_{(10)}$: $1111111111111111_{(2)}$
$-2_{(10)}$: $1111111111111110_{(2)}$

-1 **OR** -2: $1111111111111111_{(2)}$ daher -1 **OR** - 2 liefert $-1_{(10)}$

NOT X ist das Komplement von X. Mathematisch ist das -(X+1)

$5_{(10)}$: $0000000000000101_{(2)}$

Komplement: $1111111111111010_{(2)}$ daher **NOT** 5 liefert $-6_{(10)}$

2.6.4 Stringoperatoren

Als einzige zulässige Operation auf Strings ist die Verkettung von Strings zulässig.
Diese Operation wird mit dem Zeichen + dargestellt.

Beispiel:

```
10 VORNAME$= "MAXL"
20  ZUNAME$= "HUBER"
30 GESNAME$= VORNAME$ + " " + ZUNAME$
40    PRINT GESNAME$
RUN
MAXL HUBER
```

In diesem Beispiel müssen zwei Verkettungen durchgeführt werden, um das Blank zwischen Vornamen und Zunamen einzufügen.

Anmerkungen:
Für die Verarbeitung von Strings stehen eine Reihe von Anweisungen und Funktionen zur Verfügung, die sehr komplexe Operationen auf Strings erlauben.
Die Vergleichsoperatoren sind, wie in Abschnitt 2.6.2 beschrieben, ebenfalls auf Strings anwendbar.

2.7 AUSDRÜCKE

Jede Operation ist eine Verbindung aus Operator/en und Operand/en und liefert ein Datenelement (Datum) eines bestimmten Types.

Da innerhalb von vielen Anweisungen nur die Verwendung eines bestimmten Datentyps vorgeschrieben ist, nicht aber seine Darstellungsform, können an solchen Stellen verwendet werden:

- Konstanten
- Variablen
- Die Verbindung von Konstanten und Variablen durch Operationen
- Standard- oder definierte Funktionen.

2.7.1 Numerische Ausdrücke

Numerische Ausdrücke sind numerische Konstanten oder numerische Variablen oder das Ergebnis von numerischen Funktionen oder Vergleichsausdrücke bzw. logische Ausdrücke (vgl. Kapitel 2.7.3) oder die Verknüpfung solcher numerischer Werte durch arithmetische Operationen.

Prioritätsregeln können durch die Verwendung von Klammern verändert werden.

Beispiele:

Numerische Ausdrücke sind:

```
-1
LAENGE# + BREITE#
SQR(UMFANG)
HOEHE!
(3*A+C*D)/(SIN(A%*B%)^SQR( A%)-D)
RADIUS^2*3.141592
```

Anmerkung: Bei der Berechnung numerischer Ausdrücke von numerischen Werten verschiedenen Typs sind Typenumwandlungen erforderlich.
Dabei können erhebliche Signifikanzverluste auftreten.
Es werden alle Operanden in die höchste vorkommende Genauigkeitsstufe umgewandelt. Dadurch werden in der Operation auch nicht-signifikante Ziffern verarbeitet, die das Ergebnis verfälschen können.

Beispiel: Um den Inhalt der Variablen A! (Typ: einfache Genauigkeit) mit dem konstanten Wert 1.Ø auf Gleichheit zu testen, ist folgende Formulierung nötig:

```
IF  ABS(A!-1.Ø)<1.ØE-6 THEN.......
```

2.7.2 Stringausdrücke

Für Stringausdrücke können verwendet werden:

- Stringkonstanten
- Stringvariablen
- Standard- oder definierte Funktionen, die als Ergebnis einen String liefern
- Die Verbindung von solchen Elementen derart, daß als Ergebnis ein String entsteht.

Beispiele:

Stringausdrücke sind:

```
"MAXL"
FRAGE$
""            (der Leerstring)
ZEICHENSATZ$+CHR$(I%)
VORNAME$+" "+ZUNAME$
```

2.7.3 Vergleichsausdrücke und logische Ausdrücke

Vergleichsausdrücke sind:

a) Numerischer Ausdruck Vergleichsoperator numerischer Ausdruck

b) Stringausdruck Vergleichsoperator Stringausdruck ,

wobei Vergleichsoperatoren die in Abschnitt 2.6.2 definierten Operatoren sind.

Vergleichsausdrücke liefern als Ergebnis den Wahrheitswert -1 (="wahr"), wenn die Vergleichsbedingung erfüllt ist. Sie liefern den Wahrheitswert Ø (="falsch"), wenn die Vergleichsbedingung nicht erfüllt ist. Dieser Wahrheitswert wird als Integer dargestellt und kann somit als numerischer Wert behandelt werden.

Beispiele:

```
VORNAME$=""
ZUNAME$="HANS"
R$< CHR$(I%)
MID$(ADRESSE$,I%,L!)> =STRASSE$+NUMMER$
```

Das Gleichheitssymbol = steht hier nicht für Zuweisung, sondern bedeutet: Führe einen Vergleich durch und errechne den Wahrheitswert.

Solche Vergleiche (bzw. deren Wahrheitswert) können, ähnlich wie Funktionen, nicht allein stehen. Es muß in einer Anweisung festgelegt werden, was mit dem Ergebnis (dem Wahrheitswert der Vergleichsoperation) geschehen soll.

Vergleichsausdrücke können zu logischen Ausdrücken verbunden werden.

Logische Ausdrücke sind Vergleichsausdrücke oder ihre Verknüpfung durch logische Operatoren (siehe Abschnitt 2.6.3). Da logische Operatoren Prioritätsregeln genügen, ist die Verwendung von Klammern zur Änderung der Reihenfolge zulässig.

Beispiele:

```
MONAT$="DEZEMBER" AND TAG%=24
A=1Ø AND(B< D  OR B> E AND B = Ø)
```

Solche Ausdrücke können wiederum nur in Anweisungen eingekleidet vorkommen.

Die zweite Form von logischen Ausdrücken wird durch die Verbindung mit Operationen der logischen Operatoren auf Bitebene erreicht.

Beispiele:

```
A% AND B% OR C%
(A< B) OR NOT X
```

Logische Ausdrücke liefern als Ergebnis je Bit immer einen Integer-Wert.

Die Kombinationen solcher Werte können zur Steuerung von Abläufen im Programm verwendet werden. Wegen der Darstellung als Integer ergeben sich durch solche Operation auf Bitebene numerische Werte, die somit auch in numerischen Ausdrücken verwendet werden können.

Anmerkung: Treten numerische Werte in logischen Ausdrücken außerhalb von Vergleichen (also z.B. alleinstehend) auf, so sind sie genau dann "wahr", wenn sie ungleich Ø sind und genau dann "falsch", wenn sie gleich Ø sind.

2.8 DATEN IN EINEM PROGRAMM

Variablen, Konstanten und Ausdrücke können in einem Programm nach den Erfordernissen der Aufgabenstellung verwendet werden.

Für die Verwendung von Variablennamen gilt:

1.) Sie dürfen von 1 bis 39 Zeichen lang sein (genauere Vorschriften vgl. Abschnitt 4.1.1) und müssen durch Blanks von Schlüsselwörtern getrennt sein.

2.) Das letzte Zeichen des Namens legt den Typ der Variablen fest (sofern nicht **DEFINT**, **DEFSNG**, **DEFDBL** oder **DEFSTR** verwendet wurde). Es ist daher z.B. A% verschieden von A! und von A# . A! und A (z.B.) ist jedoch die gleiche Variable, es sei denn, es wurde eine andere Typvereinbarung für A getroffen als **DEFSNG**.

3.) Einige Variablennamen haben im System eine spezielle Bedeutung. Diese Variablennamen können im Programm unter Berücksichtigung ihrer Wirkung auf das System verwendet werden wie vom Anwender vereinbarte Variable. Das sind die Variablen **DATE$** und **TIME$** zum Setzen und Abfragen von Datum und Uhrzeit und **ERR** bzw. **ERL** für die Verwendung in Fehlerbehandlungsroutinen sowie die Variable **NAME$**, die das System selbst benötigt (vgl. **DRAW**-Anweisung). Variablennamen dürfen nicht mit der Zeichenfolge **FN** beginnen. Für weitere Einschränkungen (z.B. bei Einsatz von indexsequentiellen Files, die mit ISAM-Assembler-Routinen verarbeitet werden oder bei normierter Programmierung nach den Richtlinien der DEUTSCHEN OLIVETTI) vgl. Kapitel 2.5.3.2.

4.) Schlüsselwörter (in diesem Handbuch **fett** gedruckt) dürfen nicht als Variablennamen verwendet werden. Sie können aber Bestandteil von Variablennamen sein.

Für die Verwendung von Arrays gilt:

Arrays sollten vor ihrer Verwendung in einer **DIM**-Anweisung vereinbart werden.

Unterbleibt diese Vereinbarung, so wird bei jedem Index als Default-Wert für den höchstmöglichen Index 1Ø gesetzt.

Durch die Verwendung der **ERASE**-Anweisung können Arrays während der Programmausführung aufgelöst und in der Folge redimensioniert werden.

Werden Variablen in einem Programm erstmals verwendet, so wird numerischen Variablen der Wert Ø bzw. Stringvariablen der Leerstring zugewiesen.

Bei der Verkettung mehrerer Programme besteht die Möglichkeit, durch Angabe des Parameters **ALL** in der Anweisung **CHAIN** alle Variablen des rufenden Programmes mit ihren aktuellen Werten an das gerufene Programm zu übergeben.

Wird **ALL** nicht angegeben, so kann durch eine **COMMON**-Anweisung (in beiden Programmen) festgelegt werden, welche Variable/n mit ihrem Wert übergeben werden soll/en.

2.9 ZUSAMMENFASSUNG DER DEFINITIONEN

Folgende Begriffe wurden im vorangehenden Abschnitt definiert:

Programm:	Folge von mit Zeilennummern versehenen Programmzeilen
Programmzeile:	Zeile mit max. 255 Zeichen, die aus Zeilennr. und Anweisung(en) besteht.
Anweisung:	Ein nach den Regeln der Sprache BASIC gebildeter Bestandteil eines Programmes. Eine Anweisung besteht aus einem oder mehreren Schlüsselworten und Parametern. Mehrere Anweisungen in einer Programmzeile sind mit : zu trennen.
Daten:	Daten können sein: Konstanten oder Variablen bzw. numerisch oder Strings.
num. Konstante:	Zahl, die mit ihrem Wert und gegebenenfalls Typkennzeichen angegeben wird.
Stringkonstante:	String, der, in Anführungszeichen eingeschlossen, im Programm verwendet wird.
Variable:	Variablen sind Speicherplätze, die mit einem Namen identifiziert werden. Variablen haben Namen, die maximal 39 Zeichen lang sind, mit einem Buchstaben beginnen und keine Sonderzeichen (außer dem Punkt) enthalten dürfen. Am Ende (gegebenenfalls auch als 40. Zeichen) darf zusätzlich noch ein Typkennzeichen stehen.
Array:	Arrays sind Variablen, die mehr als einen Wert enthalten können. Sie werden unter einem Namen identifiziert und das gewünschte Element wird durch Indices ausgewählt. Die Anzahl der Indices und die Obergrenzen der einzelnen Indices sind frei wählbar.
arithm. Operatoren:	^,-,*,/,\,**MOD**,+,-
Vergleichs-Operatoren:	=, <> , < , > , <=, >=
logische Operatoren:	**NOT**, **AND**, **OR**, **XOR**, **IMP**, **EQV**
num. Ausdruck:	numerische Konstanten oder numerische Variablen oder Verknüpfung solcher Größen mit arithmetischen oder logischen oder Vergleichsoperatoren oder Funktionen mit der Möglichkeit zur Prioritätsänderung durch Klammern.

Stringausdruck:	Stringkonstanten, Stringvariablen oder ihre Verknüpfung durch + und die Anwendung von Stringfunktionen.
Vergleichsausdruck:	Str.ausdruck Vergleichsoperator Stringausdr. oder num. Ausdr. Vergleichsoperator num. Ausdr..
logischer Ausdruck:	Verknüpfung von Vergleichsausdrücken und/oder numerischen Ausdrücken durch logische Operatoren.
Zeilennummer:	steht am Anfang einer Anweisung(sfolge) und zeigt an, an welcher Stelle im Programm (im Arbeitsspeicher) die Anweisung(sfolge) steht. Die Zeilennummern können zwischen Ø (einschließlich) und 65529 (einschließlich) liegen.

3. ARBEITSWEISE DES INTERPRETERS

Das Betriebssystem des M 2Ø enthält neben seinem Befehlsvorrat zur Ausführung verschiedener Operationen auf Volume- und Fileebene (PCOS-Befehle) einen BASIC-Interpreter zur Verwaltung und Ausführung von Programmen. Dem Anwender stehen somit PCOS-Befehle und - bei aktiviertem BASIC-Interpreter- sowohl BASIC-Befehle als auch BASIC-Anweisungen zur Verfügung.

Der BASIC-Interpreter kann in verschiedenen Betriebszuständen arbeiten, die nachfolgend einzeln beschrieben werden:

- Der Command-Mode
  erlaubt die Eingabe von Programmzeilen und die Eingabe von unmittelbar auszuführenden BASIC-Befehlen und -Anweisungen; im letztgenannten Falle befindet sich das System im Direkt-Mode;

- Der Execute-Mode
  wird bei der Ausführung von Programmen und von direkt auszuführenden Zeilen erreicht;

- Der Edit-Mode
  wird für die Korrektur von Programmzeilen aufgerufen.

3.1 EINGABE VON BASIC-BEFEHLEN

BASIC-Befehle werden hauptsächlich für die Eingabe, Korrektur, Dokumentation und Speicherung von Programmen angeboten.
Der Betriebszustand des Interpreters, der die Eingabe (und unmittelbare Ausführung) von Befehlen zuläßt, heißt Command-Mode bzw. Direkt-Mode.

3.1.1 Übergang in den Command-Mode

Der BASIC-Command-Mode wird erreicht,

- falls sich das Betriebssystem in der PCOS-Ebene befindet und der (PCOS)-Befehl **ba** eingegeben wird;

- falls während der Ausführung eines BASIC-Programmes (oder einer direkt auszuführenden Zeile) C eingegeben wird;

- falls bei der Korrektur von Programmzeilen (Edit-Mode) einer der nachfolgenden Edit-Subbefehle eingegeben wird: **E** (Exit), **Q** (Quit Editing) oder eine Eingabeabschluß-taste (CR).

Der BASIC-Command-Mode wird selbstständig aktiviert, falls die Ausführung eines Programmes oder einer direkt auszuführenden Zeile korrekt beendet wird.

Bei jedem Übergang aus einem beliebigen anderen Betriebszustand in den BASIC-Command-Mode erfolgt die Meldung "ok".

3.1.2 Eingaben im Command-Mode

Der BASIC-Command-Mode erlaubt die Eingabe und Ausführungen von BASIC-Befehlen, direkt auszuführenden Zeilen und von Programmzeilen.

Der Command-Mode und der Direkt-Mode haben keinen Einfluß auf ein im Arbeitsspeicher befindliches Programm. Eingaben von Programmzeilen im Command-Mode löschen jedoch sofort den gesamten Datenbereich im Arbeitsspeicher sowie alle Stack-Speicher.

Jede Eingabe wird im BASIC-Command-Mode erst verarbeitet, wenn sie mit einer Eingabeabschlußtaste (CR) beendet wurde.

Bei jeder Eingabe werden führende Blanks ignoriert. Die Interpretation richtet sich nach dem ersten eingegebenen Zeichen, das von Blank verschieden ist.

- Ist das erste von Blank verschiedene Zeichen eine Ziffer, so wird die Eingabe als Programmzeile behandelt und im Arbeitsspeicher abgelegt. Dabei werden alle Daten im Datenbereich des Arbeitsspeichers sofort gelöscht.

- Ist das erste von Blank verschiedene Zeichen keine Ziffer, so wird die Zeile als direkt auszuführend interpretiert. Dabei werden die Daten im Datenbereich des Arbeitsspeichers nicht gelöscht.

Für die Erstellung eines Programmes werden somit die einzelnen Programmzeilen, mit einer Zeilennummer beginnend, eingegeben. Diese Zeilen werden nach Eingabe von CR übernommen und im Arbeitsspeicher abgelegt. Die weitere Behandlung dieser Programmzeilen wird durch BASIC-Befehle bestimmt. (So kann z.B. durch Eingabe von **RUN** die Ausführung dieser Zeilen verlangt werden.)

Ist einer Eingabezeile keine Zeilennummer vorangestellt, so wird sie unmittelbar nach dem Drücken von CR ausgeführt. Neben den BASIC-Befehlen können auch die meisten Anweisungen direkt ausgeführt werden. Die Unterscheidung wird in Abschnitt 3.2 erläutert.

Korrektur von Eingaben im Command-Mode

Eingabezeilen können vor dem Drücken der Abschlußtaste (CR) korrigiert werden.
Durch (wiederholtes) Drücken von H können die Zeichen von der aktuellen Cursorposition bis zur ersten fehlerhaften Stelle in der Zeile gelöscht werden. Es ist ab dieser Stelle die Eingabe richtig fortzusetzen.

3.1.3 Verlassen des Command-Modes

Der Command-Mode wird in vielen Fällen automatisch durch die Ausführung des gewünschten Befehles verlassen. (z.B. bei Eingabe von **RUN** zur Ausführung eines Programmes).

In folgenden Fällen kann der Command-Mode explizit verlassen werden:

SYSTEM	als Befehl bewirkt den Übergang in die PCOS-Ebene (**SYSTEM** kann auch im Programm verwendet werden);
RESET + Umschalttaste (shift)	löscht den Inhalt des Speichers und bewirkt nachfolgend den Übergang in die PCOS-Ebene;
EDIT	dient zum Aufruf des Edit-Modes zur Korrektur von Programmzeilen (siehe Abschnitt 3.4).

3.2 DIREKT-MODE

Der BASIC-Interpreter kann neben Befehlen auch die meisten Anweisungen direkt ausführen. Umgekehrt können die meisten Befehle auch innerhalb von Programmen vorkommen und sinnvoll angewendet werden.

Befehle, die unmittelbar zum Umgang mit Programmen (z.B. **LIST**, **LLIST**) vorgesehen sind, werden zwar innerhalb von Programmen ausgeführt, bewirken jedoch einen Übergang in den BASIC-Command-Mode bzw. Direkt-Mode.

Kommt daher ein solcher Befehl in einem Programm vor, so wird die Programmausführung nach der Ausführung dieses Befehles abgebrochen.
Andere Befehle, wie z.B. **KILL**, **TRON**, **TROFF**, können sinnvoll innerhalb von Programmen verwendet werden.

BASIC-Anweisungen, die ohne vorangehende Zeilennummer eingegeben werden, werden als direkt auszuführend interpretiert.
Dadurch ist eine sehr komfortable Möglichkeit verfügbar, kleinere Aufgaben direkt zu lösen oder Programme sehr wirkungsvoll zu testen und zu korrigieren.

Einige Gruppen von Anweisungen sind nicht im Direkt-Mode ausführbar und haben entweder keine Wirkung oder führen zu einer Fehlermeldung.
Das sind im wesentlichen:
- einzelne Anweisungen für Files
- **DATA**
- **DEF FN** .

Sinnvoll im Direkt-Mode anwendbar sind jene Anweisungen, die zu direkten Ergebnissen (z.B. von Berechnungen) führen oder die Aufschluß über den Zustand von Variablen eines aktuellen Programmes bieten (z.B. **PRINT**, **LPRINT**, **LET** usw.).

3.3 ERSTELLEN EINES PROGRAMMES

Wie bereits in Kapitel 2 beschrieben, besteht ein Programm aus einer Folge von Programmzeilen, die mit einer Zeilennummer beginnen und eine oder mehrere durch : getrennte Anweisungen enthalten.

Aus Abschnitt 3.1 ist bekannt, daß im Command-Mode des BASIC-Interpreters eingegebene Zeilen, die mit einer Ziffer beginnen, als Programmzeilen interpretiert werden.

Dieser Abschnitt befaßt sich jetzt mit den Bedienungselementen, die zur Eingabe eines neuen Programmes notwendig bzw. möglich sind.

3.3.1 Vorbereiten des Arbeitsspeichers

Um ein neues Programm eingeben zu können, muß der Arbeitsspeicher mit dem Befehl **NEW** gelöscht werden.
Es ist dadurch gewährleistet, daß eventuell im Arbeitsspeicher enthaltene Programmzeilen oder Daten von früheren Arbeiten gelöscht sind.

Nach Ausführung von **NEW** kann unmittelbar mit der Eingabe der ersten Programmzeile begonnen werden.
Zusätzlich bietet der Interpreter die Möglichkeit der selbständigen Vergabe der Zeilennummern mit dem Befehl **AUTO**.

3.3.2 Eingabe der Programmzeilen

Die Programmzeilen werden entsprechend den Regeln von BASIC eingegeben.
Sobald eine der Abschlußtasten CR gedrückt wird, wird die Zeile entsprechend ihrer Zeilennummer den bereits bestehenden Programmzeilen hinzugefügt oder eingereiht.

Wird eine Zeilennummer doppelt vergeben, so wird die bestehende Zeile durch die neue Programmzeile vollständig ersetzt. Erzeugt der Befehl **AUTO** eine bereits bestehende Zeilennummer, so wird nach der Zeilennummer ein * (Sternchen) angezeigt, um den Anwender auf die bereits bestehende Zeile hinzuweisen. Wird unmittelbar CR gedrückt, so bleibt die alte Zeile erhalten; erfolgen Eingaben, so wird die alte Programmzeile wie gewohnt durch die neue Programmzeile ersetzt.

Die Korrektur von Programmzeilen während der Eingabe kann vor dem Drücken der Abschlußtaste CR mit H erfolgen:
Es werden durch Drücken von H alle Zeichen bis zur fehlerhaften Stelle gelöscht und anschließend die Zeile korrekt eingegeben.

Wurde die Abschlußtaste CR bereits gedrückt, so kann die Korrektur auf zwei Arten erfolgen:

- durch neuerliche Eingabe der Zeile mit der gleichen Zeilennummer;

- durch die Möglichkeit des Edit-Modes (siehe Abschnitt 3.4).

3.3.3 Möglichkeiten nach dem Ende der Eingabe

Die Programmzeilen werden während der Eingabe im Arbeitsspeicher abgelegt. Sind alle Programmzeilen eingegeben, so muß durch Befehle bestimmt werden, was mit dem soeben erstellten Programm geschehen soll.

Wurde bei der Eingabe die automatische Vergabe der Zeilennummer durch **AUTO** gewählt, so muß diese Vorgabe der Zeilennummern mit C abgebrochen werden.

Es kann jetzt jede im BASIC-Command-Mode zulässige Eingabe durchgeführt werden.
Es ist jedoch zu empfehlen, das Programm mit dem Befehl **SAVE** zuvor auf eine Diskette zu speichern und mit **LLIST** einen Ausdruck des Programmes zu erstellen.

3.4 KORREKTUR BESTEHENDER PROGRAMME

Zur Korrektur eines bestehenden Programmes muß dieses Programm im Arbeitsspeicher vorhanden sein. Das ist der Fall, wenn das Programm unmittelbar vorher eingegeben oder z.B. mit **RUN** ausgeführt wurde. Ist das Programm noch nicht im Arbeitsspeicher, so muß es mit **LOAD** von der Diskette geladen werden.

Wurden Programme mit **SAVE** und Parameter **P** geschützt gespeichert, so ist eine nachträgliche Korrektur nicht möglich. Alle Befehle, die eine Änderung des bestehenden Programmes oder den Druck von Programmzeilen bewirken, führen zu einer Fehlermeldung.
Das sind folgende Befehle:
LIST, LLIST, DELETE, EDIT, RENUM
sowie das Einfügen von neuen Zeilen.

3.4.1 Einfügen zusätzlicher Programmzeilen:

Zusätzliche Programmzeilen können durch einfache Eingabe dieser Zeilen mit den entsprechenden Zeilennummern eingegeben werden. Diese Zeilen werden entsprechend einsortiert oder angefügt.

3.4.2 Ersetzen von Programmzeilen

Soll eine bestehende Programmzeile zur Gänze ersetzt werden, so ist die neue Programmzeile mit der Zeilennummer der zu ersetzenden Zeile einzugeben.

3.4.3 Löschen von Programmzeilen

Eine oder mehrere Programmzeilen können mit dem Befehl **DELETE** gelöscht werden.

3.4.4 Ändern bestehender Programmzeilen (Edit-Mode)

Für die Korrektur bestehender Programmzeilen stellt der BASIC-Interpreter einen eigenen Betriebszustand, den Edit-Mode, zur Verfügung.

Der Edit-Mode wird auf zwei Arten erreicht:

1) Durch Anwendung des Befehles **EDIT**

2) Falls bei der Ausführung eines Programmes ein Syntax-Fehler erkannt wird, so wird dieser Fehler gemeldet und automatisch der Edit-Mode aufgerufen.

Innerhalb des Edit-Modes stehen eine Reihe von Edit-Subbefehlen zur Verfügung, die nachfolgend tabellarisch angeführt und anhand zweier Beispiele erläutert werden.

Edit-Befehl	Bedeutung
EDIT nnnnn	Es wird die Programmzeile mit der Nummer nnnnn aufgerufen und die Zeilennummer im Bildschirm angezeigt. Der Cursor befindet sich an der ersten Stelle nach der Zeilennummer.
EDIT .	Der Befehl **EDIT** bezieht sich auf die aktuelle Zeilennummer, das ist z.B. die zuletzt eingegebene oder ausgeführte Zeile.

Edit-Subbefehl	Bedeutung
L	Die Zeile wird im Display angezeigt. In der nächsten Zeile erscheint wieder die Zeilennummer und der Cursor an der ersten Position der Zeile.
A	Die laufende Edit-Operation wird unterbrochen und es bleibt die ursprüngliche Zeile erhalten. Die Zeilennummer wird erneut angezeigt und der Cursor befindet sich an der ersten Stelle der Zeile.
Blank	Der Cursor wird um eine Stelle nach rechts bewegt und das entsprechende Zeichen wird angezeigt.
CTRL H	Der Cursor wird um eine Stelle nach links (zurück) bewegt. Das vorherige Zeichen wird jedoch nicht gelöscht!
I	Ab der gegenwärtigen Cursorposition sollen Zeichen eingefügt werden. Nach **I** sind die Zeichen einzugeben und entweder mit CR oder CTRL 5 abzuschließen. Die eingefügten Zeichen werden angezeigt und der Cursor befindet sich an der ersten Stelle nach den eingefügten Zeichen.
X	Erlaubt das Anfügen von Zeichen an das Ende einer bestehenden Zeile. Wird **X** eingegeben, so wird die bestehende Zeile angezeigt, der Cursor ist an der ersten freien Stelle und es können weitere Zeichen eingegeben oder mit CTRL H die letzten Zeichen gelöscht werden.
CTRL HOME CTRL 5	Bewirkt das Ende des Einfügens, es bleibt jedoch der Edit-Mode erhalten. Wird CR gedrückt, so wird zwar ebenfalls das Ein- (bzw.An-)fügen beendet, jedoch auch der Edit-Mode verlassen.

nD löscht ab der Cursorposition die nächsten n Zeichen. Die gelöschten Zeichen werden zwischen "Ö" angezeigt, der Cursor steht an der ersten Stelle nach dem letzten gelöschten Zeichen.
Ist n größer als die Anzahl der rechts vom Cursor stehenden Zeichen, so wird der Rest der Zeile ab der Cursor-Position gelöscht.

H löscht den Rest der Zeile ab der Cursorposition und erlaubt anschließend das Eingeben von Zeichen.

nSZeichen Dieser Subbefehl sucht das n-te Auftreten des nach **S** angegebenen Zeichens.
Das Suchen beginnt an der ersten Position rechts vom Cursor. Wird das Zeichen gefunden, so wird der Cursor unmittelbar vor dem gefundenen Zeichen angezeigt.
Wird das Zeichen nicht gefunden, so steht der Cursor am Ende der Zeile. Alle während des Suchens durchlaufenen Zeichen werden angezeigt.

nKZeichen Dieser Subbefehl ist ähnlich dem Subbefehl **S**, jedoch mit dem Unterschied, daß alle während des Suchens durchlaufenen Zeichen gelöscht werden. Die gelöschten Zeichen werden zwischen "Ö" angezeigt. Der Cursor steht an der Stelle unmittelbar vor dem im Subbefehl angegebenen Zeichen.

nCZeichen Mit **C** kann das nächste Zeichen durch das eingegebene Zeichen ersetzt werden. Wird **nC** gewählt, so können ab der Cursorposition die nächsten n Zeichen durch die eingegebenen Zeichen ersetzt werden. Ist ein Zeichen zu ersetzen, kann n weggelassen werden. Sobald n Zeichen eingegeben wurden, wird automatisch wieder der Edit-Mode angewählt.

CR Das Drücken der Abschlußtaste bewirkt die Anzeige der modifizierten Zeile und die Rückkehr in den Command-Mode.

E Bewirkt ebenfalls die Rückkehr in den Command-Mode, die modifizierte Zeile wird jedoch nicht angezeigt.

Q Die Eingabe von **Q** bewirkt die Rückkehr in den Command-Mode, es bleibt jedoch der ursprüngliche Inhalt der Zeile gespeichert und alle durchgeführten Änderungen bleiben unberücksichtigt! Auch der Datenbereich im Arbeitsspeicher bleibt ungelöscht, wenn keine anderen Edit-Subbefehle vorher eingegeben wurden.

Anmerkungen:

Die Edit-Subbefehle werden am Bildschirm nicht dargestellt.

Im Edit-Mode werden Zeichen unmittelbar übernommen. Es ist daher nicht nötig, eine der Abschlußtasten zu drücken. (Durch Drücken der Abschlußtaste wird der Edit-Mode verlassen.)

Beim Übergang in den Edit-Mode werden alle noch bestehenden Variableninhalte gelöscht. Sollen also im Zuge des Testen eines Programmes noch Variableninhalte geprüft werden (z.B. weil der Edit-Mode automatisch nach einem Syntaxfehler aufgerufen wird), so ist unbedingt der Subbefehl **Q** einzugeben.
Es wird dann der Edit-Mode verlassen und die Inhalte der Variablen können überprüft werden. Jeder andere Subbefehl als **Q** löscht die Inhalte der Variablen!

Mit dem Parameter **P** des BASIC-Befehles **SAVE** geschützte Programme können nicht verändert werden. Die Anwahl des Edit-Modes bewirkt die Meldung "Illegal function call".

Beispiele:

1) Die Zeile

```
500 FOR I=1 TO 15 STEP 2
```

soll korrigiert werden auf:

```
500 FOR I=2 TO 16 STEP 2
```

Schritt	Eingabe	Anzeige
1	**EDIT** 500 CR	500 ■
2	**L**	500 FOR I=1 TO 15 STEP 2
		500 ■
3	Blank 6 mal	500 FOR I=■
4	**C**2	500 FOR I=2■
5	Blank 5 mal	500 FOR I=2 TO 1■
6	**C**6	500 FOR I=2 TO 16■
7	CR	500 FOR I=2 TO 16 STEP 2
		■

2) Die Zeile

```
510 LET A(I)=I*SIN(X)
```

soll geändert werden auf:

```
510 LET A(I)=I*COS(X):PRINT A(I)
```

Schritt	Eingabe	Anzeige
1	**EDIT** 510 CR	510 ■
2	**L**	510 LET A(I)=I*SIN(X)
		510 ■
3	Blank 11 mal	510 LET A(I)=I*■
4	3**C** COS	510 LET A(I)=I*COS■
5	**X**:PRINT A(I) CR	510 LET A(I)=I*COS(X):PRINT A(I)
		■

3.5 AUSFÜHREN VON PROGRAMMEN

3.5.1 Programmstart

Für die Ausführung von Programmen stehen mehrere BASIC-Befehle und -Anweisungen zur Verfügung.

1) Das Programm befindet sich im Arbeitsspeicher.

   Ist das Programm bereits im Arbeitsspeicher, weil es z.B. unmittelbar zuvor eingegeben wurde oder zu Edit-Zwecken geladen war, so wird die Ausführung mit **RUN** gestartet.

   RUN ohne Parameter bewirkt die Ausführung des im Arbeitsspeicher befindlichen Programmes, beginnend bei der niedrigsten Zeilennummer.

   Soll das Programm ab einer bestimmten Zeile ausgeführt werden, so gilt:

   RUN nnnnn startet die Ausführung bei der Zeilennummer nnnnn. Selbstverständlich muß nnnnn so gewählt werden, daß eine sinnvolle Programmausführung ab dieser Stelle möglich ist.

2) Das Programm befindet sich als File auf der Diskette.

   In diesem Fall stehen folgende Möglichkeiten zur Verfügung:

   1) **RUN** Filename
   2) **LOAD** Filename,**R**
   3) **ba** Filename, falls das System in der PCOS-Ebene ist ('Filename' ist hier nicht in Anführungszeichen einzugeben!)
   4) Start des Programmes über INIT.BAS
   5) Aufruf des Programmes durch ein anderes Programm, das die Anweisung **CHAIN** verwendet.

   'Filename' ist dabei ein nach den PCOS-Regeln aufgebauter Filename.

   Beispiele für Programmnamen:
   "Ø:EINGABE"
   "1:PROGRAMM25"
   "DISKA:ENDE/PASSWORD"

zu 1) **RUN** Filename

Das Programm 'Filename' wird auf der angegebenen Diskette gesucht, in den Arbeitsspeicher geladen und mit der Ausführung bei der niedrigsten Zeilennummer begonnen.

Wird zusätzlich der Parameter **R** angegeben (**RUN** Filename,**R**) so bleiben alle Datenfiles offen, die zum Zeitpunkt des **RUN**-Befehls offen sind.

zu 2) **LOAD** Filename,**R**

LOAD mit Parameter **R** ist vollkommen gleichbedeutend mit **RUN**.
Anmerkung: **LOAD** alleine bewirkt das Laden des Programmes ohne Start der Ausführung.

zu 3) **ba** Filename

Befindet sich der M 20 in der PCOS-Ebene, so kann unmittelbar die Ausführung eines Programmes mit **ba** Filename gestartet werden. (Der Filename ist hier nicht in Anführungszeichen eingeschlossen anzugeben, da es sich um einen PCOS-Befehl handelt.) Es wird dann sofort in die BASIC-Ebene gewechselt, das Programm auf Diskette gesucht und mit seiner Ausführung begonnen.

zu 4) Start über INIT.BAS

Der M 20 sucht nach dem Einschalten sofort nach dem Programmnamen INIT.BAS. Ist ein solches Startprogramm vorhanden, so wird ohne weitere Bedienungseingriffe mit der Ausführung des Programmes begonnen.
INIT.BAS kann selbstverständlich auch mit einer der unter 1) - 3) angegebenen Möglichkeiten gestartet werden.

zu 5) Programmaufruf aus anderen Programmen heraus

Ein Programm kann durch Anweisungen selbst andere Programme aufrufen. Es wird dann das Programm im Arbeitsspeicher durch das aufgerufene Programm ersetzt und mit der Ausführung dieses Programmes begonnen (vgl. Anweisung **CHAIN**).

Anmerkung:

Für die erfolgreiche Ausführung eines Programmes müssen bestimmte Voraussetzungen hinsichtlich der Systemvorbereitung (z.B. **sb**, **ss**, **sf** in der PCOS-Ebene) und des Zustands des Arbeitsspeichers gewährleistet sein (vgl. **CLEAR**- **COMMON**- und **CHAIN**-Anweisungen). Führt der Start eines Programmes, dessen Funktionsfähigkeit bereits bekannt ist oder vorausgesetzt werden kann, zu Fehlermeldungen, so ist zu überprüfen, ob die Ursache in solchen fehlerhaften Voraussetzungen liegen kann (z.B. das vorangehende Programm wurde vorzeitig beendet oder es wird die reguläre Ausführung vorangehender Programme verlangt).

3.5.2 Unterbrechen der Programmausführung

Üblicherweise sollte der vorgesehene Programmablauf eingehalten werden.

Besonders zum Testen von neu erstellten Programmen ist jedoch die abschnittsweise Ausführung von Programmen oder die Unterbrechung der Ausführung sinnvoll und notwendig.

Durch Eingabe von

C kann die Programmausführung an der aktuellen Stelle unterbrochen werden. Das System meldet "ok". Die Datenfiles bleiben offen, die Inhalte der Variablen unverändert.

Es können alle Befehle und direkten Anweisungen ausgeführt werden, die Variablen haben ihren aktuellen Wert. Variablenwerte können somit mit **PRINT** Variablenname oder **?** Variablenname abgefragt oder z.B. mit **LET** verändert werden.
Die Ausführung kann aber an der Stelle der Unterbrechung durch Eingabe von **CONT** fortgesetzt werden.

Durch Eingabe von

S wird die Ausführung unterbrochen, es wird jedoch nicht in den Command-Mode verzweigt.
Das Drücken einer beliebigen Taste bewirkt die Fortsetzung der Programmausführung.

3.5.3 Testen und Korrektur von Programmen

Für den Test von Programmen stehen verschiedene Möglichkeiten zur abschnittsweisen Ausführung zur Verfügung.

1) Unterbrechung des Programmlaufs durch den Bediener.

Durch Eingabe von C während der Ausführung kann an der aktuellen Stelle unterbrochen werden.

Es stehen alle Möglichkeiten des Direkt-Modes zur Verfügung.
Die Ausführung kann mit **CONT** fortgesetzt werden.

2) Unterbrechungen aus den Programm heraus

Soll an bestimmten Stellen unterbrochen werden, so sind an diesen Stellen **STOP**-Anweisungen im Programm aufzunehmen. Die Ausführung wird unterbrochen.
Es stehen alle Möglichkeiten des Direkt-Modes zur Verfügung.
Die Ausführung kann mit **CONT** mit der auf **STOP** folgenden Anweisung fortgesetzt werden.

3) Überprüfung des Programmweges

Der aufgrund von Schleifen, Bedingungen oder Kontrollanweisungen ausgeführte Ablauf in der Ausführung kann durch die Anweisungen **TRON** und **TROFF** verfolgt werden. **TRON** und **TROFF** können auch im Programm verwendet werden.

4) Fortsetzung an beliebigen Stellen

Mit **CONT** kann nach C oder **STOP** ab der Stelle der Unterbrechung fortgesetzt werden.
Soll mit einer anderen Anweisung fortgesetzt werden, so kann mit **RUN** Zeilennummer oder mit der direkt auszuführenden Anweisung **GOTO** Zeilennr. angegeben werden, ab welcher Anweisung die Ausführung wieder gestartet wird.

5) Fortsetzung nach Fehlermeldungen

Werden Syntaxfehler gemeldet, so erfolgt automatisch der Übergang in den Edit-Mode.
Der Übergang in den Edit-Mode bewirkt das Löschen aller Variablenwerte. Sollen also nach Auftreten des Syntaxfehlers Variablenwerte überprüft werden, so ist der Edit-Mode mit **Q** (Quit) zu verlassen.
Das Drücken jeder anderen Taste bewirkt das Löschen der Variablen!
Bei der Fortsetzung nach solchen Fehlern ist zu beachten, daß eventuell die Variablenwerte nicht mehr zur Verfügung stehen.

Bei anderen Fehlermeldungen ist nach der jeweiligen Situation zu entscheiden.
Einige Meldungen lassen die Fortsetzung mit **CONT** zu, andere erfordern vor der Fortsetzung die Behebung der Fehlerursache (Siehe Meldungen des Interpreters; Kapitel 7).

3.5.4 Ende einer Programmausführung

Das reguläre Ende einer Programmausführung wird mit Ausführung der Anweisung **END** erreicht.
Es werden alle offenen Files geschlossen und in den BASIC-Command-Mode verzweigt. Es wird die Meldung "ok" angezeigt.

Nach der Ausführung von **END** stehen die Variablenwerte weiterhin zur Verfügung und können im Direkt-Mode verarbeitet werden.

Hinweis: Jedes andere Programmende außer der Ausführung von **END** bewirkt, daß Datenfiles offen bleiben, es sei denn, diese wurden mit einer **CLOSE**-Anweisung geschlossen.

3.5.5 Verbindung mehrerer Programme

Das M 20-BASIC erlaubt die Verbindung von Programmen auf mehrere Arten:

1) Mit **RUN** oder **LOAD**
2) Mit **CHAIN** oder **CHAIN MERGE**

Der Hauptunterschied dieser Möglichkeiten besteht in der Behandlung der zu übergebenden DATEN.

1) **RUN** und **LOAD**

Werden Programme mit **RUN** oder **LOAD** verbunden, so werden
- alle Datenfiles geschlossen (außer, es wurde der Parameter **R** angegeben)
- alle numerischen Variablen mit Ø und alle Stringvariablen mit dem Leerstring ("") vorbelegt.

2) **CHAIN**

Die Anweisung **CHAIN** erlaubt unterschiedliche Möglichkeiten zur Weitergabe von Daten zwischen einzelnen Programmen:

CHAIN läßt die Datenfiles offen;

CHAIN unter Verwendung von einer oder mehreren **COMMON**-Anweisungen erlaubt die Übergabe einzelner Variablen;

CHAIN mit Parameter **ALL** bewirkt, daß alle Variablen des aufrufenden Programmes mit ihren Werten an das aufgerufene Programm übergeben werden.

3) **CHAIN MERGE**

Mit **RUN** bzw. **CHAIN** können mehrere Programme sequentiell verbunden werden.
CHAIN MERGE erlaubt hingegen, daß innerhalb eines Programmes Overlays von Programmteilen gebildet werden.

CHAIN MERGE bewirkt, daß ein als ASCII-File gespeicherter Programmteil mit dem im Arbeitsspeicher befindlichen Programm vermischt wird. Die Ausführung wird bei der Zeile begonnen, bei der das Overlay geladen wurde.

Alle Daten des aufrufenden Programmes bleiben erhalten.

Wird zusätzlich der Parameter **DELETE** angegeben, so werden vor dem Laden des Overlays die angegebenen Zeilennummern gelöscht.

4. DIE VERWENDUNG DER SPRACHELEMENTE

4.1 BILDUNG VON VARIABLENNAMEN; FORMATBESCHREIBUNG; SYNTAX

4.1.1 Bildung von Variablennamen (vgl. auch Abschnitt 2.5.3.2)

Für jeden einzelnen Variablennamen können von 1 bis zu 39 Zeichen verwendet werden. Variablennamen, die sich in mindestens einem Zeichen bzw. dessen Position unterscheiden, sind verschieden. Zulässig sind neben dem Punkt alle Groß- und Kleinbuchstaben außer Ä, Ö, Ü, ä, ü, ö und ß sowie die Ziffern Ø und 9.

Das erste Zeichen eines Variablennamens muß ein Buchstabe sein.

Für Variablennamen oder Befehle oder Anweisungen oder Funktionen gilt:
Groß- und Kleinbuchstaben werden vom M 2Ø nicht unterschieden. Werden Kleinbuchstaben eingegeben, so ersetzt der M2Ø diese beim Listen sofort durch die entsprechenden Großbuchstaben. Das Gesagte gilt nicht bei Stringkonstanten (vgl. Abschnitt 2.5.3.1) und Filenamen.

Bei Variablenbezeichnungen können zusätzlich zum Namen am Ende noch die Zeichen **%**, **!**, **#** oder **$** angehängt sein, um den Typ der Variablen zu bestimmen, gegebenenfalls sogar als 40. Zeichen (vgl. Abschnitt 2.5).
Die Zeichen (und) dienen dazu, Elemente von Arrays zu spezifizieren (vgl. Abschnitt 2.5.3.3). Arrays können ebenfalls von jedem Typ sein (z.B. NAMEN$() oder KENN.NR!()).

Variablennamen, die sich nur durch das Typkennzeichen unterscheiden, sind verschieden. Ebenso sind Namen, die durch () als Array gekennzeichnet sind, verschieden von solchen ohne ().

Schlüsselwörter sowie reservierte Variablen (in diesem Handbuch stets **fett** gedruckt) dürfen nicht als Variablennamen, wohl aber als Bestandteil von Variablennamen verwendet werden (vgl. Kapitel 2.5.3.2).

4.1.2 Formatbeschreibung laut Syntax von Befehlen, Anweisungen und Funktionen.

Um alle möglichen Fälle der Formulierung eines Befehls oder einer Anweisung möglichst knapp, aber umfassend darstellen zu können, werden bei der FORMAT-Beschreibung die folgenden Hilfssymbole und Darstellungsmittel verwendet:

{ , } , [,] , ... ,

Großschreibung, Kleinschreibung, Unterstreichung (____), X (Negativdarstellung) und Fettdruck.

Diese Hilfssymbole sind bei der Eingabe des Befehls, der Anweisung oder der Funktion wegzulassen.

BEDEUTUNG DER SYMBOLE:

{ }

einer der in den Klammern untereinander angeführten Parameter muß verwendet werden (Alternativ-Auswahl)

[ ]

der/die Parameter in den Klammern kann verwendet werden

. . .

der/die Parameter in der zuletztgenannte/n [ ] kann beliebig oft erneut verwendet werden

___

Default-Wert = Wert wird vom System selbständig angesetzt, sofern der Parameter vom Anwender nicht eingegeben wird. Solche Default-Werte stehen immer in [] oder {} .

SPRACHELEMENT
(GROßSCHREIBUNG)

feste Vorgabe als Sprachelement; muß genau so eingegeben werden.

FETTDRUCK

Reservierte Wörter und jeweils erforderliche Sonderzeichen werden **fett** gedruckt.

Begriff
(Kleinschreibung)

Platzhalter, ist durch einen aktuellen Parameter zu ersetzen. Dieser Parameter ist eine vom Anwender zu bestimmende Zeichenfolge, die dem 'Begriff' gerecht wird.

X
(Negativdarst.)

Betätigen der Taste, die das in Negativdarstellung angegebene Zeichen (hier z.B. X) erzeugt, in Verbindung mit der blauen Taste (sog. CONTROL-Taste), aber ohne Eingabeabschlußtaste.

, od. () od.
/ od. # u.ä.

und andere Sonderzeichen, die in der Syntaxbeschreibung nicht in [] stehen, müssen gesetzt werden, auch wenn ein folgender Ausdruck entfällt, aber noch etwas folgt (Platzhalterfunktion). Wenn gar nichts mehr folgt, muß das Zeichen entfallen. () müssen voll gesetzt werden, wenn nötig.

b

Blank; dieses Symbol wird nur an einigen Stellen ausdrücklich gesetzt, um die benötigte Anzahl Leertasten sofort anzuzeigen. Im allgemeinen ist diese jedoch durch Abzählen der an der entsprechenden Position der Folgezeile stehenden Buchstaben zu ermitteln.

ACHTUNG:

Die Eingabe von Sprachelementen ist nicht "formfrei". Das heißt, daß die Sprachelemente und Parameter nicht einfach hintereinander eingegeben und trennende Blanks dabei weggelassen werden können. Schlüsselwörter müssen durch Blanks von sonstigen Parametern getrennt sein.

Beispiele:

1) Die Eingabe
30 **PRINT**A,B(1) führt auf die Fehlermeldung "Syntax error". Es muß
30 **PRINT** A,B(1) eingegeben werden.

2) 37521**GOSUB**6500:C=**ASC(**Q$**)** führt auf die Fehlermeldung "Syntax error".

Es muß

37521**GOSUB** 6500:C=**ASC(**Q$**)** eingegeben werden.

Von besonderer Bedeutung bei einer Syntaxbeschreibung (FORMAT-Beschreibung der Befehle, Anweisungen und Funktionen) ist, auf die angegebene Art von Parametern zu achten.

So muß z.B. bei

{ Stringkonstante
num.Konstante }

entweder nur eine Stringkonstante, z.B. "NAME", oder aber nur eine numerische Konstante, z.B. -7.21, eingegeben werden, aber keine Variablenbezeichnung oder Ergebnisse von Funktionen oder Operationen (dafür müßte dann 'Ausdruck' statt 'Konstante' definiert sein).

Beispiele:

1) Die **DATA**-Anweisung hat das FORMAT

DATA { num.Konst. / Stringkonst. } [, { num.Konst. / Stringkonst. }] ...

Die Anweisung
52 **DATA** "JAN",31,"FEB",28,"MRZ",31
ist laut Formatbeschreibung syntaktisch korrekt, nicht aber

47 N=31
48 **DATA** "JAN",N,"FEB",N-3,"MRZ",N

falsch falsch falsch

N ist keine num.Konstante, ebensowenig N-3.

2) Die **OPEN**-Anweisung hat das Format

OPEN Zugriffsart, [#] Filenr., Filename [, Record-Länge]

Zugriffsart:	Stringausdruck	(Ergebnis: "I", "O", "A" oder "R")
Filenr.:	num.Ausdruck	(Ergebnis: 1 - 15)
Filename:	Stringausdruck	(Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht; vgl. Kapitel 4.1.3)
Record-Länge:	num.Ausdruck	(Ergebnis: 1 - beliebig)

Nach dieser Formatbeschreibung ist die Anweisung

42316 **OPEN** M$(I%),F%(I%),**MID$**(FILEN$,A%(I%),E%(I%)),R%(I%)
syntaktisch korrekt.

M$(I%) darf nur den Inhalt "R" haben (da die Inhalte "I", "A" oder "O", bei Angabe der 'Record-Länge' den Fehler "Illegal function call" liefern!)
F%(I%) darf nur ganze Zahlen zwischen 1 und 15 enthalten;
die String-Funktion **MID$**(FILEN$,A%(I%),E%(I%)) muß als Ergebnis einen String liefern, der den auf Diskette vorhandenen, gewünschten Filenamen beschreibt;
R(I%) muß eine ganze Zahl größer als 1 ergeben.

Zulässig wäre auch die Anweisung

4117 **OPEN** "I",2,"Ø:DAFILE"

3) Die Erklärung

H löscht das zuletzt am Bildschirm abgebildete Zeichen

bedeutet:
Wird die Taste H in Verbindung mit der blauen Taste (CONTROL-Taste), aber ohne Eingabeabschlußtaste, gedrückt, wird das zuletzt am Bildschirm erzeugte Zeichen wieder gelöscht.

4.1.3 Bildung von Filenamen

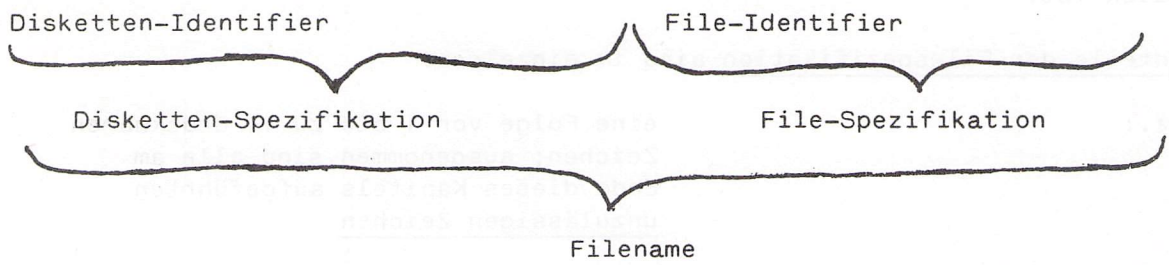
Zur vollständigen Identifikation eines Files ist anzugeben

- auf welcher Diskette oder Diskettenstation das File sich befindet
- welches File dieser Diskette gemeint ist.

Ein Filename besteht somit aus der Diskettenspezifikation und der Filespezifikation.

Der gesamte Filename hat folgendes FORMAT:

```
[[{Stationsbez.  }][[/Disketten-Password]:]Filebez.[.Typ][/File-Password]]
 [{Diskettenbez.}]
```



1. die Diskettenspezifikation

   Sie kann sein:

   - Die Bezeichnung, die einer Diskette gegeben wurde ("Name der Diskette")
   - Die Bezeichnung der Station, in der die Diskette liegt.

   Es kann wahlweise die Bezeichnung der Diskette oder die Bezeichnung der Station (Ø oder 1) angegeben werden.
   Zusätzlich zur Bezeichnung kann einer Diskette ein Password zugeordnet werden, ohne dessen Kenntnis ein Zugriff auf die Diskette nicht möglich ist.

   Anmerkung:

   Aus Vereinfachungsgründen wird im gesamten Handbuch statt des in der EDV gebräuchlichen Begriffs "Volume" stets der Begriff "Diskette" verwendet.

   Bestandteile der Diskettenspezifikation sind im einzelnen:

   Stationsbez.: Ø (rechte Station)
   1 (linke Station)

Diskettenbez.:	eine Folge von 1 bis zu 14 Zeichen; ausgenommen sind alle am Ende dieses Kapitels aufgeführten unzulässigen Zeichen
Disketten-Password:	eine Folge von 1 bis zu 14 druckbaren Zeichen; ausgenommen sind alle am Ende dieses Kapitels aufgeführten unzulässigen Zeichen

2. die Filespezifikation:

Sie enthält die Bezeichnung des Files und ggf. eine Angabe über den Typ des Files. Enthält eine Filespezifikation eine Typangabe, so muß diese als Bestandteil angegeben werden.
Zusätzlich zu Filebezeichnung .Typ kann einem File ein Password zugeordnet werden, ohne dessen Kenntnis ein Zugriff auf das File nicht möglich ist.

Bestandteile der Filespezifikation sind im einzelnen:

Filebez.:	eine Folge von 1 bis zu 14 druckbaren Zeichen; ausgenommen sind alle am Ende dieses Kapitels aufgeführten unzulässigen Zeichen
Typ:	eine Folge von mindestens einem druckbaren Zeichen; die maximal Anzahl Zeichen ergibt sich aus 13 minus Länge der Filebezeichnung; ausgenommen sind alle am Ende dieses Kapitels aufgeführten unzulässigen Zeichen. Einige Typ-Zeichenfolgen haben für PCOS spezielle Bedeutung und zwar: cmd, sav, SAV, LST, lst, BAS und bas.
File-Password:	eine Folge von 1 bis zu 14 druckbaren Zeichen; ausgenommen sind alle am Ende dieses Kapitels aufgeführten unzulässigen Zeichen

Beispiele:

Die folgenden Filenamen sind zulässig:

a) "Ø/DKENNW:ANWPROG.BAS/FPW"

b) "DISKNAME:DAFILE.DAT"

c) "1:ASCFILE"

d) "1:MODUL1.ASC/FILEPASS"

e) "KUNDEN"

Bemerkungen:

- Groß- und Kleinbuchstaben sind bei Filenamen zu unterscheiden!
- Sind Disketten- oder File-Passwords vergeben worden, sind diese stets anzugeben.
- Disketten-Passwords können mit dem PCOS-Befehl **vp** (Aufhebung durch **vd**) vergeben werden; File-Passwords können mit dem PCOS-Befehl **fp** (Aufhebung durch **fd**) vergeben werden.
- Wird beim 'Filenamen' die 'Stationsbezeichnung' bzw. die 'Disketten-bezeichnung' weggelassen, wird die Suche nach dem File auf der aktuellen Station begonnen. Das ist die zuletzt angesprochene Station.
- Beim Abspeichern eines Programmfiles (vgl. **SAVE**-Befehl mit Parameter **P**) kann eine Kodierung erreicht werden, die vor Einsicht und Änderung des Files schützt (z.B. vor den Befehlen **LOAD**, **DELETE**, **RENUM**).
- Sowohl Disketten als auch Files können mit Schreibschutz versehen werden. Ersteres geschieht durch Überkleben der Ausstanzung am oberen rechten Rand der Diskette mit einem Stück Aluminiumfolie. Der Schreibschutz von Files wird unter Verwendung der PCOS-Befehle **fw** (Aufhebung: **fu**) erreicht.
- Sind Disketten mit Schreibschutz versehen, führt jeder Schreibversuch sofort auf einen Fehler 57 (Disk I/O-Error).

Unzulässige Zeichen

:	Doppelpunkt		
.	Punkt		
/	Slash		
,	Komma		
+	Plus	Ö	
*	Sternchen		
"	Anführungszeichen	⍽	Blank
-	Bindestrich oder Minus	;	Strichpunkt
#	Nummernzeichen	$	Dollarzeichen
&	Ampersand	@	at-Zeichen
>	größer-als-Zeichen	\	Backslash
		'	Hochkomma
?	Fragezeichen	£	Pfundzeichen
!	Ausrufezeichen	§	Paragraph-Zeichen
		=	Gleichheitszeichen

4.2 GEGLIEDERTE ZUSAMMENSTELLUNG DER BEFEHLE, ANWEISUNGEN UND FUNKTIONEN

4.2.2 Liste der für Variablennamen gesperrten Sprachelemente

ABS	FN	POS
ALL	FOR	PRESET
AND	FRE	PRINT
AS	GET	PSET
ASC	GOSUB	PUT
ATN	GOTO	RANDOMIZE
AUTO	HEX$	READ
CALL	IF	REM
CDBL	IMP	RENUM
CHAIN	INKEY$	RESTORE
CHR$	INPUT	RESUME
CINT	INPUT$	RETURN
CIRCLE	INSTR	RIGHT$
CLEAR	INT	RND
CLOSE	KILL	RSET
CLS	LEFT$	RUN
COLOR	LEN	SAVE
COMMON	LET	SCALE
CONT	LINE	SCALEX
COS	LIST	SCALEY
CSNG	LLIST	SGN
CURSOR	LOAD	SIN
CVD	LOC	SPACE$
CVI	LOF	SPC( )
CVS	LOG	SQR
DATA	LPOS	STEP
DATE$	LPRINT	STOP
DEFDBL	LSET	STR$
DEF	MERGE	STRING$
DEFINT	MID$	SWAP
DEFSNG	MKD$	SYSTEM
DEFSTR	MKI$	TAB( )
DELETE	MKS$	TAN
DIM	MOD	THEN
DRAW	NAME	TIME$
EDIT	NAME$	TO
ELSE	NEW	TROFF
END	NEXT	TRON
EOF	NOT	USING
EQV	NULL	USR
ERASE	OCT$	VAL
ERL	ON	VARPTR
ERR	OPEN	WEND
ERROR	OPTION	WHILE
EXEC	OR	WIDTH
EXP	PAINT	WINDOW
FIELD	PEEK	WRITE
FILES	POINT	XOR
FIX	POKE	

4.3 FUNKTIONELLER ZUSAMMENHANG DER BEFEHLE UND ANWEISUNGEN

4.3.1 Erstellen von Programmen

Programme für den M 20 werden in der BASIC-Ebene erstellt (zur Anwahl der Ebene vgl. Kapitel 1.2). Nach Eingabe des Befehls **NEW** können BASIC-Anweisungen in den Arbeitsspeicher eingegeben werden. Dabei kann die automatische Zeilennumerierung (siehe Befehl **AUTO**) eingesetzt werden. Die Zeilennummern können auch per Hand eingetippt werden, wobei führende Nullen weggelassen werden dürfen. Die (Mehrfach-)Anweisungen werden in dem Moment im Arbeitsspeicher abgestellt, in dem eine Eingabeabschlußtaste (CR) betätigt wird. Die Reihenfolge der Erfassung von Anweisungszeilen ist beliebig; die (Mehrfach-)Anweisungen werden jedoch in der Reihenfolge der Zeilennummern im Arbeitsspeicher abgestellt.

Zur Änderung von (Mehrfach-)Anweisungen stehen die Möglichkeiten des Edit-Mode (vgl. Befehl **EDIT** und Kapitel 3.4.4) zur Verfügung.

(Mehrfach-)Anweisungen können in der BASIC-Ebene zur sofortigen Ausführung gebracht werden, indem sie ohne Zeilennummer eingegeben werden. Solche Anweisungen werden jedoch nicht im Arbeitsspeicher gespeichert (vgl. Kapitel 3.2)! Der Direkt-Mode wird z.B. folgendermaßen eingesetzt:

Ein im Arbeitsspeicher befindliches Programm wird durch **RUN** zum Laufen gebracht. (Damit werden alle Variableninhalte und Stack-Speicher des vorherigen Programms gelöscht; alle (logisch) offenen Files werden geschlossen). Durch eine BASIC-Anweisung **STOP** im Programm oder durch Betätigen von C wird das Programm unterbrochen; im Direkt-Mode werden die Inhalte bestimmter Variablen abgefragt oder Testwerte auf Variablen zugewiesen. Anschließend wird das Programm mit dem Befehl **CONT** fortgesetzt.

ACHTUNG: Wird, z.B. wegen eines Fehlers, der Edit-Mode zum Ändern von Anweisungen eingesetzt, so geht dabei der vorher geschaffene Datenbereich verloren!!

Eine zweite Möglichkeit, Programme zu erstellen, besteht darin, bereits früher in Form von ASCII-Files auf Diskette gespeicherte Programm-Module mit dem aktuellen Inhalt des Arbeitsspeichers zu verbinden (vgl. **CHAIN**, **MERGE** und Kapitel 3.5.5)

4.3.2 Vereinbarungsteil

Nach den das Programm beschreibenden **REM**s folgt der Vereinbarungsteil. In diesem werden festgelegt:

a) die im Arbeitsspeicher für das betreffende Programm (einschließlich Daten) zu reservierende Anzahl Bytes - sofern - nötig, (nur Verringerung der durch PCOS-Befehl definierten Größe ist möglich!), sowie die Anzahl Stack-Register (sofern sie von der Standard-Anzahl abweichen soll); für beides ist die Anweisung **CLEAR** einzusetzen;

b) die Typen der Variablen, sofern diese nicht durch Typkennzeichen im Gesamtprogramm vereinbart werden (**DEFINT**, **DEFSNG**, **DEFDBL**, **DEFSTR**);

c) der niedrigstmögliche Wert für alle Array-Indices (Ø oder 1; **OPTION BASE**);

d) die Dimension von Arrays; es wird definiert, wieviele Indices zur Spezifizierung eines Elements des jeweiligen Arrays nötig sind (**DIM**);

e) der höchstmögliche Wert für die einzelnen Indices der jeweiligen Arrays (**DIM**).

Beispiel:

```
10 OPTION BASE 1
20 DEFINT I-J:DEFDBL X-Z:DEFSTR S-T
30 INDEX1=5:INDEX2=5:INDEX3=5
40 DIM D3.!(INDEX1,INDEX2,INDEX3),TEXTE(INDEX1,
INDEX2,INDEX3)
50 DIM X(5),Y(5),Z(5)
```

10 legt fest, daß alle Array-Elemente als niedrigsten Index 1 haben dürfen.

20 legt fest, daß im folgenden alle ohne Typkennzeichen versehenen Variablennamen, die mit I oder J anfangen, vom Typ Integer sein sollen; diejenigen mit X, Y oder Z am Anfang doppelt genaue numerische Variablen und die mit S oder T anfangenden Strings. Mit I, J, S, T, X, Y, Z beginnende Variablen können jedoch durch Typkennzeichen am Ende von dieser Vereinbarung abweichend definiert werden.

30 setzt die Integer-Variablen INDEX1, INDEX2 und INDEX3 alle auf 5.

40 legt fest, daß der einfach genaue Array D3.!() dreidimensional ist, wobei als höchster Wert für alle drei Indices 5 gilt. Da der niedrigste Wert 1 ist, wird für D3.!() somit für 5x5x5 Elemente Platz im Arbeitsspeicher reserviert. Auch der String-Array TEXTE() wird dreidimensional mit 5x5x5 Elementen definiert.

50 legt fest, daß die doppelt genauen Arrays X(), Y() und Z() je eindimensional sind und jeder 5 Elemente hat.

Default-Werte: Numerische Variablen ohne jede Typvereinbarung sind von einfacher Genauigkeit.
Die niedrigsten Indices von Array-Elementen sind 0.
Die höchsten Indices sind 10.
Numerische Variablen haben beim ersten Aufruf ohne vorherige Wertzuweisung den Inhalt 0, alle Strings den Inhalt Leerstring.

Arrays können bis zu jeder beliebigen Dimension mit beliebig hohem Index dimensioniert werden (Begrenzung nur durch Kapazität des Arbeitsspeichers). Sie können dynamisch dimensioniert werden (vgl. Anweisung 40). Nach **ERASE**-Anweisung können sie redimensioniert werden. Der alte Inhalt und die alte Dimensionierung gehen dabei verloren.
Strings werden im Speicher dynamisch verwaltet, so daß es nicht nötig ist, für sie Maximallängen zu definieren. Das Gesagte gilt auch für String-Arrays. Der für Strings benötigte Platz im Arbeitsspeicher ist somit abhängig von der aktuellen Länge der einzelnen Strings bzw. String-Array-Elemente! Um die bei Stringmanipulationen auftretenden "Fehlreservierungen" im Arbeitsspeicher aufzuheben und ggf. die Rechengeschwindigkeit zu erhöhen, empfiehlt sich im Programmablauf von Zeit zu Zeit eine "Bereinigung" des Arbeitsspeichers unter Anwendung der Funktion **FRE**(Stringausdr.)!

Es empfiehlt sich, im Vereinbarungsteil auch alle im Programm benötigten selbstdefinierten Funktionen (vgl. Kapitel 4.3.7 und Anweisung **DEF FN**) anzuführen, denn sie müssen stets vor Aufruf definiert worden sein.

4.3.3 Zuweisungen

Für (Wert-) Zuweisungen auf numerische oder String-Variablen bieten sich folgende Möglichkeiten:

1. mit Hilfe der **LET...=** -Anweisung
   Das Sprachelement **LET** kann dabei weggelassen werden. Mehrfachzuweisungen sind nicht möglich und müssen durch Mehrfach-Anweisungen ersetzt werden.

   Array-Zuweisungen (z.B. Setzen aller Elemente eines numerischen Arrays auf 1) sind nicht möglich. Dies muß in Form einer (ggf. verschachtelten) Schleife erfolgen.

2. mit Hilfe der Anweisung
   MID$...= ,
   um in einer Stringvariablen einen Teil durch einen anderen String zu ersetzen

3. mit Hilfe der Anweisung **SWAP**, um Variableninhalte zu vertauschen

4. mit Hilfe der Anweisung **LSET** und **RSET** auf Feldvariablen eines Random-File-Puffers bzw. zum links- oder rechtsbündigen Einsetzen von String-Inhalten in vorhandene String-Variablen-Inhalte

5. mit Hilfe der **FOR...NEXT**-Anweisung auf deren Laufvariable

6. mit Hilfe der Anweisung **ERROR** auf die reservierte Variable **ERR**

7. mit Hilfe von Tastatur-Inputs (**INPUT**, **LINE INPUT**)

8. mit Hilfe von Inputs aus einem internen Datenfile (**READ**)

9. mit Hilfe von Inputs aus einem sequentiellen externen Datenfile (**INPUT#** , **LINE INPUT#**)

10. mit Hilfe von Inputs aus einem Random-File-Puffer (**GET**, **INPUT#** , **LINE INPUT#**).

Bei allen Wertzuweisungen numerischer Größen auf numerische Variablen ist auf eine ggf. notwendige Typumwandlung (vgl. Kapitel 2.5.1.4) und daraus resultierende Signifikanzverluste zu achten (vgl. z.B. Kapitel 2.7.1)!!

Solche Signifikanzverluste lassen sich auch durch Anwendung der Funktionen **CDBL** u.ä. nicht vermeiden.

Die Problematik kann jedoch mit Hilfe einer Anweisung der Art

```
D# =VAL(STR$(G!))
```

umgangen werden, also unter Zuhilfenahme der Stringverarbeitung.

4.3.4 Stringverarbeitung

Es werden in diesem Kapitel nur die folgenden Stringmanipulationen besprochen:

1. Stringverkettung
2. Teilstringbildung und Ersetzungen in Strings
3. Verwandlung von numerischen Werten in Strings und umgekehrt (Konvertierungen)

Formatierungsmöglichkeiten (Masken) werden in Kapitel 4.3.9.2 erörtert, Fragen im Zusammenhang mit Inputs über Tastatur in Kapitel 4.3.8 und die Verarbeitung von Strings beim Schreiben und Lesen auf externen Datenfiles im Kapitel 4.3.10.2.

4.3.4.1 Stringverkettung (String-Addition)

Zur Verkettung von Stringausdrücken dient der String-Operator +.

Beispiel:

```
1 CLS
2 DEFSTR T
10 A$="Personal":T="Computer":T(1)="M":T(2)="20"
20 E$=A$+"-"+T+" "+T(1)+" "+T(2)
30 LPRINT E$
```

```
Personal-Computer M 20
```

4.3.4.2 Teilstringbildung (Substrings) und Ersetzungen in Strings

Zur Bildung von Teilstrings können die Funktionen **LEN**, **INSTR**, **LEFT$**, **RIGHT$** und **MID$** (als Funktion!) eingesetzt werden.

Beispiel: (der obige Programmteil wird übernommen)

```
40 POSITION%=INSTR(E$,"-")

50 T(3)=LEFT$(E$,POSITION%-1)
60 T(4)=RIGHT$(E$,LEN(E$)-POSITION%)
70 T(5)=MID$(E$,POSITION%+1,INSTR(E$," ")-POSITION%)
80 LPRINT T(3):LPRINT T(4):LPRINT T(5):LPRINT POSITION%
```

```
Personal-Computer M 20
Personal
Computer M 20
Computer
 9
```

Ersetzungen von Teilen eines Strings durch einen anderen String können entweder durch Verkettung der betroffenen Teilstrings oder aber mit Hilfe der Anweisung **MID$...=** durchgeführt werden.

Beispiel: (das obige Programm wird fortgesetzt)

```
90 MID$(E$,1)="      Ihr "
91 LPRINT E$
92 E$=RIGHT$(E$,LEN(E$)-5)
95 LPRINT E$
```

```
Personal-Computer M 20
Personal
Computer M 20
Computer
 9
     Ihr Computer M 20
Ihr Computer M 20
```

Wie man sieht, läßt sich bei Verwendung der **MID$**-Anweisung die aktuelle Länge des Ausgangsstrings nicht verändern.

Um in Strings eine bestimmte Anzahl von Blanks einzubringen, kann die Funktion **SPACE$** eingesetzt werden.

Mit der Funktion **STRING$** lassen sich Strings erzeugen, die ausschließlich aus einem gewählten Zeichen bestehen (z.B. Blanks für das Löschen von Texten im Bildschirm).

Beispiel: (obiges Programm wird fortgesetzt)

```
96 ANZAHL%=9
97 LPRINT STRING$(LEN(E$)+ANZAHL%,"*")
100 LPRINT LEFT$(E$,13)+SPACE$(ANZAHL%)+RIGHT$(E$,4)
```

```
Personal-Computer M 20
Personal
Computer M 20
Computer
 9
     Ihr Computer M 20
Ihr Computer M 20
**************************
Ihr Computer          M 20
```

Mit Hilfe der Anweisungen **LSET** und **RSET** lassen sich in Strings mit einer vorher festgelegten Länge (z.B. für Ausgabefelder) rechtsbündige oder linksbündige Ersetzungen vornehmen. Nicht von der Ersetzung betroffene Zeichen werden durch Blanks ersetzt.

Beispiel: (neues Programm)

```
1 LPRINT STRING$(40,"*")
10 FLAENG%(1)=17:FLAENG%(2)=20
20 FELD$(1)=STRING$(FLAENG%(1),"@")
30 FELD$(2)=STRING$(FLAENG%(2),"@")
42 LPRINT FELD$(1):LPRINT FELD$(2)
45 RSET FELD$(1)="1. Feld"
50 LSET FELD$(2)="2. Feld"
70 LPRINT FELD$(1):LPRINT FELD$(2)
```

```
****************************************
@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@
          1. Feld
2. Feld
```

4.3.4.3 Konvertierungen

Für die Umwandlung von numerischen Werten zu Strings bieten sich folgende Möglichkeiten:

a) Konvertierung von ISO-Codes (vgl. Kapitel 4.3.17) zu den durch sie definierten Zeichen mit Hilfe der Funktionen **CHR$** (Umkehrung: **ASC**) und **STRING$**;

b) direkte Konvertierung von dezimal, oktal oder hexadezimal dargestellten numerischen Werten in die optisch deckungsgleiche Stringdarstellung mit Hilfe der Funktion **STR$** (Umkehrung: **VAL**);

c) Konvertierung von dezimal dargestellten numerischen Werten in die oktale oder hexadezimale Darstellung in String-Form mit Hilfe der Funktionen **OCT$** und **HEX$** (Umkehrung über Stringverkettung und **VAL**).

