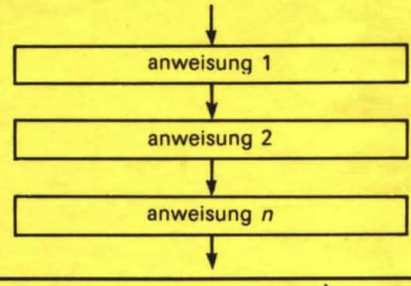
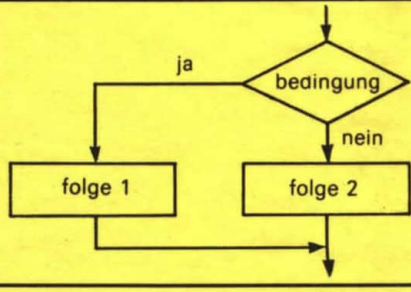
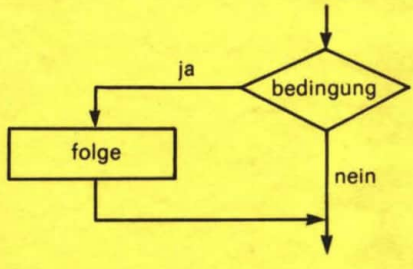
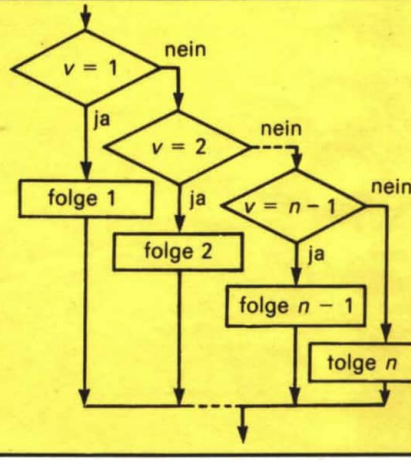
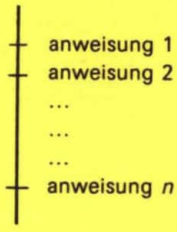
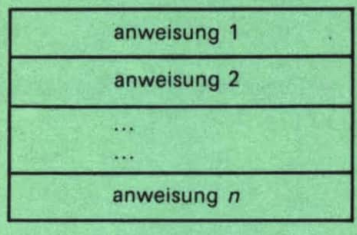
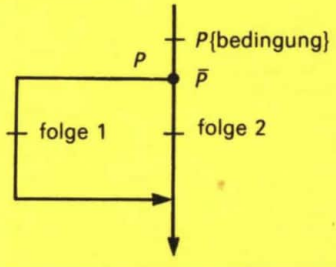
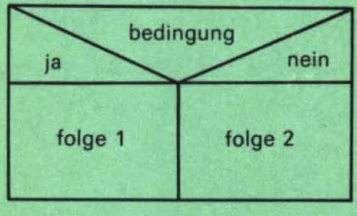
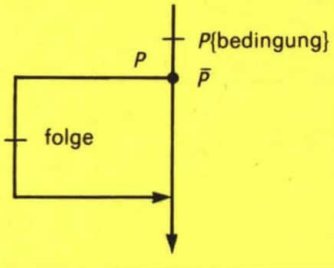
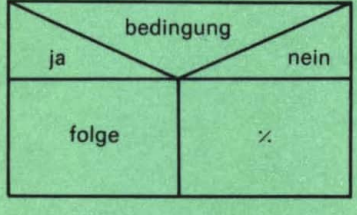
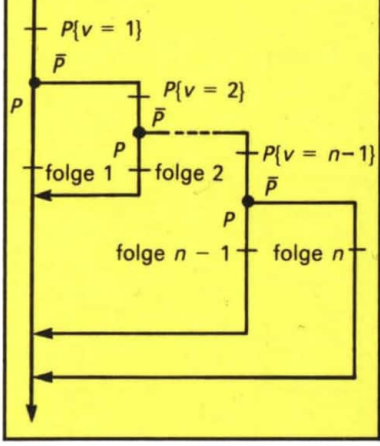
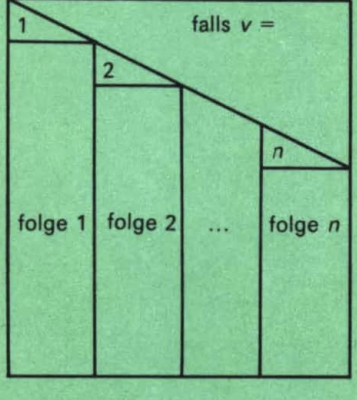
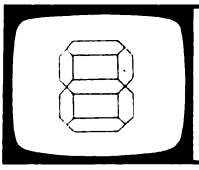
SCHILLING · TÖPFER

INFOR-MATIK

Strukturelemente der Algorithmierung in verschiedenen Darstellungsformen

	verbal formalisierte Notation	Sinnbilddarstellung
1. Folge (Sequenz)	anweisung 1 anweisung 2 ... ... anweisung *n*	
2. Auswahl (Selektion) 2.1. zweiseitige	WENN bedingung DANN folge 1 SONST folge 2	
2.2. einseitige	WENN bedingung DANN folge	
2.3. mehrseitige	FALLS *v* = 1: folge 1 2: folge 2 ... ... *n*: folge *n* ENDE	

Leitlinien-darstellung	Struktogramme	BASIC der Kleincomputer
		a1 anweisung 1 a2 anweisung 2 ... ... an anweisung n
		ak IF NOT(bedingung) THEN an al folge 1 am GOTO ap an folge 2 ap REM ende der zweiseitigen auswahl
		ak IF NOT(bedingung) THEN ap am folge ap REM ende der einseitigen auswahl
		ak ON v GOTO a1,a2, ..., an a1 folge 1 ... GOTO am a2 folge 2 ... GOTO am ... an folge n ... am REM ende der mehrseitigen auswahl



Informatik

Lehrbuch
für das strukturierte Programmieren

Volk und Wissen
Volkseigener Verlag Berlin

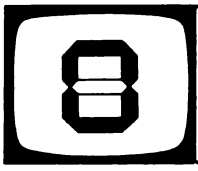
Autoren:

Dr. Ing. Alfred Schilling
Wolfgang Töpfer

Redaktion: Willi Wörstenfeld

ISBN 3-06-062502-6

© Volk und Wissen Volkseigener Verlag Berlin 1988
1. Auflage
Lizenz-Nr. 203 1000/88 (E 062502-1)
LSV 1083
Typografische Gestaltung: Atelier vwv, Uwe Rogal
Einband: Manfred Behrendt
Zeichnungen: Heinz Grothmann
Printed in the German Democratic Republic
Gesamtherstellung: Grafischer Großbetrieb Völkerfreundschaft Dresden
Schrift: 9/10 p Maxima, TVS
Redaktionsschluß: 18. 09. 87
Bestell-Nr.: 709 408 3
01650



Inhaltsverzeichnis

	Zum Gebrauch des Buches	6
1.	**Einführung in die Informatik**	**7**
1.1.	Grundbegriffe der Informatik	7
	Aufgaben	13
1.2.	Der Computer als Mittel der Informationsverarbeitung	13
	Wir nutzen ein fertiges Programm	14
	Aufgaben	24
1.3.	Alltagsalgorithmen	25
1.4.	Beispiele elementarer Algorithmen	26
1.4.1.	Wertetabellen – ein numerisches Beispiel	26
	Zusammenfassung	35
	Aufgaben	35
1.4.2.	Namen schreiben – ein nichtnumerisches Beispiel	36
	Zusammenfassung	40
	Aufgaben	40
2.	**Strukturierte Programmierung und Kleincomputer**	**42**
2.1.	Problemlösen in der Informatik	42
2.2.	Darstellungsmöglichkeiten für Algorithmen	46
	Aufgaben	50
2.3.	Grundstrukturen der strukturierten Programmierung und ihre Umsetzung in BASIC	51
2.3.1.	Folge	51
	Zusammenfassung	55
	Aufgaben	56
2.3.2.	Auswahl	57
	Zweiseitige Auswahl	58
	Einseitige Auswahl	61
	Mehrseitige Auswahl	65
	Anwendung der Auswahl bei der Simulation eines automatischen Verkaufs von S-Bahn-Fahrausweisen	69
	Zusammenfassung	74
	Aufgaben	74
2.3.3.	Wiederholung	76
	Zählschleife	76

	Erstes Anwendungsbeispiel zur Zählschleife	77
	Zweites Anwendungsbeispiel zur Zählschleife	79
	Drittes Anwendungsbeispiel zur Zählschleife	82
	Weitere Anwendungsbeispiele zur Zählschleife	90
	Zusammenfassung	97
	Aufgaben	98
	Wiederholschleife und Solangeschleife	101
	Anwendungsbeispiele zur Wiederholschleife	103
	Anwendungsbeispiel zur Solangeschleife	103
	Zusammenfassung	105
	Aufgaben	106
2.3.4.	Anwendungen der Grundstrukturen – Menütechnik	106
	Erstes Anwendungsbeispiel zur Menütechnik	106
	Zweites Anwendungsbeispiel zur Menütechnik	108
	Drittes Anwendungsbeispiel zur Menütechnik	113
	Zusammenfassung	118
	Aufgaben	119
2.3.5.	Unteralgorithmen	121
	Vereinbarung und Aufruf von Unteralgorithmen	121
	Vereinbarung und Aufruf von benutzerdefinierten Funktionen	126
	Zusammenfassung	127
	Aufgaben	127
2.4.	Datenstruktur Feld	128
	Zusammenfassung	134
	Aufgaben	134
2.5.	Arbeiten mit Menüs und Unterprogrammen bei komplexen Problemen	137
2.5.1.	Arten von Menüs	137
2.5.2.	Anwendung von Menüs und Unterprogrammen	140
2.6.	Regeln zur Gestaltung von BASIC-Programmen	147
	Aufgaben	150
3.	**Die gesellschaftliche Bedeutung der Informatik**	**155**
3.1.	Die Entwicklung der Informationsverarbeitung	155
3.2.	Computer in Forschung, Produktion und Verwaltung	159
4.	**Lösen nichtnumerischer Probleme mittels Kleincomputer**	**162**
4.1.	Dateien und Sortierprogramme	162
4.1.1.	Problembeschreibung	162
4.1.2.	Algorithmierung	164
	Entwickeln der Hilfsalgorithmen	166
	Erstellen strukturell ähnlicher Programmteile am Beispiel des Untermenüs 1 ANLEGEN EINER DATEI	172
	Programme zur Dateneingabe und Datenausgabe	173
4.1.3.	Verarbeiten der Datei	181
	Sortieren	181
	Suchen	190
	Zusammenfassung	193
	Aufgaben	194
4.2.	Textanalyse und Grafik	197
4.2.1.	Problemanalyse	197
4.2.2.	Algorithmierung	197

4.2.3.	Prozeduren und Unteralgorithmen zur Igel-Geometrie	205
	Zusammenfassung	212
	Aufgaben	212
5.	**Lösen numerischer Probleme mittels Kleincomputer**	**215**
5.1.	Elementare Näherungsverfahren	215
	Zusammenfassung	234
	Aufgaben	235
5.2.	Probleme der elementaren Stochastik	236
	Zusammenfassung	248
	Aufgaben	248
6.	**Vorprojekt: Meßwerterfassung und Meßwertauswertung**	**252**
6.1.	Problemanalyse und schrittweise Verfeinerung	252
6.1.1.	Fachlicher Hintergrund	252
6.1.2.	Experimentgestaltung	253
6.1.3.	Programmtechnische Problemanalyse	253
6.2.	Modulprogrammierung	254
6.2.1.	Hardwareergänzungen	254
6.2.2.	Modulprogrammierung	256
6.3.	Einzel- und Gesamttest	261
6.4.	Dokumentation	262
6.5.	Pflege und Wartung von Programmen	263
7.	**Projektarbeit**	**264**
7.1.	Allgemeines	264
7.2.	Projekt „Vokabeltrainer"	265
8.	**Weitere Problemklassen und andere Programmiersprachen**	**272**
8.1.	Weitere Problemklassen	272
8.2.	Vergleich einiger Programmiersprachen	274
	BASIC	275
	PASCAL	275
	FORTH	277
	LOGO	278
	PROLOG	279
9.	**Anhang**	**280**
	Im Text behandelte BASIC-Sprachelemente	280
	Im Text behandelte BASIC-Fehlermeldungen	281
	Literaturverzeichnis	281
	Sachwortregister	282

Zum Gebrauch des Buches

Es wird vorausgesetzt, daß auch die Bedien- und Programmierhandbücher für den jeweils benutzten Rechnertyp und BASIC-Interpreter benutzt werden.

Für die Darstellung der Anweisungen und Kommandos von BASIC werden in Übereinstimmung mit den Programmierhandbüchern [1], [2] zu den KC 85/n (n = 1, 2, 3) folgende Vereinfachungen verwendet:

Darstellung	Bedeutung
Worte in Großbuchstaben	Schlüsselworte von BASIC, die exakt so geschrieben werden müssen (z.B. CLS).
Worte in Kleinbuchstaben	Ersatzworte (Parameter), die durch aktuelle Werte ersetzt werden müssen (z.B. zeilennummer).
Sonderzeichen	Mit Ausnahme von [ ] müssen die Sonderzeichen exakt so geschrieben werden.
[text]	Der text kann (ohne []) wahlweise auftreten oder entfallen.
[text] ...	Der text kann wahlweise mehrfach hintereinander auftreten oder entfallen.
<taste>	Taste der Tastatur des Computers

Die vollständige Sprachbeschreibung von BASIC ist den Bedien- und Programmierhandbüchern des verwendeten Rechners mit BASIC-Interpreter zu entnehmen.

Beim Abfassen des Buches wurde überwiegend von

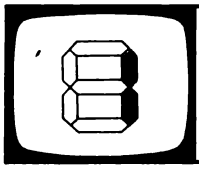
- Rechnern des Typs KC 85/2 mit BASIC-Modul und 16 KByte-RAM und
- Rechnern des Typs KC 85/3 mit mindestens 16 KByte-RAM

ausgegangen.

Im Lehrbuch verwendete Symbole:
▶ Merksätze
■ Beispiele
● Aufgaben
↗ siehe

Die Autoren danken hiermit allen, die zum Gelingen dieses Buches beigetragen haben. Für Hinweise zur weiteren Verbesserung sind Verlag und Autoren jederzeit dankbar.

Berlin/Leipzig, September 1987



Einführung in die Informatik

Was man zu verstehen gelernt hat,
fürchtet man nicht mehr.
(Marie Curie)

Wegen der verhältnismäßig kurzen Entwicklungszeit der Informatik als Wissenschaft gibt es noch keine allgemein anerkannte Definition. Deshalb soll die folgende Beschreibung einen ersten Einblick in die Aufgaben der Informatik geben.

Die **Informatik** beschäftigt sich
- mit der automatischen Verarbeitung von Informationen, d. h. ihrer Speicherung, Auswertung, Veranschaulichung und Übermittlung,
- mit den bei der Informationsverarbeitung verwendeten Verfahren und Sprachen,
- mit den technischen Hilfsmitteln der Informationsverarbeitung,
- mit den Einsatzmöglichkeiten und Auswirkungen der Informationsverarbeitung.

Die Informatik (engl. Computer science) ist heute weit mehr als nur ein technisches Hilfsmittel. Sie unterstützt auch das Denken und Problemlösen.
Man unterscheidet innerhalb der Wissenschaft Informatik noch zwischen den Teilgebieten *Kerninformatik* und *Angewandte Informatik*. Während sich die Kerninformatik vorwiegend mit den theoretischen und praktischen Problemen des Entwurfs, des Entwikkelns und des Einsatzes von Computern befaßt (theoretische und technische Informatik), untersucht die Angewandte Informatik die Aspekte der Anwendung der Informationsverarbeitung in anderen Wissenschaften, in Produktion und Verwaltung, in der Medizin, dem Bildungswesen sowie in der Kunst.
Schwerpunkt des Stoffes in diesem Buch ist die Angewandte Informatik.

1.1. Grundbegriffe der Informatik

Um sich in dem weiten Feld der Informatik orientieren zu können, sind viele neue Begriffe zu erlernen. Einige wichtige sind im folgenden erläutert.

Information. Sie umfaßt eine Nachricht zusammen mit ihrer Bedeutung für den Empfänger.

Eine äußerst nützliche menschliche Fähigkeit ist die Kommunikation, die Fähigkeit der Nachrichtenübermittlung zwischen einem Sender und einem Empfänger. Eine Nachricht wird zur Information, wenn Sender und Empfänger eine gemeinsame Sprache benutzen und wenn der Empfänger den Informationsgehalt der Nachricht erkennt. Befindet man sich im Ausland, so wird man eine Nachrichtensendung im Fernsehen als solche wahrscheinlich erkennen können, aber die dabei übermittelten Informationen kann man nur verstehen, wenn man der Fremdsprache mächtig ist.

Sprache. In der automatischen Informationsverarbeitung spielen Sprachen eine große Rolle. Jede Sprache ist ein System zur Darstellung und Übermittlung von Informationen zwischen Menschen oder zwischen Mensch und Maschine oder zwischen Maschinen. In diesem System ist eine eindeutige Menge von Zeichen – der Zeichenvorrat oder das Alphabet – definiert und sind Regeln zur Kombination der Zeichen zu größeren Einheiten – Worte, Sätze – festgelegt (Syntax). Darüber hinaus enthält das System Regeln über Wortverwendung und Wortanordnung, mit denen bestimmte Bedeutungen erzielt werden (Semantik).

Es sind vor allem formale Sprachen, deren Zeichen und Syntaxregeln vollständig exakt und eindeutig festgelegt sind, die in der Informationsverarbeitung in Form der Programmiersprachen eine so große Rolle spielen.

Daten. Auf der Grundlage einer Sprache besteht zwischen der Menge der Nachrichten und der Menge der dadurch verfügbaren Informationen eine für jeden Menschen spezifische Relation. Diese Relation läßt sich auch als Menge von Paaren aus Nachrichten und Informationen beschreiben. Diese Paare bezeichnen wir als Daten.
Ein *Datenelement* oder *Datum* besteht aus einer Nachricht und der darin enthaltenen Information.
In der Praxis der Datenverarbeitung unterscheidet man mehrere *Datenformen*:
- alphabetische Daten (z. B. Buch, Maier)
- numerische Daten (z. B. 12, 3.14)
- alphanumerische Daten (z. B. Schulze1, 1234Haus).

Daten werden im Rechner als Folgen von Binärziffern verschlüsselt (codiert) dargestellt. Diese Darstellung ist technisch günstiger zu realisieren als z. B. die dezimale Darstellung.

Bit (binary digit). Es ist die kleinste Einheit der Informationsdarstellung. Ein Bit kann genau einen von zwei möglichen Werten annehmen.
Das kann beispielsweise wahr/falsch, 0/1 oder ja/nein sein. Alle komplizierteren Sachverhalte müssen deshalb durch mehrere Bits dargestellt werden.

Zur Darstellung der zehn Ziffern der Dezimalzahlen mit Hilfe der Binärzahlen 0 und 1 werden jeweils vier dieser Zeichen zu einer Einheit – einem *Binärwort* – zusammengefaßt. So bedeuten
0010 die Dezimalzahl 2 und
1001 die Dezimalzahl 9.

Byte. Es ist eine Zusammenfassung mehrerer Bits zur Darstellung von Ziffern, Buchstaben und Sonderzeichen.
Ein Byte besteht aus
- acht Bits zur Informationsspeicherung,
- einem Bit für Prüfzwecke.

Bei den weiteren Betrachtungen wird unter Byte nur die Einheit der acht Informationsbits gesehen, da der Nutzer auf das Prüfbit keinen Einfluß hat. Es wird nach bestimmten Regeln vom Computer gesetzt.

In einem Byte können insgesamt $2^8 = 256$ verschiedene Zeichen dargestellt werden. Dies reicht aus, um
- die zehn Ziffern,
- die 26 Großbuchstaben,
- die 26 Kleinbuchstaben,
- Sonderzeichen und
- zusätzliche Zeichen (z. B. Grafiksymbole)

im Rechner darstellen zu können.

Die Ziffer 1 wird intern wie folgt abgespeichert:
0 0 1 1 0 0 0 1
Der Buchstabe A folgendermaßen:
0 1 0 0 0 0 0 1

Das Byte ist die kleinste adressierbare Informationseinheit im Computer. Bei heutigen Computern hat man Hauptspeicher, deren Größe man in Kilobyte (KB) oder in Megabyte (MB) angibt.

Es gilt: $1\ KB = 2^{10}\ Byte = 1024\ Byte$
$1\ MB = 2^{10}\ KB = 1024 \cdot 1024\ Byte = 1.048.576\ Byte$

Beachte: 1 KByte = 1024 Byte
1 kg = 1000 g!

Code. Die Vorschrift, nach deren Regeln die Abbildung eines Zeichenvorrates (z. B. das lateinische Alphabet) in einen anderen (z. B. das Maschinenalphabet eines Rechners) erfolgt, wird als Code bezeichnet.

Algorithmus. Er ist eine Verarbeitungsvorschrift, die so formuliert ist, daß sie auch von mechanischen oder elektronischen Geräten ausgeführt werden kann. Ein Algorithmus beschreibt exakt die schrittweise Umwandlung (Transformation) der Eingabedaten in Ausgabedaten.

Algorithmen besitzen folgende *Eigenschaften*:

- Ein Algorithmus dient zur Lösung einer ganzen Klasse von Problemen. Die Spezifizierung eines einzelnen Problems erfolgt über Parameter.
- Algorithmen bestehen letztendlich aus Elementaroperationen, die der Ausführende kennt.
- Algorithmen sind determiniert, d. h. unter gleichen Bedingungen liefern sie stets gleiche Ergebnisse.
- Die Beschreibung einer Problemlösung mittels eines Algorithmus besitzt eine endliche Länge.
- Ein Algorithmus endet nach endlich vielen Schritten.

Die Bezeichnung „Algorithmus" wurde aus dem Namen des arabischen Gelehrten Ibn Musa Al-Chwarismi (es existieren auch andere Schreibweisen dieses Namens) abgeleitet, der bereits im 9. Jahrhundert ein Buch über „Regeln der Wiedereinsetzung und Reduktion" schrieb.
Ein Algorithmus kann sequentiell oder nicht-sequentiell sein. Bei einem sequentiellen Algorithmus erfolgt eine Abarbeitung in der Reihenfolge der Bearbeitungsschritte. In nicht-sequentiellen Algorithmen werden Teile parallel nebeneinander bearbeitet (z. B. beim Autofahren). Entsprechend dem Stand der auch in naher Zukunft verfügbaren Computer werden wir im weiteren sequentielle Algorithmen behandeln.

Programm. Es ist ein Algorithmus, der in einer dem Computer verständlichen Sprache (Programmiersprache) formuliert ist und von diesem ausgeführt werden kann. Das Programm besteht aus *Vereinbarungen* und ausführbaren *Anweisungen*.

In den *Vereinbarungen* werden die Namen und die Eigenschaften der zu bearbeitenden Daten für das Programm festgelegt. Die *Anweisung* (statement) ist eine in der Programmiersprache abgefaßte Arbeitsvorschrift für den Computer zur Bearbeitung der Daten. Ein *Befehl* (instruction) ist eine elementare Anweisung, die sich in der benutzten Sprache nicht mehr in Teile zerlegen läßt, die selbst Anweisungen sind. Anweisung und Befehl werden in der Literatur häufig synonym verwendet.

Computer. Sie sind elektronische Systeme, die der elektronischen Informationsverarbeitung dienen. Diese Systeme bestehen aus den beiden Hauptkomponenten Hardware und Software.

Entsprechend der Leistungsfähigkeit (Verarbeitungsgeschwindigkeit und Speicherkapazität) unterscheidet man dabei
- Supercomputer,
- Großcomputer,
- Minicomputer,
- Mikrocomputer.

Supercomputer haben eine Verarbeitungsgeschwindigkeit von mehr als 1 Milliarde Operationen je Sekunde und einen Hauptspeicher von mehreren Millionen Byte.

Großcomputer haben eine Verarbeitungsgeschwindigkeit bis zu 1 Milliarde Operationen je Sekunde und einen Hauptspeicher von mehreren Millionen Byte.

Minicomputer sind kleinere Rechenanlagen mit Verarbeitungsgeschwindigkeiten bis zu mehreren Millionen Operationen je Sekunde.

Zur Gruppe der *Mikrocomputer* gehören der Personalcomputer PC 1715 und die Kleincomputer KC 85/1, KC 85/2, KC 85/3 und KC 87.

Hardware. Sammelbezeichnung für alle praktisch anfaßbaren Bestandteile eines Computers.
Dazu gehören:
- die Zentraleinheit (engl. CPU = central processing unit), oft als zentrale Verarbeitungseinheit ZVE bezeichnet,
- die Peripheriegeräte.

Zentraleinheit. Die wichtigsten Funktionseinheiten sind:
- der Prozessor (das Steuerwerk und das Rechenwerk)
- der Hauptspeicher (ROM und RAM).

Die Zentraleinheit (Abbildung 1.1/2)
- überwacht und steuert das ganze System (mit dem Steuerwerk),
- verknüpft die Daten arithmetisch oder logisch (mit dem Rechenwerk),
- speichert Daten und Programme (im Hauptspeicher),
- ruft benötigte Daten und Programme von Peripheriegeräten ab,
- leitet die Ausgabe von Daten und Programmen zu Peripheriegeräten ein.

Der *Hauptspeicher* besteht aus den Baugruppen ROM und RAM. Dies sind unterschiedliche Bauformen von Halbleiterspeichern. Dabei ist der **ROM** ein nur lesbarer Speicher (read only memory), in dem wesentliche Programme zum Betrieb des Rechners durch den Hersteller eingespeichert werden. **RAM** ist der Speicherteil, mit dem der Nutzer im Rahmen der Programmabarbeitung arbeiten kann. Der RAM ist ein Lese-/Schreibspeicher (random access memory), der die eingespeicherten Daten und Programme nur zeitweise speichert. Spätestens, wenn der Nutzer den Rechner ausschaltet, werden alle Daten im RAM gelöscht.

Bus. Die Informationsübertragung innerhalb des Computers und zu den Peripheriegeräten erfolgt über *Kanäle*. Bei den Mikrocomputern werden diese als Bus bezeichnet.
Je nach Art der zu transportierenden Informationen unterscheidet man zwischen *Datenbus, Adreßbus* und *Steuerbus*.

Periphere Geräte. Sie dienen der Ein- und Ausgabe sowie der Speicherung von Daten und Programmen.
In Abbildung 1.1/2 sind die Bezeichnungen der wichtigsten Gruppen peripherer Geräte angeführt. An Mikrocomputern können beispielsweise folgende periphere Geräte angeschlossen werden:

Gruppe	Geräte
Eingabegeräte	Tastaturen Lochschriftleser Strichmarkierungsleser
Ausgabegeräte	Bildschirme Drucker Plotter (Zeichengeräte) Sprachausgabegeräte
Speichergeräte	Magnetbandkassettengeräte Diskettengeräte

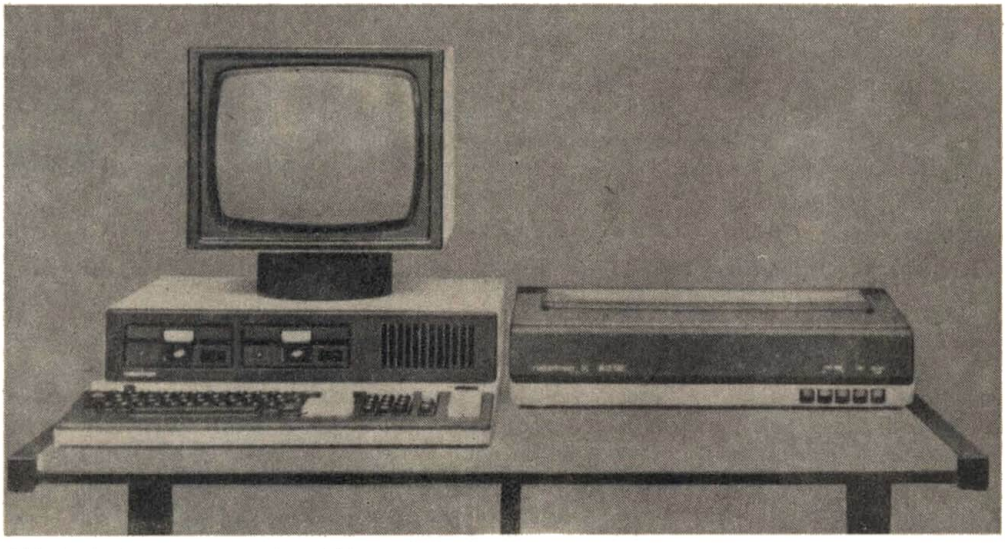


Abb. 1.1/1: Personalcomputer PC 1715

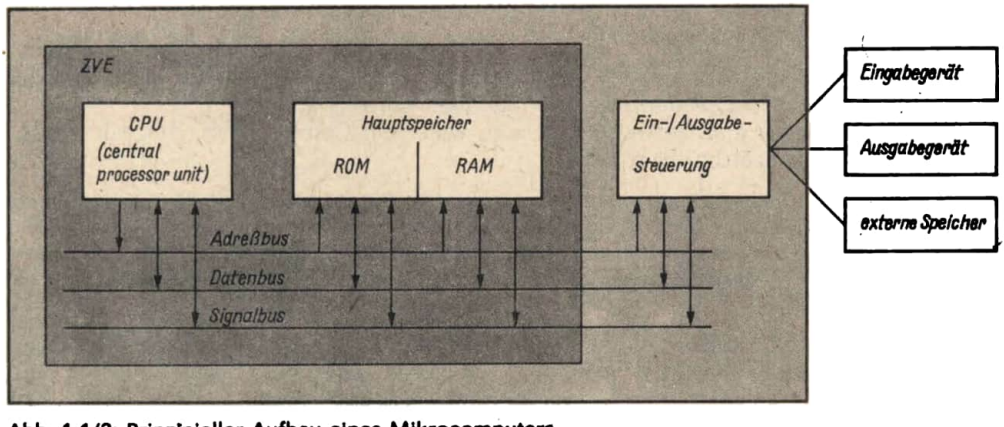


Abb. 1.1/2: Prinzipieller Aufbau eines Mikrocomputers

Im Zusammenhang mit den derzeit verfügbaren Personal- und Kleincomputern soll hier nur kurz auf die Speichergeräte eingegangen werden, da Tastaturen, Bildschirme und Drucker dem Leser bekannt sein dürften.
Grundprinzip der externen Speichermedien *Magnetbandkassette* und *Diskette* ist die Verwendung von magnetisisierbaren Schichten. In Abbildung 1.1/3 sind die Informationsspeicherung auf Kassette und Diskette gegenübergestellt.
Die *Magnetbandkassette* ist von Tonaufzeichnungen her bekannt. Wesentlicher Unterschied zwischen der Tonaufzeichnung und der Aufzeichnung digitaler Signale besteht darin, daß die Bits eines Zeichens als Rechteckimpulse in Laufrichtung sequentiell aufgezeichnet werden. Zur Suche nach einer ganz bestimmten Information muß die Kassette unter Umständen ganz abgespielt werden. Es dauert ziemlich lange, bis der Rechner die geforderte Information erhält.

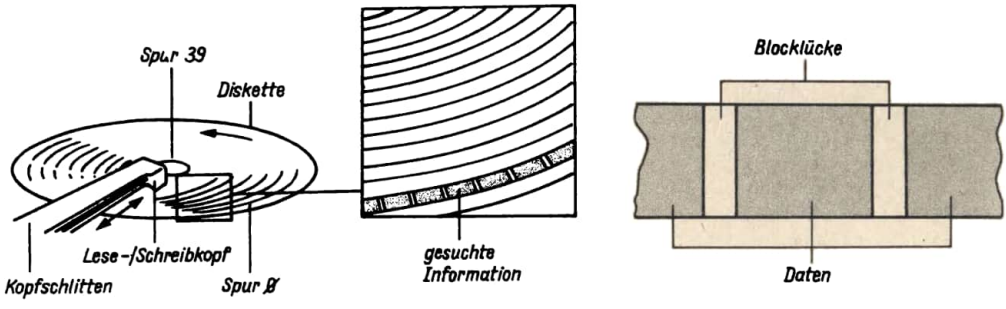


Abb. 1.1/3: Informationsspeicherung auf Diskette und Kassette

Die *Diskette* (Floppy Disk) ähnelt einer Schallplatte in Schutzhülle. Die dünne runde Kunststoffolie im Inneren der Schutzhülle ist beidseitig mit einer Magnetschicht beschichtet. Die Daten werden über einen Lese-/Schreibkopf in konzentrischen Kreisen in die Magnetschicht eingetragen bzw. von dort gelesen. In den äußeren Spuren ist das Inhaltsverzeichnis enthalten, das unmittelbar nach Einlegen der Diskette gelesen und im Hauptspeicher aufbewahrt wird.

Soll nun eine Information gesucht werden, so kann schnell im Inhaltsverzeichnis die Nummer der Spur – des entsprechenden Kreises – festgestellt und der Lese-/Schreibkopf direkt auf diese Spur eingestellt (Direktzugriff zu den Daten) werden. So können die Daten sehr schnell in den Hauptspeicher übertragen werden. Der Zugriff zu einem bestimmten Datum geht bei der Diskette also wesentlich schneller als bei der Kassette.

Software. Sie ist die Sammelbezeichnung für die Programme, die zur *Steuerung der Hardware* und zur *Manipulation von Informationen* notwendig sind.
Ein Teil der Software muß austauschbar sein, wenn ein Computer als Vielzweck- oder Universalrechner einsetzbar sein soll.

Grundsätzlich wird zwischen *System-* und *Anwendersoftware* unterschieden. Dabei ermöglicht die Systemsoftware dem Nutzer, einen Computer in Betrieb zu nehmen. Die Systemsoftware allein reicht aber nicht aus, um mit dem Computer bestimmte Probleme zu lösen; dazu bedarf es noch der Anwendersoftware. Die Systemsoftware wird von den Herstellern der Computer erarbeitet und bereitgestellt.

Das Betriebssystem besteht im wesentlichen aus Programmen, die eine effektive Nutzung der Rechenanlage gewährleisten. Auf der Grundlage des Betriebssystems kann der Nutzer dann Anwendersoftware erstellen oder erwerben. Neben dem Betriebssystem erhält der Nutzer eine Gruppe von Programmen der Systemsoftware.

Software	
Systemsoftware	Anwendersoftware
Betriebssystem Programmorganisation Ein-/Ausgabeverwaltung Externspeicherverwaltung Laufzeitorganisation	**anwenderneutrale Programme** (z. B. Lohnrechnungsprogramm)
Systemunterstützung Dienstprogramme Hilfsprogramme Programmierunterstützungen Prüf- und Mess-Programme	**anwendungsspezifische Programme** (z. B. Programme zur Ermittlung der Energiekosten im Haushalt)

Bei den Kleincomputern werden Teile der Systemsoftware vom Rechnerhersteller im Rechner (im ROM) fest eingespeichert. Dadurch ist der Computer sofort nach dem Einschalten betriebsbereit. Dies ist bei unseren Kleincomputern in unterschiedlichem Maß der Fall. Weitere Teile des Betriebssystems, wie Übersetzerprogramme der verschiedenen Programmiersprachen, Testhilfen u. a. kann man im Bedarfsfall über externe Datenträger in den Rechner einlesen bzw. in Form von steckbaren Speichermodulen (ROMs) in den Computer stecken.

Firmware. Programme, die von Software-Herstellern in ROMs eingespeichert wurden, werden auch als Firmware bezeichnet.

Beim KC 85/2 und KC 85/3 gibt es beispielsweise Module für die Programmiersprachen BASIC und FORTH.

● Aufgaben

1. Erläutern Sie die Begriffe Bit, Byte und Binärwort!
2. Wie viele Bits müssen Binärworte mindestens umfassen, die zur Codierung aller Großbuchstaben des lateinischen Alphabets geeignet sein sollen?
3. Eine Magnetbandkassette (K 60) hat beispielsweise eine Speicherkapazität von 1,5 MB. Wie viele A4-Seiten Text zu je 2,3 KB (ca. 60 Zeichen/Zeile und ca. 38 Zeilen/Seite) können auf dieser Kassette gespeichert werden?

1.2. Der Computer als Mittel der Informationsverarbeitung

In der gegenwärtigen Etappe der wissenschaftlich-technischen Revolution werden bestimmte, sich wiederholende geistige Tätigkeiten in zunehmendem Maße durch Maschinen übernommen.
Diese Maschinen – als Computer, Rechner, Datenverarbeitungsanlagen bezeichnet – verarbeiten Eingabeinformationen nach bestimmten Vorschriften und erzeugen Ausgabeinformationen.

Es werden drei Grundelemente für den Prozeß der Datenverarbeitung benötigt:

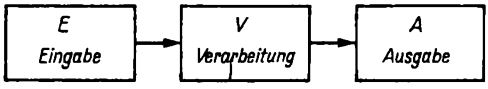


Abb. 1.2/1: E V A – Prinzip

- eine **Eingabeeinheit** zur Übermittlung der Daten von der Umwelt an die zentrale Verarbeitungseinheit,
- eine zentrale **Verarbeitungseinheit** (ZVE),
- eine **Ausgabeeinheit** zur Übermittlung der Daten von der ZVE an die Umwelt.

Diese drei Grundelemente kann man auch bei dem Computer KC 85/1 des Kombinates Robotron (VEB Meßelektronik, Dresden) und den Computern KC 85/2 sowie KC 85/3 des Kombinates Mikroelektronik (VEB Mikroelektronik „Wilhelm Pieck", Mühlhausen) unterscheiden (Abbildung 1.2/2). Bei ihnen ist die Tastatur die Eingabeeinheit und der Bildschirm des Fernsehgerätes die Ausgabeeinheit. Die zentrale Verarbeitungseinheit ist beim KC 85/1 von Robotron zusammen mit der Tastatur in einem Gehäuse untergebracht. Beim KC 85/2 und KC 85/3 sind Tastatur und ZVE getrennte Baugruppen, wobei die ZVE in einem extra Gehäuse eingebaut ist.

Die Kleincomputer KC 85/1 sowie KC 85/2 und /3 haben zwei unterschiedliche Arbeitsmodi, den *Kommando-* oder *Direktmodus* und den *Programm-Modus*.

Kommando-Modus (Direktmodus). Der KC wird in ähnlicher Weise wie ein Taschenrechner genutzt. Die eingegebenen Kommandos und Befehle werden unmittelbar nach Abschluß der Eingabe verarbeitet.

Programm-Modus. Die eingegebenen Befehle werden im Rechner gespeichert und stehen als eine Befehlsfolge zur wiederholten Verarbeitung bereit.

Wir nutzen ein fertiges Programm

Mit dem Kommando CLOAD "IGELGEO" laden wir das Programm IGEL-GEOMETRIE in den KC 85/2 oder KC 85/3. Der Name IGEL ist aufgrund des Zeichens entstanden, das wir auf dem Bildschirm bewegen können. Dieses Zeichen sieht wie die spitze Schnauze eines Igels aus. Für den KC 85/1 gibt es ein ähnliches Programm mit dem Namen PAGE.
Nach dem Start des Programms wird der aus 320*256 Bildpunkten bestehende Bildschirm in zwei Bereiche aufgeteilt. Der obere Bereich von 320*200 Punkten wird als Zeichenfläche verwendet. Der verbleibende untere Bereich umfaßt sieben Zeilen, in die die Befehle für den Igel geschrieben werden können (Befehlseingabebereich). Abbildung 1.2/3 zeigt das Startbild des IGEL-Programms.
Die Sprache, mit der man den Igel im Programm steuern kann, besteht aus zwei Gruppen von Befehlen.

Die 1. Gruppe umfaßt die acht Befehle zur Bewegung auf dem Bildschirm:

1. **VORWAERTS** *x* — Der Igel bewegt sich um *x* Bildpunkte in Blickrichtung.
2. **RECHTS** *y* — Der Igel dreht sich im Uhrzeigersinn um *y* Grad.
3. **WIEDERHOLE** *z*(..) — Wiederholung der in der Klammer beschriebenen Bewegung *z*-mal.
4. **HEBE** — Der Zeichenstift des Igels wird von der Zeichenfläche abgehoben; es wird keine Spur mehr gezeichnet.

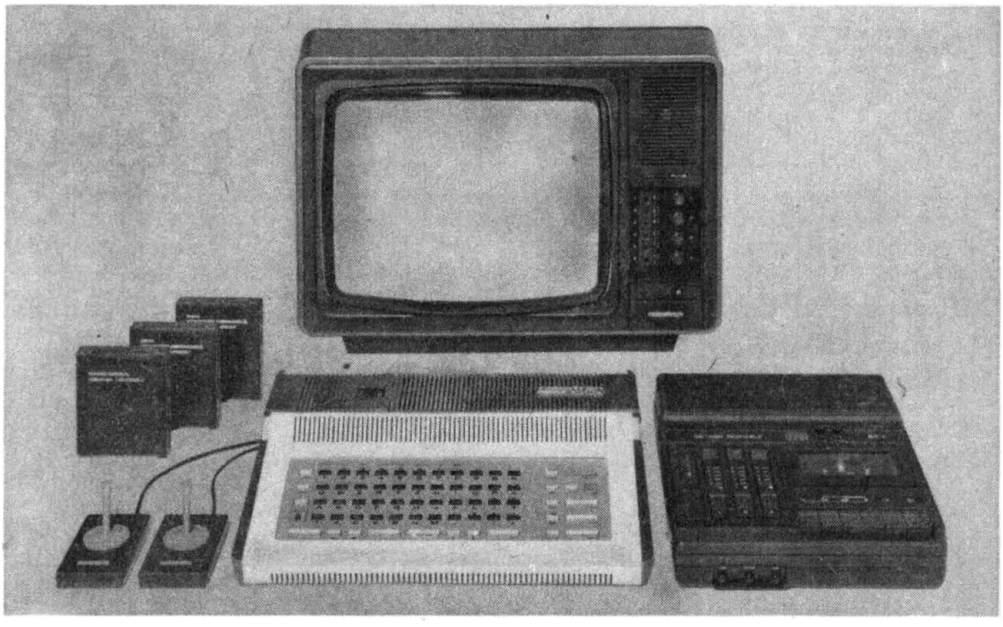


Abb. 1.2/2a: KC 85/1 des VEB Robotron Meßelektronik „Otto Schön", Dresden



Abb. 1.2/2b: KC 85/3 des Kombinates Mikroelektronik (VEB Mikroelektronik „Wilhelm Pieck", Mühlhausen)

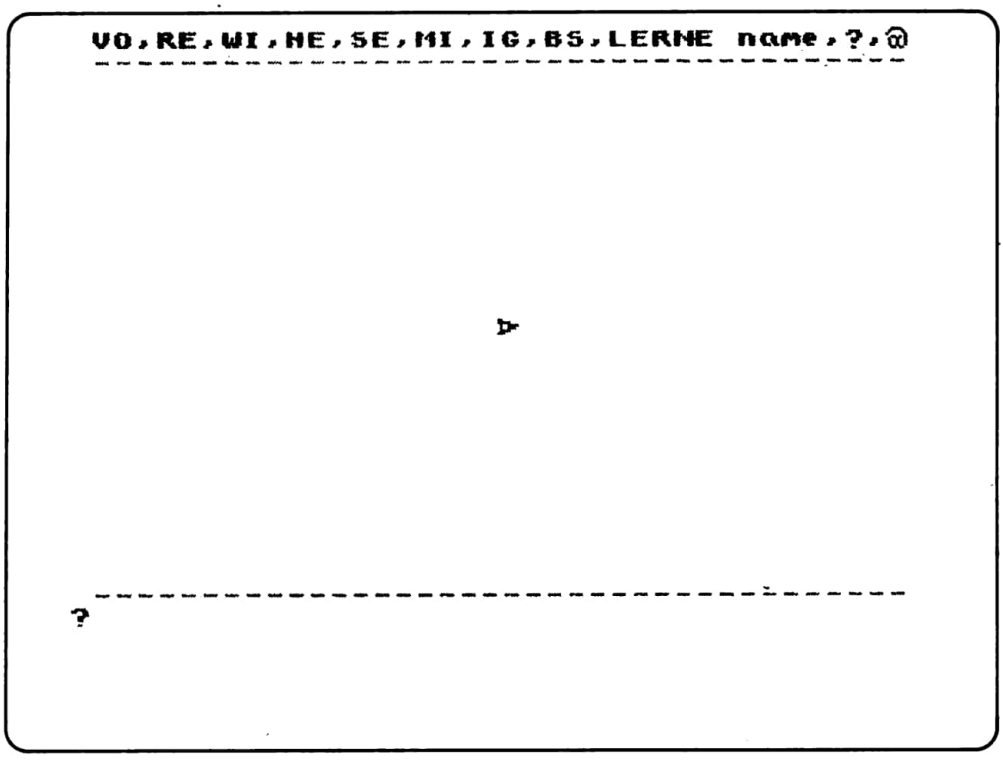


Abb. 1.2/3: Startbild IGEL-GEO

5. **SENKE** Der Zeichenstift des Igels wird wieder abgesenkt, die Spur wird wieder gezeichnet.
6. **MITTE** Der Igel wird in die Ausgangsposition gesetzt.
7. **IGEL** Der Igel wird unsichtbar gemacht, wenn er sichtbar war bzw. sichtbar, falls er unsichtbar war.
8. **BILDSCHIRM SAEUBERN** Der Bildschirm wird gelöscht (beide Bereiche).

Die 2. Gruppe enthält 2 Befehle, mit denen man dem Igel Befehlsfolgen ‚beibringen' kann.

9. **LERNE name**
10. **ENDE**

Hinweis: Es ist wichtig, daß zwischen den Befehlsworten wie VORWAERTS, RECHTS, WIEDERHOLE, LERNE und den zugehörigen Operanden jeweils *wenigstens ein Leerzeichen* steht.

▶ Die Eingabe eines vollständigen Befehls ist mit der Taste <ENTER> abzuschließen.

Wie Abbildung 1.2/3 zeigt, meldet sich das Programm mit dem Zeichen ‚?' in der ersten Zeile des Befehlseingabebereichs bereit zum Dialog mit dem Nutzer. Wir geben nun die folgenden 5 Befehle ein:

```
? VORWAERTS 80   <ENTER>
? RECHTS 90      <ENTER>
? VORWAERTS 80   <ENTER>
? RECHTS 90      <ENTER>
? VORWAERTS 80   <ENTER>
```

▶ Befehls- und Kommandoeingaben sind durch Betätigen der ENTER-Taste abzuschließen. Es wird künftig nur in Ausnahmen darauf hingewiesen!

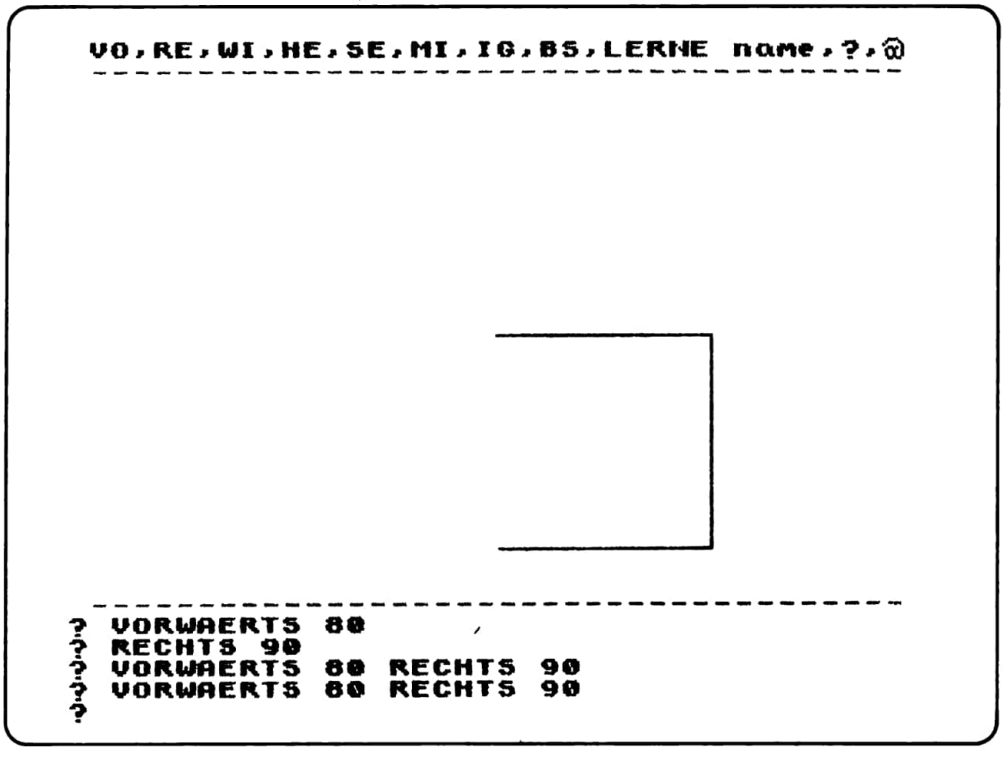


Abb. 1.2/4: Ergebnis der ersten Befehle

Entsprechend dieser Befehle hinterläßt der Igel eine Spur. Diese stellt 3 Seiten eines Rechtecks dar (Abbildung 1.2/4). Da jeder Befehl sofort nach der Eingabe vom Igel ausgeführt wird, spricht man hier vom Direkt- oder Kommando-Modus.
Fügen wir an die obige Befehlsfolge noch die Befehle

```
RECHTS 90
VORWAERTS 80
RECHTS 90
```

an, so ist auf der Zeichenfläche ein vollständiges Rechteck entstanden, und der Igel steht wieder in seiner Ausgangsstellung.

Mit BILDSCHIRM SAEUBERN stellen wir das Startbild des Programms wieder her. Es werden dabei die Zeichenfläche und der Befehlseingabebereich gelöscht.
Wollten wir erneut das Rechteck zeichnen, so müßten wir alle 8 Befehle neu eingeben. Das ist nicht besonders rationell. Es wäre besser, wenn der Igel sich ‚merken' könnte, daß das gewünschte Rechteck durch die obigen 8 Befehle gezeichnet wird. Dann könnte er ein Rechteck zeichnen, wenn wir ihm nur den Befehl ‚RECHTECK' eingeben. Wie wir aus der obigen Befehlsübersicht entnehmen, besteht im Programm ‚IGEL-GEOMETRIE' diese Möglichkeit durch die Befehle „LERNE name" und „ENDE".

