BILDUNGSCOMPUTER
r o b o t r o n A 5 1 0 5

Programmierhandbuch

VEB ROBOTRON-MESSELEKTRONIK >OTTO SCHÖN< DRESDEN
Lingnerallee 3, Postfach 211, Dresden, DDR-8012

Programmierhandbuch

BILDUNGSCOMPUTER
r o b o t r o n A 5 1 0 5

I N H A L T

12/88 b

VEB ROBOTRON-MESSELEKTRONIK >OTTO SCHÖN< DRESDEN
Lingnerallee 3, Postfach 211, Dresden, DDR-8012

0. Ausführliches Inhaltsverzeichnis

Diese Programmierhandbuch wurde vom nachstehenden Autorenkollektiv verfaßt:

Gesamtleitung

Dr. rer. nat. G. Keller — VEB Robotron-Meßelektronik "Otto Schön" Dresden

Autoren der Abschnitte

1; 2	Dr. rer. nat. G. Keller	
3	Dr. rer. nat. G. Keller Dr.-Ing. H.-J. Busch Dr.-Ing. J. Haase	
4	Dr. rer. nat. G. Keller	
5	Dr.-Ing. H.-J. Busch Dipl.-Ing. G. Gärtner	VEB Robotron-Meßelektronik "Otto Schön" Dresden
6	Dipl.-Math. K. Bernhardt	
7	Dipl.-Ing. M. Ludewig Dipl.-Ing. G. Lützner Dipl.-Math. G. Weitzmann	
8,9	Prof. Dr. S. Bohnsack	Pädagogische Hochschule Güstrow

Redaktion

Dipl.-Phys. E. Nebe
Ing. J. Ullmers

digitalisiert: Ulrich Zander, 10/2002 <zander@felix.sax.de>

1. Einleitung

Dieses Programmierhandbuch enthält eine ausführliche Beschreibung der Programmiersprache RBASIC des BILDUNGSCOMPUTERs robotron A 5105 bzw. des daraus abgeleiteten Kleincomputers sowie einige Informationen zum Betriebssystem dieser Rechner.

Der im ROM des Computers enthaltene BASIC-Interpreter (RBASIC) dient der Übersetzung (Interpretation) von BASIC-Anweisungen, die dem Menschen leicht verständlich sind und die er in den Computer eingibt, in solche, die der Zentralprozessor (CPU) des Computers versteht. Dieser RBASIC-Interpreter gliedert sich in zwei Teile, den sogenannten RBASIC-Grundumfang (innerhalb der 40 kBytes ROM des Computergrundgerätes) und die RBASIC-Disk-Erweiterung (16 kBytes ROM in der Diskettenspeichereinheit, die in der Grundversion des Kleincomputers nicht vorhanden ist).

Nach dem Einschalten des Rechners wird der RBASIC-Interpreter sofort gestartet, falls keine SCP-Systemdiskette eingelegt wird. Durch die realisierte Speicherverwaltung bietet er dann etwa 24 bis 28 kBytes Speicherplatz für Anwenderprogramme.

Wegen ihrer einfachen Erlernbarkeit ist die dialogorientierte Programmiersprache BASIC besonders für den Anfänger oder Gelegenheitsnutzer sehr zweckmäßig und deshalb international verbreitet. Aufgrund der ständigen (und noch nicht abgeschlossenen) Weiterentwicklung ist BASIC aber unterdessen auch für anspruchsvolle Aufgabenstellungen gut einsetzbar.

Mit Hilfe der Programmiersprache BASIC können sowohl einfache mathematische Operationen sofort ausgeführt werden, als auch Programme eingegeben, abgearbeitet und auf Magnetbandkassette oder Diskette gespeichert werden. Natürlich ist auch die Ausgabe von Programmen, Rechenergebnissen und anderen Informationen auf einen Drucker möglich. Durch BASIC-Anweisungen können Informationen von angeschlossenen Meßgeräten abgerufen werden und auch Steuersignale an entsprechende Geräte übertragen werden.

Besondere Vorzüge besitzt das RBASIC des Computers A 5105 durch die vielfältig unterstützten Möglichkeiten der textlichen und grafischen Bildausgabe, die die besonderen technischen Möglichkeiten diese Computers zur Geltung bringen. Es sei auch darauf hingewiesen, daß die Struktur des RBASIC verschiedene Erweiterungen erlaubt, so daß beispielsweise die einfache Programmierbarkeit eines angeschlossenen Plotters leicht gesichert werden kann.

RBASIC ist in seinem Aufbau (Anweisungs- und Kommandovorrat, syntaktische Struktur) dem weitverbreiteten Standard angeglichen, so daß eine Übertragung von RBASIC-Programmen zu oder von anderen Rechnern (z.B. PC 1715 und EC 1834) in der Regel leicht fallen wird, solange hardwarespezifische Programmelemente vermieden werden.

In der dialogorientierten Programmiersprache BASIC können Sie sowohl fertige Programme laden und abarbeiten, als auch eigene Programme schreiben (programmieren), testen und abarbeiten. Zur Unterstützung dieser Möglichkeiten enthält der RBASIC-Interpreter neben den für Programme notwendigen Anweisungen auch entsprechende (Steuer-)Kommandos. Dementsprechend kann man auch zwei typische Arbeitszustände unterscheiden:

- den Kommandomodus, in dem die Tastatureingabe von Kommandos durch den Nutzer erfolgt. Die Ausführung dieser Kommandos ist im wesentlichen Voraussetzung für die Programmabarbeitung bzw. beeinflußt diese (Programm starten, unterbrechen, fortsetzen, speichern, laden, anzeigen, ändern usw.).

- den Programmmodus, innerhalb dessen die Abarbeitung von Anweisungen des gestarteten BASIC-Programms stattfindet. Diese Anweisungen dienen der eigentlichen Problemlösung und werden interpretierend abgearbeitet, d. h. in vorgegebener Reihenfolge einzeln analysiert und unmittelbar ausgeführt.

Das vorliegende Programmierhandbuch vermittelt einen Überblick über die prinzipiellen Möglichkeiten, der Arbeitsweise des RBASICs, seines Kommando- und Anweisungsumfanges. Dabei werden nach einem einführenden Abschnitt 2, der zunächst wichtige Begriffe und notwendige Bedienerhandlungen erläutert, im 3. Abschnitt alle Grundanweisungen des RBASICs systematisch und ausführlich erläutert, wobei gleichzeitig Hinweise zu deren sinnvoller Nutzung vermittelt werden.

Die folgenden Abschnitte 4 bis 9 widmen sich jeweils einem speziellen Anwendungsgebiet und stellen die Möglichkeiten des RBASIC-Interpreters dazu systematisch dar.

Diese Abschnitte sind relativ unabhängig voneinander lesbar; sie setzen jedoch voraus, daß die wichtigsten der in den Abschnitten 2 und 3 erläuterten Grundlagen bekannt sind.

Für Zusatzinformationen steht außerdem ein Anhang zur Verfügung, in dem wichtige Tabellen und Übersichten zur Unterstützung des Programmierers zusammengefaßt sind.

Nachdem Sie den Bildungs- bzw. Kleincomputer unter Beachtung der Bedienungsanleitung ordnungsgemäß in Betrieb genommen haben (vgl. auch Bedienungsanleitung, Abschnitt 3), können Sie nun anhand des Programmierhandbuches mit der Arbeit im RBASIC beginnen.

Wir wünschen Ihnen dazu viel Spaß und Erfolg.

Hinweis:

- In der Grundversion des aus dem Computergrundgerät abgeleiteten Kleincomputers stehen die im Abschnitt 6 erläuterten Anweisungen zur Diskettennutzung nicht zur Verfügung. Alle anderen Kommandos und Anweisungen sind in vollem Umfang nutzbar.

- Die Anweisungen PLAY und SOUND zur Tonerzeugung sind in einem gesonderten Heft beschrieben.

2. Einführung in die Arbeit mit RBASIC

2.1. Nach dem Einschalten

Der Grundzustand

Das Starten des im ROM des Computers integrierten BASIC-Interpreters (RBASIC) ist in der Bedienungsanleitung, Abschnitt 3.3, ausführlich erläutert. Das Einschaltbild erscheint mit heller Schrift (ocker) auf dunklem Hintergrund (schwarz), je nachdem, ob Sie mit einem Schwarzweiß- oder Farbmonitor arbeiten.

```
RBASIC Version 2.0
robotron MKD 1988
xxxxx Bytes free
Ok
#

screen     auto       load"      list       run
```

Der Wert xxxxx gibt Ihnen an, wieviel Speicherplatz (in Bytes) für Ihre Programme und Daten zur Verfügung steht. Dieser Wert ist davon abhängig, welche Peripherie in die Arbeit mit RBASIC einbezogen wird, ob z.B. ein Diskettenlaufwerk angeschlossen ist oder nicht.

Mit

```
Ok
#
```

zeigt der RBASIC-Interpreter seine Arbeitsbereitschaft im sogenannten Kommandomodus an und erwartet nun eine Tastatureingabe (ein Kommando) von Ihnen.

Die Informationen in der letzten Bildschirmzeile zeigen die aktuellen Belegungen der Funktionstasten (PF-Tasten). Sie können diese Zeile zunächst unbeachtet lassen. Die Nutzung der PF-Tasten wird im Abschnitt 3.3 erläutert.

Zu erwähnen ist noch, daß der Bildausgabemodus so eingestellt wird, daß 25 Zeilen und 40 Spalten auf dem Bildschirm sichtbar sind. Dieser Modus wird auch als **SCREEN 0** bezeichnet.

Beenden der Arbeit im RBASIC

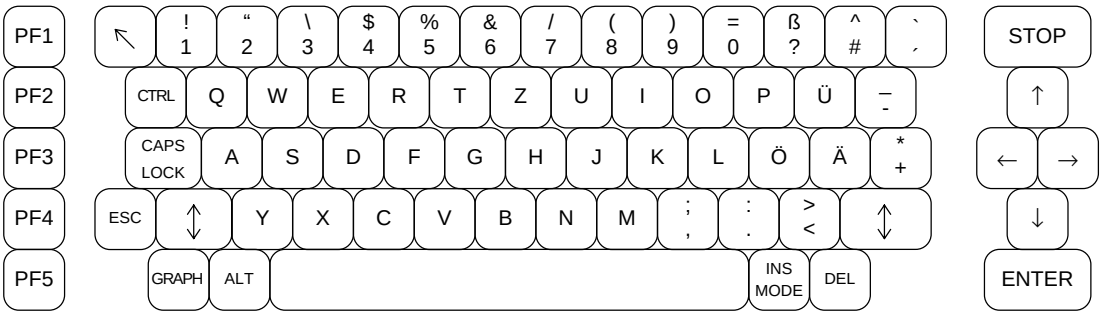
Bevor Sie Ihre Arbeit im RBASIC beenden, sollten Sie prüfen, ob das zuletzt bearbeitete Programm gespeichert werden muß, da es Ihnen sonst verloren geht. Wenn Sie dies getan haben, können Sie Ihre Arbeit einfach durch Ausschalten des Computers beenden.

Möchten Sie im Betriebssystem SCPX 5105 weiterarbeiten, so legen Sie die SCP-Systemdiskette in das Diskettenlaufwerk ein und drücken danach die Reset-Taste an der rechten Seite des Computers. Nach wenigen Sekunden meldet sich dann das Betriebssystem SCPX 5105.

Hinweis: In dieser Beschreibung wird der Kursor durch ein # gekennzeichnet.

Die Tastatur

Zur Eingabe von Kommandos und Programmen nach dem Starten von RBASIC steht Ihnen die Tastatur zur Verfügung.



Bei Betätigung einer alphanumerischen Taste wird das entsprechende Zeichen auf dem Bildschirm an der aktuellen Kursorposition (#) sichtbar. Dabei werden die Buchstaben wie bei der Schreibmaschine als Kleinbuchstaben angezeigt, und bei den Zahlen und Sonderzeichen wird jeweils das untere auf der Taste sichtbare Zeichen angesprochen. Dauerhaftes Drücken einer Taste führt zu einer ständigen Wiederholung dieses Zeichens.

Wie bei der Schreibmaschine kann die Tastatur durch zusätzliches (gleichzeitiges) Drücken einer der sogenannten SHIFT-Tasten ⇳ auf Großbuchstaben bzw. die "oberen" Symbole umgeschaltet werden.

Nach Drücken von CAPS LOCK werden alle Buchstabentasten fest auf Großbuchstaben umgestellt. Alle anderen Tasten funktionieren wie im Normalfall und sind durch SHIFT umstellbar. Gleichzeitig wird der CAPS LOCK-Modus durch eine Kontrollanzeige signalisiert. Erneutes Drücken von CAPS LOCK schaltet wieder in den Normalzustand zurück.

Auch die programmierbaren Funktionstasten (PF-Tasten) sind doppelt belegt. Im Normalfall werden die Funktionen 1 bis 5 angesprochen. In Verbindung mit SHIFT werden die Funktionen 6 bis 10 aktiviert.

Die Zuordnung der durch die Tastatur ansprechbaren Zeichen zu den Tastenkombinationen mit SHIFT, GRAPH und ALT im RBASIC (8-Bit-Zeichensatz) sind in der Bedienungsanleitung, Abschnitt 4.1, und im Anhang B des Programmierhandbuches dargestellt.

Die Nutzung der übrigen Steuer- und Funktionstasten wird später erläutert.

Einfache Rechnungen

Nachdem der Computer durch "Ok" seine Eingabebereitschaft signalisiert hat, können Sie mit wenig Aufwand viele Anweisungen (Rechenoperationen) direkt ausführen, u.a. auch mathematische Grundoperationen, etwa im Bereich eines Taschenrechners. Dieser Modus (d.h. Zustand) des RBASIC-Interpreters wird meist als Kommandomodus (es werden Kommandos ausgeführt) bezeichnet. Aber auch der Begriff Direktmodus ist üblich, da Anweisungen bzw. Operationen "direkt" bzw. "sofort" ausgeführt werden.

Numerische Operationen

Wollen Sie z.B. die Aufgabe

235+117

lösen, so geben Sie zeichenweise über die Tastatur ein

```
PRINT 235+117
```

und drücken danach die ENTER-Taste. Auf dem Bildschirm sind dann Aufgabe und Ergebnis in folgender Weise dargestellt:

```
PRINT 235+117
  352
Ok
#
```

Durch "Ok" wird dabei wieder die vollständige Abarbeitung der Rechenoperation quittiert. Ab der Position des Kursors # können Sie jetzt eine neue Aufgabe eingeben.

Im Unterschied zum Taschenrechner müssen Sie also nicht nur die Aufgabe 235+117 eingeben, sondern dem Computer noch mitteilen, was er mit dem Ergebnis anfangen soll. Durch Voranstellen des Schlüsselwortes PRINT ("Drucke") teilen Sie ihm mit, daß das Ergebnis am Bildschirm angezeigt (auf dem Bildschirm gedruckt) werden soll.

Zur Vereinfachung der Eingabe kann das häufig benötigte PRINT auch durch ein Fragezeichen ("?") abgekürzt werden. Geben Sie z.B. ein:

```
?768-374 ENTER
```

so erscheint am Bildschirm anschließend

```
?768-374
 394
Ok
#
```

Zur Ausführung solcher einfachen Rechenoperationen stehen Ihnen die Operationszeichen

+	für Addition
-	für Subtraktion
*	für Multiplikation
/	für Division
^	für Potenzierung

zur Verfügung. Weiterhin können Sie einige mathematische Standardfunktionen (siehe auch Abschnitt 3.2), wie z.B.

SQR(X)	- Quadratwurzel
SIN(X)	- Sinus (X im Bogenmaß)
LOG(X)	- natürlicher Logarithmus (X>0)

nutzen. Wenn Sie zur Übung Aufgaben lösen wollen, so beachten Sie bitte, daß jede Anweisung durch ENTER abgeschlossen werden muß. Auf dem Bildschirm ergibt sich z.B.

```
?SIN(3.14159*45/100)
 .7071063120938
Ok
?3^7+1/8
 2187.125
Ok
?0.5*LOG(10)/5-3*5)
-.1151292546497
Ok
#
```

Beachten Sie bitte:

1. Bei nichtganzzahligen Werten muß stets ein Dezimalpunkt eingegeben werden, kein Komma. Das entspricht dem internationalen Standard der Rechentechnik.

2. Zur Unterscheidung von dem Buchstaben O wird die Ziffer 0 (Null) auf dem Bildschirm und auf der Tastatur durchgestrichen dargestellt.

Zur Vereinfachung der Bedienung ist für Sie noch folgender Hinweis wichtig:

Die Schlüsselwörter des RBASICs, wie das PRINT in den obigen Beispielen, können Sie sowohl mit Großbuchstaben als auch mit Kleinbuchstaben über die Tastatur eingeben. Die Eingabe von

```
PRINT 72*1.38 ENTER
```

wird genauso abgearbeitet wie

```
print 72*1.38 ENTER
```

Da sich Kleinbuchstaben leichter schreiben lassen, wird man also in der Regel auch nur Kleinbuchstaben eingeben.

Beachten Sie bitte:

1. Zur besseren Übersichtlichkeit sind in diesem Programmierhandbuch alle Schlüsselwörter groß geschrieben.

2. Auch der Computer gibt Schlüsselwörter bei der Programmanzeige (nach LIST) in Großbuchstaben aus.

3. Unabhängig davon können Sie ihre Eingaben zur Vereinfachung in Kleinbuchstaben ausführen.

Zeichenketten

Gleich zu Beginn sollten Sie noch wissen, daß der Computer neben Zahlen auch Zeichenketten verarbeiten kann. Zeichenketten sind Folgen von Buchstaben, Ziffern, Sonderzeichen oder Grafikzeichen, die im allgemeinen durch Anführungszeichen ("...") begrenzt werden. Sie werden zur Textspeicherung und -darstellung benötigt. Geben Sie z.B. ein:

```
PRINT "robotron" ENTER
```

Der Rechner verarbeitet dann die Zeichenfolge robotron als Zeichenkette und gibt diese wieder am Bildschirm aus.

Zeichenketten werden u.a. benötigt, damit Sie ihre Bildschirmanzeigen ordentlich beschriften bzw. gestalten können. Ebenso können Sie damit textliche (allgemeiner: nichtnumerische) Daten speichern, z.B. Namen, Orte, Straßen. WichtigeInformationen zu Zeichenkette finden Sie besonders in den Abschnitten 3.2 und 3.12.

Einfache Fehler

Bereits bei den ersten einfachen Übungen am Computer wird es vorkommen, daß Sie Fehler machen. Das ist ganz natürlich. Dabei werden verschiedene Arten von Fehlern unterschieden. Wichtig ist für Sie, daß es Fehler gibt, die Sie selbst finden müssen und solche, auf die Sie durch den Computer (genauer durch RBASIC) aufmerksam gemacht werden.

Zur ersten Art gehören solche Fehler, bei denen Ihre Eingabe nicht mit der tatsächlichen Aufgabe übereinstimmt, wenn Sie z.B. statt

```
?28/4
```

die Aufgabe

```
?28*4
```

eintippen. Der Rechner kann nicht feststellen, daß Sie dann ein "falsches" Ergebnis erhalten, denn er löst "seine" Aufgabe richtig. Natürlich gibt es auch kompliziertere Fehler, die Sie selbst finden müssen.

Im zweiten Fall versteht der Computer ein Kommando oder Programmzeile nicht bzw. kann die angegebene Operation mit den vorhandenen Daten nicht ausführen. Es erfolgt automatisch ein Programmabbruch und eine Mitteilung auf dem Bildschirm, welche die Fehlermeldung und gegebenenfalls die Zeilennummer, in der der Fehler auftritt, enthält.

Zum Beispiel:

```
Syntax error in 30
```

Mit den Erläuterungen im Anhang H des Programmierhandbuches können Sie dann die Fehlerursache ermitteln. Danach muß der Fehler in Ihrer Anweisung bzw. im Programm korrigiert werden.

Eingabekorrekturen

Bereits bei einfachen Aufgabe kann es vorkommen, daß Sie sich bei der Eingabe über die Tastatur vertippen.

Wenn Sie Ihren Fehler bemerken, bevor Sie ENTER gedrückt haben, können Sie die Eingabe korrigieren. Bewegen Sie dazu den Kursor mit Hilfe der Kursortasten ← und → auf der Eingabezeile und überschreiben Sie die falschen Eingaben durch richtige.

Zur Erleichterung der Korrekturen können Sie in vielen Fällen die Tasten DEL und INS MODE nutzen.

DEL — Durch Drücken dieser Taste wird das Zeichen auf der Kursorposition gelöscht. Der rechts davon bis zum Zeilenende stehende Text wird nun eine Position nach links verschoben. Auch bei dieser Taste ist die Dauerfunktion nutzbar.

INS MODE — Durch Drücken dieser Taste wird in den Insert-Modus (insert = einfügen) umgeschaltet. Der Kursor wird dann deutlich kleiner dargestellt.
Alle jetzt eingegebenen Zeichen werden an der Kursorposition, jeweils links vom Kursor, in den Text der Zeile eingefügt. Der Insert-Modus wird nach erneutem Betätigen dieser Taste wieder verlassen, ebenso nach ENTER oder nach Benutzung der Kursortasten.

Probieren Sie diese Möglichkeit anhand selbstgewählter Beispiele aus, und vergessen Sie nicht, daß der Rechner Ihre Aufgaben erst dann abarbeitet, wenn Sie ENTER gedrückt haben.

Tastaturklick

Entsprechend Ihrem persönlichen Geschmack können Sie sich zur Eingabenunterstützung einen Tastaturklick ein- oder ausschalten.

Das Einschalten erfolgt durch `SCREEN,,,,1 ENTER` ,

das Ausschalten durch `SCREEN,,,,0 ENTER` .

Beachten Sie dabei, daß nach SCREEN jeweils 4 Kommas stehen müssen. Die komplette SCREEN-Anweisung wird später erläutert.

2.2. Einfache Programme

Programmeingabe

Als erstes Übungsbeispiel soll ein ganz einfaches Programm verwendet werden, damit Sie sich an die wichtigsten Bedienhandlungen bei der Programmeingabe und beim Programmtest gewöhnen können.

Für ein Programm zur Berechnung des Mittelwertes von zwei Zahlen müssen Sie beispielsweise folgendes eingeben:

```
10 A = 57 ENTER
20 B = 327 ENTER
30 M = (A+B)/2 ENTER
40 PRINT M ENTER
50 END ENTER
```

Beachten Sie bitte:

1. Auch bei der Programmeingabe können Sie Schlüsselwörter (PRINT, END) und Variablennnamen (hier: A, B und M) sowohl in Groß- als auch in Kleinbuchstaben eingeben. Intern wird automatisch in Großbuchstaben umgewandelt.

2. Die Eingabe jeder Programmzeile muß mit ENTER abgeschlossen werden. Erst dann wird die Zeile (intern) in das Programm übernommen.

3. Fehler bei der Eingabe einer Programmzeile (z.B. Tippfehler) können Sie korrigieren, wie das für einfache Anweisungen im Abschnitt 2. beschrieben wurde, falls Sie noch nicht ENTER gedrückt haben.

4. Bereits mit ENTER übernommene Zeilen können z.B. korrigiert werden, indem sie nochmals (mit der gleichen Zeilennummer, aber richtig) eingegeben werden. Weitere Korrekturmöglichkeiten werden noch beschrieben.

Programmabarbeitung

Nachdem Sie die 5 Zeilen eingegeben haben, wollen Sie dieses kleine Programm auch abarbeiten. Das müssen Sie dem Rechner durch das Kommando RUN (run = rennen, laufen) mitteilen. Geben Sie also ein:

```
RUN ENTER
```

Der Computer arbeitet die Anweisungen der Programmzeilen nun der Reihe nach ab. Im angegebenen Beispiel passiert dabei folgendes:

Zeile 10: Die Variable (mit dem Namen) A bekommt den Wert 57 zugewiesen.

Zeile 20: Die Variable (mit dem Namen) B bekommt den Wert 327 zugewiesen.

Zeile 30: Die Variable M bekommt das Ergebnis des Ausdrucks (der Rechenoperation) (A+B)/2 zugewiesen, wobei für A und B jeweils die aktuellen Werte verwendet werden.

Zeile 40: Der (aktuelle) Wert von M wird am Bildschirm angezeigt.

Zeile 50: Das Programm wird beendet. RBASIC geht in den Kommandomodus zurück. END könnte in diesem Fall, da keine Programmzeile nachfolgt, auch weggelassen werden.

Am Bildschirm sieht das so aus:

```
RUN
 192
Ok
#
```

Der Rechner wartet nun wieder auf eine Anweisung bzw. ein Kommando.

Programmänderung

Bevor Sie ein Programm ändern, sollten Sie sich nochmals über die tatsächliche Programmliste, den aktuellen "Quelltext", des Programms informieren. Das erreichen Sie durch Eingabe des Kommandos

```
LIST ENTER
```

Nach LIST (list = auflisten) wird am Bildschirm der vollständige (Quell)-Text des Programms angezeigt. Bei obigem kleinen Beispiel hat dieser Text sogar vollständig auf dem Bildschirm Platz.

```
LIST
10 A = 57
20 B = 327
30 M = (A+B)/2
40 PRINT M
50 END
Ok
#
```

Änderungen an einem Programm lassen sich im wesentlichen in 3 Gruppen einteilen.

Zeilen einfügen:
Eine Zeile mit einer noch nicht vorhandenen Zeilennummer läßt sich in das Programm einfügen, indem sie ganz einfach zusätzlich eingegeben wird.

```
35 Q = SQR((A*A+B*B)/2) ENTER
45 PRINT Q ENTER
```

Nach erneuter Eingabe des Kommandos LIST können Sie sich überzeugen, daß die Zeilen tatsächlich an der richtigen Position in das Programm eingefügt wurden.

Zeilen löschen:
Eine Zeile kann gelöscht werden, indem nur die Zeilennummer eingegeben wird. Mehrere Zeilen in einem Programm (Blöcke) können auch mit dem DELETE-Kommando gelöscht werden (vgl. Abschnitt 3.3).
Die im Beispiel überflüssige Zeile 50 wird also durch

```
50 ENTER
```

gelöscht. Kontrollieren Sie die Wirkung wieder durch LIST.
Nach RUN liefert das Programm nun

```
RUN
 192
 234.71045992882
Ok
#
```

Zeilen ändern:
Zum Ändern von Zeilen gibt es mehrere Möglichkeiten. Beispielsweise können Sie bei größeren Änderungen die Zeile komplett neu eingeben. Zur besseren Gestaltung der Ergebnisangabe kann im Beispiel geändert werden:

```
40 PRINT "Mittelwert       :";M ENTER
45 PRINT "Quadr. Mittel    :";Q ENTER
```

Die Ausführung des jetzt im Rechner gespeicherten Programms nach RUN liefert auf dem Bildschirm

```
RUN
Mittelwert       : 192
Quadr. Mittel    : 234.71045992882
Ok
#
```

Selbstverständlich können Sie dieses Programm beliebig oft mit RUN starten und abarbeiten.
Beachten Sie bitte, daß im letzten Beispiel in den Zeilen 40 und 45 die Zeichenkettenkonstanten "Mittelwert :" und "Quadr. Mittel :" innerhalb der PRINT-Anweisung verwendet wurden.

Programmspeicher und Bildspeicher

Sie haben sicher bereits festgestellt, daß die Ausschriften am Bildschirm um eine Zeile nach oben rücken, sobald die unterste Zeile beschrieben ist. Ihr Programm ist auch dann noch im Rechner gespeichert und kann durch RUN gestartet werden, wenn es nicht mehr auf dem Bildschirm sichtbar ist.

Beachten Sie bitte:

1. Ein Programm wird im Rechner intern im sogenannten Programmspeicher (auch: Arbeitsspeicher, vgl. auch Anhang E) gespeichert und dort auch abgearbeitet. Dieser Speicherbereich ist direkt nicht sichtbar.

2. Zusätzlich und unabhängig vom Programmspeicher gibt es den Bildspeicher, welcher der direkten Anzeige von Informationen am Bildschirm dient. Durch LIST und PRINT werden Informationen in den Bildspeicher (Text-SCREENs, vgl. auch Anhang E) geschrieben und damit sichtbar.

Löschen des Bildschirmes

Gelegentlich möchten Sie bestimmt einen "sauberen" (leeren) Bildschirm haben. Das ist zweckmäßig, wenn Sie eine Programmliste anzeigen wollen oder die (PRINT-)Ausgaben eines Programms übersichtlich am oberen Rand beginnen sollen.
Dazu dient die CLS-Anweisung (CLS = clear screen = klarer Bildschirm), die sowohl im Kommandomodus, als auch im Programm ausgeführt werden kann, und die den Bildschirm (genauer: den aktiven Bildschirmspeicherbereich) löscht.
Probieren Sie z.B.

```
CLS ENTER
```

aus, und ergänzen Sie anschließend das vorhandene Programm durch die Zeile 5:

```
5 CLS ENTER
```

Die Wirkung dieser Anweisung erkennen Sie durch mehrmaligen Programmstart.

2.3. Eingeben, Abarbeiten und Korrigieren von Programmen

Nach den ersten einfachen Übungen zur Eingabe und Korrektur von Programmen sollen Sie nun weitere Hilfsmittel dafür kennenlernen. Das erleichtert Ihnen später bei umfangreichen Programmen die Arbeit wesentlich.

Beachten Sie bitte, daß vor der Eingabe eines Programms am Computer wichtige Arbeitsschritte erledigt werden müssen, z.B. die Problemanalyse, eine Programm- und Datenkonzeption, die Festlegung von Programmstrukturen u.a. Diese Phasen der Programmerstellung werden jedoch in diesem Programmmierhandbuch nicht behandelt. Vielmehr werden die Hilfsmittel beschrieben, die Ihnen RBASIC zur Umsetzung Ihrer Probleme und Programmvorstellungen bietet.

Generell werden Sie bei der Erarbeitung solcher RBASIC-Programme oftmals bestimmte Schritte am Rechner durchlaufen müssen, die hier nur kurz skizziert sind:

- Eingabe des Programms
  Eingabe des gewünschten Programms über die Tastatur, von Magnetbandkassette oder Diskette (dabei helfen Ihnen z.B. die Kommandos AUTO, CLOAD, LOAD).

- Überprüfung bzw. Korrektur des Programms
  Ausgabe (Listen) des Programms auf den Bildschirm bzw. auf einen Drucker. Gegebenenfalls Änderung und Korrektur (LIST, LLIST).

- Abarbeitung des Programms
  Start des Programms durch RUN (oder Funktionstaste PF 5). Während des Ablaufs des Programms Dialoganforderungen beachten. Abbruch evtl. durch CTRL + STOP möglich.

- Fehler bei der Programmausführung
  Falls das Programm mit einer Fehlermeldung abbricht oder nicht den gewünschten Verlauf nimmt, erneute Überprüfung bzw. Korrektur (2. Anstrich).

- Speichern (Sichern) des Programms auf Magnetbandkassette oder Diskette (CSAVE, SAVE).

Zu den Schritten Eingabe, Korrektur und Abarbeitung von Programmen erhalten Sie anschließend einige Hinweise. Das Speichern (und Laden) von Programmen auf Diskette bzw. Kassette ist im Abschnitt 6 bzw. in der Bedienungsanleitung, Abschnitt 4, beschrieben.

Bevor Sie aber ein neues Programm in den Computer eingeben, müssen Sie das alte Programm zunächst aus dem Programmspeicher des Rechners entfernen (löschen). Das geschieht durch das Kommando

```
NEW
```

(ENTER nicht vergessen!).

Eingabe eines RBASIC- Programms mit AUTO

Die Eingabe des neuen Programms können Sie sich durch den sogenannten AUTO-Modus erleichtern. In diesem Modus nimmt RBASIC Ihnen die Zeilennumerierung ab. Nach Eingabe des Kommandos AUTO wird automatisch die Zeilennummer 10 angezeigt:

```
AUTO
10 #
```

und Sie können nun die Programmzeile mit dieser Zeilennummer eingeben und mit ENTER abschließen. Zum Beispiel:

```
AUTO
10 INPUT "Natürliche Zahl eingeben:";N
20 FOR I=1 TO 1000
30   PRINT I*N
40 NEXT I
50 #
```

Da das Programm mit Zeile 40 bereits enden soll, müssen Sie nach der vollständigen Eingabe dieser Zeile den AUTO-Modus abbrechen. Das geschieht, indem Sie nach Anzeige der Zeilennummer 50 gleichzeitig die Tasten CTRL + STOP drücken.

Danach ist RBASIC wieder im Kommandomodus, Sie können das Programm jetzt mit RUN starten. Wenn Sie richtig eingegeben haben, passiert folgendes:

Zeile 10: Die INPUT-Anweisung bewirkt am Bildschirm die Ausschrift

```
Natürliche Zahl eingeben: #
```

Sie können jetzt eine solche Zahl eingeben, müssen aber wieder mit ENTER abschließen, z.B. :

```
Natürliche Zahl eingeben: 3 ENTER
```

Der Computer übernimmt nun diesen Wert und weist ihn der Variablen N zu (d.h., die Variable N erhält den aktuellen Wert 3).

Zeile 20: Die FOR-Anweisung gibt an, daß alle folgenden Anweisungen bis zur NEXT-Anweisung (in Zeile 40) mehrmals ausgeführt werden sollen. Und zwar soll dabei die Variable I die Werte 1 bis 1000 durchlaufen (I = 1 TO 1000), d.h., für alle I = 1,2,3,...1000 werden die Anweisungen bis zum NEXT I(in Zeile 40) abgearbeitet.

Zeile 30: In unserem Beispiel steht zwischen FOR und NEXT nur die Anweisung PRINT I*N (Zeile 30). Im ersten Durchlauf der FOR...NEXT-Schleife wird Zeile 30 also mit I=1 abgearbeitet, dann mit I=2 usw.

Zeile 40: Schließt die FOR...NEXT-Schleife ab. Falls I noch kleiner als 1000 ist, wird zurück zur FOR-Anweisung (Zeile 20) gesprungen und I um 1 erhöht.

Praktisch bewirkt das Programm das Anzeigen der 3er-Reihe bis zum Wert 1000*3. Gibt man eine andere Zahl als 3 ein, so wird eine entsprechend andere Zahlenfolge angezeigt.

```
RUN
Natürliche Zahl eingeben: 3
 3
 6
 9
 12
 15
 18
 21
 .
 .
```

Programmunterbrechung und -abbruch

Die durch unser Beispielprogramm am Bildschirm angezeigten Zahlen laufen relativ schnell nach oben und sind dann nicht mehr sichtbar. Deshalb ist es relativ wichtig, daß Sie den Programmablauf auch unterbrechen oder abbrechen können, z.B., falls Sie sich nur für einen Teil der angezeigten 3er-Reihe interessieren.

Nach Drücken von STOP wird die Abarbeitung unterbrochen, und zwar nach der Anweisung, die gerade abgearbeitet wurde. Das Programm geht dann in eine PAUSE über. Damit bleibt auch das Bild stehen, es erfolgen keine weiteren Ausgaben. Sie können sich nun in Ruhe die Zahlen ansehen. Nach erneuter Betätigung von STOP läuft das Programm weiter.

Wollen Sie die Abarbeitung ganz beenden (abbrechen), so müssen Sie gleichzeitig CTRL + STOP drücken. Der Computer meldet den Abbruch in der entsprechenden Programmzeile, z.B.

```
Break in 30
Ok
#
```

und geht wieder in den Kommandomodus über.

Programmkorrektur mit dem SCREEN-Editor

In vielen Fällen wird das Programm nicht gleich fehlerfrei laufen. Häufig möchten Sie auch Verbesserungen anbringen. Das Programm muß also korrigiert (verändert) werden. Als effektive Hilfe sollen Sie deshalb den sogenannten SCREEN-Editor (SCREEN = Bildschirm, edit = bearbeiten, redigieren) kennenlernen.

Damit Sie den SCREEN-Editor nutzen können, müssen Sie das Programm (oder den zu korrigierenden Programmabschnitt) am Bildschirm anzeigen. Das geschieht natürlich durch LIST, oder z.B. durch

```
LIST 20-40 ENTER
```

falls Sie nur die Zeilen 20 bis 40 korrigieren wollen.

Danach können Sie den Kursor auf dem Bildschirm beliebig positionieren, wobei Sie die Kursortasten

- zur Bewegung des Kursors um eine Position in die angegebene Richtung

- zur Positionierung des Kursors in die linke obere Bildschirmecke

verwenden können. Sie transportieren den Kursor nun auf die zu korrigierende Programmzeile und

- korrigieren diese Programmzeile, wobei Sie die Funktionstasten INS MODE und DEL verwenden können. (Beachten Sie dabei, daß eine Programmzeile bis zu 254 Zeichen enthalten kann. Sie kann also über mehrere Bildschirmzeilen gehen.)

- Nach Korrektur der Zeile drücken Sie ENTER, bevor der Kursor auf eine andere Zeile bewegt wird.

Die wichtigsten Änderungen eines Programms können mit Hilfe des SCREEN-Editors in folgenden Varianten (siehe nachfolgenden Tabelle) durchgeführt werden:

Problem	**Ablauf**	**Beispiel/Ergebnis**
Änderung einzelner Zeichen, z.B. Korrektur PRIMT in PRINT	1. Kursor auf das zu ändernde Zeichen positionieren.	PRIMT (Kursor auf M)
	2. Vorhandenes Zeichen mit neuem Zeichen überschreiben	PRINT (Nach Überschreiben mit N rückt der Kursor eine Stelle nach rechts.)
	3. ENTER drücken (nach Korrektur der gesamten Zeile)	Geänderte Zeile wird in das Programm übernommen.
Löschen einzelner Zeichen einer Programmzeile, z.B. PRIXNT in PRINT	1. Kursor auf das zu löschende Zeichen positionieren.	PRIXNT (Kursor auf X)
	2. DEL drücken	PRINT (X ist verschwunden; alle folgenden Zeichen sind um eine Stelle nach links gerückt.)
	3. ENTER drücken.	Geänderte Zeile wird in das Programm übernommen.
Einfügen von Zeichen in einer Programmzeile, z.B. PRNT in PRINT	1. Kursor auf das Zeichen positionieren, vor dem eingefügt werden soll.	PRNT (Kursor auf N)
	2. INS MODE drücken.	Kursor wird kleiner.
	3. Fehlende Zeichen eingeben.	PRINT
	4. Wieder INS MODE drücken. (Schritt 4 kann entfallen)	Kursor wird normal
	5. ENTER drücken	Geänderte Zeile wird in das Programm übernommen.
Korrektur einer kompletten Programmzeile	1. Zeile mit der gleichen Zeilennummer überschreiben oder mit dieser Zeilennummer neu eingeben.	Geänderte Zeile.
	2. ENTER drücken.	Geänderte Zeile wird in das Programm übernommen.

Hinweis: 1. Wird ENTER nicht gedrückt, so wird die Änderung der Programmzeile intern nicht ausgeführt!

2. Wird die Zeilennummer mit verändert, so bleibt die alte Zeile mit der alten Zeilennummer erhalten. Sie muß gegebenenfalls gesondert gelöscht werden. Andererseits können auf diese Weise schnell mehrere gleichartige oder ähnliche Programmzeilen erzeugt werden.

3. Durch die freie Beweglichkeit des Kursors können Sie auf alle Anweisungen und Kommandos zurückgreifen, die noch auf dem Bildschirm sichtbar sind. Nachdem Sie den Kursor auf die entsprechende Zeile bewegt haben, können Sie das "alte" Kommando nochmals, evtl. auch in leicht modifizierter Form abarbeiten.

Die Möglichkeiten, den SCREEN-Editor zu nutzen, sind sehr vielfältig. Sie sollten sich deshalb in selbstgewählten Beispielen üben. Oft läßt sich eine Korrektur auch auf unterschiedliche Art ausführen, so daß Sie sich dann die günstigste Art selbst wählen können.
Gelegentlich ist außerdem die Verwendung folgender Tastenkombinationen für Korrekturen nützlich:

⇳	+	↖	Löscht den Bildschirm (wie CLS) bzw. das aktuelle Bildfenster.
bzw.			(Anschließend kann durch LIST ein neuer Programmbereich ange-
CTRL	+	L	zeigt werden).
CTRL	+	E	Löscht von Kursorposition bis Zeilenende.
CTRL	+	N	Positioniert den Kursor an das Zeilenende.
CTRL	+	U	Löscht die gesamte Zeile. (Nachfolgend kann ein Kommando auf der entstandenen Leerzeile eingegeben werden.)

Im Anhang A können Sie sich über weitere Korrekturvarianten mit Hilfe der CTRL-Taste informieren.

Zeilenaufbau und Kommentar mit REM

Ändern Sie nun zur Übung das obige Beispielprogramm mit Hilfe des SCREEN-Editors in

```
5  REM ---- Beispiel Kommentare und Doppelpunkt ----
10 INPUT "Natürliche Zahl eingeben:";N    :REM - Eingabe N
12 INPUT "Maximaler Faktor:";M            :REM - Eingabe M
15 PRINT " I              I*N"            :REM - Ueberschrift
16 PRINT "------------------"             :REM - unterstreichen
20 FOR I=1 TO M                           :' Schleifenanfang
30     P=I*N : PRINT I,P                  :' 2 Anweisungen
40 NEXT I                                 :' Schleifenende
```

und starten das Programm mit RUN.

Das Programm enthält jetzt einige neue RBASIC-Komponenten: die Anweisung REM und den Doppelpunkt. Außerdem werden durch die PRINT-Anweisungen in den Zeilen 15 und 16 diesmal keine Zahlen, sondern Zeichenketten (Texte) angezeigt, die eine Überschrift für die berechnete Tabelle bilden.

Dabei dient der Doppelpunkt zum Trenne mehrere Anweisungen mir einer Programmzeile. In der Reihenfolge

zeilennummer anweisung1: anweisung2: anweisung3...

können Sie durch Doppelpunkt getrennt so viele Anweisungen auf eine Programmzeile schreiben, bis die maximale RBASIC-Zeilenlänge (254 Zeichen) ausgeschöpft ist.

Hinweis: Ein Doppelpunkt innerhalb einer Zeichenkette wirkt nicht als Trennzeichen von Anweisungen. Die Doppelpunkte in Zeile 10 haben also verschiedene Bedeutung.

Hinter dem Schlüsselwort REM können Sie einen beliebigen Text (Kommentar) schreiben, der z.B. der Erläuterung des Programms dient, beim Programmablauf aber unberücksichtigt bleibt. Damit können Sie sich das Programm übersichtlicher bzw. verständlicher gestalten und wichtige Anweisungen erläutern.

REM kann auch durch den Apostroph ‘ ersetzt werden, wie Sie aus den Zeilen 20 bis 40 ersehen können. Demzufolge hätten Sie in Zeile 5 auch schreiben können :

```
5 ' ---- Beispiel  Kommentare und Doppelpunkt ----
```

Hinweis: Die nach REM (bzw. dem Apostroph ‘) eingegebenen Buchstaben werden nicht wie bei Schlüsselwerten in Großbuchstaben umgewandelt.

2.4. Einiges zur Bildgestaltung

Bildformate

Neben dem bisher verwendeten Bildformat mit 25 Zeilen und 40 Spalten unterstützt RBASIC weitere Bildmodi, sogenannte SCREENs, die auch mit der SCREEN-Anweisung eingestellt werden.

Für die Ausgabe von Text (Buchstaben, Zahlen, Zeichen) und Grafik stehen folgende Darstellungsvarianten zur Verfügung:

SCREEN 0	-	Text, 25 x 40 Zeichen (Standard im RBASIC)
SCREEN 1	-	Text, 25 x 80 Zeichen
SCREEN 2	-	Grafik, 320 x 200 Punkte, 4 aus 16 Farben
SCREEN 3	-	Grafik, 640 x 200 Punkte, 4 aus 16 Farben
SCREEN 5	-	Grafik, 320 x 200 Punkte, je Bildpunkt 16 Farben
SCREEN 8	-	Text, 25 x 40 Zeichen, mit Zeilenzwischenraum
SCREEN 9	-	Text, 25 x 80 Zeichen, mit Zeilenzwischenraum

Zu beachten ist dabei, daß in den Text-SCREENs Zeilen und Spalten jeweils mit 0 (Null) beginnend numeriert werden. Das ist z.B. für die Anwendung der LOCATE- und WINDOW-Anweisung (vgl. Abschnitt 3.7) wichtig.

(0,0)	(0,39) - SCREEN 0 und 8 (0,79) - SCREEN 1 und 9
	(24,39) - SCREEN 0 und 8
(24,0)	(24,79) - SCREEN 1 und 9

Alle Zeichen werden stets in einem 8x8-Punktraster dargestellt. In den SCREENs 1 und 9 sind die Punkte (und Buchstaben) nur wesentlich schmaler.
Die SCREEN-Anweisung hat in ihrer einfachsten Form das

Format: **SCREEN** *modus*

modus - Für den Parameter modus muß eine der Zahlen
0, 1, 8 oder 9 - für einen Text-SCREEN
2, 3 oder 5 - für einen Grafik-SCREEN
eingesetzt werden.

Funktion: Einstellen des Bildausgabemodus.

Hinweise:
1. Die SCREEN-Anweisung wird sowohl als Kommando als auch innerhalb eines Programms benutzt.

2. Wird die SCREEN-Anweisung nur in dieser einfachen Form benutzt, wird beim Wechsel des SCREEN-Modus der Bildschirm dann gelöscht, wenn sich die Spaltenzahl ändert. Text- und Grafik-SCREENs beeinflussen sich dabei gegenseitig nicht.

3. Bei Programmende oder -abbruch, Fehlermeldungen und LIST-Ausgaben wird in den zuletzt verwendeten Textmodus zurückgeschaltet.

4. Die SCREEN-Anweisung bietet noch weitaus mehr Parameter und Grundeinstellungen des Computers, die an anderen Stellen des Programmierhandbuches (z.B. Abschnitte 2.1, 4.1, 5.3, 6.2, 7.1) erläutert sind.

Beispiel**:** 1. Geben Sie ein

```
SCREEN 1 ENTER
```

und anschließend

```
LIST ENTER
```

2. Ergänzen Sie jetzt "im SCREEN 1" das Programm aus dem vorhergehenden Abschnitt 2.3, z.B. durch Zeile

```
5 SCREEN 0
```

und starten Sie mit RUN.

Die in den Abschnitten 2.2 und 2.3 beschriebenen Korrekturmöglichkeiten sind natürlich auch in den anderen Text-SCREENs anwendbar.

Umschalten der Bildgröße

Zur Anpassung an den konkret am Computer angeschlossenen Monitor bzw. das Fernsehgerät läßt sich die Bildbreite (Fläche des Ausgabebereiches) in zwei Stufen einstellen. Die Grundeinstellung des Bildes wird durch den Computer in Abhängigkeit von der Gerätekonfiguration automatisch festgelegt. Mit

CTRL X wird das Bild in die jeweils andere Darstellungsart umgewandelt.

Farbgestaltung

Wenn Sie Ihren Computer mit einem Farbfernsehgerät oder einem Farbmonitor betreiben, können Sie in den Text-SCREENs jedes Zeichen in einer von 16 verschiedenen Farben darstellen.

Folgende Farben stehen Ihnen zur Verfügung:

0 - schwarz	8 - dunkelgrau
1 - dunkelblau	9 - blau
2 - dunkelgrün	10 - grün
3 - blaugrün	11 - hellblau
4 - dunkelrot	12 - rot
5 - dunkelpurpur	13 - purpur
6 - ocker	14 - gelb
7 - grau	15 - weiß

Die Farben 0 bis 15 können für die Zeichen selbst (den Vordergrund) verwendet werden. Wenn Sie einen Farbcode zwischen 16 und 31 angeben, z.B. Farbcode 26, werden die Zeichen in der Farbe (Farbcode - 16 = 26 - 16 = 10, in diesem Falle grün) und gleichzeitig blinkend angezeigt. Für den Hintergrund der Zeichen können Sie die Farben 0 bis 7 wählen.
Außerdem läßt sich der Bildschirmrand in den Farben 0 bis 15 darstellen.

Zur Festlegung dieser Farben benutzen Sie die Color-Anweisung:

Format: **COLOR** *vordergrundfarbe, hintergrundfarbe, randfarbe*

vordergrundfarbe - Farbnummer für Vordergrundfarbe;
ganzzahliger Wert von 0 bis 31

hintergrundfarbe - Farbnummer für Hintergrundfarbe;
ganzzahliger Wert von 0 bis 7

randfarbe - Farbnummer für Randfarbe;
ganzzahliger Wert von 0 bis 15

Funktion: Festlegung der Farben entsprechend der konkret einsetzenden Farbcodierung (Konstanten oder Variable) für *vordergrundfarbe*, *hintergrundfarbe* und *randfarbe*.

Hinweise:

1. Die Farbzuweisung wirkt auf alle in den folgenden PRINT-Anweisungen enthaltenen auszugebenden Werte. Bereits vorher angezeigte Werte werden nicht verändert. (Ausnahme: Die Randfarbe wird sofort umgestellt.)

2. Die Parameter der COLOR-Anweisung können einzeln oder von rechts weggelassen werden.

3. Beim Starten von RBASIC wird COLOR 6,0,0 eingestellt.

4. Für die nicht angegebenen Parameter wird der alte Wert beibehalten.

Beispiele:

```
COLOR 6,0,0          :' Standardbelegung
COLOR 15,1,12        :' Vordergrund  - weiß
                        Hintergrund  - dunkelblau
                        Rand         - rot
COLOR ,,7            :' Rand wird sofort grau
COLOR 26             :' Schrift wird grün, blinkt
```

3. Programmierung in RBASIC

Aus den Abschnitten 1 und 2 wissen Sie jetzt schon eine ganze Menge über RBASIC. Bevor Sie sich nun intensiver mit dieser leicht erlernbaren Programmiersprache befassen, hier noch einige Ratschläge:

Die folgende Beschreibung ist so angelegt, daß Sie sowohl in der Reihenfolge der Abschnitte die Sprache und ihre Anwendung auf einfache Weise erlernen als auch, was hin und wieder nötig sein wird, durch Nachschlagen schnell die wesentlichen Informationen finden können.

Die bei der Darstellung der einzelnen Anweisungen und Kommandos gegebenen, teilweise sehr detaillierten Hinweise können Sie bei einem ersten Durcharbeiten ohne Nachteil übergehen. Wichtig ist, daß Sie praktische Erfahrungen sammeln, indem Sie möglichst viele Beispiele nachvollziehen. Sie wissen ja - "Probieren geht über Studieren"!

Trotzdem: RBASIC will es, wie jede andere Programmiersprache, von Ihnen stets ganz genau wissen. Damit Sie die "Grammatik" der Sprache, in der Sie sich mit dem Computer verständigen, möglichst schnell und sicher überblicken können, werden im nächsten Abschnitt einige Darstellungsregeln eingeführt, die später durchgängig benutzt werden.

3.1. Grundbegriffe und Darstellungsregeln

Für den Umgang mit RBASIC sind, wie auch für die meisten BASIC-Versionen, folgende Begriffe wichtig:

Das RBASIC-Programm
ist eine Folge von numerierten RBASIC-Programmzeilen (kurz: RBASIC-Zeile), die Anweisungen enthalten und die in aufsteigender Folge ihrer Zeilennummer nacheinander ausgeführt werden.

Die RBASIC-Programmzeile
enthält eine oder mehrere RBASIC-Anweisungen, die durch Doppelpunkt : voneinander getrennt werden. Sie kann bis zu 254 Zeichen (also mehrere Bildschirmzeilen) lang sein. Leerzeichen zählen dabei mit, auch die vorangestellte Zeilennummer.

Beispiel:
```
10 CLS: PRINT"RBASIC IST PRIMA"
```

Die RBASIC-Anweisung
dient der Formulierung einer bestimmten Aufgabe, die der Computer zum Zeitpunkt ihrer Ausführung erfüllen soll. Sie kann natürlich nicht länger als eine RBASIC-Programmzeile sein.

Die RBASIC-Zeilennummer
muß im Bereich von 0 bis 65529 liegen, ganzzahlig sein und nur einmal auftreten.

Die RBASIC-Programmeingabe
kann in Groß- oder Kleinbuchstaben erfolgen, wobei letztere automatisch in Großbuchstaben gewandelt werden, wenn sie nicht in Anführungszeichen eingeschlossen (als Zeichenkette), als Eingabedaten für Zeichenkettenvariable oder in Kommentaranweisungen auftreten. Leerzeichen werden in der RBASIC-Programmzeile mitgespeichert. Sie sind ohne Bedeutung, falls nicht, wenn man sie wegläßt, ein RBASIC-Schlüsselwort an falscher Stelle entsteht.

Das RBASIC-Kommando
und die sofort ausführbare RBASIC-Anweisung unterscheiden sich von der RBASIC-Programmanweisung äußerlich durch das Fehlen einer RBASIC-Zeilennummer.

Zur Darstellung der Anweisungen und Kommandos (Syntax-Erläuterungen) werden in diesem Handbuch folgende Vereinfachungen benutzt:

Darstellung	Bedeutung
Worte in Großbuchstaben	Schlüsselwörter von RBASIC, die exakt so geschrieben werden müssen (z.B. RUN, AUTO oder LIST).
Worte in Kleinbuchstaben	Parameter (Ersatzworte), die vom Programmierer durch aktuelle Zahlen, Zeichen, Variablenbezeichnungen, Ausdrücke u. ä. ersetzt werden müssen. (SCREEN modus - wird z.B. ersetzt durch SCREEN 1)
Sonderzeichen	Mit Ausnahme von [ ] Sonderzeichen von RBASIC, die exakt so geschrieben werden müssen. (COLOR vordergrundfarbe,hintergrundfarbe - ist zu schreiben als COLOR 6,0 - es muß ein Komma stehen.
[text]	Der *text* (ohne [ ]) kann wahlweise auftreten oder entfallen. (LIST [anfangszeile] - wird zu LIST oder LIST 150)
[text ...]	Der *text* (ohne [ ]) kann wahlweise mehrfach hintereinander auftreten (bis zur Maximallänge von 254 Zeichen je RBASIC-Zeile).
(p1|p2|p3)	Aus den Parametern p1, p2 und p3 muß genau einer ausgewählt und entsprechend eingesetzt werden ((R|S|Z) - einer der Buchstaben R, S oder Z muß eingesetzt werden)
0	Null (zur Unterscheidung vom Buchstaben O)
TASTE	Taste der Tastatur Ihres Computers (STOP - STOP-Taste drücken)

Beachten Sie bitte:

In den Beispielen, die in den weiteren Abschnitten dieses Handbuches angeführt sind, wird in der Regel auf die Darstellung des Eingabeabschlusses mit ENTER und auch der Vollzugsmeldung Ok verzichtet.

3.2. Elemente von RBASIC

A, B d, *, !,...	Zeichensatz
123, 5%, -9.7!, 4.75, "HAUS",...	Konstanten
X, AB, I1%, C#, Z$, TX$,...	Variable
ABS, ATN, COS, EXP, FIX, INT,	Standardfunktionen
LOG, SGN, SIN, SQR, TAN	"
CDBL, CINT, CSNG	Funktionen zur Typumwandlung
A-EXP(X-3), "Name:"+NA$(I)	Ausdrücke

Alle Anweisungen und Kommandos in RBASIC bestehen aus Schlüsselwörtern, die der englischen Sprache entstammen (z.B. READ für "Lies" oder INPUT für "Gib ein"), und Elementen, die in ähnlicher Weise im Mathematikunterricht jeder Schule Verwendung finden. Außer Zahlen kann RBASIC jedoch auch noch mit Text, sogenannten Zeichenketten, operieren. Es kann diese ähnlich wie Zahlen miteinander verknüpfen, vergleichen oder ein- und ausgeben. Allerdings muß der Programmierer beachten, ob er in einer Anweisung mit Zahlen oder Zeichenketten arbeitet, denn nur in wenigen Anweisungen dürfen diese Datentypen auch gemischt auftreten.
Bevor Sie mit der Formulierung einzelner Anweisungen oder Kommandos beginnen, informieren Sie sich zunächst über die "Bausteine", die Sie dabei immer wieder verwenden werden.

Für den "Einsteiger" in die RBASIC-Programmierung genügt zunächst ein Überblick. Einzelheiten sollten bei Bedarf später nachgeschlagen werden!

Zeichensatz

Der im RBASIC verfügbar Zeichensatz besteht aus mehreren Teilen:

- Steuerzeichen (Codierung 1 bis 31)
  dienen der internen Steuerung und sind am Bildschirm nicht sichtbar;

- alphanumerische, Grafik- und Sonderzeichen (Codierung 32 bis 255),
  diese Zeichen sind intern mit einem Byte (8 Bits) verschlüsselt und normal ansprechbar. Die Codierung entspricht dem international üblichen ASCII-Code (IBM-Grafik-Zeichensatz); z.B.: A, i, I, 7, ä (Code ..);

- 2-Byte-Grafikzeichen
  sind spezielle Grafiksonderzeichen, die durch zwei Codezahlen beschrieben werden. Als erster Code wird eine 1 und anschließend eine weitere Codenummer zwischen 64 und 95 (dezimal) angegeben.
  Beispiele: Nutzung von 2-Byte-Zeichen im Programm

```
10 PRINT CHR$(1);CHR$(77)
20 PRINT CHR$(1);CHR$(65);
30 A$=CHR$(1)+CHR$(80):PRINT A$
```

Alle Zeichen lassen sich über die Tastatur ansprechen. Grafikzeichen, Sonderzeichen und 2-Byte-Grafiksonderzeichen werden häufig mit Hilfe der Tasten SHIFT, GRAPH und ALT erreicht (vgl. Anhang B).

Der gesamte Zeichensatz ist einschließlich der Codierung im Anhang A dargestellt.

Konstanten

Konstanten sind feste, d.h. unveränderliche Werte, die im RBASIC-Programm benutzt werden. Nach dem Datentyp ist zwischen numerischen Konstanten (Zahlen) und Zeichenkettenkonstanten (Text) zu unterscheiden:

a) Ganze Zahlen (Integer)
   Sie liegen im Bereich zwischen -32768 und 32768 und werden in ihrer dezimalen Darstellung durch das Suffix % gekennzeichnet.

   Konstanten (dezimal): 187%, 20473%, -15%
   7,65% (wird automatisch zu 7%)

   Ganze Zahlen können neben der üblichen dezimalen Form auch noch in hexadezimaler, oktaler oder binärer Darstellung eingegeben werden. Bei diesen Darstellungen ist folgendes zu beachten:

   hexadezimal
   - Hexadezimale Zahlen werden durch die Zeichen 0 bis 9 und A bis F mit vorangestelltem &H dargestellt. Die Buchstaben A bis F entsprechen dabei den Zahlen 10 bis 15 in dezimaler Notation.
   - Die Zahlen &H0 bis &H7FFF entsprechen den Zahlen 0 bis 32767 (dezimal)
     Die Zahlen &H8000 bis &HFFFF entsprechen den Zahlen -32768 bis -1 (dezimal)
   - Beispiele: &HFF : 255 (dezimal)
     &HFFFE : -2 (dezimal)

   oktal
   - Oktale Zahlen werden durch die Zeichen 0 bis 7 mit vorangestelltem &O dargestellt.
   - Die Zahlen &O0 bis &O77777 entsprechen den Zahlen 0 bis 32767 (dezimal)
     Die Zahlen &O100000 bis &O177777 entsprechen den Zahlen -32768 bis -1 (dezimal)
   - Beispiele: &O177 : 127 (dezimal)
     &O177770 : -8 (dezimal)

binär - Binäre Zahlen werden durch die Ziffern 0 und 1 mit vorangestelltem &B dargestellt.
- Die Zahlen &B0 bis &B111111111111111
entsprechen den Zahlen 0 bis 32767 (dezimal)
Die Zahlen &B1000000000000000 bis &B1111111111111111
entsprechen den Zahlen -32768 bis -1 (dezimal)
- Beispiele: &B11111 : 32 (dezimal)
&B1111111111111100 : -4 (dezimal)

b) Reele Zahlen einfacher Genauigkeit
Sie werden durch maximal 6 signifikante (gültige) Ziffern dargestellt und umfassen die Bereiche

-9.99999E+62 bis -9.99999E–64
und 9.99999E–64 bis 9.99999E+62

einschließlich der Zahl 0 (Null). Diese Zahlen werden durch das Suffix **!** gekennzeichnet.

Z.B.:4.567!, 587.001!, 0.000345678!

c) Reele Zahlen doppelter Genauigkeit
Sie werden durch maximal 14 signifikante Ziffern dargestellt und umfassen die Bereiche

-9.9999999999999E+62 bis -9.9999999999999E–64
und 9.9999999999999E–64 bis 9.9999999999999E+62

einschließlich der Zahl 0 (Null). Diese Zahlen werden durch das Suffix # gekennzeichnet. Das # kann aber auch weggelassen werden, da RBASIC standardmäßig alle Zahlen als doppeltgenaue Zahlen versteht.

Z.B.:158.07, 123.45678901234
–17.4#, 3.3344556677881E+21

Beachten Sie bitte bei der Eingabe reeller Zahlen einfacher und doppelter Genauigkeit, daß

- statt des Dezimalkommas unbedingt ein Dezimalpunkt gesetzt werden muß (1/2 darf also nicht 0,5, sondern muß 0.5 geschrieben werden),
- positives Vorzeichen und auch die 0 vor dem Dezimalpunkt entfallen können (statt +5.5 genügt 5.5, statt 0.33 genügt .33).

Zur Darstellung reeller Zahlen

Zur Unterstützung der Lesbarkeit wandelt der Computer alle Zahlen, deren Betrag kleiner als 0.01 oder größer als bzw. gleich 99999999999999.5 ist, bei der Bildschirmausgabe mit PRINT in Gleitpunktzahlen (wissenschaftliche Darstellung) um. Sie können jedoch auch jede Zahl des Zahlenbereiches in dieser Form eingeben.

Diese Darstellung hat die Form

mantisse E exponent

Der Wert der Zahl berechnet sich dann zu

zahlenwert = mantisse * $10^{exponent}$

Für die Zahl mit der Darstellung -1.03E+7 ergibt sich der Wert -10300000.

Zu beachten ist, daß
- die Mantisse eine ganze oder Festpunktzahl sein muß,
- der Exponent ganzzahlig sein muß, wobei ein positives Vorzeichen auch entfallen kann.