Beispiel zu a):

```
10 ANF.Z$=CHR$(34)
20 Z%(1)=77:Z%(2)=32:Z%(3)=50:Z%(4)=48
30 M20$=ANF.Z$+CHR$(Z%(1))+CHR$(Z%(2))+CHR$(Z%(3))+CHR$(Z%(4))+ANF.Z$
40 LPRINT STRING$(20,42):LPRINT M20$
```

```
**********************
"M 20"
```

Beispiel zu b):

```
10 A(1)=7.43:A(2)=-60.93:A(3)=-.27:A(4)=3E-20:A(5)=&072:P(6)=&HB7D
20 LPRINT STRING$(15,42)
30 I%=1
40 LPRINT STR$(A(I%))
50 IF I%=6 THEN STOP
60 I%=I%+1:GOTO 40
```

```
***************
 7.43
-60.93
-.27
 3E-20
 58
 2941
```

Die über **STR$** gewonnenen Strings enthalten also die Zeichenfolge der dezimal dargestellten numerischen Werte im Standardformat (vgl. dazu die Anmerkung bei **PRINT** sowie Kapitel 4.3.9). Führende und hinter dem Komma stehende nicht-signifikante Nullen werden nicht konvertiert. Im Gegensatz zum Standardformat wird jedoch kein Blank am Ende des Strings erzeugt!!

Beispiel zu c):

```
10 LPRINT STRING$(50,42)
20 A(1)=635:A(2)=&H6B:A(3)=&030
30 LPRINT A(1),OCT$(A(1)),HEX$(A(1))
40 LPRINT A(2),OCT$(A(2)),HEX$(A(2))
50 LPRINT A(3),OCT$(A(3)),HEX$(A(3))
```

```
**************************************************
 635            1173           27B
 107            153            6B
 24             30             18
```

Die über **OCT$** bzw. **HEX$** gewonnenen Strings enthalten also die Zeichenfolge der oktal bzw. hexadezimal dargestellten numerischen Werte ohne führende Blanks wie beim Standardformat. Die Präfixe **&O** bzw. **&H**, die zur Darstellung oktaler oder hexadezimaler numerischer Konstanten verwendet werden müssen (vgl. Kapitel 2.5.3.1), werden somit nicht vorangestellt.

Für die Umwandlung von Strings in numerische Werte bieten sich folgende Möglichkeiten:

a) die Funktion **ASC** als Umkehrung von **CHR$**;
b) die Funktion **VAL** als Umkehrung von **STR$**.

Beispiel zu a):

```
10 T$="Test-Text"
20 LPRINT ASC(T$)

 84
```

Nur vom ersten Zeichen von T$ wird der ISO-Code ermittelt.

Beispiel zu b):

```
10 T$="34, Strasse"
20 LPRINT VAL(T$)

 34
```

Der numerische Wert muß, ebenso wie ein eventuell vorhandenes Vorzeichen, am Anfang des Arguments stehen, sonst liefert die Funktion **ASC** das Ergebnis Ø. Dazu ein weiteres
Beispiel:

```
10 T$="20-04-81"
20 LPRINT VAL(MID$(T$,3))

-4
```

Bei der Rückkonvertierung eines Strings, der eine oktale oder hexadezimale Zahlendarstellung enthält, ist wie folgt vorzugehen:
Beispiel:

```
10 A#=98
15 LPRINT A#
20 T$=OCT$(A#)
30 LPRINT T$
40 REM  zuerst Verwandlung in eine oktale Konstante!
50 T$="&O"+T$
60 B#=VAL(T$)
65 LPRINT T$
70 LPRINT B#

 98
142
&O142
 98
```

Bei der Rückumwandlung von Strings mit hexadezimaler Zahlendarstellung als Inhalt ist statt **OCT$** die Funktion **HEX$** und statt des Präfixes **&O** das Präfix **&H** vor den rückzukonvertierenden String zu setzen.

Bemerkung:

Bei häufigen Stringmanipulationen empfiehlt sich, im Programm hin und wieder mit Hilfe der Funktion **FRE**(Stringausdr.) den Arbeitsspeichers zu bereinigen (neu zu ordnen) (vgl. **FRE**).

4.3.5 Ablaufsteuerung

Um in einem Programm vom linearen Ablauf, der durch die Ordnung der Programm-Anweisungen im Arbeitsspeicher nach aufsteigender Zeilennummer festgelegt ist, abzuweichen, stehen Sprunganweisungen zur Verfügung. Die Zeilennummer des Sprungzieles kann grundsätzlich nicht als Variable definiert werden.
Es ist nicht möglich, bestimmte Anweisungen in einer Programmzeile anzuspringen, die aus durch : getrennten Anweisungen besteht (Ausnahmen: **RETURN** und **RESUME**).

4.3.5.1 Unbedingte Sprünge

Mit den Anweisungen **GOTO**, **GOSUB**, **RETURN** oder **RESUME** (diese wird im Zusammenhang der Fehlerbehandlungsroutine in Kapitel 4.3.12 besprochen) werden Sprünge verlangt, die immer ausgeführt werden, sobald die betreffende Anweisung erkannt wird.
Bei **GOTO** wird die Zeilennummer des Absprungs nicht gespeichert. Mit einer **GOSUB**-Anweisung wird sofort ein Stack-Speicher errichtet, in dem festgehalten wird, von wo der **GOSUB**-Aufruf erfolgte. Trifft das Programm auf eine **RETURN**-Anweisung, so wird die Absprungadresse aus dem letzten errichteten Stack-Speicher wieder gelöscht und automatisch zu der Anweisung gesprungen, die der Anweisung mit der gerade ermittelten Adresse folgt. Dies muß nicht die nächste Zeilennummer sein (bei durch : verbundenen Anweisungen, von denen eine ein **GOSUB** ist!).
Werden zuviele **GOSUB**-Anweisungen verschachtelt oder wurden mit **GOSUB**-Anweisungen zu viele Stack-Speicher errichtet, ohne daß sie durch **RETURN**-Anweisungen wieder gelöscht wurden, kann ein Overflow im Arbeitsspeicher auftreten. Die Anzahl der möglichen Stack-Speicher kann mit Hilfe der Anweisung **CLEAR** geändert werden.

Beispiel:

```
1 I%=0
10 I%=I%+1:GOSUB 2000
2000 I%=I%+1:GOSUB 10
```

Dies führt zur Fehlermeldung "Out of memory in 2000". Wird I% abgefragt (?I%), so wird bei Standard-Arbeitsspeicher 71 ausgegeben. Es konnten also nur 70 **GOSUB**-Stack-Speicher gleichzeitig verwaltet werden (Default-Wert).

Es existiert keine BASIC-Anweisung, mit der die Anzahl der angelegten Stack-Speicher direkt abgefragt werden kann oder mit der alle Stack-Speicher gelöscht werden können.

Beispiel zu **GOTO** und **GOSUB**:

```
1 REM Programmbeschreibung
      .
      .
      .
11 GOSUB 29000
      .
      .
      .
500 GOTO 15000
      .
      .
      .
15000 REM Hauptverarbeitung
      .
      .
      .
15672 GOSUB 20200:GOSUB 15900
      .
      .
      .
15899 GOTO 15000
      .
      .
      .
15900 REM Datumsverarbeitung
      .
      .
      .
15999 RETURN
      .
      .
      .

20200 REM Datumsaufruf
   .
   .
   .
20204 RETURN
   .
   .
   .
29000 REM Vereinbarungsteil
   .
   .
   .
29299 RETURN
```

4.3.5.2 Bedingte Sprünge

Häufig werden Programmverzweigungen nur dann gewünscht, wenn bestimmte Bedingungen erfüllt bzw. nicht erfüllt sind. Solche Möglichkeiten bieten die Anweisungen **IF...THEN...ELSE**, **ON...GOTO**, **ON...GOSUB** und **ON ERROR GOTO** (diese wird im Zusammenhang der Fehlerbehandlungsroutine, Kapitel 4.3.12, erörtert). Außerdem leisten die Schleifenanweisungen **FOR...NEXT** und **WHILE...WEND** Ähnliches.

Beispiel zu IF...THEN...ELSE:

```
1 REM Ziehung 6 aus 49
5 RANDOMIZE
10 DIM Q%(49)
20 ZIEHUNG%=1
30 ZAHL%=INT(RND*100)
40 IF ZAHL%=0 OR ZAHL%>49 GOTO 30
42 REM Q%(ZAHL%) ENTHAELT NR. DER ZIEHUNG,
       WENN ZAHL BEREITS GEZOGEN IST!
45 IF Q%(ZAHL%) THEN 30 ' = IF Q%(ZAHL%)<>0
50 Q%(ZAHL%)=ZIEHUNG%
60 LPRINT ZIEHUNG%", Ziehung: ";ZAHL%
70 ZIEHUNG%=ZIEHUNG%+1
80 IF ZIEHUNG%<7 THEN 30 ELSE I%=0:GOSUB 1000
90 STOP
1000 LPRINT:LPRINT
1005 I%=I%+1:IF Q%(I%) THEN LPRINT Q%(I%);
     ",Ziehung war";I%
1010 IF I%=49 THEN RETURN ELSE 1005
```

```
 1 , Ziehung:  31
 2 , Ziehung:  19
 3 , Ziehung:  39
 4 , Ziehung:  18
 5 , Ziehung:  22
 6 , Ziehung:  37


 4 ,Ziehung war 18
 2 ,Ziehung war 19
 5 ,Ziehung war 22
 1 ,Ziehung war 31
 6 ,Ziehung war 37
 3 ,Ziehung war 39
```

Beispiel zu ON...GOTO und ON...GOSUB:

```
1 REM WAHRHEITSWERTE ERMITTELN
2 DEFSTR T
3 OPTION BASE 0
5 T(0)="wahr":T(1)="falsch!"

10 INPUT "# Wert 1, # Wert 2:";W1,W2
20 INPUT "Vergleich Wert 1 mit Wert 2
 1: =  2: <  3: >  4: <>  5: <=  6: >=";Q%
30 IF Q%<1 OR Q%>6 THEN 20
35 LPRINT W1;
40 ON Q% GOSUB 100,200,300,400,500,600 : GOTO 9900
100 REM =
110 LPRINT "="W2"ist " T((W1=W2)+1):RETURN
200 REM <
210 LPRINT "<"W2"ist " T((W1<W2)+1):RETURN
300 REM >
310 LPRINT "<"W2"ist " T((W1>W2)+1):RETURN
400 REM <>
410 LPRINT "<>"W2"ist " T((W1<>W2)+1):RETURN
500 REM <=
510 LPRINT "<="W2"ist " T((W1<=W2)+1):RETURN
600 REM >=
610 LPRINT ">="W2"ist " T((W1>=W2)+1):RETURN
9900 INPUT "Neuer Vergleich ja -->1                    nein -->2
        Vertauschen vorherige Werte -->3";Q%
9910 IF Q%<1 OR Q%>3 THEN 9900
9920 ON Q% GOTO 10,9999
9930 SWAP W1,W2
9940 GOTO 20
9999 END
```

```
 5 < 5 ist falsch!
 6 <> 4 ist wahr
 4 = 6 ist falsch!
```

4.3.5.3 Programmende und Programmunterbrechung

Ein BASIC-Programm im M 2Ø kann eine oder mehrere **END**-Anweisungen an beliebigen Stellen im Programm haben. Diese Anweisung bewirkt das sofortige Programmende unter Schließung aller Datenfiles und die Meldung "ok". Die Variableninhalte im Arbeitsspeicher bleiben vorhanden.

Es kann aber auch ohne **END**-Anweisung gearbeitet werden. Erreicht das Programm die höchste Zeilennummer ohne **END**, wird ebenfalls "ok" gemeldet. Wurden nicht die Datenfiles mit **CLOSE** vorher geschlossen, so bleiben sie in diesem Falle offen.

Es kann ein vorzeitiger Programmabbruch durch C jederzeit erzwungen werden (vgl. Kapitel 3). Die Files werden nicht geschlossen und die Variableninhalte im Arbeitsspeicher bleiben vorhanden. Wird ein Programm durch die **STOP**-Anweisung verlassen, befindet sich das System danach im Direkt-Mode (vgl. Kapitel 3); alle im Programm geöffneten Datenfiles bleiben logisch offen und die Variableninhalte erhalten. Das Programm kann nach **STOP** oder C mit Hilfe des Befehls **CONT** fortgesetzt werden, und zwar wird ab der nächsten Anweisung weitergearbeitet.

Bitte beachten:

Werden durch den Befehl **EDIT** Programmzeilen nach einer **STOP**-Anweisung verändert, so geht der Inhalt aller Variablen verloren. Der Befehl **CONT** bewirkt danach, daß mit Default-Werten für die Variablen weitergearbeitet wird (Ø für num. und "" für String-Variablen)!!

Zur Verkettung von Programmen mit Hilfe von **CHAIN** vgl. Kapitel 4.3.13.

4.3.6 Schleifen

Programmteile, die eine Folge von sich gleichenden oder ähnelnden aufeinanderfolgenden Vorgängen darstellen, werden in Schleifen programmiert. Dazu bieten sich folgende Möglichkeiten:

1. eine Kombination von **GOTO** und **IF...THEN**-Anweisungen;
2. eine **IF...THEN...ELSE**-Anweisung;
3. eine aus den Anweisungen **FOR** und **NEXT** gebildete zählergesteuerte Schleife;
4. eine aus den Anweisungen **WHILE** und **WEND** gebildete, von einem Vergleichs- oder logischen Ausdruck abhängige Schleife.

Beispiel zu 1.:

```
1 REM Einlesen von Daten aus internem Datenfile
30 INPUT "Jahr";N0%
40 IF N0%<1978 OR N0%>1981 THEN 30
50 ON N0%-1977 GOSUB 978,979,980,981
60 I%=1
65 LPRINT "Jahr:";N0%
66 LPRINT "==========="
70 READ U!:LPRINT U!;
80 IF I%<12 THEN I%=I%+1 : GOTO 70
99 END
978 RESTORE 1978 : RETURN
979 RESTORE 1979 : RETURN
980 RESTORE 1980 : RETURN
981 RESTORE 1981 : RETURN
1978 DATA 10,20,40,30,20,80,70,80,60,20,40,10
1979 DATA 15,25,40,25,30,60,50,70,60,10,50,80
1980 DATA 50,60,20,40,30,80,90,120,110,70,65,80
1981 DATA 40,0,0,10,100,105,130,40,70,25,30,90

Jahr: 1980
==========
 50  60  20  40  30  80  90  120  110  70  65  80
```

Beispiel zu 2. (gleiches Programm wie oben, nur geänderte Programmzeilen (70 bis 80):

```
70 I%=I%+1
75 READ U!:LPRINT U!;
80 IF I%=12 THEN END ELSE 70

Jahr: 1979
==========
 10  20  40  30  20  80  70  80  60  20  40  10
```

Beispiel zu 3. (**DATA**- und **RESTORE**-Anweisungen von oben werden übernommen):

```
1 REM Einlesen von Daten aus internem Datenfile
20 INPUT;"von ",N0%
30 IF N0%<1978 OR N0%>1981 THEN 20
40 INPUT" bis ",N9%
50 IF N9%<1978 OR N9%>1981 OR N9%<N0% THEN 40
70 FOR I1%=N9% TO N0% STEP -1
75 LPRINT "Jahr:";I1%
76 LPRINT "==========":LPRINT
79 ON I1%-1977 GOSUB 978,979,980,981
80 FOR I2%=1 TO 12
90 READ U! : LPRINT U!;
100 NEXT I2%
110 LPRINT:LPRINT
120 NEXT
199 END

Jahr: 1981
==========

 40  0  0  10  100  105  130  40  70  25  30  90

Jahr: 1980
==========

 50  60  20  40  30  80  90  120  110  70  65  80

Jahr: 1979
==========

 15  25  40  25  30  60  50  70  60  10  50  80
```

Bei den **FOR...NEXT**-Schleifen handelt es sich um zählergesteuerte "pre-checked"loops, also Schleifen, bei denen die Endwert-Über-bzw.Unterschreitung am Beginn der Schleife geprüft wird. Für jede **FOR...NEXT**-Schleife wird ein Stack-Speicher benötigt. Ein vorzeitiger Aussprung aus der Schleife (z.B. aufgrund einer Bedingung) mit **IF...THEN** oder mit **GOTO** ist zwar möglich, löscht aber nicht den Stack-Speicher.
Das heißt, daß ein Rücksprung mit anschließender Weiterabarbeitung der Schleife möglich ist! Vor allem heißt das auch, daß häufige vorzeitige Absprünge ohne Stack-Löschung bald zum Overflow im Stack-Bereich des Arbeits-Speichers (der standardmäßig nur eine begrenzte Anzahl von Stack-Speichern gleichzeitig ermöglicht) führen (Fehler 7). Dies könnte durch Verwendung der Anweisung **CLEAR** geändert werden. Besser ist ein Verfahren wie im folgenden Bespiel demonstriert (Programmzeilen 80 bis 99):

```
0 REM Untersuchung eines String-Inputs
1 REM ES SIND NUR FOLGENDE BESTANDTEILE ZULAESSIG:
2 REM      alle Gross- und Kleinbuchstaben ausser ß = scharfes s
3 REM      alle Ziffern, Blanks, Bindestriche, Punkte, Kommas
           sowie der slash (/)
4 REM Alle Kleinbuchstaben sollen (z.B. zum Zwecke eines Sorts)
      in Grossbuchstaben verwandelt werden.
10 PRINT  "Eingabe der Zeichenfolge ":LINE INPUT EINGABE$
20 FALSCH%=0
30 FOR I%=1 TO LEN(EINGABE$)
40 C%=ASC(MID$(EINGABE$,I%,1))
50 IF C%=32 OR (C%>43 AND C%<58) OR C%=95 THEN 99
60 IF C%>64 AND C%<94 THEN 99
70 IF C%>96 AND C%<126 THEN MID$(EINGABE$,I%)=CHR$(C%-32) : GOTO
    99
80 FALSCH%=-1
90 I%=LEN(EINGABE$)              'Vorbereitung zum Loeschen des
                                  FOR-NEXT-Stack-Speichers
95 LPRINT EINGABE$;" ist eine unzulaessige Eingabe"
99 NEXT I%
100 IF FALSCH% THEN 10
110 LPRINT EINGABE$ "ist o.k.!"    ' Eingabe ist in Ordnung
120 LPRINT STRING$(I%,"_")
DIESE EINGABE EINHAELT EIN ß (scharfes s)
 ist eine unzulaessige Eingabe
/DIESE EINGABE IST IN 2 SLASHS EINGESCHLOSSEN/ist o.k.!
```

Wie sich am Beispiel zeigt, kann (Anfangs- und) Endwert über Variablen einmalig festgelegt werden. Es gelten die Werte der Variablen im Moment der erstmaligen Abarbeitung der **FOR**-Anweisung (vgl. Programmzeile 30).

Nach vollständiger, mindestens einmaliger Abarbeitung hat die 'Laufvariable' einer **FOR...NEXT**-Schleife den Wert 'Endwert' + 'Schrittweite'. (Ist 'Schrittweite' negativ, ist selbstverständlich der Betrag von 'Schrittweite' zu subtrahieren.) Wird eine solche Schleife nie abgearbeitet (da z.B. 'Endwert' über/unterschritten ist), hat die 'Laufvariable' nach dem Abarbeiten der **FOR**-Anweisung den Wert 'Anfangswert'.

Wird aus einer Schleife vorzeitig direkt ausgesprungen, ohne wie mit Hilfe von Zeilennr. 90 den Stack-Speicher zu löschen, gibt die Laufvariable Aufschluß darüber, wie oft die **FOR...NEXT**-Schleife durchgearbeitet wurde (Rücksprung ist dann möglich!)

Beispiel zu 4.:

```
5115 REM Bubble Sort fuer den Array A$() mit J% Elementen
5118 MARKE%=1 'erzwingt mindestens einen Durchlauf der Schleife
5120 WHILE MARKE%  ' kurz statt: WHILE MARKE%<>0
5122       MARKE%=0
5124       FOR I%=1 TO J%-1
5126            IF A$(I%)>A$(I%+1) THEN
                 SWAP A$(I%),A$(I%+1) : MARKE%=1
5128       NEXT I%
5130 WEND
```

WHILE...WEND-Schleifen benötigen ebenfalls Stack-Speicher und bieten sich vor allem anstelle von **FOR...NEXT**-Schleifen an, wenn die Anzahl der Durchläufe einer Schleife nicht festliegt, sondern abhängt von bestimmten Bedingungen, die sich innerhalb der Schleife ergeben.

4.3.7 Unterprogramme und selbstdefinierte Funktionen

4.3.7.1 Unterprogramme (Subroutinen)

Unterprogramme sind Bestandteile eines Hauptprogramms, die Routinen beinhalten, die in unregelmäßigen Abständen im Hauptprogramm häufiger benötigt werden. Sie sind im allgemeinen stark parametrisiert, d.h. mit Variablen versehen. Unterprogramme werden mit **GOSUB** aufgerufen; mit **RETURN** wird die Kontrolle an das Hauptprogramm zurückgegeben (weitere Abarbeitung ab der auf das betreffende **GOSUB** folgenden Anweisung).

Beispiel:

```
1 REM Datumspruefung
20 INPUT "Datum 1";D1$:D$=D1$
30 GOSUB 700 : IF O.K.% THEN TAG1%=T% : MONAT1%=M%    :AHR1%=J%
                         ELSE 20
40 INPUT "Datum 2";D2$:D$=D2$
50 GOSUB 700 : IF O.K.% THEN TAG2%=T% : MONAT2%=M'    :AHR2%=J%
                         ELSE 40
699 GOTO 5000
700 REM UP Datumspruefung
701 REM nur Eingaben der Form TTMMJJ oder TMMJJ
702 REM 01<=JJ<=99
705 O.K.%=-1    '      Setzen Schalter
710 D=VAL(D$)
720 IF D<=0 OR D<>FIX(D) OR D>999999!    O.K.%=0 : RETURN
725 J%=D-INT(D/100)*100
730 IF J%=0 THEN O.K.%=0 : RETURN
740 T%=INT(D/10000)
750 IF T% = 0 OR T%>31 THEN O.K.%=0 : RETURN
760 M%=INT(D/100) MOD 100   ' Rest der Integerdivision
765 IF M%<1 OR M%>12 THEN O.K.%=0 : RETURN
770 IF T%> 31 + ( (M%=4) OR (M%=6) OR (M%=9) OR (M%=11) ) +
           2*( (M%=2) AND (J% MOD 4 = 0) ) +

           3*( (M%=2) AND (J% MOD 4 <> 0) )   THEN O.K.%=0
            : RETURN
780 RETURN
5000 REM Hauptverarbeitung
5010 LPRINT D1$
5015 LPRINT TAG1%,MON=  %,JAHR1%
5016 LPRINT
5020 LPRINT D2$
5030 LPRINT TAG2% MONAT2% JAHR2%
9900 STOP

10381
 1              3              81

290280
 29             2              80
```

Wie sich im Beispiel zeigt, sind bei der Arbeit mit Unterprogrammen i.a. folgende Schritte nötig:

1. Zuweisung(en) der Daten, die das Unterprogramm benötigt, auf die im Unterprogramm dafür verwendeten Variablen.
2. Setzen von Schaltervariablen im Unterprogramm, falls es im Hauptprogramm später nötig ist zu wissen, welche Abläufe im Unterprogramm stattgefunden haben. (Das Unterprogramm muß ja stets über **RETURN** verlassen werden!)
3. Sichern von Variableninhalten (Ergebnissen) aus dem Unterprogramm in Hilfsvariablen, falls diese Ergebnisse im Hauptprogramm auch benötigt werden, wenn das Unterprogramm später nochmals aufgerufen wird. (Der nächste Aufruf des Unterprogramms verändert sämtliche darin vorkommenden Variablen!)
4. Abgrenzung des Unterprogramms vom Hauptprogramm, z.B. durch **GOTO**-Anweisung vor Beginn des Unterprogramms. (Läuft das Hauptprogramm beim linearen Abarbeiten ohne **GOSUB**-Anweisung direkt in ein Unterprogramm, wird dies erst entdeckt, wenn die **RETURN**-Anweisung erkannt ist, für die ja kein **GOSUB**-Stack-Speicher errichtet wurde (Fehler: RETURN without GOSUB).Im schlimmsten Fall wird sogar ein Stack-Speicher zum Rücksprung herangezogen, der von einem vorherigen Unterprogramm errichtet und bisher noch nicht mit **RETURN** gelöscht wurde.)

4.3.7.2 Selbstdefinierte Funktionen

Der M 2Ø kennt zahlreiche Standardfunktionen (festverdrahtet).
Zu unterscheiden sind

a) numerische Funktionen und
b) String-Funktionen

zu a): Numerische Funktionen liefern als Ergebnis stets einen numerischen Wert , unabhängig vom Typ des Arguments bzw. der Argumente, die in die Funktion eingereicht werden.

Beispiele: **INT**, **LEN**, **INSTR**.

zu b): String-Funktionen liefern als Ergebnis stets einen String, unabhängig vom Typ des Arguments bzw. der Argumente, die in die Funktion eingereicht werden. String- Funktionen sind stets am **$**-Zeichen am Ende des Funktionsnamens zu erkennen.

Beispiele: **STRING$**, **CHR$**, Funktion **MID$**.

Neben diesen Standardfunktionen besteht die Möglichkeit, öfter auftretende Operationen als selbstdefinierte Funktionen im Programm selbst zu definieren und diese unter ihrem bei der Definition vergebenen Namen aufzurufen. Das Argument/die Argumente der Funktion können von beliebigem Typ sein; Typ und Anzahl der Argumente werden in der Definition festgelegt und müssen beim Aufruf der Funktion mit der Definition übereinstimmen.

Beispiel (selbstdefinierte numerische Funktion):

```
10 DEFDBL A,M
20 DEF FNMITTELWERT(ARG1,ARG2)=(ARG1+ARG2)/2
30 M1=FNMITTELWERT(3,5)
40 LPRINT "Der Mittelwert aus";3;"und";5;"ist";M1
50 INPUT "Wert 3, Wert 4";W3#,W4#
60 M2=FNMITTELWERT(W3#,W4#)
65 LPRINT "Der Mittelwert aus";W3#;"und";W4#;"ist";M2
70 LPRINT "Der Mittelwert aus 3, 5,";W3#;"und";W4#;"ist";FNMITTE
LWERT(M1,M2)
999 END

Der Mittelwert aus 3 und 5 ist 4
Der Mittelwert aus 12,5 und 18 ist 15,25
Der Mittelwert aus 3, 5, 12,5 und 18 ist 9,625
```

Der Name der selbstdefinierten Funktion ist **FN**Mittelwert. **FN** ist einleitender Bestandteil einer selbstdefinierten Funktion und darf deshalb nicht am Anfang von Variablennamen verwendet werden! Mit **FN**Mittelwert unter Angabe der aktuellen Argumente (in Klammern) kann jederzeit die unter **DEF FN**Mittelwert rechts vom =-Zeichen vorher einmal im Programm definierte Operation abgerufen werden und steht als Funktionsergebnis zur Verfügung. Besonders wichtig ist die Tatsache, daß die in der Liste der Argumente bei **DEF FN** verwendeten Variablennamen nicht den Inhalt gleichnamiger Variablen im Hauptprogramm verändern (sog. 'lokale' Variablen)!!

Selbstdefinierte Funktionen können nur einzeilig sein, das heißt, die mit den Argumenten der Liste vorzunehmenden Operationen müssen hinter dem Gleichheitszeichen in Form eines Ausdrucks vorgenommen werden. Rechts vom =-Zeichen können zusätzlich zu den in der Argumente-Liste verwendeten Argumenten noch (Konstanten oder) Variablen verwendet werden, die für das Gesamtprogramm gelten (sog. 'globale' Variablen).

Beispiel (selbstdefinierte numerische Funktion mit globaler Variable):

```
10 REM  Rundungsroutine; kfm, Rundung auf die n-te Nachkomma-
        stelle
15 DEF FNRUNDE#(X#)=FIX(X#*10 NK.STELLEN%+.5*SGN(X#))/10^NK.STEL
LEN%
20 INPUT "gewuenschte Anzahl NK-Stellen";NK.STELLEN%
25 LPRINT "Es wird auf";NK.STELLEN%;"Nachkommastellen kaufmaenni
sch gerundet,"
26 LPRINT:LPRINT
30 INPUT "zu rundende Zahl";ZRZAHL#
40 LPRINT ZRZAHL#,FNRUNDE#(ZRZAHL#)
50 GOTO 30

Es wird auf 2 Nachkommastellen kaufmaennisch gerundet,


 1.3284          1.33
-1.2384         -1.24
 456.725         456.73
 .344            .34
-.344           -.34
```

Beispiel (selbstdefinierte numerische Funktion):

```
' REM Berechnung der Anfangsposition zum Zentrieren eines
      Strings in ein Feld von F% Zeichen Laenge
'
'
'
90 DEF FNSTARTPOS.FELD%(ZUZENTR.STRING$,F.LAENGE%)=INT((F.LAENGE
%-LEN(ZUZENTR.STRING$))/2)+1
100 INPUT "Feldlaenge";F%
110 IF F%<0 OR F%>64 THEN 100
120 INPUT "zu zentrierender String";S$
130 IF LEN(S$)>F% THEN PRINT "String zu lang":GOTO 120
131 LPRINT S$
135 LPRINT STRING$(F%,42)
140 LPRINT TAB(FNSTARTPOS.FELD%(S$,F%))S$
150 GOTO 100


DEUTSCHE OLIVETTI GmbH
**********************
    DEUTSCHE OLIVETTI GmbH
```

Beispiel (selbstdefinierte String-Funktion):

```
1 REM Ersetzen in einem String1 (S1$) ab Position P0% bis
      Position P9% (incl.) durch String2 (S2$)
10 DEF FNERSETZE$(S1$,S2$,P0%,P9%)=LEFT$(S1$,P0%-1)+S2$+RIGHT$(S
1$,LEN(S1$)-P9%)
15 A$="Konfiguration mit Matrixdrucker":B$="Laser"
16 P%(1)=19 : P%(2)=24
20 LPRINT FNERSETZE$(A$,B$,P%(1),P%(2))
```

```
Konfiguration mit Laserdrucker
```

Beispiel (selbstdefinierte String-Funktion mit Wahrheitswerten):

```
1 ' Aussortieren aller Strings aus einem String-Array T$(), die
    mit einer Postleitzahl aus 4 Ziffern (1000 - 8999) beginnen.
2   Genau eine Auswahl ist möglich, für die Auswahl sind 1 bis 4
    numerische Zeichen einzugeben. Die aussortierten Strings
    werden am Ende mit einem * markiert.
3 ' Die Auswahl geschieht mit Hilfe eines logischen Ausdrucks
    (Wahrheitswertes) in einer selbstdefinierten Funktion (eine
    Zeile!)
4 '
5 DEF FNTSUCHE$(T$,X%)=T$+SPACE$(5)+STRING$(-(VAL(LEFT$(T$,LEN(S
TR$(X%))-1))=X%),"*")
10 DIM T$(20)
20 E$=CHR$(0)
30 WHILE (LEN(E$) AND ZAEHLER%<21)
40       PRINT ZAEHLER%+1;:INPUT "  Eingabe (CR=Ende)";E$
45       IF E$="" THEN 90        ' Löschen des WHILE-WEND-Stack-
                                   Speichers!
50       IF VAL(E$)<1000 OR VAL(E$)>8999 THEN 40
60       ZAEHLER%=ZAEHLER%+1 : T$(ZAEHLER%)=E$
90 WEND
95 INPUT "auszusortierende Postleitzahl";PLZ%
96 IF PLZ%<1 OR PLZ%>8999 THEN 95
99 LPRINT "auszugeben sind alle Orte, die mit";PLZ%;"beginnen:":
LPRINT:LPRINT
100 FOR I%=1 TO ZAEHLER%
110 LPRINT FNTSUCHE$(T$(I%),PLZ%)
120 NEXT
999 END
```

```
auszugeben sind alle Orte, die mit 60 beginnen:


6500 Mainz
6000 Frankfurt      *
6001 bei Frankfurt      *
8000 München
6020 bei Frankfurt      *
1000 Berlin
```

4.3.8 Inputs über Tastatur

Daten können auf mehrere Arten über Tastatur eingegeben werden. Die Positionierung des Bildschirm-Cursors auf die gewünschte Eingabeposition am Bildschirm kann mit der Anweisung **CURSOR** erreicht werden.

1. **INPUT** und **LINE INPUT**

   Mit Hilfe der Anweisung **INPUT** können über Tastatur einer oder mehreren Variablen einer Liste Werte zugewiesen werden. Die einzelnen Werte sind durch Kommas getrennt einzugeben. Dabei ist es nicht möglich, Anführungszeichen zu Bestandteilen von Stringeingaben zu machen. Im String enthaltene Kommas oder führende oder am Schluß stehende Blanks können nur zu Stringbestandteilen werden, wenn die Eingabe im Anführungszeichen eingeschlossen wird.

   Mit Hilfe der Anweisung **LINE INPUT** hingegen kann auf genau eine Stringvariable ein Tastatur-Input gemacht werden, wobei alle Zeichen - auch die oben erwähnten - übernommen werden. Die Eingabe kann danach selbstverständlich auf bestimmte Eigenschaften hin untersucht werden (Trennzeichen, rein numerische Eingabe) und via Stringverarbeitung zerlegt und ggf. in numerische Variablen verwandelt werden.

   Beispiel:

```
10 REM  Folgende Eingabe wird getaetigt:
20 REM      DM 20,70
30 REM  Dabei werden je 4 Blanks vor DM und nach 20,70 getippt
40 INPUT"Eingabe ueber INPUT: ",A$,B$
50 LINE INPUT"Eingabe ueber LINE INPUT: ";C$
60 LPRINT A$,LEN(A$):LPRINT B$,LEN(B$):LPRINT C$,LEN(C$)
70 REM  2. Eingabe: Beide Male wird "    DM 20,70    "
                    eingegeben.
80 INPUT "ueber INPUT: ";E$
90 LINE INPUT "ueber LINE INPUT: ";F$
100 LPRINT E$,LEN(E$):LPRINT F$,LEN(F$)
999 END
```

```
DM 20          5
70             2
   DM 20,70                16
   DM 20,70                16
"   DM 20,70   "            18
```

Es ist besonders zu beachten, daß bei **INPUT** und **LINE INPUT** keine festen Begrenzungen der Eingabefelder möglich ist (keine Festlegung der maximal eingebbaren Zeichen), was dazu führen kann, daß ein gegebener Bildschirmaufbau vom Bediener zerstört werden kann (insbesondere bei Ausnutzung des Tastaturpuffers von 64 Zeichen!).
Ähnliches gilt für numerische Eingaben: Sind diese zu groß, um auf die betreffenden Variablen zugewiesen werden zu können, erscheint die Meldung "Overflow" in der nächsten Zeile. Der Overflow selbst kann in einer Fehlerbehandlungsroutine abgefangen werden (vgl. Kapitel 4.3.12), nicht aber die Meldung.

Bei Eingaben von nicht formatgerechten oder zu wenig Daten bei **INPUT** kann die Meldung "Redo from start?" ebenfalls nicht unterdrückt werden.

Über **INPUT** und **LINE INPUT** angeforderte Eingabe(liste)n können erst nach Auslösen der Eingabe-Abschlußtaste (CR) verarbeitet werden.

2. **INKEY$** und **INPUT$**

Die oben beschriebenen Probleme lassen sich durch Verwendung der Funktionen **INKEY$** und **INPUT$** lösen:

Bei **INKEY$** kann bis zu eine Taste gedrückt werden. Dazu gehören auch CR und [redacted]. Wird im Moment der Funktionsabfrage keine Taste betätigt, gilt dies als Eingabe des Leerstrings (""). Es muß somit in einer Schleife abgefragt werden, wann eine Taste betätigt wurde, da das System nicht abwartet, bis eine Taste betätigt wurde.

Beispiel:

```
 1 REM Eingabe einer bis zu 5stelligen ganzen Zahl
 5 REM Loeschen von Eingabezeichen mit CONTROL H ist hier nicht
       moeglich!
30 FOR I%=1 TO 5
40 K$=INKEY$
50 IF K$="" THEN 40
60 IF ASC(K$)=13 THEN I%=5 : GOTO 90
62 IF K$<"0" OR K$>"9" THEN 40
70 PRINT K$;
80 S$=S$+K$
90 NEXT I%
95 PRINT
100 PRINT VAL(S$)
999 END
```

Bei **INPUT$** muß eine genau festgelegte Anzahl von Tasten (mindestens eine) betätigt werden, wobei auch CR und [redacted] mitzählen. Neben der Tatsache, daß mehr als eine Taste festgelegt werden kann, unterscheidet sich **INPUT$** von **INKEY$** dadurch, daß das System abwartet, bis die vorgeschriebene Anzahl Tasten betätigt wurde und erst dann mit der weiteren Abarbeitung des Programms fortfährt.

Beispiel:

```
1 REM  Eingabe einer bis zu 5stelligen positiven ganzen Zahl
2 CLS : CURSOR(1,2) : PRINT "Eingabe: ";
20 KEINAUSSPRZ=-1
30 WHILE LEN(S$)<5 AND KEINAUSSPRZ
40 K$=INPUT$(1)
60 IF ASC(K$)=13 THEN KEINAUSSPRZ=0:GOTO 90
62 IF ASC(K$)<>8 THEN 66  '  CHR(8) ist CONTROL H
64 IF LEN(S$) THEN S$=LEFT$(S$,LEN(S$)-1) : CURSOR(10,2) : PRINT
   S$
66 IF K$<"0" OR K$>"9" THEN 40
68 CURSOR(10,2)
70 S$=S$+K$
72 PRINT S$
90 WEND
95 PRINT
100 PRINT  VAL(S$)
```

INPUT$ kann auch zum Einlesen von Zeichen aus externen Datenfiles verwendet werden (vgl. Kapitel 4.3.10.2).

3. Sonder-Funktionstasten und Eingabeabschlußtasten

Einige Tasten haben besondere Funktionen, z.B. [redacted], [redacted] oder [redacted] sowie die Eingabeabschlußtasten (CR), von denen es drei gibt:
Die "normale" Eingabeabschlußtaste sowie S1 und S2.

Außerdem existieren die Cursor-Steuertasten (vgl. Kapitel 1.3.5). Diese Tasten steuern den Cursor selber nicht, sondern erzeugen nur bestimmte ISO-Codes. In Abhängigkeit von diesen Codes ist per BASIC-Programm jeweils die betreffende **CURSOR**-Positionierung vorzunehmen (**CURSOR**-Anweisung).

Unter Verwendung der Funktionen **INKEY$** bzw. **INPUT$** lassen sich alle Tasten abfragen.
Hinweis:

Alle drei Eingabeabschlußtasten haben den ISO-Code 13 (CR).
Die ISO-Codes, die den übrigen Tasten zugeordnet sind, können der Tabelle in Kapitel 4.3.17 entnommen werden.

Alle über **INKEY$** bzw. **INPUT$** erzeugten Zeichen werden nicht automatisch am Bildschirm angezeigt und müssen deshalb - falls nötig - mit **PRINT** am Bildschirm ausgegeben werden.

Soll abgefragt werden, welche der drei Eingabeabschlußtasten betätigt wurde, muß nach der betreffenden **INPUT** oder **LINE INPUT**-Anweisung bzw. **INKEY$**- oder **INPUT$**-Funktion die Assembler-Routine **lt** mit Hilfe der Anweisung **CALL** aufgerufen werden. Als Argument in der Klammer ist - nach dem Präfix **%** - eine Integer-Variable anzugeben, die nach Ausführung der **CALL**-Anweisung Aufschluß darüber gibt, mit welcher Abschlußtaste die Eingabe abgeschlossen wurde, bzw. mit welcher Taste das Ergebnis CR(=**CHR$**(13)) bei Anwendung einer der Funktionen **INKEY$** bzw. **INPUT$** erzeugt wurde.
Die Integer-Variable enthält:

Ø bei Abschluß mit ↵

1 bei Abschluß mit S1

2 bei Abschluß mit S2

Beispiel:

```
1000 REM Programmverzweigung in Abhaengigkeit von der jeweiligen
     Abschlusstaste
1001 CLS
1005 PRINT TAB(20)"Bitte S , S1 oder S2 druecken"
1020 K$=INKEY$ : IF K$="" THEN 1020 ELSE IF ASC(K$)<>13 THEN
     1020
1030 CALL "lt"(%TASTE%)
1035 PRINT TASTE%
1050 ON TASTE%+1 GOTO 2000,3000
1060 PRINT "S2" : GOTO 1005     '     S2 wurde gedrueckt
2000 REM   S0
2010 PRINT "S0" : GOTO 1005
3000 REM   S1
3010 PRINT "S1": GOTO 1005
```

Es ist zu beachten, daß die hinter % angeführte Integervariable vor dem ersten Aufruf durch eine wertzuweisende Anweisung einen beliebigen Wert erhalten hat (z.B. im obigen Programm: 1 TASTE%=Ø; im Vereinbarungsteil).

Alle programmgesteuerten Tastatur-Inputs werden rein über Tastatur abgewickelt; eine Übernahme von am Bildschirm angezeigten Daten in den Tastaturpuffer ist nicht möglich.

Sollen am Bildschirm angezeigte Werte über Tastatureingaben des Bedieners geändert werden, müssen sie , falls mit **INPUT** oder **LINE INPUT** gearbeitet wird, vollständig eingegeben werden.

Sollen im Bildschirm angezeigte Daten nur teilweise geändert werden, kann wie folgt vorgegangen werden:

1. angezeigtes, zu änderndes Datum in einer Stringvariablen speichern;
2. jede einzelne Tastenbetätigung abfragen (Zuweisung auf eine Hilfsvariable über **INKEY$** bzw. **INPUT$**);
3. unterscheiden, ob eine Cursor-Steuertaste gedrückt wurde oder CR oder ein echtes Zeichen zur Änderung eingegeben wurde;
4. je nach dem Ergebnis von 3.:
   - den Inhalt der Stringvariablen, in der das zu ändernde Datum steht, entsprechend anpassen (Stringverarbeitung);
   - die geänderte Stringvariable am Bildschirm ausgegeben (nach **CURSOR-**Anweisung);
   - den Cursor auf die erforderliche neue Stelle positionieren;
   - Änderungsroutine verlassen (bei CR) bzw. zurück zu 2.
5. evtl. abfragen, mit welcher Abschlußtaste schließlich der Aussprung aus der Routine 1 - 4 bewirkt wurde.

Die Cursor-Steuertasten besitzen die folgenden ISO-Codes:

Steuertasten	ISO-Code (dezimal)
↑	158
↓	154
→	157
←	155
HOME	156

Zur Übersicht aller ISO-Codes vgl. Tabelle im Kapitel 4.3.17.

4.3.9 Outputs

4.3.9.1 Bildschirm (alphanumerisches Arbeiten)

Die Ausgabe von Tastatureingaben über **INPUT** bzw. **LINE INPUT** erfolgt direkt am Bildschirm. (Diese Ausgaben werden auch von einer **WIDTH**-Anweisung nicht beeinflußt.) Die Ergebnisse der Funktionen **INKEY$** bzw. **INPUT$** werden nicht angezeigt. Darüber hinaus können Daten mit Hilfe der Anweisungen **PRINT** und **WRITE** am Bildschirm angezeigt werden.

Beispiel zu **PRINT** und **WRITE:**

```
5000 REM   PRINT  /   WRITE
5001 CLS
5010 'EINGEGEBEN WIRD  DM 20,70
5020 INPUT "INPUT: ";A$,B
5030 LINE INPUT "LINE INPUT: ";C$
5040 PRINT STRING$(25,"*")
5050 PRINT A$;B;C$
5060 WRITE A$;B;C$
5099 END
```

(Der Unterschied, der sich hier zeigt, ist wesentlich zur Unterscheidung der File-Anweisungen **PRINT#** und **WRITE#** in Kapitel 4.3.10.2). **WRITE** bewirkt, im Gegensatz zu **PRINT**, die automatische Einfügung eines Trennzeichens (,) zwischen den auszugebenden Daten. Darüber hinaus werden alle Strings automatisch dadurch gekennzeichnet, daß sie in Anführungszeichen (") eingeschlossen werden. Die Anführungszeichen werden jedoch nicht zu eigentlichen Bestandteilen der Strings.

Bei Verwendung von **PRINT** ist es möglich, eine Zeilenschaltung (LF) dadurch zu unterdrücken, daß am Ende der Liste der Ausgabedaten ein Strichpunkt (;) gesetzt wird.

Eine Formatierung der Ausgaben, die weitergeht als mit **TAB**, (z.B. rechtsbündige Darstellung von numerischen Daten) ist möglich entweder durch Stringverarbeitung oder aber durch Verwendung des Anweisungsbestandteils **USING** (vgl. Ende dieses Kapitels).

Durch Ausgabe des ISO-Codes 7 über den Bildschirm (**PRINT CHR$(7)**) kann ein akustisches Signal erzeugt werden. Steht am Ende einer solchen Anweisung kein **;**, so bewirkt sie natürlich noch zusätzlich eine Zeilenschaltung (LF).

Beim Arbeiten mit Texten (alphanumerisches Arbeiten) ist der Bildschirm - in Abhängigkeit von im PCOS-Befehl **ss** gesetzten Default-Werten - wie folgt aufgeteilt:

Version 1:	64x16 (16 Textteilen à 64 Spalten)
Version 2:	80x25 (25 Textzeilen à 80 Spalten)

Die Darstellung der Zeichen basiert auf der Einteilung des Bildschirms in ein Punktraster von Elementarpunkten.

Bei Version 1 liegen 512 Punktspalten und 256 Punktzeilen zugrunde, bei Version 2 hingegen 480 Punktspalten und 256 Punktzeilen.

Mit der **WIDTH**-Anweisung kann die Ausgabebreite auf dem Bildschirm (Anzahl Textspalten) festgelegt werden. Die **WIDTH**-Anweisung wird weder durch **NEW** noch durch **RUN** oder **CHAIN** aufgehoben, sondern nur durch eine neue **WIDTH**-Anweisung! Außerdem läßt sich der Bildschirm per Programm in bis zu 16 verschiedene rechteckige Windows einteilen, von denen jedes einzelne getrennt angesprochen werden kann. Jedes einzelne Window kann mit bestimmten Attributen (Zeilenhöhe, Spaltenbreite, Hinter- und Vordergrundfarbe) ausgestattet werden. Jedes Window hat einen selbständigen Text-Cursor mit einer eigenen Spalten-Zeilen Adressierung bzw. einen eigenen Graphik-Cursor und eine eigene Skalierung. Zur Arbeit mit Windows vgl. Kapitel 4.3.18; zur graphischen Arbeit vgl. Kapitel 4.3.14. Bei der Arbeit mit Windows ist zu beachten:

- Nur die Anweisungen **CLOSE WINDOW** oder **CLEAR** lösen eine vorherige Aufteilung des Bildschirms in Windows auf.
- Auch für Windows werden Default-Werte durch PCOS-Befehle gesetzt und zwar durch **sb** und **ss**.

Mit der Anweisung **CURSOR** kann der Text-Cursor auf bestimmte Textspalten und -zeilen innerhalb des gesamten Bildschirms (Window-Nr. 1) oder innerhalb eines Windows gesetzt werden.

Beispiel:

```
0 CLEAR
10 W%=WINDOW(0,150)
20 INPUT "Window-Nr.(1-2), Zeichenbreite (6 oder 8) ",W%,Z%
21 IF W%<1 OR W%>2 OR Z%<>6 AND Z%<>8 THEN 20
24  WINDOW % W% : CURSOR (1,8)
25 INPUT "Druckposition Window 1, D.P. Window 2";D%(1),D%(2)
26 IF D%(1)<0 OR D%(2)<0 THEN 20
30 WINDOW% W% : W%=WINDOW(0,0,,Z%)
40 FOR I%=1 TO 6
45 CURSOR (D%(W%),I%)
50 PRINT "Dies ist Window Nr.";W%;"Ausgabe ab Spalte";D%(W%);
60 NEXT I%
62 PRINT
65 CURSOR (1,8)
70 GOTO 20
```

Zur Positionierung des Graphik-Cursors vgl. Kapitel 4.3.14.

Es ist zu beachten, daß bei einer Window-Aufteilung des Bildschirms die Text-Cursor-Positionen sich auf das jeweilige Window beziehen. Jedes Window kann somit als ein "eigener kleiner Bildschirm" interpretiert werden.

Die Position des Text-Cursors ist stets in der Form (Textspalte,Textzeile) anzugeben. Die oberste Zeile des Bildschirms (bzw. des betreffenden Windows) hat die Nr. 1, die Spalte am weitesten links ebenfalls die Nr. 1. Tiefere bzw. weiter rechts liegende Positionen haben entsprechend höhere Nummern.

Mit Hilfe der Funktion **POS** kann die aktuelle Position (Zeile oder Spalte) des Text-Cursors im Bildschirm bzw. in einem Window abgefragt werden.

Alle Ausgaben, deren Länge den rechten Rand des Bildschirms bzw. eines Windows überschreiten, werden automatisch in der nächsten Zeile ab dem linken Rand fortgesetzt.

Zur Berechnung der zu reservierenden Ausgabepositionen bei nicht über **USING** formatierten numerischen Daten ist das Standardformat zugrundezulegen.

Dabei gilt:

1. Kommazahlen werden stets mit einem Dezimalpunkt ausgegeben.
2. Bei Zahlen im Intervall $-1 < x < 1$ wird keine Ø vor dem Dezimalpunkt ausgegeben.
3. Nullen nach dem Dezimalpunkt werden nie ausgegeben, falls auf sie keine anderen Ziffern mehr folgen.
4. Zahlen, die zu viele signifikante Stellen für ihren Typ haben, werden automatisch im Gleitkomma- (bzw. Gleitpunkt-) Format ausgegeben also in der Form

   $\left\{\begin{matrix}-\\ \\ t\end{matrix}\right\}$ Mantisse $\left\{\begin{matrix}E\\ \\ D\end{matrix}\right\}\left\{\begin{matrix}-\\ \\ +\end{matrix}\right\}$ Hochzahl zur Basis 10, mit der die Mantisse multipliziert werden muß

   Beispiel:

```
0 CLS: WIDTH LPRINT 60:LPRINT STRING$(60,"*")
10 LINE INPUT"numerische Eingabe ";A$
15 LPRINT A$;VAL(A$):GOTO 10
************************************************************
1234567890.123456 1234567890.12346
12345678901234567890123456789O 1.23456789012346D+29
.123456E-28 1.234560-29
```

5. Hinter der letzten ausgegebenen Ziffer wird grundsätzlich zusätzlich ein Blank ausgegeben.

Werden numerische Daten mit der Funktion **STR$** konvertiert, gilt für das Ergebnis das eben Gesagte, mit der Ausnahme, daß 5. entfällt!

Werden am Bildschirm Strings ausgegeben, die nicht druckbare Zeichen (ISO-Codes Ø bis 31) enthalten, so werden diese übergangen (nicht angezeigt), die auf sie folgenden druckbaren Zeichen werden unmittelbar darauf folgend ausgegeben. Die Länge der Anzeige weicht dann von der aktuellen Länge des Strings (**LEN**) ab.
Einige dieser ISO-Codes haben jedoch Wirkung am Bildschirm:

ISO-Code	Wirkung
7	akustisches Signal (ohne Zeichenausgabe)
8	Löschen des vorherigen Zeichens und Positionierung des Cursors um eine Stelle nach links (H)
9	Erzeugung von 7 Blanks
10	Erzeugung von soviel Blanks, wie die Ausgabebreite des Bildschirms aktuell in Zeichen festgelegt ist (LF)
12	Löschen Bildschirm bzw. Window und Positionierung des Cursors auf Position (1,1)
13	CR (Zeilenrücklauf)

Die ISO-Codes über 127 erzeugen besondere Zeichen am Bildschirm, die auch tatsächlich ausgegeben werden.

Beispiel:

```
0 CLS : FOR I%=1 TO 10
6 L$=INPUT$(3)

7 IF VAL(L$)=0 OR VAL(L$)>255 THEN 6 ELSE CURSOR (1+4*I%,1)
 : PRINT VAL(L$) : CURSOR(2+4*I%,2) : PRINT CHR$(VAL(L$))
10 K$=K$+CHR$(VAL(L$))
15 NEXT : PRINT K$ : L$=INPUT$(1)
20 CALL "sp" 'nur PR2400 oder PR1450+Plot-Option
      125 132 214 187 195 7    2    64   82   244

      ü    °    3    à    ...          §    R    Q

ü°3à §RQ
```

Zur Formatierung von Daten (numerisch oder String) dient der Anweisungsbestandteil **USING**. Dabei ist in einem Formatstring, der sich aus bestimmten "Platzhalter"symbolen zusammensetzt, vorzuschreiben, wie die Daten zu formatieren sind (Definition der "Masken"). Die zu formatierenden Daten sind am Ende der Anweisung in einer Liste auszugeben.

Beispiel:

```
10 REM  LPRINT USING
15 WIDTH LPRINT 79
16 LPRINT STRING$(79,"*")
20 A#=123.456 : B#=-123.456   C#=.456 : D#=1.5752350+22
30 A$="Alpha-Feld 1 " : B$="2. Alpha-Feld"
40 LPRINT USING "######.#### = a_# und ######.#### = b_#";A#;B#
50 LPRINT STRING$(79,"*")
60 LPRINT USING "###.# = a_# und ###.# = b_#";A#;B#
70 LPRINT STRING$(79,"*")
80 LPRINT USING "#,###,###.## = a_# und ######.######^^^^ = d_#";A#;D#;
95 REM                                  ************
100 LPRINT USING"+#.###### = c_# und ö          ö = b_$";C#;B$
101 INPUT "Wieviele Blanks sollen in das variable Feld eingefuegt werden    ";N
%
102 IF N%<0 OR N%>35 THEN 101
103 LPRINT USING "-----&-----";STRING$(69,"*")
105 REM                               ******************
110 LPRINT USING "12345678901234560                  ö67890&Dies Feld ist variabe
l lang, hier: ## Zeichen";A$;SPACE$(N%);N%
```

```
**************************************************************************
  123.4560 = a# und  -123.4560 = b#
**************************************************************************
123.5 = a# und %-123.5 = b#
**************************************************************************
      123.46 = a# und  1575.23500D+019 = d# +.456000 = c# und 2. Alpha-Fel = b$
-----**********************************************************************-----
12345678901234560Alpha-Feld 1      67890              Dies Feld ist variabel la
ng, hier: 15 Zeichen
```

Eine Liste zu formatierender Daten kann durch ; abgeschlossen sein, was die Folgeausgabe ab der nächsten freien Position bewirkt.

Der Formatstring kann auch mit Hilfe von Variablen zusammengesetzt sein (z.B. über Stringverarbeitung oder mit Hilfe von selbstdefinierten String-funktionen).

Beispiel:

```
1 INPUT "Artikelnummer (5stellig), Bezeichnung (bis zu 25 Zeichen),Lagernummer
(1-10)  ",ART$,BEZ$,LNR%
2 REM Pruefung entfaellt hier
10 LINE INPUT "Bitte Maske angeben  ";MASKE$
20 REM  Pruefung entfaellt hier
30 LPRINT MASKE$
40 LPRINT ART$,"/",BEZ$;"/";LNR%
50 LPRINT USING MASKE$;ART$;BEZ$;LNR%
99 END


Art-Nr. ß   ß : $ (Lager ###)
E605/605. Artikel/ 14
Art-Nr. E605   : 605. Artikel (Lager  14)
```

Außerdem kann derselbe Formatstring gleichzeitig zur Ausgabe mehrerer gleich zu formatierender Daten benutzt werden.

Beispiel:

```
0 CLS: DATA 100,5,97,603,200,25,60,138,85,50,97,746
10 MASKE$="###,## <--> ##,### * "
15 FOR K%=1 TO 3 : READ I(K%),J(K%):NEXT K%
20 PRINT USING MASKE$;I(1),J(1),I(2),J(2),I(3),J(3)
30 EXEC "sp"    ' nur PR2400
100,50 <--> 97,603 * 200,25 <--> 60,138 *  85,50 <--> 97,746 *
```

4.3.9.2 Drucker

Daten können auch am Drucker ausgegeben werden. Es ist dann statt **PRINT** die Anweisung **LPRINT** und statt **PRINT USING** die Anweisung **LPRINT USING** anzuwenden.
Die am Drucker auszugebenden Zeichen können sich von denen am Bildschirm unterscheiden, falls der Zeichensatz des Druckers von dem des Bildschirms abweicht. Die Art des Drucks (Fettdruck, 10-Zoll-Druck etc.) sowie Vorschriften über Zeilenschaltung u.ä. können über die Ausgabe von Steuerzeichen (vgl. ANHANG II) mit **LPRINT** vorgeschrieben werden.

Beispiel:

```
1 REM      NUR PR1450
2 CLS:T$="Dieser Text ist auszugeben"
8 A$=MKS$(150):LPRINT "A=150    MKS$(A) wird auf Printer
ausgegeben als *";A$;"*","Laenge von MKS$(A)=";LEN(A$),"(
!!!!)"
9 REM Dies zeigt, dass ueber MKS$,MKI$,MKD$ gewonnene Str
ings zwar die korrekte Laenge haben, aber NICHTS sinnvoll
 drukcbares beinhalten!
10 LPRINT T$;  'bleibt wegen ; im Druckpuffer!
15 PRINT "Bitte CR druecken"
20 K$=INKEY$ : IF K$="" THEN 20 ELSE LPRINT CHR$(10);
30 LPRINT "Nach diesem Text erfolgt ein Seitenvorschub"+C
HR$(12)
```

```
A=150    MKS$(A) wird auf Printer ausgegeben als *CM*  Laenge von MKS$(A)= 4      (!!!!)
Dieser Text ist auszugeben
Nach diesem Text erfolgt ein Seitenvorschub
```

Es kann über eine Hardware-Schaltung festgelegt werden, ob die übrigen nicht druckbaren Zeichen in Form von ASCII ausgegeben werden sollen oder ihre Ausgabe (wie am Bildschirm) übergangen werden soll.

Zu beachten ist, daß auch für den Drucker mit Hilfe der Anweisung **WIDTH LPRINT** die Zeilenbreite per Programm festgelegt werden kann, die wiederum mit **NEW**, **RUN** oder **CHAIN** nicht aufgehoben werden kann.

Eine Hardcopy (Abbildung des aktuellen Bildschirmtextes auf dem Drucker) kann mit Hilfe der Anweisungen
CALL "sp" oder **EXEC "sp"**
erzeugt werden.

Mit dem PCOS-Befehl **sf** werden verschiedene Default-Werte für den Drucker gesetzt.

4.3.10 externe Datenfiles

Neben den Programmfiles und ASCII-Files (vgl. BASIC-Befehl **SAVE**) stehen mehrere Arten von Datenfiles zur Verfügung.

Datenfiles können mit dem PCOS-Befehl **fn** angelegt werden. Die dabei gewünschte Größe (Anzahl dafür zu reservierende Sektoren auf Diskette) ist dabei anzugegeben.
Es ist aber auch möglich, Datenfiles nur mit Hilfe der BASIC-Anweisung **OPEN** (z.B. im Programm) anzulegen. Die Anzahl der Sektoren, die dabei reserviert werden, ist abhängig von der im PCOS-Befehl **ss** definierten Anzahl Sektoren.
Wird es im Verlauf von Programmen notwendig, die physische Länge von Datenfiles zu vergrößern, so erledigt dies das System selbsttätig. Voraussetzung ist allerdings, daß noch genügend Platz auf Diskette frei ist. Die Anzahl der Sektoren, um die das System versucht, das File physisch zu vergrößern, ist ebenfalls gleich dem im PCOS-Befehl **ss** gesetzten Parameter. Ist eine Vergrößerung um diese Anzahl Sektoren nicht möglich, wird "Disk full" gemeldet, selbst wenn für die abzuspeichernde Information der Platz noch ausreichen würde.

Optimale Blockungsfaktoren brauchen nicht errechnet zu werden, da die logischen Records von den Grenzen der physischen Sektoren oder Spuren in keiner Weise beeinflußt werden. Es ist also möglich, beliebige Record-Längen festzulegen, ohne dabei Speicherplatzverluste auf Diskette befürchten zu müssen. Die maximale Anzahl von Files, die in einem Programm gleichzeitig verwaltet werden können, ist 15. Die Recordlänge von sequentiellen Files ist - von der Disketten-Kapazität abgesehen - nicht von Bedeutung. Die Record-Länge von Random-Files ist ebenfalls unbegrenzt, von der Größe der für sie erforderlichen, im Arbeitsspeicher zu reservierenden Puffer abgesehen.

Die maximale File-Anzahl in jedem Programm und die maximale Record-Länge der Random-Files wird mit dem PCOS-Befehl **sb** festgelegt. Es ist zu beachten, daß bei einer in **sb** definierten maximalen Record-Länge sofort für jedes File ein eigener Puffer in der angegebenen Länge reserviert wird (Kapazität des Arbeitsspeichers!).

Im Arbeitsspeicher werden sowohl für alle angesprochenen Disketten als auch für alle geöffneten Files Tabellen verwaltet, in denen der Zustand der Disketten bzw. Files festgehalten ist.
Das System prüft bei geöffneten Files in regelmäßigen Abständen, ob die Diskette gewechselt worden ist (ca. alle 3 Sekunden). Wird dies festgestellt, wird sofort der Fehler "Illegal disk change" gemeldet (Fehler-Code 74). Wird die betroffene Diskette wieder eingelegt, kann weitergearbeitet werden.

Nach dem Öffnen von Files wird kein Byte auf der Diskette gesetzt, das das File als physisch offen kennzeichnet.

4.3.10.1 Organisationsmöglichkeiten

Folgende File-Typen stehen als externe Datenfiles zur Verfügung

1. in Form von ASCII-Files abgespeicherte, mit Zeilennummer versehene Textzeilen;
2. Sequentielle Datenfiles mit fester oder variabler Record-Länge
3. Random-Files mit fester Record-Länge, deren Records unter ihrer Nummer aufgerufen werden (sog. Direct-Files);
4. nur bei Einsatz des Programmpakets ISAM in Assembler-Version: Index-sequentielle Files mit folgenden Möglichkeiten:
   - Länge des alphanumerischen Zugriffskriteriums (Key) bis 255 Zeichen
   - Sequentielles Lesen in logischer Folge
   - automatische Suche des nächsten Records
   - Löschungsmöglichkeit des Records und/oder des Keys (ohne Reaktivierungsmöglichkeit)
   - partieller Key
   - sekundärer Key
   - doppelte Keys auf Wunsch

1.: ASCII-Files

ASCII-Files sind wie sequentielle Files zu behandeln. Sie können allerdings nur gelesen werden (Speichern mit **SAVE, A**). Am Ende des Records (mit Zeilennummer versehene Zeile) werden beim Speichern vom System automatisch Trennzeichen gesetzt, so daß durch **LINE INPUT#** jede 'Zeilennummer' automatisch von der nächsten unterschieden wird. Somit kann sehr einfach jede Zeile einer Stringvariablen zugewiesen werden.

2.: sequentielle Files

In sequentiellen Files sind die Daten lückenlos aufeinanderfolgend gespeichert. Die Datenelemente müssen, um identifiziert werden zu können, durch Trennzeichen voneinander getrennt geschrieben werden. Sequentielle Files können vom Programm in jeweils einem der folgenden Arbeitsmodi verwendet werden:

a) Lesen von Anfang an (Input-Mode, Lese-Mode)
b) Schreiben von Anfang an (Output-Mode, Schreib-Mode)
c) Schreiben anschließend an bestehenden Inhalt (Append-Mode)

Beim Öffnen des Files wird festgelegt, in welchem dieser Modi gearbeitet wird. Eine Änderung ist über Schließen des Files und neuerliches Öffnen im gewünschten Modus möglich.

Während ein sequentielles File geöffnet ist, wird vom System ein Pointer für dieses File geführt, der entsprechend dem Modus auf die Stelle im File zeigt, ab der beim nächsten Zugriff gelesen bzw. geschrieben wird. Eine frei wählbare Änderung der Pointerposition innerhalb des Files ist beim Schreiben nicht und beim Lesen nur durch Setzen an den Anfang mit Schließen und neuerlichem Öffnen möglich.

Zu a) Lesen des sequentiellen Files (Input-Mode)

Nach dem Öffnen des Files zeigt der Pointer auf das erste Datenelement im File. Die Daten können in der Reihenfolge ihrer Speicherung entsprechend ihrem Typ gelesen werden.
Das Zurücksetzen des Pointers an die erste Stelle im File ist möglich.

zu b) Schreiben in sequentielle Files von Beginn an (Output-Mode)

Nach dem Öffnen des Files wird der Pointer an die erste Stelle im File gesetzt und der bisher bestehende Inhalt dadurch nicht mehr verfügbar. Beim Schreiben von Datenelementen wird jedes Element an das vorangehende angeschlossen und der Pointer an die nächste freie Stelle gesetzt.

zu c) Anhängen an bestehende Daten (Append-Mode)

Dieser Arbeitsmodus erlaubt das Hinzufügen von Daten an ein bestehendes File. Es wird nach dem Öffnen der Pointer an die erste freie Stelle im File gesetzt und anschließend wie im Schreibmodus verfahren.

3.: Random-Files

Random-Files erlauben den direkten Zugriff auf Daten. Um diesen direkten Zugriff zu ermöglichen, müssen die Daten in Records logisch organisiert werden. Records eines Files müssen die gleiche Länge haben und bilden die Einheit, über die vom Programm auf Daten der Diskette zugegriffen werden kann.
Der Zugriff erfolgt auf gleiche Weise, unabhängig davon, ob nachfolgend Daten gelesen oder verändert werden sollen. Der Zugriff auf einen Record des Files erfolgt über Angabe der Filenummer und die Ordnungsnummer des Records innerhalb des Files.
Innerhalb des Programmes wird ein Record durch eine Feldvereinbarung beschrieben (**FIELD**-Anweisung). Diese Feldvereinbarungen können innerhalb des Programmes verändert werden. Durch eine neue Feldvereinbarung für einen bestimmten Random-File-Puffer wird die zuletzt gültige aufgehoben. Unterschiedliche Feldvereinbarungen für das gleiche File sind also möglich. Bei unterschiedlicher Recordstruktur im gleichen File kann durch eine Füllvariable ("Filler") eine Anpassung auf einheitliche Recordlänge erreicht werden. Wurde auf einen Record zugegriffen, so stehen seine Daten in den Feldvariablen der für das File zuletzt gültigen **FIELD**-Anweisung zur Verfügung und können vom Programm weiterverarbeitet werden. Beim Schreiben in ein Random-File müssen zunächst die Feldvariablen im Random-File-Puffer die entsprechenden Werte enthalten. Anschließend kann der Inhalt des Random-File-Puffers unter einer Ordnungsnummer als Record in das File geschrieben werden.

Die Angabe der Ordnungsnummer beim Lesen oder Schreiben kann unterbleiben. In diesem Fall wird der Record im File gelesen bzw. geschrieben, dessen Ordnungsnummer auf die zuletzt - durch Lesen oder Schreiben - angesprochene Ordnungsnummer folgt.

Haben die Records eines Random-Files einen Aufbau wie Datenelemente eines sequentiellen Files (Trennzeichen!) ist es möglich, Random-Files als sequentielle Files zu bearbeiten.

4.: Index-sequentielle Files

In verschiedenen Anwendungen ist es nötig, häufig auf Records zuzugreifen, die durch ein Zugriffskriterium gekennzeichnet sind, das nicht die Record-Nummer ist.

Beispiel:
Die Artikel einer Artikeldatei sind mit einer alphanumerischen Artikelnummer versehen, aufgrund derer auf die benötigten Records zugegriffen wird. Es soll außerdem möglich sein, die Artikel jederzeit - ohne größeren vorherigen Sortierlauf - sofort sequentiell so auszulisten, wie sie aufgrund der logischen Reihenfolge der Artikelnummern vorhanden sind.

4.3.10.2 Befehle, Anweisungen und Funktionen

1. Befehle zum Arbeiten mit Datenfiles

Mit dem Befehl **NAME** kann ein auf Diskette vorhandenes Datenfile umbenannt werden, mit **KILL** kann es gelöscht werden (der Platz auf Diskette wird wieder frei).
Mit **FILES** kann das Inhaltsverzeichnis einer Diskette auf dem Bildschirm angezeigt werden.
Mit dem PCOS-Befehl **sb** wird die Anzahl von Datenfiles festgelegt, die sich in einem Programm gleichzeitig verwalten lassen. Außerdem wird angegeben, bis zu welcher maximalen Recordlänge bei Random-Files gearbeitet werden kann. Für alle geöffneten Random-Files wird automatisch ein eigener Random-File-Puffer in dieser Größe im Arbeitsspeicher reserviert.
Mit dem PCOS-Befehl **fn** kann ein File auf Diskette angelegt werden (Reservierung einer bestimmten Anzahl von Sektoren zu 256 Bytes auf Diskette). Mit den PCOS-Befehlen **fp** und **fd** können File-Passwords vergeben bzw. wieder gelöscht werden.
Mit **fc** können Datenfiles kopiert werden, mit **fw** mit Schreibschutz versehen. Mit **fu** kann ein Schreibschutz wieder aufgehoben werden.

2. Öffnen und Schließen von Datenfiles

Alle Datenfiles müssen mit der Anweisung **OPEN** auf Zugriffe vorbereitet werden. Dabei ist für jedes geöffnete File eine Nummer zu vergeben, unter der das File von da an bei allen Befehlen anzusprechen ist.
Ist das betreffende File unter dem 'Filenamen' noch nicht angelegt, so erledigt dies **OPEN** ebenfalls.
Mit der Anweisung **CLOSE** können bestimmte offene Files oder alle aktuell offenen Files geschlossen werden. Die Anweisung **END** sowie der Befehl **NEW** schließt alle offenen Files.

Es dürfen mit **OPEN** nur Files geöffnet werden, wenn

a) Die 'Filenr.' noch nicht aufgrund einer vorherigen **OPEN**-Anweisung vergeben ist, es sei denn, die vergebene Filenummer wurde vorher mit **CLOSE** wieder freigegeben;

b) die 'Filenr.' im Rahmen der mit **sb** gesetzten maximalen Anzahl liegt;

c) der 'Filename' vorher nicht in einer **OPEN**-Anweisung verwendet wurde, die noch nicht durch **CLOSE** aufgehoben ist;

d) (falls das File noch nicht angelegt ist) noch ausreichend Platz auf Diskette ist, um das File in soviel Sektoren anzulegen, wie durch **ss** festgelegt.

