BASIC

Einmaleins des Programmierens

Eine Einführung in die Programmiersprache BASIC

Von Dr. Ursula Grote und Prof. Dr. Horst Völz

Vorwort

Nicht zuletzt durch den großen Erfolg der ausgestrahlten Rundfunksendungen haben sich die Verfasser entschlossen, diese Einführung in die Programmierprache BASIC im Rahmen einer URANIA-Extraausgabe herauszugeben. BASIC ist in der Zwischenzeit zur am weitesten verbreiteten Programmiersprache geworden. BASIC, eine Abkürzung für Beginners Allpurpose Symbolic Instruction Code wurde ursprünglich als leicht erlernbare höhere Programmiersprache vorwiegend zur Lösung technisch-wissenschaftlicher Probleme entwickelt. Der Sprachaufbau lehnt sich sehr stark an FORTRAN und ALGOL an. Der zur Zeit zur Verfügung stehende Sprachumfang von BASIC gestattet es, sehr viel mehr als einfache Rechenoperationen zu lösen. Heute existiert eine große Anzahl unterschiedlicher, mehr oder minder komfortabel ausgebauter BASIC-Versionen.

Insgesamt sind es fast einhundert Programme aus verschiedenen Gebieten, doch fast nie aus dem Bereich der Mathematik, die hier veröffentlicht werden. Sie werden Ihnen kaum Probleme hinsichtlich des Verstehens bringen. Von eingen Ausnahmen abgesehen, sind es Programme, die lediglich die Programmierung demonstrieren sollen. Daher fehlen auch Aussagen zur Korrektheit z. B. bzgl. falscher Eingaben usw.

Für die Rundfunksendungen war eine spezifische Aufbereitung des Stoffes notwendig. Sie erfordert insbesondere kurze Programme, die leicht einer verbalen Beschreibung zugänglich waren. Diese Methodik wird hier beibehalten.

Die Verfasser haben sich bemüht, bereits in den ersten Kapiteln auf die in den meisten Lehrbüchern fehlenden Optimierungsfragen einzugehen. Bei den veröffentlichten Programmen – so hoffen wir – ist es uns gelungen, unübliche Aspekte mit interessanten Fragestellungen zu finden. Infolge dieser Vorgehensweise ist diese Einführung nicht nur für Anfänger nützlich, sondern es wird sehr wahrscheinlich auch jenen hilfreich sein, die bereits mit BASIC ein wenig vertraut sind. Dieser Kurs ist also in erster Linie als ein neuartiger Zugang und damit als Ergänzung zu den vorhandenen Lehrbüchern zu verstehen. Ebenso hoffen wir, daß er hilfreich für Computerklubs und Arbeitsgemeinschaften sein werde.

Um den Umfang des Heftes nicht zu stark auszuweiten, wird die Maschinensprache der Computer nicht behandelt, auch auf mitbenutzbare Routinen des Interpreters kann nur begrenzt eingegangen werden. Das gleiche gilt für die Behandlung der Ton- und Farbbefehle.

Das im Material verwendete BASIC ist so gehalten, daß es für die meisten BASIC-Dialekte direkt verwendbar ist. In erster Linie sind die Programme für die Rechner KC 85/2 und KC 85/3 geschrieben. Wenn von wenigen Besonderheiten (Bildschirmformat, Grafik und Ton) abgesehen wird, sind sie auch auf dem KC 85/1 bzw. KC 87 direkt lauffähig.

Horst Völz und Ursula Grote

1. Der Kleincomputer als Taschenrechner

Mit dem breiten Einsatz der Mikrorechentechnik brauchte man eine einfach zu erlernende und technisch mit geringen Mitteln zu verwirklichende Programmiersprache. BASIC erfüllt diese Anforderungen. 70 % der in der Welt vorhandenen Kleincomputer beherrschen BASIC, d. h., sie können alle in BASIC geschriebenen Programme in Ihren Computer mit nur geringen Veränderungen eingeben und nutzen.
Grundsätzlich wird bei allen BASIC-Versionen im Dialog mit dem Benutzer gearbeitet. Es gibt immer den Direktbetrieb mit Kommandos, bei dem alle Tastatureingaben sofort auf dem Bildschirm ausgegeben werden, und den Programmbetrieb, bei dem ein im Speicher stehendes Programm abgearbeitet wird.
Wenn wir behaupten, daß Ihr Computer nicht einmal addieren oder multiplizieren, geschweige denn einen bestimmten Sinuswert berechnen kann, so werden Sie dies sicherlich nicht ohne weiteres hinnehmen.
Diese Aussage wird jedoch durchschaubar, wenn man das Grundkonzept eines Computersystems analysiert. Das Herzstück jedes Computers ist die CPU (Central Prozessor Unit). Dieser Baustein ist der Teil des Computers, der alle Anweisungen ausführt. Allerdings nicht BASIC-Anweisungen, sondern sehr viel einfachere Maschinensprachbefehle, die u. a. aus der Addition und Subtraktion von ganzen Zahlen, Speicheroperationen, gewissen logischen Operationen u. ä. bestehen. Damit die CPU BASIC-Anweisungen ausführen kann, muß ein Übersetzer, d. h. ein Programm, vorhanden sein, der diese Befehle auf der Ebene der Maschinensprache ausführt. Diesen Übersetzer nennt man BASIC-Interpreter.
Im allgemeinen braucht sich der Progammierer nicht darum zu kümmern, wie sein Interpreter arbeitet, also in welche Befehle der Maschinensprache seine BASIC-Anweisungen übersetzt werden. Findet der Interpreter beim Progammlauf einen Syntaxfehler, so bricht er den Programmlauf ab. Auf dem Bildschirm werden die Art und, wenn möglich, der Ort des Fehlers gemeldet.

1.1. Die Entertaste und der PRINT-Befehl

Kommen wir nun zu der eingangs gestellten Behauptung zurück, daß ein Computer weder addieren noch multiplizieren noch einen Sinuswert berechnen kann.
Unser Computer wartet darauf, das auszuführen, was wir ihm anweisen.
Als höflicher Mensch versuchen wir es mit der Eingabe:

Guten Tag ⟨ENTER⟩

Die Anzeige der Wortkombination erscheint auf dem Bildschirm, aber nicht mehr. Damit der Befehl ausgeführt wird, müssen Sie nun die ENTER-Taste betätigen. Nach Ausführung dieses Befehls erscheint auf Ihrem Bildschirm:

?SN ERROR

Wir haben eine Fehlermeldung vom Rechner erhalten. Die Fehlermeldung hat im Direktbetrieb folgendes Format:
?xx ERROR
(xx-Fehlercode)
Eine Liste von Fehlermeldungen finden Sie im Anhang. An dieser Stelle wollen wir nur auf Fehlermeldungen bzgl. der Handhabung der PRINT-Anweisung eingehen.

SN
d. h.,
der R
Besinnen ... auf, daß es sich u ... einen Rechner handelt. Also versuchen wir es wie bei einem Taschenrechner mit einer einfachen Rechenaufgabe.
Wir geben in den Rechner ein:

(4+5)*3 ⟨ENTER⟩

Der Rechner antwortet mit:

?SN ERROR

Auch das versteht der Rechner nicht, er kann offensichtlich weniger als ein herkömmlicher Taschenrechner.
Versuchen wir es mit einer Zahl.
Wir geben in den Rechner ein:

4 ⟨ENTER⟩

Rechnerantwort:

?UL ERROR

Dies bedeutet undefined line. Es wurde eine nicht existierende Zeilennummer eingegeben.
So einfach ist also die Kommunikation mit einem Computer nicht. Mit Ihren Eingaben konnte entweder der Interpreter oder der Computer im Maschinencode nichts anfangen. Also lag der Fehler bei Ihnen.
Uns fehlt offensichtlich noch so etwas wie eine Ausgabe. Wir erinnern uns jetzt daran, daß es einen Befehl „PRINT" (drucke) gibt.
Wir geben in den Rechner ein:

PRINT 4 ⟨ENTER⟩

Rechnerantwort:

4
OK

Wir versuchen es weiter und geben in den Rechner ein:

PRINT (4+5)*3 ⟨ENTER⟩

Rechnerantwort:

27
OK

Unser Kleincomputer führt uns also bei Anwendung der entsprechenden Anweisungen auch die mathematischen Operationen aus, die wir von unserem Taschenrechner gewohnt sind. Er besitzt jedoch auch einige Besonderheiten gegenüber der üblichen mathematischen Schreibweise:

- Wir müssen stets den Buchstaben O und die Ziffer Ø (Null) unterscheiden.
  Aus diesem Grund wird die Null durchgestrichen.
- Führende Nullen vor dem Komma können weggelassen werden.
- Bei der Zahlendarstellung wird die englische Schreibweise gewählt. Wir schreiben also einen Dezimalpunkt statt eines Dezimalkommas.

– Bei der Exponentendarstellung schreiben wir statt „mal zehn hoch" einfach E.

Nun einige Beispiele:

Mathematische Schreibweise	Computer-Schreibweise
3,1	3.1
0,12	.12
$2{,}81 \cdot 10^{-15}$	2.81E–15
$8{,}06 \cdot 10^{13}$	8.06E13
6^{12}	6^12

1.2. Konstanten

Konstanten sind festgelegte, unveränderliche Werte.
Eine sehr bekannte Konstante ist die Zahl π (Rechnerschreibweise PI).
Geben Sie bitte die Konstante PI in den Rechner ein:

PRINT PI ⟨ENTER⟩

Rechnerausdruck:

```
3.14159
OK
```

Der Interpreter unseres Kleinrechners verarbeitet Zahlen und Zeichenketten als Konstanten. Als Zahlen (numerische Konstanten) stehen uns ganze Zahlen von −999999 bis +999999 sowie reelle Zahlen mit einem Betrag zwischen 9.40396E–39 bis 1.70141E+38 und Null zur Verfügung.
Nun versuchen wir es noch einmal, unseren Computer zu einer höflichen Begrüßung zu bewegen. Geben Sie in den Rechner ein:

PRINT "GUTEN TAG" ⟨ENTER⟩

Rechnerantwort:

```
GUTEN TAG
OK
```

Damit haben wir ein weiteres Erfolgserlebnis erreicht. Den in Anführungszeichen gesetzten Text nennen wir Zeichenkette. Sie ist eine Aneinanderreihung beliebiger Zeichen und wird in der Fachsprache als String bezeichnet.

1.3. Variablen

Variablen sind, wie schon der Name sagt, veränderliche Größen. Zum besseren Verständnis stellen wir uns vor, unser Computer besitzt eine Menge von Fächern, die Zahlen oder Wörter enthalten können. Den Fächern können wir Namen geben. Diese Namen kennzeichnen unsere Variablen.
Nehmen wir einmal an, daß der Radius eines Kreises 35 cm beträgt, und Sie wollen sich das merken.
Geben Sie ein:

RADIUS = 35 ⟨ENTER⟩

Der Computer hat nun als erstes ein Fach reserviert, in dem Sie ihre Zahl speichern können, und zweitens gab er diesem Fach den Namen RADIUS, damit Sie es später wiederfinden. Diese Verbindung von Speicherplatz und Kennzeichnung wird Variable genannt. Als dritte Aktion hat er die Zahl 35 in dem Fach gespeichert: man sagt, daß er der Variablen mit dem Namen RADIUS den Wert 35 zugewiesen hat (Wertzuweisung). Da der Wert von Radius eine Zahl ist, wird sie als numerische Variable bezeichnet.
Wollen Sie nun erfahren, wie groß der Radius ist?
Schreiben Sie:

PRINT RADIUS ⟨ENTER⟩

Der Rechner antwortet mit:

```
35
OK
```

Das bisher Gelernte wollen wir zur Verbesserung unseres Druckbildes (auf dem Bildschirm) heranziehen.
Wir geben in unseren Rechner ein:

PRINT "RADIUS ="; RADIUS ⟨ENTER⟩

Rechnerausdruck:

```
RADIUS = 35
OK
```

Fragen wir also optimistisch weiter nach dem Umfang eines Kreises. Der mathematische Zusammenhang zwischen Umfang und Radius ist formelmäßig gegeben durch die Beziehung: U=2*PI*r
Daraus folgt in der Computerschreibweise:

PRINT "UMFANG ="; RADIUS*2*PI ⟨ENTER⟩

Rechnerantwort:

```
UMFANG = 219.912
OK
```

Ein weitere Möglichkeit der Bildschirmausgabe ist durch die folgende Darstellung gegeben.
Wir geben in den Computer ein:

PRINT "RADIUS", "UMFANG": PRINT RADIUS, RADIUS*PI*2 ⟨ENTER⟩

Rechnerausgabe:

```
RADIUS        UMFANG
35            219.912
OK
```

Mit dem Doppelpunkt wird nach dem ersten PRINT-Befehl indirekt eine neue Zeile erzeugt, und dann folgt in der zweiten Zeile der Text, der hinter dem Doppelpunkt steht.
Durch dieses Beispiel werden zwei neue Möglichkeiten veranschaulicht:
Mit dem Doppelpunkt können mehrere Anweisungen an den Interpreter gegeben werden, d. h., es können somit mehrere Anweisungen auf eine Zeile geschrieben werden. Durch die Angabe von mehreren Variablen in der PRINT-Anweisung können mehrere Werte mit einer Anweisung dargestellt werden, es entsteht damit eine sog. PRINT-Liste.
Testen wir nun, ob die Variablen wirklich so lang sein müssen.

Wir versuchen es mit der Rechnereingabe:

```
PRINT R (ENTER)
```

Rechnerantwort:

```
Ø
OK
```

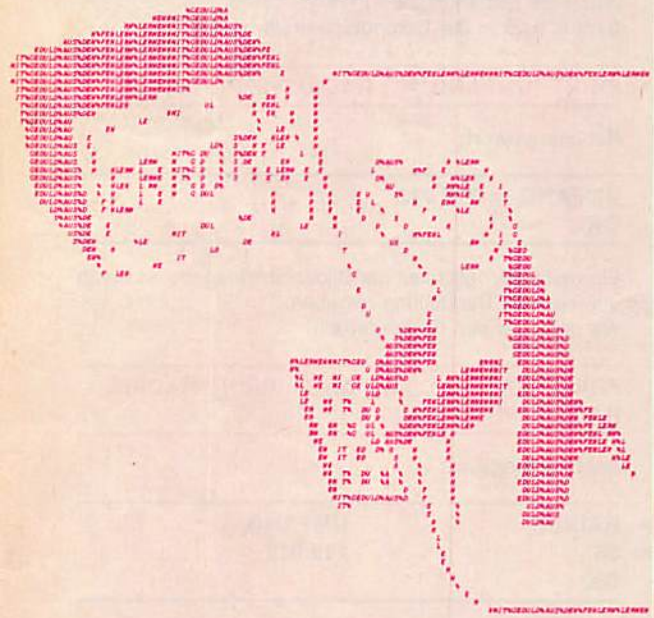
Ein Buchstabe reicht also nicht aus.
Versuchen wir es weiter:

```
PRINT RA (ENTER)
```

Der Rechner antwortet mit

```
35
OK
```

Mit weiteren Versuchen stellen wir fest:
Zwei Buchstaben reichen zur Kennzeichnung des Namens einer Variablen aus. Unser BASIC-Interpreter prüft nur die ersten beiden Zeichen eines Namens. Sie würden also keinen Unterschied zwischen RADIUS, RA, RAD oder gar RADIO und RADIANT bemerken.
Es gibt auch andere BASIC-Versionen, die längere oder kürzere Kennzeichnungen der Variablen verwenden.



1. 4. Die Grundrechenarten

Mit Zahlen und den von Ihnen definierten Variablen können Sie nun nach Belieben rechnen. Geben Sie z. B. ein:

Falls Sie den Wert für RADIUS noch nicht wieder neu festgelegt haben, werden Sie eben als Ergebnis 33.208 erhalten haben.
Bei allen Rechnungen ist zu beachten, daß der BASIC-Interpreter ihm übertragene Aufgaben in einer genau festgelegten Reihenfolge verarbeitet, d. h., er gibt bestimmten Operationen Vorrang vor anderen. Er weiß natürlich auch, daß Punkt- vor Strichrechnung geht. Das bedeutet, daß er, wie z. B. in der letzten Anweisung, erst die Multiplikation 5*PI und die Division RADIUS/5 ausführt. Danach wird addiert bzw. subtrahiert.
Sie können aber selbstverständlich bestimmten Operationen durch Klammern den Vorrang geben, z. B.:

```
PRINT 5*PI+(RADIUS+10.5)/5 (ENTER)
```

Nun erhalten Sie ein anderes Ergebnis als vorhin, da wir Klammern gesetzt haben, nämlich 24.808.
Unsere Aufgabe wurde wie folgt abgearbeitet:
1. Klammerinhalt berechnen
2. Multiplikation und Division ausführen
3. Addieren bzw. subtrahieren.

Beachten Sie, daß das Multiplikationszeichen „*" auch bei Klammerrechnungen und bei Variablen angegeben werden muß.

Zusammenfassung Kapitel 1

1. Der Rechner nimmt die Eingaben auf einer Bildschirmzeile entgegen. Die Zeile ist mit der ENTER-Taste abzuschließen. Auf einigen Rechnern heißt dieses Kommando auch RETURN, CR (Carrigde Return = Wagenrücklauf) oder NEW LINE (= neue Zeile).
2. Nur mit Befehl PRINT gibt der Rechner Texte auf dem Bildschirm aus.
3. Nach der Eingabe von für den Rechner Unverständlichem antwortet er mit einem Fragezeichen sowie zwei Buchstaben, die die Art des Fehlers kennzeichnen. Dann kommt die Mitteilung ERROR, zu deutsch: Fehler.
4. Texte, die in Anführungszeichen gesetzt sind, gibt der Rechner nach PRINT exakt wieder. Diese Texte heißen STRING, zu deutsch Zeichenkette.
5. Texte, die nicht in Anführungszeichen gesetzt werden, sind Variablen. Sie müssen mit einem Buchstaben beginnen. Variablennamen dürfen keine reservierten Worte enthalten, die schon mit einer festen Bedeutung in BASIC benutzt werden.
6. Obwohl eine Variable beliebig lang sein kann, unterscheidet der Computer nur die ersten beiden Zeichen einer Variablen. So kann er z. B. die Variablen RA, RAD, RADIO und RADIANT nicht unterscheiden.
7. Verwendet man eine Variable mit nur einem Buchstaben, so setzt der Rechner automatisch an diesen Buchstaben ein Leerzeichen. Das Leerzeichen heißt auf englisch space, also Zwischenraum.
8. Der Variablen wird der Zahlenwert durch ein Gleichheitszeichen zugewiesen. Zum Beispiel RA=35 bedeutet, daß der Variablen mit dem Namen RA der Wert 35 zugewiesen wird.
9. Strings und Variablen kann man unmittelbar hintereinander schreiben, wenn sie durch ein Semikolon getrennt werden.
10. Ein Komma in einer PRINT-Anweisung setzt das folgende Zeichen 13 Stellen weiter rechts auf den Bildschirm.
11. Mit einem Doppelpunkt können wir in einer Zeile zwei PRINT-Anweisungen verwenden, die dann auf zwei nacheinanderfolgende Zeilen Bezug haben.

2. Programmieren mit dem Computer

2. 1. Anwendung als Textspeicher

Ein großer Vorteil unseres Computers gegenüber einem Taschenrechner ist die Möglichkeit der Textverarbeitung.
Geben Sie ein:

PRINT "HERMANN HESSE: DER STEPPENWOLF" ⟨ENTER⟩ 1)

Sie sehen, die Zeichenkette innerhalb der Anführungszeichen wird auf dem Bildschirm geschrieben. Wir können so beliebige Zeichenketten mit Hilfe der PRINT-Anweisung in Verbindung mit den Anführungszeichen ausdrucken lassen. Diese Arbeitsweise bezeichnet man als Direkt-Mode. Wollen wir den obigen Text wiederholt ausgeben lassen, so sind wir gezwungen, die obige Anweisung jeweils neu einzugeben.
Hier kommt uns die Möglichkeit der BASIC-Programmierung entgegen. Wir wollen erreichen, daß der Rechner sich diesen Text merkt und dieser Text jederzeit abrufbar ist. Dazu besitzt der Rechner einen anderen Mode, den sog. Eingabe-Mode. Hierbei schaltet der Rechner zwischen zwei Zuständen um. Diese Umschaltung erfolgt bei BASIC automatisch, und zwar immer dann, wenn am Anfang der Zeile eine Zahl steht. Dann wird der Eingabe-Mode erreicht, und der folgende Text wird in den Rechner beständig abgelegt. Diese Zahl kann im Bereich von Ø und 65535 frei gewählt werden. Sie weist ähnlich wie der Name einer Variablen auf einen Speicherplatz im Rechner hin. Da unter jeder dieser Zahlen genau eine Zeile Text abgelegt werden kann, heißt die Zahl Zeilennummer.
Wir geben ein:

1) Aus drucktechnischen Gründen mußte die Zeile geteilt werden. Das geschieht im folgenden öfter. ⟨ENTER⟩ steht am Zeilenende.

10 "HERMANN HESSE: DER STEPPENWOLF" ⟨ENTER⟩

Zu unserem Erstaunen passiert jetzt gar nichts, nicht einmal das gewohnte OK erscheint. Wo ist also der Text geblieben?
Wir versuchen es mit der Eingabe:

PRINT 10 ⟨ENTER⟩

Die Rechnerantwort lautet:

```
10
OK
```

Das gleiche Ergebnis erhalten wir mit der Eingabe::

PRINT "10" ⟨ENTER⟩

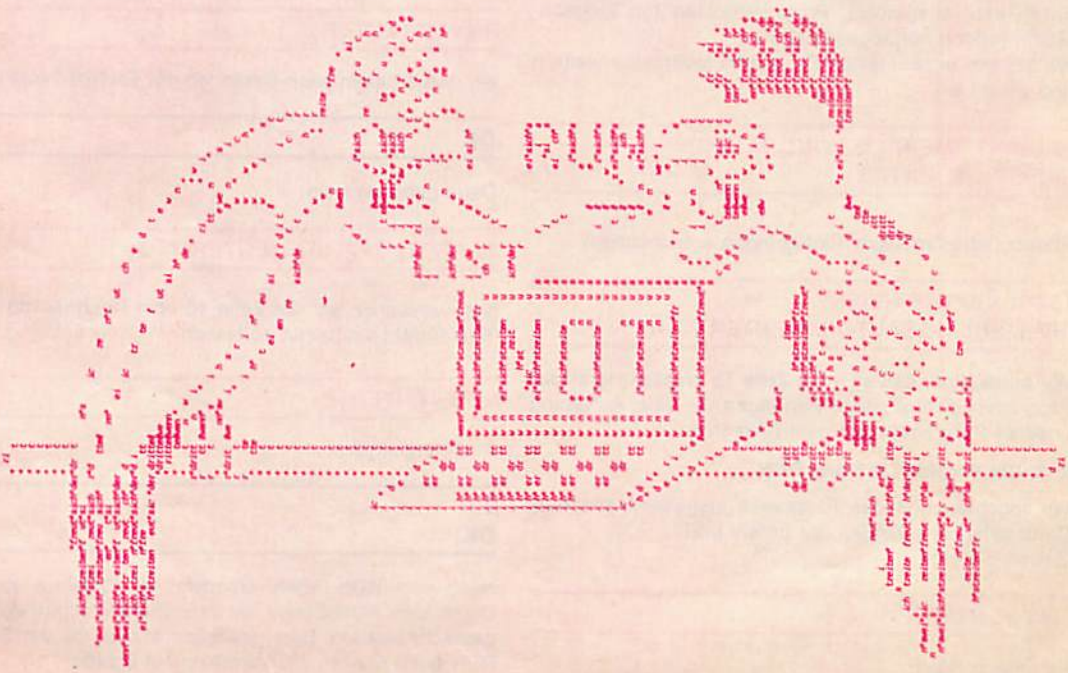
Auch dieser Versuch führt nicht zum Erfolg. Wir brauchen also einen neuen Befehl: den LIST-Befehl. Mit diesem Befehl wird das im Speicher befindliche Programm aufgelistet.
Rechnereingabe:

LIST ⟨ENTER⟩

Auf dem Bildschirm erscheint jetzt das folgende:

```
10 "HERMANN HESSE: DER STEPPENWOLF"
OK
```

Der Text ist also in der Zeile 10 abgelegt, und wir können ihn mit dem LIST-Befehl jederzeit abrufen.
Damit haben wir eine einfache Möglichkeit zum Speichern von Texten kennengelernt. Hiermit ist im Prinzip sogar eine einfache Textverarbeitung möglich.



Neben dem Speichern von Texten soll das Gespeicherte auch als Programm nutzbar sein. Dann benötigen wir den Befehl RUN. Dieser Befehl heißt soviel wie laufe, tue, was ich dir in den Zeilen befohlen habe. Er bewirkt, daß der BASIC-Interpreter von der Kommandoebene in die Programmebene übergeht. Er versucht also, den Text der Zeilen als Befehl an das Betriebssystem zu übersetzen.
Da der vorliegende Text kein Programmtext ist, muß in diesem Fall das RUN-Kommando zu einer Fehlermeldung führen.
Trotzdem wollen wir die Eingabe tätigen:

```
RUN ⟨ENTER⟩
```

Die Antwort lautet:

```
?SN ERROR ON 10
```

Das heißt, der Interpreter hat in der Zeile 10 einen Fehler entdeckt, der den Programmablauf nicht ermöglicht. Er hört daher auf zu arbeiten.

2. 2. Das Programm

Jetzt dürfen Sie endlich ein lauffähiges Programm schreiben. Versuchen wir also ein Programm zu entwikkeln, das den obigen Text ausdruckt.
Geben Sie ein:

```
10 PRINT "HERMANN HESSE:
DER STEPPENWOLF" ⟨ENTER⟩
```

Der Computer hat die Anweisung offensichtlich nicht ausgeführt. Das ist richtig, denn der PRINT-Befehl bekam eine Nummer (die 10), deshalb führte ihn der Computer nicht gleich aus, sondern speicherte ihn für später. Eine Reihe solcher gespeicherter Anweisungen nennen wir ein Programm.
Das ist die zweite wesentliche Funktion der ENTER-Taste: Kommandos (im Direkt-Mode) werden mit Hilfe der Enter-Taste ausgeführt; Programmzeilen (im Eingabe-Mode) jedoch nur abgespeichert.
Wir können unser kleines Programm jederzeit erweitern und geben ein:

```
20 PRINT "MEIN LIEBLINGSAUTOR
HEISST..." ⟨ENTER⟩
```

Ebenso ist es möglich, Einfügungen vorzunehmen.

```
15 PRINT "WIE HEISST
IHR LIEBLINGSAUTOR" ⟨ENTER⟩
```

Wir behaupten, daß sich die Zeile 15 problemlos in das Programm einfügt. Dazu benötigen wir eine Auflistung unseres Programms auf dem Bildschirm.

2. 3. Die Befehle LIST und RUN

Wir möchten uns das Programm insgesamt ansehen. Dafür existiert in BASIC der Befehl LIST.
Geben Sie ein:

```
LIST ⟨ENTER⟩
```

Rechnerantwort:

```
10 PRINT "HERMANN HESSE:
DER STEPPENWOLF"
15 PRINT "WIE HEISST IHR
LIEBLINGSAUTOR?"
20 PRINT "MEIN LIEBLINGSAUTOR
HEISST..."
```

Wir schauen uns an Hand unseres Beispiels noch einmal an, was eben auf dem Bildschirm passierte. Das Programm ist in der richtigen Reihenfolge dargestellt: die Programmzeilen werden also vom Computer entsprechend ihrer Zeilennummer aufgelistet.
Damit das Programm nun ausgeführt wird, müssen Sie RUN eingeben. Es erscheint nur der Text auf dem Bildschirm; der Computer meldet sich mit OK, und das Programm ist abgearbeitet. Dieses Programm können Sie nun beliebig oft mit der Anweisung RUN starten und ablaufen lassen.
Die Anweisung RUN bewirkt also die Abarbeitung eines Programms. Beginnend mit der niedrigsten Zeilennummer, werden die Programmzeilen in aufsteigender Reihenfolge abgearbeitet.

2. 4. Die Befehle NEW, INPUT und GOTO

Nun wollen wir aber ein „wirkliches" Programm, welches Berechnungen ausführt, d. h., allgemein gesprochen, Information verarbeitet, schreiben und dazu unseren Speicher verwenden. Falls wir vorher schon Zeilen in den Speicher eingegeben haben, so müssen wir diese zunächst löschen, damit wir den Speicher frei haben für unser Programm. Hierzu gibt es einen Befehl, der NEW lautet. Der Befehl NEW löscht alle im Speicher befindlichen Programme.
Wir geben NEW ein, und es erscheint das bekannte OK. Und nun der Test mit dem LIST-Kommando: es kommt ebenfalls nur OK, der Speicher ist also leer; d. h., keine einzige Programmzeile ist mehr im Speicher vorhanden.
Jetzt wollen wir mit dem Computer wirklich eine kleine Rechenaufgabe lösen.
Wir geben zunächst im Direkt-Mode den Radius

```
RA=35 ⟨ENTER⟩
```

ein und erhalten nach Betätigen der ENTER-Taste das

```
OK
```

Dann geben wir ein:

```
10 PRINT 2*PI*RA ⟨ENTER⟩
```

Nun versuchen wir die Zeile 10 vom Rechner mit dem RUN-Befehl ausführen zu lassen.

```
RUN ⟨ENTER⟩
```

Rechnerausgabe:

```
0
OK
```

Nach dem RUN-Befehl erscheint als Ergebnis eine 0. Durch den RUN-Befehl ist also offensichtlich RA zu 0 gemacht worden. Dies geschieht immer, da der Befehl RUN beim Starten alle Variablen auf 0 setzt.

Wenn wir also unseren Radius zu 35 realisieren wollen, müssen wir eine Zeilennummer für diese Anweisung reservieren. In gewohnter Weise fügen wir jetzt die Zeile 5 vor die Zeile 10 ein, indem wir eingeben:

```
5 RA = 35 (ENTER)
```

Nach Ausführung des RUN-Befehls erhalten wir nun das richtige Ergebnis: 219.912.

Unser erstes Programm ist erfolgreich gelaufen. Doch es gefällt uns noch nicht, daß wir für jeden anderen Radius die Zeile 5 ändern müssen.

Glücklicherweise gibt es hierfür ebenfalls einen Befehl. Dieser Befehl heißt INPUT, was soviel wie Eingabe heißt. Wir überschreiben unsere ursprüngliche Zeile 5 mit der Anweisung:

```
5 INPUT RA (ENTER)
```

Wenn wir jetzt den LIST-Befehl ausführen, dann ist in dem Rechner enthalten:

```
5 INPUT RA
10 PRINT 2*PI*RA
OK
```

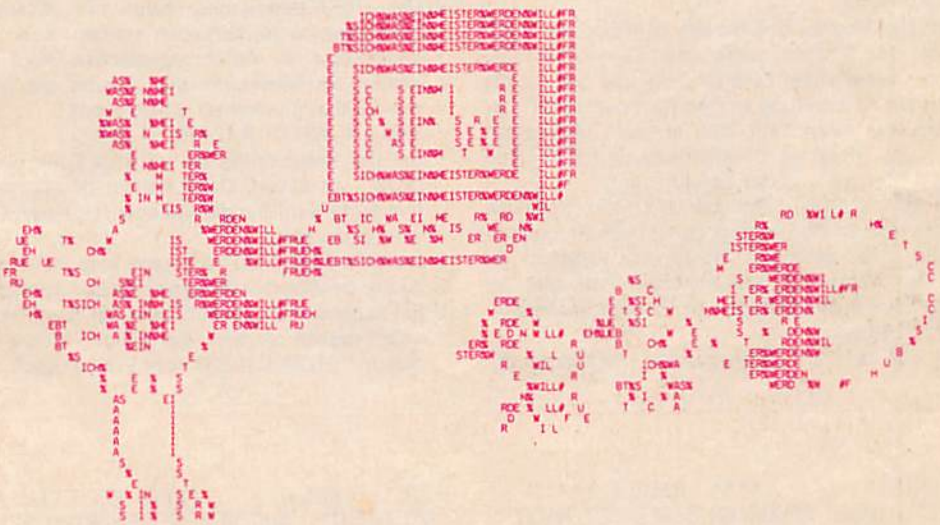
Nach Ausführung des RUN-Befehls erscheint ein Fragezeichen und dahinter der Cursor. Der Rechner fragt, welchen Radius wir eingeben wollen. Wir geben eine Zahl für den Radius ein und beenden mit dem Druck auf die ENTER-Taste die Eingabe. Der Computer legt den eingegebenen Wert unter der hinter INPUT stehenden Variablen ab. Von nun an kann man den eingegebenen Wert im Programm nach Belieben durch Benutzen der Variablen zu Rechnungen verwenden.

Wir können jetzt mit jedem neuen Start von RUN einen neuen Radius eingeben und erhalten sofort den neuen Umfang des Kreises. Unbequem ist an dieser Stelle jedoch noch, daß wir jedesmal neu mit RUN beginnen müssen.

Es wäre ja viel einfacher (insbesondere dann, wenn wir sehr oft den Radius in den Umfang umwandeln wollen), wenn wir sofort wieder zur neuen Eingabe des INPUT-Befehls kämen.
Hierzu gibt es den GOTO-Befehl, was soviel heißt wie: gehe zur Zeile mit einer festgelegten Zeilennummer.
An das Ende unseres Programmes fügen wir jetzt die Zeile 20 an mit dem Befehl GOTO 5.
Der LIST-Befehl zeigt uns folgendes Ergebnis:

```
5 INPUT RA
10 PRINT 2*PI*RA
20 GOTO 5
OK
```

Und nun lassen wir unser um die Zahl 20 erweitertes Programm mit der Anweisung RUN ablaufen. Es entsteht eine endlose Kette: der Rechner fragt nach dem Radius – wir geben eine Zahl ein – danach folgt das Ergebnis – der Rechner fragt erneut nach dem Radius – wir können wieder eine Zahl eingeben – der Rechner gibt uns wieder das Ergebnis.
Dieses Spiel hört nie auf, es sei denn, wir schalten den Rechner ab. Das ist unbequem, wir benötigen eine



Unterbrechung unseres Programms. Unterbrechen heißt im Englischen „break" (BRK-Taste). Wir betätigen also die BRK-Taste unseres Computers. Dadurch wird der Programmablauf unterbrochen, und der BASIC-Interpreter meldet sich mit

```
BREAK IN 5
OK
```

(oder BREAK IN 10 bzw. 20, je nachdem, in welcher Programmzeile mehr oder weniger zufällig die Abarbeitung unterbrochen wurde).
Jetzt testen wir, ob unser Programm noch im Rechner ist. Wir geben den Befehl LIST ein, und siehe da: unser Programm ist noch vorhanden.
Damit haben wir eine Möglichkeit kennengelernt, unser Programm einmal ständig laufen zu lassen, und zum anderen, mit der BRK-Taste zu unterbrechen.
In größeren Programmen können wir im Programmab-

lauf nicht feststellen, für welche Variable wir einen Wert eingeben. Günstiger ist es, wenn sich unser Computer während des Programmablaufes bei der INPUT-Anweisung mit einer textlichen Eingabeforderung meldet. Das ist möglich. Wir müssen beim INPUT-Befehl genauso wie beim PRINT-Befehl lediglich den Text, den er ausgeben soll, unmittelbar in Anführungszeichen folgen lassen; d. h., wir müssen schreiben:

```
5 INPUT "RADIUS ="; RA   ⟨ENTER⟩
7 PRINT "UMFANG =";      ⟨ENTER⟩
```

Lassen Sie nun das Programm ablaufen. Sie sehen, ein durch Anführungszeichen erzeugter String direkt nach der INPUT-Anweisung wird im Programmablauf entsprechend ausgedruckt. Dieser nach Belieben einzufügende String ist von der folgenden Variablen in jedem Fall durch ein Semikolon abzutrennen. Durch die Hinzunahme des PRINT-Befehls können wir ebenso unsere Ergebnisanzeige komfortabler gestalten.

Zusammenfassung Kapitel 2

1. Ein Programm besteht aus Programmzeilen mit Zeilennummern. Eine Programmzeile wird durch Betätigen der ENTER-Taste gespeichert. Für die Ordnung der Ablage gilt hierbei die Reihenfolge der Zahlen. Deshalb ist es sinnvoll, die Zeilennummerierung beispielsweise in Zehnerabständen zu wählen. Dadurch ist es möglich, hinterher etwas einzufügen.
2. Der im Speicher abgelegte Text kann mit dem Befehl LIST auf dem Bildschirm wieder sichtbar gemacht werden.
3. Nach der Eingabe des Befehls RUN liest der Interpreter den Text der numerierten Zeilen der Reihe nach, übersetzt ihn Zeile für Zeile und gibt ihn danach zur Abarbeitung an den Rechner. Findet der Interpreter einen Text, den er nicht übersetzen kann, so erfolgt eine Fehlermeldung mit Angabe des Fehlertyps und der Zeilennummer.
4. Der Befehl RUN setzt zunächst alle Variablen gleich Null. Das ist wichtig, damit beim Programmablauf genau definierte Verhältnisse vorliegen. Es ist also notwendig, alle benutzten Variablen im Programm selbst mit den benötigten Zahlenwerten zu belegen.
5. Soll ein Text im Zeilenspeicher vollständig gelöscht werden, so erfolgt dies mit dem Befehl NEW.
6. Die Eingabe von Daten während des Programms kann mit der INPUT-Anweisung erfolgen. Es ist möglich, mehreren, durch Kommata voneinander getrennten Variablen Werte mit einer INPUT-Anweisung zuzuordnen. Der Computer meldet sich mit einem Fragezeichen und erwartet die Dateneingabe. Erfolgen nicht die vereinbarten Eingaben (z. B. String anstatt numerischer Werte), so meldet sich der Computer: ?REDO FROM START, und die Eingabe muß wiederholt werden.
7. Die INPUT-Anweisung kann mit einem Text (Stringkonstante) versehen werden, wenn dieser unmittelbar in Anführungszeichen folgt. Dann kommt ein Semikolon und dahinter die Variable, welche den Zahlenwert erhalten soll.
   Beispiel: INPUT "RADIUS ="; RA
   Bei der Abarbeitung ist auf dem Bildschirm RADIUS= zu sehen. Dann kommt der Cursor, der uns auffordert, für die Variable RA einen Zahlenwert einzugeben.
8. Mit dem Befehl GOTO n kann man im Programm zu der bestimmten Programmzeile n springen.
9. Ein laufendes Programm kann mit der BRK-Taste unterbrochen werden. Bei einigen Rechnern ist dies mit STOP, CTRL C oder ESC möglich.

```
10 CLS:PRINT"        ****  RADIUS ****"
20 PRINT" DIESES PROGRAMM BERECHNET NACH"
30 PRINT"       EINGABE DES RADIUS"
40 PRINT"   UMFANG, FLAECHE UND VOLUMEN"
50 INPUT"RADIUS              =";RA
60 PRINT"UMFANG  VOM KREIS =";2*PI*RA
70 PRINT"FLAECHE VOM KREIS =";PI*RA*RA
80 PRINT"VOLUMEN DER KUGEL =";PI*RA*RA*4/3
90 PRINT:GOTO 50
```

```
10 CLS:PRINT"     ##### RECHTECK #####"
20 PRINT"  AUS DEN SEITEN WIRD BERECNET"
30 PRINT"UMFANG UND FLAECHE DES RECHTECKS"
40 PRINT:INPUT"SEITE A =";A
50 INPUT"SEITE B =";B
60 PRINT"UMFANG  =";2*(A+B)
70 PRINT"FLAECHE =";A*B
80 PRINT:GOTO 40
```

```
10 CLS:PRINT"          &&&&& SPIEL &&&&&":PRINT
20 PRINT"    ES SPIELEN  DREI SCHUELER MIT"
30 PRINT"     IMMER EINER GIBT AM RECHNER "
40 PRINT" OHNE ES DIE ANDEREN SEHEN ZU LASSEN"
50 PRINT" GEMAESS DER  FORDERUNG DES RECHNERS"
60 PRINT"  EIN MOEGLICHST WITZIGES WORT EIN"
70 PRINT"AM ENDE ZEIGT DER RECHNER DEN TEXT AN"
80 PRINT:PRINT"ARTIKEL UND SUBSTANTIV Z.B. DER MOND":INPUTA$
90 CLS:PRINT"ADJEKTIV, ALSO Z.B. ROT ODER SCHNELL":INPUTB$
100 CLS:PRINT"ADJEKTIV, ALSO Z.B. LIEB ODER GEBORGEN":INPUTC$
110 CLS:PRINTA$;" IST ";B$;" UND "C$
120 PRINT:INPUT"IRGEND EINE TASTE";D$:GOTO10
```

3. Variablen und der LET-Befehl

Variablen sind Ihnen schon in den Kapiteln 1 und 2 begegnet.
Daten können in einem Programm als Konstanten (feste Größen) und Variablen (veränderbare Größen) auftreten. Konstanten behalten während der gesamten Programmausführung ihren Wert, Variablen können im Verlaufe der Programmausführung verschiedene Werte annehmen.
Eine Variable können wir mit einem Speicher in einem Taschenrechner vergleichen. Unser Computer bietet jedoch den Vorteil, daß wir sehr viele solcher Speicher anlegen können.
Mit Hilfe der Anweisung LET können wir einer Variablen einen Wert zuweisen.

3.1. Der Begriff der Variablen

Variablen sind in der Mathematik veränderbare Größen. Zum besseren Verständnis stellen wir uns vor, unser Computer besitzt eine Menge von Fächern, die Zahlen oder Wörter aufnehmen können. Den Fächern können wir Namen geben. Diese Namen kennzeichnen unsere Variablen. Der Inhalt des Schubfaches selbst ist der Wert der Variablen. Dieser läßt sich durch entsprechende Computeranweisungen beliebig ändern.
Wir wollen einmal feststellen, wie viele solcher Variablen BASIC besitzt. Zunächst einmal sind alle Großbuchstaben zugelassen. Das sind von A bis Z gerechnet 26 Variablen. Dann kann hinter jedem Großbuchstaben noch eine der Ziffern 0, 1,...,9 folgen oder wiederum einer der 26 Buchstaben angeführt werden. Das sind 26 (10+26) also 936 Variablen. Zählen wir die 26 Buchstaben ohne Ergänzung hinzu, so ergibt dies 962 mögliche Variablen.
Wir haben also in der hier beschriebenen BASIC-Version 962 Variablen verfügbar; eine Anzahl, die selbst für große Programme kaum benötigt werden wird.
In der Praxis bestehen dennoch Probleme, mit diesen Variablenbezeichnungen umzugehen: Es kostet selbst den erfahrenen Programmierer oft Mühe, zu wissen, was er z. B. unter der Variablen KX versteht, was dort enthalten oder abgespeichert ist.
Aus diesem Grunde ist es zugelassen, längere Wörter als Variablenname zu verwenden, z. B. DURCHMESSER, RADIUS oder Z127. Der Rechner verwendet intern aber davon nur die ersten beiden Zeichen, nämlich DU, RA, oder Z1. Nun könnte man auch auf den Gedanken kommen, kleine Buchstaben zu verwenden: Dies bringt jedoch keine weitere Vielfalt, da der Rechner Kleinbuchstaben automatisch in große umwandelt.
Wie verhält es sich mit den Sonderzeichen wie Klammern, Multiplikations-, Plus-, Minus- und Ausrufezeichen? Dies wollen wir einmal am Computer ausprobieren.
Wir geben ein:

```
A!=7 (ENTER)
```

Rechnerausdruck:

```
?SN ERROR
```

Das heißt, diese Eingabe ist nicht zulässig. Es ist generell nicht erlaubt, Sonderzeichen in den Variablennamen zu verwenden. Versuchen wir noch etwas anderes.
Wir geben jetzt ein:

```
AT=7 (ENTER)
```

Auch hier erscheint der Ausdruck „? SN ERROR", d. h., die Variable AT existiert nicht; ihr kann demzufolge nicht der Wert 7 zugewiesen werden. Hierbei handelt es sich nämlich um ein sog. reserviertes Wort.
Es gibt eine Vielzahl von reservierten Worten, die in BASIC mit einer festen Bedeutung benutzt werden wie PRINT, GOTO, INPUT usw. Es gibt auch solche Anweisungen – wir kommen später auf sie zu sprechen –, die nur aus 2 Buchstaben bestehen, und zu ihnen zählt das AT. Bei der Verwendung von längeren Variablennamen ist diesen reservierten Wörtern noch viel größere Beachtung zu schenken.

Wegen AT sind so z. B. die folgenden Variablennamen unzulässig: ATOM, PATENT, MATHEMATIK.

3.2. Die Ergibt-Anweisung: LET-Befehl

Das „LET" kommt aus dem Englischen und heißt soviel wie setze, mache die Variable zu diesem Wert.
Die LET-Anweisung hat folgendes allgemeines Format:
LET Variable=Ausdruck, z.B.
LET A= 3*7+4
Die Wirkung der LET-Anweisung besteht darin, daß zunächst der Ausdruck berechnet und dann der errechnete Wert der Variablen zugewiesen wird. Die nachfolgenden Beispiele sollen die Wirkung der LET-Anweisung veranschaulichen.

Wir geben in den Rechner das folgende ein:

```
LET AG=25 (ENTER)
```

Der Variablen AG wird also der Wert 25 zugeordnet. In den letzten Jahren wurde dieser Befehl immer seltener verwendet. Es hat sich eingebürgert, einfach
AG=25
zu schreiben.
Bei vielen Rechnern benötigt der Interpreter jedoch für die letzte Form mehr Zeit, denn er muß zuerst ermitteln, ob es sich um eine Wertzuweisung zu einer Variablen handelt. Deshalb ist es günstig, den Rechner darauf zu prüfen, ob er mit LET schneller arbeitet.

Hierzu empfehlen wir Ihnen das folgende zweizeilige Programm (vgl. LET1):

```
10 LET X=1: LET A=1.01 (ENTER)
20 LET X=X*A: GOTO 20 (ENTER)
```

Dieses Programm läuft solange, bis ein „OVERFLOW-ERROR" erfolgt, d. h. bis die Zahl X so groß ist, daß sie der Rechner nicht mehr verarbeiten kann. Der Rechner meldet sich dann:

```
?OV ERROR
```

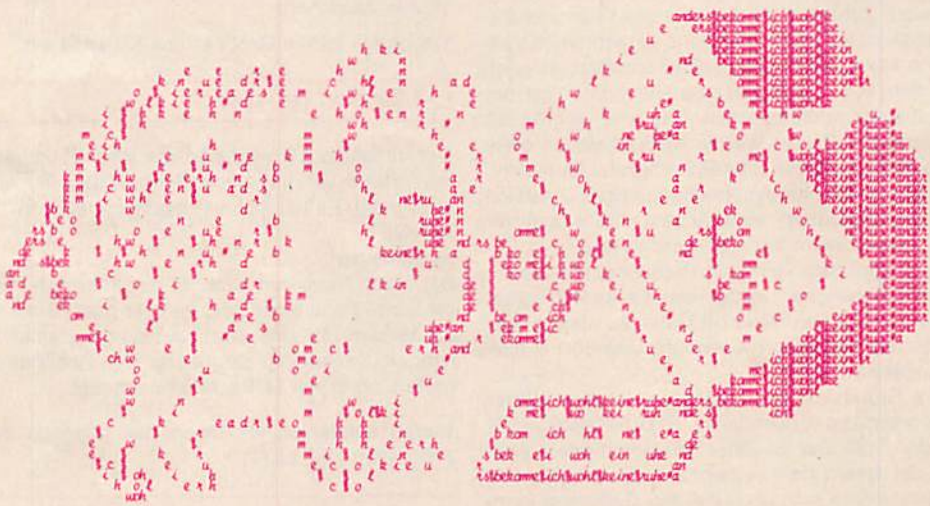
Der KC 85/3 benötigt bis zum OVERFLOW genau 58 s. Mit PRINT X kann man sich die größte Zahl, die er errechnet hat, ausdrucken lassen: sie beträgt in diesem Fall 8.57003 E+37. Die prinzipiell größte Zahl liegt beim KC 85/3 bei 1.70141 E+38.
Nun ändern Sie bitte die Zeile 20 in

```
20 X=X*A: GOTO 20 (ENTER)
```

und bestimmen erneut die Laufzeit. Sie wird bei vielen Rechnertypen oft erheblich größer sein. Beim KC 85/3 ist jedoch kein Zeitunterschied feststellbar. Für ihn kann

also fast immer ohne Bedenken die LET-Anweisung entfallen.
Die Variablen wurden zu Beginn des Kapitels mit geordneten Fächern verglichen. In das erste Fach legt der Rechner jene Variable ab, die im Programm zuerst angesprochen wird. Gleichermaßen tut er es mit allen Variablen in der Reihenfolge ihres Auftretens im Programm. Er arbeitet also die Wertzuweisung der Variablen nicht in alphabetischer Reihenfolge oder nach anderen ordnenden Prinzipien ab, sondern die Reihenfolge des ersten Aufrufs der Variablen im Programm ist entscheidend.
Bei jedem Aufruf eines Variablennamens testet der Rechner die Fächer in der ihm eigenen Reihenfolge; stößt er dabei auf die gewünschte Variable, so entnimmt er den dort vorhandenen Zahlenwert bzw. legt dort den neuen ab – findet er den Namen nicht, so eröffnet er für ihn ein neues Fach. Die Kenntnis dieses Mechanismus ist wichtig, wenn man schnelle Programme benötigt. Dann sind die häufig verwendeten Variablen zuerst zu verwenden (man spricht auch vom Initialisieren der Variablen). Zur Veranschaulichung des Gesagten möge das folgende Programm (s. auch LET2) dienen:

```
10 A=1: B=1.01: C=1: D=1: E=1: F=1:
G=1: H=1.01: X=1 ⟨ENTER⟩
20 A=A*B: GOTO 20 ⟨ENTER⟩
```



Dieses Programm benötigt, wie nicht anders zu erwarten, wiederum die 58 s.
Wird jedoch die Zeile 20 durch

```
20 H=H*X: GOTO 20 ⟨ENTER⟩
```

ersetzt, so benötigt die inhaltlich gleiche Aufgabe bereits 73 s.