Wenn wir dem Igel also ‚beibringen' wollen, welche Befehlsfolge für das Rechteck abzuarbeiten ist, teilen wir ihm durch den Befehl

```
LERNE RECHTECK
```

mit, daß er die anschließend einzugebende Folge von Befehlen nicht sofort ausführen, sondern sich zunächst nur merken soll. Erst wenn wir den Namen RECHTECK aufrufen, soll er die entsprechenden Befehle vollständig in der vorgegebenen Reihenfolge abarbeiten. Im folgenden soll dieser Weg noch einmal beschrieben werden.
Nach Eingabe von LERNE RECHTECK löscht das Programm beide Bildschirmbereiche und meldet sich bereit zur Eingabe des ersten zu ‚lernenden' Befehls mit:

```
PROZEDUR RECHTECK
1 ?■
```

Wir geben den Befehl VORWAERTS 80 ein. Der Rechner ist sofort zur zweiten Eingabe bereit:

```
2 ?■
```

Haben wir auf diese Weise alle acht Befehle eingegeben, so meldet sich das Programm mit

```
9 ?■
```

wieder. Da nun alle notwendigen Befehle für das Zeichnen des Rechtecks eingegeben sind, teilen wir dem Igel mit, daß der LERN-MODUS zu beenden ist:

```
9 ?ENDE
```

Danach löscht das Programm wieder beide Bildschirmbereiche. Der Igel steht in seiner Startposition und wartet auf Befehle. Er hat den neuen Befehl ‚RECHTECK' gelernt, der nun zu seinem Befehlsvorrat gehört. In der Abbildung 1.2/5 wird dieser neue Befehl verwendet und die Abbildung 1.2/6 zeigt das Ergebnis dieser Befehlsfolge.
Das bisher erworbene Wissen wollen wir nutzen, um ein Haus zu zeichnen, wie es in Abbildung 1.2/9 dargestellt ist.
Wir erkennen, daß das Haus aus den Grundelementen Rechteck (Haussockel), Dreieck (Dach) und Quadratvierlinge (Fenster) besteht. Davon ist dem Igel bisher nur das Rechteck bekannt.

```
VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@
-------------------------------------

  PROZEDUR RECHTECK

1     ? VORWAERTS 80
2     ? RECHTS 90
3     ? VORWAERTS 50
4     ? RECHTS 90
5     ? VORWAERTS 80
6     ? RECHTS 90
7     ? VORWAERTS 50
8     ? RECHTS 90
9     ? ENDE
```

Abb. 1.2/5: Programm

```
VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@
-------------------------------------



-------------------------------------
? RECHTECK
?
```

Abb. 1.2/6: Ergebnis

Für die anderen Grundelemente müssen wir ihm noch Programme mitteilen. Ein geeignetes Dach erzeugt beispielsweise das Programm in Abbildung 1.2/7. An diesem Programm ist ersichtlich, daß mehrere Befehle auf eine Zeile geschrieben werden können, wenn zwischen den einzelnen Befehlen (mindestens) ein Leerzeichen steht.
Beim Fenster sind am einfachsten vier kleine Quadrate zu zeichnen, wobei jedes Quadrat durch die folgende Befehlsfolge erstellt wird.

```
VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@
-------------------------------

  PROZEDUR DACH

1     ? VORWAERTS 85 RECHTS 210
2     ? VORWAERTS 52 RECHTS 300
3     ? VORWAERTS 51 RECHTS 210
4     ? VORWAERTS 5
5     ? ENDE
```

```
VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@
--------------------------------------

                    /\
                   /  \
                  /    \
                 /      \
                /        \
               /          \
              /            \
             /              \
            /                \
           /                  \
          /                    \
         /                      \
        /                        \
       /                          \
      /                            \
     /                              \
    /                                \
   /                                  \
  /                                    \
 /                                      \
/                                        \
-----------------------------------------

? DACH
```

Abb. 1.2/7: Programm ‚DACH' und Abarbeitungsergebnis

```
WIEDERHOLE 4 (VORWAERTS 10 RECHTS 90)
```

Durch das Programm in Abbildung 1.2/8 wird ein geeignetes Fenster gezeichnet.
Jetzt sind dem Igel alle oben genannten Grundelemente für das Haus bekannt. Auf dieser Grundlage kann nun das Programm zusammengestellt werden. Abbildung 1.2/9 zeigt das Programm und das Ergebnis einer Programmabarbeitung.
Schließlich kann man nun auch eine Straße mit beispielsweise drei Häusern zeichnen. Man nutzt dazu das folgende Programm (Abbildung 1.2/10).
Mit den bisherigen Programmen haben wir immer feste Typen von Fenstern, Dächern, Häusern und Straßen definiert. Noch vielfältiger wären die Variationen, wenn man die Möglichkeit hätte, die Abmessungen dieser Elemente variabel zu definieren. Dies soll an einer neuen Definition des Elementes RECHTECK gezeigt werden (Abbildung 1.2/11).

```
VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@
-------------------------------------

  PROZEDUR FENSTER

1    ? WI 4 (
2    ?    WI 4 ( VO 10 RE 90 )
3    ?       RE 90 )
4    ? ENDE
```

```
VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@
---------------------------------------

                    +---+---+
                    |   |   |
                    +---+---+
                    |   |   |
                    +---+---+

---------------------------------------
? FENSTER
?
```

Abb. 1.2/8: Programm ‚FENSTER' und Abarbeitungsergebnis

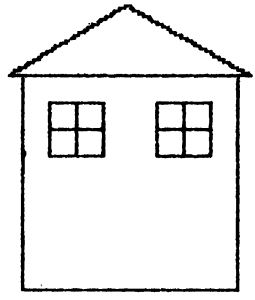
Der Aufruf des neu definierten Programms RECHTECK in Abbildung 1.2/11 zeigt die erreichte Variabilität, da nun mit dem gleichen Programm auch Quadrate gezeichnet werden können. Auch andere Elemente könnte man in dieser Weise neu definieren und so recht variable Stadtansichten auf der Suche nach einer optimalen Anordnung auf dem Bildschirm erstellen. Das Programm IGEL-GEOMETRIE deutet die vielfältigen Anwendungsbereiche für Grafikprogramme an. Es zeigt aber auch, wie ein computergestützter Entwurf (engl. computer aided design – CAD) in den Konstruktionsbüros eine Rationalisierung bewirken kann. Am Bildschirm ist es ein Leichtes, ein Flachdach gegen ein steileres Dach oder einen verputzten Haussockel gegen einen mit Klinkern gemauerten auszutauschen.
Zur Beurteilung der optischen Wirkung reicht es also schon aus, mit einem Computer Ansichten zu entwerfen.

```
VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@
-------------------------------

  PROZEDUR HAUS

 1    ? RECHTECK DACH
 2    ? HE VO 20 RE 90 VO 20 SE
 3    ? FENSTER RE 270 HE VO 40 SE
 4    ? FENSTER RE 180 HE VO 70
 5    ? RE 90 VO 20 RE 90
 6    ? ENDE
```

```
VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@
---------------------------------------
```



```
---------------------------------------
? HAUS
?
```

Abb. 1.2/9: Programm ‚HAUS' und Abarbeitungsergebnis

- Bei Vorhandensein entsprechender Programme können am Bildschirm eines Computers modische Verzierungen von Schuhen ebenso durchprobiert werden wie die äußere Gestaltung von Fahrzeugen. Besonders effektiv wird es, wenn das grafische Programm um ein Programm ergänzt wird, das sofort die ökonomischen Konsequenzen (Kosten, Materialverbrauch) einer gestalterischen Veränderung errechnet.

  Solche Programme sind natürlich wesentlich komplizierter und umfangreicher als die IGEL-GEOMETRIE. Ihre Nutzung muß wie alles im Leben erlernt werden. Die Verständigung zwischen Nutzer und Programm erfordert in diesen Fällen einen wesentlich umfangreicheren Wortschatz als bei unserem Demonstrationsprogramm. Aber das Prinzip der Nutzung gilt auch für andere Anwendungsgebiete.

- Ein Roboter, der als Ganzes um 360 Grad drehbar ist, seinen Arm heben und senken kann und damit das Anheben bzw. Absetzen von metallischen Gegenständen mittels Elek-

```
VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@
-------------------------------

  PROZEDUR STRASZE

 1    ? MI  RE  180  HE  VO  150
 2    ? RE  180  SE  HAUS
 3    ? VO  110  SE  HAUS
 4    ? VO  110  SE  HAUS
 5    ? ENDE
```

```
VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@
---------------------------------------
```

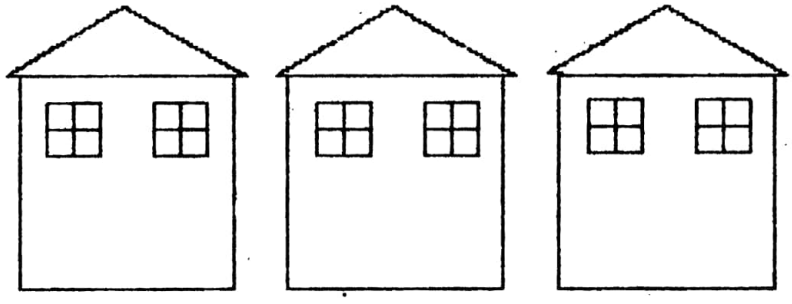


Abb. 1.2/10: Programm ‚STRASZE' mit drei Häusern und Ergebnis

tromagneten realisiert, könnte mit einem zur IGEL-GEOMETRIE ähnlichen Sprachumfang gesteuert werden:

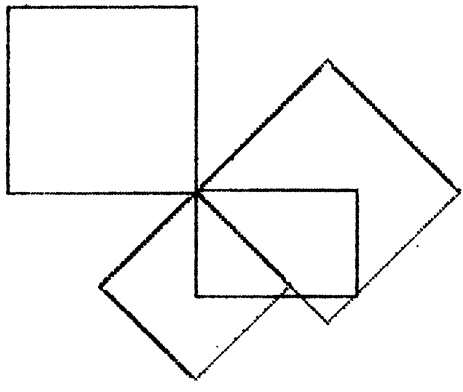
```
{LINKS DREHEN x, RECHTS DREHEN y, AUF z, AB t, MAGNET AN,
MAGNET AUS, MERKE,'ENDE, START}.
```

Bewegungsabläufe von einem Start- zu einem Zielpunkt ‚erlernt' der Computer des Roboters mit einem geeigneten Programm dadurch, daß die Endpunkte über Handsteuerung angefahren und die Koordinaten der Punkte gespeichert werden. Man sagt: ‚Der Roboter wird nach dem „Teach in" – Prinzip programmiert'. Das Programm errechnet den optimalen Verbindungsweg selbst. Dann steuert der Computer den Roboter gemäß dem berechneten Weg und führt an bestimmten Punkten die verlangten Tätigkeiten aus.

```
VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@
-------------------------------

  PROZEDUR RECHTECK
1     ? WI 2 (
2     ?        VO L RE 90 VO 6 RE 90 )
3     ? ENDE
```

```
VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@
---------------------------------------
```



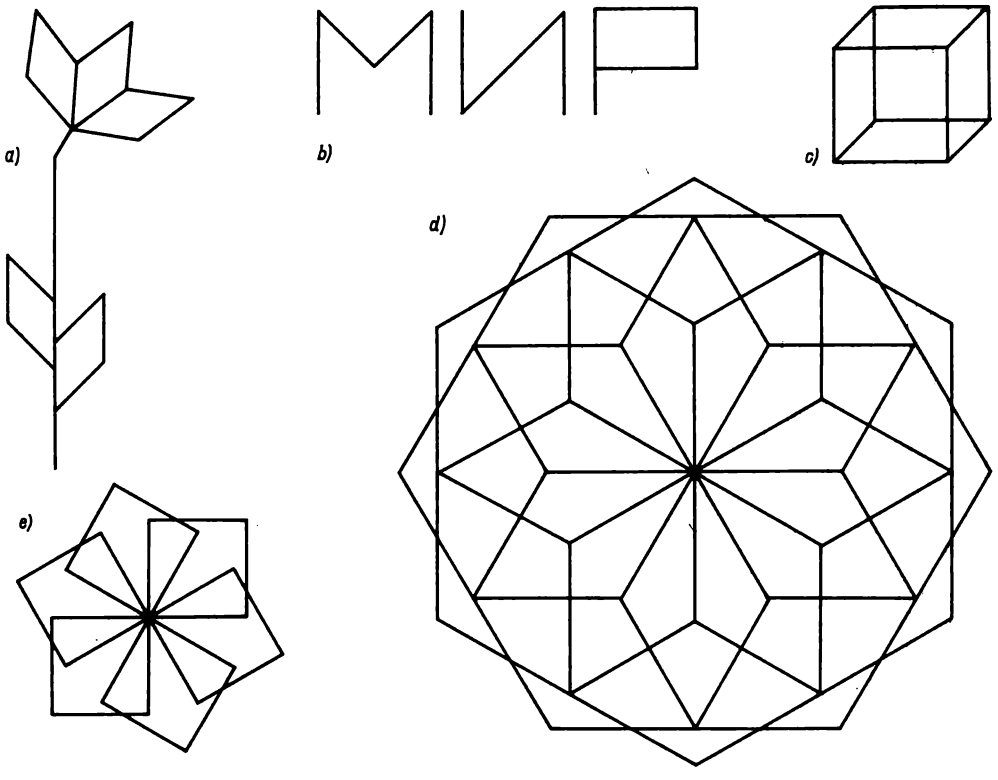
```
-----------------------------------------
? RECHTECK 60 40
? RE 45 RECHTECK 50 50
? WI 2 ( RE 135 RECHTECK 70 70 )
?
```

Abb. 1.2/11: Programm und Ergebnis zu RECHTECK

● Aufgaben

1. Überlegen Sie, welche geometrischen Objekte durch folgende Programme der IGEL-GEOMETRIE gezeichnet werden:
   a) WIEDERHOLE 5 (VORWAERTS 20 RECHTS 144) ENDE
   b) WIEDERHOLE 6 (VORWAERTS 50 RECHTS 60) ENDE
   c) WIEDERHOLE 3 (VORWAERTS 10 RECHTS 90)
      VORWAERTS 10 RECHTS 135
      VORWAERTS 14.14 RECHTS 75
      VORWAERTS 10 RECHTS 120
      VORWAERTS 10 RECHTS 75
      VORWAERTS 14.14
      ENDE

2. Nutzen Sie die IGEL-GEOMETRIE, um folgende Zeichnungen auf dem Bildschirm zu erzeugen:



3. Überlegen Sie, welches geometrische Objekt durch das folgende Programm der IGEL-GEOMETRIE gezeichnet wird:
WI 12 (RE 30 WI 4 (VO 20 RE 90) RE 180 VO 30 RE 210) ENDE

1.3. Alltagsalgorithmen

Im täglichen Leben begegnen uns immer wieder Algorithmen, was vielen von uns aber gar nicht bewußt wird. Kaufen wir uns ein technisches Gerät – Staubsauger, Waschmaschine oder Fernsehgerät – so erwerben wir stets auch eine Bedienungsanleitung mit. Betrachtet man beispielsweise die Anleitung für das Fernsehgerät, so heißt es dort:

- Netzanschluß herstellen!
- Antennenanschluß herstellen!
- Netzschalter am Gerät betätigen!
- Am Kanalwähler gewünschten Sender einstellen, dabei auf entsprechendes Übertragungsband achten!
- Lautstärke regulieren!
- Bildhelligkeit und -kontrast mittels entsprechender Regler einstellen!

Noch deutlicher als Algorithmus erkennbar sind die Bedienungsanleitungen in öffentlichen Telefonzellen. Dort lautet es in einer Bildsprache (Pictogramme):
- Telefonhörer abnehmen!
- Bei Amtszeichen Geldmünze einwerfen!
- Die einzelnen Ziffern des Anschlusses wählen!
- Bei Freizeichen warten, bis Teilnehmer sich meldet, sonst Hörer auflegen und neu beginnen!
- Bei Besetztzeichen Hörer auflegen und neu beginnen!

Auch bei den Fahrkartenautomaten der Deutschen Reichsbahn finden wir einen Algorithmus:

- Fahrtziel in Tabelle suchen!
- Bei betriebsbereitem Gerät die Ziffern des Fahrtziels einzeln über die Berührungstastatur eingeben!
- Fragen des Automaten bezüglich weiterer Einzelheiten der Fahrkarte beantworten!
- Mindestens den angegebenen Preis in Hartgeld einwerfen!
- Druck und Ausgabe von Fahrkarte und eventuellem Wechselgeld abwarten! Diese dann dem Fach entnehmen!

Selbst die Spielregeln von Sport- und Unterhaltungsspielen enthalten häufig Algorithmen, die wir meistens gar nicht als solche betrachten.

Jeder Hobbykoch und -bäcker hat in Form von Koch- und Backbüchern Sammlungen von Algorithmen zur Herstellung von Gerichten bzw. Backwerk im Bücherschrank zu stehen.

■ Strickanleitungen, Modellbauanleitungen und Notenhefte sind weitere Beispiele von Algorithmen, die sich sowohl vom Inhalt als auch von der Beschreibungsform her unterscheiden. Sie sind alle nur für den verständlich und abarbeitbar, der die jeweilige Beschreibungssprache versteht und die Elementaroperationen beherrscht.

Die Ausführenden der beispielhaft aufgezählten Algorithmen sind Menschen mit einem gewissen Erfahrungsschatz sowie vielen Fähigkeiten und Fertigkeiten. In der Informationsverarbeitung sollen aber die Algorithmen durch Maschinen abgearbeitet werden, die bisher nicht über solche Kenntnisse verfügen. Deshalb sind die Anforderungen an Algorithmen, die maschinell abgearbeitet werden sollen, noch umfangreicher.

1.4. Beispiele elementarer Algorithmen

1.4.1. Wertetabellen – ein numerisches Beispiel

1 Für die Zahlen 1,2,3,...,100 sollen die Quadratzahlen und die Quadratwurzeln auf dem Bildschirm des Computers in Form einer Tabelle ausgegeben werden.
Eine Variable X soll die natürlichen Zahlen von 1 bis 100 schrittweise durchlaufen. Auf dem Bildschirm sollen die jeweiligen Werte der Variablen X, das Quadrat X*X und die Quadratwurzel von X in Tabellenform ausgegeben werden.

Bestandteil unserer Programmiersprache ist eine Menge von Funktionen, deren Anwendung das Bearbeiten verschiedener Aufgaben für den Nutzer vereinfacht. Diese Funktionen werden als Standardfunktionen bezeichnet. Es handelt sich dabei um fertige Programme. Zur Berechnung der Quadratwurzel gibt es die Funktion SQR (engl. square root, dtsch. Quadratwurzel), die für ein Argument X die Quadratwurzel mit SQR(X) berechnet.

Das Argument bzw. die Argumente von Standardfunktionen werden grundsätzlich in runde Klammern eingeschlossen (↗ [1]).

Der Rechner soll sich im arbeitsbereiten Zustand mit aufgerufenem BASIC-Interpreter befinden. Wir geben ihm eine ‚ihm verständliche Anweisungsfolge' im Direktmodus ein. Dabei werden die einzelnen Anweisungen durch Doppelpunkt voneinander getrennt, damit der Rechner die einzelnen Befehle erkennen kann.

```
>FOR X=1 TO 100 STEP 1 : PRINT X, X*X, SQR(X) : NEXT X
```

Mit der ENTER-Taste schließen wir die Eingabe ab. Die Abarbeitung der Anweisungsfolge beginnt.
Auf dem Bildschirm erscheint in Sekundenschnelle:

```
1           1           1
2           4           1.41421
3           9           1.73205
4           16          2
5           25          2.23607
.           .           .
.           .           .
.           .           .
97          9409        9.84886
98          9604        9.8995
99          9801        9.94988
100         10000       10
OK
>
```

Da der Bildschirm des Computers nicht gleichzeitig alle 100 Zeilen zeigen kann, sind nur die letzten 22 oder 30 Zeilen zu sehen, je nach Zeilenanzahl des Bildschirms von 24 bzw. 32 Zeilen.
Wollen wir die tabellarische Ausgabe wiederholen lassen, so ist das nicht möglich. Die Anweisungsfolge ist im Computer nicht gespeichert, da er im Direktmodus arbeitet. Eine Wiederholung verlangt daher eine erneute Eingabe der Anweisungsfolge.

```
>FOR X=1 TO 100 STEP 1:PRINT X,X*X,SQR(X):NEXT X
```

Beim Betrachten der Eingabe auf dem Bildschirm bemerken wir, daß die eingegebene Anweisungsfolge länger als eine Bildschirmzeile ist. Der Rechner hat aber die Anweisungsfolge angenommen. Für die von uns verwendete Programmiersprache gilt die Vereinbarung, daß eine Programmzeile *maximal 72 Zeichen* lang sein darf.

Hinweis: Bei zukünftigen Angaben von Programmtexten werden wir aus Platzgründen den Text in der Breite des Buchtextes und nicht nur in der Breite des Bildschirms (40 Zeichen/Zeile) darstellen. Bei Vergleich des Textes im Buch mit den Zeichen auf dem Bildschirm können deshalb Unterschiede auftreten.

Um die Anweisungsfolge im Computer zu speichern, müssen wir also den Programm-Modus anwenden. Den Hinweis für den BASIC-Interpreter (das Übersetzungsprogramm

des Betriebssystems für BASIC), daß man im Programm-Modus arbeiten möchte, gibt man in folgender Weise ein:

Zeilennummer Anweisung,
oder Zeilennummer Anweisungsfolge.

▶ Die Kennzeichnung des Abschlusses einer Zeileneingabe erfolgt durch Betätigen der ENTER-Taste.

Die Eingabe der obigen Anweisungsfolge im Programm-Modus sieht damit so aus:

```
>10 FOR X=1 TO 100 STEP 1   <ENTER>
>15 PRINT X,X*X,SQR(X)      <ENTER>
>20 NEXT X                  <ENTER>
```

Diese drei Anweisungszeilen stellen unser erstes ‚Programm' dar. Es ist im Speicher des Computers gespeichert und kann beliebig oft mit dem Kommando **RUN** gestartet werden.
Wir wollen uns daran gewöhnen, jedes Programm mit END abzuschließen und fügen deshalb noch die folgende Zeile an:

```
99 END
```

Die Vergabe der Zeilennummern ist an bestimmte Vereinbarungen gebunden, da die Abarbeitung der einzelnen Anweisungen eines Programms entsprechend der aufsteigenden Folge der Zeilennummern geschieht. Unter Beachtung dieser Tatsache sind die Zeilennummern bei den genutzten Computern im Bereich von 1 bis 65529 beliebig wählbar.

Soll in unserem Beispiel noch vor PRINT eine weitere Anweisung eingeschoben werden, so kann dies ohne Schwierigkeiten mit den Zeilennummern 11, 12, 13 oder 14 erfolgen. Dabei kann diese Zeile, z.B. mit der Nummer 13, irgendwann zwischen der ersten und der letzten oder auch nach der letzten Zeile geschrieben werden. Die richtige Sortierung nimmt der BASIC-Interpreter vor.
Unser Programm hat offensichtlich noch einige Mängel. So müssen wir uns merken, welche Bedeutung die drei Spalten haben. Die 300 Zahlen werden so schnell auf dem Bildschirm angezeigt, daß ein Abschreiben unmöglich ist. Die Zahlen erscheinen beim Programmstart etwa in der Mitte des Bildschirms, auf dem oben auch noch das Programm steht, das uns jetzt gar nicht mehr interessiert.
Alle diese Dinge lassen sich natürlich verbessern.
Wir fügen deshalb weitere Anweisungen in das Programm ein. Den Bildschirm säubern wir zum Programmbeginn mit

```
1 CLS
```

(**cl**ean = säubern, **s**creen = Schirm).
Einen Tabellenkopf erzeugen wir durch:

```
2 PRINT"-----------------------------------------"
3 PRINT"ZAHL", "Quadratzahl", "Quadratwurzel"
4 PRINT"-----------------------------------------"
```

Die Tabellenwerte werden in ein 12 Zeilen hohes Fenster (window) mittels

```
8 WINDOW 9,20,0,39
```

eingeordnet.
Das verbesserte Programm läßt aber immer noch die 100 Zahlengruppen ohne Unterbrechung über den Bildschirm laufen – wenn auch im festgelegten Fenster.
Starten wir das Programm erneut mit RUN, so stellen wir fest, daß der Tabellenkopf ein zweites Mal in die Zeilen 9–11 geschrieben wird. Das ergibt sich, weil CLS jedesmal das aktuelle Fenster ‚säubert' (löscht), und das ist in diesem Fall immer noch WINDOW 9,20,0,39 – also der Bereich von der 9. bis zur 20. Zeile und von der 0. bis zur 39. Spalte.
Um zu Beginn des Programms immer den gesamten Bildschirm zu löschen und den Tabellenkopf in die Zeilen 0 bis 2 zu schreiben, korrigieren wir die Programmzeile 1 dadurch, daß wir eine neue Zeile mit der Nummer 1 eingeben:

```
 1 WINDOW 0,31,0,39 : CLS
(1 WINDOW 0,23,0,39 : CLS      beim KC 85/1)
```

Diese Anweisungsfolge bewirkt ein Löschen des gesamten Bildschirms. Diese Zeile 1 zeigt, daß auch im Programm-Modus mehrere Anweisungen auf einer Zeile eingegeben werden dürfen. Um eine gute Lesbarkeit zu gewährleisten, wollen wir nur dann mehrere Anweisungen in eine Programmzeile schreiben, wenn sie in einem sehr engen Zusammenhang stehen; wie z. B. der Fensterbefehl und das Löschen dieses Fensters.
Damit man Teile einer Tabelle längere Zeit betrachten oder abschreiben kann, muß die fortlaufende Anzeige durch die PAUSE- (KC 85/1) bzw. STOP-Taste angehalten werden. Mit dem Kommando CONT (continue) kann die Anzeige fortgesetzt werden (beim KC 85/1 gibt es eine Taste CONT).
Bequemer wäre es daher, wenn das Programm so verändert wird, daß jeweils nur eine überschaubare Menge von Zahlengruppen – z. B. zehn – ausgegeben wird und dann der Rechner mit der Fortsetzung des Programms wartet, bis ein Befehl zum Weiterlaufen vom Nutzer erfolgt.
Nach jeweils 10 Zahlen wird in unserem Beispiel eine durch 10 teilbare Zahl für X erreicht. Die Zahlen 10, 20, 30, ..., 90, 100 ergeben bei Division durch 10 eine ganze Zahl, während alle restlichen Zahlen der betrachteten Menge dies nicht tun. Die Standardfunktion INT (integer = ganze Zahl) berechnet mit INT(X) die größte ganze Zahl, die kleiner gleich X ist. So gilt INT(3.14)=3, INT(7)=7 und INT(-3.7)=−4. Die gewünschte Unterbrechung der Ausgabe kann dadurch erreicht werden, daß abgetestet wird, ob die Zählgröße ohne Rest durch 10 teilbar ist oder nicht, d. h. ob X/10 = INT(X/10) wahr oder falsch ist. Wir fügen ins Programm ein:

```
18 IF X/10=INT(X/10) THEN PRINT AT(22,8);"weiter mit CONT":
PAUSE: CLS
```

Wenn die Bedingung X/10=INT(X/10) erfüllt ist, werden alle Anweisungen, die nach ‚THEN' in der Programmzeile stehen, ausgeführt. Erst danach wird die Anweisung in der nächsten Anweisungszeile abgearbeitet. Ist die Bedingung nicht erfüllt, wird die Anweisung auf der nächsten Programmzeile sofort abgearbeitet. Die Erweiterung der PRINT-Anweisung mit AT(22,8) bewirkt die Ausgabe des Textes auf Zeile 22 ab Spalte 8.

Mit diesen Erweiterungen und Änderungen haben wir nun ein Programm erarbeitet, das unsere Zahlen in übersichtlicher Weise auf den Bildschirm schreibt. Nach Eingabe der Kommandofolge CLS : LIST erhalten wir auf dem Bildschirm diese Programmliste:

```
1  WINDOW 0,31,0,39 : CLS
2  PRINT "---------------------------------------"
3  PRINT "ZAHL", "QUADRATZAHL", "QUADRATWURZEL"
4  PRINT "---------------------------------------"
8  WINDOW 9,20,0,39
10 FOR X=1 TO 100 STEP 1
15 PRINT X,X*X, SQR(X)
18 IF X/10=INT(X/10) THEN PRINT AT(22,8); "weiter mit CONT":
PAUSE:CLS
20 NEXT X
99 END
```

Wenden wir uns einem weiteren Beispiel zu.

2 Ab einer wählbaren Zahl A bis zu einer wählbaren Zahl E > A sollen in wählbaren Schrittweiten S die Quadratzahlen und die Quadratwurzeln auf dem Bildschirm des Computers in Form einer Tabelle ausgegeben werden.
Dieses Beispiel ist dem ersten sehr ähnlich. Daher werden wir versuchen, unser bisheriges Programm zu erweitern und zu ändern. Mit einem solchen Programm könnte eine unüberschaubar große Anzahl von unterschiedlichen Tabellen erzeugt werden – je nach dem, was wir als Anfangswert A, als Schrittweite S und als Endwert E eingeben.
Da wir vom ersten Programm ausgehen wollen, holen wir uns den Programmtext mit den Kommandos

```
CLS : LIST
```

auf den Bildschirm. Wir ersetzen dann die Zeile 10 durch

```
10 FOR X= ATO E STEP S
```

und starten dann das Programm mit RUN.
Statt der erwarteten Tabelle erscheinen die Zeilen

```
? SN ERROR IN 10
OK
>
```

auf dem Bildschirm. Es liegt ein Verstoß gegen die Regeln von BASIC vor (SN ERROR → Syntax-Fehler). In unserem Fall besagt diese Meldung:

▶ Variablennamen dürfen keine BASIC-Schlüsselworte enthalten.

Die Variable ATO enthält AT. Das Schlüsselwort ‚AT' haben wir schon kennengelernt (↗ S. 29). Wir müssen deshalb für den Anfangswert der Laufvariablen einen anderen Variab-

lennamen verwenden, z. B. A1. Um unseren Variablennamen A beizubehalten, da dieser Name gut auf die Bedeutung ‚Anfangswert' hinweist, setzen wir zwischen A und das Schlüsselwort TO ein Leerzeichen. Damit ist der syntaktische Fehler beseitigt. Nach dem erneuten Start mit RUN wird zwar kein syntaktischer Fehler mehr angezeigt, aber die Tabelle besteht nur aus einer einzigen Zeile mit den Werten 0, 0, 0. Da wir versäumt haben, den Variablen A, E und S Werte zuzuweisen, haben diese numerischen Variablen standardmäßig durch den BASIC-Interpreter den Wert 0 erhalten. Wir fügen in das Programm Anweisungen zur Eingabe der Anfangswerte ein.

```
5 INPUT "Anfangswert  :"; A
6 INPUT "Endwert      :"; E
7 INPUT "Schrittweite :"; S
```

Bei der Abarbeitung des Programms bewirken diese INPUT-Anweisungen, daß jeweils der in Anführungszeichen gesetzte Text auf einer neuen Bildschirmzeile ausgegeben wird. Der Rechner wartet dann auf eine Eingabe durch den Nutzer.

Bei der Eingabe von Zahlenwerten ist zu beachten, das gebrochene rationale Zahlen nicht mit Dezimalkomma, sondern mit dem Dezimalpunkt einzugeben sind.

Wir geben für A den Wert 1, für E den Wert 100 und für S auch 1 ein. Das Programm arbeitet wie erhofft. Wählen wir aber beispielsweise A=0.3, E=19 und S=0.2, so erfolgt eine fortlaufende Ausgabe aller Gruppen. Das abschnittsweise Ausgeben funktioniert nicht mehr in jedem Fall. Der Grund besteht darin, daß die Laufvariable X, die im bisherigen Programm zur Steuerung der Ausgabe auf dem Bildschirm mittels Test auf Teilbarkeit durch 10 genutzt wurde, nun nicht immer nach 10 Durchläufen eine durch 10 teilbare Zahl ergibt. Für die Ausgabesteuerung führen wir deshalb eine Hilfsvariable H ein, die beim ersten Test in Zeile 18 den Wert 1 haben soll und bei jedem Durchlauf der FOR ... NEXT-Schleife um 1 erhöht wird. Wir ergänzen das Programm noch durch die Ergibtanweisungen (Wertzuweisungen)

```
9  H = 1
19 H = H + 1
```

Bisher haben wir Korrekturen von Programmzeilen so vorgenommen, daß wir die zu korrigierende Zeile durch eine neue Zeile mit gleicher Zeilennummer ersetzt haben. Bei der recht langen Zeile 18 ist der zweimalige Austausch je eines Zeichens vorzunehmen. Wir führen die Korrektur deshalb im Edier-Modus (d.h. unter Verwendung des Korrekturprogramms EDITOR) mit dem Kommando

```
EDIT 18
```

durch.

Nach <ENTER> erscheint die Zeile 18 auf dem Bildschirm und der Kursor steht hinter dem letzten Zeichen der ganzen Zeile. Durch Betätigung der Kursortasten gelangen wir an die Stelle, an der X steht. Dort schreiben wir jetzt ein H hin und wiederholen diesen Vorgang an der zweiten Stelle mit X. Die Korrektur der Zeile wird mit der ENTER-Taste abgeschlossen. Der Rechner bietet jetzt die nächste Zeile zur Korrektur an, also die

Zeile 19. Da eine weitere Korrektur nicht notwendig ist, gelangt man mit der STOP- (KC 85/1) bzw. der BREAK-Taste wieder in den Kommando-Modus. Wir können uns nun das veränderte Programm mit CLS:LIST auf dem Bildschirm ansehen.

```
1  WINDOW 0,31,0,39 : CLS
2  PRINT  "--------------------------------------"
3  PRINT  "ZAHL", "QUADRATZAHL", "QUADRATWURZEL"
4  PRINT  "--------------------------------------"
5  INPUT  "Anfangswert  :";A
6  INPUT  "Endwert      :";E
7  INPUT  "Schrittweite :";S
8  WINDOW 9,20,0,39
9  H = 1
10 FOR X=A TO E STEP S
15 PRINT X, X*X, SQR(X)
18 IF H/10=INT(H/10) THEN PRINT AT(22,8);"weiter mit CONT":
PAUSE:CLS
19 H = H + 1
20 NEXT X
99 END
```

▶ Das Zeichen ‚=' wird in zweifacher Bedeutung im BASIC-Programm benutzt, als Ergibt- und als Gleichheitszeichen.

Die 1. Bedeutung ist in den Zeilen 9, 10 und 19 ersichtlich. Dort steht es in Ergibtanweisungen. Hier bewirkt es als Ergibtzeichen die Berechnung des rechts stehenden Ausdrukkes und die Zuweisung des berechneten Wertes an die links davon stehende Variable. Somit wird in Zeile 19 zur Variablen H der Wert 1 summiert. Man kann diese Zeile mit

H (neu) **ergibt sich aus** H (alt) plus 1

aussprechen.
In der vorhin geänderten Zeile 18 steht das Zeichen ‚=' in seiner zweiten Bedeutung als Gleichheitszeichen innerhalb einer Bedingung. Je nachdem, ob die Bedingung ‚H/10 = INT(H/10)' erfüllt ist oder nicht, werden die Anweisungen auf der Zeile nach THEN abgearbeitet oder übergangen.

Es gibt Programmiersprachen – z. B. ALGOL 60 oder PASCAL –, in denen das Ergibtzeichen in der Form ‚:=' in Ergibtanweisungen bereits im Schriftbild den Bedeutungsunterschied zum Gleichheitszeichen ‚=' in Bedingungen ausweist.

In manchen BASIC-Versionen ist es notwendig, bei Ergibtanweisungen das Schlüsselwort LET nach der Zeilennummer zu benutzen. Es ist dann z. B. zu schreiben

```
9  LET H = 1
19 LET H = H + 1
```

In den BASIC-Versionen der Kleincomputer KC 85/n ist die Angabe von LET erlaubt, aber nicht unbedingt notwendig.

Wir wollen in Zukunft die Unterscheidung zwischen Ergibtanweisung und Bedingung zu-

mindest in der Sprechweise vornehmen. So läßt sich eine Unterscheidung der Ergibtanweisung

19 H = H + 1 **(H ergibt sich aus H + 1)**

zu der durchaus nach IF möglichen, aber nie erfüllbaren Bedingung

H = H + 1 **(H gleich H + 1)**

im Sprechen erreichen.

Da wir entwickelte Programme stets austesten (auf Korrektheit prüfen) müssen, veranlassen wir den Rechner, Tabellen mit Quadratzahlen und Quadratwurzeln zu unterschiedlichen Gruppen (A, E, S) auszugeben. Wenn wir z. B. die Werte (1.2, 7.0, 0.05) wählen, so stoßen wir auf einen unerwarteten Effekt. Die letzte auszugebende Gruppe

```
7            49            2.64575
```

wird nicht erreicht. Auf der Suche nach der Ursache stellen wir fest, daß ab 5.85 der nächste Wert des Arguments immer um 0.00001 größer ist als der erwartete:

```
5.8          33.64         2.40832
5.85         34.2226       2.41868
5.90001      34.8101       2.42899
5.95001      35.4026       2.43926
...
6.95001      48.3026       2.63629
```

Durch die Umwandlung der Dezimalzahlen in die interne binäre Codierung und durch die Rechnung in der internen Darstellung kommt es zu diesem ‚kleinen' Rundungsfehler. Er scheint hier vernachlässigbar klein zu sein, aber er bewirkt immerhin, daß die letzte Gruppe nicht mehr berechnet wird, weil 7.00001 größer als der Schleifenendwert 7 ist. Bei unseren Arbeiten sollten wir beachten, daß solche Fehlereffekte auftreten können.

Wir wollen uns nun überlegen, was an unserem Programm zu verändern ist, um z. B. statt der Tabelle für Quadratzahlen und Quadratwurzeln eine Tabelle für Kubikzahlen und Kubikwurzeln zu erhalten. Nach einiger Überlegung stellen wir fest, daß nur die Programmzeilen 3 und 15 zu verändern sind:

```
3  PRINT "ZAHL", "KUBIKZAHL", "KUBIKWURZEL"
15 PRINT X, X*XX, X↑(1/3)
```

Wir prüfen nach der Änderung das Programm mit dem Anfangswert 1, dem Endwert 23 und der Schrittweite 0.3.

Am Beispiel der neuen Zeile 15 soll eine Fehlerart demonstriert werden, die gerade dem Anfänger häufig unterläuft und deren Entdeckung oft Probleme bereitet. Bei der Eingabe des Ausdruckes „X*X*X" wurde versehentlich ein Multiplikationszeichen nicht mit eingegeben (nur X*XX). Nach dem Start mit RUN gibt der Computer etwas Verblüffendes aus. In der zweiten Spalte steht stets eine 0. Ein syntaktischer Fehler wurde nicht angezeigt. Obwohl ein Multiplikationszeichen vergessen wurde, erkennt der Computer darin keinen syntaktischen Fehler. XX ist für den Rechner ein zulässiger Variablenname. Die Variable XX hat, da ihr bisher in unserem Programm kein Wert zugewiesen wurde, vom Computer den Wert 0 erhalten. Deshalb ist X*XX stets 0. Solche Fehler zu finden ist oft nicht einfach!

Eine Programmiersprache, die den Programmierer zwingen würde, alle im Programm vorkommenden Variablen zu vereinbaren – also neben dem Namen auch die Eigenschaften der Variablen dem Computer mitzuteilen –, könnte während der Übersetzung des Programms eine Kontrolle auf die Zulässigkeit aller vorkommenden Variablen realisieren. Das verwendete BASIC verlangt keine solche Vereinbarung der Variablen.

Um Programme besser lesbar zu machen, ist es günstig, dem Programm einen Vereinbarungsteil voranzustellen. Es ist dadurch auch im Falle eines Fehlers leichter, das Programm zu korrigieren. Solche Vereinbarungen werden nach dem Schlüsselwort REM (remark → Bemerkung) notiert und dadurch vom Interpreter übergangen. Diese REM-Zeilen werden wir an den Anfang eines Programms setzen.
In unserem letzten Programm sind die ersten Zeilennummern aber schon vergeben, so daß wir uns über ein Neunumerieren mit dem Kommando

```
RENUMBER 1,99,100,10
```

Platz schaffen. Dieses Kommando bewirkt, daß die bisherige Zeile 1 jetzt die Zeilennummer 100 erhält und die folgenden Zeilen in Zehnerschritten aufwärts folgen, was sich als günstig erwiesen hat, da bei weiteren Änderungen am Programm neue Anweisungszeilen eingefügt werden können. Wir haben jetzt Platz, um im Programm den Titel, die Namen der Programmierer und den Entstehungszeitpunkt zu vermerken.

```
10  REM KUBIKZAHL-, KUBIKWURZEL- TABELLE
20  REM AUTOREN: Schilling/Toepfer Leipzig/Berlin, Okt. 85
30  REM numerische Variablen:
40  REM     A - Anfangswert
50  REM     E - Endwert
60  REM     S - Schrittweite
70  REM     X - Laufvariable
80  REM     H - Hilfsvariable zum Zaehlen
90  REM ------------------------------------------------
100 WINDOW 0,31,0,39 : CLS
110 PRINT "--------------------------------------"
120 PRINT "ZAHL", "KUBIKZAHL", "KUBIKWURZEL"
130 PRINT "--------------------------------------"
140 INPUT "Anfangswert :"; A
150 INPUT "Endwert     :"; E
160 INPUT "Schrittweite:"; S
170 WINDOW 9,20,0,39
180 H = 1
190 FOR X = A TO E STEP S
200 PRINT X, X*X*X, X↑(1/3)
210 IF H/10 = INT(H/10) THEN PRINT AT(22,8);"weiter mit CONT":
PAUSE:CLS
220 H = H + 1
230 NEXT X
240 END
```

Soll ein größeres Programm im Ganzen eingegeben werden, so kann diese Eingabe durch die automatische Zeilennumerierung mit dem Kommando AUTO unterstützt werden.

Zusammenfassung

Beim Arbeiten mit Kleincomputern unterscheidet man den **Kommando-Modus** und den **Programm-Modus**.

Kommandos werden nach Eingabeende sofort ausgeführt. Wir kennen bisher die Kommandos:

RUN LIST AUTO EDIT RENUMBER CONT.

Mit der Taste **BREAK** kann die Arbeit von RUN, LIST, AUTO und EDIT beendet werden.

Programmzeilen beginnen mit einer Zeilennummer und enthalten eine oder mehrere durch Doppelpunkt getrennte Anweisungen. Programmzeilen werden nach Eingabeende für eine spätere Abarbeitung als Teil eines Programms gespeichert.

Ein **Programm** sollte im allgemeinen in
- **Programmkopf** mit Programmname und Variablenliste und
- **Programmrumpf** mit Anweisungen

gegliedert sein.

Variablennamen (Bezeichner) müssen mit einem Buchstaben beginnen. Bezeichner dürfen keine Schlüsselwörter (Befehlsworte) von BASIC beinhalten. Zur Unterscheidung verschiedener Namen werden nur die ersten zwei Zeichen benutzt.

Wir kennen **Anweisungen**

INPUT, PRINT	– zur Ein- und Ausgabe von Daten
FOR .. TO .. STEP NEXT	– zur Realisierung von Zyklen (Wiederholung, Schleife)
IF ... THEN ...	– zur Realisierung von Bedingungen
END **PAUSE** **CONT**	– zum Beenden, Unterbrechen bzw. Fortsetzen der Abarbeitung von Programmen
WINDOW **CLS**	– Zur Einteilung des Bildschirms bzw. zum Löschen.

Das **Zeichen ‚='** wird in zwei **Bedeutungen** verwendet:
1. als **Gleichheitszeichen** in Bedingungen und
2. als **Ergibtzeichen** in Ergibtanweisungen.

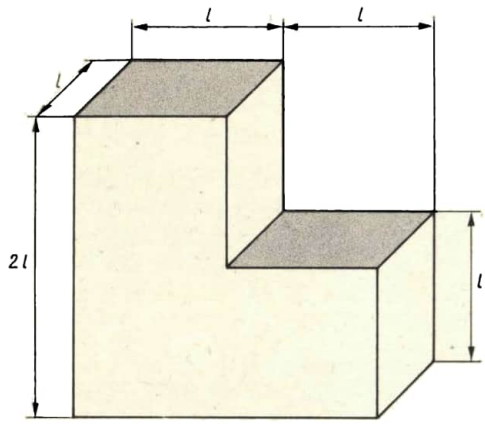
Werte lassen sich auch mit Hilfe von **Standardfunktionen** ermitteln. Wir kennen bisher die Funktionen

INT und **SQR**.

● Aufgaben

1. In den Programmen verwendeten wir X*X bzw. X*X*X statt X↑2 bzw. X↑3. Ersetzen Sie im Programm auf S. 34 X*X*X durch X↑3! Testen Sie beide Programmvarianten und führen Sie einen Genauigkeits- und einen Rechenzeitvergleich (mit der Stoppuhr) durch! Begründen Sie, gestützt auf Ihre Versuchsergebnisse, daß es in Rechenprogrammen sinnvoll ist, Operationen höherer Stufe durch Operationen niederer Stufe zu ersetzen!

2. Eine Fabrik stellt Formstücke laut Skizze her. Für den Versand von n Formstükken müssen Volumen und Masse bestimmt werden (ϱ_{Fe} = 7,5 g/cm³). Entwickeln Sie einen Algorithmus und ein Programm zu diesem Problem!



1.4.2. Namen schreiben – ein nichtnumerisches Beispiel

Computer können außer Zahlen auch andere Zeichen (Buchstaben, Sonderzeichen und Grafiksymbole) verarbeiten. Solche Zeichen oder auch eine Folge solcher Zeichen wird als *Zeichenkette* oder *String* bezeichnet.

1 Entwickeln Sie ein Programm zur Ausgabe Ihres Namens auf dem Bildschirm des Computers für folgende Fälle:

Fall 1: Vorname und Nachname sind in Großbuchstaben in 20 Zeilen untereinander zu schreiben.

Fall 2: Der Nachname ist 20mal in Großbuchstaben hintereinander – jeweils durch genau ein Leerzeichen getrennt – auszugeben.

Fall 3: Der Nachname ist 20mal in Großbuchstaben zu schreiben. Die Namen beginnen jeweils in den Spalten 0, 13 und 26 (Standard-Tabulatorpositionen).

Fall 4: Der Nachname soll auf jeder Zeile einmal geschrieben werden und zwar so, daß er von Zeile zu Zeile um 1 Spalte nach rechts versetzt wird. Wenn er am rechten Bildrand angekommen ist, soll er wieder jeweils um eine Spalte nach links versetzt werden, bis er den linken Bildschirmrand erreicht. Diese Ausgabeform ist neunmal zu wiederholen.

Fall 5: Der Nachname soll – zufällig positioniert – zehnmal auf dem Bildschirm erscheinen. Nach jeder Anzeige soll er wieder gelöscht werden.

Alle fünf Teilaufgaben haben etwas mit der Art und Weise der Ausgabe einer Zeichenkette auf dem Bildschirm zu tun. Vom Grundaufbau her ähneln sich die Programme. Wie bei den schon besprochenen Beispielen sind auch hier bestimmte Handlungen mehrmals auszuführen und zwar 10 bzw. 20 mal. Wir können die FOR-NEXT-Schleife anwenden.
Bevor wir die Namen in der geforderten Weise anzeigen lassen können, müssen sie erst eingegeben werden. Da die Programmnutzer meist unterschiedliche Namen besitzen, die meist auch unterschiedlich lang sind, benötigen wir Variablen zur Abspeicherung der Namen und der Länge der jeweiligen Zeichenkette. Die Variablen nehmen nicht nur Ziffern als Werte an. Sie beinhalten beliebige Zeichen aus dem Zeichenvorrat des Computers – bei Namen in erster Linie Buchstaben. Die Namen der Zeichenkettenvariablen sind durch ein ‚$'-Zeichen am Ende des Namens gekennzeichnet.

```
10  REM  Namen schreiben
20  REM  Erstes Programm zur Zeichenketten-Verarbeitung
30  REM  Darstellung der unterschiedlichen
         Ausgabemoeglichkeiten
40  REM  Autoren: Schilling/Toepfer Leipzig/Berlin, Mai 85
50  REM  numerische Variable:
60  REM  I  - Laufvariable
70  REM  Zeichenkettenvariablen:
80  REM  VN$ - Vorname
90  REM  NN$ - Nachname
100 WINDOW 0,31,0,39 : CLS
110 INPUT "NACHNAME, VORNAME:"; NN$, VN$
120 REM
200 REM Vorname Nachname - 20mal untereinander
210 WINDOW : CLS
220 FOR  I = 1 TO 20 STEP 1
230      PRINT VN$; " "; NN$
240 NEXT I
250 END
```

▶ Die Angabe der Schrittweite STEP 1 kann in der Schleifenanweisung entfallen, da BASIC dies als Standardfestlegung verwendet.

Um die Schachtelungstiefe sofort beim Blick auf ein Programm zu erkennen, sollten Anweisungen im Schleifenkörper eingerückt werden. Das ist allerdings nur möglich, solange die maximale Zeichenzahl je Zeile nicht überschritten wird.

Das Semikolon in der PRINT-Anweisung bewirkt, daß das folgende Element unmittelbar an das Vorhergehende auf dem Bildschirm anschließt. Damit wir den Vornamen gut vom Nachnamen unterscheiden können, wird die Ausgabe eines Leerzeichens zwischen beiden Namen eingefügt. Damit ist der erste Fall gelöst.

Der zweite Fall verlangt eine kleine Änderung in der Zeile 230.

```
230 PRINT NN$ + " ";
```

Die Zeichenkette NN$ wird durch ein Leerzeichen ergänzt, damit die Nachnamen nicht unmittelbar aufeinander folgen. Das Semikolon zum Abschluß der PRINT-Anweisung bewirkt, daß die Ausgabe der Werte der nächsten PRINT-Anweisung unmittelbar im Anschluß an das gerade Ausgegebene erfolgt.
Ein Komma am Ende einer PRINT-Anweisung verhindert ebenfalls den Zeilenvorschub.
Unter Beachtung dieser Festlegung hat die Zeile 230 im dritten Fall folgendes Aussehen:

```
230 PRINT NN$,
```

Steht ein Komma zwischen den Elementen einer Datenliste, so werden die Werte auf dem Bildschirm spaltenweise (3 Spalten a 13 Anzeigestellen je Bildschirmzeile) angezeigt.

Das Komma bewirkt ein Setzen des Anfangszeigers für ein Ausgabefeld (Tabulator) auf die 0., dann die 13. und die 26. Spalte einer 40spaltigen Einteilung der Bildschirmzeile.
Für die Lösung des vierten Falls löschen wir die Anweisungszeilen 200 bis 250 mit dem neuen Kommando

```
DELETE 200, 250
```

Der Nachname soll von Zeile zu Zeile jeweils um eine Spalte nach rechts versetzt angezeigt werden. Der Funktionsaufruf

```
LEN(NN$)
```

bestimmt die Länge (length) der Zeichenkette NN$.

LEN ist eine Zeichenkettenstandardfunktion, so daß das Argument – hier die Zeichenkette NN$ – in runde Klammern einzuschließen ist.

Mit der Anweisung PRINT TAB(I); NN$ wird der Tabulator in die I. Spalte der Ausgabezeile gesetzt. NN$ wird von dort ausgegeben.

Es ist möglich, mit zwei FOR-NEXT-Schleifen, die aufeinander folgen, das Wandern von links nach rechts und das anschließende Wandern von rechts nach links zu bewirken. Da diese ‚Bewegung' neunmal zu wiederholen ist, stellen diese beiden Schleifen das Innere einer weiteren, dritten FOR-NEXT-Schleife dar. Die Eingabe der folgenden Zeilen wird durch

```
AUTO 200
```

eröffnet, so daß die Zeilennummern ab 200 automatisch in Zehnerschritten vorgegeben werden.

```
200 REM Name wandert im Zickzack 10mal ueber den Bildschirm
210 WINDOW : CLS
220 FOR J = 1 TO 10
230     FOR I = 1 TO 39 - LEN(NN$)
240         PRINT TAB(I); NN$
250     NEXT I
260     FOR I = 38 - LEN(NN$) TO 0 STEP -1
270         PRINT TAB(I); NN$
280     NEXT I
290 NEXT J
300 END
```

Hier ist etwas Typisches für rationelles Arbeiten in der Programmierung passiert. Ein Programmgerüst ist erhalten geblieben, und nur eine logisch zusammengehörige Anweisungsfolge ist vollständig durch eine andere ersetzt worden. Später werden wir auf solche Arbeitsweisen näher eingehen. Sie werden uns helfen, weniger leistungsfähige Programmteile durch bessere zu ersetzen.