Beispiele: ganze Zahl, Festpunktzahl	Gleitpunktzahl
30000	3E4
-20500	-20.5E+3
12300000	1.23E+7
0.000000123	1.23E-7

Zahlen, deren Absolutwert größer ist als die angegebenen größten Zahlen, führen zu einem Überlauffehler. Bei Unterschreitung des zulässigen Zahlenbereiches wird automatisch der Wert 0 angenommen.
Bei der Ein- und Ausgabe dezimaler numerischer Konstanten (Zahlen) werden automatisch Zahlenum-wandlungen durchgeführt, da die drei Zahlentypen ganze Zahlen, einfach- und doppeltgenaue Zahlen intern unterschiedlich (mit 2, 4 bzw. 8 Bytes) gespeichert werden. Die gleichen Umwandlungen (Konvertierungen) werden durchgeführt, wenn einer Variablen Werte eines anderen Zahlentyps zugewiesen werden.

Sie sollten dabei beachten, daß bei der Umwandlung reeller in ganze Zahlen die "Nachkommastellen" stets abgeschnitten werden. Dagegen wird bei der Umwandlung von doppeltgenauen in einfchgenaue Zahlen gerundet.

Beispiele:	A! =4.123456789	ergibt 4.12346 für A! (einfach genau)
	B! =5/3	ergibt 1.66667 für B! (einfach genau)
	C%=5*7E2/3	ergibt 1166 für C% (Integer)

Zeichenkettenkonstanten

Zeichenkettenkonstanten können aus alphanumerischen, grafischen oder Sonderzeichen des Zeichensatzes gebildet werden. Sie werden mit Anführungszeichen vom übrigen Text abgegrenzt. Es gibt aber auch die Zeichenkette der Länge 0. Man spricht dann von einer "leeren" Zeichenkette. Diese darf nicht verwechselt werden mit einer Zeichenkette, die ein oder mehrere Leerzeichen enthält! Leerzeichen in Zeichenketten werden wie jedes andere Zeichen behandelt.

Beispiele:	"COMPUTER sind vielseitig"	
	""	(leere Zeichenkette)
	" "	(Zeichenkette, die 2 Leerzeichen enthält)
	"1234"	(numerische Zeichenkette, keine Zahl)
	"/*"	(Zeichenkette mit Sonderzeichen)

Variable

Unter einer Variablen versteht man eine Größe, deren Wert (im Gegensatz zur Konstanten) verändert werden kann. Im RBASIC-Programm enthält jede Variable, wie in der Mathematik auch, einen Namen. Der Computer reserviert außerdem für jede Variable Speicherplatz, auf dem er sich den aktuellen Wert dieser Variablen merkt. Somit kann er mit einer Variablen rechnen. Er kann ihr Werte zuweisen oder auch Operationen mit ihr (d.h. mit ihrem aktuellen Wert) ausführen.

Beispiel: Statt der Konstanten 3 und 4 in der Anweisung

```
PRINT 3+4
```

können ebensogut 2 Variable stehen, denen Sie vor der Addition die von Ihnen gewünschten Werte zuweisen, also

```
A=3
B=4
PRINT A+B
```

Die Verwendung dieser Variablen A und B hat den Vorteil, daß unsere Zahlenwerte unter ihrem Namen erhalten bleiben, solange keine neue Wertzuweisung für sie vorgenommen wird. Wir können im obigen Beispiel weiter Anweisungen ergänzen, z.B.

```
PRINT B-A
```

oder auch (mit Änderung des Wertes der Variablen A auf 21)

```
PRINT A+B+14
```

Diese Anweisung macht deutlich, daß Sie durch das Zeichen "=" keine Gleichheit im üblichen mathematischen Sinn verstehen dürfen. Statt dessen ist obige Anweisung zu verstehen als:

A	=	A + B + 14
der neue Wert der Variablen A	"ergibt sich aus"	der Summe der (gegenwärtigen) Werte der Variablen A und B plus der Konstanten 14,

demzufolge bedeutet die häufig verwendete Anweisung

I	=	I + 1
der neue Wert der Variablen I	"ergibt sich aus"	der Summe (des alten) Wertes von I plus (der Konstanten) 1

d.h., zur Variablen I wird 1 addiert, I erhält also einen neuen Wert.

Erst die Verwendung von Variablen ermöglicht in den meisten Fällen die Lösung von praktischen Aufgaben. Sie gestatten es, Eingabedaten, Zwischenergebnisse usw. zu speichern und wieder aufzufinden. Außerdem gelingt mit ihrer Hilfe die von den Daten des speziellen Anwendungsfalles unabhängige und deshalb wiederverwendungsfähige Lösung.

Ebenso wie bei den Konstanten sind verschiedene Arten (Typen) von Variablen zu unterscheiden, die durch Nachstellen eines der Zeichen !, #, % oder $ am Ende des Variablennamens gekennzeichnet werden:

Numerische Variable	Zusatzzeichen
- vom Typ Ganzahlig bzw. Integer	%
- mit einfacher Genauigkeit	!
- mit doppelter Genauigkeit	#
Zeichenkettenvariable	$

Variable mit dem gleichen Name , aber unterschiedlichem Suffix werden als unterschiedliche Variable behandelt. Wenn ein Variablenname kein Suffix besitzt, wird die Variable reell und doppeltgenau angesehen. Ausnahmen bilden die durch eine DEF-Vereinbarung spezifizierten Variablen (vgl. Abschnitt 3.6).

Beispiel:	A, A!, A%, A$	sind verschiedene Variable
	A und A#	sind im Normalfall identisch

Sonst können die Variablennamen bei Beachtung folgender Einschränkungen gewählt werden:

- Sie müssen mit einem Buchstaben beginnen und dürfen beliebig lang sein. Für den Computer sind jedoch nur die ersten beiden Zeichen bedeutsam, die übrigen werden zwar mitgespeichert, dienen aber nicht zur internen Unterscheidung verschiedener Variablen.
- Im Namen dürfen an keiner Stelle RBASIC-Schlüsselwörter enthalten sein. (siehe Anhang G).

Variablennamen können mit Klein- oder Großbuchstaben eingegeben werden. Sie werden intern immer in Großbuchstaben umgewandelt und nach LIST angezeigt.

Beispiele:

Variablenname	Bemerkung
NAME	richtig, aber nur die ersten zwei Buchstaben sind von Bedeutung
X	richtig
A1%	richtig, ganzzahlige Variable
1Y	falsch, Variablenname muß mit einem Buchstaben beginnen
C X	richtig, Leerzeichen sind bedeutungslos
TOR	falsch, enthält das reservierte Wort OR (logischer Operator)
NA	richtig, aber identisch mit der Variablen NAME
TEXT$	richtig, Zeichenkettenvariable
na	richtig, der RBASIC-Interpreter wandelt die Kleinbuchstaben in Variablenbezeichnungen bei der Eingabe in Großbuchstaben um identisch mit NAME und NA
TE$	richtig, identisch mit TEXT$

Felder

Die bislang verwendeten Variablen heißen auch "einfache Variable", da jede einzelne Variable ihren eigenen Namen besitzt. Es können auch Speicherplätze für mehrere Werte unter einem gemeinsamen Namen zusammengefaßt werden. Man spricht dann von einer Feldvariablen oder kurz, einem Feld (siehe Abschnitt 3.8). Hierbei verkörpern die Feldelemente Einzelwerte und sind durch Angabe von Indizes (in runden Klammern hinter dem Feldnamen) eindeutig bestimmt. Ein solches Feldelement heißt deshalb auch "indizierte Variable" und ist inhaltlich einer einfachen Variablen gleichzusetzen. Die Regeln zur Namensvergabe gelten gleichermaßen für Feldvariable wie für einfache Variable.

Beispiele:

```
A(5)=13
```

Dem Feldelement des Feldes A mit der Nummer (dem Index) 5 wird der Wert 13 zuge-wiesen. Die Zählung der Feldelemente beginnt stets mit Null!

```
PRINT T$(3,5)
```

Eine Zeichenkette des 2dimensionalen Zeichenkettenfeldes T$ wird ausgegeben.

Standardfunktionen

Anstelle von Konstanten oder Variablen können Sie in Anweisungen ebensogut eine der im RBASIC enthaltenen mathematischen Standardfunktionen einsetzen, z.B.

```
A=128/2
PRINT SQR(A)
```

Dann wird die Quadratwurzel (engl. square root) des vorher bestimmten Wertes der Variablen A berechnet und ausgegeben. (Beachten Sie stets, daß für die Division der Schrägstrich / satt Doppelpunkt : geschrieben werden muß!)

Folgende Standardfunktionen stehen Ihnen zur Verfügung:

ABS(X)	- Absoluter Betrag
ATN(X)	- Arcustangens (Ergebnis im Bogenmaß)
COS(X)	- Cosinus (X im Bogenmaß)
EXP(X)	- Exponentialfunktion e^x (X<145.0628605862)
FIX(X)	- Ganzer Teil der Zahl X
INT(X)	- Nächstkleinere ganze Zahl zu X
LOG(X)	- Natürlicher Logarithmus (X>0)
SGN(X)	- Vorzeichenfunktion: SGN(X)= (1 für X>0 (0 für X=0 (-1 für X<0
SIN(X)	- Sinus (X im Bogenmaß)
SQR(X)	- Quadratwurzel (X≥0)
TAN(X)	- Tangens (X im Bogenmaß)

Beachten Sie bitte, daß zur Berechnung der Winkelfunktionen die Konstante PI = 3.1415926535898 im Programmtext (z.B. als Anweisung) vorher definiert werden muß. RBASIC stellt diesen Wert nicht bereit!

Zur Umwandlung der verschiedenen Typen numerischer Konstanten und Variablen stehen außerdem folgende Funktionen bereit:

CINT(X)	- wandelt X in eine ganze Zahl um (INT = Integer)
CSNG(X)	- wandelt X in eine Zahl einfacher Genauigkeit um (SNG = single = einfach)
CDBL(X)	- wandelt X in eine Zahl doppelter Genauigkeit um (DBL = double = doppelt)

Die wichtigsten Unterschiede der Funktionen FIX, INT und CINT sollten Sie sich anhand von Beispielen selbst klarmachen. FIX und INT liefern nur für negative Argumente unterschiedliche ganzzahlige Werte (ohne Typumwandlung), während CINT den Zahlentyp in Integer umwandelt, sonst aber einen Wert wie FIX liefert.

Ausdrücke

Ein Ausdruck besteht im einfachsten Fall aus einer einzelnen Konstanten oder einer Variablen, im allgemeinen aus einer Verknüpfung von Operanden (Konstanten, Variablen, Funktionsaufrufen, Ausdrücken) mittels Operationszeichen (kurz: Operatoren). Er stellt wiederum genau einen aktuellen Wert dar, der numerisch oder eine Zeichenkette sein kann.

Beispiel:
```
X1!
15% + 38%*CINT(-6.27)
SIN(3.14159/4) * EXP(SQR(10.47))
A$+"ABCD"+"    TEXT5"
```

Demzufolge gibt es numerische und Zeichenkettenausdrücke, deren Wert, falls er einer Variablen zugewiesen werden soll, vom selben Dateityp (numerisch oder Zeichenkette) wie die Variable sein muß. Wichtig ist, daß die Wertbestimmung eines Ausdrucks nach festen Regeln (Reihenfolge!) erfolgt, die mit den in der Elementarmathematik üblichen Rechenregeln übereinstimmen.

Operation mit Zahlen

Die nachfolgende Tabelle zeigt Rang und Schreibweise der Operatoren:

Rang	Operator	Bedeutung	
1	^	Potenzieren	)
2	–	negatives Vorzeichen	)
3	*	Multiplikation	)
	/	Division	) arithmetische
4	\	ganzzahlige Division	) Operatoren
5	MOD	Rest bei Division mit Integerzahlen	)
6	+	Addition	)
	–	Subtraktion	)
7	<	kleiner als	)
	<=	kleiner als oder gleich	)
	=	gleich	) Vergleichs-
	<>	ungleich	) operatoren
	>	größer als	)
	>=	größer als oder gleich	)
8	NOT	Negation (Verneinung)	)
9	AND	Konjunktion (UND)	)
10	OR	Disjunktion (ODER)	) logische
11	XOR	exklusives ODER	) Operatoren
12	EQV	Äquivalenz	)
13	IMP	Implikation	)

Die Berechnung eines Ausdrucks erfolgt unter Berücksichtigung des Rangs der Operatoren. Dem Rang 1 kommt die höchste Priorität zu, diese Operation wird zuerst durchgeführt. Besitzen zwei Operatoren gleichen Rang, erfolgt ihre Abarbeitung von links nach rechts. Klammerungen und Funktionsaufrufe werden noch vor den in der Tabelle mit Rang angegebenen Operationen ausgewertet.
Bei der ganzzahligen (Integer-) Division (/) werden zunächst alle Operanden durch Rundung in ganze Zahlen umgewandelt und anschließend dividiert. Das Ergebnis wird durch Abschneiden der Dezimalstellen in eine ganze Zahl gewandelt. Der Rest der ganzzahligen Division wird durch die MOD-Operation bestimmt.

Beispiele:
```
K=13.36\3          ergibt den Wert 4 für K
R=13.36 MOD 3      ergibt den Wert 1 für R
```

Trifft der RBASIC-Interpreter während der Auswertung eines Ausdrucks auf eine Division durch Null, so wird die Fehlermeldung "Division by zero" angezeigt. Die gleiche Meldung erfolgt bei Potenzen von Null.

Wird der Betrag des Ergebnisses einer Berechnung größer als der maximal erlaubte Wert, reagiert RBASIC mit der Fehlermeldung "Overflow".

Beispiele zur Bildung von numerischen Ausdrücken:

RBASIC-Darstellung	mathematische Schreibweise	Bemerkungen
A+5/(B-3)	a+5:(b-3)	richtig, Division mit /
K*-52	-52 K	richtig, Multiplikation mit *
-52*K	-52 K	richtig
A*A*H	a^2h	richtig
SIN(A^2+B^2)	$\sin(a^2+b^2)$	richtig, Verwendung einer Funktion
A+FELD(-3)		falsch, -3 ist ein unzulässiger Index

Logische Ausdrücke

logische Ausdrücke werden ausführlich im Abschnitt 3.5 in Zusammenhang mit der IF...THEN-Anweisung behandelt.

3.3. Programmabarbeitung in RBASIC

Programmaufbau
Programmzeile

RUN Starten von Programmen

Editierkommandos:
NEW Löschen von Programmen
AUTO Automatische Zeilennumerierung
LIST Programmanzeige
DELETE Streichen von Programmzeilen
RENUM Numerieren von Programmzeilen

Funktionstasten:
KEY LIST Anzeige der Funktionstastenbelegung
KEY OFF Ausschalten der Standardanzeige
KEY ON Einschalten der Standardanzeige
KEY fkt-tastennummer,string Neubelegen einer Funktionstaste

Programmaufbau

Sofort ausführbare Anweisungen und einfache Programmbeispiele haben Sie in den Kapiteln 2.1 bis 2.3 schon kennengelernt.

Kennzeichen für eine RBASIC-Programmzeile ist die vorangestellte Zeilennummer. Beginnt also eine eingegebene Zeile mit einer Zahl, so wird sie als Programmzeile angesehen und unter der entsprechenden Zeilennummer im Programm eingeordnet. Ohne Zeilennummer wird die Anweisung sofort ausgeführt.

Für die Programmeingabe sind folgende Punkte zu beachten:

- Eine Zeilennummer muß im Bereich von 0 bis 65529 liegen.
- Die Abstände zwischen den Zeilennummern sind beliebig. 10 ist üblich und für weitere Programmänderungen günstig.
- Eine Programmzeile darf maximal 254 Zeichen lang sein.
- Die Zeilen werden entsprechend ihrer Nummer aufsteigend im Programm angeordnet.
- Bei der Eingabe ist nach jeder Zeile die ENTER-Taste zu drücken.
- Um ein Programm zu verkürzen und auch um etwas Rechenzeit einzusparen, können mehrere Anweisungen in eine Programmzeile geschrieben werden., getrennt durch ":".

Programmzeile

Format: *zeilennummer anweisung [:anweisung]...*

zeilennummer - ganze zahl von 0 bis 65529

anweisung - RBASIC-Anweisung

Beispiele: Die beiden Programmbeispiele zeigen den prinzipiellen Aufbau eines Programms und verdeutlichen, daß der Interpreter bei der Abarbeitung etwas Zeit für das Erkennen und Auswerten der Zeilennummern benötigt (TIME siehe Abschnitt 8.5).

```
10 TIME = 0                    :' Zeitzähler initialisieren
20 FOR I = 0 TO 5000
30 NEXT
40 PRINT TIME                  :' Zeitzähler abfragen
50 END

10 TIME = 0                    :' Zeitzähler initialisieren
20 FOR I = 0 TO 5000 : NEXT
30 PRINT TIME                  :' Zeitzähler abfragen
40 END
```

Starten von Programmen

Format: **RUN** *[zeilennummer]*

zeilennummer - ganze Zahl von 0 bis 65529

Funktion: Das im Speicher stehende RBASIC-Programm wird am Anfang oder an der angegebenen Programmzeile gestartet.

Hinweise:

1. Ist die eingegebene Zeile im Programm nicht vorhanden, erscheint die Fehlermeldung "Undefined line number".
2. Bevor die Programmabarbeitung begonnen wird, werden alle Variablen gelöscht, und der READ-Zeiger wird zurückgestellt. RUN schließt also CLEAR und RESTORE ein.

Editierkommandos

Erste Möglichkeiten zur Programmeingabe und -änderung wurden Ihnen im Abschnitt 2.3 schon vorgestellt. Zusammen mit dem Bildschirmeditor liefern Ihnen die im folgenden beschriebenen Kommandos komfortable Möglichkeiten der Programmeingabe, -änderung und -wartung.

Löschen von Programmen

Format: **NEW**

Funktion: Ein im Speicher vorhandenes Programm und alle Variablen werden gelöscht.

Hinweise:
1. Dieses Kommando sollte vor der Neueingabe eines RBASIC-Programms gegeben werden, um eine ungewollte Überlagerung mit einem vorhandenen Programm zu vermeiden.

2. Durch CLEAR eingestellte Speicherbereiche und -adressen bleiben unverändert.

Automatische Zeilennumerierung

Format: **AUTO***[[anfz][,schrittweite]]*

anfzn - Anfangszeilennummer (ganze Zahl von 0 bis 65529)
schrittweite - Schrittweite für Zeilennummern (ganze Zahl von 1 bis 65529)

Funktion: Zur Erleichterung der Programmeingabe wird die automatische Erzeugung der Zeilennummern angewiesen. Die Möglichkeiten der Angabe von Parametern beim Kommando AUTO sind folgende:

AUTO - Ohne Parameter, die Zeilennumerierung beginnt bei Anfangs-zeilennummer 10, die Schrittweite ist 10.

AUTO anfzn - Die Numerierung beginnt bei *anfzn*, die Schrittweite ist 10.

AUTO anfzn,schrittweite - Die Numerierung beginnt bei *anfzn* mit der angegebenen Schrittweite.

AUTO,schrittweite - Die Zeilennumerierung beginnt bei 0 mit der angegebenen Schrittweite.

Hinweise:
1. Wenn für *anfzn* "**.**" geschrieben wird, dann ist die zuletzt behandelte Zeile gemeint. Das ist entweder die zuletzt eingegebene, die zuletzt angezeigte oder die zuletzt bearbeitete Zeile (bei Fehlerabbruch).

2. Die automatische Erzeugung der Zeilennummern wird beendet mit CTRL + STOP, die gerade angefangene Zeile wird nicht ins Programm übernommen.

3. Wird nach einer automatisch erzeugten Zeilennummer ein * ausgegeben, so ist eine Zeile mit dieser Nummer schon im Programm vorhanden. Soll diese Zeile unverändert bleiben, drücken Sie nur die ENTER-Taste. Sie können aber diese Zeile auch einfach überschreiben, indem Sie sie neu eingeben. Das Zeichen * braucht dabei nicht gelöscht zu werden.

Programmanzeige

Formate: **LIST** *[anfzn[- [endzn]]]*
LIST*[[[anfzn] -]endzn]*

anfzn - Anfangszeilennummer
endzn - Endzeilennummer
anfzn, endzn - ganze Zahlen von 0 bis 65529

Funktion: Ein im Speicher stehendes Programm wird entweder ganz oder entsprechend den Parametern teilweise angezeigt. Anschließend können die angezeigten Zeilen mit dem Bildschirmmonitor bearbeitet werden.

Entsprechend der Parametereingabe gibt es folgende Möglichkeiten:

LIST	- Ohne Parameter wird das vollständige Programm angezeigt.
LIST anfzn	- Die Programmzeile *anfzn* wird angezeigt.
LIST anfzn-	- Das Programm wird ab Zeile *anfzn* angezeigt.
LIST anfzn-endzn	- Das Programm wird ab Zeile *anfzn* bis Zeile *endzn* angezeigt.
LIST -endzn	- Das Programm wird von der ersten bis zur Zeile *endzn* angezeigt.

Hinweise:

1. Für *anfzn* oder *endzn* kann wieder "**.**" geschrieben werden. Gemeint ist dann wieder die zuletzt behandelte Zeile.

2. Die laufende Programmanzeige kann mit CTRL + STOP abgebrochen werden.

3. Ein Anhalten der laufenden Programmanzeige ist möglich durch Drücken der STOP-Taste; ein nochmaliges Drücken von STOP setzt die Anzeige fort.

Beispiel: Das folgende Beispielprogramm sei im Speicher vorhanden.

```
10 REM --- Beispiel zu LIST ---
20 PRINT "Testausgabe"
30 PRINT "-----------"
40 END
```

LIST liefert das vollständige Programm.
Weitere LIST-Kommandos liefern folgende Anzeigen:

LIST 20
```
20 PRINT "-----------"
```

LIST 30-
```
30 PRINT "Textausgabe"
40 END
```

LIST 20 - 30
```
20 PRINT "Testausgabe"
30 PRINT "-----------"
```

LIST - 20
```
10 REM --- Beispiel zu LIST ---
20 PRINT "Testausgabe"
```

LIST .-
```
20 PRINT "Testausgabe"
30 PRINT "-----------"
40 END
```

Nach Einfügen von
```
25 NEXT
```
und Programmstart erscheint die Fehlermeldung
```
"NEXT without FOR in 25"
```
Das Kommando LIST -.
bringt dann die Anzeige:

```
10 REM --- Beispiel zu LIST ---
20 PRINT "Testausgabe"
25 NEXT
```

Streichen von Programmzeilen

Format: **DELETE** *[anfzn][-endzn]*

anfzn, endzn - ganze Zahl von 0 bis 65529

Funktion: Aus einem im Speicher stehenden Programm werden eine oder mehrere Zeilen gelöscht. Die Parameterangaben haben dabei folgende Wirkung:

DELETE anfzn - Die Zeile mit dieser Nummer wird gelöscht.

DELETE anfzn-endzn - Die Zeilen von *anfzn* bis *endzn* werden gelöscht.

DELETE -endzn - Die Zeilen von der ersten bis zu *endzn* werden gelöscht.

Hinweise:

1. Soll nur eine Zeile gelöscht werden, so genügt es, die Zeilennummer und ENTER einzugeben.
2. Für *anfzn* oder *endzn* kann wieder "**.**" geschrieben werden. Die Wirkung ist dann analog zu den vorhergehenden Kommandos.
3. Ist eine angegebene Zeile im Programm nicht vorhanden, erscheint "Illegal function call".

Numerieren von Programmzeilen

Format: **RENUM** *[znneu[,znalt[,schrittweite]]]*
RENUM *[znneu],[znalt],schrittweite*

znneu - niedrigste Zeilennummer des neu zu numerierenden Programmteiles (0 bis 65529, Standardwert: 10)

znalt - Zeilennummer, ab der umnumeriert werden soll (0 bis 65529, Standardwert: niedrigste vorhandene Zeilennummer)

schrittweite - Differenz zweier aufeinanderfolgender Zeilennummern (1 bis 65529), Standardwert: 10)

Funktion: Mit diesem Kommando werden die Zeilen eines im Speicher stehenden Programms entweder von Anfang an oder ab der (alten) Zeilennummer *znalt* neu numeriert. Die neue erste Zeilennummer für den umnumerierten Bereich ist entweder 10 oder die angegebene Zeilennummer *znneu*. Die Schrittweite ist entweder angegeben oder 10. Die Parameter können einzeln oder von rechts beginnend weggelassen werden.

Beispiele:

1. `RENUM 5,,2`
   Das gesamte Programm wird umnumeriert. Danach beginnt das Programm mit Zeile 5, der Zeilenabstand beträgt 2.
2. `RENUM ,38,5`
   Das Programm wird von Zeile 38 beginnend mit einer Schrittweite von 5 umnumeriert. Die alte Zeile 38 ist dann Zeile 10.

Hinweise:

1. Für *znneu* und *znalt* kann wieder "**.**" geschrieben werden mit einer entsprechenden Wirkung wie bei den vorhergehenden Kommandos.
2. Unverträgliche Parametereingaben werden abgelehnt mit "Illegal function call".
3. Ist die Zeile *znalt* nicht im Programm vorhanden, beginnt die Neunumerierung bei der darauf folgenden Zeilennummer.

4. Sich auf Zeilennummern beziehende Anweisungen (GOTO usw.) werden beim Neunumerieren mit bearbeitet. Sind diese Zeilennummern nicht im Programm vorhanden, erscheint die Fehlermeldung "Undefined line ... in ... ".

Funktionstasten

Mit den programmierbaren Funktionstasten (PF1 bis PF10) und den Anweisungen zur Behandlung dieser Funktionstasten stehen Ihnen effektive Mittel zur Erleichterung der Bedienung des Computers zur Verfügung.
Jeder dieser Funktionstasten können bis zu 15 Zeichen (auch Steuerzeichen) zugeordnet werden, und ein Drücken einer dieser Tasten wirkt wie das einzelne Eingeben der zugeordneten Zeichen.
Nach dem Einschalten des Rechners wird den Funktionstasten eine Standardbelegung zugeordnet. In Bildschirmzeile 24 werden die augenblicklich erreichbaren Zuordnungen angezeigt, ohne SHIFT also die Belegungen von PF1 bis PF5, mit SHIFT die Belegungen von PF6 bis PF10.

Anzeige der Funktionstastenbelegung

Format: **KEY LIST**

Funktion: Die augenblickliche Belegung der Funktionstasten PF1 bis PF10 wird angezeigt, nach dem Einschalten des Computers also die Standardbelegung. Zugeordnete Steuerzeichen sind dabei auf dem Bildschirm nicht darstellbar, werden aber durch ein Leerzeichen angedeutet.

Hinweise: 1. Die Standardbelegung der Funktionstasten lautet:

```
PF1     "screen"
PF2     - chr$(21)+"auto"
PF3     - chr$(21)+"load"+chr$(34)
PF4     - chr$(21)+"list"
PF5     - chr$(21)+"run"+chr$(13)
PF6     - chr$(21)+"color 6,0,0"+chr$(13)
PF7     - chr$(21)+"files"+chr$(13)
PF8     - chr$(21)+"save"+chr$(34)
PF9     -
        chr$(21)+"list"+chr$(13)+chr$(30)+chr$(30)
PF10    - chr$(21)+"run"
```

2. Auf dem Bildschirm erscheint folgende Anzeige nach KEY LIST:

```
screen
 auto
 load"
 list
 run
 color 6,0,0
 files
 save"
 list.
 run
```

3. Die Standardbelegung wird nach dem Einschalten oder RESET zugeordnet. Wenn den Funktionstasten neue Belegungen während eines Programmablaufes zugeordnet wurden und das Programm wird verlassen, so muß bei Bedarf die alte Belegung regeneriert werden. Sonst gilt die neue Zuordnung weiter

Ausschalten der Funktionsanzeige

Format: **KEY OFF**

Funktion: Die standardmäßig eingeschaltete Anzeige der Funktionstastenbelegungen wird abgeschaltet.

Hinweis: Die Funktionstastenanzeige ist nur abgeschaltet, die Zeile 24 ist noch nicht für Ausgaben erreichbar. Das muß über eine entsprechende WINDOW-Anweisung programmiert werden.

Einschalten der Funktionsanzeige

Format: **KEY ON**

Funktion: Die Funktionstastenanzeige wird eingeschaltet. Sie ist sichtbar auf der Bildschirmzeile 24.

Hinweis: Nach WINDOW 0,24,0,39 und KEY ON ist die Zeile 24 des Bildausgabebereiches zwar erreichbar, aber nicht sichtbar, da auf Bildschirmzeile 24 die Funktionstasten angezeigt werden. Der Bildausgabebereich sollte dann durch WINDOW wieder auf die Zeilen 0 bis 23 eingeschränkt werden.

Neubelegen einer Funktionstaste

Format: **KEY** *fkt-tastennummer,string*

fkt-tastennummer - ganze Zahl von 1 bis 10
string - Zeichenkettenausdruck

Funktion: Der angegebenen Funktionstaste wird die Zeichenkette *string* zugeordnet. Dabei können auch Steuerzeichen (mit Hilfe der CHR$-Funktion) in der Zeichenkette enthalten sein. Damit ist es möglich, die Funktionstasten mit einzelnen oder mehreren Anweisungen oder Kommandos oder Teilen davon (einschließlich ENTER) zu belegen.

Hinweis: Die Belegung der Funktionstasten ist nicht abrufbar und kann auch nicht (z.B. von Diskette) geladen werden. Die Standardbelegung (s. KEY LIST) muß also immer bei Bedarf "per Hand" oder durch ein RBASIC-Programm regeneriert werden.

Beispiel: KEY 1,"circel("

3.4. Einfache Anweisungen

[LET]	Wertzuweisung
PRINT,TAB,SPC	Ausgabeanweisung
INPUT	Eingabeanweisung
BEEP	Ausgeben eines akustischen Signals
SWAP	Tauschen von Variablenwerten

In diesem Abschnitt werden Sie die einfachsten RBASIC-Anweisungen ausführlich kennenlernen. Mit ihnen sind Sie in der Lage, eigene kleine Programme zu formulieren und auszuprobieren.
Zu den ständig benötigten Anweisungen, eigentlich in jedem Programm, gehören Wertzuweisungen und Anweisungen zur Ein- und Ausgaben von Daten über Tastatur bzw. Bildschirm. BEEP und SWAP sind nützliche Ergänzungen.

Wertzuweisungen

Format: **[LET]***variable = ausdruck*

variable - Bezeichnung einer einfachen oder indizierten Variablen (numerische oder Zeichenkettenvariable)

Funktion: Der aktuelle Wert des rechts stehenden Ausdrucks wird bestimmt und der Variablen zugewiesen.

Hinweise:

1. Das Zeichen "=" stellt in der Anweisung kein Gleichheitszeichen dar, sondern wirkt als Zuweisungsoperator. Rechte und linke Seite der Anweisung dürfen also nicht vertauscht werden.
2. Das RBASIC-Schlüsselwort LET kann weggelassen werden.
3. Die Anweisung kann (ohne Zeilennummer) wie fast alle anderen RBASIC-Anweisungen im Kommandomodus ausgeführt werden.

Beispiele:

1. Das Programm berechnet die Länge der Hypotenuse c eines rechtwinkligen Dreiecks mit den Seitenlängen a = 10; b = 20.

```
10 LET A = 10
20 LET B = 20
30 LET C = SQR(A*A+B*B)
40 PRINT C
50 END

RUN

 22.360679774998
```

Mit

```
20 B = 15
```

erhalten Sie nach RUN für C den Wert

```
 18.027756377319.
```

Die Rechenergebnisse müssen natürlich noch vernünftig interpretiert werden.

2. Arbeiten mit Zeichenkettenvariablen

```
100 X$ = " Robotron"
110 Y$ = "Einen Computer?"
120 X$ = Y$+" Dann von"+X$+"!"
130 PRINT X$
140 END

RUN 100

Einen Computer? Dann von Robotron!
```

Ausgabeanweisung

Format: **PRINT** *[[ausdruck][trennzeichen]]...*

ausdruck - numerischer oder Zeichenkettenausdruck
trennzeichen - Komma(**,**) oder Semikolon (**;**)

Funktion: Die aktuellen Werte der angegebenen Ausdrücke werden bestimmt und auf dem Bildschirm (Textbildschirm) angezeigt.
Sind keine Ausdrücke angegeben, wird eine Leerzeile ausgegeben.

Hinweise:

1. Wird als Trennzeichen ein Komma (,) verwendet, so werden zwischen die Ausgaben Leerstellen so eingefügt, als wäre alle 14 Stellen ein Tabulator gesetzt. Die nächste Ausgabe nach dem Komma beginnt dann an der jeweils nächsten Tabulatorposition. Sind für einen Wert nicht mehr genügend Stellen auf der Seite frei, so erfolgt die Ausgabe auf einer neuen Zeile.

2. Wird als Trennzeichen ein Semikolon (;) verwendet, werden die Werte unmittelbar hintereinander ausgegeben.

3. Wird am Ende der Anweisung kein Trennzeichen geschrieben, wird als letztes ein Zeilenvorschub ausgegeben.
   Wird am Ende ein Trennzeichen geschrieben, so wird die nächste PRINT-Ausgabe in derselben Zeile fortgesetzt.

4. Bei der Ausgabe von numerischen Werten wird das positive Vorzeichen (+) weggelassen, statt dessen wird ein Leerzeichen geschrieben. Wird als Trennzeichen das Semikolon (;) verwendet, wird an numerische Werte ein Leerzeichen als Trennung zum nächsten Ausgabewert angehängt.

5. Numerische Werte werden in ihrer natürlichen Darstellung ausgegeben, d.h. als ganze Zahl oder als Festkommazahl ohne Exponent. Nur wenn ihr Betrag kleiner als 0.01 oder größer gleich 99999999999999.5 ist, erfolgt die Ausgabe in Gleitkommadarstellung.

6. Als Abkürzung kann bei der Eingabe für PRINT das Fragezeichen (?) geschrieben werden. Der Interpreter ersetzt es dann durch das Schlüsselwort PRINT.

Beispiel:

```
10 x=123:Y=0.001
20 A$="abc":B$="ABC"
30 PRINT X,Y
40 PRINT X;Y
50 PRINT
60 PRINT A$,B$
70 PRINT A$;B$
80 PRINT+10,-10

RUN


 123     IE-03
 123  IE-03

abc     ABC
abcABC
 10     -10
```

Mit den Funktionen TAB und SPC erhalten Sie die Möglichkeit, auf einfache Weise die Ausgaben auf dem Bildschirm oder Drucker (siehe LPRINT, Abschnitt 7.1) zu positionieren.

Format: **TAB***(n)*

n - numerischer Ausdruck, ganzzahliger Wert von 0 bis 255

Funktion: Die TAB-Funktion füllt den Bereich von der augenblicklichen Kursorposition bis zur Position n mit Leerzeichen auf. Der Kursor steht anschließend auf Position n+1.

Hinweise:
1. Die TAB-Funktion ist nur als Ausdruck in der PRINT- oder LPRINT-Anweisung zulässig.
2. Als Trennzeichen vor und nach TAB(n) sollte nur das Semikolon verwendet werden.
3. Ist die eigentliche Kursorposition größer als oder gleich der geforderten Position n, so ist TAB(n) wirkungslos.

Format: **SPC***(n)*

n - numerischer Ausdruck, ganzzahliger Wert von 0 bis 255

Funktion: Die SPC-Funktion erzeugt bei Textausgaben die angegebene Anzahl von Leerzeichen.

Hinweis: Die SPC-Funktion ist nur als Ausdruck in der PRINT- oder LPRINT-Anweisung zulässig.

Beispiel:
```
10 PRINT"A"
20 PRINT"A";TAB(10);"BCD"
30 PRINT TAB(10);"EFG"
40 PRINT "A";SPC(10);;"BCD"
50 END

RUN


A
A          BCD
           EFG
A           BCD
```

Eingabeanweisung

Format: **INPUT***["hinweis";]var[,var]...*

hinweis - Zeichenkettenkonstante, enthält Hinweis für den Programmanwender
var - Variable (numerische oder Zeichenkettenvariable), denen ein Wert zugewiesen werden soll.

Funktion: Die INPUT-Anweisung dient der bedienerabhängigen Eingabe und Veränderung von Variablen. Mit diesen Werten werden die weiteren Berechnungen ausgeführt, oder sie beeinflussen in anderer Weise den Programmablauf.
Die laufende Programmabarbeitung wird unterbrochen, und die Werte (Konstanten) sind über Tastatur einzugeben.

Hinweis:
1. Ist der *hinweis*-Text in der Anweisung angegeben, so erscheint er auf dem Bildschirm an der aktuellen Kursorposition. Ohne *hinweis*-Text erscheint als Eingabeaufforderung ein Fragezeichen ("? ").
2. Reihenfolge, Typ und Anzahl der einzugebenden Werte müssen mit denen der Variablen übereinstimmen. Als Trennzeichen zwischen den Werten dient das Komma (,), die Eingabe wird mit dem Drücken der ENTER-Taste abgeschlossen.

3. Bei einzugebenden Zeichenkettenkonstanten können die Anführungszeichen ("...") weggelassen werden, wenn kein Komma (,) enthalten sein soll. Ansonsten sind sie mitzuschreiben.

4. Werden bei der Eingabe zuwenig Werte angegeben, so erfolgt mit "?? " die Aufforderung zur Eingabe der restlichen Werte. Wurden zu viele Werte angegeben, werden die überzähligen ignoriert und die Meldung "?Extra ignored" wird angezeigt. Anschließend wird das Programm fortgesetzt.

5. Versucht man, einer numerischen Variablen durch die Eingabe eine Zeichenkette zuzuordnen, so erfolgt die Meldung "?Redo from start", und die gesamte Eingabe für diese INPUT-Anweisung muß wiederholt werden.

6. Durch eine Leereingabe (nur Drücken der ENTER-Taste) bleiben die Werte der Variablen unverändert.

Beispiel: Das Programm berechnet wieder die Länge der Hypotenuse c eines rechtwinkligen Dreiecks. Die Werte für a und b werden jetzt aber während des Programmablaufes eingegeben. Das Programm braucht also für unterschiedliche Werte a und b nicht mehr geändert zu werden.

```
10 INPUT"Werte für a und b?";A,B
20 C=SQR(A*A+B*B)
30 PRINT "c=";C
40 END
```

Nach RUN erscheint zunächst

```
Werte für a und b? #
```

Sie geben nun Ihre Werte ein

```
10,20ENTER
```

und auf dem Bildschirm steht:

```
Werte für a und b? #
c= 22.360679774998
```

Die Werte für a und b können Sie auch einzeln abfragen durch

```
10 INPUT"Wert für a?";A
15 INPUT"Wert für b?";B
```

Ausgeben eines akustischen Signals

Format: **BEEP**

Funktion: Über den Lautsprecher der DSE (oder des Monitors oder Fernsehgerätes) wird ein kurzer Ton erzeugt.

Hinweis: Mit dieser Anweisung kann die Aufmerksamkeit des Programmnutzers wieder geweckt werden, nachdem z.B. länger dauernde Rechnungen abgeschlossen oder Fehler aufgetreten sind.

Beispiel: Im folgenden Programm soll die Eingabe negativer Werte abgelehnt werden. Anschließend ist die Eingabe zu wiederholen.

```
10 INPUT"Zahl?";Z
20 IF Z>=0 THEN GOTO 60
30 BEEP
40 PRINT "Negative Zahl unzulässig!"
50 GOTO 10
60 PRINT "Eingegeben wurde:",Z
70 END
```

Tausch von Variablenwerten

Format: SWAP var1,var2

var1,2 - numerische oder Zeichenkettenvariable oder -feldvariable

Funktion: Die Werte der beiden Variablen werden vertauscht.

Hinweis: Beide Variable müssen vom gleichen Typ sein.

Beispiel:

```
10 A=2:B=7
20 SWAP A,B
30 PRINT"a=";A
40 PRINT"b=";B
50 END

RUN


a= 7
b= 2
```

3.5. Programmablauf

GOTO	unbedingte Verzweigung
NOT, AND, OR, XOR, ...	logische Operationen
IF ... THEN ... ELSE	bedingte Verzweigung
ON ... GOTO	berechnete Verzweigung
FOR ... NEXT	Wiederholungsanweisung
REM	Kommentar
PAUSE	Programmunterbrechung
STOP	Programmabbruch
CONT	Programmfortsetzung
END	Programmende

Mit den bisher dargestellten Möglichkeiten können Sie nur Programme schreiben, die aus Anweisungen bestehen, die in starrer Folge nacheinander abgearbeitet werden. Sie können genau abzählen, wann die Ausführung welcher Anweisung an der Reihe ist. Programme, die so aufgebaut sind, sind zwar einfach, in der Regel aber wenig leistungsfähig.
Voraussetzung für höhere Effektivität ist es, Entscheidungsmöglichkeiten zu schaffen. Diesem Zweck dienen die Programmsteueranweisungen, die im weiteren erläutert werden sollen. In Abhängigkeit von ermittelten Zwischenergebnissen erlauben sie es, den Programmablauf unterschiedlich zu gestalten. Damit wird es möglich, Programme zu schreiben, die angepaßt auf verschiedene Eingabewerte reagieren. Der Ablauf wird vom Wahrheitsgehalt zu prüfender Aussagen abhängig gemacht.
Zunächst soll aber eine Anweisung zur Programmverzweigung erläutert werden, bei der auf den Test einer Bedingung verzichtet wird.

Unbedingte Verzweigung

Format: **GOTO** *zeilennummer*

zeilennummer - RBASIC-Zeilennummer, ab der die Ausführung des Programms fortgesetzt werden soll

Funktion: Die Programmausführung wird an der durch die angegebene Zeilennummer bestimmten Programmzeile fortgesetzt.

Hinweise:

1. Mit GOTO kann nur zu einer tatsächlich im Programm existierenden Zeile gesprungen werden. Wird versucht, eine nicht existierende Zeilennummer anzuspringen, kommt es zu einer Fehlermeldung. Die möglichen Zeilennummern können zwischen 0 und 65529 liegen.

2. Wird die GOTO-Anweisung im Kommandomodus eingegeben, so wird die Ausführung des im Speicher des Computers stehenden Programms mit der durch GOTO bestimmten Zeile begonnen oder fortgesetzt. Von dieser Möglichkeit wird beim Programmtest häufig Gebrauch gemacht.

Beispiel: Für einzugebende ganze Zahlen wird die Quadratzahl berechnet.

```
10 INPUT "Zahl     ?";X%
20 PRINT "Quadrat =";X%*X%
30 PRINT
40 GOTO 10
```

Dieses Programm muß mit CTRL + STOP beendet werden.

Das Beispiel ist etwas "an den Haaren herbeigezogen". Ein sinnvoller Einsatz der GOTO–Anweisung ist in vielen Fällen erst zusammen mit der noch zu erläuternden IF...THEN-Anweisung möglich. Die GOTO-Anweisung sollte allerdings nur sparsam in Programmen verwendet werden, andernfalls können diese leicht unverständlich und unübersichtlich werden.
Bevor nun die bedingte Verzweigung erläutert wird, müssen einige Erklärungen zur Bildung von logischen Ausdrücken, die zur Formulierung von Bedingungen dienen, gemacht werden.

Einfache logische Ausdrücke (Vergleiche)

Einfache Aussagen, die wahr oder falsch sein können, ergeben sich im RBASIC durch den Vergleich von numerischen Konstanten, Variablen oder Ausdrücken. Auch der Vergleich von Zeichenkettenkonstanten, -variablen oder –ausdrücken führt zu derartigen Aussagen.

Im folgenden sind die möglichen Vergleichsoperatoren zusammengestellt.

Vergleichsoperator	Bezeichnung
=	gleich
<	kleiner als
>	größer als
<> , ><	ungleich
<= , =<	kleiner als oder gleich
>= , =>	größer als oder gleich

Einer kleinen Schwierigkeit gilt es Beachtung zu schenken. Das Zeichen = dient in logischen Ausdrücken als Vergleichsoperator, in numerischen Ausdrücken dagegen zur Zuweisung eines Wertes zu einer Variablen. Aus dem Zusammenhang ergibt sich immer, was durch dieses Zeichen bewirkt wird.

Formal können mit den Vergleichsoperatoren beliebige Aussagen gebildet werden. Von Interesse ist im Weiteren nur der Wahrheitswert einer Aussage. Stehen links und rechts vom Operator numerische Ausdrücke, kann durch Vergleich ihrer Werte festgestellt werden, ob die Aussage wahr oder falsch ist. So ist 1 < 2 eine wahre und 5 > 7 eine falsche Aussage. Z + 2 >= 9 ist eine wahre Aussage, wenn Z beispielsweise den Wert 8 besitzt, und eine falsche Aussage, wenn Z der Wert 6 zugeordnet worden ist.

Um für Zeichenketten in ähnlicher Weise Vergleiche ausführen zu können, muß festgelegt werden, wie sie zu ordnen sind. Die Ordnung von Zeichenketten geschieht auf der Grundlage einer alphabetischen Anordnung. Diese unterscheidet sich allerdings etwas von der in Nachschlagewerken üblicherweise verwendeten. Die Ordnung von Zeichenketten wird mit Hilfe der in Anhang A zusammengestellten Codezahlen der Zeichen vorgenommen. Von zwei Zeichenketten ist diejenige die "kleinere", die mit dem Zeichen beginnt, dem die kleinere Codezahl zugeordnet ist. Deshalb ist beispielsweise "A" < "B" eine wahre Aussage. Die dezimale Codezahl 65 des Zeichens A ist kleiner als die Codezahl 66 des Zeichens B. "a" < "B" ist dagegen eine falsche Aussage. Die dezimale Codezahl des Zeichens a beträgt 97. Beginnen zwei Zeichenketten mit gleichen Zeichenfolgen, wird zum Vergleich die erste Stelle herangezogen, an der sie sich unterscheiden. "BAC" < "BB" ist also eine wahre Aussage, "BBC" < "BA" dagegen ist eine falsche Aussage. Die leere Zeichenkette "" ist kleiner als alle übrigen Zeichenketten. Die hier beschriebenen Aussagen werden auch als logische Ausdrücke bezeichnet.

Als Werte logischer Ausdrücke werden ganze Zahlen ermittelt. Die den Wahrheitswerten zugeordneten ganzen Zahlen können folgender Tabelle entnommen werden.

Aussage	Wert des logischen Ausdrucks (ganzzahlig)
wahr	-1
falsch	0

Logische Operatoren

Zusammengesetzte logische Ausdrücke ergeben sich durch Verknüpfung einfacherer logischer Ausdrücke mit Hilfe logischer Operationen.

Der Wert des neuen logischen Ausdruckes ergibt sich nach den Regeln der Aussagenlogik.

Im folgenden stehen X und Y für die zu verknüpfenden logischen Ausdrücke

Logischer Operator	Bezeichnung	Format
NOT	Negation ("nicht")	NOT X
AND	Konjunktion ("und")	X AND Y
OR	Disjunktion ("oder")	X OR Y
XOR	Antivalenz ("entweder-oder")	X XOR Y
EQV	Äquivalenz ("genau dann, wenn")	X EQV Y
IMP	Implikation ("wenn-dann")	X IMP Y

Zusammengesetzte logische Ausdrücke können wieder miteinander verknüpft werden. Die zu verknüpfenden Ausdrücke sind in Klammern einzuschließen. Unter Beachtung der Vorrangregeln (Prioritäten) für die Operatoren können die Klammern weggelassen werden. Die Operatoren sind in der Tabelle nach ihrer Priorität geordnet. NOT besitzt die höchste, IMP die niedrigste Priorität.

So liefert X AND Y OR Z z.B. dasselbe Ergebnis wie (X AND Y) OR Z. Soll dagegen X AND (Y OR Z) bestimmt werden, müssen die Klammern gesetzt werden. Es wird also ähnlich wie bei der Auswertung numerischer Ausdrücke vorgegangen. Das wird Ihnen sofort deutlich, wenn Sie in dem vorangegangenen Beispiel AND durch die Multiplikation * und OR durch die Addition + ersetzen.

In Abhängigkeit von den Werten der logischen Ausdrücke X und Y sind im weiteren die Werte der Verknüpfungen zusammengefaßt. –1 steht dabei für einen wahren und 0 für einen falschen logischen Ausdruck.

X	NOT X
0	-1
-1	0

X	Y	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
0	0	0	0	0	-1	-1
0	-1	0	-1	-1	0	-1
-1	0	0	-1	-1	0	0
-1	-1	-1	-1	0	-1	-1

Von der Tatsache, daß logischen Ausdrücken in RBASIC numerische Werte zugewiesen werden, kann auch Gebrauch gemacht werden, wenn logische in numerischen Ausdrücken verwendet werden. In den numerischen Ausdrücken wird dann in Abhängigkeit vom Wahrheitswert des verwendeten logischen Ausdruckes mit 0 oder –1 gerechnet. Um Mißverständnisse auszuschließen, empfiehlt es sich, den logischen Ausdruck in Klammern zu setzen.

Beispiel:

```
10 INPUT "1. Zahl =";A
20 INPUT "2. Zahl =";B
30 M=A+(A<B)*(A_B)
40 PRINT "Maximum =";M

RUN
```

Bei Eingabe von 117 und 197 erhalten Sie folgendes Ergebnis:

```
1. Zahl = 117
2. Zahl = 197
Maximum = 197
```

Es wird das Maximum der eingegebenen Zahlen A und B bestimmt. Ist A<B, so erhält man für M den Wert von A+(-1)*(A-B)=B, andernfalls von A+0*(A-B)=A.

Mit diesen Kenntnissen über logische Ausdrücke können sie eine wichtige Anweisung zur Steuerung von Programmabläufen, die IF...THEN...ELSE-Anweisung, verstehen. Diese Anweisung erlaubt es, eine Entscheidung über den weiteren Ablauf eines Programmes in Abhängigkeit vom Wahrheitswert eines logischen Ausdrucks zu treffen.

Bedingte Verzweigung

Format: **IF** *bedingung* **THEN** anweisung[:anweisung]...[:**ELSE** anweisung[:anweisung]...]

bedingung - logischer Ausdruck

Funktion: Ist der logische Ausdruck wahr, d.h. die Bedingung erfüllt, werden die zwischen THEN und ELSE stehenden Anweisungen ausgeführt. Ist kein ELSE-Zweig vorhanden, werden bei wahrem Ausdruck alle auf THEN folgenden Anweisungen abgearbeitet. Ist die durch den logischen Ausdruck formulierte Bedingung falsch, werden die nach ELSE stehenden Anweisungen ausgeführt. Ist kein ELSE-Zweig vorhanden, wird bei falschem Ausdruck die Programmabarbeitung in der nächsten Programmzeile fortgesetzt.

Hinweise:

1. Nach Übertragung der Schlüsselworte ins Deutsche läßt sich die inhaltliche Bedeutung der Anweisung leicht merken. WENN die Bedingung erfüllt ist, DANN wird die erste Folge von Anweisungen ausgeführt, sonst die zweite.

2. Erfolgt in einer abzuarbeitenden Anweisung kein Sprung zu einer anderen Programmzeile, wird die Programmabarbeitung nach Ausführung der IF...THEN...ELSE-Anweisung in der folgenden Programmzeile fortgesetzt.

3. Steht unmittelbar nach THEN oder ELSE eine GOTO-Anweisung, ist nur die Angabe der Zeilennummer erforderlich, zu der gesprungen werden soll. Das GOTO kann in diesem Fall weggelassen werden, z.B. IF A<B THEN 150

4. Man sollte weitere IF...THEN...ELSE-Anweisungen in den auszuführenden Anweisungsfolgen vermeiden. Wenn das nicht möglich ist, beachten Sie bitte: Besitzt die betrachtete Anweisung einen THEN- und einen ELSE-Zweig, muß in der Anweisungsfolge nach THEN wieder eine Anweisung mit THEN- und ELSE-Zweig stehen. Andernfalls werden die auf ELSE folgenden Anweisungen der falschen Bedingung zugeordnet.

5. Zwischenergebnisse bei der Berechnung reellwertiger numerischer Ausdrücke werden von RBASIC nur bis zu einer bestimmten Stellenzahl genau ermittelt. Daher können Ausdrücke, die eigentlich dasselbe Ergebnis liefern müßten, infolge unterschiedlicher Berechnungswege zu verschiedenen numerischen Werten führen. Testen Sie im logischen Ausdruck der IF...THEN...ELSE-Anweisung auf Gleichheit, kann es daher passieren, daß Ihr Programm anders als erwartet abläuft. Das können Sie vermeiden, wenn Sie die Bedingung so formulieren, daß der zu testende *ausdruck* innerhalb eines kleinen Intervalls liegt, das auch den eigentlich abzufragenden Wert enthält.

Beispiel:

1. Es ist der Absolutbetrag einer Zahl ohne Verwendung der ABS-Funktion auszugeben.

```
10 INPUT "Zahl   =";A
20 PRINT "Betrag =";
30 IF A>=0 THEN PRINT A ELSE PRINT -A

RUN

Zahl   = -365
Betrag = 365
```

2. Nach dem vorangegangenen einfachen Programm folgt nun ein komplizierteres zur Bestimmung des größten gemeinsamen Teilers zweier Zahlen mit Hilfe der Euklidschen Algorithmen. Erinnern Sie sich, daß M% MOD N% als Ergebnis den Rest bei Division von M% durch N% liefert. Als zweite Zahl darf nicht Null eingegeben werden.

```
10 INPUT "1. Zahl   =";M%
20 INPUT "2. Zahl   =";N%
30 R%=M% MOD N%
40 IF R%<>0 THEN M%=N%:N=R%:GOTO 30
50 PRINT "ggT       =";ABS(N%)

RUN

1. Zahl = 88
2. Zahl = 121
ggT     = 11
```

Mit Erreichen einer IF...THEN...ELSE-Anweisung kann ein Programm alternativ auf zwei verschiedenen Wegen fortgesetzt werden. Zur Programmfortsetzung auf mehreren sich ausschließenden Wegen in Abhängigkeit vom Wert eines numerischen Ausdrucks dient die ON...GOTO-Anweisung.

Berechnete Verzweigung

Format: **ON** *ausdruck zeilennummer*[,*zeilennummer*]...

ausdruck - numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 255 liegt

zeilennummer - Programmzeilennummer, ab der die Ausführung des Programms fortgesetzt werden soll

Funktion: Der ganzzahlige Teil des Wertes des Ausdruckes wird bestimmt und sei gleich I. Das Programm wird dann in der Zeile fortgesetzt, deren Zeilennummer von links gezählt an I-ter Stelle in der angegebenen Zeilennummernliste steht.

Hinweise:

1. Ist der Wert des Ausdrucks Null oder größer als die Anzahl der in der Liste angegebenen Zeilennummern, wird die Programmabarbeitung mit der nächsten Anweisung fortgesetzt.

2. Liefert der berechnete Ausdruck keinen ganzzahligen Wert, wird mit dem Wert gearbeitet, den man bei Weglassen der Stellen nach dem Dezimalpunkt erhält. Liegt der so ermittelte Wert außerhalb des Bereiches von 0 bis 255, kommt es zu einer Fehlermeldung.

Beispiel: Es sind die Nullstellen der Funktion $f(x)=x^2+px+q$ zu bestimmen. Drei Fälle sind bekannterweise in Abhängigkeit vom Wert von

$$D = \frac{p^2}{4} - q$$

zu unterscheiden. Mit SQR wird dabei in den angegebenen Ausdrücken die Funktion zur Berechnung der Quadratwurzel bezeichnet (SQR = sqare root = Quadratwurzel).

Fall 1 (D>0): $x_{01} = -\frac{p}{2} + SQR(\frac{p^2}{4} - q)$

$x_{02} = -\frac{p}{2} - SQR(\frac{p^2}{4} - q)$

Fall 2 (D=0): $x_0 = -\frac{p}{2}$

Fall 3 (D<0): keine reellen Nullstellen

```
10 PRINT "Nullstellenbestimmung"
20 PRINT "f(x)=x^2 + p*x + q"
30 PRINT
40 INPUT "p   =";P
50 INPUT "q   =";Q
60 '--- Fallauswahl ---
70 D=P*P/4-Q
80 IF D>0 THEN FALL=1
90 IF D=0 THEN FALL=2
100 IF D<0 THEN FALL=3
110 ON FALL GOTO 130,180,210
120 '--- 1. Fall ---
130 S1=-P/2:S2=SQR(D)
140 PRINT "x01 =";S1+S2
150 PRINT "x02 =";S1-S2
160 GOTO 230
170 '--- 2. Fall ---
180 PRINT "x0  = ";-P/2
190 GOTO 230
200 '--- 3. Fall ---
210 PRINT "Keine reellen Nullstellen"
220 '--- Ende der Fallauswahl ---
230 PRINT:AN$="j":INPUT "Weitere Berechnungen (j)/n ?";AN$
240 IF AN$="j" THEN 30
```

Da wahren logischen Ausdrücken der Wert –1 zugewiesen wird, können zur Programmvereinfachung die Zeilen 80 bis 100 durch

```
80 FALL=-((D>0)*1+(D=0)*2+(D<0)*3
```

ersetzt werden.

Die begrenzte Rechengenauigkeit des Computers kann zu fehlerhaften Ergebnissen führen, wenn eine der Lösungen betragsmäßig sehr viel kleiner als die zweite ist. Versuchen Sie, das Programm so zu ändern, daß dieser Fehler vermieden wird. Ermitteln Sie die betragsmäßig größere Nullstelle mit Hilfe der Lösungsformel, die andere unter Verwendung des Wurzelsatzes von Vieta ($q = x_{01} * x_{02}$).

Bisher sind Möglichkeiten gezeigt worden, wie zwischen verschiedenen Anweisungsfolgen in Abhängigkeit von Bedingungen gewählt werden kann.

Die nächste Anweisung, die FOR...NEXT-Anweisung, erlaubt es, Anweisungsfolgen wiederholt abzuarbeiten, ohne die Anweisungen mehrfach ins Programm aufnehmen zu müssen. Die Anzahl der Wiederholungen wird durch die Variablen und Ausdrücke nach FOR festgelegt.

Wiederholungsanweisung

Format: **FOR** *laufvariable* = *anfwert* **TO** *endwert* [**STEP** *schrittweite*] **NEXT** [*laufvariable*]

laufvariable - einfache numerische Variable

anfwert - numerischer Ausdruck, der den Wert der Laufvariablen beim ersten Durchlauf bestimmt

endwert - numerischer Ausdruck, dessen Wert maximal (oder minimal) von der Laufvariablen bei positiver (bzw. negativer) Schrittweite angenommen werden kann

schrittweite - numerischer Ausdruck, dessen Wert nach jedem Schleifendurchlauf zur Laufvariablen addiert wird. Fehlt die Angabe, wird 1 addiert.

Funktion: Die zwischen FOR und NEXT stehenden Anweisungen werden wiederholt ausgeführt. Beim ersten Durchlauf wird der Laufvariablen der Wert von *anfwert* zugewiesen. Beim Erreichen des zugehörigen NEXT wird zur Laufvariablen der Wert der Schrittweite addiert. Ist bei positiver (negativer) Schrittweite der neue Wert der Laufvariablen größer (kleiner) als der Endwert, wird die Programmbearbeitung nach der auf NEXT folgenden Anweisung fortgesetzt, andernfalls wird nach FOR weitergearbeitet.

Hinweise:

1. Die durch FOR und NEXT eingeschlossenen Anweisungen werden in jedem Fall mindestens einmal abgearbeitet, da der Vergleich zwischen der Laufvariablen und dem Endwert erst nach einem Durchlauf erfolgt.

2. FOR...NEXT-Anweisungen können ineinander geschachtelt werden. Dabei ist darauf zu achten, daß jede "innere Schleife" vollständig innerhalb der "äußeren" liegt. Der FOR-Teil der äußeren Schleife muß vor dem FOR der inneren Schleife stehen. Der NEXT-Teil der äußeren Schleife muß nach dem NEXT-Teil der inneren Schleife stehen. Die Laufvariablen geschachtelter Schleifen dürfen nicht gleich sein.

3. Mehrere unmittelbar aufeinander folgende NEXT-Teile können zu einem NEXT, gefolgt von einer Liste der Laufvariablen in entsprechender Reihenfolge, zusammengefaßt werden.

4. Wird auf die Angabe einer Laufvariablen hinter NEXT verzichtet, bezieht sich das NEXT immer auf die zuletzt eröffnete Schleife.

5. Ein "Hineinspringen" (z.B. mit GOTO) in die zwischen FOR und NEXT stehende Anweisungsfolge ist nicht erlaubt und führt bei Erreichen von NEXT zu einem Fehler.

6. Es ist möglich, den Wert der Laufvariablen und ggf. die Werte der Variablen, die in den Ausdrücken zur Bestimmung von *anfwert* und *endwert* verwendet werden, durch die Anweisungen innerhalb einer Schleife zu verändern. Es ist jedoch dringend abzuraten, von dieser Möglichkeit vorsätzlich Gebrauch zu machen. Bei der Fehlersuche sollte man sie aber in Betracht ziehen.

Beispiele:

1. Die Summe der Zahlen von 1 bis 100 ist zu berechnen.

```
10 S%=0
20 FOR I%=1 TO 100
30     S%=S%+I%
40 NEXT I%
50 PRINT "Summe 1 bis 100 beträgt ";S%;"."
```

2. Das folgende Beispiel demonstriert die Schachtelung zweier Schleifen.

```
10 FOR I=3 TO 1 STEP -1
20     FOR J=1 TO I STEP 2
30         PRINT I,J
40     NEXT J
50 NEXT I

RUN

3           1
3           3
2           1
1           1
```

Anstelle der Zeilen 40 und 50 hätten Sie auch schreiben können:

```
40 NEXT J,I
```

Die folgenden Anweisungen dienen der Gestaltung von Programmen.

Kommentar

Format: REM [*kommentar*]

kommentar - beliebiger Text

Funktion: Mit der REM-Anweisung können zum besseren Verständnis Kommentare in ein Programm eingefügt werden. Auf die Programmabarbeitung haben sie keinen Einfluß.