3. Arbeit mit Random-Files

Random-Files sind unter einer Filenummer mit Angabe des Parameters "R" für 'Zugriffsart' und immer unter Angabe der 'Record-Länge' des Random-Files zu öffnen.
Anschließend wird in einer **FIELD**-Anweisung für die betreffende 'Filenr.' die logische Struktur des Records festgelegt, mit der von da an gearbeitet werden soll. Die Felder sind ausschließlich in Form von Stringvariablen zu definieren (feste Feldlängen).
- Für Strings gilt:
  Für jedes Zeichen ist ein Byte zu reservieren;
- Für numerische Werte gilt:
  Für Integer-Werte ist ein String mit zwei Bytes zu reservieren;
  für jeden einfach genauen Wert ein String mit vier Bytes;
  für jeden doppelt genauen Wert ein String mit acht Bytes.

a) Lesen eines Records in den Random-File-Puffer

In der Anweisung **GET** ist die Filenr. und die Nummer des einzulesenden Records zu spezifizieren. Nach Durchführung der **GET**-Anweisung stehen in den Feldvariablen der betreffenden **FIELD**-Anweisung die Feldinhalte aus dem File im Random-File-Puffer zur Verfügung. Die Variableninhalte können ausgegeben werden, z.B. durch **PRINT**. Sollen diese Variableninhalte später wieder auf das File gebracht werden (z.B. nach Veränderung), so dürfen dazu nicht die Feldvariablen selbst Zuweisungen erhalten (außer durch **LSET** bzw. **RSET**). Ebensowenig dürfen sie als Argument in Funktionen oder innerhalb von Vergleichsausdrücken verwendet werden!!

b) Schreiben eines Records auf das File

Zuerst ist auf alle Feldvariablen eine Wertzuweisung durchzuführen. Diese darf nur über die Anweisungen **LSET** oder **RSET** erfolgen! Diese Anweisungen bringen die zugewiesenen Inhalte automatisch an die durch **FIELD** definierte Position des Feldes in den Random-File-Puffer.
Numerische Daten sind - je nach Typ - mit Hilfe der Funktionen **MKI$**, **MKS$** bzw. **MKD$** in Strings zu verwandeln.
Nach dem **LSET** bzw. **RSET** ist nur der Random-File-Puffer gefüllt. Mit Hilfe der **PUT**-Anweisung kann nun der Inhalt des Random-File-Puffers auf das File gebracht werden, wofür die betreffende 'Filenr.' und die Nummer des gewünschten Records anzugeben ist.

Bemerkungen:

- Sowohl beim Lesen als auch beim Schreiben kann die Angabe der Record-Nr. unterbleiben. Es wird dann der Record bearbeitet, der dem zuletzt gelesenen oder geschriebenen folgt.
- Wurden Records auf dem File noch nicht beschrieben, können sie trotzdem gelesen werden. Jedes Byte dieser Records ist mit dem Zeichen **CHR$(Ø)** belegt.
- Unterscheidet sich die Summe der Längen aller Feldvariablen in der **FIELD**-Anweisung von der Recordlänge, sind zwei Fälle zu unterscheiden:
  a) die Feldvereinbarung ist zu niedrig ausgelegt: Nur die nicht belegten Zeichen in den definierten Feldvariablen werden beim Schreiben des Records wie üblich mit Blanks belegt; der nicht von der Feldvereinbarung betroffene Rest des Records (rechter Teil) auf dem File bleibt unverändert.
  b) die Feldvereinbarung ist zu groß: Es wird der Fehler "FIELD overflow" (Fehler-Code 5Ø) gemeldet.

Beispiel zum Random-File:

```
200 OPEN "R",#1,"1:randfile",38 : GOTO 15000
2000  ' Arbeiten mit dem Random-File "randfile" auf Station 1
2006  ' Die Personal-Nummer entspricht der Record-Nr.
2007  ' Es handelt sich um ein Neuerfassungsprogramm, nur noch n
icht belegte Personal-Nummern sollen aufs File gebracht werden.
2010  ' Folgender Record-Aufbau liegt zugrunde:
2020  ' von  bis   I Laenge  I Progr.var.  I Var.Typ I Feldvar.
2030  '  incl.,    Iin BytesI              I         I
2040  '  1    4    I    4    I PERS.NR!    I single  I PNR$
2050  '  5    24   I    20   I GES.NAME$   I string  I N$
2060  '  25   32   I    8    I LOHN#       I double  I LO$
2070  '  33   34   I    2    I ST.KLASSE%  I integer I KL$
2080  '  35   38   I    4    I  (offen)    I (offen) I FILLER$
2090 REM DIE PERSONAL-NR. MUESSTE NICHT AUF DAS FILE GESCHRIEBEN
WERDEN, DA SIE DER RECORD-NUMMER ENTSPRICHT.
2096 REM FELDVEREINBARUNGEN
2097 REM a) mit Einzelvariablen
2100 FIELD #1,4 AS PNR$,20 AS N$,8 AS LO$,2 AS KL$,4 AS FILLER$
2110 RETURN
2120 REM b) mit Feldvariable fuer alle Felder

2130 FIELD #1,38 AS GESFELD.RANDFILE$ : RETURN
7000 REM GESAMTER RECORD IN EINER FELDVARIABLEN
7010 GOSUB 2120
7020 REM WIE 15030-15070
7030 REM WENN NICHT VORHANDEN, IST GESFELD.RANDFILE$ MIT 38 x
     CHR$(0) BELEGT
7040 IF GESFELD.RANDFILE$<>STRING$(38,0) THEN PRINT "Vorhanden"
7045 REM  Da Programmvariablen vom Typ String immer nur von
          aktueller Laenge sind, wird es hier noetig, die String
          variable GES.NAME$ zur Laenge von 20 aufzufuellen
          (hier: mit Blanks, rechtsbuendig)
7050 LSET GESFELD.RANDFILE$=MKS$(PERS.NR!)+GES.NAME$+SPACE$(20-L
EN(GES.NAME$))+MKD$(LOHN#)+MKI$(ST.KLASSE%)
7060  ' Der Filler wird automatisch mit Blanks belegt
7070 REM WIE 15140
7080 GOTO 7020
15000 REM HAUPTVERARBEITUNG
15010 REM FELDVEREINBARUNG (EINZELN) (GES. S. AB ZEILENNR. 7000)
15020 GOSUB 2096
15030 INPUT "PERS.NR.(1-500), NAME, BRUTTO, ST.-KL.(1-6)    ***
0,0,0,0 = ENDE    *** ";PERS.NR!,GES.NAME$,LOHN#,ST.KLASSE%
15031 IF PERS.NR!=0 THEN 16900
15035 IF PERS.NR!<0 OR FIX(PERS.NR!)<>PERS.NR! OR PERS.NR!>500
THEN 15030
15037 IF LOHN#<=0 THEN 15030
15040 IF LEN(GES.NAME$)>20 THEN 15030
15050 IF ST.KLASSE% <=0 OR ST.KLASSE%>6 OR FIX(ST.KLASSE%)<>ST.K
LASSE% THEN 15030
15060 REM FESTSTELLEN,OB P-NR. BEREITS VORHANDEN
15070 GET #1,CINT(PERS.NR!)  ' RECORD-NUMMER MUSS INTEGER SEIN!!
15080 REM WENN NICHT VORHANDEN, STEHT IN DER FELDVARIABLEN PNR$
4 x chr$(0) !
15090 IF PNR$<>CHR$(0)+CHR$(0)+CHR$(0)+CHR$(0) THEN PRINT "Pers.
-Nr. ist bereits vorhanden!" : GOTO 15030
15095 REM ZUWEISUNG DER PROGRAMMVARIABLENWERTE AUF DIE FELDVARIA
BLEN (VORBEREITUNG ZUM SCHREIBEN = PUT)
15100 RSET PNR$=MKS$(PERS.NR!)
15115 LSET N$=GES.NAME$
15120 RSET LO$=MKD$(LOHN#)
15130 RSET KL$=MKI$(ST.KLASSE%)
15135 REM FILLER$ BRAUCHT NICHT GEFUELLT ZU WERDEN; DEFAULT-WERT
IST : SOVIEL CHR$(0) WIE IN FIELD ALS LAENGE FESTGELEGT
15140 PUT #1,CINT (PERS.NR!)
15145 PRINT PERS.NR!;"ist jetzt erfasst" : GOTO 15030
16900 REM ENDVERARBEITUNG
16910 CLOSE #1 : END
```

4.3.11 Spezielle und reservierte Variablen

Die speziellen Variablen **DATE$** und **TIME$** werden vom System auf PCOS-Ebene verwaltet und bieten die Möglichkeit, den M 20 eine Echtzeit führen zu lassen.

Zuweisungen auf diese Variablen können erfolgen

a) durch den PCOS-Befehl **ss;**

b) durch eine **LET**-Zuweisung in der BASIC-Ebene. Eine Zuweisung über **INPUT** o.ä. ist nur über Hilfsvariablen möglich; bei **INPUT DATE$** z.B. wird zwar keine Fehlermeldung beim Input gegeben, der Inhalt von **DATE$** zum Zeitpunkt der Eingabe wird jedoch durch diese nicht verändert.

Mit den reservierten Variablen **ERR** und **ERL** kann der Fehler-Code des zuletzt aufgetretenen BASIC-Fehlers (vgl. folgendes Kapitel) bzw. die Zeilennr. abgefragt werden, in der der letzte BASIC-Fehler auftrat. Zuweisungen auf **ERL** sind auf normalem Wege nicht möglich, Zuweisungen auf **ERR** nur mit der Anweisung **ERROR.**

Die besondere Variable **NAME$** darf in BASIC-Programme überhaupt nicht verwendet werden, da sie von PCOS selbst für ein Assembler-Makro benötigt wird, das bei der Abarbeitung einer **DRAW**-Anweisung automatisch aufgerufen wird.

4.3.12 Fehlerbehandlungsroutine

Durch Fehler bei der Erstellung des Programmes oder durch unzulässige oder unvorhergesehene Datenkonstellationen können Situationen entstehen, die das System meldet.

Befindet sich das System in der PCOS-Ebene oder wurden PCOS-Routinen vom BASIC-Interpreter aufgerufen, erfolgt die Meldung vom Betriebssystem PCOS. Diese Meldungen werden im weiteren kurz PCOS-Fehler genannt. Befindet sich das System in der BASIC-Ebene, erfolgen die Meldungen vom BASIC-Interpreter (Ausnahme: Aufruf von PCOS-Routinen in einem BASIC-Programm).Diese Meldungen werden im weiteren kurz BASIC-Fehler genannt. Die einzelnen BASIC-Fehlermeldungen werden in Kapitel 7 erläutert.

Mit Hilfe der Anweisungen **ON ERROR GOTO, RESUME** sowie der reservierten **ERR** und **ERL** kann in Programmen in einer Fehlerbehandlungsroutine definiert werden, was geschehen soll, wenn der Interpreter einen BASIC-Fehler meldet.

Ohne Fehlerbehandlungsroutine werden alle Fehler, die das System beim Abarbeiten einer Anweisung oder eines Befehls entdeckt, in der nächsten Bildschirm- bzw. Window-Zeile gemeldet.

BASIC-Fehler werden dabei mit ihrer Bedeutung ausgegeben, PCOS-Fehler hingegen mit ihrem Fehler-Code.

Treten BASIC-Fehler bei der Abarbeitung von Anweisungen im Rahmen eines Programms auf, wird immer die Zeilennummer mit angegeben, in der der Fehler erkannt wurde. Treten BASIC-Fehler während der Abarbeitung von Anweisungen im Direkt-Mode auf, wird keine Zeilennummer angegeben.

In Fehlerbehandlungsroutinen können BASIC-Fehler abgefangen werden. Ist im Programm eine Fehlerbehandlungsroutine aktiviert (**ON ERROR GOTO**), so verzweigt das Programm selbsttätig zu der bei **ON ERROR GOTO** festgelegten Zeilennummer, sobald der Interpreter einen Fehler erkennt. Der Fehler wird nicht angezeigt und kann in der Fehlerbehandlungsroutine behandelt werden (z.B. durch korrigierende Wertzuweisung oder Ausgabe einer Meldung, was vom Bediener zu tun ist). Die Behandlung kann von Fehler zu Fehler verschieden definiert werden, und zwar in Abhängigkeit von den Inhalten der Variablen **ERR** und/oder **ERL**.
Der Rücksprung ins Hauptprogramm (z.B. nach Behebung der Fehlerursache) wird durch **RESUME** erreicht. Es kann dabei vorgeschrieben werden, daß

a) die Anweisung, bei der der Fehler erkannt wurde, nochmals abgearbeitet werden soll (**RESUME** oder **RESUME 0**);

b) die Anweisung als nächste abgearbeitet werden soll, die der Anweisung unmittelbar folgt, bei der der Fehler verursacht wurde (**RESUME NEXT**);

c) ab einer bestimmten Zeilennr. weitergearbeitet werden soll. Diese Zeilennr. muß außerhalb der Fehlerbehandlungsroutine liegen.

Im Fall a) ist zu beachten, daß gewisse Fehler nicht behoben werden können, ohne das Programm selbst zu verändern (z.B. Syntax-Fehler); erfolgt dies nicht, verzweigt das System sofort wieder in die Fehlerbehandlungsroutine, so daß das Programm in eine Art Endlosschleife gerät. Im Falle des Fehlers 6 ("Overflow") gilt bei **INPUT** bzw. **LINE INPUT** auf numerische Werte: Der durch eine zu große Eingabe verursachte "Overflow" kann zwar in der Fehlerbehandlungsroutine bearbeitet werden und die den Fehler verursachende Anweisung wiederholt werden; in diesem Falle wird die Meldung "Overflow" jedoch in der nächsten Zeile trotzdem ausgegeben.

Durch **RESUME** erhält die reservierte Variable **ERR** automatisch den Wert 0 zugewiesen; der Inhalt von **ERL** hingegen bleibt unverändert.

Beispiel:

```
10 REM   TEST, OB EIN DATENFILE VORHANDEN IST (BEI OPEN "
I" ... WIRD NICHT SOFORT ANGELEGT, SONDERN 'File not foun
d' GEMELDET!)
15 ON ERROR GOTO 22800
20 INPUT "STATION (0 ODER 1), FILEBEZEICHNUNG";S$,F$
30 IF S$<>"1" AND S$<>"0" THEN 20
40 OPEN "I",#1,S$+":"+F$ ' GEHT AUCH FUER RANDOM-FILES!
50 REM WEITER
60 S=SQR(-50)
9999 CLOSE : END
22800 REM ERROR-AUSGANG
22801 IF ERR=53 AND ERL=40 THEN PRINT "BITTE RICHTIGE DIS
KETTE  EINLEGEN UND CR DRUECKEN!":K$=INPUT$(1):IF K$<>CHR
$(13) THEN 22801 ELSE CLOSE : RUN
22802 IF ERR=64 AND ERL=40 THEN PRINT "UNZULCSSIGE FILEBE
ZEICHNUNG" : RESUME 20
22803 PRINT "FEHLER";ERR;"IN ZEILE";ERL : ON ERROR GOTO 0
 : CLOSE : STOP
```

PCOS-Fehler können in Fehlerbehandlungsroutinen nicht abgefangen werden, die Fehlermeldung erscheint immer. Allerdings hat **ERR** nach Meldung eines PCOS-Fehlers (z.B. bei **CALL**-oder **EXEC**-Anweisungen innerhalb eines BASIC-Programms) immer den Wert 40.

Mit Hilfe der Anweisung **ERROR** kann auf **ERR** ein Wert zugewiesen werden. (Davon wird indirekt auch **ERL** betroffen!) Dies läßt sich zu Testzwecken ausnutzen (Simulation von Fehler-Konditionen).

Es bietet sich aber auch eine weitere Möglichkeit: Da die dem Interpreter bekannten Fehler-Codes nur von 1 bis 78 gehen (0 kann nicht auf **ERR** zugewiesen werden), können "künstliche" Fehler vom Anwender definiert und erzeugt werden, die sich z.B. zur Programmsteuerung aus der Fehlerbehandlungsroutine heraus einsetzen lassen.

Beispiel:

```
0 ON ERROR GOTO 22800
1 REM Simulation eines Fehlers
5 I%=1 : LGRENZE#=-50 : RGRENZE#=50
10 INPUT "Eingabe eines Wertes (Intervall -50 <= 50)";WER
T#(I%)
20 GOSUB 800
25 I%=2 : LGRENZE#=-.999 : RGRENZE#=.999
30 INPUT "Eingabe eines Wertes (Intervall -.999 <= +.999)
";WERT#(I%)
40 GOSUB 800
50 REM weiter
799 END
800 IF WERT#(I%)<LGRENZE# OR WERT#(I%)>RGRENZE# THEN ERRO
R 100+I%
809 RETURN
22800 IF ERR < 100 THEN PRINT "Programmfehler" : ON ERROR
 GOTO 0 : STOP
22810 PRINT "Bitte Eingabe im korrekten Intervall"
22820 ON ERR-100 GOTO 22821,22822
22821 RESUME 10
22822 RESUME 30
```

4.3.14 Graphisches Arbeiten

In Abschnitt 4.3.18 wird beschrieben, auf welche Weise der Bildschirm in mehrere Teile (Windows) unterteilt werden kann. Die nachfolgende Beschreibung bezieht sich auf das Arbeiten innerhalb eines Windows.

Jedem Window kann über die Anweisung **SCALE** ein gültiger Wertbereich in problembezogenen Koordinaten zugewiesen werden. Diese Zuweisung erfolgt druch Angabe der gültigen Minimal- und Maximalkoordinaten. Dabei werden der linken unteren Ecke die Minimal- und der rechten oberen Ecke die Maximalwerte zugeordnet. Aus der Größe des Windows (das heißt der Anzahl Elementarpunkte) und dem gültigen Wertebereich ergibt sich ein Maßstab, der für horizontale und vertikale Richtung unterschiedlich sein kann:
Die horizontale Richtung von links nach rechts entspricht der positiven X-Richtung, die vertikale Richtung von unten nach oben entspricht der positiven Y-Richtung.

Für die graphische Darstellung stehen eine Reihe von Anweisungen zur Verfügung, die sich auf das für das Window gültige **SCALE** beziehen. Mit **LINE** können Geraden oder Rechtecke entweder in Absolut- oder Relativkoordinaten definiert werden. **CIRCLE** erlaubt das Zeichnen von Kreisen oder Ellipsen.
Durch diese Anweisungen können Darstellungen graphischer Elemente sehr einfach erfolgen. Zusätzliche Möglichkeiten ergeben sich durch das Verändern von Vorder- und Hintergrund bzw. der für die Darstellung gültigen Farbe (vgl. Kapitel 4.3.18.3). Sowohl **LINE** als auch **CIRCLE** erlauben über einen eigenen Operanden die Ausführung von logischen Operationen auf den durch die Darstellung angesprochenen Elementarpunkten.

Beispiel:

```
5 SCALE 0,511,0,255
10 INPUT "text";A$
20 INPUT "position (spalte,zeile)";X,Y
55 CLS
60 CURSOR (X,Y) :PRINT A$
70 LINE ((X-1)*8,259-Y*16)-STEP(LEN(A$)*8,12),,BF,XOR
80 GOTO 10
90 END
```

(REVERSE; 16 x 64)

Die Anweisung **PAINT** erlaubt das Ausfüllen eines beliebigen umschlossenen Gebietes in einer bestimmten Farbe. Dabei wird - ausgehend von einem Punkt innerhalb dieses Gebietes - die gewählte Farbe für alle Punkte gesetzt, bis zu einer Zeile oder Spalte ein Elementarpunkt einen anderen Farbcode hat.
Es ist möglich, Teile des Bildschirmes in einen Array zu speichern oder in einem Array gespeicherte Bilder auf dem Bildschirm darzustellen. Die Anweisung **GET %** erlaubt die Auswahl dieses Bereiches und die Angabe, in welchem Array die Speicherung erfolgen soll. **PUT %** erlaubt die Darstellung eines gespeicherten Bildes an einer Stelle des Bildschirmes.

Das Ansprechen von einzelnen Elementarpunkten ist über die Anweisungen **PSET** und **PRESET** möglich. Über **DRAW** können sehr komplexe Darstellungen von Figuren erreicht werden. Durch die Angabe in Relativkoordinaten können solche Figuren leicht an beliebigen Stellen dargestellt werden.

ALPHANUMERISCHER TASTATURBLOCK

Angabe auf Taste	KLEINSCHREIBUNG dez. Code	KLEINSCHREIBUNG hexa. Code	KLEINSCHREIBUNG erzeugtes Zeichen	GROSSCHREIBUNG (+ SHIFT) dez. Code	GROSSCHREIBUNG (+ SHIFT) hexa. Code	GROSSCHREIBUNG (+ SHIFT) erzeugtes Zeichen	+ CONTROL dez. Code	+ CONTROL hexa. Code	+ CONTROL erzeugtes Zeichen	+ COMMAND dez. Code	+ COMMAND hexa. Code	+ COMMAND erzeugtes Zeichen
A	97	&61	a	65	&41	A	1	&01	(SOH)	128	&80	
B	98	&62	b	66	&42	B	2	&02	(STX)	129	&81	
C	99	&63	c	67	&43	C	3	&03	(ETX)	130	&82	
D	100	&64	d	68	&44	D	4	&04	(EOT)	131	&83	
E	101	&65	e	69	&45	E	5	&05	(ENQ)	132	&84	
F	102	&66	f	70	&46	F	6	&06	(ACK)	133	&85	
G	103	&67	g	71	&47	G	7	&07	(BEL)	134	&86	
H	104	&68	h	72	&48	H	8	&08	(BS)	135	&87	
I	105	&69	i	73	&49	I	9	&09	(HT)	136	&88	
J	106	&6A	j	74	&4A	J	10	&0A	(LF)	137	&89	
K	107	&6B	k	75	&4B	K	11	&0B	(VT)	138	&8A	
L	108	&6C	l	76	&4C	L	12	&0C	(FF)	139	&8B	
M	109	&6D	m	77	&4D	M	13	&0D	(CR)	140	&8C	
N	110	&6E	n	78	&4E	N	14	&0E	(SO)	141	&8D	
O	111	&6F	o	79	&4F	O	15	&0F	(SI)	142	&8E	
P	112	&70	p	80	&50	P	16	&10	(DLE)	143	&8F	
Q	113	&71	q	81	&51	Q	17	&11	(DC1)	144	&90	
R	114	&72	r	82	&52	R	18	&12	(DC2)	145	&91	
S	115	&73	s	83	&53	S	*1			146	&92	
T	116	&74	t	84	&54	T	20	&14	(DC4)	147	&93	
U	117	&75	u	85	&55	U	21	&15	(NAK)	148	&94	
V	118	&76	v	86	&56	V	22	&16	(SYN)	149	&95	
W	119	&77	w	87	&57	W	23	&17	(ETB)	150	&96	
X	120	&78	x	88	&58	X	24	&18	(CAN)	151	&97	
Y	121	&79	y	89	&59	Y	25	&19	(EM)	152	&98	
Z	122	&7A	z	90	&5A	Z	26	&1A	(SUB)	153	&99	

ALPHANUMERISCHER TASTATURBLOCK

Angabe auf Taste	KLEINSCHREIBUNG dez. Code	hexa. Code	erzeugtes Zeichen	GROSSCHREIBUNG (+ SHIFT) dez. Code	hexa. Code	erzeugtes Zeichen	+ CONTROL dez. Code	hexa. Code	erzeugtes Zeichen	+ COMMAND dez. Code	hexa. Code	erzeugtes Zeichen
1	49	&31	1	59	&3B	;	225	&E1		237	&ED	
2	50	&32	2	34	&22	"	226	&E2		238	&EE	
3	51	&33	3	61	&3D	=	227	&E3		239	&EF	
4	52	&34	4	37	&25	%	228	&E4		240	&F0	
5	53	&35	5	38	&26	&	229	&E5		241	&F1	
6	54	&36	6	40	&28	(	230	&E6		242	&F2	
7	55	&37	7	41	&29	)	231	&E7		243	&F3	
8	56	&38	8	95	&5F	_	232	&E8		244	&F4	
9	57	&39	9	64	&40	§	233	&E9		245	&F5	
ø	48	&30	ø	47	&2F	/	224	&E0		236	&EC	
ß	126	&7E	ß	58	&3A	:	234	&EA		246	&F6	
^	94	&5E	^	96	&60	`	235	&EB		247	&F7	
Ü	125	&7D	ü	93	&5D	Ü	0	&00	(NUL)	19	&13	(DC3)
+	43	&2B	+	42	&2A	*	251	&FB		28	&1C	(FS)
Ö	124	&7C	ö	92	&5C	Ö	30	&1E	(RS)	252	&FC	
Ä	123	&7B	ä	91	&5B	Ä	31	&1F	(US)	253	&FD	
$	36	&24	$	35	&23	#	29	&1D	(GS)	159	&9F	
<	60	&3C	<	62	&3E	>	127	&7F	(DEL)	248	&F8	
,	44	&2C	,	63	&3F	?	254	&FE		249	&F9	
.	46	&2E	.	33	&21	!	160	&A0		250	&FA	
-	45	&2D	-	39	&27	'	255	&FF			*2	
CR	13	&0D	(CR)	13	&0D	(CR)	13	&0D	(CR)	13	&0D	(CR)
BLANK	32	&20	( )	32	&20	( )	32	&20	( )	32	&20	( )
RESET		**			**			*3			**	

NUMERISCHER TASTATURBLOCK

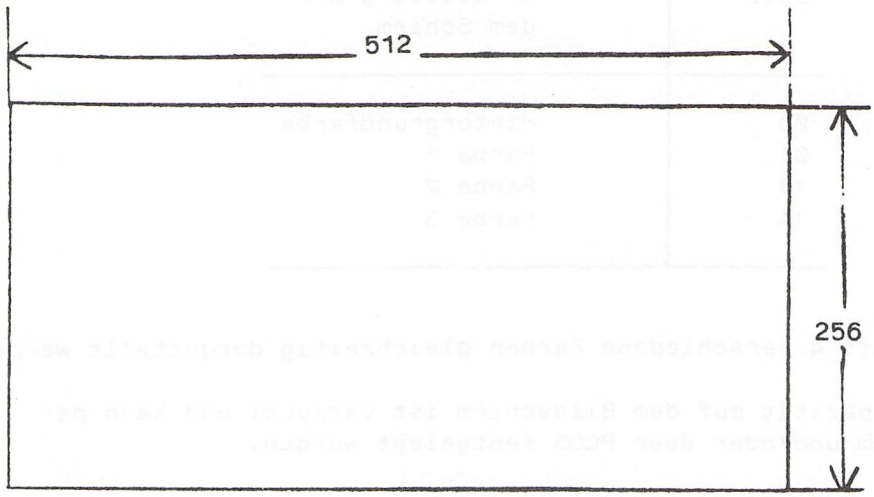
Angabe auf Taste	KLEINSCHREIBUNG dez. Code	hexa. Code	erzeugtes Zeichen	GROSSCHREIBUNG (+ SHIFT) dez. Code	hexa. Code	erzeugtes Zeichen	+ CONTROL dez. Code	hexa. Code	erzeugtes Zeichen	+ COMMAND dez. Code	hexa. Code	erzeugtes Zeichen
Ø	48	&30	Ø	48	&30	Ø	48	&30	Ø	48	&30	Ø
1	49	&31	1	28	&1C	(FS)	49	&31	1	49	&31	1
2	50	&32	2	154	&9A	(↓)	*4			50	&32	2
3	51	&33	3	29	&1D	(GS)	51	&33	3	51	&33	3
4	52	&34	4	155	&9B	(←)	*5			52	&34	4
5	53	&35	5	156	&9C	(HOME)	27	&1B	(ESC)	53	&35	5
6	54	&36	6	157	&9D	(→)	*6			54	&36	6
7	55	&37	7	30	&1E	(RS)	55	&37	7	55	&37	7
8	56	&38	8	158	&9E	(↑)	*7			56	&38	8
9	57	&39	9	31	&1F	(US)	57	&39	9	57	&39	9
+	43	&2B	+	43	&2B	+	**			43	&2B	+
-	45	&2D	-	45	&2D	-	**			45	&2D	-
*	42	&2A	*	42	&2A	*	42	&2A	*	42	&2A	*
/	47	&2F	/	47	&2F	/	47	&2F	/	47	&2F	/
.	46	&2E	.	46	&2E	.	46	&2E	.	46	&2E	.
ØØ	*8			*8			*8			*8		

ANMERKUNGEN:

**	kein Code im Puffer
*1	unterbricht Programmablauf (Pausenfunktion); kein eigener Code im Puffer
*2	Umschaltfunktion; kein Code im Puffer
*3	(Übergang zur PCOS-Ebene und Neu-Start)
*4	10 x &9A (↓)
*5	10 x &9B (←)
*6	10 x &9D (→)
*7	10 x &9E (↑)
*8	2 x &3Ø (also zweimal die Wirkung von Ø)

4.3.18 Bildschirmorganisation und Windows

Der M 20-Bildschirm ist physisch eingeteilt in ein Raster von 512 x 256 Elementarpunkten (= 131072)



Darstellungsfläche
auf M 20-Bildschirm

Sowohl die graphische als auch die alphanumerische Darstellung basiert auf dem Leuchten oder Nichtleuchten von einzelnen Elementarpunkten innerhalb der 512x256-Punkte-Matrix. Für die Darstellung des Bildschirminhaltes steht beim einfarbigen Schirm ein Speicherbereich von 16 KB zur Verfügung. Für den Farbschirm werden 32 KB benötigt; 16 KB davon werden im Anwender-Speicher belegt.
Dieser Speicherbereich wird als Bitmap bezeichnet.

Beim einfarbigen Schirm ist ein Bit innerhalb der Bitmap einem Elementarpunkt auf dem Bildschirm zugeordnet:

Bit	Darstellung auf dem Schirm
Ø	Hintergrundfarbe (leuchtet nicht)
1	Vordergrundfarbe (leuchtet)

Beim mehrfarbigen Schirm werden für jeden dargestellten Elementarpunkt zwei Bits in der Bitmap reserviert:

Bits	Darstellung auf dem Schirm
ØØ	Hintergrundfarbe
Ø1	Farbe 1
1Ø	Farbe 2
11	Farbe 3

Es können somit 4 verschiedene Farben gleichzeitig dargestellt werden.

Die Zeichenkapazität auf dem Bildschirm ist variabel und kann per BASIC-Programm und/oder über PCOS festgelegt werden.

Zeichenkapazität	Anz.Zeichen pro Zeile	Anzahl Zeilen	Anz.Zeichen auf Schirm
minimal	64	16	1024
maximal	80	25	2000

Im folgenden soll die Zeichendarstellung, die Bestimmung der Anzahl Zeichen pro Zeile und die Festlegung der Anzahl Zeilen des Schirms erläutert werden.

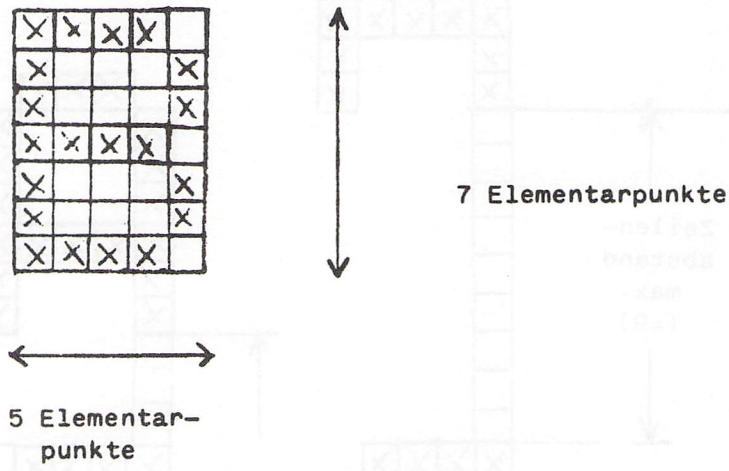
4.3.18.1 Zeichendarstellung und Bildschirmorganisation

Zeichendarstellung

Die Größe der Zeichen ist durch die Darstellung in einer 5x7-Punkte-Matrix definiert.

(Sonderfälle bilden z.B. Buchstaben mit Unterlänge wie g,q und j oder die Umlaute Ä,Ü und Ö)

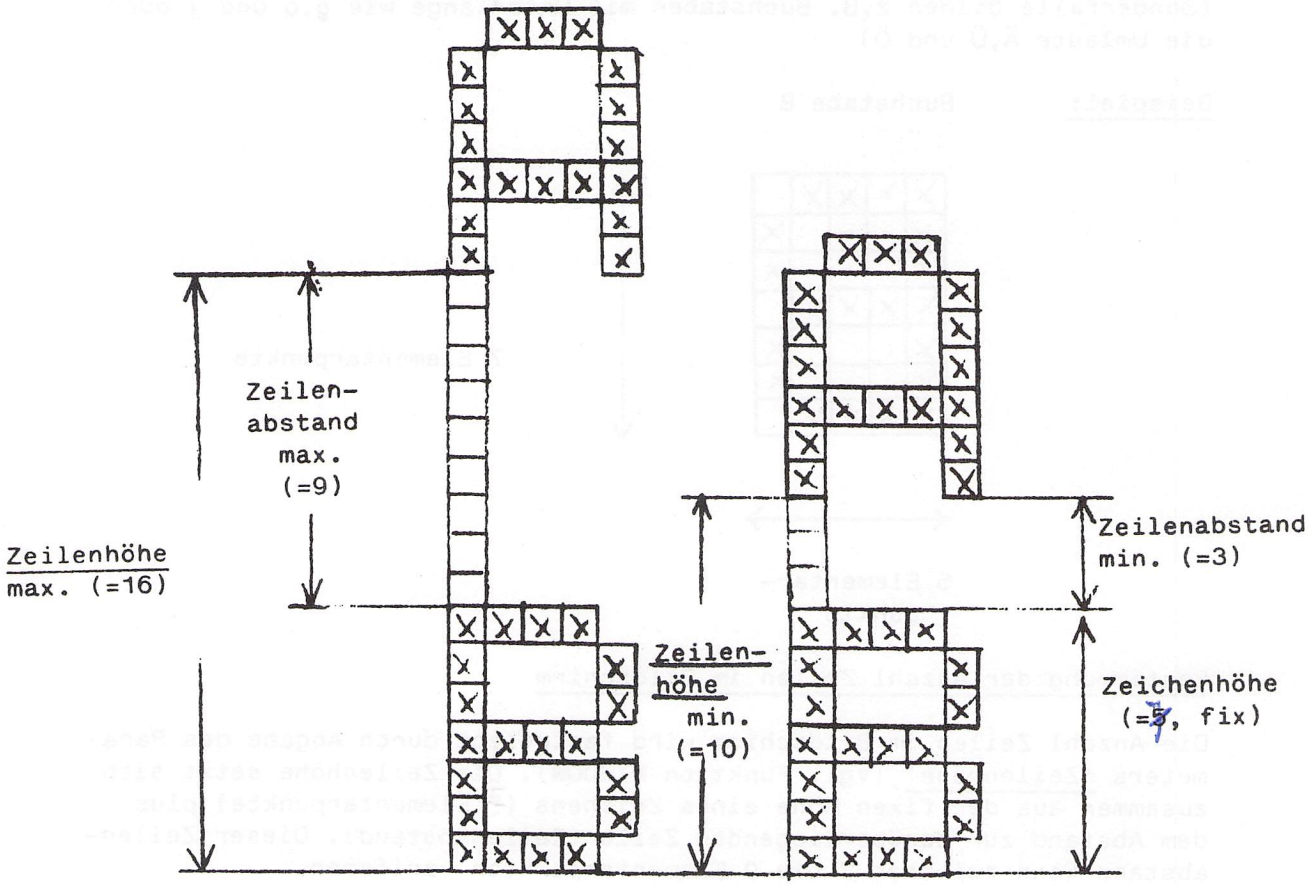
Beispiel: Buchstabe B



Bestimmung der Anzahl Zeilen im Bildschirm

Die Anzahl Zeilen im Bildschirm wird festgelegt durch Angabe des Parameters 'Zeilenhöhe' (vgl. Funktion **WINDOW**). Die Zeilenhöhe setzt sich zusammen aus der fixen Höhe eines Zeichens (7 Elementarpunkte) plus dem Abstand zur darüberliegenden Zeile (Zeilenabstand). Dieser Zeilenabstand kann zwischen 3 bis 9 Elementarpunkten variieren.

Beispiel:


Zeilen-
abstand
max.
(=9)
Zeilenhöhe
max. (=16)
Zeilen-
höhe
min.
(=10)
Zeilenabstand
min. (=3)
Zeichenhöhe
(=7, fix)

Durch Angabe der Zeilenhöhe wird somit der Zeilenstand bestimmt:

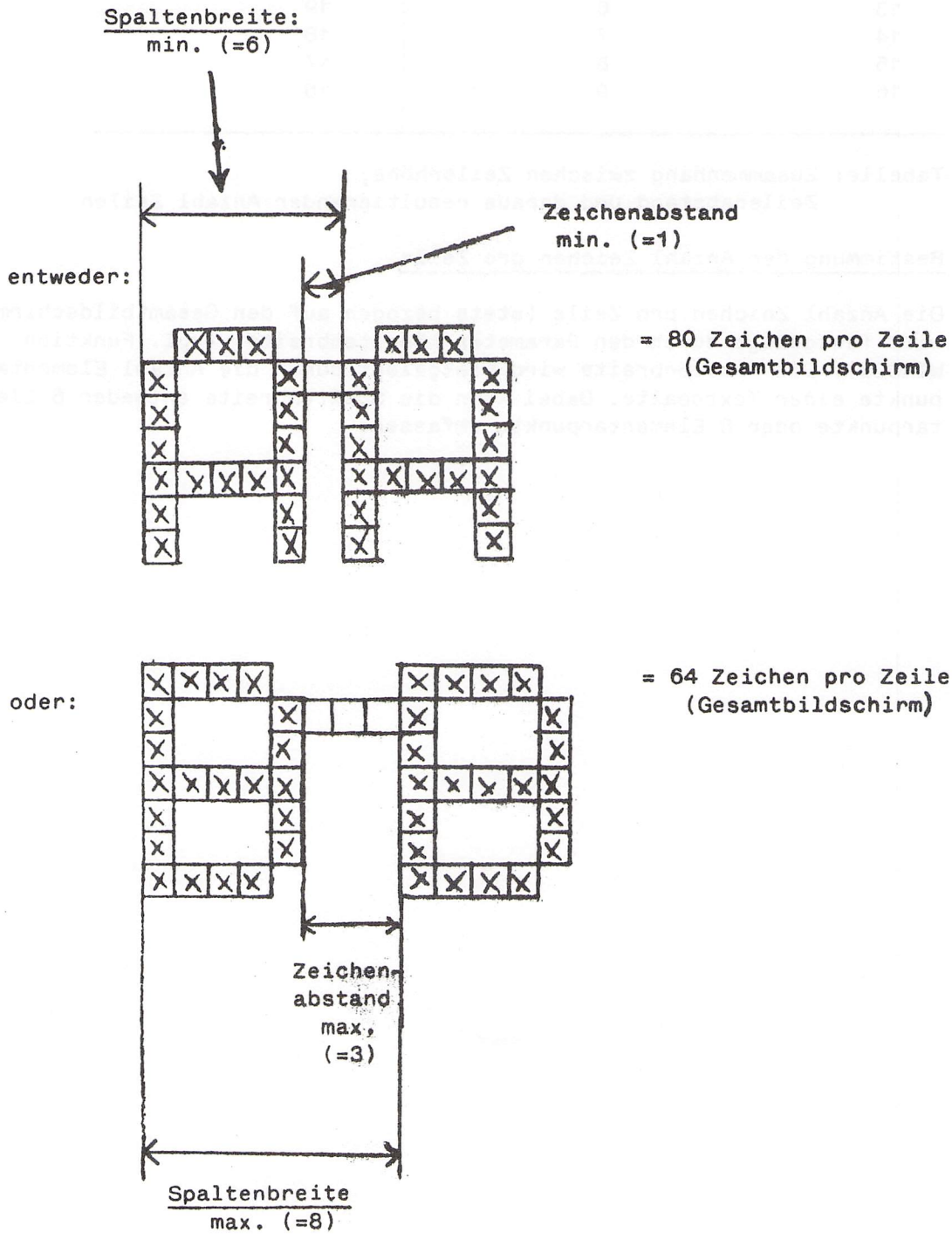
Zeilen-höhe (Elementarpunkte)	Zeilen-abstand (Elementarpunkte)	Anzahl Zeilen bzgl. Gesamtschirm
10	3	25
11	4	23
12	5	21
13	6	19
14	7	18
15	8	17
16	9	16

Tabelle: Zusammenhang zwischen Zeilenhöhe, Zeilenabstand und daraus resultierender Anzahl Zeilen

Bestimmung der Anzahl Zeichen pro Zeile

Die Anzahl Zeichen pro Zeile (stets bezogen auf den Gesamtbildschirm) wird festgelegt durch den Parameter 'Spaltenbreite' (vgl. Funktion **WINDOW**). Die Spaltenbreite wird festgelegt durch die Anzahl Elementarpunkte einer Textspalte. Dabei kann die Spaltenbreite entweder 6 Elementarpunkte oder 8 Elementarpunkte umfassen:

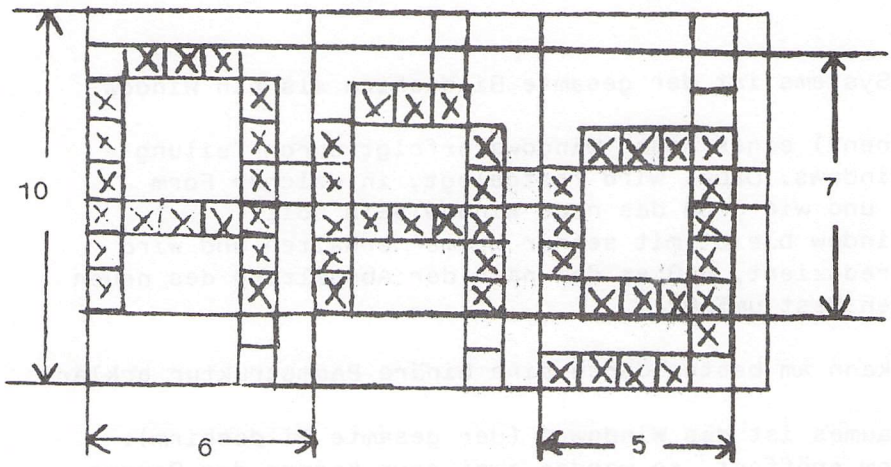
Beispiel:


Spaltenbreite:
min. (=6)
Zeichenabstand
min. (=1)
entweder:
= 80 Zeichen pro Zeile
(Gesamtbildschirm)
oder:
= 64 Zeichen pro Zeile
(Gesamtbildschirm)
Zeichen-
abstand
max.
(=3)
Spaltenbreite
max. (=8)

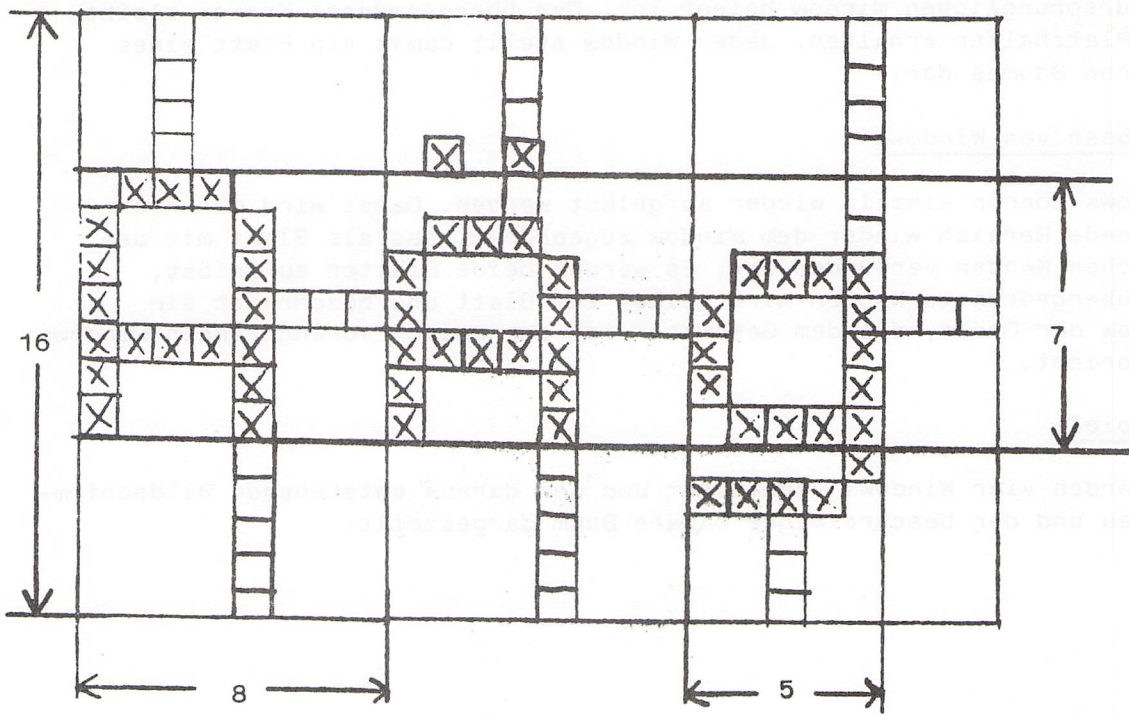
Spalten-breite (Elementarpunkte)	Zeichen-abstand (Elementarpunkte)	Zeichendichte (bzgl. Gesamtschirm)
8	3	64
6	1	80

Tabelle: Zusammenhang zwischen Spaltenbreite, Spaltenabstand und daraus resultierende Anzahl Zeichen pro Zeile

Zusammenfassung der Zeichendarstellung am Beispiel 80x25 und 64x16:



80 x 25



64 x 16

4.3.18.2 Arbeit mit Windows

Das Bildschirm-Handling des M 20 zeichnet sich durch die Möglichkeit der Window-Technik aus. Per Definition ist der M 20-Bildschirm als ein Window zu betrachten, aus dem bis zu 16 unabhängige Windows durch Teilung gebildet werden können. Diese Windows können als "eigene Bildschirme" betrachtet werden, die unabhängig voneinander angesprochen werden können. Das Anlegen von Windows geschieht mit der Funktion **WINDOW**. Innerhalb der Windows kann all das gemacht werden, was auf dem Gesamtbildschirm (Window-Nr. 1) möglich ist; d.h., es existiert in jedem Window ein Text- und ein Graphik-Cursor. Das Window kann einzeln aufgelöst werden (Anweisung **CLOSE WINDOW**). Es ist möglich, eine Hard-Copy für ein einzelnes Window zu machen, ein eigenes Scrolling durchzuführen oder die Anzahl Zeilen bzw. Anzahl Zeichen pro Zeile für jedes Window festzulegen.

Anlegen von Windows

Nach dem Laden des Systems ist der gesamte Bildschirm als ein Window zu betrachten.
Das Anlegen ("Eröffnen") eines neuen Windows erfolgt durch Teilung eines bestehenden Windows. Dabei wird festgelegt, in welcher Form die Teilung erfolgt und wie groß das neue Window sein soll.
Das ursprüngliche Window bleibt mit seiner Nummer erhalten und wird in seiner Größe so reduziert, daß es den nach der Abspaltung des neuen Windows verbleibenden Rest umfaßt.

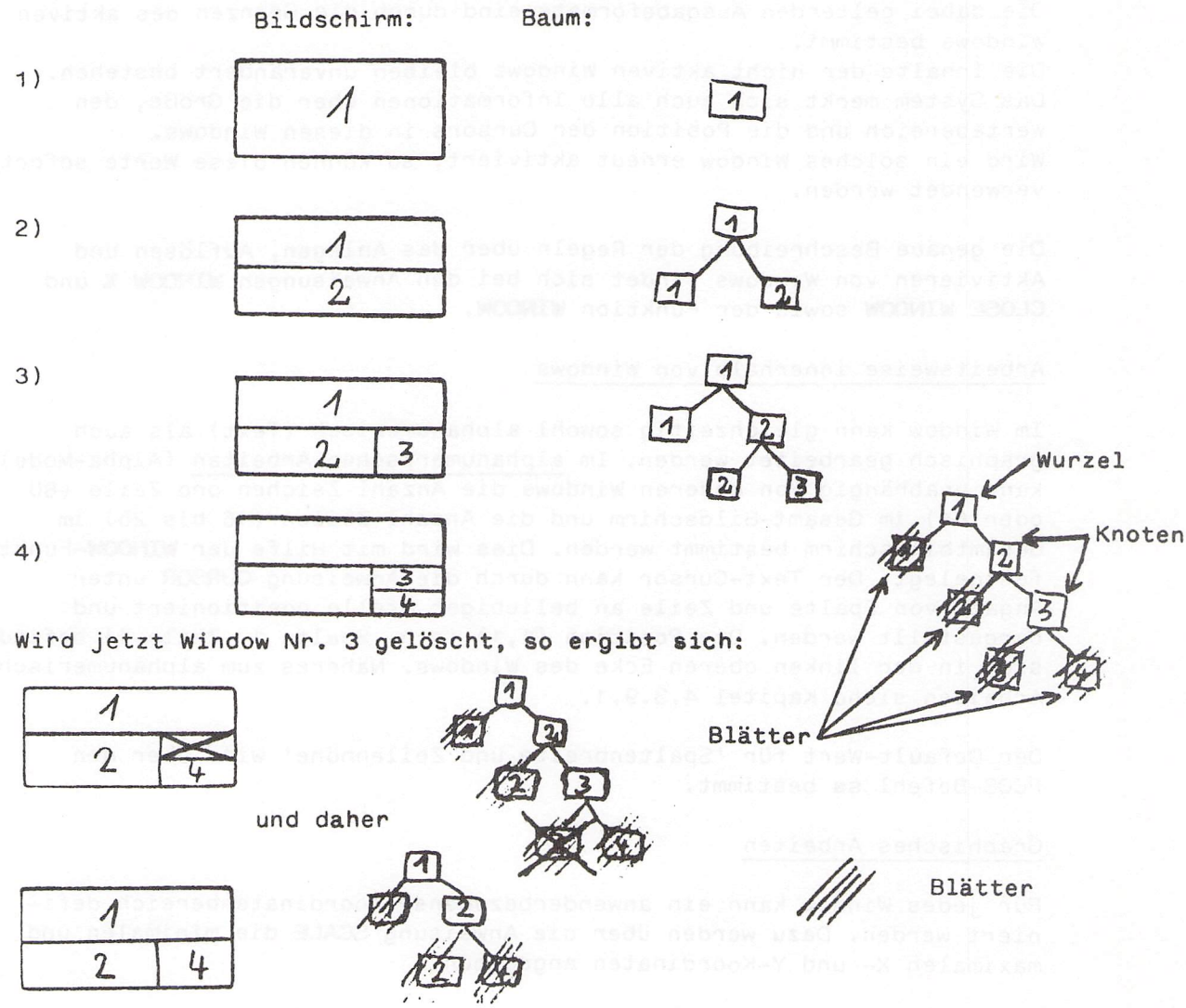
Diese Unterteilung kann am besten durch eine binäre Baumstruktur erklärt werden.
Die Wurzel dieses Baumes ist das Window 1 (der gesamte Bildschirm). Wird ein neues Window eröffnet, so werden zwei neue Knoten des Baumes angelegt, wovon ein Knoten vom neuen Window und ein Knoten vom verkleinerten ursprünglichen Window belegt ist. Der übergeordnete Knoten bleibt als Platzhalter erhalten. Jedes Window stellt damit ein Blatt eines binären Baumes dar.

Auflösen von Windows

Windows können einzeln wieder aufgelöst werden. Dabei wird der freiwerdende Bereich wieder dem Window zugeordnet, das als Blatt mit dem gleichen Knoten verbunden ist. Es werden beide Blätter aufgelöst, der übergeordnete Knoten wird selbst zum Blatt und beschreibt ein Window der Größe, die dem Gesamtbereich der beiden vorangehenden Windows entspricht.

Beispiel:

Es werden vier Windows vereinbart und der daraus entstehende Bildschirmaufbau und der beschreibende binäre Baum dargestellt:



Aktivieren eines Windows

Jedes Window stellt einen logisch selbständigen Bildschirm dar.
Da sinnvoll nur mit jeweils einem Bildschirm gearbeitet werden kann, muß jeweils eines der eröffneten Windows als aktives Window vereinbart werden.
Alle Anweisungen und Meldungen, die sich auf den Bildschirm beziehen und nicht einen eigenen Parameter 'Window-Nr.' haben, beziehen sich immer auf das aktive Window.

Die dabei geltenden Ausgabeformate sind durch die Grenzen des aktiven Windows bestimmt.
Die Inhalte der nicht aktiven Windows bleiben unverändert bestehen.
Das System merkt sich auch alle Informationen über die Größe, den Wertebereich und die Position der Cursors in diesen Windows.
Wird ein solches Window erneut aktiviert, so können diese Werte sofort verwendet werden.

Die genaue Beschreibung der Regeln über das Anlegen, Auflösen und Aktivieren von Windows findet sich bei den Anweisungen **WINDOW %** und **CLOSE WINDOW** sowie der Funktion **WINDOW**.

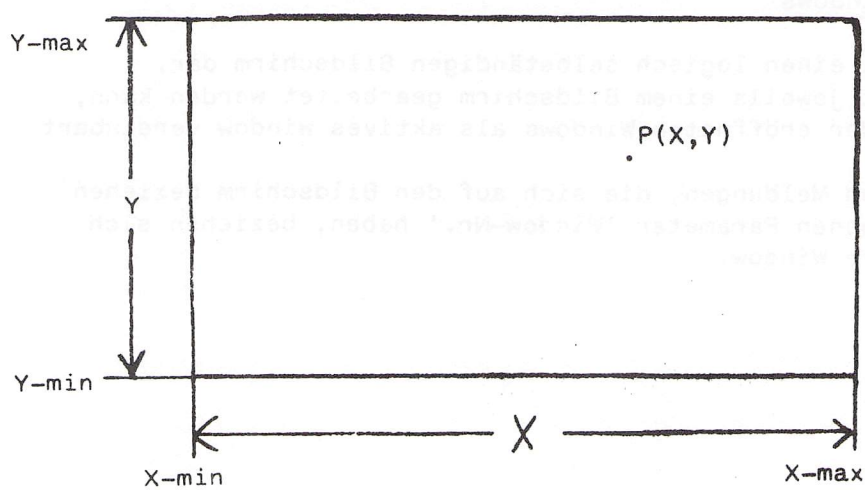
Arbeitsweise innerhalb von Windows

Im Window kann gleichzeitig sowohl alphanumerisch (Text) als auch graphisch gearbeitet werden. Im alphanumerischen Arbeiten (Alpha-Mode) kann unabhängig von anderen Windows die Anzahl Zeichen pro Zeile (80 oder 64) im Gesamt-Bildschirm und die Anzahl Zeilen (16 bis 25) im Gesamtbildschirm bestimmt werden. Dies wird mit Hilfe der **WINDOW**-Funktion festgelegt. Der Text-Cursor kann durch die Anweisung **CURSOR** unter Angabe von Spalte und Zeile an beliebiger Stelle positioniert und dargestellt werden. Die Position (1,1) (d.h. Spalte 1, Zeile 1) befindet sich in der linken oberen Ecke des Windows. Näheres zum alphanumerischen Arbeiten siehe Kapitel 4.3.9.1.

Der Default-Wert für 'Spaltenbreite und Zeilenhöhe' wird über den PCOS-Befehl **ss** bestimmt.

Graphisches Arbeiten

Für jedes Window kann ein anwenderbezogener Koordinatenbereich definiert werden. Dazu werden über die Anweisung **SCALE** die minimalen und maximalen X- und Y-Koordinaten angegeben.



Die Default-Werte für die Koordinaten innerhalb eines Windows sind:

X-min	0
Y-min	0
X-max	Anzahl horizontaler Elementarpunkte - 1
Y-max	Anzahl vertikaler Elementarpunkte - 1

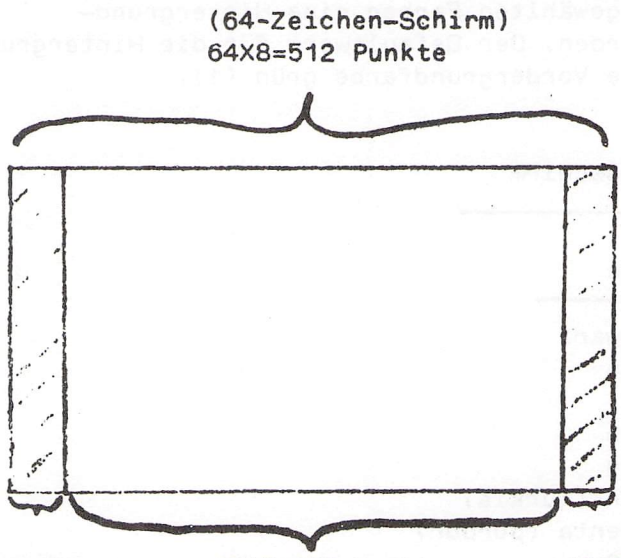
Die Default-Werte von **SCALE** für den Gesamtschirm sind:

bei 64 Zeichen pro Zeile:

X-min	0
Y-min	0
X-max	479
Y-max	255

bei 80 Zeichen pro Zeile:

X-min	0
Y-min	0
X-max	511
Y-max	255



Skizze:
Matrix der Elementarpunkte für beide Bildschirm-Modes

Der graphische Cursor kann auf einen beliebigen Elementarpunkt innerhalb eines Windows positioniert werden. Die Position (0,0) befindet sich in dem Elementarpunkt in der linken unteren Ecke, falls mit **SCALE** nicht eine andere Koordinateneinteilung definiert wurde. Näheres über Graphik-Anweisungen siehe Kapitel 4.3.14.

4.3.18.3 Verwendung von Farbe

Beim einfarbigen Bildschirm stehen die Farben schwarz und weiß zur Darstellung der Elementarpunkte zur Verfügung. Dabei unterscheidet man in Hintergrund- und Vordergrund-Farbe. Die Hintergrund-Farbe ist die Farbe, welche der Bildschirm nach Ausführung der Anweisung **CLS** annimmt. Der Default-Wert für die Hintergrundfarbe beim einfarbigen Schirm ist schwarz.

Einfarbiger Bildschirm

Farbcode	Farbe
Ø	schwarz
1	weiß

Beim mehrfarbigen Schirm stehen für die Darstellung der Elementarpunkte acht Farben zur Verfügung. Aus diesen acht Farben können mit der Anweisung **COLOR=** (globale **COLOR**-Anweisung) vier Farben ausgewählt werden, mit denen gleichzeitig gearbeitet werden kann. Den Farben ist ein Farbcode (Ø-7) zugeordnet, mit dem diese Farben angesprochen werden. In jedem Window kann aus den vorgewählten Farben eine Hintergrund- und Vordergrundfarbe bestimmt werden. Der Defaultwert für die Hintergrundfarbe ist schwarz (Ø) und für die Vordergrundfarbe grün (1).

Mehrfarbiger Bildschirm

Farbcode	Farbe
Ø	schwarz
1	grün
2	rot
3	gelb
4	blau
5	cyan (türkis)
6	magenta (purpur)
7	weiß

Die Bestimmung einer Hintergrundfarbe und einer Vordergrundfarbe erfolgt durch die Anweisung **COLOR**. In der Anweisung **COLOR** ist jedoch nicht der Farbcode anzugeben, sondern die Position (Ø-3) der Farben in der Anweisung **COLOR=** (von links abzählend). Diese Position wird in den Format-Beschreibungen mit 'Farbindex' bezeichnet.

FUNKTION: **ABS**

ZWECK: liefert den Absolutwert (Betrag) eines numerischen Werts

FORMAT:

```
ABS(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet und in seinen positiven Wert verwandelt.

BEMERKUNGEN:

- Das Ergebnis der Funktion ist von der Genauigkeit des Arguments.

BEISPIELE:

```
10 LPRINT ABS(12.5);ABS(-12.5)
 12.5  12.5
```

VERWEISE: Kapitel 2.4 und 4.13

FUNKTION: **ASC**

ZWECK: liefert den ISO-Code eines Zeichens

FORMAT:

```
ASC(Stringausdruck)
```

WIRKUNG: Der Stringausdruck wird errechnet und das erste Zeichen bestimmt. Von diesem wird der ISO-Code ermittelt.

BEMERKUNGEN:

- Ist das Ergebnis von 'Stringausdruck' der Leerstring, wird der Fehler "Illegal function call" gemeldet.
- **ASC** ist die Umkehrung der Funktion **CHR$**.

BEISPIELE:

```
10 LPRINT ASC("abc")
 97
```

VERWEISE: Kapitel 4.3.4.3 und 4.3.17
Funktion: **CHR$**

FUNKTION: **ATN**

ZWECK: liefert von einem numerischen Wert seinen Arcustangens im Bogenmaß

FORMAT:

```
ATN(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet, als Tangens interpretiert und der daraus resultierende Winkel im Bogenmaß ermittelt.

BEMERKUNGEN:

- Das Ergebnis ist im Bogenmaß ausgedrückt. Es liegt zwischen -pi/2 und +pi/2. Umrechnungen müßten ggf. in Unterprogrammen oder mit einer vom Anwender definierten Funktion (s. **DEF FN**) erfolgen.
- Die Funktion ist die Umkehrung der Funktion **TAN**.

BEISPIELE:

```
10 PI=3.14
20 LPRINT ATN(1.6)*200/PI
 64.47411
```

VERWEISE: Kapitel 2.4 und 4.13
Funktion: **TAN**

BEFEHL: **AUTO**
(automatic line numbering)

FUNKTION: Automatische Erzeugung von Zeilennummern für einzugebende Anweisungen eines Programmes

FORMAT:

AUTO [{Zeilennummer / .}] [,] [Schrittweite]

Zeilennummer: positive ganze Zahl zwischen 1 und 65529, welche die Nummer der nächsten Anweisung angibt
Schrittweite: positive ganze Zahl, welche die jeweilige Erhöhung zur nächsten Zeilennummer bestimmt

WIRKUNG: Nach jedem Drücken der Eingabeabschlußtaste wird eine Zeilennummer generiert. 'Zeilennummer' bestimmt den Anfangswert, den die erste einzugebende Anweisung erhält. 'Schrittweite' gibt an, um wieviel die laufende Zeilennummer zur nächsten Anweisung erhöht werden soll. Als Default-Wert wird für beide Parameter der Wert 10 angenommen.

BEMERKUNGEN:

- Fehlt einer der Parameter 'Zeilennummer' oder 'Schrittweite', so wird für diesen Parameter der Standardwert 10 angenommen.
- Mit **AUTO.** beginnt die Generierung der Zeilennummern ab der letzten definierten Zeilennummer.
- **AUTO.,** erzeugt Zeilennummern ab der zuletzt definierten Zeilennummer mit der vorher gewählten Schrittweite (eventuell Standardwert 10).
- Wird bei der automatischen Zeilennumerierung eine Zeilennummer erzeugt, unter der bereits eine Programmzeile im Arbeitsspeicher ist, so erscheint neben der Zeilennummer ein *. Durch Drücken der Eingabeabschlußtaste unmittelbar nach * bleibt diese Programmzeile im Arbeitsspeicher erhalten. Jegliche Eingabe ändert die im Arbeitsspeicher vorhandene Programmzeile.
- C beendet die automatische Zeilennumerierung. Die Zeilennummer, bei der C gedrückt wurde, wird nicht gespeichert. Das System befindet sich anschließend im Direkt-Mode.

- Wird bei der automatischen Zeilennumerierung der höchste Wert (65528) erreicht, so wird die Zeilennumerierung beendet; das System befindet sich danach im Direkt-Mode.

BEISPIELE:

```
new
auto

auto 25,80

auto ,15

auto ,,1
```

VERWEISE: Kapitel 3.3

ANWEISUNG: **CALL**

FUNKTION: Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine

FORMAT:

```
CALL { Stringausdruck                          }
     { Name d. Routine  (Liste d. Parameter)   }
```

Stringausdruck:	enthält Namen der Routine und Parameter in der Form, wie sie in der PCOS-Ebene einzugeben sind
Name d. Routine:	Stringausdruck, der den Namen der abzuarbeitenden Routine bestimmt
Liste d. Parameter:	Folge von num. und/oder String-variablen oder Konstanten, die durch Komma getrennt sind

WIRKUNG: Das System prüft, ob die aufgerufene Assembler-Routine bereits im Arbeitsspeicher vorhanden ist. Falls nicht, wird diese Routine auf der Diskette gesucht und in den Arbeitsspeicher geladen. Anschließend wird diese Routine sofort abgearbeitet. Der momentane Zustand des Arbeitsspeichers bleibt erhalten.

BEMERKUNGEN:

- Nach Ausführung der Assembler-Routine wird das BASIC-Programm mit der der **CALL**-Anweisung folgenden Anweisung fortgesetzt.
- Werden an die Assembler-Routine numerische Variablen übergeben, so müssen diese Variablen als Integer-variablen deklariert werden.
- Variablen, die von der Assembler-Routine an das BASIC-Programm übergeben werden, müssen vor dem Variablennamen das **$**-Zeichen stehen haben.
- Zur Beschleunigung des Aufrufs von Assembler-Routinen kann der PCOS-Befehl **pl** verwendet werden. Dann bleiben die Routinen ständig im Arbeitsspeicher erhalten.
- Routinen, die mit dem PCOS-Befehl **pl** bereits geladen wurden, oder beim Laden des Betriebssystems automatisch geladen wurden (siehe PCOS-Befehl **ps**), können nicht noch einmal durch den Befehl **pl** geladen werden.

- Im Arbeitsspeicher resident gehaltene Routinen verringern die Kapazität des Arbeitsspeichers.
- Die am meisten verwendeten PCOS-Routinen innerhalb von BASIC-Programmen sind:

1) **lt** zur Abfrage, welche Eingabeabschlußtaste betätigt wurde
2) **pk** Anwender-definierte Tastenbelegung
3) **la** Ausgabe von Text in der graphischen Arbeitsweise
4) **sp** für Hardcopy
5) **mi** Ansprechen von Byteadressen im Arbeitsspeicher
6) **pl** Laden einer Assembler-Routine in den Arbeitsspeicher als resident gehaltene Routine

BEISPIELE:

```
10 CLEAR
15 CALL "pl la"
20 CALL "la 'OLIVETTI',0,180,4,0"
30 CALL "la 'TEXT',300,10,8,1"
40 CALL "la 'M 20',30,50,10,0"
50 CALL "sp"
60 END
```

```
10 INPUT  "Drücken Sie eine der drei Eingabeabschlusstasten
(CR,S1,S2     S2 = Ende)";I%
20 CALL "lt"(@I%)
30 ON I%+1 GOTO 40,50,60
40 LPRINT "Sie haben die Eingabeabschlusstaste 'S ' gedrückt
": GOTO 10
50 LPRINT "Sie haben die Eingabeabschlusstaste 'S1' gedrückt
": GOTO 10
60 LPRINT "Sie haben die Eingabeabschlusstaste 'S2' gedrückt
"
70 LPRINT "Das Programm ist beendet"
80 END
```

VERWEISE:

Kapitel: 4.3.15, ANHANG IV
Anweisung: **EXEC**
Funktion: **VARPTR**

FUNKTION: **CDBL**
(convert double precision)

ZWECK: Konvertiert einen numerischen Wert in doppelte Genauigkeit

FORMAT:

```
CDBL(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Als Ergebnis steht sein Wert in doppelter Genauigkeit zur Verfügung.

BEMERKUNGEN: ACHTUNG: Auch bei Anwendung der Funktion **CDBL** kann bei der Umwandlung eines Wertes in einfacher Genauigkeit Signifikanzverlust eintreten; d.h., das Ergebnis der Funktion ist nur annähernd gleich dem Anfangswert (vgl. Kapitel 2.5.1.4, Punkt 5. sowie 2.7.1)! Diese Problematik kann mit einer Anweisung der Art **D# =VAL(STR$(G!))** umgangen werden.

BEISPIELE:

```
10 A=2.35768 : B=3.75645
20 LPRINT A*B;CDBL(A*B)
 8.85651  8.856507301330 57
```

VERWEISE: Kapitel 2.4, 2.5.1 und 4.13

ANWEISUNG: **CHAIN**

FUNKTION: Aufruf eines Programmes, eventuell mit Übergabe von Variablen vom gegenwärtigen Programm an das aufgerufene sowie gegebenenfalls Laden von Overlays und/oder Löschen von Programmzeilen

FORMAT:

```
CHAIN [MERGE] Filename [,[Zeilennummer] [,ALL]
[,DELETE{Zeilennr. 1 - Zeilennr. 2}]]
        {    - Zeilennr. 2      }
```

Filename : Stringausdruck, dessen Ergebnis den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

Zeilennummer: numerischer Ausdruck, dessen Ergebnis eine Zeilennummer definiert

Zeilennr. 1,
Zeilennr. 2 : Konstante, die eine Zeilennummer angibt

WIRKUNG: Die Verarbeitung des laufenden Programmes wird beendet. Das Programm mit dem Namen 'Filename' wird von Diskette geladen und die Ausführung ab der ersten Zeile oder gegebenenfalls ab der Zeile 'Zeilennummer' begonnen.

Parameter **ALL**

Ist der Parameter **ALL** gesetzt, so werden alle Variablen des Ausgangs-Programmes an das Folgeprogramm übergeben, wobei der Wert der Variablen erhalten bleibt. Sollen nur bestimmte Variable übergeben werden, siehe unter Anweisung **COMMON**.

Parameter **MERGE**

Wird der Parameter **MERGE** gesetzt, so wird das unter 'Filename' angegebene Programm als Overlay in das gegenwärtige Programm zugeladen. Danach wird das Gesamtprogramm gestartet. Das durch 'Filename' bestimmte Programm muß in Form eines ASCII-Files auf Diskette vorliegen. (Z.B. **SAVE** "1:ABC",**A**).

Parameter **DELETE**

Der Parameter **DELETE** wirkt sich nur aus, wenn auch der Parameter **MERGE** gesetzt wurde. Es werden die unter **DELETE** angegebenen Zeilennummern des gegenwärtigen Programms gelöscht, bevor **MERGE** (Laden des Overlay-Programms) ausgeführt wird.

BEMERKUNGEN:

- Die Zeilennummern 'Zeilennr. 1' und 'Zeilennr. 2' werden durch den Befehl **RENUM** unnumeriert, während 'Zeilennummer' nicht (!) unnumeriert wird.
- Durch **CHAIN** werden Daten-Files nicht geschlossen.
- Der aktuelle Wert von **OPTION BASE** bleibt bei **CHAIN** erhalten.
- Wird der Parameter **MERGE** benutzt, so werden die Daten-Files geschlossen, alle Variablen gelöscht und die Routinen **FOR...NEXT**, **WHILE...WEND** und **GOSUB...RETURN** abgeschlossen.

BEISPIELE:

```
10 CLS : CURSOR(1,1) :INPUT "Bitte Programm waehlen (1-4)",A$
20 IF ASC(A$)<48 OR ASC(A$)>52 THEN 10
30 ON ASC(A$) GOTO 40,50,60
35 CHAIN"1:prog4"
40 CHAIN "0:prog1",50
50 CHAIN "1:prog2",,ALL
60 CHAIN MERGE "1:prog3",,ALL,DELETE 10-40
```

VERWEISE:

Kapitel: 4.3.13
Befehl: **SAVE**
Anweisung: **COMMON**

FUNKTION: **CHR$**

ZWECK: liefert dasjenige Zeichen, das durch einen ISO-Code bestimmt ist

FORMAT:

```
CHR$(num.Ausdruck)
```

WIRKUNG: Der numerische Ausdruck wird berechnet und auf Ganzzahligkeit gerundet. Es wird dasjenige Zeichen ermittelt, das dem ISO-Code des Zeichens entspricht.

BEMERKUNGEN:

- Das Ergebnis von 'num.Ausdr.' muß zwischen Ø und 256 liegen, sonst wird der Fehler "Illegal function call" gemeldet.
- **CHR$** ist die Umkehrung der Funktion **ASC**.
- Zahlreiche nicht druckbare Zeichen mit den ISO-Codes Ø bis 31 dienen als Steuerzeichen am Bildschirm oder Drucker.
- Häufig werden bestimmte Zeichen als Trennzeichen zwischen Daten eingesetzt (vgl.z.B. **PRINT#** , **INPUT#**).

BEISPIELE:

```
10 LPRINT CHR$(66)
B
```

VERWEISE: Kapitel 4.3.4.3 und 4.3.17
Funktion: **ASC**

FUNKTION: **CINT**
(convert integer)

ZWECK: Konvertiert einen numerischen Wert in einen ganzzahligen (Integer-) Wert

FORMAT:

```
CINT(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet.
Seine erste Nachkommastelle wird kaufmännisch auf den Vorkommateil gerundet.