3.3. Zahlendarstellung und Genauigkeit

Der verwendbare Zahlenbereich kann von Rechner zu Rechner recht unterschiedlich sein. Darüber hinaus sind mehrere Zahlenbereiche verwendbar (s. Programm ZAHL 1). Meist existieren drei Gruppen:
Die erste Gruppe besteht aus den ganzen Zahlen. Sie werden auch integer bzw. entier genannt. Wenn Ihr Rechner, wie beim KC 85 6stellige Zahlen verwendet, sind die Zahlen von −999 999 über −999 998 und −1, 0, 1 bis 999 999 nutzbar.
Daneben gibt es Zahlen mit Komma (in BASIC-Schreibweise Dezimalpunkt), z. B. 0.123456; 1.23456; 12.3456; 12345.6 usw. Sie erzeugt der Rechner in einem gewissen Zahlenbereich automatisch:

```
PRINT 12/7 ⟨ENTER⟩

 1.71429
```

oder

```
PRINT 1/15 ⟨ENTER⟩

 .0666667
```

In dieser Form werden Zahlen im Bereich von .01 bis 9.99999 dargestellt. Bei kleineren Zahlen geht der Rechner automatisch in die Exponentialdarstellung über, also

```
PRINT 1/134 ⟨ENTER⟩

 7.46269E-03
```

Dasselbe geschieht bei zu großen Zahlen, also

```
PRINT 123456*987654 ⟨ENTER⟩

 1.21932E+11
```

Die kleinste so darstellbare Zahl beträgt beim KC 85/3 9.40396E-39. Kleinere Zahlen rundet der Rechner automatisch zu 0, und das ganz ohne Warnung:

```
PRINT 9E-39/2 ⟨ENTER⟩

 0
```

Eine Zahl in Exponentialdarstellung benötigt folgende Positionen:
SX.XXXXXESYY
Hierin bedeutet S das mögliche Vorzeichen + oder −, wobei für + am Beginn ein Leerzeichen steht. Die X stehen für Ziffern der Mantisse und die Y für Ziffern des Exponenten.

Das sind genau 12 Positionen. Hinter jeder Zahl wird in BASIC noch ein Leerzeichen angefügt, so sind die 13

die meiste Zeit für die Bildschirmausgabe benötigt wird. Deshalb schreiben wir für die Zeituntersuchungen ein etwas modifiziertes neues Programm.
Es lautet:

```
10 FOR X=1 TO 1000 (ENTER)
20    FOR Y=1 TO 50 (ENTER)
30    NEXT (ENTER)
40 NEXT (ENTER)
```

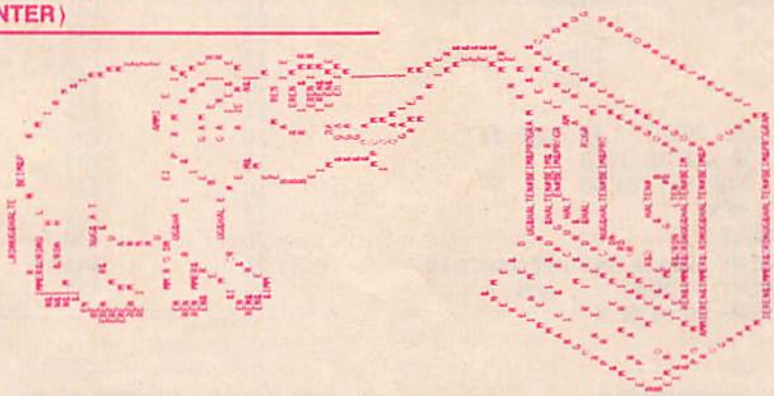
Lassen wir das Programm in dieser Form laufen, dauert es 105 s. Fügen wir Y bzw. X hinzu, so werden 120 s benötigt. Dies ist verwunderlich und erklärt sich wie folgt: Der Interpreter testet, wenn er NEXT erreicht, ob eine Variable folgt. Ist dies nicht der Fall, so setzt er voraus, daß es jene Variable ist, die beim letzten FOR verwendet wurde. Andernfalls übernimmt er die Variable hinter NEXT und sucht das dazugehörende FOR. Dieser zweite Fall dauert beim Interpreter des KC 85 länger. Es gibt aber auch Interpreter, bei denen es umgekehrt ist.

4.4. Art der Verschachtelung

Werden gleichzeitig mehrere Schleifen ineinandergeschachtelt, so sind einige Richtlinien zu beachten. Sie seien an folgendem Beispiel demonstriert:

```
10 FOR X=A TO B (ENTER)
20    FOR Y=C TO D (ENTER)
30    NEXT X (ENTER)
40 NEXT Y (ENTER)
```

Diese Anordnung ist nicht zulässig, da hier eine Überlappung bezüglich der zwei Schleifen existiert.
Nur ein sehr guter Interpreter erkennt diesen Fehler und antwortet darauf mit dem NF-NEXT ohne FOR-ERROR. Viele Interpreter liefern völlig falsche Ergebnisse.
Die Ursachen liegen darin, daß bei der Anweisung FOR u. a. die zugehörige Variable mit ihrem Wert auf den Stack gelegt wird. Die entsprechende Variable wird bei der NEXT-Anweisung geändert. Beispielsweise wird hier mit NEXT X der Wert von X geändert. Deshalb muß vorher das Y vom Stack entfernt werden. Es steht damit für die nachfolgende Anweisung NEXT Y nicht mehr zur Verfügung.
Es muß also immer erst jene Variable mit NEXT abgeschlossen werden, die zuletzt mit FOR verwendet wurde. Weiterhin ist zu beachten, daß für innere und äußere Schleifen unterschiedliche Laufvariablen benutzt werden müssen. Der Eintritt in eine FOR-NEXT-Schleife ist in jedem Fall nur über die FOR-Anweisung zulässig, d. h., Sprünge „in eine Schleife" sind nicht erlaubt.
Eine weitere Bemerkung betrifft die übersichtliche Schreibweise bei mehreren verschachtelten FOR-NEXT-Schleifen. Hier ist es günstig, nach den Zeilennummern bei jedem neuen FOR eine oder zwei Leertasten (space) einzufügen. Ebenfalls zurückzusetzen sind die dann folgenden Zeilen mit den entsprechenden Befehlen. Das zugehörige NEXT wird wieder auf den Abstand des entsprechenden FOR vorgezogen. So entsteht eine Struktur, die deutlich zeigt, welcher Textteil in einer bestimmten FOR-NEXT-Schleife steht.



Zusammenfassung Kapitel 4

1. Die Befehlsfolge FOR-TO-NEXT ermöglicht es uns, mit einem Zahlenwert, z. B. X, einen Zahlenbereich, z. B. von A bis B, zu durchlaufen. Die Befehlszeile dazu lautet:
   FOR X=A TO B
   Dabei wird der Variablen X als erstes der Zahlenwert von A zugeordnet. Danach, im zweiten Schritt, erhält X den Wert A+1 usw., bis der Wert von B erreicht ist.
   A und B sind beliebige Zahlenwerte. Sie können direkt oder über eine Variable im Programm festgelegt werden. Nach der FOR-TO-Anweisung steht, was gemäß diesen Zahlenwerten getan werden soll. In der Abschlußzeile steht die NEXT-Anweisung (s. Beispiel FOR1).
2. Hinter dem Befehl NEXT kann oder muß, je nach Interpreter, die Variable des zugehörigen FOR-Befehls stehen. Wenn diesbezüglich Freiheit besteht, existiert ein Zeitunterschied in der Ausführung beider Varianten. Oft ist der Fall ohne Variable schneller.
3. Es können in einem Programm mehrere FOR-NEXT-Schleifen ineinander gefügt werden. Der Abschluß der Schleife kann dann unterschiedlich aussehen:
   NEXT X: NEXT Y
   NEXT X, Y
   NEXT: NEXT
   Die beiden ersten Varianten sind gleichwertig.
4. Werden FOR-NEXT-Schleifen ineinandergeschachtelt, dann ist unbedingt darauf zu achten, daß die innere Schleife vollständig innerhalb der äußeren liegt. Die Laufbereiche dürfen sich also mit anderen Worten nicht überschneiden.
5. Wird mit NEXT eine weiter zurückliegende Variable aufgerufen, so werden nachfolgende andere Variablen vom Rechner nicht beachtet.
6. Es ist für die Lesbarkeit von Programmen nützlich, sie bezüglich der FOR-NEXT-Schleifen durch passend gewählten Abstand von den Zeilennummern übersichtlich zu gestalten.

```
0 REM: Programm: ## FOR1 ##
10 CLS:PRINT"Tabelle der Quadrat- und Kubikzahlen"
20 PRINT"Wert","Quadrat","Kubus"
30 FOR X=1 TO 25
40  PRINT X, X*X, X*X*X
50 NEXT
```

```
Tabelle der Quadrat- und Kubikzahlen
Wert          Quadrat       Kubus
 1             1             1
 2             4             8
 3             9             27
 4             16            64
 5             25            125
 6             36            216
 7             49            343
 8             64            512
 9             81            729
 10            100           1000
 11            121           1331
 12            144           1728
 13            169           2197
 14            196           2744
 15            225           3375
 16            256           4096
 17            289           4913
 18            324           5832
 19            361           6859
 20            400           8000
 21            441           9261
 22            484           10648
 23            529           12167
 24            576           13824
 25            625           15625
```

```
10 CLS: PRINT"  ## FOR3 ##"
20 FOR X=1 TO 1000
30   FOR Y=1 TO 50
40   NEXT
50 NEXT
60 REM: zweite Zeitmessung mit
70 REM: 30 NEXT Y ; und
80 REM: 40 NEXT X
```

```
0 REM: Programm: ## FOR2 ##
10 CLS: PRINT"Tabelle"
20 PRINT"Wert","Quadrat","Kubus"
30   FOR X=1 TO 35
40     PRINT"-";
50   NEXT: PRINT
60   FOR X=1 TO 22: I=1
70     FOR Y=1 TO 3: I=I*X
80       PRINT I,
90     NEXT
100  NEXT
110  FOR X=1 TO 35
120    PRINT"-";
130  NEXT
```

```
Tabelle
Wert          Quadrat       Kubus
-----------------------------------
 1             1             1
 2             4             8
 3             9             27
 4             16            64
 5             25            125
 6             36            216
 7             49            343
 8             64            512
 9             81            729
 10            100           1000
 11            121           1331
 12            144           1728
 13            169           2197
 14            196           2744
 15            225           3375
 16            256           4096
 17            289           4913
 18            324           5832
 19            361           6859
 20            400           8000
 21            441           9261
 22            484           10648
-----------------------------------
```

5. Die STEP-Anweisung

Wir haben in Kapitel 4 die FOR-TO-NEXT-Anweisung behandelt. Dabei wurde die Laufvariable, z. B. X, jeweils um 1 erhöht. Es handelt sich also um eine Zählschleife. Der Laufvariablen wird ein Anfangswert zugewiesen, dieser wird um die Schrittweite 1 erhöht, bis die Laufvariable den Endwert überschreitet. Wenn der Wert der Laufvariablen größer als der Endwert ist, wird zu der Programmzeile übergegangen, die auf die NEXT-Anweisung folgt.
In der FOR-Anweisung ist ein zusätzlicher Teil STEP notwendig, wenn die Schrittweite nicht den Wert 1 hat, sondern wenn wir z. B. auch das Quadrat von 1.5; 2.5 usw. benötigen.
Folglich lautet das Format jetzt:
FOR X=A TO B STEP C
Dann folgt die Anweisungsfolge, mit der die Änderungen durchgeführt werden. Den Abschluß bildet dann wieder NEXT.
Es wird jetzt also X zunächst mit A belegt, und jedesmal, wenn der Interpreter NEXT erreicht, wird X um den Wert von C erhöht. Wird dabei der Wert B überschritten, so verläßt der Interpreter die Schleife. Andernfalls wird sie weiter abgearbeiet.

Die neue Schleife ist folglich mit der alten, ohne STEP, gleichwertig, wenn C=1 gesetzt wird. Genau dann kann STEP entfallen.

5.1. Ganzzahlige Schrittweite

Soll der Wert der Laufvariablen um Schrittweiten ungleich 1 geändert werden, so ist die erweiterte Form der FOR-TO-NEXT-Anweisung mit der zusätzlichen Angabe der Schrittweite (STEP) zu verwenden. Die Schrittweite kann auch negativ sein.
Probieren wir es:

```
10 FOR X=4 TO 9 STEP-1 (ENTER)
20 PRINT X, (ENTER)
30 NEXT (ENTER)
```

Bei RUN wird nur 4 angezeigt.
Danach wird nämlich X um 1 verkleinert, nimmt also den Wert 3 an. Er ist aber kleiner als der Endwert 9. Dieses Ergebnis wird nur verständlich, wenn man weiß, daß bei negativen Schrittweiten der Interpreter den Wert nach TO, also hier die 9, daraufhin prüft, ob er noch kleiner als der Laufparameter X ist. Bei positiven Schrittweiten er-

folgt dagegen die Prüfung, ob der Grenzwert noch größer ist.
Wählen wir also den Laufbereich von 4 bis −3 und die Schrittweite zu −2, so muß die Zahlenfolge 4; 2; 0; −2 entstehen.

5.2. Reelle Schrittweite und einige Bemerkungen zum Nulldurchgang

Wir haben in Kapitel 2 den Begriff der Variablen und den LET-Befehl eingeführt.
Dort wurde gezeigt, daß die Erhöhung um 1/7, von −1 beginnend, nicht exakt durch Null führt. Der Grund lag darin, daß alle Zahlen im Rechner binär gespeichert sind und 1/7 dabei zu einer Zahl führt, die nicht exakt bei 7maliger Addition 1 ergibt. Dieses Problem wollen wir nun genauer untersuchen.
Das Pogramm STEP 1 veranschaulicht noch einmal den Sachverhalt. Es verlangt zunächst die Eingabe einer Schrittweite X. Dann wird der Laufparameter I von −10*X bis .1 mit dieser Schrittweite geändert, also:

```
30 FOR I=−10*X TO .1 STEP X ⟨ENTER⟩
```

In der nächsten Zeile drucken wir I gemäß

```
40 PRINT I, ⟨ENTER⟩
```

(Komma für Formatierung) aus, und dann folgt

```
50 NEXT ⟨ENTER⟩
```

Bei einem RUN erfolgt dann z. B. nach Eingabe von .25 die Anzeige: −2.5; −2.25; −2 usw., und das Ende wird mit −.25 und 0 erreicht.
Die Schrittweite .25, also 1/4, führt folglich exakt zu Null, wie man es auch erwartet. Anders verhält sich aber bereits die Eingabe der Schrittweite .3. Hier sieht die Zahlenfolge so aus: −3; −2.7; −2.4 usw. bis −1.2. Es tritt der erste unerwartete Effekt mit −.900001; −.600001 und −.300001 auf, und schließlich endet die Reihe mit −4.76837E-07.
Die Eingabe .3 zeigt also noch deutlicher als 1/7 den möglichen Fehlereinfluß der binären Zahlen im dezimalen Ablauf. Die 10malige Addition einer binären Zahl kann somit im dezimalen Bereich zu einem Wert führen, der ein ganz klein wenig anders ist als jener, der bei Multiplikation mit 10 entsteht. Die Differenz (der Rundungsfehler) zwischen beiden beträgt in diesem Fall zwar nur 0.5 Millionstel. Sie macht sich aber in der unmittelbaren Umgebung von Null bemerkbar. Sie überschreitet praktisch nie diesen relativ kleinen Wert und wird daher normalerweise auch nicht feststellbar sein. Jedoch in der Nähe von Null sollte diese Abweichung immer beachtet werden (vgl. PROGRAMM STEP 2).

5.3. Schrittweite Null

Es bleibt noch die interessante Frage zu klären, was der Interpreter tut, wenn wir der Schrittweite den Wert Null zuweisen. Die einzelnen Interpreter verhalten sich bei diesem Spezialfall recht unterschiedlich.
Beim KC 85 tritt das folgende ein: Ist in

```
10 FOR X=A TO B STEP 0 ⟨ENTER⟩
```

A=B, so wird die Zählschleife genau einmal durchlaufen. In allen anderen Fällen erfolgt der Schleifendurchlauf ohne Ende. Dieses Verhalten ist für den Fall, daß B<A, nicht unmittelbar einsichtig. Es erklärt sich aber aus der Funktion SGN(C), welche vom Interpreter aus der Schrittweite abgeleitet wird.
Wir untersuchen diese Funktion mit

```
10 FOR X=−2 TO 2 ⟨ENTER⟩
20 PRINT X, SGN(X) ⟨ENTER⟩
30 NEXT ⟨ENTER⟩
```

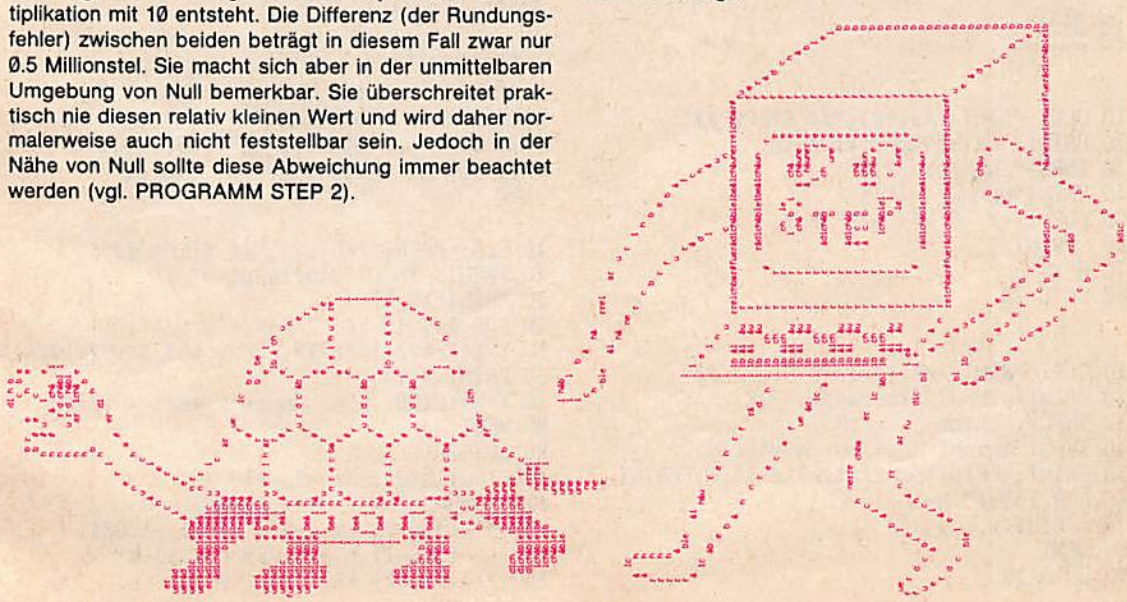
und erhalten bei RUN

```
−2  −1                          kleiner A
−1  −1
 0   0      Testet ab auf B     gleich A
 1   1
 2   1                          größer A
OK
```

Die Testung des Endwertes erfolgt bei STEP 0 nur auf Gleichheit, und das Ende wird für jedes unterschiedliche A und B nie erreicht.

5.4 Logarithmische Schrittweite

Nach diesen vielen Möglichkeiten wollen wir nun die Frage stellen, ob es vielleicht auch möglich ist, die Schrittweite so zu wählen, wie es z. B. bei den Tonhöhen der wohltemperierten Musikskala der Fall ist. Sie ist ja gemäß dem Faktor der 12. Wurzel aus 2 gestuft. Denn gehörmäßig sind in der Oktave 12 gleich große Intervalle, Halbtöne enthalten. Ähnliche Verhältnisse kennen die Elektroniker. Die Widerstände sind gemäß einer n-ten Wurzel aus 10 gestuft, wobei n dann 6, 12, 24, 48 oder 96 beträgt.



Dieses Ziel ist nun leider nicht ganz einfach zu realisieren, denn hierfür gibt es keinen direkten Befehl. Aber ein kurzes Programm leistet das Geforderte. Es liegt als STEP 3 vor. Hier der Grundgedanke:
Wir geben die Anzahl der Stufen, also bei den Noten 12, bei den Widerständen 6, 12 usw., als Anzahl N ein und lassen den Laufparameter I von 1 bis N+1 laufen.
Hierbei benötigen wir das STEP nicht. Die Schrittweite erzeugen wir indirekt einmal aus der Eingabe eines Faktors M, bei den Noten mittels der Oktave, d. h. 2, bei den Widerständen aus der Zehnerpotenz also 10. Zum zweiten aus dem M gemäß der n-ten Wurzel.
Die multiplikative Schrittweite W berechnet sich daher: W = M(1/N).
Nun benötigen wir noch einen Startwert X. Hierfür können wir in der Musik z. B. den Kammerton a mit 440 Hz wählen. Dann zeigen wir nach jedem Durchlauf zuerst X an und multiplizieren es anschließend mit W. In diesem Beispiel sind also die Zählschleife und die multiplikative Schrittweite über verschiedene Variablen realisiert. Dadurch wird deutlich, daß innerhalb der Zählschleife nahezu unabhängig von den Parametern der Zählschleife operiert werden kann (vgl. auch RUN am Ende). Auf diese Weise wird die Anweisungsfolge FOR-NEXT zu einem mächtigen Befehl.

5.5 Adaptive Schrittweitensteuerung

Das Programm STEP 4 demonstriert auf anschauliche Weise die Möglichkeit einer adaptiven Schrittweitensteuerung. Es berechnet nach Eingabe eines Startwertes X die Nullstellen der kubischen Gleichung X^3−4*X^2−19X−14. Sie liegen bei: −2; −1; 7.
Es wird hier eine bedingt gültige, besonders einfache Methode verwendet. Sie soll eben nur das Wesentliche von einer sich selbst anpassenden Schrittweite demonstrieren (adaptive Schrittweitensteuerung). Nach Eingabe eines Startwertes X wird der Funktionswert Y berechnet, und dann wird ein 1/100 dieses Wertes als adaptive Schrittweite vom vorhandenen X-Wert abgezogen. Das Ganze erfolgt im Programm 16mal. Dabei sieht man, welchem Wert sich X nähert und wie Y gegen Null strebt.
Das Programm STEP können Sie verwenden, um festzustellen, daß die Variablen A, B, C nur in der Zeile 30 lokalen Charakter besitzen.

Zusammenfassung Kapitel 5

1. In einer FOR-NEXT-Schleife kann die Schrittweite abweichend von 1 gewählt werden. Das Format der Anweisung lautet: FOR X=A TO B STEP C.
   Die Schleife wird mit dem Befehl NEXT oder NEXT X abgeschlossen.
   In einer FOR-NEXT-Anweisung ohne STEP setzt der Rechner die Schrittweite automatisch gleich 1.
2. Eine FOR-NEXT-Schleife wird unabhängig von allen Parametern mindestens einmal durchlaufen.
3. Bei negativen Schrittweiten erfolgt die Prüfung bei NEXT daraufhin, ob nach Abzug des Wertes vom Laufparameter der Grenzwert bereits erreicht oder unterschritten ist.
4. Mit der Schrittweite Null kann eine unendliche Zählschleife erzeugt werden.
5. In der Zählschleife können unabhängig vom Laufparameter zusätzliche Variablen verwendet werden. Auf diese Weise ist es möglich, verschiedene Arten von Schrittweiten zu realisieren. Die Zählschleife legt dann nur noch die Anzahl der Durchläufe fest.
6. In der Anweisungsfolge FOR X=A TO B STEP C haben die Variablen A, B und C nur lokale Bedeutung. Sie werden sofort vom Interpreter in spezielle Speicherzellen abgelegt und danach für den Schleifenablauf nicht mehr benötigt.
7. Innerhalb einer Schleife kann der Laufparameter durch eine Anweisung verändert werden. Diese Möglichkeit ist zu beachten und kann in Sonderfällen von Nutzen sein.

```
10 CLS: PRINT TAB(12);"## STEP1 ##"
20 PRINT: INPUT"Schrittweite";X
30 FOR I=-10*X TO .1 STEP X
40  PRINT I,
50 NEXT
60 GOTO20
```

```
10 CLS: PRINT TAB(12);"## STEP2 ##"
20 PRINT: PRINT"Schrittweite"
30 INPUT"Zaehler =";X
40 INPUT"Nenner  =";Y
50 FOR I=-X TO X/Y STEP X/Y
60  PRINT I,
70 NEXT
80 GOTO 20
```

```
10 CLS: PRINT TAB(12);"## STEP3 ##"
20 PRINT: INPUT"Startwert =";X
30 INPUT"Faktor     =";M
40 INPUT"Anzahl     =";N: W=M^(1/N)
50 PRINT: PRINT"Schrittweite =";W: PRINT
60 FOR I=1 TO N+1
70  PRINT X,: X=X*W
80 NEXT
90 GOTO 20
```

```
          ## STEP3 ##
Statwert = 440
Faktor   = 2
Anzahl   = 12

Schrittweite = 1.05946
 440          466.164      493.883
 523.251      554.365      587.33
 622.254      659.255      698.457
 739.989      783.991      830.61
 880
```

```
10 CLS: PRINT TAB(12);"## STEP4 ##"
20 PRINT: INPUT"Startwert =";X
30 PRINT"X","Y"
40 FOR I=1 TO 16: Z=X*X
50 Y=X*Z-4*Z-19*X-14: REM: x^3-4*x^2-19X-14
60 PRINT X,Y
70 X=X-Y/100: REM: neuer x-Wert - Y/100
80 NEXT
90 GOTO20
100 !Nullstellen: -2, -1, 7
110 !Grenzen:     -14, (+oo)
120 ! -13 (+7) -9; -8.4  (-2) -1.001
130 ! -1  (-1)   ; -.999 (7) 13.4
140 !13.5 (-2) 15;  20   (-oo)
```

```
          ## STEP4 ##

Startwert = 2
X             Y
 2            -60
 2.6          -72.864
 3.32864      -84.6827
 4.17547      -90.2747
 5.07822      -82.6808
 5.90502      -59.7686
 6.50271      -31.7239
 6.81995      -12.4185
 6.94413      -3.96954
 6.98383      -1.15994
 6.99543      -.328857
 6.99872      -.092392
 6.99964      -.0258636
 6.9999       -7.27844E-03
 6.99997      -2.01416E-03
 6.99999      -5.79834E-04
```

```
10 CLS: PRINT TAB(12);"## STEP5 ##"
20 PRINT: A=1: B=3: C=.2
30 FOR I=A TO B STEP C
40  PRINT I,: A=3: B=10: C=.1
50 NEXT
60 !mit 35 I=5 wird sofort Ende erreicht
```

6. Spezielle PRINT-Befehle und die Anweisung REM

In BASIC sorgt die Ausgabeanweisung PRINT für die Anzeige aller Art, wie Zahlen, Zeichen, Zeichenketten, auf dem Bildschirm des Computers. Eine kurze Form der PRINT-Anweisung lautet: PRINT Ausdruck.
Der Ausdruck kann im einfachsten Fall nur eine Konstante oder eine Variable sein. Ansonsten wird der Wert des Ausdrucks berechnet und auf dem Bildschirm ausgegeben. Nach Abschluß der Ausgabe wird auf eine neue Bildschirmzeile übergegangen.
Spezielle PRINT-Befehle wie TAB(X) oder SPC(X) ermöglichen es uns, eine übersichtliche Gestaltung der Anzeige auf dem Bildschirm bzw. beim Druck zu realisieren. Hierfür existiert auch der Fachausdruck Formatieren. Mit Hilfe dieser Funktionen können wir eine Ausgabe exakt innerhalb einer Zeile plazieren.
Die Kommentar- oder REM-Anweisung (engl.: to remark – bemerken) ist für das Einfügen von erklärenden Texten innerhalb eines Programms gedacht. Der Programmierer kann sie beliebig oft und an fast jeder Stelle des Programms verwenden. Insbesondere in größeren Programmen ist das Einfügen von Kommentaren nützlich und wird oft angewendet.

6.1. Die Formatisierungsfunktion TAB(X)

Die TAB-Funktion dient der programmtechnischen Steuerung des Ausgabegerätes. Sie darf nur in einer PRINT-Anweisung benutzt werden und hat folgendes Format: TAB(X).
Die TAB-Funktion bewirkt, daß danach als aktuelle Ausgabeposition auf der Bildschirm- oder in der Druckzeile die durch den Wert von X bezeichnete Position eingestellt wird. Das Argument von TAB gibt den Abstand der Ausgabe zum linken Bildschirmrand in Zeichen an. Für das Argument X kann eine Zahl, eine numerische Variable oder allgemein ein numerischer Ausdruck stehen. Ergibt sich für X ein gebrochener numerischer Wert, so wird vom Computer der ganzzahlige Anteil genutzt. TAB(1) entspricht der äußersten linken Position einer Zeile.
Stellen Sie sich bitte einmal vor, Sie wollen auf dem Bildschirm ein Dreieck erzeugen. Dann müssen Sie oben in der Mitte auf dem Bildschirm z. B. einen Multiplikationspunkt (Stern) erzeugen. Wir nehmen an, dies ist wie beim KC 85 die Zeilenposition 19. Dann würde der entsprechende Befehl lauten:

```
30 PRINT TAB(19); "*" ⟨ENTER⟩
```

Die in PRINT stehende Anweisung TAB(19) bewirkt, daß der Cursor genau auf die Position 19 geführt wird. Infolge des Semikolons wird er hier festgehalten, und anschließend wird der Stern gesetzt. In der dann folgenden Zeile muß je ein Stern um +−1 von dieser Position gedruckt werden. Eine weitere Zeile tiefer um +−2 usw. Um die Breite unseres Bildschirms optimal zu nutzen, ist dies genau 18mal möglich. Es ist daher mit der folgenden FOR-NEXT-Sequenz auszuführen:

```
50 FOR I=1 TO 18                    ⟨ENTER⟩
60 PRINT TAB(19−I); "*"; TAB(19+I); "*"
                                    ⟨ENTER⟩
70 NEXT                             ⟨ENTER⟩
```

Nachdem auf diese Art nun die beiden Schenkel des Dreiecks erzeugt sind, muß noch die Grundlinie gestaltet werden. Sie muß genau 2*18+1 (oben) +2 (Schenkelrest) =39 Sterne enthalten. Also lautet die Befehlsfolge:

```
80 FOR I=1 TO 39: PRINT"*"; : NEXT   ⟨ENTER⟩
```

(Vgl. Programm und Ausdruck DREIECK.)

6.2. Die Formatierungsfunktion SPC(X)

Ergänzend zu TAB gibt es in BASIC noch einen zweiten, ähnlichen Befehl, nämlich SPC(X) (sprich space). Während TAB stets in der aktuellen Zeile auf den linken Rand Bezug nimmt, geht SPC von der jeweils aktuellen Cursor-Position in der Zeile aus. Von hier ab wird der Cursor genau um die in der Klammer von SPC stehende Zahl nach rechts geführt. Dadurch ist SPC im Gegensatz zu TAB immer real ausführbar. Im ungünstigsten Fall wird dabei der rechte Zeilenrand überschritten und deshalb der Cursor weiter in die nächstfolgende Zeile an die entsprechende Position gebracht.
Dies läßt sich mit einem kurzen Programm TEST sehr schnell erproben.

```
10 CLS                              ⟨ENTER⟩
20 FOR X=1 TO 80                    ⟨ENTER⟩
30 PRINT SPC(9);"TEST"              ⟨ENTER⟩
40 NEXT                             ⟨ENTER⟩
```

Dabei wird das Wort TEST in schräger Linie untereinander auf dem Bildschirm gedruckt.
Wir wollen Ihnen die Funktion von SPC(X) noch einmal an Hand des Programms SPACE verdeutlichen. Es verwendet allerdings zwei neue Anweisungen, die mit Zufallsfolgen zusammenhängen. Also ähnlich, wie Sie es beim Würfeln tun. Anstatt der dabei möglichen Zahlen 1 bis 6 erzeugt der Zufallsgenerator in BASIC gleichmäßig verteilte Zufallszahlen zwischen 0 und 1. Der Hauptbefehl hierfür ist RND(X).

Wir verwenden hier ein beliebiges positives X (weitere Details s. Kapitel 17).
Mit dieser Methode erzeugen wir nun einen zufälligen Abstand zwischen Zeichen bzw. Wörtern. Es hat sich für ein ansehnliches Bild als günstig erwiesen, den Zufallsabstand zwischen Ø und 1Ø zu wählen.
Deshalb wählen wir die Zeile

```
40 PRINT SPC(10* RND(1) ; A$; ⟨ENTER⟩
```

Wenn wir dies etwa 22Ømal nach einem Löschen der Bildschirmanzeige ausführen, wird in etwa der Bildschirm zufällig vollgeschrieben. Mit A$="*" entsteht so ein Sternenhimmel. Mit A$="BASIC" erhalten Sie eine interessante Anordnung des sich ständig wiederholenden Wortes BASIC auf dem Bildschirm. Das Programm SPACE ist so geschrieben, daß Sie eine Zeitlang experimentieren können. Dies sehen Sie am Ausdruck mit dem Wort BASIC. Die Zeile 60 verhindert das Weiterlaufen des Programmes. Die beiden Anführungszeichen unterdrücken das Fragezeichen bei der Eingabe.
Zu den reservierten Wörtern TAB(und SPC(gehört die linke Klammer. Dadurch sind ausnahmsweise Variablenbezeichnungen, in denen TAB und SPC vorkommen, wie z. B. TABAK, zulässig.

6.3. Die PRINT-Funktion AT

Mit der Funktion AT können wir unsere Ausgaben an jeder beliebigen Stelle des Bildschirms plazieren. Dabei ist folgende Form zu beachten:
PRINT AT(Zeile, Spalte); Ausdruck.
Hierzu ist die Gliederung des Bildschirms zu kennen. Es gibt eine Zählung der Zeilen und Spalten. Innerhalb einer einzelnen Zeile wirken bezüglich der Spaltenposition die Befehle TAB und SPC. Zusätzlich läßt AT auch einen direkten Zugriff auf die Zeilen zu. Sie werden auf dem Bildschirm, von oben beginnend, gezählt. Beim KC 85/2 und KC 85/3 sind die Zeilen Ø bis 31 beim KC 85/1 und KC 87 Ø bis 23 vorhanden.
Da mit PRINT AT sofort auf jede Stelle des Bildschirms zugegriffen werden kann, ist es für die Druckausgabe nicht brauchbar.
Wir wollen jetzt hiermit ein einfaches Programm realisieren, bei dem wir wieder mit dem Zufall arbeiten, und uns so erneut einen Sternenhimmel erzeugen. Hierzu müssen wir die Positionen X(Spalte) und Y(Zeile) erzeugen:

```
30 X=RND(1)*40 ⟨ENTER⟩
40 Y=RND(1)*30+1 ⟨ENTER⟩
```

(beim KC 85/1 und KC 87 statt 3Ø nur 22). Das +1 ist sinnvoll, da im Normalfall nicht die Zeile Ø verwendet wird. Die Druckzeile lautet dann:

```
50 PRINT AT(Y,X); "*" ⟨ENTER⟩
```

Wenn wir die Zeilen 3Ø bis 5Ø bis zu 24Ømal wiederholen, wird die Bildschirmfläche zu etwa 20 % mit Sternen gefüllt (vgl. Programm AT).
Die Anwendung von PRINT AT soll Ihnen noch einmal durch das Programm WURZEL veranschaulicht werden. Dieses Programm läuft nur auf dem KC 85/2 und KC 85/3. Es erzeugt auf dem Bildschirm ein Achsenkreuz mit Beschriftung und darin die Punkte der Funktion Y=SQR(X)
Die Bezeichnung SQR steht für square-root, also Quadratwurzel. Wenn Sie dieses Programm analysieren, werden Sie schnell die Vielfalt der Anwendungen von PRINT AT erfassen.

6.4. Die Kommentar- oder REM-Anweisung

Für eine Analyse von umfangreichen Programmen ist es nützlich, mehrere Hilfstexte (Kommentare) einzufügen. Sie werden alle durch das reservierte Wort REM eingeleitet. Eine REM-Anweisung dient nur der Erläuterung und Erhöhung der Lesbarkeit des Programms. Sie hat keine Wirksamkeit auf die Programmausführung. Die REM-Anweisung hat einen einfachen Aufbau:
REM Kommentar.
Der Kommentar ist ein Text mit beliebigen Zeichen aus dem gesamten Zeichensatz von BASIC.
Wenn Sie fremde Programme lesen oder auch von Ihnen selbst geschriebene, werden Sie dankbar für diese Hinweise sein. Sie ermöglichen es Ihnen nämlich, schnell die Wirkungsweise eines Programms zu verstehen. Ein Programmierer sollte also mit Kommentaren nicht zu sparsam umgehen.
Allerdings ist auch zu beachten, daß REM-Anweisungen in einem BASIC-Programm Speicherplatz belegen und schließlich auch die Laufzeit des Programms nachteilig beeinflussen, obwohl sie bezüglich der Abarbeitung des Programms sonst keine Wirkung verursachen.
Im Programmablauf werden nämlich die Kommentare vom Interpreter einfach ignoriert, d. h., sie haben keinen Einfluß auf die Programmausführung. Sobald der Interpreter auf eine REM-Anweisung stößt, hört er in der Zeile auf zu übersetzen. Für das Erkennen von REM benötigt er aber zusätzlich etwas Zeit.
Zu REM gibt es noch ein zweites äquivalentes Zeichen, das Ausrufezeichen:

```
1Ø! KOMMENTAR ⟨ENTER⟩
```

Obwohl beide Zeichen auf dem Bildschirm unterschiedlich lang sind, verbrauchen sie im Speicher des Rechners gleich viel Platz und beim Programmablauf die gleiche Zeit. Es ist also nur Geschmackssache, welches von beiden man verwendet.
Bemerkung: Während der Interpreter bzgl. des Programms automatisch alle Eingaben in Großbuchstaben umwandelt, tut er dies mit hinter REM bzw. ! stehenden Kommentaren nicht. Sie bleiben diesbezüglich wie Stringvariablen in Anführungszeichen exakt erhalten.

```
10 CLS: PRINT TAB(13);"## DREIECK ##": PRINT
20 INPUT"Zeichen =";A$
30 PRINT TAB(19);A$
40 REM: Die Schenkel
50 FOR I=1 TO 18
60  PRINT TAB(19-I); A$; TAB(19+I); A$
70 NEXT
80 REM: Grundlinie
90 FOR I=1 TO 39
100  PRINT A$;
110 NEXT: PRINT
```

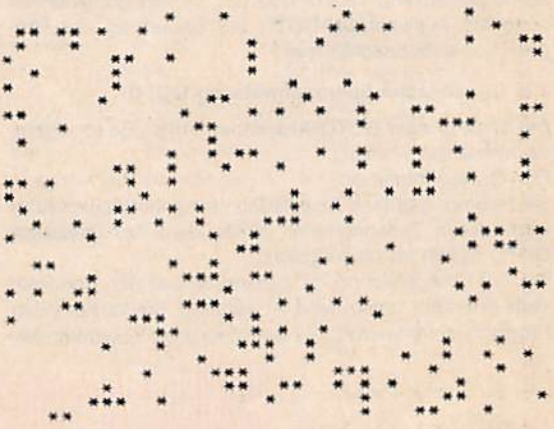
```
          ## DREIECK ##
Zeichen = OK
                   OK
                  OKOK
                 OK  OK
                OK    OK
               OK      OK
              OK        OK
             OK          OK
            OK            OK
           OK              OK
          OK                OK
         OK                  OK
        OK                    OK
       OK                      OK
      OK                        OK
     OK                          OK
    OK                            OK
   OK                              OK
  OK                                OK
OKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOK
OKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOKOK
OK
>
```

Zusammenfassung Kapitel 6

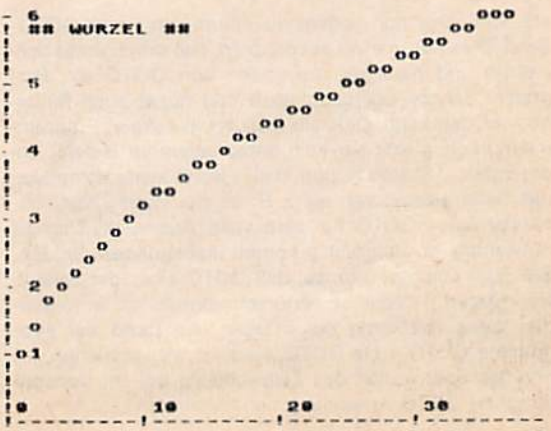
1. Wird hinter PRINT das Wort TAB gesetzt, so führt der Interpreter den Cursor in der aktuellen Zeile in die Position, die in Klammern nach TAB steht. Sie wird genau vom linken Bildschirmrand aus berechnet.
2. Hinter TAB(X) ist es sinnvoll, ein Semikolon folgen zu lassen. So wird der dahinter stehende Text deutlich von dem TAB-Teil unterschieden.
3. Hinter PRINT können, durch Semikolon getrennt, mehrere TAB-Teile folgen. So ist es möglich, in einer Zeile verschiedene Positionen für den Druck zu erreichen. Dabei ist aber zu beachten, daß alle später folgenden TAB-Werte so angeordnet werden, daß die Drucktexte enthalten sind.
4. SPC nach PRINT bewirkt, daß der aktuelle Cursor um die entsprechende Anzahl von Positionen weitergeführt wird. Dabei kann auch der rechte Bildschirmrand überschritten werden. Dann wird der Cursor an die entsprechende Stelle in der nächsten Zeile gesetzt.
5. RANDOMIZE erzeugt für den im Interpreter eingebauten Zufallsgenerator eine zufällige Startposition.
6. RND(1) ruft den Zufallsgenerator auf, wodurch eine Zahl zwischen Ø und 1 erzeugt wird.
7. Mit PRINT AT(Z,S); kann der Cursor für die Ausgabe sofort an jeden beliebigen Punkt des Bildschirms geführt werden. Z und S stehen für Zeilennummern und Spaltennummer.
8. PRINT AT ist nicht für Druckausgabe geeignet.
9. Damit die Programme gut lesbar sind, ist es nützlich, Kommentare einzufügen. Dies kann immer hinter einem REM oder einem ! geschehen. Der Interpreter übergeht beim Programmablauf alles, was auf dieser Zeile danach steht.
10. TAB(und SPC(enthalten im reservierten Wort am Ende die offene Klammer. Deshalb können TAB und SPC ohne Klammer zur Bezeichnung für oder in Variablen verwendet werden.

```
10 CLS: RANDOMIZE: REM: ## SPACE ##
20 INPUT"Text =";A$: CLS
30 FOR X=1 TO 220
40  PRINT SPC(10*RND(1)); A$;
50 NEXT
60 INPUT"";A: GOTO 10
```

```
10 CLS: RANDOMIZE: REM: ## AT ##
20 FOR I=1 TO 240
30  X=RND(1)*40
40  Y=RND(1)*30+1
50  PRINT AT(Y,X);"*"
60 NEXT
70 INPUT"";A: GOTO 10
```



```
10 CLS: PRINT AT(1,2);"## WURZEL ##"
20 REM: x-Achse
30 FOR J=0 TO 30 STEP 10
40  FOR I=1 TO 9
50   PRINT AT(30,I+J);"-";
60   PRINT AT(29,J);J
70  NEXT: PRINT AT(30,J);"!";
80 NEXT
90 REM: y-Achse
100 FOR J=30 TO 1 STEP -5
110  FOR I=1 TO 4
120   PRINT AT(J-I,0);"!";
130  NEXT: PRINT AT(J-5,0);"-";7-J/5
140 NEXT
150 REM: Funktion
160 FOR X=1 TO 37
170  Y = SQR(X)*5 + .5
180  PRINT AT(31-Y,X);"o";
190 NEXT
200 REM: OK unterdruecken
210 INPUT"";X
```



7. Die Sprunganweisung GOTO

Die Anweisungen eines BASIC-Programms werden normalerweise in der Reihenfolge der Zeilennummern ausgeführt. Mit Hilfe von Anweisungen zur Steuerung des Programmablaufs ist es jedoch möglich, diese Reihenfolge zu verlassen oder die Programmausführung zu beenden. So kann man auch komplizierte algorithmische Strukturen, wie z. B. Verzweigungen, programmieren.
Wir wollen uns in diesem Kapitel mit der einfachen Sprunganweisung GOTO und der Verzweigungsanweisung mit Auswahl ON-GOTO zur Steuerung des Programmablaufs beschäftigen.

7.1. Die einfache Sprunganweisung GOTO

Die Sprung- oder GOTO-Anweisung (engl.: go to – gehe nach) hat das Format:
GOTO Zeilennummer.
Sie bewirkt, daß jene Anweisung als nächste ausgeführt wird, deren Zeilennummer hinter dem Schlüsselwort GOTO notiert ist (Sprungziel).
Die GOTO-Anweisung ist zumindest bei den Theoretikern eine sehr umstrittene Anweisung. Sie verführt den Programmierer leicht zu einem schlechten Programmierstil. Im Fachjargon spricht man dann von einem GOTO-Salat. Dies soll in etwa ausdrücken, daß durch undisziplin iertes und häufiges Anwenden von GOTO ein Programm schnell unübersichtlich und damit auch fehlerhaft werden kann. Deshalb existiert in einigen Programmiersprachen erst gar kein entsprechender Befehl. Andererseits läßt sich zeigen, daß – wenn keine komplexeren Befehlsstrukturen als z. B. im Standard-BASIC bereitstehen – GOTO für eine volle Ausdrucksfähigkeit notwendig ist. Schließlich kommt insbesondere bei BASIC noch nachteilig hinzu, daß GOTO einer der zeitaufwendigsten Befehle ist. Natürlich können wir im folgenden diese Probleme nur streifen. An Hand der Programme GOTO 1 bis GOTO 4 wollen wir versuchen, Ihnen den Sachverhalt des Zeitproblems bei der Verwendung der GOTO-Anweisung nahezubringen.