Hinweise:
1. Statt des Schlüsselwortes REM kann ein Apostroph (**'**) geschrieben werden.
2. Tritt in einer Programmzeile eine Kommentaranweisung auf, kann dahinter keine weitere Anweisung in derselben Zeile stehen, da der gesamte Text bis zum Zeilenende als Kommentar betrachtet wird.

Beispiel:

```
10 REM --- REM-Anweisung ---
20     Stand: 8. 8. 88
30 G=9.81                 :' Erdbeschleunigung
40 INPUT "Zeit/s ?";T     :' Zeit in s
50 S=G/2*T*T              :' Fallweg in m
60 PRINT "Weg/m =";S
```

Programmunterbrechung

Format: **PAUSE** [*dauer*]

dauer - numerischer Ausdruck mit Wert zwischen –32768 und 65535. Die Länge der Pause beträgt *dauer* * 0,02 s, wenn der Wert von *dauer* positiv ist. Bei negativem Wert ist sie (*dauer*+65535) * 0,02 s lang.

Funktion: Die Programmausführung wird für die durch *dauer* bestimmte Zeit unterbrochen, ohne daß der Programmodus verlassen wird. Fehlt *dauer*, wird das Programm nach Betätigung der Taste STOP fortgesetzt.

Hinweis: Während der Programmunterbrechung sind keine Eingaben möglich.

Beispiel: Die Quadrate der Zahlen von 1 bis 40 sollen ausgegeben werden. Zwischen zwei Ausgaben soll das Programm für jeweils eine Sekunde unterbrochen werden.

```
10 PRINT "x","x^2":PRINT
20 FOR Z%=1 to 40
30     PRINT Z%,Z%*Z%
40     PAUSE 50
60 NEXT Z%
```

Programmabbruch

Format: **STOP**

Funktion: Das laufende Programm wird unterbrochen. Unter Angabe der Zeilennummer kehrt der RBASIC-Interpreter in den Kommandomodus zurück.

Hinweis:

1. Nach STOP bleiben alle aktuellen Variablenwerte und Systemzustände erhalten.

2. STOP-Anweisungen können an beliebigen Stellen im Programm stehen. Sie lassen sich vorteilhaft zum Programmtest einsetzen. So können z.B. die Werte von Variablen nach Programmunterbrechung mit PRINT-Anweisungen im Kommandomodus ausgegeben werden.

3. Wird das Programm nicht durch Editieren, Löschen oder Hinzufügen von Zeilen verändert, bleiben die aktuellen Variablenwerte erhalten. Nach Eingabe einer CONT-Anweisung im Kommandomodus wir das Programm mit der Anweisung fortgesetzt, die dem STOP folgt.

   Mit der Anweisung

   ```
   GOTO zeilennummer
   ```

   kann die Programmfortsetzung in einer anderen gewünschten Programmzeile erreicht werden.

Beispiel: In das vorangegangene Beispiel zur PAUSE-Anweisung ist

```
25 IF Z%>6 THEN STOP
```

einzufügen. Starten Sie das Programm mit RUN, und geben Sie die Werte von Z% nach jeder Programmunterbrechung mit

```
PRINT Z%
```

aus. Sie können die Änderung des Wertes der Laufvariablen Z% auf diese Weise verfolgen. Nach der ersten Unterbrechung wird 7 ausgegeben. Durch Eingabe von

```
CONT
```

wird das Programm fortgesetzt. Sie können vor einer Programmfortsetzung Z% im Kommandomodus auch einen neuen Wert zuweisen, z.B. durch Eingabe von

```
Z%=38
```

Programmfortsetzung

Format: **CONT**

Funktion: Ein mit CTRL + STOP oder mit der STOP-Anweisung unterbrochenes Programm wird fortgesetzt.

Hinweis: Die Fortsetzung beginnt mit der nächsten Anweisung. Mit CTRL + STOP unterbrochene INPUT-Anweisungen werden wiederholt.

Programmende

Format: **END**

Funktion: Das laufende Programm wird beendet. Der RBASIC-Interpreter kehrt in den Kommando-modus zurück.

Hinweise:

1. Mit Erreichen einer END-Anweisung wird die Programmabarbeitung abgeschlossen. Der RBASIC-Interpreter kehrt wie nach END in den Kommandomodus zurück, wenn auf die letzte abgearbeitete Anweisung keine weitere Anweisung folgt.

2. Nach END stehende Programmteile können nur mit Sprunganweisungen oder Unterprogrammaufrufen (vgl. Abschnitt 3.11) erreicht werden. Durch die Endanweisung können Hauptprogramm und Unterprogramme getrennt werden.

3. Sie sollten zukünftig zum ordnungsgemäßen Programmabschluß immer die Endanweisung verwenden.

Beispiele: Vergleichen Sie die Beispiele im Abschnitt 3.11.

3.6. Typvereinbarungen

DEFINT Definiert ganzzahlige Variable
DEFSNG Definiert Variable einfacher Genauigkeit
DEFDBL Definiert Variable doppelter Genauigkeit
DEFSTR Definiert Zeichenkettenvariable

Im Abschnitt 3.2 Haben Sie die Elemente von RBASIC kennengelernt, Sie haben gelernt, wie numerische und Zeichenkettenkonstanten dargestellt werden und welche Wertebereiche welchen Variablentypen zugeordnet sind.
Die möglichen Wertebereiche (und Genauigkeiten) für die Variablen werden entsprechend ihrem Suffix unterschieden. Variablennamen ohne Suffix ordnet RBASIC standardmäßig zu: numerisch, reell, doppelt genau.
Die Zuordnung von Typ, Wertebereich und Genauigkeit zu den Variablennamen ohne Suffix ist in RBASIC aber programmierbar.
Zum einen erspart das Schreibarbeit, wenn viele Variable im Programm nicht das Standardformat tragen sollen, und zum anderen wird das Programm besser lesbar.

Format: **DEFINT** *buchstabe1*[-*buchstabe2*]

buchstabe1 - A,...,Z
buchstabe2 - A,...,Z

Funktion: Alle Variablen mit den Anfangsbuchstaben von *buchstabe1* bis *buchstabe2* wird als neuer Variablentyp Integer zugewiesen, d.h., alle Variablennamen mit diesen Anfangsbuchstaben und ohne Suffix bezeichnen ganzzahlige Variable. Ganzzahlige Werte benötigen zur Speicherung 2 Bytes.

Formate: **DEFSNG** *buchstabe1*[-*buchstabe2*]
DEFDBL *buchstabe1*[-*buchstabe2*]
DEFSTR *buchstabe1*[-*buchstabe2*]

buchstabe1 - A,...,Z
buchstabe2 - A,...,Z

Funktion: Die Variablen mit den angegebenen Anfangsbuchstaben erhalten als Standardtyp bei
DEFSNG - numerisch, reell, einfache Genauigkeit (4 Bytes)
DEFDBL - numerisch, reell, doppelte Genauigkeit (8 Bytes)
DEFSTR - Zeichenkette (3 Bytes + Länge der Zeichenkette),

Hinweis: Überschneiden sich die Bereiche der Variablennamen bei DEF...-Anweisungen, so ist die zuletzt getroffene Zuordnung gültig.

Beispiel:

```
10 DEFSTR T
20 T="T ist eine Zeichenkettenvariable!"
30 A=123456789#
40 DEFINT A-F
50 A=128
60 PRINT T
70 PRINT A%,A#,A
80 END

RUN

T ist eine Zeichenkettenvariable
 128          123456789
 128
```

3.7. Dialoggestaltung

WINDOW		Einstellen des aktuellen Ausgabebereiches
LOCATE		Positionieren des Kursors
CSRLIN, POS		Abfragen der Kursorposition
LINE INPUT	)	
INPUT$	)	Weitere Eingabemöglichkeiten
INKEY$	)	
PRINT USING		Formatierte Ausgabe von Daten

Programme sollten immer so geschrieben sein, daß sie für mehrere Aufgaben mit der gleichen Bearbeitungsvorschrift anwendbar sind. Das bedeutet aber, daß während des Programmablaufs Daten bzw. Informationen in das Programm eingebbar sein müssen, die entweder weiterverarbeitet werden sollen oder die direkt den Programmablauf beeinflussen.
Eine einfache Möglichkeit dafür haben Sie schon in Abschnitt 3.4 mit der INPUT-Anweisung kennengelernt.

Die Aufforderung an den Programmnutzer, Informationen über die Tastatur in das Programm einzugeben, erfolgt in der Regel über den Bildschirm. Dieser Vorgang der Informationsanforderung und –eingabe heißt Dialog. Natürlich lassen sich so nicht immer alle Informationen eingeben, aber z.B. der Name der Datei, die die zu verarbeitenden Daten enthält (vgl. Abschnitt 6).
Die Art der Dialoggestaltung, in der der Nutzer seine Eingabemöglichkeiten auf dem Bildschirm vollständig angeboten bekommt und sie nur auszuwählen braucht, nennt man Menü-Technik.
Die Auswahl der angebotenen Möglichkeiten kann dabei auf verschiedene Weise erfolgen. Entweder wird ein Kennbuchstabe, eine Kennziffer oder ein anderes Zeichen eingegeben, das der gewünschten Möglichkeit zugeordnet wurde, oder die Auswahl wird getroffen, indem der Kursor dorthin positioniert wird.
Die erste Variante läßt sich über PRINT- und INPUT-Anweisungen recht einfach realisieren. Die zweite Variante wird Ihnen am Ende dieses Abschnittes in einem abschließenden Beispiel vorgestellt.

Wie nutzerfreundlich ein Programm ist, hängt in hohem Maße davon ab, wie der Dialog gestaltet wird. Der Nutzer eines Programms ist in der Regel nicht der Programmierer, und Verständigungsschwierigkeiten sollten ausgeschlossen werden. Aus diesem Grund gibt es einige prinzipielle Anforderungen an die Dialoggestaltung, die in jedem Nutzerprogramm eingehalten werden sollten:

- Programmbeginn immer mit einem Eröffnungsbild,
- verständliche Eingabeaufforderung,
- weitgehender Verzicht auf Abkürzungen,
- Anzeige der möglichen Eingaben,
- vernünftige Reaktion auf falsche Eingaben,
- sparsame Nutzung der Bildschirmfläche im Dialog (weniger als 50%),
- kurze Reaktionszeiten des Programms.

Die nachfolgend erläuterten Anweisungen und Funktionen sollen Ihnen helfen, diese Forderungen effektiv im Programm umzusetzen.

Einstellen des aktuellen Ausgabebereiches

Format: **WINDOW** [*zeile1*,*zeile2*,*spalte1*,*spalte2*]

zeile1 - erste Zeile des Ausgabebereiches
zeile2 - letzte Zeile des Ausgabebereiches
spalte1 - erste Spalte des Ausgabebereiches
spalte2 - letzte Spalte des Ausgabebereiches

zeile1,2	- 0 bis 24,	zeile1 <= zeile2
spalte1,2	- 0 bis 39 in SCREEN 0 und 8,	spalte1 <= spalte2
	0 bis 79 in SCREEN 1 und 9,	spalte1 <= spalte2

Funktion: Einstellen des aktuellen Ausgabebereiches.
Innerhalb des Ausgabebereiches erscheinen sämtliche Ausgaben von PRINT-Anweisungen, Eingabeaufforderungen von INPUT-Anweisungen, Fehlermeldungen und alle Ein- und Ausgaben im Kommandomodus.
WINDOW ohne Parameter stellt den Standardbereich ein.

Der Standardbereich ist abhängig vom eingestellten Bildschirmmodus und der Funktionstastenanzeige:

SCREEN 0 und 8 - 40 Spalten (0...39)
SCREEN 1 und 9 - 80 Spalten (0...79)

Funktionstastenanzeige (ein - 24 Zeilen (0...23)
(aus - 25 Zeilen (0...24)

Hinweise:

1. Die gleiche Wirkung wie bei Anweisung WINDOW (ohne Parameter) hat die Eingabe CTRL + W.

2. Alle einzugebenden Zeichen übernimmt RBASIC aus dem Ausgabebereich des Bildspeichers. Wenn dieser Bereich zu klein definiert wurde, hilft nur die Eingabe von CTRL + W.

3. Nach dem Ausführen der Anweisung steht der Kursor im Ausgabebereich links oben, der Bereich wird nicht automatisch gelöscht. CLS und die Kursorbewegungen beziehen sich nur auf diesen Ausgabebereich.

4. Die außerhalb des Ausgabebereiches liegenden Teile des Bildschirmes bleiben "eingefroren".

5. SCREEN-Umschaltungen stellen den jeweiligen Standardausgabebereich ein.

6. Die Zeile 24 des Bildschirmes kann in den Ausgabebereich einbezogen werden, ist aber nur nach KEY OFF sichtbar (vgl. Abschnitt 3.3, Funktionstasten).

Beispiel: Zunächst wird der gesamte Bildschirm als Ausgabebereich definiert und gelöscht. Danach wird die Überschrift in Zeile 0 ausgegeben. Die nachfolgenden PRINT-Anweisungen wirken auf die Zeilen 2 bis 23.

```
10 WINDOW:CLS
20 PRINT "Zahl","Quadratzahl"
30 WINDOW 2,23,0,39
40 FOR I=0 TO 50
50 PRINT I,I*I:PAUSE 25
60 NEXT I
70 END
```

Stellen Sie bitte nach Ausführung des Programms durch CTRL + W den Standardausgabebereich wieder ein.

Positionieren des Kursors

Format: **LOCATE** *zeile*[, *spalte*[,*kursor*]]
LOCATE [*zeile*], [*spalte*},*kursor*

zeile - neue Kursorzeile (0 bis 24), ohne Angabe unverändert
spalte - neue Kursorspalte (0 bis 39 bzw. 79), ohne Angabe unverändert

kursor - 1 - ein
0 - aus
ohne Angabe - unverändert
Standard - ein (nach Neustart RBASIC)

Funktion: Mit LOCATE kann der Kursor beliebig auf dem Bildschirm positioniert und ein- oder ausgeschaltet werden. Die Parameter können von rechts oder einzeln weggelassen werden.

Hinweis:

1. Der Kursor sollte mit LOCATE sinnvollerweise nur innerhalb des aktuellen Ausgabebereiches (WINDOW) positioniert werden. Andernfalls wird er an den Grenzen des Ausgabebereiches "eingefangen".

2. Werden für *zeile* und *spalte* größere Werte als die angegebenen verwendet, so werden die maximal möglichen angenommen. Werte größer als 255 werden mit "Illegal function call" abgelehnt.

3. Im Standardfall ist der Kursor während des Programmlaufs immer zu sehen, auch wenn das Programm nicht für Eingaben bereit ist.
Das kann einerseits zu unerwünschten Flimmereffekten bei PRINT-Anweisungen führen, andererseits Eingabebereitschaft vortäuschen.

4. Während der Abarbeitung von INPUT-Anweisungen ist der Kursor immer zu sehen, er wird für die Zeit der INPUT-Anweisung eingeschaltet (gilt auch für LINE INPUT und INPUT$, nicht für INKEY$).

Abfragen der Kursorposition

Format: **CSRLIN**

Typ: BASIC-Funktion

Funktion: CSRLIN liefert als Funktionswert die Zeile, in der der Kursor augenblicklich steht.

Format: **POS**(X)

X - beliebiger numerischer Wert, ganzzahlig

Typ: BASIC-Funktion

Funktion: POS liefert als Funktionswert die Spalte, in der der Kursor augenblicklich steht.

Hinweis: Der Wert des numerischen Argumentes hat keinen Einfluß auf den Funktionswert.

Beispiel:

```
10 INPUT A$
20 PRINT A$;
30 S=POS(0):Z=CSRLIN
40 LOCATE Z+3,S+5
50 PRINT "Ende"
60 END
```

"Ende" wird unabhängig von der Länge der Zeichenkette A$ (0...255) drei Zeilen unter und fünf Zeichen hinter dem letzten Zeichen von A$ ausgegeben.

Weitere Eingabemöglichkeiten

Außer der INPUT-Anweisung (vgl. Abschnitt 3.4) gibt es noch weitere Anweisungen und Funktionen, mit deren Hilfe Nutzereingaben programmiert werden können.
Die INPUT-Anweisung hat einige Eigenschaften, die bei speziellen Anwendungen von Nachteil sein können. Ein Koma als Teil einer Zeichenkette läßt sich z.B. nur umständlich einer Variablen zuweisen; von allen eingegebenen Zeichen erscheint ein Echo auf dem Bildschirm, und alle Eingaben müssen durch Drücken der ENTER-Taste abgeschlossen werden.

Format: **LINE INPUT**["*hinweis*";]*var*

hinweis - Zeichenkettenkonstante, enthält Hinweis für den Programmanwender
var - Zeichenkettenvariable, der eine Zeichenkette zugewiesen werden soll

Funktion: Der Zeichenkettenvariablen *var* wird über die Tastatur die eingegebene Zeichenkette zugewiesen.

Hinweise:

1. Die Eingab wird mit dem Drücken der ENTER-Taste abgeschlossen.
2. Der Variablen werden maximal 254 Zeichen zugewiesen. Werden mehr Zeichen über die Tastatur eingegeben, werden die überzähligen ignoriert.
3. Alle eingegebenen Zeichen - auch Komma (**,**) und Anführungszeichen (**"**) - werden der Variablen zugeordnet und haben keine Sonderfunktion.
4. Ist der *hinweis*-Text in der Anweisung angegeben, so erscheint er auf dem Bildschirm an der aktuellen Kursorposition. Ohne Hinweis-Text erscheint nur der Kursor.

Beispiel:

```
10 LINE INPUT "Name, Datum: ";N$
20 PRINT N$
30 END

RUN

Name, Datum: Erwin, 19.10.1988
Erwin, 19.10.1988
```

Format: **INPUT$**(*n*)

n - numerischer Ausdruck, ganzzahlig, Wert von 1 bis 255

Typ: BASIC-Funktion

Funktion: INPUT$ liest die angegebene Anzahl von Zeichen, die über die Tastatur eingegeben werden und gibt als Funktionswert eine entsprechende Zeichenkette zurück.

Hinweise:

1. Alle Zeichen, die über die Tastatur eingegeben werden, werden der Variablen zugeordnet, auch Steuerzeichen.
2. Die eingegebenen Zeichen werden nicht auf dem Bildschirm angezeigt. Deswegen werden Steuerzeichen auch in die Zeichenkette übernommen und wirken nicht auf den Eingabebereich.
3. Die Eingabe wird nach der angegebenen Zeichenzahl beendet, nicht vorher. Sie kann durch CTRL + STOP abgebrochen werden.
4. Eine Echofunktion auf dem Bildschirm muß mit einer PRINT-Anweisung programmiert werden.

Beispiel:

```
10 CLS
20 PRINT "Nutzerkennzeichen"
30 X$=INPUT$(4)
40 CLS
50 IF X$<>"Chef" THEN NEW
60 PRINT "Guten Tag!"
70 END
```

Wird das falsche Nutzerkennzeichen eingegeben, so zerstört sich das Programm selbst.

Format: **INKEY$**

Typ: BASIC-Funktion

Funktion: INKEY$ übernimmt in Zeichen von der Tastatur und ordnet es einer Zeichenkettenvariablen zu. Wurde keine Taste gedrückt, so wird der Variablen die leere Zeichenkette ("", LEN("")=0) zugeordnet, sonst eine Zeichenkette mit der Länge 1.

Hinweis:

1. INKEY§ wartet nicht auf die Eingabe, sondern versucht, sofort ein Zeichen zu übernehmen. Wurde keine Taste gedrückt, ist das Ergebnis deswegen die leere Zeichenkette.

2. Gedrückte Tasten werden in einem Tastaturpuffer zwischengespeichert, er kann maximal 39 Zeichen aufnehmen. INKEY$ liest aus diesem Tastaturpuffer das "älteste" Zeichen, u. U. also ein lange vor dieser Anweisung eingegebenes.

3. Soll ein Zeichen unmittelbar von der Tastatur eingelesen werden, so muß vorher der Tastaturpuffer geleert werden.

4. Nur Tasten, die einen Tastencode entsprechend Anhang A liefern, werden durch INKEY$ erkannt. CTRL + STOP führt zum Programmabbruch.

Beispiele:

1.
```
10 CLS
20 X$=INKEY$
30 IF X$<>"" THEN PRINT X$;
40 GOTO 20

RUN
```
Über Tastatur eingegebene Zeichen, einschließlich Steuerzeichen, werden auf den Bildschirm ausgegeben.

2. Nach Einfügen von Zeile

```
5 PAUSE 250
```

können vor dem Löschen des Bildschirmes schon Zeichen "auf Vorrat" eingegeben werden, die dann als erstes von INKEY$ erkannt werden.

3. Mit Zeile

```
15 IF INKEY$<>"" THEN GOTO 15
```

wird nach dem Löschen des Bildschirmes zunächst der Tastaturpuffer geleert. Während der Pause (Zeile 5) eingegebene Zeichen werden damit wirkungslos.

Formatierte Ausgabe von Daten

Wesentlich umfangreichere Möglichkeiten der Datenausgabe auf den Bildschirm (oder andere Geräte, vgl. Abschnitte 4.5, 6.4 und 7.1) erhalten Sie mit der Anweisung PRINT USING. Die einfache PRINT-Anweisung aus Abschnitt 3.4 gibt die Daten unformatiert, d.h., so wie sie vom Rechner kommen, auf den Bildschirm aus. Kommastellung und Zeichenanzahl lassen sich damit nur schwer beeinflussen. Das ist bei Tabellenausgaben, wie z. B. Preislisten, sehr störend. Die "PRINT USING"-Anweisung ist hier, aber auch bei Ausgabe von Zeichenketten, sehr hilfreich.

Format: **PRINT USING** *format*;*ausgabeliste*

format - Zeichenkettenausdruck, beschreibt das Ausgabeformat, in dem die Ausdrücke der *ausgabeliste* angezeigt werden

ausgabeliste - enthält auszugebende numerische und Zeichenkettenausdrücke (auch Konstanten oder Variable)

Funktion: Die Werte der in *ausgabeliste* enthaltenen Ausdrücke, Konstanten oder Variablen werden entsprechend *format* aufbereitet und auf den Bildschirm ausgegeben

Hinweise:

1. Die Typen von *format* und auszugebendem Ausdruck müssen übereinstimmen.

2. Der Parameter *format* kann als Zeichenkettenausdruck angegeben und in der "PRINT USING"-Anweisung damit variabel gestaltet werden.

3. Der Parameter *format* selbst kann mehrere "Formatanweisungen" enthalten. Die Ausdrücke der *ausgabeliste* werden der Reihe nach den einzelnen Formaten zugeordnet. Sind weniger Formate als Ausdrücke in *ausgabeliste* vorhanden, beginnt die Zuordnung von vorn.

4. Als Trennzeichen in *ausgabeliste* dienen Komma (,) oder Semikolon (;). Sie haben, anders als bei PRINT, keinen weiteren Einfluß auf die Ausgabegestaltung.

5. Ein Trennzeichen am Ende der *ausgabeliste* unterdrückt die Zeilenschaltung.

6. Alle Zeichen aus *format*, die keine Bedeutung entsprechend Hinweis 7 tragen, werden unverändert in die Ausgabe übernommen.

7. Die Möglichkeiten der Formatgestaltung zeigen die nachfolgenden Übersichten.

Ausgabe von Zeichenketten

Symbol	Funktion, Beispiel

"!"	Ausgabe des jeweils ersten Zeichens der Elemente von *ausgabeliste*
	`PRINT USING "!"; "Personal","Computer"`
	`PC`

"\ \"
n Leerzeichen
	Ausgabe von n+2 Zeichen der Elemente der *ausgabeliste*; bestehen diese aus weniger als n+2 Zeichen, so wird mit Leerzeichen aufgefüllt.
	`PRINT USING "\      \";"Personalcomputer"`
	`Person`

"&"	Ausgabe der vollständigen Elemente von *ausgabeliste*.
	`PRINT USING "&& von robotron.":"Personal","computer"`
	`Personalcomputer von robotron.`

"#"	Jedes # kennzeichnet eine auszugebende Stelle: das negative Vorzeichen (-) zählt dabei mit, das positive (+) wird weggelassen. Der Punkt kennzeichnet die Stellung des Dezimalpunktes.
	Die Werte werden entsprechend der Formatangabe gerundet, führende Nullen werden durch Leerzeichen ersetzt, fehlende angehängt, die Ausgabe erfolgt rechtsbündig.
	Sind zuwenig numerische Stellen vorgesehen, so werden die Werte vollständig entsprechend dem Format, aber mit einem führenden Prozentzeichen (%), ausgegeben.
	`10 PRINT USING "Preis: ##.## M";1.3`
	`20 PRINT USING "Preis: ##.## M";128.7`
	`RUN`
	`Preis:  1.30 M`
	`Preis: %128.70 M`

"+"	Ein Plus (+) vor oder nach den numerischen Stellen bewirkt die Ausgabe des positiven oder negativen Vorzeichens unmittelbar vor bzw. nach den numerischen Werten.
	`10 PRINT USING "+###.# ";25.3,-149`
	`20 PRINT USING "###.#+ ";-149,25.3`
	`RUN`
	` +25.3 -149.0`
	`149.0-  25.3+`

"-"	Ein Minus (-) nach den numerischen Stellen bewirkt die Ausgabe des negativen Vorzeichens, anstelle des positiven wird ein Leerzeichen geschrieben.
	`PRINT USING "###-";-111,222,-333`
	`111-222 333`

","	Wird ein Komma (,) innerhalb der Formatanweisung und vor dem Dezimalpunkt (.) angegeben, so erfolgt die Ausgabe eines Kommas als Stellenanzeige nach jeder dritten Position links vom Dezimalpunkt.
	`PRINT USING "##,######.##";1234567`
	`1,234,567.00`

"**"	Ein Stern (*) stellt eine numerische Stelle dar. Der Bereich vor den numerischen Werten wird aber entsprechend der Gesamtstellenzahl mit "*" aufgefüllt.
	`PRINT USING "**###.##";1.7`
	`****1.70`

"MM"	Ein M stellt eine numerische Stelle dar. Unmittelbar vor den numerischen Werten wird ein "M" ausgegeben.
	`PRINT USING "MM##.##;5.7`
	`  M5.70`

"**M"	Jedes Zeichen stellt eine numerische Stelle dar. Unmittelbar vor den numerischen Werten wird ein "M" ausgegeben, der Freiraum links wird mit "*" aufgefüllt.
	
	`PRINT USING "**M#.##";12.3`
	`*M12.30`

"^^^^"	Die numerischen Werte werden in Gleitpunktdarstellung ausgegeben, wobei "^^^^" nur den Stellen für den Exponententeil entspricht.
	
	`PRINT USING "##.#^^^^";276`
	`2.8E+02`

Abschlußbeispiel: Dialoggestaltung

10 ... 260	Hauptprogramm
10 ... 60	Initialisierung
70 ... 110	Menü-Angebot
120 ... 130	Eingabeaufforderung
140 ... 200	Auswahl der Eingabe
160	Lern des Tastaturpuffers
210 ... 260	Eingabeauswertung
300 ... 380	Zweig 1, Wurzelberechnung
400 ... 480	Zweig 2, Quadratberechnung
500 ... 560	Unterprogramm zu Zweig 1,2 Bildtitel, Einstellungen des Ausgabeformates
600 ... 660	Zweig 3, Programmende

```
10  --- Dialoggestaltung ---
20 Z=23: S=0
30 KEY OFF
40 SCREEN 1
50 WINDOW
60 CLS
70 LOCATE 22,0,0
80 PRINT STRING$(80,"M");
90 LOCATE 23,6: PRINT "Wurzel";
100 LOCATE 23,36: PRINT "Quadrat";
110 LOCATE 23,68: PRINT "Ende";
120 LOCATE 11,6
130 PRINT "Bitte mit Kursor anwaehlen   ---> ENTER"
140 ' --- Auswahl ---
150 LOCATE Z,S,1
160 IF INKEY$<>"" THEN 160
170 T$=INKEY$
180 IF T$=CHR$(13) THEN 160
190 PRINT T$;
200 GOTO 170
210 Z=CSRLIN: S=POS(0)
220 IF Z <>23 THEN 160
230 IF S>5 AND S<12 THEN 270
240 IF S>35 AND S<43 THEN 360
250 IF S>67 AND S<72 THEN 520
260 GOTO 160
300 ' --- Zweig 1 Wurzel ---
310 E$ = "Wurzel"
320 F$ = "##.###"
330 GOSUB 450
340 FOR I=0 TO 100
350     PRINT USING F$;I,SQR(I)
```

```
360 NEXT
370 PAUSE 50
380 GOTO 30
400 ' --- Zweig 2 Quadrat ---
410 E$ = "Quadrat"
420 F$ = "#####"
430 GOSUB 450
440 FOR I=0 TO 100
450     PRINT USING F$;I,I*I
460 NEXT
470 PAUSE 50
480 GOTO 30
500 # ---UP Titel, Format ---
510 CLS
520 LOCATE 1,28,0: PRINT "Zahl"
530 LOCATE 1,47: PRNT E$
540 WINDOW 3,24,2855
550 F$="###"+STRING$(16," ")+F$
560 RETURN
600 ' --- Zweig 3 Ende ---
610 CLS
620 SCREEN 0
630 LOCATE ,,1
640 KEY ON
650 WINDOW
660 END
```

3.8. Felder

DIM Feldvereinbarung
ERASE Löschen von Feldern
CLEAR Löschen aller Variablen, Einstellen der Speicherbereiche
FRE Größenbestimmung der freien Speicherbereiche

Feldvereinbarung

Mit der Programmgröße nimmt auch die Anzahl der Variablen zu, die im Programm benötigt werden. Variablennamen sind aber maximal 2 Bytes lang. Bei vielen Variablen führt das dazu, daß bald die Übersicht über die Variablenverwendung erschwert wird. Wenn viele zusammengehörige Variable noch in ähnlicher Weise initialisiert werden müssen, ist außerdem der Schreibaufwand unerträglich hoch. Abhilfe schafft hier nur die Verwendung von Feldern bzw. Feldvariablen.
Unter einem Feld versteht man die Zusammenfassung von mehreren Variablen desselben Typs (numerisch oder Zeichenkette) unter einem gemeinsamen Namen, dem Feldnamen. Der Zugriff auf die einzelnen Variablen, die Feldelemente, erfolgt über diesen Feldnamen und einen oder mehrere Indizes. Entsprechend der Anzahl der Indizes nennt man das Feld ein- oder entsprechend mehrdimensional. Da die Indizes als numerische Ausdrücke angegeben werden können, läßt sich der Zugriff auf die einzelnen Feldelemente sehr effektiv programmieren.
Für die meisten praktischen Probleme sind ein- und zweidimensionale Felder ausreichend, mehr Dimensionen lassen sich aber angeben.

Format: **DIM** *feldname*(*maxindex*[,*maxindex*]...)[,*feldname*(...),...]

feldname - Name der Feldvariablen, Typ entsprechend Suffix (vgl. Abschnitt 3.2, 3.6)

maxindex - numerischer Ausdruck, maximaler Indexwert für diese Dimension

Funktion: Ein Speicherbereich für die geforderte Anzahl von Feldelementen wird unter *feldname* reserviert, ihr Typ entspricht dem Suffix von *feldname* oder dem geltenden Standard (vgl. Abschnitt 3.6).

Hinweise:

1. Die Zählung der Indexwerte beginnt immer bei Null.
2. Felder müssen nicht immer expliziert dimensioniert werden. Wird die DIM-Anweisung weggelassen, so nimmt RBASIC mit dem ersten Auftreten des Feldelementes im Programm eine implizite Vereinbarung vor, *maxindex* ist für jede verwendete Dimension dann 10.
3. Felder lassen sich nicht umdimensionieren.
4. Nach der Dimensionierung haben alle Elemente eines numerischen Feldes den Wert Null. Elementen von Zeichenkettenfeldern ist die leere Zeichenkette (Länge Null) zugeordnet.

Beispiel:

1. In dem eindimensionalen Feld W werden einigen Feldelementen Werte zugewiesen, anschließend wird das ganze Feld ausgegeben.

```
10 DIM W(5)
20 FOR I=0 TO 3
30 W(I)=10+I
40 NEXT I
50 FOR I=0 TO 5
60 PRINT "W(";I;")=";W(I)
70 NEXT I
80 END

RUN

W( 0 )= 10
W( 1 )= 11
```

```
W( 2 )= 12
W( 3 )= 13
W( 4 )= 0
W( 5 )= 0
```

2. Mit der folgenden DIM-Anweisung werden 16*11=176 Variable vom Typ reell, einfach genau vereinbart, und das Feld hat den gezeigten Aufbau:

```
DIM R!(15,10)
```

R!(0,0)	R!(0,1)	R!(0,2)	...	R!(0,10)
R!(1,0)	R!(1,1)	R!(1,2)	...	R!(1,10)
.	.	.		.
.	.	.		.
.	.	.		.
R!(15,0)	R!(15,1)	R!(15,2)	...	R!(15,10)

3. Für eine Personendatei sollen ein zweidimensionales Zeichenkettenfeld für Name und Vorname und ein numerisches ganzzahliges Feld für das Alter vereinbart werden. Die Anzahl der zu erfassenden Personen soll variabel sein.

```
10 INPUT "Wieviel Personen?";N
20 IF N<1 THEN 10
30 DIM NA$(N-1,1),AL%(N-1)
40 FOR I=0 TO N-1
50 INPUT "Name, Vorname, Alter?";NA$(I,0),NA$(I,1),AL%(I)
60 NEXT I
70 FOR I=0 TO N-1
80 PRINT NA$(I,0);" ";NA$(I,1),AL%(I)
90 NEXT
```

Löschen von Feldern

Am Aufbau und der Größe einmal vereinbarter Felder läßt sich nichts mehr ändern. Versuche einer erneuten Dimensionierung (Umdimensionierung) werden mit "Redimensioned array" abgelehnt. Soll ein Feld in einem Programm dennoch anders verwendet werden, so bleibt nur übrig, das Feld komplett aus der Variablenliste zu streichen und neu zu dimensionieren. Dadurch gehen natürlich alle Werte des ehemaligen gleichnamigen Feldes verloren. Durch das Löschen von Feldern kann auch Platz für andere neu zu dimensionierende Felder geschaffen werden.

Format: **ERASE** *feldname*[,*feldname*]...

Funktion: Die angegebenen Felder werden gelöscht, der von Ihnen belegte Speicherbereich wird freigegeben, die angegebenen Feldnamen werden aus der Variablenliste gestrichen.

Hinweis: Anschließend können die Parameter *feldname* in DIM-Anweisungen neu verwendet werden.

Beispiel:

```
10 DIM A(10),X$(25)
20 ERASE A,X$
30 DIM A(20),X$(100)
40 ERASE A
50 ERASE X$
```

Löschen aller Variablen, Einstellen der Speicherbereiche

Anhang E enthält eine Übersicht zur Speicheraufteilung durch RBASIC (siehe auch Abschnitt 9). Diese wird eingestellt mit dem Start des Interpreters. Der Anwenderspeicher, das ist der Speicher, der für das RBASIC-Programm, die numerischen und Zeichenkettendaten, die Dateikontrollblöcke (FCBs) und den Kellerspeicher (Stack) des Interpreters zur Verfügung steht, wird initialisiert. Die obere Grenze des Anwenderspeichers wird auf 0E000H eingestellt; es werden zwei FCBs angelegt (entspricht MAXFILES=1), einer für RBASIC selbst (SAVE/LOAD) und einer für die Dateiarbeit des Anwenders, der Zeichenkettenspeicherbereich erhält eine Länge von 200 Bytes, der Stack wird innerhalb dieses Bereiches angelegt, der Programm- und Datenbereich wird geleert.
Die Einstellung und Belegung dieses Anwenderspeichers kann nun durch verschiedene Anweisungen beeinflußt werden. NEW (siehe Abschnitt 2.3) leert den Programm- und Datenbereich, läßt aber die Anzahl der FCBs und die Länge des Zeichenkettenspeicherbereichs unverändert.

Format: **CLEAR** [*zeichenbereich*[,*endadr*]]

zeichenbereich - Länge des Zeichenkettenspeicherbereichs, ganzzahliger numerischer Ausdruck von 0 bis maximal 23526, Standard: 200

endadr - Adresse des ersten freien Speicherplatzes nach dem Anwenderspeicher von RBASIC, ganzzahliger numerischer Ausdruck von 33817 bis 57344, Standard: 57344

Funktion: Mit CLEAR werden alle bisher verwendeten bzw. vereinbarten einfachen und Feldvariablen gelöscht. Numerische Variable erhalten den Wert Null, Zeichenkettenvariablen wird die leere Zeichenkette (Länge Null) zugewiesen. Ein im Speicher stehendes Programm wird nicht verändert. Ohne Parameter bleibt die Einteilung des Anwenderspeichers erhalten.
Mit der Angabe von *zeichenbereich* wird zusätzlich der Zeichenkettenspeicherbereich auf diese Länge eingestellt
Ist *endadr* angegeben, so wird die Obergrenze des Anwenderspeichers neu eingestellt.

Hinweise:

1. Theoretisch sind Werte von 0 bis 65535 für die Parameter zulässig, andere als die angegebenen werden aber mit "Out of memory" abgelehnt.

2. Der Speicherbereich zwischen *endadr* und dem Standardwert 57344 (&HE000) kann für andere Zwecke verwendet werden (siehe Abschnitt 9).

3. Mit CLEAR wird gleichzeitig ein RESTORE ausgeführt (vgl. Abschnitt 3.9).

4. CLEAR schließt alle eventuell noch offenen Dateien (CLOSE, vgl. Abschnitt 6.3).

Größenbestimmung der freien Speicherbereiche

Über die Belegung des Anwenderspeichers müssen Sie sich spätestens dann informieren, wenn RBASIC die Meldung "Out of memory" ausgibt. Dann reicht der noch vorhandene freie Speicher nicht mehr aus, Ihr Problem zu bearbeiten.
Als Gründe für diese Meldungen sind möglich:

- das Programm ist zu lang
- ein zu vereinbarendes Feld ist zu groß
- zu viele oder zu lange Zeichenketten sollen gespeichert werden.

Format: **FRE** (*argument*)

argument - beliebiger numerischer oder Zeichenkettenausdruck

Typ: BASIC-Funktion

Funktion: Entsprechend dem Typ von *argument* wird die Größe des noch freien Speicherbereiches bestimmt:

numerisch - freier Bereich im Programm- und Variablenteil des Anwenderspeichers (vgl. Anhang E),

Zeichenkette - freier Bereich im Zeichenkettenspeicherbereich.

Hinweise:

1. Der Wert von *argument* wird nicht ausgewertet, nur der Typ (numerisch oder Zeichenkette)

2. Der Funktionswert ist eine Zahl zwischen 0 und 23673, Er ist abhängig von der Programmlänge, der Anzahl und dem Typ der Variablen, der Gesamtlänge des Zeichenkettenspeicherbereiches und der Anzahl der gleichzeitig zu bearbeitenden Dateien.

3. Standardmäßig nach dem Start von RBASIC, Disk Version, ist eingestellt.

   Größe des Zeichenkettenspeicherbereiches 200 Bytes,
   Größe des freien Bereiches (vgl. Anhang E) 23473 Bytes.

   Wird nur RBASIC (ohne Diskette) gestartet, so beträgt die Größe des freien Bereiches 28473 Bytes.

4. Die obere Grenze des Anwenderspeichers (2. Parameter von CLEAR) ist nicht abfragbar. Sie kann nur eingestellt werden.

3.9. Interne Daten und Zufallszahlen

DATA Vereinbarung von Daten
READ Lesen von Daten
RESTORE Setzen des DATA-Zeigers
RND Erzeugung einer Zufallszahl

Zur Zuweisung von Werten zu Variablen haben Sie schon die LET- und die INPUT-Anweisung kennengelernt. Jetzt soll Ihnen mit den Anweisungen DATA, READ und RESTORE eine Möglichkeit gezeigt werden, die es Ihnen leicht gestattet, einer größeren Anzahl von Variablen konstante Werte zuzuweisen. Gegenüber LET bieten diese Anweisungen bei größeren Datenmengen den Vorteil, daß ihre Anwendung zu kürzeren und unübersichtlichen Programmen führt. Eine Programmunterbrechung zur Eingabe ähnlich wie bei der INPUT-Anweisung erfolgt nicht.

Format: **DATA** *konstante*[,*konstante*] ...

konstante - numerische Konstante oder Zeichenkettenkonstante

Funktion: Mit den DATA-Anweisungen eines Programms wird eine Dateiliste, die numerische Konstanten und Zeichenkettenkonstanten enthält, erzeugt. In dieser Datenliste sind die Konstanten in der Reihenfolge ihres Auftretens in den DATA-Anweisungen nacheinander abgespeichert. Der Zugriff auf die Konstanten der Datenliste erfolgt ausschließlich mittels der READ-Anweisung.

Hinweise:

1. Die DATA-Anweisung ist eine nichtausführbare Anweisung. Sie ist nur im Programmodus erlaubt.

2. DATA-Anweisungen dürfen nicht in den Anweisungsfolgen einer IF...THEN...ELSE-Anweisung stehen. Ansonsten ist die Stellung der DATA-Anweisungen innerhalb eines Programms beliebig. Beispielsweise können alle DATA-Anweisungen am Schluß eines Programms stehen.

3. Anführungszeichen (") zur Begrenzung von Zeichenkettenkonstanten können entfallen, wenn diese keine Schlüsselwörter, Kommas, Doppelpunkte, führende oder nachfolgende Leerzeichen enthalten.

Format: **READ** *variable*[,*variable*]...

variable - Name einer numerischen Variablen oder Zeichenkettenvariablen

Funktion: Mit einer READ-Anweisung kann nacheinander auf die Konstanten der durch die DATA-Anweisungen erzeugten Datenliste zugegriffen werden. Bei Ausführung einer READ-Anweisung werden fortlaufend Konstanten aus der Datenliste gelesen und den nach READ folgenden Variablen zugewiesen.

Hinweis:

1. Der Typ der Variablen muß mit dem Typ der ihr zugewiesenen Konstanten aus der DATA-Anweisung übereinstimmen.

2. Die Verwaltung der Datenliste erfolgt über einen gedachten DATA-Zeiger. Der DATA-Zeiger zeigt auf die Konstante der Datenliste, die als nächste ausgelesen werden kann. Nach Lesen einer Konstanten wird der DATA-Zeiger auf die folgende Konstante der Datenliste gesetzt. Nach einem Programmstart mit RUN wird der DATA-Zeiger auf die erste Konstante der Datenliste gesetzt.

3. Enthält die Datenliste keine ausreichende Anzahl von Konstanten für die READ-Anweisungen, so erfolgt die Fehlermeldung

```
Out of DATA
```

Beispiel:

```
10 DATA 2000              :'Jahresangabe
20 DATA Völkerschlacht,1813
30 DATA "Pariser Kommune",1871
40 DATA "Gründung der DDR",1949
50 READ J
60 PRINT "Im Jahr";J;"liegt die":PRINT
70 FOR I=1 TO 3
80     READ E$,JA
90     PRINT "- ";E$;J-JA;"Jahre zurück."
100 NEXT I
110 END

RUN
```

Auf dem Bildschirm erscheint nach Start des Programms

```
Im Jahr 2000 liegt die

- Völkerschlacht 187 Jahre zurück.
- Pariser Kommune 129 Jahre zurück.
- Gründung der DDR 51 Jahre zurück.
```

Der Variablen J wird eine Jahresangabe zugewiesen. Anschließend werden die Variablen E$ und JA mit den Bezeichnungen der historischen Ereignisse bzw. den Angaben der Jahre, in denen diese stattgefunden haben, belegt, und die Differenz der Jahre zu J wird ausgegeben.

Format: **RESTORE** [*zeilennummer*]

zeilennummer - Zeilennummer einer DATA-Anweisung

Funktion: Mit der nächsten READ-Anweisung wird auf die erste Konstante der DATA-Anweisung zugegriffen, die in der nach RESTORE angegebenen Programmzeile steht. Fehlt die Zeilennummer, erfolgen die weiteren Zugriffe ab der ersten Konstanten der ersten DATA-Anweisung des Programms.

Hinweise:

1. Mit der RESTORE-Anweisung wird der gedachte DATA-Zeiger auf eine durch die Zeilennummer bestimmte Stelle der Datenliste gesetzt.

2. Steht in der Zeile mit der angegebenen Zeilennummer keine DATA-Anweisung, wird mit der Zeilennummer der nächstfolgenden DATA-Anweisung des Programms gearbeitet.

Beispiel:

```
10 DATA "1. Wert",10
20 DATA "2. Wert",20
30 DATA 50,40
40 READ E$,A:PRINT E$,A
50 READ E$,B:PRINT E$,B
60 READ C,A :PRINT C,A:PRINT
70 RESTORE 20
80 READ B$,W:PRINT B$,W
90 END

RUN

1. Wert       10
2. Wert       20
 50           40

2. Wert       20
```

Nacheinander werden die Konstanten aus den DATA-Anweisungen gelesen. In Zeile 70 wird der DATA-Zeiger auf die erste Konstante der DATA-Anweisung in Zeile 20 gesetzt. Daher werden in Zeile 80 der Variablen B$ die Konstante "2. Wert" und der Variablen W der Wert 30 zugewiesen.
Oft sollen in Programmen Variablen zufällige Werte zugewiesen werden. Zum Auslosen oder Auswürfeln der Werte steht in RBASIC die Funktion RND zur Verfügung. Mit ihrer Hilfe kann beispielsweise in Spielprogrammen die Funktion eines Würfels nachgebildet oder in Lehr- und Lernprogrammen die Reihenfolge der gestellten Fragen zufällig gewählt werden. Auch bei Rechenexperimenten zu Vorgängen, die sich nur unbestimmt beschreiben lassen, weil sie vom zufälligen, nicht streng vorhersagbaren Eintreten gewisser Ereignisse abhängen, kann die RND-Funktion angewendet werden.

Format: **RND**(*ausdruck*)

Typ: BASIC-Funktion

Funktion: RND liefert einen zwischen Null und Eins liegenden numerischen Wert (0 <= RND (*ausdruck*) < 1). In Abhängigkeit vom Argument *ausdruck* wird der Wert von RND aus einer Folge von im Intervall [0,1] gleichverteilten Zufallszahlen ermittelt.

Argument (X=Ausdruck)	Erläuterung
x > 0	Als Wert von RND ergibt sich unabhängig vom Argument das nächste Glied der Zufallszahlenfolge. Auch bei gleichem positiven Argument erhält man somit bei jedem Aufruf von RND stets denselben Wert.
x < 0	Jedem negativen Wert ist ein festes Glied der Zufallszahlenfolge zugeordnet, das bei Aufruf von RND mit einem Argument, das diesen Wert besitzt, ausgegeben wird. Bei gleichem negativen Argument erhält man also bei Aufruf von RND stets denselben Wert. Wird danach RND mit positivem Argument aufgerufen, wird das folgende Glied der Zufallszahlenfolge ausgegeben.
x = 0	Der zuletzt ermittelte Wert von RND wird nochmals als Funktionswert geliefert.

Hinweise:

1. Eine ganze Zahl G zwischen dem unteren Wert UW und dem oberen Wert OW wird durch die Anweisung

   G=UW+INT(RND(1)*(OW-UW+1))

   ausgelöst. Für den Würfel ergibt sich mit UW=1 und OW=6

   G=1+INT(6*RND(1))

   als erforderliche Anweisung zur Funktionsnachbildung.

2. Nach jedem Programmstart beginnt die Ausgabe der Zufallszahlenfolge mit demselben Wert. Ist das nicht erwünscht, muß die RND-Funktion zunächst mit einem zufälligen negativen Argument aufgerufen werden. Dieses Argument kann beispielsweise aus dem aktuellen Stundenstand der Systemuhr abgeleitet werden (siehe TIME-Anweisung im Abschnitt 8.5).

Beispiele:

1. Das Beispiel dient zur Demonstration der Wirkung positiver und negativer Argumente.

```
10 W=RND(1):PRINT W:PRINT
20 PRINT RND(-1),RND(1),RND(1):PRINT
30 PRINT RND(-1),RND(0),RND(2):PRINT
40 PRINT RND(-.3049489791414l)
50 END

RUN

 .59521943994623

 .04389820420821
 .0962486816692
 .21069655852301
 .04389820420821
 .04389820420821
 .0962486816692

 0
```

2. Eine zwischen 1 und 1000 liegende Zahl soll geraten werden.

```
10 Z=1+INT(1000*RND(1))
20 INPUT "Zahl     ?";Z1
30 IF Z1<Z THEN PRINT "zu klein":PRINT
40 IF Z1>Z THEN PRINT "zu groß":PRINT
50 IF Z1<>Z THEN GOTO 20
60 PRINT "richtig":PRINT
70 ' ---
80 E$="J":INPUT "Noch einmal  (J)/N ?";E$
90 IF E$="J" THEN GOTO 10
100 END
```

   Versuchen Sie selbst, das Programm so zu erweitern, daß die Anzahl der Rateversuche mitgezählt wird.

3.10. Nutzerfunktionen

DEF FNname	Definition der Nutzerfunktion
FNname	Aufruf der Nutzerfunktion

RBASIC stellt Ihnen eine Reihe von Standardfunktionen zur Verfügung. Daneben können Sie als Nutzer in einem Programm eigene Funktionen definieren. Diese Funktionen, für die Sie durch Angabe eines Ausdrucks bestimmen, wie der Funktionswert zu berechnen ist, können von Ihnen wie

Standardfunktionen eingesetzt werden. Besonders für häufig im Programm zu berechnende Ausdrücke bringt die Verwendung einer Nutzerfunktion erhebliche Vorteile.

Format: **DEF FN*name*** [(*formalparameter*[,*formalparameter*]...)] = *ausdruck*

name - wird wie bei einer Variablen gebildet. Das letzte Zeichen bestimmt den Funktionstyp (% bei einer ganzzahligen Funktion, $ bei einer Zeichenkettenfunktion).

formalparameter - formale numerische Variable oder Zeichenkettenvariable, die im rechtsstehenden *ausdruck* verwendet wird. Es sind maximal neun formale Parameter erlaubt.

ausdruck - numerischer Ausdruck oder Zeichenkettenausdruck

Funktion: Durch die Anweisung wird eine numerische Funktion oder Zeichenkettenfunktion definiert. Der Ausdruck gibt die Vorschrift zur Berechnung des Funktionswertes in Abhängigkeit von den formalen Parametern an.

Hinweise:

1. Die Anweisung DEF FN muß vor der ersten Verwendung der durch sie definierten Funktion FNname durchlaufen werden.

2. Die durch die Anweisung definierte Funktion darf nicht im rechtsstehenden Ausdruck zur Funktionsdefinition verwendet werden, d.h., ein rekursiver Aufruf ist nicht zulässig. Die Länge der Anweisung darf eine RBASIC-Zeile nicht überschreiten.

3. Die Anweisung DEF FN ist keine ausführbare Anweisung. Ihre Anwendung ist nur im Programmodus erlaubt.

4. Die in der Funktionsdefinition angegebenen Parameter sind Platzhalter. Sie werden daher als formale Parameter bezeichnet. Es gibt keine Beziehung zu Variablen mit gleichem Namen, die im übrigen Programm verwendet werden.

Format: **FN*name*** [(*ausdruck*[,*ausdruck*]...)]

name - zweiter Teil des Namens einer durch eine Anweisung DEF FN definierten Funktion

ausdruck - Ausdruck vom selben Typ wie der Typ des formalen Parameters in der zugehörigen Anweisung zur Funktionsdefinition an der entsprechenden Stelle

Typ: BASIC-Funktion

Funktion: Der Funktionswert ergibt sich durch Berechnung des in der zugehörigen Anweisung DEF FN auf der rechten Seite stehenden Ausdrucks. Dabei werden für die formalen Parameter die aktuellen Werte der Ausdrücke verwendet, die anstelle der Parameter beim Funktionsaufruf stehen. Für alle anderen Variablen werden bei der Funktionswertbetrachtung die im übrigen Programm gültigen Werte benutzt.

Hinweis: Bei ganzzahligen Funktionen wird der ganzzahlige Teil des die Funktion definierenden Ausdrucks als Funktionswert geliefert. Der Funktionswert einer Zeichenkettenfunktion ist eine Zeichenkette.

Beispiel: Die Übergabe der Parameter soll Ihnen das folgende Beispiel verdeutlichen.

```
10 DEF FNBSP(A,B)=A*(B+C)
20 A=3:B=4:C=5:D=2
30 PRINT FNBSP(2*D,1)
40 PRINT A,B
50 END
```

```
RUN

 24
 3            4
```

In Zeile 10 wird die Funktion FNBSP mit den formalen Parametern A und B definiert. Beim Aufruf dieser Funktion in der Zeile 30 wird der formale Parameter A durch den Wert von 2*D, also 4 und der formale Parameter B durch 1 ersetzt. Diese Werte werden bei der Berechnung des Funktionswertes anstelle von A und B verwendet. C ist kein formaler Parameter der Funktion. Für ihn wird der in Zeile 20 zugewiesene Wert benutzt. In Zeile 30 wird daher 4*(1+5)=24 als Wert ausgegeben. Die Bestimmung der Werte der formalen Parameter hat keinen Einfluß auf die Werte der Variablen A und B, die durch Zuweisungen in Zeile 20 festgelegt sind. In Zeile 40 wird daher für A der Wert 3 und für B der Wert 4 ausgegeben.

3.11. Unterprogramme

GOSUB Aufruf eines Unterprogramms
RETURN Rücksprung aus einem Unterprogramm
ON...GOSUB Berechneter Aufruf von Unterprogrammen

Der Einsatz von Unterprogrammen (auch "SUBroutinen" genannt) dient vorrangig einer effektiven Programmgestaltung. Er soll

- zu einer klaren und übersichtlichen Programmstruktur führen, die eine Voraussetzung für gute Testbarkeit und Änderungsfreundlichkeit eines Programmes ist.

- kürzere, speicherplatzsparende Programmtexte durch die mehrfache Verwendung nur einmal formulierter Anweisungsblöcke erlauben.

- die Wiederverwendung vorhandener Programmlösungen beim Erarbeiten neuer Programme unterstützen.

Ein Unterprogramm besteht aus einer Folge von RBASIC-Programmzeilen, die mit einer RETURN-Anweisung abgeschlossen wird. Ein Unterprogramm ist für sich allein nicht abarbeitungsfähig. Es Bedarf eines übergeordneten Programms, von dem aus es aufgerufen wird. Das Programm, dem alle Unterprogramme untergeordnet sind, wird Hauptprogramm genannt.

Aufruf eines Unterprogramms

Format: **GOSUB** *zeilennummer*

zeilennummer - Zeilennummer, ab der die Programmabarbeitung fortgesetzt wird

Funktion: Mit dieser Anweisung wird das Unterprogramm aufgerufen, das die RBASIC-Zeile *zeilennummer* enthält. Die Programmabarbeitung wird in der angegebenen Zeile fortgesetzt. Nach Auftreten einer RETURN-Anweisung erfolgt die Rückkehr und Programmfortsetzung mit der Anweisung , die auf die GOSUB-Anweisung folgt, oder mit der nach RETURN angegebenen RBASIC-Zeile.

Hinweise:
1. Eine Parameterübergabe mit Hilfe formaler Parameter wie bei Nutzerfunktionen ist nicht möglich. Alle im übrigen Programm verwendeten Variablen sind auch im Unterprogramm uneingeschränkt gültig. Achten Sie deshalb darauf, daß keine Fehler durch ungewollte Mehrfachnutzung von Variablen entstehen. Fügen Sie ggf. vor Ihr Unterprogramm Kommentare ein, aus denen die verwendeten Variablen ersichtlich sind.

2. Aus einem Unterprogramm können weitere Unterprogramme aufgerufen werden.

3. In der Regel sollten Programme so erstellt werden, daß die Anweisungen des Hauptprogramms den Anfang bilden. Danach folgen die Anweisungen der einzelnen Unterprogramme. Am Schluß des Hauptprogramms muß die END-Anweisung stehen, da sonst das nachfolgende Unterprogramm bei der Programmabarbeitung ungewollt erreicht wird.

Format: **RETURN** [*zeilennummer*]

zeilennummer - RBASIC-Programmzeile, in der die Programmabarbeitung fortgesetzt wird.

Funktion: Das Ende des Unterprogramms ist erreicht. Die Programmabarbeitung wird im untergeordneten Programmteil fortgesetzt. Wenn die Angabe der zeilennummer fehlt, erfolgt die Fortsetzung mit der auf GOSUB folgenden Anweisung, andernfalls mit der angegebenen Zeile.

Hinweis: Zu jeder RETURN-Anweisung, mit der ein Unterprogramm verlassen wird, gehört während der Programmabarbeitung eine GOSUB-Anweisung, mit der das Unterprogramm erreicht worden ist. Ist das nicht der Fall erfolgt eine Fehlermeldung.

Beispiel:

```
10 ' --- Hauptprogramm ---
20 N=0:A=1:B=10
30 PRINT "Hauptprogrammanfang":PRINT
40 GOSUB 130
50 PRINT "Hauptprogrammzeile 50":PRINT
60 GOSUB 250
70 PRINT "Hauptprogrammende"
80 END
90 ' ---
100 '--- Unterprogramm 1 ---
110 '--- Eingabe: A, B, N ---
120 '--- Ausgabe: A, B, N ---
130 N=N-1
140 PRINT TAB(3*N);"Unterprogramm 1"
150 PRINT TAB(3*N);"A = ";A:PRINT TAB(3*N);"B =";B
160 C=A+B
170 PRINT TAB(3*N);"C =";C:PRINT
180 GOSUB 250
190 N=N-1
200 RETURN
210 '---
220 '--- Unterprogramm 2 ---
230 '--- Eingabe: A, N ---
240 '--- Ausgabe: A, N ---
250 N=N+1
260 PRINT TAB(3*N);"Unterprogramm 2"
270 PRINT TAB(3*N);"A =";A:PRINT
280 A=A+1
290 PRINT TAB(3*N);"A =";A:PRINT
300 N=N+1
310 RETURN
```

Anhand der Ausschriften des Programms können Sie die Unterprogrammaufrufe und die Änderungen der Variablen verfolgen. N dient zur Kennzeichnung der Aufrufebene.

```
Hauptprogrammanfang

   Unterprogramm 1
   A = 1
   B = 10
   C = 11

      Unterprogramm 2
      A = 1
      A = 2
```

```
Hauptprogrammzeile 50

   Unterprogramm 2
   A = 2
   A = 3

Hauptprogrammende
```

Berechneter Aufruf von Unterprogrammen

Format: **ON** *ausdruck* **GOSUB** *zeilennummer* [,*zeilennummer*]...

ausdruck - numerischer Ausdruck mit einem Wert größer als oder gleich Null, dessen ganzer Anteil benutzt wird

zeilennummer - RBASIC-Zeilennummer des aufzurufenden Unterprogramms

Funktion: Der ganzzahlige Teil des Ausdrucks wird bestimmt und sei I. Das Programm wird dann durch Eintritt in ein Unterprogramm bei der Zeilennummer fortgesetzt, die an I-ter Stelle in der angegebenen Zeilennummernliste steht.
Nach Auftreten einer RETURN-Anweisung erfolgt die Rückkehr und Programmfortsetzung mit der Anweisung, die auf die ON...GOSUB-Anweisung folgt. Steht nach Return eine Zeilennummer, wird das Programm in der angegebenen Zeile fortgesetzt.

Hinweis: Ist der ganzzahlige Wert des Ausdrucks gleich Null oder größer als die Anzahl der in der ON...GOSUB-Anweisung angegebenen Zeilennummern, wird das Programm mit der auf ON...GOSUB folgenden Anweisung fortgesetzt.

Beispiel: Im Hauptprogramm kann durch den Nutzer die auszuführende Aufgabe ausgewählt werden. Zur Ausführung der Aufgabe (Quader- oder Kugeloberflächenberechnung) wird dann ein Unterprogramm aufgerufen. Dabei werden von den Unterprogrammen für die Abarbeitung einzelner Teilprobleme (Werteingabe, Fehlermeldung, Programmunterbrechung) wieder Unterprogramme verwendet.
Mit den Anweisungen KEY OFF und KEY ON wird die Anzeige der Funktionstastenbelegung in Zeile 20 aus- bzw. in Zeile 120 wieder eingeschaltet.

```
10 ' --- Hauptprogramm ---
20 KEY OFF:PI=3.141592653
30 WINDOW:CLS:LOCAT 8,4
40 PRINT "1 Quaderoberfläche":PRINT
50 PRINT TAB(4);"2 Kugeloberfläche":PRINT
60 PRINT TAB(4);"3 Ende"
70 WINDOW 20,20,4,39
80 F%=1:INPUT "Funktion:  ";F%
90 IF F%<0 OR F%>3 THEN GOTO 80
100 ON F% GOSUB 150,190
110 IF F%<>3 THEN GOTO 30
120 WINDOW:CLS:KEY ON
130 END
140 ' --- UP Quaderoberflächenberechnung ---
150 T$="Seitenlänge/cm":GOSUB 250
160 PRINT "Quaderoberfläche/cm^2";TAB(23);"=";6*W^2:GOSUB 310
170 RETURN
180 ' --- UP Kugeloberflächenberechnung ---
190 T$="Radius/cm":GOSUB 250
200 PRINT "Kugeloberfläche/cm^2";TAB(23);"="
210 PRINT USING "##.###^^^^";4*PI*W^2:GOSUB 310
220 RETURN
230 ' --- UP Werteingabe ---
240  '       Eingabe: T$        Ausgabe: W
250 WINDOW:CLS:LOCATE 10,0
260 W=-1:PRINT T$;TAB(23);"=";
```

```
270 INPUT "";W:PRINT
280 IF W<0 THEN GOSUB 330:GOTO 250
290 RETURN
300 ' --- UP Programmunterbrechung ---
310 PRINT:PRINT TAB(25);:INPUT ">ENTER<";E$:RETURN
320 ' --- UP Fehlermeldung ---
330 PRINT "Wert darf nicht kleiner Null sein!":GOSUB 310
340 RETURN
```

3.12. Funktionen zur Arbeit mit Zeichenketten

LEN	Länge einer Zeichenkette
LEFT$)	
RIGTH$)	Übernahme einer Teilzeichenkette
MID$)	
MID$	Verändern einer Zeichenkette
INSTR	Suche einer Zeichenkette in einer anderen Zeichenkette
STRING$	Wiederholung eines Zeichens
SPACE$	Wiederholung eines Leerzeichens
ASC	Bestimmung des Codes eines Zeichens
CHR$	Zeichen zu gegebenem Code
VAL	Bilden einer dezimalen Zahl aus gegebenen Zeichenketten
STR$	Zeichenkette mit Dezimaldarstellung einer gegebenen Zahl
BIN$	Zeichenkette mit Binärdarstellung einer gegebenen Zahl
HEX$	Zeichenkette mit Hexadezimaldarstellung einer gegebenen Zahl
OCT$	Zeichenkette mit Octaldarstellung einer gegebenen Zahl

Diese Funktionen bieten eine wirksame Unterstützung bei der Arbeit mit Zeichenketten. Ihre Benutzung erfolgt in der gleichen Weise wie die der numerischen Standardfunktionen. Als Argumente sind, wie für die einzelnen Funktionen im folgenden beschrieben wird, numerische Ausdrücke und Zeichenkettenausdrücke möglich.