BEMERKUNGEN:
- -Ø.5 wird zu Ø
  -Ø.6 wird zu -1
- Ist das Argument nicht im Intervall zwischen -32768 und 32767, wird "Overflow" gegeben.

BEISPIELE: **CINT**(3.7) liefert 4 und **CINT**(-3.7) liefert -4.
CINT(3.2) liefert 3 und **CINT**(-3.2) liefert -3.

```
10 DEFDBL X
20 X=32.57465846#:LPRINT X;CINT(X)
 32.57465846   33
```

VERWEISE: Kapitel 2.4, 2.5.1 und 4.13
Funktionen: **FIX**, **INT**

ANWEISUNG: **CIRCLE**

FUNKTION: Erlaubt das Zeichnen von Kreisen oder Ellipsen

FORMAT:

CIRCLE [**%** Window-Nr.,] (X,Y), Radius [, [Farbindex] [,Achsenverhältnis] [,Operand]]

Window-Nr.	:	numerischer Ausdruck (Wert: 1 bis 16)
X,Y	:	numerische Ausdrücke, die den Mittelpunkt bestimmen
Radius	:	numerischer Ausdruck, der den Radius des Kreises bestimmt
Farbindex	:	numerischer Ausdruck, der die Farbe des Kreises bezogen auf die aktuelle globale **COLOR**-Anweisung bestimmt (Wert: Ø-3)
Achsenverhältnis:		numerischer Ausdruck, der das Verhältnis der gültigen Radien von X- zu Y-Richtung bestimmt.
Operand	:	Operand kann sein: **AND**, **OR**, **XOR**, **NOT**, **PSET** oder **PRESET**

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, so wird der Kreis in diesem Window gezeichnet. Fehlt dieser Parameter, so wird der Kreis im aktiven Window gezeichnet. Das Koordinatenpaar (X,Y) bestimmt den Mittelpunkt des Kreises in Absolut-Koordinaten bezogen auf das für das Window gültige **SCALE**.

Radius

Dieser Parameter bestimmt den Radius des Kreises. Der Wert ist bezogen auf das für das Window geltende **SCALE** in horizontaler (X-)Richtung.

Farbindex

Dieser Parameter bestimmt, in welcher Farbe der Kreis gezeichnet wird. Dabei gibt der Parameter 'Farbindex' die Position (Ø-3) der entsprechenden Farbe in der globalen **COLOR**-Anweisung an.
Fehlt die Angabe des Farbcodes, so wird der Kreis in der aktuellen Vordergrundfarbe gezeichnet.

Achsenverhältnis

Fehlt dieser Parameter, so wird ein Kreis gezeichnet. Ist 'Achsenverhältnis' angegeben, so wird eine Ellipse nach folgenden Regeln gebildet:
Als Ausgangswert für die horizontale und vertikale Halbachse wird der gültige Wert von 'Radius' im jeweils gültigen Maßstab gemäß **SCALE** genommen.
Der Parameter 'Achsenverhältnis' wird für die vertikale Halbachse als Faktor für den sich aus Radius und Maßstab ergebenden Wert genommen.
Aus den sich dadurch ergebenden Werten für die Halbachse wird eine Ellipse gezeichnet.

Operand

Operand kann eines der Sprachelemente **AND**, **OR**, **XOR**, **NOT**, **PSET** oder **PRESET** sein. Es wird dann auf dem für die Darstellung maßgeblichen Bereich der Bitmap (vgl. Kapitel 4.3.18) die entsprechende logische Operation für jeden Punkt durchgeführt.
Wird **PSET** angegeben, so werden alle betroffenen Punkte auf die angegebene Farbe gesetzt, bei **PRESET** werden alle betroffenen Punkte auf die Farbe des Hintergrundes gesetzt.

BEMERKUNGEN:

- Kreise und Ellipsen werden gezeichnet, indem jeweils die Punkte in der entsprechenden Farbe dargestellt werden, die den sich aus der Rechnung ergebenden Werten am nächsten liegen.

- Es wird jeweils der Teil des Kreises oder der Ellipse gezeichnet, der innerhalb des Windows liegt.
- Je kleiner der Wert für 'Achsenverhältnis' ist, desto flacher wird die Ellipse, je größer dieser Wert ist, um so höher wird die Ellipse.
- Der durch 'Radius' bestimmte Wert bezieht sich auf die Skaleneinteilung in horizontaler Richtung.

BEISPIELE:

```
10 CLEAR
20 CIRCLE (100,100),30
30 CIRCLE (300,150),40,1,2
40 CALL"sp"
50 END
```

```
10 CLEAR
20 FOR I=1 TO 17
30 CIRCLE (250,180-5*I),40+5*I
40 CIRCLE(250,80+5*I),40+5*I
50 CIRCLE (295+5*I,130),40+5*I
60 CIRCLE (215-5*I,130),40+5*I
70 NEXT
80 CALL"sp"
90 END
```

VERWEISE : Kapitel : 4.3.14 und 4.3.18
Anweisungen: **COLOR=, SCALE**
Funktion : **WINDOW**

ANWEISUNG: **CLEAR**

FUNKTION: Löschen des Datenbereiches und der Bitmap für den Bildschirm im Arbeitsspeicher; Setzen des gesamten Bildschirms auf Default-Hintergrundfarbe; Aufhebung aller Window-Vereinbarungen; ggf. Definition der logischen Größe des Arbeitsspeichers und/oder des für Stack-Speicher zu reservierenden Platzes im Arbeitsspeicher

FORMAT:

CLEAR [, [Arbeitsspeicher-Größe] [,Stack-Größe]]

Arbeitsspeicher-Größe: num.Ausdruck
Stack-Größe: num.Ausdruck

WIRKUNG: Der gesamte Datenbereich im Arbeitsspeicher wird gelöscht (incl. aller Variablen, die aufgrund einer **COMMON**-Anweisung weitergereicht werden könnten). Bei späterem Aufruf (ohne vorherige Zuweisung) haben alle numerischen Variablen den Wert Ø und alle String-Variablen den Wert Leerstring ("").
Alle Vereinbarungen über den Bildschirm (Windows, Farbauswahl) werden aufgehoben (Setzen der PCOS-Default-Werte). Der Bildschirm wird komplett auf die Default-Hintergrundfarbe gelöscht.
Mit 'Arbeitsspeicher-Größe' kann definiert werden, auf wieviele Bytes der physisch dem Anwender zur Verfügung stehende Platz im Arbeitsspeicher reduziert werden soll.
Mit 'Stack-Größe' kann der Wert, der für Stack-Speicher im Arbeitsspeicher in Bytes standardmäßig oder aufgrund der letzten **CLEAR**-Anweisung reserviert wurde, verringert oder vergrößert werden.

BEMERKUNGEN:
- **CLEAR** schließt keine offenen Files.
- **CLEAR** löscht alle aktiven Windows und setzt den Bildschirm auf die Hintergrundfarbe zurück.
- Die Summe aus 'Arbeitsspeicher-Größe' und 'Stack-Größe' muß kleiner sein als der mit PCOS-Befehl **sb** zur Verfügung des Anwenders freigegebene Platz im Arbeitsspeicher (normalerweise 38ØØØ Bytes).

- Für einen Stack-Speicher müssen zwischen 9.7 Bytes (bei einer sehr geringen Anzahl) und 6.1 Bytes (bei einer sehr hohen Anzahl) kalkuliert werden.
- Eine durch **CLEAR** definierte 'Arbeitsspeicher-Größe' kann nur durch Neu-Aktivieren des Systems über die PCOS-Ebene rückgängig gemacht werden.
- Eine durch **CLEAR** definierte 'Stack-Größe' kann (auch in neuen Programmen) erst durch eine neue **CLEAR**-Anweisung oder durch Neu-Aktivieren des Systems über die PCOS-Ebene rückgängig gemacht werden.

BEISPIELE:

```
10 CLEAR ,20000
```

```
10 CLEAR ,,3000
```

VERWEISE: Kapitel 4.3.2 und 4.3.5
Befehle: **RUN**, **LOAD**
Anweisungen: **GOSUB**, **ON...GOSUB**, **RETURN**, **FOR...NEXT**, **WHILE...WEND**, **COMMON**, **CHAIN**
Fehler-Code: 7

ANWEISUNG: **CLOSE**

FUNKTION: Schließen von einem, mehreren oder allen externen Datenfiles

FORMAT:

CLOSE [[#] Filenr. [, [#] Filenr.]] ...

Filenr.: numerischer Ausdruck

es ist die Nummer anzugeben, mit der das Datenfile bei der Anweisung **OPEN** geöffnet wurde.

WIRKUNG: Es wird der ganzzahlige Teil des numerischen Ausdrucks ermittelt.

Daten, die sich noch im File-Puffer befinden und noch nicht gespeichert sind, werden automatisch in das entsprechende File geschrieben, wenn:

- sich ein sequentielles File im Output-Mode befindet (siehe Parameter "O" und "A" bei der Anweisung **OPEN**)
- bei einem Random-File die Anweisung **PUT** ausgeführt, aber das Schreiben physisch noch nicht erfolgt ist.

Sämtliche Informationen, die sich auf das File beziehen, werden anschliessend im Arbeitsspeicher gelöscht, d.h. die dem File zugeordnete Nummer wird wieder frei.

BEMERKUNGEN:

- Die **END**-Anweisung sowie die Befehle **NEW** und **RUN** bewirken automatisch ein **CLOSE** für alle geöffneten Files.
- Files sollten im Programm immer durch **CLOSE**-Anweisungen geschlossen werden. Von jedem anderen Schliessen ist wegen des möglichen Datenverlustes abzusehen.
- Die **STOP**-Anweisung schliesst Files nicht.

BEISPIELE:

```
10 REM Oeffnen der Files 'sequen','random','sequen1'
20 OPEN "I",#1,"1:sequen"
30 A$="1:sequen1"
40 OPEN "I",#2,A$
50 OPEN "R",#3,"1:random"
70 '
150 REM Schliessen der Files
160 A=1
170 CLOSE A,2:CLOSE 3
```

VERWEISE:

Kapitel : 4.3.10.2
Anweisung: **OPEN**
Befehle : **RUN, NEW**

ANWEISUNG: **CLOSE WINDOW**

FUNKTION: Auflösen von Windows

FORMAT:

CLOSE WINDOW [%Window-Nr.]

Window-Nr.: numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen 2 und 16 enthalten muß

WIRKUNG: Wird nur **CLOSE WINDOW** angegeben, so werden alle zur Zeit definierten Windows mit Ausnahme des Windows 1 aufgelöst. Es existiert also nur noch ein Window (Gesamt-Schirm), und dieses ist aktiv.
Wird die 'Window-Nr.' angegeben, so wird dieses Window aufgelöst und der freiwerdende Bereich nach der Regel der binären Baumstruktur anderen Windows zugeordnet. Dieser Vorgang wird in Kapitel 4.3.18 beschrieben.
Es gelten dann für den Gesamtbereich aus aufgelöstem Window und dem Window, dem dieser Bereich zugeordnet wurde, die Parameter des verbleibenden Windows.
Der Bereich des/der aufgelösten Windows wird vollständig mit der Hintergrundfarbe gefüllt (wie bei **CLS**).

BEMERKUNGEN:

- Window 1 kann nie aufgelöst werden.
- Man beachte den Unterschied zwischen dem Auflösen eines Windows mit **CLOSE WINDOW** und dem Löschen des Inhaltes eines Windows mit **CLS**.
- Der durch die Auflösung des Windows frei werdende Bereich wird einem anderen Window zugeordnet. Da dieses Window sich dadurch vergrößert, ergibt sich eine Änderung des geltenden Maßstabes in diesem Window.
- **CLEAR** löst ebenfalls alle offenen Windows auf.
- Es ist empfehlenswert, vor dem Programmende alle eröffneten Windows aufzulösen, da andernfalls beim nachfolgenden Arbeiten mehrere eröffneten Windows behindern oder zu Fehlern führen können.

BEISPIELE:

```
10 CLEAR
20 W=WINDOW(0,0)
30 W1=WINDOW(3,40)
40 WINDOW%W
50 FOR I=1 TO 20:PRINT "das ist Window1":NEXT
60 WINDOW%W1
70 COLOR 0,1
80 FOR I=1 TO 20 :PRINT "das ist Window2":NEXT:COLOR 1,0
90 FOR I=1 TO 3000:NEXT
100 CLS %W
110 WINDOW%W
120 FOR I=1 TO 10:PRINT" Window2 wird geschlossen":NEXT
130 FOR I=1 TO 3000 :NEXT
140 CLOSE WINDOW %W1
150 FOR I=1 TO 30 :PRINT" es wird wieder mit dem ganzen Bild
schirm gearbeitet":NEXT

160 END
```

VERWEISE: Kapitel 4.3.18
Anweisungen: **WINDOW%, CLEAR, CLS, SCALE**
Funktion: **WINDOW**

ANWEISUNG: **CLS**
(clear screen)

FUNKTION: Löscht den Inhalt des aktiven oder des angegebenen Windows

FORMAT:

```
CLS [%Window-Nr.]
```

Window-Nr.: numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen 1 und 16 enthalten muß

WIRKUNG: Wird nur **CLS** angegeben, so wird der Inhalt des zur Zeit aktiven Windows gelöscht und das Window in der Hintergrundfarbe dargestellt. Wird zusätzlich die 'Window-Nr.' angegeben, so wird der Inhalt des dadurch spezifizierten Windows gelöscht und das Window in der Hintergrundfarbe dargestellt.

BEISPIELE:

```
10 CLS
20 FOR I=1 TO 10
30 PRINT " TEXT TEXT TEXT TEXT TEXT TEXT TEXT TEXT TEXT TEXT
"
40 NEXT
70 PRINT:PRINT:PRINT " in ca. 2 Sekunden wird der Bildschirm
 mit 'CLS' gelöscht":PRINT
90 FOR I=1 TO 2000:NEXT  CLS : END
```

VERWEISE: Kapitel 4.3.14

ANWEISUNG: **COLOR**

FUNKTION: Auswahl oder Neufestlegung von Vorder- und Hintergrundfarbe in einem Window

FORMAT:

COLOR [%Window-Nr.,] Vordergrundfarbe [,Hintergrundfarbe]

Window-Nr. : num. Ausdruck (Wert: Ø-3)

Vordergrundfarbe: num. Ausdruck (Wert: Ø-3), dessen Ergebnis die Position der gewünschten Vordergrundfarbe in der globalen **COLOR**=-Anweisung bestimmt ('Farbindex')

Hintergrundfarbe: num. Ausdruck (Wert: Ø-3), dessen Ergebnis die Position der gewünschten Hintergrundfarbe in der globalen **COLOR**=-Anweisung bestimmt ('Farbindex')

WIRKUNG: Wird der Parameter **%** Window-Nr. angegeben, bezieht sich die Anweisung auf das damit bestimmte Window, andernfalls auf das aktive Window.
Mit 'Vordergrundfarbe' wird festgelegt, die wievielte der in der globalen **COLOR**=-Anweisungen zur Verfügung gestellten Farben zur Vordergrundfarbe werden soll.
Mit 'Hintergrundfarbe' wird die Hintergrundfarbe ausgewählt.

BEMERKUNGEN:

- 'Vordergrundfarbe' und 'Hintergrundfarbe' sind 'Farbindex' (vgl. Kapitel 4.3.18.3) und geben an, der wievielte Farbcode der globalen **COLOR**=-Anweisung gewünscht wird. Dabei steht Ø für den dort zuerstgenannten, 1 für den zweitgenannten, 2 für den drittgenannten und 3 für den zuletztgenannten.
  Wurde noch keine **COLOR**=-Anweisung abgearbeitet, erfolgt die Auswahl aus den Default-Werten, bei denen die Reihenfolge der 'Farbcodes' festliegt ('Farbindex', vgl. **COLOR**=-Anweisung).

- Wird 'Hintergrundfarbe' nicht angegeben, bleibt die bisherige Hintergrundfarbe erhalten. Unterscheidet sich dabei die Vordergrundfarbe nicht mehr von der aktuellen Hintergrundfarbe, wird die **COLOR**-Anweisung jedoch nicht beachtet.
- Die **COLOR**-Anweisung wirkt sich nicht sofort auf das gesamte Window aus, sondern erst bei den einzelnen Anweisungen, die den Inhalt des Windows ändern, (z.B. **PRINT**, **CLS** oder graphische Anweisungen). Es wird immer nur die aktuell angesproche Window-Zeile angepaßt, und zwar von der Anfangsposition (die z.B. durch **CURSOR** festgelegt wurde), aber grundsätzlich bis zum äußersten rechten Window-Rand.

BEISPIELE:

```
10 CLS
20 COLOR 1,0
30 EXEC "la 'color 1,0',20,20,5,0"
40 COLOR 0,1
50 EXEC "la 'color 0,1',20,200,5,0"
60 CURSOR (5,10):PRINT "dieser Text wurde mit 'PRINT' geschr
ieben"
70 CALL "sp"
80 COLOR 1,0:CLS:END
```

VERWEISE: Kapitel 4.3.18
Anweisungen: **COLOR=**, **WINDOW%**, **CLS**

ANWEISUNG: **COLOR=**
(globale **COLOR**-Anweisung)

FUNKTION: Auswahl von vier aus acht Farben für das Arbeiten mit dem Farbschirm

FORMAT:

```
COLOR = Farbcode1, Farbcode2, Farbcode3, Farbcode4
```

Farbcode: numerischer Ausdruck, dessen Ergebnis gerundet wird und eine Zahl zwischen Ø und 7 ergibt

WIRKUNG: Die Anweisung wirkt sich nur auf den Farbbildschirm aus. Beim Bildschirm schwarz/weiß hat sie keine Bedeutung. Es werden durch Angabe der Farbcodes aus der nachstehenden Tabelle 4 Farben ausgewählt, die dann gleichzeitig auf dem Bildschirm dargestellt werden können:

Farbcode	Farbe
Ø	schwarz
1	grün
2	rot
3	gelb
4	blau
5	cyan (türkis)
6	magenta (purpur)
7	weiß

Farbtabelle für farbigen Bildschirm

BEMERKUNGEN:
- Wird keine **COLOR=**-Anweisung benutzt, so werden vom System die folgenden Defaultwerte gesetzt:

einfarbiger Schirm	Farb-Schirm	Farb-Code	Farb-index
schwarz	schwarz	Ø	Ø
weiß	grün	1	1
	rot	2	2
	blau	4	3

Defaultwerte für Bildschirm

- Die an zweiter Position genannte Farbe ist die Vordergrundfarbe.
- Die zuerst genannte Farbe (Default-Wert: schwarz) ist die Hintergrundfarbe.
- Die Vordergrundfarbe hat die Defaultwerte weiß (einfarbiger Schirm) und grün (Farbschirm).
- Die Vorder- und Hintergrundfarbe kann mit der Anweisung **COLOR** für jedes Window definiert oder geändert werden.
- Die mit **COLOR=** ausgewählten Farben gelten für alle Windows, d.h., in allen Windows kann nur mit diesen Farben gearbeitet werden.
- Die bis zu vier ausgewählten Farben sind für alle Anweisungen, die später das Arbeiten mit Farben ermöglichen, unter den Nummern Ø bis 3 ('Farbindex') in der Reihenfolge der mit 'Farbcode' in der **COLOR=**-Anweisung ausgewählten Farben anzusprechen.

BEISPIELE:

```
10 COLOR=3,6,4,7  :   REM nur Farbschirm
20 REM Damit ist die Hintergrundfarbe gelb und die Vorder
grundfarbe magenta. Ausserdem stehen die Farben blau und
weiss zur Verfuegung.
30 REM DIE ANWEISUNG COLOR 3,1 BEWIRKT, DASS VON DA AN AL
LE DARSTELLUNGEN IN DER VORDERGRUNDFARBE  WEISS UND DER H
INTERGRUNDFARBE MAGENTA ERFOLGEN.
```

VERWEISE: Kapitel 4.3.14 und 4.3.18.3
Anweisung: **COLOR**

ANWEISUNG: **COMMON**

FUNKTION: Definition der Variablen in einem BASIC-Programm, deren Werte bei der Verkettung von Programmen erhalten bleiben

FORMAT:

COMMON {Array ( )} [,{Array ( )}] ...
{Variable} [{Variable}]

Array : Name des Arrays (String oder numerisch)

Variable: Name der String- oder numerischen Variablen

WIRKUNG: Die **COMMON**-Anweisung legt für das laufende Programm die Variablen fest, die ihre Werte an ein anderes Programm übergeben und/oder von einem anderen Programm Werte übernehmen sollen. Der Speicherplatz für die Variablen der **COMMON**-Anweisung wird in der dort festgelegten Reihenfolge mit den gültigen Deklarationen und Dimensionen im Common-Bereich des Arbeitsspeichers reserviert.

BEMERKUNGEN:

- **COMMON**-Anweisungen sind im Zusammenhang mit der Anweisung **CHAIN** zu verwenden.
- Ein Programm kann eine oder mehrere **COMMON**-Anweisungen enthalten.
- Die Speicherzuweisung an die **COMMON**-Variablen erfolgt in der Reihenfolge, in der die Variablen in der Anweisung gelistet sind.
- Die mit **CHAIN** verketteten Programme müssen die gleiche Anzahl **COMMON**-Variablen haben, die in ihrer Reihenfolge nach dem Typ (integer, single precision, double precision und String) übereinstimmen. Die Variablennamen können jedoch verschieden sein.
- Beziehen sich Deklarationsanweisungen (wie **DEFINT**, **DEFSNG**, **DEFDBL** oder **DEFSTR**) auf **COMMON**-Variablen, so sind diese im Folgeprogramm zu wiederholen. Dabei müssen die Deklarationsanweisungen jeweils vor den **COMMON**-Anweisungen stehen.

- Die Arrays der **COMMON**-Anweisung müssen durch die Anweisung **DIM** dimensioniert sein. Diese muß vor der **COMMON**-Anweisung stehen.
- Eine Variable darf innerhalb der Gruppe der **COMMON**-Anweisungen nur einmal auftreten.

BEISPIELE:

```
10 COMMON A()
20 DIM A(5,2)
30 FOR I=1 TO 5:FOR K=1 TO 2
40 A(I,K)=I*K
50 NEXT K,I
60 CHAIN "1:common1"
```

```
10 COMMON A()
20 FOR I=1 TO 5:FOR K=1 TO 2
30 LPRINT USING "  #####.###  ";A(I,K);

40 NEXT K,I
50 END
      1.000          2.000          2.000          4.000          3.
000          6.000          4.000          8.000          5.000
  10.000
```

VERWEISE : Kapitel 4.3.13

BEFEHL: **CONT**

FUNKTION: Fortsetzung der Programmausführung nach einer Unterbrechung

FORMAT:

```
CONT
```

WIRKUNG: Die Programmausführung wird mit der Anweisung fortgesetzt, die auf diejenige Anweisung folgt, bei der das Programm unterbrochen wurde.

BEMERKUNGEN:

- Eine Programmunterbrechung kann erfolgen durch
  1. Betätigen der Taste C
  2. Ausführen der Anweisung **STOP**
  3. Auftreten einer Fehlermeldung
- Beim Auftreten einer Fehlermeldung kann mit der Programmausführung nur dann fortgesetzt werden, wenn die Fehlermeldung keinen Fehler anzeigt, der die sinnvolle Fortsetzung des Programms verhindert (vgl. Kapitel 7).
- Wird während der Ausführung der Anweisung **INPUT** oder **LINE INPUT** die Taste C gedrückt und anschließend mit **CONT** die Programmausführung fortgesetzt, so wird die Anweisung **INPUT** oder **LINE INPUT** wiederholt.
- Wurde vor **CONT** im Edit-Mode gearbeitet bzw. im Command-Mode eine neue Anweisungszeile eingegeben, sind alle Variableninhalte wieder auf Ø bzw. Leerstring gesetzt sowie alle Stack-Speicher (z.B. für **GOSUB...RETURN** oder **FOR...NEXT**) gelöscht. In diesem Fall kann die Meldung "Can't continue" erscheinen, falls in einer Schleife unterbrochen wurde.

BEISPIELE:

```
10 LPRINT "das Programm wird mit 'STOP' unterbrochen"
20 LPRINT "zur Fortsetzung des Programmes ist 'CONT' einzuge
ben"
30 LPRINT
40 STOP
50 LPRINT "das Programm wird mit Zeile 50 fortgesetzt"
60 END
das Programm wird mit 'STOP' unterbrochen
zur Fortsetzung des Programmes ist 'CONT' einzugeben

das Programm wird mit Zeile 50 fortgesetzt
```

VERWEISE: Kapitel 3
Befehl: **STOP**

FUNKTION: COS

ZWECK: liefert den Cosinus eines numerischen Werts im Bogenmaß

FORMAT:

```
COS(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet, als Bogenmaß interpretiert und davon der Cosinus ermittelt.

BEMERKUNGEN:

- Das Argument ist im Bogenmaß anzugeben. Umrechnungen müßten ggf. über Unterprogramm oder eine vom Anwender definierte Funktion (s. **DEF FN**) erfolgen.
- Das Argument muß im Intervall -65537.574≤x≤65534.427 liegen, sonst erfolgt die Fehlermeldung "Illegal function call".
- Der Arcus-Cosinus steht nicht als Standardfunktion zur Verfügung. Er kann berechnet werden über ARCCOS(X)=**ATN(X/SQR(-X*X+1))**

BEISPIELE:

```
5 PI=3.14
10 X=30:LPRINT"Winkel x=";X;"Grad","Cosinus x=";COS(X*PI/200)
Winkel x= 30 Grad            Cosinus x= .891115
```

VERWEISE: Kapitel 2.4 und 4.13

FUNKTION: **CSNG**
(convert single precision)

ZWECK: Konvertiert einen numerischen Wert in einfache Genauigkeit

FORMAT:

```
CSNG(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Als Ergebnis steht er in einfacher Genauigkeit zur Verfügung.

BEMERKUNGEN:
- Die siebte signifikante Stelle von doppelt genauen Werten wird kaufmännisch gerundet.

ACHTUNG:- Werden doppelt genaue Werte außerhalb des für einfach genaue Werte zulässigen Intervalls (vgl. Kapitel 2.5.1.2) konvertiert, sind zwei Fälle zu unterscheiden:
1. Liegt der zu konvertierende doppelt genaue Wert außerhalb des Intervalls
   $-1.80925139D75 \leq x \leq 1.80925139D75$
   wird die Meldung "Overflow" gegeben. Es wird jedoch mit den verschiedensten völlig falschen Werten weitergerechnet!!!
2. Liegt der zu konvertierende doppelt genaue Wert zwar unterhalb des Intervalls zu 1., aber über dem Intervall
   $-3.40282D38 \leq x \leq 3.40282D38$,
   erfolgt keine Overflow-Meldung. Intern wird mit dem niedrigsten bzw. höchsten in einfacher Genauigkeit intern darstellbaren Wert (vgl. Intervall) weitergerechnet!

BEISPIELE:

```
10 DEFDBL X-Z:X=2.34567:Y=3.97654
20 Z=X*Y:LPRINT Z;:Z  =CSNG(X*Y):LPRINT Z
30 END
 9.32765072937821   9.3[illegible]5102386475
```

VERWEISE: Kapitel 2.4, 2.5.1 und 4.13

ANWEISUNG: **CURSOR**

FUNKTION: Steuerung des Text- oder des Graphik-Cursors und gegebenenfalls Festlegung der Cursor-Eigenschaften

FORMAT:

```
CURSOR [POINT] [(X,Y)] [Schalter] [,Rate] [,Darstellung]
```

X,Y : numerische Ausdrücke, dessen Ergebnisse die Position des Cursors bestimmen

Schalter : numerischer Ausdruck, dessen Ergebnis gerundet wird und dann Ø oder 1 enthalten muß

Rate : numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen Ø und 2Ø enthalten muß

Darstellung: Element eines eindimensionalen Integer-Arrays, ab dem die Cursor-Darstellung gespeichert ist.

WIRKUNG: Der alphanumerische oder der graphische Cursor wird in der durch den Parameter 'Darstellung' definierten Form auf die Position (X,Y) gesetzt. Dabei bestimmt der Parameter 'Schalter', ob der Cursor sichtbar ist oder nicht und der Parameter 'Rate' legt die Rate fest, mit der der sichtbare Cursor blinkt.

POINT

Wird das Schlüsselwort **POINT** angegeben, so bezieht sich die Anweisung auf den graphischen Cursor und somit auf die durch **SCALE** definierten Anwender-Koordinaten.
Fehlt das Schlüsselwort **POINT**, so wird der Text-Cursor über Textspalte und Textzeile angesprochen.

X,Y

Die Ergebnisse der numerischen Ausdrücke werden auf Ganzzahligkeit gerundet. Beim Text-Cursor bestimmt X die Text-Spalte und Y die Text-Zeile, in der der Cursor positioniert wird. Beim graphischen Cursor bestimmt X die horizontale und Y die vertikale Anwender-Koordinate (vgl. **SCALE**).

Schalter

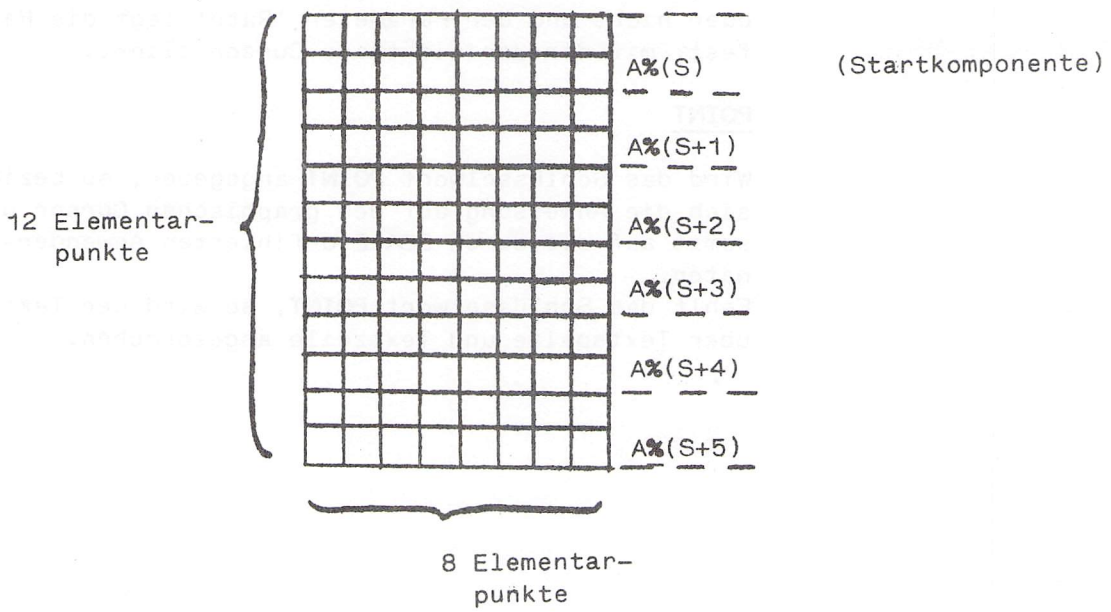
Hat das gerundete Ergebnis des numerischen Ausdrucks 'Schalter' den Wert Ø, so wird der gesetzte Cursor nicht sichtbar angezeigt. Besitzt es den Wert 1, so wird der Cursor auf der definierten Position angezeigt.

Rate

Durch den Parameter 'Rate' wird die Rate bestimmt, mit der der Cursor blinkt. Dabei können Werte von Ø bis 2Ø benutzt werden. Ist der Wert gleich Ø, so bleibt der Cursor ständig sichtbar, d.h. er blinkt nicht. Ist der Wert z.B. 5, heißt das, der Cursor soll pro Sekunde 5 mal blinken.

Darstellung

Der Cursor besteht aus einer 8 x 12-Matrix von Elementarpunkten:



Ordnet man je zwei Reihen von Elementarpunkten (à 8 Punkten) einer Komponenten eines eindimensionalen Integer-Arrays zu, so kann die Darstellung des Cursors, die innerhalb der 8x12 Matrix frei wählbar ist, in einem eindimensionalen Integer-Array von 6 Komponenten gespeichert werden. Durch Angabe der ersten Komponente - hier 'Startkomponente' genannt - dieses 6-elementigen (Teil-) Arrays (im Bild: Elemente S bis S+5) ist die Cursor-Darstellung eindeutig definiert.
In den 2 Bytes einer Komponente des Integer-Arrays wird die 16-Bit-Maske einer Zweierreihe von Elementarpunkten gespeichert.

BEMERKUNGEN:

- Es ist immer nur ein Cursor aktiv und zwar entweder der graphische oder der alphanumerische.
- Die Anweisung **CURSOR** bezieht sich immer auf das momentan aktive Window.
- Die Startkomponente des durch 'Darstellung' definierte eindimensionalen Arrays muß nicht die Komponente Ø oder 1 (bzgl. **OPTION BASE**) sein, sondern kann ein beliebiges Array-Element sein, dem 5 weitere folgen.
- Die **CURSOR**-Anweisung hat solange Gültigkeit, bis eine neue definiert wird, oder durch RESET bzw. Ausschalten das System wieder der Default-Cursor gesetzt wird.

BEISPIELE:

```
10 CLS
20 DEFINT A,B
30 A(1)=32:A(2)=4104:A(3)=3086:A(4)=3596:A(5)=2064:A(6)=8192
40 CURSOR (2,4):PRINT"Cursor in Spalte 2,Zeile 5 (er wird ni
cht angezeigt)
50 CURSOR (2,5)0:GOSUB 180
60 CURSOR (2,6):PRINT"Cursor in Spalte 2,Zeile 7 (blinkend)"
70 CURSOR (2,7)1:GOSUB 180
80 CURSOR (2,8):PRINT"Cursor in Spalte 2,Zeile 9 (10*blinken
d/sek)"
90 CURSOR (2,9)1,10:GOSUB 180
100 CURSOR (2,10):PRINT"cursor in Spalte 2,Zeile 11 (neue Fo
rm)"
110 CURSOR (2,11)1,4,A%(1):GOSUB 180
120 FOR I=1 TO 5:B(I)=21802:NEXT:B(6)=21760
130 CURSOR (2,11)1,2,B%(1)
140 CURSOR (2,15):PRINT "der graphische Cursor wird angezeig
t"
150 CURSOR POINT(300,100)1,1:GOSUB 180
160 CURSOR (1,1)1,2,B%(1):CLS
170 END
180 FOR I=1 TO 8000:NEXT:RETURN
```

VERWEISE : Kapitel 4.3.14 und 4.3.18
Funktion: **WINDOW, POS**

FUNKTION: **CVD**
(convert double)

ZWECK: Verwandlung der ersten 8 Bytes eines Strings in einen doppelt genauen numerischen Wert

FORMAT: **CVD**(Stringausdr.)

WIRKUNG: Der Stringausdruck, der vorher meist über **MKD$** gewonnen wurde, wird berechnet. Seine ersten acht Bytes werden in einen numerischen Wert (doppelt genaue Zahl) verwandelt.

BEMERKUNGEN:
- **CVD** ist die Umkehrfunktion von **MKD$**.
- Ist 'Stringausdruck' mehr als 8 Zeichen lang, werden nur die ersten acht Zeichen berücksichtigt.
- Ist 'Stringausdruck' weniger als 8 Zeichen lang, wird die Fehlermeldung "Illegal function call" gegeben.

BEISPIELE:

```
10 CLS:D#=9.99999999999998D+149 : S!=1E-20 : I%=32000
20 D$=MKD$(D#):S$=MKS$(S!):I$=MKI$(I%)
30 LPRINT "D$ ist";LEN(D$);"Bytes lang und ist numerisch";CV
D(D$)
40 LPRINT "S$ ist";LEN(S$);"Bytes lang und ist numerisch";CV
S(S$)
50 LPRINT "I$ ist";LEN(I$);"Bytes lang und ist numerisch";CV
I(I$)
D$ ist 8 Bytes lang und ist numerisch
 9.99999999999998D+149
S$ ist 4 Bytes lang und ist numerisch 1E-20
I$ ist 2 Bytes lang und ist numerisch 32000
```

VERWEISE: Kapitel 4.3.10.2
Funktion **MKD$**

FUNKTION: **CVI**
(convert integer)

ZWECK: Verwandlung der ersten 2 Bytes eines Strings in einen Integer-Wert

FORMAT:

```
CVI(Stringausdr.)
```

WIRKUNG: Der Stringausdruck, der vorher meist über **MKI$** gewonnen wurde, wird berechnet. Seine ersten zwei Bytes werden in einen numerischen Wert (Integerzahl) verwandelt.

BEMERKUNGEN:

- **CVI** ist die Umkehrung der Funktion **MKI$**.
- War ein Wert einfach oder doppelt genau, bevor er mit **MKI$** in einen String verwandelt wurde, wird bei der Rückumwandlung mit **CVI** sein ganzzahliger Teil (vgl. Kapitel 2.5.1.1) erzeugt.
- Ist 'Stringausdruck' mehr als 2 Zeichen lang, werden nur die ersten zwei Zeichen berücksichtigt.
- Ist 'Stringausdruck' weniger als 2 Zeichen lang, wird die Fehlermeldung "Illegal function call" gegeben.

BEISPIELE:

```
10 CLS:D#=9.99999999999998D+149 : S!=1E-20 : I%=32000
20 D$=MKD$(D#):S$=MKS$(S!):I$=MKI$(I%)
30 LPRINT "D$ ist";LEN(D$);"Bytes lang und ist numerisch";CV
D(D$)
40 LPRINT "S$ ist";LEN(S$);"Bytes lang und ist numerisch";CV
S(S$)
50 LPRINT "I$ ist";LEN(I$);"Bytes lang und ist numerisch";CV
I(I$)
D$ ist 8 Bytes lang und ist numerisch
 9.99999999999998D+149
S$ ist 4 Bytes lang und ist numerisch 1E-20
I$ ist 2 Bytes lang und ist numerisch 32000
```

VERWEISE: Kapitel 4.3.10.2
Funktion: **MKI$**

FUNKTION: **CVS**
(convert single)

ZWECK: Verwandlung der ersten 4 Bytes eines Strings in einen einfach genauen numerischen Wert

FORMAT:

```
CVS(Stringausdr.)
```

WIRKUNG: Der Stringausdruck, der vorher meist über **MKS$** gewonnen wurde, wird berechnet. Seine ersten vier Bytes werden in einen numerischen Wert (einfach genaue Zahl) verwandelt.

BEMERKUNGEN:

- **CVS** ist die Umkehrung der Funktion **MKS$**.
- War ein Wert doppelt genau, bevor er mit **MKS$** in einen String verwandelt wurde, ist er nach der Rückumwandlung mit **CVS** an der siebten signifikanten Stelle kaufmännisch gerundet.
- Ist 'Stringausdruck' mehr als 4 Zeichen lang, werden nur die ersten vier Zeichen berücksichtigt.
- Ist 'Stringausdruck' weniger als vier Zeichen lang, wird die Fehlermeldung "Illegal function call" gegeben.

BEISPIELE:

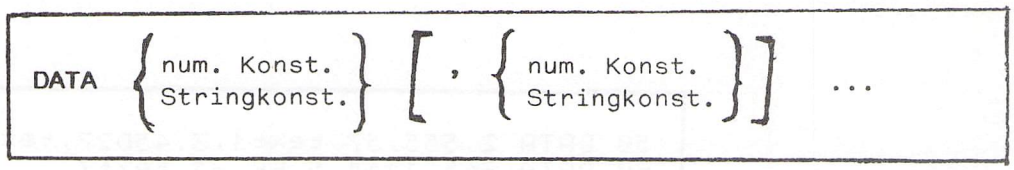
```
10 CLS:D#=9.999999999999998D+149 : S!=1E-20 : I%=32000
20 D$=MKD$(D#):S$=MKS$(S!):I$=MKI$(I%)
30 LPRINT "D$ ist";LEN(D$);"Bytes lang und ist numerisch";CV
D(D$)
40 LPRINT "S$ ist";LEN(S$);"Bytes lang und ist numerisch";CV
S(S$)
50 LPRINT "I$ ist";LEN(I$);"Bytes lang und ist numerisch";CV
I(I$)
D$ ist 8 Bytes lang und ist numerisch
 9.999999999999998D+149
S$ ist 4 Bytes lang und ist numerisch 1E-20
I$ ist 2 Bytes lang und ist numerisch 32000
```

VERWEISE: Kapitel 4.3.10.2
Funktion: **MKS$**

ANWEISUNG: **DATA**

FUNKTION: Erzeugung eines internen Files von Daten, die anschliessend den Variablen aus **READ**-Anweisungen zugewiesen werden

FORMAT:



WIRKUNG: Vor Beginn der Programmausführung wird im Arbeitsspeicher eine Tabelle erzeugt, die alle Konstanten aller **DATA**-Anweisungen des Programmes enthält. Ein Pointer zeigt auf das erste Element der Tabelle. (Die Konstanten der Tabelle werden später aus **READ**-Anweisungen auf Variablen zugewiesen). Nach jeder Zuweisung zeigt der Pointer auf das jeweils nächste Element der Tabelle. (Mit der Anweisung **RESTORE** kann der Pointer wieder auf das erste Element einer **DATA**-Zeile zurückgesetzt werden.)

BEMERKUNGEN:

- Die **DATA**-Anweisungen können an jeder Stelle im Programm stehen. Die Reihenfolge des Abstellens der Konstanten im internen Datenfile wird bestimmt durch die physische Reihenfolge der **DATA**-Anweisungen gemäß aufsteigender Zeilennummer.
- **DATA**-Anweisungen werden beim Programmlauf selbst nicht mehr ausgeführt.
- Folgen innerhalb einer Programmzeile einer **DATA**-Anweisung andere mit : angehängte Anweisungen, werden diese dem logischen Programmablauf gerecht abgearbeitet.
- **DATA**-Anweisungen können zwar angesprungen werden, erzeugen dabei aber keine neuen Elemente im internen Datenfile!
- Stringkonstanten in einer **DATA**-Anweisung, die am Anfang oder am Ende ein Blank oder irgendwo ein Komma enthalten, müssen in Anführungszeichen stehen. Ansonsten können die Anführungszeichen weggelassen werden.
- Anführungszeichen können nicht als Bestandteile von Strings übernommen werden.
- Die Konstanten-Typen (String, numerisch) in der **DATA**-Anweisung müssen mit den Variablen-Typen der **READ**-Anweisung übereinstimmen.

- **DATA**-Anweisungen können zwar im Direkt-Mode ohne Fehlermeldung eingegeben werden, haben aber keine Wirkung.

BEISPIELE:

```
50 DATA 2,555,57,text1,3,45D27,text2
60 DATA 345,1111,5,66,-1,text3
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **READ**, **RESTORE**

SPEZIELLE VARIABLE: **DATE$**

FUNKTION: dient zur Verwaltung des von der Echtzeituhr gesteuerten Datums

FORMAT:

DATE$

WIRKUNG: Die Variable **DATE$** enthält das aktuelle Datum in der Form:

TT/MM/JJJJ

Der Variablen **DATE$** können Werte zugewiesen werden; die ein gültiges Datum darstellen.
Gültig ist:

01≤TT≤31 (Tag)
01≤MM≤12 (Monat)
00≤JJ≤99 (Jahr)
oder 1000≤JJ≤9999

und ein beliebiges druckbares Trennzeichen, das keine Ziffer sein darf.

Eine Zuweisung in diesem Format bewirkt eine Übernahme vom System als gültiges Datum.
Wurde das Jahr zweistellig zugewiesen, so wird automatisch 1900 addiert.

Wird versucht, **DATE$** einen String zuzuweisen, der nicht diesem Format entspricht, so erfolgt keine Zuweisung und das aktuelle Datum bleibt unverändert.

BEMERKUNGEN:

- Die Variable **DATE$** ist auch ohne Wertzuweisung definiert, da der Wert vom PCOS-Befehl **ss** ständig zur Verfügung steht.
- Die Überprüfung des Tagesdatums erfolgt nicht in Abhängigkeit vom geltenden Monat, d.h. der 31.02.82 ist ein nach dem Format zulässiges Datum.
- Das Datum wird von der Echtzeituhr richtig auch unter Berücksichtigung der Schaltjahre verwaltet. Das Datum wird vom System jeweils verändert, wenn die Uhrzeit 00/00/00 beträgt (vgl. **TIME$**).

- **DATE$** darf in allen Anweisungen verwendet werden, die keine Wertänderung bewirken. Die Wertänderung ist nur über die Anweisung **LET** möglich. Das Schlüsselwort **LET** muß allerdings weggelassen werden.

BEISPIELE:

```
10 INPUT "Datum (TT/MM/JJ)";A$
20 INPUT "Zeit  (HH:MM:SS)";B$
30 C$="ss "+A$+","+B$
40 EXEC C$
50 LPRINT DATE$, TIME$
60 EXEC "sp" ' nur PR2400
03/04 1982     16/26/35
Datum (TT/MM/JJ)? 03/04/82

Zeit  (HH:MM:SS)? 16/26/35

 Date = 03/04/1982  Time = 16/26/35  Disk Verify = 0 (Off)

 Extent size = 8  Display = 0 (64 x 16)
```

VERWEISE: Kapitel 4.3.11
Spezielle Variable: **TIME$**

ANWEISUNG: **DEFDBL**
(define double precision)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als doppelt genaue numerische Variablen

FORMAT:

DEFDBL Anf.b.st.1 [-Anf.b.st.2] [,Anf.b.st.k [-Anf.b.st.1]] ...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä,ö,ü, Ä,Ö,Ü und ß); gibt den Anfangsbuchstaben der als doppelt genau vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st. 1' (incl.) und 'Anf.b.st. 2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von # an den Namen - als doppelt genaue numerische Variablen interpretiert.

BEMERKUNGEN:

- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2', ebenso 'Anf.buchst.2' näher an 'Anf.b.st.1'als 'Anf.b.st.k' usw.
- Eine neue Typvereinbarung durch **DEFSNG**, **DEFINT** oder **DEFSTR** hebt **DEFDBL** auf. Dabei erhalten ohne Fehlermeldung alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen wieder den Default-Wert (Ø für numerische; ""(=Leerstring) für Strings)!
- Beim Verketten von Programmen mit Variablenübergabe (vgl. Anweisungen **CHAIN** und **COMMON**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen.
- **DEFDBL** bezieht sich bei einer vom Anwender definierten Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.

- Durch Anhängen von **%**,**!** oder **$** an Variablennamen, die wegen **DEFDBL** normalerweise doppelt genaue Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.
  ACHTUNG: Ist z.B. **DEFDBL** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNR und KUNDENNR# nicht mehr möglich. Eine Wertänderung von KUNDENNR# bewirkt dann automatisch eine Wertänderung von KUNDENNR .

BEISPIELE:

```
10 DEFDBL X-Y,Z
20 A#=678901.23456789#
30 XKOORD=A#-200000!:YKOORD=A#:ZKOORD=525.256
40 LPRINT XKOORD,YKOORD,ZKOORD
 478901.23456789              678901.23456789
 525.255981445313
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2
Anweisungen: **DEFINT**, **DEFSNG**, **DEFSTR**, **DEF FN**, **COMMON**, **CHAIN**

ANWEISUNG: **DEF FN**

FUNKTION: Definition einer numerischen oder einer Stringfunktion innerhalb einer BASIC-Zeile durch den Anwender

FORMAT:

```
DEF { FNVar.Name     (Parameterliste) = num. Ausdruck    }
    { FNStr.Var.Name (Parameterliste) = Stringausdruck   }
```

Var.Name: beliebiger numerischer Variablenname mit Ausnahme eines Arrays

Str.Var.Name: beliebiger Stringvariablenname mit Ausnahme eines Arrays

Parameterliste: Folge von numerischen und/oder String-Variablen, die durch Komma getrennt werden. Es sind keine Array-Elemente zugelassen.

WIRKUNG: Die in einer **DEF**-Anweisung innerhalb der Klammern stehenden Parameter sind lokale Variablen, die keinen Bezug zu gleichnamigen Variablen ausserhalb der Funktion haben und deren Inhalt im Gesamtprogramm nicht beeinflussen. Beim Funktionsaufruf (s. **FN**) sind diesen Parametern entsprechende Ausdrücke zuzuweisen, wodurch den lokalen Variablen der aktuelle Wert der Ausdrücke zugewiesen wird.

Auf der rechten Seite vom Gleichheitszeichen können ausser den Elementen der Parameterliste auch andere Variablen, sogenannte globale Variablen, benutzt werden.

Die Funktion **FN**Var.Name oder **FN**Str.Var.Name wird durch den Ausdruck auf der rechten Seite definiert. Die Funktion wird in einem Programm so oft ausgeführt, wie ihr Name **FN**Var.Name oder **FN**Str.Var.Name zusammen mit den aktuellen Parametern im Programm aufgerufen wird (vgl. **FN**).

BEMERKUNGEN:

- Eine Funktion darf nur ein einziges Mal, und zwar vor dem ersten Aufruf, definiert werden und in dieser Form nur eine Zeile und eine Anweisung umfassen.
- Verwendet man bei der Definition einer Funktion einen Stringvariablennamen, ist das Ergebnis der Funktion ein String; benutzt man den Namen einer num.Variablen, so ist das Ergebnis ein numerischer Wert.

- Für den Variablennamen/Stringvariablennamen sind Typendeklarationen, z.B. mit **DEFSTR**, möglich.
- Wird eine Funktion vor ihrer Definition aufgerufen, so erfolgt die Meldung "Undefined user function".
- Es dürfen in der Parameterliste nicht mehr als zwölf Variablennamen (lokale Variablen) verwendet werden. Es muß aber mindestens ein Variablenname darin vorkommen.
- Direkte rekursive Aufruffolgen (z.B. 10 **DEF FNA**(Y)=**FNA**(K)+Y) sind nicht erlaubt.
- Aufrufe anderer definierter Funktionen rechts vom Gleichheitszeichen sind möglich (z.B. 30 **DEF FNS$**(X,R$)= **FNA$**(X)+" "+R$).
- Die beim Aufruf der Funktion aktuell eingereichten Parameter müssen in Typ und Anzahl mit den in der 'Parameterliste' aufgeführten lokalen Variablen übereinstimmen.
- Funktionsdefinitionen dürfen nicht im Direkt-Mode eingegeben werden.
- Syntaxfehler in **DEF FN**-Anweisungen werden beim Aufruf der Funktion durch **FN** gemeldet.
- ACHTUNG: Durch **DEFINT**, **DEFSTR**, **DEFDBL**, **DEFSNG** getroffene Typenvereinbarungen beziehen sich auch auf den 'Var.Namen' und auch auf die in der 'Parameterliste' angegebenen lokalen Variablen!

BEISPIELE: numerische selbstdefinierte Funktion:

```
30 DEF FNU1(P,R)=2*P*R - P^2 + 3*R^3-1
```

selbstdefinierte Stringfunktion:

```
10 DEFSTR T
20 DEF FNTFUNCT(T,U1%,U2%)=CHR$(U1%)+T+CHR$(U2%)
```

VERWEISE: Kapitel 4.3.7
Anweisung: **FN**

ANWEISUNG: **DEFINT**
(define integer)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als numerische Integer-Variablen

FORMAT:

DEFINT Anf.b.st.1 [-Anf.b.st.2] [,Anf.b.st.k [-Anf.b.st.1]] ...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä,ö,ü, Ä,Ö,Ü und ß); geben den Anfangsbuchstaben der als 'Integer' vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st.1' (incl.) 'Anf.b.st.2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von **%** an den Namen - als numerische Integer-Variablen interpretiert.

BEMERKUNGEN:

- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2' ebenso 'Anf.b.st.2' näher an 'Anf.b.st.1' als 'Anf.b.st.k' usw.
- Eine neue Typvereinbarung durch **DEFDBL**, **DEFSNG** oder **DEFSTR** hebt eine vorher erfolgte auf. Dabei erhalten ohne Fehlermeldung alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen wieder den Default-Wert (Ø für numerische, "" (=Leerstring) für Strings)!
- Beim Verketten von Programmen mit Variablenübergabe (vgl. Anweisungen **CHAIN** und **COMMON** bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen.
- **DEFINT** bezieht sich bei einer vom Anwender definierten Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.

- Durch Anhängen von !,# oder $ an Variablennamen, die wegen **DEFINT** normalerweise Integer-Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.
  ACHTUNG: Ist z.B. **DEFINT** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNR und KUNDENNR% nicht mehr möglich. Eine Wertänderung von KUNDENNR% bewirkt dann automatisch eine Wertänderung von KUNDENNR .

BEISPIELE:

```
10 DEFINT I-N,Z
20 A#=123.456789012346#
30 I=A#:LOSNUMMER=A#:N1=A#:ZEILE=A#
40 LPRINT I,LOSNUMMER,N1,ZEILE
 123            123            123            123
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2
Anweisungen: **DEFDBL, DEFSNG, DEFSTR, DEF FN, COMMON, CHAIN**

ANWEISUNG: **DEFSNG**
(define single precision)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als einfach genaue numerische Variablen

FORMAT:

DEFSNG Anf.b.st.1 [-Anf.b.st.2] [,Anf.b.st.k [-Anf.b.st.l]] ...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä,ö,ü, Ä, Ö, Ü und ß); geben den Anfangsbuchstaben der als einfach genau vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st. 1' (incl.) und 'Anf.b.st. 2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von ! an den Namen - als einfach genaue numerische Variablen interpretiert.

BEMERKUNGEN:

- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2', ebenso Anf.b.st.2' näher an 'Anf.b.st.1' als 'Anf.b.st.k usw.
- Eine neue Typvereinbarung durch **DEFDBL**, **DEFINT** oder **DEFSTR** hebt eine vorher erfolgte auf. Dabei erhalten ohne Fehlermeldung alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen wieder den Default-Wert (Ø für numerische, "" (=Leerstring) für Strings)!
- Beim Verketten von Programmen mit Variablenübergabe (vgl. Anweisungen **CHAIN** und **COMMON**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen.
- **DEFSNG** bezieht sich bei einer vom Anwender definierten Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.
- Durch Anhängen von **%**,**#** oder **$** an Variablennamen, die wegen **DEFSNG** normalerweise einfach genaue Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.

ACHTUNG: Ist z.B. **DEFSNG** K erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNR und KUNDENNR! nicht mehr möglich. Eine Wertänderung von KUNDENNR!bewirkt dann automatisch eine Wertänderung von KUNDENNR .

BEISPIELE:

```
10 DEFSNG A-H
20 A#=123.456789012346#
30 ALPHA=A#:BETA=A#:HOEHE=A#
40 LPRINT ALPHA,BETA,HOEHE
 123.457         123.457         123.457
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2
Anweisungen: **DEFINT, DEFDBL, DEFSTR, DEF FN, COMMON, CHAIN**

ANWEISUNG: **DEFSTR**
(define string)

ZWECK: Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als String-Variablen

FORMAT:

DEFSTR Anf.b.st.1 [-Anf.b.st.2] [,Anf.b.st.k [-Anf.b.st.l]] ...

Anf.b.st.: Klein- oder Großbuchstabe (außer ä,ö,ü, Ä,Ö,Ü und ß); geben den Anfangsbuchstaben der als String vereinbarten Variablen an (vgl. Kapitel 2.5.3.1)

WIRKUNG: Alle Variablen, deren Namen mit den Buchstaben beginnt, die zwischen 'Anf.b.st.1' (incl.) und 'Anf.b.st.2' (incl.) liegen, werden - auch ohne Typvereinbarung durch Anhängen von **$** an den Namen - als String-Variablen interpretiert.

BEMERKUNGEN:

- 'Anf.b.st.1' muß im Alphabet näher an A liegen als 'Anf.b.st.2', ebenso 'Anf.b.st.2' näher an 'Anf.b.st.1' als 'Anf.b.st.k' usw.
- Eine neue Typvereinbarung durch **DEFDBL**, **DEFSNG** oder **DEFINT** hebt eine vorher erfolgte auf. Dabei erhalten ohne Fehlermeldung alle vorher unter diesem Anfangsbuchstaben mit Werten belegten Variablen wieder den Default-Wert (Ø für numerische)!
- Beim Verketten von Programmen mit Variablenübergabe (vgl. Anweisungen **CHAIN** und **COMMON**) bleiben die Typvereinbarungen erhalten, sofern sie nicht neu erfolgen.
- **DEFSTR** bezieht sich bei einer Funktion (siehe Anweisung **DEF FN**) auch auf den Funktionsnamen sowie die in der 'Parameterliste' aufgeführten lokalen Variablen.

- Durch Anhängen von **%**, **!** oder **#** an Variablennamen, die wegen **DEFSTR** normalerweise String-Variablen wären, kann eine für den betreffenden Anfangsbuchstaben abweichende Typvereinbarung erreicht werden.
  ACHTUNG: Ist z.B. **DEFSTR K** erfolgt, ist eine Unterscheidung z.B. zwischen den Variablen KUNDENNAME und KUNDENNAME**$** nicht mehr möglich. Eine Wertänderung von KUNDENNAME**$** bewirkt dann automatisch eine Wertänderung von KUNDENNAME .

BEISPIELE:

```
10 DEFSTR T,E,Z
20 T="Text":E="Erklärung":ZEILE=T+"-"+E
30 LPRINT T,E
40 LPRINT ZEILE
Text            Erklärung
Text-Erklärung
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.2
Anweisungen: **DEFINT**, **DEFSNG**, **DEFDBL**, **DEF FN**, **COMMON**, **CHAIN**

BEFEHL: **DELETE**

FUNKTION: Löschung einer oder mehrerer Programmzeilen im Arbeitsspeicher

FORMAT:

DELETE { Zeilennummer$_1$ [-Zeilennummer$_2$] | . | -Zeilennummer$_2$ }

Zeilennummer$_1$: ganze positive Zahl, die die Nummer der zu löschenden Zeile oder die erste Zeile des zu löschenden Abschnittes angibt

Zeilennummer$_2$: ganze positive Zahl, die die letzte Zeile des zu löschenden Abschnitts angibt

WIRKUNG: Die durch 'Zeilennummer$_1$' angegebene Programmzeile oder alle Programmzeilen des durch 'Zeilennummer$_1$' und 'Zeilennummer$_2$' (inklusive) begrenzten Abschnittes werden im Arbeitsspeicher gelöscht. Ist der erste Operand der Punkt (.), so wird für 'Zeilennummer$_1$' die zuletzt angesprochene oder abgearbeitete Programmzeile genommen. Bei Angabe nur des Operanden '-Zeilennummer$_2$' werden alle Programmzeilen vom Anfang des Programmes bis zu 'Zeilennummer$_2$' gelöscht.

BEMERKUNGEN:
- Die im Befehl angegebenen Zeilennummern müssen im Programm vorhanden sein (sonst: "Illegal function call").
- Die Eingabe von 'Zeilennummer' und unmittelbares Drücken einer Eingabeabschlußtaste hat dieselbe Wirkung wie **DELETE** Zeilennummer.
- Nach Ausführung des Befehls **DELETE** befindet sich das System im Direkt-Mode.

BEISPIELE:

```
delete .
delete 2255-3450
delete -150
```

VERWEISE: Kapitel 3.3

ANWEISUNG: **DIM**

FUNKTION: Festlegung der Dimension und der maximalen Werte der Indices von Arrays

FORMAT:

```
DIM Arrayname(Indexliste) [,Arrayname(Indexliste)] ...
```

Format der Indexliste:
num.Ausdr.1 [,num.Ausdr.2] ... [,num.Ausdr.k] ... [,num.Ausdr.n]

Arrayname: Name des Arrays

num.Ausdr.k: Gibt den höchsten Wert des Index in der Dimension k an

n: Dimension des Arrays

WIRKUNG: Der num.Ausdr.k wird berechnet und gerundet. (-Ø.5 liefert Ø; -0.6 liefert -1). Für jeden angegebenen Array wird die Dimension und der maximale Wert der Indices definiert.

BEMERKUNGEN:

- Für nicht mit **DIM** definierte Arrays gilt: Der Default-Wert der höchsten Indices beträgt 1Ø.
- In Abhängigkeit der Definition von **OPTION BASE** ist der kleinste mögliche Index-Wert eines Arrays gleich Ø oder 1.
- Der vom Interpreter zugelassene Höchstwert für die Dimension ist 255, und die maximale Anzahl Elemente pro Dimension beträgt 32767. Die tatsächlichen Obergrenzen ergeben sich aus der Größe des jeweils vorhandenen Arbeitsspeicher.
- Die **DIM**-Anweisung muß vor dem ersten Auftreten des entsprechenden Arrays stehen.

- Werden den Elementen der Arrays keine Werte zugewiesen, so haben die Elemente der numerischen Arrays den Wert Ø und die Elemente der Stringarrays den Wert Leerstring (="").
- Verschieden dimensionierte Arrays dürfen nicht den gleichen Namen haben. (z.B. ist **DIM** A(2), A(3,3) nicht zulässig).
- Zum Löschen von Arrays (er ist dann nicht mehr dimensioniert) siehe **ERASE**.
- Da für die Indices numerische Ausdrücke zugelassen sind, ist eine dynamische Dimensionierung von Arrays möglich. Vor einer Redimensionierung muß die Anweisung **ERASE** verwendet werden. Ansonsten erfolgt die Fehlermeldung "Duplicate definition".

BEISPIELE: siehe Anweisung **OPTION BASE**

```
10 DIM P(2,5,9),Q11(2*T,5),TEXT1$(2,3,5)
```

```
10 DIM P(2,5,9),Q11(2*T,5),TEXT1$(2,3,5)
```

VERWEISE: Kapitel 2.5.3.3 und 4.3.2
Anweisungen : **OPTION BASE, ERASE**
Fehler-Codes: 7, 9, 10

ANWEISUNG: **DRAW**

FUNKTION: Festlegung von Zeichenvorgängen

FORMAT:

```
DRAW [% Window-Nr.] Befehlsstring
```

Window-Nr. : numerischer Ausdruck; 1 bis 16

Befehlsstring: Stringkonstante oder Stringvariable, die eine Befehlsfolge enthält

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, so wird das angesprochene Window gewählt. Fehlt dieser Parameter, so bezieht sich die Anweisung auf das aktive Window. Der Befehlsstring wird der vom System benutzten Variablen **NAME$** zugewiesen und beschreibt, wie gezeichnet werden soll. Der Befehlsstring für diesen String besteht aus einem oder zwei Buchstaben und Koordinatenwerten. Die Form der Darstellung kann mit der Vorstellung einer "Bewegungen" durchführenden "Feder" verbunden werden.
Koordinaten Angaben werden hier mit 'x' und 'y' bezeichnet, wenn es Absolutkoordinaten sind und mit 'dx' und 'dy' bezeichnet, falls Relativkoordinaten angegeben werden.

Folgende Elemente für den Befehlsstring sind zulässig:

B	Dieses Element kann anderen Elementen vorangestellt werden und bedeutet, daß nur Bewegungen der "Feder" ohne Zeichnen ausgeführt werden. Ist **B** nicht angegeben, so wird eine Zeichenbewegung ausgeführt.
C Farbindex	Bewirkt die Vorwahl der durch 'Farbindex' angegebenen Farbe, bezogen auf die für das Window gültige **COLOR**-Anweisung
J x,y	Es wird die Feder linear zum Punkt mit den Koordinaten (x,y) bewegt

M dx,dy	Bewirkt eine lineare Federbewegung um dx und dy Einheiten
L dx	Bewirkt eine Federbewegung um dx nach links
R dx	Bewirkt eine Federbewegung um dx nach rechts
U dy	Bewirkt eine Federbewegung um dy nach oben
D dy	Bewirkt eine Federbewegung um dy nach unten

Jedem dieser Befehlselemente außer **C** kann ein zusätzlicher Operand zugeordnet werden, der eine logische Operation für jeden Elementarpunkt bestimmt. Dieser Operand entspricht in seiner Wirkung denen der Anweisungen **LINE**, **CIRCLE** und **PUT %.**

Diese Operanden sind:

P	für	**PSET**
X	für	**XOR**
O	für	**OR**
R	für	**PRESET**
N	für	**NOT**
A	für	**AND**

Sie müssen ohne Trennzeichen unmittelbar nach dem Befehlselement **J**, **M**, **L**, **R**, **U** oder **D** folgen:

ANMERKUNGEN:

- Alle Angaben von Koordinatenwerten (absolut und relativ) beziehen sich auf die gültige **SCALE**-Anweisung.
- Die Buchstaben für die Befehle müssen als Großbchstaben im Befehlsstring enthalten sein.
  Die Koordinatenwerte sind als num. Konstanten oder Variablen im String enthalten und sind bei zwei Angaben durch Komma zu trennen. Wird ein Koordinatenwert durch eine Variable angegeben, so muß die Variable durch Voran- und Nachstellen des Zeichens '=' gekennzeichnet werden. Die Verwendung von Blanks zwischen Befehlscodes und Koordinaten ist beliebig.

- Die Anweisung **DRAW** verwendet die reservierte Variable **NAME$**. Die Zuweisung an **NAME$** kann somit vom Anwender nicht verwendet werden und führt zur Fehlermeldung "Syntax error".
- In einem Befehlsstring können hintereinander mehrere Befehlselemente mit ihren Operanden definiert sein.
- Der Befehlsstring muß den Vorschriften entsprechen. Ist der String fehlerhaft, so erfolgt die Fehlermeldung "Syntax error". Diese kann in Fehlerbehandlungsroutinen nicht sinnvoll bearbeitet werden.

BEISPIELE:

```
10 CLEAR : N%=WINDOW(2,32)
20 WINDOW %N% : FOR J%=1 TO 2
26 SCALE %J% 0,100,0,100 :CLS %J% : NEXT
30 DRAW "BJ 0,0" : DRAW "J 100,100" : DRAW "D 50"
40 X1=20 : Y1=40 : RELX=-5 : RELY=50
50 BEFEHLSSTRING$="BJ =X1=,=Y1= M =RELX=,=RELY="
60 DRAW %1 BEFEHLSSTRING$
999 K$=INKEY$ : IF K$="" THEN 999 ELSE END
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **SCALE**, **COLOR=**, **COLOR**
Funktion : **WINDOW**

BEFEHL: **EDIT**

FUNKTION: Ändern bestehender Programmzeilen eines Programms im Arbeitsspeicher

FORMAT:

EDIT { Zeilennummer / . }

Zeilennummer: ganze positive Zahl, die die Nummer der zu ändernden Programmzeile angibt

WIRKUNG: Der Command-Mode wird verlassen und der Edit-Mode gewählt. Die Zeilennummer der durch Zeilennummer angegebenen Programmzeile wird angezeigt und kann mit Hilfe von Edit-Subbefehlen verändert werden.
Wird anstelle von 'Zeilennummer' der Parameter ' . ' verwendet, so bezieht sich **EDIT** auf die laufende Zeile.

Edit-Subbefehle

Edit-Subbefehle werden verwendet, um den Cursor innerhalb der Zeile zu bewegen oder um Zeichen ändern, löschen, einfügen oder anfügen zu können.

Edit-Subbefehle werden am Bildschirm nicht angezeigt. Werden Zeichen, die keinen Subbefehl bilden, eingegeben, so ertönt das akustische Signal.

Anmerkung: In der nachfolgenden Beschreibung bedeuten:

Zeichen — ein beliebiges Zeichen

Text
a) eine beliebige Zeichenfolge mit beliebiger Länge
b) eine beliebige Zeichenfolge mit genau n Zeichen, falls dem Subbefehl der Parameter 'n' vorangestellt ist.

n — Ganze positive Zahl, die einigen Subbefehlen vorangestellt wird und einen Wiederholungsfaktor für diesen Subbefehl darstellt. Enfällt 'n' im Subbefehl, so wird 1 angenommen.

Subbefehle zur Cursorsteuerung

Blank

bewegt den Cursor um eine Stelle nach rechts. Das soeben durchlaufene Zeichen wird angezeigt. Wird das Zeilenende erreicht, so haben nachfolgende Blanks keine Wirkung.

nBlank

bewirkt die n-fache Wiederholung des Subbefehles Blank.

H

Bewegt den Cursor um eine Stelle nach links (zurück). Die Zeichen, um die der Cursor nach links verschoben werden, werden nicht mehr angezeigt. Die durchlaufenen Zeichen bleiben erhalten. Werden mehr H angegeben, als der Cursorposition innerhalb der Zeile entspricht, so bleibt die Cursorposition vor dem ersten Zeichen der Zeile konstant. Es bleibt jedoch die Zeilennummer und ein nachfolgendes Blank erhalten.

n H

Bewirkt die n-fache Wiederholung des Subbbefehles H. Es wird der Cursor um n-Stellen nach links oder an den Zeilenanfang gesetzt. Alle Zeichen, die dabei durchlaufen werden, werden in der Reihenfolge des Durchlaufens, also in umgekehrter Reihenfolge, angezeigt.

Subbefehle für das Einfügen von Zeichen

IText

Die als 'Text' eingegebene Zeichenfolge wird rechts von der aktuellen Cursorposition eingefügt. Das Einfügen kann auf folgende Arten beendet werden:

HOME 5

5 beendet das Einfügen und der Edit-Mode bleibt erhalten. Der Cursor steht an der ersten Stelle nach dem eingefügten Zeichen.
CR beendet das Einfügen und der Edit-Mode wird verlassen.
Während des Einfügens kann H wie bei Eingaben außerhalb des Edit-Modes zum Löschen von fehlerhaft eingegebenen Zeichen verwendet werden.
Wird durch das Einfügen versucht, in einer Zeile mehr als 255 Zeichen einzugeben, so ertönt das akustische Signal und weitere Zeichen werden nicht akzeptiert.

XText

Erlaubt das Anfügen von Zeichen an eine bestehende Zeile. Es wird der Cursor an die Stelle nach dem letzten Zeichen der Zeile gesetzt. Alle unter I beschriebenen Möglichkeiten stehen ohne weiteren Subbefehl zur Verfügung.

Subbefehle für das Löschen von Zeichen

D löscht das Zeichen rechts zum Cursor.

nD löscht n Zeichen beginnend ab der Cursorposition. Ist n größer als die Anzahl der Zeichen rechts vom Cursor, so wird die Zeile bis zum Ende gelöscht. Die gelöschten Zeichen werden zwischen dem Zeichen Ö angezeigt, bzw. \ bei ASCII-Zeichensatz. Der Cursor befindet sich vor dem ersten nicht gelöschten Zeichen.

H Es werden alle Zeichen rechts vom Cursor bis zum Zeilenende gelöscht und das Einfügen von Zeichen wie beim Subbefehl **X** ermöglicht. Der Subbefehl **H** ist besonders zum Ersetzen von Anweisungen oder Anweisungsteilen am Zeilenende sinnvoll.

Subbefehle für das Ersetzen von Zeichen

CZeichen
n**C**Text Das Zeichen rechts von der Cursorposition wird durch das eingegebene Zeichen ersetzt.
Es werden - beginnend ab dem Zeichen rechts von der Cursorposition - 'n' Zeichen durch die im 'Text' angegebene Zeichenfolge ersetzt.
Enthält 'Text' mehr als 'n' Zeichen, so werden die nachfolgenden Zeichen aus 'Text' ignoriert. Der Cursor befindet sich nach **C** an der Stelle vor dem ersten nicht geänderten Zeichen.