Das erste Programm – GOTO 1 – besteht nur aus 3 Zeilen. Aus ihm werden sich dann in einfacher Weise die weiteren drei Programme ableiten lassen.
Also:

```
10 X = 1 : K = 1.01 : I = 1  (ENTER)
20 X =X*K                    (ENTER)
30 GOTO 20                   (ENTER)
```

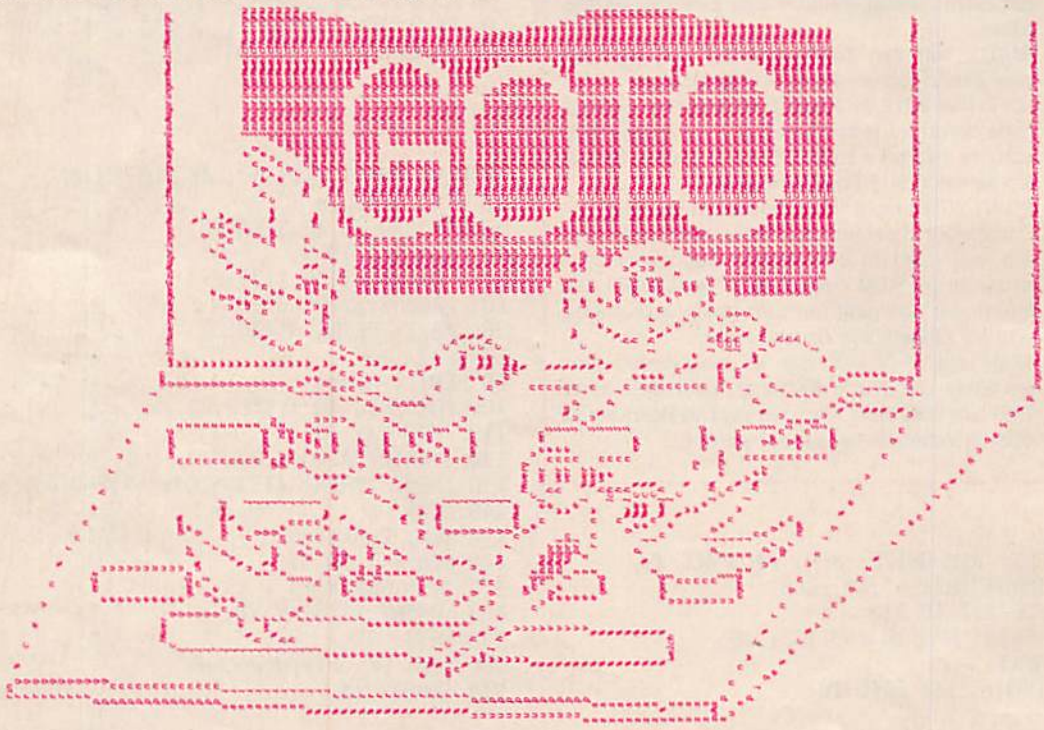
Dieses Programm wurde bereits in Kapitel 3 in Zusammenhang mit dem LET-Befehl behandelt. Es läuft solange, bis ein OV-ERROR erfolgt. Dies geschieht beim KC 83/3 nach ca. 58.3 s.

Wir versuchen jetzt festzustellen, wie oft die Schleife bis zum OV durchlaufen wird. Zu diesem Zweck fügen wir in Zeile 20 eine Addition als Zähler ein (GOTO 2):

```
20 X = X*K : I = I + 1       (ENTER)
```

Jetzt läuft das Programm 96.4 s, und mit

```
PRINT I (ENTER)
```



erhalten wir die Anzahl der Durchläufe zu

```
8779
```

Aus der Differenz beider Zeiten, nämlich 38,1 s, und der gerade bestimmten Anzahl der Durchläufe erhalten wir die Zeit für eine Addition zu 4.3 ms.

7.2. Zum Zeilenaufbau

Dieser Versuch war eigentlich nur eine Vorbereitung für unser GOTO. Für die weiteren Zeitmessungen müssen wir uns zunächst noch damit beschäftigen, wie GOTO vom Interpreter benutzt wird. Dazu müssen wir uns die Struktur einer BASIC-Zeile ansehen (s. Bild S. 22). Sie wird wie folgt im Speicher abgelegt. Die Speicherzellen werden wir hier einfach je Byte wählen.

Die ersten beiden Byte des BASIC-Programms weisen nun auf jene Adresse hin, wo die zweite Zeile beginnt. Die dort stehenden ersten beiden Byte verweisen auf den Beginn der dritten BASIC-Zeile usw. Es liegt also mittels der jeweils ersten beiden Byte jeder Zeile eine Verkettung der einzelnen BASIC-Zeilen vor. Man nennt daher diese beiden Byte auch Pointer, zu deutsch Zeiger.
Hinter den jeweils ersten beiden Byte jeder Zeile stehen nun wieder jeweils zwei Byte. In ihnen ist die Zeilennummer der Zeile kodiert.
Dahinter folgt der eigentliche BASIC-Zeilentext.
Das Ende jeder Zeile wird mit Null gekennzeichnet.
Ein GOTO-Befehl enthält nun immer die Zeilennummer, zu der der Interpreter dann gehen soll. Gelangt der Interpreter zu einem GOTO-Befehl, so wird der folgende Mechanismus in Gang gesetzt:
Der Interpreter liest die hinter GOTO stehende Zeilennummer, übersetzt sie ins Binäre und legt sie an einer bestimmten Stelle im Interpreter-Hilfsspeicher ab. Danach geht er zum Programmanfang und sieht im dritten und vierten Byte nach, ob dort die richtige Zeilenzahl steht.
Ist dies der Fall, dann ist das Ziel erreicht.
Ist die Zeilenzahl größer als die gesuchte, dann gibt er die Fehlermeldung:
?UL ERROR IN XX (unzulässige Zeile),
wobei XX jene Zeilennummer ist, wo der GOTO-Befehl stand.
Der Normalfall wird jedoch sein, daß die Zeilennummer noch zu klein ist. Dann greift der Interpreter zwei Byte zurück auf den Pointer zur nächsten Zeile und springt dorthin, um hier die Zeilennummer zu lesen und mit dem gesuchten Wert zu vergleichen. Bei zu kleiner Zeilennummer wiederholt sich dieser Vorgang so lange, bis entweder die Zeile gefunden oder ein zu großer Wert zum UL-ERROR führt.

7.3. Zeitanalyse

Nun ändern wir unser Programm GOTO 1 dadurch, daß wir in den Zeilen 11, 12 bis 19 je eine Kommentarzeile (REM-Anweisung) erzeugen (Programm GOTO 3). Das Programm läuft 67.1 s, also 8.8 s länger als GOTO 1. Bei 9 eingefügten REM und 8779 Durchläufen erhalten wir für jede Zeilennummer-Suche etwa Ø.1 ms. Es ist also für schnelle Programme wichtig, sie so zu gestalten, daß alle GOTO-Sprünge möglichst weit nach vorn im Programm verweisen. Leider läßt sich dies jedoch nur sehr bedingt realisieren. Aber gerade hier liegen die Zeitlimite für das GOTO, und es kommt infolgedessen sogar darauf an, GOTO-Sprünge – wo irgend möglich – zu vermeiden. Leider berücksichtigen selbst kommerzielle Programme dies nur bedingt und laufen dann relativ langsam.
Das bisher behandelte Problem ist nun relativ leicht in ein FOR-NEXT-Programm überführbar. Dabei entstehen 4 Zeilen:

```
10 X=1: K=1.01                ⟨ENTER⟩
15 FOR I=1 TO 2 STEP Ø        ⟨ENTER⟩
20 X=X*K                      ⟨ENTER⟩
30 NEXT                       ⟨ENTER⟩
```

Der Überlauf erfolgt hier nach 59.1 s.
Wählt man andere Schrittweiten als Ø und macht dazu die Oberschranke nach dem TO hinreichend groß, so liegt die Laufzeit immer bei ca. 63.7 s. Das hierfür übliche Programm ist GOTO 4. Damit liegt es in der Laufzeit zwischen GOTO 1 und GOTO 2. Hieraus folgt, daß FOR-NEXT-Programme fast immer schneller als entsprechende GOTO-Programme sind. Der Unterschied wird dann besonders groß, wenn GOTO-Sprünge zu relativ hohen Zeilennummern führen. Der hier gemachte Vergleich ist in diesem Sinne geradezu untypisch, denn fast immer werden die GOTO-Sprünge sogar zu wesentlich höheren Zeilenzahlen zurückführen. FOR-NEXT ist dann meist erheblich schneller.
Um den Zeitverlust, der bei GOTO-Sprüngen auftritt, noch deutlicher zu machen, fügen wir jetzt in unser Ausgangsprogramm GOTO 1 die folgenden 4 Zeilen ein:

```
22 GOTO 24                    ENTER
24 GOTO 26                    ENTER
26 GOTO 28                    ENTER
28 GOTO 30                    ENTER
```

Dies ist zwar fast unsinnig hinsichtlich eines günstigen Programmablaufs. Es ist aber optimal, wenn man den Zeitverlust bzgl. eines GOTO ermitteln will. Das Programm GOTO 5 benötigt bis zum Überlauf 104.8 s, also 46.5 s mehr als GOTO 1. Infolge der 4 zusätzlichen GOTO und der 8779 Durchläufe kommen auf jedes GOTO also 1.3 ms. Zieht man die Zeilensuche ab, so benötigt nun das GOTO allein ca. 5 ms.
Um die Zeitbetrachtung bzgl. GOTO abzuschließen, sei noch auf das Programm GOTO 6 eingegangen. Es ist wie folgt strukturiert:

```
10 X=1: K=1.01                ⟨ENTER⟩
20 X=X*K                      ⟨ENTER⟩
30 GOTO 240                   ⟨ENTER⟩
```

Die Zeilen 40, 50 usw. bis 230 enthalten nur ein REM.

```
240 GOTO 20                   ⟨ENTER⟩
```

Kritisch hinsichtlich der Laufzeit ist hier die Zeile 30, in der 24 Zeilen abzusuchen sind. Dies allein führt nach unseren bisherigen Abschätzungen auf (24*.1+5)*8779 ms, also fast 26 s Verlust. Gemessen wurden 89.8 s bis zum Überlauf, also insgesamt 31.6 s Verlust.
Nun ersetzen wir in Zeile 30 das GOTO 240 durch ein REM, dann beträgt die Zeit zum Overflow 187.2 s. Daraus ist zu folgern, daß auch das Erkennen von REM erhebliche Zeit erfordert. Sie kann je REM-Zeile auf etwa 5 ms geschätzt werden.
Bezüglich der Laufzeiten sei noch auf einige generelle Meßmethoden hingewiesen. Sie dienen zum Rechnervergleich und heißen bench-mark, zu deutsch etwa Höhenmarke. Sie verwenden möglichst alle wichtigen Befehle in einer Häufigkeit, die den mittleren Anwendungsfällen entspricht. Daher existiert auch der MIX, soviel wie Mischung aus Befehlen. Wir geben zum Schluß des Kapitels des Programm BENCH an, was fast exakt jenem Programm in dem Buch von D. Werner, BASIC für Mikrorechner, (Verlag Technik) auf S. 67 entspricht. Dort sind auch Werte für verschiedene Rechner angegeben. Der KC 85/3 liegt mit einem Wert von 62 s für die Grundrechenarten recht gut.

7.4. Die Verzweigungsanweisung mit Auswahl ON-GOTO

Die ON-GOTO-Anweisung, auch berechnete Sprunganweisung genannt, verzweigt in Abhängigkeit vom Wert eines numerischen Ausdrucks zu einer von mehreren möglichen Programmzeilen.
Sie hat die allgemeine Form: ON numerischer Ausdruck GOTO N1, N2, ..., NM.
Hat der numerische Ausdruck den Wert Ø oder 1, so ver-

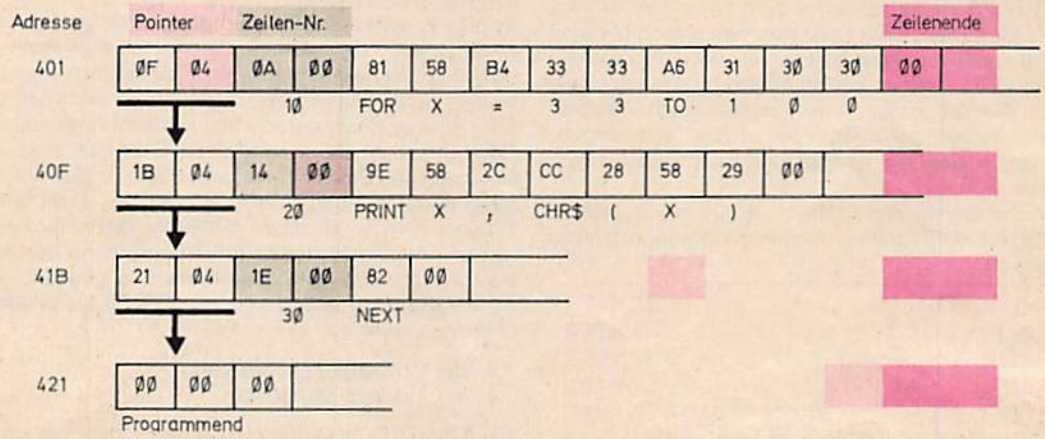
zweigt sich das Programm zur Zeile N1, beim Wert 2 zur Zeile N2 usw. Für M und Werte größer als M wird der Sprung der Zeile NM realisiert. Ist der Wert des Ausdrucks nicht ganzzahlig, so wird er vor der Auswertung auf den ganzzahligen Teil reduziert.
Eine Anwendung des eben Dargelegten finden Sie im Programm RADIUS 2. Bei ihm wird der Radius über INPUT eingegeben, und dann erscheint das folgende Menü auf dem Bildschirm:

```
1:Kreisdurchmesser
2:Kreisumfang
3:Kreisinhalt
4:Kugeloberfläche
5:Kugelvolumen ■
```

Jetzt wartet der Cursor hinter dem Kugelvolumen auf Eingabe einer Zahl nach A. Anschließend wird dieser Wert zur Verzweigung in ON A GOTO verwendet. In den entsprechenden Zeilen werden die Menügrößen über eine entsprechende Formel berechnet und dann angezeigt. Danach führt das Programm zur Eingabe des Radius zurück. Hier kann eine neue Auswahl getroffen werden. Die vorherigen Werte werden beibehalten, wenn nur mit ENTER die Abfrage bestätigt wird.
In diesem Programm sind 13 GOTO-Zeilennummern enthalten. Dadurch ist es nicht ganz einfach, die Struktur des Programms zu durchschauen. Versuchen Sie, es von diesem GOTO-Salat zu befreien!
Zum Abschluß des Kapitels möchten wir Ihnen noch das Programm KURS vorstellen. Hierbei ist der Kurs für Rubel, Kronen, Forint und Zloty einzugeben. Das Programm rechnet selbständig diese Währungen zum eingegebenen Kurs in die Währung Mark um bzw. umgekehrt. Das Programm benötigt zweimal die ON-GOTO-Folge, nämlich auch für die zweite Währung im Vergleich zur Mark.
Das Programm Kurs ist außerdem eine Demonstration der GOTO-Verflechtungen, denn es enthält sogar 18 GOTO-Zeilennummern.

Zusammenfassung Kapitel 7

1. Jede BASIC-Zeile hat zu Anfang zwei Byte, die auf die Adresse des Beginns der nächsten Zeile verweisen. Über diesen Pointer, zu deutsch Zeiger, sind alle BASIC-Zeilen verkettet.
2. Nach den ersten beiden Pointer-Byte folgen in jeder BASIC-Zeile zwei Byte, welche binär kodiert die Zeilennummer der Zeile enthalten.
3. Erst dann folgt der eigentliche BASIC-Text der Zeile, welcher mit einer Null abschließt.
4. Der GOTO-Befehl enthält eine Zeilennummer, zu der gesprungen werden soll. Hierbei sucht der Interpreter, vom Beginn des Programms angefangen, diese Zeilennummer. Dabei stößt er nach einiger Zeit entweder auf die richtige oder auf eine zu große Zeilennummer. Bei richtiger ist das Ziel erreicht, bei zu großer Zeilennummer erscheint UL-ERROR.
5. Für schnelle BASIC-Programme sollte darauf geachtet werden, daß GOTO-Sprünge möglichst vermieden werden oder aber zu niedrigen Zeilennummern führen.
6. Es sollte im Interesse einer kurzen Laufzeit der Programme immer versucht werden, GOTO-Befehlsfolgen durch entsprechende FOR-NEXT-Routinen zu ersetzen.
7. Der Zeitverlust eines GOTO-Sprunges besteht aus zwei Anteilen: zum einen für das GOTO allein, zum anderen für die Zeilensuche vom Beginn des Programms bis zur Soll-Zeile. In der Regel überwiegt letzterer Anteil.
8. Programme mit Kommentarzeilen, also REM, benötigen längere Laufzeiten. Es ist deshalb sinnvoll, je Programm zwei Varianten zu speichern: eine mit und eine ohne REM.
9. Die Befehlsfolge ON GOTO dient der Programmverzweigung. Sie besitzt das Format: ON Variable GOTO N1, N2, N3, ..., NM. Bei der Abarbeitung der Befehlsfolge erfolgt in Abhängigkeit des Wertes der Variablen (des arithmetischen Ausdrucks) ein bedingter Sprung. Ist die Variable gleich Ø oder 1, so erfolgt der Sprung in die Zeile N1, ist sie gleich 2, dann in die Zeile N2 usw. Für alle Werte der Variablen gleich oder größer M erfolgt der bedingte Sprung nach Zeile NM. Besitzt die Variable keinen Integerwert, so wird dieser vom Interpreter gebildet.




```
10 X=1: K=1.01: I=1
20 X=X*K
30 GOTO 20
40 !
50 REM: 58.3 s
60 REM: ## GOTO1 ##
```

```
10 X=1: K=1.01: I=1
20 X=X*K: I=I+1
30 GOTO 20
40 !
50 REM: 96.4 s
60 REM: ## GOTO2 ##
```

```
10 X=1: K=1.01: I=1
11 !
12 !
13 !
14 !
15 !
16 !
17 !
18 !
19 !
20 X=X*K
30 GOTO 20
40 !
50 REM: 67.1 s
60 REM: ## GOTO3 ##
```

```
10 X=1: K=1.01: INPUT"Zahl =";A
15 FOR I=1 TO 9000 STEP A
20 X=X*K
30 NEXT
40 !
50 REM: A=0 59.1 S, sonst ca. 63.7 s
60 REM: ## GOTO4 ##
```

```
10 X=1: K=1.01: I=1
20 X=X*K
22 GOTO 24
24 GOTO 26
26 GOTO 28
28 GOTO 30
30 GOTO 20
40 !
50 REM: 104.8 s
60 REM: ## GOTO5 ##
```

```
10 X=1: K=1.01: I=1
20 X=X*K
30 GOTO 240
40 !
50 !
60 !
70 !
80 !
90 !
100 !
110 !
120 !
130 !
140 !
150 !
160 !
170 !
180 !
190 !
200 !
210 !
220 !
230 !
240 GOTO 20
250 !
260 REM: 89.8 s
270 REM: bei 30 ! 187.2 s
280 REM: ## GOTO6 ##
```

```
10 PRINT TAB(10);"## BENCH ##"
20 K=0: DIM M(5)
30 K=K+1
40 A=K/2*3+4-5
50 GOSUB 110
60 FOR L=1 TO 5
70 M(L)=A
80 NEXT
90 IF K<1000 THEN 30
100 END
110 RETURN
```

```
10 CLS: PRINT TAB(9);"## RADIUS2 ##"
20 PRINT: INPUT"Radius =";R
30 PRINT "1:Kreisdurchmesser"
40 PRINT "2:Kreisumfang"
50 PRINT "3:Kreisinhalt"
60 PRINT "4:Kugeloberflaeche"
70 INPUT "5:Kugelvolumen";A
80 ON A GOTO 90, 100, 110, 120, 130, 140
90 X=R+R: GOTO 150
100 X=2*PI*R: GOTO 150
110 X=PI*R*R: GOTO 150
120 X=4*PI*R*R: GOTO 150
130 X=4*PI*R*R*R/3: GOTO 150
140 PRINT"Eingabefehler": GOTO 20
150 PRINTX: GOTO 20
```

```
10 CLS: PRINT TAB(9);"## KURS ##": PRINT
20 PRINT" 1 Mark ist gleich"
30 INPUT"Rubel";R
40 INPUT"Kronen";K
50 INPUT"Forint";F
60 INPUT"Sloty ";S
70 PRINT: PRINT"Umrechnung"
80 PRINT"1:Mark nach andere"
90 INPUT"2:andere nach Mark";A: PRINT
100 PRINT"1:Rubel  2:Kronen"
110 INPUT"3:Forint 4:Sloty ";B
120 ON A GOTO 130, 190
130 INPUT"Mark =";M
140 ON B GOTO 150, 160, 170, 180
150 PRINT M*R; "Rubel": GOTO 70
160 PRINT M*K; "Kronen": GOTO 70
170 PRINT M*F; "Forint": GOTO 70
180 PRINT M*S; "Sloty": GOTO 70
190 ON B GOTO 200, 210, 220, 230
200 INPUT"Rubel =";X: M=X/R: GOTO 240
210 INPUT"Kronen =";X: M=X/K: GOTO 240
220 INPUT"Forint =";X: M=X/F: GOTO 240
230 INPUT"Sloty =";X: M=X/S
240 PRINT M; "Mark": GOTO 70
```

8. Die Verzweigungsanweisung IF-THEN und ihre Modifikationen

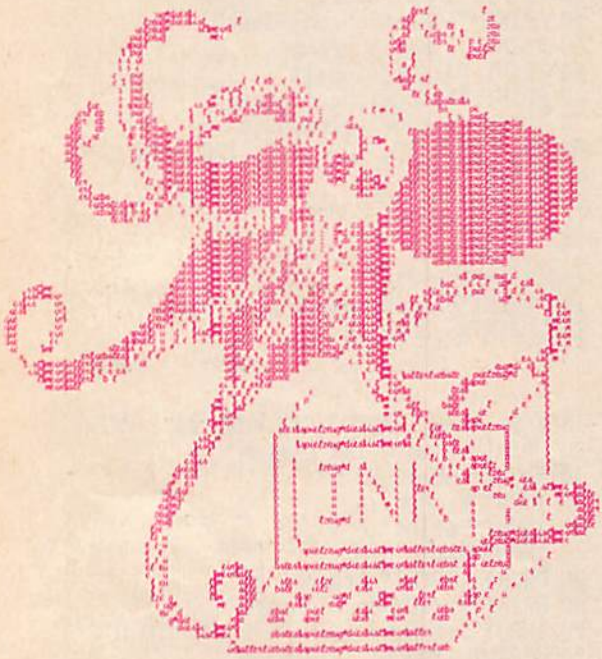
Oft sind Algorithmen so gestaltet, daß in Abhängigkeit vom Ergebnis eines Vergleichs zwei verschiedenen Anweisungsfolgen auszuführen sind (Programmverzweigung). Verzweigungen lassen sich in BASIC mit der IF-THEN-Anweisung (engl.: if ... then – wenn ... dann) realisieren. Sie hat das Format:

IF Ausdruck Vergleichsoperator Ausdruck THEN Handlung

Wenn der Vergleichsoperator den Wert „wahr" hat, dann wird die Programmausführung mit der nach THEN bezeichneten Handlung ausgeführt. Anderenfalls hat diese Anweisung keine Wirkung, und als nächste wird die Anweisung der folgenden Programmzeile ausgeführt.

8.1. Die IF-GOTO-Anweisung

Mit Hilfe der IF-Anweisung teilen wir unserem Computer die Entscheidungskriterien mit, d. h., wir sagen ihm, in welchem Fall er was zu tun hat.



Schauen wir uns dazu ein kleines Beispiel an. Hier soll der Computer eine erteilte Note, die in das Programm eingegeben wird, mit „gut" oder „schlecht" bewerten. Dabei werden die Noten 1 und 2 pauschal mit „gut" und die anderen mit „schlecht" bewertet. Geben Sie ein:

```
10 INPUT A                          (ENTER)
20 IF A=1 OR A=2 GOTO 50            (ENTER)
30 PRINT "SCHLECHT"                 (ENTER)
40 END                              (ENTER)
50 PRINT "GUT"                      (ENTER)
60 END                              (ENTER)
```

Mit folgendem Kommentar ist das Programm sofort verständlich:
Wenn A gleich 1 oder 2 ist, so gehe zur Zeile 50, ansonsten arbeite die nächste Zeile ab.
Die IF-GOTO-Anweisung funktioniert verallgemeinert wie folgt:
IF X GOTO n
Wenn der Ausdruck X wahr ist, so gehe zur Programmzeile n, ansonsten arbeite die nächste Zeile ab. Nebenbei bemerkt, haben Sie eben Bekanntschaft mit einem der Booleschen Operatoren gemacht (mehr dazu im Kapitel 12).

8.2. Die IF-THEN-Anweisung

Es gibt auch noch eine zweite Variante der IF-Anweisung. Diese sieht verallgemeinert wie folgt aus:
IF X THEN Anweisung
Das bedeutet inhaltlich: Wenn der Ausdruck X wahr ist, dann führe die hinter THEN stehende Anweisung aus, ansonsten ignoriere die Anweisung und fahre mit der Abarbeitung der nächsten Zeile fort. Statt einer Anweisung kann auch eine Zeilennummer stehen, zu der bei erfüllter Bedingung gesprungen wird. GOTO kann hierbei entfallen. Der Anweisung nach THEN können weitere durch Doppelpunkt getrennte Anweisungen folgen. Diese werden jedoch auch nur dann ausgeführt, wenn die vorherstehende Bedingung erfüllt wurde.
Wir wollen nun diese Befehlsfolge für die Konstruktion einer Digitaluhr verwenden.
Unser Rechner soll also so arbeiten wie eine Digitaluhr, d. h., ständig die richtige Uhrzeit anzeigen. Sie erhalten dazu am Ende des Kapitels das Programm UHR.
Die Funktionsweise ist folgende:
Es gibt drei Variablen:
- S enthält die Stunde,
- M trägt die Minutenwerte, und
- T ist für die Sekunden zuständig.

Weiter existiert eine Zeitschleife, die zusammen mit den anderen Programmteilen genau 1 s benötigt. Nach Ablauf der Zeitschleife wird T um 1 erhöht.
Dieser Teil lautet:

```
70 FOR X=1 TO 433: NEXT: T=T+1      (ENTER)
```

Die Zahl 433 bestimmt die Ganggenauigkeit der Uhr und kann durch Sie selbständig noch etwas verbessert werden. Es sind aber nur ganzzahlige Werte sinnvoll. Deshalb ist die Genauigkeit nicht zu weit zu treiben.
Was muß nun aber geschehen, wenn 60 s erreicht werden? Dann muß T auf Null gestellt werden, und der Minutenwert ist gleichzeitig zu erhöhen. Also:

```
80 IF T>59 THEN T=0: M=M+1          (ENTER)
```

Das gleiche muß auch für die Minuten gelten:

```
90 IF M>59 THEN M = 0 :S = S + 1    (ENTER)
```

Natürlich könnte man jetzt das Datum usw. berechnen lassen. Wir vermuten aber, daß Sie den Rechner nicht tage- oder gar wochenlang ausschließlich als Digitaluhr verwenden wollen. Deshalb hören wir im Programm bei den Stunden auf und zeigen den Wert in der Bildschirmmitte an:

```
PRINT AT (14,5); S; "Uhr"; M; "Minuten"; T;
"Sekunden"  (ENTER)
```

Hinter Sekunden fügen wir noch mehrere Space ein, damit die alte Zeit mit eventuell längeren Zeitangaben überschrieben werden kann.
Jetzt sind nur noch Schönheitsergänzungen für unser Programm notwendig. Natürlich muß die Uhr beim Start durch eine Eingabe gestellt werden. Auch wäre eine schöne Überschrift am oberen Teil des Bildschirms gut. Damit dürften Sie das angegebene Programm UHR ohne Schwierigkeiten verstehen und können es eventuell auch erweitern und ergänzen. Das Programm ist bewußt im GOTO-Salat geschrieben. Versuchen Sie es bitte im Sinne des vorigen Kapitels weitgehend auf FOR-NEXT-Schleifen umzustellen. Dann gewinnen Sie Zeit, und der Wert 433 in Zeile 70 muß größer gewählt werden. Damit ist eine höhere Genauigkeit erreichbar.

Nun wollen wir uns dem besonders interessanten Programm HYDRA zuwenden. Stellen Sie sich bitte ein Fabelwesen vor. Es besitzt eine bestimmte Anzahl von Köpfen, die wir durch Eingabe in das Programm festlegen. Ein Ritter hat nun mit diesem Fabelwesen zu kämpfen, und er hat gesiegt, wenn er dem Tier so viele Köpfe abgeschlagen hat, daß es nur noch einen besitzt. Doch in diesem Kampf gelten strenge Regeln. Hat die Hydra eine gerade Anzahl von Köpfen, so kann der tapfere Ritter genau die Hälfte abschlagen. Besitzt sie danach jedoch eine ungerade Anzahl, so wachsen ihr sofort wieder 3mal so viele und einer zusätzlich nach. Damit ist die Anzahl wieder gerade und der Ritter schlägt dem Wesen wieder die Hälfte der Köpfe ab. In diesem Spiel nimmt also die Anzahl der Köpfe ab oder zu, und zwar in ziemlich unregelmäßiger Folge. Das Interessante an diesem Problem ist, daß es der Mathematik bis heute nicht zu beweisen gelang, ob bei jeder Anfangszahl, der Ritter die Chance hat, das Wesen zu töten. Dies soll uns aber erst in einem späteren Kapitel interessieren. Hier geht es uns zunächst nur um das IF-THEN.
Mit X als Anzahl der Köpfe gilt:
IF X gerade THEN X = X/2
IF X ungerade THEN X = 3*X + 1
und der Test auf Sieg bedeutet:
IF X = 1 THEN END
Auf gerade prüfen wir nun durch Halbieren der Zahl, also:
IF X/2 = INT (X/2) THEN
Damit ist dieses Programm für Sie leicht lesbar, ja Sie können es vielleicht sogar schon selbst schreiben. Versuchen Sie es einmal, bevor Sie sich das am Ende des Kapitels angegebene Programm ansehen.
Schauen Sie sich dann beim Ablauf des Programms einmal die Zahlenfolge etwas genauer an. Interessant sind z. B. jene Startzahlen für die Köpfe, die zu einem langen Lauf, also zu einer langen Zahlenkette führen. Vielleicht ändern Sie das Programm auch so ab, daß Sie automatisch jene Eingaben erhalten, bei denen der Kampf mehr als 20, 50 oder 70 Schwertschläge des Ritters verlangt.

8.3. Zu einigen Programmen mit Anwendng von Grafikbefehlen

Das „Standardformat" des Bildschirms, welches stets nach dem Einschalten des Computers KC 85/2 bzw. 3 eingestellt ist, gliedert sich in 30 Schriftzeilen mit je 40 Zeichen.
Darüber hinaus können wir aber noch die verschiedensten Bildausschnitte, also Fenster (engl.: window), einstellen. Das ist der erste neue Grafikbefehl:
WINDOW ZS, ZZ, SS, SZ.
Dieser Befehl stellt dann also ein Fenster auf dem Bildschirm ein, auf das allein im weiteren ein direkter Zugriff besteht (eine Ausnahme bildet der Befehl PRINT AT).
Es bedeuten: ZS die Anfangszeile, ZZ die Zeilenzahl, SS die Anfangsspalte, SZ die Spaltenzahl.
Der Befehl WINDOW ist im Gegensatz zu den noch folgenen beiden Grafikbefehlen PSET und PRESET auch beim KC 85/1 und KC 87 anwendbar.

Der KC 85/2 und der KC 85/3 bieten Ihnen einen für einen Kleincomputer beispielhaften Grafikkomfort. Wir sind in der Lage, das Fernsehbild in 320 Punkte in der Waagerechten und 255 Punkte in der Senkrechten aufzulösen. Dadurch ist es möglich, mathematische Funktionen oder vom Computer überwachte Prozesse grafisch darzustellen.
Wie zeichnen wir nun konkret?
Jeder einzelne Punkt (Pixel) ist ansprechbar durch seine Koordinaten. Die X-Koordinate gibt den Abstand vom linken Bildschirmrand (von links nach rechts von 0 bis 319) und die Y-Koordinate den Abstand vom unteren Bildschirmrand (von unten nach oben von 0 bis 255) des jeweiligen Punktes an.
Wollen wir nun einen Punkt mit den Koordinaten X und Y in einer bestimmten Farbe zeichnen, so geben wir diese in Form der Anweisung
PSET X-Koordinate, Y-Koordinate, Farbcode
ein.
So, wie wir einen bestimmten Punkt setzen können, ist es auch möglich, diesen mit der Anweisung PRESET zu löschen:
PRESET X-Koordinate, Y-Koordinate
Mit diesen beiden Befehlen können wir nun bequem auf dem Bildschirm malen (vgl. Programm GRAFIK).
Wir definieren dazu noch die Richtungen 1 bis 8. Die Zahl 1 weist nach oben, 3 nach rechts, 5 nach unten usw. Sobald die Richtungen auf diese Weise festgelegt sind, kann man in dieser Richtung mit einem Miniatur-Cursor auf dem Bildschirm wandern. Dabei sind drei Fälle für das Wandern zu unterscheiden.
G: Der Punkt wandert nur und bewirkt auf dem Bildschirm nichts
S: Der Punkt wandert und hinterläßt auf dem Bildschirm seine Spur
R: Der Punkt wandert und löscht alles was auf seiner Spur war.
Mit diesen wenigen Befehlen ist es möglich, wunderschöne Strichzeichnungen anzufertigen. Das zugehörige Programm GRAFIK zeigt Ihnen in einem kleinen Fenster rechts oben, was Sie tun (G, S, R), und rechts unten die Richtung, in der Sie es tun. Mit 0 können Sie darüber hinaus die Art Ihres Tuns ändern. Die Richtungen und Arten werden natürlich mit IF-THEN angewählt.
Das Verständnis des Programms MOSAIK setzt bereits gewisse Programmierkenntnisse voraus. Versuchen Sie es bitte zu beschreiben.

Wenn Sie dieses Programm starten, entsteht auf dem Bildschirm eine eigenartig strukturierte Folge von Zeichen. Nach einer Weile kommt dieser spannende Ablauf zur Ruhe. Er kann aber für einen anderen zufälligen Ablauf durch Betätigen irgendeiner Taste wieder in Gang gesetzt werden. In diesem Programm sind auf eigenwillige Weise Zufall und Gesetzmäßigkeit gekoppelt, und das bewirkt die erregenden Abläufe.
Das Programm OSTERN schließlich gestattet es Ihnen, das Osterdatum für jedes beliebige Jahr zu berechnen. Hierbei sind sehr viele Bedingungen zu erfassen. Wir sind im Programm allerdings mit 4 IF-Anweisungen ausgekommen, und dabei ist es sogar noch gleichgültig, ob Sie für das Jahr beispielsweise 1987 oder bloß 87 eingeben. Es gilt jedoch nur für einen Zeitraum von 1850 bis 2000.

Zusammenfassung Kapitel 8

1. Eine Bedingung ist eine Aussage, die entweder wahr oder falsch sein kann. In BASIC verwendet man u. a. Zahlenvergleiche dafür, also:
   X = 0, X < I, X = A, X < A usw.
   Es sind aber auch Stringvergleiche wie
   A$ = "JA" zugelassen
   In einer IF-THEN-Anweisung wird der nach THEN stehende Befehl ausgeführt, wenn die Bedingung nach IF wahr ist. Ist die Bedingung falsch, so geht der Interpreter zur nächsten Programmzeile über.
2. Die Folge IF – Bedingung kann auf drei Arten fortgesetzt werden:
   - mit THEN und einem Befehl, z. B. Y = 3 X+1
   - mit GOTO und einer Zeilennummer
   - mit THEN und einer Zeilennummer
3. Bei THEN und Befehl können nach THEN auf derselben Zeile, jedoch durch Doppelpunkt getrennt, auch mehrere Befehle und zum Schluß sogar noch ein GOTO folgen. Auf diese Weise ist die IF-THEN-Folge sehr ausdrucksstark.
4. WINDOW ZS, ZZ,, SS, SZ bewirkt, daß auf dem Bildschirm ein Ausschnitt folgender Größe, beginnend vom linken oberen Bildrand, ausgewählt wird:
   ZS – Anfangszeile ZZ – Zeilenzahl
   SS – Anfangsspalte SZ – Spaltenzahl
   Nach der Anweisung WINDOW kann mit dem üblichen PRINT-Befehl nur auf diesen Bildschirmausschnitt zugegriffen werden. Eine Ausnahme bildet jedoch der Befehl PRINT AT.
5. Bezüglich der Zeilenzahl weichen der KC 85/1 bzw. KC 87 vom KC 85/2 und KC 85/3 ab. Dies ist bei der Anwendung von WINDOW zu beachten.

Zeiten aus den GOTO-Programmen

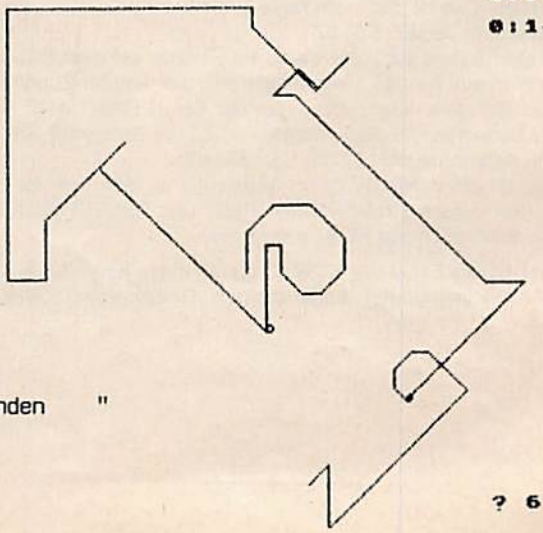
(2) – (1) = 38.1 s / 8779 Additionen
je Addition also 4.3 ms
(3) – (1) = 8.8 s/ 9 REM / 8779
je Zeile suchen .1 ms
FOR-NEXT mit GOTO im Vergleich
59.1 58.3 ohne Addition bzw. Add. 0
63.7 96.4 mit Addition
(5) – (1) = 46.5 s/ 4 GOTO / 8779
je GOTO ca. 1.3 ms
Abzug für Zeilensuche ca..8 ms – ca..5 ms
(6b) – (6a) = 97.4 s / 21 REM / 8779 je REM erkennen .5 ms

```
10 CLS: PRINT TAB(9)"## HYDRA ##"
20 INPUT"Startzahl =";X
30 IF X/2=INT(X/2) GOTO 70
40 X=3*X+1
50 PRINT X,
60 GOTO 30
70 X=X/2
80 PRINT X,
90 IF X=1 THEN END
100 GOTO 30
```

```
        ## HYDRA ##
Startzahl = 45
 136          68          34
 17           52          26
 13           40          20
 10           5           16
 8            4           2
 1
```

```
10 REM: ## GRAFIK ##
20 CLS: X=150: Y=120: PSET X,Y,7
30 !    Fenster rechts oben
40 WINDOW 0,2,33,39: CLS
50 !    G=gehen, S=setzen, R=ruecksetzen
60 INPUT "G:S:R";B$
70 !    Richtung 1=hoch 3=rechts 5=runter
80 !    7=links: 0=neu festsetzen
90 PRINT "0:1-8";
100 !   Fenster rechts unten
110 WINDOW 30,31,33,39
120 INPUT C: A=0: B=0
130 !   Festlegung der Aenderungen
140 IF C=0 GOTO 40
150 IF C=1 THEN A=1
160 IF C=2 THEN A=1: B=1
170 IF C=3 THEN      B=1
180 IF C=4 THEN A=-1:B=1
190 IF C=5 THEN A=-1
200 IF C=6 THEN A=-1:B=-1
210 IF C=7 THEN      B=-1
220 IF C=8 THEN A=1: B=-1
230 !   alte und neue Werte
240 XA=X: YA=Y: X=X+B: Y=Y-A
250 !   ausfuehren
260 IF B$="S" THEN PSET   X,Y,7
270 IF B$="R" THEN PRESET X,Y
280 IF B$="G" THEN PRESET XA,YA: PSET X,Y,7
290 GOTO 110
```

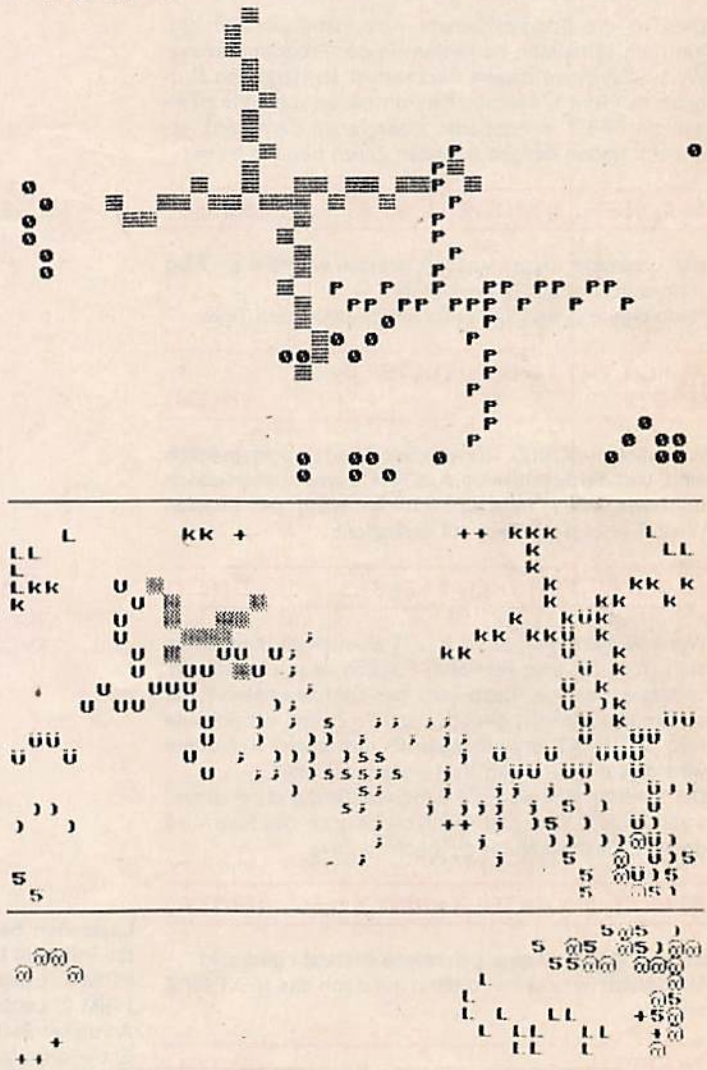
```
G:S:R
0:1-8
```



```
? 6
```

```
10 CLS: T=2: REM: ## UHR ##
20 PRINT"Eingabe der Uhrzeit"
30 PRINT"Start bei 0 Sekunden"
40 INPUT"Stunde,Minute ";S,M: CLS
50 PRINT TAB(10);"## DIGITALUHR ##"
60 PRINT AT (14,5);S;"Uhr";M"Minuten";T"Sekunden    "
70 FOR X=1 TO 433: NEXT: T=T+1
80 IF T>59 THEN T=0: M=M+1
90 IF M>59 THEN M=0: S=S+1
100 GOTO 60
```

```
10 REM: ## MOSAIK ##
20 WINDOW 0,31,0,39:CLS: RANDOMIZE
30 A=INT( RND(1) *1230): B=A+1: C=B-40: D=C+1
40 A$=CHR$ (RND(1) *224 +32)
50  FOR I=0 TO 5 *RND(1)
60   FOR J=0 TO INT(3 *RND(1) )
70    A=ABS(A+J-41): B=ABS(B+40*J-39): C=ABS(C-J+41): D=ABS(D-40*J+39)
80    IF A>1225 THEN A=A-78
90    IF B>1225 THEN B=B-78
100   IF C>1225 THEN C=C-78
110   IF D>1225 THEN D=D-78
120   ZA=INT(A/40): SA=A-40*ZA: ZB=INT(B/40): SB=B-40*ZB
130   ZC=INT(C/40): SC=C-40*ZC: ZD=INT(D/40): SD=D-40*ZD
140   PRINT AT(ZA,SA); A$: PRINT AT(ZB,SB); A$
150   PRINT AT(ZC,SC); A$: PRINT AT(ZD,SD); A$
160  NEXT
170 NEXT:IF RND(1)<.8 GOTO 30
180 INPUT"";Z: CLS: GOTO 40
```



```
10 CLS:PRINT TAB(13);"## OSTERN ##"
20 PRINT TAB(13);"============":PRINT
30 INPUT "Jahr =";J: IF J<100 THEN J=J+1900
40 P=INT(J/100): N=J-P*100: Q=INT(P/3): R=INT(P/4)
50 X=15+P-Q-R: X=X-INT(X/30)*30: Y=P+4-R: Y=Y-INT(Y/7)*7
60 A=J-INT(J/19)*19: B=J-INT(J/4)*4: C=J-INT(J/7)*7: D=19*A+X
70 D=D-INT(D/30)*30: E=B+B+4*C+6*D+Y: E=E-INT(E/7)*7
80 IF D+E<=9 THEN OT=22+D+E: OM=3: GOTO 120
90 IF D=28 AND E=6 AND A>10 THEN OT=18: OM=4: GOTO 120
100 IF D=29 AND E=6 THEN OT=19: OM=4: GOTO 120
110 OT=D+E-9: OM=4
120 PRINT OT;"."; OM;".": PRINT: GOTO 30
```

9. Vorzeitiges Verlassen einer FOR-NEXT-Schleife

Mit der Laufanweisung FOR-TO-NEXT haben wir uns bereits in Kapitel 4 befaßt. Hier sind grundsätzliche Ausführungen zum Gebrauch dieser Anweisung gemacht worden. In diesem Kapitel wollen wir uns der Problematik des vorzeitigen Verlassens einer FOR-NEXT-Schleife zuwenden.
Es existieren drei Möglichkeiten, um die FOR-NEXT-Schleife vor dem Erreichen des Endwertes verlassen zu können.

9.1. Setzen des Laufparameters auf seinen Maximalwert mittels der IF-THEN-Anweisung

Dies ist die übersichtlichste Austrittsmöglichkeit. Sie führt am seltensten zu Fehlern in der Programmierung. Wir wollen Ihnen diesen Sachverhalt an Hand des Programms PRIM 1, welches die Primzahlen von 5 bis zu einer mit INPUT eingebbaren Obergrenze berechnet, erläutern. Neben einigen formalen Zeilen beginnt es mit

```
30 FOR Z=5 TO N STEP 2: V=0          (ENTER)
```

Nur ungerade Zahlen werden geprüft, wobei V ein Flag (Kennzeichen) auf Teilbarkeit ist.
Nach dieser Zeile folgen nun die zu testenden Teiler:

```
40 FOR T=3 TO SQR(Z) STEP 2:
W=Z/T                                (ENTER)
```

Wir wählen SQR(Z), da größere Werte nicht möglich sind, und die Schrittweite zu STEP 2, weil ausschließlich ungerade Teiler zulässig sind. An Hand der Division W=Z/T erfolgt der Test auf Teilbarkeit:

```
50 IF W=INT(W) THEN T=Z: V=1         (ENTER)
```

Wenn W ganzzahlig ist, d. h. Z/T also teilbar, handelt es sich nicht um eine Primzahl. Folglich kann die Schleife verlassen werden. Dazu wird der Laufparameter T auf seinen Maximalwert gesetzt, also T=Z, und die Schleife wird bei NEXT ordnungsgemäß verlassen. Außerdem wird dies mit dem Flag V=1 gekennzeichnet.
Die Schleife wird also nur dann vollständig durchlaufen, wenn Z nicht teilbar ist. Dann bleibt auch das Flag V=0 gesetzt. Deshalb lautet die nächste Zeile:

```
60 NEXT: IF V=0 THEN PRINT Z         (ENTER)
```

Es wird also bloß eine gefundene Primzahl gedruckt.
Mit der letzten Zeile wird dann nur noch das NEXT für Z erhöht:

```
70 NEXT                              (ENTER)
```

9.2. Setzen der inneren Laufvariablen auf den Wert der höheren Schleifenstufe mittels der IF-THEN-Anweisung

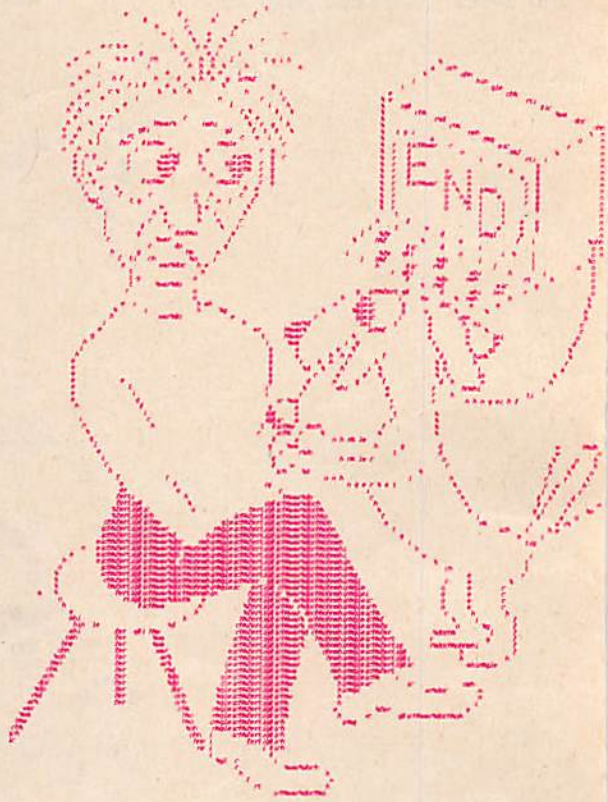
Wir hatten schon in Kapitel 4 die Möglichkeit der falschen Verwendung einer Variablen in der Verschachtelung von Laufanweisungen aufgezeigt. Das hatte zur Folge, daß die „richtige" Variable vom Stack geworfen wurde. Dies läßt sich nun auch für das vorzeitige Verlassen der Schleife verwenden. Eine so variierte Version unseres Primzahlprogramms ist das Programm PRIM 2. Die Zeile 50 lautet hierbei:

```
50 IF W=INT(W) THEN NEXT Z: END      (ENTER)
```

Im Falle der Teilbarkeit wird hier sofort mittels der IF-THEN-Anweisung auf die Laufvariablen der höheren Schleifenstufe übergegangen. Dazu benötigen wir auch nicht mehr das Flag V. Wir müssen aber hinter NEXT Z ein END einfügen.
Zeile 60 ist wie folgt zu ändern:

```
60 NEXT: PRINT Z, : NEXT             (ENTER)
```

und die Zeile 70 entfällt. Das Programm ist damit sogar noch kürzer geworden. Dies wird außerdem bei den



Laufzeiten beider Programme deutlich. Berechnen wir die Primzahl bis 400, so gilt:
PRIM 1: Laufzeit 26.8 s
PRIM 2: Laufzeit 24.6 s
An dieser Stelle erhebt sich nun die Frage, was mit dem Schleifenparameter wird, wenn man einfach die aktuellen Werte vom Stack fegt. Um dies für Sie zugänglich zu machen, haben wir das Programm NEXT 1 geschrieben. Es prüft, welchen Wert die Variable des vom Stack gefegten Laufparameters nachher besitzt. Dazu wird die Variable vorher und nachher ausgedruckt. Hier einige erklärende Bemerkungen: X und Y werden variiert, und die Variable I wird immer um den Faktor 1.1 erhöht. Wird I>20 erreicht, so wird zunächst Y gedruckt und dann mit der THEN-Anweisung NEXT X gewählt. Dadurch wird Y vom Stack entfernt. Dennoch behält die Variable Y den letzten Wert bei, wie ein erneuter Ausdruck von Y zeigt. Demzufolge kann der zuvor aktuelle Wert Y weiter genutzt werden.