Endet der Name der Funktion mit dem Zeichen $, ist der Funktionswert eine Zeichenkettenkonstante. Diese kann in einem Zeichenkettenausdruck unter Verwendung des Operationszeichens + mit anderen Zeichenketten verkettet werden. Funktionen, deren Funktionswert eine Zeichenkette ist, können aber auch überall dort allein stehen, wo die Verwendung eines Zeichenkettenausdruckes verlangt wird.

Eine numerische Konstante wird als Funktionswert ermittelt, wenn der Funktionsname nicht mit einem $-Zeichen endet. In diesen Fällen können die Funktionen wie die übrigen numerischen Standard-funktionen verwendet werden.

Länge einer Zeichenkette

Format: **LEN**(*zeichenkette*)

zeichenkette - Zeichenkettenausdruck

Typ: BASIC-Funktion

Funktion: Die Anzahl der in der Zeichenkette enthaltenen Zeichen (einschließlich nicht darstellbarer Zeichen und Leerzeichen) wird als Funktionswert geliefert. Die leere Zeichenkette, die kein Zeichen enthält, hat die Länge Null.

Beispiele:
```
PRINT LEN("ABCD"),LEN("")
4                 0
```

Im folgenden Beispiel wird die Verwendung der Funktion zur rechtsbündigen Ausgabe einer Folge von Zeichenketten demonstriert.

```
10 DATA Rechtsbündige,Ausgabe,von,Zeichenketten,.
20 FOR I=1 TO 5
30     READ T$:PRINT TAB(15-LEN(T$));T$
40 NEXT I
50 END
```

Übernahme einer Teilzeichenkette

Format: **LEFT$**(*zeichenkette*,*z*)

zeichenkette - Zeichenkettenausdruck

z - Numerischer Ausdruck, dessen Wert die Anzahl der aus der Zeichenkette (von links beginnend) zu übernehmenden Zeichen bestimmt. Der ganzzahlige Teil des Wertes muß zwischen 0 und 255 liegen.

Typ: BASIC-Funktion

Funktion: Als Funktionswert ergibt sich eine Zeichenkette, die die ersten *z* Zeichen der als Argument verwendeten Zeichenkette enthält.

Hinweis: Ist die geforderte Zeichenanzahl größer als die vorhandene, wird die gesamte Zeichenkette übernommen. Falls die gewünschte Zeichenanzahl Null ist, ergibt sich als Funktionswert die leere Zeichenkette, die kein Zeichen enthält.

Beispiel:
```
PRINT LEFT$("ABCD",2)
AB
```

Format: **RIGTH$**(*zeichenkette*,*z*)

zeichenkette - Zeichenkettenausdruck

z - Numerischer Ausdruck, dessen Wert die Anzahl der aus der Zeichenkette (rechts stehenden) zu übernehmenden Zeichen bestimmt. Der ganzzahlige Teil des Wertes muß zwischen 0 und 255 liegen.

Typ: BASIC-Funktion

Funktion: Als Funktionswert ergibt sich eine Zeichenkette, die die letzten *z* Zeichen der als Argument verwendeten Zeichenkette enthält.

Hinweis: Ist *z* größer als die Länge der auszuwertenden Zeichenkette, wird die gesamte Zeichenkette übernommen. Nimmt *z* den Wert Null an, ergibt sich als Funktionswert die leere Zeichenkette.

Beispiel:
```
PRINT RIGHT$("ABCD",3)
BCD
```

Format: **MID$**(*zeichenkette*,*ab_position*[,*z*])

zeichenkette - Zeichenkettenausdruck

ab_position - Position des ersten zu übernehmenden Zeichens der Zeichenkette; numerischer Ausdruck mit Wert zwischen 1 und 255

z - Anzahl der aus Zeichenkette (von *ab_position* nach rechts fortschreitend) zu übernehmenden Zeichen; numerischer Ausdruck zwischen 0 und 255

Typ: BASIC-Funktion

Funktion: Der Funktionswert wird aus dem Teil der Zeichenkette im Argument gebildet, der mit dem an der *ab_position* stehenden Zeichen beginnt. Ist die Zeichenanzahl angegeben, werden ab dieser Position maximal *z* Zeichen übernommen.

Hinweise:
1. Liegt die *ab_position* hinter dem letzten vorhandenen Zeichen, wird die leere Zeichenkette geliefert.

2. Ist die geforderte Zeichenanzahl ab der vorgegebenen Position größer als die Zahl der vorhandenen Zeichen, erhält man als Funktionswert die gesamte restliche Zeichenkette.

Beispiel:
```
A$="ABCDEF":PRINT MID$(A$,3,2)
CD
```

Verändern einer Zeichenkette

MID$ kann auch, abweichend von allen übrigen Funktionen, auf der linken Seite einer Zuweisung stehen. MID$ bewirkt dann allerdings nicht die Ausgabe einer Teilzeichenkette, sondern erlaubt, eine vorgegebene Zeichenkette teilweise zu ändern.

Format: **MID$**(*zeichenkette*,*n*,[*z*])=*neuer_teil*

zeichenkette - Zeichenkettenvariable

n - Numerischer Ausdruck, dessen Wert zwischen 1 und der Länge von *zeichenkette* liegen muß.

z - Numerischer Ausdruck, dessen Wert zwischen 1 und der Länge von *zeichenkette* liegen muß.

neuer_teil - Zeichenkettenausdruck

Funktion: Das n-te Zeichen und die folgenden *z* Zeichen von *zeichenkette* werden durch die ersten *z* Zeichen der Zeichenkette *neuer_teil* ersetzt. Danach wird *zeichenkette* erforderlichenfalls auf die ursprüngliche Länge reduziert.

Beispiel:
```
10 A$="ABCDEFG":B$="1234"
20 MID$(A$,3,2)=B$
30 PRINT A$

RUN

AB12EFG
```

Suchen einer Zeichenkette in einer anderen Zeichenkette

Format: **INSTR**(*zeichenkette*,*testzeichenkette*)

zeichenkette - Zeichenkettenausdruck
testzeichenkette - Zeichenkettenausdruck

Typ: BASIC-Funktion

Funktion: Es wird ermittelt, ob *testzeichenkette* vollständig in *zeichenkette* enthalten ist. Als Funktionswert wird die Position des ersten Zeichens von *testzeichenkette* in *zeichenkette* ermittelt. Wird *testzeichenkette* nicht gefunden, ergibt sich der Funktionswert Null.

Hinweis: Ist *zeichenkette* die leere Zeichenkette, erhält man als Funktionswert Null, wenn *testzeichenkette* ebenfalls die leere Zeichenkette ist. Andernfalls ist der Funktionswert Eins.

Beispiele: 1.
```
T$="Beispiel":PRINT INSTR(T$,"IE")
 6
```

2.
```
10 T$="Meier, Fritz"
20 N$=LEFT$(T$,INSTR(T$,",")-1)
30 PRINT N$

RUN

Meier
```

Es wird der vor dem Komma stehende Teil der Zeichenkette T$ ausgegeben.

Bestimmung des Codes eines Zeichens

Zeichen und Zeichenketten werden nicht direkt im Rechner gespeichert. Dazu ist ein Zwischenschritt erforderlich. Jedem Zeichen wird in der Regel eine bestimmte Zahl zwischen 0 und 255 zugeordnet. Es gibt hier natürlich viele verschiedene Zuordnungsmöglichkeiten. Die Zuordnungen werden auch Codierungen genannt. Weit verbreitet ist die Codierung gemäß ASCII (American Standard Code for Information Interchange = amerikanische Standardcodierung zum Informationsaustausch) für die Zuordnung zu den Zahlen zwischen 0 und 127. Im Anhang A wird ein Überblick über die in Ihrem Computer verwendete Codierung gegeben, die auf der ASCII-Regelung aufbaut. Der einem Zeichen zugeordnete Wert wird im folgenden als Code des Zeichens bezeichnet.

Es wird zwischen darstellbaren und nicht darstellbaren Zeichen unterschieden. Die darstellbaren Zeichen liefern bei Ausgabe auf dem Bildschirm oder Drucker das angegebene Bild. Nicht darstellbare Zeichen können Sie bei einer Ausgabe nicht sehen. Durch sie werden die im Anhang A angeführten Wirkungen erreicht. Beispielsweise bewirkt die Ausgabe des Zeichens mit dem Code 12 ein Löschen des Bildschirms.

Eine besondere Rolle spielt das nicht darstellbare Zeichen mit dem Code 1. Seine Ausgabe hat zur Folge, daß als folgendes Zeichen ein Grafiksonderzeichen entsprechend der Tabelle für die 2-Byte-Zeichen ausgegeben wird. Das Zeichen mit dem Code 1 bewirkt also ein Umschalten zwischen der normalen Zeichendarstellung und den Grafiksonderzeichen. Auf diese Weise wird der Umfang der darstellbaren Zeichen erweitert.

Format: **ASC**(*zeichenkette*)

zeichenkette - Zeichenkettenausdruck

Typ: BASIC-Funktion

Funktion: Als Funktionswert ergibt sich der Code des ersten Zeichens der als Argument verwendeten Zeichenkette (vgl. Anhang A).

Hinweise: 1. Steht als Argument die leere Zeichenkette, erfolgt bei Funktionsabarbeitung eine Fehlermeldung.

2. Ist das erste Zeichen ein 2-Byte-Zeichen (vgl. Anhang A), wird Eins als Funktionswert geliefert.

Beispiel:
```
PRINT ASC("ABC")
 65
```

65 ist der Code von A.

Zuordnung eines Zeichens zu gegebenem Code

Format: **CHR$(***ausdruck*)

ausdruck - numerischer Ausdruck, dessen ganzzahliger Wert zwischen 0 und 255 liegen muß

Typ: BASIC-Funktion

Funktion: Als Funktionswert wird das Zeichen geliefert, dessen Code gleich *ausdruck* ist.

Beispiel:
```
PRINT CHR$(65)
A
```

Es wird das Zeichen A auf dem Bildschirm ausgegeben, da das Zeichen A den dezimalen Code 65 hat.

Schauen Sie sich noch einmal die darstellbaren Zeichen des Zeichensatzes auf dem Bildschirm an.

```
10 CLS
20 PRINT "Alpha- und Sonderzeichen":PRINT
30 For I=32 TO 126
40     PRINT CHR$(I)
50 NEXT I
60 PRINT:PRINT:PRINT
70 PRINT "Sonder- und Grafikzeichen":PRINT
80 FOR I=128 TO 255
90     PRINT CHT$(I)
100 NEXT I
110 PRINT:PRINT:PRINT
120 PRINT "2-Byte-Zeichen":PRINT
130 FOR I=65 TO 95
140     PRINT CHR$(1);CHR$(I)
150 NEXT I
160 PRINT
170 END
```

Wiederholung einer Zeichenkette

Format 1: **STRING$**(*wiederholungen*,*zeichenkette*)

wiederholungen - numerischer Ausdruck, dessen ganzzahliger Wert festlegt, wie oft das erste Zeichen der Zeichenkette zu wiederholen ist. Wert zwischen 0 und 255.

zeichenkette - Zeichenkettenausdruck

Typ: BASIC-Funktion

Funktion: Als Funktionswert wird eine Zeichenkette gebildet. Diese ergibt sich durch die angegebene Anzahl von Wiederholungen des ersten Zeichens der als Argument verwendeten Zeichenkette.

Hinweis:
1. Die Länge der erzeugten Zeichenkette darf die Größe des Zeichenkettenspeicherbereiches (Standard: 200 Zeichen) nicht überschreiten. Die Größe dieses Bereiches kann mit der CLEAR-Anweisung verändert werden.

2. Ist die Zahl der Wiederholungen Null, ergibt sich als Ergebnis die leere Zeichenkette.

Beispiel:
```
PRINT STRING$(3,"ABCD")
AAA
```

Für den Fall, daß der Code eines zu wiederholenden Zeichens bekannt ist, kann auch die folgende Anweisung verwendet werden.

Format 2: **STRING$**(*wiederholungen*,*code*)

wiederholungen - numerischer Ausdruck, dessen ganzzahliger Wert festlegt, wie oft das Zeichen mit dem angegebenen Code wiederholt werden soll. Wert zwischen 0 und 255.

code - numerischer Ausdruck, dessen Wert den Code des zu wiederholenden Zeichens bestimmt. Wert zwischen 0 und 255.

Typ: BASIC-Funktion

Funktion: Das Zeichen mit dem angegebenen *code* wird so oft verfielfacht, wie der Parameter *wiederholungen* angibt , und als Funktionswert bereitgestellt.

Hinweis: STRING$(W,N) und STRING$(W,CHR$(N)) liefern dasselbe Ergebnis.

Beispiel: Unterstreichen Sie eine Überschrift durch Wiederholung des Zeichens mit dem Code 205. Die LEN-Funktion nimmt Ihnen die Mühe ab, die Zahl der erforderlichen Wiederholungen zu ermitteln.

```
10 UE$="Sonder- und Grafikzeichen"
20 PRINT UE$
30 PRINT STRING$(LEN(UE$),205)
```

Sollen nur Leerzeichen wiederholt werden, kann die folgende Anweisung genutzt werden.

Format: **SPACE$**(*wiederholungen*)

wiederholungen - numerischer Ausdruck, dessen ganzzahliger Wert festlegt, wie oft das Leerzeichen wiederholt werden soll. Wert zwischen 0 und 255.

Typ: BASIC-Funktion

Funktion: Die Zeichenkette, die als Funktionswert gebildet wird, enthält die angegebene Anzahl von Leerzeichen

Hinweis: Die Aufrufe SPACE$(W) und STRING$(W," ") sowie STRING$(W,32) liefern denselben Funktionswert.

Beispiel: Aus zwei Zeichenketten A$ und B$ soll eine Zeichenkette C$ mit 40 Zeichen gebildet werden, die durch Einfügen einer entsprechenden Anzahl von Leerzeichen zwischen A$ und B$ entsteht.

```
10 A$="LEER":B$="ZEICHEN"
20 C$=A$+SPACE$(40-LEN(A$)-LEN(B$))+B$
30 PRINT A$:PRINT B$
40 PRINT:PRINT C$
```

Als Zeichenkette können Sie auch eine Folge von Zeichen definieren, die sich als Zahl lesen läßt (z.B. "-123" oder "&B101"). Eine derartige Zeichenkette kann aber nicht in numerischen Ausdrücken verwendet werden.

Der Umwandlung von Zeichenketten, die Zahlen darstellen, in numerische Werte und den umgekehrten Schritt dienen die beiden folgenden Anweisungen.

Wandlung von Zeichenkette in Zahl und umgekehrt

Format: **VAL**(*zeichenkette*)

zeichenkette - Zeichenkettenausdruck, dessen Wert eine Zahl darstellt

Typ: BASIC-Funktion

Funktion: Als Funktionswert wird die Zahl geliefert, die der im Argument übergebenen Zeichenkette entspricht. Führende Leerzeichen werden ignoriert.
Kann nicht die gesamte Zeichenkette als Zahl interpretiert werden, werden möglichst viele links stehende aufeinander folgende Zeichen, die eine Zahlendarstellung ergeben, zur Funktionswertermittlung verwendet. Die folgenden Zeichen werden nicht ausgewertet.

Hinweis: Die leere Zeichenkette oder eine Zeichenkette, die keine einer Zahl entsprechende linke Teilzeichenkette enthält (z.B. "P 14") liefert den Funktionswert Null.

Beispiele:
```
A$="1234":PRINT VAL(A$)+1000
 2234

A$="5678":PRINT VAL(A$+".5")
 5678.5

PRINT VAL("5152-45-56047")
 5152
```

Im letzten Beispiel wird nur der linke Teil der Zeichenkette bis zum ersten Bindestrich ausgewertet.

Format: **STR$**(*ausdruck*)

Typ: BASIC-Funktion

Funktion: Als Funktionswert wird die Zeichenkette geliefert, die den Wert des im Argument angegebenen numerischen Ausdruckes in dezimaler Darstellung enthält.

Hinweis: Wenn der Wert des numerischen Ausdruckes nicht negativ ist, bekommt die Zeichenkette ein Leerzeichen anstelle eines Vorzeichens vorangestellt.

Beispiele:

1. Eine Summe wird in eine Zeichenkette umgewandelt.

```
PRINT STR$(1234+1000)
 2234
```

2. Die Zeichenkettendarstellung ganzer Zahlen erfolgt nicht in Exponentialform. Damit kann für diese Zahlen das folgende einfache Programm zur Quersummenberechnung verwendet werden.

```
10 PRINT "Zahl       =";Z%
20 Q=0
30 Z$=STR$(Z%)
40 FOR I=2 TO LEN(Z$)
50     Q=Q+VAL(MID$(Z$,I,1))
60 NEXT I
70 PRINT "Quersumme =";Q
80 END
```

Versuchen Sie selbst, das Programm mit möglichst geringem Aufwand so abzuändern, daß die Summe der Quersumme aller Zahlen von 1 bis zu einer eingegebenen positiven Zahl berechnet wird. Sie erhalten dann ein weiteres Beispiel für geschachtelte Schleifen.

Für die Summe der Quersummen aller Zahlen von 1 bis 100 lautet das Ergebnis 901. Übrigens gibt es einen einfachen Weg, diesen Wert direkt zu bestimmen. Vielleicht finden sie ihn.

Verschiedene Zahlendarstellungen

Für Zahlendarstellungen wird heute üblicherweise ein Positionssystem, wie es auch das gebräuchliche Zahlensystem darstellt, verwendet. Die Zeichen innerhalb einer solchen Darstellung haben verschiedene Stellenwerte.

Beim Dezimalsystem erhält man den Zahlenwert als eine Summe von Zehnerpotenzen. Beim Binärsystem ergibt sich der Zahlenwert als eine Summe von Potenzen der Zahl 2, multipliziert mit den Werten 0 oder 1, die an den zugehörigen Stellen stehen. Binärzahlen sind durch die vorangestellten Zeichen &B zu kennzeichnen. Beim Oktalsystem ist die 8 die Basis für die Zahlendarstellung. Die Konstanten werden durch &O gekennzeichnet. Die Basis der mit &H gekennzeichneten Hexadezimalzahlen ist 16.

RBASIC verfügt über Zeichenkettenfunktionen, die für numerische Ausdrücke Zeichenketten, die den Zahlendarstellungen in den angegebenen Positionssystemen entsprechen, ermitteln. Die STR$-Funktion, die die dezimale Darstellung liefert, haben Sie schon kennengelernt.

Die folgenden drei Funktionen dienen der Ermittlung von Zeichenketten für die anderen genannten Darstellungsarten. Dabei wird auf die vorangestellte Kennzeichnung der Darstellungsart verzichtet.

Format: **BIN$(***ausdruck*)

ausdruck - numerischer Ausdruck, Wert zwischen -32768 und 65535.

Typ: BASIC-Funktion

Funktion: Funktionswert ist eine Zeichenkette, die den ganzzahligen Teil des im Argument stehenden positiven Ausdrucks in Binärdarstellung enthält.

Hinweise: Der Funktionswert enthält keinen Hinweis auf die Darstellungsart. &B fehlt zu Beginn der Zeichenkette. Die Werte von *ausdruck* und VAL("&B"+BIN$(ausdruck)) stimmen überein.

Beispiel:
```
PRINT BIN$(253)
11111101
```

Format: **OCT$**(*ausdruck*)

ausdruck - numerischer Ausdruck, Wert zwischen -32768 und 65535.

Typ: BASIC-Funktion

Funktion: Bei positivem Parameter *ausdruck* ist der Funktionswert eine Zeichenkette, die den Wert des ganzzahligen Teils des Arguments in Oktaldarstellung enthält. Bei negativem Argument wird der Funktionswert nach Addition von 65535 zum *ausdruck* ermittelt.

Beispiel:
```
PRINT OCT$(253)
375
```

Es ist $253 = 3*8^2 + 7*8^1 + 5*8^0 =$ &O375

Format: **HEX$**(*ausdruck*)

ausdruck - numerischer Ausdruck, Wert zwischen -32768 und 65535.

Typ: BASIC-Funktion

Funktion: Der Funktionswert ist gleich dem ganzzahligen Teil eines positiven Ausdrucks, der als Argument steht, in Hexadezimaldarstellung. Bei negativem Argument wird der Funktionswert nach Addition von 65535 zum *ausdruck* ermittelt.

Beispiel:
```
PRINT HEX$(253)
FD
```

Es ist 253 = $15*16^1 + 13*16^0 = F*16^1 + D*16^0$ = &HFD

3.13. Testunterstützung

TRON Einschalten der Ablaufverfolgung
TROFF Ausschalten der Ablaufverfolgung

Nachdem Sie ein RBASIC-Programm geschrieben haben, sollten Sie überprüfen, ob es in der gewünschten Weise arbeitet. Führen Sie dazu einen Programmtest durch, in dessen Verlauf Sie möglichst alle Fehler Ihres Programms beseitigen. Testen Sie erst einzelne Programmteile, z.B. Unterprogramme, und danach das Gesamtprogramm.

Programmteile können Sie testen, indem Sie vor das Ende des betreffenden Teils eine STOP-Anweisung einfügen. Variablen können Sie im Kommandomodus Werte zuweisen. Starten Sie den mit der Zeile *startzeile* beginnenden Programmteil mit *GOTO startzeile*. Nach Unterbrechung des Programms bei Erreichen der STOP-Anweisung können Sie vom Programm berechnete Variablenwerte ausgeben und mit den erwarteten vergleichen. Ist alles in Ordnung, können Sie die zum Test eingefügte STOP-Anweisung wieder entfernen. Oft werden Sie aber Fehler beseitigen und den Test wiederholen müssen.

Einige Fehler teilt Ihnen Ihr Computer mit. Ein Fehler wird beispielsweise gemeldet, wenn der Computer eine Anweisung Ihres RBASIC-Programms nicht versteht. Das ist der Fall, wenn Sie gegen die Regeln, nach denen eine Anweisung zu bilden ist, verstoßen. Fehler dieser Art werden syntaktische Fehler genannt. Diese Fehler müssen Sie in jedem Fall korrigieren.

Auch Laufzeitfehler werden Ihnen mitgeteilt. Sie treten erst zur Laufzeit des Programms auf, wenn der Computer einzelne formal richtige Anweisungen nicht ausführen kann. Ein Laufzeitfehler kann entstehen, wenn die Werte von einzelnen Variablen, die in der Anweisung verwendet werden, nicht innerhalb zulässiger Grenzen liegen. So führt z.B. der Versuch, durch eine Variable mit Wert Null zu dividieren, zu einem Laufzeitfehler. Sie sollten in Ihrem Programm die Ursachen dieser Fehler beseitigen. Eine Reaktion auf Laufzeitfehler kann aber auch durch Ihr Programm erfolgen. Näheres erfahren Sie dazu im Zusammenhang mit der ON...ERROR...GOTO-Anweisung in Abschnitt 5.1.

Unangenehmer sind die logischen Fehler. Das Programm arbeitet ohne Beanstandungen des Computers, aber es löst seine Aufgabe nicht wie beabsichtigt. In diesem Fall haben Sie beim Entwurf oder Schreiben Ihres Programms etwas übersehen. Es könnte z.B. sein, daß Sie die begrenzte Rechengenauigkeit bei der Verwendung von Gleitpunktzahlen nicht gebührend berücksichtigt haben. Derartige Fehler müssen Sie selbst finden. Aber keine Angst, der Rechner läßt Sie mit diesem Problem nicht allein. Sie müssen zwar wie ein Detektiv alle Wege verfolgen, die zu einem Fehler geführt haben könnten, aber Sie haben im Computer einen bereitwilligen Auskunftspartner, der Ihre Fragen wahrheitsgemäß beantwortet. Mit der STOP-Anweisung können Sie das Programm an einer beliebigen Stelle unterbrechen, sich mit PRINT Werte von Variablen ausgeben lassen und Variablen neue Werte zuweisen. Eine Fortsetzung nach der die Programmunterbrechung bewirkenden STOP-Anweisung ist mit CONT möglich. Wünschen Sie eine Programmfortsetzung mit einer anderen *zeile*, müssen Sie *GOTO zeile* eingeben.
Beachten Sie in diesem Fall jedoch, daß der Zusammenhang aller GOSUB...RETURN- oder FOR...NEXT-Anweisungen erst bei Neustart des Programms hergestellt wird.

Nicht nur Variablenwerte können Sie sich anzeigen lassen. Sie können den Computer auch veranlassen, Spuren bei der Abarbeitung eines Programms zu hinterlassen. Zusätzlich zu allen übrigen Bildschirmausgaben können Sie verlangen, daß die Zeilennummern, eingeschlossen in eckige Klammern [ ], in der Reihenfolge angezeigt werden, in der sie vom Programm abgearbeitet werden. Dem Ein- bzw. Ausschalten der Ablaufverfolgung dienen die beiden folgenden Kommandos. Sie können diese Kommandos auch im Programmodus verwenden.

Format: TRON

Funktion: Die Ablaufverfolgung wird eingeschaltet. Nach Ausführung dieses Kommandos werden in eckigen Klammern zusätzlich zu allen übrigen Angaben die Nummern der Programmzeilen in der Reihenfolge Ihrer Abarbeitung angezeigt.

Format: TROFF

Funktion: Die Ablaufverfolgung wird ausgeschaltet.

Falls Sie trotz all dieser Hilfsmittel einen Fehler nicht entdecken können, lesen Sie bitte noch einmal in diesem Programmierhandbuch die Erklärungen zu den von Ihnen verwendeten Anweisungen durch.

4. Grafik

4.1. Darstellungmöglichkeiten und Grundbegriffe

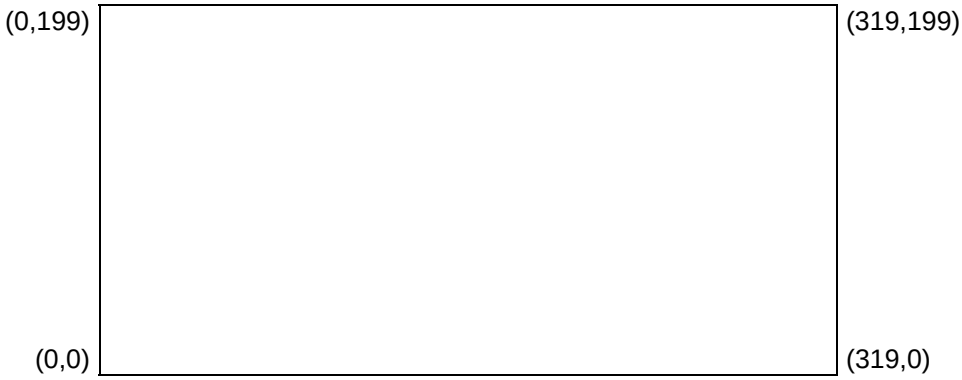
SCREEN Einstellen des Bildausgabemodus
COLOR Einstellen der Farbpalletten
CLS Löschen des Bildes im Grafikmodus

Bevor Sie die vielfältigen Möglichkeiten einer vollgrafischen Bildgestaltung (Pixelgrafik) nutzen können, sollten sie sich über die prinzipielle Wirkungsweise solcher Darstellungen im klaren sein. Die Grundlagen dazu vermittelt Ihnen dieser Abschnitt.

Zur grafischen Ausgabe von Informationen stehen Ihnen beim Computer A 5105 drei Darstellungsmöglichkeiten (Bildmodi und SCREENs) zur Verfügung, die sich von denen für die Ausgabe von Textbildern (Zahlen, Texte und quasigrafische Zeichen) generell unterscheiden. Diese drei grafischen Bildspeichervarianten (auch Grafik-SCREENs) sind durch folgende Merkmale gekennzeichnet.

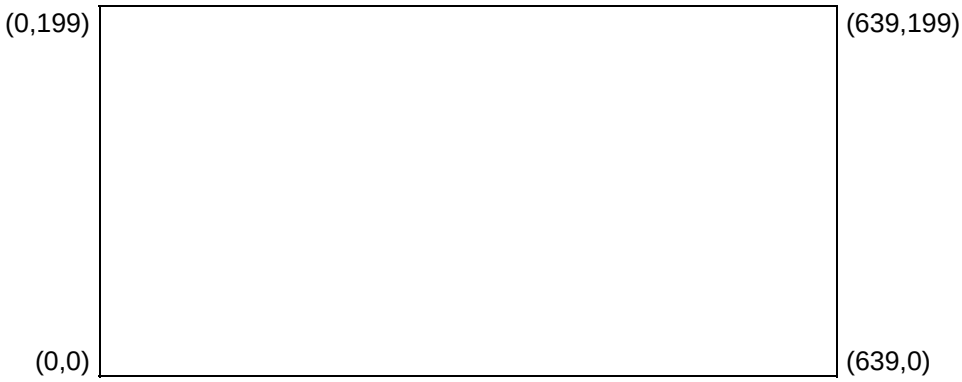
SCREEN 2 (normaler Grafikmodus)

Darstellung: 200 x 320 Bildpunkte, (jeder Punkt in 4 Farben darstellbar
(Farbauswahl über 2 Paletten mit je 4 aus 16 Farben)



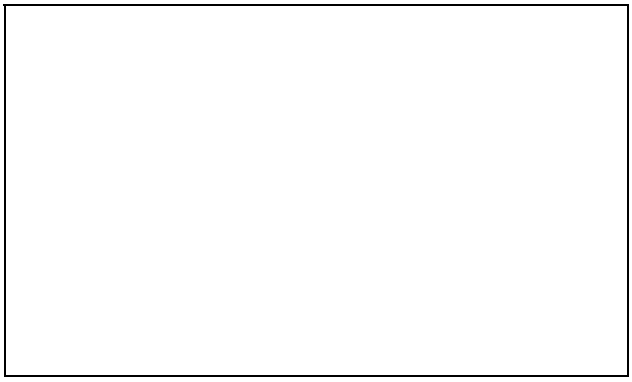
SCREEN 3 (feinauflösender Grafikmodus)

Darstellung: 200 x 640 Bildpunkte, (jeder Punkt in 4 Farben darstellbar
(Farbauswahl über 2 Paletten mit je 4 aus 16 Farben)



SCREEN 5 (Multicolormodus)

Darstellung: 200 x 320 Bildpunkte, (jeder Punkt in 16 Farben darstellbar
(Farbauswahl entsprechend der Farbtabelle für Text-SCREENs)



(0,199)	(319,199)	SCREEN 2 u. 5
	(639,199)	SCREEN 3
	(319,0)	SCREEN 2 u. 5
(0,0)	(639,0)	SCREEN 3

Hinweis: Die SCREENs 2 und 3 sind in Bildaufbau und Formgestaltung zu den Grafik-SCREENs der Programmiersprache BASI des Personalcomputers EC1834 kompatibel.

Bei der Gestaltung von Programmen (Programmabschnitten) zur grafischen Ausgabe sind folgende Schritte zu beachten.

1. Aktivierung des Grafikmodus und (in der Regel) Löschen des zugehörigen Grafik-Bildspeichers durch die Anweisungen SCREEN und CLS.

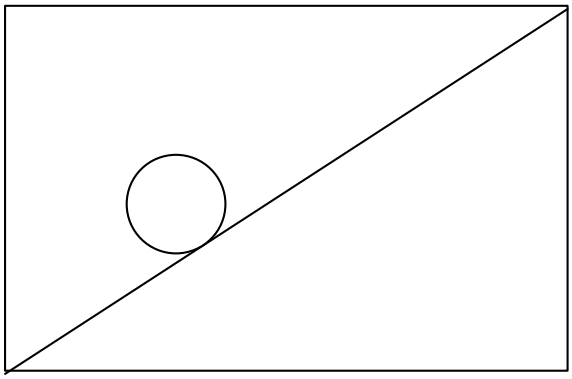
2. Anweisungen zur grafischen Bildgestaltung einschließlich spezieller Textgestaltung (PRINT#).

3. Programmpause bzw. Vermeidung der Rückkehr in den Textmodus (durch PAUSE-Anweisung, Warteschleifen u.ä.), da sonst bei Programmende automatisch in den letzten Textmodus zurückgekehrt wird.

Achtung!
Beim Erreichen des Programmendes oder bei Programmabbruch mit CTRL+STOP sowie durch Fehlermeldung schaltet der RBASIC-Interpreter in den Kommandomodus zurück und stellt den letzten Textmodus ein. Das Grafikbild ist dann nicht mehr sichtbar, im Bildspeicher aber noch vorhanden.

Beispiel:
```
10 SCREEN 2              :' Einschalten des Grafik-SCREENs
20 CLS                   :' Löschen des zugehörigen Bildspeichers
30 LINE (0,0)-(319,199)   :' Zeichnen einer Linie
40 CIRCLE (100,100),30    :' Zeichnen eines Kreises
50 PAUSE                 :' Anhalten des Programms und des Bildes
```

Nach CTRL+STOP wird das Programm abgebrochen und in den letzten Textscreen zurückgekehrt.



Wollen Sie Grafikbilder, die im Bildspeicher bereits vorhanden sind, aus dem Kommandomodus (Text-SCREEN) sichtbar machen, so können Sie das z.B. durch folgende Anweisungen erreichen.

```
SCREEN 2 : PAUSE ENTER
```

Für das Verständnis der Wirkungsweise aller Grafik-SCREENs sind folgende Begriffe wichtig:

Nullpunkt
Der Nullpunkt der Koordinatensysteme liegt standardmäßig in der linken unteren Bildecke. Im Gegensatz dazu liegt er beim Textmodus - SCREEN 0 - links oben. Der Nullpunkt kann durch die WINDOW-Anweisung (vgl. Abschnitt 4.3) verschoben werden.

Koordinatenangaben
Die Angabe der Koordinaten in Grafikanweisungen kann in der Regel

sowohl	absolut durch (X,Y)
als auch	relativ durch STEP(X,Y)

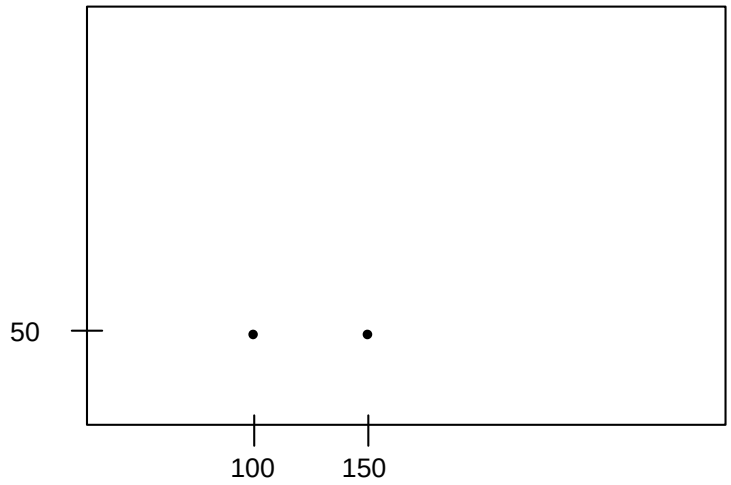
erfolgen. Die relativen Koordinatenangaben beziehen sich dann auf den "letzten angesprochenen Punkt" (siehe unten).

Koordinatensystem
Alle Koordinatenangaben beziehen sich immer auf das aktuelle Koordinatensystem. Dieses wird durch SCREEN standardmäßig eingestellt. Ein Bildpunkt entspricht dabei einer Einheit.
Durch WINDOW können die Orientierung des Koordinatensystems und die Größe einer Einheit verändert werden.

Letzter angesprochener Punkt - LP (last point)
In jedem Grafikmodus wird ein sogenannter LP (letzter angesprochener Punkt; auch: interner Grafikkursor) mitgeführt, auf den sich z.B. relative Koordinatenangaben beziehen können.
Der LP wird durch die meisten Grafikanweisungen in seiner Position verändert. Er kann oft als Bezugspunkt zur rationellen Programmierung von grafischen Darstellungen genutzt werden.

Beispiel:

```
10 SCREEN 2
20 CLS
30 PSET (100,50): ' Punkt (100,50) setzen
40 PSET STEP(50,0):' relative Koordinaten (50,0)
50 PAUSE
```



Durch die erste PSET-Anweisung wird der Punkt (100,50) im (absoluten) Koordinatensystem gezeichnet. Gleichzeitig wird der LP auf diesen Punkt gesetzt. Durch Anweisung 40 wird, ausgehend von diesem LP, im Abstand (50,0), d.h., 50 Einheiten in X-Richtung und 0 Einheiten in Y-Richtung , der nächste Punkt gezeichnet. Zusätzlich wird der LP auf diesen Punkt gesetzt.

Farbgestaltung bei grafischer Ausgabe

Die Farbgestaltung in den einzelnen Grafik-SCREENs ist unterschiedlich. Im SCREEN 5 erfolgt die Festlegung der Farben analog zu den Text-SCREENs. Es können die Farbcodierungen von 0 bis 15 gemäß der Tabelle aus Abschnitt 2.4 (bzw. Anhang E) verwendet werden. Punkte, Linien, Kreise u.a. werden dann in diesen 16 Farben dargestellt, da je Bildpunkt intern 4 Bits zur Verfügung stehen.

Die Möglichkeiten der Farbgestaltung in den Grafik-SCREENs 2 und 3 unterscheiden sich generell von denen bei der Textdarstellung. Da in diesen SCREENs je Bildpunkt jeweils nur 2 Bits zur Verfügung stehen, kann jeder Punkt unabhängig von allen anderen in 4 Farben dargestellt werden. Welche 4 der insgesamt 16 verfügbaren Farben dabei zur Auswahl stehen, wird durch die sogenannte Farbpalette festgelegt. Standardmäßig werden 2 Farbpaletten zur Verfügung gestellt.

Farbe-Nr.	Palette 0	Palette 1
0	Hintergrundfarbe (wählbar)	Hintergrundfarbe (wählbar)
1	dunkelgrün	grünblau
2	dunkelrot	dunkelpurpur
3	ocker	grau

Dabei wird Farbe Nr.0 als Hintergrundfarbe und Farbe Nr.3 als Standardzeichenfarbe angesehen. Beim Start von RBASIC werden als Hintergrund- und Randfarbe schwarz und Palette 0 eingestellt. Palette sowie Hintergrund- und Randfarbe können durch die COLOR-Anweisung verändert werden, die im Grafikmodus für die SCREENs 2 und 3 jedoch eine andere Gestalt hat als im Textmodus

Formate: **COLOR** *hintergrundfarbe*[,*palette*[,*randfarbe*]]
COLOR [*hintergrundfarbe*],[*palette*],*randfarbe*

hintergrundfarbe - Nummer der gewählten Hintergrundfarbe gemäß Farbcodierung im Textmodus (Farbnummern 0 bis 15; Standard: letzte Farbe)

palette - Nummer der gewählten Farbpalette (0 oder 1)

randfarbe - Nummer der gewählten Randfarbe (Farbnummern 0 bis 15; Standard: letzte Farbe)

Funktion: Festlegung von Farbpalette und Hintergrundfarbe für das sichtbare Grafikbild. Die Farbfestlegung wirkt sowohl sofort auf das bereits gezeichnete Bild als auch auf die nachfolgenden Grafikanweisungen.

Hinweise:

1. Die Anweisung ist in dieser Syntax nur wirksam, wenn ein Grafikmodus aktiv ist. Andernfalls werden die Parameter so wie für die COLOR-Anweisung im Textmodus interpretiert. Die Wirkung der COLOR-Anweisung ist also im Text- bzw. Grafikmodus unterschiedlich.

2. Für *palette* wird ein gerader Wert in 0 (Null), ein ungerader Wert in 1 umgewandelt.

3. Wird ein Parameter weggelassen, so wird der alte Wert beibehalten.

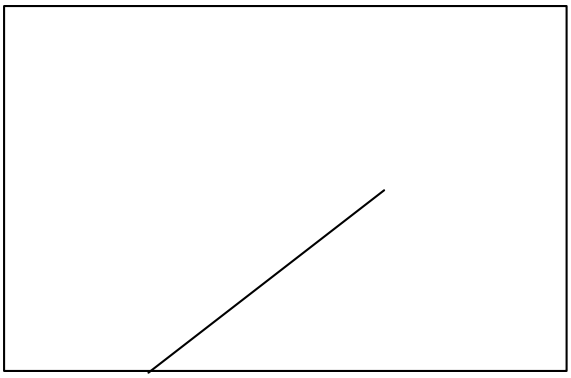
4. COLOR ohne Parameterangabe führt zu einer Fehlermeldung

5. Die COLOR-Anweisung in diesen Formaten ist nur für die SCREENs 2 und 3 gültig. Im SCREEN 5 wirkt COLOR wie in den Text-SCREENs.

Beispiele:

1. Im folgenden Programm werden im SCREEN 2 die Hintergrundfarbe 9 (blau) und die Palette 1 eingestellt. Der Rand wird weiß gefärbt. Die Linie (Zeile 20) wird in Farbe 2 (dunkelpurpur) der Palette 1 gezeichnet.

```
10 SCREEN 2:CLS:COLOR 9,1,15
20 LINE (100,0)-(200,100),2
30 PAUSE
```



2. Die folgende Ergänzung des ersten Beispiels zeigt, daß bei einem Grafikbild die Farbzuweisungen verändert werden können, ohne daß sich der Bildinhalt verändert. Bei einem Textbild ist dies nicht möglich.

```
25 CIRCLE (100,120),50
30 FOR I=1 TO 15
35     J=1 MOD 2:COLOR I,J:PAUSE 150
40 NEXT I
```

Grafik und Text

Auf die drei Grafik-SCREENs wirken alle grafischen Anweisungen, wie z.B.

PSET, PRESET, LINE, CIRCLE, PAINT, DRAW, POINT

die im Abschnitt 4 beschrieben werden. Diese Anweisungen wirken jedoch nicht auf die Text-SCREENs (SCREEN 0, 1, 8 und 9).
Gleichzeitig muß beachtet werden, daß mit den bisher genutzten Anweisungen zur Textausgabe (PRINT, PRINT USING, INPUT, LOCATE) im allgemeinen nicht auf die Grafik-SCREENs zugegriffen werden kann.

Eine Ausnahme stellen dabei folgende Anweisungen dar:

CLS - löscht jeden beliebigen (aktiven) SCREEN

WINDOW - wirkt in den Grafikmodi anders als in den Textmodi (vgl. Abschnitt 4.3)

COLOR - besitzt für die SCREENs 2 und 3 eine andere Syntax und andere Wirkung als in einem Textmodus

Die Textgestaltung bei grafischer Ausgabe erfolgt durch die Anweisungen

OPEN, PRINT#, PRINT# USING, CLOSE

Nähere Erläuterungen dazu finden Sie in Abschnitt 4.5.

4.2. Grafische Grundanweisungen

PSET Setzen eines Punktes
PRESET Löschen eines Punktes
LINE Zeichnen einer Linie oder eines Rechteckes
CIRCLE Zeichnen eines Kreises, eines Kreisbogens oder einer Ellipse

Diese vier grafischen Anweisungen bieten die grundlegenden Voraussetzungen zur effektiven Ausgabe einfacher grafischer Darstellungen auf dem Bildschirm (bzw. auf einem Plotter). Durch die sehr variabel nutzbaren Parameter kann bereits eine große Vielfalt der Darstellungen erreicht werden (z.B. ausgefüllte Rechtecke und verschiedenartige Kreissegmente).

Bitte beachten Sie:

- Alle Grafikanweisungen wirken nur, wenn ein Grafik-SCREEN aktiv ist.

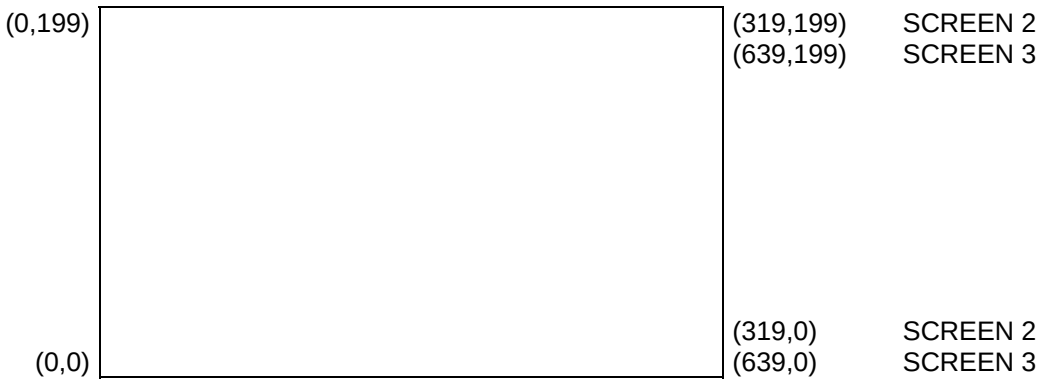
- Der Farbparameter bezieht sich in den SCREENs 2 und 3 auf die durch COLOR eingestellte Farbpalette, im SCREEN 5 auf die Farbtabelle der Text-SCREENs.

- Alle Koordinatenausgaben sind absolut und relativ (bezüglich des LP, des "letzen angesprochenen Punktes") möglich.

- Die Koordinaten beziehen sich auf das aktuelle durch SCREEN (bzw. WINDOW, vgl. Abschnitt 4.3) definierte Koordinatensystem.

- Durch WINDOW lassen sich die Koordinatenbereiche fast beliebig umdefinieren.

- Standardmäßig werden durch die Grafik-SCREENs folgende Koordinatensysteme eingestellt.



Die Koordinatenangaben in den Grafikanweisungen sollten Punkte innerhalb dieser Grenzen bezeichnen. Bei Koordinatenangaben außerhalb des eingestellten Koordinatenbereiches werden die beschriebenen Figuren (Linien, Rechtecke, Kreise usw.) nur soweit dargestellt, wie sie innerhalb dieses Bereiches liegen. Es erfolgt keinerlei Fehlermeldung.

- In Abhängigkeit vom eingesetzten Monitor bzw. Fernsehgerät kann es vorkommen, daß die Einheiten (Punkte) am Bildschirm in X- bzw. Y-Richtung auch in den SCREENs 2 und 3 unterschiedlich groß sind. Diese "Verzerrung" muß dann bei der Festlegung der Koordinaten beachtet werden.

Setzen eines Punktes

Formate: **PSET** (*X_koordinate*,*Y_koordinate*)[,*farbe*]
PSET STEP(*x_rel*,*y_rel*)[,*farbe*]

X_koordinate - absoluter Koordinatenwert für X, auf dem positioniert werden soll

Y_koordinate - absoluter Koordinatenwert für X, auf dem positioniert werden soll

x_rel - Abstand des zu setzenden Punktes vom bisherigen LP in X-Richtung

y_rel - Abstand des zu setzenden Punktes vom bisherigen LP in Y-Richtung

farbe - Nummer der Farbe, in der der Punkt gesetzt wird. Im SCREEN 2 und 3 bezieht sich *farbe* auf die aktuelle Farbpalette, im SCREEN 5 auf die Farbtabelle analog Textmodus

Funktion: Setzen eines Punktes (PSET = point set) an der angegebenen Position in der bezeichneten Farbe. Dabei können die Koordinaten absolut bzw. relativ bezüglich des LP angegeben werden.

Hinweise:

1. Die PSET-Anweisung wird vorwiegend zum Zeichnen einzelner Punkte im Grafikmodus verwendet.

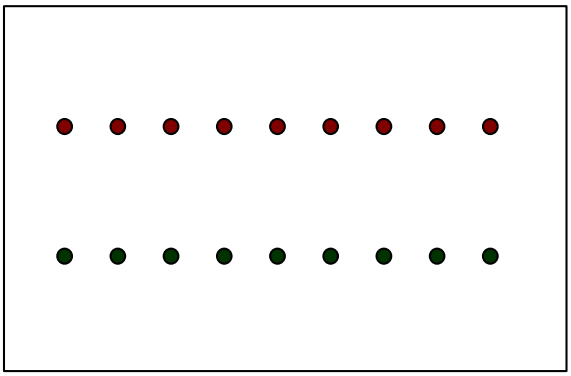
2. Wird *farbe* weggelassen, so erfolgt in den SCREENs 2 und 3 die Ausgabe in der aktuellen Farbpalette. Bei *farbe* = 0 wird der Punkt gelöscht.
   Wird *farbe* im SCREEN 5 weggelassen, so erfolgt die Darstellung in der letzten durch COLOR (gegebenenfalls im Textmodus) eingestellten Vordergrundfarbe.

3. Werden die Koordinaten relativ mit STEP(x_rel,y_rel) angegeben, bezieht sich die "Entfernung" (x_rel,y_rel) auf den LP.

4. Nach Ausführung der PSET-Anweisung wird der LP auf den angegebenen Punkt gesetzt.

Beispiel: Mit dem folgenden Programm werden Punkte durch Angabe absoluter Koordinaten in unterschiedlichen Farben gesetzt.

```
10 SCREEN 2:COLOR 0,0:CLS                    :' Grundeinstellung
20 FOR I=20 TO 300 STEP 20
30     PSET(I,59),1:PSET(I,150),2            :' Setzen der Punkte
40 NEXT I
50 PAUSE
```



FARBE 2
(dunkelrot)

FARBE 1
(dunkelgrün)

Löschen eines Punktes

Format: **PRESET** (*X*,*Y*)[,*farbe*]
PRESET STEP(*X*,*Y*)[,*farbe*]

(*X*,*Y*) - Koordinatenpaar, absolut bzw. relativ

farbe - Farbnummer der gültigen Farbpalette (SCREEN 2 und 3) bzw. der Farbtabelle (SCREEN 5)

Funktion: Löschen des Punktes mit der angegebenen Koordinate bzw. ändern seiner Farbe

Hinweise:

1. Die PRESET-Anweisung wird vorwiegend zum Löschen einzelner Punkte verwendet. Sie kann aber auch (wie PSET) zum Setzen von Punkten benutzt werden.

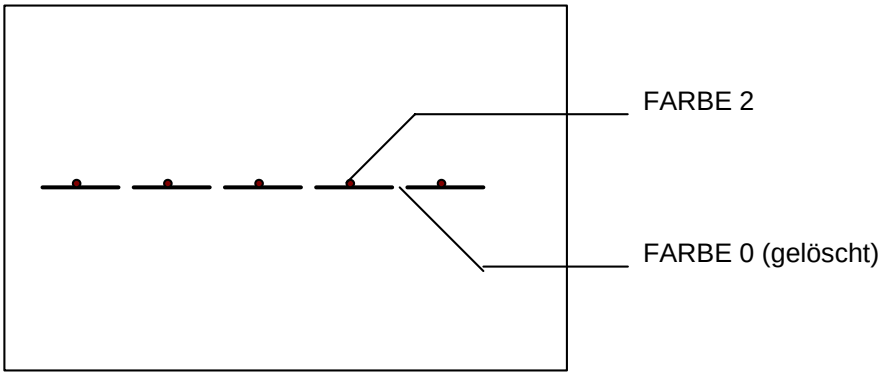
2. Wird *farbe* nicht angegeben, so wird der Punkt in der durch die COLOR-Anweisung festgelegten Hintergrundfarbe gesetzt. (Das ist praktisch der einzige Unterschied zur PSET-Anweisung.)

3. Werden die Koordinaten relativ mit STEP(X,Y) angegeben, bezieht sich die Entfernung (X,Y) auf den LP.

4. Nach Ausführung der PRESET-Anweisung wird der LP auf den angegebenen Punkt gesetzt.

Beispiel: Durch die PRESET-Anweisungen werden einzelne Punkte einer Linie gelöscht bzw. in anderer Farbe (dunkelpurpur) gesetzt.

```
10 SCREEN 2:COLOR 4,1:CLS          :' Grundeinstellung
20 LINE (10,100)-(310,100),1
30 FOR I=20 TO 310 STEP 40
40     PRESET(I,100):PRESET(I+20,100),2:' Löschen/Setzen
50 NEXT I
60 PAUSE
```



Achtung!
Die Formatangaben mit relativen Koordinaten STEP(X,Y) werden in den folgenden Beschreibungen nicht mehr aufgeführt. Trotzdem können Sie STEP(X,Y) in allen Anwendungen stets statt (X,Y) benutzen.

Zeichnen von Linien und Rechtecken

Format 1: **LINE** [(*X1*,*Y1*)] - (*X2*,*Y2*) [,*farbe*]

Format 2: **LINE** [(*X1*,*Y1*)] - (*X2*,*Y2*),[*farbe*],*B*
LINE [(*X1*,*Y1*)] - (*X2*,*Y2*),[*farbe*],*BF*

(*X1*,*Y1*) - Anfangspunkt der Linie bzw. Eckpunkt des Rechteckes

(*X2*,*Y2*) - Endpunkt der Linie bzw. gegenüberliegender Eckpunkt des Rechteckes

farbe - Farbnummer der gültigen Farbpalette (SCREEN 2 und 3) bzw. der Farbtabelle (SCREEN 5)

B - bewirkt Zeichnen eines Rechteckes (box)

BF - bewirkt Zeichnen und Ausfüllen eines Rechteckes (box fill)

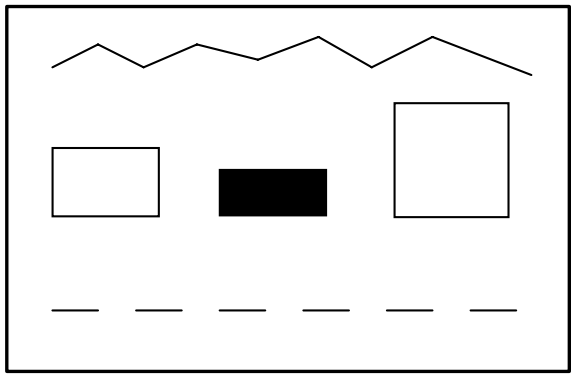
Funktion: Zeichnen einer Linie oder eines Rechteckes im Grafikmodus. Wahlweise kann das Rechteck auch ausgefüllt werden.

Hinweise:

1. Im Format 1 wird eine Linie zwischen den angegebenen Punkten in der angegebenen Farbe gezeichnet.
2. Im Format 2 wird ein Rechteck mit den Eckpunkten (X1,Y1), (X2,Y1), (X2,Y2) und (X1,Y2) gezeichnet.
   Bei Angabe von *B* (box) wird nur das Rechteck gezeichnet. Wird *BF* (box fill) angegeben, so wird das Rechteck mit *farbe* ausgefüllt.
3. Wird *farbe* weggelassen, so wird in den SCREENs 2 und 3 in Farbe 3 der eingestellten Farbpalette und im SCREEN 5 in der letzten durch COLOR eingestellten Vordergrundfarbe gezeichnet.
4. Wird der erste Koordinatenpunkt (X1,Y1) weggelassen, wird dafür der letzte gezeichnete Punkt (LP) verwendet.
5. Werden relative Koordinaten (STEP) verwendet, so beziehen sie sich in beiden Fällen auf den LP als Ursprungsort.
6. Nach Ausführung der LINE-Anweisung wird der LP auf (X2,Y2) gesetzt.
7. Bei Überschreitung des darstellbaren Bereiches wird nur bis zum Bildrand gezeichnet. Eine Fehlermeldung erfolgt nicht.

Beispiel:

```
10 SCREEN 2:COLOR 11,0:CLS
20 LINE (0,0)-(319,199),2,B       :' Rechteck (Rahmen)
25 PSET (20,170),0              :' Punkt
30 FOR I=40 TO 300 STEP 20
40     LINE -(I,170+10*RND(1))    :' Linien ab LP
50 NEXT I
60 LINE (20,80)-(80,120),,B       :' Rechteck
70 LINE (120,80)-(180,110),2,BF   :' Rechteck, ausgefüllt
80 LINE (220,80)-(280,140),2,B    :' Rechteck
90 FOR I=20 TO 280 STEP 40
100    LINE (I,20)-STEP(20,0),1   :' Linie mit Länge 20
110 NEXT I
120 PAUSE
```



Zeichnen von Kreisen und Ellipsen

Format 1: **CIRCLE** (*X*,*Y*),*radius*[,*farbe*]

Format 2: **CIRCLE** (*X*,*Y*),*radius*,[*farbe*],[*anfwinkel*],*endwinkel*

Format 3: **CIRCLE** (*X*,*Y*),*radius*,[*farbe*],[*anfwinkel*],[*endwinkel*],*ellipse*

(*X*,*Y*) - Mittelpunkt des Kreises bzw. der Ellipse (absolut oder relativ angegeben)

radius - Radius des Kreises bzw. der großen Halbachse der Ellipse

farbe - Farbnummer der aktuellen Farbpalette (SCREEN 2 und 3) bzw. der Farbtabelle (SCREEN 5)

anfwinkel - Anfangswinkel für Kreis- bzw. Ellipsenbogen; Wert zwischen -2π und 2π (Standard: 0)

endwinkel - Endwinkel für Kreis- bzw. Ellipsenbogen; Wert zwischen -2π und 2π (Standard: 2π)

ellipse - Ellipsenfaktor; gibt das Verhältnis zwischen Y- und X-Achse der Ellipse an. Wert größer als Null.

Funktion: Zeichnen eines Kreises (Format 1), eines Kreisbogens (Format 2) bzw. einer Ellipse (Format 3). Der Mittelpunkt wird jeweils durch (*X*,*Y*) und der Radius durch *radius* festgelegt.

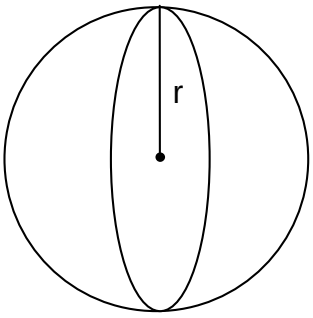
Hinweise:

1. Der Kreis (Kreisbogen, Ellipse) wird mit der durch *farbe* festgelegten Farbe gezeichnet. Wird *farbe* nicht angegeben, so wird Farbe 3 der aktuellen Farbpalette (SCREENs 2 und 3) bzw. die letzte durch COLOR eingestellte Vordergrundfarbe (SCREEN 5) verwendet.

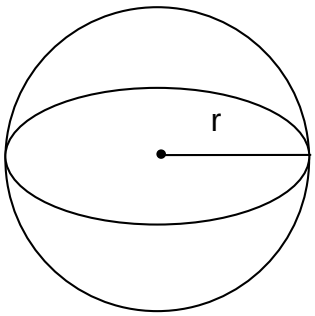
2. Die Werte für *anfwinkel* und *endwinkel* beschreiben das zu zeichnende Kreissegment, werden im Bogenmaß angegeben und müssen zwischen -2π und 2π liegen. Werden diese Werte nicht angegeben, so wird standardmäßig 0 bzw. 2π angenommen.

3. Werden die Winkel negativ angegeben, so wird der Kreisbogen zwischen den Beträgen dieser Werte gezeichnet. Zusätzlich werden die entsprechenden Radien (Sektorgrenzen) gezeichnet.

4. Als Ellipsenparameter *ellipse* wird ein positiver Wert angegeben. Er gibt das Verhältnis von Y- zu X-Radius an. Die Ellipse wird immer entsprechend der Abbildung innerhalb des Kreises mit dem angegebenen Radius gezeichnet.



ellipse > 1



ellipse < 1

ellipse > 1: der angegebene Radius ist der Y-Radius der Ellipse (X-Radius = *radius*/*ellipse*)

ellipse < 1: der angegebene Radius ist der X-Radius der Ellipse (Y-Radius = *radius***ellipse*)

Gegebenenfalls kann durch *ellipse* auch eine Bildschirmverzerrung eines Kreises ausgeglichen werden.

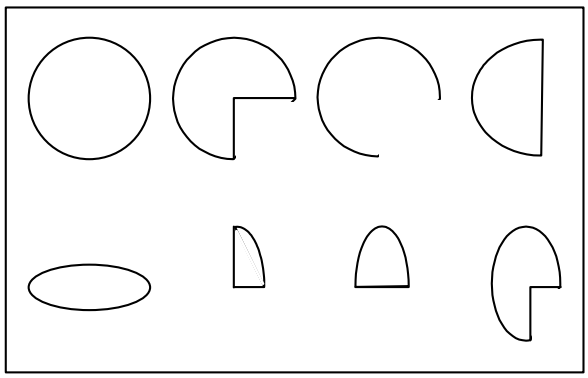
5. Der LP wird nach Zeichnen des Kreises (der Ellipse) auf den Mittelpunkt positioniert.