Subbefehle für das Suchen von Zeichen

SZeichen
n**S**Zeichen Es wird das 'n'-te Auftreten von 'Zeichen' in der Zeile gesucht und der Cursor an die Stelle vor diesem Zeichen gesetzt. Das Zeichen an der aktuellen Cursorposition wird in die Suche nicht einbezogen. Wird das 'Zeichen' nicht oder nicht genügend oft gefunden, so wird der Cursor an das Ende der Zeile gesetzt.
Die bei der Suche durchlaufenen Zeichen werden angezeigt.

KZeichen
n**K**Zeichen Der Subbefehl **K** hat eine ähnliche Wirkung wie **S**, es werden jedoch alle beim Suchen durchlaufenen Zeichen gelöscht. Der Cursor wird an die Stelle vor dem gefundenen 'Zeichen' gesetzt. Die gelöschten Zeichen werden zwischen Ö angezeigt, bzw. \ bei ASCII-Zeichensatz.

Subbefehle für die Steuerung

CR	Bewirkt die Anzeige der restlichen Zeile, speichert die durchgeführten Änderungen und beendet den Edit-Mode.
E	Die Änderungen werden gespeichert und der Edit-Mode verlassen. Im Gegensatz zu CR wird der Zeilenrest jedoch nicht angezeigt.
Q	Beendet den Edit-Mode ohne Speicherung der durchgeführten Änderungen. Die Zeile bleibt in ihrer ursprünglichen Form gespeichert.
L	Bewirkt die Ausgabe der restlichen Zeile. Die Zeilennummer wird erneut angezeigt und der Cursor befindet sich an der ersten Stelle der Zeile.
A	Erlaubt den Neubeginn des Änderns einer Zeile. Die ursprüngliche Zeile wird wieder bereitgestellt und der Cursor wieder an die erste Stelle der Zeile gesetzt.

Anmerkungen:

1.) Der Edit-Mode wird vom System automatisch gewählt, wenn während der Programmausführung ein Syntaxfehler erkannt wird.

2.) Der Edit-Mode löscht alle Variablenwerte.
Sollen daher (z.B. nach einem Syntaxfehler) noch Variablenwerte überprüft werden, so muß der Edit-Mode mit dem Subbefehl **Q** verlassen werden.
Jeder andere Subbefehl löscht die Variablenwerte.

3.) Wird versucht, **EDIT** für ein mit dem Parameter **P** geschütztes Programm auszuführen oder wird der Edit-Mode automatisch infolge eines Syntaxfehlers aufgesucht, so wird 'Illegal function call' gemeldet.

BEISPIELE:

```
50 print "zu koorrigieren"";?
edit 50
50 PRINT "zu koorrigieren"";?
50 PRINT "zu korrigieren":?
list 50
50 PRINT "zu korrigieren":PRINT
ok
```

VERWEISE: Kapitel 3.4

ANWEISUNG: **END**

FUNKTION: Gibt das logische Ende eines Programmes an

FORMAT:

```
END
```

WIRKUNG: Die Programmausführung wird beendet, und externe Files werden geschlossen. Der BASIC-Interpreter befindet sich anschließend in der Befehlsebene.

BEMERKUNGEN:

- Die Anweisung **END** kann an jeder Stelle in einem Programm stehen.
- Das Herbeiführen eines Programmendes ohne Ausführung der **END**-Anweisung gewährleistet nicht, daß die benutzten Files ordnungsgemäß geschlossen werden.
- Die Ausführung der **END**-Anweisung verursacht die Meldung 'ok'.
- Die Meldung 'ok' erscheint auch, wenn die letzte Anweisung eines BASIC-Programmes abgearbeitet ist, ohne daß eine **END**-Anweisung das physische Programmende anzeigt.
- Ein Programm kann keine oder mehrere **END**-Anweisungen haben.

BEISPIELE:

```
10 ' Ein Programm kann mehrere oder keine END-Anweisung haben
20 CLS : PRINT "ENDE" : END
```

```
10 REM Programmbeginn
9900 K$=INKEY$ : IF K$="" THEN 9900 ELSE IF K$<>CHR$(13)
THEN END ELSE  10
```

VERWEISE: Kapitel 4.3.5

FUNKTION: **EOF**
(end of file)

ZWECK: Abfrage auf Ende eines sequentiellen Files beim Input von sequentiellen Files

FORMAT:

```
EOF(Filenr.)
```

WIRKUNG: Die Funktion **EOF** liefert den Wahrheitswert -1, falls beim Versuch eines Inputs von einem sequentiellen File das Fileende erreicht wurde.

BEMERKUNGEN:
- Wird der Versuch eines Inputs über das Fileende hinaus gemacht, ohne mit **EOF** abzufragen, wird der Fehler 62 ("Input past end") gemeldet.

BEISPIELE:

```
10 OPEN "O",#1,"1:sequen"
20 MASKE$="###.#####  "
30 FOR I=1 TO 10
40 PRINT#1, USING MASKE$;RND
50 NEXT I
60 CLOSE #1
70 OPEN "I",#1,"1:sequen"
80 IF EOF(1) THEN 90 ELSE LINE INPUT#1,A$:LPRINT A$:GOTO 80
90 CLOSE
   .08225
   .25584
   .77617
   .68201
   .07324
   .37856
   .95686
   .37209
   .01480
   .59631
```

VERWEISE: Kapitel 4.3, 10.2
Anweisungen: **INPUT#**, **LINE INPUT#**
Fehler-Code: 62

ANWEISUNG: **ERASE**

FUNKTION: Aufheben der Dimensionierung eines Arrays

FORMAT:

ERASE Arrayname [,Arrayname] ...

Arrayname: Name des Arrays (ohne Klammern und Indices)

WIRKUNG: Für jeden angegebenen Array wird die Dimensionierung, die in der zuletzt durchlaufenen **DIM**-Anweisung für den/die Array/s gegeben wurde, aufgehoben.

BEMERKUNGEN:

- Nach der Aufhebung der Dimensionierung kann der Array neu dimensioniert werden; ansonsten gelten die Default-Werte (siehe **DIM**-Anweisung).
- Es kann mit **ERASE** auch die Dimensionierung von Arrays aufgehoben werden, die mit Default-Werten dimensioniert wurden, also ohne **DIM**-Anweisung.

BEISPIELE:

```
5 CLS:OPTION BASE 1
10 DIM Z$(100),A(20,20),B(5,10,20)
20 A(15,15)=4711
30 PRINT A(15,15),A(20,20),A(5,5)
50 REM   Redimensionierung
70 ERASE A
80 PRINT A(5,5)
90 PRINT A(15,15)
 4711          0                0

 0

Subscript out of range in 90
Ok
exec "sp"      nur PR2400
```

VERWEISE: Kapitel 2.5.3.3 und 4.3.2
Anweisungen **DIM**, **OPTION BASE**

RESERVIERTE VARIABLE: **ERL**

ZWECK: enthält die Zeilennr. derjenigen Programmzeile, in der der zuletzt vom System entdeckte Fehler auftrat bzw. die Zeilennr., in der die letzte **ERROR**-Anweisung auftrat

FORMAT:

```
ERL
```

EINSATZ: Mit Hilfe einer Anweisung der Art

IF ERL=Fehlerzeile **THEN** Zeilennr. / Anweisung(sfolge) ...

kann eine Behandlung von aufgetretenen ("echten" bzw. mit **ERROR** "künstlich" erzeugten) Fehlern in Abhängigkeit von der Programmzeile, in der der Fehler auftrat (='Fehlerzeile'), erreicht werden.
Ein Fehler führt jedoch immer zu sofortigem Programmabbruch und Fehlermeldung, wenn er nicht in einer Fehlerbehandlungsroutine bearbeitet wird. (Ausnahme: gewisse Fälle des "Overflows"; vgl. z.B. Kapitel 2.5.1.2 bzw. der Anweisungen **INPUT#** bzw. **LINE INPUT#** bei File-Ende).

BEMERKUNGEN:

- **ERL** enthält einen ganzzahligen Wert.
- Wird im Gegensatz zur obigen Formulierung die Abfrage in der Form

  IF Fehlerzeile=**ERL THEN** ...

  durchgeführt, wird bei einer Neu-Numerierung des Programms (vgl. Befehl **RENUM**) der Parameter 'Fehlerzeile' nicht angepaßt.
- Auf die reservierte Variable **ERL** kann mit **LET** kein Wert zugewiesen werden, sondern nur über Umwege mit der Anweisung **ERROR**.

- Tritt ein Fehler im Direkt-Mode auf, so enthält die reservierte Variable **ERL** stets den Wert 65535.

BEISPIELE:

```
1 ON ERROR GOTO 22800
10 OPTION BASE 1 : DIM A$(10,6)
30 I%(1)=2 : I%(2)=5
50 GOSUB 1000
60 I%(2)=0 : GOSUB 1000
70 END
1000 A$(I%(1),I%(2))=STRING$(I%(1),I%(2)) :LPRINT A$(I%(1
),I%(2)):RETURN
22800 LPRINT "Fehler ";ERR;" in Zeile ";ERL : RESUME NEXT
Ok
Fehler  9  in Zeile  1000
Fehler  9  in Zeile  1000
```

VERWEISE: Kapitel 4.3.12 und 7
Anweisungen: **ON ERROR GOTO, ERROR, RESUME**
Reservierte Variable: **ERR**

RESERVIERTE VARIABLE: **ERR**

ZWECK: enthält den Fehler-Code des zuletzt vom System entdeckten Fehlers bzw. nach einer **ERROR**-Anweisung den dadurch zugewiesenen Wert

FORMAT:

```
ERR
```

EINSATZ: Mit Hilfe einer Anweisung der Art

IF ERR=Fehler-Code **THEN** Zeilennr.
Anweisung(sfolge) ...

kann eine Behandlung von aufgetretenen ("echten" bzw. mit **ERROR** "künstlich erzeugten") Fehlern in Abhängigkeit vom Fehler-Code erreicht werden.

Ein Fehler führt jedoch immer zu sofortigem Programmabbruch und Fehlermeldung, wenn er nicht in einer Fehlerbehandlungsroutine bearbeitet wird. (Ausnahmen: gewisse Fälle des "Overflows"; vgl. z.B. Kapitel 2.5.1.2; bzw. der Anweisungen **INPUT#** bzw. **LINE INPUT#** bei File-Ende).

BEMERKUNGEN:

- **ERR** enthält einen ganzzahligen numerischen Wert.
- Auf die reservierte Variable **ERR** kann ein Wert nur mit der Anweisung **ERROR** zugewiesen werden.
- PCOS-Fehler werden nicht in **ERR** festgehalten.

- **ERR** hat den Wert **0**, nachdem das Programm mit **RESUME** aus einer Fehlerbehandlungsroutine zurückgekehrt ist.

BEISPIELE:

```
1 ON ERROR GOTO 22800
10 OPTION BASE 1 : DIM A$(10,6)
30 I%(1)=2 : I%(2)=5
50 GOSUB 1000
60 I%(2)=0 : GOSUB 1000
70 END
1000 A$(I%(1),I%(2))=STRING$(I%(1),I%(2)) :LPRINT A$(I%(1
),I%(2)):RETURN
22800 LPRINT "Fehler ";ERR;" in Zeile ";ERL : RESUME NEXT
Ok
Fehler  9  in Zeile  1000
Fehler  9  in Zeile  1000
```

VERWEISE: Kapitel 4.3.12 und 7
Anweisungen: **ON ERROR GOTO, ERROR**
Reservierte Variable: **ERL**

ANWEISUNG: **ERROR**

FUNKTION: Zuweisung eines Fehler-Codes auf die reservierte Variable **ERR** oder Simulation eines "Fehlers" durch den Anwender

FORMAT:

```
ERROR num.Ausdr.
```

WIRKUNG: Der numerische Ausdruck wird berechnet und zur Ganzzahligkeit kaufmännisch gerundet.
Entspricht der Wert einem standardmäßig für den Interpreter definierten Fehlercode, wird die entsprechende Fehlermeldung gegeben und das Programm angehalten bzw. bei vorhandener Fehlerbehandlungsroutine in diese verzweigt (vgl. **ON ERROR GOTO**).
Anderenfalls wird der Fehler "Unprintable error" gemeldet und das Programm angehalten.
Dies geschieht nicht, wenn in einer vorher aktivierten Fehlerbehandlungsroutine der betreffende Fehlercode, z.B. mit einer "künstlichen" Fehlermeldung, belegt ist.

BEMERKUNGEN:
- Das Ergebnis des 'numerischen Ausdruck' muß größer als Ø und kleiner als 255 sein.
- Es empfiehlt sich, "künstlichen" Fehlercodes Werte über 7Ø (höchster dem Interpreter bekannter Fehler-Code) zu geben.
- PCOS-Fehler können nicht simuliert werden.

BEISPIELE:

```
1 CLS
10 ERROR 3
RETURN without GOSUB in 10

Ok

call "sp"      nur PR2400
```

```
10 CLS:S=100:T=5
20 ERROR S+T
Unprintable error in 20

Ok

exec "sp"      nur PR2400
```

VERWEISE: Kapitel 4.3.11, 4.3.12 und 7
Anweisungen: **ON ERROR GOTO, RESUME**
Reservierte Variablen: **ERR, ERL**

ANWEISUNG: **EXEC**

FUNKTION: Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine

FORMAT:

EXEC { Stringausdruck
"Name d. Routine Liste d. Parameter " }

Stringausdruck: enthält Namen der Routine und Parameter in der Form, wie sie in der PCOS-Ebene einzugeben sind

Name d. Routine: bezeichnet den Namen der Assembler-Routine; muß explizit angegeben werden wie in der PCOS-Ebene

Liste d. Parameter: Folge von num. und/oder String-konstanten, die durch Komma getrennt sind

WIRKUNG: Das System prüft, ob die aufgerufene Assembler-Routine bereits im Arbeitsspeicher vorhanden ist. Falls nicht, wird diese Routine auf der Diskette gesucht und in den Arbeitsspeicher geladen. Anschließend wird diese Routine sofort abgearbeitet. Der momentane Zustand des Arbeitsspeichers bleibt erhalten.

BEMERKUNGEN:

- Im Gegensatz zu der Anweisung **CALL** können mit der Anweisung **EXEC** nur Konstanten an die Assembler-Routine übergeben werden.
- Mit der Anweisung **EXEC** können keine Werte von der Assembler-Routine an das BASIC-Programm übergeben werden.
- Zur Beschleunigung des Aufrufs von Assembler-Routinen kann der PCOS-Befehl **pl** verwendet werden. Danach bleiben die Routinen ständig im Arbeitsspeicher erhalten.
- Routinen, die mit dem PCOS-Befehl **pl** bereits geladen wurden oder beim Laden des Betriebssystems automatisch geladen wurden (siehe PCOS-Befehl **ps**), können nicht noch einmal durch den Befehl **pl** geladen werden.

- Im Arbeitsspeicher resident gehaltene Routinen verringern die Kapazität des Arbeitsspeichers.

BEISPIELE:

```
10 INPUT "Datum (MM/TT/JJ)";A$
20 INPUT "Zeit  (HH:MM:SS)";B$
30 C$="ss "+A$+","+B$
40 EXEC C$
50 LPRINT DATE$, TIME$
60 EXEC "sp"   ' nur PR2400
03/04/1982      16/26/35
Datum (MM/TT/JJ)? 03/04/82
Zeit  (HH:MM:SS)? 16/26/35
 Date = 03/04/1982  Time = 16/26/35  Disk Verify = 0 (Off)
 Extent size = 0  Display = 0 (64 x 16)
```

VERWEISE:

Kapitel: 4.3.15, ANHANG IV
Anweisung: **CALL**
Funktion: **VARPTR**

FUNKTION: **EXP**
(exponent)

ZWECK: Berechnung einer Potenz der Eulerschen Zahl e

FORMAT:

```
EXP(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Die Eulersche Zahl e wird mit diesem Wert potenziert.

BEMERKUNGEN:

- e selbst ist mit **EXP(1)** zu bestimmen (einfache Genauigkeit!).
- Ist das Argument größer als 88, wird "Overflow" gemeldet.
  Ist das Argument kleiner als -87, wird mit Ø weitergerechnet.

BEISPIELE:

```
10 LPRINT EXP(1);EXP(2.645)
 2.71828   14.0834
```

VERWEISE: Kapitel 2.4 und 4.13

ANWEISUNG: **FIELD**

FUNKTION: Vereinbarung der Feldlängen der Records eines bestimmten Random-Files, Festlegung der Variablennamen und der Position im Random-File-Puffer sowie Bereitstellung des Random-File-Puffers für folgende **GET**- oder **PUT**-Anweisungen

FORMAT:

FIELD [#] Filenr.,Feldlänge **AS** Feldvariable [,...]

(Die Folge ,Feldlänge **AS** Feldvariable kann wiederholt werden.)

Filenr.:	num.Ausdr.	(Ergebnis: 1-15)
Feldlänge:	num.Ausdr.	(Ergebnis: größer Ø, aber kleiner oder gleich der bei **OPEN** festgelegten Record-Länge)
Feldvariable:	Stringvariable	

WIRKUNG: Für das unter 'Filenr.' geöffnete Random-File wird im betreffenden Random-File-Puffer Platz für 'Feldvariablen', reserviert, und zwar jeweils für die in 'Feldlänge' vereinbarte Anzahl Zeichen (Bytes).

BEMERKUNGEN:

- In einem Random-File sind alle Daten in Form von Strings gespeichert.
- Die Daten können nur über Stringvariablen vom File in den Puffer bzw. vom Puffer in das File übertragen werden (**GET** bzw. **PUT**).
- Die 'Feldvariablen' des Puffers dürfen ihre Inhalte nur mit Hilfe der Anweisungen **LSET** bzw. **RSET** oder durch Lesen mit Hilfe von **GET** erhalten.
- Die **FIELD**-Anweisung überträgt keine Variableninhalte in den Puffer, sondern stellt nur Platz im Puffer bereit.
- Für eine 'Filenr.' können beliebig viele **FIELD**-Anweisungen gegeben werden.
- Bei mehreren **FIELD**-Anweisungen für dieselbe 'Filenr.' gilt immer nur die zuletzt abgearbeitete.
- Überschreitet 'Feldlänge' bzw. die Summe der Feldlängen aller für eine bestimmte 'Filenr.' vereinbarten 'Feldvariablen' die bei **OPEN** festgelegte Record-Länge, wird der Fehler "FIELD overflow" gegeben.

BEISPIELE:

```
10 OPEN "R",#1,"1:random",128          'Oeffnen des Files "ra
ndom" auf Station 1:
20 OPEN "R",#2,"1:random1",24          'Oeffnen des Files "ra
ndom1" auf Station 1:

30
40 GOTO 5000
70 REM Recorddefinition fuer File#1
75 REM  Record-Typ 1
80 FIELD #1,2 AS I$(1),16 AS K$(1),4 AS H$(1),10 AS E$(1),32
 AS DUMMY1$,2 AS I$(2),16 AS K$(2),4 AS H$(2),10 AS E$(2),32
 AS DUMMY2$
85 RETURN
90 REM  Record-Typ 2
95 FIELD #1,2 AS TYP2.1$,4 AS TYP2.2$,8 AS TYP2.3$,114 AS DU
MMY1$
100 RETURN
130 '
140 REM Recorddefinition fuer File#2
150 FIELD #2,24 AS TEXT$
155 RETURN
160 '
5000 REM Hauptverarbeitung
5010 GOSUB 75    ' Record-Typ 1 von "random"
5020 REM Verarbeitung
5030 GOSUB 90    ' Record-Typ 2 von "random"
5040 REM Verarbeitung
5050 GOSUB 140   ' File#2
5060 REM Verarbeitung
6998 CLOSE #2,1
9999 END
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **OPEN, RSET, LSET, GET, PUT**
Funktionen: **MKI$, MKS$, MKD$**
Fehler-Code: 50

BEFEHL: **FILES**

FUNKTION: Ausgabe des kurzen Inhaltsverzeichnisses einer Diskette oder der Namen von Files mit gemeinsamen Eigenschaften auf dem Bildschirm

FORMAT:

```
FILES { Diskettenspezifikation }
      { Filename(nauswahl)      }
```

Diskettenspezifikation: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Diskettenspezifikationen entspricht (vgl. Kapitel 4.1.3)

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3); mit Hilfe der Parameter * und ? kann eine Auswahl bestimmter Files mit gemeinsamen Eigenschaften im Namen getroffen werden.

WIRKUNG: Es wird das kurze Inhaltsverzeichnis der Diskette ausgegeben. Ist kein Operand angegeben, wird das Inhaltsverzeichnis der Diskette angezeigt, die zuletzt durch einen Befehl oder eine Anweisung angesprochen wurde. Bei Angabe des Operanden 'Diskettenspezifikation' wird das Inhaltsverzeichnis der durch 'Diskettenspezifikation' bestimmten Diskette ausgegeben. Ist der Operand 'Filename' angegeben, so wird nur der Name des angesprochenen Files angezeigt. Ist eine 'Filenamenauswahl' angegeben, werden die Namen der dadurch selektierten Files ausgegeben. Im Falle, daß nur die Filespezifikation angegeben ist, muß die Station, auf der sich das File befindet, aktiv sein.
Der Befehl **FILES** gibt folgende Informationen:

1. Station (**0** oder **1**), in welcher sich die Diskette befindet
2. Diskettenbezeichnung
3. Anzahl freier Sektoren auf der Diskette.

Anschließend werden die File-Identifier - in 4 Spalten aufgeteilt - in der Reihenfolge ausgegeben,wie sich die Files im Inhaltsverzeichnis der Diskette befinden.

BEMERKUNGEN:

- Der PCOS-Befehl **vq** entspricht dem BASIC-Befehl **FILES.**
- Soll das kurze Inhaltsverzeichnis am Drucker ausgegeben werden, ist die Anweisung

  CALL "vq +dprt: Diskettenspezifikation"

  einzugeben.
  Das ausführliche Inhaltsverzeichnis kann durch

  CALL "vl [**+dprt:**] Disk-Spez. [Filenamen(auswahl)]"

  abgerufen werden.

BEISPIELE:

```
files "1."
files "DISKNAME test.prg"
files "0:*.prg"
files "1:??A*"
call "vq +dprt: 1:"
call "vl +dprt: 0:"
```

VERWEISE:

Kapitel 4.1.3
PCOS-Befehle: **vq, vl**

FUNKTION: **FIX**

ZWECK: Ermittlung des ganzzahligen Teils eines numerischen Werts

FORMAT:

```
FIX(num. Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet.
Vom Ergebnis wird der Nachkomma-Anteil abgeschnitten (ohne Rundung).

BEMERKUNGEN: - Das Ergebnis ist in der Genauigkeit des Arguments.

BEISPIELE: **FIX**(3.7) liefert 3 und **FIX**(-3.7) liefert -3
FIX(3.2) liefert 3 und **FIX**(-3.2) liefert -3

```
10 A=2.8572
20 LPRINT A;FIX(A)
30 END
 2.8572  2
```

VERWEISE: Kapitel 2.4 und 4.13
Funktionen: **INT, CINT**

FUNKTION: **FN**

ZWECK: Eine vom Anwender definierte Funktion wird aufgerufen und abgearbeitet

FORMAT:

```
{ FNVar. Name    }
{ FNStr.Var.Name } (Liste von Ausdrücken)
```

Var.Name, Str.Var.Name: siehe Anweisung **DEF FN**

Liste von Ausdrücken: Folge von num. oder Stringausdrücken, die durch Komma getrennt sind.

WIRKUNG: Die einzelnen Ausdrücke werden berechnet und entsprechend der Parameterliste (siehe Anweisung **DEF FN**) der Funktion übergeben. Anschließend wird die Funktion berechnet. Der Typ des Ergebnisses (num.Wert, String) hängt ab vom Typ der Funktion.

BEMERKUNGEN:
- Die Elemente von 'Liste von Ausdrücken' müssen in Typ (!!) und Anzahl mit den Elementen von 'Parameterliste' der zugehörigen **DEF FN**-Anweisung übereinstimmen.
- siehe Bemerkungen bei der Anweisung **DEF FN**

BEISPIELE:

```
30 DEF FNU1(P,R)=2*P*R - P^2 + 3*R^3-1
40 A=3.5742:B=(2*A)^.3
50 X=FNU1(A,B):LPRINT X
60 END
 16.7377
```

VERWEISE: Kapitel 4.3.7
Definition einer Funktion durch den Anwender siehe Anweisung **DEF FN**
Fehler-Code: 18

ANWEISUNG: **FOR**

FUNKTION: Startanweisung für die mehrmalige Ausführung von Anweisungen innerhalb einer Schleife

FORMAT:

```
FOR Laufvar.1=Anfangswert TO Endwert [STEP Schrittweite]
  .
BASIC-Anweisung(en)
  .
NEXT [Laufvar.1] [,Laufvar.2]    ...
```

Laufvar.:	numerische Variable; darf kein Array-Element und nicht doppelt genau sein
Anfangswert:	numerischer Ausdruck
Endwert:	" "
Schrittweite:	" "

WIRKUNG: Die zwischen **FOR** und **NEXT** liegende Anweisung(sfolge) wird so oft ausgeführt, wie es sich aus den Parametern 'Anfangswert', 'Endwert' und 'Schrittweite' ergibt.

1. Stößt das Programm auf eine **FOR**-Anweisung, wird zu dieser eine zugehörige **NEXT**-Anweisung ermittelt.

2. Der 'Laufvariablen' wird der 'Anfangswert' zugewiesen. Es wird der aktuelle Wert des 'Endwerts' und der 'Schrittweite' ermittelt und im Arbeitsspeicher festgehalten. Diese ermittelten Werte stellen von nun an feste, der betreffenden **FOR...NEXT** Schleife zugeordnete Parameter dar.

3. Vor Ausführung der folgenden BASIC-Anweisungen wird überprüft, ob der 'Endwert' überschritten ist (falls die 'Schrittweite' größer Ø ist) bzw., ob der 'Endwert' unterschritten ist (falls die 'Schrittweite' kleiner Ø ist).

4. Ist 'Endwert' über- bzw. unterschritten, wird mit derjenigen Anweisung fortgefahren, die auf die vorher ermittelte, der **FOR**-Anweisung zugeordnete **NEXT**-Anweisung folgt.

5. Andernfalls werden die auf **FOR**..... folgenden Anweisungen ausgeführt.

6. Stößt das Programm dann auf die **NEXT**-Anweisung, wird zum aktuellen Wert der Laufvariablen der Wert der 'Schrittweite' addiert. (Im Falle von negativer 'Schrittweite' vermindert sich der Wert von 'Laufvariable' entsprechend). Dieser Wert wird an die 'Laufvariable' zugewiesen und das Programm verzweigt wieder zu Punkt 3.

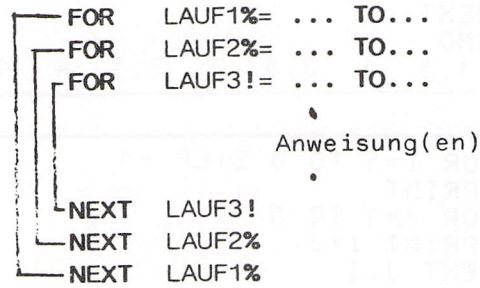
BEMERKUNGEN:

- Ist die Schrittweite nicht angegeben, wird der Default-Wert 1 angesetzt.
- Ist die 'Schrittweite' Ø, wird die Schleife nie verlassen, falls der Endwert anfangs nicht sofort über- bzw. unterschritten ist.
- Zur Beschleunigung der Schleife empfiehlt es sich (wenn aufgrund der 'Schrittweite' möglich), die 'Laufvariable' als Integer zu deklarieren (z.B. durch Anhängen von **%** an den Variablennamen).
- Ist die 'Laufvariable' ein Array-Element, erscheint die Fehlermeldung "Syntax error". Ist die 'Laufvariable' doppelt genau definiert, wird "Type mismatch" gemeldet.
- Der Name der 'Laufvariablen' bei der **NEXT**-Anweisung kann weggelassen werden. Die **NEXT**-Anweisung bezieht sich dann auf diejenige davorstehende **FOR**-Anweisung, die noch nicht aufgrund von 'Endwert'-Unter/Überschreitung abgearbeitet ist.
- Jeder **FOR**-Anweisung muß genau eine **NEXT**-Anweisung zugeordnet sein, sonst Fehlermeldung "FOR without NEXT".
- Der Wert von 'Laufvariable' wird während der Abarbeitung verändert.

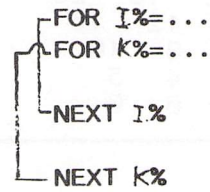
Verschachtelung von **FOR**...**NEXT**-Schleifen:

- Fallen mehrere **NEXT**-Anweisungen an einer Programmstelle zusammen, kann dies durch die Angabe aller betroffenen 'Laufvariablen' hinter einem **NEXT**, getrennt durch Komma, erklärt werden.
  Diese Formulierung kann jedoch bei bestimmten Datenkonstellationen zu einer fehlerhaften Anzahl von Schleifendurchläufen führen!

Die 'Laufvariablen' müssen in der Reihenfolge genannt werden, daß die Zuordnung nach folgendem Schema möglich ist:



Eine Verschachtelung nach dem Schema



oder ähnlichen ist nicht zulässig (Fehlermeldung: "NEXT without FOR").

- In verschachtelten **FOR...NEXT**-Schleifen müssen sich die Namen der 'Laufvariablen' unterscheiden.
- Achtung: Bei vorzeitigem Aussprung aus **FOR-NEXT**-Schleifen (z.B. mit **GOTO**) ohne späteren Rücksprung bleibt ein Stack-Speicher belegt. Dies kann bei wiederholtem Auftreten in einem Programm zu Speicher-Overflows führen. Zum Löschen von Stack-Speichern in solchen Fällen siehe Beispiel in Kapitel 4.3.6.

BEISPIELE:

```
10 FOR I!=1 TO 5 STEP .5
20 LPRINT I;
30 NEXT
40 END
 1   1.5   2   2.5   3   3.5   4   4.5   5
```

```
10 FOR I=5 TO 0 SIEP -1
15 LPRINT
20 FOR J=1 TO 5
30 LPRINT I*J;
40 NEXT J,I
50 END

 5   10   15   20   25
 4   8   12   16   20
 3   6   9   12   15
 2   4   6   8   10
 1   2   3   4   5
 0   0   0   0   0
```

VERWEISE:

Kapitel 4.3.5 und 4.3.6
Anweisung: **NEXT, CLEAR**
Fehler-Codes: 1,7,2,6

FUNKTION: **FRE**

ZWECK: Ermittlung des im Arbeitsspeicher noch freien Platzes (in Bytes) und ggf. Bereinigung des Arbeitsspeichers

FORMAT:

FRE({ Ø / Stringausdr. })

WIRKUNG: Bei Angabe der Ø wird nur der freie Platz (in Bytes) im Arbeitsspeicher ermittelt.
Bei Angabe von 'Stringausdr.' wird zusätzlich der Arbeitsspeicher bereinigt (von Überresten gereinigt und segmentweise neu geordnet).

BEMERKUNGEN:

- Eine Bereinigung des Arbeitsspeichers kann bis zu 2 Minuten dauern.
- Durch eine Bereinigung des Arbeitsspeichers kann einem Fehler 14 ("Out of string space") eine Zeitlang vorgebeugt werden.
- Nach einer Bereinigung des Arbeitsspeichers wird eine neue Bereinigung weniger Zeit benötigen und die Rechengeschwindigkeit möglicherweise größer sein.
- Eine Bereinigung des Arbeitsspeichers kann nur durch **FRE**(Str.Ausdr.) erfolgen.

BEISPIELE:

```
10 LPRINT FRE(0)
20 LPRINT FRE("a")
 8268
 8268
```

VERWEISE: Kapitel 4.3.4 und 4.13
Anweisung: **CLEAR**
Fehler-Code: 14

ANWEISUNG: **GET**

FUNKTION: Übertragen eines logischen Records von einem Random-File in den Random-File-Puffer ("Lesen")

FORMAT:

GET [#] Filenr. [,Record-Nr.]

Filenr.: num.Ausdruck (Ergebnis: 1-15)
Record-Nr.: num.Ausdruck (Ergebnis 1-32767)

WIRKUNG: Von dem unter 'Filenr.' geöffneten Random-File wird der durch 'Record-Nr.' spezifizierte Record in den durch die letzte **FIELD**-Anweisung für die betreffende 'Filenr.' definierten Random-File-Puffer gebracht ("Lesen").

BEMERKUNGEN:

- Nach **GET** stehen die Feldinhalte in den Feldvariablen der **FIELD**-Anweisung für 'File-Nr.' zur Verfügung. Diejenigen Stringvariablen, die numerische Werte beinhalten, können mit Hilfe der Funktionen **CVI**, **CVS** oder **CVD** zurückverwandelt werden.
- Records, die vorher noch nicht beschrieben wurden (vgl. **PUT**), sind mit sovielen Zeichen mit ISO-Code Ø belegt, wie bei **OPEN** unter 'Record-Länge' angegeben wurde.
- 'Record-Nr. muß ein positiver Integer-Wert sein; Ø liefert die Fehlermeldung "Bad record number"; der maximale Wert ist 32767.
- Wird 'Record-Nr.' weggelassen, wird derjenige Record des Files mit 'Filenr.' in den Puffer übertragen, der auf den letzten durch **GET** gelesenen Record folgt ("sequentielles Lesen").
- Mit der Funktion **LOC** kann die letzte angesprochene Record-Nr. abgefragt werden.
- Wird ein **GET**-Versuch über das File-Ende hinaus gemacht, wird der Fehler "Input past end" gegeben. Mit der Funktion **EOF** kann jedoch vorher abgefragt werden, ob das Ende des Files erreicht ist.
- Der Inhalt des Puffers oder Teile davon können auch mit **INPUT#** bzw. **LINE INPUT#** Variablen(listen) zugewiesen werden.
- Jeder Versuch, dabei den Puffer über seine definierte Größe hinaus (vgl. **OPEN**) zu lesen, führt zur Fehlermeldung "FIELD overflow".

BEISPIELE:

```
10 DIM R!(5,10)
20 OPEN "R",#1,"0:RANDOM",200
30 FIELD #1,40 AS A$    ' 10 Werte von single precision!
50 FOR I%=1 TO 5
55 GET #1,I%
60 FOR J%=1 TO 10
70 R!(I%,J%)=CVS(MID$(A$,(J%-1)*4+1,4))
80 LPRINT R!(I%,J%);
85 NEXT J%
86 LPRINT :LPRINT : NEXT I%
90 LPRINT :CLOSE #1 : END
 .082254   .255843   .776169   .682012   7.32367E-02
 .378558   .956858   .372086   1.48002E-02   .596312

 .433519   .977703   .27769   .495957   .138067   .30685
 .94467   .709296   .838979   .346897

 .665805   .494194   .570306   2.68581E-02   .555742
 .650659   .360494   .603685   .836101   .287501

 .839139   .563209   .676119   .705208   4.97852E-02
 .676352   .434646   .410655   .546917   .601238

 .256648   2.92721E-02   .605242   .171833   .727317
 .630305   .119415   .26372   3.69306E-02   .730778
```

VERWEISE:

Kapitel 4.3.10.2
Anweisungen: **OPEN, FIELD, INPUT#, LINE INPUT#**
Funktionen: **EOF, LOC, CVI, CVS, CVD**
Fehler-Codes: 54, 62, 63

ANWEISUNG: **GET %**

FUNKTION: Bereiche eines Windows werden in einem Array gespeichert

FORMAT:

```
GET [% Window-Nr.,] (X1,Y1)-(X2,Y2),Arrayelement
```

Window-Nr. : numerischer Ausdruck; Wert: 1 bis 16

X1, Y1 : numerische Ausdrücke, die den Anfangspunkt des Bereiches definieren

X2, Y2 : numerische Ausdrücke, die den Endpunkt des Bereiches definieren

Arrayelement: Bezeichnung des Elementes eines eindimensionalen Integer-Arrays, bei dem mit der Speicherung begonnen wird

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, so wird das entsprechende Window gewählt. Fehlt dieser Parameter, so bezieht sich die Anweisung auf das aktive Window. Die Koordinatenpaare (X1,Y1) und (X2,Y2) definieren, bezogen auf die gültige **SCALE**-Anweisung, den linken unteren und den rechten oberen Endpunkt eines achsenparallelen Rechteckes. Der diesem Rechteck entsprechende Bereich der Bitmap wird in den angegebenen Integer-Array so übertragen, daß das angegebene Arrayelement die Anzahl Elementarpunkte des Rechteckes in der horizontalen Richtung enthält, das nächste die Anzahl in vertikaler Richtung und jedes weitere Arrayelement je 16 Bits der Bitmap, aufsteigend von links oben nach rechts unten.

BEMERKUNGEN:

- Die Anzahl der erforderlichen Arrayelelemente ergibt sich aus der Anzahl Elementarpunkte im angegebenen Rechteck dividiert durch 16 plus 2 bis 40. Die letzte Zahl ist abhängig von der Anzahl abzuspeichernder Punkte.

- Liegen Ausgangs- und Endpunkt des Rechteckes außerhalb des Windows, so wird nur der im Window liegende Teil des Rechteckes erfaßt.

BEISPIEL: Siehe **PUT %**

```
10 CLEAR
20 CURSOR POINT 0
30 SCALE 0,511,0,255
40 FOR I=1 TO 7
50 CIRCLE (250,180-5*I),40+5*I
60 CIRCLE(250,80+5*I),40+5*I
70 CIRCLE (295+5*I,130),40+5*I
80 CIRCLE (215-5*I,130),40+5*I
90 NEXT
100 DIM ARRAY%(8500)
110 GET (0,0)-(511,255),ARRAY%(0)
120 CLS : COLOR 0,1
140 CURSOR (5,8):PRINT "Das Bild wurde mit der Anweisung
 'GET%' in den Arbeitspseicher geladen"
150 PRINT TAB(5)"Es wird mit 'PUT%' wieder auf den Bilds
chirm gegeben"
160 FOR I=1 TO 5000!:NEXT:CLS
170 PUT (0,0)-(511,255),ARRAY%(0)
175 COLOR 1,0 : CALL "sp"    ' nur PR2400
180 END
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **PUT %, SCALE**
Funktion : **WINDOW**

ANWEISUNG: **GOSUB**

FUNKTION: Bewirkt den Sprung zu einer bestimmten Anweisung, bei der ein Unterprogramm beginnt

FORMAT:

```
GOSUB Zeilennummer
```

WIRKUNG: Das Programm fährt bei der durch die Zeilennummer definierten Anweisung fort. Die zuletzt ausgeführte Anweisung des Unterprogrammes muss **RETURN** sein, damit das Programm zu der Anweisung nach **GOSUB** zurückspringt.

BEMERKUNGEN:

- In einem Unterprogramm können auch mehrere **RETURN**-Anweisungen vorkommen.
- Ein Unterprogramm kann auch **GOSUB**-Anweisungen enthalten. Durch **RETURN** wird jedesmal zu der Anweisung nach dem letzten **GOSUB** gesprungen.
- Die Anzahl der möglichen Schachtelungen von Unterprogrammen ist abhängig vom Platz, der bei der Ausführung des Programmes im Arbeitsspeicher zur Speicherung der Rücksprungadressen freibleibt (Stack-Speicher).
- Der rekursive Aufruf von Unterprogrammen ist möglich.
- Wird versucht, eine nicht vorhandene Zeilennummer anzuspringen, wird "Undefined line number" gemeldet.

BEISPIELE:

```
40 GOSUB 100 : REM  -Sprung ins 1.UP
50 REM **** Rueckkehr aus 1.UP
60 GOSUB 130 : REM  -Sprung ins 2.UP
70 REM **** Rueckkehr aus 2.UP
90 END
100 LPRINT "erstes Unterprogramm erreicht"
110 RETURN
130 LPRINT "zweites Unterprogramm erreicht"
140 RETURN
```

VERWEISE: Kapitel 4.3.5 und 4.3.7.1
Anweisungen: **RETURN, ON ... GOSUB, CLEAR**
Fehler-Codes: 3, 8

ANWEISUNG: **GOTO**

FUNKTION: Sprung zu einer bestimmten Zeile des Programms

FORMAT:

GOTO Zeilennummer

WIRKUNG: Die Verarbeitung des Programmes wird mit der in **GOTO** bezeichneten Zeile fortgesetzt.

BEMERKUNGEN:
- Sprünge aus **FOR/NEXT**-Schleifen hinaus und zurück sind möglich.
- Wird versucht, eine nicht vorhandene Zeilennr. anzuspringen, wird "Undefined line number" gemeldet.

BEISPIELE:

```
10 INPUT A,B
20 LPRINT A;B;A*B
30 GOTO 10
40 END
 5  3  15
```

VERWEISE: Kapitel 4.3.5
Anweisung: **ON...GOTO**
Fehler-Code: 8

FUNKTION: **HEX$**

ZWECK: liefert den hexadezimalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung

FORMAT:

```
HEX$(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet, kaumännisch gerundet und das Ergebnis als Dezimalzahl interpretiert. Davon wird die hexadezimale Darstellung ermittelt und in einen String verwandelt.

BEMERKUNGEN:

- Das Ergebnis von 'num.Ausdruck' muß zwischen Ø und 65536 liegen, sonst wird "Overflow" gemeldet. Negative Zahlen bis -65536 werden ebenfalls umgeformt; es werden allerdings die 16-Bit-Komplemente der positiven Zahl entwickelt.
- Die hexadezimale Zahlendarstellung selbst wird in einen String verwandelt. Dabei wird kein führendes Blank und kein **&H**-Präfix vorangestellt.

BEISPIELE:

```
10 A$=HEX$(29):LPRINT A$
1D
```

VERWEISE: Kapitel 2.5.3.1 und 4.3.4.3
Funktionen: **OCT$**, **VAL**

ANWEISUNG: **IF ... THEN ... ELSE**

FUNKTION: Bedingte Verzweigung(en) in einem Programm

FORMAT:

IF log. Ausdruck { [,] **THEN** Anweisung(en) / **GOTO** Zeilennr. / **THEN** Zeilennr. } [**ELSE** { Anweisung(en) / Zeilennr. }] ...

logischer Ausdruck: Vergleichsausdruck, logischer Ausdruck oder Verknüpfung von Vergleichsausdrücken bzw. logischen Ausdrücken durch logische Operatoren; siehe Kapitel 2.7.3

WIRKUNG: Der logische Ausdruck wird berechnet, die Ergebnisse untereinander verglichen und ihr Wahrheitswert gebildet: Ist der logische Ausdruck eine Verknüpfung von Vergleichen, die mit Boole'schen Operatoren wie **AND**, **OR** etc. verknüpft sind, wird der Wahrheitswert der Verknüpfung gebildet.

Ist das Ergebnis des logischen Ausdruckes "wahr" also ungleich Ø, wird **THEN** oder **GOTO** ausgeführt.

Ist das Ergebnis des logischen Ausdruckes "unwahr", also Ø, so wird der **THEN**- oder **GOTO**-Zweig ignoriert und wenn vorhanden der **ELSE**-Zweig ausgeführt. Ist kein zugeordneter **ELSE**-Zweig vorhanden, wird die nächste Programmzeile verarbeitet. Für alle Fälle gilt: Fehlt eine Sprunganweisung oder folgt kein weiteres verwertbares **ELSE**, wird die Verarbeitung mit der nächsten Programmzeile fortgesetzt.

BEMERKUNGEN:

- Wird versucht, eine nicht vorhandene Zeilennummer anzuspringen, wird "Undefined line number" gemeldet.
- Vor dem Schlüsselwort **THEN** darf ein Komma stehen (Übersichtlichkeit des Programmes).
- **IF**-Anweisungen dürfen verschachtelt werden. Die Tiefe der Verschachtelung ist nur durch die Zeilenlänge begrenzt.

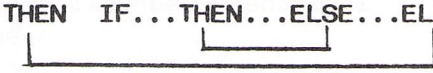
- Liegt eine verschachtelte **IF**-Folge vor, sind drei Fälle zu unterscheiden:

  . Anzahl **THEN** = Anzahl **ELSE**; jedes einzelne **ELSE** bezieht sich auf das zuletzt davor aufgetretene **THEN**.

  . Anzahl **THEN** größer Anzahl **ELSE**: jedes **ELSE** bezieht sich auf dasjenige vorangehende **THEN**, dem noch kein **ELSE** zugeordnet wurde.

Beispiel:

IF...THEN IF...THEN IF...THEN...ELSE...ELSE



Das erste **THEN** hat kein **ELSE**. Es ist somit nicht möglich, bei Nichterfüllung der ersten Bedingung auf eine Anweisung innerhalb der **IF**-Folge zu kommen.

  . Anzahl **THEN** kleiner Anzahl **ELSE**: die überflüssigen **ELSE** werden ignoriert.

BEISPIELE:

```
10 INPUT "Eingabe a und b: ",A,B
15 LPRINT A,B;TAB(30)"Druck von";
20 IF A<B THEN LPRINT A ELSE LPRINT B
25 IF A=B THEN END
30 GOTO 10
 10              20              Druck von 10
 20              10              Druck von 10
 20              20              Druck von 20
```

VERWEISE: Kapitel 4.3.5
Fehler-Code: 8

FUNKTION: **INKEY$**

ZWECK: Überprüfung, ob eine Taste an der Tastatur betätigt wurde

FORMAT:

```
INKEY$
```

WIRKUNG: Das Ergebnis der Funktion **INKEY$** ist das durch die betätigte Taste dargestellte Zeichen. Wurde keine Taste betätigt, ergibt sich der Leerstring. Das Ergebnis der Funktion wird nicht auf dem Bildschirm dargestellt.

BEMERKUNGEN:

- Das Programm arbeitet auch ohne Betätigung irgendeiner Taste weiter (im Gegensatz zu der Funktion **INPUT$**).
- Es ist jederzeit eine Eingabe möglich, ohne daß das Programm auf den Abschluß der Eingabe durch die Eingabeabschlußtaste wartet (im Gegensatz zu der Anweisung **INPUT**).
- In einer Schleife kann ständig abgeprüft werden, ob eine Taste betätigt wurde und wenn, welche.
- Es können alle Steuerzeichen eingegeben werden.
- Alle Eingabeabschlußtasten (=**CHR$(13)**) gelten als Eingabezeichen.
- **C** bewirkt eine Unterbrechung des Programmablaufs.
- Werden in einer Schleife mehrere Zeichen eingegeben und aufaddiert, so können mit **H** (=**CHR$(8)**) bereits eingegebene Zeichen gelöscht werden.

BEISPIELE:

```
10 FOR I=1 TO 10000
20 LPRINT I;
30 IF INKEY$="a" THEN 50
40 NEXT I
50 LPRINT
60 LPRINT"ENDE"
70 END
 1   2   3   4   5   6   7   8   9   10   11   12   13   14   15   16   17
 18   19   20   21   22   23   24   25   26   27   28   29   30   31
 32   33   34   35   36   37   38   39   40   41   42   43   44   45
 46   47   48   49   50   51   52   53   54   55   56   57   58   59
 60   61   62   63   64   65   66   67   68   69   70   71   72   73
 74   75   76   77   78
ENDE
```

VERWEISE:

Kapitel : 4.3.8
Anweisungen: **INPUT**, **CURSOR**, **WINDOW %**
Funktion : **INPUT$**
Assembler-Routine: **lt**

ANWEISUNG: **INPUT**

FUNKTION: Das Programm wartet auf Eingaben über die Tastatur und weist die Eingaben auf Variablen zu

FORMAT:

INPUT [;] [Stringkonstante] [{;}] Variablen(liste)

Variablenliste: Folge von numerischen und/oder Stringvariablen, die durch Kommas getrennt sind

WIRKUNG: Die 'Stringkonstante' und ggf. ein Fragezeichen werden angezeigt und das Programm wartet auf die Dateneingabe. Der Bediener kann nun die angeforderten Daten eingeben. Werden mehrere Daten erwartet, müssen diese durch Kommas getrennt eingegeben werden. Die Eingaben werden den Variablen der **INPUT**-Anweisung der Reihe nach zugewiesen. Die Zuweisung erfolgt erst, wenn der Programmablauf nach dem Eintasten aller Werte durch eine zulässige Eingabeabschlußtaste fortgesetzt wird.

BEMERKUNGEN:

- Es ist nicht möglich, bereits am Bildschirm vorhandene Eingaben durch Auslösen einer zulässigen Eingabeabschlußtaste zu übernehmen! Die Eingaben müssen erneut über Tastatur eingegeben werden. Nur die aktuelle Tastatureingabe wird übernommen!!
- Die Eingabewerte müssen mit dem Typ der entsprechenden Variablen übereinstimmen.
- Die Anzahl der eingegebenen Werte muß mit der Anzahl der Variablen in 'Variablenliste' übereinstimmen.
- Es ist nicht möglich, mit **INPUT** Eingabefehler vor der aktuellen Eingabe zu begrenzen (Einschränkung: mit **WINDOW**-Funktion definierte Windows).
- Es ist nicht möglich, vor **INPUT**-Anweisungen die Elemente der 'Variablenliste' (seien es String- oder numerische Variablen) auf eine maximale oder minimale Länge zu begrenzen.

- Stringeingaben können ohne Anführungszeichen eingegeben werden. Soll eine Stringeingabe führende oder am Ende stehende Blanks oder Kommas enthalten, muß der String mit Anführungszeichen eingegeben werden.
- Anführungszeichen können über **INPUT** nicht zum Bestandteil einer Stringeingabe werden.
- Der Strichpunkt unmittelbar nach dem Schlüsselwort **INPUT** bewirkt das Unterbleiben des Zeilenvorschubes, der sonst stets nach dem Betätigen jeder zulässigen Eingabeabschlußtaste erfolgt. Dieser Strichpunkt muß auf jeden Fall gesetzt werden, wenn eine Eingabe in der letzten Bildschirm-bzw. Windowzeile erfolgen soll. Dadurch wird das Scrolling (automatischer Vorschub des Bildschirm-bzw. Window-Inhalts um eine Zeile nach oben) verhindert.
- Der Strichpunkt direkt vor der 'Variablenliste' bewirkt, daß die Eingabe direkt hinter dem Fragezeichen und nicht in der nächsten Zeile erwartet wird.
- Das Komma direkt vor der 'Variablenliste' bewirkt, daß das Fragezeichen nicht erscheint und die Eingabe hinter der 'Stringkonstanten' erwartet wird. ACHTUNG: Im Falle des Kommas muß vorher eine 'Stringkonstante' definiert sein, notfalls der Leerstring.
- Werden bei der **INPUT**-Anweisung zu viele oder zu wenige Daten (d.h. eigentlich, zu wenige oder zu viele Kommas zur Trennung der Eingabewerte) eingegeben oder wird der Versuch gemacht, einer numerischen Variablen eine Stringeingabe durch die Eingabe zuzuordnen, erscheint die Meldung "?Redo from start". Es findet keine Zuweisung statt und alle Eingabedaten dieser **INPUT**-Anweisung müssen neu eingegeben werden.
- Der Inhalt der reservierten Variablen **ERR** und **ERL** wird durch "?Redo from start" nicht verändert; ist eine Fehlerbehandlungsroutine vorhanden, wird nicht in diese verzweigt.
- Die Fehlermeldung "Overflow" kann zwar in einer Fehlerbehandlungsroutine abgefangen werden (Fehlercode 6); die Anzeige nach Rückkehr aus der Routine (**RESUME**-Anweisung) findet dennoch statt, wenn mit **RESUME** oder **RESUME Ø** die **INPUT**-Anweisung wiederholt werden soll.
- Wird nichts eingegeben bzw. bei Variablenlisten nur die benötigte Anzahl Kommas, so erhalten num. Variablen den Wert Ø und Strings den Wert Leerstring zugewiesen.

BEISPIELE:
(↑: Position des Cursors beim Erwarten der Eingabe)

1.) 15 **INPUT**;"Eingabe ";E$
RUN
EINGABE ?
↑

2.) 25 **INPUT**;"",E$
RUN
↑

3.) 37 **INPUT**;"Ku.-Nr.,PLZ,Ort";N%,P%,O$
RUN
Ku.-Nr.,PLZ,Ort?
↑

Eingabe + zulässige Eingabeabschluß-taste	Inhalt von N%	P%	* O$
47112,6000,Ffm	47112	6000	"Ffm"
,,	0	0	""(Leerstring)
,7,	0	7	""(Leerstring)
3,,5	3	0	"5"
3, Meldung:	?Redo from Start		
,,"Ffm, Stadt"	0	0	"Ffm, Stadt"

* Die Anführungszeichen dienen hier nur der Kenntlichmachung und sind selbst nicht Bestandteil von O$.

VERWEISE: Kapitel 4.3.8
Anweisungen: **LINE INPUT, TAB, POS, CURSOR, WINDOW %**
Funktionen: **INPUT$, INKEY$**
Assembler-Routine: **lt**

ANWEISUNG: **INPUT#**

FUNKTION: Lesen von Daten von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf Variablen

FORMAT:

```
INPUT# Filenr., Variable(nliste)
```

Filenr.: num.Ausdr. (Ergebnis: 1-15)
Variablenliste: Folge von numerischen und/oder String-variablen, die durch Kommas getrennt sind.

WIRKUNG:

a) sequentielle Files:
Aus dem unter 'Filenr.' zum Input geöffneten File (Zugriffsart: "I") wird ab der Position des Pointers gelesen.

b) Random-Files:
Aus dem Random-File-Puffer, der dem unter 'Filenr.' geöffneten Random-File (Zugriffsart: "R") per **FIELD**-Anweisung zugeordnet wurde, wird von links gelesen.

Die Daten werden von links kommend elementweise auf die einzelnen Variablen der 'Variablenliste' zugewiesen.

Als Trennzeichen zwischen zwei Daten auf dem File bzw. im Random-File-Puffer werden interpretiert:

1) das Komma (,) =**CHR$**(44)
2) LF =**CHR$**(10)
3) CR =**CHR$**(13)
4) nur bei numerischen Daten:
   das Blank: =**CHR$**(32)
5) nur bei Strings:
   das Anführungszeichen ("): =**CHR$**(34),
   aber nur wenn der String-Anfang durch ein Anführungszeichen markiert war.

- numerische Werte:

Das erste Zeichen seit dem letzten Trennzeichen, das nicht ein Blank, LF oder CR ist, wird als Beginn des Wertes interpretiert.
Auf Ende des Wertes wird erkannt, sobald eines der Zeichen zu 1) bis 4) auftritt.

- Strings: Das erste Zeichen seit dem Trennzeichen, das nicht ein Blank, LF oder CR ist, wird als Beginn des Wertes interpretiert.
Auf Ende des Wertes wird erkannt, sobald eines der Zeichen zu 1) bis 3) auftritt oder auf die betroffene Stringvariable bereits 255 Zeichen zugewiesen worden sind.

Ist das erste für einen String zulässige Zeichen nach dem Trennzeichen jedoch das Anführungszeichen, wird auf das dem Anführungszeichen folgende Zeichen als erstes Zeichen des Strings erkannt (auch wenn dieses Blank, LF oder CR ist). Auf Ende des Strings wird dann erst beim nächsten Anführungszeichen erkannt. Zwischen " " stehende Trennzeichen werden in diesem Falle also nicht als solche interpretiert.

BEMERKUNGEN:

- Die eingelesenen Daten müssen für die Zuweisung formatgerecht sein.
- Bei **INPUT#** wird anders als bei **INPUT** kein ? auf dem Bildschirm ausgegeben.
- Wird während der Ausführung der **INPUT#** -Anweisung das File- bzw. Puffer-Ende erreicht, wird keine Fehlermeldung gegeben und das Ende des gerade betroffenen Datums sowie des Lesens angenommen.
- Das Anführungszeichen (") kann mit **INPUT#** nicht zum Bestandteil eines Strings gemacht werden.
- Führende Blanks oder Kommas können zum Bestandteil eines Strings gemacht werden, wenn der Wert des Strings in Anführungszeichen eingeschlossen auf das File gebracht wurde (vgl. **WRITE#** und **PRINT#**).
- Ein **INPUT#** auf die speziellen Variablen **DATE$** und **TIME$** und die reservierten Variablen **ERR** und **ERL** ist nicht möglich.

BEISPIELE:

```
10 OPEN "O",#1,"1:daten1"
20 A=1:B=2:C=3
30 PRINT#1,A,B,C
40 CLOSE#1
50 OPEN "I",#1,"1:daten1"
60 INPUT#1,A1,B1,C1
70 LPRINT A1,B1,C1
80 CLOSE
90 END
 1             2             3
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **OPEN, LINE INPUT#** , **PRINT#** , **PRINT#..USING, WRITE#**
Fehler-Code: 54

FUNKTION: **INPUT$**

ZWECK: Eine bestimmte Zeichenanzahl wird über Tastatur eingegeben oder von einem Datenfile eingelesen

FORMAT:

INPUT$(Anzahl Zeichen [,[#] Filenr.])

Anzahl Zeichen: num.Ausdruck;
bestimmt die Anzahl Zeichen, aus der das Ergebnis der Funktion besteht

Filenr. num.Ausdruck;
es ist die Nummer anzugeben, unter der das Datenfile bei der Anweisung **OPEN** geöffnet wurde

WIRKUNG: Die 'Filenr.' wird berechnet und der ganzteilige Anteil bestimmt. 'Anzahl Zeichen' wird berechnet und gerundet. Ist der Parameter 'Filenr.' angegeben, so wird aus dem durch 'Filenr.' definierten Datenfile (siehe Anweisungen **OPEN**) die durch 'Anzahl Zeichen' bestimmte Anzahl Zeichen eingelesen. Fehlt der Parameter, 'Filenr.', so ist das Ergebnis von **INPUT$** die Folge der über Tastatur eingegebenen Zeichen. Die Anzahl der zu betätigenden Tasten ergibt sich aus 'Anzahl Zeichen'. Das Ergebnis der Funktion wird nicht auf dem Bildschirm dargestellt.

BEMERKUNGEN:

- 'Anzahl Zeichen' muß einen Wert zwischen 1 und 255 aufweisen.
- Beim Aufruf **INPUT$** wartet das Programm, bis die durch 'Anzahl Zeichen' bestimmte Anzahl Zeichen eingegeben wurden (im Gegensatz zu der Funktion **INKEY$**).
- Der Abschluß der Eingabe muß nicht mit Eingabeabschlußtaste erfolgen (im Gegensatz zu der Anweisung **INPUT**), sondern wird erkannt, sobald die in 'Anzahl Zeichen' spezifizierte Anzahl Tasten betätigt wurden.
- Alle Eingabeabschlußtasten (=**CHR$**(13)) gelten als Eingabezeichen.

- Es können alle Steuerzeichen eingegeben werden.
- C bewirkt eine Unterbrechung des Programmablaufs
- H löscht bereits eingegebene Zeichen. H (=**CHR$**(8)) gilt als Betätigen einer Taste und wird mitgezählt.

BEISPIELE:

```
10 A$=INPUT$(3)
20 LPRINT A$
30 END
123
```

VERWEISE:

Kapitel 4.3.8 und 4.3.9.1
Anweisungen: **OPEN**, **INPUT**, **INPUT#**
Funktion: **INKEY$**
Fehler-Code: 54
Assembler-Routine: **lt**

FUNKTION: **INSTR**
(instring)

ZWECK: Aufsuchen der Anfangsposition eines Teilstrings in einem String

FORMAT:

```
INSTR( [Position,] Ausgangsstring, Teilstring)
```

Position: num. Ausdruck;
bestimmt Position, ab der der Teilstring gesucht werden soll

Ausgangsstring: Stringausdruck, in dem gesucht werden soll

Teilstring: Stringausdruck, der gesucht wird

WIRKUNG: Die 'Position' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet.Als Ergebnis wird die Stelle des 'Ausgangsstrings' geliefert, an der das erste Zeichen des 'Teilstrings' ab 'Position' das erste Mal im 'Ausgangsstring' auftritt.
Ist 'Position' nicht angegeben, so beginnt die Suche beim ersten Zeichen des Ausgangsstrings.

BEMERKUNGEN:

- 'Position' muß einen Wert zwischen 1 und 255 aufweisen.
- Die Funktion **INSTR** liefert als Ergebnis den Wert 0, falls:
  - . 'Position' größer als die Länge des 'Ausgangsstrings' ist.
  - . der 'Ausgangsstring' der Leerstring ist.
  - . der 'Teilstring' im 'Ausgangsstring' nicht genau enthalten ist.
- Ist der 'Teilstring' der Leerstring, so ist das Ergebnis der Funktion **INSTR** gleich 'Position', falls diese angegeben wurde, andernfalls gleich 1.

BEISPIELE:

```
10 A$="Olivetti Computersysteme"
20 LPRINT A$,"'Computer' beginnt bei Pos.";INSTR(1,A$,"Compu
ter")
30 END
Olivetti Computersysteme     'Computer' beginnt bei Pos. 10
```

VERWEISE: Kapitel 4.3.4

FUNKTION: **INT**
(integer)

ZWECK: Ermittlung derjenigen ganzen Zahl aus einem numerischen Wert, die als nächste auf der Zahlenskala links von dem angegebenen num. Wert liegt ("Gauß-Funktion")

FORMAT:

```
INT(num. Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Es wird diejenige ganze Zahl bestimmt, die als nächste links auf der Zahlenskala liegt.

BEMERKUNGEN:
- Das Ergebnis ist in der Genauigkeit des Arguments.
- Ist das Argument selbst ganzzahlig, ergibt es sich als Ergebnis.

BEISPIELE: **INT**(3.7) liefert 3 und **INT**(-3.7) liefert -4.
INT(3.2) liefert 3 und **INT**(-3.2) liefert -4.

```
10 A=-3.2:B=-3.7
20 LPRINT ABS(A);INT(ABS(A)),ABS(B);INT(ABS(B))
30 LPRINT A;INT(A),B;INT(B)
40 END
 3.2  3          3.7  3
-3.2 -4         -3.7 -4
```

VERWEISE: Kapitel 2.4 und 4.13
Funktionen: **FIX, CINT**

BEFEHL: **KILL**

FUNKTION: Löschen eines gespeicherten Files auf Diskette

FORMAT:

```
KILL Filename
```

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Das System prüft, ob das mit 'Filename' bestimmte File vorhanden ist. Wenn ja, wird dieses File gelöscht. Der dadurch frei gewordene Platz kann für die Speicherung weiterer Files wieder verwendet werden.

BEMERKUNGEN:

- Die Diskette oder das File 'Filename' darf keinen Schreibschutz haben.
- File-Password und/oder Disketten-Password, wenn vorhanden, müssen angegeben werden.
- Nachdem das File 'Filename' gelöscht wurde, kann 'Filename' zur Speicherung eines neuen Files wiederverwendet werden.

BEISPIELE: *

```
kill "0:Datenfile1"
kill "links:meinPROGRAMM"
kill "1:MOD219.asc"
```

VERWEISE: Kapitel 4.1.3

* - **KILL** löscht nur das betreffende File aus dem Inhaltsverzeichnis der Diskette und vermerkt dort, daß die betroffenen Sektoren wieder frei sind. Die Sektoren des Files selbst werden hingegen nicht mit **CHR$(Ø)** vorbelegt!

FUNKTION: **LEFT$**

ZWECK: Einem String wird von links kommend ein Teilstring entnommen

FORMAT:

```
LEFT$(Ausgangsstring, Anzahl Zeichen)
```

Ausgangsstring: Stringausdruck

Anzahl Zeichen: num. Ausdruck;
bestimmt die Anzahl Zeichen des Teilstrings

WIRKUNG: Die 'Anzahl Zeichen' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet. Aus dem 'Ausgangsstring' wird von links kommend die berechnete Anzahl von Zeichen entnommen.

BEMERKUNGEN:
- Ist die 'Anzahl Zeichen' größer als die Länge des 'Ausgangsstrings', so wird der gesamte Ausgangsstring übergeben.
- Ist die 'Anzahl Zeichen' = Ø, so wird kein Zeichen entnommen und der 'Teilstring' ist der Leerstring.
- 'Anzahl Zeichen' muß einen Wert zwischen Ø und 255 aufweisen

BEISPIELE:

```
10 A$= "Personalcomputer M 20"
20 LPRINT A$, LEFT$(A$,8)
30 END
Personalcomputer M 20        Personal
```

VERWEISE: Kapitel 4.3.4
Funktionen: **RIGHT$, MID$**

FUNKTION: **LEN**

ZWECK: Bestimmung der Länge eines Stringausdruckes

FORMAT:

```
LEN(Stringausdruck)
```

WIRKUNG: Die Funktion **LEN** ermittelt die im Stringausdruck enthaltene Anzahl von Zeichen.

BEMERKUNGEN: - Alle nicht druckbaren Zeichen und Blanks werden mitgezählt

BEISPIELE:

```
10 LPRINT "Das Wort 'Computer' enthaelt ";LEN("Computer");"
Zeichen"
Das Wort 'Computer' enthaelt  8  Zeichen
```

VERWEISE: Kapitel 4.3.4

ANWEISUNG: **LET...=**

FUNKTION: Zuweisung des Wertes eines Ausdrucks an eine Variable

FORMAT:

```
[ LET ]  { numerische Variable = num. Ausdruck  }
         { Stringvariable      = Stringausdruck }
```

WIRKUNG: Der Ausdruck wird berechnet und das Ergebnis der Variablen zugewiesen.

BEMERKUNGEN:

- Das Schlüsselwort **LET** muß nicht eingegeben werden.
- Die aktuelle Länge der Stringvariablen in einer Zuweisung ist gleich der Anzahl Zeichen, die sich als Ergebnis des Stringausdruckes ergeben.
- Die Zuweisung erfolgt in dem Datentyp, in dem die 'numerische Variable' definiert ist.

ACHTUNG: Bei der Zuweisung eines einfach genauen num. Werts auf eine doppelt genaue Variable kann Signifikanzverlust eintreten; d.h., der Inhalt der doppelt genauen Variablen ist nur annähernd gleich dem einfach genauen Wert (vgl. Kapitel 2.5.1.4, Punkt 5 sowie 2.7.1)! Diese Problematik kann mit einer Anweisung der Art

D# =VAL(STR$(G!))

umgangen werden.

- Mehrfachzuweisungen sind nicht möglich. Sie müssen durch mehrere **LET**-Anweisungen bewirkt werden.

BEISPIELE:

```
0 LET P=1 : REM entspricht P=1
20 VAR1=2*A+R22
30 STRING1$=""
50 P#=A1+SIN(2*RAD.3)
```

VERWEISE: Kapitel 4.3.3

ANWEISUNG: **LINE**

FUNKTION: Zeichnen von Linien oder Rechtecken oder mit einer Farbe gefüllten Rechtecken

FORMAT:

```
LINE [% Window-Nr.]  [STEP] [(X1,Y1)-]
  [STEP]  (X2,Y2)  , [Farbindex] , [B[F]]  [, Operand]
```

Window-Nr.: numerischer Ausdruck; Wert: 1 bis 16

X1,Y1 : numerische Ausdrücke, deren Wert den Anfangspunkt bestimmen

X2,Y2 : numerische Ausdrücke, deren Wert den Endpunkt bestimmen

Farbindex : numerischer Ausdruck, der die Farbe, bezogen auf die aktuelle globale **COLOR**-Anweisung, bekommt (Wert: Ø-3)

Operand : Operand kann sein: **AND**, **OR**, **XOR**, **NOT**, **PSET**, **PRESET**

WIRKUNG: Es wird in einem Window eine Strecke oder ein Rechteck oder ein ausgefülltes Rechteck gezeichnet. Ist der Parameter **%** Window-Nr. angegeben, so erfolgt die Ausgabe im angegebenen Window.
Fehlt dieser Parameter, so erfolgt die Ausgabe im aktiven Window.

1. Zeichnen von Strecken in Absolutkoordinaten

Eine Strecke in Absolutkoordinaten wird gezeichnet, wenn die Anweisungselemete **STEP** und **BF** nicht angegeben sind.

X1 und Y1 bestimmen den Anfangspunkt der Strecke, bewogen auf das für das Window gültige **SCALE**. Fehlt dieser Parameter, so wird die Strecke vom vorangehenden Endpunkt zu den angegebenen Koordinaten gezogen.

2. Zeichnen von Strecken in Relativkoordinaten

Wird vor einem Koordinatenpaar das Sprachelement **STEP** angegeben, so wird dieses Koordinatenpaar relativ zur zuletzt erfolgten Angabe errechnet und entsprechend die Verbindung gezeichnet.

3. Zeichnen von Rechtecken

Wird in der Anweisung das Sprachelement **B** ("box") angegeben, so bilden die Koordinatenpaare (X1,Y1) und (X2,Y2) diagonal liegende Eckpunkte eines parallel zu den Achsen liegenden Rechteckes, wobei (X1,Y1) die linke untere Ecke bestimmt. Fehlen in der Anweisung die Schlüsselwörter **STEP**, so wird das Rechteck nach Absolutkoordinaten, bezogen auf das gültige **SCALE**, gezeichnet. Ist vor einem Koordinatenpaar **STEP** angegeben, so wird dieses Koordinatenpaar relativ zu den vorangegangenen Koordinaten interpretiert.

Wird der Parameter **BF** ("box fill") anstelle von **B** angegeben, so wird das Rechteck mit der mit 'Farbindex' angegebenen Farbe ausgefüllt gezeichnet.

Farbindex

Dieser Parameter gibt an, in welcher Farbe die Strecke, das Rechteck oder das ausgefüllte Rechteck gezeichnet werden soll. Dabei gibt der Parameter 'Farbindex' die Position (Ø-3) der entsprechenden Farbe in der globalen **COLOR**=-Anweisung an. Fehlt der Parameter, so wird in der aktuellen Vordergrundfarbe gezeichnet.

Operand

'Operand' kann eines der Sprachelemente **AND**, **OR**, **XOR**, **NOT**, **PSET** oder **PRESET** sein.
Es wird auf dem für die Darstellung maßgebenden Bereich der Bitmap (vgl. Kapitel 4.3.18) die entsprechende logische Operation für jeden Elementarpunkt durchgeführt. Wird **PSET** angegeben, so werden alle betroffenen Punkte gesetzt, bei **PRESET** werden alle betroffenen Punkte auf die Farbe des Hintergrundes gesetzt.

ANMERKUNGEN:

- Strecken werden so dargestellt, daß jeweils die Punkte in der angegebenen Farbe dargestellt werden, die dem bei der Interpolation errechneten Wert am nächsten liegen.
- Wird die Strecke bezogen auf den vorangehenden Endpunkt, so kann der dann für die Strecke maßgebende Anfangspunkt z.B. durch eine zuvor ausgeführte **LINE**-, **PSET**- oder **PRESET**-Anweisung definiert werden.
- ACHTUNG:
  Wird eine Grenze der Skalierung - sei es aufgrund von Default-Wert oder aber einer **SCALE**-Anweisung - unter- bzw. überschritten, wird aus Anfangs-und Endpunkt ein Verhältnis entwickelt, das den Verlauf der Strecke bzw. die Form des Rechteckes bestimmt. Dieses Verhältnis ist von X-und Y-Koordinate abhängig. Es empfiehlt sich deshalb, mit einer **SCALE**-Anweisung statt mit Default-Werten zu arbeiten, sowie die linke und rechte **SCALE**-Grenze nicht zu unter- bzw. überschreiten.

BEISPIELE:

```
1 CLEAR
10 SCALE 0,511,0,255
30 LINE (0,50)-(320,50)
40 LINE (50,50)-(60,200),,B
50 LINE (100,50)-(115,125),,B
60 LINE (150,50)-(170,40),,B
70 LINE (200,50)-(280,190),,BF
80 CALL "=P"
90 END
```

VERWEISE : Kapitel 4.3.14 und 4.3.18
Anweisungen: **COLOR=**, **COLOR**, **SCALE**
Funktion : **WINDOW**

[redacted]: **LINE INPUT**

FUNKTION: Eingabe einer ganzen Kette von Zeichen ohne Trennzeichen und Zuweisung auf eine Stringvariable über Tastatur

FORMAT:

LINE INPUT [;] [Stringkonstante] [{;,}] Stringvariable

WIRKUNG: Die 'Stringkonstante' und ggf. ein Fragezeichen werden angezeigt und das Programm wartet auf die Dateneingabe. Die Eingabe, die durch Eingabeabschlußtaste zu beenden ist, wird auf die Stringvariable zugewiesen.

BEMERKUNGEN:

- Es ist nicht möglich, bereits am Bildschirm vorhandene Eingaben durch Auslösen einer zulässigen Eingabeabschlußtaste zu übernehmen! Die Eingaben müssen erneut über Tastatur eingegeben werden. Nur die aktuelle Tastatureingabe wird übernommen!!
- Es ist nicht möglich, bei **LINE INPUT** Eingabefelder vor der aktuellen Eingabe zu begrenzen (Einschränkung: über **WINDOW**-Funktion definierte Windows).
- Es ist nicht möglich, vor **LINE INPUT**-Anweisung die Anzahl der auf 'Stringvariable' zuzuweisenden Zeichen auf eine maximale oder minimale Länge zu begrenzen.
- Die Anzahl der eingegebenen Zeichen darf 254 nicht überschreiten.
- Die Eingabe kann ohne Anführungszeichen eingegeben werden. Führende oder am Ende stehende Blanks oder in der Eingabe enthaltene Kommas werden übernommen.
- Eingegebene Anführungszeichen werden über **LINE INPUT** zum Bestandteil der Stringeingabe gemacht.
- Der Strichpunkt unmittelbar nach dem Schlüsselwort **LINE INPUT** bewirkt das Unterbleiben des Zeilenvorschubes, der sonst stets nach dem Betätigen jeder zulässigen Eingabeabschlußtaste erfolgt. Dieser Strichpunkt muß auf jeden Fall gesetzt werden, falls die Eingabe in der letzten Bildschirm-bzw. Window-Zeile erfolgt. Nur so kann das Scrolling (Vorschub des Bildschirm-bzw. Window-Inhalts um eine Zeile nach oben) verhindert werden.

- Der Strichpunkt direkt vor der 'Stringvariablen' bewirkt, daß die Eingabe direkt hinter dem Fragezeichen und nicht in der nächsten Zeile erwartet wird.
- Das Komma direkt vor der 'Stringvariablen' bewirkt, daß das Fragezeichen nicht erscheint und die Eingabe hinter der 'Stringkonstanten' erwartet wird.

  ACHTUNG: Im Falle des Kommas muß vorher eine 'Stringkonstante' definiert sein, notfalls der Leerstring.

- Wird nichts eingegeben, so erhält die 'Stringvariable' den Wert Leerstring zugewiesen.
- Die Eingabeabschlußtaste wird nicht zum Bestandteil der Eingabe.
- Ein **LINE INPUT** auf die Variablen **DATE$** und **TIME$** sowie auf die reservierten Variablen **ERR** und **ERL** ist nicht möglich.

BEISPIELE:

```
10 LINE INPUT "Titel:";T$
20 LPRINT T$
30 END
*** Input.String **
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **INPUT, LINE INPUT#, CURSOR, WINDOW %**
Funktionen: **TAB, POS, INPUT$, INKEY$**
Assembler-Routine: **lt**

ANWEISUNG: **LINE INPUT #**

FUNKTION: Lesen einer Folge von Zeichen von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf eine Stringvariable

FORMAT:

```
LINE INPUT# Filenr.,Stringvariable
```

Filenr.: num.Ausdr. (Ergebnis: 1-15)

WIRKUNG: Die 'Filenr.' wird berechnet und auf Ganzzahligkeit gerundet.

a) sequentielle Files:
   Aus dem unter 'Filenr.' zum Input geöffneten File (Zugriffsart: "I") wird ab der Position des Pointers gelesen.
b) Random-Files:
   Aus dem Random-File-Puffer, der dem unter 'Filenr.' geöffneten Random-File (Zugriffsart: "R") per **FIELD**-Anweisung zugeordnet wurde, wird von links gelesen.