9.3. Verlassen einer FOR-NEXT-Schleife mittels der GOTO-Anweisung

Es ist eine gefährliche Möglichkeit, mittels der GOTO-Anweisung beim Erreichen einer Bedingung aus der FOR-NEXT-Schleife zu springen. Hierbei können Stack-Probleme auftreten, welche zu einem fehlerhaften Ablauf des Programms führen können. Deshalb sollte dieser Weg nur ausnahmsweise gegangen werden (vgl. Programm NEXT 2). Die äußerste Schleife mit der Laufvariablen X wird hier nur 2mal durchlaufen und nicht, wie es richtig wäre, 5mal.

Wenn keine Schleifenschachtelung vorliegt, treten selbstverständlich mit der GOTO-Anweisung in einer Laufschleife keine Probleme auf. Die Gefahr wird lediglich dann ziemlich groß, wenn in ungünstig geschachtelten Schleifen gearbeitet wird.

Verwendet man jedoch konsequent die Variable hinter NEXT, stellt sich auch in geschachtelten Schleifen der Stack automatisch richtig ein.

Mit dem Programm NEXT 3 sollten Sie ein wenig experimentieren, um so den Eigenschaften des vorzeitigen Verlassens einer FOR-NEXT-Schleife auf die Spur zu kommen. Wenn Sie in Zeile 20 z. B. STEP 1 + 1/I ergänzen, ändert sich das Verhalten sehr. Jedesmal wird dann die Schleife mit neuer Schrittweite durchlaufen. Sie können auch in der Schleife den Laufparameter zurücksetzen, wenn eine bestimmte Bedingung erfüllt ist. Wir wünschen Ihnen bei derartigen Experimenten viel Vergnügen.

9.4. Spezialprogramm zu chaotischen Systemen

Das Programm APFEL verwendet eine hochauflösende Grafik und läuft nur auf dem KC 85/2 und KC 85/3.

Das Programm behandelt das berühmte Apfelmännchen.

Wir möchten unseren Ausführungen zu diesem Programm ein klein wenig Fachwissen voranstellen.

Es gibt einfache Gleichungen, die stark von der Wahl der Parameter abhängen. Solche Gleichungen liegen u. a. bei Wettermodellen vor. Das allereinfachste Beispiel ist wohl

X = R*X*(X–1)

Diese Gleichung wird in eine FOR-NEXT-Schleife eingebaut. Ihr Verhalten hängt dann ganz wesentlich vom Parameter R ab. Für R > 3.5 und R < = 4 ergeben sich die interessanten Werte:

- R = 4 Chaosverhalten
- R < 4 mehr oder weniger regelmäßiges Schwanken zwischen zwei Knotenpunkten
- R > 4 schnelles Wachsen über alle Grenzen

Das Apfelmännchen ist eine ähnliche Folge für X- und Y-Werte. Die Eigenschaften der X- und Y-Parameter können jetzt auf unterschiedliche Art zur grafischen Darstellung genutzt werden. Genau dies tut unser Apfelmännchen. Es besteht aus den stabilen Punkten. Lassen Sie dieses Programm einmal laufen. Sie werden an den Bildern Ihre Freude haben. Interessant ist es auch, sich Teilstrukturen anzuschauen. Doch hier gleich eine Warnung: Jeder Bildaufbau benötigt – trotz relativ effektiver Programmierung – ca. 3 h. Damit Sie den Aufbau des Bildes verfolgen können, haben wir einen speziellen Pixelcursor für Sie hinzugefügt. Hier einige Beispielwerte für die Eingaben: Das Apfelmännchen liegt im Bereich von –.8 < X < 2.4 und -1.4 < Y < .4. Ein interessanter Ausschnitt ist z. B. 1 < X < 1.5 und 2 < Y < 1.4.

Schnellere Bilder, aber auch mit weniger Auflösung, bekommen Sie durch Änderungen in den Zeilen 40, 50, 100 und 110.

Die entscheidende Gleichung steht in der Zeile 130.

Wir wünschen Ihnen Geduld und Freude bei diesem Programm.

Zusammenfassung Kapitel 9

1. Eine FOR-NEXT-Schleife kann vorzeitig verlassen werden. Wenn die geforderte Bedingung erfüllt ist, wird der zugehörige Laufparameter auf seinen Maximalwert oder größer gesetzt. Dadurch wird die Schleife ordnungsgemäß beim Erreichen des NEXT verlassen. Diese Maßnahme ist deshalb so günstig, weil damit zugleich der Stack bereinigt wird und bei späteren Rechnungen nicht mehr stören kann.
2. Es ist zuweilen notwendig, den Wert der Schleifenvariablen bzw. andere Größen beim Erreichen der Bedingung zu retten. Sie müssen dann nach der THEN-Anweisung auf geeignete Variablen übergeben werden.
3. Die innere von geschachtelten FOR-NEXT-Schleifen kann bei erfüllter Bedingung dadurch vorzeitig verlassen werden, daß nach der Anweisung THEN ein NEXT mit der Variablen der äußeren Schleife folgt.
4. Es ist möglich, eine FOR-NEXT-Schleife einfach durch vorzeitiges Herausspringen mittels einer GOTO-Anweisung zu verlassen. Hierbei können Stackprobleme auftreten, welche zu einem unüberschaubaren bzw. fehlerhaften Programmablauf führen. Deshalb sollte dieser Weg nur in Ausnahmefällen beschritten werden.
5. Das einfache Herausspringen mit GOTO aus einer Schleife ist mit gewisser Vorsicht unter folgenden Bedingungen zulässig:
   – wenn keine Schleifenverschachtelung vorliegt oder
   – wenn konsequent Variablen hinter NEXT verwendet werden.

```
10 CLS: PRINT TAB(10);"## PRIM1 ##": PRINT
20 INPUT"Primzahlen bis zu =";N
30  FOR Z=5 TO N STEP 2: V=0
40    FOR T= 3 TO SQR(Z) STEP 2: W=Z/T
50     IF W = INT(W) THEN T=Z: V=1
60    NEXT: IF V=0 THEN PRINT Z,
70  NEXT
```

```
10 CLS: PRINT TAB(10);"## PRIM2 ##": PRINT
20 INPUT"Primzahlen bis zu =";N
30  FOR Z=5 TO N STEP 2
40    FOR T= 3 TO SQR(Z) STEP 2: W=Z/T
50     IF W = INT(W) THEN NEXT Z: END
60    NEXT: PRINT Z,: NEXT
```

```
10 CLS: PRINT TAB(10);"## NEXT1 ##": PRINT
20 PRINT"X","Y","Y in 30"
30  FOR X=1 TO 20: PRINT Y: I=X: PRINT X,
40    FOR Y=1 TO 40: I=I*1.1
50      IF I>20 THEN PRINT Y,: NEXT X: END
60  NEXT: NEXT
run
```

```
          ## NEXT1 ##

X             Y             Y in 30
 0
 1             32            32
 2             25            25
 3             20            20
 4             17            17
 5             15            15
 6             13            13
 7             12            12
 8             10            10
```

```
10 CLS: PRINT TAB(10);"## NEXT2 ##": PRINT
20  FOR X=1 TO 5
30   A=A+1: PRINT"X:";X;Y;A,
40    FOR Y=1 TO 8
50     A=A+1: PRINT"Y:";X;Y;A,
60     IF A>11 GOTO 80
70    NEXT
80   A=A+2: PRINT"Z:";X;Y;A,
90  NEXT

RUN
          ## NEXT2 ##

X: 1  0  1    Y: 1  1  2    Y: 1  2  3
Y: 1  3  4    Y: 1  4  5    Y: 1  5  6
Y: 1  6  7    Y: 1  7  8    Y: 1  8  9
Z: 1  9  11   X: 2  9  12   Y: 2  1  13
Z: 2  1  15   Y: 2  2  16   Z: 2  2  18
Y: 2  3  19   Z: 2  3  21   Y: 2  4  22
Z: 2  4  24   Y: 2  5  25   Z: 2  5  27
Y: 2  6  28   Z: 2  6  30   Y: 2  7  31
Z: 2  7  33   Y: 2  8  34   Z: 2  8  36
```

```
10 CLS: PRINT TAB(10);"## NEXT3 ##": PRINT
20  FOR X=0 TO I
30   PRINT X;
40  NEXT
50 I=I+1: IF I<10 THEN PRINT: GOTO20
60 ! in 50 ist statt I=I+1 auch moeglich I=X

RUN
          ## NEXT3 ##

 0
 0  1
 0  1  2
 0  1  2  3
 0  1  2  3  4
 0  1  2  3  4  5
 0  1  2  3  4  5  6
 0  1  2  3  4  5  6  7
 0  1  2  3  4  5  6  7  8
```

10. Die Anweisung IF-THEN-ELSE

Wir haben uns in Kapitel 8 mit der Verzweigungsanweisung IF-THEN befaßt. Es existiert noch die Möglichkeit, eine alternative Anweisung in die IF-Anweisung einzubauen. Dies geschieht mit Hilfe des Anweisungsteils ELSE (engl.: else – sonst), der an die uns schon bekannten Formate der IF-Anweisung angehängt wird. Die ELSE-Anweisung wird dann wirksam, wenn der Ausdruck den Wert „falsch" hat. Hinter ELSE kann wiederum entweder eine Zeilennummer oder eine Anweisung notiert werden.

10.1. Der Anweisungsteil ELSE

Der ELSE-Befehl ist nicht gerade ein besonders wichtiger Befehl, viele BASIC-Dialekte besitzen ihn nicht einmal. Er ist aber trotz allem meist sehr nützlich. Der Anweisungsteil ELSE hat nur im Zusammenhang mit der IF-THEN-Anweisung einen Sinn. Die Modifikation der IF-THEN-Anweisung hat die allgemeine Form:

IF Ausdruck THEN { Zeilennummer / Anweisung } ELSE { Zeilennummer / Anweisung }

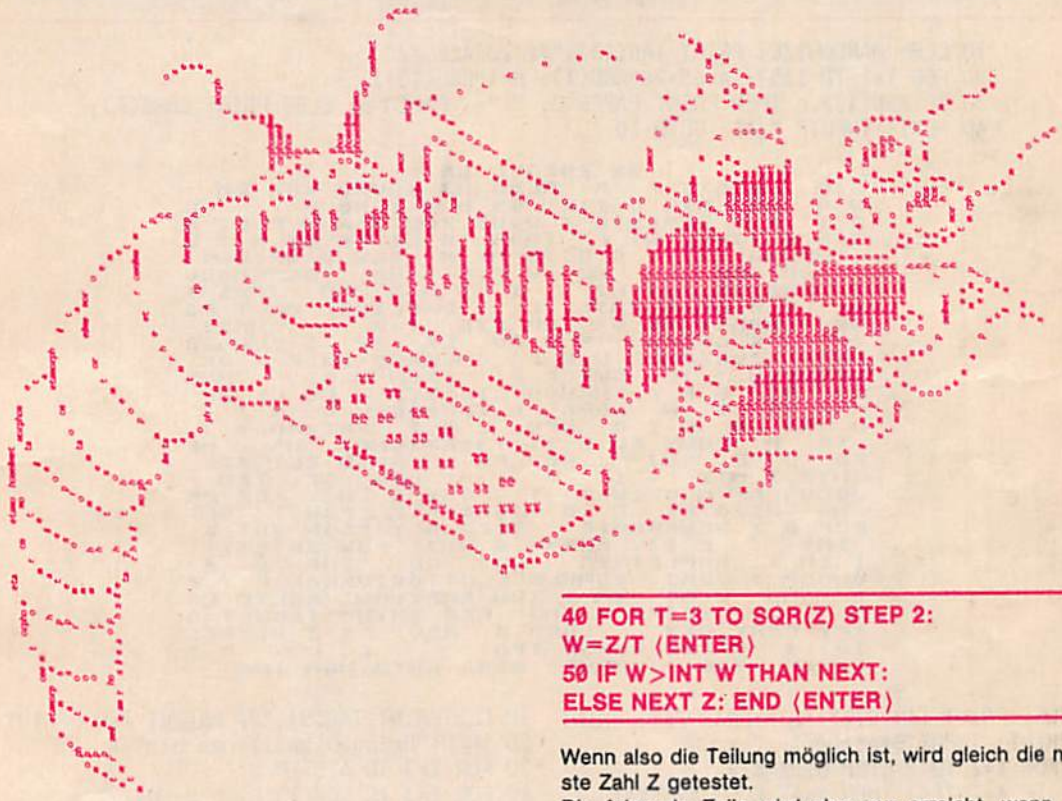
Wir wollen jetzt hierzu ein einfaches Programm behandeln: Es arbeitet mit einem Zufallsgenerator. Wenn sein Ergebnis >.6 ist, dann wird ein Leerzeichen dargestellt; andernfalls ein Großbuchstabe. Also:
IF RND(1) >.6 THEN PRINT " ": ELSE PRINT CHR$(A);
Neu ist der Befehl
CHR$(A).
Hierin bedeutet A eine Zahl zwischen 0 und 127, und der Befehl CHR$ macht daraus ein Zeichen. Zum Beispiel bedeutet A = 70 den Buchstaben F, A=52 die Zahl 4 usw.
Für unser Beispiel wird 65<=A<=90 gewählt.
Wenn wir die obige Befehlsfolge einbauen wollen, ist also auch noch das A per Zufall zu bilden:
A=65+26*RND(1).
Damit ist es durch Wiederholung mittels der FOR-NEXT Schleife möglich, den Bildschirm mit Buchstaben und Leerzeichen zu füllen, wobei die Leerzeichen 40 % ausmachen. Es entsteht so eine Struktur auf dem Bildschirm, die etwa einem ausgefüllten Kreuzworträtsel gleicht. In der Tat können Sie darin zuweilen – wenn auch selten – sinnvolle Wörter finden. Damit Sie aber den richtigen Spaß haben, wurde das Ganze noch farbig hinterlegt. Das Programm haben wir „ZUFALL" genannt, weil hier 3mal der Zufallsgenerator benutzt wurde.

```
0 !  ## APFEL ##
10 WINDOW 0,31,0,39: CLS
20 !Initiieren der Variablen
30 A=0: B=0: M=0: K=0: J=0: I=-1: G=500: X=0: Y=0
40 !Ausschnitt
50 INPUT "X1,X2 ="; X1,X2: XD=(X2-X1)/158
60 INPUT "Y1,Y2 ="; Y1,Y2: YD=(Y2-Y1)/124: CLS
70 !Ueber die Flaeche jeder 2. Punkt
80 !PSET ist Curser; G=Divergenzgrenze
90 !K=Laufparameter bzgl.Divergenz
100 FOR Y=Y1 TO Y2 STEP YD: I=I+2: J=-1
110   FOR X=X1 TO X2 STEP XD: J=J+2: A=X: B=Y: PSET J,I,7
120     FOR K=1 TO 25
130       M=A*A-B*B-X: B=A*B: B=B+B-Y: A=M
140       IF ABS(A)+ABS(B)>G THEN PRESET J,I: NEXT X: NEXT: GOTO 170
150 NEXT:NEXT:NEXT
160 !Anzeige der Parameter
170 PRINT AT(0,7);X1;"X";X2; ":";Y1;"Y";Y2
```

10.2. Einige Bemerkungen zur Bedeutung des Anweisungsteils ELSE

Der besondere Vorteil des Anweisungsteils ELSE besteht in der übersichtlicheren und meist auch kürzeren Schreibweise von Programmen. Wenn es nur um das Anspringen von zwei verschiedenen Zeilen geht, ist jedoch die Befehlsfolge ON-GOTO meist vorteilhafter, die bereits in Kapitel 7 behandelt wurde. Wir kehren jetzt aber noch einmal zum Kapitel 8 zurück. Dort haben wir den Kampf des tapferen Ritters mit der Hydra behandelt. Dieses Programm lief darauf hinaus, daß die Hydra bei ungerader Anzahl ihrer Köpfe diese vervielfachen konnte. Bei gerader Anzahl konnte dagegen der Ritter die Hälfte der Köpfe abschlagen. Dies ist also ein Problem für die IF-THEN-ELSE-Anweisung. Nachdem X die aktuelle Zahl der Köpfe ist, wird sofort A=X/2 gebildet. Dann lautet die entsprechende Zeile:

IF A=INT(A) THEN X=A: ELSE X=3*X+1



Das Programm haben wir HYDRA 2 genannt. Es ist deutlich kürzer, übersichtlicher und schneller als das Programm HYDRA (vgl. Kap. 8). Wenn Sie die beiden Programme vergleichen, werden Sie den großen Vorteil von ELSE erkennen. Und vielleicht ist dies für Sie ein erneuter Anlaß, sich mit dem interessanten Hydra-Problem noch einmal auseinanderzusetzen. Vielleicht modifizieren Sie es so, daß Sie alle Eingaben suchen, bei der die Hydra zwischendurch mehr als 1000 Köpfe besitzt. Unter den Eingaben, die kleiner als 100 sind, existieren 12.

Vielleicht suchen Sie auch die Kampfdauer als Funktion der Eingabe zu ermitteln. Dann tritt nämlich etwas Seltsames auf: Es gibt zuweilen dicht beieinanderliegende Zahlen von großer Länge. So etwas nennt man Zwillinge. Bei der Hydra sind es z. B. die Eingaben 95 und 97.

10.3. Zur Problematik der Primzahlzwillinge

Wir wollen hier an das Primzahlprogramm aus Kapitel 9 anknüpfen. Jedoch werden wir jetzt das Programm für Primzahlzwillinge mit der IF-THEN-ELSE-Anweisung formulieren. Weiterhin werden wir die Erfahrungen des vorigen Kapitels bezüglich des vorzeitigen Verlassens von FOR-NEXT-Schleifen nutzen.

Dabei entsteht eine sehr effektive Programmierung, die mit genau fünf entscheidenden Zeilen auskommt. Dies ist der Grund, daß wir das Programm PRIZWI hier ausführlicher behandeln wollen.

Wir setzen zwei Fakten voraus:

- In Y=5 befindet sich die letzte gefundene Primzahl.
- A enthält den höchsten Wert, bis zu dem wir suchen wollen. Dann lautet die Schleife für die möglichen Zahlen:

```
30 FOR Z=7 TO A STEP 2 (ENTER).
```

Die Zeile für die notwendigen Teiler lautet wieder:

```
40 FOR T=3 TO SQR(Z) STEP 2:
W=Z/T (ENTER)
50 IF W>INT W THAN NEXT:
ELSE NEXT Z: END (ENTER)
```

Wenn also die Teilung möglich ist, wird gleich die nächste Zahl Z getestet.

Die folgende Zeile wird also nur erreicht, wenn eine Primzahl vorliegt. Dann erfolgt der Vergleich mit der vorangegangenen, also:

```
60 IF Z>Y+2 THEN Y=Z:
ELSE PRINT Y,Z: Z=Y+4 (ENTER)
```

Wenn der Abstand zu groß ist, wird die errechnete Zahl auf Y übertragen. Falls ein Primzahlzwilling vorliegt, erfolgt nach ELSE der Ausdruck, und der Zählparameter Z wird um 4 erhöht. Die nun folgende Zeile hat in beiden Fällen nur noch NEXT für Z auszuführen.

```
60 NEXT (ENTER)
```

Es gibt an diesem extrem kurzen und leistungsfähigen Programm nur einen Trick, und der betrifft das Z=Y+4. Diesen möchten wir Ihnen sozusagen als kleine Übungsaufgabe offenlassen. Doch dafür einen Hinweis: Analy-

sieren Sie einmal mit dem Programm den auftretenden Abstand der Primzahlzwillinge. Hier gibt es interessante Gesetze!
Vielleicht dazu noch eine Aussage aus der Theorie: Obwohl bewiesen ist, daß der Abstand der Primzahlen über alle Grenzen wächst, ist zur Zeit nicht bewiesen, ob es eine Obergrenze für Primzahlzwillinge gibt. Es sind Primzahlzwillinge mit Hunderten von Ziffern bekannt.

Zusammenfassung Kapitel 10

1. Die Anweisung ELSE hat nur in einer Zeile Sinn, in der auch IF und THEN stehen. Diese Zeile besteht dann aus drei Teilen:
   - der Bedingung hinter IF
   - dem auszuführenden Teil hinter THEN, wenn die Bedingung erfüllt ist
   - dem auszuführenden Teil, wenn die Bedingung nicht erfüllt ist. Dieser Teil wird nach einem Doppelpunkt und dem danach folgenden ELSE abgelegt.

   Das Format lautet also:
   IF Bedingung THEN Anweisung: ELSE Anweisung
2. Sowohl hinter THEN als auch hinter ELSE können, durch Doppelpunkt getrennt, mehrere Anweisungen stehen.
3. Wie bei THEN kann hinter ELSE auch auf eine beliebige Zeile verwiesen werden.
4. Eine Primzahl liegt vor, wenn die Zahl nur durch 1 und sich selbst ohne Rest teilbar ist.
5. Primzahlzwillinge sind zwei Primzahlen, die genau den Abstand 2 besitzen. Beispiele sind
   5 und 7, 311 und 313 oder 999959 und 999961.

```
10 CLS: RANDOMIZE: PRINT TAB(13) "## ZUFALL ##"
20 FOR I=1 TO 1157: A=65+26*RND(1): B=5*RND(1)+1
30 IF RND(1)>.6 THEN PRINT PAPER 0; " ";: PAPER B: ELSE PRINT CHR$(A);
40 NEXT: INPUT" ";A$: GOTO 10
```

```
                ## ZUFALL ##
G M   LEQXJ A  R  RLBG NJ HAVKQ XUH EM
W Q  UN  BXL D IIE  GS B T Y HB P     C
H Y  D P OCQJQIG Y VVUX ZOM J R U PCPJ O
RJS B JJAFL WT F  IQDS  B EI  R QY H J L
  NMXIA LYI   A DJJS MZ K  EGW B WPZVWX
   AH GGB     NC  V EYP   HH   BGZAMGDV
J  MCATVB  UK LJ    X  Y N S  AW  JFFS J
KZ  H B BM K HMJF  L    G HLIORL MD T PQ
WG  LMQQSI KV U   NH EXE  I Z     QNHXI
     K CP KV   P D PV  LK  ZZ  J BVMS P
WWQI  ZVGZCQ Z U W L  CMLRRWU ZV F X UK
XTQMIFW Y IF  OWFOP EI  CXON M B    YNNO
D F Q   O R Z  MKNUD LTEPBYT  NU LK AE
H R UV   T  M  LQVP  B XFOASQLV O  H Z
D  RX  O  N I O  HFV  C B T  RBXXHD Y J
 YO  H  EUNX EM E ST THBRZXGNP   MS   ME
KO  D  P   WJ    AB  RQ  P RP M ZLT KX
 HYB X MTX    I   F    M  OBW  GZ  TFM
DUCVBFRF N VWIW J  I   OQY J  W    DZ CK
 NQ  VXJOTEX  C  B YJUKI B Z  PBU T  GC
ELF Q X DEWHRRHZP   TYEO O Y PSUN JDI K
 IMT    E IJ  KKEJK Q WYJ  PVW JF ETI
I IM    RAMLEI KC   VG   CG   YUF  W  P
OWGDP Y VBKJ  ZSNHLR ELOYTOHIUKADL P Y P
Z  GNO  W MG  XN I XUW RMPYWMWT ODLYC LP
D    X   YEJKZ M  NJ  FEE YHXUM TEBWLY S
TFY CPKX GYX  L LOPO R  RQV   Q Z ACXRZS
IO  I  KIPTCLFHHSP TPG      Z  L X  D I
I  EG  YXR   HMANL  GIOQ KUYD NGN TMA
```

```
10 CLS: PRINT TAB(9);" ## HYDRA2 ##": PRINT
20 PRINT: INPUT"Startwert =";X
30  FOR I=1 TO 2 STEP 0: A=X/2
40   IF A=INT(A) THEN X=A: ELSE X=3*X+1
50   PRINT X,: IF X>1 THEN NEXT
60 GOTO 20
```

```
10 CLS:PRINT TAB(9);"## PRIZWI ##": PRINT: Y=5
20 INPUT"Primzahlzwillinge bis";A
30 FOR Z=7 TO A STEP 2
40 FOR T=3 TO SQR(Z) STEP 2: W=Z/T
50 IF W>INT(W) THEN NEXT: ELSE NEXT Z: END
60 IF Z>Y+2 THEN Y=Z: ELSE PRINT Y,Z: Z=Y+4
70 NEXT
```

11. Die Unterprogrammtechnik

Wird ein bestimmter Programmabschnitt mehrmals im Programm benötigt, so ist es effektiv, diesen als Unterprogramm (engl.: subroutine) zu schreiben. Mit der Anweisung GOSUB n wird zu dem auf der Programmzeile n beginnenden Unterprogramm gesprungen und dieses abgearbeitet. Nach der Abarbeitung des Unterprogramms, das immer mit der Anweisung RETURN abzuschließen ist, „springt" der Computer zur Aufrufstelle des Hauptprogramms zurück und führt die Arbeit mit der dort folgenden Anweisung fort. Ein Programm kann mehrere Aufrufe des gleichen Unterprogramms enthalten, und ein Unterprogramm kann wiederum andere Unterprogramme aufrufen. Auf diese Weise kann der Speicherplatzbedarf des Gesamtprogramms verringert werden. Außerdem erhält man dadurch eine übersichtliche Programmstruktur. Die Unterprogrammtechnik bietet auch die Möglichkeit, den Aufwand für die Erarbeitung neuer Programme zu reduzieren.

11.1. Die Anweisungen GOSUB und RETURN

Die Anweisung für den Aufruf eines Unterprogramms hat das Format GOSUB Zielzeilennummer.
GOSUB bewirkt einen Sprung zur angegebenen Zielzeile im Unterprogramm. Dieses kann auf alle im übergeordneten Programm benutzten Variablen direkt zugreifen (Variablen, die im aufrufenden und im aufgerufenen Programm die gleichen Namen haben, belegen glei-

che Speicherplätze. BASIC unterscheidet also keine lokalen und globalen Variablen). Die Rückkehr aus dem Unterprogramm in das übergeordnete Programm bewirkt die Anweisung RETURN.
Die Programmausführung wird dann im übergeordneten Programm mit der auf den Unterprogrammaufruf folgenden Anweisung fortgesetzt.
Subroutinen sind vor allem in großen Programmen sinnvoll. In ihnen wiederholen sich recht oft gleichartige Programmteile, und gerade hier kann man sich die Schreibarbeit durch Subroutinen schnell vereinfachen. Man schreibt die Subroutinen eben nur einmal und ruft sie dann an den gewünschten Stellen einfach auf. Wir wollen das hier an einem sehr simplen Beispiel versuchen. Aus Gründen, auf die wir später noch eingehen, werden wir nur die ganzen Zahlen von Ø bis 255 zulassen. Alle anderen Zahlen sollen mit Eingabefehler und Rückkehr zur Eingabe beantwortet werden. Wir schreiben zunächst die Subroutine:

```
130 PRINT"**Eingabefehler**": RETURN ⟨ENTER⟩
```

Wenn wir irgendwo GOSUB 13Ø schreiben, so geschieht das Folgende:
Ähnlich wie bei GOTO wird die Zeile 13Ø gesucht, dann wird der Eingabefehler ausgedruckt, und danach bewirkt RETURN, daß der Rechner wieder an die Stelle zurückkehrt, von der er kam. RETURN heißt etwa soviel wie Rückkehr.

Wenden wir dies nun auf die Zahleneingabe an:

```
20 FOR X=1 TO 2 STEP Ø ⟨ENTER⟩
```

Ständige Wiederholung des nun folgenden

```
30 INPUT "Zahl"; A ⟨ENTER⟩
```

Die Eingabe wird in A abgelegt.

```
40 IF A<Ø THEN GOSUB 130: NEXT ⟨ENTER⟩
```

Wenn A kleiner als Null ist, dann wird ein Eingabefehler angezeigt und danach infolge NEXT die Eingabe wiederholt.
Jetzt werden Zahlen >255 ausgeschaltet:

```
50 IF A>255 THEN GOSUB 130: NEXT ⟨ENTER⟩
```

und weiter:

```
60 IF A>INT(A) THEN GOSUB 130: NEXT ⟨ENTER⟩
```

Jetzt existieren nur noch Zahlen, die zugelassen sind, und wir können wunschgemäß mit ihnen arbeiten. Selbst an diesem trivialen Beispiel ist zu sehen, wie GOSUB das Programm verkürzt und damit Speicherkapazität einspart. Das vollständige Programm finden Sie am Ende des Kapitels als Programm DEZBIN. Es soll also seinem Namen gemäß die dezimalen Zahlen Ø bis 255 in binäre Zahlen wandeln. Binäre Zahlen sind Zahlen, die nur aus Ø und 1 bestehen und genau den Bits entsprechen, mit denen unser Rechner intern arbeitet. Diese Zahlen enthalten 8 Positionen, an denen die Nullen oder Einsen stehen. Nach 4 Zeichen wollen wir zur besseren Übersicht einen Zwischenraum, also ein Leerzeichen (Space), einfügen.

Wir setzen B=256 und wählen unsere 8 Stellen:

```
80 FOR I=1 TO 8 : B=B/2 ⟨ENTER⟩
```

Jetzt prüfen wir, ob unsere eingegebene Zahl A größer als 128 ist, dann wäre nämlich die erste binäre Ziffer gleich 1, anderenfalls Ø:

```
90 IF A>=Ø THEN A=A-B : PRINT "1";:
ELSE PRINT "Ø"; ⟨ENTER⟩
```

Bei 1 ziehen wir auch gleich den durch 1 dargestellten Anteil von 128 ab. Jetzt könnten wir eigentlich in der nächsten Zeile NEXT schreiben. Es wird immer die Hälfte von B gebildet und durch Vergleich 1 bzw. Ø gedruckt sowie bei 1 der entsprechende Wert von A abgezogen. Wir wollten aber nach vier Ø/1-Ziffern einen Leerraum einfügen. Damit folgt:

```
100 IF I=4 THEN PRINT " ";          ⟨ENTER⟩
110 NEXT: PRINT                     ⟨ENTER⟩
120 NEXT                            ⟨ENTER⟩
```

Viel Freude an diesem Programm, und eventuell ist es Ihnen sogar nützlich. Vielleicht probieren Sie einmal, dieses Programm auf die Zahlen von Ø bis 65 535 zu erweitern. Dann müssen nämlich 16 binäre Werte entstehen. Es ist nicht schwer, das Programm in diesem Sinne zu ändern.

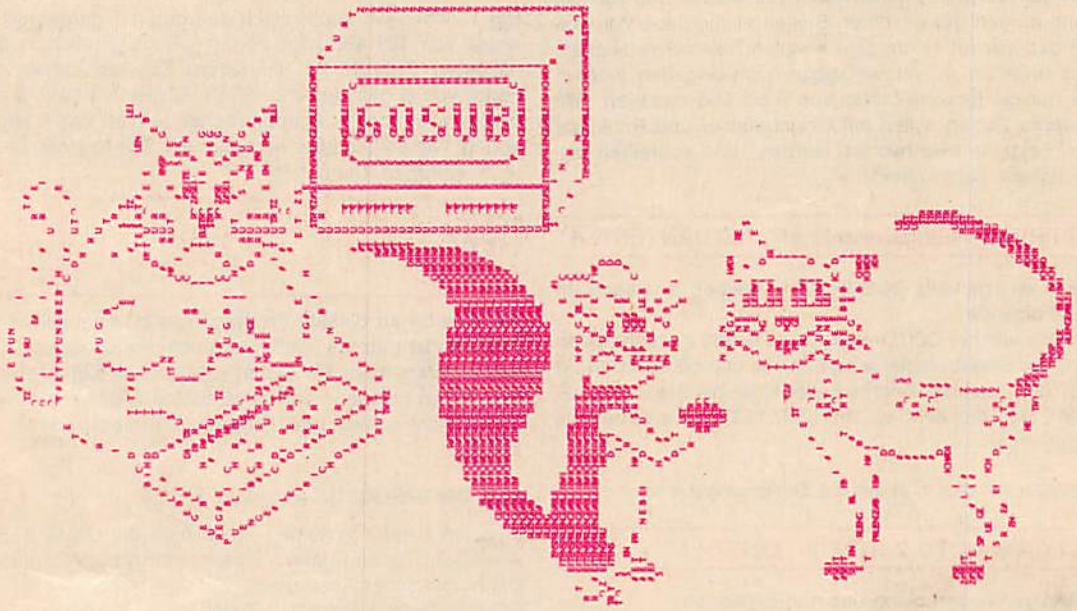
11.2. Die ON-GOSUB-Anweisung

In vielen BASIC-Systemen steht auch die ON-GOSUB-Anweisung für den Unterprogrammaufruf zur Verfügung:
ON numerischer Ausdruck
GOSUB Zielnr.1,Zielnr.2,...,Zielnr.n
In dieser Anweisung sind hinter dem Schlüsselwort GOSUB mehrere Zielzeilennummern (Zielnr.) notiert. Vom Wert des numerischen Ausdrucks ist es, wie bei der ON-GOTO-Anweisung, abhängig, zu welcher Programmzeile gesprungen wird. Damit ist es möglich, unter verschiedenen Unterprogrammen bzw. verschiedenen Startpunkten eines Unterprogramms eine Auswahl zu treffen. Im Gegensatz zu ON GOTO sind hier aber die Zahlenwerte <1 und größer N verboten.
Die vorgestellte Thematik wollen wir Ihnen an Hand des Programms RADIUS 3 veranschaulichen. Es soll Ihnen zugleich zeigen, wie man ein angenehmes Menü aufbaut.
Wir wollen den Radius eingeben und daraus für den Kreis je nach Wunsch den Durchmesser, den Umfang oder den Flächeninhalt sowie für die Kugel die Oberfläche oder das Volumen berechnen.
Ähnlich wie ON GOTO wollen wir jetzt ON GOSUB verwenden.
Daher erzeugen wir das folgende Menü:

```
20 PRINT: INPUT"Radius=";R          ⟨ENTER⟩
30 PRINT"1:Kreisdurchmesser"        ⟨ENTER⟩
40 RPINT"2:Kreisumfang"             ⟨ENTER⟩
50 PRINT"3:Kreisinhalt"             ⟨ENTER⟩
60 PRINT"4:Kugeloberfläche"         ⟨ENTER⟩
70 PRINT"5:Kugelvolumen";A          ⟨ENTER⟩
90 ON A GOSUB 100, 110, 120, 130, 140:
GOTO 20                             ⟨ENTER⟩
```

Hier wird der Vorteil von GOSUB besonders deutlich. Es strukturiert in hervorragender Weise unser Problem. Mit dem soeben besprochenen Programmteil können wir

unser Programm schrittweise erweitern und erproben. Wir setzen zunächst in den Zeilen 100, 110, 120, 130 und 140 je ein RETURN nach. Dann ist die eben geschriebene Teilroutine sofort testbar. Sie stellt die spätere Hauptroutine (Main) dar. Wenn sie läuft, kann einzeln in jeder Zeile die Berechnung eingefügt und erprobt werden.
Diese Methode ist ein Teil der sog. strukturierten Programmierung und hat den Vorteil, daß wir unser Programm schrittweise von oben nach unten ausbauen können.



11.3. Einige Bemerkungen zum Speicherplatzbedarf bei der Subroutinetechnik mittels der GOSUB-Anweisung

Nun wollen wir einmal abschätzen, was unsere Subroutinetechnik an Speicherplatz einspart. Wir nehmen an, sie sei I Byte lang und werde m-mal benötigt. Dann würden bei direktem Einbau an den entsprechenden Programmstellen insgesamt I*m Byte benötigt. Als Subroutine geschrieben, werden jedoch zunächst einmal Speicher direkt für die Subroutine: Pointer und Nummer der Zeile sowie für den Befehl RETURN, also zusätzlich 5 Byte, benötigt. An jeder Stelle, wo die Subroutine benötigt wird, steht GOSUB und die Zeilennummer, also etwa 4 Byte; d. h. m*4 Byte. Insgesamt also bei Anwendung von Subroutinen I+5+4*m gegenüber I*m Byte ohne Verwendung der Subroutinentechnik. Daraus ist ersichtlich, daß Subroutinen, allein vom Speicherplatzbedarf betrachtet, immer dann vorteilhaft sind, wenn sie mindestens 2mal bei Textlängen über 13 Byte oder bei Textlängen von über etwa 5 Byte oftmals eingesetzt werden (vgl. Tabelle).

Bei dem hier betrachteten Gesichtspunkt der speicherplatzsparenden Anwendung ist aber noch nicht der Vorteil der o. g. strukturierten Programmierung berücksichtigt.
Zusätzliche Erkenntnisse zu dieser Problematik werden uns durch das Programm GOSUB vermittelt. Wir lassen einfach GOSUB sich selbst aufrufen.

Die entscheidende Zeile des Programms ist die Zeile 30:

```
30 PRINT I; : I=I+1:GOSUB 30 ⟨ENTER⟩
```

Es wird also immerfort die nach GOSUB 30 aktuelle Adresse auf dem Stack abgelegt. Der Stackbereich wächst dadurch pausenlos, bis der Rechner überfordert ist. Dann druckt er

```
?OM ERROR IN 30
```

aus und beendet die Arbeit.

Wir wissen, daß mit CLEAR n der Speicherbereich verändert werden kann. Deshalb setzen wir die Zeile davor:

```
20 INPUT A: CLEAR A ⟨ENTER⟩
```

Wählen wir A=15 200, so ist das Ende bereits bei 012 erreicht und für A=14 200 tritt das Ende erst nach 202 ein. Hieraus können wir berechnen, daß je aktuelle GOSUB (ohne RETURN) 5 Byte auf dem Stack gelegt werden. Mit ähnlichen Methoden ist es möglich, den erforderlichen Stack zu bestimmen, man spricht dabei auch von dynamischem Speicherraum. Bei großen Programmen können Sie manchmal erleben, daß Ihr Rechner mitten im Programm die Arbeit mit OM ERROR abbricht. Dann wurde der dynamische Speicherraum zu klein bzw. die Stacktiefe zu groß.
Hierbei ist zu beachten, daß nicht nur die Rückkehradresse von GOSUB auf den Stack abgelegt wird, sondern auch anderes. Wir haben das schon bei der FOR-NEXT-Schleife erwähnt.
Im Prinzip läßt sich auf der o. g. Basis sogar eine rekursive Programmierung aufbauen. Versuchen Sie einmal das folgende Programm zu verstehen, welches die Fakultät berechnet:

```
10 INPUT N: FK=1: M=1                          ⟨ENTER⟩
20 IF N=0 OR N=1 GOTO 50                       ⟨ENTER⟩
30 M=M+1: IF M<N THEN GOSUB 30                 ⟨ENTER⟩
40 FK=FK*M: M=M-1: IF M>1 THEN RETURN
                                               ⟨ENTER⟩
50 PRINT FK                                    ⟨ENTER⟩
```

11.4. Zeitprobleme in Zusammenhang mit der Anwendung der GOSUB-Anweisung

Der Zeilenansprung von GOSUB benötigt genau wie bei

der GOTO-Anweisung eine Suchzeit, die durch den Abstand der gesuchten Zeile vom Programmanfang gegeben ist. Deshalb sollen alle Unterprogramme mit häufigem Aufruf möglichst weit vorn im Programm festgelegt werden. Die Rückkehr bei RETURN ist wegen der auf dem STACK abgelegten Adresse dagegen sehr schnell.

Zusammenfassung Kapitel 11

1. Existieren in einem Programm gleichartige Programmteile, so braucht man sie als Subroutine (Unterprogramm) nur einmal zu schreiben. Das Format der Anweisung lautet:
   GOSUB Zeilennummer
2. Die Subroutine kann aus einer oder mehreren Zeilen bestehen. Ihr Ende ist durch ein RETURN zu kennzeichnen.
3. Der Interpreter arbeitet dabei folgendermaßen: Erscheint in einer Programmzeile GOSUB, so wird
   - die folgende Zeilennummer wie bei GOTO behandelt,
   - die Adresse hinter der Zeilennummer auf dem BASIC-Stack abgelegt,
   - die Subroutine abgearbeitet, bis RETURN erreicht ist,
   - der Interpreter kehrt zu der auf dem Stack befindlichen Adresse zurück und arbeitet dort weiter.
4. Mittels der Befehlsfolge ON GOSUB ist es möglich, schrittweise ein Programm zu erweitern. Dabei wird zunächst die Hauptroutine geschrieben. Die einzeln anzuspringenden Zeilen erhalten zuerst nur ein RETURN. Dann wird diese Hauptroutine lauffähig gemacht und danach schrittweise die Subroutinen. Dies geschieht, indem das RETURN durch die entsprechende Befehlsfolge mit dem Ende RETURN ersetzt wird. Dabei entsteht noch der weitere Vorteil, daß diese Subroutinen, ja selbst Teile von ihnen, auch an anderen Stellen im Programm verwendet werden können.
5. Subroutinen ermöglichen eine übersichtliche, strukturierte Programmierung, die vor allem bei der Programmentwicklung von großem Vorteil ist.
6. Subroutinen gestatten bei häufigem Aufruf oder größerer Länge, Speicherplatz einzusparen.
7. Der Zeilenansprung von GOSUB benötigt wie bei GOTO eine Suchzeit, die durch den Abstand der gesuchten Zeile vom Programmanfang gegeben ist. Deshalb sollten alle Unterprogramme mit häufigem Aufruf möglichst weit vorn im Programm angelagert sein.
8. Die Rückkehr bei RETURN ist wegen der auf dem Stack abgelegten Adresse dagegen sehr schnell.
9. Bei der folgenden Operation wird der BASIC-Stack verwendet und damit dynamischer Speicherraum benötigt:
   - FOR-NEXT 15 Byte
   - GOSUB 5 Byte
   - offene Klammer 4 Byte
   - Zwischenergebnisse bei Rechnungen 6 Byte

   Diese Werte gelten für den KC 85 und den KC 87. Bei anderen Rechnern treten zuweilen erheblich größere Werte auf.

```
10 CLS: PRINT TAB(10);"## DEZBIN ##": PRINT
20  FOR X=1 TO 2 STEP 0
30   INPUT"Dezimal =";A
40   PRINT"Binaer  =";: B=256
50   IF A<0       THEN GOSUB 130: NEXT
60   IF A>255     THEN GOSUB 130: NEXT
70   IF A>INT(A) THEN GOSUB 130: NEXT
80    FOR I=1 TO 8: B=B/2
90     IF A=>B THEN A=A-B: PRINT"1";: ELSE PRINT"0";
100    IF I=4 THEN PRINT" ";
110   NEXT: PRINT
120 NEXT
130 PRINT"** Eingabefehler **": RETURN
```

Notwendige Länge L einer Subroutine, damit Speicherplatz gespart wird. Sie ist eine Funktion der Anzahl der Wiederholungen M und der Stellenzahl der Zeilennummer K:

$$L > \frac{5 + M * (K+1)}{M - 1}$$

	K=				
M	1	2	3	4	5
2	10	12	14	16	18
3	6	8	9	11	12
4	5	6	8	9	10
5	4	6	7	8	9
6–8	4	5	6	7	8
9	3	5	6	7	8
10	3	4	6	7	8

```
10 CLS: PRINT TAB(9);"## RADIUS3 ##"
20 PRINT: INPUT"Radius ="; R
30 PRINT "1:Kreisdurchmesser"
40 PRINT "2:Kreisumfang"
50 PRINT "3:Kreisinhalt"
60 PRINT "4:Kugeloberflaeche"
70 INPUT "5:Kugelvolumen"; A
80 IF A<1 OR A>5 THEN PRINT"** Eingabefehler **": GOTO 30
90 ON A GOSUB 100, 110, 120, 130, 140: GOTO 20
100 PRINT R+R: RETURN
110 PRINT 2*PI*R: RETURN
120 PRINT PI*R*R: RETURN
130 PRINT 4*PI*R*R: RETURN
140 PRINT 4*PI*R*R*R/3: RETURN
```

```
10 CLS: PRINT TAB(10);"## GOSUB ##": PRINT
20 INPUT A: CLEAR A
30 PRINT I;: I=I+1: GOSUB 30
```

12. Der Feldbegriff

Die Variablen, die wir in den vorangegangenen Abschnitten bisher benutzt haben, repräsentierten Einzeldaten wie X, H3, Y$ oder R. Jeder dieser Variablen kann nur ein Wert zugewiesen werden. Man bezeichnet sie daher auch als einfache oder skalare Variablen. Treten jedoch größere Variablenmengen auf, so bedient man sich in der Mathematik indizierter Variablen, also solcher, die sich durch Indizes (kleine am Fuß der Variablen befindliche Zahlen wie z. B. x_1, x_2) unterscheiden. Diese Möglichkeit bietet selbstverständlich auch unser Computer. Man nennt eine solche Menge gleichartiger Variablen Feld (engl.: array) und eine Variable, der ein ganzes Datenfeld zugeordnet werden kann, Feldvariable. Eine Feldvariable wird wie eine einfache Variable durch einen Namen (Feldnamen) bezeichnet. Der Rechner unterscheidet sie von der einfachen Variablen durch die nach dem Namen stehende Klammer. Der Zugriff auf ein bestimmtes Element eines Feldes ist durch Angabe des Feldnamens und der in Klammern stehenden Indexfolge möglich. Für die Speicherung der Feldelemente muß im Computer eine entsprechende Zahl von Speicherplätzen reserviert werden. Dazu dient die Dimensionierungsanweisung. Sie hat das Format: DIM Feldname (Konstante oder numerischer Ausdruck). Der hinter den Feldnamen in Klammern angegebene ganzzahlige numerische Wert bezeichnet den Größtwert des entsprechenden Index. Der Anfangswert der Indexzählung ist im allgemeinen auf Ø festglegt. In einer DIM-Anweisung können (durch Komma getrennt) mehrere Felder dimensioniert werden.