Die beiden inneren FOR-NEXT-Schleifen unseres Programms verwenden I als Laufvariable – einmal vorwärts und einmal rückwärts zählend. Das ist erlaubt, weil die Schleifen hintereinander liegen. Sind aber Schleifen in Schleifen geschachtelt, so müssen unter-

schiedliche Laufvariablen verwendet werden – zum Beispiel I und J. Die Variable J sollte in der Variablenaufzählung am Programmanfang genannt werden. Zeile 60 lautet dann:

```
60 REM I,J - Laufvariablen
```

Der Fall 5 verlangt die zufällige Positionierung des Namens auf dem Bildschirm. Der Computer kann mit der Standardfunktion RND(X) eine Folge von Pseudozufallszahlen aus dem Intervall <0,1) produzieren. Unter dieser Voraussetzung wird das Programm ab Anweisungszeile 200 mit AUTO 200 wieder neu gestaltet:

```
200 REM 10mal zufaellige Positionierung des Nachnamens
210 WINDOW : CLS
220 L = LEN(NN$)
230 FOR I = 1 TO 10
240     Z = RND(1)*24
250     S = RND(1)*(40-L)
260     PRINT AT(Z,S); NN$
270     PAUSE (20)
280     PRINT AT(Z, S); STRING$(L," ")
290 NEXT I
300 END
```

Die Spaltenanzahl wird durch den ganzzahligen Anteil von RND(1)*(40–L) so bestimmt, daß der Name nicht über den rechten Bildschirmrand hinausgeschrieben wird! Da der angezeigte Name wieder zu löschen ist, überschreiben wir ihn in der Zeile 280 mit einer Zeichenkette, die aus L mal dem Leerzeichen besteht. Vor Zeile 280 haben wir mit der Anweisung

```
PAUSE (20)
```

dafür gesorgt, daß eine kurze Zeit (20 * 0,1 s) der angezeigte Name auf dem Bildschirm zu sehen ist. Die Anweisungszeilen

```
55 REM   Z,S - Zeilen- bzw. Spaltenvariable
58 REM   L   - Laenge des Namens NN$
```

komplettieren das Programm.

```
10  REM Namen schreiben
20  REM Programme zur Zeichenkettenverarbeitung
30  REM Darstellung der unterschiedlichen Ausgabemoeglichkeiten
40  REM Autoren: Schilling/Toepfer, Leipzig/Berlin, Mai 86
50  REM numerische Variablen:
55  REM    Z,S - Zeilen- bzw. Spaltenvariable
58  REM    L   - Laenge des Namens NN$
60  REM    I   - Laufvariable
70  REM Zeichenkettenvariablen:
```

```
80  REM     VN$ - Vorname
90  REM     NN$ - Nachname
100 WINDOW : CLS
110 INPUT "Nachname, Vorname:";NN$,VN$
120 REM
200 REM 10-mal zufaellige Positionierung des Nachnamens
210 CLS
220 L = LEN(NN$)
230 FOR I = 1 TO 10
240     Z = RND(1)*24
250     S = RND(1)*(40-L)
260     PRINT AT(Z,S);NN$
270     PAUSE (20)
280     PRINT AT(Z,S);STRING$(L," ")
290 NEXT I
300 END
```

Zusammenfassung

Computer können mit **Zahlen** und mit **Zeichenketten** umgehen. Zeichenketten sind Folgen von beliebigen Zeichen aus dem Zeichenvorrat des Computers.

Zeichenkettenkonstanten werden in Anführungszeichen eingeschlossen. Eine **Zeichenkettenvariable** wird durch das **$-Zeichen** am Ende des Variablennamens gekennzeichnet.

Auch für Zeichenketten gibt es Standardfunktionen, zum Beispiel **LEN** und **STRING$**.

Die Ausgabe auf dem Bildschirm kann mit **PRINT** realisiert werden. Die Ausgabeform wird durch das Fehlen bzw. Vorhandensein der Trennzeichen Komma oder Semikolon bestimmt.

Mit dem Kommando **DELETE n1 [,n2]** kann man die Zeile n1 [oder die Zeilen von n1 bis n2] in einem BASIC-Programm löschen.

Bestimmte Positionierungen des Ausgabebeginns erzielt man mit **PRINT TAB** und mit **PRINT AT**.

● Aufgaben

1. Benutzen Sie das Programm zum Fall 4, um Muster auf dem Bildschirm zu erzeugen! Probieren Sie vor allem Kombinationen von Sonderzeichen des Zeichenvorrates!

2. Die Güte einer Zufallszahlenfunktion ist um so höher, je unsystematischer sie die Zufallszahlen in einem vorgegebenen Bereich erzeugt. Ein einfaches Verfahren zur Güteprüfung ermittelt, wie weit der Mittelwert einer erzeugten Zufallszahlenreihe vom idealen Mittelwert dieser Reihe abweicht. Ein Beispiel soll dieses Verfahren demonstrieren.

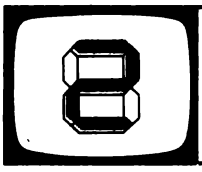
Bereich: natürliche Zahlen von 1 bis 6
Anzahl: 600
idealer Mittelwert ergibt sich zu
(100 * 1 + 100 * 2 + 100 * 3 + 100 * 4 + 100 * 5 + 100 * 6)/600 = 3,5
Entwickeln Sie ein Programm, das die Zufallszahlenfunktion des Kleincomputers nach diesem Verfahren prüft!

3. Entwickeln Sie ein Programm, das die angegebenen Sternchenmuster auf dem Bildschirm anzeigt!

a)
```
*
**
***
****
***
**
*
```

b)
```
*       *
 *     *
  *   *
   * *
  *   *
 *     *
*       *
```

4. Ein Spielautomat hat drei Räder, die sich bei Start des Automaten unabhängig von einander in Bewegung setzen. Jedes Rad hat Felder für die Ziffern 0 bis 5.
Zeigen alle 3 Räder die gleiche Ziffer, so bedeutet dies einen Hauptgewinn. Zeigen alle Räder unterschiedliche Ziffern, so bedeutet dies die Rückgabe des Einsatzes. In allen anderen Fällen hat der Spieler verloren.
Erstellen Sie ein Programm, das diesen Spielautomaten durch Anwendung des Zufallszahlengenerators simuliert! Die Ziffern der 3 Räder sollen bei Stillstand in der Mitte des Bildschirms in den Spalten 15, 20 und 25 angezeigt werden. Drei Zeilen tiefer soll die Räderstellung durch „Hauptgewinn", „Einsatzgewinn" oder „Verlust" in der Zeilenmitte kommentiert werden.



Strukturierte Programmierung und Kleincomputer

Es gibt die erstaunliche Möglichkeit,
daß man einen Gegenstand mathematisch beherrschen kann,
ohne den Witz der Sache wirklich erfaßt zu haben.
(Albert Einstein)

In den Anfängen der Computerprogrammierung bereitete die begrenzte Leistungsfähigkeit der Maschinen besondere Probleme. Es war deshalb Aufgabe der Programmierer, Algorithmen in der Sprache der jeweiligen Maschine so effektiv wie möglich zu formulieren. Raffinierte Techniken und Tricks waren erforderlich, um Programme auf den damaligen Computern zum Arbeiten zu bringen. Die Leistungssteigerung der EDV-Anlagen zu Beginn der sechziger Jahre führte im wesentlichen nur dazu, daß neue Aufgabenstellungen mit alten Programmiermethoden gelöst wurden. Die gewachsene Komplexität der Aufgaben führte zu einer Vielzahl von Programmfehlern bzw. -mängeln, die sich nachteilig auf die Programmentwicklung auswirkten.
Die drastische Leistungssteigerung der Computer ab Beginn der siebziger Jahre – bedingt durch den konsequenten Einsatz der Mikroelektronik – und der wachsende Umfang an Programmieraufgaben brachte die Erkenntnis, daß es weniger darauf ankommt, die Grenzen des Computers durch Retten von Bits und Mikrosekunden auszuschöpfen, sondern daß es erforderlich wurde, Programme auf ganz neue Art herzustellen.
Als eine Methode dieser neuen Art der Programmentwicklung wurde Mitte der siebziger Jahre die Methode der **„strukturierten Programmierung"** vorgestellt.
Der Begriff „strukturiertes Programmieren" wurde geprägt, um auszudrücken, daß die Probleme systematisch und strukturiert bearbeitet werden sollen.
Der Holländer E. W. Dijkstra veröffentlichte 1972 auf der Grundlage von Untersuchungen und eigenen Erfahrungen die Grundgedanken der strukturierten Programmierung als Methodik zur Erstellung von Programmen, die

- leicht zu verstehen sind und dabei schon eine qualitative Beurteilung erlauben,
- einfach zu kontrollieren sind,
- eine leichte Fehlersuche ermöglichen,
- mit geringem Aufwand geänderten Aufgabenstellungen angepaßt werden können.

Man kann den Grundgedanken der strukturierten Programmierung durch das Motto ausdrücken: Erst denken, dann tippen!

2.1. Problemlösen in der Informatik

Für ein gegebenes Problem soll ein Lösungsverfahren entwickelt und mit formalen Mitteln so aufbereitet werden, daß es im Endeffekt in einer computerverständlichen Form vorliegt. Bei den meisten heute verwendeten Computern und Programmiersprachen bedeutet dies ein Problemlösen unter Verwendung von Algorithmen.

Es gibt erste Ansätze, Probleme nicht nur algorithmisch, sondern mit logischen Mitteln zu lösen. Der Mensch entwickelt dabei keine kleinschrittige Folge von Computerbefehlen, sondern formuliert eine Problemlösung in der Sprache der Logik (etwa in PROLOG, ↗ S. 279). Der Rechner gelangt aufgrund der gemachten Angaben zu Lösungen, ohne daß der Programmierer den Lösungsweg bis ins Detail vorschreibt, wie es bei algorithmischen Programmen notwendig ist.

Das Lösen von Problemen ist im allgemeinen eine sehr umfangreiche Aufgabe. Es ist deshalb möglichst zu fordern, Lösungsverfahren für ganze Gruppen ähnlich gelagerter Aufgaben (Klasse von Aufgaben) zu finden. Dadurch ergeben sich sowohl hinsichtlich der Probleme als auch der Lösungsverfahren gewisse Einschränkungen:

- Es gibt Probleme, die prinzipiell nicht mit dem Computer gelöst werden können.
- Es gibt Probleme, die mit den heute verfügbaren Computern noch nicht lösbar sind, weil die benötigte Rechenzeit für eine Lösung zu groß ist.
- Es gibt Probleme, von denen man nicht sofort weiß, ob sich das gefundene Lösungsverfahren für die Bearbeitung durch Computer eignet.

Das Suchen nach neuen Lösungsverfahren kann bisher noch nicht vom Computer selbst übernommen werden. Er kann bestenfalls aufgrund vorher eingebrachter Informationen und logischer Regeln bei diesem Suchen behilflich sein, wie es Expertensysteme in speziellen Aufgaben (Fehlersuche in automatischen Getrieben, Diagnosesysteme für spezielle Gebiete der Medizin) schon realisieren.

Bei den Expertensystemen handelt es sich derzeit um Software-Lösungen. Sie bestehen aus:
- einer Wissensbasis, die zu einem eng begrenzten Fachgebiet das Spezialwissen von Experten enthält,
- einer softwaretechnisch realisierten „Maschine" zum Ziehen von logischen Schlußfolgerungen nach festgelegter Problemlösungsmethode auf der Grundlage der Wissensbasis,
- einer geeigneten Nutzerschnittstelle (Programme zur Mensch-Maschine-Kommunikation) zur „einfachen" Arbeit mit dem System.

Die umfassende Nutzung von Expertensystemen ist ein wesentliches Merkmal der zur Zeit in Entwicklung befindlichen Gerätetechnik der „Fünften Rechnergeneration".

Das Lösen von Problemen ist ein abstrakter und kreativer Prozeß, bei dem persönliche Erfahrungen und Ideen notwendig sind.

Im Laufe der Entwicklung der Informatik hat sich eine gewisse Systematik des Problemlösens als günstig herausgestellt. Ausgangspunkt ist dabei ein gestelltes Problem, das Ziel ist die Erstellung eines Algorithmus zur Abarbeitung durch einen Computer. Dazwischen liegen im allgemeinen mehrere Bearbeitungsschritte:
- das Überprüfen der Problemstellung,
- der Entwurf eines Lösungsplans,
- das Erarbeiten des Lösungsweges,
- das Prüfen der Korrektheit des Lösungsweges,
- das Verbessern des Lösungsweges,
- die Codierung des Algorithmus und
- die Dokumentation des Problemlösungsweges.

Das Überprüfen der Problemstellung

Die erfolgreiche Bearbeitung eines Problems setzt voraus, daß die Aufgabenstellung vom Bearbeiter verstanden wird und daß alle notwendigen Informationen vorliegen. Dazu gehört, daß alle Eingabe- und Ausgabegrößen sowie deren grundsätzliche Beziehungen aus

der Problemstellung erkannt werden können. Sollten diese Informationen nicht eindeutig vorliegen, so müssen weitere Informationen eingeholt werden.

Der Entwurf eines Lösungsplans

Im zweiten Schritt, dem Entwurf eines Lösungsplans, wird untersucht, ob gleiche oder ähnliche Probleme schon bearbeitet wurden und ob diese Lösungen verwendbar sind. Ist das nicht der Fall, so sollte man versuchen, das Gesamtproblem in kleinere Teilprobleme zu zerlegen und diese zu lösen. Aus den Lösungen der Teilprobleme muß man dann zu einer Lösung des Gesamtproblems kommen. Diese Zerlegung kann für alle oder einzelne Teilprobleme weitergeführt werden, so daß man einen Zerlegungsbaum des Gesamtproblems erhält. Diese Vorgehensweise wird als *‚schrittweise Verfeinerung'* (top-down-Verfahren) bezeichnet (Abb. 2.1/1).

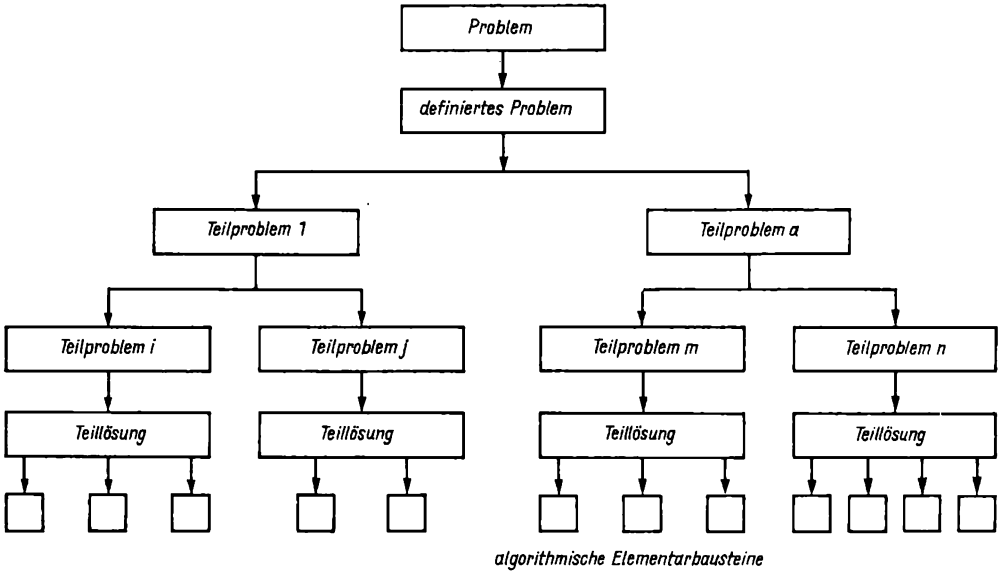


Abb. 2.1/1: Zerlegungsbaum der Problemlösung

Bei der Lösung mancher Probleme ist auch eine andere Arbeitsweise sinnvoll, bei der man von den algorithmischen Elementarbausteinen, die schon vorhanden sind, ausgehend durch die *‚schrittweise Verallgemeinerung'* (bottom-up-Verfahren) den Baum bis zur kompletten Lösung zusammensetzt. Auch Kombinationen beider Verfahren sind natürlich möglich.

Das Erarbeiten des Lösungsweges

Aus dem Zerlegungsbaum ist ersichtlich, daß der dritte Schritt eine weitere Aufgliederung des Problems beinhaltet. Dabei wird mittels schrittweiser Verfeinerung eine ständig steigende Konkretisierung der Handlungsabläufe durchgeführt, bis eine Formulierung in

Form von algorithmischen Elementarbausteinen vorliegt. In den verschiedenen Stadien dieses dritten Schrittes des Problemlösungsprozesses kann es sich herausstellen, daß Fehler gemacht wurden, so daß es zu einer Verbesserung der bisherigen oder zu einer völlig anderen Problemlösung kommen kann.
In dieser Bearbeitungsstufe muß der Lösungsweg so exakt formuliert sein, daß durch die Bearbeitung mit einem Computer aus den Eingabeinformationen die Ausgabeinformationen erzeugt werden können. Deshalb sollte der Lösungsweg in einer unmißverständlichen Darstellungsform vorliegen. Eine umgangssprachliche Fassung ist meist ungenau und mehrdeutig. Betrachten wir nur die Mehrdeutigkeiten des Satzes „Ich gehe zur Bank", so wird die Ungenauigkeit der umgangssprachlichen Formulierung deutlich, denn nur aus diesem Satz ist nicht zu erkennen, ob das Kreditinstitut, die Parkbank oder die Sandbank gemeint ist.

Das Prüfen der Korrektheit des Lösungsweges

Liegt im Ergebnis der vorangegangenen Tätigkeiten ein Lösungsweg vor, so folgt nun die Prüfung auf seine Korrektheit. Es gilt festzustellen, ob das Problem auf diese Weise vollständig und richtig gelöst werden kann. Empirische Tests mit unterschiedlichen Eingabewerten sollten alle Teile des Algorithmus überprüfen. Die Überprüfung kann Fehler ergeben, die zu beseitigen sind. Dabei ist es möglich, daß auch Rückwirkungen auf vorhergehende Bearbeitungsschritte auftreten.

Das Verbessern des Lösungsweges

Da es zur Lösung einer Aufgabe meist mehrere Varianten gibt, sollte man sich die Frage nach einer möglichen Verbesserung des Lösungsweges stellen. Dabei ist zu untersuchen, ob man dasselbe Ergebnis vielleicht schon mit einem geringeren Aufwand erzielen kann. Dies kann in Teilen oder insgesamt zu neuen Lösungswegen führen.

Die Codierung des Algorithmus

Mit dem nächsten Schritt wird die Verbindung zum Computer hergestellt. Es erfolgt die Codierung des Lösungsweges in einer dem Computer verständlichen Programmiersprache.

Aufgrund von Unzulänglichkeiten in der Programmiersprache und/oder bei dem verwendeten Computer können sich Änderungen zum bisher erarbeiteten Lösungsweg ergeben. Je nach Art dieser Änderungen ist ein erneutes Abarbeiten vorhergehender Schritte notwendig. Die Programmiersprache BASIC kann einige Grundelemente der strukturierten Programmierung nicht durch entsprechende Sprachelemente ausdrücken, so daß man Ersatzkonstruktionen anwenden muß. Deshalb wird dieser Schritt des Problemlösens im weiteren besonders ausführlich behandelt.

Danach wird das Programm getestet. Der Test muß ergeben, daß das Programm vollständig den Regeln der Sprache entspricht und die gestellte Aufgabe vollständig und sicher gelöst wird.

Selbst die umfangreichsten Tests sind keine Gewähr für die vollständige Fehlerlosigkeit eines erstellten Programms. Es gibt gegenwärtig noch keine Möglichkeit, die Korrektheit eines Programms automatisch zu beweisen. Durch Tests kann die Fehlerfreiheit nur für die verwendeten Daten demonstriert werden.

Die Dokumentation des Problemlösungsweges

Hierbei handelt es sich um einen kontinuierlichen Vorgang, der während des Erarbeitens des Lösungsweges zu realisieren ist. Die vollständige Dokumentation muß Auskunft über die Aufgabenstellung, den Lösungsalgorithmus und den Programmtext geben sowie Hinweise für den späteren Nutzer enthalten. Auf die einzelnen Bestandteile einer vollständigen Dokumentation wird später hingewiesen.

Die vorgestellten Schritte beim Problemlösen haben sich bei vielen großen Projekten bewährt. Deshalb wird versucht, den ganzen Prozeß mit Computern zu unterstützen und durch Computer zu kontrollieren.

Bei Problemen, die leicht zu überschauen sind, kann man mehrere Schritte zusammenfassen. Dabei sollte man sich aber an diese Arbeitsweise halten, weil sie zur Verringerung von Fehlern beiträgt und weil durch sie gemachte Fehler schnell erkannt werden.

2.2. Darstellungsmöglichkeiten für Algorithmen

Erarbeitete Lösungsschritte werden notiert, um Unterlagen für die weitere Bearbeitung zu haben. Im Laufe der Entwicklung der Informatik haben sich verschiedene Darstellungsmöglichkeiten (Notationsformen) für Algorithmen herausgebildet. Wesentliche Notationsformen sind

- die verbal formalisierte Notation,
- der Programmablaufplan,
- das Struktogramm.

▶ Die verbal formalisierte Notation ist eine kurzgefaßte Beschreibungsform auf der Grundlage der Umgangssprache.

Dabei werden einzelne Arbeitsschritte stichwortartig unter Verwendung festgelegter *Schlüsselwörter* notiert. Die Reihenfolge der einzelnen Arbeitsschritte legt im allgemeinen auch die Reihenfolge der Abarbeitung fest.

▶ Den Programmablaufplan (auch PAP genannt) gibt es als Programmlinien- und als Sinnbild- (auch Kästchen-) Darstellung.

Bei der *Programmliniendarstellung* werden an eine Pfeillinie die Operationen in ihrer Reihenfolge eingetragen. Einige spezielle Symbole verdeutlichen Verzweigungen und Zusammenführungen. Bei der *Sinnbilddarstellung* sind für bestimmte Operationen spezielle Symbole festgelegt. Diese Symbole werden durch Pfeile verbundenen, die die Reihenfolge der Abarbeitung der einzelnen Operationen angeben.

▶ Das Struktogramm ist eine grafische Notationsform für Algorithmen.

Diese Notationsform entstand als Folge der breiten Anwendung der strukturierten Programmierung. Der Gesamtalgorithmus wird dabei als Rechteck dargestellt, und Teilalgorithmen sowie Elementaroperationen sind darin grafisch und textlich untergebracht.

Eine ausführliche Gegenüberstellung der drei genannten Notationsformen erfolgt erst bei der Behandlung der einzelnen Algorithmenstrukturen. Einige Beispiele sollen aber schon jetzt einen ersten Eindruck vermitteln, wobei nur die Algorithmen dargestellt sind, nicht aber ihre Entwicklung.

1 Ein Schüler möchte die Zinsen errechnen, die er bis zum Jahresende erzielt, wenn er das Geld von seinen Altstoffsammlungen regelmäßig auf sein Sparkonto einzahlt.

1. verbal formalisierte Notation

```
ZINSEN
  BEGINN
    Ausgabe einer Überschrift
    Eingabe Betrag, Zinssatz, Laufzeit in Tagen
    Zinsen=(Betrag*Zinssatz*Laufzeit)/(100*360)
    Ausgabe Zinsen
ENDE ZINSEN
```

2. Programmablaufplan

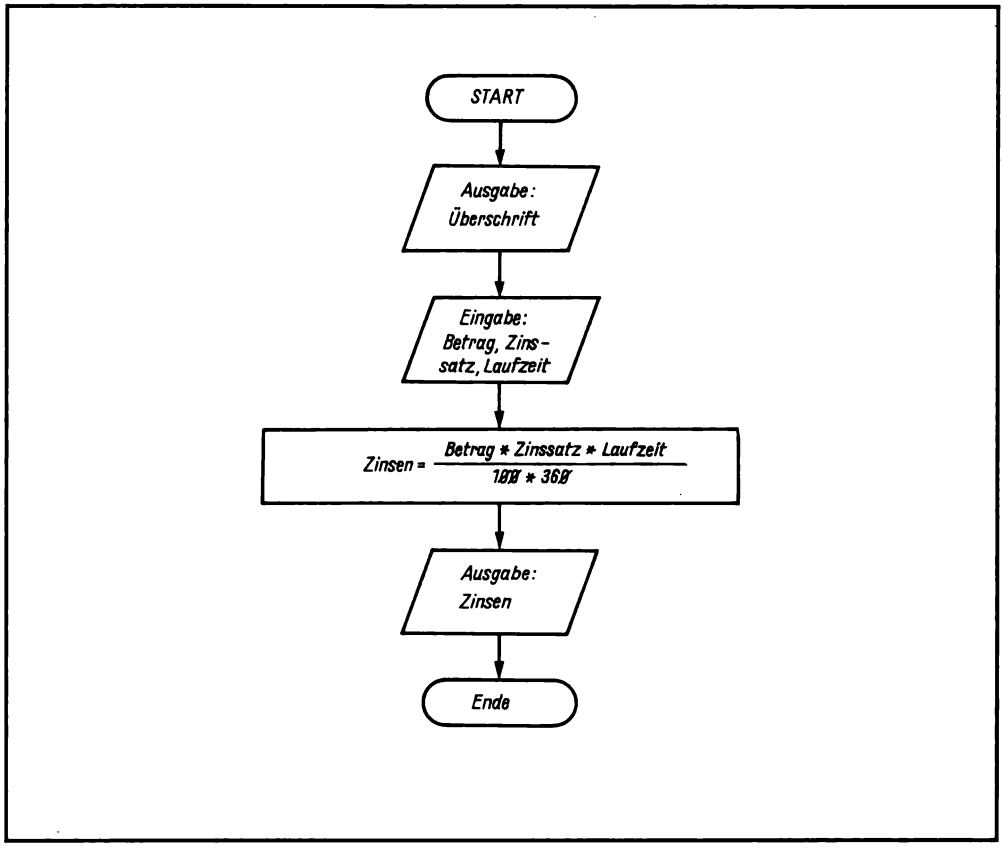


Abb. 2.2/1: Programmablaufplan „Zinsen"

3. Struktogramm

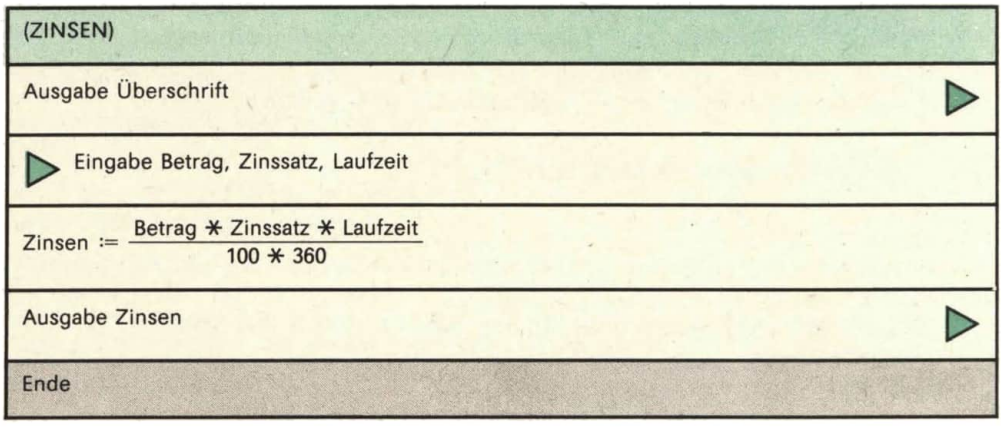


Abb. 2.2/2: Struktogramm „Zinsen"

2 Aus unserer Kindheit ist ein Spiel bekannt, in dem durch wiederholtes Aufsagen eines Abzählreims aus einer Menge von Kindern ein Kind ermittelt wird, das z. B. beim Verstekken der Sucher ist.

1. verbal formalisierte Notation

```
ABZAEHLREIM
BEGINN
  Kinder im Kreis aufstellen
  Kind als 1. Reimaufsager bestimmen
  WIEDERHOLE
    WIEDERHOLE
      Reimaufsager spricht nächste Reimsilbe aus und zeigt auf das nächste Kind
    BIS letzte Reimsilbe ausgesprochen ist
    Das Kind, auf das bei letzter Reimsilbe gezeigt wurde, scheidet aus
    Das im Kreis folgende Kind ist nächster Reimaufsager
  BIS nur noch ein Kind im ‚Kreis'
  letztes Kind ist Sucher beim Verstecken
ENDE ABZAEHLREIM
```

2. Programmablaufplan

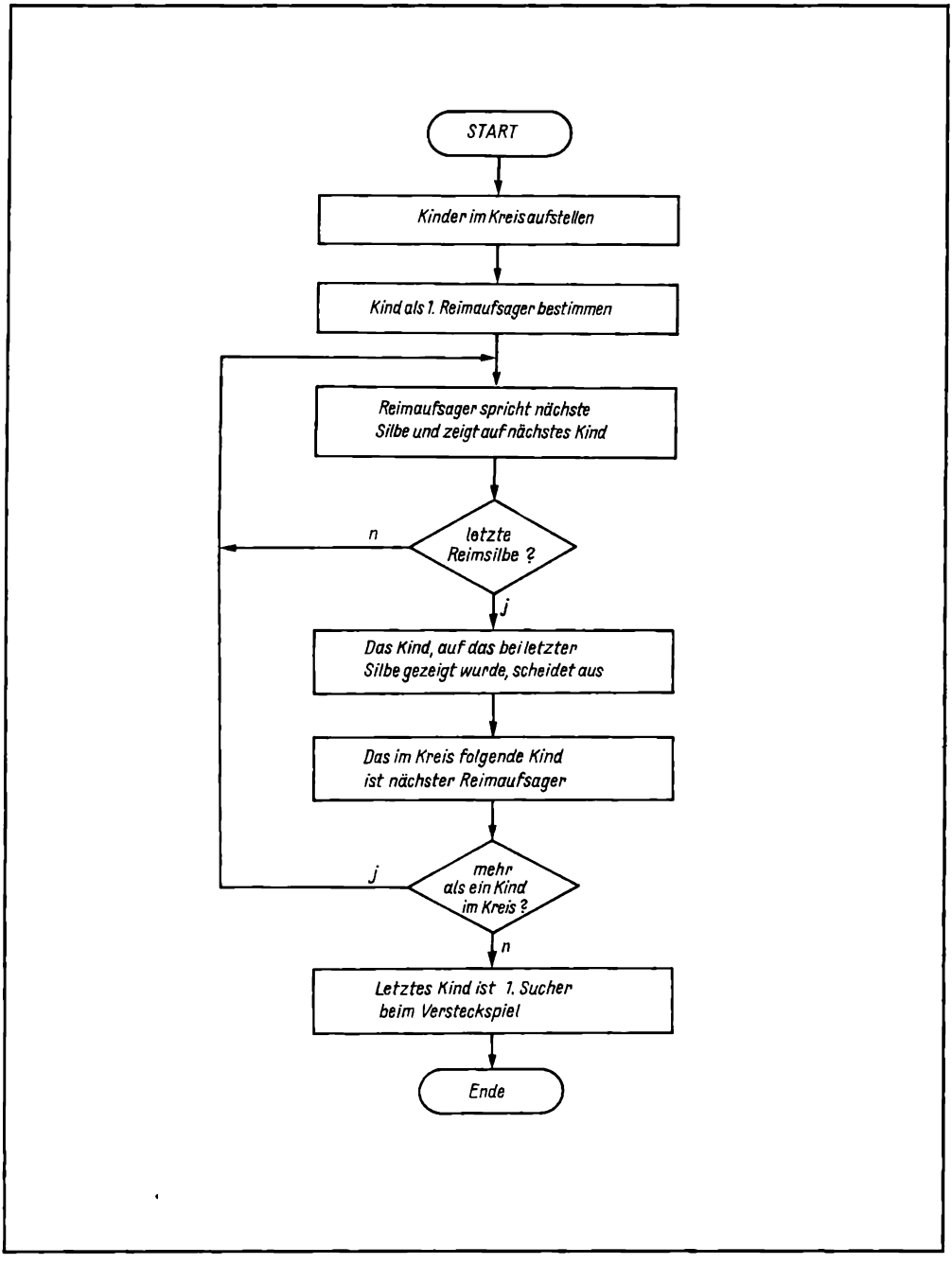


Abb. 2.2/3: Programmablaufplan „Abzählreim"

3. Struktogramm

<table>
<tr><td colspan="3">(ABZÄHLREIM)</td></tr>
<tr><td colspan="3">Kinder im Kreis aufstellen</td></tr>
<tr><td colspan="3">Kind als 1. Reimaufsager bestimmen</td></tr>
<tr><td rowspan="4">WIEDER-
HOLE</td><td rowspan="2">WIEDER-
HOLE</td><td>Reimaufsager spricht nächste Reimsilbe und zeigt auf das nächste Kind</td></tr>
<tr><td>BIS letzte Reimsilbe gesagt ist</td></tr>
<tr><td colspan="2">Ausscheiden des Kindes, auf das bei letzter Silbe gezeigt wurde</td></tr>
<tr><td colspan="2">Im Kreis folgendes Kind ist nächster Reimaufsager</td></tr>
<tr><td colspan="3">BIS nur noch ein Kind im Kreis</td></tr>
<tr><td colspan="3">Letztes Kind ist 1. Sucher beim Verstecken</td></tr>
<tr><td colspan="3">Ende</td></tr>
</table>

Abb. 2.2/4: Struktogramm „Abzählreim"

● Aufgaben

1. Welche Darstellungsformen für Algorithmen kennen Sie?

2. Erstellen Sie in Stichworten Algorithmen für
   a) Kauf eines Getränks am Getränkeautomaten,
   b) Kaffeezubereitung mittels Kaffeemaschine,
   c) Teezubereitung mittels Elektroherd!

3. Erstellen Sie die Struktogramme zu den Algorithmen aus Aufgabe 2!

2.3. Grundstrukturen der strukturierten Programmierung und ihre Umsetzung in BASIC

▶ Die Grundstrukturen der strukturierten Programmierung sind
- Folge,
- Auswahl,
- Wiederholung,
- Unteralgorithmus.

Im folgenden werden die Eigenschaften dieser Grundstrukturen, ihre Darstellung in den verschiedenen Notationsformen und ihre Umsetzung in die Programmiersprache BASIC behandelt.

2.3.1. Folge

▶ Die Folge (auch Sequenz genannt) ist die einfachste Form eines Algorithmus. Sie ist eine Aneinanderreihung von auszuführenden Operationen, die wiederum entweder Elementaroperationen oder Folgen, Auswahlen, Wiederholungen oder Unteralgorithmen sein können.

Alle Operationen müssen nacheinander abgearbeitet werden. Jede neue Operation kann erst ausgeführt werden, wenn die vorhergehende vollständig abgearbeitet ist.
Ein Beispiel für die drei Notationsformen einer Folge ist der Algorithmus zur Berechnung der Zinsen im vorhergehenden Kapitel.

in verbal formalisierter Notation	Struktogramm
anweisung 1 anweisung 2 ... anweisung m	anweisung 1 anweisung 2 ... anweisung m

Umsetzung in BASIC

```
a1    anweisung 1
a2    anweisung 2
..    ...
am    anweisung m
```

Die Programmiersprache BASIC der betrachteten Kleincomputer läßt zu, mehrere Anwei-

sungen einer Folge in eine Programmzeile zu schreiben, solange die Programmzeilenlänge ausreicht:

ad anw1:anw2: ... :anw m

Dabei sind a1, a2, ..., am, ad Zeilennummern des zulässigen Bereichs von {1 bis 65529}.
Viele Berechnungen nach Formeln sind Beispiele für die Grundstruktur Folge.

1 Entwickeln Sie einen Algorithmus zur Berechnung von Flächeninhalt und Umfang eines Dreiecks, von dem die Längen der drei Seiten a, b, c bekannt sind.
Wenn a, b und c die Seitenlängen eines Dreiecks sind und s dessen halber Umfang, so wird der Flächeninhalt des Dreiecks berechenbar mit

$F = \sqrt{s * (s - a) * (s - b) * (s - c)}$.

Damit können wir festlegen:
Die **Eingabe** umfaßt die Werte für: a, b, c $\in R^+$

(R^+ = Bereich der positiven Zahlen).

Die **Verarbeitung** erfolgt mit den Formeln

$U = a + b + c$ und

$F = \sqrt{s * (s - a) * (s - b) * (s - c)}$ mit $s = U/a$.

Die **Ausgabe** beinhaltet die Werte von U und F.
Unter Beachtung dieser Festlegungen erhalten wir den folgenden Algorithmus und das zugehörige BASIC-Programm.

Algorithmus	Programm
(UMFANG UND FLÄCHENINHALT EINES DREIECKS) Eingabe a,b,c U := a + b + c S := U/2 F := $\sqrt{s(s-a)(s-b)(s-c)}$ Ausgabe "Umfang =";U "Flächeninh. =";F Ende	10 REM UMFANG UND FLAECHENIN- 20 REM HALT DREIECK - HERON 30 INPUT "A="; A 40 INPUT "B="; B 50 INPUT "C="; C 60 U = A+B+C 70 S = U/2 80 F = SQR(S*(S-A)*(S-B)*(S-C)) 90 PRINT "Umfang =";U 100 PRINT "Flaecheninh. =";F 110 END

Wir testen (prüfen) dieses Programm mit unterschiedlichen Eingabewerten. Für A=3, B=4 und C=5 gibt das Programm wie erwartet „Umfang = 12" und „Flächeninh. = 6"

aus. In diesem Fall liefert das Programm das Verlangte. Nehmen wir aber z. B. A=3, B=5 und C=9, so meldet sich der Computer mit

```
? FC ERROR IN 80
```

Das bedeutet, daß ein unzulässiger Funktionsaufruf (engl. illegal function call) in der Zeile 80 erfolgte. Als Fehlerursache vermuten wir nach kurzer Überlegung das Radizieren aus negativen Zahlen und überprüfen deshalb das Argument. Da S=8.5 ist, trifft unsere Vermutung zu, denn der Faktor (S−C)=−0.5 ist wie das gesamte Argument der Wurzelfunktion negativ. Weil eine Dreiecksungleichung, nämlich A+B>C, nicht erfüllt ist, kann kein Dreieck mit den Seitenlängen A=3, B=5 und C=9 existieren.

Bei unserer weiteren Algorithmierung werden wir darauf zu achten haben, daß solche unzulässigen Zahleneingaben mit dem Hinweis auf die Unzulässigkeit zurückgewiesen werden oder die Eingabe so oft wiederholt werden muß, bis die Eingabe zulässig ist.

Wenn wir das Struktogramm und das BASIC-Programm vergleichen, so sehen wir, daß die Eingabe von a, b, c in drei Eingabeanweisungen umgesetzt wurde. Als zweite Möglichkeit können wir auch programmieren:

```
30 INPUT "Werte fuer A, B, C";A,B,C
```

Nach dem Semikolon kommen drei Variablen, die wertmäßig zu belegen sind und durch Kommas getrennt werden.

▶ Innerhalb der einzugebenden Werte darf kein Komma vorkommen, da es vom BASIC-Interpreter in der INPUT-Anweisung als Trennzeichen zwischen 2 Eingabewerten verwendet wird. Das ist vor allem bei der Eingabe von Werten für Zeichenkettenvariable zu beachten, da dort Kommas nur innerhalb der begrenzenden Anführungszeichen vorkommen dürfen.

2 Einfache Berechnungsaufgaben aus der Mathematik und der Physik wird man sicher schneller mit einem elektronischen Taschenrechner lösen. Wir haben hier absichtlich ein einfaches Beispiel gewählt, um die Entwicklung des Algorithmus nicht durch einen zu komplizierten mathematischen Sachverhalt zu erschweren.

Entwickeln Sie einen Algorithmus zur Ermittlung der Anzahl von Zeichen (Buchstaben und Leerzeichen) in Limericks!

Ein Limerick ist ein humorvoll-ironischer 5-Zeilen-Vers mit paradoxer oder grotesker Endzeile. Sein Reimschema ist aabba.
Wir testen das Programm (Seite 54 oben rechts) mit einem Limerick von H. Preissler:

Ein Reimer aus dem Bezirk Frankfurt,
der mal unwesentlich krank wurd,
hat im Bett sich gerekelt
und Limericks gehäkelt,
wodurch ihm die Zeit nicht lang wurd.

Nach Eingabe der 1. Zeile quittiert der Rechner mit der Fehlermeldung

```
? EXTRA IGNORED
```

Algorithmus	Programm
(ZEICHEN IN LIMERICKS) Eingabe (zeilenweise) Z1$,Z2$ Z3$,Z4$ Z5$ T$:=Z1$+Z2$+Z3$+Z4$ +Z5$ T :=Länge(T$) Ausgabe "Der Limerick" Z1$,Z2$,Z3$, Z4$,Z5$ "besteht aus";T; "Zeichen" Ende	10 REM ZEICHENZAEHLUNG IN LIMERICKS 20 INPUT "1. Zeile :";Z1$ 30 INPUT "2. Zeile :";Z2$ 40 INPUT "3. Zeile :";Z3$ 50 INPUT "4. Zeile :";Z4$ 60 INPUT "5. Zeile :";Z5$ 70 CLS 80 T$=Z1$+Z2$+Z3$+Z4$+Z5$ 90 T = LEN (T$) 100 PRINT : PRINT 110 PRINT "Der Limerick" 120 PRINT : PRINT Z1$ 130 PRINT Z2$ 140 PRINT Z3$ 150 PRINT Z4$ 160 PRINT Z5$ 170 PRINT: PRINT "besteht aus" 180 PRINT: PRINT T;"Zeichen." 190 END

Dieses Signal des Rechners an den Programmnutzer bedeutet, daß zu viele Werte eingegeben worden sind, der Rechner die überflüssigen ignoriert und die Programmabarbeitung fortsetzt. Die Computer-Reaktion ist verständlich, da das Komma am Ende der Textzeile als Trennzeichen zwischen 2 Eingabewerten interpretiert wird. Das heißt, der BASIC-Interpreter geht beim Erkennen des Kommas davon aus, daß der Wert für eine zweite Variable folgt. In der Zeile 20 steht aber nur die eine Variable Z1$.
Die Eingabe wird neu begonnen. Dieses Mal werden die Zeichenkettenkonstanten in Anführungszeichen eingeschlossen. So wird beispielsweise die erste Zeile in folgender Form bereitgestellt:

"Ein Reimer aus dem Bezirk Frankfurt,"

Diese Eingabe wird akzeptiert. Nun können wir in gleicher Weise die folgenden Zeilen eingeben. Nach der vollständigen Eingabe des Limericks wird der Bildschirm mit CLS gesäubert. Danach kommt eine weitere Fehlermeldung

```
? OS ERROR IN 80
```

Diese Fehlerausschrift bedeutet, daß die Größe des Zeichenkettenspeicherbereichs nicht ausreicht (Out of string space). Standardmäßig umfaßt der Zeichenkettenspeicherbereich 256 Bytes zur Aufnahme von 256 einzelnen Zeichen. Beim Nachzählen der Zeichen des Test-Limericks kommt man auf 152 – zählt man die Leerzeichen zwischen zwei Wörtern und die Satzzeichen mit. Dann müßten aber noch 104 Bytes im Zeichenkettenspeicherbereich frei sein. Durch die Programmzeile 80 wird aber weiterer Platz im Zeichenkettenspeicherbereich für die Speicherung von Zwischenergebnissen bei der Ausführung von Zeichenkettenoperationen benötigt. Bei der Ausführung der dort notwendigen 4 Verkettungen kommt es zum Überschreiten des Zeichenkettenbereichs.

Mit dem Kommando

```
CLEAR 500
```

wird der Zeichenkettenspeicherbereich 500 Bytes groß. Man kann dieses Kommando auch als Anweisung

```
5 CLEAR 500
```

ins Programm aufnehmen. Nach dieser Korrektur funktioniert das Programm wie gewünscht. Wir erhalten als Computer-Antwort, daß dieser Limerick aus 152 Zeichen besteht.
Auch diese Aufgabenstellung ist ein Beispiel für eine Folge, die nur aus Eingabe, einfacher Wertzuweisung und Ausgabe besteht. Man sollte meinen, daß die Zeichenmenge schneller ausgezählt ist, als extra dafür ein Programm zu schreiben. Wenn wir aber Texte mittels Computer ohnehin zu schreiben haben, um sie anschließend mehrfach über einen Drucker ausgeben zu können, dann ist diese zusätzliche Information über die Anzahl der Zeichen im Text eine Aufgabe für den Computer. Solche einfachen Folgen sind erst als Teilstrukturen in umfangreichen Algorithmen von großem Interesse.

Zusammenfassung

Die **Folge** ist eine Aneinanderreihung von Anweisungen. Eine einfache Folge besteht aus

Eingabe
Ergibtanweisung
Ausgabe.

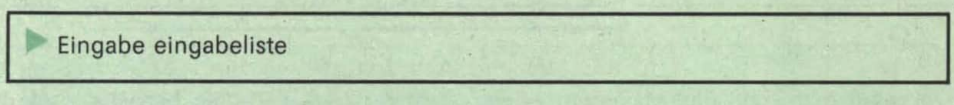
Es sind die einfachen **Datentypen Zahl** und **Zeichenkette** zu unterscheiden.

Daten können über Tastatur mit der Anweisung **INPUT** eingegeben werden. Diese Anweisung hat das Format

INPUT ["hinweistext";] variable[,variable] ...

variable – numerische oder Zeichenkettenvariable, für die ein Wert eingegeben werden soll
hinweistext – Zeichenkette, die Informationen für den Programmnutzer enthält.

Im Struktogramm wird die Eingabe durch ein eigenes Symbol gekennzeichnet



Die **Ergibtanweisung** hat das Format

[LET] variable = ausdruck

variable – Bezeichnung einer numerischen oder Zeichenkettenvariablen

ausdruck – konstante, operand operator operand
Ergebnis vom gleichen Typ wie die Zielvariable.

Daten können auf dem Bildschirm mit der Anweisung

PRINT printliste

ausgegeben werden.
printliste kann

- leer → Ausgabe Leerzeile
- eine Konstante oder Variable oder Ausdruck
- eine Liste von Konstanten und/oder Variablen und/oder Steuerangaben sein.

Auch für die Ausgabe wurde im Struktogramm ein eigenes Symbol eingeführt.

Ausgabe ausgabeliste ▶

Mit der Anweisung **CLEAR** n wird Speicherplatz für Zeichenketten bereitgestellt.

Aufgaben

1. Entwickeln Sie ein Programm zur Berechnung des arithmetischen, geometrischen, harmonischen und quadratischen Mittels von drei positiven reellen Zahlen A, B, C! Es gilt

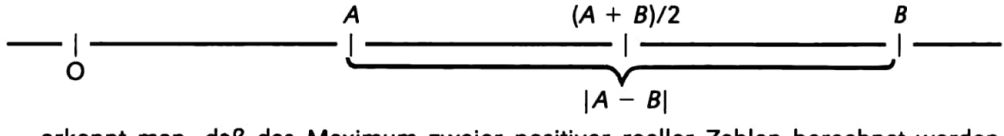
$$AM = \frac{A + B + C}{3} \qquad GM = \sqrt[3]{A * B * C}$$

$$HM = \frac{3}{\frac{1}{A} + \frac{1}{B} + \frac{1}{C}} \qquad QM = \sqrt{\frac{A^2 + B^2 + C^2}{3}}$$

   Testen Sie Ihr Programm mit mehreren Tripeln (A, B, C)! Welche Relationen existieren aufgrund der Testergebnisse vermutlich zwischen diesen 4 Mitteln? Beweisen Sie Ihre Vermutung!

2. Schreiben Sie ein Programm, das alle sechs Permutationen der 3 einzugebenden Satzanteile Subjekt (S), Prädikat (P), Objekt bzw. Adverbialbestimmung (OA) hintereinander auflistet!

3. Ausgehend von der Überlegungsskizze



   erkennt man, daß das Maximum zweier positiver reeller Zahlen berechnet werden kann nach

$$MAX(A,B) = \frac{A + B}{2} + \frac{|A - B|}{2}$$

   Finden Sie eine analoge Berechnungsvorschrift für das Minimum! Entwickeln Sie ei-

nen Algorithmus, mit dem das Maximum und Minimum der drei positiven reellen Zahlen A, B, C berechnet wird! Benutzen Sie dafür die Standardfunktion ABS(X), die zu X den Absolutbetrag ermittelt!

4. Schreiben Sie ein Programm, mit dessen Hilfe Sie ‚experimentell' überprüfen, ob folgende aus der Mathematik bekannten Gesetze auch auf der Computerzahlenmenge gelten!
   (1) für alle $a, b, c \in P$: $(a + b) + c = a + (b + c)$
   (2) für alle $a, b, c \in P$: $(a - b) * c = a * c - b * c$
   (3) für alle $a, b, c \in P$: $(a - b)^2 = a^2 - 2 * a * b + b^2$

   Testen Sie auch mit folgenden Zahlentripeln!
   a = 0.666667 b = 0.666667^2 c = 0.666667^3
   a = 0.987654 b = 0.987655 c = 987654
   a = 2222.7 b = 2222.8 c = 1

5. Schreiben Sie ein Programm zur Berechnung der Geschwindigkeit und des Weges einer geradlinigen, gleichmäßig beschleunigten Bewegung, wenn die Zeit und die Beschleunigung gegeben sind (v = a * t, s = a/2 * t^2)!

6. Entwickeln Sie ein Programm zum Schreiben einer Rechnung für genau 2 Artikel! In der Eingabe sind für den 1. und für den 2. Artikel jeweils Artikelnummer, Stückzahl und Einzelpreis einzugeben. Die Ausgabe erfolge tabellarisch:

   ARTIKEL-NR. STÜCKZAHL EINZELPREIS GESAMTPREIS

   ZU ZAHLENDER GESAMTBETRAG: .....

2.3.2. Auswahl

Die Auswahl (auch Alternative, bedingte Anweisung oder IF-Anweisung genannt) ist eine Steuerstruktur. Sie liegt vor, wenn die Ausführung eines Teils oder unterschiedlicher Teile des Algorithmus von der Beantwortung einer Frage (Bedingung) abhängig gemacht wird.

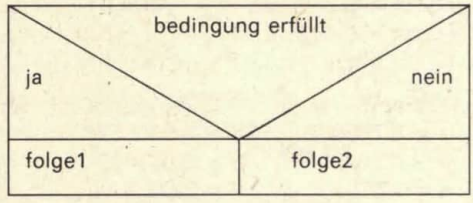
Bei der Auswahl wird eine Bedingung geprüft und dann entschieden, welche weiteren Anweisungen abgearbeitet werden sollen.

Es werden drei Formen der Auswahl unterschieden:
- die einseitige Auswahl,
- die zweiseitige Auswahl,
- die mehrseitige Auswahl.

Wir beginnen mit der Erläuterung der zweiseitigen Auswahl.

Zweiseitige Auswahl

Die zweiseitige Auswahl (auch als Alternative bezeichnet) wird dargestellt

in verbal formalisierter Notation	als Struktogramm
WENN bedingung DANN folge1 SONST folge2	

Umsetzung in BASIC

```
ak    IF NOT(bedingung) THEN an
al          folge1
am    GOTO ap
an          folge2
ap    REM   ende der zweiseitigen auswahl
```

Die Bedingung ist mit NOT (nicht) verneint. Dadurch bleibt die Reihung folge1 und dann folge2 auch im BASIC-Programm wie in den anderen Notationsformen.