Es werden solange Zeichen eingelesen, bis ein CR (=**CHR$**(13)) oder LF (=**CHR$**(10)) erkannt wird. Die betreffende Zeichenfolge wird auf die 'Stringvariable' zugewiesen (ohne CR bzw. LF).

BEMERKUNGEN:

- Bei **LINE INPUT#** wird anders als bei **LINE INPUT** kein ? auf dem Bildschirm ausgegeben.
- Wird während der Ausführung der **LINE INPUT#** -Anweisung das File- bzw. Puffer-Ende erreicht, wird keine Fehlermeldung gegeben und auf Ende des einzulesenden Strings erkannt.
- Das Anführungszeichen (") kann mit **LINE INPUT#** zum Bestandteil des Strings gemacht werden.
- Führende Blanks oder eingeschlossene Kommas werden zum Bestandteil des Strings gemacht.

- Ein **LINE INPUT#** auf die speziellen Variablen **DATE$** und **TIME$** sowie die reservierten Variablen **ERR** und **ERL** ist nicht möglich.

BEISPIELE:

```
10 OPEN "O",#1,"1:sequen"
20 A$="Artikel":B$="Preis"
30 PRINT#1,A$;B$
40 CLOSE#1
50 OPEN "I",#1,"1:sequen"
60 LINE INPUT#1,A1$:LPRINT A1$
70 CLOSE
80 OPEN "O",#1,"1:sequen"
90 PRINT#1,A$;",";B$
100 CLOSE#1
110 OPEN "I",#1,"1:sequen"
120 LINE INPUT#1,A1$:LPRINT A1$
130 CLOSE
ArtikelPreis
Artikel,Preis
```

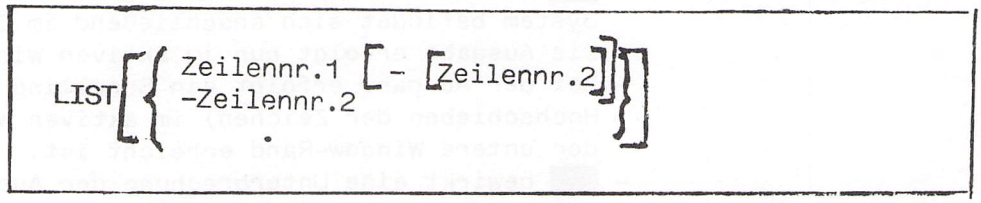
VERWEISE:

Kapitel 4.3.10.2
Anweisungen: **OPEN, INPUT# , PRINT# , PRINT#..USING, WRITE#**
Fehler-Code: 54

BEFEHL **LIST**

FUNKTION: Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programmes am Bildschirm

FORMAT:



Zeilennummer$_1$: ganze positive Zahl; gibt die Nummer der ersten auszugebenden Programmzeile an

Zeilennummer$_2$: ganze positive Zahl; gibt die Nummer der letzten auszugebenden Programmzeile an

WIRKUNG: Sind keine Operanden angegeben, so wird das im Arbeitsspeicher befindliche Programm am Bildschirm angezeigt. Die Ausgabe erfolgt beginnend mit der kleinsten Zeilennummer in aufsteigender Reihenfolge bis zur letzten Zeilennummer.
Ist nur der Operand 'Zeilennummer$_1$' angegeben, so erfolgt die Ausgabe der Programmzeile, die durch 'Zeilennummer$_1$' bestimmt ist. Der Operand '.' bestimmt, daß die zuletzt angesprochene Programmzeile ausgegeben werden soll. Der Operand '-' gibt an, daß ein Abschnitt des Programmes angezeigt werden soll. Der Abschnitt beginnt mit der ersten Programmzeile oder der im ersten Operanden angegebenen Programmzeile. Ist der Operand 'Zeilennummer$_2$' angegeben, so erfolgt die Ausgabe bis zu der Programmzeile, die durch 'Zeilennummer$_2$' bestimmt ist. Andernfalls erfolgt die Ausgabe bis zur letzten Programmzeile.

BEMERKUNGEN:
- **LIST** - hat die gleiche Funktion wie **LIST.**
- Die Ausführung des Befehls **LIST** bewirkt eine Umwandlung von Klein- in Großbuchstaben bei Schlüsselwörtern, Variablen- und Funktionsnamen. Außerdem wird das Zeichen **?** am Beginn einer Anweisung in die Anweisung **PRINT** umgewandelt.
- Nach der Ausgabe der Programmzeilen befindet sich das System im Direkt-Mode.

- Geschützte Programme (Abspeicherung durch den Befehl **SAVE** mit Operand **P**) können nicht angezeigt werden.
- **C** bewirkt eine Unterbrechung der Ausgabe. Das System befindet sich anschließend im Direkt-Mode.
- Die Ausgabe erfolgt nur im aktiven Window.
- Bei der Ausgabe erfolgt ein Scrolling (automatisches Hochschieben der Zeichen) im aktiven Window, wenn der untere Window-Rand erreicht ist.
- **S** bewirkt eine Unterbrechung der Ausgabe (und damit des Scrolling), ohne daß das System in den Direkt-Mode übergeht. Die Ausgabe kann durch Drücken einer beliebigen Taste fortgesetzt werden.
- Mit der Anweisung **WIDTH** kann festgelegt werden, wieviel Zeichen pro Zeile ausgegeben werden sollen.

BEISPIELE:

```
 list
1 ' 1.Anw.
2 ' 2.Anw.
90 letzte Anw.
ok
 list -1
1 ' 1.Anw.
ok
```

VERWEISE:

Kapitel: 3
Befehl: **SAVE**
Anweisung: **WIDTH**
Funktion: **WINDOW**

BEFEHL: **LLIST**

FUNKTION: Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programmes auf dem Drucker

FORMAT:

```
LLIST [ { Zeilennr.1 [- [Zeilennr.2]] } ]
      { -Zeilennr.2                   }
```

Zeilennummer$_1$: ganze positive Zahl; gibt die Nummer der ersten auszudruckenden Programmzeile an

Zeilennummer$_2$: ganze positive Zahl; gibt die Nummer der letzten auszudruckenden Programmzeile ab

WIRKUNG: Sind keine Operanden angegeben, so wird das gesamte im Arbeitsspeicher befindliche Programm am Drucker ausgedruckt. Die Ausgabe erfolgt beginnend mit der kleinsten Zeilennummer in aufsteigender Reihenfolge bis zur letzten Zeilennummer.
Ist nur der Operand 'Zeilennummer$_1$' angegeben, so erfolgt die Ausgabe der Programmzeile, die durch 'Zeilennummer$_1$' bestimmt ist. Der Operand '.' bestimmt, daß die zuletzt angesprochene Programmzeile ausgegeben werden soll. Der Operand '-' gibt an, daß ein Abschnitt des Programms ausgedruckt werden soll. Der Abschnitt beginnt mit der ersten Programmzeile oder der im ersten Operanden angegebenen Programmzeile. Ist der Operand 'Zeilennummer$_2$' angegeben, so erfolgt die Ausgabe bis zu der Programmzeile, die durch 'Zeilennummer$_2$' bestimmt ist. Andernfalls erfolgt die Ausgabe bis zur letzten Programmzeile.

BEMERKUNGEN:

- **LLIST** - hat die gleiche Funktion wie **LLIST**.
- Die Ausführung des Befehls **LLIST** bewirkt eine Umwandlung von Klein- in Großbuchstaben bei Schlüsselwörtern, Variablen- und Funktionsnamen. Außerdem wird das Zeichen **?** am Beginn einer Anweisung in **PRINT** umgewandelt.
- Nach der Ausgabe der Programmzeilen befindet sich das System im Direkt-Mode.

- Geschützte Programme (Abspeicherung durch den Befehl **SAVE** mit Operand **P**) können nicht ausgedruckt werden.
- C bewirkt eine Unterbrechung der Ausgabe. Das System befindet sich anschließend im Direkt-Mode.
- S bewirkt eine Unterbrechung der Ausgabe, ohne daß das System in den Direkt-Mode übergeht. Die Ausgabe kann durch Drücken einer beliebigen Taste fortgesetzt werden.
- Die Art des Drucks (Fettdruck, 10-Zolldruck, usw.) kann über Steuerzeichen vorgeschrieben werden.
- Mit dem PCOS-Befehl **sf** werden für den am System angeschlossenen Drucker Default-Werte gesetzt.
- Mit der Anweisung **WIDTH LPRINT** kann festgelegt werden, wieviel Zeichen pro Zeile ausgedruckt werden sollen.

BEISPIELE:

```
list
10 REM 1. Anweisung im Arbeitsspeicher
20 REM 2. Anw.
30 REM 3. Anw.
9999 REM letzte Anw.
ok
list 30
30 REM 3. Anw.
ok
list 20-30
20 REM 2. Anw.
30 REM 3. Anw.
ok
```

VERWEISE:

Kapitel: 3 und Anhang II
Befehl: **SAVE**
Anweisung: **WIDTH**

BEFEHL: **LOAD**

FUNKTION: Laden eines Programmes von Diskette in den Arbeitsspeicher

FORMAT:

LOAD Filename [,R]

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Das mit 'Filename' bestimmte Programm wird von der Diskette in den Arbeitsspeicher geladen. Der bisherige Inhalt des Arbeitsspeichers wird dadurch gelöscht (Programmzeilen und Variablen). Enthält der Operand 'Filename' keinen Stationsnamen, so wird das Programm auf der momentan aktiven (zuletzt angesprochenen) Station gesucht. Die Ausführung des Befehls **LOAD** ohne Operand **R** schließt alle offenen Datenfiles und bringt das Programm nicht zum Laufen.
Ist der Operand **R** angegeben, so bleiben nach dem Laden des Programmes alle bereits geöffneten Dateien offen und das Programm wird unmittelbar zum Laufen gebracht.

BEMERKUNGEN:
- Disketten- und File-Password, falls vorhanden, müssen angegeben werden.

BEISPIELE:

```
load "1:TESTPROGRAMM"
load "0/prog2"
load "disk1/pass369:p1",R
```

VERWEISE: Kapitel 3.4

FUNKTION: **LOC**

ZWECK: gibt die Anzahl gelesener bzw. beschriebener Sektoren bei sequentiellen Files oder die letzte angesprochene Record-Nr. bei Random-Files an

FORMAT: **LOC**(Filenr.)

Filenr.: num.Ausdruck (Ergebnis: 1 - 15)

WIRKUNG: Die 'Filenr.' wird berechnet. Das Ergebnis der Funktion gibt an,

a) bei sequentiellen Files:
   wieviele 256-Byte-Sektoren von dem File bereits gelesen bzw. beschrieben wurden, seit das File unter der 'Filenr.' geöffnet wurde.
b) bei Random-Files:
   welche Record-Nummer bei der letzten **GET**- bzw. **PUT**-Anweisung für das unter 'Filenr.' geöffnete Random-File verarbeitet wurde.

BEMERKUNGEN:
- Nach dem Öffnen eines Files liefert die Funktion in beiden Fällen den Wert Ø.

BEISPIELE:

```
10 OPEN "O",#1,"1:sequen"
20 MASKE$="###,#####  "
30 FOR I=1 TO 100
40 PRINT#1, USING MASKE$;RND

50 NEXT I
60 LPRINT LOC(1);" Sektoren zu 256 Byte wurden geschrieben"
70 CLOSE #1
 5  Sektoren zu 256 Byte wurden geschrieben
```

VERWEISE: Kapitel 4.3.10.2
Anweisung: **OPEN**
Funktion: **LOF**

FUNKTION: **LOF**

ZWECK: liefert für sequentielle Files die aktuelle Pointer-Position; bei Random-Files kann damit ermittelt werden, in welchem Sektor des Files der höchste beschriebene Record steht

FORMAT:

```
LOF(Filenr.)
```

Filenr.: numerischer Ausdruck (Ergebnis: 1 bis 15)

WIRKUNG: Die 'Filenr.' wird berechnet und kaufmännisch auf Ganzzahligkeit gerundet.
Von dem unter dieser 'Filenr.' geöffneten File (alle Zugriffsarten) wird ermittelt:

a) bei sequentiellen Files (Zugriffsarten: "I", "O", oder "A"): die Position des Pointers nach der letzten Operation mit dem File (wie **INPUT#** , **LINE INPUT#** , **PRINT#** , **WRITE#**).

b) bei Random-Files (Zugriffsart: "R"): die laufende Nummer desjenigen 256-Byte-Sektors im File, in den mit der bisher höchsten Record-Nr. mit **PUT** geschrieben wurde. Ausgegeben wird das Produkt aus dieser Sektor-Nummer multipliziert mit 256.

BEMERKUNGEN:

BEISPIELE :

VERWEISE: Kapitel 4.3.10.2
Anweisung: **OPEN**
Funktion: **LOC**

FUNKTION: **LOG**

ZWECK: liefert den natürlichen Logarithmus zur Basis e eines numerischen Werts

FORMAT: **LOG**(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet. Von ihm wird der natürliche Logarithmus ermittelt.

BEMERKUNGEN:
- Das Argument muß größer als Ø sein.
- Das Ergebnis ist in der Genauigkeit des Arguments.
- Logarithmen zu einer anderen Basis als e können nach folgender Formel berechnet werden (B=Basis):
  $\log_B X$=**LOG**(X)/**LOG**(B)

BEISPIELE:

```
Ø CLS
10 LPRINT LOG(2.71829);LOG(24.5748)
20 LPRINT LOG(-5)
 1   3.20172
Illegal function call in 20

Ok

exec "sp"          nur PR2400
```

VERWEISE: Kapitel 4.3.16

FUNKTION: **LPOS**
(line position)

ZWECK: liefert die aktuelle Position des Druckpointers innerhalb des Druckpuffers

FORMAT:

```
LPOS(num.Variable)
```

num.Variable: Pseudo-Argument;
darf kein Array-Element sein

WIRKUNG: Als Ergebnis wird angegeben, das wievielte Element des Druckpuffers als nächstes ausgegeben würde.

BEMERKUNGEN:
- Der Wert von 'num.Variable' wird nicht verändert.
- Es wird nicht notwendigerweise die physische Spalten-Position des Druckkopfes angegeben.

BEISPIELE:

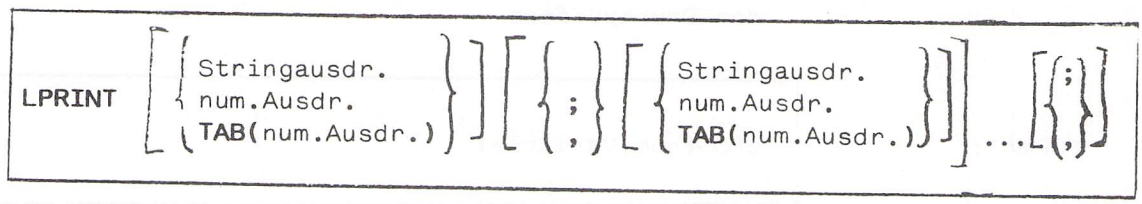
```
1 X=100
10 LPRINT "123456789";
20 POS1%=LPOS(X)
30 LPRINT"ABCDEFG"
40 POS2%=LPOS(X)
50 LPRINT POS1%,POS2%,X
123456789ABCDEFG
 10             1             100
```

VERWEISE: Kapitel 4.3.9.2
Anweisungen: **LPRINT, LPRINT USING, WIDTH**

ANWEISUNG: **LPRINT**
(line print)

FUNKTION: Ausgabe von Zahlen und/oder Strings auf dem Drucker

FORMAT:



WIRKUNG:

Die Ergebnisse der Ausdrücke werden berechnet und von links nach rechts der Reihe nach auf dem Drucker ausgegeben. Numerische Werte werden im Standardformat ausgegeben.

Die Position der Zeichen in der Druckzeile kann mit den Anweisungselementen
- **TAB**(num.Ausdruck);
- , (Komma)
- ; (Strichpunkt)

beliebig festgelegt werden.
LPRINT ohne Angabe von Ausgabeelementen bewirkt die Ausgabe einer Leerzeile.

Stellenkontrolle der Zeichen in der Druckzeile

Die **LPRINT**-Anweisung erzeugt eine der auszugebenden Folge von Ausdrücken entsprechende Folge von Zeichen in einem Puffer.

Ein Pointer weist auf die erste freie Stelle des Puffers. Erst wenn der Puffer gefüllt ist, wird sein Inhalt als eine Druckzeile ausgegeben und der Pointer auf die erste Stelle des Puffers zurückgesetzt.

Mit den Elementen , , ; und **TAB** (num.Ausdruck) kann das Ausgabeformat einer Druckzeile festgelegt werden.

Eine weitergehende Formatierung der Ausgabe kann durch **LPRINT USING** erreicht werden.

BEMERKUNGEN:
- Im Standardformat dargestellte Zahlen belegen bei Zahlen größer oder gleich Ø die erste Druckposition mit Blank, bei negativen Zahlen die erste Druckposition mit -. Außerdem wird am Ende bei allen Zahlen im Standardformat ein Blank gedruckt.

- Die Funktion **TAB** bewirkt die Ausgabe des nächsten Elementes ab einer bestimmten Position.
- Durch Verwendung des Kommas (,) als Trennzeichen wird der Puffer des Druckers in Druckzonen zu je 14 Stellen aufgeteilt. Es wird ein Druck der Ergebnisse ab den Spalten 'aktuelle Ausgabeposition', 'akt.A.p.'+14, 'akt.A.p.'+28 etc. versucht. Sind diese belegt, wird ab der nächsten freien Druckposition, ggf. in der nächsten Zeile, angedruckt.
- Durch Verwendung des Strichpunktes (;) als Trennzeichen wird erreicht, daß die nächste Ausgabe unmittelbar nach der letzten erfolgt.
- Die Ausgabe der Ausdrücke in der nächsten **LPRINT**-Anweisung im Programm erfolgt in derselben Zeile, wenn die Liste der Ausgabeelemente der vorangegangenen **PRINT**-Anweisung durch Komma oder Strichpunkt abgeschlossen wurde und aufgrund der aktuellen Position sich nicht zwingend eine Zeilenschaltung ergibt.
- Mit Hilfe der Anweisung **WIDTH LPRINT** kann der Default-Wert für die Druckzeilenlänge (132 Zeichen) im Programm verändert werden. Festlegung des Default-Werts geschieht mit dem PCOS-Befehl **sf**.
- Ist am Ende einer **LPRINT**-Anweisung kein Trennzeichen vorhanden, beginnt die Ausgabe in einer neuen Zeile.

BEISPIELE:

```
10 LINE INPUT A$
20 PRINT A$:LPRINT A$;
30 LINE INPUT B$
40 LPRINT B$
50 LINE INPUT C$
60 LPRINT TAB(50);C$
1. Eingabe2. Eingabe
                                        3. Eingabe
```

VERWEISE: Kapitel 4.3.9.2
Anweisungen: **PRINT, LPRINT USING, WIDTH**
Funktionen: **TAB, SPC, LPOS**
Fehlercode: 23

ANWEISUNG: **LPRINT USING**
(line print using)

FUNKTION: Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Drucker

FORMAT:

LPRINT [**TAB**(num.Ausdruck)] [;]
USING Formatstring; Liste von Ausdrücken [{;}]
[{,}]

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG: Die Daten der 'Liste von Ausdrücken' werden so formatiert wie durch den Anweisungsbestandteil **USING** definiert und am Drucker ausgegeben.

BEISPIELE: Siehe Anweisungsbestandteil **USING** und Anweisung **LPRINT**

VERWEISE: Siehe Anweisungsbestandteil **USING** und Anweisung **LPRINT**
Kapitel 4.3.9.2

ANWEISUNG:

LSET
(left set)

FUNKTION:

Zuweisung eines linksbündigen, rechts eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable

FORMAT:

LSET { Feldvar. / Stringvar. } =Stringausdr.

WIRKUNG:

Der 'Stringausdruck' wird berechnet. Das Ergebnis wird linksbündig in das durch 'Feldvariable' definierte Feld (vgl. **FIELD**-Anweisung) eines Random-File-Puffers gebracht, wobei Feldreste automatisch mit Blanks gefüllt werden.

Bei einer **LSET**-Zuweisung auf eine normale Stringvariable wird deren alter Inhalt linksbündig durch das Ergebnis von 'Stringausdruck' ersetzt. Diejenigen Zeichen von 'Stringvariable', die bei der linksbündigen Ersetzung nicht betroffen werden, werden durch Blanks ersetzt.

BEMERKUNGEN:

- Hat das Ergebnis von 'Stringausdruck' mehr Zeichen als 'Feldvar.' bzw. 'Stringvar.', werden die überschüssigen Zeichen rechts abgeschnitten.
- Bevor numerische Werte in den Random-File-Puffer gebracht werden, müssen sie - je nach Typ - mit Hilfe der Funktionen **MKI$**, **MKS$** oder **MKD$** in Strings konvertiert werden.

BEISPIELE:

```
10 A$="                                                  "
20 B$="                                                  "
30 LSET A$="olivetti M20":RSET B$="olivetti M20"
80 LPRINT ">";A$;"<":LPRINT ">";B$"<"
>olivetti M20                                            <
>                                  olivetti M20<
```

VERWEISE:

Kapitel 4.3.10.2
Anweisungen: **FIELD**, **PUT**, **RSET**
Funktionen: **MKI$**, **MKS$**, **MKD$**

BEFEHL: **MERGE**

FUNKTION: Kombination eines sich im Arbeitsspeicher befindlichen Programmes mit einem File, das im ASCII-Format (Parameter A) bei **SAVE** abgespeichert ist

FORMAT:

```
MERGE Filename
```

Filename: Stringausdruck, dessen Ergebnis einen Begriff ergibt, der den Regeln zur Bildung von Filenamen gerecht wird (vgl. Kapitel 4.1.3)

WIRKUNG: Es wird das unter 'Filename' angegebene Programm, das ASCII-Format haben muß, in den Arbeitsspeicher geladen. Dort wird es mit dem vorhandenen Programm kombiniert.

BEMERKUNGEN:

- Die Zeilen des Programmes im Arbeitsspeicher werden mit den Zeilen des ASCII-Files verschachtelt.
- Sind Zeilennummern in beiden Programmen identisch, so werden die Zeilen im Arbeitsspeicher durch die des ASCII-Files ersetzt.
- Sowohl das File im Arbeitsspeicher als auch das ASCII-File kann ein 'Text-File' sein, d.h. kann aus Zeilennummer plus beliebigem alphanumerischem Text bestehen.
- **MERGE** schließt offene Files und löscht alle Variablen.
- Nach der Ausführung von **MERGE** befindet sich das System im Command-Mode (vgl. Kapitel 3).

BEISPIELE:

```
save "0:PROGRAMM1"
load "1:progteil"
renum 5000,1
save "1:progteil",A
load "0:PROGRAMM1"
merge "1:progteil"
```

VERWEISE: Kapitel 3.5.5 und 4.3.13
Befehl: **SAVE**

ANWEISUNG: **MID$...=**

FUNKTION: Ersetzen eines String-Bestandteils durch einen anderen String

FORMAT:

MID$(Stringvar.,Position[,Anzahl Zeichen])=Stringausdruck

Position: num. Ausdruck
Anzahl Zeichen: num. Ausdruck

WIRKUNG: Der Stringausdruck wird berechnet.
'Position' und 'Anzahl Zeichen' werden berechnet und kaufmännisch gerundet. In 'Stringvar.' wird ab dem durch 'Position' definierten Zeichen (einschl.) die durch 'Stringausdruck' definierte Zeichenfolge eingesetzt. Dabei werden - von links kommend - nur soviel Zeichen der 'Stringvariablen' durch den 'Stringausdruck' verändert, wie durch 'Anzahl Zeichen' festgelegt.

BEMERKUNGEN:
- 'Position' muß größer als Ø sein.
- Ist 'Anzahl Zeichen' gleich Ø, wird nichts ersetzt.
- Ist das Ergebnis von 'Stringausdruck' der Leerstring, wird nichts ersetzt.
- Der Inhalt von 'Stringvar.' muß mindestens so lang sein, daß ein Zeichen existiert, das als 'Position' der Ersetzung herangezogen werden kann.
- Die von der Ersetzung nicht betroffenen Zeichen des Inhalts von 'Stringvar.' bleiben unverändert.
- Wird 'Anzahl Zeichen' nicht angegeben, wird der volle 'Stringausdruck' übernommen.
- Würde die vorherige Länge von 'Stringvar.' überschritten, wird die Ersetzung nur bis zu dieser durchgeführt, auch wenn durch 'Anzahl' explizit mehr verlangt wird.

- Die Länge des Strings, in dem ersetzt wird, kann nicht verändert (z.B. verkürzt) werden.

BEISPIELE:

```
10 A$="Textmanipulationen !"
15 LPRINT A$,
20 MID$(A$,5,14)="Variationen    "
25 LPRINT A$
40 END
Textmanipulationen !         Textvariationen
```

VERWEISE: Kapitel 4.3.2 und 4.3.4
Funktion: **MID$**

FUNKTION: **MID$**

ZWECK: Einem String wird ab einer bestimmten Position ein Teilstring entnommen

FORMAT:

MID$(Ausgangsstring, Position [,Anzahl Zeichen])

Ausgangsstring:	Stringausdruck
Position:	num. Ausdruck; bestimmt Position, ab der der Teilstring entnommen wird
Anzahl Zeichen:	num. Ausdruck; bestimmt die Anzahl Zeichen des Teilstrings

WIRKUNG: Die 'Position' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet.
'Anzahl Zeichen' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet.
Aus dem Ausgangsstring wird ab 'Position' (inklusive) ein Teilstring in der Länge von 'Anzahl Zeichen' entnommen.
Ist 'Anzahl Zeichen' nicht angegeben, so wird der gesamte Rest übernommen.

BEMERKUNGEN:
- 'Position' und 'Anzahl Zeichen' müssen einen Wert zwischen 1 und 255 aufweisen.
- Ist 'Position' größer als die Länge des Ausgangsstrings, so ist der Teilstring der Leerstring.

- Ist die Differenz zwischen Länge des Ausgangsstrings und 'Position' kleiner als 'Anzahl Zeichen' so wird der gesamte Rest übernommen.

BEISPIELE:

```
10 A$="Computersystem M 20"
20 LPRINT A$,MID$(A$,9,6)
30 END
Computersystem M 20          system
```

VERWEISE: Kapitel 4.3.4
Anweisung: **MID$...=**
Funktionen: **LEFT$, RIGHT$**

FUNKTION: **MKD$**
(make double precision)

ZWECK: Konvertierung eines doppelt genauen numerischen Wertes in einen 8 Zeichen (Bytes) langen String

FORMAT:

MKD$(num.Ausdruck)

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird in einen 8 Zeichen (Bytes) langen String verwandelt.

BEMERKUNGEN:

- Doppelt genaue numerische Werte müssen zum Schreiben in ein Random-File vorher über **MKD$** in einen String verwandelt und mit der Anweisung **LSET** oder **RSET** in den Random-File-Puffer gebracht werden.
- **MKD$** ist die Umkehrfunktion von **CVD**.

BEISPIELE:

```
10 LPRINT MKD$(1254.3),LEN(MKD$(1254.3))
@■3@■              8
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **LSET, RSET, FIELD**
Funktionen: **CVD, MKI$, MKS$**

FUNKTION: **MKI$**
(make integer)

ZWECK: Konvertierung eines Integer-Wertes in einen 2 Zeichen (Bytes) langen String

FORMAT:

```
MKI$(num.Ausdruck)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird in einen 2 Zeichen (Bytes) langen String verwandelt.

BEMERKUNGEN:
- Numerische Integer-Werte müssen zum Schreiben in ein Random-File vorher über **MKI$** in einen String verwandelt und mit der Anweisung **LSET** oder **RSET** in den Random-File-Puffer gebracht werden.
- **MKI$** ist die Umkehrfunktion von **CVI**.

BEISPIELE:

```
10 A%=899!
20 LPRINT MKI$(A%),LEN(MKI$(A%))
30 END
RUN
                2
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **LSET**, **RSET**, **FIELD**
Funktion: **CVI**, **MKS$**, **MKD$**

FUNKTION: **MKS$**
(make single precision)

ZWECK: Konvertierung eines einfach genauen numerischen Wertes in einen 4 Zeichen (Bytes) langen String

FORMAT:

```
MKS$(num.Ausdruck)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird in einen 4 Zeichen (Bytes) langen String verwandelt.

BEMERKUNGEN:
- Einfach genaue numerische Werte müssen zum Schreiben in ein Random-File vorher über **MKS$** in einen String verwandelt und mit der Anweisung **LSET** oder **RSET** in den Random-File-Puffer gebracht werden.
- **MKS$** ist die Umkehrfunktion von **CVS**.

BEISPIELE:

```
10 P!=243.55
20 LPRINT MKS$(P!),LEN(MKS$(P!))
30 END

MC=              4
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **LSET, RSET, FIELD**
Funktionen: **CVS, MKI$, MKD$**

NAME

FUNKTION: Änderung des Namens eines Programm- oder Datenfiles auf Diskette

FORMAT:

```
NAME Filename AS Filebezeichnung
```

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

Filebezeichnung: Stringausdruck; bestimmt die neue Filebezeichnung (vgl. Kapitel 4.1.3)

WIRKUNG: Dem mit Hilfe von 'Filename' spezifizierten File wird die mit 'Filebezeichnung' definierte neue Filebezeichnung zugeordnet. Der 'Filename' muß auf der angegebenen Diskette oder Station vorhanden sein. Es darf sich jedoch kein File mit dem Namen 'Filebezeichnung' auf der Diskette oder Station befinden. Nach der Ausführung des Befehls **NAME** befindet sich das angesprochene File auf derselben Diskette am gleichen Platz mit neuem Namen.

BEMERKUNGEN:

- Die Diskette oder das unter 'Filename' spezifizierte File darf keinen Schreibschutz haben.
- File-Password und Disketten-Password, wenn vorhanden, werden nicht geändert.
- File-Password und/oder Disketten-Password, wenn vorhanden, müssen bei 'Filename' angegeben werden.
- Das File-Password kann mit **NAME** nicht geändert werden, sondern nur mit dem PCOS-Befehl **fp**.

BEISPIELE:

```
name "1:alterPname" as "neuerPname"
name "FIBU:DATEN/KENNW174" as "DATEN2.rnd"
```

VERWEISE: Kapitel 4.1.3

BEFEHL: **NEW**

FUNKTION: Löschen des Arbeitsspeicher-Inhalts; Vorbereitung zur Eingabe eines BASIC-Programmes oder mit Zeilennr. versehenen Textes

FORMAT:

```
NEW
```

WIRKUNG: Der Inhalt des Arbeitsspeichers wird gelöscht. Der gesamte Datenbereich wird gelöscht. Alle offenen Datenfiles werden geschlossen. Wurde vorher der Befehl **TRON** gegeben, so wird dieser aufgehoben. Der Befehl **NEW** dient zur Vorbereitung des Systems auf die Neueingabe eines Programmes oder mit Zeilennr. versehenen Textes über die Tastatur.

BEMERKUNGEN:

- Da der bestehende Inhalt des Arbeitsspeichers durch die Eingabe des Befehls **NEW** gelöscht wird, muß vor diesem Befehl der Befehl **SAVE** stehen, wenn der alte Inhalt gespeichert werden soll.
- Soll ein Programm mit den Befehlen **RUN** oder **LOAD** von der Diskette in den Arbeitsspeicher geladen werden, ist die Ausführung des Befehls **NEW** nicht notwendig.
- Wurden vorher Windows angelegt und aktiviert, so bewirkt **NEW** nicht das Schließen aller Windows. Es sollte deshalb zusätzlich noch **CLOSE WINDOW** (z.B. im Direkt-Mode) eingegeben werden.
- **NEW** bewirkt auch nicht die Wiederherstellung des mit **ss** gesetzten Default-Werts für die Anzahl Zeichen pro Bildschirmzeile bzw. des mit **sf** gesetzten Default-Werts für die Anzahl Zeichen pro Druckerzeile, falls diese mit der Anweisung **WIDTH** bzw. **WIDTH LPRINT** vorher geändert wurden. Eine neue **WIDTH**-Anweisung - ggf. im Direkt-Mode - ist nötig.

BEISPIELE:

```
new
list
ok
```

VERWEISE: Kapitel 3
Anweisungen: **CLOSE WINDOW**, **WIDTH**

ANWEISUNG: **NEXT**

FUNKTION: Endanweisung einer oder mehrerer **FOR-NEXT** Schleifen

FORMAT:

NEXT [Laufvar.1] [,Laufvar.2] ...

Laufvar.: numerische Variable; darf nicht Element eines Arrays und nicht doppelt genau sein

WIRKUNG: siehe Anweisung **FOR**

BEMERKUNGEN:

- Die am weitesten links angegebene Laufvariable bezieht sich auf die innerste Schleife.
- Wird die 'Laufvariable' weggelassen, wird diejenige 'Laufvariable' verarbeitet, die bisher am weitesten innen steht und deren Endwert noch nicht über/unterschritten ist.
- Fallen mehrere **NEXT**-Anweisungen an einer Programmstelle zusammen, kann dies durch die Angabe aller betroffenen 'Laufvariablen', hinter einem **NEXT**, getrennt durch Komma, erklärt werden.
  Diese Formulierung kann jedoch bei bestimmten Datenkonstellationen zu einer fehlerhaften Anzahl von Schleifendurchläufen führen.

BEISPIELE: siehe **FOR**

VERWEISE: Kapitel 4.3.6
Anweisung: **FOR**, **CLEAR**
Fehler-Code: 1

ANWEISUNG: **NULL**

FUNKTION: verzögert die Ausführung von Anweisungen, die Ausgabeeinheiten betreffen

FORMAT:

```
NULL num. Ausdruck
```

WIRKUNG: Der numerische Ausdruck wird berechnet und auf Ganzzahligkeit gerundet.
Die Ausführung von Anweisungen, die Ausgaben bewirken (z. B. **PRINT**) wird verzögert. Je größer der Wert von 'num. Ausdruck ist', desto größer ist die Verzögerung.

BEMERKUNGEN:

- 'Num. Ausdruck' muß zwischen Ø und 255 liegen; sonst: Meldung "Illegal function call".
- Die Anweisung kann nur durch eine neue **NULL**-Anweisung aufgehoben werden.
- Anwendungsmöglichkeiten:
  - Verzögerung von Ausgabezeilen am Bildschirm
  - Synchronisation der Übertragungsraten an periphere Einheiten (z.B. Lochstreifenstanzer oder Teletype-kompatible Bildschirme)

BEISPIELE:

```
Ø NULL 255 : GOSUB 150
100 NULL Ø : GOSUB 150
110 STOP
150 FOR I%=1 TO 150 : PRINT I% : NEXT
155 RETURN
```

VERWEISE:

FUNKTION: **OCT$**

ZWECK: liefert den oktalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung

FORMAT:

```
OCT$(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet, kaufmännisch gerundet und das Ergebnis als Dezimalzahl interpretiert. Davon wird die oktale Darstellung ermittelt und in einen String verwandelt.

BEMERKUNGEN:

- Das Ergebnis von 'num.Ausdruck' muß zwischen Ø und 65536 liegen, sonst wird "Overflow" gegeben. Für negative Zahlen bis -65536 werden ebenfalls Umformungen vorgenommen; diese sind jedoch die oktalen Darstellungen der 16-Bit-Komplemente der positiven Zahl.
- Die oktale Zahlendarstellung selbst wird in einen String verwandelt. Es wird kein führendes Blank und kein &O-Präfix vorangestellt.

BEISPIELE:

```
10 LPRINT OCT$(800)
1440
```

VERWEISE: Kapitel 2.5.3.1
Funktionen: **HEX$, VAL**

ANWEISUNG: **ON ERROR GOTO**

FUNKTION: Fehlererkennung und Verzweigung zu einer Fehlerbehandlungsroutine

FORMAT:

ON ERROR GOTO { Ø / Zeilennummer }

Zeilennummer: Zeilennummer des Programms

WIRKUNG: Fehler, die nach der Ausführung der **ON ERROR GOTO**-Anweisung auftreten, können behandelt werden. Dabei springt das System auf die angegebene Zeilennummer und führt dort das Programm weiter aus, bis eine **RESUME**-Anweisung für den Rücksprung auftritt.

BEMERKUNGEN:

- Die Fehlerbehandlungsroutine ist ein vom Anwender definiertes Unterprogramm, das mit der **RESUME**-Anweisung endet.
- Gegebenenfalls kann die Fehlerursache in der Fehlerbehandlungsroutine behandelt werden oder aber die BASIC-Zeile wiederholt werden, die den Fehler verursachte.
- Durch eine neue **ON ERROR GOTO**-Anweisung kann eine neue Fehlerbehandlungsroutine definiert werden. Die alte Routine wird dann nicht mehr angesprungen, wenn ein Fehler auftritt.
- Wird der Parameter Ø angegeben, wird die zuletzt definierte Fehlerbehandlungsroutine desaktiviert, ohne daß eine neue gilt.

BEISPIELE:

```
10 ON ERROR GOTO 100
20 A=-2.5
30 X=LOG(2*A) : LPRINT X
50 END
100 A=ABS(A)
110 LPRINT"Log. eines negativen Wertes unmöglich - Resultat aus
Absolutwert:"
120 RESUME
Log. eines negativen Wertes unmöglich - Resultat aus Absolutwert
:
 1.60944
```

VERWEISE: Kapitel 4.3.11 und 4.3.12
Anweisungen: **RESUME**, **ERROR**
reservierte Variablen: **ERR**, **ERL**
Fehler-Codes: 8, 19

ANWEISUNG: **ON ... GOSUB**

FUNKTION: Sprung in ein Unterprogramm; das Sprungziel ist abhängig vom Wert eines num. Ausdruckes

FORMAT:

ON num. Ausdr. **GOSUB** Zeilennr. [,Zeilennr.] ...

Zeilennummer: gibt das Sprungziel an

WIRKUNG: Der numerische Ausdruck wird berechnet und das Ergebnis gerundet.
Diese ganze Zahl gibt an, zur wievielten der Zeilennummern rechts von **ON...GOSUB** gesprungen werden soll. Jede Zeilennummer in der Anweisung gibt die erste Zeile des jeweiligen Unterprogrammes an. Jedes Unterprogramm enthält die Anweisung **RETURN**, die den Rücksprung in die Anweisung nach **ON ... GOSUB** bewirkt.

BEMERKUNGEN:
- Wird versucht, eine nicht vorhandene Zeilennr. anzuspringen, wird "Undefined line number" gemeldet.
- In einem Unterprogramm können auch mehrere **RETURN**-Anweisungen vorkommen.
- Ergibt der numerische Ausdruck nach der Rundung eine Zahl, die Ø ist oder größer als die Anzahl Zeilennummern in der Anweisung **ON ... GOSUB**, wird das Programm mit der nächsten Anweisung nach **ON ... GOSUB** fortgesetzt.
- Ist der numerische Ausdruck im Ergebnis negativ oder größer als 255, erscheint die Fehlermeldung "Illegal function call".

- Die Rundung des num. Ausdruckes ist kaufmännisch.
  Achtung: -Ø.5 wird zu Ø gerundet;
  -Ø.6 zu -1.

BEISPIELE:

```
10 INPUT "Waehlen Sie ein Unterprogramm (Nr. 1-2, Ø=Ende)";I
30 LPRINT "I=";I
50 ON I GOSUB 110,130 : IF I=Ø THEN LPRINT "ENDE" : END ELSE
     ON I GOTO 10,10
90 LPRINT"Kein Unterprogramm unter dieser Nummer":GOTO 10
110 LPRINT"Das ist das erste Unterprogramm"
120 RETURN
130 LPRINT"Das ist das zweite Unterprogramm"
140 RETURN
I= 2
Das ist das zweite Unterprogramm
I= 1.5
Das ist das zweite Unterprogramm
I= 1.4
Das ist das erste Unterprogramm
I= 3
Kein Unterprogramm unter dieser Nummer
I= Ø
ENDE
```

VERWEISE: Kapitel 4.3.5 und 4.3.7
Anweisungen: **GOSUB, RETURN, CLEAR**
Fehler-Code: 8

ANWEISUNG: **ON...GOTO**

FUNKTION: Sprung in eine bestimmte Programmzeile; das Sprungziel ist abhängig vom Wert eines num. Ausdruckes

FORMAT:

ON num. Ausdr. **GOTO** Zeilennr. [,Zeilennr.] ...

Zeilennummer: gibt das Sprungziel an

WIRKUNG: Der numerische Ausdruck wird berechnet und das Ergebnis gerundet. Diese ganze Zahl gibt an, zur wievielten der Zeilennummern rechts von **ON...GOTO** gesprungen werden soll.

BEMERKUNGEN:

- Wird versucht, eine nicht vorhandene Zeilennr. anzuspringen, wird "Undefined line number" gemeldet.
- Ergibt der numerische Ausdruck (nach der Rundung) eine Zahl, die Ø ist oder größer als die Anzahl Zeilennummern der Anweisung **ON...GOTO**, wird kein Sprung ausgeführt, sondern das Programm mit der nächsten Anweisung nach **ON...GOTO** fortgesetzt.
- Ist der numerische Ausdruck im Ergebnis negativ oder größer als 255, erscheint die Fehlermeldung "Illegal function call".
- Die Rundung des num. Ausdrucks ist kaufmännisch.
  ACHTUNG: -Ø.5 wird zu Ø gerundet;
  -Ø.6 zu -1.

BEISPIELE:

```
10 INPUT "Waehlen Sie ein Unterprogramm (Nr. 1-2, 0=Ende)";I
30 LPRINT "I=";I
50 ON I GOSUB 110,130   IF I=0 THEN LPRINT "ENDE" : END ELSE
      ON I GOTO 10,10
90 LPRINT"Kein Unterprogramm unter dieser Nummer":GOTO 10
110 LPRINT"Das ist das erste Unterprogramm"
120 RETURN
130 LPRINT"Das ist das zweite Unterprogramm"
140 RETURN
I= 2
Das ist das zweite Unterprogramm
I= 1.5
Das ist das zweite Unterprogramm
I= 1.4
Das ist das erste Unterprogramm
I= 3
Kein Unterprogramm unter dieser Nummer
I= 0
ENDE
```

VERWEISE: Kapitel 4.3.5
Anweisung: **GOTO**
Fehler-Code: 8

ANWEISUNG: **OPEN**

FUNKTION: Öffnen eines sequentiellen oder Random-Files für Zugriffe (evtl. auch Neuanlage eines nicht vorhandenen Files bzw. Vorbereitung zur automatischen Verlängerung eines vorhandenen Files)

FORMAT:

OPEN Zugriffsart,[#] Filenr.,Filename [,Record-Länge]

Zugriffsart:	String-Ausdr. (Ergebnis:"I", "O", "A" oder "R")
Filenr.:	num.Ausdr. (Ergebnis: 1 bis 15)
Filename:	String-Ausdr., muß als Ergebnis einen String liefern, der den Regeln zur Bildung von Filenamen genügt (vgl. Kapitel 4.1.3)
Record-Länge:	num.Ausdr. (Ergebnis: 1 bis beliebig)

WIRKUNG: Die 'Filenr.' und die 'Record-Länge' werden errechnet und ggf. durch kaufmännische Rundung der ersten Nachkommastelle davon der ganzzahlige Wert ermittelt.

Ist das durch 'Filename' definierte File auf der angesprochenen Diskette noch nicht vorhanden und die Zugriffsart ist "O" oder "R", so wird es auf Diskette angelegt.

Es erhält die ermittelte 'Filenr.' zugeordnet, unter der es von da an anzusprechen ist.

Liefert 'Zugriffsart' den Wert "I", wird das File als sequentielles File nur für Inputs geöffnet, bei "O" als sequentielles File nur für Outputs ab File-Beginn. Zugriffsart "O" löscht sofort den gesamten bisherigen File-Inhalt. Ist der Wert "A", wird das File als sequentielles File nur für Outputs ab dem bisherigen logischen File-Ende geöffnet ("Append"-Wirkung). Ist der Wert "R", wird es als Random-File sowohl für Inputs als auch für Outputs geöffnet.

Handelt es sich um 'Zugriffsart' "R", so muß 'Recordlänge' angegeben sein; bei allen anderen 'Zugriffsarten' darf 'Recordlänge' nicht angegeben sein.

BEMERKUNGEN:

- Noch nicht vorhandene Datenfiles sollten stets mit dem PCOS-Befehl **fn** angelegt werden.
- Die Angabe von 'Record-Länge' muß bei Random-Files immer erfolgen; bei sequentiellen Files hingegen führt sie zur Fehlermeldung "Illegal function call".
- Die 'Record-Länge' bei Random-Files ist beliebig, allerdings größer als Ø.
- Die Anzahl Filenummern, die mit **OPEN**-Anweisungen vergeben werden können (maximal 15) sowie die maximal einsetzbare Record-Länge wird mit dem PCOS-Befehl **sb** festgelegt. Normalerweise sind die Default-Werte 3 bzw. 128. Die Fehlermeldungen "Bad file number" oder "Illegal function call" können aus solchen zu niedrigen Default-Werten herrühren.
- Ist die 'Filenr.' im Programm bereits vergeben, wird die Fehlermeldung "File already open" gegeben.
- Files auf Disketten mit Schreibschutz können geöffnet, aber später nicht durch **PUT**, **PRINT#** , **PRINT#...USING**, oder **WRITE#** beschrieben werden.
- Disketten-Passwords oder File-Passwords müssen beim 'Filenamen' enthalten sein.
- Wird der 'Filename' auf Diskette nicht gefunden, wird die Fehlermeldung "File not found" gegeben. Dies gilt nur bei Zugriffsart "**i**", angewandt auf alle Typen von Datenfiles. Für die anderen Zugriffsarten gilt: Ist das File nicht vohanden, wird es angelegt. Reicht dafür der Platz aufgrund des PCOS-Befehls **ss** nicht aus, wird der Fehler 61 ("Disk Full") gemeldet.
- Ist das Password des Files, falls vergeben, nicht korrekt angegeben, wird die Fehlermeldung "Invalid Password" gegeben.
- Entspricht die 'Record-Länge' nicht der bei Anlage des Files festgelegten, wird keine Fehlermeldung gegeben!!
- Wird nach **OPEN** ein File automatisch angelegt oder vergrößert, so werden im Inhaltsverzeichnis zwar die dabei betroffenen Sektoren als "belegt" vermerkt; die Sektoren des Files selbst werden hingegen nicht mit **CHR$(Ø)** vorbelegt!

BEISPIELE:

```
10 REM das sequentielle File 'sequen' wird in den INPUT-Mode
 gesetzt
20 OPEN "I",#1,"1:sequen"
30 '
60 REM das sequentielle File 'sequen1' wird in den APPEND-Mo
de gesetzt
65 OPEN "A",#2,"1:sequen1"
70 '
100 REM das sequentielle File 'sequen' wird geschlossen und
in den OUTPUT-Mode gesetzt
120 CLOSE #1
130 OPEN "O",#1,"1:sequen"
150 '
160 REM das Random-File 'random' wird in den INPUT/OUTPUT -M
ode gesetzt
170 OPEN "R",#3,"1:random",128
190 '
200 REM alle offenen Files werden geschlossen
210 CLOSE 1,2,3
220 END
```

VERWEISE:

Kapitel 4.1.3 und 4.3.10

Anweisungen: **CLOSE, FIELD, GET, PUT, RSET, LSET, INPUT# , LINE INPUT# , PRINT# , PRINT# USING, WRITE#**

Funktionen: **CVI, MKI$, CVS, MKS$, CVD, MKD$, EOF, LOC, LOF**

Fehler-Codes: 53, 54, 55, 61, 64, 67

ANWEISUNG: **OPTION BASE**

FUNKTION: Festlegung des kleinsten möglichen Index-Wertes aller Arrays

FORMAT:

OPTION BASE { Ø / 1 }

WIRKUNG: Es wird der kleinste mögliche Index-Wert für alle Arrays des Programmes festgelegt. Der kleinstmögliche Index-Wert kann Ø oder 1 sein.

BEMERKUNGEN:

- Der Default-Wert ist Ø, d.h., daß alle Arrays mit dem Index Ø beginnen, falls keine **OPTION BASE**-Anweisung existiert.
- Es darf pro Programm nur eine **OPTION BASE**-Anweisung (vor allen **DIM**-Anweisungen) vorkommen.
- Wird ein Array mit höchstem Index X dimensioniert (**DIM** A(X)), so enthält der Array

  bei **OPTION BASE** 1 X Elemente und
  bei **OPTION BASE** Ø X+1 Elemente.

BEISPIELE:

```
10 OPTION BASE 0
20 A(5,5)=234:LPRINT A(5,5)
30 LPRINT B(0,0)
 234
 0
```

```
1 CLS
10 OPTION BASE 1
20 A(5,5)=234:LPRINT A(5,5)
30 LPRINT B(0,0)
40 END
 234
Subscript out of range in 30

Ok

exec "sp"   'nur PR2400
```

VERWEISE: Kapitel 2.5.3.3 und 4.3.2
Anweisung: **DIM**

ANWEISUNG: **PAINT**

FUNKTION: Ausfüllen eines ganzen oder eines Teils eines Windows

FORMAT:

PAINT [%Window-Nr.,] (X,Y)[,Farbindex1][,Farbindex2]

Window-Nr.:	numerischer Ausdruck (Ergebnis: 1 bis 16)
X, Y:	numerische Ausdrücke
Farbindex1, Farbindex2:	numerische Ausdrücke (Ergebnis: Ø bis 3)

WIRKUNG: Ausgehend vom Koordinatenpunkt (X,Y) wird ein zuvor definierter geschlossener Bereich (Poligon) mit der durch 'Farbindex1' angegebenen Farbe ausgefüllt. Liegt der Punkt (X,Y) außerhalb des Polygons, so wird der Rest des Windows ausgefüllt. Durch 'Farbindex2' wird die Farbe bestimmt, mit der der Rand des Polygons gezeichnet wird.

BEMERKUNGEN:

- Die Parameter 'Farbindex1' und 'Farbindex2' sind Zahlen zwischen Ø und 3, die der Position der Farbdefinition in der globalen **COLOR=**-Anweisung entsprechen.
- Der Default-Wert für 'Farbindex1' und 'Farbindex2' ist die Vordergrundfarbe.

BEISPIELE:

```
10 CLS
20 CIRCLE (200,100),80
30 PAINT (200,100),1
40 CALL "sp" ' nur PR2400
```

VERWEISE: Kapitel 4.3.14
Anweisung: **COLOR=**

FUNKTION: **POINT**

ZWECK: Ermittlung der Farbe in einem Koordinatenpunkt

FORMAT:

```
POINT(X,Y)
```

X,Y: numerische Ausdrücke, die den zu betrachtenden Elementarpunkt bestimmen

WIRKUNG: Für die Koordinatenwerte 'X' und 'Y' wird entsprechend dem für das aktive Window geltenden Maßstab der zugehörige Elementarpunkt ermittelt. Es wird derjenige Farbindex innerhalb der globalen **COLOR=**-Anweisung ermittelt, der der Farbe entspricht, die sich in dem Elementarpunkt befindet.

BEMERKUNGEN:

BEISPIELE:

```
10 CLS :COLOR = 1,2,3,0
20 SCALE -50,50,-50,50
30 CIRCLE (0,0),30,,,.5
40 PAINT (0,0),3,2
50 FARBINDEX%=POINT(30,50/30*.5)
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisungen: **COLOR**, **COLOR=**, **SCALE**, **WINDOW %**
Funktion: **WINDOW**

FUNKTION: **POS**
(position)

ZWECK: liefert die aktuelle Spalten- oder Zeilen-Position des Bildschirm-Cursors

FORMAT:

```
POS(Position)
```

Position: num. Ausdruck (Ergebnis: Ø oder 1)

WIRKUNG: 'Position' wird berechnet und kaufmännisch gerundet. Ist das Ergebnis Ø, liefert die Funktion die aktuelle Spalten-Position, an der sich der Bildschirm-Cursor befindet. Ist 'Position' nicht gleich Ø, liefert die Funktion die aktuelle Zeilenposition des Bildschirm-Cursors.

BEMERKUNGEN:
- Die Spalten- bzw. Zeichen-Position bezieht sich stets auf das aktuelle Window.

BEISPIELE:

```
10 CLS
20 A=POS(0):B=POS(1):FOR I=1 TO 10000:NEXT
30 PRINT "der Cursor war in Zeile ";B;" position";A
40 END
```

VERWEISE: Kapitel 4.3.9.1
Anweisungen: **PRINT, PRINT USING, CURSOR, LINE INPUT, INPUT, WINDOW%, CLS**
Funktionen: **INPUT$, INKEY$, WINDOW**

ANWEISUNG: **PRESET**
(pixel reset)

FUNKTION: Zeichnen eines Elementarpunktes in der Hintergrundfarbe

FORMAT:

PRESET [**%**Window-Nr.] (X,Y)

Window-Nr.: numerischer Ausdruck
(Ergebnis: 1 bis 16)

X, Y: numerische Ausdrücke

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, so wird das entsprechende Window gewählt. Fehlt der Parameter, so wird die Anweisung im aktiven Window ausgeführt. Das Koordinatenpaar (X,Y) bezieht sich auf die für das Window gültige **SCALE**-Anweisung. Der entsprechend dem Maßstab diesen Koordinaten zugeordnete Elementarpunkt wird auf die aktuelle Hintergrundfarbe gesetzt.

BEMERKUNGEN: siehe **PSET**

BEISPIELE:

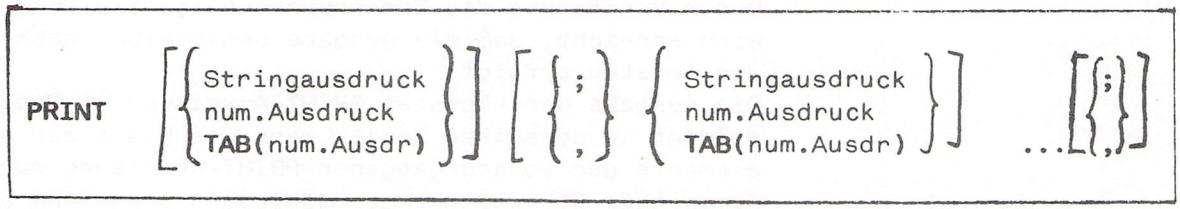
```
1 CLEAR
10 SCALE 0,100,-100,0
20 FOR I%=1 TO 100
30 PSET (I%/2,-I%) : NEXT
40 PRESET (25,-50) : PRESET (26,-52) : PRESET (28,-56)
45 K$=INKEY$ : IF K$="" THEN 45
50 END
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **COLOR=**, **COLOR**, **SCALE**

ANWEISUNG: **PRINT**

FUNKTION: Druck von Zahlen und Strings im Standardformat auf dem Bildschirm

FORMAT:



, oder ; sind zur Trennung der Ausgabeelemente möglich

WIRKUNG: Die Ergebnisse der Ausdrücke werden berechnet und von links nach rechts der Reihe nach am Bildschirm ausgegeben. Numerische Werte werden im Standardformat ausgegeben.

Die Position der Zeichen in der ausgegebenen Zeile kann mit den Anweisungselementen

- **TAB** (num.Ausdruck);
- , (Komma)
- ; (Strichpunkt)

beliebig festgelegt werden.
PRINT ohne Angabe von Ausgabeelementen bewirkt die Ausgabe einer Leerzeile.

Mit den Elementen , , ; und **TAB** (num.Ausdruck) kann das Ausgabeformat einer Ausgabezeile festgelegt werden.

Eine weitergehende Formatierung der Ausgabe kann durch **PRINT USING** erreicht werden.

BEMERKUNGEN:

- Bei Verwendung des Kommas (,) als Zeichen zur Trennung der ausgegebenen Daten wird die Bildschirmzeile ab der aktuellen Ausgabeposition in Druckzonen zu je 14 Stellen aufgeteilt. Es wird ein Druck der Ergebnisse ab den Spalten 'aktuelle Ausgabeposition', 'akt.A.p.'+14, 'akt.A.p.'+28 etc. versucht. Sind diese belegt, wird ab der nächsten freien Position, ggf. in der nächsten Zeile, ausgegeben.

- Im Standardformat dargestellte Zahlen belegen bei Zahlen größer oder gleich Ø die erste Position mit Blank, bei negativen Zahlen die erste Position mit -. Außerdem wird am Ende bei allen Zahlen im Standardformat ein Blank erzeugt.
- Die Funktion **TAB** bewirkt die Ausgabe des nächsten Elementes ab einer bestimmten Position.
- Durch Verwendung des Strichpunktes (;) als Trennzeichen wird erreicht, daß die Ausgabe unmittelbar nach der letzten erfolgt.
- Die Ausgabe der nächsten **PRINT**-Anweisung im Programm erfolgt in derselben Zeile, wenn die Liste der Ausgabeelemente der vorangegangenen **PRINT**-Anweisung durch Komma oder Strichpunkt abgeschlossen wurde und aufgrund der aktuellen Position sich nicht zwingend eine Zeilenschaltung ergibt.
- Mit Hilfe der Anweisung **WIDTH** kann der Default-Wert für die Bildschirmzeilenlänge (64 oder 8Ø Zeichen) im Programm verändert werden. Die Festlegung des Default-Werts geschieht mit dem PCOS-Befehl **s$**.
- Ist am Ende einer **PRINT**-Anweisung kein Trennzeichen vorhanden, beginnt die Ausgabe in einer neuen Zeile.
- Anstelle des Schlüsselwortes **PRINT** kann ein **?** (Fragezeichen) eingegeben werden. Dieses wird in Listings und beim Speichern des Programms durch **PRINT** ersetzt.

BEISPIELE:

```
1 REM Dieses Programm gibt den Zeichenvorrat des BILDSCHIRMS
          bei nationaler deutscher Tastatur aus
10 PRINT"Zeichenvorrat bei nationaler deutscher Tastatur:"
20 PRINT
30 FOR I=33 TO 126
40 PRINT CHR$(I);
50 NEXT
60 END
```

VERWEISE: Kapitel 4.3.9.1
Anweisungen: **LPPRINT**, **PRINT USING**, **PRINT#**, **WIDTH**
Funktionen: **TAB**, **SPC**

ANWEISUNG: **PRINT#**

FUNKTION: Schreiben von Daten auf ein externes sequentielles File; Trennzeichen werden nicht automatisch auf das File geschrieben; am Ende wird ggf. ein CR und ein LF auf das File geschrieben

FORMAT:

PRINT# Filenr.,Liste von Ausdrücken [;] [,]

Filenr.: num.Ausdruck (Ergebnis: 1-15)

Liste von Ausdrücken: numerisch und/oder String; getrennt durch , oder ;

WIRKUNG: Die 'Filenr.' wird berechnet und gerundet.
Auf das unter dieser Filenr. geöffnete sequentielle File (Zugriffsart: "O" oder "A") werden ab der Position des Pointers die Ergebnisse der (durch Trennzeichen getrennten) Ausdrücke geschrieben.
Als Zeichen zur Trennung der Ausdrücke in der 'Liste von Ausdrücken' gelten:
das Komma (,) und
der Strickpunkt (;).

a) zwischen numerischen Ausdrücken:
- Es empfiehlt sich die Trennung durch Strichpunkt (;). Die numerischen Daten werden im Standardformat (vgl. **PRINT**-Anweisung) direkt hintereinander auf das File geschrieben.
- Das Komma (,) als Trennzeichen bewirkt, daß unter Verwendung von Blanks auf das File tabuliert wird (vgl. Funktion von , bei **PRINT**-Anweisungen).

b) zwischen Stringausdrücken:
- Auch hier empfiehlt sich die Trennung durch Strichpunkt (;). Dadurch werden allerdings die Daten direkt hintereinander, ohne Kennzeichnung durch Anführungszeichen, auf das File gebracht (vgl. Funktion von ; bei **PRINT**-Anweisungen), so daß eine Trennung bei späterem **INPUT#** nicht mehr möglich ist. Es müssen also zusätzlich solche Zeichen zwischen String-Daten auf das File geschrieben werden, die bei **INPUT#** bzw. **LINE INPUT#** als Trennzeichen erkannt werden (z.B. Komma (=**CHR$**(44)) oder Anführungszeichen (=**CHR$**(34)).

- Enthält ein String-Datum ein Komma oder sonstige bei **INPUT#** als Trennzeichen wirkende Zeichen, so muß der String mit **CHR$(34)** von den anderen Datenelementen getrennt auf das File geschrieben werden.

Am Ende einer auf das File geschriebenen 'Liste von Ausdrücken' werden ein CR (=**CHR$(13)**) und ein LF (=**CHR$(10)**) auf das File geschrieben. Diese Trennzeichen entfallen jedoch, wenn am Ende der 'Liste von Ausdrücken' ein ; steht.

BEMERKUNGEN:

- **PRINT#** ist bei Disketten mit Schreibschutz nicht möglich und führt zum Fehler "Disk I/O-Error".

- Numerische Werte werden im Standardformat auf das File gebracht (vgl. **PRINT**).
- Enthalten Stringvariablen führende oder am Ende stehende Blanks, werden diese auf das File geschrieben.
- **PRINT#** schreibt die Daten im Grunde so auf das File, wie sie durch **PRINT**-Anweisung auf dem Bildschirm dargestellt würden, allerdings einschließlich nicht druckbarer Zeichen.
- Im Gegensatz zu **WRITE#** werden die Daten nicht automatisch durch ein Trennzeichen getrennt auf das File gebracht; die Strings werden nicht durch Anführungszeichen gekennzeichnet und das Ende der 'Liste von Ausdrücken' nur auf Wunsch mit CR und ggf. LF gekennzeichnet (Weglassen des ; am Ende der 'Liste von Ausdrücken').
- Wird die bisherige (oder durch PCOS-Befehl **fn** festgelegte) File-Größe überschritten, wird automatisch versucht, für das File soviel neue Sektoren auf Diskette zu reservieren, wie durch den PCOS-Befehl **ss** spezifiziert wurde. Dies kann sofort zum Fehler "Disk full" führen, obwohl für den Record selbst noch ausreichend Platz vorhanden wäre.

- Mit Hilfe der Funktion **LOC** kann abgefragt werden, ob auf Diskette noch genügend Platz für die von **PRINT#** betroffenen Daten ist.

BEISPIELE:

```
10 OPEN "O",#1,"1:daten1"
20 A=1:B=2:C=3
30 PRINT#1,A,B,C
40 CLOSE#1
50 OPEN "I",#1,"1:daten1"
60 INPUT#1,A1,B1,C1
70 LPRINT A1,B1,C1
80 CLOSE
90 END
 1              2              3
```

VERWEISE:

Kapitel 4.3.10.2
Anweisungen: **OPEN, INPUT# , LINE INPUT# , WRITE# , PRINT**
Funktion: **LOC, LOF**
Fehler-Codes: 54, 61

ANWEISUNG: **PRINT USING**

FUNKTION: Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Bildschirm

FORMAT:

PRINT [**TAB**(num.Ausdruck)] [;]
USING Formatstring; Liste von Ausdrücken [{;/,}]

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG: Die Daten der 'Liste von Ausdrücken' werden so formatiert wie durch den Anweisungsbestandteil **USING** definiert und am Bildschirm angegeben.

BEISPIELE: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT**

VERWEISE: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT** Kapitel 4.3.9.2

ANWEISUNG: **PRINT# ... USING**

FUNKTION: Schreiben von formatierten Daten (numerisch und/oder String) auf ein sequentielles File

FORMAT:

PRINT# Filenr. **USING** [**TAB**(num.Ausdruck)] [;]
Formatstring; Liste von Ausdrücken [{;,}]

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder Stringausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG: Die 'Filenr.' wird berechnet und kaufmännisch auf Ganzzahligkeit gerundet. Die Daten der 'Liste von Ausdrücken' werden so formatiert wie durch den Anweisungsbestandteil **USING** definiert und in dieser Form auf das unter 'Filenr.' geöffnete File (Zugriffsart "O" oder "A") geschrieben.

BEISPIELE:

```
10 OPEN "O",#1,"1:sequen"
20 MASKE$="###.#####  "
30 FOR I=1 TO 10
40 PRINT#1, USING MASKE$;RND
50 NEXT I
60 CLOSE #1
70 OPEN "I",#1,"1:sequen"
80 IF EOF(1) THEN 90 ELSE INPUT#1,A$:LPRINT A$:GOTO 80
90 CLOSE
 .08225
 .25584
 .77617
 .68201
 .07324
 .37856
 .95686
 .37209
 .01480
 .59631
```

VERWEISE: Siehe Anweisungsbestandteil **USING** und Anweisung **PRINT#** Kapitel 4.3.10.2

ANWEISUNG: **PSET**
(pixel set)

FUNKTION: Darstellung eines Elementarpunktes in vorgewählter Farbe

FORMAT:

```
PSET [% Window-Nr.] (X,Y) [,Farbindex]
```

Window-Nr.: numerischer Ausdruck; Wert 1 bis 16

X,Y : numerische Ausdrücke, die die Koordinaten des gewählten Punktes bestimmen

Farbindex : numerischer Ausdruck; Wert Ø bis 3

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, so wird das dadurch bestimmte Window gewählt. Fehlt dieser Parameter, so bezieht sich die Anweisung auf das aktive Window.
Gemäß der für das Window gültigen **SCALE**-Anweisung wird derjenige Elementarpunkt ermittelt, der dem Koordinatenpaar (X,Y) am nächsten liegt. Dieser Elementarpunkt erhält die Farbe, die der durch 'Farbindex' definierten Position in der geltenden globalen **COLOR**=-Anweisung entspricht. Ist 'Farbindex' nicht angegeben, wird der Punkt in der aktuellen Vordergrundfarbe dargestellt.

BEMERKUNGEN:
- Liegen die durch 'X' und 'Y' bestimmten Koordinaten außerhalb des definierten Windows, so hat **PSET** keine Wirkung.

BEISPIELE:

```
10 CLS
20 FOR I=1 TO 100
25 PSET (3*I+10,RND*I*5) 1
30 NEXT
```

VERWEISE : Kapitel 4.3.14 und 4.3.18
Anweisungen: **COLOR**, **COLOR=**, **SCALE**

ANWEISUNG: **PUT**

FUNKTION: Übertragen eines logischen Records von einem Random-File-Puffer in das zugehörige Random-File ("Schreiben")

FORMAT:

PUT [#] Filenr. [,Record-Nr.]

Filenr.: num.Ausdruck (Ergebnis: 1-15)
Record-Nr.: num.Ausdruck (Ergebnis: 1-32767)

WIRKUNG: Aus dem Random-File-Puffer wird der durch 'Record-Nr.' spezifizierte Record in das unter 'Filenr.' geöffnete Random-File gebracht ("Schreiben").

BEMERKUNGEN:

- **PUT** ist bei Diskette mit Schreibschutz nicht möglich und führt zum Fehler "Disk I/O-Error".

- 'Record-Nr.' muß ein positiver Integer-Wert sein; Ø liefert die Fehlermeldung "Bad record number"; der maximale Wert ist 32767.
- Wird 'Record-Nr.' weggelassen, wird der Puffer-Inhalt unter derjenigen Record-Nr. in das mit 'File-Nr.' spezifizierte File gebracht, die auf den letzten durch **PUT** geschriebenen bzw. durch **GET** gelesenen Record folgt ("sequentielles Schreiben").
- Wird die bisherige (oder durch PCOS-Befehl **fn** festgelegte) File-Größe überschritten, wird automatisch versucht, für das File soviel neue Sektoren auf Diskette zu reservieren, wie durch den PCOS-Befehl **ss** spezifiziert wurde. Dies kann sofort zum Fehler "Disk full" führen, obwohl für den Record selbst noch ausreichend Platz vorhanden wäre.
- Mit der Funktion **LOC** kann die letzte angesprochene Record-Nr. abgefragt werden.
- Im Random-File-Puffer ist vor **PUT** durch **FIELD**-Anweisung Platz für die Feldvariablen zu reservieren, die anschließend (evtl. unter Zuhilfenahme der Funktionen **MKI$**, **MKS$** oder **MKD$**) über **LSET** bzw. **RSET** Werte zugewiesen erhalten.