12.1. Eindimensionale Felder

Bei den eindimensionalen Feldern wird zur Beschreibung der Reihenfolge der Feldelemente nur ein Index verwendet. Beispielsweise DIM A(17), B(6), C(4) usw. Unterbleibt eine solche einleitende Anweisung und stößt dann der Rechner im Programm erstmalig auf ein Feld, so setzt er die Feldgröße automatisch auf 10.
Eindimensionale Felder nennt man in der Physik Vektoren. Die maximale Größe der Dimension liegt meist bei 65 535. Sie ist also so groß, daß die vorhandene Speicherkapazität meist vorher begrenzt wird (Anzeige OM-ERROR).
Wir wollen nun das Prinzip des Arrays zum Sortieren von Zahlen verwenden. Dazu dient uns das Programm ZASORT, welches wir am Ende dieses Kapitels vollständig angeben. Hierfür benötigen wir drei Teilroutinen, die in einer Hauptroutine mit Menü verknüpft sind. Neu sind nur die Teilroutinen: Eingabe, Anzeige und Sortieren.
Das Programm ist mit mehreren REM versehen. Daher genügt es, wenn nur die Teilroutinen kurz besprochen werden.
Zunächst muß aber das Array dimensioniert werden. Hierzu dient die Eingabe:

```
30 INPUT „Max. Anzahl der Zahlen = "; N:
DIM A(N)                                  ⟨ENTER⟩
```

Danach folgt die Eingabe der Daten:

```
90 FOR I = Ø TO N                         ⟨ENTER⟩
100 PRINT I;: INPUT"=";A(I)               ⟨ENTER⟩
120 NEXT                                  ⟨ENTER⟩
```

Der Reihe nach werden also den mit I gekennzeichneten Elementen des Feldes die Zahlenwerte A (I) zugewiesen. Damit ist das Feld belegt. Im eigentlichen Programm sind noch einige Modifikationen vorhanden, die ein Abbrechen und den Neubeginn der Eingabe zu. Diese probieren Sie aber am besten selbst aus.
Dieses Beispiel zeigt bereits die neuen Möglichkeiten von Feldern, die mit keiner anderen Methode erreichbar sind. Sie stellen also eine wichtige Erweiterung der Variablen dar. Noch deutlicher wird dies bei der Anzeige:

```
90 FOR I=Ø TO N: PRINT A(I),              ⟨ENTER⟩
100 NEXT                                  ⟨ENTER⟩
```

Es genügt dieser kurze Programmteil, um das ganze Feld anzeigen zu lassen!
Nun kommen wir zum wichtigsten Teil unseres Programms, nämlich dem Sortieren der Zahlen nach ihrer Größe. Sortieren zählt zu den besonders wichtigen Aufgaben der Rechentechnik. Beispielsweise ist es bei der Textverarbeitung notwendig, Namen alphabetisch zu ordnen. Solche Prozesse sind sehr zeitaufwendig, deshalb wurde hierfür eine Vielzahl von Methoden entwickelt. Wir wählen übersichtshalber ein besonders einfaches und leicht erklärbares Verfahren. Es heißt „bubble sort". Der Name „bubble", welcher Blase bedeutet, deutet bereits das Prinzip an. Es werden zwei benachbarte Werte verglichen, und wenn sie falsch angeordnet sind, so werden sie vertauscht. Ein falsch angeordneter Wert bewegt sich dann wie eine Luftblase durch das Feld, bis er die richtige Stelle erreicht. Dieses Verfahren wird so oft wiederholt, bis keine Blase mehr entsteht. Es ist daher eine Kennzeichnung für das Auftreten einer Blase beim Vergleich notwendig. Wir verwenden hierfür die Variable. Ihretwegen und wegen der eventuellen Zeit für das Sortieren beginnt die Teilroutine mit:

```
170 PRINT" Ich sortiere ": C=Ø            ⟨ENTER⟩
180 FOR I=Ø TO N−1                        ⟨ENTER⟩
190 IF A(I)<=A(I+1) THEN NEXT;
GOTO 220                                  ⟨ENTER⟩
```

Wenn also zwei Werte falsch geordnet sind, wird der folgende Teil für das Vertauschen beider Werte erreicht:

```
220 A=A(I): A(I)=A(I+1): A(I+1)=A: C=1    ⟨ENTER⟩
```

Es wird also über C=1 der Austausch, also die Blase, gekennzeichnet.

```
210 NEXT                                  ⟨ENTER⟩
```

schließt den einen Durchlauf ab, und es folgt das Testen auf die Blase:

```
220 IF C=1 THEN C=Ø: GOTO 180:
ELSE END                                  ⟨ENTER⟩
```

Mit diesem Beispiel sind aber die Möglichkeiten von Arrays erst angedeutet. Insgesamt sind sie so vielfältig, daß wir sie hier nicht einmal vollständig nennen können.

12.2. Mehrdimensionale Felder

In einer Tabelle haben wir Zeilen und Spalten. Jedes Element der Tabelle kann daher durch je einen Index, der die Zeilen- bzw. Spaltennummer trägt, gekennzeichnet werden. So etwas ist ein zweidimensionales Feld. Seine Datenelemente sind also nach Zeilen und Spalten geordnet. Die Position eines Feldelements wird durch zwei Indizes gekennzeichnet, von denen in der Regel der erste

die Zeile, der zweite die Spalte benennt. Beispielsweise ist die indizierte Variable X(2,3) das Element aus Zeile 2 und Spalte 3 des Feldes X. Für ein zweidimensionales Feld ist auch die Bezeichnung Matrix gebräuchlich.
Außer ein- und zweidimensionalen Feldern können auch solche mit höherer Dimensionszahl gebildet werden. In einem n-dimensionalen Feld wird jedes Element durch n Indizes bezeichnet. Für den zulässigen Höchstwert von n existieren in den einzelnen Sprachversionen unterschiedliche Festlegungen; üblich sind 255 Dimensionen. Sie werden praktisch jedoch nie benötigt.
Zur Veranschaulichung des Sachverhaltes möchten wir Sie mit dem Magischen Quadrat etwas näher vertraut machen (vgl. Programm MAGISCH).
Hierzu wählen wir ein quadratisches Feld Q(N,N). Es erhält (n+1)*(n+1) Teilfelder, die nach der Berechnung die Zahlen 1 bis (n+1)*(n+1) so tragen, daß in jeder Zeile, in jeder Spalte und auch in den beiden Diagonalen die Summierung zur gleichen Summe führt. Wir haben hier nicht alle Möglichkeiten berücksichtigt, sondern nur je eine Variante für die ungeradzahligen Quadrate. Infolge der begrenzten Größe des Bildschirms sollten nur Magische Quadrate für 3, 5, 7 und 9 berechnet werden. Das Prinzip ist nicht ganz einfach darzustellen. Vielleicht versuchen Sie sich einmal selbst am Programm MAGISCH. Hier möchten wir nur drei Teile erklären:
Als erstes die Dimensionierung gemäß:

```
40 DIM Q(N,N) (ENTER)
```

Die Anzeige wird wie folgt realisiert:

```
120 FOR I=1 TO N                (ENTER)
130     FOR J=1 TO N            (ENTER)
140 PRINT TAB(4*J-4);Q(I,J);    (ENTER)
150     NEXT: PRINT             (ENTER)
160 NEXT                        (ENTER)
```

Schließlich berechnen wir noch die Summe aus der ersten Zeile:

```
170 S=0 FOR I=1 TO N: S=S+Q(1,I): NEXT
(ENTER)
```

und lassen sie anzeigen:

```
180 PRINT" Die Summe ist=";S (ENTER)
```

12.3. Zur Verwendung von Arrays in Arrays

Neben der Mehrdimensionalität von Arrays besteht nun auch noch die Möglichkeit, Arrays in Arrays zu verwenden. Also jenes, was beim Selbstaufruf von GOSUB im vorigen Kapitel zum Fehler führte, ist bei den Arrays zulässig. Ich kann also z. B. verwenden A(B(C(7))).
Es ist natürlich schwierig, dies übersichtlich zu gestalten. Wir möchten hierfür auf ein Kinderspiel zurückgreifen, auf den Abzählreim: „Ene mene muh, ab bist du".
Die Kinder stellen sich also in einem Kreis auf, und jemand zählt nach diesem Vers ab, so daß jeder sechste ausscheidet. Dadurch wird der Kreis immer kleiner, und einer bleibt übrig, der dann im Spiel eine bestimmte Funktion übernimmt. Dieses Spiel hat ernsthafte, wenn auch makabre Vorgänger in der Geschichte; wobei das Abzählen dann über Tod oder Leben des einzelnen entschied. Gemäß seiner ersten Beschreibung durch den jüdischen Historiker Flavius Josephus heißt es oft Josef-Spiel. Unser Programm JOSEF gibt eine allgemeine Lösung. Es existieren N Teilnehmer, und davon soll beim Abzählen jeder M-te ausscheiden. N und M werden über INPUT eingegeben, und danach zeigt das Programm die Reihenfolge der ausscheidenden Nummer an, und wer am Ende übrigbleibt. Versuchen Sie einmal, dieses Programm zu beschreiben.

12.4. Stringvariablenfelder

Bisher haben wir nur numerische Felder behandelt. Ebenso wie es neben den numerischen Variablen auch Stringvariablen gibt, existieren auch Stringarrays.
Das Programm SORT demonstriert am Beispiel eines eindimensionalen Stringvariablenfeldes einfache Möglichkeiten zur effektiven Nutzung von Variablenfeldern. Es entspricht in seinen wesentlichen Zügen dem Programm ZASORT. Der Unterschied besteht eigentlich nur darin, daß jetzt Stringvariablen verwendet werden. Es ist

Fortsetzung auf S. 38

Zusammenfassung Kapitel 12

1. Werden für eine Anweisung mehrere zusammengehörige Daten benötigt, so sind sie am besten durch ein Array (Feld) zu behandeln. Ein Feld wird genauso mit einem Namen gekennzeichnet wie die üblichen Variablen. An einen Feldnamen schließt sich jedoch unmittelbar in einer Klammer ein Zahlenwert oder ein Ausdruck an, der auf das ausgewählte Teilfeld verweist. Hier können genau wie bei den einfachen Variablen Zahlen oder Zeichenketten abgelegt werden.
2. Damit der Rechner nicht unnötigen oder zu kleinen Speicherraum für ein Feld reserviert, ist es sinnvoll, zu Beginn des Programms das Feld richtig zu dimensionieren. Dies erfolgt mit der Anweisung:
   DIM Variable (Zahlenwert)
3. Es ist zu beachten, daß die Zählung der Teilfelder mit 0 beginnt.
4. Wird vom Interpreter im Programm ein Feld gefunden, das zuvor nicht dimensioniert wurde, so wählt er automatisch die Größe 10.
5. Ein zweimaliges Dimensionieren von Feldern ist in BASIC nicht möglich. Felder können im Verlauf eines Programms nicht vergrößert oder verkleinert werden.
6. Felder lassen sich auch mehrdimensional definieren. Dann sind in der Klammer die Größen der einzelnen Dimensionen durch Kommata zu trennen. Ein mit DIM A(3,7,9,5) festgelegtes Feld A hat also vier Dimensionen mit den Längen 4, 8, 10 und 6. Es legt einen Speicherraum für 1920 Werte an.
7. Bei den Namen werden Felder unterschiedlicher Dimensionen nicht auseinandergehalten. Jedes Feld, egal welcher Dimension, muß folglich einen anderen Namen erhalten. Es werden lediglich numerische und Stringfelder unterschieden.
8. Innerhalb der Klammer eines Feldes ist es möglich, wieder ein Feld anzuordnen. Dies ist sogar wiederholt möglich. Damit lassen sich komplexe Umordnungsprozesse in Feldern vornehmen.
9. Zeichenketten (Strings) lassen sich genau wie Zahlen miteinander vergleichen. Dabei entsteht ein Ergebnis, welches durch die ASCII-Nummer bestimmt wird. Insbesondere ist auf diese Weise eine alphabetische Sortierung möglich.

ein großer Vorteil der Codierung von Strings, die wir noch ausführlicher in den nächsten Kapiteln behandeln werden, daß hierbei automatisch eine fast vollständig richtige alphabetische Sortierung erfolgt.
Die entscheidenden, vom Programm ZASORT abweichenden Zeilen lauten jetzt:

```
180 IF A$(I-1)<=A(I) THEN NEXT:
GOTO 210 (ENTER)
190 B$=A$(I-1): A$(I-1)=A$(I):A$(I-1)=B$:
C=1 (ENTER)
```

Stringvariablen lassen sich miteinander genau wie Zahlen vergleichen. Dabei entsteht ein Ergebnis, welches durch die ASCII-Code-Nummer bestimmt wird.

```
10 CLS: PRINT TAB(8); "## ZASORT ##": PRINT
20 !----Dimensionieren des Feldes----
30 INPUT"Max. Anzahl der Zahlen =";N: DIM A(N): M =-1
40 !----Menu und Verteiler----
50 INPUT "1:Eingabe 2:Anzeige 3:Sortieren"; A
60 ON A GOSUB 80, 130, 170: GOTO 50
70 !-----Eingabe-----
80 CLS: PRINT "Eingabe": PRINT "Abbruch mit -100": PRINT
90   FOR I=M+1 TO N
100   PRINT I;: INPUT"="; A(I)
110   IF A(I)=-100 THEN I=N : ELSE M=I
120  NEXT: RETURN
130 CLS:!-----Anzeige-----
140  FOR I=0 TO M: PRINTA(I),
150  NEXT: PRINT: RETURN
160 !-----Sortieren-----
170 CLS: PRINT"** Ich sortiere **": C=0
180  FOR I=0 TO M-1
190   IF A(I)<=A(I+1) THEN NEXT: GOTO 220
200   A=A(I): A(I)=A(I+1): A(I+1)=A: C=1
210  NEXT
220 IF C=1 THEN C=0: GOTO 180: ELSE RETURN
```

```
10 CLS: PRINT TAB(10); "## MAGISCH ##": PRINT
20 INPUT"Ordnung (ungerade) =";  N
30 IF N<2 OR 2*INT(N/2)=N GOTO 20
40 DIM Q(N,N): I=N-1: J=I/2
50  FOR U=0 TO N-1
60   FOR V=0 TO N-1
70    IF I<N THEN I=I+1: ELSE I=1
80    IF J<N THEN J=J+1: ELSE J=1
90    Q(I,J)=U*N+V+1
100  NEXT: I=I-2: J=J-1
110 NEXT: PRINT: PRINT
120 FOR I=1 TO N
130  FOR J=1 TO N
140   PRINT TAB(4*J-4);Q(I,J);
150  NEXT: PRINT
160 NEXT: PRINT
170 S=0: FOR I=1 TO N: S=S+Q(1,I): NEXT
180 PRINT"Die Summe ist =";S:PRINT
```

```
        ## MAGISCH ##

Ordnung (ungerade) = 7

 22  31  40  49  2   11  20
 21  23  32  41  43  3   12
 13  15  24  33  42  44  4
 5   14  16  25  34  36  45
 46  6   8   17  26  35  37
 38  47  7   9   18  27  29
 30  39  48  1   10  19  28

Die Summe ist = 175
```

```
10 CLS: PRINT TAB(13);"## JOSEF ##": PRINT
20 PRINT"    N Teilnehmer bilden einen Kreis"
25 PRINT TAB(7);"jeder M-te scheidet aus": PRINT
30 INPUT"N,M =";N,M: DIM K(N): PRINT
35 PRINT"Der Reihe nach scheiden aus:"
40  FOR Z=1 TO N : K(Z)=Z+1: NEXT
50 K(N)=1: Z=N : REM= Kreis + Anfang
60  FOR I=1 TO M-1: Z=K(Z): NEXT
70 PRINT K(Z),: K(Z)=K(K(Z))
80 IF K(Z)<>Z GOTO 60
90 PRINT: PRINT: PRINT"Der letzte ist:";Z: PRINT
```

```
          ## JOSEF ##

    N Teilnehmer bilden einen Kreis
       jeder M-te scheidet aus

N,M = 16      7

Der Reihe nach scheiden aus:
 7            14           5
 13           6            16
 10           4            2
 1            3            9
 15           8            11


Der letzte ist: 12
```

```
10 CLEAR 500: DIM A$(50)
20 CLS: PRINT TAB(15);"## SORT ##": PRINT
30 !-----Main und Verteiler-----
40 PRINT"1:Eingabe": PRINT"2:Anzeige"
50 INPUT"3:Sortieren"; A: PRINT
60 ON A GOSUB 80, 120, 160: GOTO 20
70 !-----Eingabe------
80 CLS: PRINT"Eingabe": PRINT"beenden mit@"
90  FOR I=B TO 50: INPUT A$(I)
100   IF A$(I)="@" THEN B=I: I=50
110 NEXT: RETURN
120 CLS: !-----Anzeige-----
130  FOR I=0 TO B-1: PRINT A$(I): NEXT
140 INPUT"";B$: RETURN
150 !-----Sortieren-----
160 PRINT"Ich sortiere; bitte Moment warten !": C=0
170  FOR I=1 TO B-1
180   IF A$(I-1)<= A$(I) THEN NEXT: GOTO 210
190   B$=A$(I-1): A$(I-1)=A$(I): A$(I)=B$: C=1
200  NEXT
210 IF C=1 THEN C=0: GOTO 170: ELSE RETURN
```

13. Zeichenketten (Strings)

Zeichenketten, sog. Strings, bestehen aus einer Folge von Einzelzeichen (fast alle ASCII-Zeichen). Dazu gehören auch viele Sonderzeichen. Die zulässige Länge einer Zeichenkette kann verschieden sein, jedoch meist nicht länger als eine Eingabezeile (indirekt sind aber 255 Zeichen realisierbar). Zeichenketten finden vor allem in der Textverarbeitung breite Anwendung. Es gibt mehrere Möglichkeiten, Strings in einem Programm festzulegen. Dabei sind zwei Methoden zu unterscheiden:

- die direkte Erzeugung durch Anführungszeichen, zwischen denen die gewünschte Zeichenkette steht,
- die indirekte Realisierung. Hierbei ist das $ (sprich String) zur Kennzeichnung notwendig.

13.1. Zur Stringerzeugung

Zeichenketten können also, ähnlich wie bei den Zahlen, unterschiedlich, d. h. direkt oder über eine Variable, gekennzeichnet werden.
Wenn wir beispielsweise schreiben PRINT "gut", liegt die direkte Bezeichung vor. Schreiben wir aber PRINT A$, so liegt die indirekte Bezeichnung vor, denn A$ ist eine Variable, der ein String zugewiesen werden muß. Dies könnte z. B. erfolgen durch:

A$ = "gut" ⟨ENTER⟩

Zahlen besitzen stehts die gleich lange interne Darstellung. Bei den KC-Rechnern nämlich immer 4 Byte. Die Strings dagegen können unterschiedlich lang sein, möglich sind Längen von 0 bis 255. Dies hat für BASIC beachtliche Konsequenzen, auf die wir noch genauer eingehen werden. Am eigenwilligsten ist dabei wohl der String der Länge 0. Er besteht aus keinem Zeichen und heißt deshalb auch Leerstring. Er wird wie folgt erzeugt:

A$ = "" ⟨ENTER⟩

Es gibt noch andere Möglichkeiten der Eingabe von Strings, die dann aber auf speziellen Befehlen beruhen. Wir kennen schon die Eingabe mit dem INPUT-Befehl, beispielsweise INPUT A$, bzw. mit Text INPUT "Radius ="; A$
Eine sehr ähnliche Befehlsfolge ist:

A$ = INKEY$ ⟨ENTER⟩

INKEY$ holt während des Programmablaufs eine Information von der Tastatur, ohne daß das Programm angehalten oder die ENTER-Taste gedrückt werden muß. Dabei ist es jedoch nur möglich, ein Tastenzeichen als String einzulesen. So kann man mit INKEY$ den Dialogbetrieb einfacher gestalten. Wenn während der Abfrage keine Eingabe erfolgt, dann behält A$ seinen alten String bei; ansonsten übernimmt er das betätigte Tastaturzeichen. Ein Einzelzeichen kann auch mittels des ASCII-Codes A zugewiesen werden:

A$=CHR$(A) ⟨ENTER⟩

oder aber mittels

A$=VGET$ ⟨ENTER⟩

Die Funktion VGET$ (V von Video und get = nehmen) liefert den Inhalt der Cursorposition als String. Man bewegt hierzu den Cursor an eine Stelle des Bildschirms, und das dort stehende Zeichen liefert die Eingabe.
Schließlich besteht noch die Möglichkeit, die Zahl in einen String zu wandeln.
Beispiel:

A$=STR$ (1.27) ⟨ENTER⟩

Dies erscheint auf den ersten Blick überflüssig. Der Befehl ist jedoch notwendig, weil in BASIC deutlich die numerischen Variablen und Stringvariablen unterschieden werden müssen. Ein Vermischen führt zu dem TM-ERROR (type mismatch).
Eine weitere Möglichkeit, einer Stringvariablen einen Wert zuzuweisen, bietet die DATA-Anweisung (s. Kapitel 16).
Es existieren also 7 Methoden der indirekten Stringbildung:

- A$ = "XYZ"
- INPUT A$
- A$=INKEY$ von INPUT und keybord = Tastatur
- A$=CHR$(ASCII-Code) von character = Zeichen
- A$=STR$(Zahl) STR von String (also sprich: String-string)

- A$=VGET$ von Video und get = nehmen
- DATA-Zeilen.

Eine Kombination dieser Stringbefehle bzw. ihrer Möglichkeiten enthält unser Programm INKEY. Laden Sie dieses Programm und schreiben Sie, von oben beginnend, einen Text auf den Bildschirm. Danach rufen Sie das Programm mit RUN auf und beobachten, was geschieht. Besonders interessant dürfte für Sie dabei die Funktion VGET$ sein.

13.2. Stringablage im Speicher

Nun ist es notwendig, daß wir uns der Ablage der Strings im Speicher zuwenden. Sie erfolgt wesentlich anders als bei den numerischen Variablen bzw. den Arrays. Sie erinnern sich vielleicht noch an Kapitel 3. Dort

hatten wir festgestellt, daß ein BASIC-Programm die Variablen in folgender Weise in einem speziellen Speicherraum ablegt: Zunächst wird beim Start des Programms der gesamte Speicherraum getrennt für die Variablen und Felder ermittelt und reserviert. Dann werden in der Reihenfolge des Auftretens die Plätze für die Variablen bzw. Felder belegt. Beim erstmaligen Belegen spricht man vom Initialisieren. Bei dieser Methode benötigen die zuerst initialisierten Variablen bzw. Felder bei Ihren Aufrufen immer die geringste Zeit.

Nun wollen wir uns damit beschäftigen, wo diese Speicherbereiche liegen. Wir nehmen dazu an, Sie arbeiten beim KC 85/1 bzw. 2 mit dem BASIC-ROM-Modul oder direkt mit dem KC 85/3 bzw. KC 87 und verwenden in allen Fällen keinen RAM-Erweiterungs-Modul. Dann liegt der aktiv für BASIC verwendbare Speicherraum von 3Ø0 bis 4Ø0 H. Ab 4Ø0 H beginnt ihr BASIC-Programm. Es möge beispielsweise bis 450 H reichen. Daran schließt sich dann der Speicherraum für die Variablen an. Er reiche bis 5Ø0 H. Dann folgt der Raum für die Felder, z. B. bis 6Ø0 H. Wenn Sie Ihr Programm nun ändern, so verschieben sich auch diese Speicherbereiche. Deshalb sind danach und nach RUN-Beginn immer alle Variablen zu Ø gesetzt.

Im Speicherbereich für die Variablen liegen nun sowohl die numerischen Variablen als auch die Stringvariablen. Es kann daher A von X$ gefolgt angeordnet sein. Dann steht im Speicher zunächst die Kennzeichnung (1 Zeichen) für eine numerische Variable. Hierfür werden 2 Byte verwendet. Dann folgen 4 Byte, die den aktuellen Zahlenwert enthalten; binär codiert, wie wir es in Kapitel 3 besprachen. Unmittelbar daran schließt sich die Kennzeichnung für A$ als Stringvariable, ebenfalls 2 Byte, an. In den folgenden zugehörenden 4 Byte kann nun aber nicht der dazugehörige String untergebracht werden, denn er kann ja Ø bis 255 Byte lang sein. Folglich stehen jetzt dort seine Länge (1 Byte) und die Adresse, wo er sich wirklich befindet. Um nun zu erfahren, wo der String selbst steht, müssen wir den Speicher, von oben beginnend, untersuchen. Einfluß hat darauf auch die Anweisung:

CLEAR n, m

Das m legt, sofern möglich, die obere Speichergrenze fest. CLEAR n, 12288 bedeutet z. B., daß die obere Speichergrenze auf dem dazugehörigen HEX-Wert von 3ØØØ H liegt. Von hier an, abwärts gerechnet, beginnt der String-Speicherraum. Seine Größe wird durch das n unter CLEAR bestimmt, also CLEAR 5ØØ, 12288 reserviert 5ØØ Byte vom HEX-Wert 3Ø0 H abwärts. Von da an abwärts liegt weiter der BASIC-Stack, auf den wir schon mehrmals eingegangen sind. Er kann also maximal bis zur Grenze der Felder wachsen. Wenn er bis dahin wächst, gibt es das berüchtigte OM-ERROR.

Doch wir wollen uns jetzt ansehen, wie ein String abgelegt wird. Es werden zunächst seine Kennung, Länge und Adresse im Variablenraum abgelegt. Dann wird er selbst im Stringraum gespeichert, und der nächste String folgt danach. Doch was geschieht nun, wenn dem String eine neue, eventuell längere Zeichenkette zugewiesen wird? Sie paßt dann ja nicht in den alten Speicherplatz. Deshalb wird bei jeder neuen Zuweisung immer neuer Speicherplatz im Stringraum verwendet und im Variablenraum nur die neue Adresse und Länge eingetragen. Der Interpreter verschwendet also den Speicherplatz im Stringraum mehr als großzügig. Vielleicht fällt Ihnen ein einfaches, effizienteres Verfahren ein. Allerdings, unter Berücksichtigung aller Umstände, können Sie kaum Glück haben.

Doch nun kommt erst der eigentliche Clou. Irgendwann gelangt der Interpreter bei dieser Methode an die

Grenze des zulässigen Stringraums, und das, obwohl vielleicht im ökonomischen Sinn nur 10 % genutzt sind. Was wird er da wohl machen? Er hält den gesamten Rechenprozeß an und räumt erst einmal im Stringraum auf. Dafür benötigt er Zeit. Der Rechner wirkt während dieser Zeit wie defekt. Nichts funktioniert mehr. Nach Beendigung dieser Säuberungsaktion geht alles unvermittelt weiter. Diese Säuberung heißt in der Fachsprache Garbage-Collection, zuweilen auch garbage connection. Wir übersetzen es am besten mit Müllabfuhr. Damit sie sich hiervon überzeugen können, haben wir Ihnen das Programm GARBAGE geschrieben.
Sie starten es am besten mit einem Stringraum von 2000 und dem String "Versuch", also einem String, der aus sieben Zeichen besteht. Dann wird das Programm bis zur Zahl 250 laufen, den Bildschirm löschen und neu mit 1 beginnen. Es bleibt unvermittelt bei 34 für 11 s stehen, dann läuft es weiter bis zur Zahl 67 und pausiert wieder für 11 s, um dann die 100 für die neue Pause zu erreichen. Und so geht es weiter: Nun können Sie die Parameter variieren und sich einen Überblick über die Garbage-Collection verschaffen. Dabei werden Sie folgende Erkenntnisse gewinnen:
Es sind zwei Zeiten zu unterscheiden, und zwar
– für die Dauer der Garbage-Collection,
– für den Abstand zwischen der Garbage-Collection.
Die Dauer der Garbage-Collection hängt fast ausschließlich von der Anzahl der aktuellen Strings und kaum von deren Länge ab. Sie ist dem Quadrat der Anzahl der Strings proportional. Deshalb gilt z. B. für den KC 85/2,3:

Zeiten für die garbage collection

Anzahl der String	50	100	250	500	1000
Zeit in Sekunden	.4	1.7	11	41	165

Ab etwa 50 Strings wird die Zeitdauer deutlich spürbar; ab 500 fast unerträglich.
Neben dem automatischen Entstehen der Garbage-Collection, ist es auch möglich, sie gezielt vom Interpreter ausführen zu lassen. Dies geschieht z. B. mit dem Befehl FRE(X$). Man fragt den Rechner ab, wieviel Speicherraum im Stringraum noch verfügbar ist. Um diesen Wert zu bestimmen, muß er zuvor die Redundanz im Stringraum beseitigen.

13.3. Die Stringfunktionen STR$, VAL$, CHR$ und ASC

Diese Stringfunktionen betreffen Befehle, welche Zahlen in Strings umwandeln und ASCII-Nummern in ASCII-Zeichen bzw. umgekehrt.
STR$ (Zahl) wandelt die Zahl zu einem String um, VAL(A$) gibt den numerischen Wert des Strings an. Ist das erste Zeichen des Strings kein +, –, und keine Zahl, so ist VAL (String) = 0.
CHR$ (ASCII) stellt das Zeichen der ASCII-Nummer her und ASC (Zeichen) liefert die ASCII-Nummer.

13.4. Ein spaßiges Programm

Das Programm MOPS soll Ihnen, so hoffen wir, auf heitere Weise den Gebrauch von Strings nahebringen. Es wurde in Anlehnung an Carl Reinhardts Gedicht gestaltet:
Wenn der Mops mit der Wurst über den Spucknapf springt und der Storch in der Luft den Frosch verschlingt.
Hierzu gehören mehrere Variationen. Theoretisch sind 6!, also 120 möglich, Reinhardt hat aber nur ca. 10 genutzt. Wir haben statt der 6 zu vertauschenden Wörter 9 eingeführt, und daher gäbe es 9!, also 362 880 Variationen. Sie können natürlich nicht alle durchprobieren, aber das Programm macht es auf eine spezielle Art für Sie. Probieren Sie es aus. Sie brauchen dazu nur 9 Substantive mit den bestimmten Artikeln einzugeben. Der Rest geschieht automatisch. Wir haben es z. B. mit:
DER GOTT, DIE JUNGFRAU, DIE HOELLE
DER TEUFEL, DIE WELT, DAS GELD
DIE SUENDE, DIE DISCO, DAS ELEND
probiert. Am Ende des Kapitels finden Sie den entsprechenden RUN.

Zusammenfassung Kapitel 13

1. BASIC-Zeichen sind die großen und kleinen Buchstaben, die Ziffern und die Sonderzeichen (Komma, Punkt, Fragezeichen...).
2. Die Zusammenfassung von mehreren Zeichen zu einer Einheit heißt Zeichenkette oder String.
3. Zeichenketten sind in vielfältiger Weise erzeugbar. Dabei sind zwei Methoden zu unterscheiden:
   – die direkte Erzeugung durch Anführungszeichen, zwischen denen die gewünschte Zeichenkette unmittelbar steht,
   – die indirekte Realisierung. Hierbei ist das angefügte $ zur Kennzeichnung notwendig. Es wird als String gesprochen.
4. Die Länge einer Zeichenkette kann von 0 bis 255 Zeichen betragen.
5. Genau wie bei den Zahlenwerten existieren auch bei den Zeichenketten Felder (Arrays).
6. Der Speicherraum wird bei BASIC in unterschiedliche Teilbereiche zerstückelt. Von unten beginnend folgen in Reihe:
   – das BASIC-Programm
   – der Variablenraum
   – der Arrayraum
   und von oben beginnend:
   – ein reservierbarer freier Speicherraum für besondere Anwendungen
   – der Stringraum
   – der BASIC-Stack.
7. Wenn der BASIC-Stack in die Nähe des Arrayraums gelangt, erfolgt die Meldung OM-ERROR.
8. Der Stringraum mit seinen beiden Grenzen wird automatisch beim Eintritt in BASIC festgelegt. Dies kann mit der Anweisung CLEAR n, m verändert werden. m legt dabei die obere Adresse und n die Größe des Stringraums fest.
9. Zeichenketten werden effektiv und redundant im Stringraum abgelegt. Deshalb ist in gewissen Abständen ein Aufräumen mit der Garbage-Collection notwendig. Während dieser Zeit verhält sich der Rechner, als wäre er defekt.
10. Nicht im Stringraum abgelegt werden jene Zeichenketten, die im Programm direkt erzeugt werden.
11. Die Dauer einer Garbage-Collection ist im wesentlichen nur von der Zahl der im Programm verwendeten Stringvariablen abhängig. Sie wächst mit dem Quadrat dieser Anzahl. Deshalb wird die Garbage-Collection unter 50 Stringvariablen kaum bemerkt. Über 300 wird sie deutlich störend, und über 600 kann oft nicht mehr effektiv gearbeitet werden.
12. Der Abstand zwischen zwei Garbage-Collectionen hängt von dem noch vorhandenen Speicherraum ab und von der Häufigkeit, mit der im Programm die Zeichenketten verwendet werden.
13. Die Garbage-Collection kann auch gezielt durch den Befehl FRE(X$) ausgelöst werden.

```
10 ! ## INKEY ##
20 DIM A$(38): PRINT CHR$(16);
30 FOR I=0 TO 38
40   A$(I) = VGET$: PRINT CHR$(9);
50 NEXT: PRINT
60 FOR I=0 TO 38
70   PRINT A$(I);
80 NEXT: PRINT CHR$(25);
90 GOTO 30
100 !Bedeutung der ASCII-Zeichen:
110 !  9 Cursor 1 nach rechts
120 ! 16 Cursor home
130 ! 25 Cursor zum Zeilenbeginn
```

```
LIST
LIST  ##  INKEY  ##
LIST  ##  INKEY  ##RINT  CHR$(16);
LIST  ##  INKEY  ##RINT  CHR$(16);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
LIST  ##  INKEY  ##RINT  CHR$(16);(9);
BREAK IN 40
OK
>
```

```
10 CLS: PRINT TAB(12); "##  GARBAGE  ##"
20 PRINT: INPUT"Stringraum ="; A: CLEAR A
30 INPUT"String ="; A$
40 INPUT"DIM ="; N: DIM A$(N): CLS
50  FOR I=1 TO 2 STEP 0
60    FOR K=0 TO N
70      A$(K)=A$: L=L+1: PRINT L;
80      IF L=250 THEN CLS: L=0: M=M+1: PRINT M
90    NEXT
100 NEXT
```

```
Aufteilung des Speicherraumes bei BASIC
=======================================

Speicher- !-----------------!
ende      ! freier Speicher !
          !-----------------!  <---------●
          !  Stringraum     !     CLEAR n,m
          !-----------------!  <-------●
          !     Stack       !
          !       ↓         !
          !                 !
          !                 !
          !                 !
          !-----------------!
          !    Arrays       !
          !-----------------!
          !  Variablen      !
          !-----------------!
          !  Programm       !
          !-----------------!
          !                 !
Speicher- !                 !
beginn    !-----------------!
```

```
10 WINDOW 0,15,0,39: CLS: PRINT TAB(12);"## MOPS ##": B$="Fehler !"
20 PRINT TAB(4); "frei nach Carl Reinhardt 1850"
30 PRINT: DIM A(8), B(8), C$(8), D$(2), E$(2), F$(2)
40 D$(0)= "DER": D$(1)= "DAS": D$(2)= "DIE"
50 E$(0)= "DEM": E$(1)= "DEM": E$(2)= "DER"
60 F$(0)= "DEN": F$(1)= "DAS": F$(2)= "DIE"
70   FOR I=0 TO 8
80    INPUT"Wort mit Artikel";A$
90    IF MID$(A$,4,1)<>" " THEN PRINTB$: GOTO 80
100   IF LEFT$(A$,3)="DER" THEN A(I)=0: GOTO 140
110   IF LEFT$(A$,3)="DAS" THEN A(I)=1: GOTO 140
120   IF LEFT$(A$,3)="DIE" THEN A(I)=2: GOTO 140
130   PRINT B$: GOTO 80
140   C$(I)=MID$(A$,4)
150  NEXT: WINDOW 2,31,0,39: CLS
160 WINDOW 12,31,0,39: CLS
170  FOR I=0 TO 8
180   X=INT(8*RND(1)+.5)
190   C=0: IF I=0 THEN B(I)=X: NEXT
200    FOR J=0 TO I-1
210     IF B(J)=X THEN C=1: J=I
220    NEXT: IF C=1 GOTO180
230   B(I)=X
240  NEXT
250 PRINT"WENN ";D$(A(B(0)));C$(B(0));" MIT ";E$(A(B(1)));C$(B(1))
260 PRINT"  UEBER ";F$(A(B(2)));C$(B(2));" SPRINGT"
270 PRINT"UND ";D$(A(B(3)));C$(B(3));" IN ";E$(A(B(4)));C$(B(4))
280 PRINT"  ";F$(A(B(5)));C$(B(5));" VERSCHLINGT"
290 PRINT"DANN ";D$(A(B(6)));C$(B(6));" AUS ";E$(A(B(7)));C$(B(7))
300 PRINT"  ALS";C$(B(8));" ERKLINGT"
310 INPUT"";A:GOTO160
```

```
## MOPS ##
frei nach Carl Reinhardt 1850

WENN DAS GELD MIT DEM GOTT
  UEBER DIE DISKO SPRINGT
UND DAS ELEND IN DEM TEUFEL
  DIE SUENDE VERSCHLINGT
DANN DIE WELT AUS DER JUNGFAU
  ALS HOELLE ERKLINGT
WENN DIE JUNGFAU MIT DEM GOTT
  UEBER DIE SUENDE SPRINGT
UND DAS ELEND IN DER HOELLE
  DIE DISKO VERSCHLINGT
DANN DER TEUFEL AUS DEM GELD
  ALS WELT ERKLINGT

WENN DER TEUFEL MIT DER WELT
  UEBER DIE HOELLE SPRINGT
UND DAS ELEND IN DER DISKO
  DAS GELD VERSCHLINGT
DANN DIE JUNGFAU AUS DER SUENDE
  ALS GOTT ERKLINGT

WENN DAS ELEND MIT DEM TEUFEL
  UEBER DIE JUNGFAU SPRINGT
UND DAS GELD IN DER WELT
  DIE SUENDE VERSCHLINGT
DANN DIE DISKO AUS DEM GOTT
  ALS HOELLE ERKLINGT

WENN DAS ELEND MIT DER JUNGFAU
  UEBER DIE DISKO SPRINGT
UND DIE HOELLE IN DER SUENDE
  DEN GOTT VERSCHLINGT
DANN DIE WELT AUS DEM GELD
  ALS TEUFEL ERKLINGT

WENN DAS GELD MIT DER WELT
  UEBER DIE JUNGFAU SPRINGT
UND DIE DISKO IN DEM GOTT
  DEN TEUFEL VERSCHLINGT
DANN DIE HOELLE AUS DEM ELEND
  ALS SUENDE ERKLINGT
```

14. Stringmanipulationen

Wir können auch mit Strings in gewissem Umfang operieren. So ist es z. B. möglich, Strings durch das Operationszeichen "+" miteinander zu verknüpfen.
Selbstverständlich können Sie die Strings auch unter Variablen ablegen und diese dann miteinander oder mit Stringkonstanten verknüpfen.
Weiterhin lassen sich Strings durch Vergleichsoperationen miteinander vergleichen oder Teilstrings aus Strings bilden.

14.1. Zusammenfügen von Strings

Bemühen wir uns zunächst um das Einfachste, nämlich das Zusammenfügen von zwei Strings. Nehmen wir an, wir haben C$="Rechner" und B$="Freude" bereits im Programm definiert. Dann könnten wir einfach mit PRINT C$;B$ das neue Wort Rechnerfreude erhalten. Für viele Anwendungen ist es aber günstiger, einen neuen String hierfür zu definieren: Dies ist durch die Konstruktion A$=C$+B$ möglich. Und danach genügt PRINT A$, um den neuen String zu erhalten. Diese Konstruktion erscheint auf den ersten Blick fast trivial. Bedenken Sie aber einmal, was jetzt unser + im Rechner alles bedeuten kann:

- Vorzeichen bei Zahlen und Exponenten
- Addition von Zahlen
- Verbindung von Strings, die auch Konnektion heißt
- einfaches ASCII-Zeichen in Texten.

Zwischen diesen vier Fällen muß der BASIC-Interpreter automatisch unterscheiden. Er muß also aus der Umgebung von "+" erkennen, welcher Fall hier vorliegt, und danach dann handeln. Darüber hinaus muß er sogar erkennen, ob ein unerlaubter Fall vorliegt. Er muß z. B. ERROR anzeigen, wenn Sie versuchen, eine Variable mit "+" zu verwenden:

14.2. Zerstückelung von Strings

Der wichtigere und interessante Fall ist das Zerstückeln von Zeichenketten. Bevor wir darauf eingehen, jedoch noch eine kurze Bemerkung zur Stringfunktion LEN (String). LEN(A$) erzeugt eine Zahl, welche der Anzahl der Zeichen entspricht, die in der Kette A$ enthalten sind.
Also liefert

```
PRINT LEN(A$) ⟨ENTER⟩
```

bei unserem Wort Rechnerfreude die Zahl 13.
Nun wollen wir hieraus wieder die beiden Teilwörter erzeugen. Das sind einmal die 7 ersten bzw. die 6 letzten Zeichen. Sie sind über die Befehle LEFT$(A$,7) bzw. RIGHT$(A$,6) zu erhalten. LEFT$ wählt also immer eine frei wählbare Anzahl, vom Ende gezählt.
Ich meine, dies ist sogar anschaulich leicht zu merken. Wir könnten also mit folgendem Programm alle Teilwörter vom Beginn und vom Ende des String her erhalten:

```
10 INPUT A$: L=LEN(A$)                    ⟨ENTER⟩
20 FOR I=1 TO L                           ⟨ENTER⟩
30 PRINT LEFT$(A$,I), RIGHT$(A$,L-I)
                                          ⟨ENTER⟩
40 NEXT                                   ⟨ENTER⟩
```

Dann ständen die voneinander getrennten Wortteile immer brav nebeneinander, also

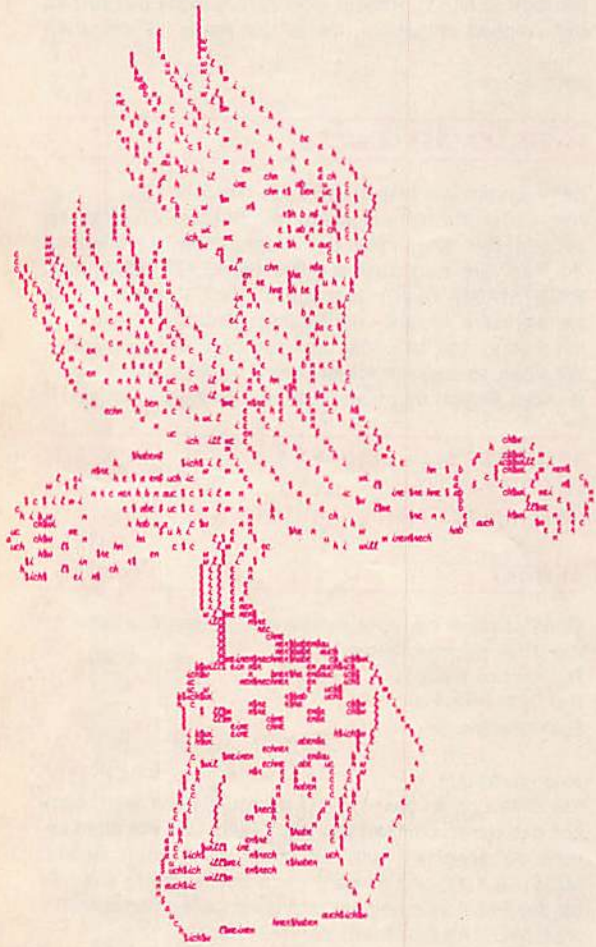
```
R     echnerfreude
Re    chnerfreude
Rec   hnerfreude
...
Rechnerfreude
```

Sie sehen daran bereits, was eventuell noch fehlt, nämlich das Heraustrennen von Wortteilen. Und wie nicht anders zu erwarten, gibt es auch dazu einen Befehl: MID$(A$,X,Y), und er wählt jene Zeichenkette aus, die ab der Position X beginnt und dann Y-Zeichen enthält.
Also MID$(A$,5,3) liefert der Teil "ner".
Nun können wir unsere Zeichenketten nach Herzenslust zerlegen und dann mit + wieder in anderer Weise zusammenfügen.

14.3. Einige Programme

Die Möglichkeiten der Stringmanipulation wollen wir Ihnen an Hand von vier Programmen verdeutlichen. Sie werden sich sicher fragen, warum so viele Programme und sowenig zu deren Beschreibung? Uns ist in der Fachliteratur aufgefallen, daß zwar alle diese Stringfunktionen gut erklärt werden, aber kaum sinnvolle Anwendungen bekannt sind. Hier möchten wir Ihnen Ideen aufzeigen. Unser erstes Programm – STRING1 – ermöglicht es, alle Teilketten aus einem eingegebenen String zu bestimmen. Wenn Sie es probieren, werden Sie ihre Freude daran haben. Wir möchten hier nur ein Beispiel anführen. Was meinen Sie, was alles in dem Wort Mond-

nacht enthalten ist? Wir haben Ihnen am Ende des Kapitels das Ergebnis des RUN-Laufs angegeben. Das Programm verwendet für alles nur den MID$-Befehl. Es werden dabei aber alle zulässigen Werte für X und Y genutzt. Wozu natürlich zuerst die Länge des eingegebenen String zu bestimmen ist.
Unser zweites Programm haben wir SPIEGEL genannt. Wir nehmen an, Sie kennen den schönen Satz von Spoerl:
Ein Neger mit Gazelle zagt im Regen nie.
Er hat die Eigenschaft, von vorn und von hinten gelesen, den gleichen Text zu ergeben. So wie Sie es sicher auch schon bei Wörtern wie ANNA, OTTO, EHE usw. bemerkt haben. Das Programm leistet nun das Folgende: Sie geben einen Text ein, und das Programm ergänzt ihn so,



daß etwas entsteht, was vorwärts und rückwärts gleich ist. Geben Sie z. B. AN ein, so entsteht ANNA. Geben Sie ein: EIN NEGER, so entsteht: EIN NEGER REGEN NIE. Viel Glück wünschen wir Ihnen beim Probieren. Sie gewinnen sicher bald Übung, symmetrisch lesbare Texte zu erzeugen. Dieses Programm arbeitet natürlich bereits in komplexerer Weise mit den neuen Befehlen, und vor allem wird hierbei umfangreich von der Konnexion Gebrauch gemacht.
Wir kommen nun zu unserem dritten Programm. Das Programm KINDER bildet eine Kindersprache nach. Ihre Struktur besteht darin, daß immer dann, wenn ein Vokal im Wort auftritt, etwas Umständliches geschieht. Nehmen wir das Wort gut, so wird daraus "gulefut". Es wird also nach dem Selbstlaut die Silbe LEF eingefügt und dann noch einmal der Selbstlaut wiederholt. Und genau das macht nun unser Programm. "Ich singe mein Lied" wird dann "ilefich silefingelef meilefein lielefied".
Das Programm verwendet alle String-Befehle, die wir bis jetzt besprochen haben.
So und nun unser letztes Programm. Es macht mit den Befehlen etwas Nützliches. Haben Sie es nicht auch schon bedauert, daß unser BASIC nur 6 gültige Stellen besitzt? So können Sie vielleicht nicht einmal Ihr „großes" Sparguthaben verwalten! Denn bei 9999,99 Mark ist das Ende erreicht. Und was soll da erst der Hauptbuchhalter Ihres Betriebes machen? Mit unserem BASIC kann er nicht arbeiten, oder doch? Wir möchten Ihnen jetzt eine simple Methode erklären, mit der es immer geht. Wir formen unsere Zahlen in einen String um. Dann können Sie eine Zahl mit 255 Ziffern verwalten. Das übersteigt sogar das Nationaleinkommen der DDR um ein Vielfaches. Und wenn Sie dies getan haben, brauchen Sie darauf nur eine Arithmetik zu implementieren. Das klingt gelehrt und kompliziert. Aber mit dem Programm LANGARI können Sie bereits 30stellige Zahlen eingeben und addieren. Vielleicht macht Ihnen dieses Programm Mut, weitere Teile zu ergänzen.
Viel Freude beim Knobeln ist unser Wunsch dazu!