Wird bei Abarbeitung des Programms die Zeile ak erreicht und die verneinte Bedingung ist wahr, so wird mit ‚folge1' der Ja-Zweig der Alternative ausgeführt. Durch die Anweisung GOTO ap wird dieser Zweig mit einem Sprung zur Zeile ap beendet und damit der Nein-Zweig übersprungen. Die Verwendung der GOTO-Anweisung wird in der Auswahl häufig notwendig sein, da die Begrenzung von 72 Zeichen für die ganze Anweisung sehr enge Grenzen setzt.

In Bedingungen werden logische Operatoren für die Verknüpfung von Aussagen und Vergleichsoperatoren für den Vergleich zwischen Werten von Ausdrücken benötigt. Die Abarbeitung einer Bedingung oder eines Ausdrucks erfolgt entsprechend der Priorität der Operatoren. Bei Operatoren gleicher Priorität erfolgt die Abarbeitung von links nach rechts. Soll diese Abarbeitungsfolge verändert werden, so sind Klammern zu setzen.

Priorität	Operator	Bedeutung	Art
1	^	Potenzierung	arithmetische Operatoren
2	–	negatives Vorzeichen	
3	*	Multiplikation	
	/	Division	
4	+	Addition	
	–	Subtraktion	
5	<	kleiner als	Vergleichsoperatoren
	<=	kleiner gleich	
	=	gleich	

Priorität	Operator	Bedeutung	Art
5	<> > >=	ungleich größer als größer gleich	Vergleichsoperatoren
6 7 8	NOT AND OR	Negation Konjunktion Alternative	logische Operatoren
9	+	Verkettungsoperator für Zeichenketten (Strings)	

Wenn nur so wenige Anweisungen in Abhängigkeit von der Bedingung ausgeführt werden müssen, die auf einer BASIC-Programmzeile Platz finden, dann kann die Umsetzung in BASIC nach

```
ad IF bedingung THEN folge 1 : ELSE folge 2
```

erfolgen.

1 Es sind ein Algorithmus und ein Programm zur Überprüfung von Dreiecken auf Rechtwinkligkeit zu entwickeln.

Die **Eingabe** umfaßt die Zahlenwerte für die Längen der 3 Dreieckseiten a,b,c $\in R^+$.
Die **Verarbeitung** besteht im Bilden von c^2 und $a^2 + b^2$.
Die **Ausgabe** ist entweder der Text "RECHTWINKLIG"
oder der Text "NICHT RECHTWINKLIG".

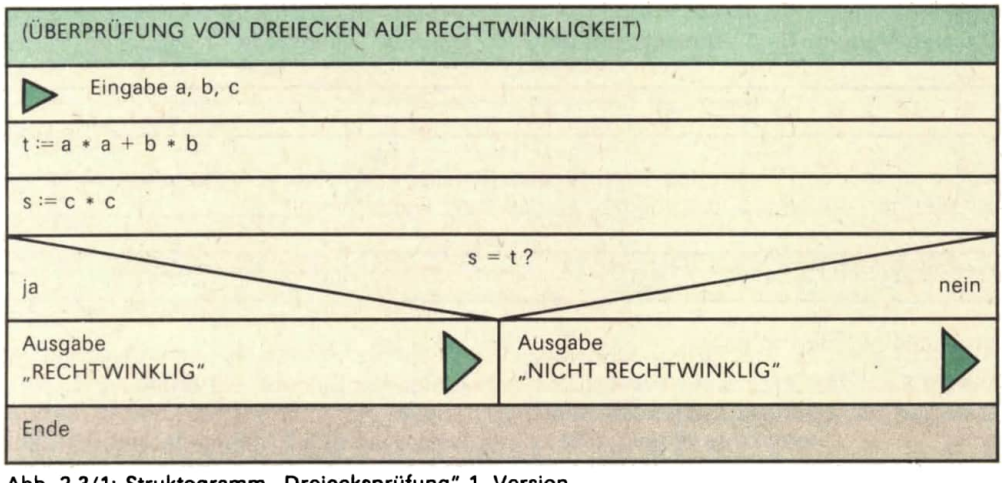


Abb. 2.3/1: Struktogramm „Dreiecksprüfung" 1. Version

Programm

```
10 REM UEBERPRUEFUNG VON DREIECKEN AUF RECHTWINKLIGKEIT
20 INPUT "A, B, C (C MAXIMAL)";A,B,C
```

```
30 T = A * A + B * B
40 S = C * C
50 IF T=S THEN PRINT "RECHTWINKLIG" : ELSE PRINT
   "NICHT RECHTWINKLIG"
60 END
```

Auch hier sind aus Gründen der Rechengeschwindigkeit und Rechengenauigkeit statt A↑2, B↑2, C↑2 (↑ ist der Potenzoperator in BASIC) die Ausdrücke mit Operationen niederer Stufe A*A, B*B, C*C gewählt worden.

Natürlich kann dieser Algorithmus verbessert werden. Es sollte

- die Ausgabe aussagekräftiger sein,
- der Bildschirm ordentlich und übersichtlich gestaltet werden und
- die Sicherung, daß die Eingabe zulässig ist, vom Programm übernommen werden.

Wenn ein längerer Text ausgegeben werden soll, dann paßt die zweiseitige Auswahl nicht mehr auf eine Programmzeile. Es muß jetzt im BASIC-Programm mit der allgemeinen Form der Auswahl gearbeitet werden.

```
50  IF NOT(T = S) THEN 90
60     PRINT "DAS DREIECK MIT DEN 'SEITEN'";A;B;C
70     PRINT : PRINT "IST RECHTWINKLIG."
80  GOTO 110
90     PRINT "DAS DREIECK MIT DEN 'SEITEN'";A;B;C
100    PRINT : PRINT "IST NICHT RECHTWINKLIG."
110 REM ende der zweiseitigen auswahl
120 END
```

Die Negation von T=S ist hier leicht gebildet. Die Anweisungszeile

```
50  IF T <> S THEN 90
```

würde auch dem Verlangten entsprechen. Bei komplizierteren Aussageformen ist es schwieriger, die Negation zu bilden, so daß auch weiterhin mit

```
IF NOT(bedingung) THEN ...
```

gearbeitet werden sollte.

Die im Sinne des Programms fehlerhafte Reihenfolge der Eingabe der Daten A=3, B=5, C=4 darf nicht zur unzutreffenden Antwort

```
„DAS DREIECK MIT DEN 'SEITEN' 3 5 4 IST NICHT RECHTWINKLIG"
```

führen. Unsinnige Antworten des Programms wie

```
„DAS DREIECK MIT DEN 'SEITEN' -3 4 5 IST RECHTWINKLIG"
```

signalisieren, daß der Algorithmus unbedingt zu verbessern ist. Das Struktogramm in Abbildung 2.3/2 zeigt eine 2. Version.

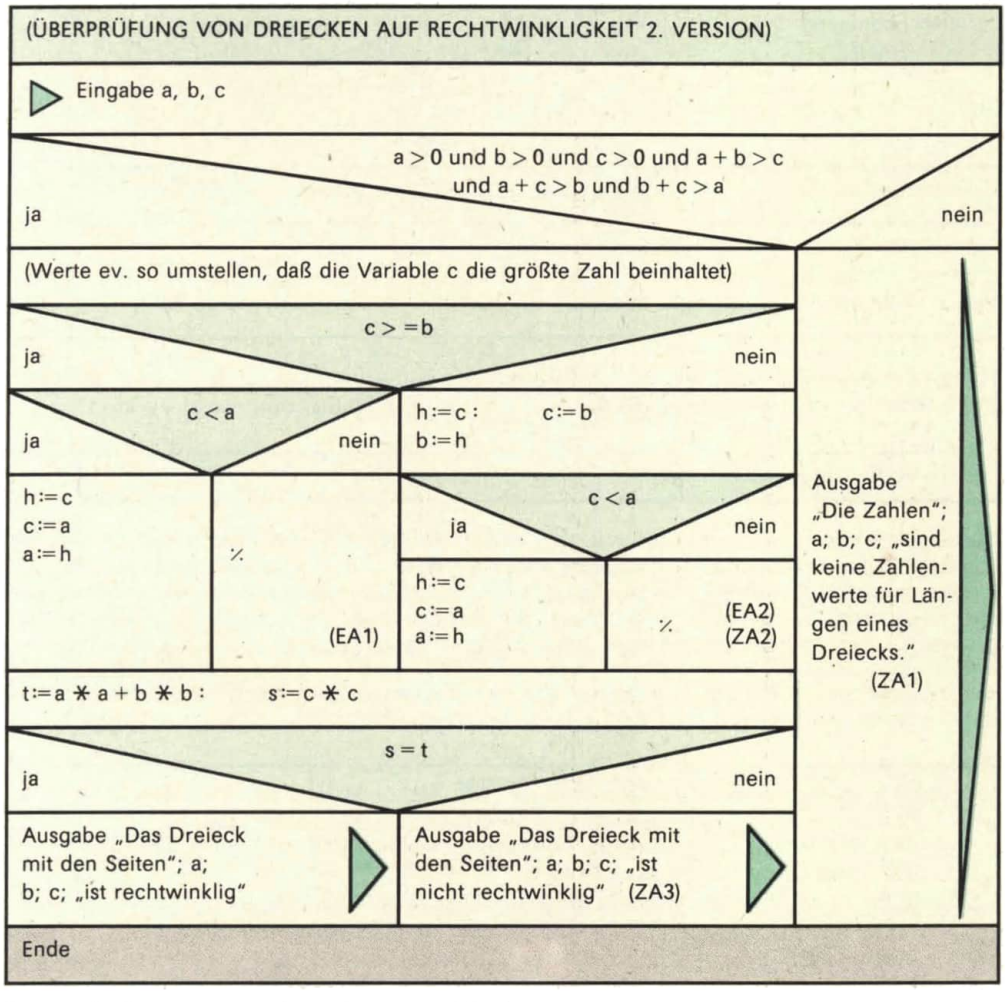
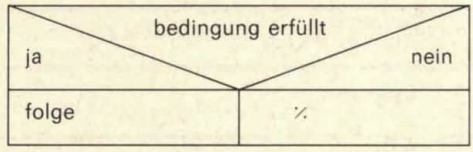


Abb. 2.3/2: Struktogramm „Dreiecksprüfung" 2. Version

Im Vergleich zum ersten Algorithmus sieht dieser recht kompliziert aus. Eine Überprüfung dieses Algorithmus mit den oben angeführten Zahlenwerten zeigt, daß der Algorithmus auf diese Zahlenwerte richtig reagiert. Zum gleichen Problem kann es unterschiedliche Algorithmen geben, die je nach Güte und Leistung variieren können.

Einseitige Auswahl

Es gibt im obigen Beispiel Auswahlen, bei denen nur in einem Fall etwas zu tun ist. Diese Struktur stellt einen Spezialfall der zweiseitigen Auswahl dar und heißt einseitige Auswahl. Sie wird wie folgt dargestellt:

in verbal formalisierter Notation	als Struktogramm
WENN bedingung DANN folge	

Umsetzung in BASIC

```
ak    IF NOT(bedingung) THEN ap
am       folge
ap    REM ende der einseitigen auswahl
```

Haben die Anweisungen, die beim Erfülltsein der Bedingung auszuführen sind, auf einer Zeile Platz, so kann die einseitige Auswahl wie folgt in BASIC umgesetzt werden.

ak IF bedingung THEN folge

In einem Teil des Algorithmus ist eine weitere Idee zur Ermittlung des Maximums von 3 Zahlen mittels Größenvergleich dargestellt. Wenn die Variable c nicht mit dem größten Wert belegt ist, sind die Werte zweier Variabler gegenseitig auszutauschen. Zum Zwischenspeichern eines Wertes wird eine Hilfsvariable H verwendet. Die 3 Anweisungen H=C, C=A, A=H werden in eine BASIC-Programmzeile geschrieben, um die enge Zusammengehörigkeit zu betonen. Da alle 6 Bedingungen a>c, b>c, c>0 (Positivität von Längen), a+b>c, a+c>b und b+c>a (Dreiecksungleichung) erfüllt sein müssen, um im JA-Zweig der zweiseitigen Auswahl ZA1 weiterzugehen, sind sie konjunktiv – durch das aussagenlogische AND (und) – zu verknüpfen.

```
10  REM UEBERPRUEFUNG VON DREIECKEN AUF
20  REM RECHTWINKLIGKEIT (2. VERSION)
30  REM Autoren: Schilling/Toepfer; Leipzig/Berlin; 5'86
40  REM numerische Variablen:
50  REM     A,B,C  -  Zahlenwerte für Seitenlaengen
60  REM     T,S    -  Hilfsvariable für Zwischenergebnisse
70  REM     H      -  Hilfsvariable zum Umspeichern
80  REM ------------------------------------------------
100 WINDOW : CLS
110 PRINT "  UEBERPRUEFUNG VON DREIECKEN"
120 PRINT : PRINT "  MIT DEN SEITEN A, B, C"
130 PRINT : PRINT "  AUF RECHTWINKLIGKEIT"
140 PRINT : PRINT
150 INPUT "A, B, C"; A,B,C
160 IF NOT(A>0 AND B>0 AND C>0 AND A+B>C AND A+C>B AND
    B+C>A) THEN 370
170    IF NOT(C>=B) THEN 220
180       IF NOT(C<A) THEN 200
190          H=C : C=A : A=H
200       REM ende der eins. auswahl 1  EA1
210       GOTO 260
```

```
220        H=C : C=B : B=H
230        IF NOT(C<A) THEN 250
240           H=C : C=A : A=H
250        REM ende der eins. auswahl 2  EA2
260     REM ende der zweis. auswahl 2   ZA2
270     T = A * A + B * B
280     S = C * C
290     IF NOT(T=S) THEN 330
300        PRINT : PRINT "DAS DREIECK MIT DEN SEITEN";A;B;C
310        PRINT : PRINT "IST RECHTWINKLIG!"
320     GOTO 350
330        PRINT : PRINT "DAS DREIECK MIT DEN SEITEN";A;B;C
340        PRINT : PRINT "IST NICHT RECHTWINKLIG!"
350     REM ende der zweis. auswahl 3  ZA3
360 GOTO 400
370     PRINT : PRINT "DIE ZAHLEN";A;B;C
380     PRINT : PRINT "SIND KEINE ZAHLENWERTE FUER DIE"
390     PRINT : PRINT "LAENGEN EINES DREIECKS!"
400 REM ende der zweis. auswahl 1  ZA1
410 END
```

2 Jeder Bürger der DDR bekommt bei seiner Geburt nicht nur einen Namen, sondern auch eine Personenkennzahl (PKZ). Sie ist eine amtlich festgelegte Kennzahl, die über Geburtsdatum und Geschlecht Auskunft gibt und der Rationalisierung in der Verwaltung dient. Für den Einsatz in Dateiprogrammen ist ein Teilprogramm zu entwickeln, das aus der PKZ die richtige Anrede für einen Adreßaufkleber ermittelt. Es werden nur PKZ von Schulkindern eingegeben.

Die **Eingabe** besteht aus der 12stelligen PKZ.

Die **Verarbeitung** ist darauf gerichtet, die 7. Ziffer der PKZ von links zu analysieren. Zulässig ist nur eine 4 oder eine 5, so daß die Person als männlich oder weiblich ausgewiesen wird. Andernfalls muß ein Eingabefehler gemeldet werden.

Die **Ausgabe** besteht aus den Wörtern „Schüler" oder „Schülerin".

Wird die Folge von Ziffern der Personenkennzahl als eine Zeichenkette aufgefaßt, so können Zeichenkettenfunktionen angewendet werden. Die Funktion TEILKETTE bildet z. B. mit

```
TEILKETTE („PERSONENKENNZAHL",7,4)
```

mitten aus der Zeichenkette „PERSONENKENNZAHL" eine Teilkette, die mit der 7. Position beginnt und die folgenden Zeichen – insgesamt 4 – umfaßt. Die auf diese Weise gebildete Zeichenkette ist „ENKE". Da meist etwas aus der Mitte herauskopiert wird, heißt die entsprechende BASIC-Zeichenkettenfunktion MID$. Sie hat das Format

```
MID$(zeichenkette, ab-position [,zeichenanzahl])
```

Das Schlüsselwort MID kommt von middle (Mitte). Das angehängte Zeichen ‚$' weist darauf hin, daß der zurückgegebene Funktionswert wieder eine Zeichenkette ist.

Interessiert das 7. Zeichen der Zeichenkette PKZ$, so erhält man es in BASIC mit der Anweisung

```
G$ = MID$(PKZ$,7,1)
```

Der Zeichenkettenvariablen G$ wird bei Abarbeitung dieser Anweisung die Teilkette von PKZ$ zugewiesen, die ab der 7. Position beginnt und 1 Zeichen lang ist.

Algorithmus

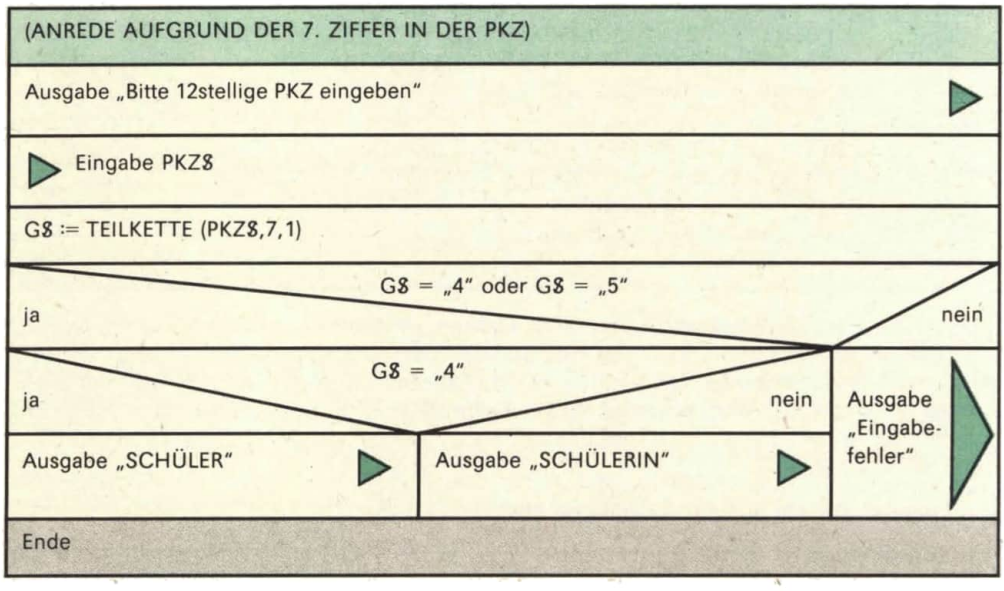


Abb. 2.3/3: Struktogramm „Anrede aus PKZ"

Hinweis: In den folgenden Programmen wird auf Angaben zu den Programmautoren im Programmkopf verzichtet, um Platz zu sparen.

Programm

```
10  REM ANREDE AUFGRUND DER 7. ZIFFER IN DER PKZ
20  REM Zeichenkettenvariablen:
30  REM     PKZ$   -  Personenkennzahl
40  REM     G$     -  Geschlecht
45  REM ----------------------------------------
50  WINDOW : CLS
60  PRINT "BITTE 12STELLIGE PKZ EINGEBEN!" : PRINT
70  INPUT ""; PKZ$
80  G$ = MID$(PKZ$,7,1)
90  IF NOT(G$="4" OR G$="5") THEN 160
100    IF NOT(G$="4") THEN 130
110       PRINT : PRINT "SCHUELER"
```

```
120    GOTO 140
130       PRINT : PRINT "SCHUELERIN"
140    REM ende der inneren zweis. auswahl
150 GOTO 170
160    PRINT : PRINT "EINGABEFEHLER"
170 REM ende der aeusseren zweis. auswahl
180 END
```

Dieses BASIC-Programm kann wegen der Kürze der auszugebenden Texte vereinfacht werden. Es sind die Programmzeilen 90 bis 170 durch

```
90  PRINT
100 IF NOT(G$="4"ORG$="5")THEN PRINT"EINGABEFEHLER":GOTO 180
110 IF G$="4" THEN PRINT"SCHUELER" : ELSE PRINT"SCHUELERIN"
```

ersetzbar. Dabei geht allerdings die angestrebte strenge Analogie zum Struktogramm verloren.

Es ist zulässig, Variablennamen zu benutzen, die länger als 2 Zeichen sind – wie hier PKZ$. Das kann die Lesbarkeit von Programmen verbessern. Trotzdem werden nur die ersten 2 Zeichen zur Unterscheidung von Variablen benutzt.

In Programmzeile 70 wurde als auszugebender Text bei INPUT die Leerzeichenkette "" verwendet, um die Anzeige des ? auf dem Bildschirm zu unterdrücken.

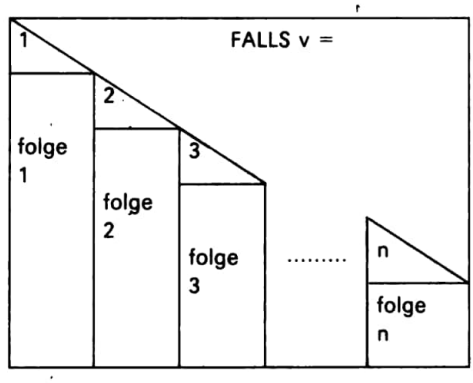
Mehrseitige Auswahl

Eine Teilaufgabe in größeren Programmen kann im Ausgeben des Geburtsmonats in der ausgeschriebenen Form bestehen (ausgehend von einer Personenkennzahl). In der Praxis werden solche Daten in Dateien (Datensammlungen) bereitgestellt. Für unser Beispiel sollen die zu analysierenden Daten mit der Tastatur eingegeben werden.

Eingabe: Nachname, Vorname und Personenkennzahl
Verarbeitung: Analyse der 3. und 4. Ziffer in der PKZ und Zuordnung der entsprechenden Monatsnamen
Ausgabe: „... hat im Monat ... Geburtstag."

Aufgrund der Zahlen 1 bis 12 ist fallweise die Ausgabe der Monatsnamen Januar bis Dezember zu veranlassen. Für eine derartige Aufgabe eignet sich die Algorithmenstruktur mehrseitige Auswahl (auch Fallauswahl genannt), die in den drei Notationsformen wie folgt dargestellt wird:

Hinweis: In der Informatik wird die Ziffer Null oft mit einem Schrägstrich versehen (Ø). Wenn Verwechslungen der Ziffer Null mit dem Buchstaben O unwahrscheinlich sind, wurde in diesem Buch auf eine besondere Kennzeichnung der Null verzichtet. Lautet eine Zeichenfolge beispielsweise MO > 0 (Seiten 67 und 68), dann weist MO auf die Variablenbedeutung MONAT hin, so daß es keine Verwechslung der Ziffer Null mit dem Buchstaben O geben kann. Anders bei der Zeichenfolge XO = 50. Da in der Mathematik oft x_0 verwendet wird, hier aber XO eine Koordinatenbezeichnung für den oberen Punkt bedeutet, wurde in dem Listing auf Seite 78 die Ziffer 0 durch Ø dargestellt, also XØ = 5Ø. Die Zeilennummern wurden dabei wie in jedem anderen Programm geschrieben.

in verbal formalisierter Notation	als Struktogramm
FALLS v = 1 : folge 1 2 : folge 2 3 : folge 3 .. .. n : folge n ENDE	

Umsetzung in BASIC

```
ak    ON v GOTO a1,a2,a3,...,an
a1        folge 1
. .   GOTO am
a2        folge 2
. .   GOTO am
a3        folge 3
. .   GOTO am
. .       . . . . .
an        folge n
am    REM ende der mehrseitigen auswahl
```

Das BASIC-Schlüsselwort **‚ON'** kann man mit **‚falls'** übersetzen. Es wird der Wert der Variablen ‚v' ermittelt und der ganzzahlige Wert gebildet. Dieser Wert muß zwischen 1 und der Anzahl von Zeilennummern nach GOTO liegen. Liegt ein gültiger Wert vor, so erfolgt ein Sprung zur Anweisung mit der ersten, zweiten, ..., n-ten Zeilennummer in der Liste nach GOTO.

Die Personenkennzahl wird auch hier als Zeichenkette eingegeben, um eine Teilkette TK$ = TEILKETTE (PKZ$,3,2) herauskopieren zu können. Der Zulässigkeitstest läuft auf die Abfrage hinaus, ob TK$ =„01" oder TK$=„02" oder TK$=„03" oder TK$=„04" oder TK$=„05" oder TK$=„06" oder TK$=„07" oder TK$= „08" oder TK$=„09" oder TK$=„10" oder TK$=„11" oder TK$=„12" zutrifft. Das ist sehr umständlich.
Deshalb wird von der Teilkette TK$ der numerische Wert MO=WERT(TK$) bestimmt, so daß der Zulässigkeitstest nur in der Bedingung besteht:
„Ist MO ganzzahlig und größer als 0 und kleiner als 13"

Die BASIC-Funktion, die den numerischen Wert einer Zeichenkette ermittelt, heißt VAL und hat das Format.

```
VAL(zeichenkette)
```

Das Schlüsselwort kommt von dem englischen Wort ‚value' und heißt ‚Wert'.

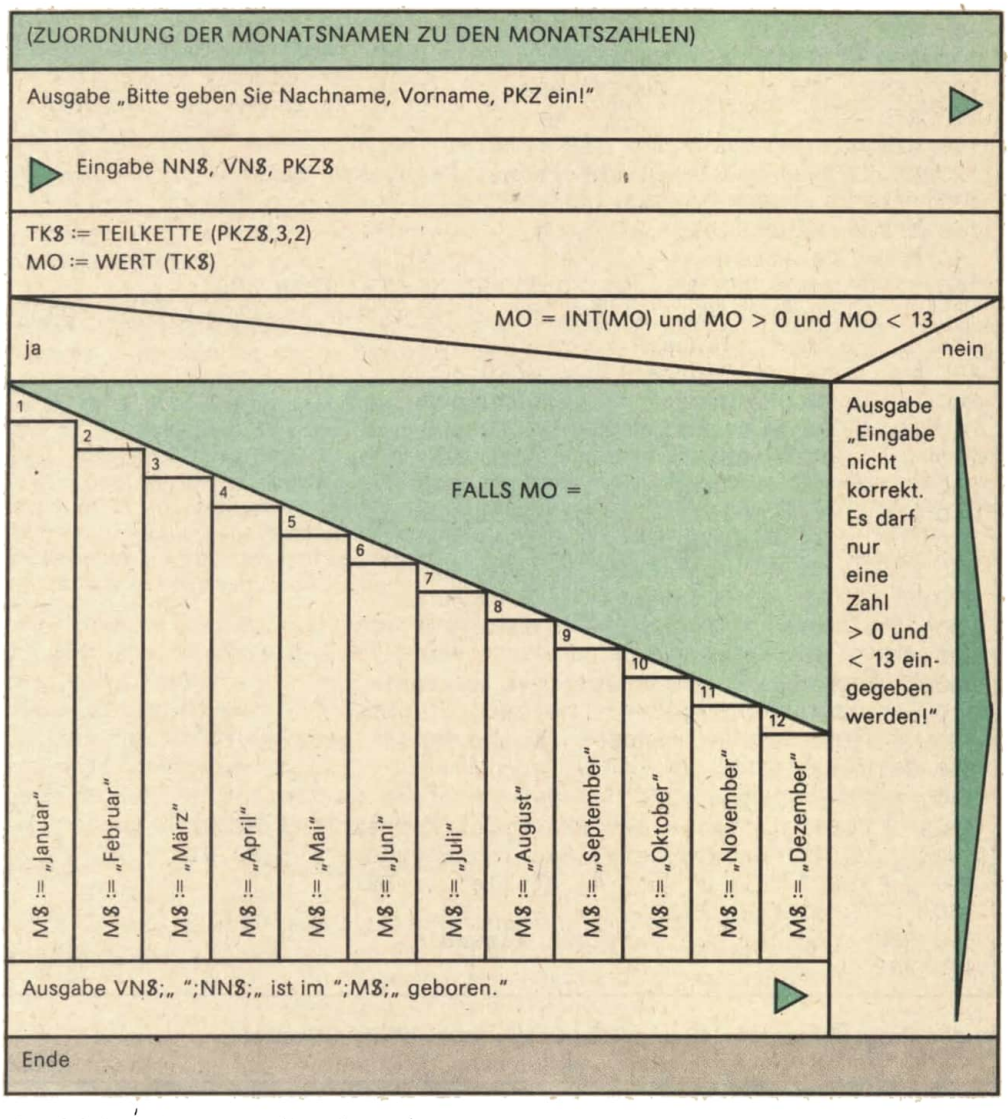


Abb. 2.3/4: Struktogramm „Zuordnung der Monatsnamen"

Programm

```
10  REM ZUORDNUNG DER MONATSNAMEN
20  REM ZU DEN MONATSZAHLEN ======================
30  REM Programm mit mehrseitiger Auswahl
40  REM numerische Variable:
50  REM     MO       - Monatszahl
60  REM Zeichenkettenvariablen:
70  REM     VN$,NN$ - Vor- und Nachname
```

```
80  REM     PKZ$     - Personenkennzahl
90  REM     TK$      - Teilkette
100 REM     M$       - Monatsname
105 REM ----------------------------------------
110 WINDOW : COLOR 7,1 : CLS : PRINT
120 PRINT "Geben Sie ein: Vorname, Nachname, PKZ"
130 PRINT : INPUT "";VN$, NN$, PKZ$
140 TK$ = MID$(PKZ$,3,2)
150 MO = VAL(TK$)
160 IF NOT(MO=INT(MO) AND MO>0 AND MO<13) THEN 340
170    ON MO GOTO 180,190,200,210,220,230,240,250,260,270,280,290
180       M$ = "Januar"     : GOTO 300
190       M$ = "Februar"    : GOTO 300
200       M$ = "Maerz"      : GOTO 300
210       M$ = "April"      : GOTO 300
220       M$ = "Mai"        : GOTO 300
230       M$ = "Juni"       : GOTO 300
240       M$ = "Juli"       : GOTO 300
250       M$ = "August"     : GOTO 300
260       M$ = "September"  : GOTO 300
270       M$ = "Oktober"    : GOTO 300
280       M$ = "November"   : GOTO 300
290       M$ = "Dezember"
300    REM ende der mehrseitigen auswahl
310    WINDOW 10,23,0,39 : COLOR 6,2 : CLS
320    PRINT VN$+" "+NN$;" ist im ";M$;" geboren."
330 GOTO 390
340    CLS
350    PRINT "Eingabe der PKZ nicht korrekt!" : PRINT
360    PRINT "Ueberpruefen Sie insbesondere"  : PRINT
370    PRINT "die 3. und 4. Stelle"  : PRINT
380    PRINT "der PKZ!"
390 REM ende der zweiseitigen auswahl
400 END
```

In den Programmzeilen 110 und 310 tritt die neue BASIC-Anweisung

```
COLOR v,h
```

auf. Durch diese Anweisung werden die Farben für den Hintergrund (‚h') und die Schreibfarbe (‚v') neu eingestellt. Dabei bewirkt das folgende CLS, daß nach dem Löschen des Bildschirms dieser in der neuen Hintergrundfarbe erstrahlt. In der Zeile 320 wurden Leerzeichen in den Ausgabetext eingefügt, damit nicht mehrere Wörter ohne Zwischenraum aufeinander folgen.

Anwendung der Auswahl bei der Simulation eines automatischen Verkaufs von S-Bahn-Fahrausweisen

Im folgenden Beispiel kommen alle 3 Formen der Auswahl vor, die ein-, die zwei- und die mehrseitige Auswahl.

4 Will man mit der Berliner S-Bahn fahren, benötigt man – wie für jedes öffentliche Verkehrsmittel – einen gültigen Fahrausweis. Zu manchen Tageszeiten sind die Schalter geschlossen, und man muß einen Fahrkartenautomaten benutzen. Es ist ein Programm zu entwickeln, das die Ausgabe einer gewünschten S-Bahnfahrkarte simuliert.

Eine Möglichkeit eines automatischen Fahrkartenverkaufs besteht darin, daß der Käufer sein Fahrtziel über eine geeignete Tastatur zeichenweise eingibt. Danach sucht der Computer heraus, welcher Fahrpreis zu bezahlen ist, und nach Eingabe des geforderten Geldbetrages wird die gewünschte Fahrkarte ausgegeben. Dafür ist eine Tastatur mit mindestens 30 Tasten (26 Buchstaben des Alphabetes, Ä, Ö, Ü, ß) notwendig. Tastaturen sind teuer und relativ störanfällig. Wie wir bereits wissen, muß die Verständigung mit einem Computer nach festen Regeln geführt werden. Orthographische Fehler führen bei Computern i. a. zum Nichterkennen eines Wortes. Wenn z. B. statt ZEUTHEN nur ZEUTEN als Fahrtziel eingegeben wird, dann kommt als Computerreaktion „Eingabefehler" oder „Fahrtziel ist dem Fahrkartenautomaten nicht bekannt" heraus. Zwar könnten hinreichend leistungsfähige, nutzerfreundliche Programme derartige orthographische Schwierigkeiten von Nutzern weitgehend meistern, aber dieser erste Vorschlag zum automatischen Fahrkartenverkauf ist auch wegen der Tastatur aus ökonomischen Gründen zu verwerfen.

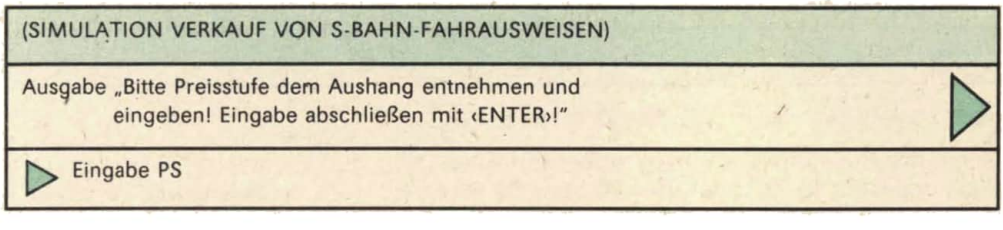
Die Fahrpreise der S-Bahn sind nach Preisstufen gestaffelt. Man kann die zu den Preisstufen gehörenden Fahrpreise einer Tabelle entnehmen. Es gibt eine zweite Tabelle, in der allen Zielbahnhöfen die jeweilige Preisstufe zugeordnet ist. Aus der ersten Tabelle geht hervor, daß es die Preisstufen 1 bis 8 gibt. Also ist eine weniger umfängliche Tastatur für die Ziffern als im ersten Vorschlag notwendig. Nachdem der Reisende der Tabelle 2 die Preisstufe entnommen hat, gibt er die entsprechende Ziffer ein. Danach fordert der Computer zur Zahlung eines entsprechenden Betrages auf. Ist dieser vom Reisenden eingezahlt, erfolgt die Fahrkartenausgabe.

Eingabe:
- Ziffer aus der Menge {1,2,3,4,5,6,7,8}; sie repräsentiert die Preisstufe,
- Geldbetrag über Tastatur (statt echter Eingabe).

Verarbeitung:
- Bestimmung des Fahrpreises aufgrund der Preisstufe,
- Prüfung, ob ausreichend viel Geld eingegeben wurde,
- ggf. Berechnung eines Restbetrages,
- Erzeugen eines Bildes der Fahrkarte

Ausgabe:
- ggf. Restbetragsangabe auf dem Bildschirm,
- Bild eines Fahrtausweises entsprechender Preisstufe auf dem Bildschirm (statt Druckerausgabe).

Hinweis: Die schwierigen Probleme der Zulässigkeitsprüfung einer Münzeingabe, der Erkennung einer eingegebenen Münze und das Ermitteln der Münzausgabe für den Restbetrag werden im Algorithmus (Abb. 2.3/5) nicht simuliert.



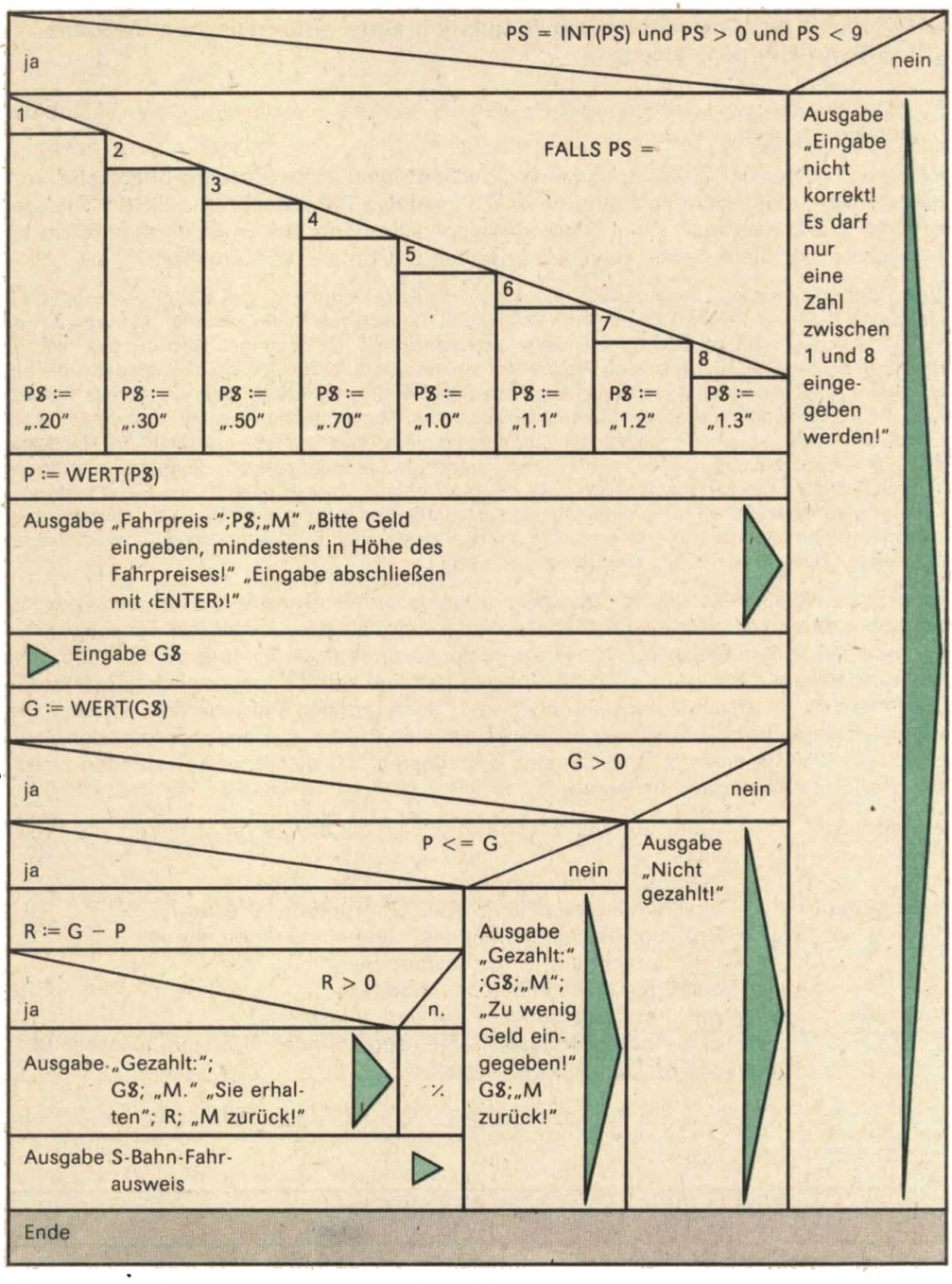


Abb. 2.3/5: Struktogramm „Simulation S-Bahn-Fahrkarte"

Damit der Preis und das zurückzugebende Geld stellengerecht angezeigt werden, sind Preis und Geld als Zeichenketten P$ bzw. G$ vereinbart. Um Vergleiche und Berechnun-

gen durchführen zu können, sind die numerischen Werte zu bilden. Die stellengerechte Ausgabe eines möglichen Restbetrages R ist noch nicht verwirklicht, so daß unter Umständen unbefriedigende Antworten auftreten können, wie z. B.

„Sie erhalten .2 M zurueck"
oder „Sie erhalten 1.5 M zurueck".

Die unterschiedlichen Fälle sind in die Form *.** zu überführen, d. h. beispielsweise:

.2	...	0.20
.25	...	0.25
1.5	...	1.50
1.75	...	1.75
1	...	1.00

Es sind die Zahlen R in Vorkommazahl INT(R) und Nachkommazahl R-INT(R) zu zerlegen:

R	INT(R)	R-INT(R)
.2	0	.2
.25	0	.25
1.5	1	.5
1.75	1	.75
1	1	0

Nachdem diese Zahlen in Zeichenketten gewandelt wurden, kann man sie zeichenkettenmäßig verknüpfen. Die numerischen Werte können durch die BASIC-Funktion STR$ mit dem Format

```
STR$(zahl)
```

in eine Zeichenkette umgewandelt werden, so daß sich die stellenmäßig gewünschte Form des Restbetrages durch

```
R$ = STR$(INT(R))+"."+MID$(STR$(R-INT(R))+"00",3,2)
```

ergibt.

Die Umrahmung der S-Bahnfahrkarte stellt ein Rechteck aus 4 Strecken dar. Jede dieser Strecken wird von einem Punkt P(xa,ya) zu einem Punkt Q(xe,ye) in einer Farbcodierung f durch die Anweisung

```
LINE xa, ya, xe, ye [,f]
```

gezeichnet.

Für die Ausgabe von Informationen auf dem Bildschirm sind Zeichen- und Punktraster zu unterscheiden. Jedes Zeichen wird innerhalb eines 8 * 8 Bildpunktrasters dargestellt. Bei den Kleincomputern KC 85/2 und Nachfolger kann man jeden einzelnen Bildpunkt (256 * 320) adressieren. Beim KC 85/1 kann nur jedes Zeichen auf dem Bildschirm – 24 Zeilen * 40 Spalten – adressiert werden.
Die folgende Abbildung zeigt Zeichen- und Punktraster für die KC 85/2 oder /3 mit je einem Befehlsbeispiel.

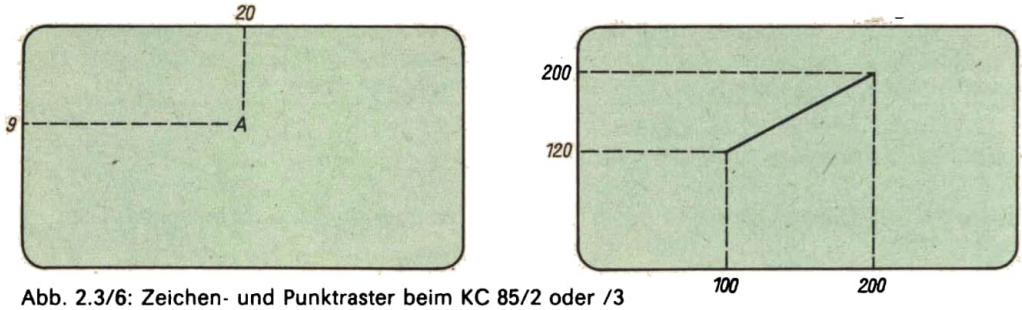


Abb. 2.3/6: Zeichen- und Punktraster beim KC 85/2 oder /3

Programm

```
10  REM SIMULATION EINES VERKAUFS VON S-BAHN FAHRAUSWEISEN
20  REM MITTELS AUTOMAT  ============================
30  REM Programm mit ein-, zwei- und mehrseitiger Auswahl
40  REM numerische Variablen:
50  REM   PS      - Preisstufe
60  REM   P       - Preis
70  REM   G       - Geld
80  REM   R       - Restgeld
90  REM Zeichenkettenvariablen:
100 REM   P$      - Preis
110 REM   G$      - Geld
120 REM   R$      - Restbetrag
130 REM -------------------------------------------------
200 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
210 PRINT "Bitte Preisstufe dem Aushang entnehmen"
220 PRINT : PRINT "und eingeben!"
230 PRINT AT(22,3);"Eingabe abschliessen mit <ENTER>!"
240 PRINT : INPUT ""; PS
250 CLS : PRINT "Sie waehlten Preisstufe"; PS
260 IF NOT(PS=INT(PS) AND PS>0 AND PS<9) THEN 810
270   ON PS GOTO 280,290,300,310,320,330,340,350
280     P$ = "0.20" : GOTO 360
290     P$ = "0.30" : GOTO 360
300     P$ = "0.50" : GOTO 360
310     P$ = "0.70" : GOTO 360
320     P$ = "1.00" : GOTO 360
330     P$ = "1.10" : GOTO 360
340     P$ = "1.20" : GOTO 360
350     P$ = "1.30"
360   REM ende der mehrseitigen auswahl
370   P = VAL(P$)
380   PRINT : PRINT "Fahrpreis: ";P$;" M"
390   WINDOW 5,22,0,39 : COLOR 0,2 : CLS
400   PRINT "Bitte Geld eingeben," : PRINT
410   PRINT "mindestens in Hoehe" : PRINT
420   PRINT "des Fahrpreises!" : PRINT
430   PRINT AT(21,3);"Eingabe abschliessen mit <ENTER>!"
```

```
440   INPUT "";G$ : G = VAL(G$)
450   G$ = STR$(INT(G))+"."+MID$(STR$(G-INT(G))+"00",3,2)
460   IF NOT(G > 0) THEN 770
470     IF NOT(P <= G) THEN 700
480       R = G - P : CLS
490       PRINT "Gezahlt: ";G$;" M"
500       IF NOT(R > 0) THEN 550
510         WINDOW 18,22,0,19 : COLOR 0,6 : CLS
520         R$=STR$(INT(R))+"."+MID$(STR$(R-INT(R))+"00",3,2)
530         PRINT "Sie erhalten" : PRINT
540         PRINT R$;" M zurueck."
550       REM ende der einseitigen auswahl
560       WINDOW 5,22,20,39 : COLOR 0,6 : CLS
570       PRINT AT(7,26);"Berliner"
580       PRINT AT(9,27);"S-Bahn"
590       PRINT AT(12,24);"gueltig fuer"
600       PRINT AT(13,25);"eine Fahrt"
610       PRINT AT(14,24);"bis zu einem"
620       PRINT AT(15,25);"Bahnhof der"
630       PRINT AT(18,25);"Preisstufe";PS
640       PRINT AT(20,23);"(siehe Aushang)"
650       LINE 178,208,310,208,7
660       LINE 178, 80,310, 80,7
670       LINE 178,208,178, 80,7
680       LINE 310,208,310, 80,7
690     GOTO 750
700       CLS : PRINT "Gezahlt: ";G$;" M"
710       WINDOW 15,22,0,39
720       COLOR 15,6 : CLS
730       PRINT "zu wenig Geld eingegeben!"
740       PRINT : PRINT G$;" M zurueck"
750     REM ende der zweiseitigen auswahl 3
760   GOTO 790
770     WINDOW 10,23,0,39 : COLOR 0,6 : CLS
780     PRINT "Nicht gezahlt!"
790   REM ende der zweiseitigen auswahl 2
800 GOTO 870
810   WINDOW 9,31,0,39 : COLOR 15,6 : CLS
820   PRINT "Eingabe nicht korrekt!" : PRINT
830   PRINT "Es darf nur" : PRINT
840   PRINT "eine Zahl" : PRINT
850   PRINT "zwischen 1 und 8" : PRINT
860   PRINT "eingegeben werden!"
870 REM ende der zweiseitigen auswahl 1
880 END
```

Auch hier dienen die leeren PRINT-Anweisungen sowie die verschiedenen WINDOW-Anweisungen einer übersichtlicheren Bildschirmgestaltung. Durch verschiedene Farbgebungen von Bildschirmausschnitten – mit COLOR v,h – wird der Blick des Nutzers auf Aktuelles bzw. Wichtiges gerichtet.

Zusammenfassung

Die **Auswahl** ist eine Steuerstruktur. Wir unterscheiden dabei die

- einseitige Auswahl,
- zweiseitige Auswahl (Alternative),
- mehrseitige Auswahl (Fallauswahl).

In Abhängigkeit von einer Bedingung wird bei der Auswahl entschieden, welche Folge von Anweisungen als nächste ausgeführt wird.

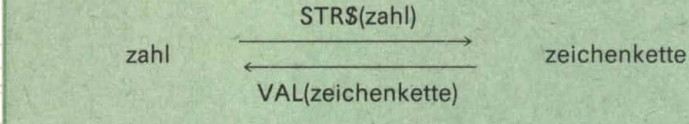
In **Bedingungen** können

- logische Operatoren,
- arithmetische Operatoren und
- Vergleichs-Operatoren

vorkommen. Die Bedingung kann aus logisch verknüpften Teilbedingungen zusammengesetzt sein. Die Priorität der Operatoren bestimmt die Auswertungsreihenfolge.
Die Bildung von Teilketten ermöglicht die Funktion **MID$** mit dem Format

MID$(zeichenkette,ab–position [,zeichenanzahl]).

Die Funktionen **STR$** und **VAL** ermöglichen Datenumwandlungen



Mit dem Befehl

LINE xa,ya,xe,ye [,f]

kann auf dem Bildschirm eine Linie vom Anfangspunkt mit den Koordinaten xa, ya zum Endpunkt xe, ye in einer wählbaren Farbe f gezeichnet werden.

Aufgaben

1. Testen Sie das Programm „Anrede aufgrund PKZ" mit Ihrer eigenen Personenkennzahl und mit denen Ihrer Mitschüler!

2. Schreiben Sie ein Programm zur Berechnung des Funktionswertes zu einem eingegebenen x-Wert für die Funktion

   $$f(x) = \frac{1}{x^4 - 10x^3 + 35x^2 - 50x + 24}\ !$$

   Im Fall, daß der Nenner Null ist, ist dem Programmnutzer mitzuteilen, daß für den gewählten x-Wert die Funktion nicht definiert ist.

3. Bei einem Abschlusszeugnis werden Zensuren in der Wortform geschrieben. Entwikkeln Sie ein Programm, das eine als Zahl eingegebene Zensur in ihrer Wortform ausgibt!

4. Schreiben Sie ein Programm, das das Zeichnen einer Strecke in Abhängigkeit von der mit RND(1) gebildeten Zufallszahl veranlaßt! Die Strecke soll vom Punkt P(200,200) zum Punkt Q(300,100) für RND(1) > 0.5 und zum Punkt Q(100,100) für RND(1) <= 0.5 gezeichnet werden.

5. Pädagogen, Arbeiter und technische Angestellte in den Einrichtungen der Volksbildung erhalten ab vollendetem zweiten Dienstjahr jährlich eine zusätzliche Vergütung. Die Höhe der zusätzlichen Vergütung ist von der Anzahl der Dienstjahre abhängig. Sie beträgt

   nach 2 Dienstjahren 4 Prozent bis max. 450,- M,
   nach 5 Dienstjahren 6 Prozent bis max. 600,- M und
   nach 10 Dienstjahren 8 Prozent bis max. 750,- M

   des Bruttoeinkommens des letzten Jahres.
   Schreiben Sie ein Teilprogramm für ein Lohn- und Gehaltsprogramm, das die jährliche zusätzliche Vergütung ermittelt! Der Einfachheit halber sind das Bruttojahresgehalt und die Anzahl der Dienstjahre über die Tastatur einzugeben.