6. Die Parameter *farbe*, *anfwinkel*, *endwinkel*, *ellipse* können einzeln oder von rechts weggelassen werden.

Beispiel:

```
5  PI=3.1415926:WI=1E-14
10 SCREEN 5:COLOR 5,0:CLS
20 CIRCLE (50,150), 30
30 CIRCLE (130,150),30,12,-WI,-1.5*PI :' Kreisbogen mit Grenzen
40 CIRCLE (210,150),30,,0,1.5*PI      :' Kreisbogen ohne Grenzen
50 CIRCLE STEP(80,0),30,1,-PI/2,1.5*PI:' Kreisbogen
```

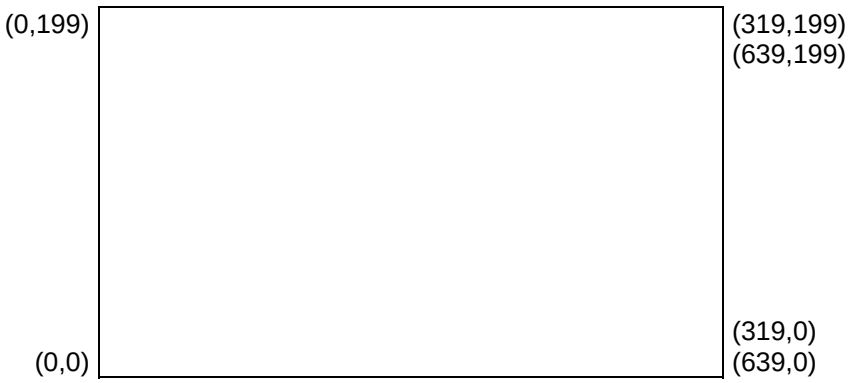
```
60 CIRCLE (50,50),30,2,,,.5              :' Ellipse
70 FOR I=1 TO 3                          :' Ellipsenfolge
80     X=50+80*I:FI=I+12
90     CIRCLE (X,50),30,FI,-WI,-I*PI/2,2
100 NEXT I
110 PAUSE
```



4.3. Koordinatensysteme

WINDOW Festlegung eines Koordinatensystems in einem Grafikmodus

Die bisher beschriebenen grafischen Ausgaben bezogen sich alle auf das Koordinatensystem mit den Eckpunkten (0,0), (0,199) usw., das durch die Anzahl der im jeweiligen SCREEN darstellbaren Bildpunkte festgelegt ist.



Diese (sogenannten Geräte-)Koordinatensysteme haben jedoch mehrere Nachteile, wie Sie sicher bereits festgestellt haben:

- Die gewünschten Darstellungen müssen stets in diese Koordinatensysteme umgerechnet werden, auch wenn das teilweise sehr umständlich und unzweckmäßig ist (z.B. bei der Darstellung einfacher Funktionen).

- In Abhängigkeit vom eingesetzten Monitor ist die Verzerrung zwischen X- und Y-Achse unterschiedlich. Die Anweisungen

  ```
  LINE (50,50)-(100,100),,B
  CIRCLE (150,150),30
  ```

  ergeben deshalb nur bei Nutzung eines Standardmonitors und nur in den SCREENs 2 und 5 ein Quadrat bzw. einen Kreis. Um das bei beliebigen Ausgabegeräten (z.B. auch Drucker und Plotter) bzw. im SCREEN 3 zu erreichen, müssen in die entsprechenden Koordinatenangaben Korrekturfaktoren eingearbeitet werden.

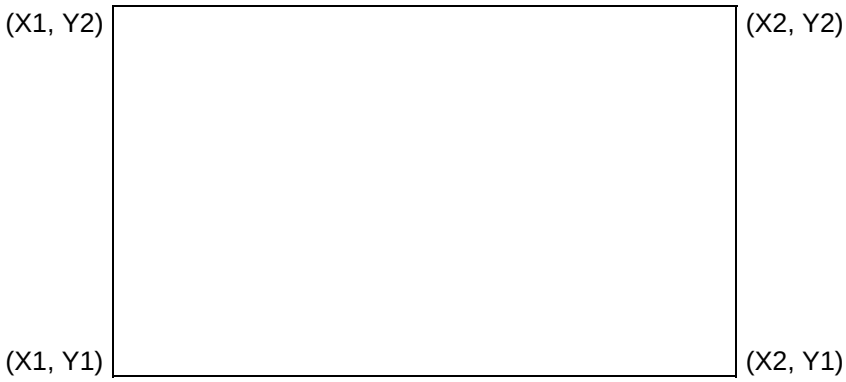
Um die Programmierung von Darstellungen zu vereinfachen, kann man deshalb ein beliebiges (Nutzer-)Koordinatensystem definieren. Dies erfolgt durch die Anweisung WINDOW, die im Grafikmodus anders als im Textmodus wirkt.

Format: **WINDOW** [(*X1*,*Y1*)-(*X2*,*Y2*)]

(*X1*,*Y1*) - Koordinaten des linken unteren Eckpunktes des Ausgabebereiches am Bildschirm

(*X2*,*Y2*) - Koordinaten des rechten oberen Eckpunktes des Ausgabebereiches am Bildschirm

Funktion: Festlegung des Nutzerkoordinatensystems (bzw. Weltkoordinatensystems) für die grafische Ausgabe auf SCREEN 2, 3 und 5.



Hinweise:

1. Die Anweisung bezieht sich stets auf den "aktiven" Grafik-SCREEN (vgl. Abschnitt 5.3.
2. WINDOW ohne Parameter stellt das Grundkoordinatensystem des aktiven SCREENs ein.
3. Die Werte für X1,...,X2 können im gültigen Zahlenbereich des RBASIC-Interpreters liegen.
4. Wird X2<X1 bzw. Y2<Y1 gewählt, so erfolgt eine Umkehrung der üblichen Axenorientierung (Spiegelung).
5. Der Nullpunkt des Koordinatensystems kann an eine beliebige Stelle (innerhalb oder außerhalb des sichtbaren Bereiches) gelegt werden.
6. Für X1=X2 bzw. Y1=Y2 erfolgt eine Fehlermeldung.
7. Die WINDOW-Anweisung ist auch dann nützlich, wenn Sie Grafikbilder außer auf den Bildschirm auch auf einen Plotter (oder Drucker) ausgeben wollen. Die WINDOW-Anweisung hilft Ihnen dann, den geeigneten Maßstab festzulegen.
8. Mit WINDOW können Sie bei der Darstellung gewöhnlicher Funktionen sehr leicht Zoom-Effekte programmieren.

Beispiele:

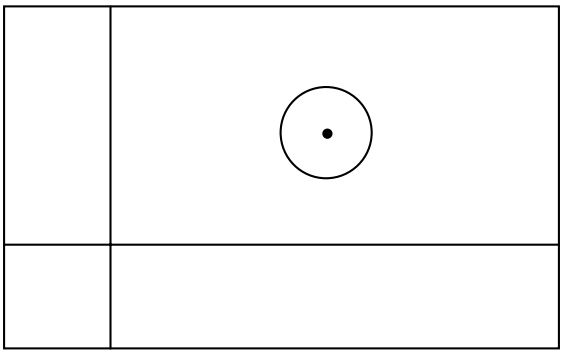
1. Wollen Sie erreichen, daß Kreise wirklich "Kreise" sind, so muß das Verhältnis von X- und Y-Achse entsprechend dem eingesetzten Monitor und dem verwendeten SCREEN abgestimmt werden. Beim Standardmonitor ist das der Fall, wenn

$$\text{etwa} \quad \frac{X2-X1}{Y2-Y1} = \frac{320}{200} = 1{,}6 \text{ gilt.}$$

Durch das Programm

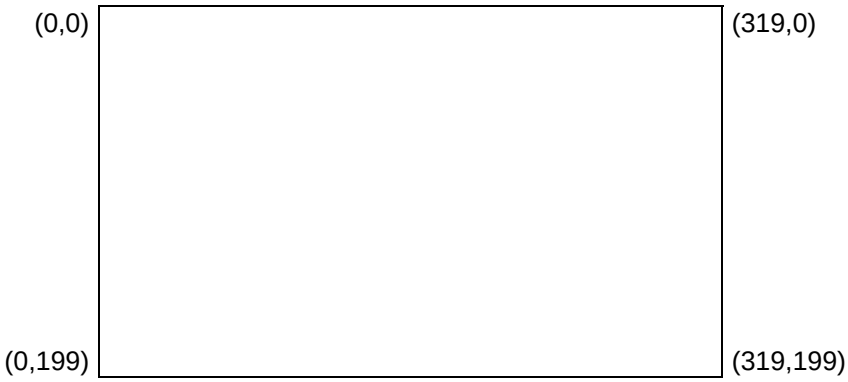
```
10 SCREEN 3:CLS
20 WINDOW (-10,-10)-(40,22)
30 CIRCLE (10,10),5
40 PAUSE
```

wird somit tatsächlich auch im SCREEN 3 ein Kreis gezeichnet. Der Ursprung des Koordinatensystems liegt im Beispiel innerhalb der Zeichenfläche



2. Eine einfache Spiegelung der Y-Achse erreichen Sie durch

```
20 WINDOW (0,199)-(319,0)
```

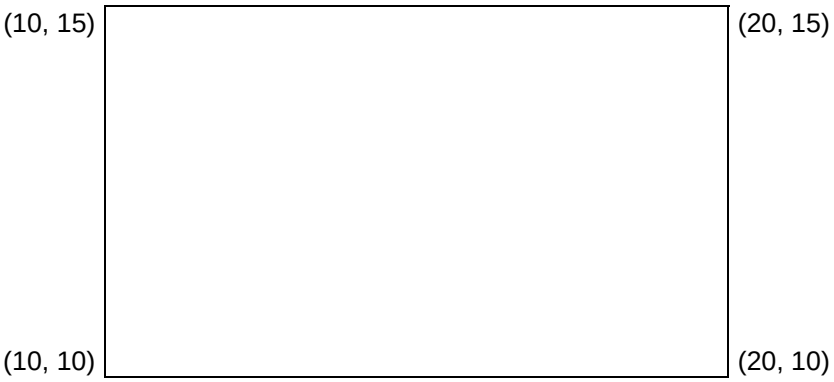


Koordinatensysteme mit dem Ursprung links oben werden bei manchen Computern standardmäßig verwendet. Durch Nutzung dieser WINDOW-Anweisung können Sie Programme leichter übertragen.

3. Durch die Anweisung

```
20 WINDOW (10,10)-(20,15)
```

wird eine Ausschnittsvergrößerung erreicht und gleichzeitig der Nullpunkt außerhalb der Darstellungsfläche gelegt.



4.4. Weitere Zeichenanweisungen

PAINT Ausfüllen von Flächen
DRAW Zeichnen von Figuren (Polygonzügen)
POINT Abfragen von Koordinaten

Diese grafischen Anweisungen sind für die Gestaltung spezieller Grafikbilder zweckmäßig. Durch einfache Mittel kann dadurch eine sehr schnelle Erzeugung auch komplizierterer Bilder erreicht werden.

Beachten Sie bitte, daß auch für die Anweisungen PAINT und POINT die Hinweise bezüglich Koordinatensystem und absoluter bzw. relativer Koordinatenangabe aus den Abschnitten 4.1 bis 4.3 gültig sind. Dagegen beziehen sich alle Angaben bei DRAW nur auf das durch SCREEN festgelegte Gerätekoordinatensystem.

Ausfüllen von Flächen

Formate: **PAINT** (*X*,*Y*)[,*farbe*[,*randfarbe*]]
PAINT (*X*,*Y*),[*farbe*],*randfarbe*

(*X*,*Y*) - Koordinatenpaar, absolut oder relativ

farbe - Farbnummer der gültigen Farbpalette (SCREEN 2 und 3) bzw. der Farbtabelle (SCREEN 5)

randfarbe - Randfarbe der auszufüllenden Fläche: Werte wie bei *farbe*

Funktion: Ausfüllen einer begrenzten Fläche, die durch den Punkt (x,y) in ihrem Inneren und gegebenenfalls durch die angegebene Randfarbe bezeichnet wird. Das Ausfüllen der Fläche erfolgt in der gewählten Farbe.

Hinweise:

1. Die jeweiligen Rand- und Zeichenfarben werden durch den Farbcode der gültigen Farbpalette (SCREEN 5) festgelegt. Wird die Randfarbe nicht angegeben, so wird *randfarbe* = *farbe* angenommen.

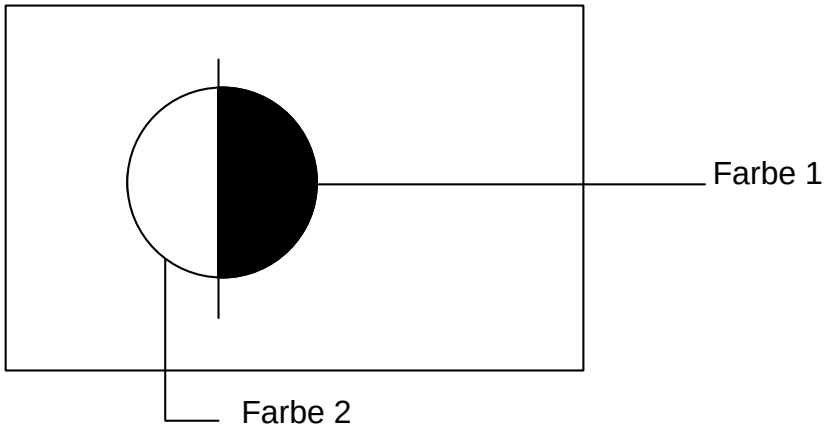
2. Wird die Zeichenfarbe nicht angegeben, so wird für Farbe 3 der gültigen Farbpalette (SCREENs 2 und 3) bzw. die zuletzt durch COLOR eingestellte Vordergrundfarbe (SCREEN 5) verwendet.

3. Falls der durch (x,y) angegebene Punkt die gleiche Farbe wie *randfarbe* besitzt (oder außerhalb des darstellbaren Bereiches liegt), so wird die Anweisung nicht ausgeführt.

4. Wenn die durch die Randfarbe gekennzeichnete Grenzlinie nicht geschlossen ist, wird der gesamte Bildschirm ausgefüllt.

Beispiel: Eine durch einen Kreis und eine Gerade begrenzte Fläche (Halbkreis) wird mit Farbe 1 (dunkelgrün) der Palette 0 ausgefüllt.

```
10 SCREEN 2:COLOR 8,0:CLS
20 CIRCLE (100,100),50,2
30 LINE (100,40)-(100,160),2
40 PAINT (110,100),1,2
50 PAUSE
```



Zeichnen spezieller Figuren

Format: **DRAW** *zeichenkette*

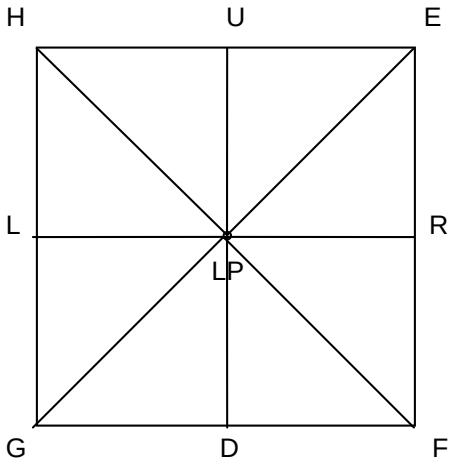
zeichenkette - Zeichenkette (Konstante oder Variable), die grafische Unteranweisungen zum Zeichnen von Polygonzügen enthält

Funktion: Zeichnen von Figuren (Polygonzügen) im Grafikmodus. Durch *zeichenkette* wird eine Figur (Polygonzug) definiert, die gezeichnet werden soll. Zur Definition der Figur stehen verschiedene Grafik-Macro-Kommandos zur Verfügung, die innerhalb der Zeichenkette kombiniert werden können. Diese Kommandos beziehen sich stets auf den LP, der jedoch durch diese Macro-Kommandos verändert werden kann.

Grafik-Macro-Kommandos

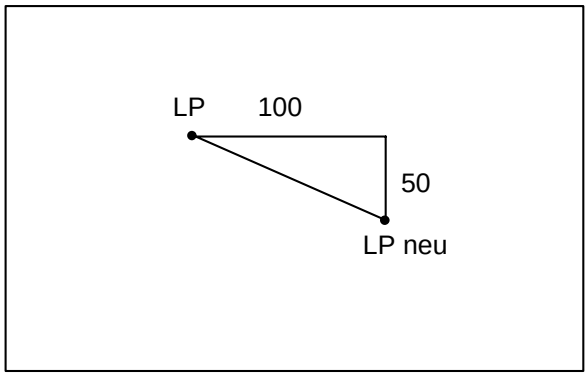
Eine Verschiebung des Bezugspunktes (LP) um *anzahl* Punkte und Zeichen der Verbindungslinie vom alten LP kann durch folgende Macro-Kommandos erfolgen:

U[*anzahl*]	- nach oben (up)
D[*anzahl*]	- nach unten (down)
L[*anzahl*]	- nach links
R[*anzahl*]	- nach rechts
E[*anzahl*]	- nach rechts oben
F[*anzahl*]	- nach rechts unten
G[*anzahl*]	- nach links unten
H[*anzahl*]	- nach links oben
(*anzahl*	- Verschiebung um *anzahl* Bildpunkte; Standard: 1)



M X-Koordinate,
Y-Koordinate
- Zeichnet eine Linie von LP zu den angegebenen Koordinaten.
- Ist die erste Koordinate mit einem Vorzeichen versehen (+ oder -), so bezeichnen sie relative Koordinaten.
- Liegen die Koordinaten außerhalb des darstellbaren Bereiches, so wird der Punkt auf den Bildrand projiziert.

Beispiel:
```
DRAW"M+100,-50"
```
(entspricht LINE-STEP(100,-50))



S
- Beim nachfolgenden Kommando (U,D,...,H,M) wird nur der LP bewegt und nicht gezeichnet.

N
- Nachfolgendes Kommando zeichnet Linien, ohne den LP zu ändern

A *zahl*
- Dreht die durch U, D, L, R, E, F, G, H definierten Figuren in 90°-Schritten.
- Der Winkel wird durch eine Zahl zwischen 0 und 3 beschrieben. (Andere Werte führen zu einer Fehlermeldung).

  0 = 0° 1 = 90° 2 = 180° 3 = 270°
- Die Drehung erfolgt für alle folgenden Figuren. Bei Programmende sollte wieder "A0" eingestellt werden.

C *farbe*
- Stellt die Farbe für die nachfolgenden DRAW-Unteranweisungen entsprechend der gültigen Farbpalette (SCREENs 2 und 3) bzw. der Farbtabelle (SCREEN 5) ein. Als Standards werden die gleichen Werte wie bei PSET, LINE usw. verwendet.

S *sfaktor*
- Die in U, D, L, R, E, F, G, H angegebene *anzahl* Punkte wird mit *sfaktor* multipliziert.
- *sfaktor* kann nur im Bereich 1...255 liegen. Der tatsächliche Skalierungsfaktor ist *sfaktor*/4.
- Die Skalierung gilt für alle folgenden Figuren. Bei Programmende sollte wieder "S4" gesetzt werden.

X *zeichenkette*
- Führt die in *zeichenkette* enthaltenen Macro-Befehle aus.

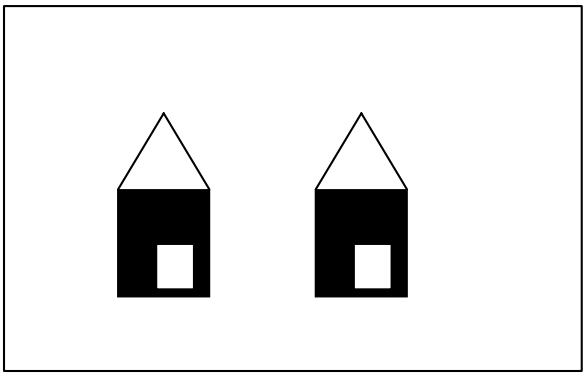
= *num_variable*
- Erlaubt die Verwendung einer numerischen Variablen zur Angabe der Werte für *anzahl*, *zahl*, *farbe* usw. (numerische Ausdrücke sind nicht erlaubt).

Hinweis:
1. Alle Koordinaten- und Abstandsangaben innerhalb der DRAW-Anweisung beziehen sich grundsätzlich auf das durch SCREEN festgelegte Gerätekoordinatensystem. Veränderungen durch WINDOW werden nicht berücksichtigt.

2. Die Macro-Kommandos können einzeln oder in beliebiger Kombination verwendet werden.

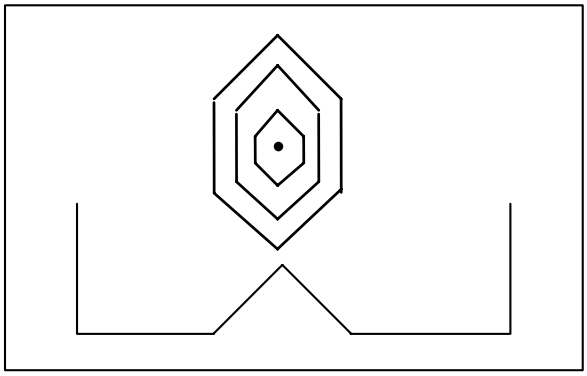
Beispiele: 1. Die durch die Zeichenkette Z$ definierte Figur wird in den Zeilen 50 und 60 an zwei verschiedenen Positionen gezeichnet und anschließend jeweils teilweise ausgefüllt.

```
10 SCREEN 2:CLS
40 Z$ = "U50E25F25D50L50R20U20R20D20"    :' Haus
50 PSET (50,50):DRAW Z$                  :' zeichnen
60 PSET (150,50):DRAW Z$                 :' zeichnen
70 PAINT (60,80):PAINT (160,80)          :' ausfüllen
80 PAUSE
```



2. Die durch A$ definierte Figur wird mit unterschiedlichen Größen mehrfach gezeichnet. Die Figur in Zeile 70 beginnt in Punkt (50,100).

```
10 SCREEN 2:CLS
20 A$="BD3H2U2E2F2D2G2BU3"          :' Grundfigur
30 DRAW "BM160,120"
40 FOR I=10 TO 80 STEP 20
50     DRAW "S=I;XA$;"
70 DRAW "S4C2BM50,100D80R80E30F30R80U80"
80 PAUSE
```



Abfragen einzelner Punkte

Format: **var=POINT(X,Y)**
var=POINT STEP(X,Y)

Typ: BASIC-Funktion

Funktion: Liefert den aktuellen Farbcode des bezeichneten Koordinatenpunktes im Grafikmodus.

Hinweise: 1. Der gelieferte Farbcode bezieht sich auf die aktuelle Farbpalette (SCREENs 2 und 3, ganzzahliger Wert 0,...,3) bzw. auf die Farbtabelle (SCREEN 5, Werte 0,...,15).

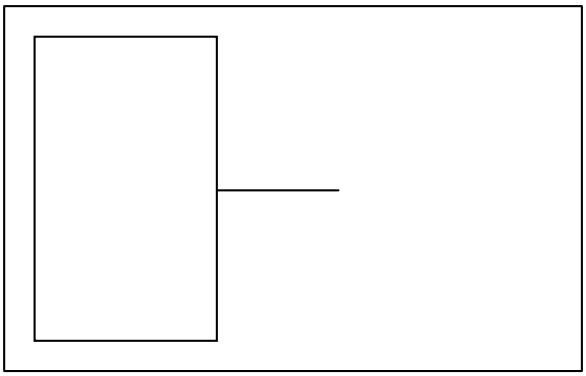
2. Der LP wird durch die POINT-Funktion nicht verändert.

3. Liegt der angegebene Punkt außerhalb der darstellbaren Fläche, wird der Wert -1 geliefert.

Beispiele:

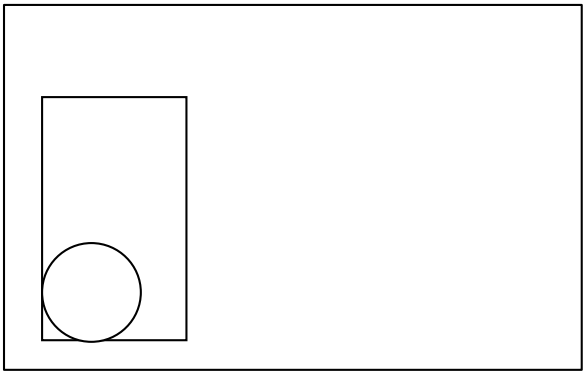
1. Mit dem folgenden Programm wird ein Rechteck mit einer zufälligen Begrenzung gezeichnet. Anschließend wird eine Punktfolge bis zum rechten Rand des Rechtecks geführt.

```
10 PRINT "TASTE DRÜCKEN"
20 R=RND(1):IF INKEY$="" THEN 20
30 SCREEN 2:COLOR 15,1:CLS
40 R1=100+INT(30*R)
50 LINE(20,20)-(R1,180),2,B
60 X=200
70 PSET(X,100),3:X=X-1
80 A=POINT(X,100)
90 IF A<>2 THEN 70
100 PAUSE
```



2. Im folgenden Programm kann mittels der Kursortasten ein Punkt innerhalb eines vorgegebenen Rechteckes bewegt werden. Die Überschreitung des Randes wird mit Hilfe der POINT-Funktion verhindert. (Sie können selbst die Rechteckfläche durch eine beliebige Umrandung ersetzen.)

```
10 SCREEN 2:CLS
20 LINE(20,20)-(300,180),3,B
30 X=160:Y=100:PSET(X,Y)2
40 ' ---
50 A$=INKEY$: IF A$="" THEN 50 ELSE A=ASC(A$)
60 IF A=13 THEN 600                          :' Test ENDE
70     IF A<28 OR A>31 THEN 50               :' Test Kursor
80     ON (A-27) GOTO 100,200,300,400
100       X1=X+1:IF POINT(X1,Y)<>3 THEN X=X1:GOTO 500
200       X1=X-1:IF POINT(X1,Y)<>3 THEN X=X1:GOTO 500
300       Y1=Y+1:IF POINT(X,Y1)<>3 THEN Y=Y1:GOTO 500
400       Y1=Y+1:IF POINT(X,Y1)<>3 THEN Y=Y1:GOTO 500
500    ' ---
510    PSET(X,Y),2                           :' neuer Punkt
520    GOTO 50                               :' Sprung Abfrage
530    ' ---
600 BEEP: PAUSE
```



4.5. Textausgabe im Grafikmodus

OPEN	Eröffnen einer Datei für Textausgabe
PRINT#	Textausgabe im Grafikmodus
PRINT# USING	Formatierte Textausgabe im Grafikmodus
CLOSE	Schließen der Datei für Textausgabe

In den Abschnitten 4.1 bis 4.4 haben Sie viele Möglichkeiten der Gestaltung von grafischen Darstellungen in den Grafik-SCREENs kennengelernt. Zur Beschriftung dieser Grafiken können Sie die üblichen Anweisungen zur Textanzeige (PRINT, PRINT USING, INPUT, ...) jedoch nicht benutzen. Die Textausgabe auf einen Grafikbildschirm erfolgt über eine Ausgabedatei. Die hierzu erforderlichen Anweisungen werden im folgenden erläutert. Auf die Arbeit mit beliebigen Ein-/bzw. Ausgabedateien wird in Abschnitt 6 ausführlich eingegangen.

Prinzipiell werden bei der Ausgabe von Text auf einen Grafikbildschirm drei Schritte durchlaufen:

Schritt 1: Eröffnen der Ausgabedatei durch eine OPEN-Anweisung

Schritt 2: Beschreiben der Datei und damit Ausgabe an den Bildschirm durch eine (oder mehrere) PRINT#-Anweisung(en).
Gegebenenfalls kann durch PSET oder PRESET die Ausgabeposition des Textes (der Zahl) festgelegt werden.

Schritt 3: Schließen der Ausgabedatei durch eine CLOSE-Anweisung.

Eröffnen einer Textdatei für Ausgabe im Grafikmodus

Format: **OPEN "GRP:" AS $***dateinummer*

dateinummer - ganze Zahl (Werte: 1 bis 10 bzw. MAXFILES)

Funktion: Eröffnen einer Datei für die Ausgabe im aktuellen Grafikmodus. Anschließend können mit PRINT# Texte bzw. Zahlen im Grafikmodus angezeigt werden.

Hinweise:
1. *dateinummer* darf die durch MAXFILES festgelegte maximale Dateianzahl nicht überschreiten. Zu Beginn wird MAXFILES = 1 eingestellt (ausführliche Erläuterung im Abschnitt 6).
2. *dateinummer* darf nicht mit der einer bereits eröffneten Datei übereinstimmen.
3. Wenn die eröffnete Datei nicht mehr benötigt wird, sollte sie durch CLOSE geschlossen werden.

Ausgabe von Texten bzw. Zahlen

Format 1: **PRINT #***dateinummer* [,*ausgabeliste*]

Format 2: **PRINT #***dateinummer*,**USING** *format*;*ausgabeliste*

dateinummer - ganze Zahl, die der Dateinummer der zugehörigen OPEN-Anweisung entspricht (Werte: 1 bis 10)

ausgabeliste - enthält die auszugebenden Konstanten und Variablen

format - beschreibt das Format, in dem ausgegeben wird

Funktion: Ausgabe (Anzeige) der in *ausgabeliste* spezifizierten numerischen oder Zeichenketten-variablen bzw. Konstanten auf den aktuellen Grafikbildschirm.

Hinweise: 1. *dateinummer* muß der in der OPEN-Anweisung verwendeten Nummer entsprechen.

2 Für *ausgabeliste* und *format* gelten die gleichen Regeln wie in der PRINT bzw. PRINT USING-Anweisung. Zahlen und Zeichenketten werden in der gleichen Form im Grafikbild angezeigt.

3. Die auszugebenden Zeichen werden wie im Textmodus in einem 8x8-Raster angezeigt. In Abhängigkeit vom eingestellten Grafikmodus können 40 (SCREEN 2 und 5) bzw. 80 Zeichen (SCREEN 3) je Zeile angezeigt werden. Die Zeichengröße und Schriftrichtung wird durch WINDOW nicht beeinflußt.

4. Die Ausgabe des ersten Zeichens *ausgabeliste* erfolgt an der Position des LP (linke untere Ecke des Zeichens). Diese Position kann z.B. durch eine PSET oder PRESET-Anweisung festgelegt werden. Die Position des LP wird durch die Textausgabe nicht verändert.

5. Die Darstellung des Textes erfolgt in der zuletzt in einer Grafikanweisung (PSET, LINE, CIRCLE,...) verwendeten Farbe.
   Durch PRESET kann der LP positioniert werden, ohne daß ein Punkt gezeichnet wird und ohne Änderung der eingestellten Farbe.
   Zum Einstellen der Schriftfarbe für einen Text ab dem LP=(X,Y) kann deshalb in jedem Fall die Anweisungsfolge

   PSET(X,Y),schriftfarbe:PRESET(X,Y),hintergrundfarbe

   verwendet werden.

6. Wird durch die anzuzeigenden Zeichen der rechte Bildschirmrand erreicht, so erfolgt die Fortsetzung der Ausgabe automatisch am Beginn der nächsten "Textzeile", d.h. 8 Punktzeilen tiefer in Spalte 0 (bezogen auf das Gerätekoordinatensystem).

7. Durch das Steuerzeichen CHR$(13) in *ausgabeliste* wird ebenfalls ein Zeilenvorschub zur nächsten "Textzeile" bewirkt.

8. Durch PRINT# bzw. PRINT#..USING können alle darstellbaren Zeichen des Zeichensatzes (Codierung 32 bis 255) im Grafikmodus angezeigt werden. 2-Byte-Zeichen werden wie üblich durch Voranstellen von CHR$(1) angesprochen.

Schließen der Datei zur Textausgabe im Grafikmodus

Format: **CLOSE** #*dateinummer*

Funktion: Schließen (Abmelden) der durch OPEN eröffneten Datei mit der entsprechenden Nummer.

Hinweise: 1. Nach Abarbeitung der CLOSE-Anweisung kann *dateinummer* zur Eröffnung anderer Dateien wieder verwendet werden.

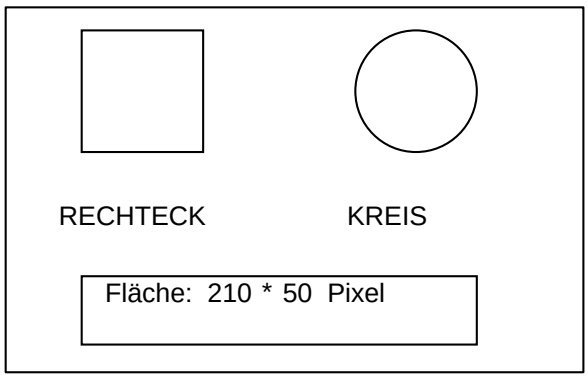
2. Durch CLOSE wird der am Bildschirm sichtbare Text nicht gelöscht!

Das Anzeigen von Text in einem Grafikmodus erfordert also die Nutzung der Anweisungen OPEN, PRINT# und CLOSE. Das Zusammenwirken dieser Anweisungen können sie dem folgenden Beispiel entnehmen.

Beispiel: Im Programm wird eine Ausgabedatei mit der Nummer 1 definiert. Die Beschriftung des Bildes erfolgt in den Zeilen 70...100.

```
10 SCREEN 2:COLOR 0,0:CLS
20 LINE (50,120)-(120,190),3,B
30 CIRCLE (230,150),30,2
```

```
40 X=210:Y=50:LINE (50,30)-STEP(X,Y),2,BF
50 '--- Anzeigen Text ---
60 OPEN "GRP:" AS #1                    :' Eröffnen der Datei
70 PSET(50,90),1 :PRESET (50,90)        :' LP und Farbe setzen
75 PRINT#1,"RECHTECK"                   :' Ausgabe Text
80 PSET(210,90),3:PRESET (120,90)       :' LP und Farbe setzen
85 PRINT#1,"KREIS"                      :' Ausgabe Text
90 PSET (65,53),0
100 PRINT#1,"Fläche:";X;"*";Y;"Pixel)
120 CLOSE #1
130 PAUSE
```



Löschen von Beschriftungen im Grafikmodus

Auch im Grafikmodus kann es, z.B. für die Dialoggestaltung erforderlich sein, bereits geschriebene Texte zu löschen. Durch den vergleichsweise langsamen Bildaufbau in diesem Modus ist es aber oft wünschenswert, nur eine teilweise Löschung von Text (evtl. auch von Grafik) auszuführen, dabei aber den wesentlichen Bildinhalt beizubehalten. Die im Grafikmodus auf den gesamten SCREEN wirkende CLS-Anweisung läßt sich dafür nicht verwenden.
Auch ein "Überschreiben" des Textes durch einen anderen führt nicht zum Ziel, da die Buchstaben dann nur "übereinander" angezeigt werden.
Deshalb seien hier zwei Möglichkeiten der selektiven Bildlöschung skizziert.

Beispiele:

1. Ein Text kann gelöscht werden, indem der gleiche Text an der gleichen Position in der Hintergrundfarbe nochmals geschrieben wird. Im vorhergehenden Beispiel können dazu die folgenden Zeilen ergänzt werden:

   ```
   110 PAUSE
   112 PSET (50,90),0: PRINT#1,"RECHTECK"
   114 PSET (210,90),0:PRINT#1,"KREIS"
   ```

2. Die zu löschende Fläche kann durch ein Rechteck begrenzt und in der gewünschten Hintergrundfarbe ausgefüllt werden. Statt der Zeilen 112 bis 114 könnte dann diese Ergänzung erfolgen:

   ```
   112 LINE (50,90)-STEP(8*8-1,7),0,BF
   114 LINE (210,90)-STEP(5*8-1,7),0,BF
   116 LINE (65,53)-STEP(220*8-1,7),2,BF
   ```

   Beachten Sie bei dieser Variante, daß ein durch WINDOW definiertes Koordinatensystem bei der Berechnung des Rechtecks gegebenenfalls berücksichtigt werden muß.

Tastatureingaben im Grafikmodus

Wie Sie sicher bereits bemerkt haben, kann die im Textmodus übliche INPUT-Anweisung im Grafikmodus nicht zur Eingabe benutzt werden. Für die Dialogführung ist es aber häufig erforderlich, daß der Nutzer sowohl einen Informationstext als Eingabeaufforderung als auch ein unmittelbares

Echo seiner Tastatureingaben am Bildschirm erhält. Für eine solche "Eingabe im Grafikmodus" stellt RBASIC keine spezielle Anweisung bereit, jedoch läßt sich dieses Problem auf verschiedene Weise lösen.

Eine einfache Variante ist im folgenden Beispiel unter Verwendung der Anweisungen PRINT# und INPUT# angedeutet. Weitere Möglichkeiten können Sie sich entsprechend dem konkreten Dialogaufbau selbst überlegen.

```
10 SCREEN 2: COLOR 1,0:CLS
20 OPEN "GRP:" AS #1
30 PRESET (20,25) : PRINT#1,"Zeichnen eines Rechtecks"
40 PRESET (20,15) : PRINT#1,"Breite (<=250): ";
50 GOSUB 100:A=X:IF X>250 THEN PSET STEP(128,0),0:PRINT#1,Z$:GOTO40
60 PRESET (20,5):PRINT#1,   "Höhe   (<=150): "
70 GOSUB 100:B=X:IF X>150 THEN PSET STEP(128,0),0:PRINT#1,Z$:GOTO60
80 LINE (20,40)-STEP(A,B),2,B
90 PAUSE
95 CLOSE:END                                          :' Programmende
100 '--- UP Eingabe dreistelliger Zahlen ---
110 Z$=""
120 FOR I=1 TO 3
130     H$=INPUT$(1)                                  :' Eingabe Ziffer
140     IF H$<"0" OR H$>"9" THEN 130
150     PRINT#1,H$;                                   :' Anzeige Ziffer
160     Z$=Z$+H$
170 NEXT I
180 X=VAL(Z$)
190 RETURN
```

5. RBASIC für Fortgeschrittene

5.1. Fehler in Anweisungen und Programmen

Fehlerbehandlung

ON ERROR GOTO	Festlegen eines Fehlerbehandlungsprogramms
RESUME	Rückkehr aus der Fehlerbehandlung
ERR	Fehlernummer
ERL	Fehlerzeile

Beim Erarbeiten von Programmen treten nahezu immer Programmierfehler auf, die in einer Testphase vor der eigentlichen Nutzung der Programme erkannt und beseitigt werden sollten. Es kann vorkommen, daß in eigentlich richtigen Programmen Variable unzulässige Werte annehmen, z.B. durch ungenügend kontrollierte oder unsinnige Eingaben. Diese und ähnliche Umstände würden zu Fehlern führen, die eine normale Weiterarbeit im Programm bzw. die Ausführung von Anweisungen im Kommandomodus un-möglich machen.

Auf fehlerhafte Anweisungen und Programmfehler, die die verschiedensten Ursachen haben können, reagiert der RBASIC-Interpreter auf drei mögliche Arten. Auf welche, ist dabei abhängig vom Arbeitsmodus des Interpreters und davon, ob eine programmierte Fehlerbehandlung erfolgen soll oder nicht.

Jeder Fehler ist durch seine Nummer und die Fehlermeldung (den zum Fehler gehörenden Text) charakterisiert. Fehlernummern und Fehlermeldungen finden Sie im Anhang H. Auf Fehler im Kommandomodus reagiert der RBASIC-Interpreter sofort mit der Ausgabe der Fehlermeldung, zum Beispiel auf

```
READ A      mit    Out of DATA,
```

da die READ-Anweisung eine Programmzeile mit einer DATA-Anweisung voraussetzt. Die Fehlernummer ist abrufbar über die Funktion ERR.

```
PRINT ERR
 4
```

Laufende Programme werden beim Auftreten eines Fehlers standardmäßig abgebrochen. Der RBASIC-Interpreter geht in den Kommandomodus über, die Fehlermeldung wird angezeigt, und die Funktionen ERR und ERL liefern die Angaben über den Fehler:

ERR - Fehlernummer,
ERL - Nummer der Programmzeile, in der der Fehler auftrat.

Die Anzeige der Fehlermeldung erfolgt mit zusätzlicher Ausgabe der Zeilennummer von ERL.

Beispiel: Nach dem Fehler "Subscript out of range in 80" liefern die Funktionen ERR und ERL zum Beispiel:

```
PRINT ERR, ERL
 9           80
```

Programmiert Fehlerbehandlung mit ON ERROR

Format: **ON ERROR GOTO** *zeilennummer_a*

zeilennummer_a - Beginn des Programmteiles zur Fehlerbehandlung

Funktion: Mit ON ERROR GOTO wird die Stelle im Hauptprogramm bestimmt, an der im Falle eines Fehlers weitergearbeitet werden soll.
Mit ON ERROR GOTO 0 wird die programmierte Fehlerbehandlung wieder ausgeschaltet.

Format: **RESUME**
RESUME NEXT
RESUME *zeilennummer_b*

zeilennummer_b - Fortsetzungszeile nach einer Fehlerbehandlung

Funktion: Das Teilprogramm zur Fehlerbehandlung wird verlassen, und die Programmabarbeitung wird fortgesetzt mit

RESUME oder RESUME 0	- in der Fehlerzeile (ERL) der fehlererzeugenden Anweisung
RESUME NEXT	- in der folgenden Anweisung
RESUME zeilennummer_b	- in der angegebenen Zeile

Format: ERR
ERL

Typ: BASIC-Funktion

Funktion: ERR und ERL liefern in Teilprogrammen zur Fehlerbehandlung Angaben darüber, in welcher Programmzeile (ERL) welcher Fehler (ERR) aufgetreten ist.

Hinweis: Über ON ERROR GOTO...RESUME ist es einfach möglich, deutschsprachige Fehlermeldungen zu erzeugen.

Beispiel:
```
10 ON ERROR GOTO 100
20 INPUT "X=";X
30 PRINT SQR(X)
40 IF X=0 THEN 60
50 GOTO 20
60 END
100 REM --- Fehlerbehandlung ---
110 PRINT "Fehler";
120 IF ERR = 5 THEN 140
130 PRINT ERR :RESUME 60
140 PRINT "nur Zahlen >=0 eingeben!"
150 RESUME NEXT
```

Zeile	Erläuterung
10	Im Fehlerfall wird in Zeile 100 weitergearbeitet
20 ... 60	Hauptprogramm. Vom eingegebenen Wert X wird die Quadratwurzel berechnet und ausgegeben. Ist X=0, so wird das Pro-gramm beendet, ansonsten wird auf die Eingabe des nächsten X gewartet.
100 ... 150	Fehlerbehandlung. Wurde in Zeile 20 ein negativer Wert eingegeben, so entsteht normalerweise in Zeile 30 ein "Illegal function call" mit der Fehlernummer 5. In der Fehlerbehandlung wird hier die Aufforderung aus Zeile 140 ausgegeben, und das Programm wird fortgesetzt (in Zeile 40). Bei anderen Fehlern (die eigentlich nicht auftreten dürften) wird das Programm beendet, in Zeile 60.

Fehlererzeugung, Fehlersimulation mit ERROR

Format: **ERROR** *fehlernr*

fehlernr - Nummer des zu erzeugenden Fehlers

Funktion: Der RBASIC-Interpreter verhält sich so, als wäre in der Programmzeile mit der ERROR-Anweisung der Fehler tatsächlich aufgetreten.

Hinweise:

1. Mit der ERROR-Anweisung können so an beliebiger Stelle Fehlerbedingungen gesetzt werden, für die die entsprechenden Bedeutungen feststehen; Fehler können so simuliert werden.

2. Es können auch Fehlerbedingungen gesetzt werden, für die im RBASIC keine speziellen Bedeutungen feststehen und die als nicht druckbare Fehler (unprintable errors) ausgewiesen werden (s. Anhang Fehlermeldungen). Diese Fehlerbedingungen können im Programm mit eigenen Bedeutungen versehen werden.

Beispiele:

1. Das vorhergehende Programmbeispiel soll so geändert werden, daß die Werte für die Quadratwurzel für Zahlen zwischen 0 und 20 berechnet werden. Dazu ist Zeile 25 einzufügen

```
25 IF X>20 THEN ERROR 5
```

und Zeile 140 ist zu ändern in

```
140 PRINT ":Zahlen von 0 bis 20 eingeben!"
```

Für Zahlen <0 erzeugt nach wie vor Zeile 30 die Fehlerbedingung.

2. Das Programm könnte entsprechend Hinweis 2 auch folgendermaßen geändert werden:

```
25 IF X>20 THEN ERROR 100
125 IF ERR=100 THEN 135
135 PRINT ":Zahlen <=20 eingeben!":RESUME 20
140 PRINT ":Zahlen >=0 eingeben!"
```

5.2. Ereignis- und zeitabhängige Unterbrechungen

Anweisungen zur Unterbrechungsbehandlung

ON INTERVAL GOSUB	)	Zeitunterbrechung
INTERVAL ON/OFF/STOP	)	
ON KEY GOSUB	)	Unterbrechung durch
KEY(n) ON/OFF/STOP	)	Funktionstasten
ON STOP GOSUB	)	Unterbrechung durch
STOP ON/OFF/STOP	)	CTRL + STOP
ON STRIG GOSUB	)	Unterbrechung durch
STRIG(n) ON/OFF/STOP	)	Leer- bzw. Aktionstasten

Unter einer Unterbrechung (einem Interrupt) verstehen wir hier die Unterbrechung der Abarbeitung des laufenden Hauptprogramms. Das Hauptprogramm wird verlassen, und die Abarbeitung wird im Unterprogramm zur Unterbrechungsbehandlung fortgesetzt. Die Weiterarbeit im Hauptprogramm erfolgt nach Beendigung des Unterprogramms durch RETURN.
RBASIC kennt vier Arten von Unterbrechungen, die jeweils mit ähnlichen Anweisungsgruppen bearbeitet werden können. Diese Unterbrechungen sind entweder an bestimmte Ereignisse gebunden (z.B. Drücken einer Funktionstaste) oder kehren in festgelegten Zeitabständen wieder.

Mit ON...GOSUB werden die Zeilennummern festgelegt, bei denen die Unterprogramme zur Unterbrechungsbehandlung beginnen, wobei für jede Unterbrechungsquelle eine andere Zeilen-

nummer angegeben werden kann. Eine Zeilennummer 0 hebt diese Zuordnung wieder auf. Die Anweisungen ...ON, ...OFF, ...STOP bestimmen die Art und Weise, wie die Unterbrechungen bearbeitet werden.

...ON Unterbrechungsanmeldung erlauben,
Unterbrechungsanmeldung im Unterprogramm bearbeiten, das zur Unterbrechungsquelle gehört.

...OFF Unterbrechungsanmeldung verbieten (Standard).

...STOP Unterbrechungsanmeldung erlauben,
Bearbeitung bis zum nächsten ...ON aussetzen und dann ausführen

Die Wirkungsweise der einzelnen Anweisungen wird am Beispiel der Funktionstasten ausführlich erläutert und gilt in gleicher Weise auch für die anderen Unterbrechungsquellen.

Hinweis: Alle Anweisungen ON...GOSUB können nicht in Teilprogrammen zur Fehlerbehandlung verwendet werden.

Unterbrechungen durch Funktionstasten

Format: **ON KEY GOSUB** *zlnr1*[,*zlnr2*[,*zlnr3*[...[,*zlnr10*]...]]]
ON KEY GOSUB [*zlnr1*],[*zlnr2*],...,*zlnr10*
KEY(n) ON
KEY(n) OFF
KEY(n) STOP

zlnr1 ... *zlnr10* - Zeilennummern der Unterprogramme für die Behandlung der Unterbrechungen durch die einzelnen Funktionstasten.

zlnr1 - Zeilennummer für PF-Taste 1 usw.
Die Zeilennummern *zlnr1* bis *zlnr10* können von rechts oder einzeln weggelassen werden (0...65529)

n - Nummer der Funktionstaste

Funktionen: Mit ON KEY GOSUB wird jeder gewünschte Funktionstaste ein Unterprogramm zur Unterbrechungsbehandlung zugeordnet, gleiche Zeilennummern für verschiedene Funktionstasten sind möglich.

Mit KEY(n) ON wird die Anmeldung und Bearbeitung einer Unterbrechung durch Funktionstaste *n* erlaubt, mit KEY(n) OFF verboten. KEY(n) STOP erlaubt die Anmeldung der Unterbrechung, die Bearbeitung wird aber ausgesetzt bis zum nächsten KEY(n) ON.

Hinweise:

1. Die Anweisungen KEY(n) ON/OFF/STOP müssen für jede Funktionstaste als Unterbrechungsquelle einzeln ausgeführt werden.

2. Je Unterbrechungsquelle ist eine Anmeldung zur Unterbrechungsbehandlung möglich.

3. Mit Beginn der Unterbrechungshandlung gilt KEY(n) STOP, bis das Unterprogramm verlassen wird. Andere Einstellungen können im Behandlungsprogramm vorgesehen werden. Die Unterbrechung durch andere Funktionstasten wird davon nicht beeinflußt, d.h., auch das Programm zur Unterbrechungsbehandlung kann unterbrochen werden.

4. Nach dem RETURN des Behandlungsprogramms gilt wieder KEY(n) ON, mit RETURN *zeilennummer* bleibt die letzte Einstellung gültig.

Beispiel: Das folgende Beispiel mit den unten angegebenen Ergänzungen verdeutlicht die Wirkungsweise der Anweisungen zur Unterbrechungsbehandlung bei verschiedenen Einstellungen. Ausführliche Erläuterungen folgen im Anschluß an das Programm.

```
10  ON  KEY  GOSUB  100,,200       :'  Behandlungsprogramme
festlegen
20 KEY(1) ON                  :' Unterbrechung KEY(1) erlauben
30 KEY(3) ON                  :' Unterbrechung KEY(2) erlauben
40 CLS
50 LOCATE 0,5                 :' Anfang Schleife
60 K=K+1
70 PRINT K
80 GOTO 50                    :' Ende Schleife
100 REM --- Unterbrechung durch PF1 ---
110 CLS
120 LOCATE 10,5
130 PRINT"Taste PF1"
140 FOR I=0 TO 39
150     LOCATE 11,I:PRINT"X"
160     PAUSE 2
170 NEXT I
180 CLS
190 RETURN
200 REM --- Unterbrechung durch PF3 ---
210 LOCATE 15,5:PRINT"Taste PF3"
220 FOR J=1 TO 500:NEXT
230  LOCATE  15,5:PRINT"              "     :'  9 Leerzeichen
ausgeben
240 LOCATE 0,5
250 RETURN
```

Zeile	**Erläuterungen**
10 ... 30	Festlegungen der Unterbrechungsbearbeitung für PF1 und PF3
40 ... 80	Hauptprogramm, fortlaufende Ausgabe von Zahlen
100 ... 190	Unterprogramm zur Behandlung von Unterbrechungen durch PF1; nach der Ausschrift "Taste PF1" wird Zeile 11 verzögert mit X beschrieben.
200 ... 250	Unterprogramm zur Behandlung von Unterbrechungen durch PF3, die Meldung "Taste PF3" wird ausgegeben und wieder gelöscht. Zeile 240 regeneriert den Kursor für das Hauptprogramm.

Das Hauptprogramm und auch wechselseitig die Unterprogramme zur Unterbrechungsbehandlung können unterbrochen werden.
Mit dem Start der Unterprogramme zur Unterbrechungsbehandlung wird automatisch ein KEY(n) STOP erzeugt. das bedeutet, daß erneute Unterbrechungsanmeldungen durch die gleiche Taste zwar angenommen, aber erst nach dem zum Unterprogramm gehörenden RETURN bearbeitet werden.
Wenn zweimal hintereinander PF1 gedrückt wird, wird zweimal hintereinander Zeile 11 vollständig mit X beschrieben, d.h., eine erneute Behandlung der Unterbrechung erfolgt erst, wenn die vorhergehende abgeschlossen ist.
Eine Unterbrechung mit Taste PF3 kann immer erzeugt werden.

Mit den Einstellungen der Anweisung

```
105 KEY(1) ON
```

ändert sich die Abarbeitung so, daß das Programm zur Behandlung von PF1 ebenfalls unterbrochen werden kann.
Das Beschreiben der Zeile 11 mit X wird jetzt nach Drücken von PF1 abgebrochen und von vorn begonnen.

Mit der Anweisung

```
155 IF I=20 THEN KEY(1) STOP
```

werden erneute Unterbrechungen durch Drücken der Taste PF1 nur so lange sofort bearbeitet, wie die linke Hälfte der Zeile 11 beschrieben wird.

Wenn während der Bearbeitung der Unterbrechungsanforderung durch PF1 die Unterbrechung durch PF3 unterdrückt werden soll, so kann das erreicht werden durch

```
106 KEY(3) STOP
185 KEY(3) ON.
```

Unterbrechung durch CTRL + STOP

Format: **ON STOP GOSUB** *zeilennummer*

STOP ON
STOP OFF
STOP STOP

zeilennummer - Zeilennummer der ersten Anweisung zur Behandlung der Stoptasten-unterbrechung (0...65529)

Funktionen: Mit ON STOP GOSUB wird der Stoptaste CTRL + STOP ein Unterprogramm zugeordnet. Mit STOP ON wird die Anmeldung und Bearbeitung der Unterbrechung erlaubt, mit STOP OFF verboten. STOP STOP erlaubt die Anmeldung der Unterbrechungen, die Bearbeitung wird aber ausgesetzt bis zum nächsten STOP ON.

Hinweise:

1. Für die Dauer der Unterbrechungsbearbeitung gilt wieder STOP STOP.

2. Ein Programm mit einem Unterprogramm zur Stoptastenbehandlung kann nach STOP ON nicht mehr abgebrochen werden, denn nach CTRL + STOP wird ja das Unterprogramm angesprungen.

Beispiel:

```
10 ON STOP GOSUB 100
20 STOP ON
30 I=0
40 PRINT I
50 I=I+1
60 GOTO 40
100 REM --- Unterbrechung mit CTRL+STOP ---
110 CLS
120 PRINT "CTRL-STOP-Unterbrechung"
130 RETURN
```

Zeile	Erläuterung
10, 20	CTRL+STOP-Unterbrechung erlauben
30 ... 60	Hauptprogramm, fortlaufende Ausgabe von Zahlen
100 ... 130	Unterprogramm zur Behandlung der CTRL+STOP-Unterbrechung. Der Bildschirm wird gelöscht, und die Meldung von Zeile 120 wird ausgegeben.

Achtung: Speichern Sie das Programm ab, bevor Sie es starten!
Nach RESET müßten Sie es erneut eingeben!

Mit den folgenden zusätzlichen Anweisungen läßt sich das Programm nach Drücken von PF1 wieder abbrechen.

```
15 ON KEY GOSUB 200
16 KEY(1) ON
200 REM --- CTRL+STOP-Unterbrechung verbieten ---
210 STOP OFF
220 RETURN
```

Zeitabhängige Unterbrechung

Format: **ON INTERVAL** = *zeit* **GOSUB** *zeilennummer*

INTERVAL ON
INTERVAL OFF
INTERVAL STOP

zeit - Zeitintervall zwischen zwei Unterbrechungsanmeldungen, Angabe in 1/50 Sekunden. 0 <= zeit <= 65536

zeilennummer - Zeilennummer des Unterprogramms zur Behandlung der zeitabhängigen Unterbrechung (0...65529).

Funktionen: Mit **ON INTERVAL** = *zeit* **GOSUB** wird vereinbart, in welchem Zeitabstand eine erneute Unterbrechungsanmeldung veranlaßt wird. Mit **INTERVAL ON** wird die Anmeldung und Bearbeitung der zeitabhängigen Unterbrechung erlaubt, mit **INTERVAL OFF** verboten. **INTERVAL STOP** erlaubt die Anmeldung der Unterbrechung, die Bearbeitung wird aber ausgesetzt bis zum nächsten **INTERVAL ON**.

die Zeitangabe erfolgt in 1/50 Sekunden, d.h., für *zeit* = 50 wird das Behandlungs-progamm genau alle ganzen Sekunden aufgerufen.

Hinweise:

1. Mit Beginn der Unterbrechungsbehandlung gilt wieder INTERVAL STOP.
2. Das maximal angebbare Zeitintervall beträgt 1310,72 Sekunden, das sind knapp 22 Minuten.

Beispiel:

```
10 ON INTERVAL = 50 GOSUB 200
20 INTERVAL ON
30 IF INKEY$="" THEN 30
40 END
200 REM --- zeitabhängige Unterbrechung ---
210 BEEP
220 RETURN
```

Zeile	Erläuterungen
10, 20	Zeitabhängige Unterbrechung erlauben. Zeitintervall 1 Sekunde
30, 40	Hauptprogramm, auf Tasteneingabe in Zeile 30 warten, dann Ende in Zeile 40.
200 ... 220	Unterprogramm zur Behandlung der zeitabhängigen Unterbrechung, Erzeugung eines Pieptones.

Mit dem Programm wird so lange im Sekundenrhythmus ein Piepton erzeugt, bis eine Taste gedrückt wird.

Unterbrechung durch Aktionstasten bzw. die Leertaste

Format: **ON STRIG GOSUB** *zlnr0*,*zlnr1*,*zlnr2*,*zlnr3*,*zlnr4*

Funktion und Verwendung dieser Anweisung werden in Abschnitt 7.2 im Zusammenhang mit der Nutzung von Steuerhebeln erklärt.

5.3. Bildspeicher-SCREEN-Anweisung

SCREEN Festlegen des SCREEN-Modus und der aktiven und visuellen Seiten

Aktive und visuelle SCREENs

Bisher haben Sie für die Bildausgabe

SCREEN	0 bzw. 8) 1 bzw. 9)	für Textdarstellung
SCREEN	2) 3) 5)	für Grafik

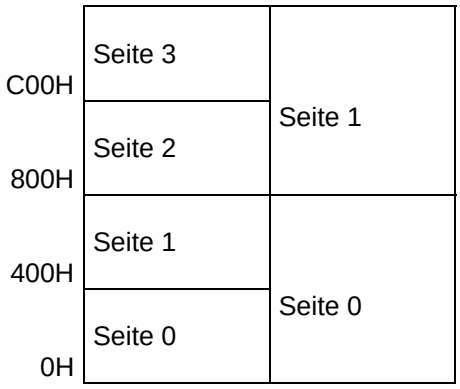
kennengelernt. Der sehr große Bildspeicher des Computers (64 k x 16 Bit Video-RAM) bietet aber weitaus mehr Möglichkeiten der Bildgestaltung. Er ist aufgeteilt in

16 k x 16 Bit	für Text-SCREENs (Video-RAM-Bereich: 0H - 3FFFH)
48 k x 16 Bit	für Grafik-SCREENs (Video-RAM-Bereich: 4000H - FFFFH)

Der Bildbereich für die Textdarstellung bietet Platz für 16 bzw. 8 unabhängige Bildschirmseiten, je nachdem, ob 40 oder 80 Zeichen je Zeile dargestellt werden sollen. Die Aufteilung der Bildschirmseiten entnehmen Sie bitte der folgenden Tabelle.

	SCREEN 0,8	SCREEN 1,9
3FFFH	Seite 15	Seite 7
3C00H	Seite 14	
3800H	Seite 13	Seite 6
3400H	Seite 12	
3000H	Seite 11	Seite 5
2C00H	Seite 10	
2800H	Seite 9	Seite 4
2400H	Seite 8	
2000H	Seite 7	Seite 3
1C00H	Seite 6	
1800H	Seite 5	Seite 2
1400H	Seite 4	
1000H		

	SCREEN 2	SCREEN 3,5
FF40H	Systembereich	
E000H	Seite 5	Seite 2
C000H	Seite 4	
A000H	Seite 3	Seite1
8000H	Seite 2	
6000H	Seite 1	Seite 0
4000H	Seite 0	



Nach dem Starten von RBASIC bzw. nach RESET ist standardmäßig die Seite 0 im 40-Zeichen-Modus eingeschaltet. Durch SCREEN 1 wird Seite 0 des 80-Zeichen-Modus eingestellt.
Ebenso bietet der Bereich für die Grafik-SCREENs Platz für 6 bzw. 3 unabhängige Bildschirmseiten, je nachdem, ob SCREEN 2, SCREEN 3 oder SCREEN 5 eingeschaltet ist.

Auch hier wird durch SCREEN 2, SCREEN 3 bzw. SCREEN 5 standardmäßig die Seite 0 zugewiesen.

Durch Nutzung dieser vielen einfachen programmierbaren Bildschirmseiten lassen sich eine Reihe besonderer gestalterischer Effekte erreichen.
Von besonderer Bedeutung ist die Tatsache, daß Sie dabei zwischen einer aktiven und einer visuellen Bildschirmseite (SCREEN-Seite) unterscheiden können.

Aktive SCREEN-Seite: bezeichnet den Bereich (Seite) des Bildspeichers, auf den geschrieben (Text-SCREEN) bzw. gezeichnet (Grafik-SCREEN) werden soll. Zur eindeutigen Bezeichnung gehört die Festlegung eines aktiven SCREEN-Modus und einer aktiven SCREEN-Seite.

Visuelle SCREEN-Seite: bezeichnet die gerade am Bildschirm sichtbare Seite des Bildspeichers (Video-RAMs); sie kann unabhängig von der aktiven SCREEN-Seite definiert werden. Zur eindeutigen Bezeichnung gehört die Festlegung des visuellen SCREEN-Modus und der visuellen SCREEN-Seite.

Zur Festlegung der aktiven und visuellen SCREEN-Seiten kann die bekannte SCREEN-Anweisung in folgender Weise erweitert werden:

Format: **SCREEN** [*aktivmodus*],[*aktivseite*],[*vismodus*][,*visseite*]

aktivmodus - SCREEN-Modus, in dem die Bildausgabe (Text oder Grafik) erfolgt

aktivseite - SCREEN-Seite zu *aktivmodus*, in welche die Information (Text oder Grafik) ausgegeben wird

vismodus - SCREEN-Modus des angezeigten Bildes

visseite - Nummer der SCREEN-Seite, die im *vismodus* am Bildschirm angezeigt wird

Funktion: Festlegung der aktiven und visuellen Bildausgabemodi und -seiten.

Hinweise:
1. Die Parameter der SCREEN-Anweisung können einzeln oder von rechts weggelassen werden.
   - Fehlt *aktivmodus*, wird der Modus der vorhergehenden SCREEN-Einstellung verwendet.
   - Fehlt *aktivseite*, wird die Seitennummer der vorhergehenden SCREEN-Einstellung verwendet.
   - Fehlt *vismodus*, wird der aktive Modus übernommen.
   - Fehlt visseite, wird die Nummer der aktiven Seite übernommen.

2. Bei Programmende oder -abbruch, Fehlermeldungen, LIST-Ausgaben wird stets auf Seite 0 des letzten verwendeten Textmodus zurückgeschaltet.

3. Wird durch die SCREEN-Umschaltung ein Speicherbereich in einem anderen Modus verwendet, löscht der Computer vor dem Zuschalten den betreffenden Bereich.

   Beim Umschalten von Text- auf Grafik-SCREEN und umgekehrt bleibt der Bildinhalt erhalten. Ebenso ist das Umschalten zwischen verschiedenen Seiten im gleichen Modus ohne Verlust des Bildinhaltes möglich.

4. Der Kursor wird beim Umschalten auf Textseite 0 auf

   die alte Position,

   beim Umschalten auf andere Seiten nach

   links oben im Textmodus,
   links unten im Grafikmodus

   positioniert.