Zusammenfassung Kapitel 14

1. Es existieren 7 Methoden der indirekten Stringbildung:
   - A$="XYZ"
   - INPUT A$
   - A$=INKEY$
   - A$=CHR$(ASCII-Zahl)
   - A$=STR$ (Zahl)
   - A$=VGET$
   - aus DATA-Zeilen
2. Folgende Bedeutungen für das Zeichen + werden in BASIC verwendet:
   - als Vorzeichen bei Zahlen und Exponenten
   - bei der Addition von Zahlen
   - bei der Verbindung von Zeichenketten (Konnektion)
   - als direktes ASCII-Zeichen in Texten.
3. Eine Zeichenkette läßt sich mittels drei verschiedener Befehle in Teilketten zerlegen:
   - LEFT$(A$,X) wählt die ersten X-Zeichen von A$ aus,
   - RIGHT$(A$,X) wählt die letzten X-Zeichen von A$ aus,
   - MID$(A$,X,Y) wählt beginnend mit dem X-ten Zeichen von A$ die folgenden Y Zeichen aus.
4. Mit dem Befehl LEN(A$) wird die Anzahl der Zeichen bestimmt, aus der die Zeichenkette A$ besteht.
5. Während STR$(Zahl) eine Zeichenkette erzeugt, bewirkt VAL(A$) die Umwandlung eines String in eine Zahl, sofern das möglich ist. Andernfalls entsteht Ø. VAL kommt von value, zu deutsch: Wert.
6. Während CHR$(ASCII-Zahl) das Zeichen der Zahl herstellt, liefert ASC(Zeichen) die ASCII-Nummer.

```
10 CLS: PRINT TAB(12);"## LANGARI ##":PRINT: DIM A(2,10), S(2)
20 ! Eingabe der beiden Zahlen
30   FOR X=0 TO 1
40    PRINT X+1;: INPUT"Zahl ="; A$: L=LEN(A$): J=0
50    S=INT (L/3): R=L-3*S: IF R=0 THEN S=S-1
60    S(X)=S
70 ! Zerlegung in 3-er Gruppen
80     FOR I=L-2 TO 1 STEP -3
90      A(X,J) = VAL (MID$ (A$,I,3) ): J=J+1
100    NEXT
110 ! Rest beachten
120   IF R>0 THEN A(X,J) = VAL (LEFT$ (A$,R) )
130  NEXT: R=0
140 ! Addition gemaess groesserer Laenge
150 IF S<S(0) THEN S=S(0)
160  FOR X=0 TO S
170   A(2,X) = A(1,X) + A(0,X) + R: R=0
180   IF A(2,X) > 999 THEN A(2,X) = A(2,X) - 1000: R=1
190  NEXT
200 IF R=1 THEN S=S+1: A(2,S) = 1
210 S(2)=S
220 ! Anzeige der Summe
230  FOR I = S(2) TO 0 STEP -1
240   PRINT A(2,I);
250  NEXT: PRINT
```

```
10 CLS: CLEAR 2000: PRINT TAB(9);"## SPIEGEL ##": PRINT
20 PRINT "Text =": INPUT"";A$: L=LEN(A$): B$=""
30 FOR I=0 TO L-1: B$ = B$ + MID$ (A$,L-I,1): NEXT
40 PRINT A$+B$: PRINT :GOTO 20
```

```
10 CLS: PRINT TAB(12)"## KINDER ##": S$="EAOIUY"
20 PRINT: PRINT: PRINT "Text": INPUT ""; A$: L=LEN(A$)
30  FOR I=1 TO 100: B$ = MID$ (A$,I,1)
40   IF I=L+1 THEN I=100: GOTO 150
50    FOR J=1 TO 6
60     IF B$ = MID$ (S$,J,1) THEN A=I: B=1: I=I+1: J=8
70    NEXT
80   IF J<8 THEN NEXT: GOTO 160
90   C$ = MID$ (A$,I,1)
100   FOR K=1 TO 6
110    IF C$ = MID$ (S$,K,1) THEN A=A+1: B=B+1: I=I+1: K=9
120   NEXT: IF K>7 GOTO 90
130  IF J>6 THEN A$ = LEFT$ (A$,A) + "LEF" + RIGHT$ (A$,L-A+B)
140  IF J>6 THEN I=I+3+B: L=LEN(A$): A=0: B=0
150 NEXT
160 PRINT A$: GOTO 20
```

```
10 !:  ## STRING1 ##
20 CLS: INPUT"Wort";A$: L=LEN(A$)
30 FOR I=1 TO L-1
40  FOR J=2 TO L-I+1
50   PRINT MID$(A$,I,J),
60  NEXT: PRINT
70 NEXT: INPUT"";A$: GOTO 20
```

```
## KINDER ##


Text
ICH LIEBE ROTE ROSEN
ILEFICH LIELEFIEBELEFE ROLEFOTELEFE ROLEFOSELEFEN
```

```
MO          MON         MOND
MONDN       MONDNA      MONDNAC
MONDNACH    MONDNACHT
ON          OND         ONDN
ONDNA       ONDNAC      ONDNACH
ONDNACHT
ND          NDN         NDNA
NDNAC       NDNACH      NDNACHT

DN          DNA         DNAC
DNACH       DNACHT
NA          NAC         NACH
NACHT
AC          ACH         ACHT

CH          CHT
HT
```

15. Logische Größen

Das Prinzip der logischen Variablen haben wir indirekt schon in Kapitel 8 eingeführt. Alles was hinter IF vor dem THEN steht, ist nämlich eine logische Größe, die wahr oder falsch sein kann. Wir haben sie bisher jedoch als Bedingung bezeichnet.
Eine Aussage wie z. B. "heute haben wir Vollmond" kann je nach dem Datum wahr oder falsch sein. Eine andere Aussage wäre "3 ist größer 5". Hier wissen wir, daß sie falsch ist.
Unser BASIC-Interpreter verfügt über drei logische Operationen, das logische Oder (Anweisung OR), das Und und die Negation (Anweisung AND und NOT). Durch geeignete Kombination dieser Operationen können wir weitestgehend alle Probleme der Booleschen Algebra bewältigen. Wir können diese Operatoren analog der Umgangssprache einsetzen.

15.1. Die Verarbeitung der logischen Größen im Rechner

Unser Rechner drückt die logische Aussage "falsch" durch eine Ø und die Aussage "wahr" durch eine −1 aus. Dies können wir wie folgt erkennen:
In Direktmode geben wir ein:

```
A=5 und B=7 ⟨ENTER⟩
```

Jetzt lassen wir ihn die Relationen anzeigen:

```
PRINT A>B ⟨ENTER⟩
```

und er zeigt uns

```
Ø
```

Er vergleicht in diesem Fall also A mit B und stellt fest, daß die Aussage falsch ist, folglich ist das Ergebnis Ø.
Bei PRINT A<B ist die Aussage wahr, folglich erhalten wir

```
−1
```

Auch die Aussage PRINT A=B ist falsch, und folglich gilt Ø. Wir wissen, daß diese Problematik für die meisten von Ihnen, besonders für jene, die schon etwas mehr in BASIC programmiert haben, überraschend ist. Die logischen Variablen werden in fast allen Lehr- und Handbüchern nur unter der Bedingung IF behandelt. Dort ist es nicht einmal notwendig, darauf hinzuweisen, daß es sich hier um eine logische Variable handelt. Das Problem liegt im Konzept von BASIC. Andere Programmiersprachen besitzen speziell ausgewiesene logische Variablen, die nach dem Mathematiker Boole auch als Boolesche Variablen bezeichnet werden. In BASIC fehlen sie und müssen deshalb indirekt aus den Zahlenwerten abgeleitet werden. Dies macht die Problematik jedoch nicht nur schwieriger, sondern schafft auch neue Möglichkeiten, auf die wir noch eingehen werden.
Unser kleines Programm WAHR soll Ihnen den Sachverhalt noch einmal verdeutlichen.

```
10 FOR X=−2 TO 2 STEP .25          ⟨ENTER⟩
20 PRINT X,: IF X THEN PRINT"wahr":
ELSE"falsch"                       ⟨ENTER⟩
30 NEXT                            ⟨ENTER⟩
```

Es zeigt für alle Werte außer Ø „wahr" und bei Ø „falsch" an.
Dies ist unser Beweis (vgl. RUN am Programmende). Zugleich haben wir damit auch erkannt, was eine Boolesche Variable ist. Nach der IF-Anweisung wird also immer getestet, ob der Ausdruck den Wert Ø hat bzw. ob er ungleich Ø ist.
Dieser Test kann aber auf zweierlei Art erfolgen: Entweder liegen Ausdrücke vor, die direkt solche Zahlen erzeugen, oder es liegen die in den vergangenen Kapiteln behandelten Vergleichsoperationen vor.

15.2. Vergleichsoperationen

Für Vergleiche stehen die folgenden Operationen zur Verfügung:
<,>,<=,>=,<>oder =
Die Vergleichsoperatoren können auf Zahlen oder Strings angewendet werden. Bei den Strings bestimmen dann die ASCII-Nummern das Ergebnis. Die Ausdrücke, aus denen der Vergleichsausdruck gebildet wird, müssen entweder beide numerische Ausdrücke oder beide Zeichenkettenausdrücke sein, weil eine Zahl nur mit einer Zahl und eine Zeichenkette nur mit einer Zeichenkette verglichen werden kann.
Eine neue Qualität entsteht dann, wenn wir das Ergebnis des Vergleichs, also bei falsch die Ø und bei wahr die −1, einer Variablen zuweisen. Der Übersichtlichkeit halber möchten wir das zunächst mittels Zeichenketten erklären.
Ein String ist "kleiner" als ein anderer, wenn er im Alphabet vorher steht.

```
A$="ALT"; B$="NEU" ⟨ENTER⟩
```

A$ ist also im Vergleich kleiner als B$.
A$<B$ ist also wahr, d. h. −1
Mit

```
LET A=A$<B$ ⟨ENTER⟩
```

erhält A den Wert −1, was sich leicht durch PRINT A überprüfen läßt. Damit bekommt sogar eine völlig unsinnig aussehende Zeile A=B=C einen Sinn.
Das erste Gleichzeichen weist hierin A einen Zahlenwert zu; er entsteht aus dem Vergleich von B mit C.
Setzen Sie einmal für A und B Zahlenwerte ein und prüfen dann das Ergebnis der obigen Zeile mit PRINT A.
Wenn wir der Variablen die Werte −1 für wahr und Ø für falsch zugewiesen haben, so wirkt sie nach einer IF-Anweisung genauso, wie der direkte Vergleich.
Wir können folglich auch schreiben IF A THEN... und nur bei A=Ø wird THEN genutzt. Diese Methode ist dann von großem Vorteil, wenn die Vergleiche kompliziert sind bzw. mehrmals für verschiedene IF-Anweisungen verwendet werden.

15.3. Die Booleschen Operationen

Zur Bildung komplexer logischer Ausdrücke stehen uns in BASIC drei Funktionen zur Verfügung. Wir möchten diese Funktionen umgangssprachlich einführen.
Da ist zunächst das bekannte „nein", „nicht": „Wenn nicht der Rechner defekt ist, dann kann ich mit ihm arbeiten."
In BASIC würde dies etwa lauten:
IF NOT Rechner defekt THEN kann ich mit ihm arbeiten.
Die Anweisung NOT vertauscht die Boolesche Variable in ihrem Wert. NOT bezieht sich immer nur auf einen Ausdruck bzw. auf eine Bedingung. Die Negation NOT ist genau dann wahr, wenn dieser Ausdruck falsch ist bzw. die Bedingung nicht zutrifft.

Da es nur eine Umkehrung ergibt, wird das NOT relativ selten verwendet. Ohne NOT wären folglich die THEN- und ELSE-Zweige zu vertauschen!
Anders verhält es sich mit der Verknüpfung von logischen Ausdrücken. Wir wollen z. B. ein Datum in den Rechner eingeben und alle unzulässigen Eingaben unterdrücken. Beim Monat wären infolgedessen nur die Zahlen 1;2; 3...12 zulässig. Dann würde unsere komplexe Bedingung lauten:
IF A>12 OR A>OR A>INT(A) THEN GOSUB Fehlerroutine.

Das OR entspricht hier also unserem umgangssprachlichen „oder".
Die zweite Verknüpfung von logischen Ausdrücken ist das AND, welches umgangssprachlich dem „und" entspricht. Wir können es gebrauchen, wenn der Monat Februar vorliegt und der Tagewert größer als 29 ist. Nehmen wir an, der Monat sei wieder in A und der Tag in B abgelegt, dann müßten wir programmieren:
IF A=2 AND B>29 THAN Fehlerroutine

An diesen beiden Beispielen erkennen Sie die Möglichkeiten der Befehle AND und OR. Am Ende des Kapitels geben wir Ihnen das Programm WOCHENTAG an. Es realisiert folgendes: Sie geben ein Datum ein und erhalten aus einem immerwährenden Kalender, der ab etwa 1750 gilt, den dazugehörigen Wochentag. Mit diesem Kalender können Sie z. B. feststellen, an welchem Wochentag Sie geboren wurden.
Vielleicht sind Sie ein Sonntagskind!
In diesem Programm sind vielfältige Anwendungen von AND und OR enthalten, die Sie einmal genauer analysieren sollten.

15.4. Anwendung der Boolschen Operatoren auf Zahlenwerte

Wir wissen, daß Wahrheitswerte durch Vergleiche mit eventueller Kombination aus NOT, AND und OR entstehen. Andererseits können hinter IF auch beliebige Zahlenwerte stehen, die dann auf 0, d. h. falsch abgetestet werden. Was ist da nun naheliegender, als auch die Anweisungen NOT, AND und OR auf beliebige Zahlenwerte anzuwenden und zu prüfen, was dabei geschieht. Wir und tun dies im Direktmode und wählen dazu:

```
A=137: B-56                    (ENTER)
```

und testen:

```
PRINT NOT A, NOT B             (ENTER)
```

und erhalten so

```
-138                            -57
```

Es entsteht also eine negative Zahl, die noch um 1 verkleinert wird. Dies bekräftigt die gemachte Aussage, daß durch die Anweisung NOT aus 0, nämlich falsch, −1, also wahr, gemacht wird.

Für die Umkehrung testen wir zwei negative Zahlen:

```
C=-3: D=-1200                  (ENTER)
```

Mit

```
PRINT NOT C, NOT D (ENTER)
```

erhalten wir:

```
2                               1199
```

Auch hier wird folglich das umgekehrte Vorzeichen gebildet und dann 1 abgezogen.
Nun sind wir natürlich neugierig, was der Interpreter mit den Zahlen A und B bei AND bzw. OR anfängt, also:

```
PRINT A AND B, A OR B          (ENTER)
```

```
8                               185
```

Dies ist natürlich für Sie nicht auf Anhieb zu durchschauen. Deshalb geben wir Ihnen hierzu am Ende des Kapitels das Programm ANDOR an. Das Programm macht das Folgende: Sie geben zwei Zahlen ein. Diese Zahlen werden mit AND bzw. OR verknüpft. Aber nicht nur dies, sondern auch noch die zugehörige binäre Folge in 16 Bit hinter jeder Zahl wird erzeugt. Davon erfaßt das erste Bit das Vorzeichen, und die folgenden 15 Zahlen sind eine binäre Codierung der vorzeichenfreien Zahl.
Die Operatoren AND und OR wirken, wie aus den RUN vor ANDOR ersichtlich, einzeln auf alle Bit-Positionen und erzeugen so neue Zahlen. Diese Methode kann für verschiedene Sonderanwendungen von großem Nutzen sein.

Zusammenfassung Kapitel 15

1. Eine Aussage kann wahr oder falsch sein. Im Rechner werden diese beiden möglichen Zustände durch Boolesche Variablen beschrieben. In BASIC werden hierfür meist die Zahlenwerte 0 für falsch und −1 für wahr verwendet.
2. Treten diese Zahlenwerte in einem Ausdruck hinter der IF-Anweisung auf, so wird bei −1 die nach THEN stehende Anweisung ausgeführt. Bei 0 wird die Anweisung nach ELSE oder die nächste Zeile ausgeführt.
3. Allgemein gilt, daß jede Zahl ungleich 0, die nach IF steht, den Zustand wahr bewirkt.
4. Vergleiche erzeugen die Werte −1 für wahr und 0 für falsch. Diese Werte können auch Variablen zugewiesen werden. Steht eine solche Variable nach einer IF-Anweisung, so wirkt sie genauso wie der direkte Vergleich. Diese Methode ist dann von großem Vorteil, wenn die Vergleiche kompliziert sind bzw. mehrmals für verschiedene IF-Anweisungen verwendet werden müssen.
5. In logischen Ausdrücken, also insbesondere hinter IF, sind Verknüpfungen mittels logischer Anweisungen möglich.
   Hierzu stehen zur Verfügung:
   - NOT: Es kehrt den Wahrheitswert, also falsch – wahr, um und entspricht damit der Verneinung.
   - AND: Es sagt aus, daß die davor und dahinter stehenden Aussagen wahr sein müssen, wenn die Gesamtaussage wahr sein soll. Es entspricht also dem sowohl–als–auch.
   - OR: Hier genügt es, wenn einer der beiden Ausdrücke (vor oder hinter OR) wahr ist, damit der Gesamtausdruck wahr ist. Falsch tritt nur auf, wenn beide Ausdrücke zugleich falsch sind.
6. Bei einigen BASIC-Interpretern wird das logische wahr durch +1 dargestellt.

```
10 CLS: PRINT TAB(7);"## WAHR ##": PRINT
20 PRINT" Zahl   Wahrheitswert": PRINT
30 FOR X=-2 TO 2 STEP .25
40 PRINT X, :IF X THEN PRINT "wahr": ELSE PRINT "falsch"
50 NEXT
```

```
      ## WAHR ##

 Zahl   Wahrheitswert

-2           wahr
-1.75        wahr
-1.5         wahr
-1.25        wahr
-1           wahr
-.75         wahr
-.5          wahr
-.25         wahr
 0           falsch
 .25         wahr
 .5          wahr
 .75         wahr
 1           wahr
 1.25        wahr
 1.5         wahr
 1.75        wahr
 2           wahr
```

```
10 CLS: PRINT TAB(15);"## WOCHENTAG ##": PRINT
20 DIM A$(6): A$(0)="Sonntag":A$(1)="Montg":A$(2)="Dienstag"
30 A$(3)="Mittwoch":A$(4)="Donnerstag":A$(5)="Freitag":A$(6)="Samstag"
40 PRINT"Datumseingabe in Zahlen"
50 INPUT" Tag  =";T: IF T>31 THEN GOSUB 160: GOTO 50
60 INPUT"Monat =";M: IF M>12 THEN GOSUB 160: GOTO 60
70 IF M=2 AND T>29 THEN GOSUB 160: GOTO 50
80 IF (M=4 OR M=6 OR M=9 OR M=11) AND T=31 THEN GOSUB 160: GOTO 50
90 INPUT" Jahr =";J: IF J<100 THEN J=J+1900
100 Z=J/100: Z=J-100*INT(Z): X=J/400
110 IF T=29 AND M=2 AND Z=0 AND X>INT(X) THEN GOSUB 160: GOTO 40
120 M=M-2: IF M<1 THEN M=M+12: J=J-1
130 C=INT(J/100): A=J-100*C: B=INT((13*M-1)/5)+INT(A/4)+INT(C/4)
140 W=B+A+T-C-C: W=W-INT(W/7)*7
150 PRINT"dies ist ein ";A$(W): PRINT: GOTO 40
160 PRINT"## Eingabe-Fehler ##": PRINT: RETURN
```

```
10 CLS: PRINT TAB(12); "## ANDOR ##": PRINT
20 DIM A(3), A$(3)
30  FOR X=1 TO 2 STEP 0
40   INPUT"Zahlen ="; A(0), A(1)
50   A(2) = A(0) OR A(1): A(3) = A(0) AND A(1)
60    FOR I=0 TO 3: B=32768: A=A(I)
70     IF A<0 THEN A = 32768 + A: A$(I)="1": ELSE A$(I)="0"
80      FOR J=1 TO 15: B=B/2
90       IF A>=B THEN A=A-B: A$(I)=A$(I)+"1": ELSE A$(I)=A$(I)+"0"
100      IF J=3 OR J=7 OR J=11 THEN A$(I) = A$(I) + " "
110     NEXT
120   NEXT
130   FOR I=0 TO 3
140    PRINT A(I), A$(I)
150    IF I=1 THEN PRINT"    A OR B"
160    IF I=2 THEN PRINT"    A AND B"
170   NEXT: PRINT
180 NEXT
```

```
           ## ANDOR ##

Zahlen = 127  3456
 127         0000 0000 0111 1111
 3456        0000 1101 1000 0000
   A OR B
 3583        0000 1101 1111 1111
   A AND B
 0           0000 0000 0000 0000

Zahlen = 2345              1235
 2345        0000 1001 0010 1001
 1235        0000 0100 1101 0011
   A OR B
 3579        0000 1101 1111 1011
   A AND B
 1           0000 0000 0000 0001
```

16. Wertzuweisung über Datenliste: die Anweisungen DATA, READ und RESTORE

Bisher haben wir zwei Möglichkeiten kennengelernt, einer Variablen einen Wert zuzuweisen. Einmal können wir dies durch eine direkte Wertzuweisung (z. B. X = 1.2), zum anderen haben wir die Möglichkeit, die Wertzuweisung mit Hilfe des INPUT-Befehls zu realisieren. Müssen wir nun größere Datenmengen verarbeiten, so erweisen sich beide Methoden auf Grund der immer wiederkehrenden Eingabe als langwierig. BASIC bietet die Möglichkeit, den Variablen Werte aus einer Datenliste zuzuordnen. Grundsätzlich sind zwei neue Befehle erforderlich: DATA und READ.

16. 1. Die DATA-Anweisung

Die Folge der Elemente der Datenliste wird mit Hilfe der DATA-Anweisung festgelegt. Die DATA-Anweisung hat das Format:
DATA Konstante, Konstante,..., Konstante.
Die Werte der Datenliste können sowohl numerisch als auch Strings sein.
Hinter der Anweisung DATA sind die Daten, also die Werte, die wir verarbeiten wollen, durch Komma getrennt zu schreiben. Damit diese DATA-Zeilen für GOTO und GOSUB nicht unnötig viel Zeit verbrauchen, legt man sie am besten an das Ende des Programms. Dies hat auch für die Programmentwicklung Vorteile. Die Zeilen können nämlich unproblematisch entfernt, ergänzt und eventuell sogar nachgeladen werden. Als Beispiel haben wir Ihnen das Programm ZAHLEN geschrieben, welches zu den eingegebenen Ziffern 0 bis 9 die englischen und französischen Zahlwörter verbal ausgibt. In den DATA-Zeilen steht also u. a.:

```
70 DATA six, six, seven, sept... ⟨ENTER⟩
```

Zugeordnet werden diese Zahlwörter einem Array

```
20 DIM A$ (1,10),                    ⟨ENTER⟩
```

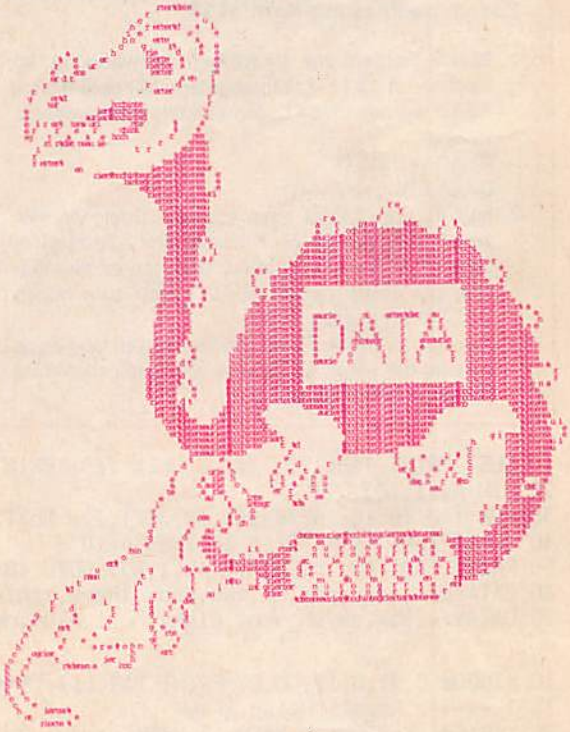
wobei die erste Dimension 0 = englisch, 1 = französisch und die zweite Dimension die Zahlen 0, 1,...,9 bedeutet.
Dementsprechend lautet die Leseoperation:

```
30 FOR I = 0TO10: READ A$ (0,I), A$ (1,I): NEXT
⟨ENTER⟩
```

Über die Eingabe einer Zahl werden danach im Programm die beiden Zahlwörter angezeigt.
Das Programm GLOBUS ist ein besonders langes Programm. Es enthält ca. 600 Datenwerte.
Die Daten in diesem Programm sind die Konturen der Erdoberfläche, und sie werden so verwendet, daß man einen Globus aus jeder möglichen Ansicht auf den Bildschirm zaubern kann. Ein Bild befindet sich am Ende des Kapitels. Wegen der hochauflösenden Grafik ist das Programm nur für den KC 85/2 und 3 geeignet. Wir hoffen, Sie werden Ihre Freude an dem Programm haben und es zeigt Ihnen deutlich, daß eine solche Aufgabe nur mit DATA-Zeilen effizient zu lösen ist.
In den DATA-Zeilen ist es möglich, Zahlenwerte und Strings zu vermischen. Wenn mit der READ-Anweisung versucht wird, einen String einer numerischen Variablen zuzweisen, so erfolgt ein Abbruch mit ERROR-Meldung. Die Umkehrung ist dagegen zulässig. Ja, in den DATA-Zeilen kann für Strings sogar das Anführungszeichen entfallen. Aus DATA 1, 2, 3, können die Zahlen daher sowohl einer numerischen als auch einer Stringvariablen zugewiesen werden. Dies kann vorteilhaft genutzt werden.
An Hand des Programms KEYWORD, zu deutsch: Schlüssel- bzw. Kennwort, wollen wir Ihnen die letzte Aussage verdeutlichen.
Dieses Programm simuliert auf einfache Weise, wie man ein Programm vor falschem Zugriff schützen kann. Sie benötigen dazu ein sog. Paßwort oder eben das Kennwort.
In unserem Programm ist dieses Wort in Zeile

```
110 DATA KYX,3 ⟨ENTER⟩
```

codiert abgelegt. Wir müssen Ihnen hier natürlich die Auflösung geben: Die ersten drei Zeichen, KYX, sind ein String. Seine ASCII-Werte sind um die Zahl 4 größer gewählt. Reduziert man die ASCII-Werte um 4, so entsteht daraus das Kennwort, nämlich „GUT".
Die 3 in der DATA-Zeile ist die Länge des Paßwortes. Das Programm vergleicht nun zuerst die Längen vom Paßwort mit dem eingegebenen Wort. Nur wenn beide gleich lang sind, wird die Zeichenkette weiter verglichen. Wenn auch dies stimmt, können Sie in das Programm gelangen. Wenn Sie ein falsches Wort eingeben, wird sofort das gesamte Programm zerstört:
Zeile 50 endet nämlich mit NEW.



16.2. Lesen der Datenliste: READ-Anweisung

Mit der Anweisung READ Variable, Variable, ..., Variable weisen wir den hinter READ stehenden Variablen die hinter DATA stehenden Werte zu. Diese Anweisung ordnet der ersten Variablen das erste Element der Datenliste, der zweiten Variablen das zweite Element usw. zu.
In einem Programm können mehrere READ-Anweisungen stehen. Jede der READ-Anweisungen setzt dann das Lesen an der Stelle der Datenliste fort, wo es die vorhergehende READ-Anweisung beendete.
Damit der Computer sich merken kann, welchen der

Werte er schon ausgelesen hat und welcher somit der nächste ist, gibt es einen internen DATA-Zeiger. Dieser Zeiger wird vom Startkommando RUN auf das erste Listenelement gesetzt und nach jeder Ausführung einer READ-Anweisung um soviel Positionen weitergestellt, wie Variable gelesen wurden.
Als ein Beispiel haben wir für Sie eine mehrfache Fenstertechnik als Programm FENSTER geschrieben. Nach einer Kopfzeile schreibt das Programm den wesentlichen Teil unseres Bildschirms mit Doppelkreuzen voll. Da wir mit diesem Programm beide Rechnertypen zugleich befriedigen wollten und der KC 85/1 bzw. KC 87 weniger Bildschirmzeilen besitzt, bleibt ein Teil des Bildschirms beim KC 85/2 bzw. 3 frei. Sie können diesen kleinen Mangel bestimmt leicht beheben und lernen dann gleich, noch effektiver mit DATA und READ umzugehen.
Nachdem der Bildschirm also im wesentlichen mit Doppelkreuzen gefüllt ist, werden die 6 verschiedenen Fenster erzeugt. Dies geschieht wie folgt:

```
30 FOR I=1 TO 6 (ENTER)
40 READ A,B,C,D (ENTER)
```

Hiermit werden aus den DATA-Zahlen die ersten 4 Werte den Variablen A bis D zugewiesen. Damit wird nun ein Fenster gebildet und gelöscht:

```
50 WINDOW A,B,C,D : CLS (ENTER)
```

Darauf wird in das Fenster seine Bezeichnung geschrieben:

```
60 PRINT „Fenster"; I (ENTER)
```

Mit der Zeile

```
70 NEXT (ENTER)
```

wird das genau sechsmal durchgeführt. Auf dem Bildschirm sind die verschiedenen Fenster deutlich mit ihrer unterschiedlichen Lage und Größe zu erkennen.

16.3. Rücksetzen des Datenzeigers: RESTORE-Anweisung

Benötigen Sie die gleichen Werte in der gleichen Reihenfolge noch einmal, so brauchen Sie bloß den DATA-Zeiger auf den Anfang zurückzusetzen. Dies können Sie ganz einfach mit der Anweisung RESTORE bewerkstelligen.
Mit RESTORE n kann darüber hinaus der DATA-Zeiger auf den Anfang einer durch n gekennzeichneten Zeile gesetzt werden. Damit ist es möglich, aus den DATA-Werten spezielle Werte auszuwählen.

Zusammenfassung Kapitel 16

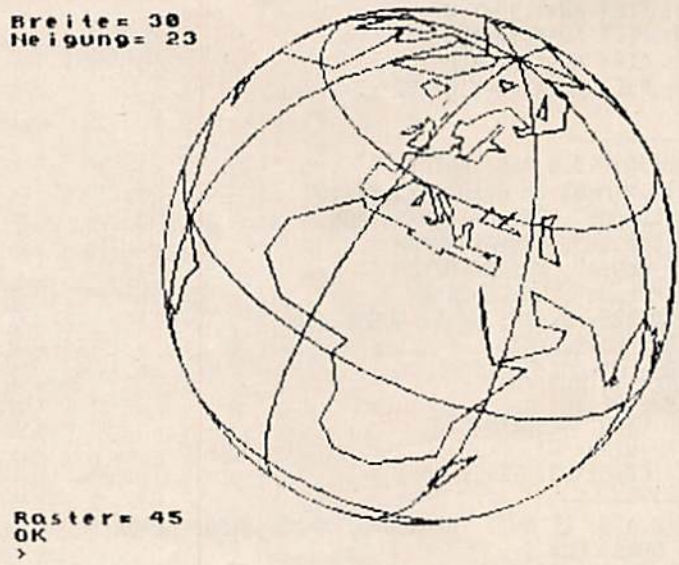
1. Daten werden zur vielfältigen Verwendung im Rechner in DATA-Zeilen abgelegt. Mit dem Befehl READ werden diese Daten Variablen zugewiesen.
   Beispiel:
   READ A,B,C$,D
   DATA 5,12.3,GUT,–35
2. Im Interpreter wird eine spezielle Speicherzeille verwendet, welche die Adresse des zuletzt gelesenen DATA-Wertes notiert. Dadurch ist es möglich, bei einem neuen READ gleich zum neuen DATA-Wert überzugehen.
3. Werden in DATA-Zeilen Zeichenketten abgelegt, so sind die sonst für Strings notwendigen Anführungszeichen nicht nötig.
4. Es ist möglich in DATA-Zeilen Zahlenwerte und Zeichenketten gemischt abzulegen. Die READ-Anweisung darf aber nur zweckmäßig darauf zurückgreifen. Wenn fälschlich mit READ versucht wird, eine Zeichenkette einer numerischen Variablen zuzuweisen, so erfolgt eine Fehlermeldung. Dagegen ist es aber möglich, einen numerischen Wert einer Stringvariablen zuzuweisen.
5. Der Befehl RESTORE setzt den Adressenzeiger für das READ an den Anfang des Programms. Dadurch können Daten immer wieder vom Beginn gelesen werden.
6. Mit RESTORE n kann der Zeiger auf den Anfang einer durch die Zahl n gekennzeichneten Zeile gesetzt werden. So können aus DATA-Tabellen spezielle Zeilen ausgewählt werden.

```
10 CLS: PRINT TAB(12); "## ZAHLEN ##": PRINT
20 DIM A$(1,10)
30 FOR I=0 TO 10: READA$(0,I),A$(1,I): NEXT
40 INPUT"Zahl =";A: IF A 10 THEN A=10
50 PRINT TAB(14); A$(0,A), A$(1,A): GOTO 40
60 DATAzero,zero,one,un,two,deux,three,trois,four,quatre,five,cinq
70 DATAsix,six,seven,sept,eight,huit,nine,neuf,error,defaut
```

```
10 WINDOW 0,31,0,39: CLS: PRINT TAB(11);"**** GLOBUS ****": GOTO 210
15 !------- Kugel- Polar -----
20 Y=R*CP: X=Y*CL: Y=Y*SL: Z=R*SP
30 XX=AX*X+AY*Y+AZ*Z: YY=BX*X+BY*Y+BZ*Z
40 ZZ=CX*X+CY*Y+CZ*Z: IF YY 0 THEN F2=0: F1=0: RETURN
50 X2=INT(X0+XX+H): Y2=INT(Y0-ZZ+H): F2=1
60 IF F1=0 THEN X1=X2: Y1=Y2: F1=1: RETURN
65 !--------- Gerade ---------
70 IF X2=X1 THEN U=1E38: GOTO 110
80 U=(Y2-Y1)/(X2-X1): IF ABS(U)>1 GOTO 110
90 Y=Y1+H: T=SGN(X2-X1): IF T<0 THEN U=-U
100 FOR X=X1 TO X2 STEP T:PSET X,INT(Y),0: Y=Y+U: NEXT: GOTO 130
110 X=X1+H: U=1/U: T=SGN(Y2-Y1): IF T<0 THEN U=-U
120 FOR Y=Y1 TO Y2 STEP T:PSET INT(X),Y,0: X=X+U: NEXT
```

```
130 F1=F2: X1=X2: Y1=Y2: RETURN
135 !-------- Kreis ----------
140 Y2=0: D=INT(R/SQR(2)+H): X2=R*R: X=R: R2=X2+D
150  FOR Y=0 TO D: IF Y2+X2 R2 THEN X2=X2-X-X+1: X=X-1
160   PSET X0+X,Y0+Y,0: PSET X0+Y,Y0+X,0
170   PSET X0-Y,Y0+X,0: PSET X0-X,Y0+Y,0
180   PSET X0-X,Y0-Y,0: PSET X0-Y,Y0-X,0
190   PSET X0+Y,Y0-X,0: PSET X0+X,Y0-Y,0: Y2=Y2+Y+Y+1
200  NEXT: RETURN
205 !--------- Main ----------
210 PRINT" H.Voelz";TAB(30)"13.4.85": PRINT
220 PRINT TAB(10); "Alle Winkel in Grad": BM=PI/180
230 INPUT"Geografische Laenge"; CW: C=-(CW+90)*BM
240 INPUT"Geografische Breite"; AW: A=-AW*BM
250 INPUT"Neigung der Achse"; BW: B=-BW*BM
260 INPUT"Abstand der Kreise"; QW: Q=QW*BM
270 CLS: O=4: X0=190: Y0=128: R=120: H=.5: GOSUB 140
280 S1=SIN(A): S2=SIN(B): S3=SIN(C)
290 C1=COS(A): C2=COS(B): C3=COS(C)
300 AX=C2*C3:  AY=-C2*S3: AZ=S2
310 BX=C1*S3+S1*S2*C3:  BY=C1*C3-S1*S2*S3
320 BZ=-S1*C2:  CX=S1*S3-C1*S2*C3
330 CY=S1*C3+C1*S2*S3:  CZ=C1*C2: F1=0: CO=1
335 !--------- Karte ----------
340 FOR J=1 TO 292: READ A,B: IF A=.1 THEN DO=B: A=AO: B=BO: CO=0
350 IF CO=1 THEN AO=A: BO=B: CO=-1
360 L=A*BM: P=-B*BM: CL=COS(L): SL=SIN(L): CP=COS(P): SP=SIN(P): GOSUB 20
370 IF CO=0 THEN A=DO: READ B: CO=1: F1=0: GOTO 350
380 NEXT
385 !-------- Laengenkreise -----
390 FOR L=0 TO PI-Q STEP Q: F1=0: CL=COS(L): SL=SIN(L)
400   FOR P=0 TO PI+PI+BM STEP 2.5*BM: CP=COS(P): SP=SIN(P): GOSUB 20
410 NEXT:NEXT
415 !-------- Breitenkreise -----
420 FOR P=-PI/2+Q TO PI/2-Q STEP Q: F1=0:CP=COS(P): SP=SIN(P)
430  FOR L=0 TO PI+PI+BM STEP 2.5*BM: CL=COS(L): SL=SIN(L): GOSUB 20
440 NEXT:NEXT
450 PRINT"Laenge="CW: PRINT"Breite="AW: PRINT"Neigung="BW
460 LOCATE 30,0: PRINT"Raster=";QW
470 PAUSE (4): A$=INKEY$:IF A$="" GOTO 470
480 CLS: GOTO 220
485 !----------- Daten ---------
490 DATA355,50,1,51,2,52,358,56,359,57,357,58,353,58,355,55,356,55,.1,28,72
500 DATA15,68,9,63,5,62,6,58,8,58,10,59,12,56,11,56,11,57,9,56,9,54,6,53,3,51
510 DATA356,48,358,45,358,43,352,43,352,36,355,35,358,36,4,44,6,44,9,45,16
520 DATA39,16,38,13,38,15,37,18,39,17,40,18,41,19,40,13,44,14,45,19,42,20,40
530 DATA23,36,25,38,24,40,26,41,27,36,37,36,36,32,21,33,21,32,11,33,12,34
540 DATA357,33,355,34,342,20,342,11,351,5,10,5,8,0,13,-8,12,-18,18,-24,28
550 DATA-24,41,-15,40,-5,52,12,43,12,38,20,34,28,43,12,58,18,60,22,51,23,48
560 DATA30,68,23,77,8,80,10,80,7,82,8,80,10,80,15,90,22,95,16,97,17,99,8
570 DATA104,2,100,13,106,8,110,13,106,20,109,21,109,19,121,27,121,40
580 DATA124,39,126,33,130,35,131,41,140,48,140,57,160,61,155,51
590 DATA163,55,190,67,130,72,95,82,95,76,69,69,35,65,33,67,38,67
600 DATA42,68,.1,12,54,13,55,15,54,21,55,23,59,30,60,22,61,26,66,24,66
610 DATA16,55,.1,23,35,26,35,.1,32,35,34,35,.1,44,-25,47,-25,50,-12,44,-18
620 DATA.1,165,-47,172,-47,178,-38,172,-35,172,-40,.1,128,0,141,-9,153,-12
630 DATA145,-5,.1,118,9,125,5,126,11,122,19,.1,90,8,107,-8,126,-10,106,-5
640 DATA.1,109,2,110,-3,117,-4,119,8,.1,118,-5,123,-5,125,2,120,2,.1,130,30
650 DATA140,36,145,45,142,45,142,55,138,38,130,33,.1,49,38,55,37,51,45,54,48
660 DATA48,47,.1,28,42,42,42,36,44,39,47,34,43,36,44,30,46,31,46,.1,350,51
670 DATA353,53,354,55,353,56,350,53,.1,51,-67,102,-66,140,-67,172,-72
680 DATA160,-81,210,-85,200,-76,225,-75,300,-63,298,-78,354,-71,.1
690 DATA315,60,338,70,345,82,330,84,295,81,290,76,300,75,312,62,.1
700 DATA346,63,346,68,336,67,337,63,.1,18,76,32,80,10,79,.1,50,72,55,71
710 DATA69,78,.1,276,22,278,23,295,20,280,18,278,22,.1,288,13,283,9
720 DATA279,9,276,11,276,15,272,17,273,22,270,21,269,19,263,20,263,28
730 DATA277,30,279,24,278,31,307,48,296,60,282,63,280,51,265,60,280,70
740 DATA286,68,282,63,296,62,298,66,280,74,300,83,228,70,200,70,193,67
750 DATA195,62,202,57,190,53,215,60,235,48,236,35,250,22,245,32,255,18,273,12
760 DATA279,8,283,8,278,-5,290,-19,283,-52,292,-55,296,-54,292,-51,294,-47
```

```
770 DATA293,-45,315,-23,320,-23,322,-12,325,-6,310,0,308,5,298,11,.1
780 DATA114,-22,132,-12,137,-12,140,-16,142,-11,152,-26,148,-43,135,-35
790 DATA115,-35,.1,9,43,9,42,8,41,8,39,10,40,.1,-2,60,-2,62,.1,-7,63,-7,65
```




```
10 CLS: REM: ## KEYWORD ##
20 INPUT"Geben Sie Ihr Kennwort ein";A$: L=LEN(A$)
30 READ K$,K
40 IF L=K GOTO 60
50 PRINT"Sie sind fuer dieses Programm nicht zugelassen": NEW
60 FOR I=1 TO K
70  A=ASC (MID$ (K$,I,1)) -4
80  IF MID$ (A$,I,1) <>CHR$(A) GOTO 50
90 NEXT
100 PRINT"Herzlich willkommen": PRINT"Hier muesste das Programm beginnen"
110 DATA KYX,3
```

```
10 CLS: PRINT TAB(12);"## FENSTER ##"
20 FOR I=1 TO 879: PRINT"#";: NEXT
30  FOR I=1 TO 6
40   READ A, B, C, D
50   WINDOW A, B, C, D : CLS
60   PRINT"Fenster";I
70  NEXT
80 INPUT A,: WINDOW 0,22,0,39
90 DATA 2,7,1,7,    6,10,9,12,    2,3,10,30
100 DATA14,19,3,12, 9,19,15,22,  6,21,24,36
```

17. Der Befehl RND(X)

17.1. Der Zufall

Bereits mehrfach haben wir den Befehl RND(X) verwendet. In diesem Abschnitt wollen wir ihn systematisch behandeln. Dabei ist es notwendig, ein wenig genauer auf den Zufall und seine Realisierung durch diese Funktion einzugehen. Außerdem sind objektive Gegebenheiten und subjektive Feststellungen deutlich zu unterscheiden.

Das allen am besten bekannte Gerät zum Erzeugen des Zufalls ist der Würfel. Wenn er nicht manipuliert ist, treten die Augenzahlen 1, 2,...,6 mit gleicher Wahrscheinlichkeit auf. So sagt es die Physik. Die Praxis zeigt aber, daß zeitweilig erhebliche Abweichungen zu erwarten sind. Jemand hat eben zu bestimmten Zeiten Glück oder Pech. So wird es zumindest von den meisten gesehen. Diesem scheinbaren Widerspruch wollen wir zunächst nachgehen. Die Aussage der Physik gilt dann, wenn man über sehr lange Zeiten mittelt. Bei wenigen Würfen gibt es dagegen fast notwendig Abweichungen. Die Beispiele werden das noch genauer zeigen.

17.2. Aufruf des RND(X)-Befehls

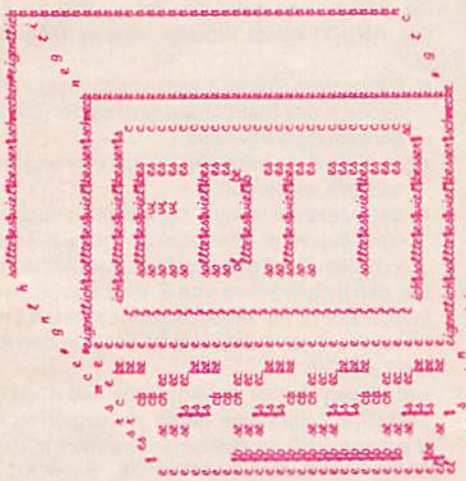
Jeder Aufruf dieses Befehls erzeugt zufällige Zahlenwerte zwischen Ø und 1. Dies funktioniert – auch wenn es nicht einfach vorstellbar ist – ähnlich wie ein Würfel. Jedoch nicht nur mit 6, sondern mit sehr vielen Flächen, auf denen Zahlen stehen. Der Interpreter macht das natürlich elektronisch. In seinem Speicher befinden sich zwei Zahlen, die miteinander multipliziert werden. Von dem Ergebnis wird dann nur jener Teil verwendet, der nach dem Komma steht. Dadurch liegen die Zahlen zwischen Ø und 1. Damit diese Zahlen zufällig auftreten, müssen spezielle Parameter ausgewählt werden, und meist ist auch die Arithmetik noch ein wenig komplizierter. Es gehört schon zu den schwierigen Problemen, diese Arithmetik so zu organisieren, daß in den Zahlenfolgen, die beim Aufruf der Funktion erzeugt werden, keine Gesetzmäßigkeit auftritt. Genauer gesagt, werden so lange Perioden erzeugt, daß Wiederholungen erst nach ca. 1 Million Zufallszahlen auftreten. Außerdem ist dafür gesorgt, daß alle Zahlenbereiche zwischen Ø und 1 mit gleicher Häufigkeit auftreten. Diesem Problem wollen wir uns mit dem ersten Programm RND1 zuwenden. Es erzeugt 4ØØ Zufallszahlen, gemäß:

```
30 FOR I=1 TO 400 (ENTER)
40 A=20*RND(1)... (ENTER)
```

Durch den Faktor 20 werden Zahlen zwischen 0 und 19.9999 erzeugt. Davon wird aber nur der ganzzahlige Wert genutzt. Bei jeder erzeugten Zahl wird dann der zugehörige String des Feldes Z$(19) um ein # verlängert. Deshalb geht das Programm wie folgt weiter:

```
40... Z$(A)=Z$(A)+"#" (ENTER)
50 NEXT (ENTER)
```

Nach dem Ablauf des Programms ist daher die Häufigkeit für jeden Zahlenbereich durch die Länge des String bzw. die Anzahl der # hinter der Zahl unmittelbar zu sehen. Schauen Sie sich darauf das Programm und den gezeigten Ablauf am Ende dieses Abschnittes an. In grober Näherung kann von einer Gleichverteilung gesprochen werden. Aber bei den ausgewählten 400 Werten waren eben die Zahlen mit 1 und 15 besonders häufig (subjektiv Glückszahlen) und die mit 3 und 11 relativ selten (Pechzahlen). Jeder andere Durchlauf des Programms erzeugt andere Verhältnisse. Wenn Sie das Programm oft genug laufen lassen, werden Sie die einleitenden Worte dieses Abschnittes besser verstehen. Das Glück oder Pech ist eben nichts anderes als eine Abweichung, die für eine bestimmte Zeit nun eben einmal auftritt.