6. Aus den Informationen über Leistungen und Gebühren der Deutschen Post kann man folgende Tabelle der Telegrammgebühren zusammenstellen.
   Wortgebühr für Telegramme (Mindestgebühr 10 Wörter):

Bezeichnung	Kürzel	Ortsverkehr Mark	Fernverkehr Mark
Gewöhnliche Telegramme	T	–,10	–,15
Dringende Telegramme	DT	–,20	–,30
Brieftelegramme	BT	–,05	–,05
Not-Telegramme	NT	gebührenfrei	
Staatstelegramme	ST	–,10	–,15
Wettertelegramme	WT	–,05	–,075*
Pressetelegramme			
– gewöhnliche	P	–,10	–,10
– dringende	DP	–,20	–,20

* Gebühren für Wettertelegramme werden auf volle 5 Pfennige aufgerundet.

   a) Schreiben Sie ein Teilprogramm, das die Gebühren für Wettertelegramme auf volle 5 Pfennige aufrundet!
   b) Entwickeln Sie einen Algorithmus zur Berechnung von Telegrammgebühren in Form eines Struktogramms und übersetzen Sie dieses in ein Programm zur Berechnung von Telegrammgebühren! Es ist davon auszugehen, daß die Art des Telegramms durch ein Kürzel aus der Menge {T, DT, BT, NT, ST, WT, P, PD}, die Wörteranzahl und die Verkehrsart (O für Ortsverkehr, F für Fernverkehr) über Tastatur eingegeben werden.

7. Zur Steuerung von Bewegungen – insbesondere in Spielen – gibt es für den KC 85/1 Spielhebel (engl. joystick). Mit der BASIC-Funktion

```
JOYST(nummer)
```

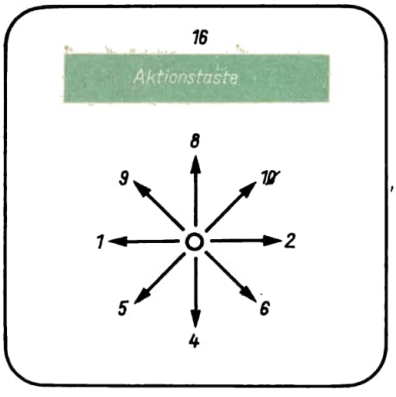


Abb. 2.3/7: Spielhebel

mit nummer=1 für Spielhebel 1, mit nummer=2 für Spielhebel 2 wird der Spielhebel abgefragt, dessen Nummer in Klammern steht. Bei A=JOYST(1) erhält die Variable A einen numerischen Wert, der durch die Stellung des Spielhebels 1 bestimmt wird und der aus der skizzierten Draufsicht auf einen Spielhebel zu entnehmen ist. Wird der Hebel 1 z. B. nach unten gedrückt, so ist A = 4. Drückt man die Aktionstaste, so ist A = 16.
Schreiben Sie ein Programm, das bei Betätigen des Spielhebels 5 Sterne von der Kursorposition 10. Zeile und 19. Spalte aus in Richtung der Spielhebelbewegung zeichnet! Wird die Aktionstaste gedrückt, so soll ein akustisches Signal mit der BASIC-Funktion BEEP gegeben werden.

2.3.3. Wiederholung

Die **Wiederholung** (auch Zyklus, Schleife oder Laufanweisung genannt) ist eine Steuerstruktur.
Wir unterscheiden:

- die **Zählschleife**
  (auch zählergesteuerte Schleife genannt),
- die **Solangeschleife**
  (auch While-Schleife genannt),
- die **Wiederholschleife**
  (auch Repeat-Schleife genannt).

Zählschleife

Wenn bekannt ist, wie oft eine Anweisungsfolge wiederholt werden soll, läßt sich die Wiederholung dieser Anweisungsfolge mit einer Zählschleife steuern.

In den drei Notationsformen (S. 77) bedeuten:

- l Name der Laufvariablen
- anf Anfangswert
- end Endwert
- sw Schrittweite.

Die Abbildung 2.3/8 zeigt unten eine Form des Struktogramms für die Zählschleife, die im Zusammenhang mit BASIC verwendet wird. Wir werden im weiteren die obere Form verwenden.

Aus einer großen Anzahl von Einsatzfällen werden hier nur wenige Beispiele angeführt. Wie auch schon in dem im Kapitel 1.4. beschriebenen Beispielalgorithmus zu erkennen war, ist die zählergesteuerte Schleife so leicht erfaßbar, daß hier für die Beispiele gleich die BASIC-Programme angegeben werden.

Die Zählschleife in den Notationsformen

verbal formalisierte Notation	Struktogramm
FÜR I = anf BIS end [SCHRITT sw] folge NÄCHSTES I	FÜR I = anf BIS end [SCHRITT sw] folge FÜR I = anf BIS end [SCHRITT sw] folge NÄCHSTES I

Umsetzung in BASIC

```
ak    FOR I = anf TO end [STEP sw]
al        folge
am    NEXT I
an    REM ende der zaehlschleife
```

Abb. 2.3/8: Struktogrammformen der Zählschleife

1 Erstes Anwendungsbeispiel zur Zählschleife

Eine anfangs senkrecht an einer Hauswand lehnende 200 cm lange Leiter rutscht in die waagerechte Lage. Dabei gleitet das obere Ende der Leiter mit den Koordinaten XO,YO längs der Wand nach unten und das untere Ende der Leiter mit den Koordinaten XU, YU von der Ecke aus waagerecht nach rechts.
Die Abbildung 2.3/9 zeigt eine Momentaufnahme zu dem Zeitpunkt, zu dem das untere Ende der Leiter sich L cm nach rechts bewegte.

Wie sieht die Hüllkurve aus?

Schreiben Sie ein Programm, das diesen Gleitvorgang in N Momentaufnahmen demonstriert!

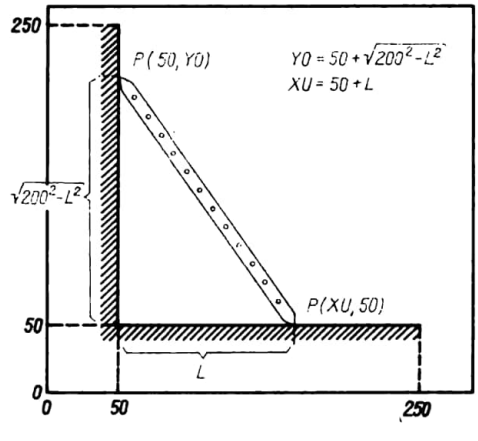


Abb. 2.3/9: Momentaufnahme

```
10  REM HUELLKURVE EINER RUTSCHENDEN LEITER
20  REM ==========================
30  REM  numerische Variablen:
40  REM    XO, YO  -  Koordinaten oberer Punkt
50  REM    XU, YU  -  Koordinaten unterer Punkt
60  REM    L       -  Laufvariable
70  REM    N       -  Anzahl Momentaufnahmen
80  REM ---------------------------------------
100 WINDOW 0,31,0,39 : COLOR 7,4 : CLS
110 REM  Ausgangssituation:
120 REM  Leiter steht senkrecht
130 XO = 50 : YO = 250
140 XU = 50 : YU = 50
150 INPUT "Anzahl der Momentaufnahmen:"; N
160 REM Momentaufnahmen:
170 REM Es bleiben konstant: XO=50 YU=50
180 FOR L=0 TO 200 STEP 200/N
190     XU = 50 + L
200     YO = 50 + SQR(200*200-L*L)
210     LINE XO,YO,XU,YU,7
220 NEXT L
230 END
```

Weil XO und YU konstant bleiben, kann die LINE-Anweisung auch heißen:

```
210     LINE 50,YO,XU,50,7
```

Die Farbkodierung 7 am Ende der LINE-Anweisung bewirkt, daß die einzelnen Linien weiß gezeichnet werden. In der Programmzeile 100 wurde mit COLOR 7,4 festgelegt, daß mit der Vordergrundfarbe 7 (weiß) auf einem grünen Hintergrund (Farbcodierung 4) geschrieben wird.
In der BASIC-Version des KC 85/2 gibt es die LINE-Anweisung nicht. Das Zeichnen einer Linie kann nur durch Setzen der einzelnen Bildpunkte erreicht werden. Die Anweisung dafür heißt PSET und hat das Format

```
PSET x, y [,f]
```

Die Anweisung bewirkt das Setzen des Bildpunktes mit den Koordinaten x, y in der Farbe f. Wird die Farbe nicht angegeben, so gilt die zuletzt gesetzte Zeichenfarbe.

Mit der Anweisungsfolge

```
121 FOR Y=50 TO 250
122    PSET 50,Y
123 NEXT Y
124 FOR X=50 TO 300
125    PSET X,50
126 NEXT X
```

werden Linien für die Hauswand und den Erdboden gezeichnet. In ähnlicher Weise kann man mit PSET auch einen Kreis zeichnen. Im BASIC des KC 85/3 gibt es dafür die Anweisung CIRCLE mit dem Format

```
CIRCLE xm, ym, r [,f]
```

Bei Abarbeitung dieser Anweisung wird ein Kreis mit dem Radius r um den Mittelpunkt mit den Koordinaten xm, ym in der Farbe f gezeichnet.

Da man oft nur den Hintergrund (das Papier, engl. paper) oder den Vordergrund (die Tinte, engl. ink) farblich ändern möchte, gibt es die beiden BASIC-Anweisungen

```
PAPER h     und     INK v
```

Die Anweisungsfolge

```
INK 7 : PAPER 4     entspricht     COLOR 7,4
```

Die folgende *Farbcodetabelle* ist für die KC 85/2 und /3 gültig.

Vordergrundfarbe	Codezahl	Hintergrundfarbe
schwarz	0	schwarz
blau	1	blau
rot	2	rot
purpur	3	purpur
grün	4	grün
türkis	5	türkis
gelb	6	gelb
weiss	7	weiss
schwarz	8	
violett	9	
orange	10	
purpurrot	11	
grünblau	12	
blaugrün	13	
gelbgrün	14	
weiss	15	

Zweites Anwendungsbeispiel zur Zählschleife

2 Schreiben Sie ein Programm, das einen Überblick über alle Farbkombinationen der 8 möglichen Hintergrundfarben (0 bis 7) mit den 16 Vordergrundfarben (0 bis 15) ermöglicht! In einem 2 Zeilen hohen und 16 Spalten breiten Fenster der jeweiligen Hintergrundfarbe ist 16mal das Zeichen ,*' in den 16 Vordergrundfarben darzustellen.

Algorithmus

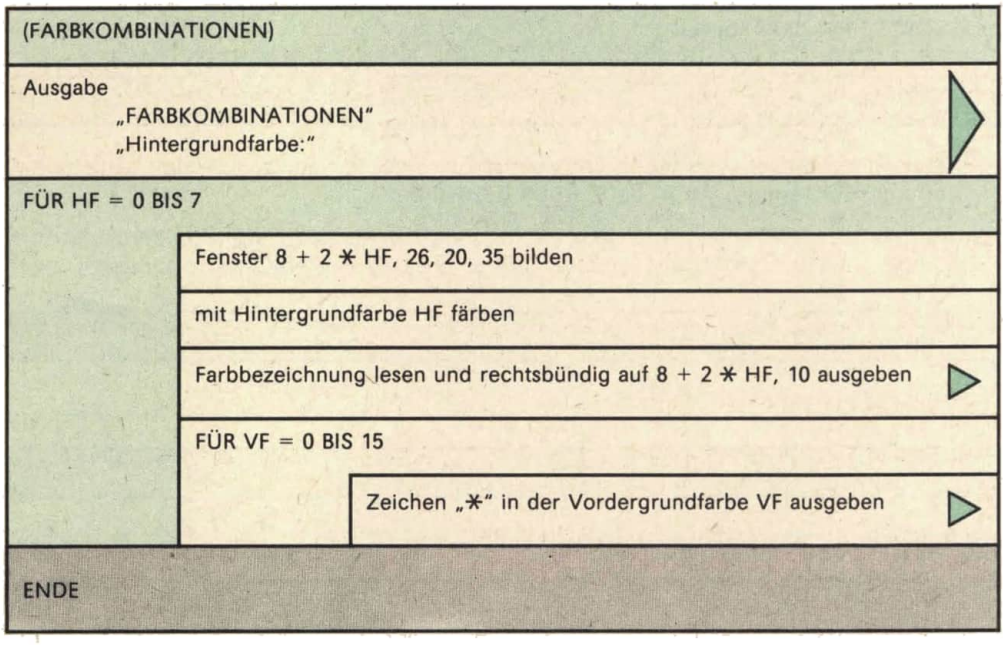


Abb. 2.3/10: Struktogramm „Farbkombinationen"

Programm

```
10  REM FARBKOMBINATIONEN
20  REM ================================
30  REM numerische Variablen:
40  REM   VF  -  Code Vordergrundfarbe
50  REM   HF  -  Code Hintergrundfarbe
60  REM Zeichenkettenvariable:
70  REM   F$  -  Bezeichnung Hintergrundfarbe
80  REM ----------------------------------------
100 WINDOW 0,31,0,39 : COLOR 0,7 : CLS
110 PRINT AT(2,11);"FARBKOMBINATIONEN"
120 PRINT AT(6,0);"Hintergrundfarbe:"
130 FOR HF = 0 TO 7
140     WINDOW 8+HF+HF,26,20,35
150     PAPER HF : CLS
160     READ F$
170     F$ = RIGHT$("    "+F$,7)
180     PRINT AT(8+HF+HF,10);F$
190     FOR VF = 0 TO 15
200        PRINT INK VF;"* ";
210     NEXT VF
220     REM ende innere zaehlschleife
230 NEXT HF
240 REM ende aeussere zaehlschleife
```

```
250 END
260 DATA schwarz,blau,rot,purpur,gruen
270 DATA tuerkis,gelb,grau
```

Um im Endbild (acht jeweils 2 Zeilen hohe und 16 Spalten breite) Fenster in der gewünschten Hintergrundfarbe zu bekommen, kann man die Fensterfolge

```
WINDOW 8,26,20,35
WINDOW10,26,20,35
WINDOW12,26,20,35
...
WINDOW22,26,20,35
```

bilden, in dem der Anfangszeilenparameter sich aus 8+2*zählvariable ergibt. Da die Variable für die Hintergrundfarbe HF von 0 bis 7 läuft, kann sie auch für die Bildung der Fensterfolge genutzt werden. Wie im vorigen Kapitel begründet, werden Operationen höherer Stufe möglichst durch Operationen niedrigerer Stufe ersetzt. So wird im Programm statt 8+2*HF besser 8+HF+HF programmiert. Das jeweilige Fenster wird schließlich mit der Farbe zur Code-Nummer 0 (schwarz), 1 (blau), 2 (rot), ..., 7 (grau) mit den Befehlen

```
PAPER HF : CLS
```

eingefärbt. Das letzte Fenster umfaßt 5 Zeilen in der Farbe grau, die auch für den restlichen Bildschirm als Hintergrundfarbe in Zeile 100 gewählt ist, so daß der Eindruck der Zweizeiligkeit der Fenster erhalten bleibt.

Um eine ästhetisch ansprechende Gestaltung des Bildschirms zu erreichen, werden die Farbbezeichnungen des Hintergrunds neben das Feld der Farbkombinationen in der Hintergrundfarbe rechtsbündig ausgegeben:

schwarz
blau
rot
purpur
grün
türkis
gelb
grau

Wie kann die gewünschte Rechtsbündigkeit erzielt werden? Man sieht, daß die längste Farbbezeichnung 7 und die kürzeste 3 Buchstaben umfaßt. Es wird eine 4 Leerzeichen umfassende Zeichenkette " " mit der Farbbezeichnung F$ verknüpft. Man erhält z. B. die Zeichenketten " schwarz"," rot". Nun ist eine Teilkette aus den jeweils 7 von rechts gezählten Zeichen zu bilden. In BASIC gibt es die Standardfunktionen RIGHT$ und LEFT$ mit dem Format

```
RIGHT$(zeichenkette, zeichenanzahl)
LEFT$(zeichenkette, zeichenanzahl)
```

die aus ‚zeichenkette' vom rechten bzw. linken Ende beginnend eine Teilkette der Länge

‚zeichenanzahl' herauskopiert. Muß man in einer langen Zeichenkette eine bestimmte Teilkette suchen, so gibt es dafür die Standardfunktion **INSTR** mit dem Format

```
INSTR(zeichenkette1,zeichenkette2)
```

Das Ergebnis der Anwendung dieser Funktion ist die Position, ab der ‚zeichenkette2' vollständig in ‚zeichenkette1' steht. Wird ‚zeichenkette2' nicht vollständig gefunden, so ist das Ergebnis der Funktionswert 0.

Im Programm werden die Farbbezeichnungen in DATA-Zeilen bereitgestellt, die innerhalb des Programms mit der Anweisung READ F$ gelesen werden. Dabei steht zu Beginn des Lesevorgangs ein Zeiger (der Datenzeiger) auf der ersten Bezeichnung in der ersten DATA-Zeile, und bei jedem Lesen rückt der Zeiger auf die nächste Bezeichnung weiter. Daher ist zu sichern, daß ausreichend viele Daten in DATA-Zeilen erfaßt sind.
Mit

```
PRINT [farbfestlegung;][AT(zeile,spalte);] ausgabeliste
```

kann eine lokale Änderung von Vorder- oder Hintergrundfarbe erfolgen. Die gewünschte Rechtsbündigkeit und Farbgestaltung der Farbbezeichnungen werden durch die Programmzeilen

```
160         READ F$
170         F$ = RIGHT$("     "+F$,7)
180         PRINT COLOR HF,7; AT(8+HF+HF,10)+ F$
```

realisiert.

Drittes Anwendungsbeispiel zur Zählschleife

3 Für die Anzeige von Meßwerten auf einem Bildschirm – z. B. im Physikunterricht – ist ein Programm zu entwickeln, das das Anzeigen von vierstelligen Zahlen im Großformat auf dem Bildschirm veranlaßt. Jede Ziffer soll im 7*5-Zeichen-Raster dargestellt werden.

Eingabe: Vierstellige Zahl über Tastatur.
Verarbeitung: Die Zahl ist ziffernweise zu analysieren. Zu jeder Ziffer werden die Daten für die großformatige Ziffer herausgesucht.
Ausgabe: Eingegebene Zahl in großformatigen Ziffern.

Bei jeder Zahleneingabe liegt fest, wie oft Handlungen zu wiederholen sind. Da eine einzugebende Zahl aus genau vier Ziffern bestehen soll, ist die Analyse für 4 Ziffern durchzuführen. Jede große Ziffer soll aus 7 Zeilen mit jeweils 5 Spalten bestehen. Deshalb sind zwei ineinander geschachtelte Zählschleifen zu programmieren.
Als Beispiele sind für die Ziffern 1 und 6 zwei Entwürfe in Abbildung 2.3/11 dargestellt.
Da auch die nichtgesetzten Zeichen von Bedeutung sind, muß jedes Datum genau 5 Zeichen umfassen, die z. B. der Vollkursor "■" oder das Leerzeichen " " sein können. Diese werden in Anführungsstriche gesetzt. Die Daten für die "1" nach Abbildung 2.3/11 sind durch folgende DATA-Zeile erfaßt:

```
710 DATA "  ■  ","  ■■  ","■ ■  ","  ■  ","  ■  ","  ■  ",
         "  ■■■  "
```

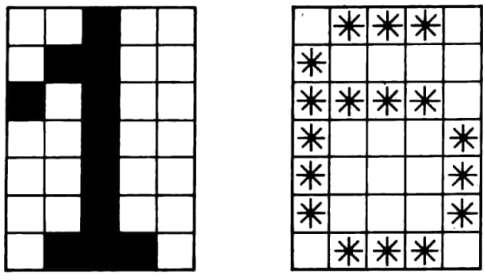


Abb. 2.3/11: Zeichenentwürfe

Sind die beiden ersten Anführungsstriche vergessen worden, so werden alle Leerzeichen vor dem ersten Komma ignoriert, und es erscheint auf dem Bildschirm folgendes Zeichen.

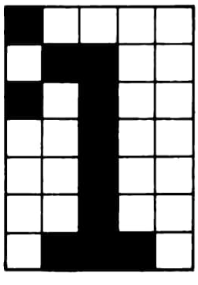


Abb. 2.3/12: Ausgabe des fehlerhaften Zeichens

Das entspricht aber nicht der entworfenen "1" nach Abbildung 2.3/11.

Beim Start eines Programms steht der Datenzeiger auf dem ersten Datum derjenigen DATA-Zeile, die die niedrigste Zeilennummer besitzt. Bei jeder READ-Anweisung im Programm rutscht der Datenzeiger eine Position in der Datenliste weiter. Da aber die zu lesenden Daten von der einzugebenden 4-stelligen Zahl abhängen, muß der Datenzeiger in Abhängigkeit von der auszugebenden großformatigen Ziffer gesetzt werden. Das wird in BASIC mit der Anweisung RESTORE mit dem Format

```
RESTORE [zeilennummer]
```

realisiert. Fehlt die Angabe des Parameters ‚zeilennummer', dann wird der Datenzeiger auf die DATA-Anweisung mit der niedrigsten Zeilennummer gesetzt.

Programm

```
100 REM ZAHLENGROSSANZEIGE - 1. Version   ==========/=====
110 REM numerische Variablen:
120 REM   I,J   - Laufvariable
130 REM   ZI    - Ziffer
```

```
140 REM Zeichenkettenvariablen:
150 REM   ZA$   - vierstellige Zahl
160 REM   ZI$   - Ziffer
170 REM   ZEI$  - Zeile einer Grosszziffer
190 REM ------------------------------------------
200 REM Eingabe einer vierstelligen Zahl
210 WINDOW 0,31,0,39 : COLOR 0,6 : CLS
220 WINDOW 29,31,5,39 : COLOR 7,1 : CLS
230 INPUT "vierstellige Zahl eingeben:"; ZA$
240 REM Analyse einer Zahl und
250 REM Ausgabe der Grosszziffern
260 FOR I = 1 TO 4
270    ZI$ = MID$(ZA$,I,1)
280    ZI = VAL(ZI$)
290    IF ZI = 0 THEN ZI = 10
300    REM beginn: mehrseitige auswahl
310    ON ZI GOTO 320,330,340,350,360,370,380,390,400,405
320       RESTORE 710 : GOTO 410
330       RESTORE 720 : GOTO 410
340       RESTORE 730 : GOTO 410
350       RESTORE 740 : GOTO 410
360       RESTORE 750 : GOTO 410
370       RESTORE 760 : GOTO 410
380       RESTORE 770 : GOTO 410
390       RESTORE 780 : GOTO 410
400       RESTORE 790 : GOTO 410
405       RESTORE 800
410    REM ende der mehrseitigen auswahl
420    WINDOW 8,17,(I-1)*8+1,(I-1)*8+6 : COLOR 0,6 : CLS
430    FOR J = 1 TO 7
440       READ ZEI$
450       PRINT ZEI$
460    NEXT J
470 NEXT I
480 PAUSE 50 : WINDOW 0,31,0,39 : COLOR 7,1 : CLS : END
490 REM ------------------------------------------------
700 REM Daten fuer die Ziffern 1 bis 0
710 DATA "    ■    "," ■■    "," ■■  ■■   "," ■■■■  ",
         "    ■■   "," ■■■   "," ■■  ■■■  ":REM  1
720 DATA "  ■■■■  "," ■■■■■ "," ■■■■■■■■ "," ■■■■■■ ",
         "    ■■   "," ■■■■■ "," ■■■■■■■■":REM  2
730 DATA "  ■■■■  "," ■■■■■ "," ■■■■■■■■ "," ■■■■■■ ",
         "       ■■"," ■■■■■ "," ■■■■■■■■":REM  3
740 DATA "        *"," *  *  *"," *  *  *  "," *  *  *",
         " * * * * *"," *  *  *"," *  *  *  ":REM  4
750 DATA " * * * * *"," *  *  *"," *  *  *  "," *  *  *",
         "        *"," *  *  *"," *  *  *  ":REM  5
760 DATA "  * * *  "," *  *  *"," *  *  *  "," *  *  *",
         " *      *"," *  *  *"," *  *  *  ":REM  6
770 DATA " * * * * *"," *  *  *"," *  *  *  "," *  *  *",
         " *      *"," *  *  *"," *  *  *  ":REM  7
```

```
780 DATA "  * * *  ","*      *","  * * *  ","*      *",
         " *     *","*      *","  * * *  ":REM  8
790 DATA "  * * *  ","*      *","*      *","  * * * *",
         "      *","*      *","  * * *  ":REM  9
800 DATA "  * * *  ","*      *","*      *","*      *",
         " *     *","*      *","  * * *  ":REM  0
```

Wenn in der Anweisungszeile 800 das letzte Datum fehlt, meldet der Computer bei Eingabe einer Null ein

```
? OD ERROR IN 440
```

OD (out of data → außerhalb der Daten) weist darauf hin, daß durch DATA-Anweisungen nicht genügend Daten für READ-Anweisungen bereitgestellt wurden.
Wählt man als Eingabe die Zahl −1234, so erscheint auf dem Bildschirm 0123. Die eingegebene Zahl als Zeichenkette interpretiert, hat als erstes Zeichen das Minus. Der mit der Standardfunktion VAL ermittelte Wert für das Zeichen „−" ist 0. Testet man das Programm mit 12.34, so gibt der Computer in großen Ziffern 1 2 0 3 aus, da der Wert des dritten Zeichens – das ist der Dezimalpunkt – ebenfalls Null ist. Bei Eingabe der Zahl 1 ist die Großziffernanzeige 1 0 0 0. Daraus erkennt man, daß das Programm nur dann die gewünschten Ausgaben liefert, wenn alle vier Stellen einer vierstelligen ganzzahligen nichtnegativen Zahl eingegeben werden.
Noch eine Bemerkung zur Anweisungszeile 290. Warum wird vor der mehrseitigen Auswahl abgetestet, ob die isolierte Ziffer die „0" ist? Die Zeile 290 wird bei der Programmabarbeitung übergangen, wenn ein REM eingefügt wird:

```
290 REM   IF ZI = 0 THEN ZI = 10
```

Nach der Eingabe der Zahl 3030 bzw. 1A56 bzw. 1.234 erscheint auf dem Bildschirm in großen Ziffern 3 1 3 1 bzw. 1156 bzw. 1123. D. h., alle Zeichen, die nicht gleich 2 oder 3 oder 4 ... 9 sind, werden als die Großziffer interpretiert, die durch die erste RESTORE-Anweisung nach der ON ... GOTO-Anweisung zugewiesen wird.

4 Für die Ausgabe von Meßwerten sind neben den Zahlenwerten auch die Einheiten erforderlich. Wie kann die Großanzeige z. B. von ‚m' für Meter realisiert werden?

Mögliche Entwürfe für große Zeichen im 7*5-Raster zeigt Abbildung 2.3/13.

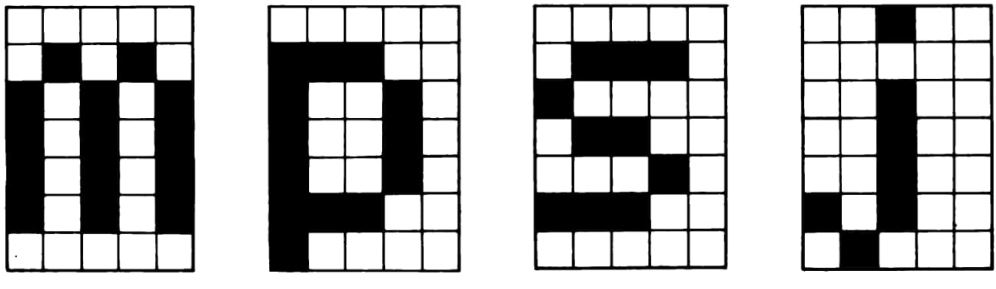


Abb. 2.3/13: Zeichenentwürfe

Wieder kann man jeweils 7 Zeichenketten als Zeilen eines großformatigen Zeichens in DATA-Anweisungen ablegen. Wie aber kann der Datenzeiger in Abhängigkeit vom Zeichen, das nun keine Ziffer ist, gesetzt werden?

Die VAL-Funktion kann zur Buchstabenunterscheidung nicht herangezogen werden, da sie für jedes Zeichen außer 1 bis 9 den Wert 0 liefert. Zeichen sind im Computer im sogenannten ASCII-Code verschlüsselt. ASCII ist die Abkürzung für American Standard Code for Information Interchange und vermittelt zu allen Zeichen (Ziffern, Buchstaben, Sonderzeichen) und zu Steuerzeichen aus 8 Bit bestehende Bitmuster, so daß der Computer sie erkennen und verarbeiten kann. Die Standardfunktion **ASC** mit dem Format

```
ASC(zeichenkette)
```

liefert für eine beliebige Zeichenkette die ASCII-Zahl des ersten Zeichens der angegebenen Zeichenkette.

Mit dem folgenden Programm werden für 10 beliebig eingegebenen Zeichen die ASCII-Zahlen ausgegeben:

```
10  REM ADRESSEN VON ZEICHEN IN
20  REM DER ZEICHENCODETABELLE ======================
30  REM numerische Variable:
40  REM   I    - Laufvariable
50  REM Zeichenkettenvariable:
60  REM   ZE$ - Zeichen
70  REM ----------------------------------------
100  WINDOW : COLOR 7,1 : CLS
110  PRINT : PRINT ,"ZEICHEN","ASCII- ZAHL"
120  FOR I = 1 TO 10
130     WINDOW 28,31,0,39 : COLOR 0,6 : CLS
140     INPUT "Zeichen eingeben:";ZE$
150     PRINT INK 7, PAPER 1; AT(5+I+I,14); ZE$
160     PRINT INK 7, PAPER 1; AT(5+I+I,28); ASC(ZE$)
170 NEXT I
180 END
```

Ein Ausschnitt aus dem Protokoll eines Probelaufs:

ZEICHEN	ASCII-ZAHL
E	69
A	65
a	97
1	49
2	50
&	38
*	42
ENDE	69
,	44
z	122

Wird das Zeichen " eingegeben, so kommt die Ausschrift

```
? FC ERROR IN 160
```

In der Anweisungszeile 160 liegt ein Fehler in einem Funktionsaufruf vor. Die Eingabe "

interpretiert der Computer als leere Zeichenkette "" und diese führt hier zum Programmabbruch.
Insgesamt gibt es die ASCII-Zahlen 0, 1, 2, ..., 255. Die Zeichenkettenfunktion **CHR$** (character → Schriftzeichen) vermittelt zu den ASCII-Zahlen das zugehörige Zeichen. Diese Funktion besitzt das Format

```
CHR$(ausdruck)
```

Dabei muß für den ganzzahligen Wert W des in der Klammer stehenden Ausdrucks gelten:

0 <= W <= 255.

Mit dem folgenden Programm kann man sich einen Überblick über die 224 druckbaren Zeichen des KC 85/3 verschaffen:

```
10 REM ZEICHENZUORDNUNG
20 REM ZU ASCII-ZAHLEN
30 WINDOW : CLS
40 FOR I = 32 TO 255
50    PRINT I; CHR$(I),
60 NEXT I
70 END
```

Im Programm wurden die Zahlen 0, 1, 2, ..., 31 in der Laufanweisung auf Zeile 40 ausgeschlossen, da diese ASCII-Zahlen unterschiedliche BASIC-Befehle und Steuerfunktionen auslösen, wie z. B. CHR$(13) für <ENTER>.

Der vom Programm erzeugten Tabelle kann man entnehmen, daß die Ziffern, die Großbuchstaben und die Kleinbuchstaben bestimmten Folgen von ASCII-Zahlen zugeordnet sind. Diese Zuordnung wird noch einmal in der Tabelle verdeutlicht.

ASCII-Zahlen-Gruppe	Bedeutung
48... 57	Ziffern 0 bis 9
65... 90	Großbuchstaben A bis Z
97...122	Kleinbuchstaben a bis z

Mit diesen Kenntnissen läßt sich ein Programm entwickeln, das zu eingegebenen Kleinbuchstaben die entsprechenden Zeichen im 7*5-Raster auf dem Bildschirm erzeugt. Es kann analog zum Programm für das dritte Beispiel gestaltet werden. Da die ASCII-Zahlen zu den Kleinbuchstaben a, b, c, ..., z die Zahlen 97, 98, 99, ..., 122 sind, müssen die Anweisungszeilen ab 280 im Programm ZAHLENGROSSANZEIGE geändert werden:

```
280     ZI = ASC(ZI$) - 96
```

Setzt man voraus, daß wirklich nur Kleinbuchstaben eingegeben werden, so kann ZI damit nur eine der ganzen Zahlen 1, 2, 3, ..., 26 sein. Die Liste nach GOTO besteht aus dreistelligen Anweisungsnummern und Trennzeichen. Da 26 * 4 = 104 ist, wird man nicht alle möglichen Anweisungsnummern in eine Programmzeile bekommen. Deshalb wird

die mehrseitige Auswahl in zwei Teile gegliedert. Damit entsteht ein Teil des Programms für Zahlengroßanzeige neu (2. Version).

```
290      IF ZI > 13 THEN  445
300      REM beginn der mehrseitigen auswahl
310      ON ZI GOTO 320,330,340,350,360,370,380,390,400,
  410,420,430,440
320          RESTORE 710 : GOTO 590
330          RESTORE 720 : GOTO 590
340          RESTORE 730 : GOTO 590
350          RESTORE 740 : GOTO 590
360          RESTORE 750 : GOTO 590
370          RESTORE 760 : GOTO 590
380          RESTORE 770 : GOTO 590
390          RESTORE 780 : GOTO 590
400          RESTORE 790 : GOTO 590
410          RESTORE 800 : GOTO 590
420          RESTORE 810 : GOTO 590
430          RESTORE 820 : GOTO 590
440          RESTORE 830 : GOTO 590
445          ZI = ZI - 13
450      ON ZI GOTO 460,470,480,490,500,510,520,530,540,
  550,560,570,580
460          RESTORE 840 : GOTO 590
470          RESTORE 850 : GOTO 590
...          ....... ...
570          RESTORE 950 : GOTO 590
580          RESTORE 960
590      REM ende der mehrseitigen auswahl
```

Die Programmzeilen 420 bis 490 der ersten Version des Programms ZAHLENGROSSANZEIGE (↗ S. 84) können in die zweite Version übernommen werden. Entweder schreibt man sie erneut ein, oder man numeriert sie mit

```
RENUMBER 420,490,600,10
```

vor der Eingabe der obigen Anweisungszeilen 290 bis 590 um. Mit diesen Veränderungen des Programms ZAHLENGROSSANZEIGE ist ein Programm zur Großanzeige der 26 Kleinbuchstaben des lateinischen Alphabets entstanden.

5 Wenn Texte fachlichen Inhalts zu schreiben sind, benötigt man oft spezielle Zeichen, z. B. griechische Buchstaben. So wie bei der Großanzeige von Ziffern oder Kleinbuchstaben ein Vollkursor oder ein Leerzeichen pro Zeichenfeld geeignet kombiniert wurden, um z. B. ein ‚m' im Großformat darzustellen, so besteht jedes druckbare Zeichen aus einer Ansammlung von hellen und dunklen Bildpunkten (Pixeln). Jedes Zeichenfeld besteht aus einem 8*8-Punkt-Raster. Da pro Rasterpunkt entweder ein Punkt gesetzt ist oder nicht, kann diese Information binär erfaßt werden. Ist ein Punkt gesetzt, schreibt man eine 1, sonst eine 0. Um noch die Position eines Rasterpunktes zu markieren, werden die Zeilen 0, 1, 2, 3, 4, 5, 6, 7 des 8*8-Rasters jeweils als eine 8-stellige Dualzahl aufgefaßt.
Achtstellige Dualzahlen zu merken, ist kompliziert, so daß man die zusammengehörigen 8 Stellen in zwei Vierergruppen unterteilt und dann nur noch eine zweistellige Hexadezimalzahl zu merken hat.

Davon ausgehend können nun die verschiedensten Zeichen selbst entworfen und dem Computer eingeprägt werden. So z. B. auch Schachfiguren, Spielkartensymbole, Buchstaben unterschiedlicher Alphabete, Schaltzeichen der Elektrotechnik, mathematische Symbole.

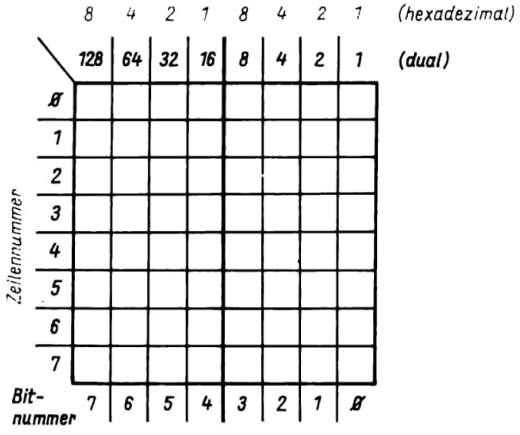


Abb. 2.3/14: Zeichenraster

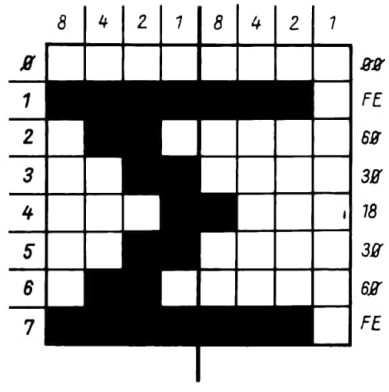


Abb. 2.3/15: Summenzeichen

Nach dem Zeichenentwurf auf Papier ist die Folge der 8 Bytes, die das Zeichen beschreiben, im Computer abzulegen. Im folgenden ist an einem Beispiel ein mögliches Vorgehen bei Nutzung des Kleincomputers KC 85/3 dargestellt.
Im Grundmenü des KC 85/3 gibt es das Kommando MODIFY. Mit diesem Kommando lassen sich die Inhalte der Speicherzellen ab einer angegebenen hexadezimalen Adresse modifizieren. Gestartet wird mit

```
%MODIFY BA00  <ENTER>
```

Der Computer bietet nun Byte für Byte zur Veränderung des Inhalts an. Die Folge der zweistelligen hexadezimalen Zahlen für ∑ wird eingegeben. Nach acht Eingaben ist folgendes auf dem Bildschirm zu sehen:

```
BA00 00
BA01 FE
BA02 60
BA03 30
BA04 18
BA05 30
BA06 60
BA07 FE
BA08 00
```

Aus dem MODIFY-Modus kommt man mit der Eingabe eines Dezimalpunktes und <ENTER> heraus, und der Computer zeigt das Bereitschaftszeichen auf der Betriebssystemebene %■ an.

Soll das Zeichen ∑ z. B. der ASCII-Zahl 160 zugeordnet werden, so ist der Zeiger, der auf die Adresse der Zeichenbildtabelle für die Zeichen mit den ASCII-Zahlen 160 bis 223 ge-

richtet ist, mit der Adresse BA00 zu belegen. Aus den Systemunterlagen zum KC 85/3 entnimmt man, daß dieser Zeiger die Anfangsadresse B7AA hat. Das Byte 00 ist auf die Adresse B7AA und das Byte BA ist auf die nächstfolgende Adresse B7AB mit MODIFY einzuschreiben.

Geht man dann zurück ins BASIC und gibt

```
PRINT CHR$(160) <ENTER>
```

ein, so erscheint in der nächsten Bildschirmzeile – wie gewünscht – das Zeichen $\sum$.

Weitere Anwendungsbeispiele zur Zählschleife

Für eine Menge, die aus n Elementen besteht, $n >= 1$, berechnet sich die Anzahl der verschiedenen Anordnungen dieser Menge zu $n!$ (sprich: n Fakultät). Dabei ist $n!$ definiert durch

$1! = 1$
$n! = (n\text{-}1)! * n$ für $n > 1$.

Es ist ein Programm zu entwickeln, mit dem der Computer $n!$ berechnet.

Da nach der Eingabe der natürlichen Zahl n genau festliegt, daß – ausgehend von der Definition von $n!$ – n-mal ein Produkt aus dem bisher ermittelten ‚Zwischenwert' und dem jeweils aktuellen Faktor zu bilden ist, eignet sich die Zählschleife. Bekam der ‚Zwischenwert' durch das Programm keinen Wert zugewiesen, so ist er standardmäßig Null. Kommt die Programmabarbeitung erstmals zur Produktbildung, so führt ein Faktor Null zum Produkt Null. Um diesen Fehler bei der Produktbildung zu vermeiden, muß dem ‚Zwischenwert' vor Eintritt in die Zählschleife ein Wert ungleich Null zugewiesen werden. Bei Produktbildungen ist der Anfangswert oft die Zahl 1 oder der Wert des ersten Faktors. Ein Lösungsalgorithmus in Form eines BASIC-Programms kann wie folgt aussehen:

```
10  REM BERECHNUNG VON N! ====================
20  REM numerische Variablen:
30  REM   I    - Laufvariable
40  REM   N    - Eingangsvariable
50  REM   NFAK - Zwischenwert
60  REM ------------------------------------
100 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
110 PRINT
120 PRINT TAB(10);"BERECHNUNG VON N!"
130 PRINT : PRINT
140 INPUT "N = ";
150 NFAK = 1
160 FOR I = 1 TO N
170    NFAK = NFAK * I
180 NEXT I
190 PRINT N;"! =";NFAK
200 END
```

● Testen Sie, ab welcher natürlichen Zahl es zu einem

```
? OV ERROR IN 160
```

kommt! Die Ausschrift OV (numerical overflow) besagt, daß der Betrag des Ergebnisses einer Berechnung größer als die größte Zahl der Computerzahlenmenge ist.

● Entwickeln Sie ein Programm zur Ausgabe der Fakultäten 1!, 2!, 3!, ..., k!, wobei k die größte der natürlichen Zahlen sein soll, die nicht zum OV ERROR führt!

7 Es ist ein Programm zur Berechnung der Summe $1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots+\frac{1}{2^{50}}$

zu entwickeln. Dabei sollen alle Zwischensummen in folgender Weise angezeigt werden:

a) in tabellarischer Form in den Bildschirmzeilen 9 bis 26,
b) so, daß alle Zwischensummen ab der Bildschirmstelle 28. Zeile, 9. Spalte übereinander und zwar in der Form *.***** ausgegeben werden, um die jeweiligen Veränderungen in den Dezimalstellen verfolgen zu können.

Eine erste Programmversion hat folgendes Aussehen:

```
10  REM SUMME VON LAENGEN BEI
20  REM FORTLAUFENDER HALBIERUNG
30  REM DER SUMMANDEN
40  REM numerische Variablen:
50  REM   I    - Indexvariable
60  REM   Z    - Zeilennummer
70  REM   SU   - Summand
80  REM   ZS   - Zwischensumme
90  REM Zeichenkettenvariablen:
100 REM   I$   - Index (rechtsbuendig)
110 REM   HZ$  - Hilfszeichenkette
120 REM   ZS$  - Zwischensumme (stellengerecht)
130 REM   D$   - Dummyvariable: Ausgabe-Unterbrechung
190 REM ------------------------------------------------
200 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
210 PRINT TAB(10);"BERECHNUNG DER SUMME"
220 PRINT
230 PRINT TAB(8);"     1    1    1          1  "
240 PRINT TAB(8);"1 + - + - + - +...+ --- "
250 PRINT TAB(8);"     2    4    8        2↑50"
260 PRINT : PRINT
270 PRINT"INDEX";TAB(10);"SUMMAND";TAB(26);"ZWISCHENSUMME"
280 WINDOW 9,26,0,26 : COLOR 0,6 : CLS
290 ZS=1 : SU=1 : Z=9 : REM anfangswerte
300 FOR I = 1 TO 50
310    I$ = RIGHT$(" "+STR$(I),2)
320    PRINT AT(Z,2);I$
330    PRINT AT(Z,10);SU
340    PRINT AT(Z,26);ZS
350    HZ$=LEFT$(STR$(ZS),2)+"."+MID$(STR$(ZS),4,5)
```

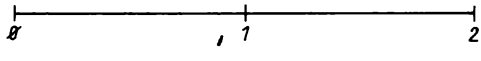
```
360     HZ$=HZ$+"00000"
370     ZS$=LEFT$(HZ$,8)
380     PRINT AT(30,10);"weiter mit <ENTER>"
390     PRINT COLOR 0,5;AT(28,9);"Zwischensumme";ZS$
400     SU=SU/2
410     ZS=ZS+SU
420     INPUT "";D$
430     IF Z=26 THEN WINDOW 9,26,0,39 : CLS : Z=8
440     Z=Z+1
450 NEXT I
460 WINDOW 31,31,39,39
470 END
```

Was liefert ein Programmtest dieser ersten Version?
Für I = 19 gibt der Computer erstmals als Zwischensumme die Zahl 2.00000 aus. Die Addition der folgenden Summanden, die durchaus nicht gleich 0 sind, führt nicht zur Änderung der Summe. Es stellt sich die Frage, ob das Ergebnis 2.00000 für

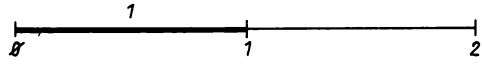
$$\sum_{i=0}^{50} \frac{1}{2^i}$$

überhaupt korrekt ist.

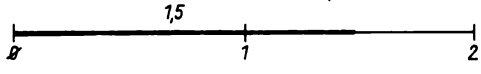
Die Summenbildung kann an einer Strecke der Länge 2 geometrisch veranschaulicht werden.



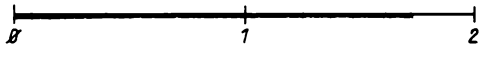
Der erste Summand entspricht der halben Länge der Gesamtstrecke.



Der zweite Summand entspricht der halben Länge der noch verbliebenen unmarkierten Strecke.



Der dritte Summand entspricht der halben Länge der noch verbliebenen unmarkierten Strecke



usw.

Demzufolge kann der Wert 2 auf diese Weise nach endlich vielen Schritten nicht erreicht werden. Da die Computerzahlenmenge jedoch diskret ist, sind Rundungen unumgänglich.
Die Bildung der Summe aus der
18. Zwischensumme des Computers 1.99999
und dem 19. Summanden 0.0000038147
(in der Form 3.8147E-6 auf dem Bildschirm angezeigt), führt zu der auf 6 Stellen gerundeten Zahl

```
2.00000
```

Da bei Fortsetzung des geschilderten Gedankengangs die Zahl 2 nicht überschritten wird, ist das Computerergebnis 2.00000 für die zu berechnende Summenformel kein schwerwiegender Fehler.

Wem es nur um die Zahlenergebnisse geht, erhält diese auch mit dem Programm:

```
500 CLS
510 ZS = 1 : SU = 1
520 FOR I = 1 TO 50
530     PRINT I,SU,ZS
540     SU = SU/2
550     ZS = ZS + SU
560 NEXT I
570 END
```

Die Mehrzahl der Anweisungen im Programm zum letzten Beispiel dient der Bildschirmgestaltung. So werden zur besseren Lesbarkeit des Textes auf dem Bildschirm durch die PRINT-Anweisungen in den Zeilen 220 und 260 Leerzeilen erzeugt. Für die Überschrift und für die Hinweise zum Fortsetzen in der Programmabarbeitung sind die Zeilen 210, 230, 240, 250 und 380 eingefügt. Die Überschrift für die tabellarische Ausgabe realisiert die Anweisung in der Zeile 270. Die farbliche Gestaltung der Tabelle – schwarze Schrift auf gelbem Hintergrund – wird in der Zeile 280 erzeugt. Die Zeile 390 sorgt für die Ausgabe in schwarzer Schrift auf türkisem Hintergrund. Die Anweisungen in den Zeilen 310, 350, 360 und 370 realisieren die Aufbereitung von Zahlen in gewünschter Form (rechtsbündig bzw. als Zahl mit einer Stelle vor dem Dezimalpunkt und fünf Stellen nach dem Dezimalpunkt).
Die gute Bildschirmgestaltung ist für den Programmnutzer wichtig, weil sie den Dialog Mensch – Computer unterstützt.

In den folgenden Beispielen steht die algorithmische Durchdringung des Problems im Mittelpunkt. Die Beispiele werden vorwiegend nur bis zum Algorithmus in Form eines Struktogramms besprochen.

8 Es ist näherungsweise der Inhalt I der Fläche zu berechnen, die durch die x-Achse, durch die Parallelen zur y-Achse durch $x=a$ bzw. $x=b$ und durch den Graph der Funktion $f(x)=x^2$ ($x \in \langle a,b \rangle$) begrenzt wird. In die Fläche werden erst n Rechtecke der Breite $h = (b-a)/n$ und der Längen $a^2, (a+h)^2, \ldots (a+(n-1)h)^2$ einbeschrieben. Zum zweiten wird die Fläche durch n Rechtecke der Breite $h = (b-a)/n$ und der Längen $(a+h)^2$, $(a+2h)^2$, $\ldots$ $(a+nh)^2$ überdeckt.
(Abbildung 2.3/16 und 2.3/17).

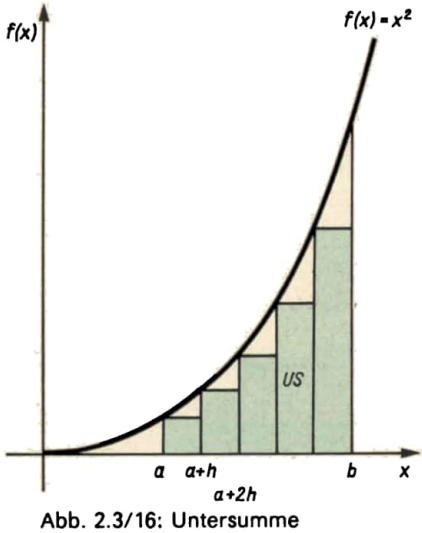


Abb. 2.3/16: Untersumme

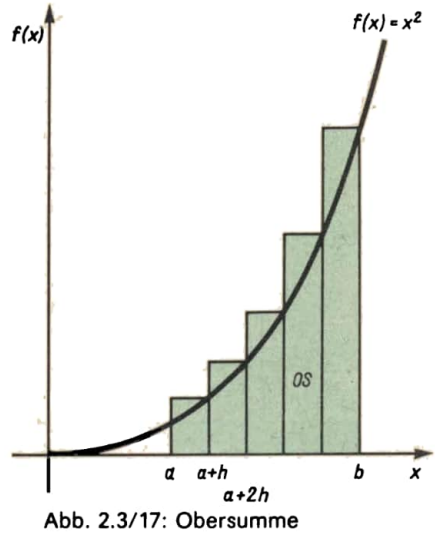


Abb. 2.3/17: Obersumme

Die Summe der Flächeninhalte der einbeschriebenen Rechtecke wird Untersumme *US* und die der überdeckenden Rechtecke Obersumme *OS* genannt. Sofort einsichtig ist, daß *US* <= *I* <= *OS* gelten muß.
Verfeinert man die Zerlegung des Intervalls <*a,b*> – d. h., die Rechteckzahl wächst und die Rechteckbreiten werden kleiner –, so sind die neue Untersumme *US*neu und die Obersumme *OS*neu verbesserte Näherungswerte für *I* und es gilt:
US <= *US*neu <= *I* <= *OS*neu <= OS.

Eingabe:
- untere Intervallgrenze *a*,
- obere Intervallgrenze *b*,
- Anfangsanzahl der Teilintervalle *N*,
- Anzahl der Verfeinerungen *V*

Verarbeitung:
- für jeweilige *N* werden Untersumme und Obersumme für die vorgegebene Funktion $f(x)=x^2$ im Intervall <*a,b*> berechnet,
- dann wird *V*-mal die Teilintervallzahl verdoppelt

Ausgabe:
- Anzahl der Teilintervalle,
- Länge der Teilintervalle (Schrittweite),
- die zugehörige Untersumme,
- die zugehörige Obersumme.