5. Die SCREEN-Umschaltung verändert die Farbeinstellung nicht. Wird eine neue aktive Seite eingestellt, müssen die seitenspezifischen Farben nach der SCREEN-Anweisung mit einer COLR-Anweisung eingestellt werden.

6. Beim Einstellen einer Textseite wird jeweils das Standard-WINDOW angenommen. Im Grafikmodus beeinflußt die SCREEN-Anweisung das durch WINDOW eingestellte Koordinatensystem nicht.

Beispiele:

1. Während der Anzeige eines Textbildes kann im Hintergrund z.B. eine grafische Darstellung vorbereitet werden. Nach Drücken von ENTER wird die Grafik sofort sichtbar.

```
10 SCREEN 0:COLOR 6,0,0:CLS
20 PRINT"Der Computer zeichnet im Hintergrund!"
30 SCREEN 2,0,0,0 : COLOR 0,0 :CLS :' Zeichen
40 FOR R=10 TO 80 STEP 10          :' einer neuen
50     CIRCLE (100,100),R          :' aktiven Seite
60 NEXT R                          :' im Hintergrund
70 SCREEN 0
80 LOCATE 24,35
90 INPUT "ENTER";A
100 SCREEN 2                       :' Anzeigen des Bildes
110 PAUSE
```

2. Durch den Aufbau mehrerer unabhängiger Grafikbilder können schnell bewegte Bilder leicht programmiert werden.

```
10 FOR I=0 TO 5                    :' Aufbau der Bilder
20     COLOR 0,0 :SCREEN 2,I
30     CIRCLE(160,100),10*(I+1),1
40     PAINT (160,100),2,1
50 NEXT I
60 FOR I=0 TO 5                    :' Wechseln der Bilder
70     SCREEN ,,,I
80     PAUSE 10
90 NEXT I
100 IF INKEY$=""<THEN 70
```

   Mit dem Parameter der PAUSE-Anweisung in Zeile 90 läßt sich die Geschwindigkeit der Bewegung beeinflussen.

5.4. Änderung des Zeichensatzes

CHR$ Änderung eines Zeichens

Nach dem Starten des RBASIC-Interpreters wird stets der gleiche Standardzeichensatz aktiviert (vgl. Anhang A). Zur Erweiterung der Darstellungsmöglichkeiten können Sie jedes dieser Zeichen mit der Anweisung CHR$ noch definieren.

Format: **CHR$(***n*) = *zeichenkette*

n - Nummer des Zeichens in der Codetabelle
zeichenkette - 8-Byte-Zeichenkette, die das neue Zeichenmuster definiert

Funktion: Festlegung des Zeichenmusters, eines Zeichens zur Darstellung auf dem Bildschirm.

Hinweise:

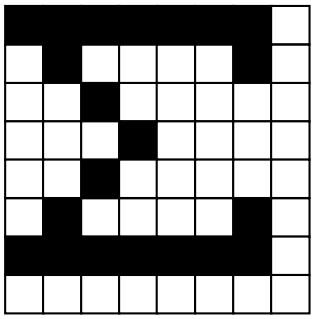
1. Ein Zeichen belegt stets 8 x 8 Bildpunkte. Zur Definition des Zeichens müssen Sie festlegen, welche dieser Punkte in der Vordergrundfarbe und welche in der Hintergrundfarbe dargestellt werden sollen.

2. Nach Ausführung der CHR$-Anweisung wird auf allen Textseiten das alte Zeichen durch das neu definierte ersetzt.

3. Auf den Grafikseiten bleiben alle vorher ausgegebenen Zeichen unverändert. Die CHR$-Anweisung wirkt nur auf nachfolgende Ausgaben.

4. Die Ausgabe des Zeichens *n* auf einen angeschlossenen Drucker wird von der Anweisung CHR$ nicht beeinflußt, da der Zeichensatz des Druckers durch CHR$(n)=... nicht verändert wird.

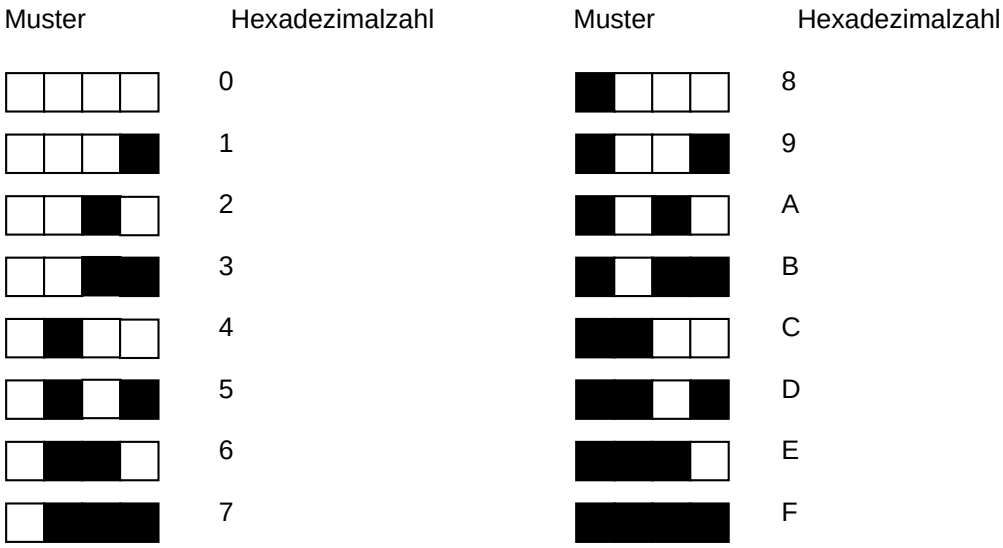
Beispiel: Ersetzen des Zeichens "S" (83) durch das Summenzeichen. Tragen Sie zunächst das neue Zeichen in ein 8x8-Raster ein.



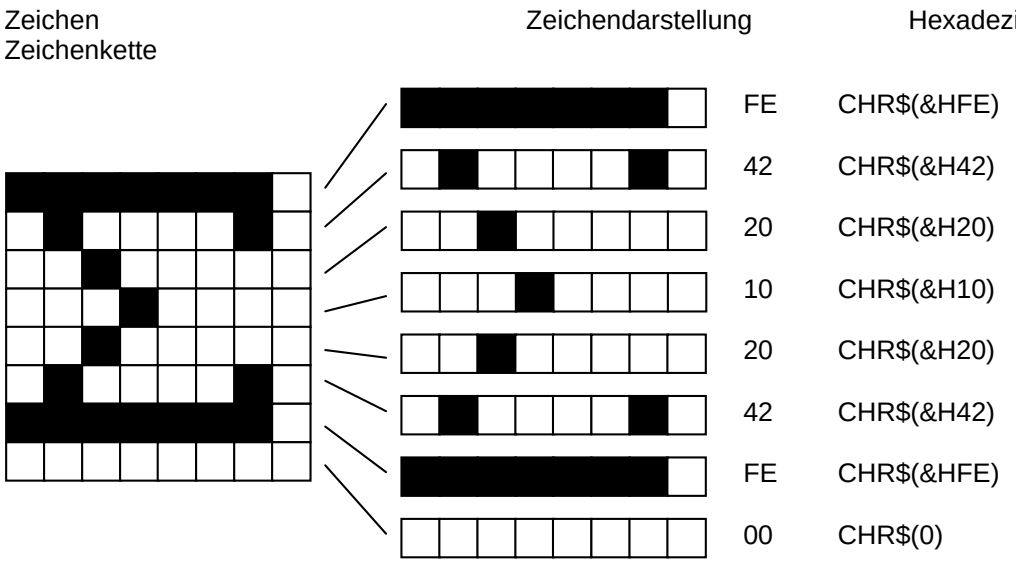
Danach zerlegen Sie dieses Muster in 8 Zeilen. Für jeden Punkt mit Vordergrundfarbe wird eine Eins und für jeden Punkt mit Hintergrundfarbe eine Null eingetragen.

```
1 1 1 1 1 1 1 0
0 1 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 1 0
1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0
```

Danach kann jede Zeile als achtstellige Binärzahl dargestellt werden. Für die erste Zeile des Beispiels ergibt das die Zahl &B11111110. Zur Verkürzung der Schreibweise können ebenfalls Hexadezimalzahlen oder auch Dezimalzahlen verwendet werden.
Nach Unterteilung der Zeile in 4 Punkte links und 4 Punkte rechts und unter Nutzung der untenstehenden Tabelle erfolgt die Umwandlung in eine zweistellige Hexadezimalzahl.



Mit dem Muster  der ersten Zeile ergibt das die Hexdazimalzahl &HFE. Dieses Bitmuster wird folglich durch CHR$(&HFE) codiert. Die Zerlegung des oben beschriebenen Zeichenmusters wird folgendermaßen vorgenommen:



Aus den 8 Bytes, die jeweils eine Zeile vertreten, wird eine Zeichenkette gebildet. Dabei können die bekannten Zahlendarstellungen (hexadezimal, binär, dezimal) verwendet werden. Bei Codierungen, die zwischen &H20 und &HFF liegen, kann für die Darstellung in der Zeichenkette auch das entsprechende Zeichen der ASCII-Tabelle verwendet werden. Für das Summenzeichen ergibt sich z.B.:

```
ZE$ = CHR$(&HFE) + "B" + CHR$(&B00100000) + CHR$(&H10) +
      CHR$(&H20) + CHR$(66) + CHR$(&HFE) + CHR$(0)
```

Mit der Anweisung

```
CHR$(83) = ZE$
```

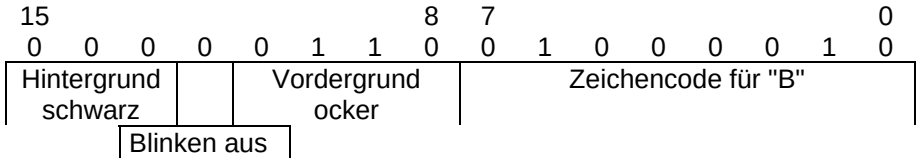
ordnen Sie dem Zeichen 83 das mit der Zeichenkette ZE$ neu definierte Summenzeichen zu.

5.5. Informationen zum Bildspeicher

VDEEK Lesen eines Wortes aus dem Bildspeicher
VDOKE Schreiben eines Wortes in den Bildspeicher

Wie Sie bereits aus Abschnitt 5.3 wissen, besitzt der Computer einen separaten Bildspeicher von 64 k * 16 Bit, d.h., er vermag 65535 16-Bit-Worte zu speichern und ist in Text- und Grafikbereich unterteilt. Auch die Anfangsadressen der Bildschirmseiten können Sie dort nachlesen. Die folgenden Informationen erlauben die Nutzung einer weiteren Ausgabemöglichkeit auf den Bildschirm und die Manipulation im Bildspeicher. Dabei ist zu beachten, daß vom Betriebssystem der freie Bereich hinter den Bildschirmseiten genutzt wird. Zwischen den Adressen 0FF40H und 0FFE0H liegt der Bereich für die Anzeige der Belegung der programmierten Funktionstasten.
Der Zugriff auf den Bildspeicher wird vom Grafik-Display-Controller gesteuert.
Entsprechend der Unterscheidung in Text- und Grafikmodus haben die 16 Bits eines Bildspeicherplatzes unterschiedliche Bedeutung.
Im Textmodus wird in einem Wort der Inhalt eines Zeichens im Bildspeicher abgespeichert. Bit 0 bis 7 (Low-Byte) enthalten den Zeichencode entsprechend ASCII-Tabelle (Anhang A) und Bit 8 bis 15 (High-Byte) den Farbcode.
Die Codierung von Bit 8 bis 11 bestimmt die Vordergrundfarbe entsprechend der Farbtabelle (s. Anhang E). Wenn Bit 12 den Wert 1 hat, blinkt das Zeichen. Die Bits 13 bis 15 enthalten die Codierung der Hintergrundfarbe.

Beispiel:

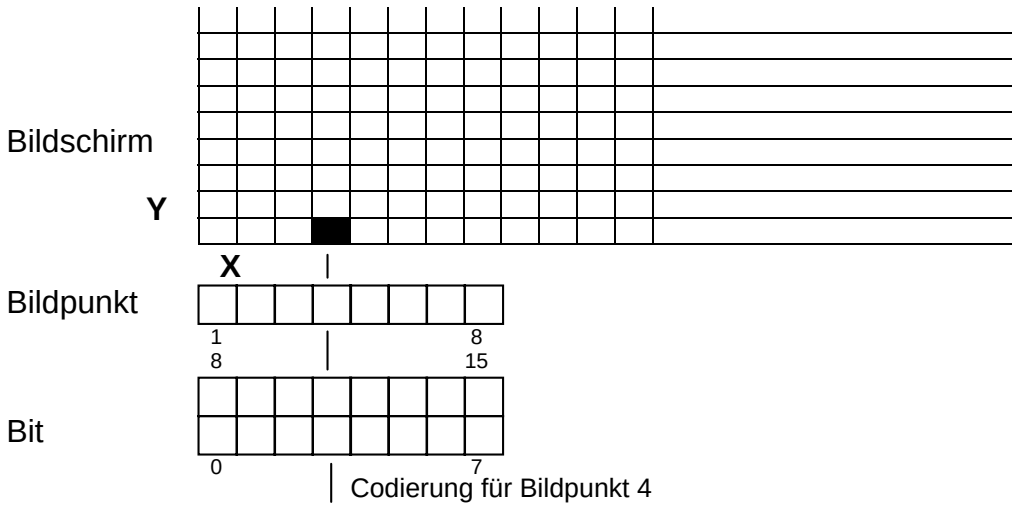


Die Adresse des Speicherwortes im SCREEN 0 wird berechnet nach:

```
Adresse = Anfangsadresse der Textseite
        + (Zeile * 40)                    ; bei SCREEN 1: (Zeile * 80)
```

Im Grafikbereich bestimmt der Inhalt eines Speicherwortes die Farbe von 8 (SCREEN 2 und 3) bzw. 4 (SCREEN 5) nebeneinanderliegenden Bildpunkten (Pixeln).
Bei SCREEN 2 und 3 enthalten jeweils 2 Bits die Codierung der Farbnummer für einen Bildpunkt (Pixel). Damit können die Hintergrundfarbe 0 oder die Farben 1 bis 3 der aktuellen Palette ausgewählt werden. Bit 8 und 0 bestimmen die Farbe des Bildpunktes, der am weitesten links steht, Bit 9 und 1 die Farbe des Bildpunktes rechts daneben und so weiter. Bit 15 und 7 bestimmen die Farbe des Bildpunktes, der von den 8 in einem Wort beschriebenen am weitesten rechts steht.
Die Adresse des Speicherwortes im Bildspeicher für den Bildpunkt (X,Y) ist nach folgender Vorschrift zu berechnen:

```
Adresse = Anfangsadresse der Grafikbildschirmseite
        + (199-Y) * 40                    ; bei SCREEN 3: (199-Y) * 80
        + (X \ 8)                         ; ganzzalige Division
```



Beispiel: Inhalt des Speicherwortes: 0CCAAH

```
8                       15
0  0   1   1   0   0   1   1
0  1   0   1   0   1   0   1
0                       7
```

Mit dem angegebenen Speicherwort und gewählter Palette 0 sowie der Hintergrundfarbe 0 haben die 8 Bildpunkte (Pixel) von links nach rechts folgende Farben:

. schwarz	(0)
. rot	(1)
. grün	(2)
. ocker	(3)
. schwarz	(0)
. rot	(1)
. grün	(2)
. ocker	(3)

Bei SCREEN 5 enthalten jeweils 4 Bits die Codierung der Farbnummer für einen Bildpunkt. Damit können die Farben 0 bis 15 der Farbtabelle direkt ausgewählt werden. Bit 12, 8, 4 und 0 des Speicherortes bestimmen die Farbe des Bildpunktes ganz links; Bit 13, 9, 5 und 1 die Farbe des Bildpunktes rechts daneben; Bit 14, 10, 6 und 2 die Farbe des Bildpunktes wiederum rechts daneben und Bit 15, 11, 7, und 3 die Farbe des Bildpunktes ganz rechts.
Die Adresse des Speicherortes wird berechnet:

Adresse = Anfangsadresse der Grafikbildschirmseite

+ (199-Y) * 80
+ (X \ 4) .

Unter Beachtung dieser Informationen können Sie direkt in den Bildspeicher schreiben bzw. daraus lesen. Dazu stehen die Anweisungen VDEEK und VDOKE zur Verfügung.

Format: **VDEEK**(*adresse*)

Typ: BASIC-Funktion

adresse - numerischer Ausdruck (ganzzahlig), der eine Adresse zwischen 0 und &HFFFF ergibt.

Funktion: Der Inhalt (16 Bit) des Bildspeicherplatzes mit der angegebenen Adresse wird als Funktionswert angenommen.

Beispiel:
```
10 SCREEN 0:COLOR 6,0,0:CLS
20 PRINT "robotron"
30 A$=HEX$(VDEEK(0))
40 PRINT "Code des 1. Zeichens:"; A$
```

Bei Abarbeitung des Programms erscheint auf dem Bildschirm links oben das Wort "robotron".
In Zeile 30 wird der Variablen A$ die Belegung des Speicherplatzes 0 (Zeichen links oben) zugewiesen, und in Zeile 40 wird diese Belegung auf dem Bildschirm angezeigt.
Die Anzeige 672 bedeutet:

Farbkombination ocker auf schwarz (06)
Zeichen 72 (hexadezimal) entspricht dem "r"

Format: **VDOKE** *adresse*,*wort*

adresse - Numerischer Ausdruck (ganzzahlig), der eine Adresse zwischen 0 und &HFFFF angibt.

wort - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis &HFFFF.

Funktion: Mit dieser Anweisung werden 16 Bits auf den Bildspeicherinhalt mit der angegebenen Adresse geschrieben.

Hinweis: Das Schreiben in Bildspeicherbereiche, die als Systemspeicher genutzt werden (außerhalb der Bildschirmseiten), kann zu Systemfehlern führen.

Beispiele:

1. Mit dem Beispielprogramm wird auf die Textseite 1 ab Position 10 (Zeile), 16 (Spalte) die Zeichenkette "robotron" geschrieben. Die Bildspeicheradresse wird berechnet aus &H400 (Seite 1) + 10*40 (Zeile 10) +16 (Spalte 16) + I.

```
10 ' --- VDOKE mit Zeichenkette ---
20 DATA &H172, &H26F, &H362, &H46F, &H574, &H672, &H76F,
   &H86E
30 SCREEN 0,1 : COLOR 6,0,0:CLS
40 ' ------
50 FOR I=0 TO 7
60     READ ZE
70     VDOKE &H400 + 400+16+I,ZE
80 NEXT
90 PAUSE
```

2. Am oberen Rand des Grafikbildschirmes wird eine gestrichelte dreifarbige Linie gezeichnet.

```
10 ' --- VDOKE mit Grafik ---
20 DATA &H33, &H3300, &H3333
30 SCREEN 2 :COLOR 0,0,0:CLS
40 ' ------
50 FOR I=0 TO 10
60     FOR J=0 TO 2
70         READ ZE
80         VDOKE &H4000 + 3*I+J,ZE
90     NEXTJ
100    RESTORE 10
110 NEXT I
120 PAUSE
```

5.6. Verknüpfung numerischer Ausdrücke mit logischen Operatoren

Die logischen Operatoren **AND**, **OR**, **XOR**, **EQU**, **IMP** und **NOT** sind im Zusammenhang mit der Bildung logischer Ausdrücke erläutert worden.
Mit diesen Operatoren können aber nicht nur die den Wahrheitswerten "wahr" und "falsch" entsprechenden ganzen Zahlen -1 und 0 verknüpft werden. Sie können auf beliebige ganzzahlige Werte zwischen -32768 und 32767 angewendet werden. Zwei ganze Zahlen (einer bei Verwendung von NOT) wird als Ergebnis eine dritte zugeordnet.

Den Ausgangspunkt bildet die erläuterte Binärdarstellung der ganzen Zahlen. Aus der Binärdarstellung der Operanden kann die Binärdarstellung des Resultats ermittelt werden.

Dazu wird aus den Koeffizienten gleicher Zweierpotenzen der Operanden der Koeffizient der entsprechenden Zweierpotenz im Ergebnis bestimmt. Die Regeln für diese Verknüpfungen können aus der folgenden Tabelle entnommen werden. X und Y sind die Werte der Koeffizienten.

Regeln zur bitweisen Verknüpfung ganzer Zahlen

X	NOT X	X	Y	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
0	1	0	0	0	0	0	1	1
1	0	0	1	0	1	1	0	1
		1	0	0	1	1	0	0
		1	1	1	1	0	1	1

Fehlt eine Zweierpotenz in der Binärdarstellung eines Operanden, bedeutet das, daß der entsprechende Koeffizient Null ist.

Beispiel: Das Programm liefert die AND-Verknüpfung zweier ganzer Zahlen.

```
10 DEF FNR$(X$)=SPACE$(16-LEN(X$))+X$
20 INPUT "1. Zahl  =";A%
30 INPUT "2. Zahl  =";B%
40 C%=A% AND B%
50 PRINT
60 PRINT TAB(10);"dezimal";TAB(21);"binär"
70 PRINT USING "A        :  ######    &";A%,FNR(BIN$(A%))
80 PRINT USING "B        :  ######    &";B%,FNR(BIN$(B%))
90 PRINT USING "A AND B  :  ######    &";C%,FNR(BIN$(C%))
```

Die Funktion FNR$ dient zur rechtsbündigen Ausgabe der Binärdarstellung.

Bei Eingabe von -1 und 0 erhalten Sie folgende Bildschirmausschrift:

```
            dezimal  binär
A         :      -1  1111111111111111
B         :       0                 0
A AND B   :       0                 0
```

Verknüpft man die ganzen Zahlen -1 und 0 nach diesen Regeln und betrachtet die Ergebnisse, so ergeben sich die aus Abschnitt 3.5 bekannten Beziehungen für die Bestimmung der Wahrheitswerte zusammengesetzter Aussagen. Damit ist ein Zusammenhang zwischen der Verknüpfung numerischer und logischer Ausdrücke mit Hilfe logischer Operatoren hergestellt.

Die Anwendungsmöglichkeiten solcher Verknüpfungen sind vielfältig. Häufig werden mit ihrer Hilfe z.B. Bitmasken für die PIO- und CTC-Programmierung gesetzt (vgl. Beispiele im Abschnitt 8.3).

6. Speicherung von Programmen und Daten

Im Abschnitt 6 soll es in erster Linie um die Arbeit mit Dateien gehen. Unter einer Datei versteht man die Zusammenfassung von logisch zusammengehörenden Daten (Programmen, Texten, Personalien aller Schüler einer Klasse). Jede solche Datei besitzt einen Dateinamen.

Der Dateiname kennzeichnet die Datei und ermöglicht z.B. das Speichern und Wiederfinden einer Datei auf einem Externspeicher (Kassette, Diskette).

Der Dateiname ist wie folgt aufgebaut:

Diskette:	name[.typ]
	name: maximal 8 Zeichen
	typ: maximal 3 Zeichen
Kassette:	name
	name: maximal 6 Zeichen

Innerhalb des Namens sind alle alphanumerischen und Sonderzeichen zugelassen außer:

. , ; : = ? * [ ] " / \ +

Der Dateiname auf der Diskette kann in einigen Kommandos mehrdeutig gewählt werden, um mit einem Kommando mehrere Dateien ansprechen zu können:

1. An einer oder mehreren Stellen des Dateinamens kann ein Fragezeichen (?) stehen. Das Kommando bezieht sich dann auf alle Dateien, die in den Positionen des Fragezeichens ein (beliebiges) Zeichen haben.

2. Wird ein Stern (*) im Dateinamen verwendet, so kennzeichnet er beliebige Zeichen oder Zeichengruppen an dieser Stelle.

3. Stern * und Fragezeichen ? können in einem Dateinamen gemischt auftreten.

Bespiele hierfür sind bei der Beschreibung der Kommandos FILES und KILL angeführt.

Wenn man Dateien auf Externspeichern ablegen bzw. wiederfinden will, muß man vor dem Dateinamen den Namen des Externspeichers (im weiteren als Gerät bezeichnet), gefolgt von einem Doppelpunkt, angeben. Die Dateibezeichnung hat dann die vollständige Form:

gerät:name.typ

Dabei gelten folgende Abkürzungen für *gerät*:

CAS:		für angeschlossenes Kassettengerät
A:		für ein angeschlossenes Diskettenlaufwerk
B:	)	für weitere angeschlossene
C:	)	Diskettenlaufwerke

Im Kapitel 6.3 wird gezeigt, daß auch noch andere Geräte möglich sind. Das Standardgerät (d.h., wenn man kein Gerät angibt ist bei angeschlossenem Diskettenlaufwerk das Laufwerk A, sonst CAS.

Wird eine Datei auf Kassette aufgezeichnet, so wird grundsätzlich vor die eigentlichen Daten der Datei der Dateiname geschrieben, um die Datei später wieder identifizieren zu können. Will man also nicht immer die gesamte Kassette nach einer bestimmten Datei absuchen, so sollte man sich den Dateinamen und den zugehörigen Zählerstand notieren.

Diese Notwendigkeit besteht bei der Diskette nicht, weil jede Diskette ein Inhaltsverzeichnis enthält, in dem die Namen der gespeicherten Dateien und die Diskettenabschnitte, die jede Datei belegt, eingetragen sind.

Anhand dieses Inhaltsverzeichnisses kann der Computer eine Datei bei vorgegebenem Namen selbständig finden.

Der folgende Abschnitt befaßt sich mit allgemeinen Kommandos für die Disketten- bzw. Verzeichnisarbeit.

6.1. Kommandos zur Dateiverwaltung

CALL FORMAT	Formatieren einer Diskette
RESET	Setzen des Schreib-Lese-Status
COPY	Kopieren einer Datei
FILES	Anzeige des Disketteninhaltsverzeichnisses
LFILES	Druck des Disketteninhaltsverzeichnisses
NAME	Umbenennen einer Datei
KILL	Löschen einer Datei

Diese Kommandos gelten ausschließlich für Diskettendateien.

In diesem Abschnitt werden wichtige und hilfreiche Kommandos vorgestellt, die das Verwalten einer Diskette erleichtern bzw. überhaupt erst ermöglichen.

Auf der Diskette kommt das Konzept der gestreuten Speicherung zur Anwendung, welches besagt, daß jede Datei in Blöcken zu je 2 kBytes gespeichert wird. Diese einzelnen Blöcke der Datei müssen nicht zusammenhängend auf der Diskette stehen (gestreute Speicherung). Aber keine Angst, das Speichern und Wiederfinden einer Datei übernimmt der Computer , und man merkt in der Regel überhaupt nicht, daß der Computer die Datei im Interesse einer optimalen Diskettenausnutzung "zerstückelt" abspeichert.

Wissen sollte man allerdings in diesem Zusammenhang, daß die kleinste für den Computer ansprechbare Einheit der Diskette ein Block von 2 kBytes ist, daß also z.B. eine Datei, die nur aus einem einzigen Zeichen (ein Byte) besteht, trotzdem 2 kBytes Diskettenplatz belegt.

Bevor man eine Diskette zur Speicherung von Programmen oder Daten nutzen kann, muß man sie formatieren (auch initialisieren genannt).

Formatieren einer Diskette

Format: **CALL FORMAT**

Funktion: Aufbringen von Spuren auf der Diskettenoberfläche und Prüfen der Diskette auf Beschreib- und Lesbarkeit.

Hinweise:

1. Die Aufzeichnung der Dateien erfolgt nicht homogen auf der gesamten Diskettenoberfläche, sondern auf insgesamt 160 Spuren, die konzentrisch auf beiden Diskettenseiten angeordnet sind.

2. Das Kommando CALL FORMAT muß für fabrikneue Disketten durchlaufen werden, erst dann ist ein Speichern von Dateien möglich. Wendet man CALL FORMAT auf eine bereits benutzte Diskette an, so wird ihr Inhalt zerstört (gelöscht).

3. Bedienablauf für das Formatieren einer Diskette:

Bildschirmausschrift	Tastatureingabe	Bemerkung
	CALL FORMAT ENTER	
Formatieren in LW (A)/B/C:	A oder ENTER	B oder C nur bei Zusatzlaufwerken möglich

Diskette stecken <ENTER>:	ENTER	
Warnung!!! Gesamte Datei wird gelöscht! J/(N):	J	bei allen anderen Tasten: Abbruch von CALL FORMAT
Formatieren auf Spur...		die Spurnummern 000 bis 159 werden angezeigt
Noch einmal? J/(N):	N oder ENTER	bei J beginnt der Dialog von vorn

4. Jede Spur wird fest in 5 Sektoren zu 1024 Bytes eingestellt, so daß eine Diskettenspeicherkapazität von 780 kBytes plus 4 Systemspuren zu je 5 kBytes zur Verfügung steht.

5. Werden beim Formatieren defekte 2-kBytes-Blöcke festgestellt (z.B. fehlerhafte Magnetschicht), so werden die Nummern der entsprechenden Blöcke am Ende des Formatierens angezeigt und automatisch für eine Benutzung gesperrt. Solche Disketten sind (mit einer entsprechend geringen Speicherkapazität) in vollem Umfang nutzbar. Erkennt der Computer mehr als 8 defekte Blöcke, so erscheint die Ausschrift "zu viele defekte Blöcke", und die Diskette ist unbrauchbar.

Eine formatierte Diskette, die man zu einer beliebigen Zeit während der Arbeit mit RBASIC in ein Diskettenlaufwerk steckt, besitzt automatisch den sogenannten "Nur-Lese-Zustand" , d.h., es kann nur lesend auf sie zugegriffen werden. um diesen "Nur-Lese-Zustand" in einen "Schreib-Lese-Zustand" umzuwandeln, wird das folgende Kommando verwendet.

Setzen des Schreib-Lese-Status

Format: **RESET**

Funktion: Ermöglicht das Aufzeichnen von Programmen oder Daten nach Diskettenwechsel.

Hinweise:

1. Versucht man nach einem Diskettenwechsel Programme oder Daten auf die gewechselte Diskette zu schreiben, ohne vorher RESET zu geben, so erscheint eine Fehlermeldung:
   "disk write protected" (Diskette schreibgeschützt).

2. Erscheint der unter Hinweis 1 angegebene Fehler beim Versuch, ein Programm abzuspeichern (siehe SAVE, Abschnitt 6.2), oder beim Versuch, eine Datei zu löschen (KILL) oder umzubenennen (NAME), so ist das Kommando RESET einzugeben und die gewünschte Operation zu wiederholen.

3. Den unter Hinweis 1 angegebenen Fehler vermeidet man während eines Programmlaufes, indem man nach jedem Diskettenwechsel die Anweisung RESET programmiert. Ansonsten wird das Programm mit dem angegebenen Fehler abgebrochen.

4. Um Disketten anderer Computer verarbeiten zu können, wird eine automatische Formaterkennung der Diskette durchgeführt und die erkannte Diskettenkapazität in der Form

   LW A = 780 k

   angezeigt.

   Eine Übersicht über die möglichen Diskettenformate ist im Anhang 2 der Bedienungsanleitung enthalten.

Beispiel:

```
.
.
100 INPUT "Datendiskette einlegen - ENTER";X
110 RESET
.
.
```

Längere Programme und größere Datenbestände sollten grundsätzlich durch eine Kopie gesichert sein. Wenn es möglich ist, sollte die Kopie auf einer anderen Diskette abgelegt werden, um in Havariefällen und bei Defekten an Disketten nicht den unter Umständen erheblichen Aufwand zur Neuerstellung der Datei betreiben zu müssen.

Nach jeder abgeschlossenen Änderung der Originaldatei sollte eine neue Sicherungskopie angelegt werden.

Wie man Daten kopiert, zeigt das folgende Kommando.

Kopieren einer Datei

Format: **COPY** "[*gerät1*:]*dateiname1*" **TO** "[*gerät2*:]*dateiname2*"

gerät1 - Name des Laufwerkes der Ausgabedatei
dateiname1 - Dateiname der Ausgangsdatei
gerät2 - Name des Laufwerkes für die Kopie
dateiname2 - Dateiname der Kopie

Funktion: Kopieren einer Datei auf die gleiche oder auf eine andere Diskette.

Hinweise:

1. Sind *gerät1* und *gerät2* gleich (z.B. Laufwerk A), so wird die Kopie auf der gleichen Diskette angelegt. Dies ist nur möglich, wenn sich die beiden angegebenen Dateinamen voneinander unterscheiden.

2. Wenn nur ein Laufwerk zur Verfügung steht und auf eine andere Diskette kopiert werden soll, so ist als gerät2 das (nichtvorhandene) Laufwerk B: anzugeben.

   Dabei wird mittels folgendem Bildschirmdialog zum Wechseln der Disketten aufgefordert:

Bildschirmausschrift	Tastatureingabe
	COPY "A:TEST.BAS" TO "B:TEST.KOP"
Diskette B stecken; weiter mit "*"	*
Diskette (A) stecken; weiter mit "*"	*
Diskette (B) stecken; weiter mit "*"	*
.	
.	
.	
Ok	

Die Anzahl der notwendigen Diskettenwechsel hängt von der Länge der zu kopierenden Datei ab.

Bei dieser Vorgehensweise kann eine Datei natürlich auch unter dem gleichen Dateinamen kopiert werden.

Beispiel:
```
COPY "A:TEST.BAS" TO "A:TESTKOP.BAS"
COPY "A:TEST.BAS" TO "B:TEST.BAS"
```

Die folgenden Kommandos beziehen sich auf das Disketteninhaltsverzeichnis. Das Verzeichnis faßt maximal 128 Einträge, auch wenn die zugehörigen Dateien die Diskettenkapazität nicht voll ausnutzen.

Anzeige des Disketteninhaltsverzeichnisses

Formate: **FILES**
FILES "*gerät*:"
FILES "*dateiname*"
FILES "*gerät*:*dateiname*"

Funktion: Anzeige des Disketteninhaltsverzeichnisses (vollständig oder auszugsweise) auf dem Bildschirm.

Beispiele:

1. Anzeige des vollständigen Inhaltsverzeichnisses der Diskette im Laufwerk A:

   ```
   FILES
   ```

   Gleichbedeutend sind die Kommandos:

   ```
   FILES "A:"
   FILES "*.*"
   FILES "A:*.*"
   ```

2. Auf der Diskette im Laufwerk A befinden sich diese Dateien:

   TEST1.BAS
   TEST2.BAS
   TEST2.KOP
   TEST1.DAT
   TEST2.DAT
   TEST10.DAT
   PROG.BAS

   Die folgende Tabelle zeigt einige Varianten des FILES-Kommandos und die Namen der angezeigten Dateien:

Kommando	angezeigte Dateinamen		
FILES "*.BAS"	TEST1.BAS	TEST2.BAS	PROG.BAS
FILES "TEST?.DAT"	TEST1.DAT	TEST2.DAT	
FILES "TEST2.*"	TEST2.BAS	TEST2.KOP	TEST.DAT
FILES "TEST*.DAT"	TEST1.DAT	TEST2.DAT	TEST10.DAT
FILES "TEST*,?A?"	TEST1.BAS	TEST2.BAS	TEST1.DAT
	TEST2.DAT	TEST10.DAT	

Druck des Disketteninhaltsverzeichnisses

Format: **LFILES**
LFILES "*gerät*:"
LFILES "*dateiname*"
LFILES "*gerät*:*dateiname*"

Funktion: Ausgabe des Disketteninhaltsverzeichnisses (vollständig oder auszugsweise) auf einen angeschlossenen Drucker.

Hinweis: Grundlegende Informationen zur Nutzung eines Druckers finden Sie im Abschnitt 7.1 bzw. im Abschnitt 6.3 der Bedienungsanleitung.

Beispiele: siehe FILES

Umbenennen einer Datei

Format: **NAME** "*gerät*:*altname*" **AS** "*neuname*"

altname - bisheriger Dateiname
neuname - neuer Dateiname

Funktion: Umbenennen einer Diskettendatei

Hinweise:
1. Die Datei mit dem Namen *altname* muß auf der Diskette existieren, sonst erscheint ein Fehler "File not found".
2. Es wird nur der Dateiname, nicht aber der Dateiinhalt geändert.

Beispiel: Die Datei TEST2.BAS wird in TEST2.ALT umbenannt:

```
NAME "TEST2.BAS" AS "TEST2.ALT"
```

Löschen einer Datei

Format: **KILL** "[*gerät*:]*dateiname*

Funktion: Löschen einer Datei von der Diskette.

Hinweise:
1. KILL streicht die Datei aus dem Inhaltsverzeichnis und gibt damit den Platz, den die Datei auf der Diskette einnimmt, zum Überschreiben frei.
2. KILL für eine bereits geöffnete Datei (siehe Abschnitt 6.3) führt zu einem Fehler.
3. Durch die Verwendung von ? und * im Dateinamen können mehrere Dateien mit einem KILL-Kommando gelöscht werden. Insbesondere ist aus Zeitgründen für das Löschen aller Dateien einer Diskette das Kommando KILL "*.*" dem Kommando CALL FORMAT vorzuziehen.

Beispiele:
1. Löschen der Datei mit dem Namen PROG.BAS:

```
KILL "PROG.BAS"
```

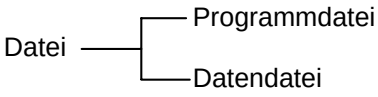
2. Löschen aller Dateien mit dem Dateityp DAT:

```
KILL "*.DAT"
```

6.2. Programmdateien

SAVE	Speichern einer Programmdatei
LOAD	Laden einer Programmdatei
RUN	Laden und Starten einer Programmdatei
MERGE	Mischen zweier Programmdateien
BSAVE	Speichern von Hauptspeicherinhalten (Maschinencode)
BLOAD	Laden von Hauptspeicherinhalten
CSAVE	Speichern einer Programmdatei auf Kassette
SCREEN	Einstellen der Baudrate
CLOAD	Laden einer Programmdatei von Kassette
CLOAD?	Kontrollesen einer Programmdatei von Kassette
MOTOR	Ein-/Ausschalten des Kassettengerätemotors

Der Begriff der Datei gliedert sich in zwei Arten:



In diesem Kapitel wollen wir uns mit dem Speichern von Programmen auf einem Externspeicher (Diskette, Kassette) befassen. So wird es möglich, Programme über längere Zeit außerhalb des Computers aufzuheben und bei Bedarf schnell in diesen zurückzuladen.

Speichern einer Programmdatei

Format: **SAVE**"[*gerät*:] *dateiname*"[,*A*]

gerät - {A|B|C} für Diskette, d.h. A, B oder C angeben
CAS für Kassette
ohne Standardgerät
(bei angeschlossenem Diskettenlaufwerk: A,
ohne Diskettenanschluß: CAS)

Funktion: Aufzeichnen eines RBASIC-Programms auf einem Externspeicher (Diskette oder Kassette).

Hinweise:

1. Wenn ein Programm unter dem angegebenen Namen auf einer Diskette bereits existiert, wird es überschrieben. Bei Nutzung einer Kassette hängt es von der Positionierung des Magnetbandes ab.

2. Wird der A-Zusatz weggelassen, erfolgt eine Aufzeichnung im sogenannten Zwischencode. Bei Angabe des A-Zusatzes (A = ASCII) wird im ASCII-Code abgespeichert. Für Sie unterscheiden sich die beiden Möglichkeiten in erster Linie dadurch, daß eine Datei im ASCII-Code mehr Platz auf dem Externspeicher benötigt. Allerdings ist es für manche Zwecke notwendig, ein Programm im ASCII-Code zu speichern (z.B. für die Anwendung des MERGE-Kommandos). Auf Kassette wird immer im ASCII-Mode aufgezeichnet, auch wenn der A-Zusatz weggelassen wird.

3. Erscheint beim Versuch, ein Programm auf Diskette abzuspeichern eine Fehlermeldung "Disk write protected", so ist das RESET-Kommando (siehe Abschnitt 6.1) einzugeben, und das SAVE-Kommando ist zu wiederholen.

Beispiel:

```
SAVE"A:PROG1.BAS"
SAVE"PROG2",A
SAVE"CAS:PROG3"
```

Laden einer Programmdatei

Format: **LOAD**"[*gerät*:]*dateiname*"[,*R*]

gerät - {A|B|C} für Diskette
CAS für Kassette
ohne Standardgerät
(bei angeschlossenem Diskettenlaufwerk: A,
ohne Diskettenanschluß: CAS)

Funktion: Laden eines mit SAVE abgespeicherten RBASIC-Programms von einem Externspeicher.

Hinweise:
1. *dateiname* ist der Name, mit dem das Programm abgespeichert wurde.
2. Bei Verwendung des R-Zusatzes (R=RUN) wird das Programm nach dem Laden automatisch gestartet.
3. LOAD schließt alle offenen Dateien und löscht alle Variablen und Programmzeilen, die im Speicher noch vorhanden sind, bevor es das gewünschte Programm lädt.
4. Bei Verwendung des R-Zusatzes werden alle offenen Datendateien offengehalten.

Beispiel:
```
LOAD "A:PROG1.BAS"
LOAD "CAS:PROG3"
```

Laden und Starten einer Programmdatei

Format: **RUN** "[*gerät*:]*dateiname*"

gerät - {A|B|C} für Diskette
ohne Diskettenlaufwerk A

Funktion: Laden und sofortiges Starten eines mit SAVE abgespeicherten RBASIC-Programms von einer Diskette.

Hinweise:
1. *dateiname* ist der Name, mit dem das Programm abgespeichert wurde.
2. RUN schließt vor dem Laden alle offenen Dateien und löscht alle Variablen und Programmzeilen, die noch im Speicher vorhanden sind.
3. Das Kommando ist in dieser Form nur für Programme auf Disketten anwendbar.

Beispiel:
```
RUN "PROG1.BAS"
```

Mischen zweier Programmdateien

Format: **MERGE** "[*gerät*:]*dateiname*"

gerät - {A|B|C} für Diskette
CAS für Kassette
ohne Standardgerät
(bei angeschlossenem Diskettenlaufwerk: A,
ohne Diskettenanschluß: CAS)

Funktion: Einfügen bzw. Anfügen eines RBASIC-Programms von einem externen Speicher in/an das im Hauptspeicher befindliche Programm.

Hinweise:
1. Das einzumischende Programm muß im ASCII-Format aufgezeichnet sein.

2. Wenn beim Mischen von Programmen eine Zeilennummer zweimal vorkommt, so enthält das resultierende Gesamtprogramm die Zeile des zuletzt geladenen Programms.

3. MERGE läßt sich für das Anhängen einzeln abgespeicherter (weil in mehreren Hauptprogrammen verwendbarer) Unterprogramme nutzen.

Beispiel: Auf der Diskette im Laufwerk A befinden sich zwei Programme:

1. PROG1.BAS (auf Diskette geschrieben mit dem Kommando SAVE "A:PROG1.BAS"

```
10 ' --- Zinsberechnung ---
20 CLS
30 INPUT "Gesamtbetrag.";B
40 Z=B*3.25/100
50 GOSUB 1000
60 PRINT "Zinsen fuer ";B;"Mark: ";Z;"Mark"
70 END
```

2. PROG2.BAS (auf Diskette geschrieben mit dem Kommando SAVE "A:PROG2.BAS",A)

```
1000 ' --- Unterprogramm Runden auf Pfennige ---
1010 Z=INT(Z*100.0.5)/100
1020 RETURN
```

Durch die Kommandofolge

```
LOAD "A:PROG1.BAS"
MERGE "A:PROG2.BAS"
```

wird das Unterprogramm PROG2.BAS an das Hauptprogramm PROG1.BAS angehängt.

Speichern von RAM-Speicherinhalten

Die beiden folgenden Kommandos sind von allgemeinerer Natur als die Kommandos SAVE und LOAD, die nur zum Speichern und Laden von RBASIC-Programmen dienen. Sie erfordern deshalb auch weitergehende Kenntnisse speziell über die Aufteilung des Arbeitsspeichers. Informationen dazu können Sie den Abschnitten 5.3 bis 5.5, 9 und dem Anhang entnehmen.

Format: **BSAVE** "[*gerät*:]*dateiname*",*anfangsadresse*,*endadresse*{[,*startadresse*]|[,*S*]|[,*Z*]}

gerät - {A|B|C} für Diskette
CAS für Kassette
ohne Standardgerät
(bei angeschlossenem Diskettenlaufwerk: A,
ohne Diskettenanschluß: CAS)

anfangsadresse)
endadresse) ganze Zahl
startadresse)

Funktion: BSAVE dient zum Aufzeichnen der Daten des durch *anfangsadresse* und *endadresse* spezifizierten Hauptspeicher-, Video-RAM- oder Zeichengenerator-RAM-Bereiches auf einem Externspeicher in binärer Form.

Hinweise: 1. Der Parameter *startadresse* wird beim Abspeichern von Maschinencode verwendet und gibt die Adresse an, ab der das Maschinencodeprogramm nach einem BLOAD-Kommando mit R-Zusatz gestartet wird.

2. Fehlt *startadresse*, so ist *anfangsadresse* = *startadresse*.

3. Bei vorhandenem S-Zusatz wird der angegebene Bereich des Video-RAMs in die Datei gespeichert. Der S-Zusatz ist nur für eine Aufzeichnung auf Diskette zugelassen. Bei Kassette wird S als Variable für die Startadresse gewertet.
   Entsprechendes gilt für den Z-Zusatz und den Zeichengenerator-RAM.

4. Folgende Werte sind für *anfangsadresse*, *endadresse* und beim Hauptspeicher für *startadresse* zulässig:

   Hauptspeicher - &H8000 bis &HFFFF
   , dezimal -32767 bis -1 oder 32768 bis 65535
   Video-RAM - 0 bis &HFFFF , dezimal 0 bis 65535
   Zeichengenerator-RAM - 0 bis &HFF , dezimal 0 bis 255

5. Die Parameter *startadresse*, S und Z können nur alternativ angegeben werden.

Beispiele:

1. Aufzeichnen des Hauptspeicherinhaltes von der Adresse &HE000 bis &HE800 auf die Diskette im Laufwerk A:

```
BSAVE "A:PROG.BIN",&HE000,&HE800
```

2. Erzeugen eines Rechteckes auf dem Bildschirm und Abspeichern des zum SCREEN 2 gehörenden Video-RAM-Bereiches auf der Diskette im Laufwerk A:

```
10 SCREEN 2
20 LINE (30,20)-(130,100),,B
30 BSAVE "RECHTECK",&H4000,&H5FFF,S:' Bildspeicherbereich
                                      SCREEN 2, Seite 0
```

Laden von Maschinencode bzw. des Video-RAM-Inhalts

Format: **BLOAD** "[*gerät*:] *dateiname*" {[,*R*]|[,*S*]|[,*Z*]} [,*versatz*]

gerät - {A|B|C} für Diskette
CAS für Kassette
ohne Standardgerät
(bei angeschlossenem Diskettenlaufwerk: A,
ohne Diskettenanschluß: CAS)

versatz - ganze Zahl von -32768 bis 32767 oder von 0 bis 65535

Funktion: BLOAD dient dem Laden eines mit BSAVE gespeicherten Speicherinhalts in den Hauptspeicher bzw. zum Laden des durch das BSAVE-Kommando abgespeicherten Video-RAM-Inhalts.

Hinweise:

1. Das Programm wird zwischen die im entsprechenden BSAVE-Kommando angegebenen Hauptspeicheradressen geladen. Ist ein *versatz* angegeben, so wird dieser vorher zur *anfangs*- und *endadresse* addiert.

2. Wird der R-Zusatz angegeben, so wird das Programm unmittelbar nach dem Laden an der durch das entsprechende BSAVE-Kommando spezifizierten *startadresse* gestartet.

3. Wird der S-Zusatz angegeben, so wird der angegebene Video-RAM-Inhalt geladen (nur Diskette). Der Video-RAM-Inhalt kann auch im "Hintergrund" geladen werden, wenn eine andere visuelle Bildseite eingestellt ist.

4. Wird der Z-Zusatz angegeben, so wird der angegebene Zeichengenerator-RAM-Inhalt geladen (nur Diskette).

Beispiele:

1. Laden (auf Adresse &HE000 bis &HE800) und Starten (ab Adresse &HE000) der Datei PROG.BIN:

```
BLOAD "A:PROG.BIN",R
```

2. Laden des im vorhergehenden BSAVE-Beispiel abgespeicherten Video-RAM-Inhalts von der Diskette im aktuellen Laufwerk und Anzeige des Rechtecks auf dem Bildschirm:

```
10 SCREEN 2:CLS
20 BLOAD "RECHTECK",S
30 PAUSE
```

Im folgenden werden drei Kommandos erläutert, die sich nur auf das Speichern bzw. Laden von Programmen auf bzw. von Kassette beziehen.

Speichern einer Programmdatei auf Magnetbandkassette

Format: **CSAVE** "*dateiname*"[,*baudrate*]

baudrate - Übertragungsrate (Werte 1 oder 2; Standard: einstellbar mit SCREEN)

Funktion: Aufzeichnen eines RBASIC-Programms auf einem Kassettengerät.

Hinweise:
1. Mit CSAVE gespeicherte Programme können mit dem CLOAD-Kommando wieder eingelesen werden. Nach Abarbeitung des CSAVE-Kommandos sollte das CLOAD?-Kommando benutzt werden, um die Richtigkeit der Aufzeichnung zu überprüfen.
2. *baudrate* ist die Übertragungsrate, sie kann mit 1 oder mit 2 angegeben werden.

   baudrate 1 = 1200 Bd
   2 = 2400 Bd

Beispiele:
```
CSAVE "PROG2"
CSAVE "PROG2",2
```

Einstellen der Baudrate für Kassettenausgabe

Format: **SCREEN**,,,,,*baudrate*

baudrate - Übertragungsrate (Werte 1 oder 2; Standard: 1)

Funktion: Verändern des Wertes der Dateiübertragungsrate für Ausgabe auf Magnetbandkassette.

Hinweis:
1. baudrate 1 = 1200 Bd
   baudrate 2 = 2400 Bd
2. Mit dieser *baudrate* wird gearbeitet, wenn in der CSAVE-Anweisung keine *baudrate* angegeben wird.
3. Die 5 Kommas in der SCREEN-Anweisung bewirken, daß alle anderen mit SCREEN einstellbaren Größen unverändert bleiben.

Laden einer Programmdatei von Magnetbandkassette

Format: **CLOAD**["*dateiname*"]

Funktion: Laden eines RBASIC-Programms vom Kassettengerät.

Hinweise:
1. Wenn die angegebene Datei gefunden wurde, erscheint zu Beginn des Ladens die Bildschirmausschrift

   Found: *dateiname*.

   Das Ende des Ladens wird durch "Ok" signalisiert.

2. Wird ein vom angegebenen Parameter *dateiname* verschiedener Dateiname gefunden, so erscheint die Meldung

   ```
   Skip: dateiname
   ```

3. Wenn kein Dateiname angegeben wird, so wird die zuerst gefundene Datei geladen.

4. CLOAD löscht alle im Speicher befindlichen Programme und Daten und schließt alle offenen Dateien.

5. Es wird immer mit der Datenübertragungsrate geladen, mit der das Programm aufgezeichnet wurde. Die Übertragungsrate wird automatisch erkannt.

Beispiele:
```
CLOAD "PROG2"
CLOAD
```

Es ist ratsam, eine mit CSAVE erfolgte Programmaufzeichnung zu überprüfen, um z.B. Aufzeichnungsfehler durch defektes Bandmaterial erkennen zu können. Die Aufzeichnung muß dann gegebenenfalls an einer anderen Bandstelle wiederholt werden.

Überprüfung der Aufzeichnung

Format: **CLOAD?**["*dateiname*"]

Funktion: CLOAD? vergleicht ein Programm auf Magnetbandkassette mit dem im Hauptspeicher befindlichen Programm.

Hinweise:

1. CLOAD? sollte nach jedem CSAVE benutzt werden.

2. Bei auftretenden Nichtübereinstimmungen wird auf dem Bildschirm

   ```
   Verify error
   ```

   angezeigt. Die komplette Übereinstimmung wird durch "Ok" bestätigt.

Beispiele:
```
CLOAD? "PROG2"
CLOAD?
```

Das folgende Kommando wird nur für Kassettengeräte mit Rechnersteuerung benötigt.

Ein-/Ausschalten des Kassettenmotors

Format: **MOTOR** [**OFF**]
MOTOR [**ON**]

Funktion: Schaltet bei Kassettengeräten mit Rechnersteuerung den Motor ein oder aus.

Hinweise:

1. MOTOR OFF schaltet den Motor aus.
   MOTOR ON schaltet den Motor ein.
   MOTOR schaltet den Motor zwischen OFF und ON um.

2. Alle RBASIC-Anweisungen, die sich auf ein Kassettengerät beziehen (siehe Abschnitte 6.2 und 6.4), schalten bei rechnergesteuerten Kassettengeräten den Motor automatisch ein und aus. Das MOTOR-Kommando ist dort also nicht notwendig.

3. Verwenden muß man MOTOR ON z.B., um bei rechnergesteuerten Kassettengeräten manuell schnellen Vorlauf bzw. Rücklauf ausführen zu können.

4. Für Kassettengeräte ohne Rechnersteuerung hat das MOTOR-Kommando keinerlei Bedeutung.

6.3. Eröffnen und Schließen von Dateien

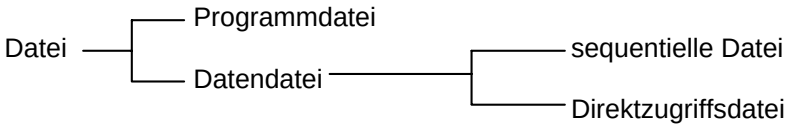
MAXFILES Festlegen der Anzahl der gleichzeitig zu bearbeitenden Dateien
OPEN Eröffnen einer Datei
CLOSE Schließen einer Datei

Datendateien

Außer Programmen möchte man natürlich auch Daten (z.B. Personalien, Telefonverzeichnisse, Termine) außerhalb des Computers speichern und damit über längere Zeit aufheben können. Auch besteht der Wunsch, neben dem Erstellen solcher Datendateien Möglichkeiten für die nachträgliche Änderung, Ergänzung, Streichung oder auch nur Wiederauffindung in die Hand zu bekommen.

Mit den dafür bereitstehenden Anweisungen wollen wir uns in den folgenden Abschnitten befassen.

In RBASIC gibt es zwei verschiedene Arten von Datendateien, deren Eigenschaften an einem kleinen Beispiel erläutert werden sollen.



Gespeichert werden soll ein Telefonverzeichnis, das der Einfachheit halber nur jeweils den Familiennamen und die zugehörige Telefonnummer enthalten soll.

In einer sequentiellen Datei kann das z.B. so aussehen:

A	D	A	M	.	5	2	8	4	0D	0A	B	A	U	M	A	N	N	,	3	8	1	1	0D	0A	

Sequentielle Dateien können stets nur hintereinander, beginnend am Anfang der Datei, geschrieben bzw. gelesen werden. Es können also keine Daten übersprungen werden, um z.B. den direkten Zugriff auf die Telefonnummer des in der Mitte der Datei stehenden Herrn Müller zu haben. Zu diesem Zwecke müßte die Datei von Anfang an gelesen werden, bis der Name Müller gefunden ist.
Eine sequentielle Datei ist vorstellbar als eine Papierrolle, auf der die Daten nacheinander angeordnet sind. Ein Satz ist dabei die Datenmenge, die bei einem Schreibvorgang aufgezeichnet wird, im obigen Beispiel in die Codierungen &H0D&H0A (Bedeutung siehe Abschnitt 6.4 PRINT#-Anweisung) eingeschlossen. Die Länge eines solchen Satzes hängt also von der Länge der Daten ab.

Eine Direktzugriffsdatei können wir uns dagegen als viele einzelne Papierstreifen vorstellen, die alle die gleiche Länge haben. Jeder dieser Papierstreifen enthält einen Satz und ist mit einer Nummer (1, 2, 3 usw.) gekennzeichnet.

A	D	A	M												5	2	8	4
B	A	U	M	A	N	N									3	8	1	1
M	U	E	L	L	E	R									9	4	2	5
								.										
								.										
								.										
W	U	N	D	E	R	L	I	C	H						6	7	3	8

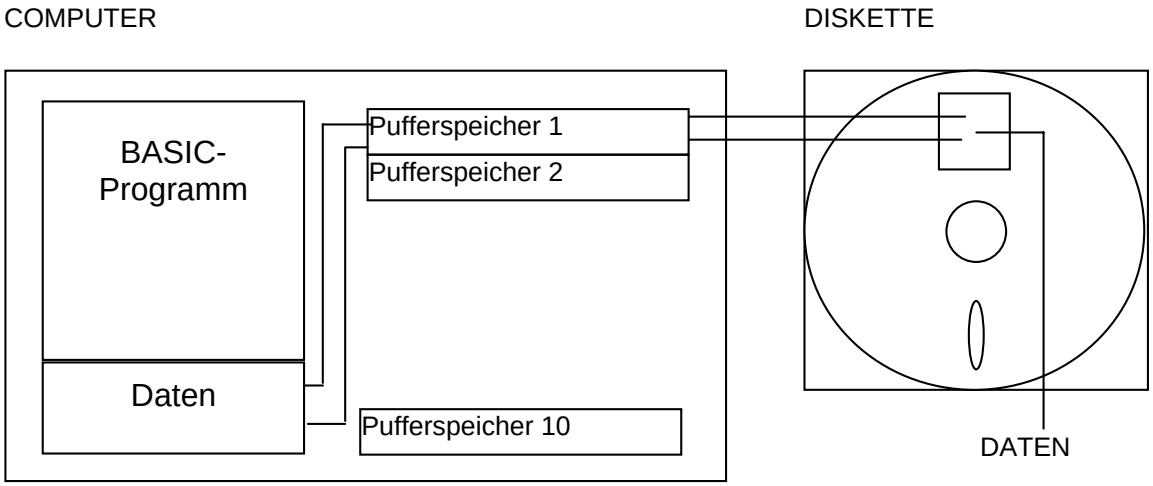
Satz 1
Satz 2
Satz 3
Satz n

In dieser Abbildung haben die Papierstreifen eine Länge von 19 Zeichen, was einer Speicherkapazität von 19 Bytes (je Papierstreifen bzw. Datensatz) entspricht. Auf Grund der konstanten Länge der Länge der Sätze läßt sich bei den Direktzugriffsdateien ein Satz unter Angabe der Satznummer direkt lesen oder schreiben. Die Position (Adresse) des Satzes kann über die Satznummer schnell berechnet werden. Das erleichtert z.B. die Datenänderung gegenüber sequentiellen Dateien ganz erheblich.

In den folgenden Kapiteln wollen wir uns nun die RBASIC-Anweisungen für die Arbeit mit Datendateien näher ansehen.

Eröffnen und Schließen

Wenn Daten von einer Datei gelesen oder in eine Datei geschrieben werden sollen, so muß eine Verbindung zwischen dem Computer und der entsprechenden Datendatei hergestellt werden. Man kann das etwa mit dem Herstellen einer Telefonverbindung über eine Vermittlung vergleichen.



Das Herstellen bzw. Löschen einer solchen Verbindung sowie das Festlegen der Anzahl der gleichzeitig nebeneinander bestehenden Verbindungen sind Inhalt diese Abschnittes.

Festlegen der Anzahl der gleichzeitig zu bearbeitenden Dateien

Format: **MAXFILES** = *anzahl*

anzahl - ganze Zahl von 0 bis 10

Funktion: Festlegen der Anzahl der im Computerspeicher anzulegenden Pufferspeicher und damit der maximalen Anzahl gleichzeitig zu bearbeitender Datendateien.

Hinweise:

1. Wird die Anweisung MAXFILES nicht verwendet, so steht standardmäßig ein Pufferspeicher zur Verfügung.

2. MAXFILES=0 bedeutet, die Freigabe des standardmäßig zur Verfügung stehenden Pufferspeichers und kann bei speicherplatzkritischen Programmen, die nicht mit Dateien arbeiten, verwendet werden.

3. Um nicht unnötig Speicherplatz zu blockieren, sollten nur so viele Pufferspeicher vereinbart werden, wie wirklich benötigt werden. Jeder Pufferspeicher schränkt den Speicherplatz für Programme bzw. Variable um ca. 440 Bytes ein.

4. MAXFILES muß sinnvollerweise am Programmanfang stehen, da mit MAXFILES die Ausführung von CLEAR verbunden ist.

Beispiel: Bei der Abarbeitung eines RBASIC-Programms soll gleichzeitig (d.h. nicht nacheinander) mit 3 Dateien gearbeitet werden:

```
10 MAXFILES=3
```

Eröffnen einer Datei

Format: **OPEN** "[*gerät*:]*dateiname*"[**FOR** *betriebsart*]**AS**[#]*dateinummer*[**LEN**=*satzlänge*]

gerät - {A|B|C} für Diskette, d.h. A, B oder C angeben
CAS für Kassette

CRT für Bildschirm
LPT für Drucker
GRP für Grafikbildschirm
ohne Standard ist bei angeschlossenem Diskettenlaufwerk A, sonst CAS

betriebsart - INPUT sequentielles Lesen
OUTPUT sequentielles Schreiben
APPEND sequentielles (Weiter-)Schreiben am Ende einer Diskettendatei
ohne wird die Klausel [FOR *betriebsart*] weggelassen, so ist Lesen und/oder Schreiben im Direktzugriff spezialisiert.

dateinummer - ganze Zahl von 1 bis 10

satzlänge - Länge eines Satzes bei Direktzugriffsdateien

Funktion: Öffnen einer Datei. Erst nach OPEN ist ein lesender/schreibender Zugriff auf diese Datei möglich.