- Eine andere Möglichkeit, den Random-File-Puffer zu füllen, besteht durch Einsetzen der Anweisungen **PRINT#**, **PRINT#...USING** oder **WRITE#**.
  Im Falle des **WRITE#** wird der Puffer rechts mit Blanks aufgefüllt.

BEISPIELE:

```
1 DIM R!(50) : FOR J%=1 TO 5
2 FOR I%=1 TO 10 : Z%=Z%+1 : R!(Z%)=RND
3 LPRINT R!(Z%); : NEXT I%
4 LPRINT :LPRINT : NEXT J%
5 OPEN "R",#1,"0:RANDOM",200   ' = 5 x 10 Werte von single
precision
6 FIELD #1, 40 AS A$    ' 10 Werte von single precision
10 FOR J%=1 TO 5
11 U$=""
12 FOR I%=1 TO 10
13 Z!=Z!+1
18 U$=U$+MKS$(R!(Z!))
19 NEXT I%
20 LSET A$=U$
30 PUT #1,J%
85 NEXT J%
90 CLOSE #1 : END
 .082254   .255843   .776169   .682012   7.32367E-02
 .378558   .956858   .372086   1.48002E-02   .596312

 .433519   .977703   .27769    .495957   .138067   .30685
 .94467    .709296   .838979   .346897

 .665805   .494194   .570306   2.68581E-02   .555742
 .650659   .360494   .603685   .836101   .287501

 .839139   .563209   .676119   .705208   4.97852E-02
 .676352   .434646   .410655   .546917   .601238

 .256648   2.92721E-02   .605242   .171833   .727317
 .630305   .119415   .26372   3.69306E-02   .730778
```

VERWEISE:

Kapitel 4.3.10.2
Anweisungen: **OPEN**, **FIELD**, **PRINT#** , **PRINT#...USING**, **WRITE#**
Funktionen: **LOC**, **MKI$**, **MKS$**, **MKD$**, **LOF**
Fehler-Codes: 54, 61, 63

ANWEISUNG: **PUT %**

FUNKTION: Ein in einem Array gespeichertes Bild wird auf dem Bildschirm dargestellt

FORMAT:

```
PUT [% Window-Nr. ,] (X1,Y1)-(X2,Y2),Arrayelement [,Operand]
```

Window-Nr. : numerischer Ausdruck; Wert: 1 bis 16

X1,Y1 : numerische Ausdrücke, die den Anfangspunkt des Bereiches definieren

X2,Y2 : numerische Ausdrücke, die den Endpunkt des Bereiches definieren

Arrayelement: Bezeichnung des Elementes eines eindimensionalen Integer-Arrays, mit dem die Speicherung des Bildes beginnt

Operand : kann sein: **AND**, **OR**, **XOR**, **NOT**, **PSET**, **PRESET**

WIRKUNG: Ist der Parameter **%** Window-Nr. angegeben, so wird das entsprechende Window gewählt. Fehlt dieser Parameter, so bezieht sich die Anweisung auf das aktive Window.
Die Koordinatenpaare (X1,Y1) und (X2,Y2) definieren - bezogen auf die gültige **SCALE**-Anweisung - die diagonal liegenden Eckpunkte eines achsenparallelen Rechteckes.
Die in diesem Array gespeicherten Werte werden als Abbildung eines Bereiches der Bitmap (vgl. Kapitel 4.3.18) interpretiert und aus dem Array in den dem Rechteck entsprechenden Bereich der Bitmap übertragen.
Die Übertragung erfolgt in aufsteigender Reihenfolge der Arrayelemente in das Rechteck, beginnend von links oben nach rechts unten.
Der Parameter kann eines der Sprachelemente **AND**, **OR**, **XOR**, **NOT**, **PSET** oder **PRESET** sein.

Ist eines dieser Sprachelemente angegeben, so wird auf dem durch **PUT %** angesprochenen Bereich der Bitmap für jeden Elementarpunkt die betreffende logische Operation durchgeführt.
Wird **PSET** angegeben, so werden alle betroffenen Punkte gesetzt, bei **PRESET** werden alle betroffenen Punkte auf die Hintergrundfarbe gesetzt.

ANMERKUNGEN:

- Das angegebene Arrayelement muß so gewählt werden, daß es die Informationen über die Größe des im Array gespeicherten Bildes enthält. Es wird daher im allgemeinen dem im **GET %** angegebenen Arrayelement entsprechen.
- Ist der in **PUT %** angegebene Bereich größer als der im Array gespeicherte Bereich, so wird zeilengerecht nur die Anzahl von Elementarpunkten übernommen, die ab 'Arrayelement' gespeichert sind.
- Ist der in **PUT %** angegebene Bereich kleiner als der im Array gespeicherte Bereich, so werden die Grenzen des in **PUT %** angegebenen Bereiches eingehalten.
- Liegt das durch (X1,Y1) und (X2,Y2) definierte Rechteck nicht vollständig im angegebenen Window, so wird nur der im Window liegende Bereich berücksichtigt.
- Die Anzahl der Arrayelemente, die für die Übertragung in das Window nötig sind, ergibt sich aus der Anzahl der im Rechteck liegenden Elementarpunkte, wobei in jedem Arrayelement 16 Bit gespeichert sind.

BEISPIELE: s. **GET %**

VERWEISE :

Kapitel : 4.3.14 und 4.3.18
Anweisungen: **GET %, SCALE**
Funktion : **WINDOW**

ANWEISUNG: **RANDOMIZE**

FUNKTION: Festlegung bzw. Auswahl des Anfangswertes und der Folge von später aufzurufenden Zufallszahlen

FORMAT:

RANDOMIZE [num.Ausdruck]

WIRKUNG: Der num. Ausdruck wird berechnet. Mit seiner Hilfe wird die erste später aufgerufene Zufallszahl **RND** eindeutig bestimmt, ebenso wie eine daraus resultierende Folge von Zufallszahlen.

BEMERKUNGEN:

- 'num.Ausdruck' muß einen Integer-Wert zwischen -32768 und 32767 ergeben, sonst erfolgt die Fehlermeldung "Overflow".
- Wird 'num.Ausdruck' weggelassen, fordert der M 20 die Eingabe eines Wertes an. Meldung: "Random number seed (-32768 to 32767)?"
- Wird die **RANDOMIZE**-Anweisung nicht gegeben, wird bei jedem Aufruf des Programms der gleiche Anfangswert und die gleiche Folge von Zufallszahlen erzeugt, wenn die Funktion **RND** ohne Argument aufgerufen wird.
- Die **RANDOMIZE**-Anweisung muß in jedem Programm gegeben werden, das von der Standardfolge abweichende Zufallszahlen haben soll; allerdings kann dies auch durch Angabe eines Arguments beim Aufruf der **RND**-Funktion erreicht werden.

BEISPIELE:

```
10 REM *** Zufallszahlen mit RANDOMIZE ***
20 FOR J=1 TO 2

25 CLS :RANDOMIZE
30 FOR I=1 TO 10
40 LPRINT RND,
50 NEXT I,J
60 EXEC "sp" ' nur PR2400
```

VERWEISE: Kapitel 4.3.16
Funktion: **RND**

ANWEISUNG: **READ**

FUNKTION: Zuweisung von Werten aus dem internen File an Variablen; die Werte wurden vorher in einer internen Tabelle mit Hilfe von **DATA**-Anweisungen generiert

FORMAT:

READ { num. Var. / Stringvar. } [, { num. Var. / Stringvar. }] ...

WIRKUNG: Die Werte der internen Tabelle werden der Reihe nach (von links nach rechts) den Variablen in der **READ**-Anweisung zugewiesen. Mit dem Lesen der Tabelle wird an der durch den Pointer bezeichneten Stelle begonnen (siehe Anweisungen **DATA** und **RESTORE**).

Nach jeder Zuweisung einer Konstanten an die entsprechende Variable wird der interne Pointer um eine Position (nächste Konstante) verschoben. Der Pointer kann mit der Anweisung **RESTORE** auf bestimmte Elemente der Tabelle zurückgesetzt werden.

BEMERKUNGEN:

- **READ** kann nur dann verwendet werden, wenn im Programm mindestens eine **DATA**-Anweisung vorkommt.
- In der **READ**-Anweisung werden die gelesenen Daten den Variablen in der Reihenfolge von links nach rechts zugewiesen. Das System verhält sich genau so, als stünde für jede Variable eine eigene **READ**-Anweisung.
- Stehen in der **DATA**-Anweisung mehr Konstante als mit dem **READ**-Statement zu lesen sind, so wird die erste nicht gelesene Konstante der **DATA**-Anweisung der ersten Variablen der nächsten **READ**-Anweisung zugewiesen.
- In einer **READ**-Anweisung dürfen nur so viele Werte verlangt werden, wie in der internen Tabelle noch vorhanden sind. Ansonsten wird die Meldung "Out of data" angezeigt.
- Die Variablen-Typen (String, numerisch) der **READ**-Anweisung müssen mit den Konstanten-Typen der **DATA**-Anweisung übereinstimmen. Andernfalls wird die Meldung "Syntax error" angezeigt.

- Es können sich sowohl mehrere **READ**-Anweisungen auf eine **DATA**-Anweisung als auch eine **READ**-Anweisung auf mehrere **DATA**-Anweisungen beziehen.

BEISPIELE:

```
10 DATA# 2,475,-4,66,34,3e34,6663,8
20 DATA text1,text2,100,texte
30 REM
40 READ WERT1,WERT2,WERT3,WERT4,WERT5
50 LPRINT WERT1,WERT2,WERT3,WERT4,WERT5
60 READ W6$,W7$,W8,W9$
70 LPRINT W6$,W7$,W8,W9$
80 RESTORE 20
90 READ T$,S$,X,X$
100 LPRINT T$,S$,X,X$
110 END
 2             475           -4,66          3,43E+35
 6663,8
text1         text2          100           texte
text1         text2          100           texte
```

VERWEISE:

Kapitel 4.3.8
Anweisungen: **DATA, RESTORE**
Fehler-Code: 4

ANWEISUNG: **REM**
(remark)

FUNKTION: Einfügung von Erläuterungen in ein Programm

FORMAT:

REM [Kommentar]

Kommentar: beliebiger Text

WIRKUNG: **REM** ist eine Anweisung, die nicht ausgeführt wird. Der Kommentar erscheint im Ausdruck (Listing), ist jedoch für die Programmausführung bedeutungslos.

BEMERKUNGEN:

- Es können auch Kommentare am Ende einer BASIC-Zeile angefügt werden. Dabei ist die Anweisung **REM** durch : von der letzten Anweisung zu trennen.
- Anstelle des Schlüsselwortes **REM** kann das Zeichen ' (Hochkomma) gesetzt werden. Dieses wird in Listings und beim Speichern des Programms nicht durch **REM** ersetzt. Dabei kann sogar ein normalerweise nötiges Zeichen : vorher entfallen.
- Eine BASIC-Zeile mit einer **REM**-Anweisung kann als Sprungziel verwendet werden.
  Das Programm wird mit derjenigen BASIC-Zeile fortgesetzt, die der **REM**-Anweisung folgt.
- Achtung: Befindet sich eine **REM**-Anweisung oder ein '(= Hochkomma) inmitten einer BASIC-Zeile mit durch : verbundenen Anweisungen, so werden alle Zeichen, die **REM** oder ' folgen, als Kommentar interpretiert, also auch nachfolgende Anweisungen innerhalb der Zeile!

BEISPIELE:

```
10 REM *** Diese Textzeile ist nur ein Hinweis fuer den Pro-
20 REM     grammierer und hat keinen Einfluss auf den Pro-
30 REM     grammablauf
40 ' Das Zeichen ' entspricht einer REM-Anweisung
50 END
```

VERWEISE: Kapitel 4.3.1

BEFEHL: **RENUM**

FUNKTION: Umnumerierung der Zeilen eines im Arbeitsspeicher vorhandenen Programms

FORMAT:

RENUM [neue Zeilennummer], [alte Zeilennummer] [,Schrittweite]

neue Zeilennummer: positive ganze Zahl; bestimmt die Zeilennummer, die der ersten umzunumerierenden Programmzeile zugeordnet wird

alte Zeilennummer: positive ganze Zahl, bestimmt die Zeilennummer, ab der umnumeriert wird

Schrittweite: positive ganze Zahl, welche die jeweilige Erhöhung der Numerierung bestimmt

WIRKUNG: Die im Arbeitsspeicher befindlichen Programmzeilen werden umnumeriert. Der ersten Programmzeile wird die mit 'neue Zeilennummer' definierte Zeilennummer zugeordnet. Jede folgende Zeile erhält eine jeweils um die 'Schrittweite' erhöhte Zeilennummer. Default-Wert für 'neue Zeilennummer' und 'Schrittweite' ist jeweils 10. Ist der Operand 'alte Zeilennummer' angegeben, so beginnt die Umnumerierung nicht mit der ersten Programmzeile, sondern mit der durch 'alte Zeilennummer' definierten Zeile.

BEMERKUNGEN:

- Wird bei der Umnumerierung eine Zeilennummer größer als 65528 erzeugt, so erfolgt die Meldung "Illegal function call".
- Ist 'neue Zeilennummer' kleiner als 'alte Zeilennummer' und existieren Programmzeilen, die eine kleinere Zeilennummer als 'alte Zeilennummer' besitzen, so erfolgt die Meldung "Illegal function call".
- Bei der Umnumerierung eines Programms modifiziert das System automatisch auch alle innerhalb von BASIC-Anweisungen vorkommenden Hinweise auf Zeilen (z.B. **GOTO**, **IF...THEN**, **GOSUB**, **ERL**)

BEISPIELE:

1.
```
10 REM  1. REM
20 IF INKEY$="" THEN 10 ELSE 30
30 REM  2. REM
Ok
renum 200,,2
```

2
```
200 REM  1. REM
202 IF INKEY$="" THEN 200 ELSE 204
204 REM  2. REM
Ok
renum 500,202,100
```

3.
```
200 REM  1. REM
500 IF INKEY$="" THEN 200 ELSE 600
600 REM  2. REM
Ok
```

VERWEISE: Kapitel 3.3

ANWEISUNG: **RESTORE**

FUNKTION: Setzen des Pointers des internen Files auf das erste Element einer **DATA**-Anweisung

FORMAT:

RESTORE [Zeilennummer]

WIRKUNG: Wird keine Zeilennummer angegeben, so wird der Pointer des internen Files auf das erste Element der **DATA**-Anweisung mit der niedrigsten Zeilennummer gesetzt.

Wird eine Zeilennummer angegeben, so wird der Pointer auf das erste Element derjenigen **DATA**-Anweisung gesetzt, die in der genannten Zeile steht.

BEMERKUNGEN: Siehe Anweisungen **DATA** und **READ**

BEISPIELE:

```
10 DATA 2,475,-4.66,34.3e34,6663.8
20 DATA text1,text2,100,texte
40 READ WERT1,WERT2,WERT3,WERT4,WERT5
50 LPRINT WERT1,WERT2,WERT3,WERT4,WERT5
60 READ W6$,W7$,W8,W9$
70 LPRINT W6$,W7$,W8,W9$
80 RESTORE 20
90 READ T$,S$,X,X$
100 LPRINT T$,S$,X,X$
 2             475          -4.66          3.43E+35       6663.8
text1         text2          100           texte
text1         text2          100           texte
```

VERWEISE: Kapitel 4.3.8
Anweisungen: **DATA**, **READ**

ANWEISUNG: **RESUME**

FUNKTION: Fortsetzung eines Programms nach einer Verzweigung in eine Fehlerbehandlungsroutine

FORMAT:

```
RESUME [[ { Ø      } ]]
       [[ { NEXT   } ]]
       [[ { Zeilennr. } ]]
```

Zeilennr.: Zeilennr. einer Anweisung(sfolge); darf nicht in Form einer Variablen definiert sein

WIRKUNG: Stößt das System innerhalb einer Fehlerbehandlungsroutine (vgl. **ON ERROR GOTO**) auf die **RESUME**-Anweisung, wird aus der Fehlerbehandlungsroutine zurück zum Hauptprogramm verzweigt.

Bei Angabe der **Ø** oder keines Parameters wird zu derjenigen Anweisung rückverzweigt, die den Sprung in die Fehlerbehandlungsroutine verursachte. Bei Angabe des Parameters **NEXT** wird zu derjenigen Anweisung verzweigt, die auf die Anweisung unmittelbar folgt, die den Sprung in die Fehlerbehandlungsroutine verursachte.
Bei Angabe von 'Zeilennr.' wird zu der mit 'Zeilennr.' bezeichneten Anweisungszeile verzweigt.

BEMERKUNGEN:
- Eine **RESUME**-Anweisung ohne vorherige Verzweigung in die Fehlerbehandlungsroutine führt zur Fehlermeldung "RESUME without error".
- Die reservierte Variablen **ERR** enthält nach der **RESUME**-Anweisung den Wert **Ø**.

- Die reservierte Variable **ERL** bleibt von **RESUME** unberührt.

BEISPIELE:

```
10 REM'Flaechenberechnung im Dreieck
20 A$="_Seite a=###.##  _Seite b=###.##  _Seite c=###.##  Fl
aeche=#####.######"
30 ON ERROR GOTO 100
40 INPUT"Seite a ,Seite b ,Seite c";A,B,C
50 IF (A<0) OR (B<0) OR (C<0) THEN ERROR 200
60 S=(A+B+C)/2
70 F2=S*(S-A)*(S-B)*(S-C)
80 IF F2<=0 THEN ERROR 210 ELSE LPRINT USING A$;A,B,C,SQR(F2
)
90 GOTO 40
100 IF ERR=200 AND ERL=50 THEN LPRINT "a oder b oder c klein
er 0":RESUME 40
110 IF ERR=210 AND ERL=80 THEN LPRINT "kein Dreieck !!!":RES
UME 40
120 ON ERROR GOTO 0
130 END
Seite a= 15,00  Seite b= 20,00  Seite c=  8,00  Flaeche=   5
3,197200
a oder b oder c kleiner 0
kein Dreieck !!!
```

VERWEISE:

Kapitel 4.3.12
Anweisungen: **ON ERROR GOTO, ERROR**
reservierte Variablen: **ERR, ERL**
Fehler-Code: 20

ANWEISUNG: **RETURN**

FUNKTION: **RETURN** bildet das logische Ende eines Unterprogrammes und bewirkt den Rücksprung zu der unmittelbar auf das aufrufende **GOSUB** oder **ON...GOSUB** folgenden Anweisung

FORMAT:

```
RETURN
```

WIRKUNG: Siehe Anweisungen **GOSUB** und **ON..GOSUB**

BEISPIELE:

```
1000 GOSUB 2000
1010 LPRINT "RETURN erfolgt" : END
2000 LPRINT "GOSUB 2000 abgearbeitet": RETURN
GOSUB 2000 abgearbeitet
RETURN erfolgt
```

VERWEISE: Kapitel 4.3.5 und 4.3.7
Anweisungen: **GOSUB**, **ON...GOSUB**, **CLEAR**
Fehler-Code: 3

FUNKTION: **RIGHT$**

ZWECK: Einem String wird von rechts kommend ein Teilstring entnommen

FORMAT:

RIGHT$(Ausgangsstring, Anzahl Zeichen)

Ausgangsstring: Stringausdruck

Anzahl Zeichen: num. Ausdruck;
bestimmt die Anzahl Zeichen des Teilstrings

WIRKUNG: Die 'Anzahl Zeichen' wird berechnet und kaufmännisch zur Ganzzahligkeit gerundet. Aus dem 'Ausgangsstring' wird von rechts kommend die berechnete Anzahl von Zeichen entnommen.

BEMERKUNGEN:

- Ist die 'Anzahl Zeichen' größer als die Länge des 'Ausgangsstrings', so wird der gesamte Ausgangsstring übergeben.
- Ist die 'Anzahl Zeichen' = Ø, so wird kein Zeichen entnommen und der Teilstring ist der Leerstring.
- 'Anzahl Zeichen' muß einen Wert zwischen Ø und 255 aufweisen.

BEISPIELE:

```
10 A$= "Personalcomputer M 20"
20 LPRINT A$, RIGHT$(A$,4)
30 END
Personalcomputer M 20        M 20
```

VERWEISE: Kapitel 4.3.4
Funktionen: **LEFT$, MID$**

FUNKTION: **RND**
(random number)

ZWECK: liefert eine Zufallszahl zwischen Ø und 1

FORMAT: **RND** [(num.Ausdr.)]

WIRKUNG: Der numerische Ausdruck wird berechnet und auf Ganzzahligkeit gerundet.
In Abhängigkeit von seinem Wert wird eine Zufallszahl zwischen Ø und 1 (einfach genau) als Ergebnis der Funktion entwickelt. Ist 'num.Ausdr.' größer als Ø, wird der nächste Wert einer Folge von Zufallszahlen bestimmt, deren erster Wert von 'num.Ausdr.' abhängt.
Ist 'num.Ausdr.' kleiner als Ø, wird immer wieder die gleiche vom 'numerischen Ausdruck' abhängige Zufallszahl erzeugt.
Ist 'num.Ausdruck'=Ø, wird stets Ø erzeugt.
Ist 'num.Ausdr.' nicht angegeben, wird die nächste Zahl der Standardfolge von Zufallszahlen erzeugt.

BEMERKUNGEN:
- Bei Weglassen von 'num.Ausdr.' kann die Standardfolge durch eine vorher erfolgte **RANDOMIZE**-Anweisung geändert werden.
- Die Zufallszahl ist von einfacher Genauigkeit.

BEISPIELE:

```
Ø FOR J%=1 TO 2
1 FOR I%=1 TO 4 : LPRINT RND;
20 NEXT I% : LPRINT : NEXT J%
30 REM 2 x run
 .082254   .255843   .776169   .682012
 7.32367E-02   .378558   .956858   .372086
 .082254   .255843   .776169   .682012
 7.32367E-02   .378558   .956858   .372086
```

VERWEISE: Kapitel 4.3.16
Anweisung: **RANDOMIZE**

ANWEISUNG: **RSET**
(right set)

FUNKTION: Zuweisung eines rechtsbündigen, links eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable

FORMAT:

RSET { Feldvar. / Stringvar. } =Stringausdr.

WIRKUNG: Der Stringausdruck wird berechnet. Das Ergebnis wird rechtsbündig in das durch 'Feldvariable' definierte Feld (vgl. **FIELD**-Anweisung) eines Random-File-Puffers gebracht, wobei Feldreste automatisch mit Blanks gefüllt werden.

Bei einer **RSET**-Zuweisung auf eine normale Stringvariable wird deren alter Inhalt rechtbündig durch das Ergebnis von 'Stringausdruck' ersetzt. Diejenigen Zeichen von 'Stringvariable', die bei der rechtsbündigen Ersetzung nicht betroffen werden, werden durch Blanks ersetzt.

BEMERKUNGEN:
- Hat das Ergebnis von 'Stringausdruck' mehr Zeichen als 'Feldvar.' bzw. 'Stringvar.', werden die überschüssigen Zeichen rechts abgeschnitten.
- Bevor numerische Werte in den Random-File-Puffer gebracht werden, müssen sie -je nach Typ- mit Hilfe der Funktionen **MKI$**, **MKS$** oder **MKD$** in Strings konvertiert werden.

BEISPIELE:

```
10 A$='                                                  "
20 B$="                                                  "
30 LSET A$="olivetti M20":RSET B$="olivetti M20"
80 LPRINT ">";A$;"<":LPRINT ">";B$"<"
>olivetti M20                                            <
>                                  olivetti M20<
```

VERWEISE: Kapitel 4.3.10.2
Anweisungen: **FIELD**, **PUT**, **LSET**
Funktionen: **MKI$**, **MKS$**, **MKD$**

BEFEHL: **RUN**

FUNKTION: Start der Programmausführung

FORMAT:

```
RUN [{Zeilennummer [,R]}]
    [{Filename     [,R]}]
```

Zeilennummer: Programmzeile, mit welcher die Verarbeitung beginnen soll; kann nicht Variable sein

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Sind keine Operanden angegeben, so wird das sich im Arbeitsspeicher befindliche Programm, beginnend mit der ersten Programmzeile, abgearbeitet. Bei Angabe des Operanden 'Zeilennummer' beginnt die Ausführung des Programmes mit der durch 'Zeilennummer' definierte Programmzeile. Die Ausführung des Befehls **RUN** löscht den gesamten Datenbereich und schließt alle offenen Datenfiles.
Ist der Operand 'Filename' angegeben, so wird das Programm auf der entsprechenden Diskette gesucht, in den Arbeitsspeicher geladen und abgearbeitet. Der Operand **R** gibt an, daß alle bereits geöffneten Datenfiles offen bleiben sollen.

BEMERKUNGEN:

- Wird **RUN** mit 'Zeilennummer' angegeben, so darf sich die 'Zeilennummer' nicht auf eine Anweisung innerhalb einer **FOR-NEXT**-Schleife (Meldung: "NEXT without FOR"), innerhalb einer **WHILE-WEND**-Schleife (Meldung: "WEND without WHILE"), oder innerhalb eines Unterprogrammes (Meldung: "RETURN without GOSUB") beziehen.

Achtung:

Bei **RUN** mit 'Zeilennummer' wird der Datenbereich gelöscht; daraus folgt daß das Programm ab 'Zeilennummer' für alle Variablen, die ohne vorherige Zuweisung aufgerufen werden, die Default-Werte ansetzt (num. Variablen: Ø; String-Variablen: "").

- Wird **RUN** mit 'Zeilennummer' angegeben und existiert die Zeilennummer nicht, so erscheint die Fehler-Meldung "Undefined line number".
- **RUN** Zeilennummer hat die gleiche Funktion wie **GOTO** Zeilennummer, jedoch wird bei **GOTO** Zeilennummer der Datenbereich nicht gelöscht.
- **RUN** Filename,R hat die gleiche Bedeutung wie **LOAD** Filename,R.
- Die Programmausführung wird beendet, wenn

  1.) die Anweisung **END** erkannt wird. Alle offenen Files werden geschlossen. Programmvariablen können angezeigt werden.
  2.) die Anweisung **STOP** erkannt wird, die Taste C gedrückt wird oder eine Fehlermeldung auftritt (Ausnahme sind Syntax-Fehler). Das System befindet sich dann im Direkt-Mode. Offene Files bleiben offen, Programmvariablen können angezeigt und/oder geändert werden. Mit dem Befehl **CONT** kann die Ausführung des Programmes fortgesetzt werden.
  3.) die Taste S gedrückt wird. Das System befindet sich nicht im Direkt-Mode. Offene Files bleiben offen. Programmvariablen können nicht angezeigt werden. Durch Drücken irgendeiner Taste wird die Ausführung des Programmes fortgesetzt.

BEISPIELE:

```
run "1:prog1"
run
run "diskrechts/P145:TEXTVERARB.prg",R
```

VERWEISE:

Kapitel 3
Befehl: **CONT**
Anweisungen: **END**, **GOTO**, **STOP**, **CHAIN**

BEFEHL: **SAVE**

FUNKTION: Abspeicherung eines BASIC-Programmes oder eines mit Zeilennummern versehenen Textes (ASCII-Files) auf Diskette

FORMAT:

SAVE Filename [, {A / P}]

Filename: Stringausdruck; Ergebnis muß einen Begriff ergeben, der den Regeln zur Bildung von Filenamen entspricht (vgl. Kapitel 4.1.3)

WIRKUNG: Das sich im Arbeitsspeicher befindliche Programm wird auf Diskette gespeichert. Das Programm wird in einer gepackten Binärform abgespeichert, wenn der Operand **A** nicht angegeben ist. Enthält der Operand 'Filename' keine Stationsbezeichnung, so wird das Programm auf die Diskette in der momentan aktiven (zuletzt angesprochenen) Station gespeichert. Bei der Abspeicherung kann dem Programm ein File-Password zugeordnet werden (siehe Kapitel 4.1.3). Befindet sich auf der Diskette bereits ein File mit dem durch 'Filename' bestimmten Namen, so wird dieses File durch das im Arbeitsspeicher befindliche Programm ersetzt. Wurde diesem File ein File-Password gegeben, so muß das File-Password bei **SAVE** mit angegeben werden.
Ist der Operand **A** angegeben, so wird das Programm in ASCII-Format auf der Diskette gespeichert. ASCII-Format benötigt mehr Platz auf der Diskette als die gepackte Binärform. Es gibt jedoch Befehle, die ein File in ASCII-Format erfordern (z.B. **MERGE**).
Der Operand **P** bestimmt, daß das Programm in gepackter Binärform abgespeichert und gegen folgende Befehle geschützt werden soll:

- **LIST**
- **EDIT**
- **SAVE** (ersetzen)

BEMERKUNGEN:
- Die Diskette darf nicht mit einem Schreibschutz versehen sein.
- Hat die Diskette ein Disketten-Password, so ist dieses bei der Abspeicherung anzugeben.

- Das System prüft, ob das abzuspeichernde Programm noch auf der Diskette Platz hat. Sollte dies nicht der Fall sein, so erscheint die Meldung "Disk full".
- Mit PCOS-Befehl **ss** wird festgelegt, wie groß die Anzahl zu reservierender Sektoren für ein abzuspeicherndes Programm sein soll. Ist diese Anzahl größer als die Anzahl freier Sektoren auf der Diskette, so erscheint bei dem Befehl **SAVE** ebenfalls die Meldung "Disk full".
- Die Operanden **A** und **P** können nicht gleichzeitig angegeben werden, d.h., ein ASCII-File kann nicht geschützt werden.

BEISPIELE:

```
save "1:SACHKTEN/FPASS"
save "0:traeger"
save "1:BWAModul",A
save "1:mitschutz/fpa546",P
save "sichdisk/dpass:sichfile/fpass",A
```

VERWEISE: Kapitel 3 und 4.3.13

ANWEISUNG: **SCALE**

ZWECK: Festlegung des Koordinationssystems innerhalb eines Windows

FORMAT:

```
SCALE [%Window-Nr.,] X-min, X-max, Y-min, Y-max
```

Window-Nr.: numerischer Ausdruck, dessen Ergebnis gerundet wird und dann einen Wert zwischen 1 und 16 enthalten muß

X-min:
X-max:
Y-min:
Y-max: numerische Ausdrücke, die Grenzen problembezogener Werte bilden

WIRKUNG: - Ist der Parameter % Window-Nr. angegeben, so bezieht sich die Anweisung auf dieses Window. Fehlt der Parameter, so bezieht sich die Anweisung auf das aktive Window.
'X-min' und 'X-max' sowie 'Y-min' und 'Y-max' stellen die Grenzwerte der Anwender-Koordinaten dar, die im angesprochenen Window gültig sind.
Dabei werden der linken unteren Ecke des Windows die Koordinaten (X-min/Y-min) zugewiesen. Die rechte obere Ecke des Windows erhält die Koordinaten (X-max/Y-max) zugeordnet:
Daraus ergibt sich:

$$X\text{-min} < X\text{-max}$$
$$Y\text{-min} < Y\text{-max}$$

Die positive Richtung der X-Achse ist somit der horizontalen Richtung von links nach rechts zugeordnet, die positive Richtung der Y-Achse entspricht der vertikalen Richtung von unten nach oben. Aus der Anzahl der Elementarpunkte, die in jede Richtung des Windows enthalten sind, wird durch den gültigen Wertebereich der Abbildungsmaßstab für später folgende graphische Angaben bestimmt.

BEMERKUNGEN:

- Werden Koordinatenpunkte, die außerhalb des durch **SCALE** festgelegten Bereiches liegen, angesprochen, so hat das für die meisten graphischen Anweisungen und Funktionen keine Wirkung. Für die imaginäre Darstellung außerhalb des Bereichs wird jedoch Rechenzeit benötigt (unnötigerweise).
- Ist 'X-max' kleiner oder gleich 'X-min' bzw. 'Y-max' kleiner oder gleich 'Y-min', so wird "Division by zero" gemeldet und weitergearbeitet.
- Der **RUN**-Befehl hebt eine im Vorprogramm erfolgte **SCALE**-Anweisung nicht auf. Dies ist nur durch neue **SCALE**-Anweisung oder durch die **CLEAR**-Anweisung möglich.
- Wird ein Programm durch **CHAIN** geladen, so bleibt die **SCALE**-Definition des alten Programmes für das neue gültig, es sei denn, dort wird eine neue **SCALE**-Anweisung verwendet.
- Die Default-Werte für **SCALE** betragen:
  bei 64x16-Zeichen-Format des Bildschirms
  (mit **ss** gesetzt!):
  X-min= Ø
  X-max= 511
  Y-min= Ø
  Y-max= 255

  bei 80x25-Zeichen-Format(mit **ss** gesetzt!):
  X-min: Ø
  X-max: 479
  Y-min: Ø
  Y-max: 255
- Für den Maßstab gilt:
  ((X-max - X-min)+1)/ (Anzahl Elementarpunkte in X-Richtung)
  ergibt die Anzahl von problembezogenen Einheiten, die auf einen Elementarpunkt abgebildet werden; analog für die Y-Richtung.

- Da der Abstand der Elementarpunkte horizontal und vertikal nicht gleich ist, muß zur Erreichung eines gleichen Maßstabes in X und Y, bezogen auf den Gesamtbildschirm, gelten:
  dx : dy = 1.575 : 1 bei 64x16-Zeichen-Schirm.
  dx : dy = 1.504 : 1 bei 80x25-Zeichen-Schirm.
  Diese Verhältnisse gelten nur, wenn das Bildschirmformat durch **ss** festgelegt wurde.

- Wird mit Hilfe der Funktion **WINDOW** der Gesamtschirm im Format geändert, so wird zwar die dem Format entsprechende Zeichenfläche zugrundegelegt; die horizontale Aufteilung in Elementarpunkten (Anzahl) bleibt jedoch die in **ss** festgelegte.
  Deshalb empfiehlt sich, stets eine **Skalierung** vorzunehmen. Ist diese nicht in Anwenderkoordinaten, so sollte das Format des Gesamtschirms über die **WINDOW**-Funktion definiert und danach folgende Skalierung gewählt werden:

Spaltenbreite	**SCALE**
8	0,511,0,255
6	0,479,0,255

BEISPIELE:

```
1 CLEAR
5 CLS
10 W=WINDOW(0,0,16,8)
20 LINE (0,0)-(479,255),,B
25 SCALE 0,511,0,255

30 LINE (0,0)-(511,255), B
35 A$=INKEY$: IF A$="" THEN 35
40 SCALE 0,80,0,25
45 LINE (0,0)-(80,25),,B
50 GOTO 50
```

PCOS-Befehl
ss :
80x25

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **CLEAR**
Funktionen: **WINDOW**, **SCALEX**, **SCALEY**

FUNKTION: **SCALEX**

ZWECK: gibt an, der wievielte Elementarpunkt des aktiven Windows durch eine Anwender-bezogene X-Koordinate angesprochen wird

FORMAT:

```
SCALEX(num. Ausdruck)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als X-Koordinate, bezogen auf das im aktiven Window gültige **SCALE**, interpretiert. Es wird ermittelt, der wievielte Elementarpunkt des aktiven Window in X-Richtung, ab zählend von Ø (= linker Window-Rand), dadurch angesprochen wird.

BEMERKUNGEN:

- Liegt das Argument außerhalb des durch **SCALE** festgelegten X-Intervalls, wird ein imaginärer Punkt ermittelt.
- Das Ergebnis der Funktion muß zwischen -32755 und 32755 liegen, sonst wird "Overflow" gemeldet.
- Eine X-Koordinate auf dem linken Window-Rand liefert den **SCALE**-Wert Ø; der maximale Wert für den rechten Window-Rand ist abhängig vom Bildschirm-Format:

Bildschirm-Format	maximaler **SCALEX**-Wert (nur am Gesamtschirm!)
64 x 16	511
80 x 25	479

- Die Funktion ist besonders bei Anwendung der Assembler-Routine **la** wichtig, da die Anfangsposition des auszugebenden Schriftzuges stets in Elementar-Positionen anzugeben sind.

BEISPIELE:

```
10 CLS
20 SCALE 0,5110,0,3180
30 CIRCLE (2555,1590),1000
40 PIXELX=SCALEX(2555)
50 PIXELY=SCALEY(1590)
60 PRINT" Die X-Koordinate 2555 entspricht Pixel ";PIXELX
70 PRINT" Die Y-Koordinate 1590 entspricht Pixel ";PIXELY
80 CALL "sp"
90 END
 Die X-Koordinate 2555 entspricht Pixel  255

 Die Y-Koordinate 1590 entspricht Pixel  127
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **SCALE**
Funktion: **SCALEY**
Assembler-Routine: **la**

FUNKTION: **SCALEY**

ZWECK: gibt an, der wievielte Elementarpunkt des aktiven Windows durch eine Anwender-bezogene Y-Koordinate angesprochen wird

FORMAT:

SCALEY(num. Ausdruck)

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als Y-Koordinate, bezogen auf das im aktiven Window gültige **SCALE**, interpretiert. Es wird ermittelt, der wievielte Elementarpunkt des aktiven Windows in Y-Richtung, abzählend von Ø (= unterer Window-Rand), dadurch angesprochen wird.

BEMERKUNGN:

- Liegt das Argument außerhalb des durch **SCALE** festgelegten Y-Intervalls, wird ein imaginärer Punkt ermittelt.
- Das Ergebnis der Funktion muß zwischen -32755 und 32755 liegen, sonst wird "Overflow" gemeldet.
- Eine Y-Koordinate auf dem unteren Window-Rand liefert den **SCALEY**-Wert Ø; der maximale Wert für den oberen Window-Rand ist, unabhängig vom Bildschirmformat, 255 (nur am Gesamtschirm!).
- Die Funktion ist besonders bei Anwendung der Assembler-Routine **la** wichtig, da die Anfangsposition des auszugebenden Schriftzuges stets in Elementarpunkt-Positionen anzugeben ist.

BEISPIELE:

```
10 CLS
20 SCALE 0,5110,0,3180
30 CIRCLE (2555,1590),1000
40 PIXELX=SCALEX(2555)
50 PIXELY=SCALEY(1590)
60 PRINT" Die X-Koordinate 2555 entspricht Pixel ";PIXELX
70 PRINT" Die Y-Koordinate 1590 entspricht Pixel ";PIXELY
80 CALL "sp"
90 END
 Die X-Koordinate 2555 entspricht Pixel  255
 Die Y-Koordinate 1590 entspricht Pixel  127
```

VERWEISE: Kapitel 4.3.14 und 4.3.18
Anweisung: **SCALE**
Funktion: **SCALEX**
Assembler-Routine: **la**

FUNKTION: **SGN**
(signum)

ZWECK: gibt Aufschluß über das Vorzeichen eines numerischen Wertes ("Signum-Funktion")

FORMAT:

```
SGN(num.Ausdruck)
```

WIRKUNG: Der numerische Ausdruck wird berechnet.
Ist er positiv, liefert **SGN** als Ergebnis +1.
Ist er negativ, liefert **SGN** als Ergebnis -1.
Ist er genau Ø, liefert **SGN** als Ergebnis Ø.

BEISPIELE:

```
10 DATA -5,.0001,-60D35,0,23D-120
20 FOR I%=1 TO 5 : READ A#(I%)
25 LPRINT "WERT =";A#(I%);TAB(20);
26 LPRINT "SGN(WERT) =";SGN(A#(I%)) : NEXT
WERT =-5              SGN(WERT) =-1
WERT = .0001          SGN(WERT) = 1
WERT =-6D+36          SGN(WERT) =-1
WERT = 0              SGN(WERT) = 0
WERT = 2.3D-119       SGN(WERT) = 1
```

VERWEISE: Kapitel 4.3.16

FUNKTION: **SIN**

ZWECK: liefert den Sinus eines numerischen Wertes im Bogenmaß

FORMAT:

```
SIN(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als Bogenmaß interpretiert und daraus wird der Sinus berechnet.

BEMERKUNGEN:

- Das Argument ist im Bogenmaß anzugeben. Umrechnungen müßten über Unterprogramm oder eine selbstdefinierte Funktion (vgl. **DEF FN**) erfolgen.
- Das Argument muß Intervall $-65535.998 \leq x \leq 65535.998$ liegen, sonst erfolgt die Fehlermeldung "Overflow".
- Der Arcussinus läßt sich berechnen mit Hilfe der Formel:
  ARCSIN(X)=**ATN(X/SQR(-X*X+1))**

BEISPIELE:

```
10 LPRINT"Winkel  W=50 Neugrad"
20 LPRINT"Winkel  W=";50*3.14/200;" im Bogenmass"
30 LPRINT"Sinus   W=";SIN(50*3.14/200)
40 END
Winkel  W=50 Neugrad
Winkel  W=  .785   im Bogenmass
Sinus   W=  .706825
```

VERWEISE: Kapitel 4.3.16

FUNKTION: **SPACE$**

ZWECK: liefert einen String aus Blanks

FORMAT:

SPACE$(num.Ausdr.)

WIRKUNG: Der numerische Ausdruck wird berechnet und zu einer Integer-Zahl kaufmännisch gerundet. Als Ergebnis liefert die Funktion einen String aus soviel Blanks wie durch die Integer-Zahl angegeben.

BEMERKUNGEN:

- Der 'numerische Ausdruck' muß zwischen Ø und 255 liegen.

BEISPIELE:

```
10 P=11
20 LPRINT">"+SPACE$(P/2)+"<"
30 LPRINT" 123456"
40 END
>      <
 123456
```

VERWEISE: Kapitel 4.3.4 und 4.3.9
Funktion: **SPC**

FUNKTION: **SPC**

ZWECK: Ausgabe einer bestimmten Anzahl von Leerzeichen (Blanks) auf dem Bildschirm oder auf dem Drucker

FORMAT:

```
SPC(num.Ausdruck)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Es werden ab der aktuellen Spalten-Position des Bildschirm-Cursors oder des Druckkopfes soviele Leerzeichen (Blanks) ausgegeben wie der numerische Ausdruck angibt.

BEMERKUNGEN:

- Die **SPC**-Funktion darf nur innerhalb der Liste von Ausgabe-Elementen bei **PRINT**- oder **LPRINT**-Anweisungen eingesetzt werden.
- Das Argument muß zwischen Ø und 255 liegen.
- Es wird höchstens eine Bildschirmzeile mit Blanks erzeugt!

BEISPIELE:

```
10 A=4.5683
20 LPRINT A SPC(5) A^2 SPC(5) A/2
30 END
 4.5683          20.8694          2.28415
```

VERWEISE: Kapitel 4.3.9.1 und 4.3.9.2
Anweisungen: **PRINT**, **LPRINT**

FUNKTION: **SQR**
(square root)

ZWECK: liefert die positive Quadratwurzel eines numerischen Wertes

FORMAT:

```
SQR(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Daraus wird die positive Quadratwurzel gezogen.

BEMERKUNGEN:
- Das Argument muß größer oder gleich Ø sein, sonst wird die Fehlermeldung "Illegal function call" gegeben.
- Das Ergebnis hat die Genauigkeit des Arguments.

BEISPIELE:

```
10 LPRINT SQR(23.465)^SQR(2.5^2)
 4 84407  2.5
```

VERWEISE: Kapitel 4.3.16

ANWEISUNG: **STOP**

FUNKTION: Unterbrechen der Programmausführung

FORMAT:

```
STOP
```

WIRKUNG: Der Programmablauf wird unterbrochen. Die Meldung "Break in line nnnnn" wird ausgegeben, worin nnnnn die Nummer der die **STOP**-Anweisung enthaltenden Zeile angibt.

Das System geht in den Direkt-Mode (siehe Kapitel 3).

BEMERKUNGEN:
- Die **STOP**-Anweisung bewirkt kein Schliessen der Files.
- Mit dem **CONT**-Befehl kann die Programmausführung fortgesetzt werden.

ACHTUNG: Nach Übergang in den Direkt-Mode und Änderung oder Neuaufnahme einer Programmzeile ist der gesamte Datenbereich gelöscht!

BEISPIELE:

```
10 LPRINT "das Programm wird mit 'STOP' unterbrochen"
20 LPRINT "zur Fortsetzung des Programmes ist 'CONT' ei
nzugeben"
30 LPRINT
40 STOP
50 LPRINT "das Programm wird mit Zeile 50 fortgesetzt"
60 END
das Programm wird mit 'STOP' unterbrochen
zur Fortsetzung des Programmes ist 'CONT' einzugeben

das Programm wird mit Zeile 50 fortgesetzt
```

VERWEISE: Kapitel 3 und 4.3.5

FUNKTION: **STR$**

ZWECK: Bildung eines Strings aus einem numerischen Ausdruck

FORMAT:

```
STR$(num.Ausdruck)
```

WIRKUNG: Der numerische Wert wird berechnet und in einen String umgewandelt. Das erste Zeichen des Strings stellt das Vorzeichen des numerischen Ausdrucks dar.

BEMERKUNGEN:

- Die Funktion **STR$** ist die logische Umkehrung der Funktion **VAL**.
- Ist das Vorzeichen der Mantisse positiv, wird an der betreffenden Position ein Blank gesetzt.
- Bei Gleitkommadarstellung wird das Vorzeichen des Exponenten ausgegeben. Bei + wird + ausgegeben, und der Exponent wird mit zwei Stellen angegeben, wenn er eine oder zwei Stellen lang ist.
- ACHTUNG: Im Gegensatz zur Darstellung der numerischen Werte im Standardformat (vgl. **PRINT**-Anweisung und Kapitel 4.3.9) wird bei der Umwandlung des numerischen Wertes mit **STR$** nach der letzten signifikanten Stelle des numerischen Werts kein Blank erzeugt! **STR$**(0.7300100) liefert den String " .73001".

BEISPIELE:

```
10 A$=STR$(-3.5729)
20 LPRINT A$,LEN(A$)
30 END
-3.5729         7
```

VERWEISE: Kapitel 4.3.4.3
Funktion: **VAL**

FUNKTION: **STRING$**

ZWECK: Erzeugung eines Strings aus vorgegebenem Zeichen mit vorgegebener Länge

FORMAT:

```
STRING$(Stringlänge, {dez. Wert     })
                     {Stringausdr.  }
```

Stringlänge: num. Ausdruck;
gibt die Länge des Ergebnisstrings an

dez. Wert: num. Ausdruck;
gibt den ISO-Code des Zeichens an

Stringausdruck: das erste Zeichen des Ergebnisses des Stringausdruckes wird übernommen

WIRKUNG: 'Stringlänge' und 'dez. Wert' werden berechnet und zur Ganzzahligkeit kaufmännisch gerundet. Je nach Angabe wird das betreffende Zeichen gemäß der ISO-Code-Tabelle ermittelt. (Bei Angabe von 'Stringausdruck' wird aus diesem das erste Zeichen ermittelt.) Der Ergebnisstring wird durch Aneinanderreihen des ermittelten Zeichens gebildet. Das Zeichen tritt in der durch 'Stringlänge' definierten Anzahl auf.

BEMERKUNGEN:
- 'Stringlänge' muß einen Wert zwischen Ø und 255 aufweisen.
- Ist 'Stringlänge' = Ø, so ist der Ergebnisstring der Leerstring.
- 'dez. Wert' muß einen Wert zwischen Ø und 255 aufweisen.

BEISPIELE:

```
10 LPRINT STRING$(10,65),STRING$(12,"a")
AAAAAAAAAA    aaaaaaaaaaaa
```

VERWEISE: Kapitel 4.3.4.2

ANWEISUNG: **SWAP**

FUNKTION: Vertauschen des Inhalts zweier Variablen vom gleichen Typ

FORMAT:

SWAP { Stringvar.1 , Stringvar.2 / num.Var.1 , num.Var.2 }

WIRKUNG: Der Inhalt von 'Stringvar.2', bzw. 'num.Var.2' wird auf 'Stringvar.1' bzw. 'num.Var.1' zugewiesen und der vorherige Inhalt von 'Stringvar.1' bzw. 'num.Var.1' auf 'Stringvar.2' bzw. 'num.Var.2'.

BEMERKUNGEN:

- Die beiden Variablen müssen vom gleichen Typ sein (Integer, einfache Genauigkeit, doppelte Genauigkeit oder String), sonst wird die Fehlermeldung "Type mismatch" gegeben.

BEISPIELE:

```
10  A=10:B=25:R1$="Text1":R2$="Text2"
20 LPRINT A;B,R1$ SPC(3) R2$
30 SWAP A,B:SWAP R1$,R2$
40 LPRINT A;B,R1$ SPC(3) R2$
50 END
 10   25         Text1    Text2
 25   10         Text2    Text1
```

VERWEISE: Kapitel 4.3.2

BEFEHL: **SYSTEM**

FUNKTION: Aufruf der PCOS-Ebene

FORMAT:

```
SYSTEM
```

WIRKUNG: Das System kehrt von der BASIC-Ebene zur PCOS-Ebene zurück. Alle offenen Datenfiles werden geschlossen. Der Inhalt des Arbeitsspeicher wird gelöscht. Anschließend können alle PCOS-Befehle abgearbeitet werden.

BEMERKUNGEN:

- Im Gegensatz zur BASIC-Ebene, wo PCOS-Befehle mit den Anweisungen **EXEC** bzw. **CALL** aufgerufen werden und in der Form von Stringkonstanten in Anführungszeichen (") eingeschlossen anzugeben sind, sind die PCOS-Befehle in der PCOS-Ebene ohne " einzugeben.

BEISPIELE:

```
system


L1.M20 PCOS  Rev. 1.00
> sp
```

VERWEISE:

FUNKTION: **TAB**
(tabulation)

ZWECK: Positionieren auf eine gewünschte Spalte am Bildschirm oder Drucker

FORMAT:

TAB(Druckspalte)

Druckspalte: num.Ausdruck

WIRKUNG: Der numerische Ausdruck wird berechnet und kaufmännisch zu einer Integer-Zahl gerundet. Der Bildschirm-Cursor bzw. der Druckkopf wird an die durch ihn definierte Spaltenposition bewegt.

BEMERKUNGEN:

- **TAB** darf nur in **PRINT**- oder **LPRINT**-Anweisungen innerhalb der Liste von Ausgabeelementen eingesetzt werden.
- Bei **PRINT USING** bzw. **LPRINT USING**-Anweisungen kann **TAB** vor dem **USING** gesetzt werden.
- Die Position 'Druckspalte' wird auch festgehalten, wenn auf **TAB** kein ; folgt.
- 'Druckspalte' muß zwischen 1 und 255 liegen.
- Liegt die errechnete Spalten-Position links von der aktuellen Position des Bildschirm-Cursors bzw. des Druckkopfes, wird die Position in der nächsten Zeile angesteuert.

BEISPIELE:

```
10 FOR I=.1 TO 1.25 STEP .05
20 LPRINT TAB(TAN(I)*20);"*";
30 NEXT
40 END
 ******* *** ** * * * * *  *  *   *   *     *      *
*
```

VERWEISE: Kapitel 4.3.9.1 und 4.3.9.2
Anweisungen: **PRINT**, **LPRINT**, **WIDTH**, **PRINT USING**, **LPRINT USING**

FUNKTION: **TAN**

ZWECK: liefert den Tangens eines numerischen Werts im Bogenmaß

FORMAT:

```
TAN(num.Ausdr.)
```

WIRKUNG: Der numerische Ausdruck wird berechnet. Er wird als Bogenmaß interpretiert und daraus der Tangens ermittelt.

BEMERKUNGEN:

- Das Argument muß im Intervall $-65535.998 \leq x \leq 65534.428$ liegen, sonst erfolgt "Overflow".
- **TAN** ist die Umkehrfunktion von **ATN**.
- Das Argument ist im Bogenmaß anzugeben. Eine benötigte Umrechnung müßte in einem Unterprogramm oder einer selbstdefinierten Funktion (vgl. **DEF FN**) erfolgen.

BEISPIELE:

```
10 LPRINT TAN(1);TAN(1.5);TAN(2)
 1.55741  14.1014 -2.18504
```

VERWEISE: Kapitel 4.3.16
Funktionen: **ATN**

SPEZIELLE VARIABLE: **TIME$**

FUNKTION: dient zur Verwaltung der von der Echtzeituhr geführten Uhrzeit

FORMAT:

```
TIME$
```

WIRKUNG: Die Variable **TIME$** enthält die aktuelle Uhrzeit in der Form:

HH/MM/SS

wobei HH die Stunde, MM die Minuten und SS die Sekunden bedeuten.
Der Variablen **TIME$** können Werte zugewiesen werden, die eine gültige Uhrzeit darstellen.
Gültig ist:

ØØ≤HH≤24
ØØ≤MM≤6Ø
ØØ≤SS≤6Ø

und ein beliebiges druckbares Trennzeichen, das keine Ziffer sein darf.

Eine Zuweisung in diesem Format bewirkt eine Übernahme vom System als gültige Uhrzeit.

Eine Zuweisung eines Strings, der nicht diesem Format entspricht, bewirkt keine Veränderung der gültigen Uhrzeit.

BEMERKUNGEN:

- Die Variable **TIME$** ist auch ohne Wertzuweisung definiert, da der Wert vom PCOS-Befehl **ss** ständig zur Verfügung steht.
- Die Uhrzeit wird korrekt mitgeführt, solange nicht mit RESET PCOS geladen oder das System ausgeschaltet wird.

- Bei Erreichen der Zeit 00/00/00 wird die reservierte Variable **DATE$** entsprechend dem neuen Datum verändert.
- **TIME$** darf in allen Anweisungen verwendet werden, die keine Wertänderung bewirken. Die Wertänderung ist nur über die Anweisung **LET** möglich; das Schlüsselwort **LET** muß allerdings weggelassen werden.

BEISPIELE:

```
10 INPUT "Datum (TT/MM/JJ)";A$
20 INPUT "Zeit  (HH:MM:SS)";B$
30 C$="ss "+A$+","+B$
40 EXEC C$
50 LPRINT DATE$, TIME$
60 EXEC "sP"  ' nur PR2400
03/04/1982    16/26/35
Datum (TT/MM/JJ)? 03 04 82
Zeit  (HH:MM:SS)? 16/26/35
 Date = 03/04/1982  Time = 16/26/35  Disk Verify = 0 (Off)
 Extent size = 8  Display = 0 (64 x 16)
```

VERWEISE: Kapitel 4.3.11
spezielle Variable: **DATE$**

BEFEHL: **TROFF**
(trace off)

FUNKTION: Aufhebung der Wirkung von **TRON**

FORMAT:

```
TROFF
```

WIRKUNG: Die Nummern der ausgeführten Programmzeilen werden nicht mehr gedruckt. (Die Gültigkeit von **TRON** wird aufgehoben.)

BEISPIELE:

```
10 TRON
20 FOR I=1 TO 10
30 NEXT
40 TROFF

50 FOR I=1 TO 20
60 NEXT
70 END
```

VERWEISE: Befehl: **TRON**

BEFEHL: **TRON**
(trace on)

FUNKTION: Ausdruck der Zeilennummern während des Programmablaufs

FORMAT:

```
TRON
```

WIRKUNG: Die Zeilennummer der jeweils ausgeführten Anweisung wird ausgedruckt.

Die Zeilennummern erscheinen in "Ä" und "Ü" eingeschlossen.

BEMERKUNGEN:
- Der **TRON**-Befehl wird durch den **TROFF**-Befehl, **TROFF** im Direkt-Mode oder den Befehl **NEW** aufgehoben.

BEISPIELE:

```
10 TRON
20 FOR I=1 TO 10
30 NEXT
40 TROFF
50 FOR I=1 TO 20
60 NEXT
70 END
```

VERWEISE: Befehl: **TROFF**

ANWEISUNGS-
BESTANDTEIL:

USING

FUNKTION:

Bestimmung des Formates, in welchem die Ausgabegrößen (Zahlen und/oder Strings) dargestellt werden

FORMAT:

Sprachelement [**TAB**(num.Ausdruck) [;]]
USING Formatstring; Liste von Ausdrücken [{;,}]

Folgende Sprachelemente sind zulässig:

PRINT, **PRINT #** , **LPRINT**

Formatstring: Stringausdruck, der definiert, in welchem Format die Ausgabegröße(n) dargestellt werden soll(en)

Liste von Ausdrücken: Folge von num. und/oder String-ausdrücken, die durch Strichpunkt oder Komma getrennt sind

WIRKUNG:

'Formatstring' wird berechnet. Er besteht aus einer Folge von Formatfeldern. Die einzelnen Ausdrücke von 'Liste von Ausdrücken' werden berechnet und in dem Format dargestellt, welches im 'Formatstring' für den entsprechenden Ausdruck angegeben ist. Jedem Ausdruck in 'Liste von Ausdrücken' wird ein Formatfeld (von links beginnend) zugeordnet. Zulässige Bestandteile von 'Formatstring' sind:

a) für numerische Daten:

1. **#**
2. .
3. +
4. -
5. ^^^^
6. **
7. $$
8. **$
9. ,

b) für Strings:

10. \ \ (Ö Ö im Deutschen Zeichensatz)
11. &
12. !

c) sonstige Zeichen:

13. _ (Unterstreichungszeichen)
14. Folge von Zeichen aller Art (außer 1-13)

Zu 1. Ein Formatfeld für ganzzahlige Größen besteht aus einer Folge von Symbolen (#### ...). Jedes #-Zeichen steht als Platzhalter für eine Ziffer. Bei negativen Zahlen ist für das Vorzeichen ein weiteres Symbol vorzusehen. Die Zahl wird rechtsbündig ins Formatfeld übertragen. Bei nicht ganzzahligen Größen wird der kaufmännisch auf Ganzzahligkeit gerundete Wert dargestellt. Sind in einem Formatfeld mehr Symbole als darstellbare Ziffern vorhanden, so wird das Formatfeld mit Blanks aufgefüllt.

Bsp.: **PRINT USING"** ###↵";12;-15;10.99
12 -15 11

Zu 2. Ein Formatfeld für Dezimalzahlen besteht aus einer Folge von # -Zeichen und einem Dezimalpunkt. Der Dezimalpunkt kann an beliebiger Stelle des Formatfeldes stehen. Er bestimmt somit, mit wieviel Vor- und Nachkommastellen die Dezimalzahl dargestellt wird. Sind für Bruchteile der Zahl nicht genügend Symbole vorgesehen, wird die Zahl auf die letzte darstellbare Stelle kaufmännisch gerundet. Hat die Zahl weniger Nachkommastellen als im Formatfeld vorgesehen, so wird bei den nicht benötigten #-Zeichen mit Ø aufgefüllt. Bei Dezimalzahlen ohne Vorkommastellen werden nur der Dezimalpunkt und die Nachkommastellen dargestellt (keine Ø vor Dezimalpunkt).

Bsp.: **PRINT USING"** ###.##"; 17.238
17.24

Zu 3. Ein + -Zeichen am Anfang des Formatfeldes bewirkt die Darstellung des Vorzeichens (positiv oder negativ) direkt vor der ersten signifikanten Ziffer. Ist das + -Zeichen am Ende des Formatfeldes, so wird das Vorzeichen unmittelbar hinter der Zahl dargestellt.

```
Bsp.: PRINT USING"+###.##↓";12.56;-13.47
       +12.56  -13.47

      PRINT USING"##.##+↓";12.56;-13.47
       12.56+  13.47-
```

Zu 4. Ein - -Zeichen am Ende des Formatfeldes bewirkt die Darstellung des negativen Vorzeichens unmittelbar hinter der Zahl. Positive Vorzeichen werden nicht dargestellt. Das - -Zeichen kann nur am Ende des Formatfeldes stehen.

```
Bsp.: PRINT USING"##.##-↓";-1.25;2.87
       1.25-  2.87
```

Zu 5. Bei numerischen Größen in einem Formatfeld für Zahlen in Exponentialdarstellung entspricht jeder Ziffer ein #-Zeichen (siehe 2). Die vier ^^^^-Zeichen stehen am Ende des Formatfeldes. Sie sind Platzhalter für die Darstellung von **E** (Exponent) oder **D**, dem Vorzeichen und 2 bzw. 3 Ziffern, die den Exponenten zur Basis 10 bilden.

Der Dezimalpunkt kann an beliebiger Stelle des #-Feldes stehen. Die signifikanten Ziffern werden im numerischen Formatfeld linksbündig dargestellt. Aus dieser Darstellung und der festgelegten Position des Dezimalpunkts wird der entsprechende Exponent berechnet. Fehlt die Angabe eines der Zeichen + oder - (siehe 3. u. 4.), so wird ein #-Zeichen links vom Dezimalpunkt für das Vorzeichen verwendet (Blank oder -).

Bsp.:

```
PRINT USING"##.## ^^^^↓";234.56;7635.23;1.45
 23.46E+01  76.35E+02  14.50E-01
```

Zu 6. Ist am Anfang des Formatfeldes ** angegeben, so werden alle nicht benötigte #-Zeichen (siehe 1) mit * dargestellt. ** wirkt ansonsten wie bei einer Formatierung mit zwei #-Zeichen.

```
Bsp.: PRINT USING"**#.#↵";12.39;-0.9;765.1
      *12.4 **-.9 765.1
```

Zu 7. Ist am Anfang des Formatfeldes $$ angegeben, so wird vor der ersten signifikanten Ziffer ein $-Zeichen dargestellt. Ein $-Zeichen kann die Funktion eines #-Zeichens übernehmen. Negative Zahlen in Verbindung mit $$ können nur dann dargestellt werden, wenn das - -Zeichen (siehe 4.) am Ende des Formatfeldes angegeben ist. Bei der Exponentialdarstellung kann für negative Zahlen kein $$ angegeben werden.

```
Bsp.: PRINT USING"$$##.## ";456.78
      $456.78
```

Zu 8. Steht am Anfang des Formatfeldes **$, so bewirkt dies eine Verbindung der Punkte 7 und 8. Führende Blanks werden mit * aufgefüllt und vor der ersten signifikanten Ziffer steht ein $-Zeichen. **$ kann die Funktion von zwei #-Zeichen übernehmen.

```
Bsp.: PRINT USING"**$#.#↵";123.5;1.2
      $123.5 **$1.2
```

Zu 9. Befindet sich in einem Formatfeld für Dezimalzahlen irgendwo links vom Dezimalpunkt ein Komma (dies reicht bereits aus), so werden die Vorkommastellen (von rechts kommend) zu Dreiergruppen zusammengefaßt dargestellt. Die Dreiergruppen werden durch ein Komma getrennt. Das erste Komma wirkt wie ein # -Zeichen. Für eventuelle weiter auftretende Kommas müssen # oder andere Platzhalter vorgesehen werden. Bei der Exponentialdarstellung hat das Komma keine Wirkung.

```
Bsp.: PRINT USING"##,###,### ,.## ";1234567.89
      1,234,567.89
```

Zu 10. \ \definiert ein Formatfeld für die Darstellung von Strings. (Im deutschen Zeichensatz ist \ durch Ö darzustellen). \ dient als Platzhalter für ein Zeichen. Blanks zwischen \ \dienen ebenso als Platzhalter. Anzahl Blanks +2 gibt

an, wieviel Zeichen des Strings dargestellt werden. Der String wird linksbündig in das Formatfeld übertragen. Ist der String länger als das Formatfeld, so werden die nicht zu übertragenden Zeichen abgeschnitten (keine Fehlermeldung). Ist der String kürzer, so wird das Formatfeld rechts mit Blanks aufgefüllt.

```
Bsp.: PRINT USING"\\";"AUTO"
      AU
      PRINT USING"\␣␣\"; "AUTO"
      AUTO
      PRINT USING"\␣␣␣␣\ ";"AUTO"
      AUTO␣␣
```

Zu 11. Das Zeichen & definiert ein Formatfeld von variabler Länge. Der String wird in seiner aktuellen Länge dargestellt.

```
Bsp.: PRINT USING"&";"COMPUTER"
      COMPUTER
      PRINT USING"&";"OS"
      OS
```

Zu 12. Das Zeichen ! definiert ein Formatfeld für ein einzelnes Zeichen. Nur das erste Zeichen des auszugebenden Strings wird dargestellt.

```
Bsp.: PRINT USING"!";"OLIVETTI"
      O
```

Zu 13. Das Unterstreichungszeichen (__) bewirkt eine Darstellung des ersten folgenden Format-Zeichens. Das nachfolgende Zeichen (eines der Zeichen 1-13) wird somit nicht als Bestandteil des Formates angesehen.

```
Bsp.: PRINT USING"_!##.##__";12.49
      !12.49_
      PRINT USING"_##.##";2.76
      #2.76
```

Zu 14. Eine beliebige Folge von Zeichen aller Art (außer 1-13) können dargestellt werden (z.B. als erläuternde Texte innerhalb von 'Formatstring'). Um solche Zeichen darstellen zu können, muß jedoch 'Formatstring' eines der unter 1.-12. definierten Formatfelder enthalten.