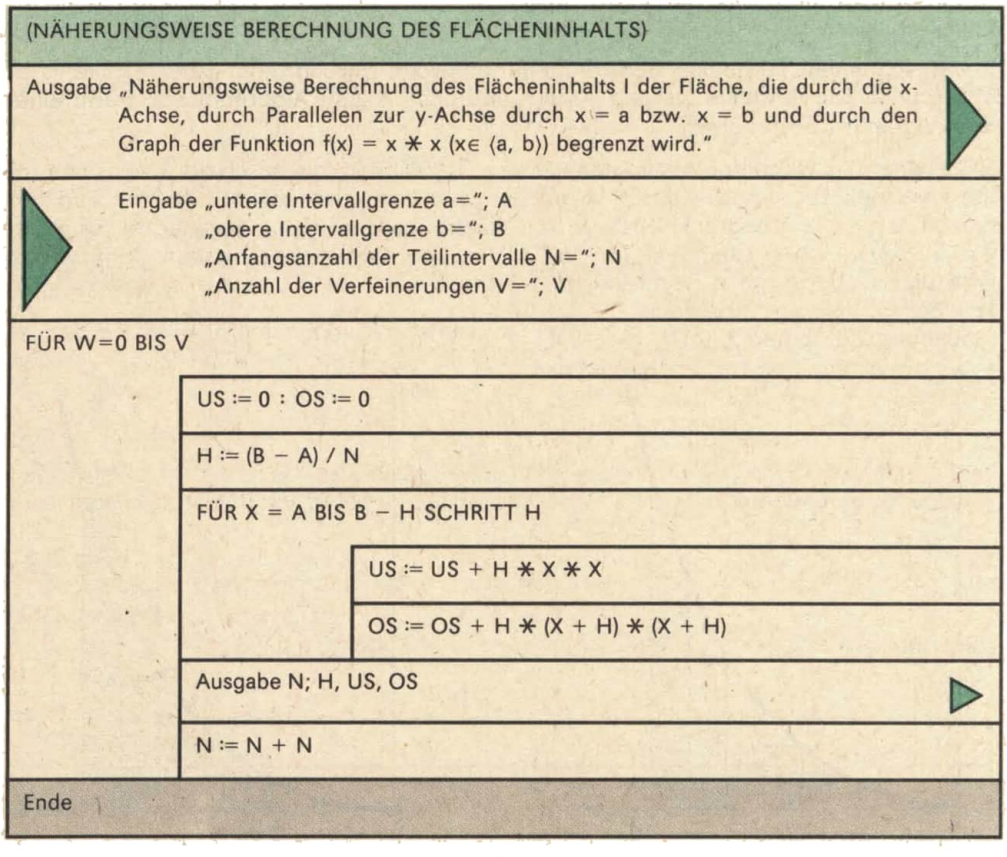


Abb. 2.3/18: Struktogramm „Flächenberechnung"

Der Berechnungsteil des Programms besteht in den Zeilen:

```
330 FOR W = Ø TO V
340     US = Ø : OS = Ø : H = (B-A)/N
350     FOR X = A TO B-H STEP H
360         US = US + H * X * X
370         OS = OS + H * (X+H) * (X+H)
380     NEXT X
390     PRINT N;TAB(5);H;TAB(16);US;TAB(29);OS
400     N = N + N
410 NEXT W
```

Für den Test sei A=2, B=3, N=5 und V=5 gewählt. Bei Abarbeitung dieser Testwerte gibt das Bildschirmprotokoll Rätsel auf, da folgendes erscheint:

N	H	Untersumme	Obersumme	
5	.2	4.272	5.04	*
10	.1	6.085	6.585	
20	.05	6.20875	6.45875	
40	.025	6.04968	6.17095	*
80	.0125	6.19056	6.25212	*
160	6.25E-03	6.31768	6.34893	

Weder nimmt die Untersummenfolge monoton zu, noch die Obersummenfolge monoton ab. Es gibt sogar mehrere Obersummen – mit * gekennzeichnet –, die kleiner als Untersummen sind.

Um den Fehler im Programm herauszufinden, wird anstatt $f(x)=x*x$ die ebenfalls monoton wachsende Funktion $f(x)=x$ genommen. Dazu sind im Programm nur die Zeilen 360 und 370 zu ändern:

```
360         US = US + H * X
370         OS = OS + H * (X+H)
```

Für diese Funktion ist der exakte Flächeninhalt sofort bestimmt. Nimmt man das Intervall <0,1>, so ist I = 0.5 der exakte Flächeninhalt. Wählt man $A=0$, $B=1$, $N=5$ und $V=5$, so wird auf dem Bildschirm protokolliert:

N	H	Untersumme	Obersumme	
5	.2	.4	.6	
10	.1	.36	.45	*
20	.05	.4275	.475	*
40	.025	.4875	.5125	
80	.0125	.49375	.50625	
160	6.25E-03	.490664	.496875	*

Auch bei diesem Test treten vergleichbare Effekte auf, die auf die Fehlerhaftigkeit des Programms hinweisen. Läßt man sich den letzten X-Wert ausgeben mit

```
PRINT X    →    .993752
```

so wird die Ursache für die nicht korrekten Ausgaben erkennbar. Es ist

```
   .993752
+  .00625    (letzter Wert von H)
  1.000002
```

Zwar ergeben die Produkte der Dezimalzahlen von N * H jeweils die obere Intervallgrenze, aber durch die interne Dualarithmetik kommt es zu Rundungseffekten. Diese bewirken, daß der Flächeninhalt der jeweils letzten Rechtecke manchmal nicht mehr zur Untersummen- bzw. Obersummenbildung herangezogen wird, weil der Endparameter B−H überschritten ist. Deshalb wird zum Endparameter der Zählschleife eine halbe Schrittweite als Reserve addiert (B−H+H/2), so daß die Zeile 350 neu die Form

```
350    FOR X = A TO B-H/2 STEP H
```

bekommt. Mit dieser Korrektur fallen dann die Programmtests erwartungsgemäß aus:

Für $f(x) = x$ im Intervall <0,1> und
für $f(x) = x * x$ im Intervall <2,3>
haben die Untersummen- und Obersummenfolgen für N=5 und V=5 folgendes Aussehen:

Untersumme	Obersumme	Untersumme	Obersumme
.4	.6	5.84	6.84
.45	.55	6.085	6.585
.475	.525	6.20875	6.45875
.4875	.5125	6.27095	6.39595
.49375	.50625	6.30212	6.36462
.496875	.503125	6.31768	6.34893

Die Untersummenfolgen wachsen und die Obersummenfolgen fallen mit zunehmender Verfeinerung (fortgesetzte Halbierung der Teilintervalle), und keine Untersumme ist größer als eine Obersumme. Die zuletzt angezeigten Untersummen und Obersummen liegen am dichtesten beim wahren Wert (I = 0.5 bei der linearen Funktion und I = $6.3\overline{3}$ bei der quadratischen Funktion). Der hier zu Schwierigkeiten führende Effekt läßt sich an einem sehr einfachen Beispiel für FOR-NEXT-Schleifen studieren:

```
                                  Für S=.25 bzw. S=.2 wird ausgedruckt:
                                        0          0
10 CLS                                  .25        .2
20 INPUT "Schrittweite S=";S            .5         .4
30 FOR I=0 TO 2 STEP S                  .75        .6
40    PRINT I                           1          .8
50 NEXT I                               1.25       1
60 END                                  1.5        1.2
                                        1.75       1.4
                                        2          1.6
                                                   1.8
```

Bei der zweiten Schrittweite wird der Endwert I=2 nicht ausgedruckt, da die binäre Darstellung von .2 und ihre fortlaufende Addition in der Laufanweisung nach dem letzten gezeigten Durchlauf einen Wert größer dem Endwert und damit den Abbruch der Schleife ergibt.

Zusammenfassung

Die **Zählschleife** ist eine Steuerstruktur. Sie ist anwendbar, wenn bekannt ist, wie oft die Schleife zu durchlaufen ist.

Zählschleifen dürfen ineinander geschachtelt werden. Dabei ist darauf zu achten, daß die zuletzt eröffnete Schleife zuerst geschlossen werden muß. Im Schleifenkörper sollte die Laufvariable nicht verändert werden.

Werden Zählschleifen zur Berechnung einer Summe S bzw. eines Produktes P genutzt, so sind der Variablen S bzw. der Variablen P vor Eintritt in die Schleife Anfangswerte zuzuweisen.

Im BASIC der Kleincomputer ist die Zählschleife so realisiert, daß der Schleifenkörper mindestens einmal durchlaufen wird, auch dann, wenn z. B. der Anfangswert bei positiver Schrittweite größer als der Endwert ist.

Die KC 85/2 und /3 verfügen über 8 Hintergrund- und 16 Vordergrundfarben, wobei die Vordergrundfarbe der Schriftfarbe entspricht. Der Hintergrund läßt sich mit der Anweisung **PAPER hf** einstellen. Der Vordergrund läßt sich mit der Anweisung **INK vf** einstellen.

Mit **COLOR vf,hf** können Vorder- und Hintergrundfarbe mit einer Anweisung festgelegt werden.

Neben der Möglichkeit, Teilketten mit MID$ zu bilden, gibt es die Bildung von Teilketten mit den Anweisungen

RIGHT$(zeichenkette,zeichenanzahl von rechts)

und

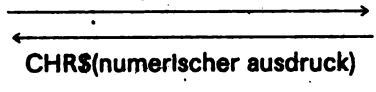
LEFT$(zeichenkette,zeichenanzahl von links).

Mit der Anweisung

INSTR(zeichenkette1,zeichenkette2)

kann ‚zeichenkette1' in ‚zeichenkette2' gesucht werden.

Die BASIC-Standardfunktionen **CHR$** und **ASC** ermöglichen die Wandlung



wobei ‚numerischer ausdruck' >= 0 und <= 255 sein muß!

Neben der externen Dateneingabe mit der Anweisung INPUT gibt es in BASIC die interne Dateneingabe mit der Anweisung

READ variable [,variable]... .

Für die interne Dateneingabe müssen die Daten mit

DATA konstante [,konstante]...

im Programmtext bereitgestellt werden. Bei Abarbeitung von READ-Anweisungen wird ein Zeiger im DATA-Block weitergestellt.

Mit der Anweisung

RESTORE [zeilennummer]

kann der Datenzeiger auf den Anfang des DATA-Blockes bzw. bestimmte Zeilennummern innerhalb dieses Blockes eingestellt werden.

Mit dem Kommando

MODIFY vierstellige hexadezimale adresse

lassen sich im Betriebssystem-Modus der KC 85/2 und /3 Speicherzelleninhalte (Bytes) verändern, um z. B. neue Zeichen zu definieren.

Aufgaben

1. Schreiben Sie ein Programm, das in tabellarischer Form in 3 Spalten die 224 Tripel (die Adresse in der Zeichencodetabelle, das zugehörige Zeichen, den Wert des zugehörigen Zeichens) der druckbaren Zeichen auf dem Bildschirm ausgibt!

2. Was bewirkt das folgende Programm?

```
10 PAPER 6 : CLS
20 FOR I=0 TO 31
30    LINE 10,10+7*I,250-7*I,I
40 NEXT I
50 END
```

3. Was bewirkt das folgende Programm?

```
10 COLOR 0,6 : CLS
20 FOR I=0 TO 15
30    CIRCLE 10+I*10, 10+I*5, I*5, I
40 NEXT I
50 END
```

4. Aus der Molekularbiologie weiß man, daß die Informationsspeicherung in Folgen von Nukleotidbasen (Adenin (A), Thymin (T), Guanin (G) und Zytosin (C)) in einem Eiweißmolekül erfolgt. Interessiert zum Beispiel, wie oft und an welchen Stellen bestimmte Aminosäuren (Kombinationen von jeweils drei Nukleotidbasen) wie Arginin GCA, Serin AGC oder Alanin CGG in einer Nukleotidbasenfolge wie

   AATCACGATCCTTCTAGGAGG...
   oder
   TCGCGTAAGCTGGCTTAGCCG...

vorkommen, so ist eine Aufgabe der Textanalyse zu lösen. Schreiben Sie ein Programm, mit dessen Hilfe ein Computer die Information über die Vorkommensstellen und über die Vorkommenshäufigkeiten eingegebener Buchstabenfolgen in einem Text ermittelt und ausgibt!

5. Die Erbinformationsübertragung erfolgt durch die Boten-RNS (Ribonukleinsäure) und zwar so, daß sich

   - an ein Zytosinnukleotid der DNS ein Guaninnukleotid der RNS,
   - an ein Guaninnukleotid der DNS ein Zytosinnukleotid der RNS,
   - an ein Thyminnukleotid der DNS ein Adeninnukleotid der RNS,
   - an ein Adeninnukleotid der DNS ein Urazilnukleotid der RNS

   anlagert. Um die Nukleotidenfolge der Boten-RNS zu bestimmen, ist ein Buchstabenaustausch

   $C \rightarrow G$, $G \rightarrow C$, $T \rightarrow A$, $A \rightarrow U$

   notwendig. Schreiben Sie ein Programm, mit dessen Hilfe der Computer einen frei wählbaren Buchstaben durch einen anderen im Text ersetzt!

6. So manchem Spiel liegen Textanalyse und Textmanipulation zugrunde. Bekannt ist das Lied

   Drei Chinesen mit dem Kontrabaß,
   gingen auf der Straße und erzählten sich 'was,
   kam ein Polizist und sprach: Was ist denn das?
   Drei Chinesen mit dem Kontrabaß.

   Die nächsten fünf Strophen entsprechen dieser ersten in allen Konsonanten. Sämtliche Vokale werden jedoch in der

   - 2. Strophe durch ‚a',
   - 3. Strophe durch ‚e',
   - 4. Strophe durch ‚i',
   - 5. Strophe durch ‚o' und
   - 6. Strophe durch ‚u'

   ersetzt. Es ist ein Programm zu schreiben, das alle sechs Strophen nacheinander auf dem Bildschirm anzeigt!

7. Ergebnisse von Manipulationen an Texten findet man häufig auch aus gestalterischen Gründen in Büchern, Programmheften, auf Plakaten oder auf Schallplattenhüllen. In Christian Morgensterns „Galgenliedern" ist „DAS AESTHETISCHE WIESEL" zentriert abgedruckt:

   Ein Wiesel
   saß auf einem Kiesel
   inmitten Bachgeriesel.
   Wißt ihr,
   weshalb?
   Das Mondkalb
   verriet es mir
   im Stillen:
   Das raffinierte Tier
   tat's um des Reimes willen.

Schreiben Sie ein Programm, mit dessen Hilfe der Computer ein eingegebenes Gedicht, das nicht mehr als 20 Zeilen umfaßt, bei der Bildschirmausgabe zentriert anzeigt!

8. Auf der Märchenplattenhülle „Die Schneekönigin" beginnt die Liste der Sprecher in einer äußerlich sehr ansprechenden Form:

Der Erzähler	Klaus Piontek
Gerda	Juliane Koren
Kay	Walter Plathe
die Großmutter	Ilse Voigt
die Schneekönigin	Jutta Hoffmann

Schreiben Sie ein Programm, das 10 Rollenbezeichnungen rechtsbündig bis zur Mitte ausgibt und die dazugehörigen Sprechernamen linksbündig nach 4 Leerzeichen anschließt!

9. Schreiben Sie ein Programm, das jedes von 10 eingegebenen Wörtern an der Bildschirmmitte spiegelt! Dabei kann man Wörter finden, die, von links nach rechts gelesen, auch ein sinnvolles Wort ergeben, wie z. B.

REGAL LAGER.

Ein Wort, das von links nach rechts gelesen gleich dem Wort ist, das von rechts nach links gelesen wurde, heißt Palindrom. Aus der Vielzahl von Beispielen seien hier RELIEFPFEILER und ANNASUSANNA genannt. Das Programm für die Wortspiegelung ist um einen Palindromtest zu erweitern. Wenn ein Palindrom vorliegt, dann soll dies durch die drei Zeichen ,*P*' gekennzeichnet werden.

10. In einem eingegebenen fortlaufenden Text soll ein Wort besonders hervorgehoben werden, indem es g e s p e r r t ausgegeben wird. Schreiben Sie ein Programm zum Sperren von markierten Wörtern!

11. Schreiben Sie ein Programm, das auf eine positive Zahl X n-mal die Quadratwurzelfunktion anwendet und dann anschließend n-mal die Quadratfunktion! Das heißt, es soll der Wert W wie folgt berechnet werden:

$$W = (\ldots\underbrace{((SQR(\ldots SQR(SQR(X))\ldots))}_{n\text{-mal}}\,\hat{}\,2)\,\hat{}\,2\ldots)\,\hat{}\,2$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{n\text{-mal}}$$

Testen Sie, ob – wie in der Mathematik – W = X auch für die Computerrechnung zutrifft! Die Werte für n sollten in DATA-Zeilen bereitgestellt werden und mindestens 5, 10, 15, 20, 25, 28, 30 umfassen. Für X sind Werte größer 1 und Werte kleiner gleich 1 auszutesten. Klären Sie beim Test, ob die Reihenfolge, in der Funktionen auf Werte angewendet werden, für die Genauigkeit von Bedeutung ist!

12. Es gilt für endliche, positive ganze Zahlen A, E, wobei E >= A ist, daß

$$\sum_{I=A}^{E}(I + 1/I) = \sum_{I=A}^{E} I + \sum_{I=A}^{E} 1/I = \sum_{I=A}^{E} 1/I + \sum_{I=A}^{E} I$$

Schreiben Sie ein Programm, das die Summen auf diesen drei Wegen (unterschiedli-

che Summationsreihenfolge) berechnet! Testen Sie Ihr Programm auch mit A=1000 und E=2000.

13. Schreiben Sie ein Programm zur Berechnung von

$$\sum_{i=1}^{100} i^2 .$$

14. Schreiben Sie ein Programm zur Berechnung von

$$\prod_{K=1}^{N} \frac{(2K)^2}{(2K-1)^2} * \frac{2}{2N+1}$$

Testen Sie das Programm mit N = 100!

15. Man kann Folgen explizit oder rekursiv definieren. Mittels vollständiger Induktion läßt sich nachweisen, daß

die explizit definierte Folge	gleich der rekursiv definierten ist
a) $y_n = (-1)^n 2^{n-6} - 1$	$\begin{cases} y_0 = -63/64 \\ y_n = -2y_{n-1} - 3 \end{cases}$
b) $y_n = (-1)^n 2^{n-8} - 1$	$\begin{cases} y_0 = -255/256 \\ y_n = -2y_{n-1} - 3 \end{cases}$

Schreiben Sie für a) und b) Programme und berechnen Sie damit die jeweils ersten 30 Glieder der Folge! Werten Sie die Genauigkeit!

16. Schreiben Sie ein Programm, mit dem Sie mit der rekursiv definierten Folge

$$\begin{cases} a_0 \\ a_{n+1} = z^2 + z - a_n \qquad z \in \mathbb{R}^+, \text{ konstant} \end{cases}$$

experimentieren können! Es ist zu untersuchen, wie sich bei gewähltem z und a_0 die Folge verhält.

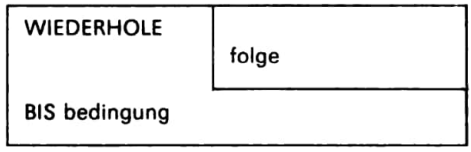
17. Diophantische Gleichungen können mit Computern mit geringem Programmieraufwand gelöst werden. Es werden alle möglichen Fälle systematisch durchprobiert. Als Beispiel ist folgende Aufgabe zu lösen:
Dieter hat die Aufgabe, 100 Stück Pralinenkästen und Bonbontüten zu kaufen. Er bekommt genau 100,- M und hat 3 Sorten zur Auswahl – eine zu 10,- M, eine zu 3,- M und eine zu 0,50 M. Wie viele Pralinenkästen und Bonbontüten holt er von jeder Sorte, wenn sein Geld für die 100 Stück genau aufgehen soll und er von jeder Sorte mindestens ein Stück mitzubringen hat?

Wiederholschleife und Solangeschleife

Soll eine Anweisungsfolge wiederholt werden und die Anzahl der Wiederholungen ist nicht von vornherein bekannt, so sind die Wiederholschleife und die Solangeschleife zu verwenden.

Falls eine Anweisungsfolge zu wiederholen ist, bis eine Bedingung (Abbruchbedingung, Endeabfrage) erfüllt wird, wählt man die WIEDERHOLSCHLEIFE.

WIEDERHOLSCHLEIFE

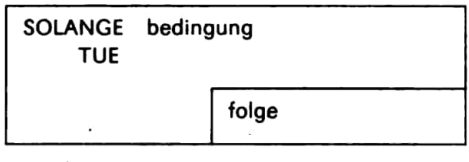
verbal formalisierte Notation	**Struktogramm**
WIEDERHOLE folge BIS bedingung	

Umsetzung in BASIC

```
ak    REM anfang der wiederholschleife
al        folge
an    IF NOT(bedingung) THEN ak
am    REM ende der wiederholschleife
```

Die Wiederholschleife wird mindestens einmal abgearbeitet, da die Abbruchbedingung erst nach der Anweisungsfolge geprüft wird. Deshalb wird diese Form in der Literatur auch häufig als ‚fußgesteuerte' oder ‚nichtabweisende' Schleife bezeichnet.

Falls eine Anweisungsfolge ausgeführt und wiederholt werden soll, solange eine Bedingung (Eingangsbedingung, Anfangsabfrage) erfüllt ist, wählt man die SOLANGE-SCHLEIFE.

verbal formalisierte Notation	**Struktogramm**
SOLANGE bedingung TUE folge ENDE	

Umsetzung in BASIC

```
ak    REM anfang der solangeschleife
al    IF  NOT(bedingung) THEN ao
am        folge
an    GOTO al
ao    REM ende der solangeschleife
```

Aus der Darstellung ist ersichtlich, daß bei der Solangeschleife die Bedingung vor der Abarbeitung der Folge geprüft wird.

Ist die Bedingung nicht erfüllt, so wird die Schleife gar nicht abgearbeitet. In der Literatur spricht man deshalb von ‚kopfgesteuerten' oder ‚abweisenden' Schleifen.

Anwendungsbeispiele zur Wiederholschleife

Einige der bisher dargestellten Algorithmen sind nicht gegen unzulässige Eingaben geschützt. So ist es z. B. beim Algorithmus „Umfang und Flächeninhalt eines Dreiecks„ (↗ 2.3.1) möglich, Werte einzugeben, die zum Fehler beim Funktionsaufruf führen. Die Wiederholschleife schafft eine brauchbare Lösung in folgender Weise:

```
WIEDERHOLE
    ▷ Eingabe a, b, c
BIS a > 0 und b > 0 und c > 0 und a + b >c
und a + c > b und b + c >a
```

Viele der bisher beschriebenen Beispiele werden erst durch das Strukturelement Wiederholschleife effektiv gelöst. So kann man mathematisch erst dann sinnvoll experimentieren (↗ 2.3.1. Aufgabe 4), wenn immer wieder neue Werte in ein Programm eingegeben werden können. Der automatische Verkauf von S-Bahn-Fahrkarten (↗ 2.3.2 Beispiel 4) wird durch ein Programm erst dann der Realität entsprechend simuliert, wenn das Programm nach dem Verkauf eines Fahrausweises in die Preisstufeneingabemöglichkeit zurückgeht.

Anwendungsbeispiel zur Solangeschleife

9 Es ist ein Algorithmus für die Berechnung von

$s_n = 1 + 1/2 + 1/3 + 1/4 + \ldots + 1/n \qquad (n >= 1)$

zu entwickeln.
Die Summation der Reziproken der natürlichen Zahlen soll erst dann abgebrochen werden, wenn die Computer-Addition weiterer Summanden zu keiner Änderung der Summe mehr führt.

```
(HARMONISCHE REIHE – 1. Version: SOLANGESCHLEIFE)
ALTE SUMME := 0 : NEUE SUMME := 1
INDEX := 2
SOLANGE ALTE SUMME <> NEUE SUMME
TUE
    ALTE SUMME := NEUE SUMME
    NEUE SUMME := NEUE SUMME + 1/INDEX
    INDEX := INDEX + 1
Ausgabe
    „Ab Index ="; INDEX; „verändert sich die Summe";
    NEUE SUMME; „nicht mehr."
Ende
```

Abb. 2.3/19: Struktogramm „Harmonische Reihe" 1. Version

Man kann diese Berechnung auch in einer Wiederholschleife durchführen:

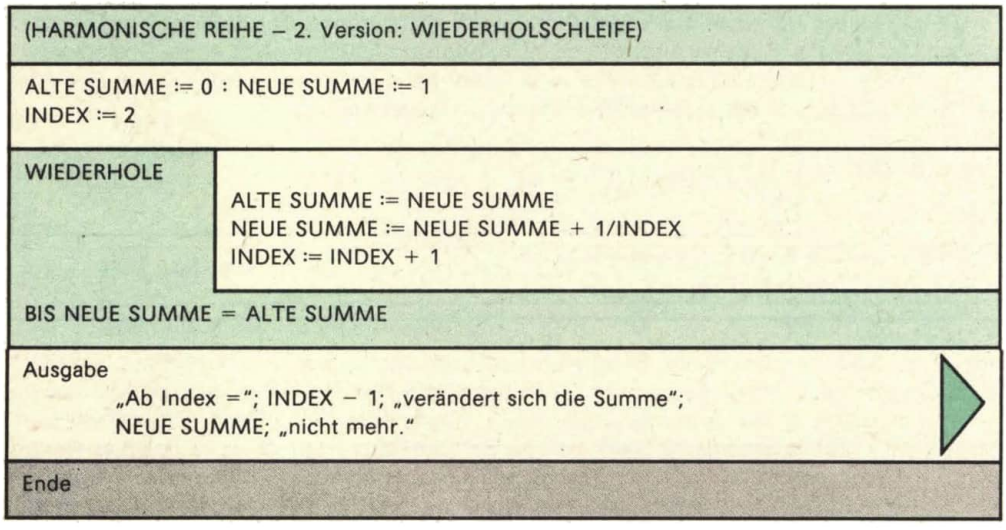


Abb. 2.3/20: Struktogramm „Harmonische Reihe“ 2. Version

Es ist zu erkennen, daß für das gleiche Problem die Solangeschleife und die Wiederholschleife verwendet werden können.

In beiden Fällen hängt die Wiederholung von einer Bedingung ab. Bei der Solangeschleife muß die Eingangsbedingung erfüllt sein, damit die Anweisungsfolge durchlaufen wird. Bei der Wiederholschleife wird erst nach Bearbeitung der Anweisungsfolge abgefragt, ob die Bedingung erfüllt ist. Und nur dann, wenn diese Bedingung nicht erfüllt ist, wird die Bearbeitung wiederholt. Das bedeutet, daß die Abbruchbedingung der Wiederholschleife die logische Negation der Eingangsbedingung der Solangeschleife sein muß, um dem gleichen Sachverhalt zu entsprechen.

Für den Programmlauf der beiden zu den Algorithmen gehörenden BASIC-Programme müßte man sich Zeit nehmen, denn erst nach etwa 15 Stunden würde der Computer ausgeben:

„Ab Index = 2.09715E+06 veraendert sich die Summe 15.4037 nicht mehr.“

Was bedeutet dieses Ergebnis? Kann man in Anlehnung an

$1 + 1/2 + 1/4 + 1/8 + \ldots 1/2^n < 2$

vermuten, daß

$1 + 1/2 + 1/3 + 1/4 + \ldots 1/n < 15.4037$

ist? Nein, diese Vermutung ist falsch!

Es werden die ersten $n = 2^k$ Summanden

$$s_2{}^k = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \ldots$$

$$\ldots + \frac{1}{2^{k-1}+1} + \frac{1}{2^{k-1}+2} + \ldots + \frac{1}{2^{k-1}+2^{k-1}-1} + \frac{1}{2 * 2^{k-1}}$$

betrachtet.

Daraus bildet man geeignete Gruppen von Summanden

$$s_2^k = 1 + \left(\frac{1}{2}\right) + \left(\frac{1}{3} + \frac{1}{4}\right) + \left(\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}\right) + \ldots$$

$$\ldots + \left(\frac{1}{2^{k-1}+1} + \frac{1}{2^{k-1}+2} + \ldots + \frac{1}{2^k}\right).$$

Der letzte Summand $1/2^i$ in der i. Klammer ($1 <= i <= k$) ist der kleinste Summand in der i. Klammer. Alle 2^{i-1} in der i. Klammer stehenden Summanden sind größer oder gleich $1/2^i$, daher läßt sich der Wert des Terms in jeder Klammer durch

$2^{i-1} * 1/2^i = 1/2$ nach unten abschätzen, so daß

$s_2^k >= 1 + k * \frac{1}{2}$ gilt.

Daraus folgt aber, daß diese Summation über jede angebbare endliche Zahl wächst, falls nur k hinreichend groß gewählt wird. Das Computerresultat, das aufgrund der Computerzahlenmenge und der Rundungseffekte entstand, spiegelt nicht nur ein quantitativ, sondern sogar ein qualitativ falsches Resultat wider.

Der Computer ersetzt nicht mathematisches Denken, aber er kann es unterstützen. Computerberechnungen sollten nur dann vorgenommen werden, wenn sie Sachverhalte qualitativ richtig widerspiegeln und die quantitativen Fehler abschätzbar bzw. beurteilbar sind.

- Übersetzen Sie die beiden Algorithmen zum Beispiel 9 in BASIC-Programme und überprüfen Sie die dargelegten Aussagen!

Zusammenfassung

Sind Anweisungen mit einer vorab nicht bekannten Anzahl zu wiederholen, so kann das mit den zwei Schleifentypen

– **Solangeschleife**

und

– **Wiederholschleife**

realisiert werden.

Bei der **Solangeschleife** muß die Eingangsbedingung erfüllt sein, damit die entsprechende Anweisungsfolge durchlaufen bzw. wiederholt wird.

Bei der **Wiederholschleife** wird erst nach Bearbeitung der Anweisungsfolge die Abbruchbedingung getestet. Nur dann, wenn diese Bedingung nicht erfüllt ist, wird die Bearbeitung wiederholt.

Wiederholschleifen eignen sich zum Beispiel für Eingaben mit Zulässigkeitstests, für die Programmsteuerung mit Auswahlmenüs, für die fortlaufende Dateneingabe bis zu einem vereinbarten Schlußzeichen und zur Wiederholung von Berechnungen, bis eine Genauigkeitsbedingung erfüllt wird oder bestimmte obere bzw. untere Schranken über- bzw. unterschritten werden.

● Aufgaben

1. Schreiben Sie das Programm ‚Näherungsweise Berechnung eines Flächeninhalts' (↗ 2.3.3) unter Verwendung der Wiederholschleife um! Die Anzahl der Teilintervalle ist so oft zu verdoppeln, bis die dazugehörige Berechnung der Untersumme *US* und der Obersumme *OS* Werte liefert, die der Abbruchbedingung $OS-US<EPS$ genügen. Der Wert für *EPS* soll vom Nutzer vor der Berechnung eingebbar sein. Auszugeben sind $(OS+US)/2$ als Näherungswert für den Flächeninhalt und die Anzahl der Teilintervalle.

2. Bei einer Population von Drosophila-Fliegen, die in einem Glasbehälter untergebracht und mit hinreichend Nahrung versorgt wurden, stellte man eine Entwicklung der Individuenanzahl fest, die sich nach der Rekursionsformel

$$\begin{cases} x_0 = 1 \\ x_{n+1} = (1+r) * x_n - s * x_n^2 \end{cases}$$

mit $n = 0, 1, 2, \ldots$, $r = 0.8$ und $s = 0.008$

näherungsweise berechnen läßt.
Entwickeln Sie ein Simulationsprogramm, mit dem die Veränderungen der Individuenanzahl in Abhängigkeit von den Parametern *r* und *s* verfolgt werden können.

2.3.4. Anwendungen der Grundstrukturen – Menütechnik

In größeren Programmen kann ein Programmnutzer oft aus mehreren Aktionsmöglichkeiten eine spezielle heraussuchen. Eine solche Angebotspalette von Programmfortsetzungsmöglichkeiten heißt *Menü*.

Erstes Anwendungsbeispiel zur Menütechnik

1 Es ist ein Algorithmus zu entwickeln, mit dem man Kurvenverläufe im Zeichen-Raster demonstrieren kann. Dabei sollen eine Parabel und eine sinusförmige Kurve mit vertikaler x-Achse und eine Sinusfunktion und der Funktionsverlauf beim horizontalen Wurf mit horizontaler x-Achse auf dem Bildschirm dargestellt werden.

Eingabe: Es ist eine Kennzahl einzugeben, die einem Menü zu entnehmen ist.

Verarbeitung: Die Kennzahl 0 soll zum Programmende führen. Die Kennzahl 1 soll die Ausgabe einer Parabel, die der Gleichung $y(x)=x^2$ genügt, veranlassen und zwar so, daß die Argumente nach unten zunehmen.
Die Kennzahl 2 soll die Ausgabe einer sinusförmigen Kurve veranlassen. Die Kennzahl 3 soll die Sinusfunktion mit nach rechts wachsendem Argument erzeugen, und die Kennzahl 4 soll die Ausgabe der Funktion, die dem grafischen Bild des horizontalen Wurfes entspricht, veranlassen.

Ausgabe: Es sind das Menü, die Eingabeaufforderung, ggf. ein Hinweis auf Eingabefehler und die Graphen im Zeichen-Raster mit dem Zeichen „*" auszugeben.

---

❶ `Y := X * X : PRINT TAB(Y); „*"`

❷ `Y := SIN(X) : YB := INT(10 * Y + 20 + .5) : PRINT TAB(YB); „*"`

❸ `Y := SIN(X) : XB := INT(6 * X + .5) : YB := INT(−6 * Y + 15 + .5) : PRINT AT(YB, XB); „*"`

❹ `Y := SQR (2 * X/9.81) : YB := INT (Y * 10 + .5) : PRINT TAB(YB); „*"`

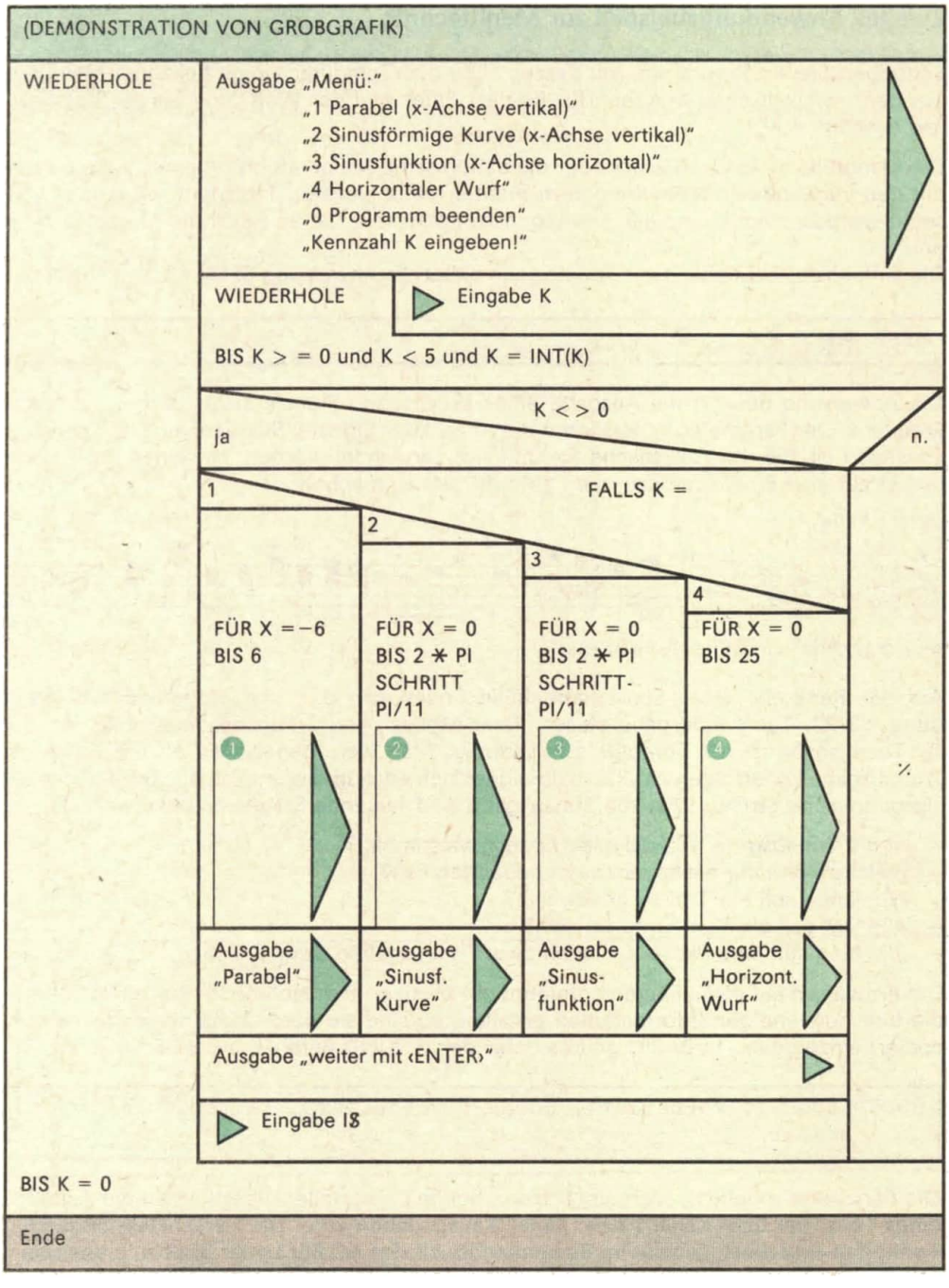


Abb. 2.3/21: Struktogramm „Grafikdemonstration"

Die in die Laufanweisungskörper (gekennzeichnet durch 1 bis 4) gehörenden Anweisungen befinden sich auf der Seite 106 unten.

Zweites Anwendungsbeispiel zur Menütechnik

2 Schreiben Sie ein Programm, mit dessen Hilfe der Computer einige bekannte Melodien aus dem musikalischen Märchen für Kinder „Peter und der Wolf" von Sergej S. Prokofjew ausgibt!

Die Klangfülle eines Orchesters oder die einfühlsame Melodieführung einer Violine kann mit den verwendeten Kleincomputern nicht erreicht werden. Trotzdem soll einmal versucht werden, dem Computer Tonfolgen zu entlocken, die an bekannte Melodien erinnern.
Die einfachste Möglichkeit zur Tonausgabe bietet die Anweisung BEEP mit dem Format

```
BEEP (n)
```

Die Anweisung bewirkt die Ausgabe eines akustischen Signals fester Länge und fester Frequenz. Der Parameter ‚n' bestimmt die Anzahl der Signale. Standardwert ist 1 und Maximalwert ist 255. Das akustische Signal kann verwendet werden, um einen Programmnutzer auf eine Eingabeanforderung aufmerksam zu machen.
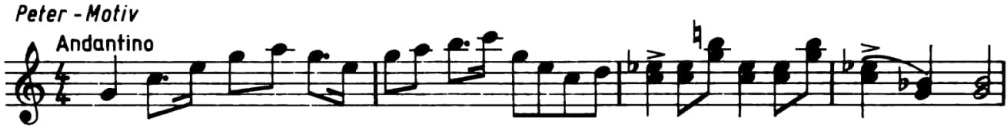


Abb. 2.3/22: Notenbild des Peter-Motivs

Auf der Rückseite einer Schallplattenhülle findet man u. a. das Notenbild aus Abbildung 2.3/22. Zur Wiedergabe dieses ‚Peter-Motivs' sind 2 Tonausgaben nötig. Ab der 16. Tonausgabe ist die Tonfolge zweistimmig. Man weiß genau, wie oft die Teilaktion ‚Tonausgabe' zu erfolgen hat. Deshalb eignet sich auch in diesem Fall die Zählschleife als algorithmische Struktur. Für die Tonausgabe sind folgende Fragen zu beantworten:

- Ist die Tonausgabe entweder ein- oder zweistimmig?
- Welche Tonhöhe besitzt ein auszugebender Ton?
- Wie lange soll ein Ton zu hören sein?
- Wie laut soll ein Ton zu hören sein?
- Wie lang soll eine Pause zwischen zwei Tonausgaben sein?

Die Antworten auf diese Fragen entnimmt ein Musiker dem Notenbild. Soll ein Computer die fünf notwendigen Informationen erhalten, so sind sie dem Computer entsprechend codiert einzugeben. In BASIC gibt es dafür den SOUND-Befehl mit dem Format

```
SOUND hoehe1, vorteiler1, hoehe2, vorteiler2, lautstaerke, ton-
      dauer
```

Die Parameter ‚hoehe1', ‚vorteiler1' bzw. ‚hoehe2', ‚vorteiler2' bestimmen die Tonhöhe eines Tons, der über Kanal 1 bzw. Kanal 2 ausgegeben wird. Die Verschlüsselungen der Tonhöhen sind einer Tabelle im Bedienhandbuch des KC 85/2 oder 85/3 zu entnehmen. Mit den Parametern ‚lautstaerke' und ‚tondauer' werden die Lautstärke (0 <= lautstaerke <= 31) und die Tondauer (0 <= tondauer <= 255) festgelegt. Die Tonwiedergabe wird z. B. über ein Kassettentonbandgerät realisiert, das mit dem Computer über ein fünfpoliges Überspielkabel verbunden ist. Das Kassettentonbandgerät ist auf Aufnahme zu stel-

len, und die Lautstärke ist nach Wunsch zu regeln. Der Lautstärkeparameter ‚lautstaerke' hat auf die Lautstärkeregelung am Kassettentonbandgerät keinen Einfluß. Es wird lautstaerke = 31 gewählt. Wie auch ein Musiker nach Empfinden das angemessene Tempo sucht, soll hier mit Tonlängen experimentiert werden. Es sei tondauer = D*F, wobei D die Tonlängenrelation (1/16-, 1/8-Note usw.) erfaßt und F ein Tonlängenfaktor ist, der die Tempoangabe des Komponisten berücksichtigen soll.

Die Werte für D seien wie folgt festgelegt:
– für eine 1/16-Note sei D = 1,
– für eine 1/8 -Note sei D = 2,
– für eine 1/4- Note sei D = 4 usw.
Eine Analyse des Peter-Motivs zeigt Abbildung 2.3/23.

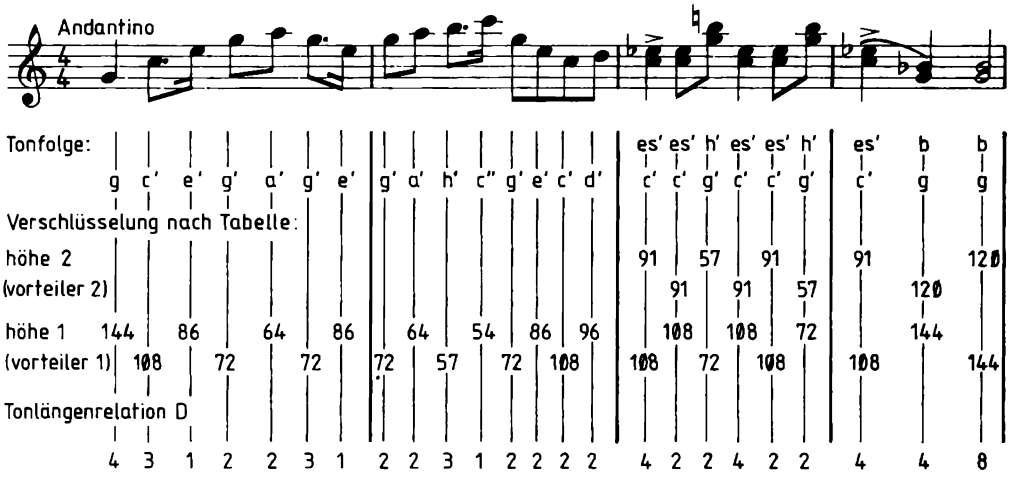


Abb. 2.3/23: Analyse des „Peter-Motivs"

Diese Daten werden in DATA-Zeilen abgelegt und können mit READ gelesen werden. Das Kernstück des Programms für die ersten zwei einstimmigen Takte besteht in den Programmzeilen:

```
400 REM TONWIEDERGABE
410 FOR T = 1 TO 15
420     READ H1, D
430     SOUND H1,0,0,0,31,D*F
440     SOUND 0,0,0,0,0,1  : REM veranlaßt kurze Unterbrechung
450 NEXT T
480 END
490 REM ------------------------------------------------
```

Dazu gehören dann die DATA-Zeilen:

```
500 REM Daten fuer das PETER-Motiv (Takte 1 und 2)
510 DATA 144,4,108,3,86,1,72,2,64,2,72,3,86,1
520 DATA 72,2,64,2,57,3,54,1,72,2,86,2,108,2,96,2
```

Vor der Wiedergabe dieses ersten Teils des Peter-Motivs ist F mit einem Wert zu belegen. Andernfalls könnte kein Ton gehört werden, da F standardmäßig mit 0 belegt ist.

Sollen auch noch die zwei zweistimmigen Takte 3 und 4 programmiert werden, so ist im Programm eine Entscheidung einzubauen, aufgrund der ab der 16. Tonwiedergabe Zweistimmigkeit vorliegt. Bisher wurden Paare (H1,D) von Daten für einen Ton in DATA-Zeilen erfaßt, nun müssen es Tripel (H1,H2,D) sein. Somit sind die Programmzeilen 410 bis 430 zu verändern, und es sind die DATA-Zeilen für die Takte 3 und 4 anzuhängen:

```
410 FOR T = 1 TO 24
420     IF T<16 THEN READ H1,D : H2=0 : ELSE READ H1, H2, D
430     SOUND H1,0,H2,0,31,D*F
...
530 REM Daten fuer das PETER-Motiv (Takt 3 u. 4)
540 DATA 108,91,4,108,91,2,72,57,2,108,91,4,108,91,2,72,57,2
550 DATA 108,91,4,144,120,4,144,120,8
```

Um auch noch die Motive der Katze und des Wolfes zu realisieren, ist das Kernprogramm zur Tonwiedergabe weiter zu verallgemeinern.
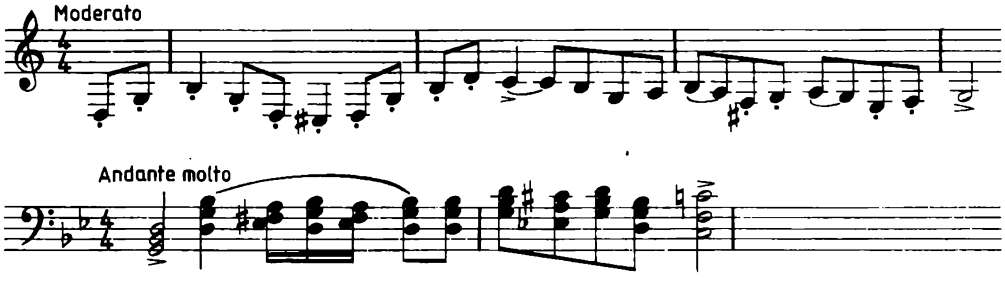


Abb. 2.3/24: Motive von Katze und Wolf

Wie man erkennt, bestehen die genannten Motive aus einer unterschiedlichen Anzahl von Tönen. Mehrstimmigkeit setzt entweder gar nicht oder an einer anderen Stelle ein. An den Tempobezeichnungen erkennt man, daß das eine Motiv ‚mäßig schnell' (Moderato) und ein anderes ‚etwas schneller gehend' (Andantino) gespielt werden soll. Diese Informationen können auch verschlüsselt in DATA-Zeilen erfaßt und mit READ bei der Abarbeitung gelesen werden. Das Wolf-Motiv ist eigentlich dreistimmig. Auch hier sollte im Experiment entschieden werden, welche beiden Stimmen zu nehmen sind, um das Charakteristikum des Motivs zu treffen. In jedem Fall kommen beim Wolf-Motiv Töne vor, die laut Tonhöhentabelle im Parameter ‚vorteiler1' bzw. ‚vorteiler2' nicht mehr den Wert 0, sondern den Wert 1 haben. Bei einem Vergleich der Verschlüsselungen aller verwendeten Töne erkennt man, daß nur Tonhöhen mit dem Parameter vorteiler1=0 bzw. vorteiler2=0 erfaßt sind, die für H1 bzw. H2 einen größeren Wert als 27 haben. Die Tonhöhen mit Vorteilerwert=1 besitzen für H1 bzw. H2 Werte, die kleiner als 28 sind, so daß im hier vorliegenden Fall aufgrund der Werte von H1 bzw. H2 auch entschieden werden kann, ob V1 bzw. V2 entweder 0 oder 1 ist.

Die Tonfolgen im Peter- und im Wolf-Motiv sind gebunden, während die Töne beim Motiv der Katze ‚abgestoßen' (staccato) zu spielen sind. Das erreicht man beim Computer durch eine etwas längere Unterbrechungsdauer zwischen zwei aufeinanderfolgenden Tönen.

Damit ergibt sich für das Kernstück des Programms in recht allgemeiner Form:

```
390 READ N, Z, F : REM Tonanzahl, Zweistimmigkeitsbeginn,
    Tonlaengenfaktor
400 REM TONWIEDERGABE
410 FOR T=1 TO N
420    IF T<Z THEN READ H1,D : H2=0 : ELSE READ H1,H2,D
424    IF H1<28 THEN V1=1 : ELSE V1=0
428    IF H2<28 THEN V2=1 : ELSE V2=0
430    SOUND H1, V1, H2, V2, 31, D*F
435    IF K=2 THEN UD=5 : ELSE UD=1
440    SOUND 0, 0, 0, 0, 0, UD
450 NEXT T
460 IF NOT(K=0) THEN 200
470 REM ende der wiederholschleife
```

Insgesamt ergibt sich somit folgender Algorithmus.