Hinweise:

1. Sequentielle Dateien können nur entweder zum Lesen oder zum Schreiben geöffnet werden, Direktzugriffsdateien können nach OPEN abwechselnd gelesen und beschrieben werden.
2. Die Dateinummer legt die Nummer des zu verwendenden Pufferspeichers fest.
3. Die Dateinummer wird weiter verwendet, um die Lese-/Schreibanweisungen auf diese Datei zu beziehen.
4. Der Parameter *satzlänge* ist nur für Direktzugriffsdateien anzugeben, die vom Standard 128 Bytes abweichen.
5. OPEN setzt einen Zeiger auf den Dateianfang. Dieser Zeiger wird durch Lese-/Schreibanweisungen in der Datei bewegt und steht jeweils an der Stelle, an der der nächste Lese-/Schreibvorgang beginnt.
6. Die Betriebsart APPEND ist nur für sequentielle Diskettendateien zulässig und wirkt wie die Betriebsart OUTPUT. Beide Betriebsarten unterscheiden sich lediglich in der Zeigerposition nach Ausführung der OPEN-Anweisung. Während bei OUTPUT der Zeiger an den Dateianfang gesetzt wird, wird bei APPEND das Dateiende gesucht und der Zeiger dort positioniert. Dieser Suchvorgang nimmt je nach Länge der Datei eine entsprechende Zeit in Anspruch.
7. Mit OPEN können sowohl neue Dateien angelegt, als auch vorhandene weiterbearbeitet werden.
8. Die Betriebsart Direktzugriff ist nicht für Kassettendateien zulässig.

Beispiele:

1. Eröffnen einer Diskettendatei mit dem Namen TELEFON.DAT zum sequentiellen Schreiben:

```
OPEN "A:TELEFON:DAT" FOR OUTPUT AS #1
```

2. Eröffnen einer Diskettendatei mit dem Namen TELVERZ.DAT für den Direktzugriff (Satzlänge 19 Bytes):

```
OPEN "A:TELVERZ.DAT" AS #1 LEN=19
```

3. Eröffnen einer Diskettendatei, an deren Ende sequentiell weitergeschrieben werden soll:

```
OPEN "TELVERZ.DAT" FOR APPEND AS#1
```

4. Vorbereiten des Schreibens auf den Grafikbildschirm:

```
OPEN "GRP:" AS#1
```

Wie anhand der möglichen Geräte zu erkennen ist, gibt es neben den üblichen Disketten- und Kassettendateien auch Bildschirm-, Drucker- bzw. Grafikbildschirmdateien. Diese sind nur für schreibenden Zugriff zugelassen und ermöglichen, Texte oder Zahlen auf vorher festlegbare Geräte ausgeben zu lassen.

Für die verschiedenen Geräte gibt es Einschränkungen im Aufbau der OPEN-Anweisung sowie für die zulässigen Ein-/Ausgabeanweisungen.

Die zugelassenen Kombinationen sind der folgenden Tabelle zu entnehmen.

Angaben in OPEN				zulässige Ein-/Ausgabeanweisungen	
gerät	*dateiname*	*betriebsart*	*satzlänge*	sequentiell	Direktzugriff
{A|B|C} (Diskette)	1	INPUT OUTPUT APPEND	0	alle aus Kapitel 6.4	
		ohne	2		alle aus Kapitel 6.5
CAS	1	INPUT OUTPUT	0	alle aus Kapitel 6.4 + EOF	0
CRT	2	OUTPUT oder ohne	0	nur PRINT#	0
LPT	2	OUTPUT oder ohne	0	nur PRINT#	0
GRP	2	OUTPUT oder ohne	0	nur PRINT#	0

Es bedeuten: 0 - Angabe bzw. Verwendung ist unzulässig
1 - Angabe bzw. Verwendung ist notwendig
2 - Angabe bzw. Verwendung ist möglich, aber nicht notwendig.

Schließen einer Datei

Format: **CLOSE** {[#]*dateinummer* [,[#]*dateinummer* ...]}
CLOSE

dateinummer - ganze Zahl von 1 bis 10

Funktion: Beendet die Ein-/Ausgabe zu einer Datei.

Hinweise:
1. *dateinummer* ist die Nummer, unter der die Datei eröffnet wurde.
2. CLOSE ohne Argumente schließt alle offenen Dateien.
3. Die Dateinummer und der zugehörige Pufferspeicher werden freigegeben und können beim Öffnen weiterer Dateien wieder verwendet werden. Dadurch kann nacheinander mit beliebig vielen Dateien gearbeitet werden.
4. Die END-Anweisung und das NEW-Kommando schließen immer alle offenen Dateien.

Beispiel: Schließen der Datei, die unter der Nummer 1 eröffnet wurde:

```
CLOSE #1
```

6.4. Sequentielle Dateien

PRINT#	sequentielles Schreiben (Ausgabe)
PRINT#, USING	sequentielles Schreiben mit Formatvorgabe
INPUT#	sequentielles Lesen (Eingabe)
LINE INPUT#	sequentielles Lesen bis Zeilenende (CR)
INPUT$	sequentielles Lesen einer vorgegebenen Zeichenanzahl

Wie man sich eine sequentielle Datei vorstellen kann, wurde bereits im Kapitel 6.3 dargestellt. Nachfolgend werden die Anweisungen behandelt, mit denen man auf solche Dateien schreibend oder lesend zugreifen kann.

Folgende Schritte sind notwendig, um

- eine neue sequentielle Datei anzulegen und mit Daten zu füllen:

  1. Öffnen der Datei mit der Betriebsart INPUT
  2. Schreiben der Daten in die Datei mit PRINT#
  3. Schließen der Datei

- Daten von einer vorhandenen sequentiellen Datei zu lesen:

  1. Öffnen der Datei mit der Betriebsart INPUT
  2. Lesen der Daten aus der Datei mit INPUT#
  3. Schließen der Datei

- eine vorhandene sequentielle Datei an ihrem Ende fortzuschreiben:

  1. Öffnen der Datei mit der Betriebsart APPEND
  2. Schreiben der Daten in die Datei mit PRINT#
  3. Schließen der Datei

Sequentielles Schreiben (Ausgabe)

Format: **PRINT** #*dateinummer*[,*ausgabeliste*]

ausgabeliste - analog PRINT-Anweisung

Funktion: Ausgabe von Daten in eine sequentielle Datei.

Hinweise:

1. *dateinummer* ist die Zahl, mit der die Datei für sequentielles Schreiben eröffnet wurde.

2. Die PRINT#-Anweisung schreibt die Daten so in die Datei, wie sie eine PRINT-Anweisung auf den Bildschirm schreiben würde. Für *ausgabeliste* gelten analoge Regeln wie bei der PRINT-Anweisung.

3. Zum Zwecke des späteren Lesens mit INPUT# ist es in der Regel erforderlich, die Daten in der Datei durch Trennzeichen voneinander zu trennen (siehe Beispiele).

4. Der Zeilenvorschub, den eine PRINT-Anweisung auf dem Bildschirm bewirkt, wird bei der PRINT#-Anweisung als Codierung &H0D&H0A in die Datei geschrieben.

5. Wundern Sie sich bitte nicht, wenn nicht bei jeder PRINT#-Anweisung ein Zugriff auf die Diskette bzw. die Kassette erfolgt. Die Daten werden im Pufferspeicher gesammelt und erst, wenn dieser gefüllt ist bzw. die Datei geschlossen wird, in die Datei geschrieben.

Beispiele:

1. Die Zeichenketten A$, B$ und C$ werden durch verschiedene Anweisungsfolgen in die als sequentielle Ausgabedatei eröffnete Datei BEISP.DAT geschrieben:

```
5 OPEN "A:BEISP.DAT" FOR OUTPUT As #1
10 A$="MEIER":B$="KLAUS":C$="DRESDEN"
50 CLOSE#1
```

Variante 1:

```
20 PRINT#1,A$
30 PRINT#1,B$
40 PRINT#1,C$
```

M	E	I	E	R	0D	0A	K	L	A	U	S	0D	0A	D	R	E	S	D	E	N	0D	0A
| Datensatz |							| Datensatz |						| Datensatz |							

Variante 2:

```
20 PRINT#1,A$,B$
```

M	E	I	E	R													K	L	A	U	S	0D	0A

Variante 3:

```
20 PRINT#1,A$;B$;C$
```

M	E	I	E	R	K	L	A	U	S	D	R	E	S	D	E	N	0D	0A
| Datensatz |																		

Variante 4:

```
20 PRINT#1,A$;",";B$;",";C$
```

M	E	I	E	R	,	K	L	A	U	S	,	D	R	E	S	D	E	N	0D	0A

Variante 5:

```
20 D$=A$+","+B$:C$=" "+C$
30 PRINT#1, D$+C$
```

M	E	I	E	R	,	K	L	A	U	S		D	R	E	S	D	E	N	0D	0A

Variante 6:

```
20 D$=A$+","+B$:C$=" "+C$
30 PRINT#1.CHR$(34);D$;CHR$(34);CHR$(34);C$;CHR$(34)
```

"	M	E	I	E	R	,	K	L	A	U	S	"	"		D	R	E	S	D	E	N	"	0D	0A

2. Auf den Grafikbildschirm ist ab den Koordinaten 1,10 das Wort "Robotron" in den Farben grün auf schwarz zu schreiben:

```
10 SCREEN 2:CLS
20 COLOR 0,0:'Hintergrund schwarz, Palette 0
30 OPEN "GRP:" AS#1
40 PSET(320,0),2:'Schriftfarbe grün
50 PRESET(1,10)
60 PRINT#1,"Robotron"
70 CLOSE#1
80 PAUSE
```

3. Das Wort "Robotron" ist wahlweise auf den Bildschirm oder auf den Drucker auszugeben:

```
10 INPUT "Bildschirm (B) oder Drucker (D)";A$
20 IF A$="B" THEN BA$="CRT:"
30 IF A$="D" THEN BA$="LPT:":ELSE GOTO 10
40 OPEN BA$ AS#1
50 PRINT#1,"Robotron"
60 CLOSE#1
```

Sequentielles Schreiben mit Formatvorgabe

Format: **PRINT**#*dateinummer*,**USING** *format*;*ausgabeliste*

format - analog Anweisung PRINT USING
ausgabeliste - analog Anweisung PRINT USING

Funktion: Ausgabe von Daten in einem vorgegebenen Format in eine sequentielle Datei.

Hinweise:
1. *dateinummer* ist die Zahl, mit der die Datei für sequentielles Schreiben eröffnet wurde.
2. Die PRINT#-USING-Anweisung schreibt die Daten so in die Datei, wie sie eine PRINT-USING-Anweisung auf den Bildschirm schreiben würde.

Sequentielles Lesen (Eingabe)

Format: **INPUT** #*dateinummer*,*variablenliste*

variablenliste - analog INPUT-Anweisung

Funktion: Einlesen von Daten aus einer sequentiellen Datei und Wertzuweisung an Variable der Variablenliste.

Hinweise:
1. *dateinummer* ist die Zahl, mit der die Datei für sequentielles Lesen eröffnet wurde.
2. *variablenliste* enthält die Variablen (auf Reihenfolge und Typ achten!), die die einzelnen Daten der Datei als Wert zugeordnet bekommen sollen.
3. Beim Einlesen gelten folgende Regeln:
   - Bei numerischen Werten werden vorangehende Leerstellen, &H0D- und &H0A-Codierungen ignoriert.
   - Das erste Zeichen, außer den angegebenen, ist der Beginn einer Zahl.
   - Eine Zahl endet an einer Leerstelle, einem Komma, einer &H0D- und &H0A-Codierung.
   - Bei Zeichenkettenwerten werden vorangehende Leerstellen, &H0D- und &H0A-Codierungen ebenfalls ignoriert.
   - Das erste Zeichen, außer den angegebenen, ist der Beginn einer Zeichenkette.
   - Ist dieses erste Zeichen ein Anführungszeichen ("), so besteht die Zeichenkette aus allen Zeichen, die zwischen dem ersten Anführungszeichen und dem nächsten gelesen werden.
   - Ist das erste Zeichen einer Zeichenkette kein Anführungszeichen, so endet die Zeichenkette an einem Komma, einer &H0D-Codierung oder nach 255 Zeichen.

Beispiele: Es werden die Dateien gelesen, die in den Beispielen 1 bis 6 mit PRINT# geschrieben wurden. Vorher wird in Zeile 5 die Datei BEISP.DAT als sequentielle INPUT-Datei eröffnet.

```
5 OPEN "A:BEISP.DAT" FOR INPUT AS #1
10 INPUT#1,A$,B$,C$
```

Liest man die in den PRINT#-Beispielen erzeugten Dateien mit der Anweisung in Zeile 10, so ergeben sich folgende Wertzuweisungen:

Variante	Länge	Wertzuweisungen
1	5	A$="MEIER"
	5	B$="KLAUS"
	7	C$="DRESDEN"
2	21	A$="MEIER KLAUS"
	0	B$=""
	0	C$=""
3	17	A$="MEIERKLAUSDRESDEN"
	0	B$=""
	0	C$=""
4	5	A$="MEIER"
	5	B$="KLAUS"
	7	C$="DRESDEN"
5	5	A$="MEIER"
	13	B$="KLAUS DRESDEN"
	0	C$=""
6	11	A$="MEIER,KLAUS"
	8	B$=" DRESDEN"
	0	C$=""

Sequentielles Lesen bis Zeilenende (CR)

Format: **LINE INPUT** #*dateinummer*, *zeichenkettenvariable*

Funktion: Einlesen einer Zeichenkette aus einer sequentiellen Datei bis zum Zeilenendezeichen (&H0D) oder von 255 Zeichen.

Hinweise: 1. *dateinummer* ist die Zahl, mit der die Datei für sequentielles Lesen eröffnet wurde.

2. Das Einlesen wird nicht durch Anführungszeichen oder Komma beendet, sondern nur durch Zeilenende (&H0D) oder nach 255 Zeichen.

Beispiel: Liest man die durch Beispiel 4 mit PRINT# erzeugte sequentielle Datei BEISP.DAT mittels

```
10 OPEN "A:BEISP.DAT" FOR INPUT AS#1
20 LINE INPUT#1,A$
```

so ergibt sich für A$ die Wertzuweisung

A$="MEIER,KLAUS,DRESDEN".

Sequentielles Lesen einer vorgegebenen Zeichenanzahl

Format: **INPUT$**(*zeichenanzahl*,[#]*dateinummer*)

zeichenanzahl - ganze Zahl von 1 bis 255

Typ: BASIC-Funktion

Funktion: INPUT$ liefert eine Zeichenkette der vorgegebenen Länge, die aus der angegebenen sequentiellen Datei gelesen wird.

Hinweise:

1. *dateinummer* ist die zahl, mit der die Datei für sequentielles Lesen eröffnet wurde.

2. INPUT$ ist keine Wertzuweisung an eine Zeichenkettenvariable, sondern eine Funktion, deren Funktionswert die gelesene Zeichenfolge ist.

3. Ist die vorgegebene Zeichenanzahl in der Datei nicht mehr vorhanden, so erscheint eine Fehlermeldung: "Input past end".

Beispiele:

1. Liest man die durch Beispiel 4 mit PRINT# erzeugte sequentielle Datei BEISP.DAT mittels

```
10 OPEN "A:BEISP.DAT" FOR INPUT AS#1
20 A$=INPUT$(11,#1)
```

so ergibt sich für A$ die Wertzuweisung

A$="MEIER,KLAUS".

2. Das folgende Programm gibt eine sequentielle Kassettendatei DATEN komplett (d.h. mit allen Trennzeichen) in hexadezimaler Form auf den Bildschirm aus:

```
10 OPEN "CAS:DATEN" FOR INPUT AS#1
20 IF EOF(1) THEN 50
30    PRINT HEX$(ASC(INPUT$(1,#1)));
40    GOTO 20
50 PRINT "**** ENDE der Kassettendatei ****"
60 END
```

Die Funktion EOF zeigt das Dateiende an (siehe Abschnitt 6.6).

6.5. Direktzugriffsdateien

FIELD	Aufteilen des Pufferspeichers
MKI$**) **MKS$) **MKD$**)	Konvertieren numerischer Werte in Zeichenketten
CVI) **CVS**) **CVD**)	Konvertieren von Zeichenketten in numerische Werte
LSET	Füllen des Pufferspeichers (linksbündig)
RSET	Füllen des Pufferspeichers (rechtsbündig)
PUT	Schreiben des Pufferspeicherinhaltes in eine Datei
GET	Füllen des Pufferspeichers mit Daten aus einer Datei

Wie man sich eine Direktzugriffsdatei vorzustellen hat, wurde bereits zu Beginn des Abschnitts 6.3 dargestellt. Deshalb sei an dieser Stelle nur noch einmal unterstrichen, daß der Direktzugriff nur für Diskettendateien möglich ist. Wer an deinem Computer also kein Diskettenlaufwerk angeschlossen hat, kann diesen Abschnitt getrost überspringen.

Folgende Schritte sind notwendig, um

- eine Direktzugriffsdatei anzulegen und mit Daten zu füllen:

1. Öffnen der Datei für den Direktzugriff
2. Aufteilen des Pufferspeichers auf die pro Satz auszugebenden Daten mittels FIELD.
3. Füllen des Pufferspeichers mit den auszugebenden Daten mittels LSET oder RSET.
4. Schreiben des Pufferspeicherinhalts als Satz in die Datei mittels PUT.

Die gleiche Vorgehensweise ist anzuwenden, um eine bestehende Direktzugriffsdatei zu ergänzen.

- von einer vorhandenen Direktzugriffsdatei zu lesen:

1. Öffnen der Datei für den Direktzugriff
2. Aufteilen des Pufferspeichers auf die pro Satz auszulesenden Daten mittels FIELD.
3. Lesen eines Satzes der Datei und Schreiben des Satzes in den Pufferspeicher mit GET.
4. Zugriff auf die Daten im Pufferspeicher durch das Programm.

In einem Programm, das sowohl lesend als auch schreibend auf dieselbe Datei zugreift, reicht die Verwendung von nur jeweils einer OPEN- und FIELD-Anweisung in der Regel aus.

Aufteilen des Pufferspeichers

Format: **FIELD**[#]*dateinummer*,*feldbreite* **AS** *zeichenkettenvariable*
[,*feldbreite* **AS** *zeichenkettenvariable*]...

feldbreite - ganze Zahl von 1 bis 255
zeichenkettenvariable - Name eines Übergabebereiches im Pufferspeicher

Funktion: Ordnet den einzelnen Daten eines Satzes einen bestimmten Abschnitt des Pufferspeichers zu.

Hinweise:

1. *dateinummer* ist die Zahl, mit der die Datei für den Direktzugriff eröffnet wurde.
2. *feldbreite* ist die maximale Anzahl der Zeichen, die der zugehörigen Zeichenkettenvariablen später durch LSET bzw. RSET zugewiesen werden kann.
3. Die Summe der Feldbreiten, die in einer FIELD-Anweisung den Zeichenkettenvariablen zugeordnet werden, darf nicht größer sein als die bei OPEN spezifizierte Satzlänge.
4. In den Pufferspeicher (und damit in die Datei) können nur Zeichenkettenwerte geschrieben werden. Numerische Werte müssen vorher mittels der Funktionen MKI$, MKS$ oder MKD$ in Zeichenketten konvertiert werden.
   Dabei gelten folgende Feldbreitenzuordnungen:

numerischer Wert vom Typ	Feldbreite in Bytes	Konvertierfunktion
ganzzahlig (integer)	2	MKI$
einfach genau	4	MKS$
doppelt genau	8	MKD$

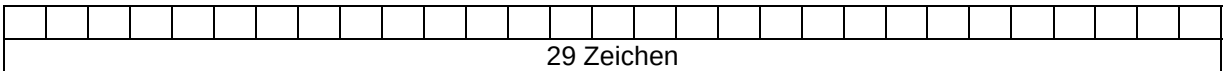
5. De in einer FIELD-Anweisung definierten Zeichenkettenvariablen zeigen auf einen Abschnitt des Pufferspeichers und dürfen deshalb nicht wie andere Zeichenkettenvariable mittels INPUT- oder LET-Anweisungen mit Werten belegt werden. Hierfür sind ausschließlich LSET- (bzw. RSET-) Anweisungen zugelassen.
6. Für die gleiche Datei kann eine beliebige Anzahl von FIELD-Anweisungen ausgeführt werden.

Beispiel: Es soll eine Direktzugriffsdatei aufgebaut werden, die für jeden Kunden einer Bank den Namen, das Datum der Kontoeröffnung und den aktuellen Kontostand enthält. Die Kundennummer soll gleich der Satznummer sein. Die Datei soll unter dem Namen KONTO.DAT auf der Diskette gespeichert werden.

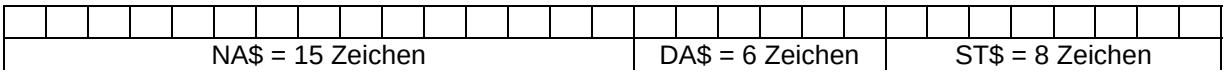
```
10 OPEN "A:KONTO.DAT" AS#1 LEN=29
20 FIELD#1,15 AS NA$,6 AS DA$,8 AS ST$
```

Um auch größere Kontostände exakt führen zu können, wird der Kontostand doppelt genau gespeichert. Man muß sich darüber im Klaren sein, daß Namen mit mehr als 15 Zeichen auf die festgelegte Länge gekürzt werden müssen, daß aber andererseits bei kürzeren Namen der entsprechende Speicherplatz auf der Diskette "verschenkt" wird.

Der Pufferspeicher sieht nach Abarbeitung von Zeile 10 wie folgt aus:



und nach der Aufteilung und Namenszuordnung in Zeile 20 so:



Konvertieren numerischer Werte in Zeichenketten

Format: **MKI$**(*ganzzahliger_ausdruck*)
MKS$(*ausdruck_einfacher_genauigkeit*)
MKD$(*ausdruck_doppelter_genauigkeit*)

Typ: BASIC-Funktion

Funktion: Konvertieren numerischer Werte in Zeichenketten fester Länge zum Zwecke des Schreibens in den Pufferspeicher von Direktzugriffsdateien.

Hinweise:
1. Diese Konvertierungsfunktionen dürfen nicht mit der Funktion STR$ verwechselt werden, bei der die Länge der Ergebniszeichenkette variabel ist.

2. Bei den Funktionen MKI$, MKS$ und MKD$ wird als Ergebnis die interne Darstellung der vorgegebenen Zahl als Zeichenkette geliefert.

3. Die zugehörigen Zeichenkettenlängen sind der Tabelle unter Hinweis 4 der FIELD-Anweisung zu entnehmen.

Beispiel: Siehe LSET-Anweisung und Beispiel am Ende dieses Abschnittes.

Rückkonvertieren von Zeichenketten in numerische Werte

Format: **CVI**(X$)
CVS(X$)
CVD(X$)

X$ - Zeichenkette, die einem numerischen Wert entspricht und die aus dem Pufferspeicher gelesen wurde.

Typ: BASIC-Funktion

Funktion: Rückkonvertieren von Zeichenketten aus dem Pufferspeicher (d.h. aus der Datei) in ihre ursprüngliche numerische Form.

Hinweise:
1. Diese Konvertierungsfunktionen dürfen nicht mit der Funktion VAL verwechselt werden.

2. CVI, CVS und CVD sind die Umkehrfunktionen zu MKI$, MKS$ und MKD$.

Beispiel: siehe Ende dieses Abschnittes.

Füllen des Pufferspeichers

Formate: **LSET** *zeichenkettenvariable=zeichenkettenausdruck*
RSET *zeichenkettenvariable=zeichenkettenausdruck*

zeichenkettenvariable - Name des Übergabebereiches aus der FIELD-Anweisung

Funktion: Die mit der FIELD-Anweisung festgelegten Abschnitte des Pufferspeichers werden mit Programmdaten (gemäß *zeichenkettenausdruck*) gefüllt.

Hinweise:
1. Wenn *zeichenkettenausdruck* kürzer ist, als in FIELD spezifiziert, so trägt LSET links- und RSET rechtsbündig ein. Es wird jeweils mit Leerzeichen aufgefüllt.
   Ist *zeichenkettenausdruck* zu lang, so wird er rechts abgeschnitten.

2. Numerische Werte sind mit MKI$, MKS$ oder MKD$ in Zeichenketten umzuwandeln, ehe sie mittels LSET oder RSET den in der FIELD-Anweisung definierten Zeichenkettenvariablen zugewiesen werden können.

3. LSET und RSET können auch auf "normale" *zeichenkettenvariable* angewendet werden, um eine Zeichenkette in einem vorgegebenen Bereich links- oder rechtsbündig zu justieren (siehe Beispiel 2).

Beispiele:
1. Führen wir das Beispiel aus der FIELD-Anweisung weiter:

```
10 OPEN "A:KONTO.DAT" AS#1 LEN=29
20 FIELD#1,15 AS NA$,6 AS DA$,8 AS ST$
30 N$=MEIER":D$="871122"
40 S#=5348.55
50 LSET NA$=N$
60 LSET DA$=D$
70 LSET ST$=MKD$(S#)
```

   Der Pufferspeicher sieht nach Abarbeitung der Programmzeilen 10 bis 70 so aus:

M	E	I	E	R										8	7	1	1	2	2	X	X	X	X	X	X	X	X
NA$														DA$						ST$							

   Die 8 Kreuze in ST$ sollen andeuten, daß in diesem Teil des Pufferspeichers etwas für uns Unlesbares steht. Aber keine Angst, mittels S#=CVD(ST$) können wir uns jederzeit unsere 5348.55 auf die Variable S# zurückholen.

2. Das Wort "robotron" soll auf einer Zeile des Bildschirms rechtsbündig zur Spalte 20 ausgegeben werden:

```
10 A$="robotron"
20 B$=SPACE$(20)
30 RSET B$=A$
40 PRINT B$
```

Schreiben des Pufferspeicherinhalts in eine Datei

Format: **PUT**[#]*dateinummer* [,*satznummer*]

dateinummer - Nummer, mit der die Datei in der OPEN-Anweisung eröffnet wurde
satznummer - Nummer (Stellung) des Satzes in der Datei (ganze Zahl von 1 bis 32767)

Funktion: Schreiben des Pufferspeicherinhalts an eine vorgegebene Stelle einer Direktzugriffsdatei.

Hinweise:
1. *dateinummer* ist die Zahl, mit der die Datei für den Direktzugriff eröffnet wurde.

2. Fehlt die Angabe von *satznummer*, so wird die nächste zur Verfügung stehende Satznummer (nach dem letzten GET bzw. PUT) verwendet.

3. Nicht bei jedem PUT muß ein Diskettenzugriff erfolgen, da die Sätze in der Regel in einem weiteren Puffer, einem sogenannten Ausgabepuffer, gesammelt werden und erst, wenn dieser gefüllt ist (bzw. bei CLOSE oder END), in die Datei geschrieben werden.

Beispiel: Der unter Beispiel 1 bei der LSET-Anweisung gefüllte Pufferspeicher soll als Satz 1 in die Datei geschrieben werden.

```
80 PUT#1,1
90 CLOSE
```

Füllen des Pufferspeichers mit Daten aus einer Direktzugriffsdatei

Format: **GET**[#]*dateinummer*[,*satznummer*]

satznummer - Nummer (Stellung) des Satzes in der Diskettendatei (ganze Zahl von 1 bis 32767)

Funktion: Lesen eines Satzes aus einer Direktzugriffsdatei in den Pufferspeicher.

Hinweise: 1. *dateinummer* ist die Zahl, mit der die Datei für den Direktzugriff eröffnet wurde.

2. Fehlt die Angabe von *satznummer*, so wird die nächste zur Verfügung stehende Satznummer (nach dem letzten GET bzw. PUT) verwendet.

Beispiele: 1. Der erste Satz der bereits aufgebauten Datei KONTO.DAT ist einzulesen und anzuzeigen:

```
10 OPEN "A:KONTO.DAT" AS#1 LEN=29
20 FIELD#1,15 AS NA$,6 AS DA$,8 AS ST$
30 GET#1,1
40 S#=MKi(ST$)
50 PRINT NA$:PRINT DA$:PRINT S#
60 CLOSE

RUN

MEIER
871122
5348.55
Ok
#
```

2. Es ist ein Programm zu erarbeiten, das für die Datei KONTO.DAT (siehe FIELD-Anweisung) folgende Funktionen realisiert:

   - Neue Kunden anfügen
   - Kontostand aktualisieren
   - Alle Kunden mit negativem Kontostand ausdrucken
   - Programmende

   Der erste Satz der Datei hat einen von anderen Sätzen verschiedenen Aufbau. In ihm wird die höchste benutzte Satznummer gespeichert.

```
10 ' ------------------------------------
20 ' ------ Verwaltung von Konten ---------
30 ' ---    Stand:       24.12.1988     ---
40 ' ---    Bearbeiter: Weihnachtsmann ---
50 ' ------------------------------------
60 OPEN "A:KONTO.DAT" AS#1 LEN 29
```

```
70 FIELD#1,2 AS LE$,27 AS L$
80 FIELD#1,15 AS NA$,6 AS DA$,8 AS ST$
90 A$="N" :INPUT "neue Datei J/(N) ";A$
100 IF A$<>"J" THEN 150
110 ' --- Datei anlegen ---
120   LSET LE$=MKI$(1)
130   LSET L$=SPACE$(27)
140   PUT#1,1
150 GET#1,1
160 LS%=CVI(LE$)
200 ' --- Menue aufbauen ---
210 CLS
220 PRINT "Verwaltung von Konten"
230 PRINT:PRINT "Funktionsauswahl:":PRINT
240 PRINT "1 - anzeigen"
250 PRINT "2 - anfügen"
260 PRINT "3 - aktualisieren"
270 PRINT "4 - negative Konten drucken"
280 PRINT "5 - Programmende"
290 PRINT:INPUT "Funktionskenzahl:";F
300 IF F<1 OR F>5 THEN 210
310 CLS:ON F GOSUB 410,600,700,800
320 LSET LE$=MKI$(LS%)
330 LSET L$=SPACE$(27)
340 PUT#1,1
350 IF F<>5 THEN 210            :' Rücksprung neues Menue
360 CLOSE: PRINT"Programmende" :' Datei schließen
370 END
400 ' ------------------------------------------
410 ' --- Datei anzeigen ---
420 IF LS%<2 THEN PRINT"Datei leer":GOTO 490
430    FOR SA%=2 TO LS%
440        GET#1,SA%
450        PRINT:PRINT "Kundennummer:";SA%-1
460        PRINT "Name:";NA$
470        PRINT "Betrag:";CVD(ST$);"Mark"
480    NEXT
490 PRINT:PRINTTAB(39);">ENTER<"
500 IF INKEY$<>CHR$(13) THEN 500
510 RETURN
600 ' --- Satz anfügen ---
610 LS%=LS%+1
620 PRINT Kundennummer:";LS%-1
630 INPUT "Name,Datum,Betrag";N$,D$,B#
640 LSET NA$=N$
650 LSET DA$=D$
660 LSET ST$=MKD$(B#)
670 PUT#1,LS%
680 RETURN
700 ' --- Satz aktualisieren ---
710 INPUT "Kundennummer:";SA%:SA%=SA%+1
720 IF SA%>LS% THEN PRINT "nicht vorhanden":GOTO 710
730 GET#1,SA%:B#=CVD(ST$)
740 PRINT "Name:";NA$
750 PRINT "alter Betrag:";B#
760 INPUT "neuer Betrag:";B#
770 LSET ST$=MKD$(B#)
780 PUT#1,SA%
790 RETURN
800 ' --- Datei drucken --- Drucker muß bereit sein! ---
810 IF LS%<2 THEN PRINT "Datei leer":GOTO 890
820    FOR SA%=2 TO LS%
830        GET#1,SA%
840        B#=CVD(ST$)
850        IF B#>=0 THEN 880
860           LPRINT:LPRINT "Kundennummer:";SA%
870           LPRINT "Name:";NA$;"  Schulden:";-B#;"Mark"
880    NEXT SA%
890 RETURN
```

6.6. Dateifunktionen

EOF	Dateiendefunktion
LOC	Zeigerpositionsfunktion
LOF	Dateigrößenfunktion
VARPTR#	Adresse des File-Control-Blockes

Von den nachfolgend beschriebenen Funktionen ist die EOF-Funktion die wichtigste. Die restlichen Funktionen werden sicher nur in Spezialfällen genutzt.

Dateiendefunktion

Format: **EOF**(*dateinummer*)

dateinummer - Nummer mit der die Datei in der OPEN-Anweisung eröffnet wurde

Typ: BASIC-Funktion

Funktion: Liefert als Funktionswert -1, wenn eine sequentielle Datei vollständig gelesen wurde, sonst 0.

Hinweise:

1. EOF ist nur für sequentielles Lesen (Betriebsart INPUT in der OPEN-Anweisung) zugelassen. Bei anderen Betriebsarten erscheint ein Fehler "Bad file mode".

2. EOF wird nach dem Lesen des letzten in der Datei vorhandenen Satzes gesetzt. Versucht man einen nicht mehr vorhandenen Satz zu lesen, erscheint ein Fehler "Input past end".

Beispiel: Vollständiges Lesen einer sequentiellen Datei:

```
10 OPEN "A:BEISP.DAT" FOR INPUT AS #1
20 IF EOF(1) THEN GOTO 40
30 INPUT #1,A$:PRINT A$:GOTO 20
40 PRINT "**** DATEIENDE ****":CLOSE:END
```

Zeigerpositionsfunktion

Format: **LOC**(*dateinummer*)

Typ: BASIC-Funktion

Funktion: Liefert die Nummer des zuletzt gelesenen oder geschriebenen Satzes.

Hinweise:

1. Sequentielle Dateien haben standardmäßig eine Satzlänge von 128 Bytes. LOC liefert hier also die Nummer des zuletzt gelesenen oder geschriebenen 128-Byte-Satzes. Bei Direktzugriffsdateien wird die Satzlänge durch die OPEN-Anweisung festgelegt.

2. Unmittelbar nach dem Eröffnen einer Datei liefert LOC den Wert 0.

Dateigrößenfunktion

Format: **LOF**(*dateinummer*)

Typ: BASIC-Funktion

Funktion: Liefert die Nummer des letzten 128-Byte-Satzes im aktuellen Extent.

Hinweise:

1. Der aktuelle Extent ist der 16-kBytes-Bereich der Datei, auf den der letzte Lese- oder Schreibzugriff erfolgte.

2. Ist eine Datei nicht länger als 16 kBytes, dann liefert LOF die Nummer des letzten 128-Byte-Satzes der Datei.

3. Auch bei Direktzugriffsdateien handelt es sich um die Nummer des letzten 128-Byte-Satzes im aktuellen Extent, unabhängig von der in der OPEN-Anweisung angegebenen Satzlänge.

Adresse des RBASIC-FCBs

Format: **VARPTR**(*#dateinummer*)

Typ: BASIC-Funktion

Funktion: Liefert die Adresse des RBASIC-FCBs (FCB - File Control Block).

Hinweis: Der FCB ist der Datenbereich im RBASIC-Arbeitsspeicher, über den der Dateizugriff organisiert wird. Manipulationen an diesem Datenbereich können zur Zerstörung der Datei führen!

Die drei zuletzt aufgeführten Funktionen sollten Sie nur bei genauer Kenntnis der Datei- bzw. FCB-Strukturen auf der Diskette benutzen. Diese Strukturen sind nicht Gegenstand dieser Beschreibung.

7. Die Arbeit mit der Standardperipherie

7.1. Ausgaben auf einen Drucker

SCREEN	Einstellen von Druckertyp und V.24-Interface-Parameter
LLIST	Listen eines Programms auf einen Drucker
LPRINT	Ausgabe von Daten auf einen Drucker
LPRINT USING	Formatierte Ausgabe von Daten auf einen Drucker
LPOS	Bestimmen der aktuellen Druckposition
PRINT#	Ausgabe von Daten auf einen Drucker mittels Dateiausgabeanweisung
PRINT#, USING	Formatierte Ausgabe von Daten auf einen Drucker mittels Dateiausgabeanweisung
CALL BICOP	Ausdrucken des aktuellen Bildschirminhaltes

Durch die Anweisungen LLIST, LPRINT und LPOS wird auf einfache Weise die Ausgabe von Programmen und Daten auf einen angeschlossenen Drucker unterstützt.

Die Anweisungen PRINT# und PRINT#...USING dienen der Ausgabe von Daten. Ihre Handhabung ist etwas anspruchsvoller, dafür bieten sie aber auch Vorteile.
Bei Ausführung aller dieser Anweisungen erfolgt die Ausgabe nur auf dem Drucker. Der Bildschirminhalt wird nicht verändert.

Durch Nutzung der Drucksteuerzeichen können weitere Ausgabe- und Gestaltungsmöglichkeiten der Drucker (Formate, Schriftarten, Zeichensätze usw.) ausgenutzt werden. In den jeweiligen Druckerhandbüchern sind die Möglichkeiten dieser speziellen Drucksteuerung erläutert.

Vorausgesetzt wird jedoch, daß ein entsprechend geeigneter Drucker gemäß Bedienungsanleitung, Abschnitt 6.3, an den Computer angeschlossen worden ist.

Achtung!
Wenn der Drucker nicht bereit ist (Drucker nicht angeschlossen, nicht eingeschaltet, defekt oder im OFF-LINE-Zustand oder Papier verbraucht), wird der Computer blockiert. Der Druckvorgang muß mit CTRL + STOP abgebrochen werden. Es erscheint die Fehlerausschrift:

```
Device I/O error
```

Die gleiche Ausschrift erscheint, wenn der laufende Druckvorgang mit CTRL + STOP abgebrochen wird.

Einstellen von Druckertyp und V.24-Interface-Parameter

Format: **SCREEN**,,,,,,[*druckertyp*][,*datenformat*]

druckertyp - Kennzahl für Druckertyp (ganzzahliger Wert von 0 bis 31; Standard: 0)

datenformat - Kennbyte für V.24-Interface-Parameter

Funktion: Softwaremäßige Anpassung des Computers an den Drucker (soweit das möglich ist).

Hinweise:

1. Die Kennzahl *druckertyp* für Ihren Drucker entnehmen Sie der Bedienungsanleitung, Anhang 2.
   Standardmäßig ist *druckertyp* = 0, d.h., ein Drucker mit IBM-Befehlssatz ist eingestellt (z.B. EPSON FX 1000).

2. Das Kennbyte *datenformat* ist in der Bedienungsanleitung, Abschnitt 6.3 erläutert.
   Standardmäßig ist *datenformat*=&HB1, d.h., die V.24-Interface-Parameter lauten:
   9600 Baud, 8 Datenbits, ohne Paritätsbit, 1 Stoppbit.

3. Die 6 Kommas in der SCREEN-Anweisung vor *druckertyp* sichern, daß alle anderen mit SCREEN einstellbaren Parameter unverändert bleiben. Die Kommas dürfen nicht entfallen.

4. Das Programm DRUEIN.BAS auf der Systemdiskette erleichtert Ihnen die Anpassung des Computers an Ihren Drucker. Mit dem Programm DRUTE.BAS können Sie anhand von Probedrucken das Ergebnis überprüfen.
   Beide Programme gehören zur Ergänzungssoftware des RBASIC und werden gesondert vertrieben.

Ausdrucken von Programmen

Formate: **LLIST** [*anfzeile*][-*endzeile*]
LLIST *anfzeile* -
LLIST.

anfzeile - Nummer der Programmzeile, ab der gedruckt werden soll
endzeile - Nummer der letzten Programmzeile, die gedruckt werden soll.

Funktion: Ausdrucken ("Listen") eines im Arbeitsspeicher befindlichen BASIC-Programms auf einen standardmäßig angeschlossenen Drucker. Das Programm kann vollständig oder teilweise gedruckt werden. Es wird am Bildschirm nicht angezeigt.

Hinweise:
1. Bezüglich der Wahl von *anfzeile* und *endzeile* gelten die gleichen Aussagen wie bei LIST.

2. Die Ausgabe des Programms kann durch CTRL + STOP abgebrochen werden. Jedoch wird der Pufferspeicher des Druckers dann noch geleert, d.h., der Drucker druckt dessen Inhalt noch aus.

Beispiele: Ausdrucken eines vollständigen Programms:

```
LLIST
```

Ausdrucken von Programmteilen

```
LLIST 250                :' nur Programmzeile 250
LLIST 110-               :' ab Zeile 110 bis Ende
LLIST -550               :' von Anfang bis Zeile 550
```

Ausgabe von Daten auf einen Drucker

Format 1: **LPRINT** [*ausgabeliste*]

Format 2: **LPRINT USING** *format*;*ausgabeliste*

ausgabeliste - enthält auszudruckende Konstanten, Variable oder Ausdrücke

format - beschreibt das Ausgabeformat, in dem die Ausdrücke von *ausgabeliste* gedruckt werden

Funktion: Ausdrucken von numerischen oder Zeichenkettendaten aus einem RBASIC-Programm.

Hinweise:
1. Für die Parameter *ausgabeliste* und *format* gelten die gleichen Regeln wie bei PRINT bzw. PRINT USING.

2. Wird im Format 1 *ausgabeliste* weggelassen, so wird eine Leerzeile ausgegeben.

Beispiele: Durch das folgende Programm werden eine Zeichenkette und 3 formatierte Zahlen auf den Drucker ausgegeben.

```
10 Z$="Robotron-Meßelektronik"
20 A=113:B=-7.1234:C=15.676
30 LPRINT Z$
40 LPRINT:LPRINT                        :' --- 2 Zeilen Vorschub
   ---
50 LPRINT USING"#####.##";A;B;C

RUN
```

Auf dem Drucker wird folgendes ausgegeben:

```
Robotron-Meßelektronik


 113.00    -7.12   15.68
```

Abfrage des Auagabepuffers

Format: **LPOS**(X)

X - Blindargument, Wert hat keinen Einfluß auf die Funktion

Typ: BASIC-Funktion

Funktion: LPOS gibt die Position (Spaltennummer) für das nächste Druckzeichen im Ausgabepuffer (aktuelle Druckzeile) an.

Hinweise:
1. Die Zählung der Druckspalten (Positionen) beginnt bei 0.
2. Die durch LPOS angegebene Position stimmt in der Regel mit der Ausgabeposition des nächsten Zeichens am Drucker überein. Das Ausdrucken erfolgt aber zeitverzögert.

Beispiel: Im folgenden Programm wird LPOS auf unterschiedliche Art ausgewertet:

```
10 LPRINT "Drucken mit RBASIC";
20 PRINT LPOS(0)
30 LPRINT " am A 5105"
40 X=LPOS(0)
.
.
```

In Zeile 20 wird die Position des nächsten zu druckenden Zeichens am Bildschirm angezeigt; in Zeile 40 wird sie der Variablen X zugeordnet.

Ausgabe von Daten auf einen Drucker mittels Dateiausgabeanweisung

Format 1: **PRINT***#dateinummer*,[*ausgabliste*]

Format 2: **PRINT***#dateinummer*,**USING** *format*;*ausgabliste*

Funktion: Ausgabe von numerischen oder Zeichenkettendaten aus einem RBASIC-Programm an die Datei *#dateinummer*, die vorher für den Drucker geöffnet werden muß.

Hinweise:
1. Vor der ersten Druckanweisung müssen im Programm die Anweisungen

```
MAXFILES = maximale_dateianzahl
OPEN "LPT:" AS #dateinummer
```

stehen. Nach der letzten Druckanweisung ist

```
CLOSE #dateinummer
```

erforderlich. Weitere Einzelheiten dazu siehe Abschnitt 6.3.

2. Der Vorteil der Druckausgabe über Dateiarbeit liegt darin, daß mit dem gleichen Programm nur durch Änderung der OPEN-Anweisung die Daten auch an den Alpha- oder Grafikbildschirm, die Kassette oder Diskette ausgegeben werden können!

3. Um den Papierverbrauch in Grenzen zu halten, sollte man daher zunächst einen "Probedruck" auf dem Bildschirm vorsehen, entscheiden, ob das Ergebnis "druckreif" ist, und nach eventuellen Korrekturen nur die endgültige Fassung drucken.

Beispiel: Das folgende Programm soll die Funktion SIN(X) in einem wählbaren Intervall tabellarisch ausdrucken. Die Tabellenausgabe ist als Unterprogramm (Zeilen 300...400) gestaltet, welches zweimal aufgerufen wird: zur Probeanzeige auf dem Bildschirm (Zeilen 130...150) und zum Drucken (Zeilen 220...250). Dazwischen erfolgt in den Zeilen 160...200 der Test, ob gedruckt werden soll. Entspricht die Anzeige nicht den Vorstellungen, so können Intervallgrenzen und Schrittweite so lange am Bildschirm variiert werden, bis die gewünschte Tabelle erreicht wird. Besonders günstig ist die Möglichkeit, den Gerätenamen in der OPEN-Anweisung (Zeile 310) als Zeichenkettenvariable (DV$) zu schreiben.

```
10 ' ---- Wertetabelle einer Funktion ----
20 ' ---  Auswahl des Intervalls ---
30 MAXFILES=1
40 PRINT CHR$(12);"Wertetabelle von SIN(X)"
50 PRINT "========================="
60 PRINT
70 XA=0: XE=6.283: DX=(XE-XA)/16
80 INPUT "Anfangswert   von X: ";XA
90 INPUT "Endwert       von X: ";XE
100 IF XA>=XE THEN PRINT "*** XA>=XE ***" :GOTO 60
110 ' ---
120 INPUT "Schrittweite von X: ";DX
130 ' --- Probeanzeige auf dem Bildschirm ---
140 DV$="CRT"
150 GOSUB 300
160 ' --- Entscheidung, ob drucken ---
170 PRINT
180 JN$="N"
190 INPUT "Tabelle drucken? J/(N): ";JN$
200 IF (JN$<>"J") AND (JN$<>"j") THEN 20   :' Neubeginn
210 ' ---
220 ' --- Richtige Tabelle drucken ---
230 DV$="LPT:"
240 GOSUB 300
250 END                                    :' Programmende
260 ' ------------------------------------------------------
300 ' --- Unterprogramm Tabellenausgabe ---
310 OPEN DV$ AS #1                         :' DV$=Gerät
320 PRINT #1, "Tabelle SIN(X) im Intervall";XA;"bis";XE
330 PRINT #1,                              :' Leerzeile
340 PRINT #1, "        X    SIN(X)"
350 FOR X=XA TO XE STEP DX
360    PRINT #1, USING "###.####";X;       :' Komma beachten
370    PRINT #1, USING "#####.####";SIN(X):' Komma beachten
380 NEXT
390 CLOSE #1
400 RETURN
```

Mit diesem Programm erhält man folgenden Ausdruck:

```
Tabelle SIN(X) im Intervall 0 bis 6.283

X             SIN(X)
0.0000         0.0000
0.3927         0.3827
0.7854         0.7071
1.1781         0.9239
1.5707         1.0000
1.9634         0.9239
2.3561         0.7071
2.7488         0.3828
3.1415         0.0001
3.5341        -0.3825
3.9268        -0.7070
4.3195        -0.9238
4.7122        -1.0000
5.1049        -0.9240
5.4975        -0.7073
5.8902        -0.3829
6.2829        -0.0003
```

Spezielle Steuerzeichen

Format: **LPRINT** *steuerzeichenfolge*

steuerzeichenfolge - spezielle Zeichenfolge, die verschiedene Parameter des angeschlossenen Druckers beeinflußt

Funktion: Änderung der Grundeinstellungen des Druckers, z.B. Druckformat, Schriftart, Zeichensatz u.a.

Hinweise:

1. Auswahl und Wirkung der Steuerzeichen hängen natürlich vom konkreten Drucker (Druckertyp und Befehlssatz!) ab und sind im Druckerhandbuch nachzuschlagen.

2. Zur Ausgabe von Steuerzeichenfolgen ist der Druckertyp in der SCREEN-Anweisung auf Steuerzeichenausgabe und anschließend wieder zurückzuschalten (s. Bedienungsanleitung, Anhang 2). Während der Umschaltung auf Steuerzeichenausgabe wird der Wert der Funktion LPOS nicht verändert.

3. Weitere Beispiele für den Gebrauch von Steuerzeichenfolgen finden Sie im Programm DRUEIN.BAS.

Beispiel: Einstellen des Zeichensatzes und des Papierformats des Druckers:

Hier soll gezeigt werden, wie Grundeinstellungen (Zeichensatz u.a.) über Steuerzeichenfolgen vom Computer aus erfolgen können. Dabei werden gleichzeitig weitere Einstellungen am Drucker vorgenommen. Es ist zu sichern, daß der Druckkopf vor dem Absenden der Steuerzeichenfolge am Seitenanfang steht. Während der Ausgabe der Steuerzeichen ist die Kennzahl für den Druckertyp in der SCREEN-Anweisung umzuschalten. Um Zeilenschaltungen zu vermeiden, werden alle Steuerzeichenfolgen mit "," abgeschlossen.

Einstellen von Befehlssatz, Papierformat A4 u. Schriftart entsprechend Bedienunganleitung, Anhang 2:

- für IBM-Drucker (EPSON FX 1000):

```
10 SCREEN,,,,,,16+0                 :' Druckertyp 0, Steuerzeichen
20 LPRINT CHR$(27);"6";             :' Zeichensatz: IBM-Tabelle 2
30 LPRINT CHR$(27);"3";CHR$(36);    :' Zeilenabstand:
                                       36/216 = 1/6 Zoll
40 LPRINT CHR$(27);"C";CHR$(72);    :' Seitenlänge: 72 Zeilen
50 LPRINT CHR$(27);"N";CHR$(4);     :' Unterer Rand: 4 Zeilen
60 LPRINT CHR$(27);":";             :' Schriftart: Elite
                                       (12 Zeichen/Zoll)
70 LPRINT CHR$(27);"X";CHR$(12);CHR$(90);  :' Linker Rand nach
                                       dem 12., rechter Rand nach
                                       dem 90. Zeichen
80 LPRINT CHR$(27);"9";             :' Papierendefühler ein
90 SCREEN ,,,,,,0                   :' Druckertyp 0, Druckzeichen
```

- für EPSON-Drucker (EPSON LX 86):

```
10 SCREEN ,,,,,, 16+9               :' Druckertyp 9, Steuerzeichen
15 LPRINT CHR$(27);"@";             :' Drucker initialisieren
20 LPRINT CHR$(27);"t";CHR$(1);
21 LPRINT CHR$(27);"6";             :' Zeichensatz: EPSON-Tab. 4
22 LPRINT CHR$(27);"R";CHR$(0);     :' US-ASCII
30 LPRINT CHR$(27);"A";CHR$(12);    :' Zeilenabstand:
                                       12/72 = 1/6 Zoll
40 LPRINT CHR$(27);"C";CHR$(72);    :' Seitenlänge: 72 Zeilen
50 LPRINT CHR$(27);"N";CHR$(4);     :' Unterer Rand: 4 Zeilen
60 LPRINT CHR$(27);"!";CHR$(1)      :' Schriftart: Elite
                                       (12 Zeichen/Zoll)
70 LPRINT CHR$(27);"l";CHR$(12);    :' Linker Rand nach
                                       dem 12. Zeichen
71 LPRINT CHR$(27);"Q";CHR$(90);    :' Rechter Rand nach
                                       dem 90. Zeichen
80 LPRINT CHR$(27);"9";             :' Papierendefühler ein
90 SCREEN,,,,,,9                    :' Druckertyp 9, Druckzeichen
```

- für ISO-Drucker (robotron K6311):

```
10 SCREEN,,,,,,16+15                :' Druckertyp 15, Steuerzeichen
20 LPRINT CHR$(27);"R";CHR$(0)      :' Zeichensatz: US-ASCII
40 LPRINT CHR$(27);"[144)";         :' Seitenlänge:
                                       144/2 = 72 Zeilen
50 LPRINT CHR$(27);"[136z";         :' Unterer Rand: 4 Zeilen
60 LPRINT CHR$(27);"[2";CHR$(32);"K";  :' Schriftart:
                                          12,5 Zeichen/Zoll
61 LPRINT CHR$(27);"[0m";
90 SCREEN,,,,,,15                   :' Druckertyp 15, Druckzeichen
```

Beispiel: Einschalten des NLQ-Modus

- für Drucker mit IBM-Befehlssatz (EPSON FX 1000):

```
10 SCREEN,,,,,,16+0                 :' Druckertyp 0, Steuerzeichen
20 LPRINT CHR$(27);"I";CHR$(2)      :' Steuerfolge für NLQ
30 SCREEN,,,,,,0                    :' Druckertyp 0, Druckzeichen
```

- für Drucker mit EPSON-Befehlssatz (EPSON LX 86):

```
10 SCREEN,,,,,,16+9                 :' Druckertyp 9, Steuerzeichen
20 LPRINT CHR$(27);"x";CHR$(1)      :' Steuerfolge für NLQ
30 SCREEN,,,,,,9                    :' Druckertyp 9, Druckzeichen
```

Einmaliger Bildschirmausdruck

Format: **CALL BICOP**

Funktion: Einmaliger Ausdruck des sichtbaren Bildschirminhaltes

Hinweise:

1. Bildschirmausdruck ist nicht mit allen Druckern möglich (siehe Bedienungsanleitung, Anhang 2) und nur mit 8 Datenbits als Interfaceparameter (siehe Bedienungsanleitung, Abschnitt 6.3).

2. Ein Bildschirmausdruck ist von allen SCREENs möglich und berücksichtigt auch vom Nutzer eventuell vorgenommene Änderungen am Zeichensatz des Computers.

3. Es ist zu beachten, daß der Drucker nicht farbig drucken kann. Es erscheint dasselbe Wild wie an einem Schwarz-Weiß-Monitor (der keine Graustufen kennt).

4. Wenn der Bildschirmausdruck mit CTRL + STOP abgebrochen wird, kann es sein, daß der Drucker nicht mehr reagiert. Dieser Zustand kann durch Aus- und Einschalten des Druckers bzw. durch Senden von "Blindzeichen" (z.B. CHR$(0)) an den Drucker behoben werden. Dabei wird spätestens nach 256 Blindzeichen der Drucker wieder aktiv.

Beispiel: Die Ausgabe eines Grafikbildes in SCREEN 2, Seite 3 aus dem Kommandomodus erfolgt z.B. durch:

```
SCREEN 2,3:CALL BICOP ENTER
```

7.2. Nutzung von Steuerhebeln

STICK	Abfragen der Kursortasten bzw. der Stellung eines Steuerhebels
STRIG	Abfragen der Aktionstasten am Steuerhebel bzw. der Leertaste
STRIG(n)ON	Unterbrechungsanmeldung erlauben
STRIG(n)OFF	Unterbrechungsanmeldung verbieten
STRIG(n)STOP	Unterbrechungsanmeldung erlauben, aber nicht ausführen
ON STRIG GOSUB	Festlegung der gültigen Unterbrechungsbehandlungsprogramme

An der rechten Seite des Computergrundgerätes können bis zu zwei Steuerhebel mit wahlweise 1 oder 2 Aktionstasten angeschlossen werden (vgl. Bedienungsanleitung, Abschnitt 3.1). Verfügen Sie über keinen Steuerhebel, können Sie auch die entsprechenden Kursortasten betätigen. Die Abfrage der Stellung dieser Hebel bzw. der Kursortasten erfolgt von RBASIC aus durch die Anweisung STICK. Durch STRIG können die Aktionstasten und außerdem die Leertaste der Tastatur des Computers abgefragt werden. In Abhängigkeit vom Ergebnis dieser Abfragen können Sie dann den weiteren Ablauf Ihres RBASIC-Programms steuern und verschiedene Reaktionen bewirken. Beachten Sie bitte, daß die Steuerhebel so gehalten werden müssen, daß die Aktionstaste in Richtung Bildschirm zeigt.

Format: **STICK** (*steuerhebel*)

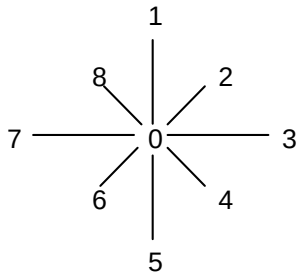
steuerhebel - numerischer Ausdruck (ganzzahliger Wert: 0, 1 oder 2)

Typ: BASIC-Funktion

Funktion: In Abhängigkeit vom Wert des numerischen Ausdrucks *steuerhebel* werden ein Steuerhebel bzw. die Kursortasten der Tastatur abgefragt.

steuerhebel = 0 Abfrage der Kursortasten
steuerhebel = 1 Abfrage Spielhebel in Buchse 1
steuerhebel = 2 Abfrage Spielhebel in Buchse 2

Der gelieferte Funktionswert (Wert zwischen 0 und 8) gibt die aktuelle Stellung des Steuerhebels an. Wird der Steuerhebel nicht betätigt bzw. keine Taste gedrückt, so wird der Wert 0 geliefert.



Hinweise:

1. Durch STICK kann nur die Stellung des Steuerhebels abgefragt werden, nicht die Betätigung der Aktionstasten.

2. Die gelieferten Funktionswerte bei Betätigung der Kursortasten entsprechen den zuordenbaren Richtungen des Steuerhebels.

Wert von STICK	Stellung des Steuerhebels	äquivalente Kursortasten
0	Ruhestellung	keine Taste gedrückt
1	↑	↑
2	↗	↑ + →
3	→	→
4	↘	→ + ↓
5	↓	↓
6	↙	↓ + ←
7	←	←
8	↖	↑ + ←

Beispiel: Das Beispielprogramm zeigt die Codenummern der eingestellten Richtungen bzw. der gedrückten Kursortasten an. Es muß durch CTRL + STOP abgebrochen werden. (Falls in Zeile 10 eine '1' eingegeben wird, sollte der entsprechende Spielhebel angeschlossen sein.)

```
10 INPUT "Abfrage Tastatur (0) oder Steuerhebel (1)";S
20 J=STICK(S)                          :' Richtung abfragen
30 IF J<>0 THEN PRINT "RICHTUNG:";J  :' Reaktion auslösen
40 GOTO 20
```

Durch geringfügige Veränderungen des Programms können Sie weitere Reaktionen hervorrufen, zum Beispiel

```
35 IF J=5 THEN BEEP
40 IF J<>8 THEN 20 ELSE END
```

Format: **STRIG** (*taste*)

taste - numerischer Ausdruck (ganzzahliger Wert: 0 bis 4)

Typ: BASIC-Funktion

Funktion: In Abhängigkeit vom Wert *taste* wird abgefragt, ob die Leertaste der Tastatur bzw. eine Aktionstaste eines Steuerhebels gedrückt wurde

taste	Abfrage
0	Leertaste
1	Aktionstaste 1 am Steuerhebel 1
2	Aktionstaste 1 am Steuerhebel 2
3	Aktionstaste 2 am Steuerhebel 1
4	Aktionstaste 2 am Steuerhebel 2

Als Funktionswert wird geliefert:
0, falls abgefragte Taste nicht gedrückt
1, falls abgefragte Taste gedrückt

Hinweise:

1. Die STRIG-Funktion wird in dieser Weise als selbständige Funktion zur Abfrage der angegebenen Tasten verwendet. Der Funktionswert kann mittels einer Variablen gespeichert und später ausgewertet werden.

2. Die Abfrage der Aktionstaste 2 ist nur sinnvoll, wenn Sie einen Spielhebel mit 2 Aktionstasten haben.

Beispiel: In Zeile 10 des folgenden Programms ist die Eingabe einer '1' nur dann sinnvoll, wenn Spielhebel 1 angeschlossen ist. Durch Verwendung der STRIG-Funktion besteht die Möglichkeit, das Programm durch Drücken der Leertaste bzw. der Aktionstaste 1 vom Spielhebel 1 zu beenden.

```
10 INPUT "Abfrage Leertaste (0) oder Aktionstaste (1)";S
20 J=STICK(S)                              :' Richtung abfragen
30 IF J<>0 THEN PRINT "RICHTUNG:";J
40 A=STRIG(0) : B=STRIG(1)                 :' Tasten abfragen
50 IF A=-1 OR B=-1 THEN 100 ELSE 20        :' )Abbruch nach
100 PRINT "Ende der Steuerhebelabfrage":' )Tastendruck
110 BEEP
```

Weiterhin steht die Anweisungsgruppe STRIG(n)ON, STRIG(n)OFF, STRIG(n)STOP und ON STRIG GOSUB zur Verfügung, die völlig analog zu den bereits im Abschnitt 5.2 beschriebenen Anweisungsgruppen ON KEY, ON STOP, ... arbeitet. Die Anweisungen werden deshalb hier nur noch kurz erläutert.

Formate: **ON STRIG GOSUB** *zeilennummer*[,*zeilennummer*...]
ON STRIG GOSUB [*zeilennummer*],...,*zeilennummer*

zeilennummer - gültige Zeilennummer im aktuellen Programm, mit der ein GOSUB-Unterprogramm beginnt.

Funktion: Die Anweisung legt die jeweils erste Zeilennummer fest, zu der verzweigt wird, wenn eine Unterbrechung durch eine Aktionstaste bzw. Leertaste erfolgt.
Es können bis zu 5 Zeilennummern angegeben werden, die in ihrer Reihenfolge den Tastennummern 0 bis 4 entsprechend der STRIG-Funktion zugeordnet sind.

Hinweise:

1. Die Verzweigung durch eine Aktionstastenunterbrechung wird erst dann ausgeführt, wenn die Unterbrechungsbehandlung durch eine STRIG(n)ON-Anweisung erlaubt wurde.

2. Die Zeilennummern in der Anweisung werden durch Kommas getrennt. Wird eine Zeilennummer weggelassen, so kann durch die zugeordnete Taste keine Unterbrechung ausgeführt werden.

3. Die Rückkehr (Beendigung) aus einem Unterbrechungsbehandlungsproramm erfolgt durch eine RETURN-Anweisung. Das Programm wird dann an der "Unterbrechungs-stelle" bzw. an der durch RETURN festgelegten Zeile (bei RETURN *zlnr*) fortgesetzt.

4. Während ein Unterbrechungsbehandlungsprogramm abgearbeitet wird, wird für die jeweilige Aktionstaste der STRIG(n)STOP-Status eingestellt. Nach RETURN wird wieder der STRIG(n)ON-Zustand eingenommen.

Format: **STRIG**(*taste*)**ON**

taste - numerischer Ausdruck (ganzzahliger Wert: 0 bis 4)

Funktion: Erlauben der Anmeldung und Bearbeitung von Unterbrechungen durch Betätigung der Aktionstasten bzw. der Leertaste. Für *taste* gelten die gleichen Zuordnungen wie bei der STRIG-Funktion.

Hinweis: Vor der STRIG(n)ON-Anweisung muß durch eine "ON STRIG GOSUB"-Anweisung den Unterbrechungstasten ein entsprechendes Unterprogramm zugeordnet werden.

Format: **STRIG**(*taste*)**OFF**

taste - numerischer Ausdruck (ganzzahliger Wert: 0 bis 4)

Funktion: Verbieten der Anmeldung einer Unterbrechung durch Betätigung der Aktionstasten bzw. der Leertaste. Für *taste* gelten die gleichen Zuordnungen wie bei der STRIG-Funktion.