17.3. Test auf Gleichverteilung

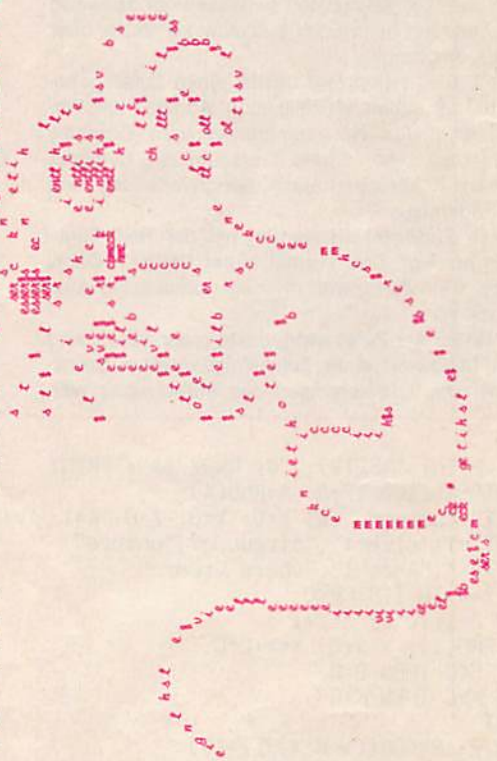
Mit einem zweiten Programm RND2 wollen wir nun die Eigenschaft der Geichverteilung exakter testen. Hierzu addieren wir alle erzeugten Zahlen etappenweise und teilen sie durch die Anzahl der Zahlen. Dann muß langfristig ein Wert von 0.5 entstehen.
Eine zweite wichtige Eigenschaft von Zufallsfolgen ist die Streuung. Vereinfacht gilt für Sie, daß mit ihr die mittlere Differenz der erzeugten Zahlen vom Mittelwert, also 0.5, ausgedrückt wird. Genauer: Die Differenz wird noch quadriert, summiert und dann die Wurzel gezogen. Für eine Gleichverteilung der Zahlen zwischen 0 und 1 muß die Streuung 0.288675 betragen. Außerdem bestimmt das Programm noch die kleinste und die größte Zahl, die während des Ablaufs auftreten.
Im Gegensatz zum vorigen Programm wird hier immer weiter addiert. Wenn Sie das Programm starten, werden Sie feststellen, daß zu jedem Zeitpunkt gewisse Abweichungen von den idealen Werten auftreten. Die Abweichungen werden aber im Mittel um so geringer, je länger das Programm läuft. Zeitweilig können die Abweichungen wieder etwas zunehmen. Wesentlich ist jedoch die Tendenz. Hier wirkt sich das Gesetz der großen Zahl aus, das folgendes aussagt: Je länger ein Zufall wirkt, um so mehr nähern sich Mittelwert und Streuung den theoretischen Grenzwerten. Die Abweichungen davon nehmen in der Tendenz deutlich ab.

17.4. Der Parameter von RND

Bei dem Programm RND2 haben wir auch einen Anfangswert zur Eingabe vorgesehen. Seine Größe beeinflußt die Eigenschaften des Zufallsgenerator insbesondere bei seinem Start. Hierbei sind drei Fälle zu unterscheiden:

- Für alle positiven Werte geschieht das bisher beschriebene: Nach dem inneren Gesetz des Zufallsgenerators wird eine neue Zufallszahl berechnet.
- Für die Eingabe 0 wird die letzte Zufallszahl noch einmal wiederholt. Hierdurch haben wir die Möglichkeit, ohne dabei den Zufallsgenerator irgendwie zu beeinflussen, die letzte Zahl erneut zu verwenden.
- Neuartig verhält sich der Zufallsgenerator dann, wenn in der Klammer von RND eine negative Zahl steht. Dann wird er nämlich aus seinem Takt gebracht. Aus der negativen Zahl wird in komplizierter Weise eine zu ihr gehörende Zufallszahl gebildet.

So kann der Zufallsgenerator in eine genau definierte, immer wiederholbare Startposition gebracht werden.



Erst mit einem darauffolgenden RND (von einer positiven Zahl) entsteht wieder das übliche Zufallsspiel. Wir können also mit einer negativen Zahl eine definierte Zahlenfolge von Zufallszahlen erzeugen. Dies ist für verschiedene Anwendungen nützlich. Denken Sie vielleicht einmal an unser Programm MOPS. Dort waren ja 9!=362880 Variationen denkbar. Sie wurden per Zufall ausgewählt. Wenn wir nun eine bestimmte davon wiederholen wollen, so ist das kaum möglich. Hätten wir

aber den Start mit einer negativen Zahl festgelegt, z. B. im einleitenden Teil mit RND(−3), so wäre dies leicht möglich.

Damit wir uns auch diesen Fakt etwas genauer ansehen können, gibt es das Programm RND3. Es erzeugt immer 12 Zufallszahlen, nachdem einmalig von Hand für die erste Zufallszahl der Parameter eingegeben wurde. Hierdurch lassen sich in augenfälliger Weise die o. g. Schlußfolgerungen beobachten.

17.5. Zufälliger Start

Aus den bisherigen Erklärungen und den zugehörigen Programmen können Sie also eine Menge über den Zufall entnehmen. Es bleibt lediglich noch ein Punkt übrig, nämlich: Wie startet der Interpreter von sich aus die Zufallsfolge? Auch hierzu ist das letzte Programm geeignet: Sie schalten den Rechner neu ein, starten BASIC, lesen das Programm RND3 ein und starten es mit 1 als Parameter. Dann entsteht bei den Rechnern KC 85/1 bis 3 und KC 87 immer die Zufallszahl .245121

Die Folge geht ohne Eingriffe dann weiter: .305003; .311866; .515163 usw.
Die Rechner starten also mit einer genau definierten Folge. Sie kann weder durch neues RUN noch durch BYE und REBASIC bzw. WBASIC in ihrem Ablauf beeinflußt werden. Es sei denn, Sie starten BASIC völlig neu oder setzen den Zufallsgenerator mit einer negativen Zahl an den Beginn einer definierten Folge. Aus dieser Problematik erkennen Sie, daß etwas fehlt, nämlich eine zufällige Startposition des Zufallsgenerators. Hierzu dient der Befehl RANDOMIZE.
Er macht das Folgende: Im Mikroprozessor unseres Rechners gibt es ein sog. Refresh-Register. Es arbeitet mit den Speichern des Rechners zusammen und ändert sehr schnell seinen Zahlenwert. Wenn der Befehl RANDOMIZE auftritt, fragt der Interpreter dieses Register nach seinem Zahlenwert ab und verwendet ihn als negative Größe in der Klammer von RND. Damit hängt es dann wirklich vom Zufall ab, wo unser Zufallsgenerator beginnt. Dieser Befehl existiert jedoch nur im KC 85/2 bzw. 3. Beim KC 85/1 bzw. KC 87 wird empfohlen, die Systemuhr abzufragen.

Zusammenfassung Kapitel 17

1. Es gibt zufällige Ereignisse, wie sie z. B. bei einem fehlerfreien Würfel existieren. Hierbei treten aus physikalischen Gründen alle Augenzahlen mit gleicher Wahrscheinlichkeit auf. Abweichungen von der Gleichverteilung existieren nur zeitweilig und werden beim Spiel subjektiv als Glück oder Pech empfunden.
2. Der BASIC-Interpreter besitzt einen Zufallsgenerator, der mittels Multiplikation, Addition und Abtrennung des Nachkommateils vom Ergebnis Zahlen zwischen Ø und 1 erzeugt. Sie genügen hohen Ansprüchen nach Gleichverteilung und Periodendauer.
3. Diese Zufallszahlen werden mit der RND-Funktion erzeugt. Das Format lautet RND(X). Dabei kann X ein Zahlenwert oder ein numerischer Ausdruck sein.
4. Je länger der Zufall wirkt, desto mehr nähert sich der Mittelwert einer Zufallsfolge seinem Grenzwert. Die Abweichungen des Mittelwertes werden dabei regelmäßig kleiner.
5. Die Bildung von zufälligen Zahlen mit der Funktion RND(X) bietet unterschiedliche Möglichkeiten:
   - Für positive Werte X wird unabhängig von der Größe X das Standardprogramm der Zufallsfolgenbildung aufgerufen.
   - Ist X = Ø, so wird die zuletzt verwendete Zufallszahl wiederholt.
   - Bei negativen X wird der Zufallsgenerator auf eine definierte Startposition gesetzt, die nur von dem Wert der negativen Zahl abhängt.
6. Bei der Initialisierung von BASIC startet der Zufallsgenerator mit einer genau definierten Zufallsfolge. Die erste Zufallszahl ist dabei in den KC 85 die Zahl .245121.
7. Um diesen Urstart zu vermeiden und auch keine definierte Zufallsfolge durch ein negatives Argument von RND zu realisieren, existiert in den KC 85/2,3 der Befehl RANDOMIZE. Beim KC 85/1 bzw. KC 87 kann der aktuelle Sekundenstand der eingebauten Systemuhr genutzt werden.

```
10 CLS: PRINT TAB(12); "## RND2 ##": PRINT
20 INPUT"Startwert";A: A=RND(A)
30 INPUT" Abstand ";B: X=0: Y=0: Z=0: K=1: G=0: IF B 2THEN B=2
40 PRINT"Mittelwert","Streuung","untere"
50 PRINT" ","Anzahl","obere Grenze"
60 FOR I=0 TO 1 STEP 0
70  FOR J=1 TO B: Z=Z+1
80   C=RND(1): X=X+C: Y=Y+C*C
90   IF G<C THEN G=C
100  IF K>C THEN K=C
110 NEXT
120 M=X/Z: S=SQR((Y-M*X)/(Z-1))
130 PRINT M, S, K: PRINT" ", Z, G
140 NEXT
```

```
                    ## RND2 ##

 Startwert .604465
  Abstand  10
 Mittelwert   Streuung     untere
              Anzahl       obere Grenze
  .485415      .218865      .109422
               10           .773794
  .514017      .244186      .0872055
               20           .932633
  .511236      .256777      .056528
               30           .965555
  .514084      .261824      .056528
               40           .965752
  .50443       .26375       .0235308
               50           .965752
```

```
10 CLEAR 1000: DIM Z$(19)
20 FOR I=0 TO 19: Z$(I)="": NEXT
30 FOR I=1 TO 400
40  A=20*RND(1): Z$(A)=Z$(A)+"#"
50 NEXT
60 CLS: PRINT TAB(13); "## RND1 ##": PRINT
70 FOR I=0 TO 19
80  PRINT I+1; TAB(5); Z$(I)
90 NEXT: GOTO 20
```

```
          ## RND1 ##

1   ##############################
2   ##################
3   ############
4   ##############
5   ####################
6   ################
7   ##################
8   ########################
9   ####################
10  ####################
11  ############
12  ################
13  ################
14  ##################
15  ##############################
16  ##################
17  ##############
18  ##################
19  ####################
20  ########################
```

18. Frei definierbare Funktionen

In BASIC gibt es eine Vielzahl fest eingebauter Funktionen. Hierzu gehören u. a. ABS(X), INT(X), RND(X), LOG(X),SIN(X). Für viele Anwendungen – vor allem auf mathematischem Gebiet – ist es nützlich, auch über andere Funktionen zu verfügen. Da es aber kaum möglich ist, alle Funktionen einzubauen, existiert in BASIC ein anderer Weg: Wir können spezielle Funktionen definieren. So gibt es z. B. die Kettenlinie. Sie entsteht, wenn wir eine Leitung an zwei Punkten halten, also bei Telegrafen- oder Freileitungen. Sie kann aus zwei Exponentialfunktionen aufgebaut werden. Um diese Funktionskombination dann nicht immer wieder neu beschreiben zu müssen, kann sie vorteilhaft für ein Programm festgeschrieben werden. Das zugehörige Befehlspaar beginnt mit dem Definieren gemäß:
DEF FN Name(Variable)
Genutzt wird die Funktion durch das Aufrufen der definierten Funktion, z. B. Y = FN Name(X)
Der Name kann wie bei den Variablen aus einem Buchstaben oder einem Buchstaben mit folgendem Buchstaben oder Zahl bestehen. Von der Variablen X wird über die definierte Funktion der Funktionswert dem Y zugewiesen. Formal hat die definierte Funktion beachtliche Ähnlichkeit mit GOSUB. Es ist, vereinfacht gesprochen, ein einerseits eingeschränkter GOSUB, aber andererseits mit zusätzlichen Möglichkeiten. Vergleiche hierzu enthält die Zusammenfassung am Ende dieses Abschnittes.

18.1. Ein einfaches Beispiel

Stellen Sie sich bitte vor, Sie haben eine Zeitdauer in Stunden, Minuten und Sekunden, und Sie wollen nun

```
10 CLS: PRINT TAB(10);"## RND3 ##": PRINT
20 PRINT: INPUT"Starwert =";A: PRINT RND(A)
30 FOR I=0 TO 11
40  PRINT RND(1),
50 NEXT: GOTO 20
```

```
          ## RND3 ##

Startwert =-1
 .308601
 .498871        .670127        .98706
 .739354        .783018        .949844
 .55241         .681371        .823571
 .244878        .421504        .775332

Startwert = 0
 .775332
 .310637        .346463        .056878
 .516925        .414609        .441716
 .424417        .893161        .0405661
 .4474          .879268        .712771
```

wissen, wieviel Sekunden das sind. Dann müssen Sie folgendes beachten:
1 Stunde hat 60 Minuten und 1 Minute hat 60 Sekunden. Effektiv rechnen Sie die Stunden zunächst in Minuten um, indem Sie die Stundenzahl mit 60 multiplizieren. Dann können Sie dazu die Minuten addieren. Danach multiplizieren Sie die zusammengefaßten Minuten wieder mit 60, erhalten so die Sekunden für die Stunden und Minuten und brauchen nur noch die Sekunden zu addieren.
In diesem Beispiel ist also zweimal dasselbe zu tun, nämlich die Multiplikation mit 60. Es ist also nützlich, hierfür eine Funktion zu definieren:
Wir setzen Z=60 und schreiben dann:

```
20 DEF FN A(X)=Z*X ⟨ENTER⟩
```

Zunächst machen wir uns klar, wie die Minuten umzurechnen sind: FN(M) erzeugt so genau die M*60s. FN(H) erzeugt aus den Stunden die Minuten.
Hier sehen wir bereits einen Vorteil der definierten Funktion: sie wird mit frei wählbaren Variablen (dummy) definiert, die bei der Anwendung durch irgendeine ersetzt werden kann. Dies ist also mehr, als mit einem GOSUB zu verwirklichen wäre.
Damit können wir nun beide Fälle für die Umrechnung von Minuten und Stunden zusammenfassen:
FN A(FN (H)+M)
Jetzt ist der Vorteil deutlich. Und die vollständige Zeile lautet:

```
80 PRINT FN A(FN (A(H) +M) + S ⟨ENTER⟩
```

Das entsprechende Programm steht am Ende als"ZEIT". Es enthält noch eine zweite Teilroutine, nämlich die Umkehrung. In ihm verwenden wir eine zweite definierte Funktion:

```
30 DEF FN B(X)= INT(X/Z) ⟨ENTER⟩
```

Hier wird X durch Z=60 dividiert und dann der ganzzahlige Anteil gebildet. 428 s sind 7 min und 8 s. Die 7 min erhält man über 428/60 = 7,13.

Dieselbe Funktion kann man auch verwenden, um aus vielen Minuten, die in ihnen enthaltenen Stunden zu bestimmen. Der genaue Vorgang zur Umrechnung von sehr vielen Sekunden in Stunden, Minuten und Sekunden ist im Programm in den Zeilen 100, 110 enthalten. Bitte versuchen Sie, diese Lösung zu verstehen.

18.2. Stringvariablen und IF-THEN-ELSE nicht zulässig

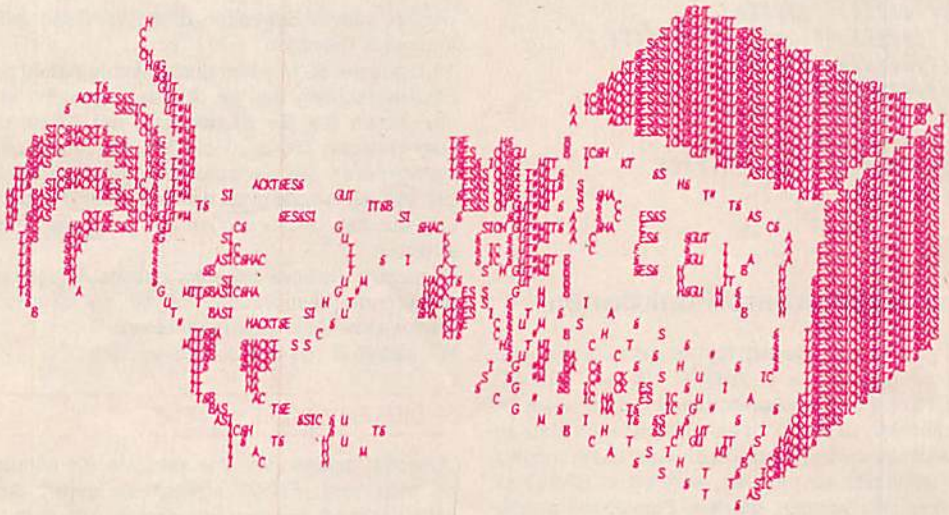
Für die definierte Funktion sind nur numerische Variablen zugelassen. Bitte versuchen Sie sich hiervon durch Experimentieren zu überzeugen. Es gibt jedoch Auswege. Strings lassen sich ja in numerische Werte verwandeln. Hierzu dienen u. a. die Befehle VAL(A$) und ASC(A$). Vielleicht operieren Sie mal mit der Funktion DEF FN ST(VAL(CHR$(X)).
Nur für X zwischen 45 und 57 geschieht etwas Interessantes.
Die definierte Funktion muß immer ein abgeschlossener Ausdruck sein. Dies bedeutet u. a., daß Konstruktionen wie IF-THEN-ELSE oder FOR-TO-STEP-NEXT nicht in der Funktionsdefinition vorkommen dürfen. Dies ist eine deutliche Einschränkung gegenüber dem Aufruf von Unterprogrammen.

```
200 DEF FN F(X)=X+1 - INT((X+1)/3)*3
                                    ⟨ENTER⟩
```

Damit wird also die Zahlenfolge 1, 2, 3, 1, 2, 3, 1, 2, 3... erzeugt. Der Wert von X wird durch diese Funktion jeweils um 1 erhöht. Wenn dabei aber der Wert 4 entsteht, wird 3 abgezogen. In der mathematischen Fachsprache wird also in der Funktion, neben der Erhöhung um 1, modulo 3 gebildet.
Außerdem können Sie in diesem Beispiel wieder eine komplexe Verwendung von Feldern feststellen. Versuchen Sie, die nicht ganz einfache Lösung zu begreifen. Auch wenn es Ihnen Mühe bereitet, lernen Sie dabei den Umgang mit Feldern noch besser zu verstehen.

18.4. Zeitverbrauch von FN, Vergleich mit GOSUB

Zeitanalysen haben wir schon mehrfach, insbesondere bei GOTO und GOSUB, durchgeführt. Sie haben hier ganz analog zu erfolgen und führen dann etwa zu folgendem Ergebnis: Für den Aufruf einer definierten Funktion, benötigen wir etwa 2,2 ms mehr, als wenn die Funktion direkt an dieser Stelle ins Programm eingefügt wäre. Dies ist fast genau der gleiche Wert, wie er für ein GO-



18.3. Die Türme von Hanoi

Wir wollen uns jetzt einem zweiten Beispiel für die definierte Funktion zuwenden. Es ist unter dem Namen Türme von Hanoi bekannt. Hierbei existieren z. B. 10 Scheiben mit unterschiedlichem Durchmesser. Die Scheiben sind durch das Feld A$(X) graphisch gegeben. Sie können auf drei Stecken gelegt werden. Es ist aber nur erlaubt, daß kleinere Scheiben auf größeren liegen. Zu Beginn befinden sich alle Scheiben, richtig geordnet, auf dem linken Stab. Sie sollen jetzt wohlgeordnet auf den rechten Stab gebracht werden. Dazu sind minimal 1024 Züge notwendig. Das Programm ist so geschrieben, daß es dies für Sie in optimaler Weise vollzieht. Damit Sie diesen Ablauf gut verfolgen können, haben wir für jeden Zug etwa 3.5 s gewählt. Teilweise wird diese Zeit durch die neue Funktion PAUSE(X) bestimmt. Mit dem Parameter in diesem Befehl können Sie in der Zeile 70 den Ablauf beschleunigen oder verlangsamen. In der programmierten Version dauert der ganze Ablauf rund 1 h! Die definierte Funktion wird in diesem Beispiel für den Wechsel zwischen den Stapeln verwendet. Sie erhalten die Nummern 0, 1, 2. Zwischen diesen Stapeln ist dann zyklisch zu wechseln. Dazu dient die Funktion:

SUB ohne Berücksichtigung der Zeilensuche benötigt wird. Bei der definierten Funktion kann, im Gegensatz zu GOSUB, die Zeilensuche vernachlässigt werden. Dies ist immer ein beachtlicher Zeitvorteil.
Etwas Zeit wird zusätzlich dann verbraucht, wenn in einem Programm mehrere Funktionen definiert wurden. Dann muß der Interpreter zwischen diesen Funktionen unterscheiden. Für jede zusätzlich definierte Funktion verlängert sich die Suche zunächst um etwa 0.15 ms. Außerdem ist die Reihenfolge der Funktionen bedeutsam. Für jede unnötige Suche müssen noch ca. 0.08 ms hinzugefügt werden. Aber selbst die Summe dieser Werte bleibt immer klein gegenüber den Zeilensuchzeiten von GOSUB mit ca 0.5 bis 0.8 ms je Zeile. Die definierte Funktion ist also immer schneller. Weitere Vorteile von ihr sind:

Die schon beschriebene Dummyvariable, welche einen nahezu universellen Einsatz zuläßt. Weiter bringt sie erhebliche Vorteile bezüglich der Übersichtlichkeit von Programmen. Bei einem Aufruf von FN A(X) weiß man, im Gegensatz zu GOSUB, sofort, was hier geschieht. Auch sind die Definitionszeilen leichter überschaubar.
Schließlich gibt es Möglichkeiten, mittels Maschinenpro-

gramm die definierte Funktion in der DEF-FN-Zeile zu ändern bzw. interaktiv dort direkt einzugeben.
Bei GOSUB haben wir uns mehr oder weniger erfolgreich mit dem Selbstaufruf beschäftigt. Versuchen wir das auch hier und definieren:

DEF FN A(X) = FN A(X)*(X−1) ⟨ENTER⟩

Wir verwenden also im Definitionsteil, d. h. links vom Gleichheitszeichen, wieder die definierte Funktion. So etwas ist in verschiedenen Sprachen, wie z. B. PASCAL und LISP, möglich. Es wird dann von rekursiver Programmierung gesprochen. In BASIC geht dies leider nicht. Auch nicht in diesem noch recht „übersichtlichen" Fall. Die Funktion legt dann nämlich beim ersten Aufruf ständig die notwendigen Parameter auf dem Stack ab, und in kurzer Zeit folgt:

?OM ERROR IN XX

Einen zusammengefaßten Vergleich von GOSUB und DEF FN finden Sie am Ende dieses Abschnittes.

18.5. Ein Grafikbeispiel

Zu Ihrer Erholung wollen wir jetzt das Programm MUSTER besprechen. Es verlangt allerdings die Vollgrafik des KC 85/2 bzw. 3 und zaubert Ihnen schöne Muster auf den Bildschirm. Dabei werden mehrfach und kombiniert die goniometrischen Funktionen COS und SIN verwendet. Die Anwendung dieser Funktion in Kombination mit einigen Parametern erfolgt in zwei DEF FN. Für den Ablauf des Programms brauchen Sie wieder etwas Geduld, je Bild ca. 20 min. Aber das Ergebnis ist sehenswert.

Zusammenfassung Kapitel 18

1. Mit DEF FN A(X) kann eine Funktion zur späteren Verwendung definiert werden. Darin ist FN der Funktionsaufruf und A der Name der Funktion. A wird wie eine Variable gebildet.
2. Der in Klammern gesetzte Ausdruck X ist ein Dummy, zu deutsch: Attrappe. Er hat nur formale Bedeutung. X kann beim späteren Aufruf der definierten Funktion durch jede andere Variable ersetzt werden. Dies ist der wichtigste Vorteil der definierten Funktion.
3. Es ist nicht zulässig, für die Definition von Funktionen Stringvariablen zu verwenden, also z. B. DEF FN A$(X). Dagegen können bei entsprechender Umrechnung Zeichenketten im Funktionsteil genutzt werden. Beispiel: DEF FN A(X) = VAL(CHR$(X)).
4. In Funktionsdefinitionen dürfen keine IF-THEN- oder FOR-NEXT-Routinen verwendet werden.
5. Um den Ablauf des Programms für eine bestimmte Zeit anzuhalten, kann die PAUSE-Funktion verwendet werden. Das Format lautet PAUSE(X).
   Die Zeitdauer der Unterbrechung beträgt X*0.1 s. Mit der PAUSE-Funktion kann man z. B. Ausschriften auf dem Bildschirm für eine definierte Zeit stehenlassen.
6. GOSUB hat Vorteile, wenn
   - im Unterprogramm IF-THEN- oder FOR-NEXT-Routinen verwendet werden sollen,
   - mehrere Zeilen oder
   - Stringoperationen verwendet werden sollen,

   DEF FN hat Vorteile, wenn:
   - die Zeit wichtig ist,
   - die Variablen in der Berechnung gewechselt werden sollen,
   - eine übersichtliche Darstellung von Funktionen gewünscht ist,
   - die Funktion für verschiedene Anwendungen leicht modifizierbar sein soll.

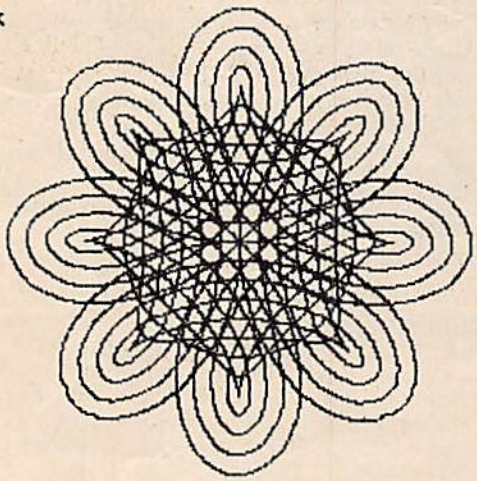
```
10 DIM A(2,11), B(2), A$(10): WINDOW 0,14,0,39: GOTO 90: ## HANOI ##
20 !------ ANZEIGE --------
30 CLS: PRINT TAB(8);"## TUERME VON HANOI ##"
40 PRINT: PRINT"Nr."Z: PRINT: Z=Z+1
50 FOR I=10 TO 1 STEP -1
60  PRINT A$(A(0,I)); A$(A(1,I)); A$(A(2,I))
70 NEXT: PAUSE(25): RETURN
80 !------- HAUPTROUTINE ------
90 A$(0)="     :      "
100 A$(1)="     H      "
110 A$(2)="    iHi     "
120 A$(3)="    HHH     "
130 A$(4)="   iHHHi    "
140 A$(5)="   HHHHH    "
150 A$(6)="  iHHHHHi   "
160 A$(7)="  HHHHHHH   "
170 A$(8)=" iHHHHHHHi  "
180 A$(9)=" HHHHHHHHH  "
190 A$(10)="iHHHHHHHHHi "
200 DEF FN F(X) = X+1 -INT((X+1)/3)*3
210 FOR I=1 TO 10 : A(0,I)=11-I: NEXT
220 A(0,0)=99: A(1,0)=99: A(2,0)=99: B(0)=10: GOSUB 30
230 A(C,B(C))=0: B(C)=B(C)-1: C=FN F(C): B(C)=B(C)+1: A(C,B(C))=1
240 GOSUB 30: D=FN F(C): E=FN F(D)
250 IF A(D,B(D))>A(E,B(E)) THEN A=D: B=E: ELSE A=E: B=D
260 Q=A(B,B(B)): A(B,B(B))=0: B(B)=B(B)-1: B(A)=B(A)+1: A(A,B(A))=Q
270 GOSUB 30: IF B(1)=10 OR B(2)=10 THEN END:ELSE 230
```

```
10 CLS: PRINT TAB(12);"## ZEIT ##": PRINT: Z=60
20 DEF FN A(X) = Z*X
30 DEF FN B(X) = INT(X/Z)
40 PRINT: PRINT"H = h, m, s oder S = s": PRINT: A$=""
50 A$=INKEY$: IF A$="" GOTO 50
60 IF A$="S" GOTO 90
70 INPUT"h, m, s =";H, M, S
80 PRINT FN A( FN A(H)+M) + S;"Sekunden": GOTO 40
90 INPUT"Sekunden =";S
100 M = FN B(S): S=S-Z*M: H =FN B(M): M=M-Z*H
110 PRINT H;"h";M;"m";S;"s": GOTO 40
```

```
10 WINDOW 0, 31, 0, 39: CLS: PRINT TAB(15);"## MUSTER ##": PRINT
20 KX=9: H=.5: U=125.5: V=U: BM=PI/90: BT=2*PI
30 INPUT "Wertepaar ="; A,B: C=KX*(A+B): D=(A+B)/B: CLS
40 DEF FN X(T)=INT(U+C*COS(T)-Q*COS(D*T))
50 DEF FN Y(T)=INT(V+C*SIN(T)-Q*SIN(D*T))
60  FOR N=-3 TO 3 STEP H: Q=N*B*KX
70   T=0:X1=FN X(T) :Y1=FN Y(T)
80    FOR T=BM TO BT STEP BM
90     X2=FN X(T): Y2=FN Y(T)
100    LINE X1, Y1, X2, Y2, 7: X1=X2: Y1=Y2
110   NEXT
120 NEXT
130 PRINT"A ="A :PRINT"B ="B: PRINT
140 ! Gute Beispielwerte:
150 !A:-6 -6 -8  4  4  6  4.5 ...
160 !B: 1  2  2  1  2  1  1.5 ...
```

```
A  =-8
B  = 2

OK
>
```



19. Hinweise zum Programmieren und Debugging

In diesem Abschnitt verfolgen wir das Ziel, Ihnen einige nützliche Hinweise für die Programmentwicklung und das Beseitigen von Fehlern zu geben. Eine umfassende Behandlung ist natürlich nicht möglich. Aber vielleicht sind die beiden Programme am Ende hilfreich.

19.1. Zur Programmentwicklung

Bei kleinen Programmen, wie wir sie bisher vorwiegend verwendet haben, existiert ausreichender Überblick. Aber meist wird man ja größere Programme schreiben, in denen viele Varianten und Möglichkeiten zu berücksichtigen sind. Hier halten wir es für wichtig, wenn zuerst das Hauptmenü oder gar alle Menüs erarbeitet werden.

Menüs sind jene Darstellungen, die uns auf dem Bildschirm die verschiedenen Möglichkeiten des Programms anzeigen und uns auffordern, aus den gebotenen Möglichkeiten zu wählen. Ein gutes Menü sollte dazu alles Wesentliche übersichtlich benennen und uns interaktiv zur Entscheidung auffordern. Es soll uns so durch das Programm führen, daß weitere Anleitung möglichst nicht nötig ist. Es muß also das Programm erklären. Wenn die Speicherkapazität ausreicht und eine gute Fenstertechnik bereitsteht, sollten wir es sehr übersichtlich und auch „ästhetisch" gestalten. Dies kann u. a. durch den Einsatz verschiedener Farben in den Fenstern erfolgen.

Wenn Sie Ihr Menü wohlüberlegt gestaltet haben, liegt auch die Grobstruktur Ihres Programms fest. Sie sollten dann, wie wir es schon im Kapitel GOSUB beschrieben haben, zunächst die Hauptroutine zum Laufen bringen, um dann nach und nach die Unterroutinen immer weiter zu verfeinern.

Die Erfahrung zeigt, daß es unproduktiv ist, ein großes Programm mit einem Mal eingeben zu wollen und dann nach Fehlern zu suchen. Dabei bestehen viel zu viele Fehlermöglichkeiten. Trösten Sie sich damit, daß es selbst den besten Programmierern fast nie gelingt, ein größeres Programm – ab etwa 30 Zeilen – auf Anhieb fehlerfrei zum Laufen zu bringen. Denken Sie auch daran: Fehler, die man nicht auf Anhieb findet – und das sind die meisten –, liegen gewöhnlich ganz woanders, als dort, wo man sie sucht. Sonst würde man sie ja sofort finden. Also suchen Sie dann immer in einem größeren Umkreis!

Alle genannten Fakten sind Gründe, um Programme gut zu strukturieren. Die einzelnen Teile sind dabei besser zu übersehen, einzeln einzugeben und zum Laufen zu bringen.

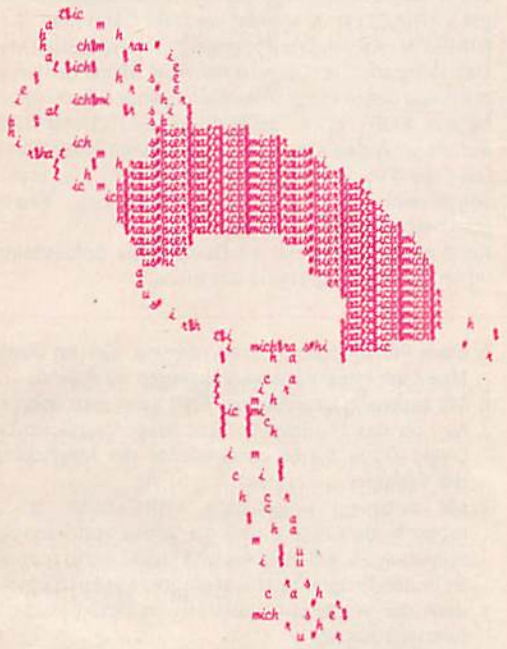
Schreiben Sie auch immer gleich reichlich Kommentare ins Programm. Man vergißt gar zu schnell, was eine Routine soll. Noch wichtiger ist es, Tricks zu vermerken. Notieren Sie sich getrennt, wenn Sie beim Programmieren eine interessante Lösung gefunden haben. So etwas sollte man später publizieren. Sie helfen sich und anderen damit.

Eine wenig bekannte und doch effektive Lösung in diesem Sinne ist es, die Teilroutinen mit RUN n anzusprin-

gen. Gegenüber GOTO oder GOSUB hat dies den Vorteil, daß dann alle Variablen Ø gesetzt werden. Weiter können Sie hierbei die Felder neu dimensionieren. Eine Variante könnte dabei wie folgt aussehen:

```
50 IF A=1 THEN RUN 200 ⟨ENTER⟩
60 IF A=2 THEN RUN 400 ⟨ENTER⟩
```

usw.
Für eine vereinfachte Programmeingabe bewährt sich der Befehl:



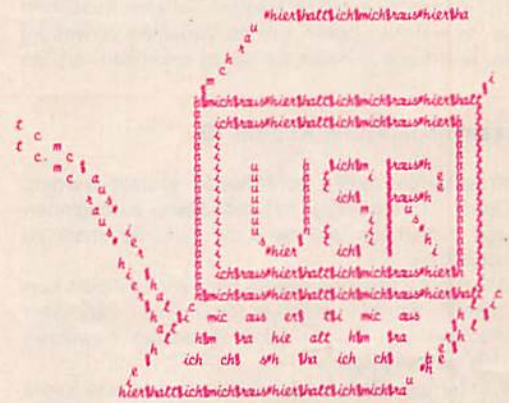
```
AUTO n, m ⟨ENTER⟩
```

Dabei verweist n auf die erste Zeilennummer und m auf den Abstand zwischen den Zeilennummern. Sie sparen sich so das Eingeben der Zeilennummern. Bezüglich der endgültigen Zeilennummern brauchen Sie sich dabei keine Sorgen zu machen. Sie können ja ständig mit

```
RENUMBER a, b, c, d ⟨ENTER⟩
```

verändert werden. Dabei steht a für die erste und b für die letzte Zeilennummer des umzunumerierenden Bereichs; c ist die neue erste Zeile und d der Abstand zwischen den Zeilen.
Wir empfehlen aber, darauf zu achten, daß durch eine solche Umnumerierung nicht die fortlaufende Zeilennumerierung von niederen zu höheren Zahlen durcheinandergebracht wird. Hier haben die KC-Rechner keinen Schutzmechanismus. Die Folge könnte sein, daß Ihr Programm dann nicht mehr lauffähig und editierbar ist.
Damit kommen wir zu einer weiteren Empfehlung: Machen Sie etwa alle 1 bis 2 Stunden Abzüge auf Kassette. Sie könnten durch eine Fehlhandlung Ihr Programm zerstören, der Strom könnte ausfallen, der Rechner abstürzen, d. h. unregelmäßiges Verhalten zeigen, usw. In der Regel sind diese Fälle zwar sehr selten. Wer aber einmal eine mehrstündige Arbeit unrettbar verloren hat, der weiß, was solche Sicherheitskopien bedeuten. Wir verwenden bei der Arbeit regelmäßig Kurzkassetten und machen die Programmabzüge abwechselnd auf beiden Seiten. Dadurch ist immer die letzte und sogar die vorletzte Fassung vorhanden.

Weiter machen wir gleich zwei Abzüge hintereinander. Dadurch ersparen wir uns das VERIFY. Aus den zwei Auszügen ist mit extrem hoher Sicherheit – selbst bei relativ schlechten Kassetten – immer wieder das Programm herzustellen.
Zum Aufbewahren von Kassetten gibt es in der Literatur sehr unterschiedliche Meinungen. Wir halten folgende Lösung für ideal: Eine vollständig kommentierte Variante des Programms kommt ins Archiv. Sie wird getrennt auf speziellen Kassetten an sicherem Ort aufbewahrt. Die Arbeitsvariante ist frei von allen Kommentaren. Sie ist also kürzer und läuft schneller. Falls Sie einmal versehentlich zerstört wird, kann sie immer wieder aus der Sicherheitskopie hergestellt werden.
Der Einfluß von Magnetfeldern ist übrigens weitaus geringer, als meist angenommen wird. Störend können unter Umständen nur starke Streufelder von Netztransformatoren und bei alten Farbfernsehgeräten die Entmagnetisierung der Maske beim Einschalten sein. Dauermagnete sollten natürlich nicht auf die Kassetten gelegt werden. In der Regel reicht überall ein Abstand von 20 cm aus.



19.2. Zur Fehlerbeseitigung

Fehler heißen im Englischen Wanzen, also bugs, und daher hat sich der Ausdruck debugging, für das Beseitigen von Fehlern eingebürgert. Ihm muß aber immer die Fehlersuche vorausgehen, und sie ist in der Regel schwieriger. Wenn der Fehler gefunden ist, läßt er sich meist relativ leicht beheben. Deshalb sind für die Fehlersuche auch viele Methoden entwickelt worden. Am einfachsten ist es noch, wenn das Programm mit irgendeinem Fehler abbricht. Dann weiß man wenigstens in welcher Zeile der Interpreter nicht mehr weiterwußte. Handelt es sich um einen Syntax-Fehler, so liegt er auch wirklich in dieser Zeile. Doch recht oft liegt er zeitlich davor. Wir haben bewußt „zeitlich davor" gesagt, denn häufig laufen die Programme ja nicht linear von Zeile zu Zeile, sondern haben vielfältige Verzweigungen. Besonders kompliziert wird dieses Rückverfolgen dann, wenn Sie Ihr Programm im GOTO-Salat-Stil geschrieben haben.
Nach dem ERROR ist es günstig, sich im Direkt-Mode die Werte wichtiger Variablen anzeigen zu lassen. Dadurch erhält man oft recht schnell nützliche Hinweise.
Zwei andere Methoden sind einmal das bewußte zeitweilige Einfügen von END ins Programm, um sich dann die Variablen anzeigen zu lassen. Zum anderen ist nützlich, PRINT-Zeilen zu ergänzen. Dadurch lassen sich beim Ablauf ausgewählte Variablen anzeigen. Auch Hinweise darauf, ob die gewünschte Teilroutine planmäßig durchlaufen wird, sind möglich.
Weiter kann das Programm an definierten Stellen von

Hand mit BREAK bzw. STOP unterbrochen werden, um sich danach im Direkt-Mode zusätzliche Werte anzeigen zu lassen. Dann kann der Ablauf mit CONT fortgesetzt werden.
Der Interpreter stellt weiterhin die beiden Befehle TRON und TROFF bereit. Hierbei werden die durchlaufenen Zeilennummern angezeigt. Der Befehl TRON (trace on = Weg an) schaltet diesen Modus an, und der Befehl TROFF (trace off – Weg zu) schaltet ihn wieder ab. Wir haben beide Befehle allerdings bisher kaum benutzt.

19.3. Zwei Hilfsprogramme

Das erste Programm heißt VARIABLEN. Es wird dem zu untersuchenden Programm angehängt. Das erfolgt einfach durch CLOAD„VARIABLEN". Es belegt die Zeilennummern 30100 bis 30137. Es wird also geladen, ohne vorher NEW einzugeben. Nach dem Dazuladen ruft man das Programm mit RUN 30100 auf. Dabei wird der Bildschirm gelöscht, und es erscheint der Programmname. Dann müssen Sie, je nach Länge Ihres Programms, einige Zeit warten. Schließlich erhalten Sie eine Ausschrift darüber, in welchen Zeilen welche Variablen verwendet werden. Hierdurch können Sie leicht erkennen, ob Sie alle Variablen wunschgemäß verwendet haben oder ob Sie womöglich die gleiche Variable in unterschiedlichen Teilroutinen so verwendet haben, daß ein Fehler auftreten muß. Außerdem können Sie nach dieser Methode leicht erkennen, in welcher Reihenfolge und wo Sie ihre Variablen initiieren.
Wenn Sie die Zeile 30103 ändern oder streichen, kann sich dieses Programm auch selbst dokumentieren. Dies sehen Sie am Ende.
Das zweite Programm heißt HILFE. Es wird genau wie das erste Programm nachgeladen. Da es die Zeilen 30000 bis 30049 beansprucht, kann es sogar gemeinsam mit VARIABLEN verwendet werden. Dann muß aber VARIABLEN als letztes Programm nachgeladen werden. Das Programm HILFE wird mit RUN 30000 gestartet und zieht aus Ihrem Programm die folgenden Befehle je Zeile heraus: FOR, NEXT, THEN, ELSE, GOTO und GOSUB. Auf diese Weise kann man sich schnell einen Überblick über die Programmstruktur verschaffen. Ja, man kann sogar relativ leicht eine Überprüfung bzgl. eines Programmablaufplans gewinnen.
Auch hierzu finden Sie als Beispiel die Selbstdokumentation dieses Programms am Ende

Zusammenfassung Kapitel 19

1. Programme sollten schrittweise erstellt werden. Dabei ist es günstig, mit dem Menü zu beginnen und dann nach und nach die Teilprogramme zu schreiben.
2. Alle 1 bis 2 Stunden sollten Sicherheitskopien hergestellt werden. Kurzkassetten mit ständigem Wechsel von Vorder- und Rückseite bewähren sich hier besonders.
3. Von fertigen Programmen sollte immer eine Kopie mit vollständigen Kommentaren an einem getrennten Ort sicher aufbewahrt werden. Als Arbeitsversion ist eine Aufzeichnung sinnvoll, die frei von Kommentaren und Leerzeichen ist.
4. Es empfiehlt sich, stets zwei Aufzeichnungen eines Programms hintereinander abzulegen, weil dann in kritischen Fällen aus beiden wieder ein vollständiges Programm hergestellt werden kann. Bei dieser Methode kann man erfahrungsgemäß auch auf das VERIFY verzichten.
5. Beim Fehlerabbruch ist es nützlich, sich im Direkt-Mode wichtige Variablen anzeigen zu lassen.
6. Mit zeitweilig eingefügtem END kann man überprüfen, ob das Programm diese Stellen erreicht. Der Direkt-Mode bietet dann wieder die Möglichkeit, die Variablen zu testen.
7. Mit zeitweilig eingefügten PRINT-Zeilen ist es möglich, den Ablauf und die Werte von Variablen zu verfolgen. Mit BREAK und CONT kann man aktiv in den Programmablauf eingreifen. Auch ein Ändern der Werte von Variablen im Direkt-Mode ist hierbei zulässig.
8. Zur besseren Verfolgung des Programmablaufs können mit der Anweisung TRON die Zeilennummern in der Reihenfolge ihrer Abarbeitung angezeigt werden. Mit der Anweisung TROFF wird dieser Zustand beendet.
9. Mit Hilfsprogrammen kann man sich zusätzliche Möglichkeiten der Fehlersuche schaffen. Oft ist es notwendig, die Verwendung der Variablen und die Struktur des Programms zu überprüfen.

```
30100 CLEAR 5000: CLS: PRINT TAB(12);"## VARIABLEN ##": PRINT: DIM ZA$(100)
30101 FOR ZI=1024 TO DEEK(983)
30102 IF PEEK(ZI)=0 THEN ZI=ZI+4: ZC$=STR$(DEEK(ZI-1)): NEXT: GOTO 30125
30103 IF ZC$>" 029999" THEN 30125
30104 ZZ=PEEK(ZI): IF ZZ=34 THEN 30134:! "
30105 IF ZZ=131 THEN 30130:! DATA
30106 IF ZZ=142 OR ZZ=156 THEN 30127:! REM, !
30107 !--- Variablensuche ---
30108 IF ZZ=32 THEN NEXT: GOTO 30125
30109 IF ZZ<65 OR ZZ>90 THEN NEXT: GOTO 30125:! keine Grossbuchstaben
30110 ZB$=CHR$(ZZ)
30111 !--- gefunden ---
30112 ZI=ZI+1: ZZ=PEEK(ZI)
30113 IF ZZ=40 THEN ZB$=ZB$+"(": GOTO 30118
30114 IF ZZ=36 THEN ZB$=ZB$+"$":GOTO 30112
30115 IF (ZZ>64ANDZZ<91)OR(ZZ>47ANDZZ<58) THEN ZB$=ZB$+CHR$(ZZ): GOTO 30112
30116 ZI=ZI-1: IF ZB$="" THEN NEXT: GOTO 30125
30117 !--- Einordnen ---
30118 ZB$=ZB$+":": ZL=LEN(ZB$)
30119 FOR ZJ=0 TO ZB-1: ZA$=ZA$(ZJ)
30120 IF LEFT$(ZA$,ZL)<>ZB$ THEN NEXT: ZA$(ZB)=ZB$+ZC$: ZB=ZB+1: GOTO 30123
30121 IF RIGHT$(ZA$,LEN(ZC$))<>ZC$ THEN ZA$(ZJ)=ZA$+ZC$
```

```
30122 ZJ=ZB: NEXT
30123 ZB$="": NEXT
30124 !--- Anzeige ---
30125 FOR ZI=0 TO ZB: PRINT ZA$(ZI): NEXT: END
30126 !--- REM ---
30127 FOR ZJ=0 TO 1 STEP 0: ZI=ZI+1:IF PEEK(ZI)=0 THEN ZJ=1
30128 NEXT: ZI=ZI-1: NEXT: GOTO 30125
30129 !--- DATA ---
30130 FOR ZJ=0 TO 1 STEP 0: ZI=ZI+1: ZA=PEEK(ZI)
30131 IF ZA=0 OR ZA=58 THEN ZJ=1
30132 NEXT: ZI=ZI-1: NEXT: GOTO 30125
30133 !--- String ---
30134 FOR ZJ=0 TO 1 STEP 0: ZI=ZI+1: ZA=PEEK(ZI)
30135 IF ZA=0 OR ZA=58 THEN ZJ=1: NEXT: ZI=ZI-1: NEXT: GOTO 30125
30136 IF ZA=34 THEN ZJ=1
30137 NEXT: NEXT: GOTO 30125

          ## VARIABLEN ##
ZA$(: 30100 30119 30120 30121 30125
ZI: 30101 30102 30104 30112 30116 30125 30127 30128 30130 30132 30134 30135
ZC$: 30102 30120 30121
ZZ: 30104 30105 30106 30108 30109 30110 30112 30113 30114 30115
ZB$: 30110 30113 30114 30115 30116 30118 30120 30123
ZL: 30118 30120
ZJ: 30119 30121 30122 30127 30130 30131 30134 30135 30136
ZB: 30119 30120 30122 30125
ZA$: 30119 30120 30121
ZA: 30130 30131 30134 30135 30136


30000 CLS: PRINT TAB(12);"## HILFE ##": PRINT: ZM=0: GOTO 30038
30001 !---- Zeilenende erreicht ----
30002 ZM=0: ZI=ZI+2: IF ZA$="" THEN NEXT: END
30003 PRINT ZC;ZA$: NEXT: RETURN
30004 !--- FOR ---
30005 GOSUB 30021: ZA$=ZA$+" FOR "+ZB$: NEXT: END
30006 !--- NEXT ---
30007 ZA$=ZA$+" NEXT ": GOSUB 30021
30008 IF ZB$>"" THEN ZA$=ZA$+ZB$
30009 NEXT: END
30010 !--- GOTO ---
30011 GOSUB 30033: ZA$=ZA$+" GOTO "+ZB$: NEXT: RETURN
30012 !--- GOSUB ---
30013 GOSUB 30033: ZA$=ZA$+" GOSUB "+ZB$: NEXT: RETURN
30014 !--- THEN ---
30015 GOSUB 30033: IF ZF=0 THEN NEXT: END
30016 ZA$=ZA$+" THEN "+ZB$: NEXT: END
30017 !--- ELSE ---
30018 GOSUB 30033: IF ZF=0 THEN NEXT: END
30019 ZA$=ZA$+" ELSE "+ZB$: NEXT: END
30020 !--- Variablensuche ---
30021 ZI=ZI+1: ZA=PEEK(ZI): ZB$="": IF ZA=0 THEN ZI=ZI-1: RETURN
30022 IF ZA=32 GOTO 30021
30023 IF ZA<65 OR ZA>90 THEN RETURN
30024 ZB$=ZB$+CHR$(ZA)
30025 ZI=ZI+1: ZA=PEEK(ZI)
30026 IF ZA=40 THEN ZB$=ZB$+"(": RETURN
30027 IF ZA=36 THEN ZB$=ZB$+"$": GOTO 30025
30028 IF(ZA>64ANDZA<91)OR(ZA>47ANDZA<58)THEN ZB$=ZB$+CHR$(ZA):GOTO 30025
30029 ZI=ZI-1: RETURN
30030 !--- Zeilennummer holen ---
30031 ZC=DEEK(ZI): ZA$="": ZI=ZI+2: ZM=1: RETURN
30032 !--- Zahlensuche ----
30033 ZB$="": ZF=0: FOR ZJ=0 TO 1 STEP 0:  ZI=ZI+1: ZA=PEEK(ZI)
30034 IF ZA=32 THEN NEXT
30035 IF ZA<48 OR ZA>57 THEN ZJ=1: ZI=ZI-1: NEXT: RETURN
30036 ZF=1: ZB$ = ZB$ + CHR$(ZA): NEXT:ZI=ZI-1: RETURN
30037 !--- Main ---
30038 FOR ZI=1027 TO DEEK(983)
30039  IF ZM=0 THEN GOSUB 30031:! Neue Zeile
```

```
30040  IF ZC=30000 THEN END
30041  ZZ= PEEK(ZI): ZM=1
30042  IF ZZ=0 THEN 30002:! Zeilenende
30043  IF ZZ=129 THEN 30005:! FOR
30044  IF ZZ=130 THEN 30007:! NEXT
30045  IF ZZ=136 THEN 30011:! GOTO
30046  IF ZZ=140 THEN 30013:! GOSUB
30047  IF ZZ=169 THEN 30015:! THEN
30048  IF ZZ=212 THEN 30018:! ELSE
30049 NEXT

          ## HILFE ##

30000  GOTO 30038
30002  NEXT
30003  NEXT
30005  GOSUB 30021 NEXT
30007  GOSUB 30021
30009  NEXT
30011  GOSUB 30033 NEXT
30013  GOSUB 30033 NEXT
30015  GOSUB 30033 NEXT
30016  NEXT
30018  GOSUB 30033 NEXT
30019  NEXT
30022  GOTO 30021
30027  GOTO 30025
30028  GOTO 30025
30033  FOR ZJ
30034  NEXT
30035  NEXT
30036  NEXT
30038  FOR ZI
30039  GOSUB 30031
30042  THEN 30002
30043  THEN 30005
30044  THEN 30007
30045  THEN 30011
30046  THEN 30013
30047  THEN 30015
30048  THEN 30018
30049  NEXT
```

20. Programmoptimierung, Grafik und Musik

Mit diesem letzten Teil wollen wir drei Gebiete streifen, die Hinweise auf weitere Möglichkeiten geben. Gewiß wäre noch viel zu behandeln. Aber einmal muß ja ein Endpunkt gesetzt werden.