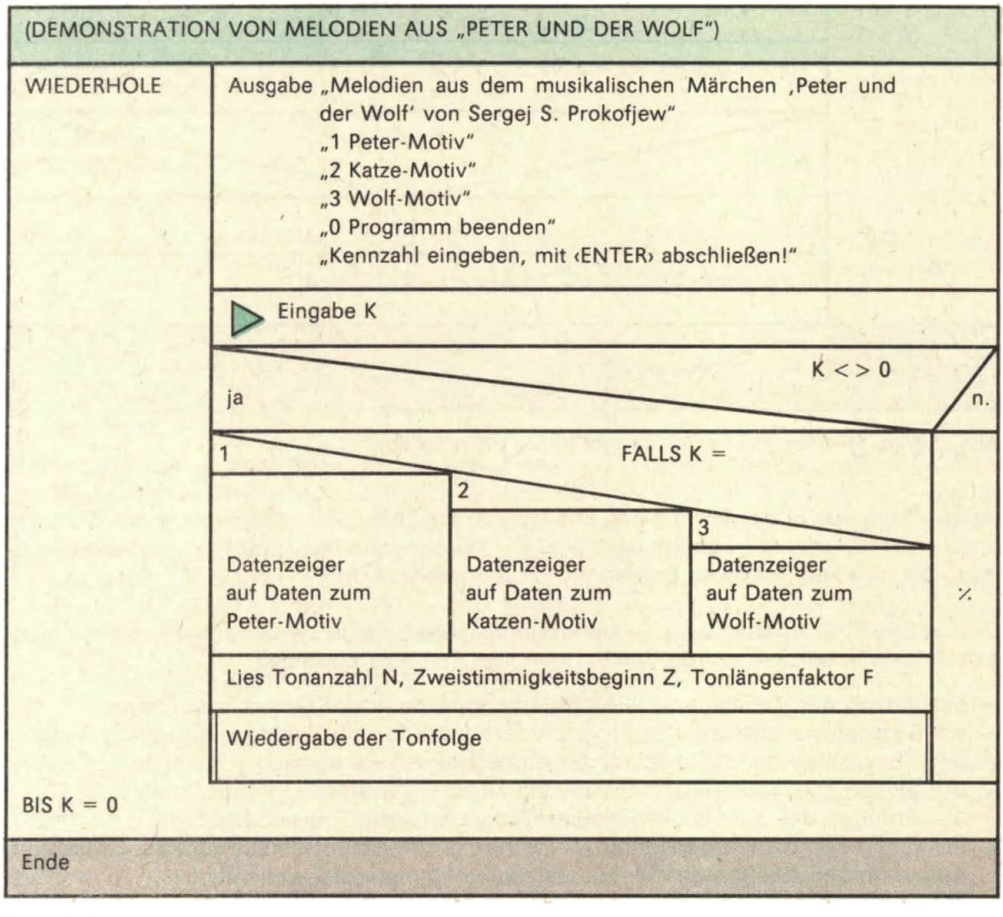


Abb. 2.3/25: Struktogramm „Demonstration von Melodien"

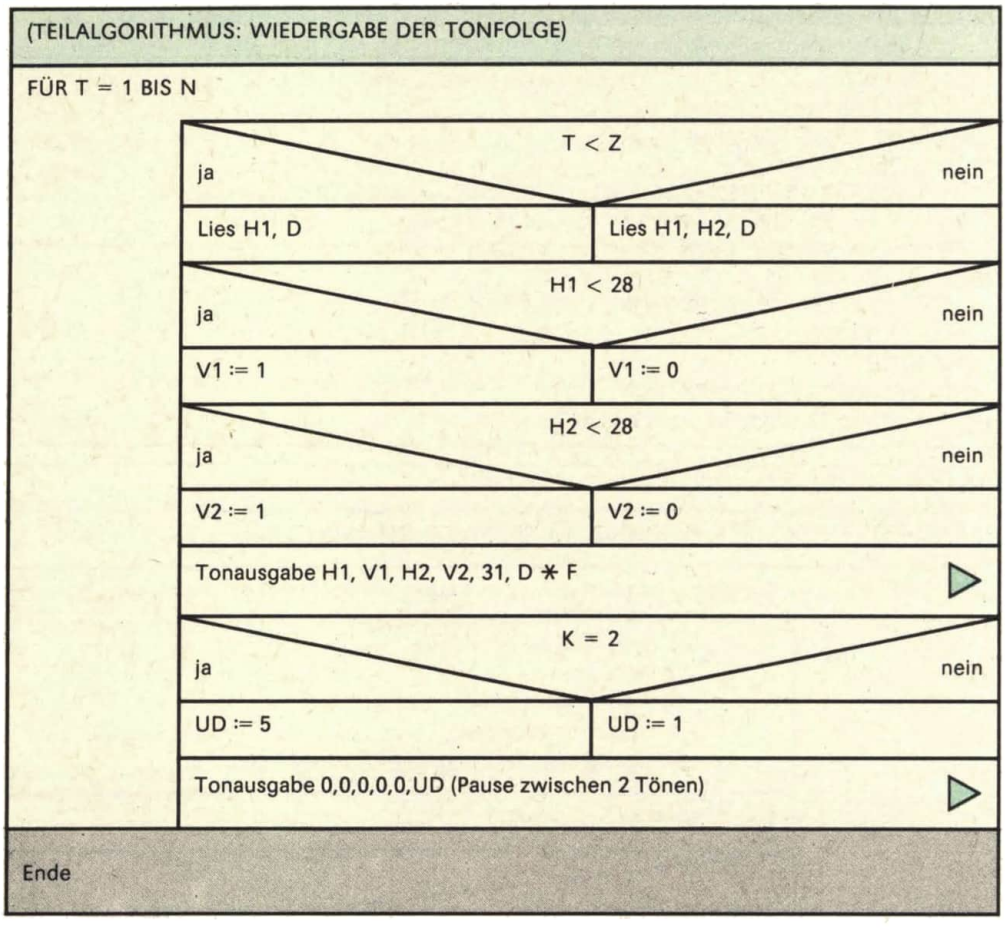


Abb. 2.3/26: Struktogramm zum Teilalgorithmus „Wiedergabe"

Der Algorithmus in der Form eines Struktogramms zeigt nochmals auf, wie das Gesamtproblem strukturell aufzubauen ist. Dabei ist „Wiedergabe der Tonfolge" ein Teilalgorithmus, der in einem weiteren Struktogramm dargestellt wird.

Da das BASIC-Programm zum Teilalgorithmus „Wiedergabe der Tonfolge" bereits in Einzelschritten entwickelt wurde, bleibt noch von Ihnen zu erledigen:

- das Fassen der Variablen in REM-Zeilen zum Anfang des Gesamtprogramms,
- die Bildschirmgestaltung des Programmtitels mit dem Auswahlmenü,
- das Überprüfen der Zulässigkeit der eingegebenen Kennzahl,
- das Stellen des Datenzeigers auf die Daten zum gewählten Motiv,
- das Anfügen des bereits entwickelten Programms zur Tonwiedergabe,
- das Einbinden der vorgenannten Teile in eine Wiederholschleife, um wiederholt die Auswahlmöglichkeiten des Menüs nutzen zu können und schließlich
- das Ergänzen der Daten zum Peter-Motiv durch die Angabe N, Z, F und
- die Angabe aller notwendigen Daten für das Katze- und das Wolf-Motiv.

Drittes Anwendungsbeispiel zur Menütechnik

Mit der Funktion RND(x) liefert der Rechner Zufallszahlen im Intervall <0,1). Die von RND (random – ‚regellos') erzeugten Zahlen folgen allerdings nicht völlig regellos aufeinander. Wie alles im Computer, werden auch sie nach einem Algorithmus, ausgehend von einem Anfangswert, eindeutig erzeugt. Weil dem Benutzer im allgemeinen der Anfangswert und die Bildungsvorschrift der Zahlenfolge unbekannt sind, erscheinen ihm die so erzeugten Zahlen zufällig verteilt. Nach einer bestimmten Anzahl von Gliedern wiederholt sich aber eine Teilfolge. Wenn die Anzahl der Zahlen groß ist, wenn die erzeugten Zahlen etwa gleichmäßig auf dem Intervall <0, 1) verteilt sind und die Aufeinanderfolge einer gewissen Regellosigkeit unterliegt, dann eignet sich eine solche Folge für die Simulation der Erzeugung von Zufallszahlen. Die von RND erzeugten Zufallszahlen werden als *Pseudozufallszahlen* (PZZ) bezeichnet, weil ihr Erzeugungsmechanismus auf eindeutig bestimmten Rechenvorschriften beruht.

Es ist ein Algorithmus zu entwickeln, der ausgehend von einem Auswahlmenü verschiedene Möglichkeiten bietet, die mittels RND erzeugten ‚Zufälligkeiten' zu demonstrieren. Dabei sollen von einem Hauptmenü

1 Erzeugen von Pseudozufallszahlen bestimmter Bereiche
2 Spiel mit dem Zufall: Lotto, Würfeln, ...
3 Zufallstexte
4 Zufallsmuster
5 Zufallsgrafik
6 Zufallsbewegungen
0 Programm beenden

Untermenüs wählbar werden, von denen ausgehend die einzelnen Teilprogramme aufrufbar sind. Untermenüs sollen im prinzipiellen Aufbau dem 1. Untermenü gleichen:

1 Pseudozufallszahlen im Intervall <0, 1)
2 PZZ im Intervall <0, x) mit $x > 0$
3 PZZ aus der Menge {1, 2, 3, ..., x} mit $x > 1$ und ganzzahlig
4 PZZ aus der Menge {x, x+1, x+2, ..., y} mit $y > x$ und x, y ganzzahlig
0 zurück ins Hauptmenü.

Damit das Arbeiten mit Menüs rationell wird, sind mindestens 3 Wahlmöglichkeiten anzubieten. Es sollten aber wegen der Übersichtlichkeit nicht mehr als 7 Auswahlmöglichkeiten in einem Menü erfaßt werden. Die Kennzahl 0 dient stets dem Übergang in das übergeordnete Menü bzw. vom Hauptmenü zum Programmende.
Es sollen hier nicht alle Teilalgorithmen entwickelt werden. Wichtig ist, daß erkennbar wird, wie man eine derartige hierarchische Struktur realisieren kann und wie man später ergänzende Programme in das entwickelte Gesamtprogramm eingliedern kann.

Jedes Untermenü ist von vergleichbarer Struktur.
Die 4 Teilalgorithmen des Untermenüs 1 sind sehr ähnlich.
Deshalb werden nur die Teilalgorithmen 1 und 4 als Struktogramm dargestellt.
Die Funktion RND(1) erzeugt Zufallszahlen im Intervall <0, 1). Man kann sich geometrisch veranschaulichen, wie diese Zahlenmenge abgebildet wird, beispielsweise auf die Zahlen 1, 2, 3, 4, 5, 6, die beim Würfeln eine Rolle spielen.
Wie man aus den beiden Struktogrammen erkennen kann, haben die Teilalgorithmen gemeinsame, gleichlautende Anteile. Mehrfach Gleiches in Programmen zu schreiben ist wegen des zusätzlichen Arbeitsaufwandes und wegen des Programmspeicherbedarfs unrationell.

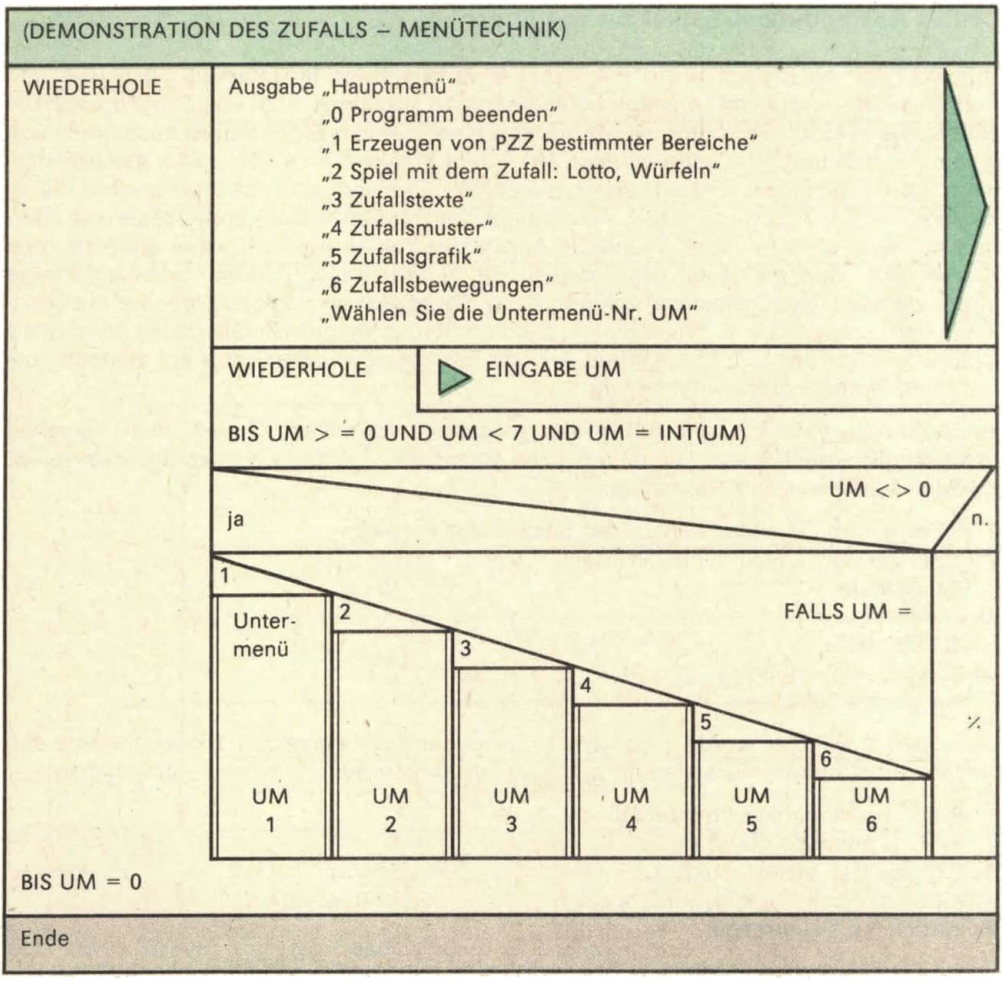
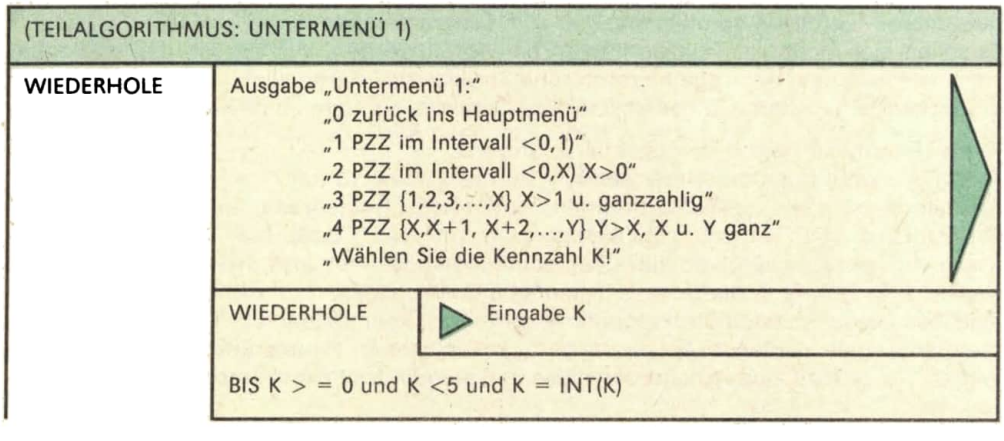


Abb. 2.3/27: Struktogramm „Demonstration des Zufalls"



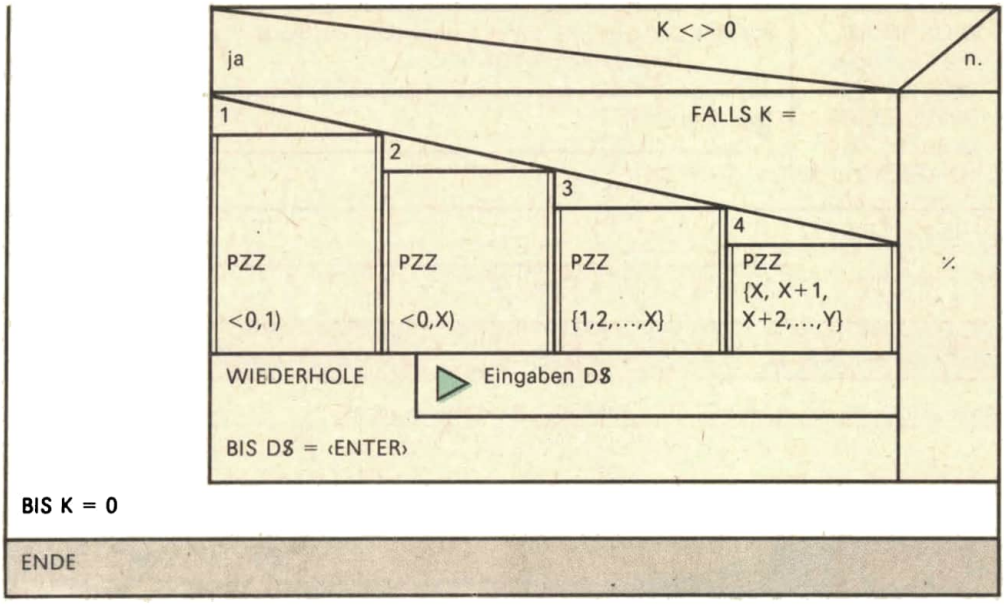


Abb. 2.3/28: Struktogramm „Untermenü 1"

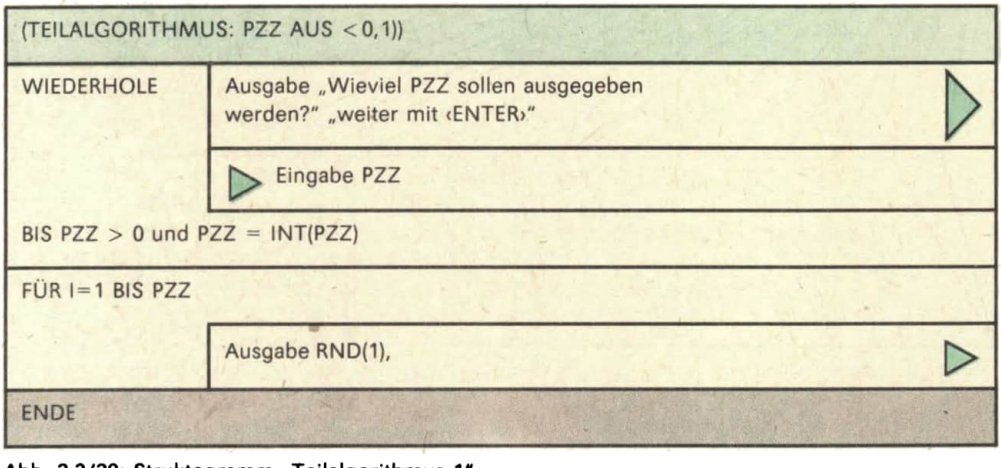
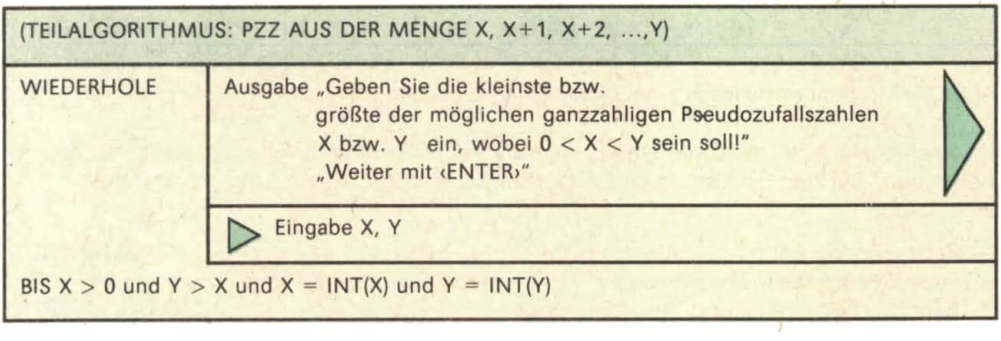


Abb. 2.3/29: Struktogramm „Teilalgorithmus 1"



WIEDERHOLE	Ausgabe „Wieviel PZZ sollen ausgegeben werden?" „Weiter mit ‹ENTER›"
	Eingabe PZZ
BIS PZZ > 0 und PZZ = INT(PZZ)	
FÜR I = 1 BIS PZZ	
	Ausgabe INT((Y−X+1) *RND (1)) + X,
ENDE	

Abb. 2.3/30: Struktogramm „Teilalgorithmus 4 des Untermenü 1"

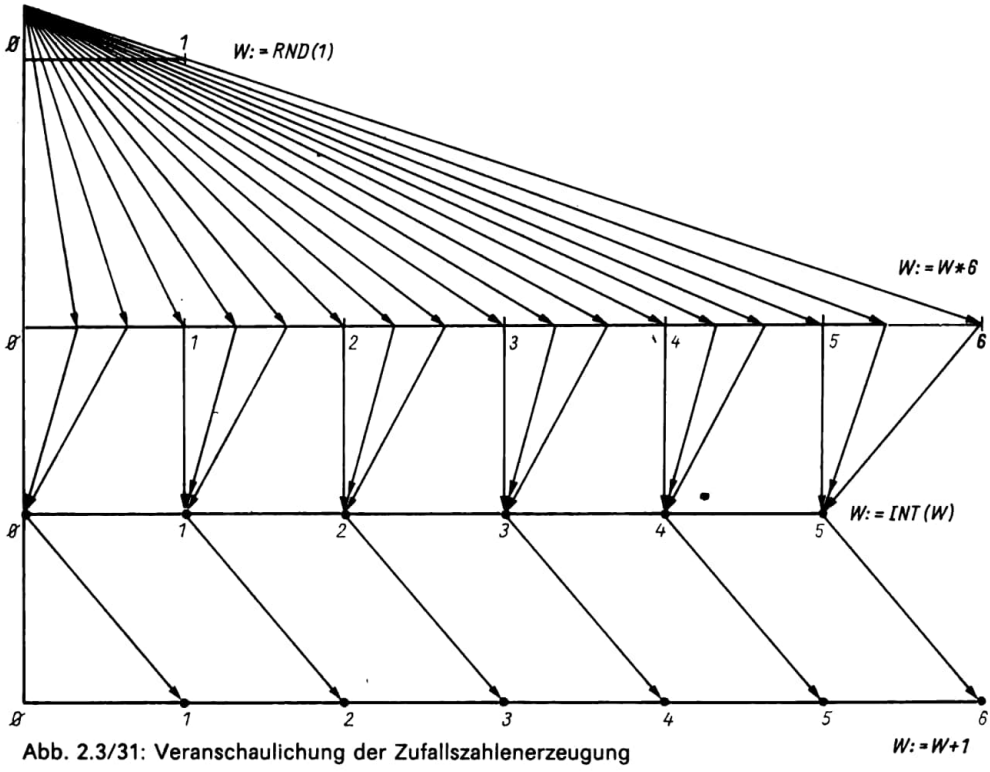


Abb. 2.3/31: Veranschaulichung der Zufallszahlenerzeugung

Im nächsten Kapitel wird die Unterprogrammtechnik behandelt, um diesem Nachteil zu begegnen. Bei der Codierung des Algorithmus sind die Anweisungsnummern wie folgt zu wählen:

- Rahmenprogramm mit Hauptmenü — 100 bis 990,
- Ende der Wiederholschleife des Rahmenprogramms und Programmende — 19000 bis 19990,

- alle Anweisungen zum
  - Untermenü 1 2000 bis 3990,
  - Untermenü 2 4000 bis 5990,
  - Untermenü 3 6000 bis 7990,
  - Untermenü 4 8000 bis 9990,
  - Untermenü 5 10000 bis 11990 und
  - Untermenü 6 12000 bis 13990.

Innerhalb der Untermenüs erfolgt ebenfalls eine analoge Zuordnung der Anweisungsnummern, damit auch durch die Anweisungsnummern die Programmgliederung gestützt wird. So sollte z. B. der Rahmen des Untermenüs 1 von 2000 bis 2190 und das Ende der Wiederholschleife des Untermenüs 1 zwischen 3900 und 3990 liegen. Für den Beginn der einzelnen Programme des Untermenüs 1 sind volle Hunderter zu wählen:

```
2200 REM -----  PZZ aus <0, 1) -----------------------
2210 WINDOW 15,31,0,39 : COLOR 0,6 : CLS
2220 PRINT "Wieviel PZZ sollen ausgegeben werden?"
2230 PRINT AT(29,25);"weiter mit";CHR$(141)
2240 INPUT ""; PZZ
2250 IF NOT(PZZ>0 AND PZZ=INT(PZZ))THEN PRINT"Eingabefehler":
PAUSE9: GOTO2210
2260 FOR I=1 TO PZZ
2270    PRINT RND(1),
2280 NEXT I
2290 GOTO 3900
2390 REM ende PZZ aus <0, 1) -------------------------

2800 REM ----- PZZ aus der Menge X,X+1,X+2,...,Y ------
2810 WINDOW 15,31,0,39 : COLOR 0,6 : CLS
2820 PRINT "Geben Sie die kleinste bzw. groesste der moegl."
2830 PRINT : PRINT "ganzzahligen PZZ X bzw. Y ein,"
2840 PRINT : PRINT "wobei 0 < X < Y sein soll!"
2850 PRINT AT(29,25); "weiter mit"; CHR$(141)
2860 INPUT " X, Y:"; X,Y
2870 IF NOT(X>0 AND Y>X AND X=INT(X) AND Y=INT(Y)) THEN 2880:
ELSE 2900
2880 PRINT : PRINT "Eingabefehler!" : BEEP : PAUSE 20
2890 GOTO 2810
2900 CLS
2910 PRINT "Wieviel PZZ sollen ausgegeben werden?"
2920 PRINT AT(29,25); "weiter mit";CHR$(141)
2930 INPUT ""; PZZ
2940 IF NOT(PZZ>0 AND PZZ=INT(PZZ))THEN PRINT"Eingabefehler":
     PAUSE20: GOTO 2900
2950 FOR I=1 TO PZZ
2960    PRINT ((Y-X+1) * RND(1)) + X,
2970 NEXT I
2990 REM ende PZZ aus der Menge X, X+1, X+2, ..., Y ---
```

Mit PRINT CHR$(141) wird das Zeichen für <ENTER> auf dem Bildschirm dargestellt.

Auf obige Weise lassen sich die einzelnen Menüs und Programme schrittweise entwikkeln. Vor Fertigstellung des Gesamtprogramms lassen sich alle einzelnen Wege testen, wenn die Untermenürahmen entworfen sind. Als Muster dient der Rahmen zum Untermenü 2:

```
4000 REM ----- UNTERMENUE 2 ----- SPIEL MIT DEM ZUFALL -----
.... ....
.... ....
5900 REM ende der mehrs. auswahl des untermenues 2
5910 PRINT AT(29,25); "weiter mit";CHR$(141)
5920 REM anfang der tastaturabfrage
5930 I$=INKEY$ : IF I$="" THEN 5930
5940 REM ende der tastaturabfrage
.... ...
5980 GOTO 19900
5990 REM ende des untermenues 2 -------------------------
```

Die Anweisung
INKEY$
auf Zeile 5930 ist eine Eingabeanweisung, die die Eingabe eines einzelnen Zeichens über Tastatur ermöglicht. Im Moment der Abarbeitung dieses Befehls wird geprüft, ob eine Eingabe gemacht wurde. War das der Fall, dann wird das eingegebene Zeichen der Zeichenkettenvariablen I$ zugewiesen. Lag keine Eingabe zum Abfragezeitpunkt vor, so läuft das Programm weiter.

In Zeile 5930 wird in der zweiten Anweisung abgefragt, ob die Zeichenkette I$ noch gleich der Leerzeichenkette ist oder nicht. Wird der Test mit ‚wahr' beantwortet, erfolgt ein Sprung wieder zur Ergibtanweisung mit INKEY$. Die Schleife wird erst verlassen, wenn I$ durch eine Tastenbelegung ein Wert zugewiesen wurde.

Zusammenfassung

Bei der Konzipierung eines Algorithmus mit Auswahlmenüs sind gewisse Regeln zu beachten:
- Die Kennzahl 0 ist stets für den Übergang ins übergeordnete Menü bzw. zum Algorithmusende zu verwenden.
- Die Anzahl der Auswahlmöglichkeiten sollte 3 bis 7 betragen.

Mit der Anweisung

SOUND hoehe1, vorteiler1, hoehe2, vorteiler2, lautstaerke,tondauer

lassen sich in BASIC zweistimmige Melodien erzeugen.

Zur Erzeugung eines Tones fester Frequenz existiert im KC-BASIC der Befehl **BEEP** mit dem Format

BEEP [n] mit 1 <= n <= 255

Der Zufall kann mittels der Pseudozufallszahlenfunktion

RND(x)

simuliert werden.

Soll die Eingabe eines einzelnen Zeichens programmiert werden, so kann die Anweisung

zeichenkette = INKEY$

verwendet werden.

Aufgaben

1. Erweitern Sie das Menü des Programms ‚Melodien aus Peter und der Wolf' um das Vogel-, das Ente- und das Großvater-Motiv! Die Notenbilder sind in Abbildung 2.3/32 dargestellt.
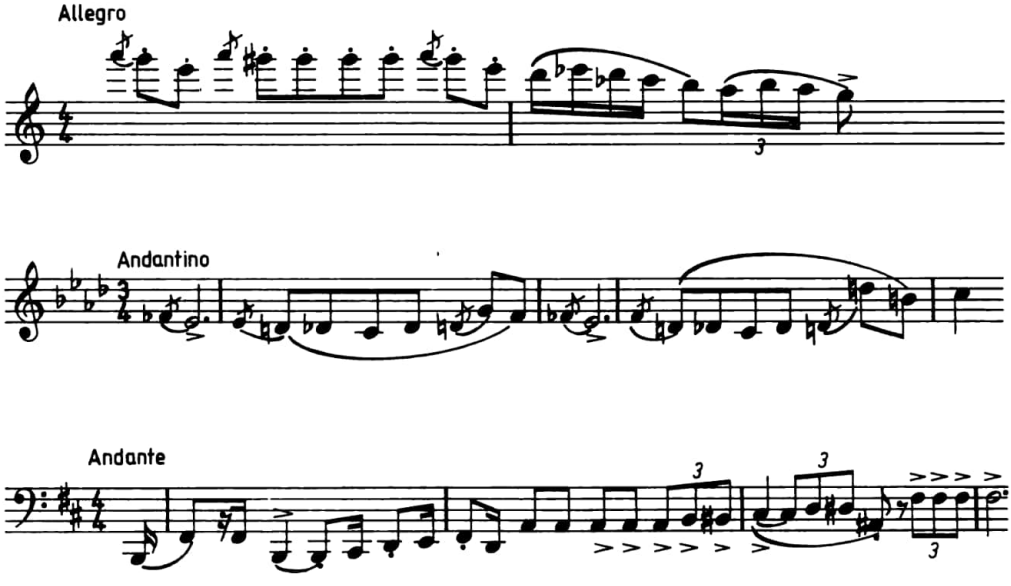


Abb. 2.3/32: Vogel-, Ente- und Großvater-Motiv

Modifizieren Sie entsprechend den Zulässigkeitstest für die Kennzahl K und die mehrseitige Auswahl zum Stellen des Datenzeigers! Ergänzen Sie die Datenliste durch Daten zum Vogel-, Ente- und Großvater-Motiv!

2. Eine der bekanntesten Codierungen ist die Codierung nach Morse, die im Tastfunk in der Form des Morse-Alphabets Anwendung findet. Entwickeln Sie ein Programm, das erstens die Morsezeichenfolge auf dem Bildschirm ausgibt und zweitens die entsprechende Folge von langen und kurzen Tönen erzeugt! Der zu codierende Text soll Großbuchstaben, Ziffern und Leerzeichen enthalten können und nicht länger als eine Zeile sein.

3. Schreiben Sie ein Programm, mit dem Sie am Computer die Melodie eines für den Musikunterricht zu erlernenden Liedes, z. B. für das ‚Lied der roten Matrosen', solange wiederholen können, bis Sie sich die Melodie eingeprägt haben!

4. Entwickeln Sie zum Programm ‚Demonstration des Zufalls' (↗ 2.3.4) die ersten 3 Programme des Untermenüs ‚2 Spiel mit dem Zufall':

   0 zurück ins Hauptmenü
   1 Tele-Lotto
   2 6 aus 49
   3 5 aus 45
   4 Würfeln mit 1 Würfel
   5 Würfeln mit 2 Würfeln
   6 Würfeln mit 3 Würfeln

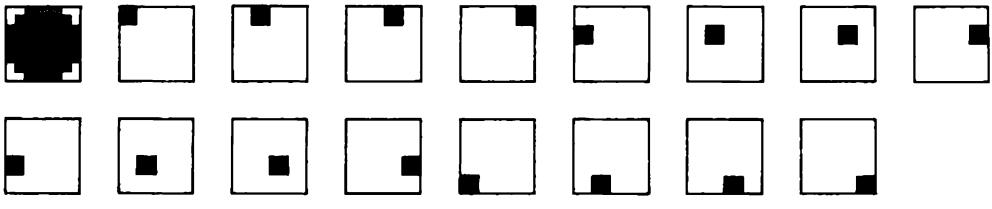
   Dabei ist beim ‚Ziehen' der Lottozahlen zu berücksichtigen, daß möglicherweise eine Zahl mehrfach gezogen wird. Bei den Spielen ‚6 aus 49' bzw. ‚5 aus 45' sind Zusatzzahlen zu ziehen.
   Die Gewinnzahlen sollen in geordneter Reihenfolge ausgegeben werden. Benutzen Sie für die Programme zum Untermenü 2 die Anweisungsnummern 4000 bis 5990!

5. Entwickeln Sie zum Programm ‚Demonstration des Zufalls' die 4 Programme des Untermenüs ‚4 Zufallsmuster':

   0 zurück ins Hauptmenü
   1 Labyrinth mit \ und /
   2 Labyrinth mit Quasigrafiksymbolen



   3 ‚Es schneit' mit Quasigrafiksymbolen



   4 ‚Gebogenes' mit den Quasigrafiksymbolen



   Sind die Quasigrafiksymbole auf dem Rechner nicht verfügbar, sind sie selbst zu entwerfen, auf einem geeigneten Speicherplatz abzulegen und der Zeichenbildtabellen-Zeiger entsprechend einzustellen. Benutzen Sie für die Programme zum Untermenü 4 die Anweisungsnummern 8000 bis 9990!

6. Entwickeln Sie zum Programm ‚Demonstration des Zufalls' die 4 Programme des Untermenüs ‚5 Zufallsgrafik':

   0 zurück ins Hauptmenü
   1 zufällige Farbrahmen
   2 zufällige Dreiecke
   3 Flickenteppich
   4 Symmetrisches

Unter zufälligen Farbrahmen sind dabei Rechtecke zu verstehen, die mit viermaligem Anwenden des LINE-Befehls erzeugt werden. Dabei soll im Punktraster die linke untere Ecke eines Rechtecks zufällig variierbar sein zwischen 0 und 250 in der Horizontalen und zwischen 30 und 180 in der Vertikalen. Die Breite bzw. Höhe soll ebenfalls zufällig bestimmt werden und zwar variierbar sein zwischen 0 und 70 Punkte bzw. zwischen 0 und 50 Punkte. Die Farben sind zufällig aus der Menge {schwarz, blau und rot} auszuwählen. Für die zufälligen Dreiecke sind die Koordinaten eines jeweiligen Dreiecks mit RND zu ermitteln. Beim Flickenteppich sind die Parameter für Bildschirmfenster mit RND zu bestimmen und mit einer zufällig ermittelten Farbe einzufärben.
Beim Programm ‚Symmetrisches' sind Kursorpositionen (Zeile, Spalte) im Zeichenraster mit RND auszuwählen, und es ist die zur vertikalen Mittellinie symmetrisch liegende Position zu bestimmen. Dort sind entweder ein Leerzeichen oder ein Vollkursor in einer zufällig bestimmten Farbe zu setzen.
Benutzen Sie für die Programme zum Untermenü 5 die Anweisungsnummern 10000 bis 11990!

2.3.5. Unteralgorithmen

Kommt in einem Algorithmus eine Folge mehrfach vor, so ist es sinnvoll, sie einmal als Unteralgorithmus zu erfassen und an allen Stellen, an denen dieser Unteralgorithmus benötigt wird, aufzurufen. Das bringt folgende Vorteile:
- geringerer Platzbedarf,
- durch die Verlagerung von Einzelheiten in Unteralgorithmen wird der Hauptalgorithmus kürzer und damit übersichtlicher,
- durch die Gliederung in übergeordnete und untergeordnete Algorithmen wird die Aufgabe der einzelnen Unteralgorithmen deutlicher und der Gesamtalgorithmus verständlicher,
- Unteralgorithmen können unabhängig vom Hauptalgorithmus entwickelt, geändert und ausgetauscht werden,
- die Lokalisierung von Fehlern wird begünstigt.

Das hier Gesagte gilt auch für die den Unteralgorithmen entsprechenden Programme.

Vereinbarung und Aufruf von Unteralgorithmen

Die Unteralgorithmusvereinbarung wird wie folgt dargestellt:

verbal formalisierte Notation	Struktogramm
UA unteralgorithmusname vereinbarungsteil BEGINN folge ENDE	(UA unteralgorithmusname) (vereinbarungsteil) BEGINN folge ENDE

Umsetzung in BASIC
ak REM upname al REM bemerkungen zu eingangs-, ausgangs- und lokale variablen am folge an RETURN

Der Unteralgorithmus kann von jeder Stelle des Hauptalgorithmus bzw. anderer Unteralgorithmen aufgerufen werden. Die Darstellung des Aufrufs erfolgt in folgender Weise:

verbal formalisierte Notation	Struktogramm
RUFE unteralgorithmusname	RUFE unteralgorithmusname
Umsetzung in BASIC	
ap GOSUB ak	

Viele BASIC-Versionen lassen nicht zu, ein Unterprogramm mit dessen Namen aufzurufen. Die GOSUB-Anweisung vermittelt das Springen zum Unterprogramm, das auf Programmzeile ak beginnt. Der Computer speichert beim Aufruf auch die Rücksprungadresse. Wenn am Ende der Abarbeitung des aufgerufenen Unterprogramms die RETURN-Anweisung erreicht wird, erfolgt der Rücksprung ins rufende Programm zur Anweisung, die auf GOSUB ak folgt.

Es sind ein Algorithmus und ein Programm zur Berechnung des Binominalkoeffizienten m über k für beliebige natürliche Zahlen m, k mit $m >= k$ zu entwickeln.

Eingabe: Die natürlichen Zahlen m, k

Verarbeitung: Eine Definition für den Binominalkoeffizienten lautet:

$$\binom{m}{k} = \frac{m!}{k!(m-k)!}$$

Für die 3 Zahlen m, k, $m-k$ sind die Fakultäten zu berechnen. Im Beispiel 6 der Zählschleife wurde ein Programm zur Berechnung von N! entwickelt, das nun in modifizierter Form als Unterprogramm verwendet werden kann.

Ausgabe: Wert des Binominalkoeffizienten m über k

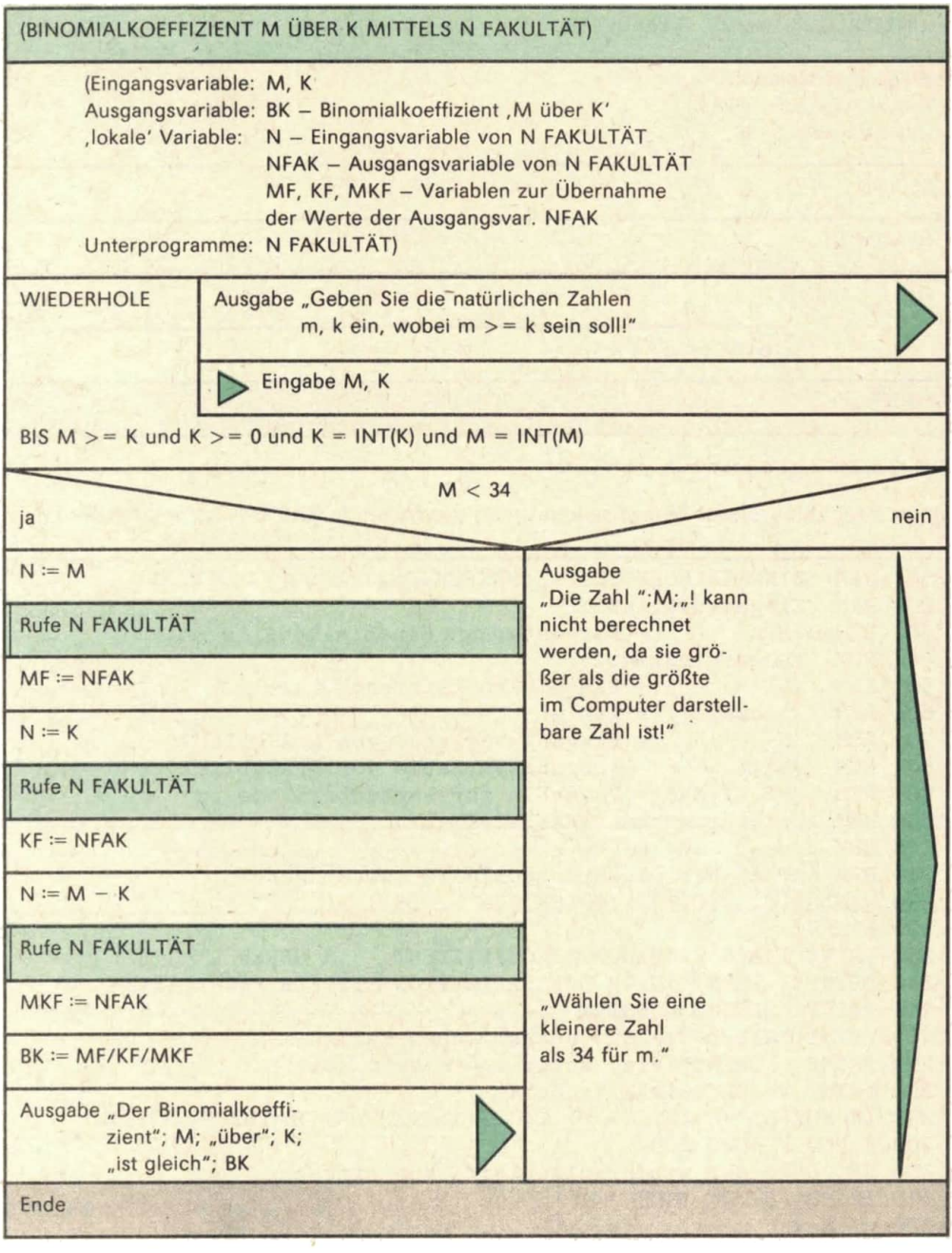


Abb. 2.3/33: Hauptalgorithmus „Binomialkoeffizient"

Es sind vor der Nutzung des Unteralgorithmus die jeweiligen aktuellen Werte an die Eingangsvariable zu übergeben. Die Werte der Ausgangsvariablen eines Unteralgorithmus sind den entsprechenden Variablen des übergeordneten Algorithmus zuzuweisen.

(UNTERALGORITHMUS: N FAKULTÄT)
(Eingangsvariable: N Ausgangsvariable: NFAK lokale Variable: I)
BEGINN
NFAK := 1
FÜR I = 1 BIS N
NFAK := NFAK * I
ENDE

Abb. 2.3/34: Unteralgorithmus „N Fakultät"

Unter Beachtung dieser Hinweise kann man das folgende BASIC-Programm schreiben:

```
10  REM  BINOMIALKOEFFIZIENT M UEBER K MITTELS N FAKULTAET
20  REM  Eingangsvariable:
30  REM   M,K        - Parameter des Binomialkoeffizienten
40  REM  Ausgangsvariable:
50  REM   BK         - Binomialkoeffizient 'M ueber K'
60  REM   'lokale' Variable:
70  REM   N          - Eingangsvariable von NFAKULTAET
80  REM   NFAK       - Ausgangsvariable von NFAKULTAET
90  REM   MF,KF,MKF - Variable zur Wertuebernahme
100 REM  Unterprogramm: NFAKULTAET
110 REM ------------------------------------------------
200 REM anfang der wiederholschleife zur eingabe
210 WINDOW 0,31,0,39 : COLOR 0,6 : CLS
220 PRINT
230 PRINT TAB(4); "BINOMIALKOEFFIZIENT    M UEBER K"
240 PRINT : PRINT "NACH DER DEFINITION MITTELS FAKULTAETEN"
250 PRINT : PRINT : PRINT
260 PRINT "Geben Sie die natuerlichen Zahlen m,k"
270 PRINT : PRINT "ein, wobei m>=k sein soll!"
280 PRINT : INPUT "m,k:"; M, K
290 IF NOT(M>=K AND K>=0 AND K=INT(K) AND M=INT(M)) THEN
GOSUB 700 : GOTO 200
300 REM ende der wiederholschleife zur eingabe
310 IF NOT(N<34) THEN 460
320     N=M
330     GOSUB 600 : REM → up: n fakultaet
340     MF=NFAK
350     N=K
360     GOSUB 600 : REM → up: n fakultaet
370     KF=NFAK
```

```
380     N=M-K
390     GOSUB 600 : REM → up: n fakultaet
400     MKF=NFAK
410     BK=MF/KF/MKF
420     LOCATE 20,3
430     PRINT "Der Binomialkoeffizient";M;"ueber";K
440     PRINT : PRINT "ist gleich";BK
450 GOTO 530
460     WINDOW 0,31,0,39 : COLOR 2,6 : CLS
470     PRINT : PRINT "Die Zahl";M;"! kann nicht berechnet"
480     PRINT : PRINT "werden, da sie groesser"
490     PRINT : PRINT "als die groesste im Computer"
500     PRINT : PRINT "darstellbare Zahl ist!"
510     PRINT:PRINT:PRINT "Waehlen Sie eine kleinere Zahl"
520     PRINT : PRINT "als 34 fuer m."
530 REM ende der zweiseitigen auswahl
540 END
600 REM UP: N FAKULTAET -------------------
610 REM Eingangsvariable: N
620 REM Ausgangsvariable: NFAK
630 REM 'lokale' Variable: I
640 NFAK=1
650 FOR I=1 TO N
660    NFAK=NFAK*I
670 NEXT I
680 RETURN
700 REM UP: ZUR FEHLERBEHANDLUNG ---------
710 PRINT AT(29,20);"EINGABEFEHLER"
720 BEEP (5)
730 RETURN
```

Der Aufruf des Unterprogramms ist in Abbildung 2.3/35 schematisch dargestellt.

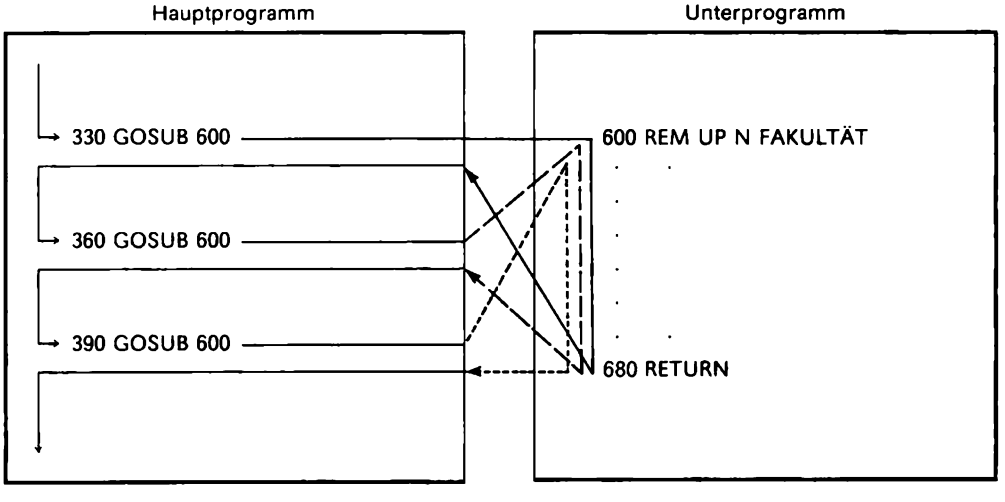


Abb. 2.3/35: Schematische Darstellung des Programmablaufs

Weil 35! größer als die größte Zahl der Computerzahlenmenge ist, kann man mit diesem Programm nicht einmal berechnen, wieviel Möglichkeiten es gibt, 5 Zahlen aus 35 Zahlen ohne Berücksichtigung der Reihenfolge auszuwählen. Schreibt man das Beispiel geeignet auf

$$\binom{35}{5} = \frac{35!}{5!\ 30!} = \frac{35 * 34 * 33 * 32 * 31 * 30!}{1 * 2 * 3 * 4 * 5 * 30!}$$

so erkennt man, daß der Binomialkoeffizient auch in der Form

$$\binom{m}{k} = \frac{\prod_{i=0}^{k-1} (m-i)}{\prod_{i=1}^{k} i} = \frac{m * (m-1) * (m-2) * \ldots * (m-(k-1))}{1 * 2 * 3 * \ldots * k}$$ dargestellt werden kann.

- Entwickeln Sie einen Algorithmus zur Berechnung von $\binom{m}{k}$ nach der rekursiven Vorschrift

$$\begin{cases} a_1 = m \\ a_{i+1} = a_i \frac{m-i}{i+1} & \text{für } i = 1, 2, \ldots, k-1 \quad \text{wobei } k >= 2 \text{ sei!} \end{cases}$$

- Testen Sie ein entsprechendes BASIC-Programm vor allem mit Werten für m und k, die mit dem BINOMIALKOEFFIZIENTEN M ÜBER K MITTELS N FAKULTÄT nicht berechnet werden können!
Erweitern Sie Ihr Programm so, daß es auch auf die Eingaben $k=0$ und $k=1$ die richtigen Ausgaben liefert.

- Stellen Sie mit einer Stoppuhr fest, wieviel Zeit für die Berechnung z. B. von $\binom{110}{100}$ mit Ihrem Programm benötigt wird. Nutzen Sie die Berechnung $\binom{m}{k} = \binom{m}{m-k}$ zur Verbesserung des Programms hinsichtlich der Rechenzeit!

Vereinbarung und Aufruf von benutzerdefinierten Funktionen

Es besteht in BASIC die Möglichkeit, Funktionen selbst zu definieren und sie wie Standardfunktionen zu benutzen. Diese sind dem Sinne nach Ergibtanweisungen, die jeweils auf einer einzigen Anweisungszeile Platz finden müssen. Die Parameterübergabe für genau einen ausgewiesenen Parameter erfolgt dadurch, daß der aktuelle Wert als Argument der benutzerdefinierten Funktion – so wie bei allen Standardfunktionen von BASIC – geschrieben wird.
Zu vereinbaren sind die benutzerdefinierten Funktionen wie folgt:

verbal formalisierte Notation	Struktogramm
FUNK fname(var)=ausdruck	FUNK fname(var)=ausdruck
Umsetzung in BASIC	
ad DEF FN fname(var)=ausdruck	

Der Aufruf erfolgt dann

verbal formalisierte Notation	Struktogramm
fname(ausdruck)	fname(ausdruck)
Umsetzung in BASIC	
an FN fname(ausdruck)	

FNfname(ausdr) kann auch ein Bestandteil eines weiteren Ausdrucks sein.
Die Vereinbarung einer benutzerdefinierten Funktion muß im Programm vor dem ersten Aufruf stehen.

Zusammenfassung

Unteralgorithmen können die Algorithmen effektivieren. Der Hauptalgorithmus stellt dann eine übersichtliche Folge von Entscheidungen und Verzweigungen in Unteralgorithmen dar.
Unteralgorithmen werden vereinbart und sind auch aus den algorithmischen Elementen Folge, Auswahl, Wiederholung und Unteralgorithmenaufruf aufgebaut.
Ein Unterprogrammaufruf kann von jeder Stelle eines Programms in BASIC mit **GOSUB** erfolgen. Dabei wird die Rückkehradresse gespeichert. Die **RETURN**-Anweisung bewirkt den Rücksprung ins rufende Programm zu der auf die GOSUB-Anweisung folgenden Anweisung.

Benutzerdefinierte Funktionen stellen Unterprogramme dar, die mit Standardfunktionen vergleichbar sind und deren Vereinbarung mit **DEF FN** jeweils auf einer Programmzeile Platz finden müssen. Der Aufruf dieser Funktion erfolgt mit **FNfname(var)**.
Die Vereinbarung einer benutzerdefinierten Funktion muß im BASIC-Programm vor dem ersten Aufruf stehen.