```
Bsp.: PRINT USING"$$#.##␣DOLLAR";12.5
      $12.50 DOLLAR
```

Steht am Ende der Anweisung ein ; , so erfolgt die nächste Ausgabe an die Folgeposition (vgl. Wirkung von ; bei **PRINT**)
Steht am Ende der Anweisung ein , , so wird das , nicht berücksichtigt, d.h. es wird nicht so tabuliert wie bei Verwendung von , in der Anweisung **PRINT**.
Zur Wirkung des Sprachelements **TAB** vgl. die Funktion **TAB**.

BEMERKUNGEN:

- Stimmen Typ des Formatfeldes (numerisch oder String) nicht mit dem entsprechenden Ausdruck von 'Liste von Ausdrücken' überein, so erfolgt die Meldung "Type mismatch".
- Reicht für einen numerischen Wert das vorgegebene Formatfeld nicht aus (Masken-Overflow), so wird der Wert unformatiert mit einem führenden %-Zeichen ausgegeben. Die dahinter folgenden Werte werden davon nicht betroffen. Masken-Overflow tritt nur auf bei numerischen Werten, und zwar:

a) wenn der ganzzahlige Anteil inklusive negativem Vorzeichen den dafür vorgesehenen Teil des Formatfeldes überschreitet
oder
b) wenn der Wert zur Beibehaltung des Signifikanzniveaus in Exponentialdarstellung ausgegeben werden müßte, jedoch diese Darstellung nicht vorgesehen ist.

Bei Strings kann kein Masken-Overflow auftreten; es ist jedoch möglich, daß überschüssige Zeichen einfach rechts abgeschnitten werden (ohne Fehlermeldung).

- Da die Möglichkeit besteht, daß numerische Werte bei der Rundung der Nachkommastellen eine weitere Vorkommastelle erhalten, kann dabei ein Masken-Overflow auftreten.

  Bsp.: **PRINT USING"#.##";**A!
  Hat A! den Inhalt 9.997, so tritt Masken-Overflow auf.

- Für numerische Formatfelder können nicht mehr als 23 **#**-Zeichen benützt werden, sonst wird "Illegal function call" gemeldet.
- Eine Maskierung von numerischen Werten mit führenden Nullen ist über **USING** nicht möglich. Es muß auf die Stringverarbeitung zurückgegriffen werden (vgl. Kapitel 4.3.4).
- In einer Anweisung, die das Sprachelement **USING** verwendet, kann die Ausgabe von mehreren zu formatierenden Ausgabeelementen vorgeschrieben werden.

- Für die Anzahl der Formatfelder im Vergleich zur Anzahl der zu formatierenden Ausdrücke sind 3 Fälle zu unterscheiden:

a) Die Anzahl der Formatfelder stimmt mit der Anzahl zu formatierender Ausdrücke überein.
b) Die Anzahl der Formatfelder ist größer als die Anzahl der zu formatierenden Ausdrücke:
Die am Ende nicht benötigten Formatfelder werden nicht berücksichtigt.
c) Die Anzahl der Formatfelder ist geringer als die Anzahl der zu formatierenden Ausdrücke:
Die Formatfelder werden zuerst von links nach rechts voll verwertet. Danach wird erneut ab dem ersten angegebenen Formatfeld weitergearbeitet.

- 'Formatstring' kann aus Stringkonstanten, Stringvariablen und/oder Stringoperationen gebildet werden.
- Ist an Bildschirm oder Drucker das Zeilenende erreicht (z.B. auch wegen einer **WIDTH**-Festlegung), die Liste der zu formatierenden Ausgabeelemente aber noch nicht zu Ende, wird in der nächsten Zeile weiter ausgegeben.
- Wird die **TAB**-Funktion vor einem **USING** eingesetzt, werden die in der aktuellen Zeile vor der durch **TAB** definierten Position befindlichen Zeichen mit Blanks gelöscht!!
- Wird ein einfach genauer Wert mit **USING** maskiert, so geschieht die Rundung bereits von der siebten auf die sechste signifikante Stelle.

BEISPIELE:

```
10 LPRINT USING "####";99,99,2,99,7
20 LPRINT USING "####.###  ";1.21,354.987,-25.7895,10000.111
#
30 LPRINT USING "+###.###  ";68.95,-68.95 ,25.7895,25.7896
40 LPRINT USING "####.##-  ";123.4,-123.4,78.986,-78.986
50 LPRINT USING  "**##.### ";2.145,-3.789,120.19,-145.5896#
60 LPRINT USING "$$##.### ";25.879,-150.12
70 LPRINT USING "**$##.#####  ";1275.96,-8009.25,95145.12345
6#
80 LPRINT USING "#########,.### ";622850.123#,-50.6789,1000
90 LPRINT USING "##.##^^^^   ";1250,-888888!
100 LPRINT USING "_Saldo ####.##_ DM   ";1.25,2500.1,80
110 LPRINT USING  "##.##  ";111.222,-88.75
120 LPRINT USING ".###   ";.9999,-2.79
130 MASKE$="_Distanz= ####.###  _m "
140 LPRINT USING MASKE$;5.25,99.987
  99  99 100
  1.210   354.987   -25.789  %10000.111
 +68.950   -68.950   +25.789   +25.790
 123.40    123.40-    78.99     78.99-
***2.145 **-3.789 *120.190 -145.590
 $25.879 %-$150.120
$1275.95896  %-$8009.25000  %$95145.12346
   622.850.123          -50.679       1.000.000
 1.25E+03   -8.89E+05
Saldo     1.25 DM   Saldo 2500.10 DM   Saldo   80.00 DM
%111.22  %-88.75
%1.000   %-2.790
Distanz=    5.250  m Distanz=   99.987  m
```

```
10 A$="zum ":B$="Beispiel"
20 PRINT USING "!";A$;B$
30 PRINT USING "!";A$,B$

40 PRINT USING  "\   \";A$,B$
50 PRINT USING  "\     \";A$,B$
60 PRINT USING "!";A$;
70 PRINT USING "&";B$
80 EXEC "sp"    ' nur PR2400
zB

zB

zum Beis

zum   Beispi

zBeispiel
```

VERWEISE: Kapitel 4.3.9 (und 4.3.4 für komplizierte Druckaufbereitungen über Stringverarbeitung)
Anweisungen: **PRINT, PRINT USING, LPRINT, LPRINT USING, PRINT#, PRINT#...USING, WIDTH**
Funktion: **TAB**

FUNKTION: **VAL**
(value)

ZWECK: Aus einem String, der am Anfang eine Zahlendarstellung enthält, wird diese in den numerischen Wert umgewandelt

FORMAT:

```
VAL(Stringausdruck)
```

WIRKUNG: Der Stringausdruck wird berechnet. Falls die ersten Zeichen dieses Strings einen num. Wert darstellen, wird dieser in den entsprechenden numerischen Wert umgewandelt.

BEMERKUNGEN:

- Die Funktion **VAL** ist die logische Umkehrung der Funktion **STR$**.
- Alle plausiblen Zahlendarstellungen werden korrekt umgewandelt.
  ACHTUNG: Overflow möglich!
- Wird keine plausible Zahlendarstellung am Anfang des 'Stringausdrucks' ermittelt, liefert die Funktion den Wert Ø.

BEISPIELE:

```
10 A$="-4657.33"
20 LPRINT 2*VAL(A$)
30 END
-9314.66
```

VERWEISE: Kapitel 4.3.4.3
Funktion: **STR$**

FUNKTION: **VARPTR**

ZWECK: liefert die Byte-Adresse einer Variablen im Arbeitsspeicher oder die Start-Adresse des I/O-Puffers, der der Filenr. eines sequentiellen Files zugeordnet ist bzw. die Adresse des Random-File-Puffers für das unter einer Filenr. geöffnete Random-File

FORMAT:

VARPTR({ Variablenname / #Filenr. })

Filenr.: numerische Konstante; Wert: 1-15

WIRKUNG:

a) Angabe eines Variablennamens:
Die Funktion liefert die Byte-Adresse der angegebenen Variablen im Arbeitsspeicher. Das Ergebnis ist eine Integerzahl zwischen -32768 und 32767. Wird eine negative Zahl ausgegeben, so ist die aktuelle Adresse diese Zahl +65536.

b) Angabe einer Filenr. in Form einer num. Konstanten:
Die Funktion liefert die Start-Adresse des I/O-Puffers des unter dieser Filenr. geöffneten sequentiellen Files (Zugriffsarten: "I", "A" oder "O"). Wurde unter der Filenr. ein Random-File geöffnet (Zugriffsart "R"), so wird die Adresse des Random-File-Puffers geliefert.

BEMERKUNGEN:

- Es können nur Adressen von Variablen ermittelt werden, die vorher über eine Zuweisung echte Werte erhielten. Die Default-Werte (Ø bzw. Leerstring) werden nicht über Zuweisung gesetzt. Auch **DIM** und **ERASE** führen keine Zuweisung aus.
- Es kann die Adresse von bestimmten Elementen eines Arrays abgefragt werden.
- Das Array-Element mit dem niedrigsten Index (Ø oder 1) hat stets die niedrigste Adresse aller Array-Elemente im Arbeitsspeicher.
- Werden Adressen von Array-Elementen abgefragt, ist zu beachten:
Jede Zuweisung auf eine neue Variable ändert die Adressen der Array-Elemente.

- Nach Stringmanipulationen können sich die Adressen aller Variablen ändern.
- Nach Verwendung der Funktion **FRE(**Str.Ausdr.**)** können sich die Adressen aller Variablen ändern.
- Es empfiehlt sich, vor Abfrage einer Adresse eine Zuweisung auf die betreffende Variable zu machen (z.B. A(2)=A(2)).

BEISPIELE:

```
10 A=2.45466:B%=44.6:C!(5)=66
20 'LPRINT VARPTR(A);VARPTR(B%);VARPTR(C!(0))
30 END
 8772  8780  8791
```

VERWEISE: Kapitel 4.3.16

ANWEISUNG: **WEND**

FUNKTION: Endanweisung einer **WHILE-WEND**-Schleife

FORMAT:

```
WEND
```

WIRKUNG: siehe Anweisung **WHILE**

BEISPIELE:

```
10 I=1:LPRINT"Schleifendurchlauf:"
20 WHILE INT(RND*10)<>9
30 LPRINT I; : I=I+1
40 WEND : LPRINT "Ende nach";I;"Durchlaeufen"
Schleifendurchlauf:
 1  2  3  4  5  6 Ende nach 7 Durchlaeufen
```

VERWEISE: Kapitel 4.3.6
Anweisung: **WHILE**, **CLEAR**
Fehler-Code: 30

ANWEISUNG: **WHILE**

FUNKTION: Start-Anweisung einer Schleife, die solange durchgeführt wird, wie eine Bedingung erfüllt ist

FORMAT:

```
WHILE log.Ausdruck
        .
      Anweisung(en)
        .
WEND
```

log. Ausdruck: Vergleichsausdruck, logischer Ausdruck oder Verknüpfung von Vergleichs- oder logischen Ausdrücken durch logische Operatoren; siehe Kapitel 2.7.3

WIRKUNG: Trifft das System auf eine **WHILE**-Anweisung, wird dieser die nächste freie **WEND**-Anweisung zugeordnet.
Der logische Ausdruck wird berechnet. Ist sein Wahrheitswert Ø, wird die zugeordnete **WEND**-Anweisung angesprungen, die dazwischen liegende(n) Anweisung(en) übergangen und die auf **WEND** folgende Anweisung abgearbeitet.

Ist der Wahrheitswert von 'log. Ausdruck' ungleich Ø, werden alle Anweisungen bis zum **WEND** ausgeführt und von dort wieder zur betreffenden **WHILE**-Anweisung zurückverzweigt.

BEMERKUNGEN:
- **WHILE**-Schleifen können verschachtelt werden.
- Jedes **WHILE** muß mit genau einem **WEND** gepaart sein.
- Achtung: Bei vorzeitigem Aussprung aus **WHILE-WEND**-Schleifen (z.B. mit **GOTO**) ohne späteren Rücksprung bleibt ein Stack-Speicher belegt. Dies kann bei wiederholtem Auftreten in einem Programm zu Speicher-Overflows führen (vgl. **FOR...NEXT**-Schleife).

BEISPIELE:

```
10 I=1:LPRINT"Schleifendurchlauf:"
20 WHILE INT(RND*10)<>9
30 LPRINT I; : I=I+1
40 WEND : LPRINT "Ende nach";I;"Durchlaeufen"
Schleifendurchlauf:
 1  2  3  4  5  6 Ende nach 7 Durchlaeufen
```

VERWEISE: Kapitel 2.7.3, 4.3.5 und 4.3.6
Anweisung: **WEND**, **CLEAR**
Fehler-Code: 29

ANWEISUNG: **WIDTH**

FUNKTION: Festlegung der Anzahl Zeichen pro Ausgabezeile für Drucker oder Bildschirm

FORMAT:

```
WIDTH [LPRINT] (Anzahl Zeichen)
```

Anzahl Zeichen: num.Ausdruck (Ergebnis: 15-255)

WIRKUNG: Die 'Anzahl Zeichen' wird berechnet und auf Ganzzahligkeit gerundet. Der Wert legt fest, wieviele Zeichen pro Ausgabezeile ausgegeben werden sollen, bevor eine automatische Zeilenschaltung erfolgt. Ist der Parameter **LPRINT** nicht angegeben, gilt die Anweisung für den Bildschirm, andernfalls für den Drucker.

BEMERKUNGEN:

- Die 'Anzahl Zeichen' muß zwischen 15 und 255 liegen.
- Der Default-Wert für 'Anzahl Zeichen' ist für den Bildschirm 64 oder 80 Zeichen und für den Drucker 132 Zeichen (werden durch die PCOS-Befehle **ss** und **sf** festgelegt).
- Ist 'Anzahl Zeichen' gleich 255, ist die Zeichenanzahl pro Zeile "unendlich", d.h., es wird niemals eine Zeilenschaltung (CR) vom System angesetzt. Die Funktionen **POS** bzw. **LPOS** werden jedoch auf 0 gesetzt, sobald die 255. Position erreicht wurde.
- Für Tastatureingaben über **INPUT** oder **LINE INPUT** kann die Zeilenlänge am Bildschirm nicht mit **WIDTH** begenzt werden.

BEISPIELE:

```
10 A$="ABCDEFGHIJKLMNOPQRSTUVW"
20 LPRINT A$
30 WIDTH LPRINT 10
40 LPRINT A$
ABCDEFGHIJKLMNOPQRSTUVW
ABCDEFGHIJ
KLMNOPQRST
UVW
```

VERWEISE: Kapitel 4.3.9.1 und 4.3.9.2
Anweisungen: **PRINT**, **WRITE**, **LPRINT**
Funktionen: **POS**, **LPOS**

FUNKTION: **WINDOW**

ZWECK: Definition der Lage und Größe eines neuen Windows ("Anlegen" bzw. "Eröffnen") und/oder (Neu-)Festlegung der Anzahl Zeichen pro Zeile und/oder der Anzahl Zeilen in einem Window

FORMAT:

```
Window-Nr.=WINDOW(Quadrant,Position,[Zeilenhöhe][,Spaltenbreite])
```

Window-Nr.:	num.Variable, die einen der Werte 1 bis 16 enthält, der der Nr. des Windows entspricht; bei neu angelegten Windows wird diese vom System automatisch zugewiesen
Quadrant:	num.Ausdruck (Ergebnis: 0: oberer Teil, 1: unterer Teil, 2: linker Teil, 3: recher Teil des aktiven Windows)
Position:	num.Ausdruck, dessen Ergebnis eine ganze Zahl sein muß und die Größe des neuen Windows bestimmt (horizontale Teilung: max. 239; bei 0,1 vertikale Teilung: max. 79) bei 2,3
Zeilenhöhe:	num.Ausdruck; zulässige Werte: 10, 11...16
Spaltenbreite:	num.Ausdruck; zulässige Werte: 6 (80 Zeichen pro Gesamtbildschirm-Zeile) und 8 (64 Zeichen pro Gesamtbildschirm-Zeile)

WIRKUNG: 'Quadrant', 'Position', 'Zeilenhöhe' und 'Spaltenbreite' werden berechnet und kaufmännisch auf Ganzzahligkeit gerundet.

I Anlegen ("Eröffnen") eines neuen Windows durch Definition der Größe und ggf. Festlegung der Anzahl Zeichen pro Bildschirmzeile und/oder der Anzahl Zeilen im Bildschirm für dieses Window:

Vorbemerkung:

Ein neues Window kann nur durch waagrechte oder senkrechte Teilung eines bereits existierenden Windows angelegt werden. Dieses Window muß aktiv sein (vgl. Anweisung **WINDOW%**).
Es kann zur gleichen Zeit immer nur ein Window aktiv, d.h. ansprechbar, sein. Zu Beginn ist der Gesamtschirm (Window 1) aktiv.

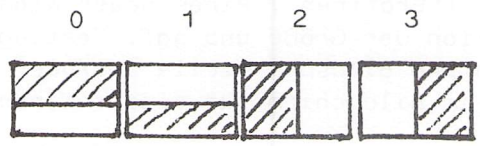
Durch Teilung des aktiven (zuletzt per **WINDOW%** angewählten) Windows wird ein neues Window angelegt und das aktive um diesen Bereich reduziert. Nach der Teilung bleibt das Ausgangswindow aktiv.

Der Variablen 'Window-Nr.' wird vom System automatisch diejenige ganze Zahl zwischen 2 und 16 zugewiesen, die dem aktiven Window folgt und noch nicht vergeben ist, um ein Window zu spezifizieren.

Das neue Window ist in Zukunft unter dieser Zahl "eröffnet" und anzusprechen (ggf. allerdings vorher zu aktivieren!). Der volle Bildschirm ist zu Beginn grundsätzlich aktiv und hat die Window-Nr. 1.

Quadrant:

Durch den Parameter 'Quadrant' wird die Lage (und nur die Lage) des neuen Windows in Bezug auf das aktive Window bestimmt. Er kann die Zahlen Ø, 1, 2 oder 3 annehmen und legt damit fest, daß das neue Window im oberen, unteren, linken oder rechten Teil des aktiven Windows angelegt wird. Die nachfolgende Skizze soll dies veranschaulichen:

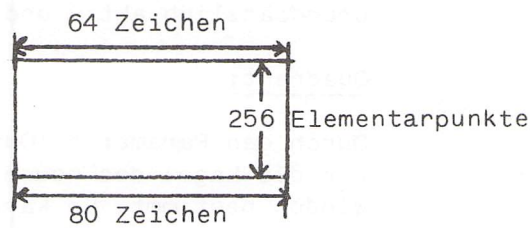


Das neu angelegte Window ist jeweils schraffiert dargestellt. Der unschraffierte Bereich ist das reduzierte, aktive Ausgangswindow.
Man beachte, das durch den Parameter 'Quadrant' nur die Lage, nicht aber die Größe des neuen Windows definiert wird.

Position:

Mit 'Position' wird die Größe des neuen Windows festgelegt. Das aktive Window, aus dem das neue durch Teilung hervorgeht, wird um den entsprechenden Bereich reduziert. Die Größe des Parameters 'Position' wird bei horizontaler Teilung in Anzahl Elementarpunkten und bei vertikaler Teilung in Anzahl Textspalten angegeben.

Der Gesamtschirm besitzt vertikal 256 Elementarpunkte und horizontal entweder 64 oder 80 Textspalten (Vergleiche dazu Kapitel 4.3.18).



Wurde für 'Quadrant' Ø oder 1 (horizontale Teilung) gesetzt, so ist für 'Position' die Anzahl Elementarpunkte vom oberen Rand des aktiven Windows anzugeben, gleichgültig ob das neue Window im oberen oder unteren Teil liegt. Das neue Window besitzt dann vertikal eine Ausdehnung, die gleich der angegebenen Anzahl Elementarpunkte ist. Das Ausgangswindow erstreckt sich danach vertikal über soviel Elementarpunkte, wie das Ausgangswindow vor der Teilung hatte, minus der für das neue Window definierten Anzahl Elementarpunkte ('Position').

Wurde für 'Quadrant' 2 oder 3 (vertikale Teilung) gesetzt, so ist für 'Position' die Anzahl Textspalten vom linken Rand des aktiven Windows anzugeben, gleichgültig ob das neue Window im linken oder rechten Teil liegt. Das Ausgangswindow besitzt danach soviel Textspalten, wie das Ausgangswindow vor der Teilung hatte, minus der für das neue Window definierten Anzahl Textspalten ('Position').

Eine Textspalte besteht - je nach Definition (80 oder 64 Zeichen pro Zeile) - aus 6 oder 8 Elementarpunkten.

Der Wertbereich für den Parameter 'Position' ist wie folgt definiert:

Parameter 'Position'

Horizontale Teilung (Quadrant Ø u. 1)	'Zeilenhöhe'+1≤'Position'≤Anzahl Elementarpunkte des aktiven Windows (vertikal) minus 'Zeilenhöhe'
vertikale Teilung (Quadrant 2 u. 3)	1≤'Position'≤ 79 bei 80 Zeichen pro Bildschirmzeile; 63 bei 64 Zeichen pro Bildschirmzeile

Tabelle 1: 'Position'

Zeilenhöhe

Durch Angabe der 'Zeilenhöhe' - in Anzahl Elementarpunkten kann der Zeilenabstand innerhalb des Windows festgelegt werden. Dabei setzt sich die 'Zeilenhöhe' zusammen aus der Zeichenhöhe (7 Elementarpunkten) plus 3 bis 9 Elementarpunkten für den Abstand zwischen den Zeilen.

In der nachfolgenden Tabelle wird Anzahl Zeilen pro Gesamtschirm in Abhängigkeit vom Parameter 'Zeilenhöhe' dargestellt:

'Zeilenhöhe' (Elementar-punkte)	Zeilenabstand (Elementar-punkte	Anzahl Zeilen im Gesamt-bildschirm
10	3	25
11	4	23
12	5	21
13	6	19
14	7	18
15	8	17
16	9	16

Tabelle 2: 'Zeilenhöhe'

Wird der Parameter 'Zeilenhöhe' nicht angegeben, so wird die 'Zeilenhöhe' des aktiven Windows übernommen.

Spaltenbreite

Durch die Angebe der 'Spaltenbreite' (6 oder 8 Elementar-punkte) kann man den Spaltenabstand innerhalb des Windows festlegen. Dabei setzt sich die 'Spaltenbreite' zusammen aus der Zeichenbreite (5 Elementarpunkte) plus Spaltenabstand (3 oder 1 Elementarpunkt). Bezogen auf den Gesamtschirm wird durch Angabe der 'Spaltenbreite' die Anzahl Zeichen pro Zeile von 80 auf 64 und umgekehrt geschaltet.

'Spaltenbreite' (Elementarpunkte)	Spaltenabstand (Elementarpunkte)	Anzahl Zeichen im Gesamtbild-schirm
6	1	80
8	3	64

Tabelle 3: 'Spaltenbreite'

Wird der Parameter 'Spaltbreite' nicht angegeben, so wird die 'Spaltenbreite' des aktiven Windows übernommen.

II Neufestlegung der Anzahl Zeichen pro Bildschirmzeile und/oder Anzahl Zeilen im Bildschirm für ein bereits angelegtes Window:

Für ein bereits existierendes Window, können 'Zeilenhöhe' und 'Spaltenbreite' neu definiert werden. Dazu muß im Aufruf der Funktion **WINDOW** die Variable 'Window-Nr.' die Nr. des Windows beinhalten, dessen Eigenschaften geändert werden sollen; das Window muß vorher aktiviert werden.
Die Parameter 'Quadrant' und 'Position' müssen den Wert Ø besitzen.

Die Parameter 'Zeilenhöhe' und 'Spaltenbreite' sind anzugeben wie unter I beschrieben.

BEMERKUNGEN:

- Beim Anlegen von Windows wird der Variablen 'Window-Nr.' automatisch eine ganze Zahl zwischen 2 und 16 zugewiesen, während bei Neufestlegung von 'Zeilenhöhe' und/oder 'Spaltenbreite' in einem bereits existierenden Window der Variablen 'Window-Nr.' vorher der betreffende Wert zugewiesen werden muß und das betreffende Window durch die **WINDOW %**-Anweisung aktiviert werden muß.
- Auf dem Bildschirm können gleichzeitig Windows mit z.B. 80 x 25-Format und 64 x 16-Format dargestellt werden. Dies sind die möglichen Default-Werte für das Format.
- Es können maximal 16 Windows gleichzeitig verwaltet werden. Im PCOS-Befehl **sb** ist die Anzahl Windows anzugeben, die maximal angelegt werden können.
- Im Window 1 (Gesamtschirm) wird für 'Zeilenhöhe' und 'Spaltenbreite' der im PCOS-Befehl **ss** angegebene Default-Wert angenommen.
- Wird beim Anlegen eines Windows 'Zeilenhöhe' und/oder 'Spaltenbreite' nicht angegeben, so werden die Werte des aktiven Windows angenommen.
- Werden beim Anlegen von Windows falsche Parameter angegeben, so wird die Meldung "Unable to create window" ausgegeben.
- In jedem Window kann alphanumerisch und/oder graphisch gearbeitet werden.

- In jedem Window kann mit dem Text-Cursor oder dem Graphik-Cursor gearbeitet werden. Dabei bezieht sich seine Position auf das Window (vgl. **CURSOR**).
- Die Anzahl Elementarpunkte im Parameter 'Position' muß keine Vielfache des Parameters 'Zeilenhöhe' sein.
- Die Anzahl Zeilen innerhalb eines Windows ergibt sich bei vorgegebener 'Zeilenhöhe' aus

  INT('Position'/'Zeilenhöhe')

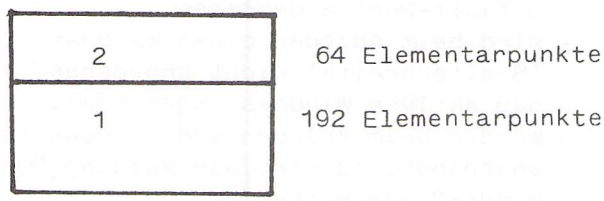
- Der Ursprungspunkt des Text-Cursors ist die oberste linke Textposition jedes Windows.
- Der Default-Wert für den Koordinatenursprung beim Arbeiten mit dem Graphik-Cursor ist der unterste linke Elementarpunkt des betreffenden Windows.
- Die letzte Position und die Eigenschaften des jeweiligen Cursors bleiben beim Aktivieren eines anderen Windows gespeichert, so daß jederzeit darauf zurückgegriffen werden kann.

BEISPIELE:

1. (Es ist noch kein Window angelegt worden):

```
Z=WINDOW(0,64)
```

Der Bildschirm wird in zwei Windows aufgeteilt, wobei das neue Window im oberen Teil entsteht und vertikal eine Ausdehnung über 64 Elementarpunkte hat. Der Variablen Z wird vom System der Wert 2 zugewiesen. Das Window 1 wird auf die Größe von 192 Elementarpunkten reduziert.



Für 'Zeilenhöhe' und 'Spaltenbreite' werden die im PCOS-Befehl **ss** gesetzten Default-Werte genommen. Sind z.B. in **ss** die 16 für 'Zeilenhöhe' und 8 für 'Spaltenbreite' definiert, so besteht das Window 2 aus 4 und Window 1 aus 12 Zeilen.
Das Window 1 ist nach wie vor aktiv, d.h. das Anlegen eines neuen Windows bezieht sich auf dieses Window.

2. (Es ist noch kein neues Window angelegt worden):

R%=WINDOW(1,64)

wie Beispiel 1, jedoch liegt das neue Window unten.
Die Variable R% hat den Wert 2.

3.

A%=1
A%=WINDOW(0,0,10,6)

Im vorhandenen Window 1 wird 'Zeilenhöhe' zu 10 und 'Spaltenbreite' zu 6 geändert, d.h. der Bildschirm besitzt danach 25 Zeilen à 80 Zeichen.

4.

```
10 CLEAR
20 WNR%=WINDOW(3,32,,6)
30 WINDOW %WNR%
40 WNR%=WINDOW(1,80,,8)
50 WINDOW %WNR%
51 FOR I%=1 TO 3 : CLS %I% : NEXT
55 PRINT "Bitte Window-Nr. 1-3 eingeben"
60 K$=INPUT$(1) : IF K$<>"1" AND K$<>"2" AND K$<>"3"
THEN 60
65 WINDOW %VAL(K$)
66 K1$="" : WHILE K1$=""
80 K1$=INKEY$
90 PRINT K$; : WEND
120 PRINT "Dies ist Window Nr " VAL(K$)     GTO 55
130 GOTO 55
```

VERWEISE: Kapitel 4.3.9, 4.3.14 und 4.3.18
Anweisungen: **WINDOW%, CLOSE WINDOW, CLS, CURSOR, SCALE, COLOR, COLOR=**

ANWEISUNG: **WINDOW%**

FUNKTION: Anwählen ("Aktivieren") eines angelegten Windows

FORMAT:

```
WINDOW% Window-Nr.
```

Window-Nr.: numerischer Ausdruck (Ergebnis: 1 bis 16)

WIRKUNG: 'Window-Nr.' wird berechnet und auf Ganzzahligkeit kaufmännisch gerundet.

Das mit **WINDOW%** angewählte Window ist dann aktiv. Alle weiteren Anweisungen beziehen sich dann auf dieses Window (Ausnahmen: Anweisungen, in denen gezielt andere als das aktive Window angesprochen werden können). Es bleibt so lange aktiv, bis ein anderes Window aktiviert wird.

BEMERKUNGEN:

- Vor Beginn ist der gesamte Bildschirm unter der 'Window-Nr.' 1 aktiviert.
- Das Anlegen von neuen Windows geschieht immer durch Teilung des aktiven Windows.
- Anweisungen wie **PRINT** oder **INPUT** beziehen sich immer auf das aktive Window.
- Soll ein neues Window aktiviert werden, ist eine neue **WINDOW%**-Anweisung nötig. Es kann nur ein Window aktiv sein.
- Von den letzten beiden Bemerkungen ausgenommen sind alle Funktionen und Anweisungen, in denen ein Window spezifiziert werden kann (erkenntlich am **%**-Zeichen als Bestandteil), z.B. **CLS %**, **GET %**.
- Ein Window kann nur aktiviert werden, wenn es zwar unter Zuhilfenahme der **WINDOW**-Funktion angelegt ("eröffnet") wurde und das betreffende Window nicht durch eine **CLOSE WINDOW**-Anweisung aufgelöst wurde. In einem solchen Falle wird der Fehler "Window not open" gemeldet.

BEISPIELE:

```
5 CLEAR
10 W1=WINDOW(1,100,10,6)
20 W2=WINDOW(3,50,10,8)
30 WINDOW%1:FOR I=1 TO 10:  PRINT"Fenster 1":NEXT
40 WINDOW%2:FOR I=1 TO 5:PRINT"Fenster 2":NEXT
50 WINDOW%3:FOR I=1 TO 5:PRINT"Fenster3":NEXT
70 CALL "sp"    ' nur PR2400
80 END
```

```
Fenster 1                                                    Fenster3
                                                             Fenster3
Fenster 1                                                    Fenster3
                                                             Fenster3
Fenster 1                                                    Fenster3
Fenster 1
Fenster 1


Fenster 2
Fenster 2
Fenster 2
Fenster 2
Fenster 2
```

VERWEISE: Kapitel 4.3.9 und 4.3.14
Anweisungen: **CLS %, CLOSE WINDOW, CURSOR, COLOR=, COLOR**
Funktion: **WINDOW**
Fehler-Code: 35

ANWEISUNG: **WRITE**

FUNKTION: Ausgabe von Daten am Bildschirm; als Trennzeichen werden automatisch Kommas ausgegeben; Strings werden automatisch in Anführungszeichen eingeschlossen ausgegeben

FORMAT:

WRITE [Liste von Ausdrücken]

Liste von Ausdrücken: numerische und/oder String-Ausdrücke; getrennt durch Kommas

WIRKUNG: Die Ergebnisse der durch Komma getrennten Ausdrücke in der 'Liste von Ausdrücken' werden auf dem Bildschirm ausgegeben. Zwischen die Elemente der 'Liste von Ausdrücken' werden automatisch Kommas gesetzt. Strings werden automatisch in Anführungszeichen eingeschlossen ausgegeben. Nach dem letzten Element der 'Liste von Ausdrücken' wird automatisch eine Zeilenschaltung vorgenommen. **WRITE** ohne Parameter bewirkt eine Zeilenschaltung.

BEMERKUNGEN:

- Numerische Werte werden im Standardformat (vgl. **PRINT**-Anweisung) ausgegeben.
- Eine weitere Formatierung mit Hilfe von **USING** ist nicht möglich.
- Eine Tabulation mit **TAB** ist nicht möglich, wohl aber eine Ausgabe ab einer durch **CURSOR** festgelegten Position.
- Der Bildschirm-Cursor kann nicht durch ; an der letzten Position der Ausgabeliste festgehalten werden; es erfolgt zwangsweise eine Zeilenschaltung.

BEISPIELE:

```
10 CLS : A!=20 : A1$="M" : A2$="ist Spitze"
20 PRINT A1$;A!;A2$ : WRITE A1$;A!;A2$
30 CALL "sp"    ' nur PR2400
M 20 ist Spitze

"M",20,"ist Spitze"
```

VERWEISE: Kapitel 4.3.9.2
Anweisungen: **PRINT**, **CURSOR**, **WIDTH**

ANWEISUNG: **WRITE #**

FUNKTION: Schreiben von Daten auf ein externes sequentielles File; als Trennzeichen wird automatisch ein Komma auf das File geschrieben; Strings werden automatisch in Anführungszeichen eingeschlossen auf das File gebracht; am Ende wird ein CR und ein LF geschrieben

FORMAT:

```
WRITE # Filenr.,Liste von Ausdrücken
```

Filenr.: num.Ausdruck (Ergebnis: 1-15)

Liste von Ausdrücken: numerische und/oder String-Ausdrücke; getrennt durch Kommas

WIRKUNG: Die 'Filenr.' wird berechnet und gerundet. Auf das unter dieser Filenr. geöffnete sequentielle File (Zugriffsart: "O", "A") werden ab der Position des Pointers die Ergebnisse der durch Komma getrennten Ausdrücke geschrieben. Zwischen die Elemente der 'Liste von Ausdrücken' wird ein Komma (=**CHR$**(44)) auf das File geschrieben. Strings werden in Anführungszeichen (**CHR$**(34)) eingeschlossen auf das File gebracht. An das Ende der 'Liste von Ausdrücken' wird automatisch ein CR (=**CHR$**(13)) und ein LF (=**CHR$**(10)) auf das File geschrieben.

BEMERKUNGEN:

- Numerische Werte werden im Standardformat (vgl. **PRINT**-Anweisung) auf das File gebracht.
- Enthalten Stringvariablen führende oder am Ende stehende Blanks, werden diese auf das File geschrieben.
- **WRITE#** schreibt die Daten im Grunde so auf das File, wie sie durch **WRITE**-Anweisung auf dem Bildschirm dargestellt würden, allerdings einschließlich nicht druckbarer Zeichen.
- Im Gegensatz zu **PRINT#** werden die Daten automatisch durch das Komma getrennt auf das File geschrieben; die Strings werden automatisch in Anführungszeichen eingeschlossen und das Ende der 'Liste von Ausdrücken' immer durch ein CR (=**CHR$**(13)) und ein LF (=**CHR$**(10)) gekennzeichnet.
- **WRITE #** ist bei Schreibschutz einer Diskette nicht möglich und führt zum Fehler "Disk I/O-Error)", der in diesem Fall nur noch durch Abschalten des M 20 behoben werden kann. (Verlust des Inhalts des Arbeitsspeichers!)

- Wird die bisherige (oder durch PCOS-Befehl **fn** festgelegte) File-Größe überschritten, wird automatisch versucht, für das File soviel neue Sektoren auf Diskette zu verservieren, wie durch den PCOS-Befehl **ss** spezifiziert wurde. Dies kann sofort zum Fehler "Disk full" führen, obwohl für den Record selbst noch ausreichend Platz vorhanden wäre.
- Mit Hilfe der Funktion **LOC** kann abgefragt werden, ob auf Diskette noch genügend Platz für die von **WRITE#** betroffenen Daten ist.

BEISPIELE:

```
10 OPEN "O",#1,"1:sequen"
20 A$="text":NR=1
30 WRITE#1,A$,NR
50 CLOSE
60 OPEN "I",#1,"1:sequen"
70 INPUT#1,A1$,NR1
80 LPRINT A1$,NR1
90 CLOSE

 text            1
```

VERWEISE:

Kapitel 4.3.10.2
Anweisungen: **OPEN**, **INPUT#** , **LINE INPUT#** , **PRINT#** , **WRITE**
Funktionen: **LOC**, **LOF**
Fehler-Codes: 54, 61

7. MELDUNGEN DES INTERPRETERS; FEHLER-CODES

7.1 LISTE DER FEHLER-CODES (BASIC-FEHLER)

Die folgende Liste ist eine Zusammenstellung der Meldungen des BASIC-Interpreters (BASIC-Fehler)

ERROR-Nr.		MELDUNG
1		NEXT without FOR
2		Syntax error
3		RETURN without GOSUB
4		Out of DATA
5		Illegal function call
6		Overflow
7	*	Out of memory
8		Undefined line number
9		Subscript out of range
10		Duplicate definition
11		Division by zero
12		Illegal direct
13	*	Type mismatch
14		Out of string space
15		String too long
16		String formula too complex
17		Can't continue
18		Undefined user function
19		No RESUME
20		RESUME without error
21		Unprintable error
22		Missing operand
23		Line buffer overflow
26		FOR without NEXT
29		WHILE without WEND
30		WEND without WHILE
31		IEEE: Invalid talker/listener address
32		IEEE: talker=listener address
33		IEEE: unprintable error
34		IEEE: Board not present
35		Window not open
36		Unable to create window
37		Invalid action-verb
38		Parameter out of range
39		Too many dimensions
40		[Ausgabe eines PCOS-Fehlers]

50		FIELD overflow
51	*	Internal error
52		Bad file number
53	*	File not found
54	*	Bad file mode
55	*	File already open
57	*	Disk I/O error
58	*	File already exists
61	*	Disk full
62	*	Input past end
63	*	Bad record number
64	*	Bad file name
66		Direct statement in file
67	*	Too many files
68		Internal error
69		Volume name not found
70		Rename error
71	*	Volume number error
72	*	Volume not enabled
73		Invalid Password
74		Illegal disk change
75		WriteProtected
76	*	Error in Parameter
77	*	Too Many Parameters
78	*	File not open

Die nicht angeführten Error-Nummern bewirken die Meldung "Unprintable error". Sie können im Anwenderprogramm beliebig besetzt werden. Die höchste zulässige Error-Nummer ist 255.

Beinhaltet die reservierte Variable **ERR** den Inhalt Ø, so ist entweder seit dem Laden des Systems über die PCOS-Ebene kein Fehler aufgetreten oder zuletzt eine Fehlerbehandlungsroutine mit **RESUME** verlassen worden.

Die mit * versehenen Fehler können auch von PCOS selbst gemeldet werden.

7.2 ERLÄUTERUNG DER FEHLER-CODES (BASIC-FEHLER)

ERROR-Nr.	BEDEUTUNG
1	Bei der Ausführung wurde eine Anweisung **NEXT** erreicht, ohne daß zuvor eine zugehörende **FOR**-Anweisung ausgeführt wurde.
2	Bei der Ausführung wurde in einer Programmzeile ein Syntaxfehler erkannt. Das System verzweigt in den Edit-Mode und zeigt sofort die entsprechende Zeile (Zeilennummer) an. Der Syntaxfehler kann mit den Möglichkeiten des Edit-Modes behoben werden. Sollen vor der Behebung noch Variablenwerte geprüft werden, so muß der Edit-Mode mit **Q** verlassen werden, da andernfalls die Variableninhalte gelöscht werden. Wurde das Programm mit **SAVE** und Parameter **P** gespeichert so erfolgt zusätzlich die Meldung "Illegal function call".
3	Es wurde während der Ausführung eine Anweisung **RETURN** erreicht, ohne daß der Rücksprung aus einem Unterprogramm möglich ist.
4	Es wird versucht, mehr Daten aus **DATA**-Anweisungen zu lesen als verfügbar sind.
5	Unzulässiger Aufruf einer Funktion. Dies kann durch falsche Angabe der Parameter oder durch Ausführung von Anweisungen oder Befehlen, die für ein geschütztes Programm unzulässig sind, verursacht sein.
6	Es wird eine zu große Zahl erzeugt. Es wird die größtmöglichste Zahl gesetzt und mit dieser Zahl unmittelbar fortgesetzt.
7	Es tritt ein Überlauf im Speicher auf. Das kann durch zu große Datenmengen, zu viele offene Rücksprungadressen oder ein zu großes Programm verursacht werden.
8	Es wird versucht, eine nicht vorhandene Programmzeile anzusprechen oder ein Datenfile mit **LOAD** in den Arbeitsspeicher zu laden.
9	Es wird versucht, ein Arralelement mit einem unzulässigen Index anzusprechen. Entweder liegt der Index über 10 und ist nicht über eine **DIM**-Anweisung als zulässiger Index festgelegt oder der Array ist mit **ERASE** aufgehoben worden, oder es liegt die Anweisung **OPTION BASE** 1 zugrunde, obwohl ein Index Ø angesprochen wird. Der Fehler kann auch auftreten, ohne daß einer dieser Fälle vorliegt. Nach Einfügen einer Anweisungszeile (z.B. **REM** xxx) direkt vor der den Fehler verursachenden Zeile wird der Fehler nicht mehr auftreten.

10 Ein bereits dimensionierter Array soll zum zweiten Male dimensioniert werden. Entweder soll eine zweite **DIM**-Anweisung für einen Array ausgeführt werden oder die **DIM**-Anweisung steht erst nach der ersten Verwendung des Arrays. Der Fehler kann auch auftreten, ohne daß einer dieser Fälle vorliegt. Nach Einfügen einer Anweisungszeile (z.B. **REM** xxx) direkt vor der den Fehler verursachenden Zeile wird der Fehler nicht mehr auftreten.

11 Es soll durch Ø dividiert werden. Es wird die größte darstellbare Zahl angenommen (abhängig vom Typ des Operanden) und unmittelbar damit weitergerechnet.

12 Es wird versucht, eine Anweisung im Direkt-Mode auszuführen, die nur in Programmen verwendet werden kann (z.B. **DEF**).

13 Es wird versucht, einem String einen numerischen Wert oder einer numerischen Variablen einen String zuzuweisen oder unterschiedliche Typen zu vergleichen oder falsche Typen als Parameter einer Anweisung zu verwenden.

14 Die Summe der Längen der verwendeten Strings übersteigt den dafür benötigten Speicherbereich.

15 Durch Ausführen der Operation würde der String mehr als 256 Zeichen lang werden.

16 Bei Ausführung der Stringoperation wird eine zu tiefe Schachtelung des Stacks erreicht.

17 Es wurde **CONT** eingegeben, es ist jedoch die Information über das Programm verloren. (Z.B., wenn nach einem Syntaxfehler durch den Edit-Mode der Stack gelöscht wurde.)

18 Es wird eine selbstdefinierte Funktion aufgerufen, die nicht zuvor mit **DEF FN** vereinbart wurde.

19 Es wird aufgrund eines Fehlers in eine Fehlerbehandlungsroutine gesprungen und kein **RESUME** erreicht.

20 Es soll eine **RESUME**-Anweisung ausgeführt werden, ohne daß zuvor aufgrund eines Fehlers in die Fehlerbehandlungsroutine gesprungen wurde.

21 Es wurde ein Fehler erzeugt, der keiner Fehlermeldung entspricht und ohne daß eine Fehlerbehandlungsroutine auszuführen ist.

22	Es fehlt ein Operand nach einem Operator.
23	Es wurde versucht, ein Random-File mit **LOAD** in den Arbeitsspeicher zu laden.
26	Es soll eine **FOR**-Anweisung ausgeführt werden, ohne daß ein zugehörendes **NEXT** gefunden wird. Die **NEXT**-Anweisung wird unmittelbar bei der Ausführung von **FOR** gesucht.
29	Es soll eine **WHILE**-Anweisung ausgeführt werden, ohne daß ein zugehörendes **WEND** vorhanden ist. Die zu **WHILE** gehörende **WEND**-Anweisung wird unmittelbar bei der Ausführung von **WHILE** gesucht.
30	Es soll eine **WEND**-Anweisung ausgeführt werden, ohne daß zuvor eine **WHILE**-Anweisung ausgeführt wurde.
31-34	Fehlermeldungen, die sich auf die Programmierung der Schnittstelle IEEE488 beziehen.
35	Es wird ein Window angesprochen, das noch nicht eröffnet wurde.
36	Es soll ein Window eröffnet werden, was nicht möglich ist, weil: - entweder in **sb** eine geringere Anzahl vereinbart wurde - dieses Window bereits existiert - die Parameter fehlerhaft sind (z.B. die Teilung außerhalb des aktiven Windows erfolgen müßte)
37	In einer Anweisung **LINE**, **CIRCLE** oder **PUT %** wird als letzter Parameter ein unzulässiger Operand eingegeben.
38	Ein Parameter überschreitet die zulässigen Grenzen. Diese Meldung tritt auch auf, wenn der in **GET %** angegebebene Array nicht genügend Platz für die Speicherung des Bildes bietet.
39	Es wird ein Array mit mehr Dimensionen angegeben, als für die Anweisung zulässig ist.
40	Während der Ausführung von **EXEC** oder **CALL** wurde ein PCOS-Fehler gemeldet.

50 Für ein Random-File wird versucht, dem Random-File-(**FIELD**)-Puffer mehr Zeichen zuzuweisen, als beim Öffnen des Files durch Recordlänge vereinbart wurde.

51 Nicht-spezifizierte Meldung.

52 Eine Anweisung versucht ein File anzusprechen, das nicht geöffnet ist, oder der Wert für "Filenummer" liegt außerhalb des gültigen Bereiches.

53 Es wird kein File mit dem verlangten Namen gefunden. (Falsche Schreibweise des Filenamens, oder falsche Diskette angesprochen oder eingelegt.)

54 Es wird versucht, eine Anweisung für ein File eines anderen Types auszuführen. (Z.B. **GET** für sequentielles File) oder es ist in **OPEN** ein falscher Parameter angeführt.

55 Es wird versucht, ein bereits offenes File erneut zu öffnen oder versucht, ein offenes File mit **KILL** zu löschen.

57 Beim Zugriff auf eine Diskette tritt ein Übertragungsfehler auf. Dieser Fehler weist auf beschädigte Disketten hin.

58 Es wird versucht, einem File einen Namen zu geben, der auf der Diskette bereits existiert, (Befehl **NAME**).

61 Die Diskette ist voll. Es sind weniger Sektoren verfügbar als aufgrund des gültigen Extent-Parameters im PCOS-Befehl **ss** in einem Zuge angelegt werden.

62 Es wird versucht, mehr Daten zu lesen als im File verfügbar sind.

63 Die Angabe der Record-Nummer für ein Random-File liegt außerhalb des gültigen Bereiches.

64 Fehlerhaftes Format für einen Filenamen.

66 Die Meldung tritt auf, wenn im Speicher Programmzeilen ohne Zeilennummer auftreten (z.B. nach Laden einer Assembler-Routine oder eines sequentiellen Datenfiles in den Arbeitsspeicher mit **LOAD** und nachfolgendem **RUN**).

67	Es wird versucht, mehr Files anzusprechen, als im entsprechenden Parameter im PCOS-Befehl **sb** vereinbart wurde.
68	Nicht-spezifizierte Meldung.
69	Bei der Diskettenspezifikation eines Filenamens wurde die Diskettenbezeichnung einer Diskette eingegeben, die nicht einliegt.
70	Fehler beim Umbenennen einer Diskette.
71	Bei der Diskettenspezifikation eines Filenamens wurde eine nicht zulässige Stationsbezeichnung angegeben.
72	Bei der Diskettenspezifikation eines Filenamens wurde das Disketten-Password nicht oder unrichtig angegeben.
73	Bei der Angabe einer Filespezifikation wurde das File-Password nicht oder unrichtig angegeben.
74	Während auf einer Diskette ein File offen ist, wurde die betreffende Diskette ausgetauscht. Nachdem die richtige Diskette wieder eingelegt ist, kann weitergearbeitet werden.
75	Das File, das zum Schreiben angesprochen wurde, ist durch PCOS-Befehl **fw** mit Schreibschutz versehen worden. Der Fehler kann auch auftreten, wenn einer Systemdiskette ein Disketten-Password zugeordnet wurde und dieses nicht korrekt angegeben wird, wenn eine Routine abgearbeitet werden soll.
76	Beim Aufruf einer PCOS-Routine durch **CALL** oder **EXEC** wurde der n-te Parameter nicht korrekt angegeben.
77	Beim Aufruf einer PCOS-Routine durch **CALL** oder **EXEC** wurden zu viele Parameter angegeben.
78	Für die File-Nummer, unter der eine Lese- oder Schreiboperation durchgeführt werden soll, ist keine **OPEN**-Anweisung mehr gültig.

PCOS - FEHLERMELDUNGEN

ERROR-CODE	BEDEUTUNG	ERLÄUTERUNG
7	exceeded memory limit	es wurde ein Programm aufgerufen, das für den zur Verfügung stehenden Speicherplatz zu groß ist; kann auftreten, weil ein PCOS-Befehl die dafür vorgesehene Segmentgrenze des Arbeitsspeichers überschreitet
13	bad data type	Der Typ eines Parameters ist nicht zulässig, z.B. keine Integer-Variable bei **CALL "lt"**
53	file not found	
54	bad file open mode	
55	file already open	
57	disk I/O-error	
58	file already exists	
60	disk has not been initialized	die angesprochene Diskette ist noch nicht mit **vf** formatiert worden
61	disk is full	
62	eof hit unexpectedly	
63	bad record number	
64	bad file name	
71	volume name not found	eine Diskette mit der angegebenen Diskettenbezeichnung wird nicht gefunden
72	name already exists	beim Versuch, ein File mit **fr** umzubenennen, wird festgestellt, daß die angegebene Filebezeichnung bereits existiert

76	match failed	das angegebene Password ist nicht richtig
77	illegal disk charge	während der Verwendung eines Files wurde die betreffende Diskette ausgetauscht
78	file is write protected	es wurde versucht, ein File mit File-Schreibschutz zu beschreiben
79	copy protected file	es wurde versucht, ein File zu kopieren, das mit Kopierschutz versehen ist
90	error in parameter	einer oder mehrere der angeführten Parameter enthalten unzulässige Werte
91	too many parameters	zu viele Parameter
92	command not found	unzulässiges Schlüsselwort oder Befehl nicht gefunden (z.B., weil benötigte Systemdiskette nicht einliegt)
96	file not open	File ist nicht offen
99	bad load file	Betriebssystem wird nicht gefunden
101	bad time or date	falscher Parameter 'Datum' oder 'Uhrzeit'
106	function key already defined	es wurde versucht, eine Taste mit **pk** zu belegen, deren Default-Belegung bereits geändert wurde
108	call user error	Fehler beim Aufruf einer Anwenderroutine in Assembler
110	time out error	die angesprochene Einheit ist nicht aktiviert (z.B. der Drucker auf **LOCAL**)
111	invalid device name	der angegebene Name der Einheit ist unkorrekt (z.B. weil unzulässige Zeichen enthalten sind)

Sind keine Erläuterungen angegeben, gelten die in der Liste der BASIC-Fehlermeldungen aufgeführten Bedeutungen.

ANHANG I:

Deutscher Zeichensatz M 20 (in () = Sonderfunktionstasten)

ISO-CODE-TABELLE

Adressierung: erst Spalte, dann Zeile

Hexadezimal (&H)	Dual	0	1	2	3	4	5	6	7
		0000	0001	0010	0011	0100	0101	0110	0111
0	0000	0 NUL	16 DLE	32 SP	48 0	64 §	80 P	96 `	112 p
1	0001	1 SOH -	17 DC1	33 !	49 1	65 A	81 Q	97 a	113 q
2	0010	2 STX	18 DC2	34 "	50 2	66 B	82 R	98 b	114 r
3	0011	(C) 3 ETX	19 DC3	35 #	51 3	67 C	83 S	99 c	115 s
4	0100	4 EOT	20 DC4	36 $	52 4	68 D	84 T	100 d	116 t
5	0101	5 ENQ	21 NAK	37 %	53 5	69 E	85 U	101 e	117 u
6	0110	6 ACK	22 SYN	38 &	54 6	70 F	86 V	102 f	118 v
7	0111	(C) 7 BEL	23 ETB	39 '	55 7	71 G	87 W	103 g	119 w
8	1000	(I) 8 BS	24 CAN	40 (	56 8	72 H	88 X	104 h	120 x
9	1001	9 HT	25 EM	41)	57 9	73 I	89 Y	105 i	121 y
A	1010	10 LF	26 SUB	42 *	58 :	74 J	90 Z	106 j	122 z
B	1011	11 VT	27 ESC	43 +	59 ;	75 K	91 Ä	107 k	123 ä
C	1100	12 FF	28 FS	44 ,	60 <	76 L	92 Ö	108 l	124 ö
D	1101	13 CR	29 CS	45 -	61 =	77 M	93 Ü	109 m	125 ü
E	1110	14 SO	30 RS	46 .	62 >	78 N	94 ^	110 n	126 ß
F	1111	15 SI	31 US	47 /	63 ?	79 O	95 _	111 o	127 DEL

dezimal

dez. Code:

↓ = 154

← = 155

HOME = 156

→ = 157

↑ = 158

ANHANG II

1. MATRIXDRUCKER PR 1450

Der Befehlssatz des **PR 1450** läßt sich in drei Gruppen einteilen:

1. ausführende Befehle
2. Löschbefehle
3. definierende Befehle

In nachfolgenden Tabellen werden die Funktion der Befehle, die Bezeichnung der ISO-Zeichen und die dazugehörigen dezimalen ISO-Codes aufgeführt.

1. Ausführende Befehle

ISO-Zeichen	Funktion	Dezimalwert
LF (Line Feed)	Druckt den Pufferinhalt aus und macht eine Zeilenschaltung	10
VT (Vertical Tabulation)	Sprung zum nächsten vertikalen Tabulationsstop. Der Pufferinhalt wird ausgedruckt. Nach dem letzten vertikalen Tabulationsstop erfolgt ein Sprung zur ersten Druckzeile der neuen Seite.	11
FF (Form Feed)	Druckt den Pufferinhalt aus und bringt das Formular in Grundstellung	12
CR (Carriage Return)	Rücklauf des Druckkopfes zum linken Rand mit Ausdruck des Pufferinhaltes	13
ESC I nnn	Es werden nnn Zeilenschaltungen ausgeführt. Die Anzahl Zeilenschaltungen nnn hat einen Wert zwischen 001 und 256	27,73,N,N,N

2. Löschbefehle

ISO-Zeichen	Funktion	Dezimalwert
DEL (Delete	Löscht den Pufferinhalt (vorgewählte Steuerungen werden nicht geändert)	127
ESC 0 (Escape 0)	General Reset; Pufferinhalt, Tabulation und definierende Befehle werden gelöscht; Drucker befindet sich im gleichen Zustand wie direkt nach dem Einschalten	27,48

3. Definierende Befehle

ISO-Zeichen	Funktion	Dezimalwert
ESC 3	Fettdruck; Druck in doppelter Breite (bezogen auf die Vorwahl für Anzahl Zeichen/Zoll) ab der momentanen Druckposition, bis ESC 4 erkannt wird	27,51
ESC 4	Rückkehr zum Normaldruck (Aufhebung von ESC 3)	27,52
ESC <	Vorwahl für Druck mit 10 Zeichen/Zoll	27,60
ESC =	Vorwahl für Druck mit 12 Zeichen/Zoll	27,61
ESC >	Vorwahl für Druck und 16 Zeichen/Zoll	27,62
ESC Q nnn [, nnn] ... ESC Z	Definiert die vertikale Tabulation. Es können 30 Tabulationsstops definiert werden. Der erste Wert bestimmt das Seitenformat (Gesamtanzahl der Zeilen pro Seite) und dadurch die Formatsteuerung. Der ersten Druckzeile ist der Wert 0 zugeordnet. Die Werte für die einzelnen Tabulationlationsstops dürfen nicht den Wert 000 haben und müssen innerhalb der definierten Seite liegen. Die Gesamtanzahl der Zeilen pro Seite kann max. 255 betragen.	27,81,N,N,N,[59,N N,N] ...,27,90
ESC Ä nnn	Definition des Zeichensatzes. nnn gibt die Nummer des Zeichensatzes an. Im folgenden sind für einige Zeichensätze das dazugehörige nnn aufgeführt: 010 - International 020 - Deutschland 090 - Schweiz 110 - USASCII Bei der Installation des PR 1450 wird ein Standard-Zeichensatz festgelegt, der nach jedem Einschalten des Druckers zur Verfügung steht.	27,91,N,N,N

BEMERKUNGEN: - n bedeutet eine Ziffer "0",...,"9" und
N den dazugehörigen dezimalen Code 48,...,57

ANHANG IV

PCOS-Befehle (Überblick)

Befehl	mnemonischer Name	Wirkung	resident (R)
ba	basic	Aktivierung des BASIC-Interpreters; Übergang in die BASIC-Ebene	
co	commands	Liste der PCOS-Befehle	
Befehl?	command	Abruf der Informationen über einen speziellen PCOS-Befehl	
eError-Code	error-code	Beschreibung eines Fehler-Codes	
er	errors	Liste der Fehler-Codes und ihrer Bedeutung	
fc	fcopy	Kopieren und/oder Neustruktuierung eines Files auf Diskette	
fd	fdepass	Aufheben eines File-Passwords	
fl	flist	Listen eines ASCII-Files	R
fn	fnew	Reservieren von Platz für ein File	
fp	fpass	Vergabe oder Änderung eines File-Passwords	
fu	funprot	Aufhebung eines File-Schreibschutzes	
fw	fwprot	Vergabe eines File-Schreibschutzes	
he	help	Auswahlmöglichkeit: PCOS-Überblick, Liste der Fehler-Codes oder Liste der PCOS-Befehle	
ie	IEEE488	Laden der IEEE-Routinen	
la	label	Plotten von Zeichen(folgen)	
lt	lterm	Bestimmung der zuletzt betätigten Eingabeabschlußtaste	R
mi	mi	Ausführung eines Z80Ø1-Maschinen-Befehls	

pk	pkey	Belegung von Tasten	
pl	pload	Ein PCOS-Befehl wird im Arbeits-speicher resident gemacht	R
ps	psave	Abspeicherung der aktuellen Betriebs-systemkonfiguration auf einer leeren Diskette	R
sb	sbasic	Festlegung von Default-Werten für den Basic-Interpreter	R
sf	sform	Festlegung von Default-Werten für den Drucker	
sp	sprint	Abbildung des Inhalts des Bild-schirms oder eines Windows auf dem Drucker	
ss	ssystem	Festlegung von Default-Werten für das Betriebssystem	R
vc	vcopy	Physisches Kopieren von Disketten-inhalten	
vd	vdepass	Aufhebung eines Disketten-Passwords	
vf	vformat	Formatierung einer Diskette	
vl	vlist	Ausgabe des Inhaltsverzeichnis einer Diskette	
vn	vnew	Löschen des Inhalts einer Diskette	
vp	vpass	Vergabe oder Änderung eines Disketten-Passwords	
vr	vrename	Vergabe oder Änderung einer Diskettenbezeichnung	

9.1 ALPHABETISCHES VERZEICHNIS DER BEFEHLE, ANWEISUNGEN UND FUNKTIONEN

NAME	ART	ZWECK/FUNKTION
ABS	FUNKTION	Liefert den Absolutwert (Betrag) eines numerischen Werts
ASC	FUNKTION	Liefert den ISO-Code eines Zeichens
ATN	FUNKTION	Liefert von einem numerischen Wert seinen Arcustangens im Bogenmaß
AUTO	BEFEHL	Automatische Erzeugung von Zeilennummern für einzugebende Anweisungen eines Programmes
CALL	ANWEISUNG	Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine
CDBL	FUNKTION	Konvertiert einen numerischen Wert in doppelte Genauigkeit
CHAIN	ANWEISUNG	Aufruf eines Programmes, eventuell mit Übergabe von Variablen vom gegenwärtigen Programm an das aufgerufene sowie ggf. Laden von Overlays und/oder Löschen von Programmzeilen
CHR$	FUNKTION	Liefert dasjenige Zeichen, das durch einen ISO-Code bestimmt ist
CINT	FUNKTION	Konvertiert einen numerischen Wert in einen ganzzahligen (Integer-) Wert
CIRCLE	FUNKTION	Erlaubt das Zeichnen von Kreisen oder Ellipsen
CLEAR	ANWEISUNG	Löschen des Datenbereiches und der Bitmap für den Bildschirm im Arbeitsspeicher; Setzen des gesamten Bildschirms auf Default-Hintergrundfarbe; Aufhebung aller Window-Vereinbarungen; ggf. Definition der logischen Größe des Arbeitsspeichers und/oder des für Stack-Speicher zu reservierenden Platzes im Arbeitsspeicher.
CLOSE	ANWEISUNG	Schließen von einem, mehreren oder allen externen Datenfiles

CLOSE WINDOW	ANWEISUNG	Auflösen von Windows
CLS	ANWEISUNG	Löscht den Inhalt des aktiven oder des angegebenen Windows
COLOR	ANWEISUNG	Auswahl oder Neufestlegung von Vorder- und Hintergrundfarbe
COLOR=	ANWEISUNG	Auswahl von vier aus acht Farben für das Arbeiten mit dem Farbschirm
COMMON	ANWEISUNG	Definition der Variablen in einem BASIC-Programm, deren Werte bei der Verkettung von Programmen erhalten bleiben
CONT	BEFEHL	Fortsetzung der Programmausführung nach einer Unterbrechung
COS	FUNKTION	Liefert den Cosinus eines numerischen Werts im Bogenmaß
CSNG	FUNKTION	Konvertiert einen numerischen Wert in einfache Genauigkeit
CURSOR	ANWEISUNG	Steuerung des Text- oder des Graphik-Cursors und ggf. Festlegung der Cursor-Eigenschaften
CVD	FUNKTION	Verwandlung der ersten 8 Bytes eines Strings in einen doppelt genauen numerischen Wert
CVI	FUNKTION	Verwandlung der ersten 2 Bytes eines Strings in einen Integer-Wert
CVS	FUNKTION	Verwandlung der ersten 4 Bytes eines Strings in einen einfach genauen numerischen Wert
DATA	ANWEISUNG	Erzeugung eines internen Files von Daten, die anschliessend den Variablen aus **READ**-Anweisungen zugewiesen werden
DATE$	SPEZIELLE VARIABLE	Dient zur Verwaltung des von der Echtzeituhr gesteuerten Datums
DEFDBL	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als doppelt genaue numerische Variable

DEF FN	ANWEISUNG	Definition einer numerischen oder einer Stringfunktion innerhalb einer BASIC-Zeile durch den Anwender
DEFINT	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als numerische Integer-Variablen
DEFSNG	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als einfach genaue numerische Variablen
DEFSTR	ANWEISUNG	Typvereinbarung aller Variablen mit bestimmten Anfangsbuchstaben als String-Variablen
DELETE	BEFEHL	Löschung einer oder mehrerer Programmzeilen im Arbeitsspeicher
DIM	ANWEISUNG	Festlegung der Dimension und der maximalen Werte der Indices von Arrays
DRAW	ANWEISUNG	Festlegung von Zeichenvorgängen
EDIT	BEFEHL	Ändern bestehender Programmzeilen eines Programms im Arbeitsspeicher
END	ANWEISUNG	Gibt das logische Ende eines Programms an
EOF	FUNKTION	Abfrage auf Ende eines sequentiellen Files beim Input von sequentiellen Files
ERASE	ANWEISUNG	Aufheben der Dimensionierung eines Arrays
ERL	RESERVIERTE VARIABLE	Enthält die Zeilennr. derjenigen Programmzeile, in der der zuletzt vom System entdeckte Fehler auftrat bzw. die Zeilennr., in der die letzte **ERROR**-Anweisung auftrat
ERR	RESERVIERTE VARIABLE	Enthält den Fehler-Code des zuletzt vom System entdeckten Fehlers bzw. nach einer **ERROR**-Anweisung den dadurch zugewiesenen Wert
ERROR	ANWEISUNG	Zuweisung eines Fehler-Codes auf die reservierte Variable **ERR** oder Simulation eines "Fehlers" durch den Anwender

EXEC	ANWEISUNG	Aufruf und Abarbeitung eines PCOS-Befehls oder einer selbstdefinierten Assembler-Routine
EXP	FUNKTION	Berechnung einer Potenz der Eulerschen Zahl e
FIELD	ANWEISUNG	Vereinbarung der Feldlängen der Records eines bestimmten Random-Files, Festlegung der Variablennamen und der Position im Random-File-Puffer sowie Bereitstellung des Random-File-Puffers für folgende **GET**- oder **PUT**-Anweisungen
FILES	BEFEHL	Ausgabe des Inhaltsverzeichnisses einer Diskette oder der Angaben über ein einzelnes File
FIX	FUNKTION	Ermittlung des ganzzahligen Teils eines numerischen Werts
FN	FUNKTION	Eine vom Anwender definierte Funktion wird aufgerufen und abgearbeitet
FOR	ANWEISUNG	Startanweisung für die mehrmalige Ausführung von Anweisungen innerhalb einer zählergesteuerten Schleife
FRE	FUNKTION	Ermittlung des im Arbeitsspeicher noch freien Platzes (in Bytes) und ggf. Bereinigung des Arbeitsspeichers
GET	ANWEISUNG	Übertragen eines logischen Records von einem Random-File in den Random-File-Puffer ("Lesen")
GET %	ANWEISUNG	Bereiche eines Windows werden in einem Array gespeichert
GOSUB	ANWEISUNG	Bewirkt den Sprung zu einer bestimmten Anweisung, bei der ein Unterprogramm beginnt
GOTO	ANWEISUNG	Sprung zu einer bestimmten Zeile des Programms

HEX$	FUNKTION	Liefert den hexadezimalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung
IF...THEN ...ELSE	ANWEISUNG	Bedingte Verzweigung(en) in einem Programm
INKEY$	FUNKTION	Überprüfung, ob eine Taste an der Tastatur betätigt wurde
INPUT	ANWEISUNG	Das Programm wartet auf Eingaben über die Tastatur und weist die Eingaben auf Variablen zu
INPUT#	ANWEISUNG	Lesen von Daten von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf Variablen
INPUT$	FUNKTION	Eine bestimmte Zeichenanzahl wird über Tastatur eingegeben oder von einem Datenfile eingelesen
INSTR	FUNKTION	Aufsuchen der Anfangsposition eines Teilstrings in einem String
INT	FUNKTION	Ermittlung derjenigen ganzen Zahl aus einem numerischen Wert, die als nächste auf der Zahlenskala links von dem angegebenen num. Wert liegt ("Gauß-Funktion")
KILL	BEFEHL	Löschen eines gespeicherten Files auf Diskette
LEFT$	FUNKTION	Einem String wird von links kommend ein Teilstring entnommen
LEN	FUNKTION	Bestimmung der Länge eines String-Ausdruckes
LET...=	ANWEISUNG	Zuweisung des Wertes eines Ausdruckes an eine Variable
LINE	ANWEISUNG	Zeichnen von Linien oder Rechtecken oder mit einer Farbe gefüllten Rechtecken
LINE INPUT	ANWEISUNG	Eingabe einer ganzen Kette von Zeichen ohne Trennzeichen und Zuweisung auf eine Stringvariable über Tastatur

LINE INPUT#	ANWEISUNG	Lesen einer Folge von Zeichen von einem sequentiellen File oder aus einem Random-File-Puffer und Zuweisung auf eine Stringvariable
LIST	BEFEHL	Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programmes am Bildschirm
LLIST	BEFEHL	Ausgabe einer oder mehrerer Zeilen eines im Arbeitsspeicher befindlichen Programmes auf dem Drucker
LOAD	BEFEHL	Laden eines Programmes von Diskette in den Arbeitsspeicher
LOC	FUNKTION	Gibt die Anzahl gelesener bzw. geschriebener Sektoren bei sequentiellen Files oder die letzte angesprochene Record-Nr. bei Random-Files an
LOF	FUNKTION	Liefert für sequentielle Files die aktuelle Pointer-Position; bei Random-Files kann damit ermittelt werden, in welchem Sektor des Files der höchste beschriebene Record steht
LOG	FUNKTION	Liefert den natürlichen Logarithmus zur Basis e eines numerischen Werts
LPOS	FUNKTION	Liefert die aktuelle Position des Druckpointers innerhalb des Druckpuffers
LPRINT	ANWEISUNG	Ausgabe von Zahlen und/oder Strings auf dem Drucker
LPRINT USING	ANWEISUNG	Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Drucker
LSET	ANWEISUNG	Zuweisung eines linksbündigen, rechts eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable

MERGE	BEFEHL	Kombination eines sich im Arbeitsspeicher befindlichen Programmes mit einem File, das im ASCII-Format (Parameter **A**) bei **SAVE** abgespeichert ist
MID$...=	ANWEISUNG	Ersetzen eines String-Bestandteils durch einen anderen String
MID$	FUNKTION	Einem String wird ab einer bestimmten Position ein Teilstring entnommen
MKD$	FUNKTION	Konvertierung eines doppelt genauen numerischen Wertes in einen 8 Zeichen (Bytes) langen String
MKI$	FUNKTION	Konvertierung eines Integer-Wertes in einen 2 Zeichen (Bytes) langen String
MKS$	FUNKTION	Konvertierung eines einfach genauen numerischen Wertes in einen 4 Zeichen (Bytes) langen String
NAME	BEFEHL	Änderung des Namens eines Programmes oder Datenfiles auf Diskette
NEW	BEFEHL	Löschen des Arbeitsspeicher-Inhalts; Vorbereitung zur Eingabe eines BASIC-Programmes oder mit Zeilen-Nr. versehenen Textes
NEXT	ANWEISUNG	Endanweisung einer oder mehrerer **FOR-NEXT** Schleifen
NULL	ANWEISUNG	Verzögert die Ausführung von Anweisungen, die Ausgabeeinheiten betreffen
OCT$	FUNKTION	Liefert den oktalen Wert einer dezimal dargestellten Zahl; das Ergebnis steht in Form eines Strings zur Verfügung
ON ERROR GOTO	ANWEISUNG	Fehlererkennung und Verzweigung zu einer Fehlerbehandlungsroutine
ON ... GOSUB	ANWEISUNG	Sprung in ein Unterprogramm; das Sprungziel ist abhängig vom Wert eines num. Ausdruckes
ON...GOTO	ANWEISUNG	Sprung in eine bestimmte Programmzeile; das Sprungziel ist abhängig vom Wert eines num. Ausdruckes

OPEN	ANWEISUNG	Öffnen eines sequentiellen oder Random-Files für Zugriffe evtl. auch Neuanlage eines nicht vorhandenen Files bzw. Vorbereitung zur automatischen Verlängerung eines vorhandenen Files
OPTION BASE	ANWEISUNG	Festlegung des kleinsten möglichen Index-Wertes aller Arrays
PAINT	ANWEISUNG	Ausfüllen eines ganzen oder eines Teils eines Windows
POINT	FUNKTION	Ermittlung der Farbe in einem Koordinatenpunkt
POS	FUNKTION	Liefert die aktuelle Spalten- oder Zeilen-Position des Bildschirm-Cursors
PRESET	ANWEISUNG	Zeichnen eines Elementarpunktes in der Hintergrundfarbe
PRINT	ANWEISUNG	Druck von Zahlen und Strings im Standardformat auf dem Bildschirm
PRINT#	ANWEISUNG	Schreiben von Daten auf ein externes sequentielles File; Trennzeichen werden nicht automatisch auf das File geschrieben; am Ende wird ggf. ein CR und ein LF auf das File geschrieben
PRINT USING	ANWEISUNG	Ausgabe von formatierten Daten (numerisch und/oder String) auf dem Bildschirm
PRINT#...USING	ANWEISUNG	Schreiben von formatierten Daten (numerisch und/oder String) auf ein sequentielles File
PSET	ANWEISUNG	Darstellung eines Elementarpunktes in vorgewählter Farbe
PUT	ANWEISUNG	Übertragen eines logischen Records von einem Random-File-Puffer in das zugehörige Random-File ("Schreiben")
PUT %	ANWEISUNG	Ein in einem Array gespeichertes Bild wird auf dem Bildschirm dargestellt

RANDOMIZE	ANWEISUNG	Festlegung bzw. Auswahl des Anfangswertes und der Folge von später aufzurufenden Zufallszahlen
READ	ANWEISUNG	Zuweisung von Werten aus dem internen File an Variablen; die Werte wurden vorher in einer internen Tabelle mit Hilfe von **DATA**-Anweisungen generiert
REM	ANWEISUNG	Einfügung von Erläuterungen in ein Programm
RENUM	BEFEHL	Umnumerierung der Zeilen eines im Arbeitsspeicher vohandenen Programms
RESTORE	ANWEISUNG	Setzen des Pointers des internen Files auf das erste Element eine **DATA**-Anweisung
RESUME	ANWEISUNG	Fortsetzung eines Programms nach einer Verzweigung in eine Fehlerbehandlungsroutine
RETURN	ANWEISUNG	Logische Ende eines Unterprogrammes; Rücksprung zu der unmittelbar auf das aufrufende **GOSUB** oder **ON...GOSUB** folgenden Anweisung
RIGHT$	FUNKTION	Einem String wird von rechts kommend ein Teilstring entnommen
RND	FUNKTION	Liefert eine Zufallszahl zwischen Ø und 1
RSET	ANWEISUNG	Zuweisung eines rechtsbündigen, links eventuell mit Blanks aufgefüllten Strings auf die Feldvariable eines Random-File-Puffers (zur Vorbereitung einer **PUT**-Anweisung) oder auf eine Stringvariable
RUN	BEFEHL	Start der Programmausführung
SAVE	BEFEHL	Abspeicherung eines BASIC-Programmes oder eines mit Zeilennummern versehenen Textes (ASCII-Files) auf Diskette
SCALE	ANWEISUNG	Festlegung des Koordinatenssystems innerhalb eines Windows
SCALEX	FUNKTION	Gibt an, der wievielte Elementarpunkt des aktiven Windows durch eine Anwenderbezogene X-Koordinate angesprochen wird

SCALEY	FUNKTION	Gibt an, der wievielte Elementarpunkt des aktiven Windows durch eine Anwender-bezogene Y-Koordinate angesprochen wird
SGN	FUNKTION	gibt Aufschluß über das Vorzeichen eines numerischen Wertes ("Signum-Funktion")
SIN	FUNKTION	Liefert den Sinus eines numerischen Wertes im Bogenmaß
SPACE$	FUNKTION	Liefert einen String aus Blanks
SPC	FUNKTION	Ausgabe einer bestimmten Anzahl von Leerzeichen (Blanks) auf dem Bildschirm oder auf dem Drucker
SQR	FUNKTION	Liefert die positive Quadratwurzel eines numerischen Wertes
STOP	ANWEISUNG	Unterbrechen der Programmausführung
STR$	FUNKTION	Bildung eines Strings aus einem numerischen Ausdruck
STRING$	FUNKTION	Erzeugung eines Strings aus vorgegebenem Zeichen mit vorgegebener Länge
SWAP	ANWEISUNG	Vertauschen des Inhalts zweier Variablen vom gleichen Typ
SYSTEM	BEFEHL	Aufruf einer PCOS-Ebene
TAB	FUNKTION	Positionieren auf eine gewünschte Spalte am Bildschirm oder Drucker
TAN	FUNKTION	Liefert den Tangens eines numerischen Werts im Bogenmaß
TIME$	SPEZIELLE VARIABLE	Dient zur Verwaltung der von der Echtzeituhr geführten Uhrzeit
TROFF	BEFEHL	Aufhebung der Wirkung von **TRON**
TRON	BEFEHL	Ausdruck der Zeilennummern während des Programmablaufs

USING	ANWEISUNGS-BESTANDTEIL	Bestimmung des Formates, in welchem die Ausgabegrößen (Zahlen und/oder Strings) dargestellt werden
VAL	FUNKTION	Aus einem String, der am Anfang eine Zahlendarstellung enthält, wird diese in den numerischen Wert umgewandelt
VARPTR	FUNKTION	Liefert die Byte-Adresse einer Variablen im Arbeitsspeicher oder die Start-Adresse des I/O-Puffers, der der Filenr. eines sequentiellen Files zugeordnet ist bzw. die Adresse des Random-File-Puffers für ein unter einer Filenr. geöffnetes Random-File
WEND	ANWEISUNG	Endanweisung einer **WHILE-WEND**-Schleife
WHILE	ANWEISUNG	Start-Anweisung einer Schleife, die solange durchgeführt wird, wie eine Bedingung erfüllt ist
WIDTH	ANWEISUNG	Festlegung der Anzahl Zeichen pro Ausgabezeile für Drucker oder Bildschirm
WINDOW	FUNKTION	Definition der Lage und Größe eines neuen Windows ("Anlegen" bzw. "Eröffnen") und/oder Neu-Festlegung der Anzahl Zeichen pro Zeile und/oder der Anzahl Zeilen in einem Window
WINDOW %	ANWEISUNG	Anwählen ("Aktivieren") eines eröffneten Windows
WRITE	ANWEISUNG	Ausgabe von Daten am Bildschirm; als Trennzeichen werden automatisch Kommas ausgegeben; Strings werden automatisch in Anführungszeichen eingeschlossen ausgegeben
WRITE#	ANWEISUNG	Schreiben von Daten auf ein externes sequentielles File; als Trennzeichen wird automatisch ein Komma auf das File geschrieben; Strings werden automatisch in Anführungszeichen eingeschlossen auf das File gebracht; am Ende wird ein CR und ein LF geschrieben

Das Handbuch dient der Information, sein Inhalt ist ohne ausdrückliche schriftliche Vereinbarung nicht Vertragsgegenstand. Technische Änderungen behalten wir uns vor. Die angegebenen Daten sind lediglich Nominalwerte.

DEUTSCHE OLIVETTI DTS GMBH · FRANKFURT-NIEDERRAD · LYONER STRASSE 34