20.1. Programmoptimierung

Von Optimierung wird immer dann gesprochen, wenn verschiedene, sich z. T. widersprechende Bedingungen existieren. Bei der Programmierung sind dies die folgenden drei:

- minimaler Speicherplatz für das Programm und den Ablauf des Programms
- schnellstmögliche Abarbeitung des Programms
- bestmögliche Dokumentation.

Diese Bedingungen gelten generell und nicht nur für BASIC.

20.2. Kommentare

Bei BASIC-Programmen wird eine gute Selbstdokumentation u. a. durch Einfügen von Kommentaren erreicht. Bereits hieraus ersieht man, daß Punkt 1 und 3 sich generell widersprechen. Deshalb haben wir ja bereits im letzten Abschnitt darauf verwiesen, daß es nützlich ist, immer zwei Programmversionen zu besitzen:

- eine für das Archiv, welche sehr gut kommentiert ist und stets an sicherem Ort aufbewahrt wird;
- eine Arbeitskopie, in der alle REM, Space usw. fehlen. Dies läßt sich sogar noch weiter treiben. So existiert ein Maschinenprogramm KOMPAKTOR, das auch noch alle überflüssigen Zeilennummern entfernt. Auf diese Weise erzeugt es eine hochdichte Arbeitsversion. Sie ist dann allerdings nicht mehr editierbar. Dieses Programm stammt von Dr. Boltze, Halle, und dürfte demnächst über den Robotron-Software-Vertrieb erhältlich sein.

Aus diesen Fakten ist ersichtlich, daß es immer nützlich ist, eine gut kommentierte Programmversion als Ausgangspunkt zu verwenden. Dazu, wie ein Programm gut kommentiert wird, haben wir mehrfach Stellung genommen. Eine Übersicht finden Sie am Ende dieses Abschnittes. Die Kommentare sind also nur ein Teil eines gut lesbaren Programms.

20.3. Speicherplatzminimierung

Auch zur Minimierung des Speicherverbrauchs sind in den einzelnen Abschnitten vielfältige Hinweise gegeben worden. Sie können hier nicht wiederholt werden. Aber allgemeine Gesichtspunkte bedürfen der Behandlung:
Zunächst sei eine Dreiteilung bzgl. des Speicherplatzes eingeführt:

- Jedes Programm beansprucht zu seiner Ablage selbst Speicherplatz. Hierauf haben u. a. die Mehrfachbefehle je Zeile, Unterprogramme, Leertasten, Weglassen von LET, Verwenden von DEF FN usw. Einfluß. Auch diese Fakten erfaßt weitgehend der o. g. KOMPAKTOR. Er ändert aber nicht die Programmstruktur. Hierzu müssen Sie selbst schon das Nötige tun.
- Beim Ablauf des Programms wird Speicherplatz für die Variablen, Arrays und Strings benötigt. Dieser Speicherplatz wird sofort beim Start des Programms reserviert und steht dann zur Verfügung. Er wird oft als statischer Speicherplatz bezeichnet. Daraus folgt, daß noch ein dritter existiert.
- Der dynamische Speicherplatz betrifft den Stack. Hier werden die aktiven FOR-NEXT, GOSUB, Klammern, DEF, FN, usw. verwaltet. Sein Bereich wird um so größer, je tiefer verschachtelt Sie ihr Programm gestalten

20.4. Geschwindigkeit

Damit ein Programm schnell abgearbeitet wird, das Ergebnis also kurzfristig zur Verfügung steht, müssen andere Bedingungen eingehalten werden. Da dies gerade für die Mikrorechner besonders wichtig ist, wurde bei allen Gelegenheiten darauf eingegangen. Deshalb haben wir auch Wert darauf gelegt, Ihnen Methoden zur Messung von Zeitabläufen zu demonstrieren. Aus diesem Grunde dürfte die Zusammenfassung am Ende auch hier ausreichend sein.

20.5. Einiges zur Grafik

Man kann in etwa 5 Grafikverfahren bei Rechnern unterscheiden. Das ist so gemeint, daß mit diesen Verfahren Bilder, also keine Texte, erzeugt werden sollen:

1. Mit Schriftzeichen kann man auch Bilder erzeugen, indem man Leerzeichen und Textzeichen zur Gestaltung verwendet. Diese Methode hat Ruth Völz zu einer gewissen Perfektion gebracht. Die „Befehlssatiren" – CAD/CAM im Tierreich – nutzen dieses Prinzip.

2. Viele Rechner, wie der KC 85/1 bzw. KC 87, besitzen spezielle Grafiksymbole, mit denen eine detailliertere Gestaltung erreichbar ist. Hier ist es erforderlich, die Bilder mit Geschick aus diesen Symbolen zu konstruieren.
3. Andere Rechner besitzen eine sog. Pseudografik, bei der es möglich ist, die Zeichen in einer Matrix von meist 8*8 Punkten – auch Pixel genannt – frei zu gestalten. Dadurch kann in einem begrenzten Rahmen die volle Auflösung des Rechners genutzt werden.
4. Bei der sog. Pixelgrafik werden alle Punkte einzeln definiert angesprochen.
5. Vektorgrafik existiert nur bei speziellen Rechnern, bei denen ähnlich wie beim Oszillografen der Lichtpunkt direkt bewegt wird. Sie können also nicht auf Fernseher übertragen werden.

Der KC 85/2, 3 verfügt über die Möglichkeiten der Pseudografik und Pixelgrafik sowie begrenzt über die Möglichkeit, mit Schriftzeichen Bilder zu erzeugen.
Nicht erwähnt ist bei dieser Aufzählung die Farbtüchtigkeit. Sie ist ein zusätzlicher Parameter, der die Methoden ergänzt.
Von hochauflösender Grafik spricht man, wenn mindestens 200*200 Pixel direkt ansteuerbar sind.
Mit dem Programm PASCAL bieten wir Ihnen eine spezielle Lösung zur Textgrafik an. Sie ist außerdem so geschrieben, daß vom Programm automatisch getestet wird, welcher Rechnertyp vorliegt. Dies erfolgt durch Abfragen des Speicherplatzes -5=FFFB.
Danach wird das maximal mögliche Fenster gewählt. Beim KC 85/2 und 3 wird das Steuerbyte B7A2=14242 für VPEEK so gesetzt, daß auch die Grafikzeichen der Kontrollcodes genutzt werden können. Erst dann folgt die eigentliche PASCAL-Routine:
Wahrscheinlich kennen Sie das Pascalsche Dreieck. Es wird um 45° gedreht, so daß seine Spitze der linke obere Zeilenanfang ist. Weiter werden die Zahlenwerte modulo einer einzugebenden Zahl umgerechnet (s. Abschn. 18.3.). Der sich dabei ergebende Wert wählt das spezielle Grafikzeichen aus. Diese wurden aus dem möglichen Zeichensatz beider Rechner speziell ausgewählt. Die DATA-Zeile 170 gilt für den KC 85/2, 3 und 180 für den KC 85/1 und KC 87.
Unseres Erachtens entstehen so recht hübsche Strukturen, und Sie können durch die Wahl des modulo-Wertes von etwa 2 bis 60000 vielfältige Muster erzeugen.
Das Programm KREISE verwendet den Grafikbefehl CIRCLE des KC 85/2, 3. Wir haben bewußt ein Programm ausgewählt, das nur zwei Zeilen benötigt und dennoch ein interessantes Bild erzeugt. Durch Änderung der Parameter haben Sie die Möglichkeit, vieles zu erproben.
Mit dem Befehl LINE Xa, Ya, Xe, Ye, FARBE, welcher Linien zwischen den Anfangspunkten a und den Endpunkten e erzeugt, arbeitet das Programm N-ECK. Es erzeugt hübsche Muster, die im wesentlichen aus hierarchisch verschachtelten N-Ecken bestehen.

20.6. Musik

Das letzte Programm nutzt die beiden Tonausgänge des KC 85/2, 3 zweistimmig für das italienische Lied „Santa Lucia". Es verwendet dazu den Befehl SOUND, desen Struktur Sie bitte im Handbuch des Rechners nachschlagen wollen.

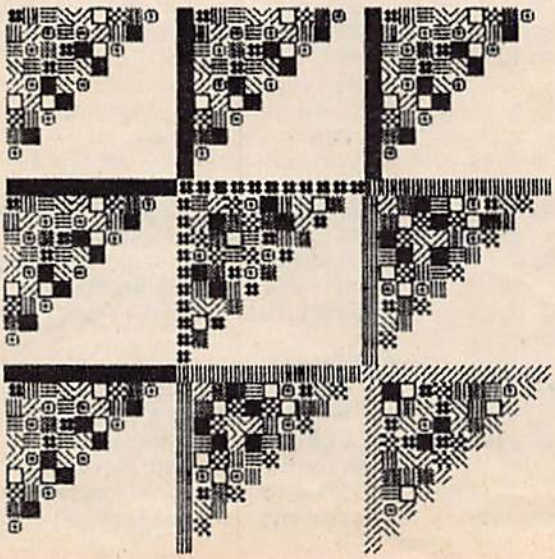
Zusammenfassung Kapitel 20

1. Die Optimierung eines Programms betrifft drei, z. T. widersprüchliche Punkte:
   - minimaler Speicherplatz des Programms
   - schnellstmögliche Abarbeitung
   - bestmögliche Selbstdokumentation.
2. Es sollten von fertigen Programmen immer zwei Versionen hergestellt werden:
   - eine sehr gut dokumentierte Archivfassung
   - eine Arbeitsversion, aus der weitgehend REM, Space usw. entfernt sind.
3. Zu einer guten Dokumentation gehören u. a. folgende Fakten:
   - Name des Programms festlegen, evtl. erklären
   - Aufgabe des Programms darlegen
   - ausweisen, was Unterprogramme bewirken
   - optische Gliederung des Programms durch Leerzeilen oder Trennzeilen, die nur mit einem ! beginnen
   - zusammengehörende FOR und NEXT gleich weit einrücken und das Schleifeninnere nochmals um ein Leerzeichen einrücken
   - die Belegung wichtiger Variablen und Zahlenwerte zusätzlich kennzeichnen.
4. Bezüglich des Speicherverbrauchs sind zu unterscheiden:
   - Länge des reinen Programms
   - Raum für Variablen, Felder und Zeichenketten
   - dynamischer Stack-Bereich infolge aktiver FOR-NEXT, GOSUB, Klammern, DEF FN und Zwischenergebnisse in Rechnungen.
5. Bezüglich der Ablaufgeschwindigkeit sind u. a. folgende Punkte wesentlich:
   - GOTO vermeiden
   - GOSUB und unvermeidbare GOTO immer zu möglichst niedrigen Zeilennummern führen (Unterprogramme vorn im Progamm anlegen)
   - DEF FN statt GOSUB verwenden, wenn möglich
   - Rechnungen vereinfachen
   - Konvertierungszeiten vermeiden, d. h., Konstanten in Variablen ablegen.
6. Die Grafik des KC 85/2, 3 läßt eine Ansteuerung von 320 (in der Waagerechten) mal 256 (in der Senkrechten) Punkten zu. Mit dem Befehl PSET X,Y,Farbcode wird ein Punkt (Pixel) mit den entsprechenden X- und Y-Koordinaten und der gewählten Farbe gesetzt. Durch den Befehl PRESET X,Y kann der entsprechende Punkt gelöscht werden.
7. Kreise können mit großer Geschwindigkeit durch den Befehl CIRCLE X,Y,R,Farbcode erzeugt werden. Hierbei bestimmen X und Y den Mittelpunkt und R den Radius des Kreises.
8. Linien können mittels LINE Xa,Ya,Xe,Ye,Farbcode gezogen werden. Dabei bedeuten Xa, Ya den Anfangspukt und Xe,Ye den Endpunkt der Strecke.

```
10 CLS:PRINT: PRINT TAB(10);"## LUCIA ##"
20 READ A,B,C,D: SOUND A,0,B,0,C,D: I=I+1: IF I=51 THEN RESTORE 110
30 IF I<>72 THEN PAUSE(2): GOTO 20
40 RESTORE 180: FOR I=1 TO 5: READ A, B, C: SOUND A,0,B,0,28,C: NEXT
50 DATA 171,144,15,20,171,144,15,30,128,108,15,10,128,108,15,10
60 DATA 144,114,15,10,144,114,15,40,192,162,15,20,192,162,15,30
70 DATA 162,128,15,10,162,128,15,10,171,144,15,10,171,144,15,40
80 DATA 171,0,15,20,128,0,15,30,144,0,15,10,144,0,15,10
90 DATA 153,0,15,10,162,0,15,40,162,0,15,20,171,0,15,20
100 DATA 192,0,15,20,162,128,15,20,171,144,15,40,0,0,0,10
110 DATA 108,86,28,20,114,96,28,20,128,108,28,20,114,0,28,10
120 DATA 128,0,28,10,162,96,28,40,114,96,24,20,128,108,24,20
130 DATA 162,128,24,20,153,0,24,10,144,0,24,10,171,108,24,40
140 DATA 86,0,15,10,108,0,15,10,108,0,15,10,144,0,15,10
150 DATA 144,0,15,10,171,0,15,10,162,0,15,10,96,0,15,10,162,96,15,40
160 DATA 162,96,15,20,162,128,15,30,144,114,15,10,162,96,15,20
170 DATA 171,108,15,40,0,0,0,20
180 DATA 162,96,20,144,86,30,162,96,10,162,96,20,171,108,40
```

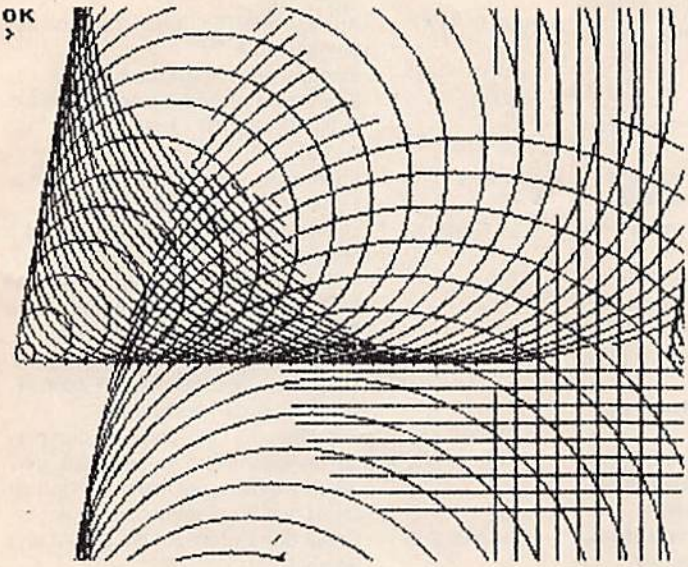
```
10 Z=PEEK(-5):IF Z<130 THEN Y=31: ELSE Y= 23
20 WINDOW 0,Y,0,39: CLS: PRINT TAB(10)"## PASCAL ##": PRINT
30 INPUT"M =";M: DIM A(32), B(32), C(11): T=14242:CLS
40 IF Y=31 THEN VPOKE T,(VPEEK(T) OR 8): ELSE RESTORE 180
50 FOR I=0 TO 11: READ C(I): NEXT
60 FOR I=0 TO Y+1: A(I)=1: NEXT: B(0)=1
70  FOR Z=1 TO Y+1
80   FOR I=1 TO Y+1
90    B(I)=B(I-1)+A(I)
100   IF B(I)=>M THEN B(I)=B(I)-M
110   A$=CHR$(C(B(I)-12*INT(B(I)/12)))
120   IF Y=31 THEN PRINT AT(Z-1,I+6);A$;: ELSE PRINT AT(Z-1,I+6);A$
130   A(I)=B(I)
140  NEXT
150 NEXT: IF Y=31 THEN VPOKE T,(VPEEK(T) AND 247)
160 PRINT "PASCAL": PRINT "MODULO": PRINT M
170 DATA 32,91,35,5,6,21,23,127,27,14,20,4
180 DATA 32,140,35,175,198,197,185,127,166,201,184,138
```

```
PASCAL
MODULO
 11
OK
>
```



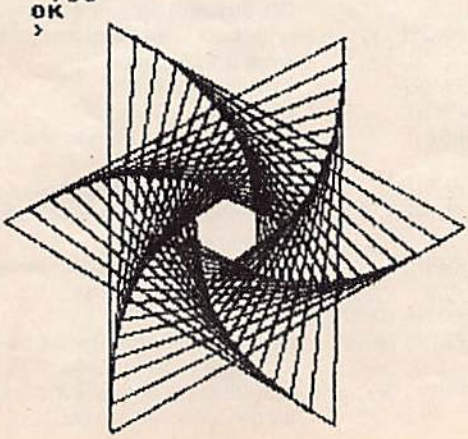
```
10 WINDOW 0,31,0,39: CLS: B=1.2:! ## KREISE ##"
20 FOR X=5 TO 350 STEP8: CIRCLEX*B+5,X+90,X,7: NEXT
```

```
OK
>
```

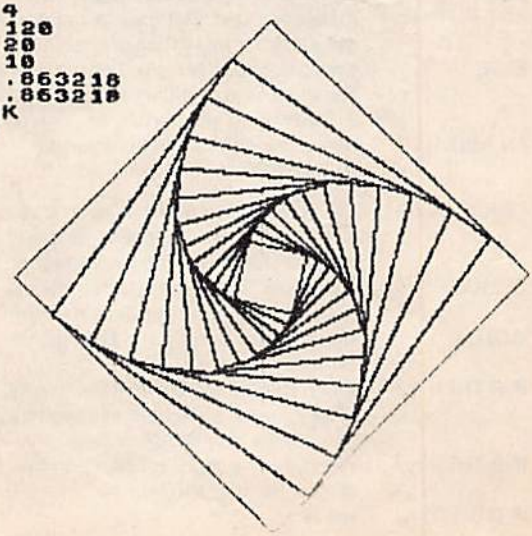


```
10 WINDOW 0,31,0,39: CLS: PRINT TAB(12);"## N-ECK ##": PRINT
20 INPUT "Ecken-Anzahl="; N: Q=2*PI/N: IF N<3 OR N>12 GOTO 20
30 DIM P(5), X(N), Y(N): P(0)=N
40 INPUT"Startlaenge ="; P(1)
50 INPUT"Endlaenge   ="; P(2): IF P(2)>=P(1) GOTO 50
60 INPUT"Drehwinkel  ="; P(3): D=P(3)*PI/180: IF D>Q/2 GOTO 60
70 S=SIN(D): C=COS(D): A=COS(Q): B=SIN(Q)
80 P(4)=B/(C*B+S-A*S): X(1)=A: Y(1)=B
90 PRINT"Idealer Wert=";P(4)
100 INPUT"aendern 0/1 ="; R: P(5)=P(4)
110 IF R=1 THEN INPUT"gewuenschter="; P(5)
120 CLS: FOR I=1 TO N-1: A=I*Q: X(I)=COS(A): Y(I)=SIN(A): NEXT
130 X(0)=1: Y(0)=0: X(N)=1: Y(N)=0: R=P(1)
140 X1=R*X(0)+190: Y1=R*Y(0)+130
150 FOR I=1 TO N
160  X2=R*X(I)+190: Y2=R*Y(I)+130: LINE X1,Y1,X2,Y2,7: X1=X2: Y1=Y2
170 NEXT
180 FOR I=0 TO N
190  A=X(I)*C-Y(I)*S: Y(I)=Y(I)*C+X(I)*S: X(I)=A
200 NEXT: R=R*P(5): IF R>P(2) GOTO 140
210 FOR I=0 TO 5: PRINT P(I): NEXT
```

```
$+BR^,    3
          110
          25
          50
          .500305
          .95
         OK
         >
```



```
          4
          120
          20
          10
          .863218
          .863218
         OK
         >
```



Anhang

Im folgenden sind nur jene Fakten zusammengestellt, die im Lehrgang benötigt werden. Vollständige Angaben finden Sie in den einschlägigen Handbüchern.

Anweisungen, Befehle, Funktionen

ABS(X)	Übergibt den Wert von X mit stets positivem Vorzeichen
A AND B	a) Wenn A und B logische Variablen sind, dann wird das Ergebnis wahr (−1), wenn A und B gleichzeitig wahr sind. b) Wenn A und B Zahlenwerte sind, dann wird das Ergebnis über die einzelnen Bit der entsprechenden 16-Bit-Binärzahl (Bit 15 = Vorzeichen) gebildet.
ASC(A$)	Erzeugt den ASCII-Code des ersten Zeichen von A$
AUTO n, m	Erzeugt im Eingabemode automatisch Zeilennummern, die bei n beginnen und den Abstand m besitzen.
BREAK	Unterbrechen des Programmablaufes
CHR$(X)	Gibt das ASCII-Zeichen mit dem Code X aus.
CIRCLE X, Y, R, Farbe	Erzeugt einen Kreis mit dem Mittelpunkt X, Y und dem Radius R.
CLEAR n, m	Setzt alle Variablen gleich Null und verändert den Speicherraum. Es werden n Byte für String reserviert und m legt das Ende des RAM fest.
CLOAD "Name"	lädt Programm mit Namen von Kassette
CLS	Löscht Bildschirm
CONT	Fortsetzen eines unterbrochenen Programms
CSAVE "Name"	Rettet Programm mit Namen auf Kassette
DATA	Macht alle folgenden Zahlen bzw. String zu Werten, die nur mittels READ zugänglich sind.
DEF FN AB(X1)= Ausdruck	Definiert eine Funktion, die über FN AB(X1) aufrufbar ist. AB ist die Bezeichnung der Funktion und X1 ist ein frei wählbarer Parameter. X1 ist also ein Dummy
DIM	Dimensioniert die dann folgenden numerischen und Stringfelder mit den in der Klammer folgenden Werten.
ELSE	Kennzeichnung für eine Teilroutine, die immer dann ausgeführt wird, wenn die IF-Bedingung nicht erfüllt ist.
FN AB(X1)	Aufruf der definierten Funktion AB. (s. DEF FN)
FOR X=A TO B	Die Variable X wird von A nach B variiert. A und B werden sofort gespeichert und sind danach frei verfügbar.
GOSUB n	Aufruf der Subroutine (Unterprogramm), welche in der Zeile n beginnt.
GOTO n	Unbedingter Sprung zur Zeile mit der Nummer n.
IF B THEN A	Wenn Bedingung B erfüllt, dann realisiere Anweisung A (und evtl. weitere), sonst gehe zur nächsten Zeile.
IF B THEN n	Wenn Bedingung B erfüllt, dann bedingter Sprung zur Zeile n.
IF B GOTO n	wie oben
IF B THEN GOTO n	wie oben
INPUT A	Eingegebene Zahl wird der Variablen A übergeben.
INPUT „TEXT";A	Anzeige von TEXT und Übergabe der Eingabe an A.
INPUT "";A	Eingabe ohne Fragezeichen an A.
INT(X)	Es wird die nächstkleinere ganze Zahl gebildet, also INT (3.5)=3; INT (−3.5)=−4.
INKEY$	übernimmt, wenn vorhanden, ein einzelnes Zeichen von der Tastatur
LEFT$(A$,X)	Liefert die ersten X Zeichen von A$ als neuen String.
LEN(A$)	Liefert die Anzahl der Zeichen von A$.
LET A=B	Setzt die Variable A auf den Wert von B. Wenn A noch nicht verwendet wurde, wird hierfür ein Speicherplatz organisiert (initialisiert). LET kann in vielen Dialekten entfallen.
LINE Xa, Ya, Xe, Ye, Farbe	Schreibt eine Linie von dem durch a gekennzeichneten Punkt bis zu dem durch e gekennzeichneten Endpunkt.
LIST	Listet den Programmspeicher auf.
LOCATE Z, S	Bringt den Cursor auf die Zeile Z und Spalte S.
MID$(A$,X,Y)	Liefert ab X-tem Zeichen die folgenden Y-Zeichen.
NEXT	Erhöht die Variable des letzten FOR. Testet, ob Endwert überschritten ist. Wenn das nicht der Fall ist, dann die Schleife ausführen. Sonst Schleife verlassen.
NEXT X	Erhöht Variable X. Wenn sie nicht zum letzten FOR gehört, dann werden die anderen Variablen zuvor vom Stack entfernt.
NEXT X, Y	Nur so wie NEXT X, jedoch erst für X, bis Maximalwert erreicht, dann für Y.
NEW	Löscht Programmspeicher.
NOT A	a) Umkehrung des Wahrheitswertes. b) a→−a−1: −a→a−1 (a=ABS(A)).
ON Y GOTO n_1, n_2,...,n_m	Wenn X=0 oder X=1, dann Sprung zur Zeile n1. Wenn X=2, dann Sprung zur Zeile n_2. ⋮ Wenn X=>M, dann Sprung zur Zeile n_m.
A OR B	Wie A AND B, jedoch mit dem Unterschied, daß: a) Nur wenn A und B falsch sind, ist auch das Ergebnis falsch, b) Die neue Zahl entsteht durch die OR-Bildung über beide 2-Byte-Zahlen.
PAUSE (X)	Unterbricht den Programmablauf für x-mal Ø.1 Sekunden.
PI	Zahlenwert 3.14159
PRINT	Erzeugt neue Zeile.
PRINT,	setzt Cursor auf die nächste 13. Position.
PRINT A	Zeigt A an und erzeugt neue Zeile.
PRINT A, B	Zeigt A und B mit dem Abstand von 13 Zeichen an.
PRINT A;B	Zeigt A und B unmittelbar nacheinander an.
PRINT "TEXT"	zeigt das Wort TEXT an.
PRINT TAB (X)	setzt nächste Ausgabe auf die Position X der aktuellen Zeile.
PRINT SPC(X);	setzt nächste Ausgabe X Zeichen hinter den aktuellen Cursor.

PRINT AT(Z,S);	Beginnt Ausgabe in Zeile Z auf der Position S. Beim KC 85/1 bzw. KC 87 muß nach der Klammer ein Komma stehen: PRINT AT (Z,S),.
PRESET X,Y	Es wird der Punkt X, Y gelöscht.
PSET X, Y, F	Setzt den Punkt X, Y mit der Farbe F. X und Y beginnen links unten. Es gilt: Ø<=X<=319 und Ø<=Y<=255.
PTEST(X)	Testet, ob auf dem vorher festgelegten Y der Punkt X gesetzt ist.
RANDOMIZE	Setzt den internen Zufallsgenerator auf einen zufälligen Startwert (aus Refresh-Register).
READ A	Liest entsprechend dem aktuellen Zeiger in DATA-Zeilen einen Wert und weist ihm der Variablen A zu. Hinter READ können auch String- und Array-Werte, ja sogar mehrere und gemischt, getrennt durch Komma, stehen.
REM	Alles nach diesem Befehl stehende wird vom Interpreter nicht zur Kenntnis genommen; ist folglich nur ein Kommentar zum Programm.
RENUMBER A, B, C, D	Ändert die Zeilennummern von A bis B in die Startnummer C und mit dem Abstand D. Es ist zu beachten, daß bei teilweiser Umnummerierung kein ineinander Verschachteln der Zeilen entsteht.
RESTORE	Rücksetzen des Zeigers auf den Programmanfang.
RESTORE n	Setzt den Zeiger für die READ-Anweisung auf den Beginn der n-ten Zeile des Programms.
RETURN	Kennzeichen für das Ende einer Subroutine.
RND(X)	Es werden Zufallszahlen zwischen Ø und 1 erzeugt, dabei gilt: X<Ø es beginnt eine neue Zufallsfolge, die durch den Wert von −X festgelegt ist. X=Ø es wird die letzte Zufallszahl wiederholt. X>Ø die begonnene Zufallsfolge wird mit einer neuen Zufallszahl fortgesetzt.
SGN(X)	Es wird das Vorzeichen von X ausgegeben: aus X<Ø folgt SGN(X) = −1 aus X=Ø folgt SGN(X) = Ø aus X>Ø folgt SGN(X) = 1
SOUND A, B, C, D, E, F	Tonausgabe auf zwei Kanälen
SQR(X)	Es wird die Quadratwurzel von X berechnet. Für X<0 erfolgt Fehleranzeige ?FC ERROR = falscher Funktionsaufruf.
STEP X	Befehl zur Festlegung der Schrittweite X
STOP	Unterbrechen des Programmablaufes beim KC 85/1 bzw. KC 87. Auch mit CTRL C möglich.
STR$(A)	Wandelt die numerische Variable A in einen String.
TROFF	Abschalten des Trace-Betriebes, bei dem jeder Zeilendurchlauf angezeigt wird.
TRON	Anschalten des Trace-Betriebes.
VAL (A$)	Wandelt A$, sofern möglich, in eine Zahl.
VGET$	Nimmt das Zeichen, auf das der Cursor im Video zeigt
WINDOW ZS, ZZ, SS, SZ, F	Definiert ein Fenster mit den Startwerten: ZS Zeile und SS Spalte sowie der Größe: ZZ Zeilen und SZ Spalten. F ist der Farbwert.

Fehlermeldungen

SN (syntax error)	Die Zeichenfolge kann nicht interpretiert werden.
UL (undefined line)	Die Zeile existiert nicht.
MO (missing operand)	Es fehlt ein zugehöriger Operand.
OV (overflow)	Die Zahl ist zu groß.
NF (next without for)	Für dieses NEXT existiert kein FOR.
REDO FROM START.	Mache die Eingabe noch einmal, sie war falsch.
OM (out of memory)	Platz für den Stack ist zu klein.
FC (function call)	unerlaubter Funktionsaufruf

Einzelzeichen

:	Trennt in einer Zeile zwei Befehle
*	Multiplikation
+	Addition (s. auch unten)
−	Subtraktion oder Vorzeichen bei Zahlen und Exponenten
/	Division
.	Dezimalpunkt
E	Exponent in Zahlen
!	Wie REM
?	Wie PRINT
^	Potenzieren, also x^y : x hoch y

Das Zeichen + kann in BASIC-Programmen folgende Bedeutungen besitzen:

1) Vorzeichen von Zahlen: z. B. +7
2) Vorzeichen im Exponenten, z. B. 1E+23
3) ASCII-Zeichen „+" mit der Code-Nr. 2BH bzw. 43
4) Additions-Zeichen, z. B. 3 + 4 = 7
5) Konnektions-Zeichen bei String

Begriffe

Array	s. Feld.
ASCII	Amerikanischer Code zur numerischen Behandlung von Steuerzeichen, Zahlen, Buchstaben und Satzzeichen.
Bedingung	Aussage, die wahr oder falsch sein kann. In BASIC z. B. X = 0, X < A oder Y$ = „JA".
Boole	Mathematiker, der wesentliche Grundlagen der Mathematischen Logik entwickelt hat.
Compiler	Übersetzer einer Programmiersprache, der nur ein vollständiges Programm zusammenfassend für den Rechner aufbereitet (Gegensatz Interpreter).
Cursor	Hilfszeichen auf dem Bildschirm, das die jeweilige Schreibposition symbolisiert.
Dialekt	Wie in der natürlichen Sprache existieren auch bei BASIC in verschiedenen Rechnern gewisse Unterschiede. Sie werden dadurch zum Ausdruck gebracht, daß man von Dialekten spricht.

Dummy	engl. Atrappe. Wert der nur formal dort steht und somit als „Lückenfüller" wirkt.
Dynamischer Speicherraum	Vom Programm benötigter Speicherraum für einwandfreie Stackverwaltung. Einfluß haben hierauf GOSUB, FOR-NEXT, Klammern und Zwischenergebnisse.
Feld	Zusammenfassung von Daten oder Strings zu einer Einheit, so daß sie über Indizierung aufgerufen werden können.
Formatieren	Ausgeben von Text und Zahlen an genau definierten Positionen.
Garbage-Collektion	= garbage connection = Müllbeseitigung Packt die Zeichenketten im Stringraum dicht zusammen. Während dieser Zeit verhält sich der Rechner so, als ob er defekt wäre.
Gleichverteilung	In einem Bereich von z. B. Ø bis 1 treten alle Zahlenwerte mit genau gleicher Wahrscheinlichkeit auf.
Grafik	Methode mit der man Bilder erzeugen kann.
Initialisieren	Es erfolgt für Variablen durch den ersten Aufruf im Programm. Hierdurch wird die Reihenfolge des Ablegens im Variablenspeicher bestimmt.
Interpreter	Programm zur Übersetzung der verschiedenen Anweisungen des BASIC. Ein Interpreter arbeitet zeilenweise. Zwischen dem Rechner und dem Menschen wirkt er ähnlich wie ein Simultandolmetscher.
Interaktiv	Methode bei Rechnern, mit der durch Fragen und Antworten auf dem Bildschirm eine unmittelbare Kommunikation möglich ist.
Konnektion	Zusammenfügen von zwei String zu einem neuen Beispiel: A$ = B$ + C$
Lineare Programmierung	Aufstellung eines Programms in der Weise, daß nur immer zu höheren Zeilenzahlen fortgeschritten wird. Solche Programme sind meist länger aber schneller.
Logik	Lehre von Aussagen, die nur falsch oder richtig sein können.
Menü	Bezeichnung einer Ausschrift, die eine interaktive Fortsetzung des Programmes erleichtert.
Mittelwert	Bei Zahlenfolgen der mittlere Wert. Hierfür gilt $M = \frac{1}{N} \sum_{i=1}^{N} X_i$; wobei X_i die einzelnen Werte, N ihre Anzahl bedeuten. Bei Gleichverteilung zwischen Ø und 1 beträgt M = Ø,5.
Mode	Betriebsart, in der sich der Rechner befindet: ● Der Direktmode liegt vor, wenn zu Beginn der Zeile keine Zahl steht. Hier werden alle Befehle sofort nach ENTER ausgeführt. ● Der Eingabemode wird erreicht, wenn zu Beginn der Eingabezeile eine Zahl steht. Der dann folgende Text wird gemäß dieser Zahl im Programmspeicher abgelegt. ● Der RUN-Mode wird mit RUN erreicht. Hier wird der Programmspeicher folgerichtig dem Interpreter übergeben. ● Der EDIT-Mode gestattet ein möglichst einfaches Ändern des Programms.
Modulo n	Spezielle Arithmetik, die nur die Zahlenwerte von Ø bis n−1 zuläßt. So gilt z. B.: 4 modulo 3 = 4–3 = 1; 11 modulo 3 = 11–3*3 = 2.
PAP = Programmablaufplan	Darstellung bezüglich der Teilaktivitäten eines Programms.
Pixel	Einzelne ansteuerbare Punkte in einer Rastergrafik.
Position	Stellung des Cursor auf dem Bildschirm. In der Regel in einer Zeile vom Anfang des Bildschirmes gezählt.
Rastergrafik	Grafik bei Rechnern, die über das Setzen einzelner Punkt (Pixel) realisiert wird.

Computerbücherkiste

Für alle, die zum Thema mehr lesen möchten, haben wir eine Liste der neuesten in der DDR erschienenen Literatur zusammengestellt, die in den Umgang mit dem Computer einführt. Ein Teil der mit dem Erscheinungsjahr 1987 ausgewiesenen Bücher wird erst im Laufe des 2. Halbjahres erscheinen. Sollte ein von Ihnen gewünschtes Buch schon vergriffen sein, denken Sie bitte auch an die nächste Bibliothek.

Bückner, U.: Kleincomputer leichtverständlich. Fachbuchverlag, Leipzig 1986

Claßen, L.; Oefler, U.: Wissensspeicher Mikrorechnerprogrammierung (2. Aufl.). Verlag Technik, Berlin 1986

Der Personalcomputer robotron 1715/R. Zeth u. a.; hrsg. i. A. d. VEB Kombinat Robotron von H. Lodahl. Verlag Die Wirtschaft, Berlin 1987

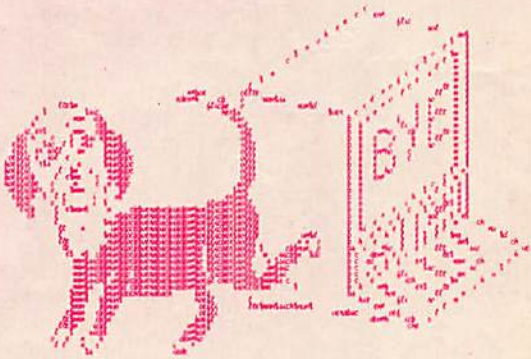
Evert, K.-P.; Hübler, B.: Ausbaufähiger Microcomputer für den Amateur (Amateurreihe electronica 227/228). Militärverlag, Berlin 1985

Fischer, P.: BASIC für Anfänger. Verlag Die Wirtschaft, Berlin 1987

Reserviertes Wort	String, der einer Anweisung entspricht, z. B. AT, OR, RND.
Sortieren	Methode bzw. Programm zur richtigen Ordnung von Zahlen und/oder Texten.
Sprung	Übergang des Interpreters zu einer neuen Zeile. Es gibt bedingte und unbedingte Sprünge. Bedingte Sprünge erfolgen nur, wenn eine Bedingung erfüllt ist.
Stack	Spezieller Speicher, der am besten durch aufeinander gelegte Zettel zu erklären ist. Der zuletzt aufgelegte muß erst entfernt werden, wenn auf einen vorhergehenden zugegriffen werden soll. Der Stack heißt auch Keller- oder Stapelspeicher bzw. last in first out.
Streuung	Maß für die Abweichungen von Zahlen von ihrem Mittelwert: Sie wird berechnet: $S = SQR(\sum X_i^2 - M*N* \sum X_i)/(N-1)$ S. hierzu auch Mittelwert. Bei Gleichverteilung zwischen Ø und 1 beträgt die Streuung S = SQR (1/9 − 1/4) = Ø.288675.
String	Zeichenkette = Eine feste Verbindung aus allen möglichen ASCII-Zeichen zu einer Einheit. Er wird u. a. durch "..." erzeugt, z. B. "Text".
Struktogramm:	Darstellung des Programmablauf in einer kästchenförmigen Art. Vgl. PAP.
Strukturierte Programmierung	Zusammenfassung von Prinzipien, welche zu einer effektiven und fehlerarmen Programmierung führen. Hierbei wird u. a. kein GOTO verwendet. Weiter wird eine schrittweise Programmentwicklung von oben nach unten, also zu den untergeordneten Teilroutinen hin empfohlen.
Token	Wert für reservierte Worte im BASIC-Programm. Alle Token, d. h. reservierten Worte, sind genau ein Byte, das größer als 7FH = 127D ist. Z. B. steht für GOTO das Token 88H.
Trace	engl. Spur. Weg, den ein Programm bezüglich der Zeilennummern durchläuft.
Variablen	Platzhalter (Speicherplätze) für Zahlenwerte. Sie beginnen mit (großem) Buchstaben und können durch meist einen Buchstaben bzw. eine Ziffer ergänzt werden. Mehr Zeichen sind möglich, werden aber intern nicht vom Rechner berücksichtigt. Zu beachten sind reservierte Worte.
Verzweigung	Möglichkeiten des Sprunges zu verschiedenen Zeilen auf Grund von Bedingungen.
Zeichenkette	String.

Tasten

ENTER	Schließt Eingabe ab. In anderen Geräten auch RETURN, CR, NEW LINE.
BRK	Unterbricht Programmablauf. In anderen Geräten auch BREAK, STOP, CTRL C.
HOME	Führt Cursor in die linke obere Ecke des Bildschirms
DEL = DELETE	Löscht das unter dem Cursor stehende Zeichen.
INS = INSERT	Fügt ein Leerzeichen an der Stelle, wo sich der Cursor befindet, ein.



Groh, J.: Kleincomputerfibel. Akademie-Verlag, Berlin 1986

Gutzer, H.: Das kann der Mikrocomputer (2. Aufl.). Urania-Verlag, Leipzig, Jena, Berlin 1986

Gutzer, H.: Spiel und Spaß mit dem Computer. Urania-Verlag, Leipzig, Jena, Berlin 1987

Gutzer, H.; Pauer, H.-D.: Wenn Kepler einen Computer gehabt hätte. Fachbuchverlag, Leipzig 1987

Heblik, P.: Wissensspeicher BASIC. Verlag Volk und Wissen, Berlin 1986

Hintze, W.: Mikrocomputer. Deutscher Verlag der Wissenschaften, Berlin 1987

Hopfer, R.: Mikrorechentechnik – allgemeinverständlich (2. Aufl.) Fachbuchverlag, Leipzig 1987

Hopfer, R.; Müller, R.: BASIC – Einführung in das Programmieren. Fachbuchverlag, Leipzig 1986

Kleinstrechner-TIPS (H. 1–6). hrsg. von H. Kreul u. a. Fachbuchverlag, Leipzig, 1984 ff.

Kramer, M.: Praktische Mikrocomputertechnik (Amateurbibliothek). Militärverlag, Berlin 1987

Müller, S.: Programmieren mit BASIC (Automatisierungstechnik 216). Verlag Technik, Berlin 1986

Paulin, G.: Kleines Lexikon der Mikrorechentechnik (Automatisierungstechnik 206; 4. Aufl.). Verlag Technik, Berlin 1986

Paulin, G.; Schiemangk, H.: Programmieren mit PASCAL (2. Aufl.). Akademie-Verlag, Berlin 1986

Posthoff, Ch.; Reinemann, G.: Computerschach – Schachcomputer. Akademie-Verlag, Berlin 1987

Prager, E.; Richter, E.: Software – was ist das? Verlag Die Wirtschaft, Berlin 1986

Werner, D.: BASIC für Mikrorechner. Verlag Technik, Berlin 1986

Die Auswahl traf für Sie
Dipl.-Ing. Heinz Waurick

Computer-unterricht in armenischen Landschulen

Im Innern eines mit Personalcomputern ausgerüsteten Busses findet der Unterricht im Lehrfach „Grundlagen der Informatik und Rechentechnik" statt. Die Schüler der Landschulen und der kleinen Stadtschulen in der Armenischen SSR eignen sich so die notwendigen Fertigkeiten im Umgang mit der EDV-Technik an.

„Die EDV-Technik dringt heute in alle Bereiche der menschlichen Tätigkeit ein", sagte der Leiter der Verwaltung Bildungseinrichtungen des Ministeriums für Volksbildung der UdSSR Waleri Rosow. „Daher muß die Schule den jungen Menschen die modernen Methoden der Organisation und Verarbeitung von Informationen beibringen und unsere Kinder sowie Kindeskinder auf die Lösung der verschiedensten Aufgaben unter Einsatz von Computern vorbereiten."

Heute werden die Grundlagen der Informatik und Rechentechnik in fast 60 000 allgemeinbildenden Schulen der UdSSR erlernt. In diesem Lehrfach werden hauptsächlich die Schüler der 9. bis 10. Klassen unterwiesen. In Armenien aber wird ein Experiment durchgeführt, das darauf abzielt, den Schülern im jüngeren Alter, u. a. bereits in der Unterstufe, den Umgang mit der EDV-Technik beizubringen. Die Leiter des Experiments haben eine Methodik ausgearbeitet, die es erlaubt, die Handhabung der EDV-Technik mit der Vertiefung der Fertigkeiten im Kopfrechen, mit der Entwicklung des sog. „Gefühls für Zahlen" sowie mit der Verbesserung der logischen Fähigkeiten der Schüler und ihres mathematischen Denkens zu verbinden.

Zu solchen Methoden gehört beispielsweise eine besondere Form der Lösung von Textaufgaben. Bei der traditionellen Methodik des Mathematikunterrichts wird ein beträchtlicher Teil der Stunde darauf verwendet, die Lösungen der Aufgaben aufzuschreiben (die Kinder schreiben ja in der Regel langsam) und Berechnungen anzustellen. Der Logik der Auflösung selbst wird nur wenig Zeit gewidmet. Mit Hilfe der EDV-Anlagen läßt sich diese Arbeit operativ und unter aktiver Beteiligung der ganzen Klasse ausführen. Der Lehrer diktiert langsam eine Aufgabe. Die Kinder schreiben

nicht mit, sondern hören aufmerksam zu und überlegen sich den Lösungsweg. Nach dem abermaligen Lesen lösen die Schüler am Computer oder mit dem Taschenrechner gleich die Aufgabe. Die Elektronenrechner ermöglichen außerdem, die Richtigkeit der Lösung rasch zu überprüfen. In der zweiten Hälfte der Stunde kommentiert ein Schüler seine Überlegungen. Bei einer solchen Form der Arbeit können die Schüler der Versuchsklassen in 15 bis 20 min acht bis zehn Aufgaben lösen.

Die Einführung des neuen Lehrfachs an den Schulen der Sowjetunion führte auch im Hochschulwesen zu einer Umstellung. An den Hochschulen des Landes wurden über 100 Fakultäten und Abteilungen eröffnet, deren Studenten neben dem Beruf eines Lehrers für Mathematik, Physik oder andere Fächer auch die fachliche Ausbildung als Lehrer für Informatik und Rechentechnik erhalten.

Die Oberschulen, Fachschulen, pädagogischen Hochschulen und Universitäten mußten mit EDV-Anlagen ausgestattet werden. Da es vorläufig nicht möglich ist, an allen Schulen Fachkabinette einzurichten, besuchen die Kinder, die sich die Grundlagen der Informatik und Rechentechnik aneignen sollen, in vielen Gegenden eine Basisschule. In der Armenischen SSR und in manchen Gebieten der RSFSR wurde beschlossen, zu diesem Zweck Busse einzusetzen, die mit den notwendigen Geräten ausgerüstet sind und deren Fahrplan mit dem Stundenplan der betreffenden Oberschulen abgestimmt wird.

Eine fahrbare EDV-Anlage