Hinweis: Nach Abarbeitung der STRIG(n)OFF-Anweisung wird nach Betätigung der bezeichneten Taste keine Unterbrechung erkannt bzw. ausgeführt.

Format: **STRIG**(*taste*)**STOP**

taste - numerischer Ausdruck (ganzzahliger Wert: 0 bis 4)

Funktion: Merken einer Unterbrechung, die durch Betätigung einer Aktionstaste bzw. der Leertaste angefordert wird, bis wieder eine STRIG(n)ON-Anweisung ausgeführt wird. Die "gemerkte" Unterbrechung wird sofort ausgeführt, wenn die STRIG(n)ON-Anweisung ausgeführt ist. Für *taste* gelten die gleichen Zuordnungen wie bei der STRIG-Funktion.

Hinweise:
1. Das nach der Unterbrechung anzuspringende Unterbrechungsbehandlungsprogramm muß durch eine "ON STRIG GOSUB"-Anweisung festgelegt sein.

2. Falls vor Abarbeitung der STRIG(n)STOP-Anweisung keine STRIG(n)ON-Anweisung ausgeführt wurde, wird keine Unterbrechung "gemerkt".

Beispiel: Im Beispiel kann das laufende Programm durch die Leertaste oder durch die Aktionstaste 1 des Steuerhebels 2 unterbrochen werden. Nach kurzer Pause wird de Zählschleife fortgesetzt:

```
10 ON STRIG GOSUB 100,,300:' Unterbrechungsprogramm festlegen
20  STRIG(0)ON:STRIG(2)ON    :'  Unterbrechung  durch  0  u.  2
erlauben
30 FOR I=1 TO 1000
40     PRINT I;
50 NEXT I
60 END                      :' Programmende
90 ' --- Unterbrechungsbehandlungsprogramme ---
100 BEEP:PRINT "Leertaste":PAUSE 50:RETURN
200 ' --- Steuerhebel 1 wirkt nicht ---
300 BEEP:PRINT "Aktionstaste 2":PAUSE 50:RETURN
```

8. Weitere Ein- und Ausgabemöglichkeiten

Für das Verständnis der Abschnitte 8 und 9 wird die Kenntnis einiger Begriffe und Zusammenhänge vorausgesetzt, die im Rahmen dieses Programmierhandbuches nicht erklärt werden. Wir empfehlen dem interessierten Leser, sich die erforderlichen Kenntnisse durch Studium der einschlägigen Literatur anzueignen (z.B.: Kieser/Meder : Mikroprozessortechnik, Verlag Technik Berlin).

8.1. Ein- und Ausgabe über Nutzerschnittstelle

INP Eingabe über externen Kanal
OUT Ausgabe über externen Kanal

Mit Hilfe der Anweisungen INP und OUT können Sie auf die externen Kanäle des Computers zugreifen. Sie gestatten sowohl das Einlesen von Meßwerten, die an den externen Kanälen anliegen, als auch die Ausgabe von Signalen über diese Kanäle an die Standard- oder Prozeßperipherie.

Format: **INP**(*portadresse*)

portadresse - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 255 bzw. 00H bis FFH

Typ: BASIC-Funktion

Funktion: Es wird ein Byte vom E/A-Port mit der angegebenen Portadresse gelesen.

Format: **OUT** *portadresse*,*byte*

portadresse - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 255 bzw. 00H bis FFH

byte - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 255 bzw. 00H bis FFH

Funktion: Die Anweisung realisiert die Ausgabe von einem Byte an die angegebene Portadresse.

Beispiele: Beispiele zur Anwendung der OUT-Anweisung oder der INP-Funktion finden Sie in den folgenden Abschnitten in komplexen Programmen.

Die Adressen für die Eingabe der Steuer- und Datenwörter des Computergrundgerätes sind (vgl. Anhang D):

		dez.	hex.
PIO-Schaltkreis Port A	DWadr.	144	90H
	STWadr.	146	92H
Port B	DWadr.	145	91H
	STWadr.	147	93H
CTC-Schaltkreis Kanal 1	ST/DWadr.	129	81H

Bitte beachten Sie:

1. PIO- und CTC-Schaltkreis des Computergrundgerätes sind bei angeschlossener Diskettenspeichereinheit des Computers nur eingeschränkt nutzbar, da sie nicht über entsprechende Stecker zugänglich sind.
2. PIO- und CTC-Schaltkreise bedürfen vor ihrer Benutzung einer INITIALISIERUNG.

3. Im Computergrundgerät stehen Ihnen zur Verfügung:

   - der Port A des PIO-Schaltkreises mit den Bits A0...A7, ARDY und ASTB,
     der Port B des PIO-Schaltkreises mit dem Bit B0

   - der Kanal 1 des CTC-Schaltkreises

4. Der Port B des PIO-Schaltkreises im Computergrundgerät wird vom System initialisiert und benutzt. Der Anwender darf diesen Port nur lesen (Adresse 91H), aber nicht initialisieren (Ausgaben auf Adresse 93H sind verboten). Mit Bit 0 dieses Ports kann ein an die Steckverbindung des Computer-grundgerätes angelegtes Eingangssignal abgefragt werden.

5. Die Kanäle 0, 2 und 3 des CTC-Schaltkreises werden vom System initialisiert und sind durch die interne Uhr belegt.

Für die PIO- und CTC-Schaltkreise in der Diskettenspeichereinheit gelten die Kanaladressen (vgl. Anhang D des Programmierhandbuches):

		dez.	hex.
PIO-Schaltkreis Port A	DWadr.	96	60H
	STWadr.	98	62H
Port B	DWadr.	97	61H
	STWadr.	99	63H
CTC-Schaltkreis Kanal 1	ST/DWadr.	81	51H
Kanal 2	ST/DWadr.	82	52H

Bitte beachten Sie:

1. Der PIO-Port A liegt am Steckverbinder "E/A-Erweiterung-1", der Port B am Steckverbinder "E/A-Erweiterung-2" der Diskettenspeichereinheit des Computers an. Die Beispiele dieses Handbuches beziehen sich alle auf diese PIO. Die zugehörigen Steckerbelegungen sind in der Bedienungsanleitung, Anhang 2, beschrieben. Gleiches gilt für die CTC-Kanäle der Dikettenspeichereinheit.

2. Der Kanal 1 des CTCs ist für externe Ein- und Ausgänge auf den Steckverbinder "E/A-Erweiterung-1" und der Kanal 2 auf den Steckverbinder "E/A-Erweiterung-2" geschaltet.

3. Alle weiteren Angaben und Programmierbeispiele beziehen sich auf die leicht zugänglichen Schaltkreise der Diskettenspeichereinheit.

8.2. Initialisierung der PIO- und CTC-Schaltkreise

Steuer- und Maskenwortbildung für den PIO-Schaltkreis ohne Interruptprogrammierung

	Struktur des Steuerwortes Bit									
MODE	7	6	5	4	3	2	1	0	hexadez.	dez.Steuerwort
Byte-Ausgabe	0	0	0	0	1	1	1	1	0FH	15
Byte-Eingabe	0	1	0	0	1	1	1	1	4FH	79
Byte-Ein/Aus	1	0	0	0	1	1	1	1	8FH	143
Bit-Ein/Ausg. Maskenwort folgt	1	1	0	0	1	1	1	1	CFH	207

Beispiel: Sechs Signale eines Analog-Digital-Umsetzers sind über einen Schaltkreis einzulesen und zwei Signale für Stellglieder auszugeben.

```
100 ' --- UP-PIO-Initialisierung ---
110 OUT 98,207                    :' Mode 3 einstellen
120 OUT 98, 63                    :' A0-A5 Eingabe = 1
                                     A6,A7 Ausgabe = 0
```

Bei maschinennaher Programmierung hat sich die hexadezimale und binäre Zahlendarstellung bewährt. Portadressen, Steuer-, Masken- und Datenwörter werden deshalb nachfolgend hexadezimal mit dem Präfix &H oder binär mit dem Präfix &B eingetragen. Das Beispiel nimmt dann folgende Form an:

```
100 ' --- UP-PIO-Initialisierung ---
110 OUT &H62,&HCF                    :' Mode 3 einstellen
120 OUT &H62,&B00111111              :' A0-A5 Eingabe = 1
```

Steuer- und Maskenwortbildung für den PIO-Schaltkreis mit Interruptprogrammierung

Die Initialisierung des PIO-Schaltkreises zur Ausführung eines Interruptzyklusses in Form eines Maschinencode- oder eines RBASIC-Programms erfordert die Einfügung der Steuer- und Maskenwörter für die Interruptlogik des PIO-Schaltkreises. Außerdem ist der niederwertige Teil des Interruptvektors einzutragen. Die Initialisierung erfolgt auf der Grundlage der nachstehenden vollständigen Tabelle:

Nr.	Bezeichnung	Struktur des Vektors/Wortes Bit 7	6	5	4	3	2	1	0	Hinweis
1	Interrupt-Vektor	V7	V6	V5	V4	V3	V2	V1	V0	D0=0
2	Modewahl	M1	M0	X	X	1	1	1	1	s.Tabelle f. Modewahl
3	Bitwahl	E/A	E/A	E/A	E/A	E/A	E/A	E/A	E/A	nur bei Mode 3 1 = Eingabe 0 = Ausgabe
4	Interrupt-Steuerwort	E/D	A/O	H/L	MA	0	1	1	1	E=1:INT erlaubt D=0:INT verboten A=1:AND O=0:OR H=1:high L=low MA=1:Maskenw. folgt MA=0:ohne Maskenwort
5	INT-Maske	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0 = regenerierbar 1 = nicht regenerierbar
6	INT ein/aus	E/D	X	X	X	0	0	1	1	E=1:INT erlaubt D=0:INT verboten

Die Auswahl des Vektors und der Steuer- und Maskenwörter erfolgt entsprechend der konkreten Aufgabenstellung.

Beispiel: Bei normaler Arbeitsweise des PIO-Schaltkreises wird zusätzlich, für den Fall, daß Bit 0 und Bit 1 gleichzeitig anliegen, eine Interruptroutine ausgelöst.

```
100 ' --- UP-PIO-Initialisierung mit INTERRUPT ---
110 POKE &HFDA0,&H..:POKE &HFDA1,&H..:' Adr.f.mc-Progr.eintragen
120 OUT &H62,&HA0                    :' INT-Vektor
130 OUT &H62,&HCF                    :' PIO-Mode 3
140 OUT &H62,&B00000011              :' Bit-E/A-Maske
```

```
150 OUT &H62,&HD7                    :' INT-Steuerwort
160 OUT &H62,&B11111100              :' INT-Maske
```

Das Programm zur PIO-Initialisierung mit Interrupt beginnt zweckmäßigerweise mit dem Eintrag der Maschinencodeprogrammadresse in die dafür vorgesehene Speicherzelle im Arbeitsspeicher. Unabhängig davon, ob aus diesem Programm ein RBASIC-Programm aufgerufen oder nur dieses Programm selbst verwendet wird, ist spätestens vor jedem Return aus dem Interrupt (RETI) der Interrupt wieder zuzulassen (EI).

Steuer- und Maskenwortbildung für den CTC-Schaltkreis ohne und mit Interrupt

Die Initialisierung des CTC-Schaltkreises ähnelt der des PIO-Schaltkreises. Durch die Einbindung eines Startsignals in die Grundprogrammierung wird gegebenenfalls eine zyklischer Wiederaufruf erforderlich.
Die CTC-Programmierung erfolgt auf der Basis nachfolgender Tabelle.

Nr.	Bezeichnung	Struktur des Vektors/Wortes Bit 7	6	5	4	3	2	1	0	Hinweis
1	Interrupt-Vektor	V7	V6	V5	V4	V3	X	X	0	
							0	0		= Kanal 0
							0	1		= Kanal 1
							1	0		= Kanal 2
							1	1		= Kanal 3
2	Modewahl	E/D	MOD	Vt	Fl	TS	TC	RS	1	E/D =1:INT erlaubt =0:INT verboten MOD =1:Zählermode =0:Zeitgebermode Vt =1:Vorteiler 256 =0:Vorteiler 16 Fl =1:positive Flanke =0:negative Flanke TS =1:externer Start =0:Zeitkonst.-Start TC =1:Zeitkonst folgt =0:keine Zeitkonst. RS =1:Kanal rücksetzen =0:NOP
3	Zeitkonstante	TC7	TC6	TC5	TC4	TC3	TC2	TC1	TC0	0 entspricht 256

Beispiel: Einfache CTC-Schaltkreisprogrammierung ohne Interruptsteuerung zur fortlaufenden Ausgabe der Zählregisterwerte.

```
10 CLS                          :' CTC-Modewahl
20 OUT &H51,&H27                :' Zeitkonstante 200
30 OUT &H51,&B11001000          :' abfragen und anzeigen
40 PRINT INP(&H51)
50 GOTO 40
```

Infolge der zu großen Geschwindigkeit, mit der das CTC-Register zählt, und der für diesen Anwendungsfall viel zu langsamen Bildschirmdarstellung können jeweils maximal 12 der 200 Werte dargestellt werden.

8.3. Anwendungsbeispiele für die Ein- und Ausgabe mittels PIO- und CTC-Schaltkreis

Beispiele:

1. Auslösung einer angezeigten Zeitschleife in einem RBASIC-Programm durch eine PIO-Interruptsteuerung bei Möglichkeit des Modewechsels. Die Nutzung und Einbindung von Maschinencodeprogrammen in RBASIC (Zeilen 100...200) wird später im Abschnitt 9 erläutert.

```
100 '--- mc-Programm mit Einladevorgang ---
110 DATA &HF5            :'PUSH AF        ;Reg. AF retten
120 DATA %HAF            :'XOR  A         ;Reg. A löschen
130 DATA &H32,&H78,&HD7 :'LD   (55160),A ;A in Speicherzelle
140 DATA &HF1            :'POP  AF         ;Reg. A rücksetzen
150 DATA &HFB            :'EI              ;INT erlaubt
160 DATA &HED,&H4D       :'RETI            ;RETURN aus INT
170 CLEAR 200,54999
180 FOR I=1 TO 9
190     READ X: POKE 54999+I,X
200 NEXT I
210 '--- Steuerworteingabe und INT-Anzeige ---
220 INPUT "Steuerworte 1,2,3:";S1,S2,S3
230 PRINT "INTERRUPT wird erwartet":GOSUB 250
240 IF PEEK(55160)<>0 THEN 240 :ELSE GOSUB 330 :GOTO 210
245 '--- ENDE PROGRAMMSCHLEIFE ---------------
250 '--- PIO-Initialisierung ---
260 POKE &HFDA0,&HD8:POKE &HFDA1,&HD6   ;'Adr.mc-Programm
270 OUT &H62,&HA0                       ;'INT-Vektor f.Adr.
280 OUT &H62,&HCF                       ;'PIO-Mode 3
290 OUT &H62, S1                        ;'Steuerwort 1
300 OUT &H62, S2                        ;'Steuerwort 2
310 OUT &H62, S3                        ;'Steuerwort 3
320 RETURN
330 '--- Interruptroutine ---
340 CLS: TIME=0
350 LOCATE 0,0,0:PRINT "Interruptroutine laeuft!";
360 PRINT TIME/50;"Sekunden"
370 IF TIME>500 THEN POKE 55160,1
380 CLS
390 RETURN
```

Eingabebeispiel: 2,151,253 für Steuerworte 1,2,3
Impuls für Bit A2 führt zum Interrupt

2. Messung einer Drehzahl in der Einheit Umdrehung/Sekunde mit einem Signalgeber für jeweils eine Umdrehung.

```
100 OUT &H51,&H7F              :'Mode einstellen
110 OUT &H51,&B01100100        :'Zeitkonstante
120 PAUSE 50
130 X=INP(&H51)                :'CTC-Registerabfrage
140 X=100-X
150 PRINT "Umdrehung/s";X
160 GOTO 100
```

Die Abrage des CTC-Registers erfolgt im Sekundenabstand, erzeugt durch die Zeitschleife mit PAUSE 50.

3. Sensorgesteuerte Ampelanlage mit PIO-Schaltkreis ohne Interrupt.

```
10 ' --- Mode 3 einstellen ---
20 ' --- B7,B6 Eingabe, B5-B0 Ausgabe ---
30 OUT &H62,&HCF
40 OUT &H62,&B11000000
100 ' --- Bildschirm einrichten ---
110 CLS:PRINT "Hauptstr. Nebenstr.":WINDOW 2,20,0,20
200 ' --- Zeitschleife für Ampel ---
210 TIME=0:OUT &H60,&B00010010:PRINT "gelb        gelb"
```

```
220 GOSUb 300:IF X=128 THEN 250:ELSE IF TIME<250 THEN 220
230 TIME=0:OUT &H0,&B00001001: PRINT "gruen       rot"
240 GOSUB 300:IF X=128 THEN 200:ELSE IF TIME<800 THEN 240
250 TIME=0:OUT &H60,&B00010010:PRINT "gelb        gelb"
260 GOSUB 300:IF X=64 THEN 200:ELSE IF TIME<250 THEN 260
270 TIME=0:OUT %H60,&B00100100:PRINT "rot         gruen"
280 GOSUB 300:IF X=64 THEN 200:ELSE IF TIME<400 ZHEN 280
290 GOTO 200
300 ' --- UP zur Portabfrage ---
310 X=INP(&H60)
320 RETURN
```

8.4. Unterbrechung der Programmausführung

WAIT Warten auf definierte Signaleingabe

Format: **WAIT** *portadresse*,*ausdruck1*[,*ausdruck2*]

portadresse - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 255 bzw. 00H bis FFH

ausdruck1,*2* - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 255

Funktion: Durch die Anweisung wird die Programmausführung angehalten, bis an der angegebenen Kanaladresse ein definierter Wert anliegt. Der an der Kanaladresse anliegende Wert wird XOR mit dem *ausdruck2* der Anweisung verknüpft. Das Ergebnis wird AND mit dem *ausdruck1* verknüpft. Die Verknüpfung erfolgt bitweise. Die Abarbeitung des Programms wird nur fortgesetzt, wenn das gebildete Ergebnis ungleich Null ist. Fehlt der *ausdruck2*, so wird sein Wert = 0 angenommen. Durch die WAIT-Anweisung kann somit das Warten auf ein externes Bitmuster programmiert werden.

Wertebelegungstabelle:

			T1	Y
X0	X1	X2	X0 XOR X2	T1 AND X1
0	0	0	0	0
1	0	0	1	0
0	1	0	0	0
1	1	0	1	1
0	0	1	1	0
1	0	1	0	0
0	1	1	1	1
1	1	1	0	0

X0 = eingelesenes Bit von der Kanaladresse
X1 = Bit des *ausdrucks1*
X2 = Bit des *ausdrucks2*

Da eine Programmfortsetzung nur für Y=1 erfolgt, sind die Bedingungen für die beiden Kombinationen

X0 = 1, X1 = 1, X2 = 0 bzw.
X0 = 0, X1 = 1, X2 = 1

erfüllt.

Beispiele: Eingabe eines geradzahligen Wertes durch Anwendung der WAIT-Anweisung von dem im Zeitgebermode programmierten CTC-Schaltkreis:

```
10 OUT &H51,&H27
20 OUT &H51,&B11111110
30 WAIT &H51,1,1
40 X=INP(&H51)
50 PRINT X;
60 GOTO 30
```

Wird der zweite Ausdruck auf 0 gesetzt, so kann mit dem analogen Programm nur ein ungeradzahliger Wert vom CTC-Kanal eingelesen werden:

```
10 OUT &H51,&H27
20 OUT &H51,&B11111111
30 WAIT &H51,1,0
40 X=INP(&H51)
50 PRINT X;
60 GOTO 30
```

8.5. Nutzung des Zeitzählers

TIME Wert des Zeitzählers setzen und abfragen

Im RBASIC existiert eine Systemvariable TIME (16 Bit), deren Wert zeitabhängig im Abstand von 20 ms (1/50 s) weitergezählt wird. Bei Erreichen des maximalen Wertes von 65535 beginnt die Zählung wieder bei 0. Die kleinste meßbare bzw. angebbare Zeiteinheit von 20 ms entspricht der in den Anweisungen PAUSE (Abschnitt 3) und ON INTERVAL GOSUB (Abschnitt 5.2). Die Systemvariable TIME ist über RBASIC-Anweisungen setzbar (TIME-Anweisung) und auswertbar (TIME-Funktion). Beachten Sie deshalb, daß TIME in zwei verschiedenen Bedeutungen, zum Setzen und zum Abfragen der Zeit verwendet wird.

Format: **TIME** = *numerischer_ausdruck*

numerischer_ausdruck - ganze Zahl von 0 bis 65535

Funktion: Die Systemvariable TIME wird mit dem Wert von *numerischer_ausdruck* initialisiert.

Hinweise:

1. Beim Start des RBASIC wird TIME mit Null initialisiert.
2. Bei der Bedienung der peripheren Geräte (Kassette, Diskette) wird der Systeminterrupt zeitweilig verboten. Aus diesem Grund "geht die Uhr dann etwas nach".
3. Das Weiterzählen von TIME erfolgt nur bei Stromzufuhr.

Format: **TIME**

Typ: BASIC-Funktion

Funktion: Liefert den augenblicklichen Wert der Systemvariablen TIME, d.h. einen Wert zwischen 0 und 65535.

Beispiele:

1. Die Zeiteinstellung soll auf 0 gesetzt und an einer beliebigen Stelle im Programm aufgerufen werden.

```
10 TIME=0
100 '--- Rechenzeit-Test ---
110 FOR I=1 TO 1000:X=I*I:NEXT I
120 '--- benötigte Zeit ---
1000 T=TIME/50:BEEP
1010 PRINT "Rechenzeit: "T;"Sekunden"
```

2. Programmierung einer Stoppuhr mit Stunden-, Minuten- und Sekundenanzeige

```
100 PRINT "Start durch >ENTER<"
110 INPUT"";X                       :'Start auslösen
120 TIME=0                          :'Start Zeit
130 PRINT "STOPPUHR LAEUFT"
140 PRINT "STOP durch >ENTER<"
150 INPUT "";X                      :'Halt auslösen
```

```
160 T=TIME                          :'Zeit abfragen
170 H=INT(T/180000!)
180 T=T-(H*180000!)
190 M=INT(T/3000)
200 T=T-(M*3000)
210 S=INT(T/50)
220 PRINT USING "##:##:##";H,M,S
```

3. Der Wert 65535 der Systemvariablen TIME entspricht 21 Minuten 50 Sekunden 70 Hundertstel Sekunden. Die Stoppuhr aus Beispiel 2 würde also nach 0:21:51 wieder bei Null beginnen. Ändern Sie das Programm zur Übung so, daß die Stunden normal mitgezählt werden.

9. Speicheraufteilung, Speicherzugriff und Maschinencodeprogramme

9.1. Maschinencodeprogrammentwicklung und Speicheraufteilung

RBASIC läuft auf einem Computer, der mit dem Mikroprozessor UA880 ausgerüstet ist. Programme auf der Ebene der Maschinensprache sind demzufolge im Maschinencode (mc) dieses Prozessortyps zu schreiben. Die Befehlscodierungen können einschlägigen Tabellen entnommen werden.

Für die Entwicklung von Programmen im Maschinencode sind folgende Schrittfolgen zweckmäßig:

1. Informieren über die Speicherbelegung und Reservieren eines Speicherbereiches für das Maschinencodeprogramm mittels CLEAR-Anweisung

2. Festlegen der Anfangsadresse des Maschinencodeprogramms durch die Anweisung DEF SR.

3. Erzeugen des Maschinencodeprogramms unter Anwendung der POKE-Anweisungen.

4. Sichern des Maschinencodeprogramms in binärer Codierung durch die Anweisung BSAVE.

5. Laden des Maschinencodeprogramms in den Speicher mittels der BLOAD-Anweisung.

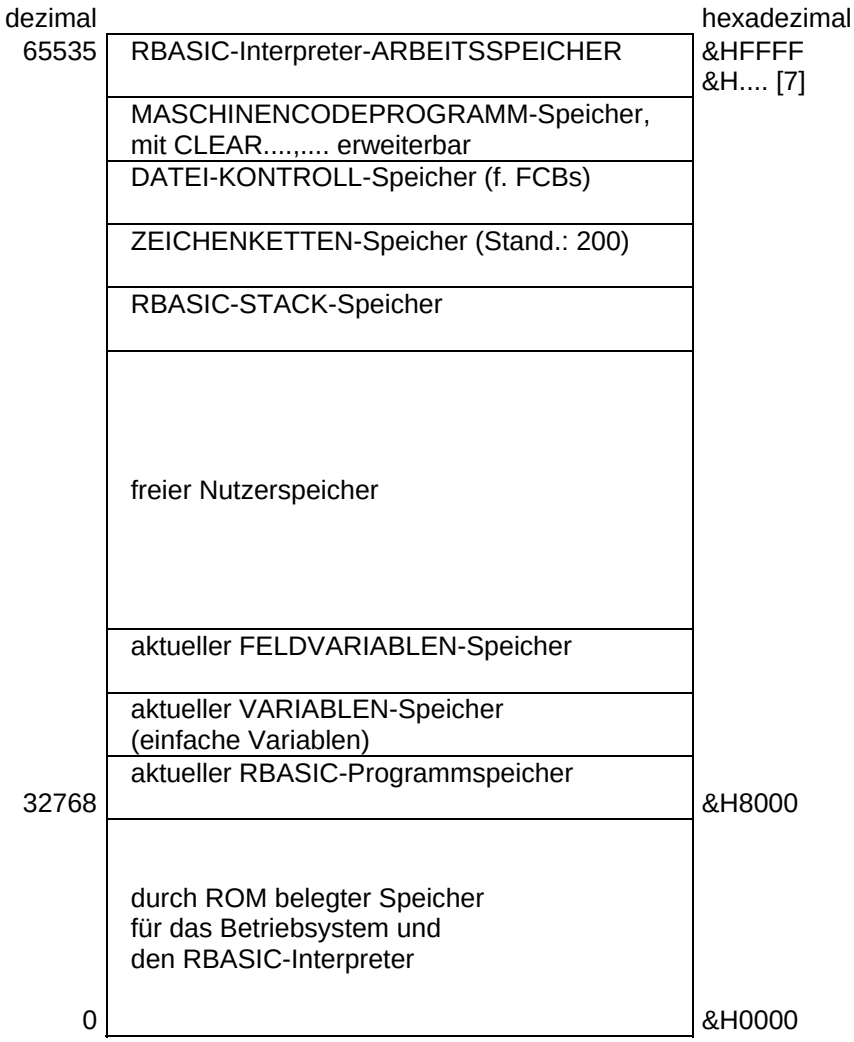
6. Aufruf des Maschinencodeprogramms durch die Funktion SR(X) aus dem RBASIC-Programm, ggf. mit Argumentübergabe in die Register A, DE und HL.

7. Auslesen einzelner Speicherinhalte (Ergebnisse) unter Anwendung der PEEK-Anweisung oder der Funktionswertübergabe an X=USR(X).

Jeder direkte Speicherzugriff, z.B. in Form der Erzeugung eines speziellen Maschinencodeprogramms, erfordert neben der Kenntnis des Maschinencodes eine genaue Kenntnis der Speicheraufteilung.

Durch unsachgemäße Speicherbelegungen können sehr leicht eigene RBASIC-Programme oder wichtige Systemzellen im Arbeitsspeicher des Betriebsystems bzw. des Interpreters überschrieben werden, was zum "Absturz" des Programms führen kann.

Aufteilung des 64-kByte-Speichers für die Arbeit mit RBASIC



Es wird erkennbar, daß für die Erzeugung von Maschinencodeprogrammen in erster Linie der Speicherbereich zwischen dem Arbeitsspeicher des RBASIC-Interpreters und des Betriebssystems und dem Datei-Kontrollspeicher zweckmäßig ist.

Hinweis: Die nach den eckigen Klammern einzufügenden Adressen sind der Erläuterungsdatei der Systemdiskette zu entnehmen.

Eine Verschiebung nach "unten" (in Richtung niedrigerer Speicheradressen) erfolgt durch den 2. Parameter der CLEAR-Anweisung. Beispielsweise reserviert die Anweisung

```
100 CLEAR 200,55000
```

einen Bereich von Adresse 55000 (&HD6D8) bis &H....[7] für die Eintragung von Maschinencode-programmen unter Beibehaltung des Standardspeichers für die Zeichenkettenvariablen.

Achtung!
Wird mit dem Interpreter BASI unter SCP gearbeitet, so gilt eine andere Speicheraufteilung, die den entsprechenden Dokumentationen zu entnehmen ist.

9.2. Direkter Speicherzugriff

PEEK Lesen des Inhalts der adressierten Speicherzelle
POKE Schreiben von einem Byte auf eine adressierte Speicherzelle
VARPTR Lesen von Variablenadressen

Die Anweisung POKE und die Funktion PEEK gestatten das Schreiben und das Lesen von Speicherzelleninhalten ausgewählter Adressen auf den bzw. vom Speicher des Computers, also den direkten Speicherzugriff.

Durch die Funktion VARPTR (VARiablen PoinTeR - Variablenzeiger) können die Variablen aller Typen einschließlich ihrer charakteristischen Parameter gelesen werden, weil alle Variablen nach einem einheitlichen Format abgespeichert sind.

Bitte beachten Sie:

- Die Anwendung aller Speicherzugriffsanweisungen und -funktionen setzt Grundkenntnisse über die Speicheraufteilung voraus. Informieren Sie sich vor ihrer Verwendung hinreichend über die Speicheraufteilung, um sich unnötigen Ärger zu ersparen.

- Mit der PEEK-Funktion ist auch ein Zugriff auf spezielle Systemzellen des Betriebssystems oder eines Maschinencodeprogramms möglich, die für eine Programmerweiterung nützlich sein können.

- Durch eine falsch adressierte POKE-Anweisung können für den Programmablauf wichtige Speicherplätze überschrieben werden, wodurch das Programm verändert bzw. zum "Absturz" gebracht wird.

- POKE-Anweisungen eignen sich, besonders in Verbindung mit der READ-DATA-Anweisung, um Maschinencodeprogramme in den Speicher einzutragen. Die Anfangsadressen der Maschinencodeprogramme müssen jedoch genau definiert sein.

Schreiben auf Speicherplätze

Format: **POKE** *adresse*,*ausdruck*

adresse - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 65535 bzw. &H0 bis &HFFFF

ausdruck - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 255 bzw. &H0 bis &HFF

Funktion: Mit der Anweisung wird ein Byte (8 Bits) auf den Speicherplatz der angegebenen Adresse geschrieben.

Hinweise:

1. Die Anweisung kann sowohl zum "Setzen" einzelner Speicherzellen als auch zum Eintragen kompletter Maschinencodeprogramme angewendet werden.

2. Durch die POKE-Anweisung können auch in RBASIC-Programmen oder in Variablenspeichern (z.B. im Zeichenkettenspeicher) partielle Änderungen vorgenommen werden, die ggf. für den Programmlauf erforderlich sind.

3. Die POKE-Anweisung erfordert Kenntnisse der allgemeinen Speicherbelegung.
   Für besondere Anwendungen sind Detailkenntnisse spezifischer Speicher, beispielsweise des Arbeitsspeichers, nötig.

4. Zum Eintragen von Maschinencodeprogrammen hat sich die hexadezimale Zahlendarstellung bewährt. Die Ausdrücke von 0 bis 255 werden daher nachfolgend als Hexadezimalzahlen mit dem Präfix &H eingetragen.

Beispiel: In den Maschinensprachenspeicher soll ein einfaches Programm zur Addition zweier Dualzahlen (0 ... 255) eingetragen werden. Das Ergebnis soll auf die Adresse 55040 dezimal ausgegeben werden.

```
10 CLEAR 200,55000       :'Speicher reservieren
20 DATA &H3E,&H0A        :'LD  A,10      ;1.Operand in A
30 DATA &H06,&H14        :'LD  B,20      ;2.Operand in B
40 DATA &H80             :'ADD B         ;Addition A und B
50 DATA &H32,&H00,&HD7   :'LD  (55040),A;A auf 55040 ablegen
60 DATA &HC9             :'RET           ;Zurück zum HP
70 FOR I=0 TO 8          :'mc-Programm einladen
80     READ P
90     POKE 55000+I,P
100 NEXT I
```

Bevor eine Eintragung in den Speicher vorgenommen wird, ist durch die CLEAR-Anweisung der Speicherbereich zu reservieren (Zeile 10).
Die Maschinencodewörter, werden bei der Adresse 55000 beginnend, in den Speicher eingetragen. Das Maschinencodeprogramm ist durch den Befehl RET (Dezimaläquivalent = 201) abgeschlossen Es kehrt also nach der Abarbeitung in das aufrufende Programm zurück. Das Ergebnis (Summe der beiden Registerinhalte A und B) wird in die Speicherzelle der Adresse 55040 dezimal (&HD700) eingetragen und kann dort leicht aus dem RBASIC-Programm gelesen werden.

Lesen von Speicherplätzen

Format: **PEEK**(*adresse*)

adresse - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 65535 bzw. &H0 bis &HFFFF

Typ: BASIC-Funktion

Funktion: Der Inhalt des Speicherplatzes der angegebenen Adresse wird als Funktionswert angenommen. Er liegt zwischen 0 und 255.

Hinweise:

1. Über die Funktion PEEK besteht die Möglichkeit, wichtige Speicherzelleninhalte des Betriebssystems zu lesen.

2. Aus Maschinencodeprogrammen können Registerinhalte, Adressen und abgelegte Ergebnisse in das RBASIC-Programm übertragen werden.

3. Die PEEK-Funktion ist besonders geeignet, Speicherinhalte auszulesen und in Form von Dezimaläquivalenten bzw. nach Umformung in hexadezimaler, dualer oder ASCII-Form darzustellen.

Beispiel: Im nachfolgenden Beispiel wird die PEEK-Anweisung gleichzeitig in mehreren Funktionen verwendet. Es werden Werte aus Speicherzellen in der normalen dezimalen Darstellung ausgelesen, und daneben wird die PEEK-Funktion innerhalb der Funktion HEX$(X) verwendet, um die Speicherinhalte sofort in hexadezimaler Darstellung anzuzeigen.
Die Einbindung in die Funktion CHR$(X) ermöglicht es weiterhin, die gelesenen Werte in Form von ASCII-Zeichen darzustellen.

```
10 ' --- Hexa- und ASCII-DUMP ---
20 WINDOW:CLS
30 INPUT "ANFANGSADRESSE(dez.):";A
40 INPUT "ENDEDRESSE    (dez.):";E
50 CLS
60 PRINT "Adr(H) hexadez.DUMP      ASCII-DUMP"
70 WINDOW 2,23,0,39                :'Kopfzeile feststellen
80 FOR I=A TO E STEP 8
90     HA$=HEX$(I):PRINT RIGHT$("000"+HA$,4);SPC(3);
100    FOR J=I TO I+7
110        HH$=HEX$(PEEK(J))
```

```
120         IF LEN(HH$)=1 THEN HH$="0"+HH$
130         PRINT HH$;
140     NEXT J
150     PRINT SPC(2);
160     FOR K=I TO I+7
170         IF PEEK(K)<31 THEN PRINT" ";
                          ELSE PRINT CHR$(PEEK(K));
180     NEXT K
190     PRINT
200 NEXT I
210 PRINT ">ENTER<";
220 IF CHR$(13)<>INKEY$ THEN 220
230 WINDOW:CLS
240 END
```

Das beschriebene Programm eignet sich also dazu, um einen HEXA- und ASCII-DUMP herzustellen.

Lesen von Variablenadressen

Format: **VARPTR**(*var*)

var - Bezeichnung einer numerischen oder Zeichenkettenvariablen durch Buchstaben mit oder ohne nachfolgendes Kennzeichen des Variablentyps.
Die Typkennzeichnung erfolgt wie üblich durch %, !, # oder $. Variable ohne Typkennzeichen bezeichnen Variable mit doppelter Genauigkeit (8-Byte-Zahl).

Typ: BASIC-Funktion

Funktion: Übergibt die Speicheradressen, vor und nach denen in den Speicherzellen die Variable kennzeichnende Parameter (TYP, Name, Zeichenkettenlängen, Folgeadressen und Daten) eingetragen sind. Ermöglicht ebenfalls das Auffinden von Startadressen für Maschinencodeprogramme (nur bei ganzzahligen Variablen) und die Übergabe von Werten in Programme.
Der Funktionswert (ausgegebene Adresse) ist eine ganze Zahl im Bereich von 0 bis 65535.

Hinweise:

1. Vor dem Aufruf von VARPTR muß der aufzurufenden Variablen ein Wert zugewiesen werden. Andernfalls wird die Fehlermeldung "Illegal function call" (ungültiger Funktionsaufruf) ausgegeben.

2. Bevor ein Aufruf mittels VARPTR erfolgt, sollte allen Programmvariablen bereits ein Wert zugeordnet sein, da sich die Adressen bei jeder Neuzuordnung ändern können.

3. Die Wertespeicherung für die Variablen ist vom Variablentyp abhängig. Sie beginnt bei der im Funktionswert VARPTR angezeigten Adresse.

   - Ganzzahlige Werte werden in zwei Bytes gespeichert, das niedrigste Byte zuerst, danach das höhere Byte.

   - Werte einfacher und doppelter Genauigkeit werden in 4 bzw. 8 Bytes in der internen Gleitpunktdarstellung gespeichert.

   - Bei Zeichenkettenvariablen gibt der erste Wert die Länge der Zeichenkette an. Die folgenden zwei Bytes enthalten die Adresse der Zeichenkette in der Folge Low-Byte, High-Byte (LB, HB).

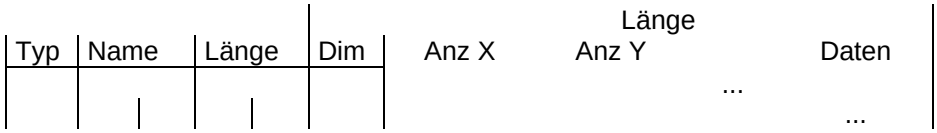
4. Vor der im Funktionswert VARPTR angezeigten Adresse sind zusätzliche Parameter für den Variablentyp in einem Byte (2 = ganzzahlig, 3 = Zeichenkette, 4 = einfach genau, 8 = doppelt genau) und die zwei ersten Zeichen des Variablennamens eingetragen.

Speicherung von numerischen und Zeichenkettenvariablen

Typ	Name		Daten							
2	..	..	LB	HB	--	--	--	--	--	--
3	..	..	Länge	LB	HB	--	--	--	--	--
4	..	..	DA1	DA2	DA3	DA4	--	--	--	--
8	..	..	DA1	DA2	DA3	DA4	DA5	DA6	DA7	DA8

↑
+-- Funktionswert (Adr.) des VARPTR

5. Numerische und Zeichenkettenfelder sind in den entsprechenden Feldspeichern abgelegt. Die Speicherformate sind denen der einfachen numerischen und Zeichenkattenvariablen gleich, jedoch sind außer TYP, NAME und DATEN für die Felder noch die LÄNGE (des nachfolgenden belegten Speicherbereiches), und die DIMENSION angegeben.



6. Da Maschinencodeprogramme häufig als ganzzahlige numerische Felder definierter Länge abgelegt sind, können die Anfangsadressen mittels VARPTR ermittelt und über Zuweisung des Funktionswertes an die DEFUSR[X]-Anweisung gestartet werden.

7. Der Funktionswert von VARPTR kann bei ganzzahligen Variablen ebenfalls für den Adresseneintrag in der POKE-Anweisung verwendet werden. Dadurch wird die Neubelegung der Variablen auch in dieser Form möglich.

Beispiel:

1. Ausgabe der formatierten Parameter einer ganzzahligen Variablen mit dem Namen AB% bei Aufruf mit der Funktion VARPTR(AB%).

```
100 INPUT "ganzzahhlige Variable AB%:";AB%     :'z.B. 520
110 AD=VARPTR(AB%)
120 IF AD<0 THEN AD=AD+65535!
130 TY=PEEK(AD-3):Z1=PEEK(AD-2):Z2=PEEK(AD-1)
140 B1=PEEK(AD):B2=PEEK(AD+1)
150 U$="## ## ## ### ###"
160 PRINT "Typ Name  Zahl"
170 PRINT USING U$;TY,Z1,Z2,B1,B2
180 PRINT "Anfangsadresse:";AD-3
190 PRINT "Endadresse    :";AD+1
200 PRINT "VARPTR-Adresse:";VARPTR(AB%)
```

Als Variablentyp erscheint nach dem Aufruf der Wert "2" für ganzzahlige Variable. Die Ziffern "65" und "66" entsprechen den Buchstaben A und B im Variablennamen. Die nächsten beiden Ziffern "8" und "2" stellen das Low- und das Highbyte dar, wenn der eingegebene Wert 520 lautet, denn

2 * 256 + 8 = 520.

2. Ausgabe der formatierten Parameter einer Zeichenkettenvariablen bei Aufruf der Funktion VARPTR(CD$).

```
100 INPUT "Zeichenkettenvariable CD$";CD$
110 AD=VARPTR(CD$)
120 IF AD$<0 THEN AD=AD+65535!
130 TY=PEEK(AD-3):Z1=PEEK(AD-2):Z2=PEEK(AD-1)
140 B1=PEEK(AD):B2=PEEK(AD+1):B3=PEEK(AD+2)
150 U$="## ## ###    ### ##"
160 PRINT "Typ Name Laenge Adresse"
170 PRINT USING U$;TY,Z1,Z2,B1,B2,B3
```

```
180 ZA=B3*256+B2
190 FOR I=0 TO B1-1
200     PRINT"Adresse:";ZA+I;"Zeichen  ";CHR$(PEEK(ZA+I))
210 NEXT I
```

Der angezeigte Variablentyp ist 3. Die Ziffern 67 und 68 entsprechen den Buchstaben C und D im ASCII-Code. Danach folgt der Wert für die Zeichenkettenlänge, die von der Zahl der Zeichen in der eingegebenen Zeichenkette abhängt (maximal 255). Es folgen der niederwertige und der höherwertige Adressenanteil für den Beginn der Speicherung der einzelnen Zeichen im Zeichenkettenspeicher.

Hier sind also, im Gegensatz zu den numerischen Variablen, keine 2, 4 oder 8 Bytes mit Daten belegt, sondern nur zwei Bytes mit einer Adresse, auf der die Zeichenkette zu finden ist.

3. Ein Speicherbereich soll durch zehn Variable mit doppelter Genauigkeit belegt werden. Anschließend sollen die Adressen des VARPTR, die Anfangs- und Endadresse und der benötigte Speicherplatz ausgegeben werden

```
10 DIM X#(9)
100 FOR I=0 TO 9
110     X#(I)=.12345678#+I
120 NEXT I
130 PRINT "Var.    VARPTR (Anfang)  Endadresse"
140 PRINT STRING$ (35,"_"):PRINT
150 FOR I=0 TO 9
160     PRINT"X#(";I;") ";VARPTR(X#(I));"            ";
        VARPTR(X#(I))+7
170 NEXT I
180 X=VARPTR(X#(9))+7-VARPTR(X#(0)+1
190 PRINT "benoetigter Speicherplatz:";X;"Bytes"
```

Der nachfolgend dargestellte Programmablauf verdeutlicht den hohen Speicherbedarf bei der Anwendung von Variablen doppelter Genauigkeit.

```
RUN

Var.    VARTPR (Anfang)  Endadresse

___________________________________

X#( 0 ) -32475           -32468
X#( 1 ) -32467           -32460
X#( 2 ) -32459           -32452
X#( 3 ) -32451           -32444
X#( 4 ) -32443           -32436
X#( 5 ) -32435           -32428
X#( 6 ) -32427           -32420
X#( 7 ) -32419           -32412
X#( 8 ) -32411           -32404
X#( 9 ) -32403           -32396
benoetigter Speicherplatz: 80 Bytes
```

9.3. Festlegen von Startadressen für Unterprogramme im Maschinencode

DEFUSR legt Startadresse für Maschinencodeprogramme fest

Maschinencodeprogramme können in verschiedenen Bereichen des Speichers installiert werden. Nur für bestimmte Speicherbereiche ändern sich im Verlaufe der Programmerstellung die Adressierungen nicht. Der Aufruf von Maschinencodeprogrammen erfordert daher genaue vorherige Adressen-definition, die gegebenenfalls auch unter Zuhilfenahme der Funktion VARPTR vorgenommen werden muß.

Format: **DEFUSR** [X] = *startadr*

startadr - numerischer Ausdruck (ganzzahlig) mit einem Wert von 0 bis 65535

X - numerischer Ausdruck (ganzzahlig) von 0 bis 9.
Bezeichnet die Nummer der USR-Funktion, deren Adresse spezifiziert wurde. Ist X nicht angegeben, wird DEFUSR 0 gesetzt.

Funktion: Bestimmt eine Startadresse, wenn ein Maschinencodeprogramm durch eine USR-Funktion aufgerufen wird.

Hinweise:

1. Die Zahl der DEFUSR-Anweisungen in einem Programm ist nicht begrenzt, so daß auf so viele Maschinencodeprogramme wie nötig zugegriffen werden kann. Die zuletzt zugewiesene Startadresse wird als Anfangsadresse benutzt.

2. Wird eine Adressenbestimmung eines Maschinencodeprogramms in einem nicht mit der CLEAR-Anweisung reservierten Maschinencodespeicherbereich erforderlich, so ist, soweit möglich, mit der Funktion VARPTR die Anfangsadresse zu bestimmen.
Dadurch können die im Maschinencode geschriebenen Unterprogramme auch in den numerischen oder Zeichenkettenspeicherbereich geladen und von dort aufgerufen werden.

3. Die Anweisung DEFUSR ist an den Programmaufruf durch die Funktion SR(X) gebunden und bereitet gleichzeitig eine Argumentübergabe vor.

Beispiele:

1. Definition der Anfangsadresse eines Maschinencodeprogramms im reservierten Maschinencodespeicherbereich. Die Größe des Zeichenkettenspeichers (200 Bytes) soll nicht verändert werden.

```
10 CLEAR 200,55000
20 DEFUSR 1=55000
```

2. Eintragung eines Maschinencodeprogramms in den reservierten Speicherbereich und Start des Programms an der mit der Funktion VARPTR ermittelten Adresse durch die Anweisung SR.

```
10 ' --- mc-Programm im Variablenspeicher --------
20 I%=0:J%=0:V%=0:X=0:Z=0          :'Platz für Variable und
30 DIM AR#(1)                      :'Felder reservieren
40 ' --- Definition mc-Programm ---
50 DATA &H21,&H00,&HD7             :'LD HL,55040
60 DATA &H36,&H30                  :'LD (HL),48
70 DATA &H11,&H01,&HD7             :'LD DE,55041
80 DATA &H01,&H80,&H00             :'LD BC,128
90 DATA &HED,&HB0                  :'LDIR
100 DATA &HC9                      :'RET
110 ' --- mc-Programm in numerischen Feld einladen ---
120 V%=VARPTR(AR#(0))
130 FOR I%=0 TO 13
140     READ J% :POKE V%+I%,J%
150 NEXT I%
160 ' --- Anzeige des mc-Programms im Speicher ---
170 FOR I%=V% TO V%+13
180     PRINT HEX$(PEEK(I%));" ";
190 NEXT I%
200 PRINT:PRINT
210 ' --- Startadresse festlegen und aufrufen ---
220 DEFUSR1=V% :X=USR1(0)
230 ' --- Speicherkontrolle ---
240 FOR Z=55030! TO 55200!
250     PRINT Z;PEEK(Z)
260 NEXT Z
```

Nachdem zunächst der Speicherraum mit der CLEAR-Anweisung reserviert wurde, erfolgt die Eintragung des Maschinencodeprogramms. Die Startadresse wird durch die Funktionswertzuweisung DEFUSR1=V%=VARPTR(AR#(0)) für den ersten

eingetragenen Maschinencodebefehl ermittelt (Zeilen 220 und 120). Der mc-Programmaufruf erfolgt mit der Anweisung X=USR1(0). Die anschließende Speicherkontrolle zeigt, daß der festgelegte Speicher tatsächlich mit Nullen beschrieben wurde (ASCII-Zeichen 48). Auf diese Weise kann beispielsweise auch ein Textspeicher mit Leerzeichen beschrieben und dadurch mittels eines Maschinencodeprogramms sehr schnell gelöscht werden.

9.4. Aufrufen von Maschinencodeprogrammen

USR Aufrufen eines Maschinencodeprogramms

Ein in dezimalem oder hexadezimalem Format geschriebenes und über die Anweisung POKE in den Speicher eingetragenes Maschinencodeprogramm, dessen Anfangsadresse entweder mit der Anweisung DEFUSR definiert oder mittels der Funktion VARPTR ermittelt wurde bzw. welches durch die Anweisung BLOAD binär codiert in den Speicher eingetragen wurde, kann über die Anweisung SR zur Ausführung gebracht werden.
Voraussetzungen für den Start des Maschinencodeprogramms aus dem RBASIC-Programm und die Rückkehr in dieses Programm mittels der Funktion SR sind:

1. Die bereits vollzogene Eintragung des Maschinencodeprogramms in einen geeigneten Speicherbereich.

2. Der Abschluß des Maschinencodeprogramms mit dem Return-Befehl (RET).

3. Die Definition der Startadresse über die Anweisung DEFUSR (s. Abschnitt 9.3) gegebenenfalls bestimmt mittels VARPTR.

4. Der numerische Ausdruck (0 bis 9) in der DEFUSR-Anweisung muß mit diesem Ausdruck in der DEFUSR-Funktion übereinstimmen.

Die USR-Funktion kann nicht nur einen Programmstart auslösen. Zusätzlich ist es möglich, auch eine Eintragung von Argumenten in Prozessorregister vorzunehmen und nach beendeter Programmausführung einen Registerwert als Ergebnis an die zugeordnete Variable des aufrufenden RBASIC-Programms auszugeben.

Format: **USR**[X](*argument*)

X - Nummer des mittels DEFUSR festgelegten Maschinencodeprogramms; ganzzahliger numerischer Ausdruck von 0 bis 9

argument - numerischer oder Zeichenkettenausdruck (ist auch als Blindargument erforderlich)

Typ: BASIC-Funktion

Funktion: Ruft ein Maschinencodeprogramm an der mit der zugehörigen DEFUSR-Funktion definierten Startadresse und mit dem *argument* auf.
X muß mit dem Wert in DEFUSR übereinstimmen.
Wird X nicht angegeben, wird es als 0 angenommen. Das Ergebnis, das nach der Ausführung des Maschinencodeprogramms erzielt wird, kann als Funktionswert übergeben werden. Dazu müssen Typ und Wert in die entsprechenden Speicherzellen eingetragen werden:

Typ : ....H [8]
Wert in FAC : ....H [9]

entsprechend Beispiel 2.

Beispiele:

1. Aufruf eines Maschinencodeprogramms ohne Nutzung der Argumentübergabe.

```
10 DEFUSR1=&HD7FF
.
.
.
100 Y=USR1(0)
.
.
```

Das Maschinencodeprogramm, für das in der Zeile 10 die Startadresse definiert wurde, wird in der Zeile 100 aus dem RBASIC-Programm ohne Argumentübergabe gestartet. Nach dem Return (RET) im Maschinencodeprogramm wird das RBASIC-Programm nach der Zeile 100 fortgesetzt.
Im gewählten Beispiel erfolgte der Aufruf des Maschinencodeprogramms für die CLS-Routine im Betriebssystem. Das übergebene Argument wird nicht ausgewertet.

2. Aufruf eines Maschinencodeprogramms zur Umwandlung der Kleinbuchstaben a...z einer Zeichenkette in die Großbuchstaben A...Z. Als Funktionswert wird die Länge der Zeichenkette zurückgegeben.
Wird keine Zeichenkette eingegeben, wird das Programm beeendet.

```
10 CLS
20 CLEAR 300,56000!
30 '--- mc-Programm ---
40  DATA &heb           :'    ex de,hl   ;Adr.str.descr. in hl
50  DATA &h46           :'    ld b,(hl)  ;Länge ZK in b
60  DATA &h78           :'    ld a,b
70  DATA &hf5           :'    push af
80  DATA &h23           :'    inc hl
90  DATA &h7e           :'    ld a,(hl)
100 DATA &h23           :'    inc hl
110 DATA &h66           :'    ld h,(hl)
120 DATA &h6f           :'    ld l,a     ;Adr ZK in hl
130 DATA &h7e           :'M2  ld a,(hl)  ;Zeichen aus ZK
140 DATA &hfe,&h61      :'    cp 'a'
150 DATA &h38,&h07      :'    jr c,M3    ;<'a'
160 DATA &hfe,&h7b      :'    cp '{'
170 DATA &h30,&h03      :'    jr nc,M3   ;>='{'
180 DATA &he6,&h5f      :'    and 05fh   ;-> Großbuchstabe
190 DATA &h77           :'    ld (hl),a
200 DATA &h23           :'M3  inc hl     ;nächstes Zeichen
210 DATA &h10,&hf1      :'    djnz M2    ;weiter, bis ZK durch
220 DATA &hf1           :'M1  pop af
230 DATA &h21,&hbe,&hf6 :'    ld hl,AC1+2
240 DATA &h77           :'    ld (hl),a  ;Länge in FAC+2
250 DATA &h23           :'    inc hl
260 DATA &h36,&h00      :'    ld (hl),0
270 DATA &h3e,&h02      :'    ld a,2     ;Typ integer
280 DATA &h32,&h29,&hf5 :'    ld (TYP),a
290 DATA &hc9           :'    ret
300 '--- mc-Programm einlesen ---
310 FOR I%=0 TO 37
320     READ A%: POKE 56000!+I%,A%
330 NEXT
340 '--- Adr. definieren ---
350 DEFUSR0 = 56000!
360 '--- Aufruf ---
370 ZK$ = ""
380 INPUT "Zechenkette: ";ZK$
390 IF ZK$="" THEN END
400 L%=USR(ZK$)
410 PRINT L%, ZK$
420 GOTO 360
```

3. Eintrag und Aufruf eines kompletten Maschinencodeprogramms zur Bedienung eines Analog-Digital-Umsetzers (C520), bei dem die übergebenen digitalen Werte (MSD, NSD, LSD) über die PEEK-Funktion in das BASIC-Programm übernommen werden.

```
10 ' --- Speicherreservierung und mc-Startadresse ---
20 CLEAR 200,55000 : DEFUSR0=55000 :CLS
30 ' --- PIO-Initialisierung ---
40 OUT &H62,&HCF          :'Mode 3
50 OUT &H62,&B11111111    :'A0-A7 Eingabe
60 ' --- mc-Programm einladen ---
70 FOR I=0 TO 63
80     READ X
90     POKE 55000!+I,X
100 NEXT I
110 ' --- mc-Programm ---
120 DATA &HD9             :'     EXX        ;Wechsel auf Hintergr.-Reg.
130 DATA &H08             :'     EXAF       ;    "          "      "
140 DATA &HDB,&H60        :'M1:  IN 96      ;Port abfragen
150 DATA &H47             :'     LD B,A     ;A in B zwischenspeichern
160 DATA &HE6,&B00110000  :'     AND 48     ;Maske für Multiplexsignale
170 DATA &HFE,&B00000000  :'     CMP 0      ;auf MSD prüfen
180 DATA &H20,&HF7        :'     JRNZ M1-#  ;zurück zu M1 bei ungl. 0
190 DATA &HDB,&H60        :'     IN 96      ;Port abfragen
200 DATA &HB8             :'     CMP B      ;Zustandskontrolle A=B
210 DATA &H20,&HF2        :'     JRNZ M1-#  ;zurück zu M1 bei ungl. 0
220 DATA &H21,&H00,&HD8   :'     LD HL,55296  ;Adr. f. Var eintragen
230 DATA &H78             :'     LD A,B     ;B zur Ausgabe vorbereiten
240 DATA &.HE6,&B00001111 :'     AND 15     ;Maske für BCD-Werte
250 DATA &H77             :'     LD M,A     ;MSD auf 55296 ablegen
260 DATA &H23             :'     INC HL     ;Ablagezeiger auf 55297
270 DATA &HDB,&H60        :'M2:  IN 96      ;Port abfragen
280 DATA &H47             :'     LD B,A     ;A in B zwischenspeichern
290 DATA &HE6,&B00110000  :'     AND 48     ;Maske f. Multiplexsignale
300 DATA &HFE,&B00100000  :'     CMP 32     ;auf LSD prüfen
310 DATA &H20,&HF7        :'     JRNZ M2-#  ;zurück zu M2 bei ungl. 0
320 DATA &HDB,&H60        :'     IN 96      ;Port abfragen
330 DATA &HB8             :'     CMP B      ;Zustandskontrolle A=B
340 DATA &H20,&HF2        :'     JRNZ M2-#  ;zurück zu M2 bei ungl. 0
350 DATA &H78             :'     LD A,B     :B zur Ausgabe vorbereiten
360 DATA &HE6,&B00001111  :'     AND 15     ;Maske f. BCD-Werte
370 DATA &H77             :'     LD M,A     ;LSD auf 55297 ablegen
380 DATA &H23             :'     INC HL     ;Ablagezeiger auf 55298
390 DATA &HDB,&H60        :'M3:  IN 96      ;Port abfragen
400 DATA &H47             :'     LD B,A     ;A in B zwischenspeichern
410 DATA &HE6,&B00110000  :'     AND 48     ;Maske f. Multiplexsignale
420 DATA &HFE,&B00010000  :'     CMP 16     ;auf NSD prüfen
430 DATA &H20,&HF7        :'     JRNZ M3-#  ;zurück zu M3 bei ungl. 0
440 DATA &HDB,&H60        :'     IN 96      ;Port abfragen
450 DATA &HB8             :'     CMP B      ;Zustandskontrolle A=B
460 DATA &H20,&HF2        :'     JRNZ M3-#  ;zurück zu M3 bei ungl. 0
470 DATA &H78             :'     LD A,B     ;zur Ausgabe vorbereiten
480 DATA &HE6,&B00001111  :'     AND 15     ;Maske f. BCD-Werte
490 DATA &H77             :'     LD M,A     ;NSD auf 55298 ablegen
500 DATA &H0B             :'     EXAF       ;Wechsel auf Vordergr.-Reg.
510 DATA &HD9             :'     EXX        ;    "          "      "
520 DATA &HC9             :'     RET        ;zurück zum BASIC-Programm
530 ' --- mc-Programm starten und Werte anzeigen ---
540 X=USR(0)
550 A = 55296!
560 MSD = PEEK(A)  :  LSD = PEEK(A+1 ) : NSD=PEEK(A+2)
570 DEW!= MSD*10+NSD+LSD*.1
580 LOCATE 5,5:PRINT"MSD NSD LSD dezimaler Wert"
590 U$ = "# #  # #  # #     # #.#"
600 LOCATE 7,5,0:PRINT USING U$;MSD,NSD,LSD,DEW!
610 GOTO 530
```

Nach der Speicherreservierung wird die Anfangsadresse mit DEFUSR definiert (Zeile 20). Die für den konkreten Anwenderfall erforderliche Initialisierung des PIO-IC steht in den Zeilen 40 und 50 (Mode 3, alles Eingabe). Die Schleife (Zeile 70 bis

100) trägt das in den Zeilen 120 bis 520 stehende Maschinencodeprogramm in den reservierten Speicherraum ein.
Das Maschinencodeprogramm selbst wird danach durch die USR-Funktion (Zeile 540) gestartet.
Vom Maschinencodeprogramm auf den Speicherzellen 55296, 55297, 55298 abgelegte Werte können im RBASIC-Programm beliebig oft durch die PEEK-Funktion ausgelesen und angezeigt werden. Wird statt dessen ein Eintrag aus dem Maschinencodeprogramm in den Gleitpunktakkumulator vorgenommen, ist auch eine direkte Funktionswerteausgabe der BCD-Werte, ohne Zwlschenspeicherung, über die USR-Funktion möglich. Es ist jedoch die geänderte Formatierung zu beachten.

4. Funktionswerteausgabe mittels USR-Funktion aus einem Frequenzmessprogramm im Maschinencode:

```
10 ' - - zur Frequenzmessung -
20 DATA &H3E,&H7F   :'    LD A,07FH  ;CTC-Mode
30 DATA &HD3,&H51   :'    OUT  051H  ;OUT CTC-Kanal
40 DATA &H3E,&H66   :'    LD A,066H  ;Zeitkonstante
50 DATA &HD3,&H51   :'    OUT  051H  ;OUT CTC-Kanal
60   DATA   &H3E,&H78          .             LD   A,078H
;AbstandsdimensionierUn~
70 DATA &H3D        :'M1: DEC A      ,-1
80 DATA &H20,&HFD   :'    JRNZ M1    ;Schleife bis A=0
90 DATA &H23        .    INC HL     ;FAC+1
100 DATA &H23       :'    INC HL     ;FAC+2
110 DATA &H23       :'    INC HL     ;FACH3
120 DATA &H36,&H00  :'    LD M,0
130 DATA &H2B       :'    DEC HL     ;FACH2
140 DATA &HDB,&H51  :'    IN   051H  ;IN CTC-Kanal
150 DATA &H77       :'    LD M,A     ;Eintrag in FAC+2
160 DATA &HC9       :'    RET        ;Rücksprung
170' - mc-Programm einladen -
180 CLEAR 200,56000
190 FOR I=56000! TO 56022!
200     READX: POKEI,X
210 NEXT I
220 ' - mc-Programm aufrufen und Werteanzeige -
230 DEFUSR1=56000!
240 LOCATE 10,10,0: PRINT (102-USR1(X96))
250 IF INKEY$="" THEN 240
260 END
```

Nachdem das mc-Programm (Zeilen 20...160) Über die Schleife (Zeilen 190...210) in den reservierten Speicher eingetragen wurde, wird in Zeile 230 die mc-Anfangsadresse definiert. Im mc-Programm befindet sich (Zeilen 90...150) die Funktionswertübergabe in den FAC. Der Wert aus dem CTC (Zählerstand) wird mit USR1(X%) in das BASIC-Programm übertragen und (Zeile 240) auf dem Bildschirm angezeigt.

S A C H W O R T V E R Z E I C H N I S

E

F

G

H

I

K