● Aufgaben

1. Es ist ein Programm zu schreiben, mit dessen Hilfe die Binomialkoeffizienten
   $\binom{m}{0}, \binom{m}{1}, \ldots, \binom{m}{n}$
   für vorgegebene natürliche Zahlen m, n mit $m>=n>=0$ berechnet und ausgegeben werden.
2. Geben Sie benutzerdefinierte Funktionen für arcsin und arccos an. Verwenden Sie nicht die oft angegebenen Formeln
   $\arcsin x = \arctan \frac{x}{\sqrt{1-x^2}}$ bzw. $\arccos x = .5 * PI - \arctan \frac{x}{\sqrt{1-x^2}}$,

weil sie für $x = \pm 1$ wegen Division durch 0 zum Fehler führen, obwohl arcsin(±1) und arccos(±1) definiert sind. Nutzen Sie die Formeln

$$\arcsin x = 2\arctan \frac{x}{1+\sqrt{1-x^2}} \quad \text{bzw.} \quad \arccos x = 5 * PI - 2\arctan \frac{x}{1+\sqrt{1-x^2}} !$$

3. Erstellen Sie ein Programm, in dem zwei einzugebende Brüche addiert werden sollen. Das Kürzen der Brüche ist mit einem geeigneten Unterprogramm zu realisieren. In diesem soll ein Unterprogramm „ggt" (größter gemeinsamer Teiler) benutzt werden.

2.4. Datenstruktur Feld

Bisher wurden Variablen oder Konstanten vom Datentyp (Zahl, Zeichenkette) verwendet. Es gibt viele Probleme, bei denen eine Zusammenfassung von Daten unter einem gemeinsamen Namen sinnvoll ist, wie z. B. die Häufigkeiten des Auftretens der Augenzahlen 1 bis 6 beim Würfeln, Schülernamen im Klassenbuch, Zahlen in einem rechteckigen Schema, Zeilen in einer Textseite oder die Koeffizienten einer ganzrationalen Funktion.

> Ein Feld ist eine Menge von Daten vom gleichen Datentyp, die in einer Reihenfolgeanordnung stehen.

Jedes Feldelement ist in dieser Reihenfolge durch Ordnungszahlen (Indizes) eindeutig festgelegt, so daß durch die Angabe des für alle Elemente einheitlichen Feldnamens und der Indizes direkt auf ein bestimmtes Element zugegriffen werden kann.

Ist nur ein Index zur Positionierung eines Feldelements notwendig, spricht man von einem *eindimensionalen Feld*. Sind zwei Indizes zur Positionierung eines Feldelements notwendig, so liegt ein *zweidimensionales Feld* vor usw. Da zweidimensionale Felder eineindeutig auf rechteckige Schemata aus Zeilen und Spalten abbildbar sind, ist es bei Feldern mit zwei Indizes üblich, von Zeilen und Spalten zu sprechen, in denen die Feldelemente stehen.

Theoretisch darf ein Feld viele Indizes besitzen (beim BASIC-Interpreter für Kleincomputer bis 255 Indizes), aber das ist praktisch nicht möglich, weil die Programmzeilenlänge nicht ausreichen würde, wenn ein Feldelement mit 255 Indizes angegeben werden sollte. In der Praxis reichen Felder aus, die bis zu drei Indizes besitzen (dreidimensionale Felder).

Da der Computer für jede einfache Variable einen Speicherplatz reserviert, muß bei der Arbeit mit Feldern die Anzahl der benötigten Speicherplätze vereinbart werden. Das geschieht mit der DIM-Anweisung, die das Format

```
DIM feldname(index[,index]...)[,feldname(index[,index]...]...
```

hat.
Dabei legt der Parameter ‚index' die höchste Ordnungsnummer der Feldelemente einer Dimension fest.
Der Wert für ‚index' muß zwischen 0 und 32766 liegen. Gleichzeitig mit der Reservierung des Speicherplatzes veranlaßt die DIM-Anweisung das Belegen aller Elemente eines Zahlenfeldes mit dem Wert 0 bzw. eines Zeichenkettenfeldes mit der leeren Zeichenkette "".

Anwendungsbeispiel für Datenstruktur Feld

1 Es ist ein Algorithmus zur Überprüfung von quadratischen Zahlenschemata zu entwikkeln. Es soll festgestellt werden, ob diese als „magisch" zu bezeichnen sind.

Ein magisches Quadrat der Ordnung n besteht aus *n* Zeilen und *n* Spalten. Darin sind die natürlichen Zahlen 1 bis *n* * *n* so angeordnet, daß die Summe der Zahlen jeder Zeile, Spalte und Diagonalen denselben Wert, nämlich die magische Zahl M, ergibt.

Dem quadratischen Zahlenschema wird ein Feld *A* der Dimension 2 zugeordnet. Jedes Element des Feldes *A* ist eine zweifach indizierte Variable *A*(i,j). Der erste Index gibt an, in welcher Zeile, und der zweite Index, in welcher Spalte sich die Variable befindet.

Eingabe: Die Ordnung N des Zahlenschemas,
die natürlichen Zahlen 1,2,...,N*N in einer vom Programmnutzer bestimmten Reihenfolge

Verarbeitung: Bildung der Summen jeder Zeile, jeder Spalte und der Diagonalen,
Vergleich der Summen (Bei Gleichheit aller Summen liegt ein magisches Quadrat vor).

Ausgabe: Zahlenquadrat und einen der Texte:
„Es ist kein magisches Quadrat"
„Es ist ein magisches Quadrat mit der magischen Zahl";M

```
(MAGISCHE QUADRATE – TEST)
Ausgabe „Sind in einem quadratischen Zahlenschema der
        Ordnung n (n Zeilen, n Spalten) die natürlichen Zahlen 1 bis n * n so angeordnet, daß alle Summen der
        Zahlen jeder Zeile, Spalte und Diagonalen denselben
        Wert, nämlich die magische Zahl M, ergeben, so
        liegt ein magisches Quadrat vor!"
WIEDERHOLE
    Alle Variablen löschen
    WIEDERHOLE
        Ausgabe „Geben Sie die Ordnung (n < 11) ein!"
        Eingabe N
    BIS N > 0 und N < 11 und N = INT(N)
    Feld dimensionieren
    RUFE ZAHLENFELDEINGABE
    RUFE ZAHLENFELDAUSGABE
    RUFE ZEILENSUMMENBILDUNG ZS
    RUFE SPALTENSUMMENBILDUNG SS
    RUFE HAUPTDIAGONALENSUMME HS
```

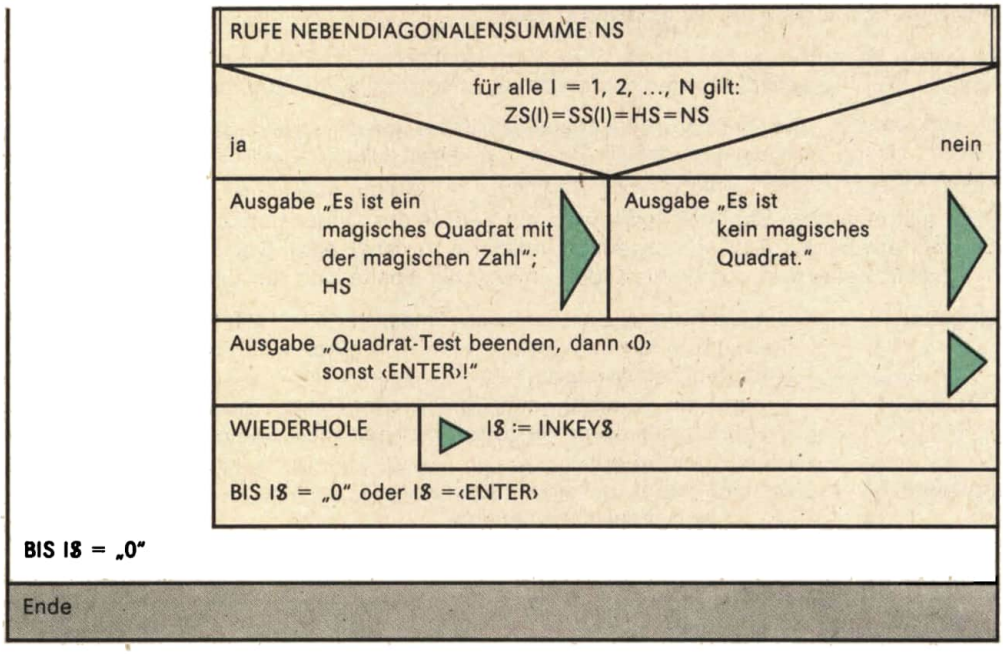


Abb. 2.4/1: Struktogramm „Magische Quadrate"

Der Übersichtlichkeit halber ist es günstig, die Ausarbeitung von Teilalgorithmen von der des Hauptalgorithmus zu trennen. Der Hauptalgorithmus ist in unserem Beispiel zwar nicht bis in jedes Detail ausgeführt, aber alle Teile sind vorhanden. Es ist darauf zu achten, daß der Hauptalgorithmus leicht lesbar ist und auf einer Druckseite Platz findet.

Unteralgorithmen besitzen oft einen solchen Allgemeinheitsgrad, daß sie als Unteralgorithmen auch bei anderen Lösungen Verwendung finden können. Daher ist es ratsam, sie in eine Programmbibliothek aufzunehmen.

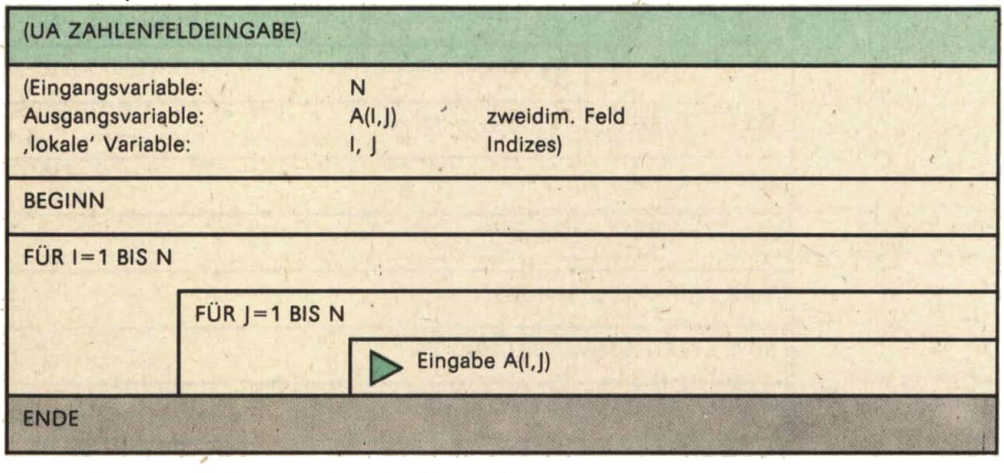


Abb. 2.4/2: Unteralgorithmus „Zahlenfeldeingabe"

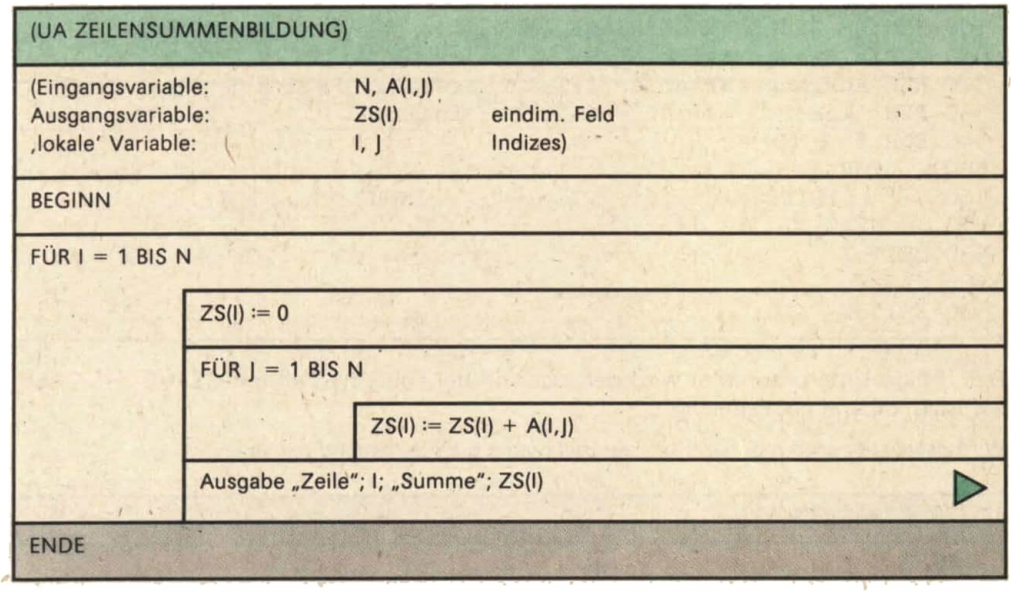


Abb. 2.4/3: Unteralgorithmus „Zeilensummenbildung"

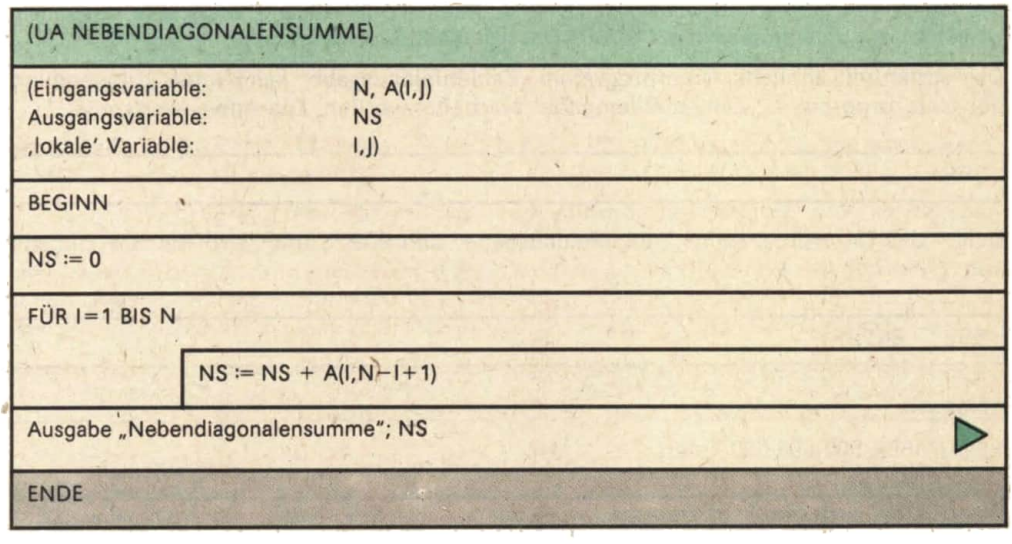


Abb. 2.4/4: Unteralgorithmus „Nebendiagonalensumme"

Die 6 Unteralgorithmen Zahlenfeldeingabe, Zahlenfeldausgabe, Zeilensummenbildung, Spaltensummenbildung, Hauptdiagonalensumme und Nebendiagonalensumme ähneln sich sehr. Deshalb werden in den Abbildungen 2.4/2 bis 2.4/4 nur die Unteralgorithmen Zahlenfeldeingabe, Zeilensummenbildung und Nebendiagonalensumme angegeben.

Bei der Programmierung in BASIC empfiehlt es sich, für die Programmzeilen zum Unteralgorithmus 'Zahlenfeldeingabe' die Anweisungsnummern 500 bis 590 zu wählen.

```
500 REM UP ZAHLENFELDEINGABE
510 REM Eingangsvariable: N
520 REM Ausgangsvariable: A(I,J)  zweidim. Feld N*N
530 REM 'lokale' Variablen: I, J   Indizes
540 FOR I=1 TO N
550    FOR J=1 TO N
560       PRINT "A(";I;",";J;")="; : INPUT "";A(I,J)
570    NEXT J
580 NEXT I
585 CLS
590 RETURN
```

Das fertige Unterprogramm wird getestet und bei Fehlerfreiheit mit CSAVE „EINGABE" auf Magnetband abgespeichert.

Wird das Unterprogramm mit RUN gestartet, endet die Abarbeitung mit einem

```
? RG ERROR IN 590
```

‚RG' kommt von ‚RETURN without GOSUB' und weist daraufhin, daß bei der Programmabarbeitung eine RETURN-Anweisung vor der Ausführung einer GOSUB-Anweisung auftrat. Da das RETURN das Aufsuchen einer Rücksprungadresse veranlaßt und diese aber nicht durch eine GOSUB-Anweisung bei diesem Test gespeichert wurde, kommt es zu dieser Fehlermeldung. Diese wird vermieden, wenn man das Unterprogramm mit GOSUB 500 statt mit RUN startet.

Das strukturell ähnliche Unterprogramm ‚Zahlenfeldausgabe' kann unter Verwendung des Unterprogramms ‚Zahlenfeldeingabe' erarbeitet werden. Die Anweisungszeile

```
585 CLS
```

ist für das Unterprogramm ‚Zahlenfeldausgabe' überflüssig und wird mit der Eingabe von

```
585 <ENTER>
```

gelöscht.

RENUMBER 500,590,600 liefert:

```
600 REM ZAHLENFELDEINGABE
610 REM Eingangsvariable: N
620 REM Ausgangsvariable: A(I,J)  zweidim. Feld N*N
630 REM 'lokale' Variable: I, J   Indizes
640 FOR I=1 TO N
650    FOR J=1 TO N
660       PRINT "A(";I;",";J;")="; : INPUT "";A(I,J)
670    NEXT J
680 NEXT I
690 RETURN
```

Textliche Umformulierungen sind in den Zeilen 600 und 620 vorzunehmen. Eine wesentliche Änderung betrifft die Zeile 660. Sie muß lauten:

```
660        PRINT AT(16-N+2*I,6-N+3*J);A(I,J)
```

Das so entstandene Programm wird einem Test unterzogen:

N=2 : A(1,1)=1 : A(1,2)=2 : A(2,1)=3 : A(2,2)=4 : GOSUB 600

Danach erfolgt das Abspeichern mit CSAVE „AUSGABE".

Auf ähnliche Weise lassen sich die anderen Unterprogramme entwickeln. Liegen alle Teilalgorithmen in Form von Unterprogrammen auf Magnetband vor, bleibt nur noch das Schreiben des Hauptprogramms – entsprechend dem Struktogramm in Abbildung 2.4/1.

Für das Hauptprogramm werden die Anweisungsnummern 10 bis 490 verwendet. So können die 6 Unterprogramme nacheinander vom Magnetband geladen werden. Damit ist das Gesamtprogramm fertig. Es kann getestet und gespeichert werden.

In dem berühmten Kupferstich „Melancholie" von A. Dürer ist ein magisches Quadrat der Ordnung 4 enthalten:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Die mittleren beiden Zahlen der letzten Zeile geben das Jahr 1514 bekannt, in dem das Bild entstand.

In den Unterprogrammen haben wir zur Bildschirmgestaltung die Anweisung PRINT AT(z,s) genutzt, die eine absolute Adressierung auf dem Bildschirm realisiert. Hat man seinen Bildschirm mit mehreren WINDOW-Anweisungen in Bereiche aufgeteilt, so ist eine absolute Adressierung ungünstig. Deshalb gibt es auch eine Möglichkeit zur relativen Adressierung im aktuellen Fenster mit der Anweisung LOCATE mit dem Format

```
LOCATE zeile,spalte
```

Der BASIC-Interpreter prüft die Gültigkeit der Angaben ‚zeile' und ‚spalte' bezüglich des aktuellen Fensters. Durch LOCATE wird die Startposition der folgenden Ausgabe einer PRINT- oder INPUT-Anweisung festgelegt. Beispielsweise würden bei Verwendung von LOCATE im Programm BINOMIALKOEFFIZIENT (↗ S. 124) die Zeilen 220 und 230 lauten:

```
220 LOCATE 1,4
230 PRINT "BINOMIALKOEFFIZIENT     M UEBER K"
```

Zusammenfassung

Neben den einfachen Datentypen **Zahl** und **Zeichenkette** gibt es den Datentyp **Feld**.

Ein Feld ist eine Menge von Daten, deren Elemente alle vom gleichen Datentyp sind und in einer Reihenfolgeanordnung stehen.

Ein Feld ist mit der **DIM**-Anweisung zu vereinbaren und damit wird die Anzahl der Feldelemente festgelegt, die sich im gesamten Programm nicht ändern darf.

Jedes Element ist über Feldname und Indizes direkt aufrufbar und veränderbar.

Die relative Adressierung einer Ausgabe im aktuellen Fenster ist mit dem Befehl **LOCATE zeile,spalte** möglich.

● Aufgaben

1. Verbessern und erweitern Sie das Programm zur Simulation eines automatischen Verkaufs von S-Bahn Fahrausweisen dadurch, daß neben Einzelfahrkarten auch Monats-, Schülermonats- und Dekadenkarten gekauft werden können! Dabei ist folgende Tabelle über ein zweidimensionales Feld zu realisieren:

Preis-stufe	Einzel-fahrkarte M	Monats-karte M	Schüler-monatskarte M	Dekaden-karten M
1	0,20	10,–	5,–	3,30
2	0,30	15,–	7,50	5,–
3	0,50	25,–	12,50	8,30
4	0,70	35,–	17,50	11,60
5	1,–	40,–	20,–	13,30
6	1,10	45,–	22,50	15,–
7	1,20	50,–	25,–	16,60
8	1,30	55,–	27,50	18,30

2. Schreiben Sie ein Programm zur Berechnung von Funktionswerten für Polynome n-ten Grades

   $p(x) = \sum_{i=0}^{n} a_i x^i$, wobei $n <= 20$ gelten soll!

   Die Koeffizienten a_i sind in einem Feld zu erfassen. Es besteht die Identität $a_n x^n + a_{n-1}x^{n-1} + \ldots a_2 x^2 + a_1 x + a_0 = (\ldots((a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0$. Testen Sie die Rechengeschwindigkeit und Genauigkeit der Berechnung des Polynomwerts nach der linken und nach der rechten Seite der Identität aus! Beurteilen Sie, welche Form der Polynomwertberechnung für Programme besser geeignet ist!

3. Die Internationale Standardbuchnummer (ISBN) besteht aus insgesamt 10 Ziffern, die mit Bindestrichen in 4 Gruppen untergliedert sind. Mit der ersten Gruppe kann die Sprache, mit der zweiten der Verleger bzw. eine Gruppe von Verlegern und danach mit der dritten der Buchtitel und die Auflage identifiziert werden. In der 4. Gruppe steht stets eine Ziffer, die Prüfziffer. Mit ihr wird eine automatische Prüfung der Rich-

tigkeit der ISBN ermöglicht. Im Lexikon für Information und Dokumentation findet man auf der Seite 226 einen ‚Algorithmus' zur Prüfziffernberechnung, der im wesentlichen folgende Anweisungen umfaßt:
Unter die ersten 9 Ziffern der ISBN werden in absteigender Reihenfolge die Zahlen 10, 9, 8, ..., 2 geschrieben. Die Ziffer der ISBN wird mit der jeweils darunter stehenden Zahl multipliziert und die Ergebnisse werden zur Endsumme zusammengezählt. Ist die Endsumme durch 11 ohne Rest teilbar, so ist die Prüfziffer 0. Andernfalls ergibt sich die Prüfziffer aus der Differenz zwischen der – bzgl. der Endsumme – nächstgrößeren, durch 11 teilbaren natürlichen Zahl und der Endsumme selbst.
Begründen Sie die Prüfziffer der ISBN des vorliegenden Buches „Informatik"!
Entwickeln Sie ein BASIC-Programm, das jede eingegebene ISBN aufgrund der Prüfziffer auf Richtigkeit untersucht! Berücksichtigen Sie dabei, daß nicht alle Zifferngruppen stets die gleiche Länge besitzen!

4. Um staatliche Planaufgaben zu lösen, ist eine möglichst genaue Prognostik z. B. für die Entwicklung der Altersstruktur zu betreiben. Es ist ein Programm zur Darstellung einer Bevölkerungspyramide zu entwickeln! Dieses Programm ist auszubauen, so daß aufgrund der Eingabe von geschätzten Geburten- und Sterberaten die Bevölkerungsentwicklung simuliert werden kann. Entnehmen Sie die notwendigen Daten dem aktuellen Statistischen Jahrbuch!

5. Entwickeln Sie Programme zur elementaren Textanalyse!
   a) Für verschiedene Sprachen ist im allgemeinen das Verhältnis der Vokale zu den Konsonanten charakteristisch. Schreiben Sie ein Programm, mit dem man einen eingegebenen Text auf die Häufigkeit der Vokale und Konsonanten untersuchen und das Verhältnis beider bestimmen kann!
   b) Schreiben Sie ein Programm, mit dem man ermitteln kann, wie oft in einem eingegebenen Text
      - ein vom Nutzer zu bestimmender Vokal insgesamt vorkommt,
      - auf diesen Vokal ein doppelter Konsonant folgt,
      - auf diesen Vokal ein ‚h' folgt und
      - dieser Vokal doppelt vorkommt!

      Die relativen Häufigkeiten sind mittels Streckendiagramm graphisch darzustellen.
   c) Schreiben Sie ein Programm, das von einem eingegebenen Text
      - die Anzahl der Wörter,
      - die Anzahl der Wörter gleicher Länge,
      - die durchschnittliche Wortlänge ermittelt und die Verteilung der Wortlängen in einem Streifendiagramm graphisch darstellt.

6. Im Schriftverkehr mit gesellschaftlichen Organisationen hat sich eine gewisse Norm herausgebildet. Die äußere Gestaltung beruht auf Übereinkunft und betrifft Wahl des Formats (A 4 oder A 5), Anordnung des Textes, des Briefkopfes, gewisser Vermerke u.a. Aber auch der Inhalt läßt sich bei zweckgebundenen Briefen in gewissen Grenzen normieren. Mit Lücken im Textgerüst werden die Stellen markiert, die der Benutzer eines Programms auszufüllen hat.
   Es ist ein Programm zu entwickeln, das das Verfassen von Zweitbewerbungen um ein Studium entsprechend dem Muster in [4, S. 180] „computergestützt" ermöglicht.

Das Programm ist so zu schreiben, daß die eingerahmten und punktierten Stellen des Briefes vom Programmnutzer ausgefüllt werden können und der dann fertige Zweitbewerbungsbrief ausgedruckt werden kann.

Frank Schäfer

5000 Erfurt, 22. 2. 87
Strasse des Friedens 15

Karl-Marx-Universität
Direktorat für Erziehung und Ausbildung
Ritterstrasse 26
Leipzig
7010

Bewerbung zum Direktstudium: . . . . . .
Ich hatte mich im . . . dieses Jahres über meine Schule, die EOS . . ., an der . . . zum . . . studium in der Fachrichtung . . . beworben. Das Direktorat der . . . teilte mir mit, daß die Anzahl der Bewerber für die genannte Fachkombination in diesem Jahr sehr hoch ist und daß ich ein Studium der gewünschten Fächer nicht werde aufnehmen können. – Mir wird empfohlen, mich für eine andere Fachkombination zu entscheiden und eine Neubewerbung einzureichen.
Nach Mitteilung des stellv. Direktors für Berufsausbildung meiner Schule sind in . . . noch Studienplätze für . . . frei.
Da ich den lebhaften Wunsch hege, . . . zu werden, bewerbe ich mich hierdurch um Zulassung zum . . . studium in der Fachrichtung . . . und bitte, meine Bewerbungsunterlagen bei der . . . anzufordern.

Frank Schäfer

7. Entwickeln Sie ein Programm, mit dessen Hilfe der Computer Sätze auf den Bildschirm schreibt! Dabei soll jeder Satz aus 4 Teilsätzen auf zufällige Weise aufgebaut werden. Die Gruppen der Teilsätze sollen in DATA-Zeilen erfaßt sein.
   1. Teilsätze:
      "Zugegebenermassen"
      "Trotz des gegenteiligen Anscheins"
      "Unter Zuhilfenahme des Prinzips von Ursache und Wirkung"
      "Wie nun jenseits jeden Zweifels bewiesen wurde"
   2. Teilsätze
      "wird eine erfolgreiche Messung"
      "wird das Streben nach einem Nobelpreis"
      "wird uebermaessiges Bedenken der Probleme der Verwaltung"
      "wird eine konstruktive Loesung"
   3. Teilsätze:
      "nur dazu dienen, mehr Gewicht zu legen auf"
      "schluessige Informationen liefern ueber"
      "orientieren auf"
      "Begeisterung hervorrufen fuer"
   4. Teilsätze:
      "die Notwendigkeit, die Rechenmaschinen weiter auszubauen."
      "einen Entwurf, der Zusammenstoesse im spaeteren Stadium produziert."
      "den Wunsch, dass manche Wissenschaftler das Unerforschte erforschen."
      "die Zukunft der Physik in Europa."

   Ein Beispiel ist der folgende Satz:
   "Trotz des gegenteiligen Anscheins wird das Streben nach einem Nobelpreis nur dazu dienen, mehr Gewicht zu legen auf die Notwendigkeit, die Rechenmaschinen weiter auszubauen."

2.5. Arbeit mit Menüs und Unterprogrammen bei komplexen Problemen

2.5.1. Arten von Menüs

Im Programmbeispiel „DEMONSTRATION DES ZUFALLS" wurde die Dialogführung zwischen Nutzer und Programm über ein **Auswahlmenü** geführt. Die Bildschirmgestaltung zum Untermenü 3 ‚Zufallstexte' soll das noch einmal verdeutlichen:

```
Untermenue 3  'Zufallstexte'

0   zurueck ins Hauptmenue
1   Krokofantenspiel
2   Eigenschaftszuordnung
3   'sinnvolle Zufallstexte'
4   Woerterraten
5   Textgeruest

Waehlen Sie die Kennzahl K
```

⇑ Abb. 2.5/1: Untermenü 3 „Zufallstexte"

```
Neueintrag eines Datensatzes:
_____________________________

   Autoren:
     Titel:
    Verlag:
 Ersch.ort:
Ersch.jahr:
  Standort:
Stichpunkte:

Eintrag beenden u. abspeichern mit ↵
```

⇓ Abb. 2.5/2: Beispiel eines Erfassungsmenüs

Es gibt als weitere Formen der Menütechnik

- Erfassungsmenüs,
- Modifikationsmenüs,
- Befehlsmenüs und
- Ikonenmenüs.

Bei **Erfassungsmenüs** zeigt das Programm das Bild eines Formulars auf dem Bildschirm, das der Benutzer ganz oder teilweise auszufüllen hat.

Bei einem **Modifikationsmenü** zeigt das Programm ein mit Daten gefülltes Formular auf dem Bildschirm. Der Benutzer kann die Daten nach Bedarf ändern, z. B. überschreiben, ergänzen oder löschen. Eintragungen können überschrieben werden. Zum Beispiel wird das Erscheinungsjahr 1986 korrigierbar zu 1988. Ergänzungen, wie Standort und Stichpunkte, kann man eintragen.

```
Aenderung des Datensatzes Nr.  3
-------------------------------------

    Autoren: Schilling/Toepfer
      Titel: Informatik
     Verlag: VWV
  Ersch.ort: Berlin
 Ersch.jahr: 1988
    Standort:
Stichpunkte:


Bewegung des Cursors mit ← → ↑ ↓
Aenderung beenden und korrigierten
Datensatz abspeichern mit ↵
```

Abb. 2.5/3: Beispiel eines Modifikationsmenüs

Bei **Befehlsmenüs** zeigt das Programm eine oder mehrere Zeilen auf dem Bildschirm, in die eine Übersicht über mehrere Befehle eingeschrieben ist. Durch Drücken einer Taste – im allgemeinen der Großbuchstabe im Befehlswort (z. B. C für Copieren oder H für Hilfe) – kann der Teil des Programms ausgewählt werden, mit dem der Benutzer arbeiten will.

Bei **Ikonenmenüs** werden die Auswahlmöglichkeiten nicht in Textform, sondern in Bildform angeboten. Mit einem Steuerhebel bzw. mit den Cursortasten fährt der Benutzer den Cursor auf die bildliche Darstellung, die den gewünschten Programmteil charakterisiert.

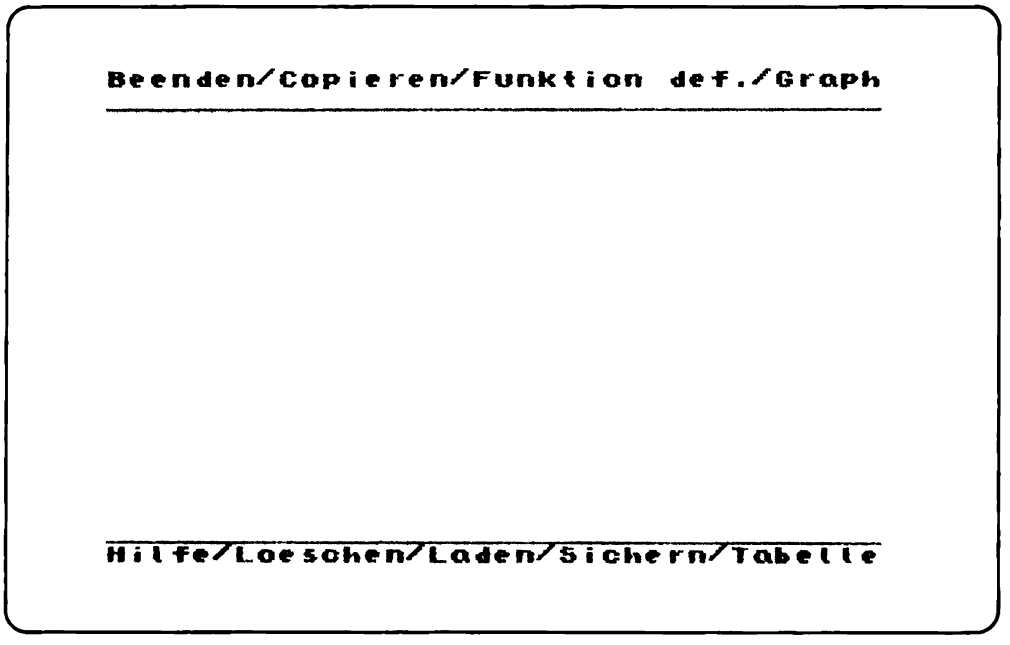


Abb. 2.5/4: Beispiel eines Befehlsmenüs

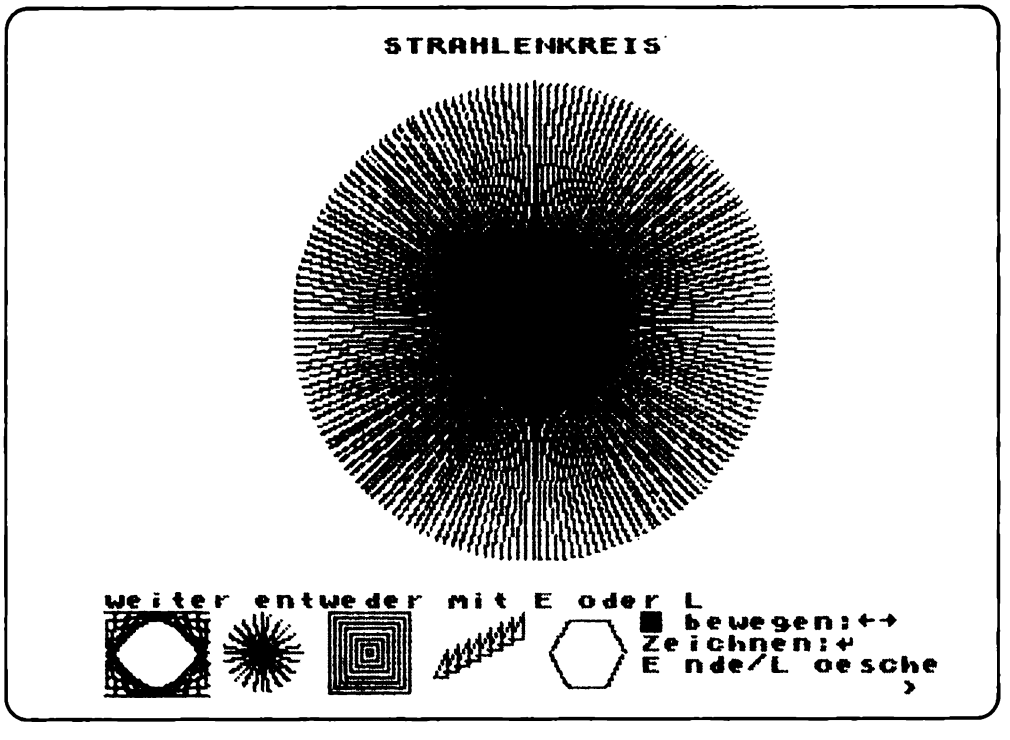


Abb. 2.5/5: Beispiel eines Ikonenmenüs

Geometrische Objekte mit Ikonen-Menüs

Es sind ein Algorithmus und ein Programm zur Darstellung von geometrischen Objekten unter Verwendung eines Ikonenmenüs zu entwickeln.

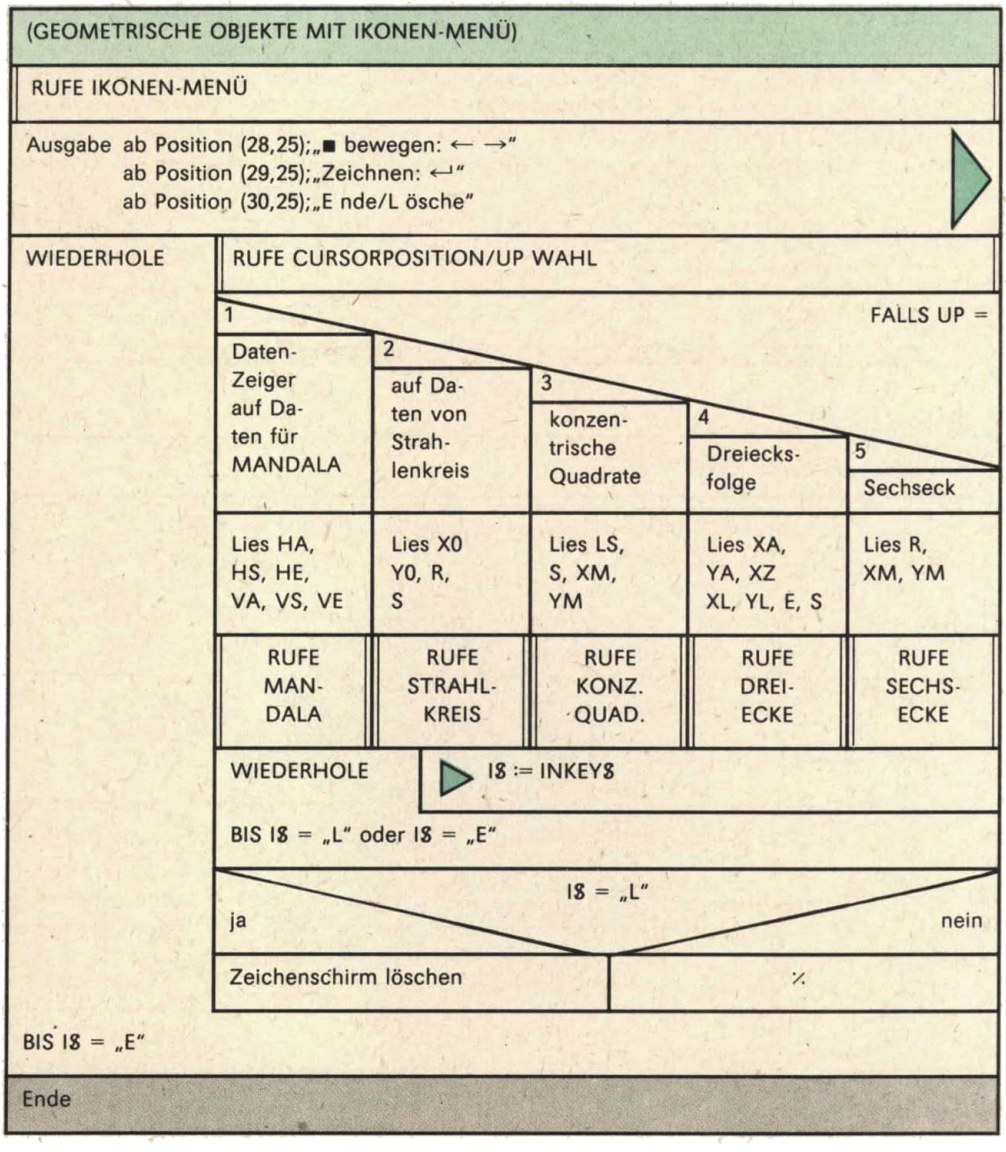
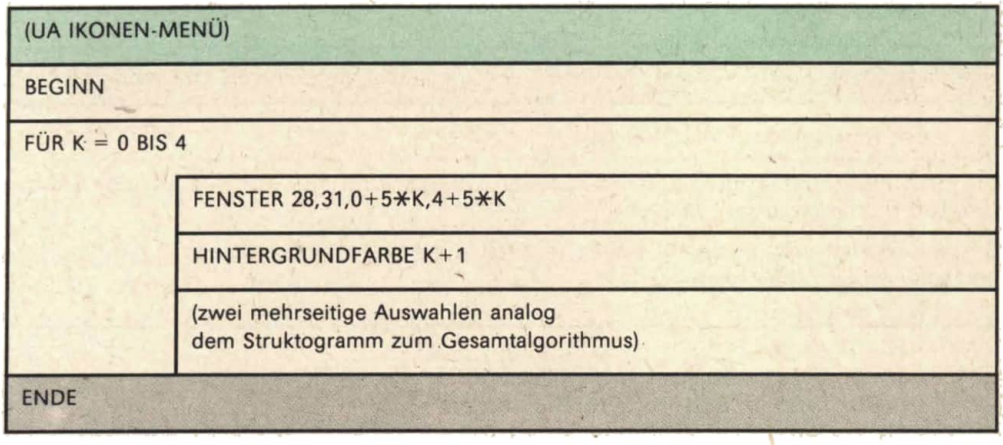


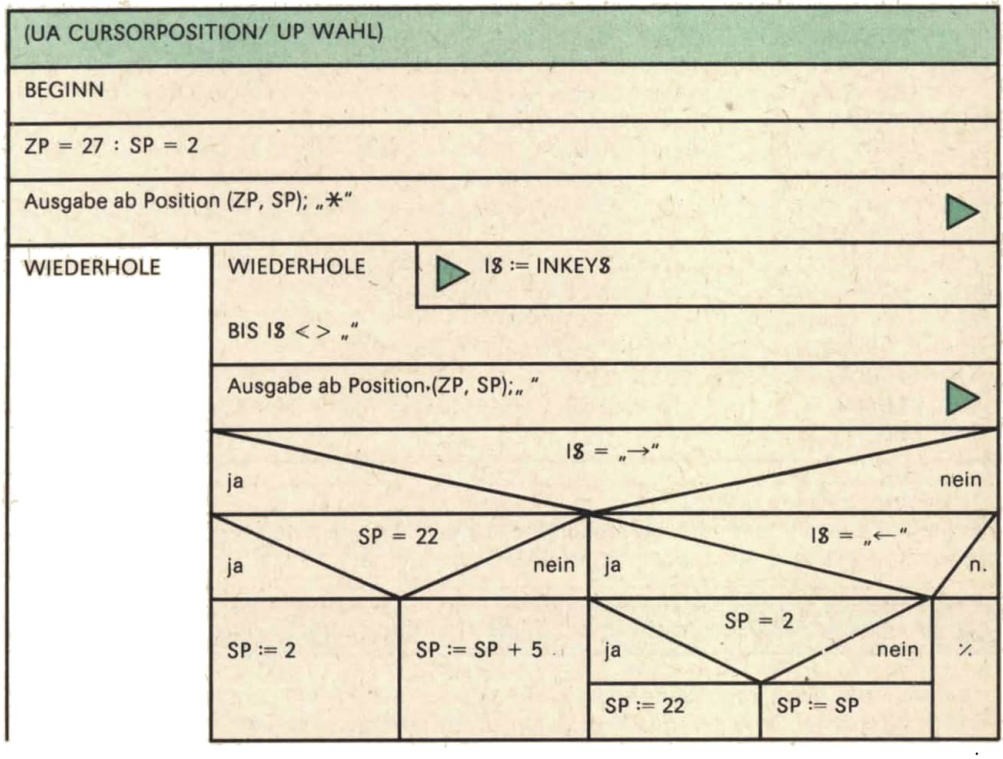
Abb. 2.5/6: Struktogramm zu „Geometrische Objekte mit Ikonenmenü"

Das Struktogramm zum IKONEN-MENUE ähnelt sehr den zwei mehrseitigen Auswahlen im Struktogramm zum Gesamtalgorithmus. Es wird der Datenzeiger allerdings auf diejeni-

gen Daten zum jeweiligen geometrischen Objekt eingestellt, die das geometrische Objekt stark verkleinert und eingeschränkt in ein kleines Fenster in die unteren 4 Zeilen zeichnen. Das wird für jedes der 5 geometrischen Objekte gemacht, so daß in den letzten 4 Bildschirmzeilen ein Überblick über alle Möglichkeiten des Programms entsteht.



In der 27. Bildschirmzeile soll ein Cursor durch die Cursortasten ← und → gesteuert werden. Aus der Cursorposition ist die Unterprogrammnummer zu bestimmen.



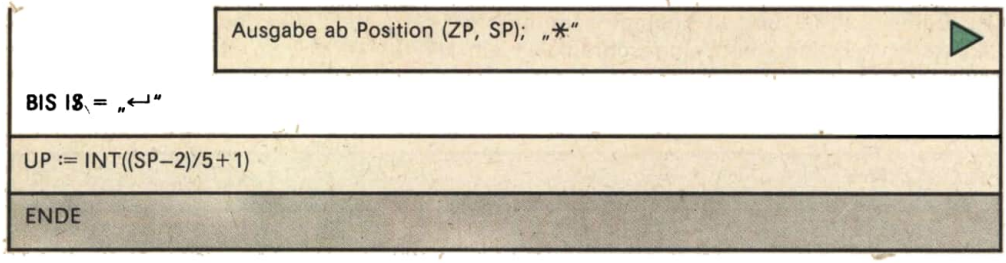


Abb. 2.5/7: Unteralgorithmus „Cursorposition"

Die Unteralgorithmen für die geometrischen Objekte sind hier nur in Form der BASIC-Unterprogramme dargestellt.

Das Hauptprogramm steuert die Aufrufe der Unterprogramme und enthält die dafür erforderlichen Entscheidungen.

```
200 WINDOW 0,31,0,39 : COLOR 0,6 : CLS
210 GOSUB 1000 : REM → IKONEN-MENUE
220 GOSUB 1400 : REM → BEFEHLS-MENUE
230 REM anfang der wiederholschleife 1
240 GOSUB 1600 : REM → CURSORPOSITION/UP WAHL
250 ON UP GOTO 260,270,280,290,300
260  RESTORE 62020:READ TZ,TS,BZ$,HA,HS,HE,VA,VS,VE:GOTO310
270  RESTORE 62040:READ TZ,TS,BZ$,XO,YO,R,S:GOTO 310
280  RESTORE 62060:READ TZ,TS,BZ$,LS,S,XM,YM:GOTO 310
290  RESTORE 62080:READ TZ,TS,BZ$,XA,YA,XZ,XL,YL,E,S:GOTO310
300  RESTORE 62100:READ TZ,TS,BZ$,R,XM,YM
310 REM ende der mehrseitigen auswahl
320 WINDOW 0,27,0,39 : COLOR 7,UP : CLS
330 ON UP GOSUB 2000,4000,3000,5000,7000
340 REM anfang innere wiederholschleife
350 PRINT COLOR 16,6;AT(27,0);"weiter entweder mit E oder L"
360   I$=INKEY$
370   IF NOT(I$="L" OR I$="E") THEN 360
380   REM anfang der einseitigen auswahl
390      IF NOT(I$="L") THEN 410
400      WINDOW 0,27,0,39 : COLOR 0,6 : CLS
410 IF NOT(I$="E") THEN 230
420 WINDOW 0,31,0,39 : COLOR 0,6 : CLS
430 END
```

Das Hauptprogramm umfaßt 24 Zeilen und paßt vollständig auf den Bildschirm. Die Zeile 330 enthält eine Variante der mehrseitigen Auswahl (berechneten Sprung zu den Unterprogrammen). Diese Anweisung hat in BASIC das Format

```
ON ausdruck GOSUB zeilennummer [,zeilennummer]...
```

Dabei muß der Wert von ‚ausdruck' nichtnegativ sein. Zur Unterprogrammadressenwahl wird der ganzzahlige Anteil des Werts von ‚ausdruck' benutzt. Ist dieser gleich 0 oder

größer als die Anzahl der angegebenen Zeilennummern, wird das Programm mit der auf die ON...GOSUB-Anweisung folgenden Anweisung fortgesetzt.
Im Befehls-Menü sind Hinweise zur Fortsetzung des Programms zusammengefaßt:

```
1400 REM === BEFEHLS-MENUE ====================
1410 WINDOW 28,31,25,39 : COLOR 0,6 : CLS
1420 PRINT AT(28,25);"■  bewegen:";CHR$(136);CHR$(137)
1430 PRINT AT(29,25);"Zeichnen:";CHR$(141)
1440 PRINT AT(30,25);"E nde/L oesche"
1490 RETURN
```

Das Ikonen-Menü entspricht dem Struktogramm und gleicht den Hauptprogrammzeilen 250 bis 330, die nur in den Datenzeigeradressen auf 60020 bis 60100 verändert sind, und das Ganze ist in eine FOR...NEXT-Schleife eingebaut. Die Anweisungsnummern gehen von 1000 bis 1290. Auch dieses Unterprogramm umfaßt nur so viele Programmzeilen, daß sie auf eine Bildschirmseite passen. Das gilt auch für das Unterprogramm CURSOR-POSITION/UP WAHL:

```
1600 REM ==== CURSORPOSITION/ UP WAHL ===
1610 REM Ausgangsvariablen:
1620 REM  ZP,SP - Cursorposition
1630 REM  UP    - UP-Nummer
1640 REM 'lokale' Variable:
1650 REM  I$    - Eingabezeichenkette
1660 REM ------------------------------------
1700 WINDOW 0,27,0,39:COLOR 0,6:ZP=27:SP=2:PRINT COLOR 16,6;
     AT(ZP,SP);"*"
1710 REM anfang der wiederholschleife
1720    I$=INKEY$:IF I$="" THEN 1720
1730    PRINT AT(ZP,SP);" "
1740    IF NOT(I$=CHR$(9)) THEN 1770
1750      IF SP=22 THEN SP=2 : ELSE SP=SP+5
1760    GOTO 1790
1770      IF I$<>CHR$(8) THEN 1790
1780         IF SP=2 THEN SP=22 : ELSE SP=SP-5
1790    REM ende der zweiseitigen auswahl
1800    PRINT COLOR 16,6;AT(ZP,SP);"*"
1810 IF NOT(I$=CHR$(13)) THEN 1710
1820 REM ende der wiederholschleife
1830 UP=INT((SP-2)/5+1)
1890 RETURN
```

Die Unterprogramme zu den einzelnen geometrischen Objekten sind ebenfalls auf das Format von einer Seite geschrieben:

```
2000 REM ========== MANDALA ==============
2010 REM Eingangsvariablen:
2020 REM  BZ$,TZ,TS - Bezeichnung mit Anfangsposition
2030 REM  HA,HS,HE  - horizontale Parameter
2040 REM  VA,VS,VE  - vertikale Parameter
```

```
2050 REM ------------------------------------
2200 LOCATE TZ,TS : PRINT BZ$
2210 N=INT((HE-HA)/HS)
2220 FOR I=0 TO N
2230    X1=HA+HS*I
2240    X2=HE-HS*I
2250    Y1=VA+VS*I
2260    Y2=VE-VS*I
2270    LINE X1,VA,HE,Y1,7
2280    LINE X1,VE,HE,Y2,7
2290    LINE X2,VA,HA,Y1,7
2300    LINE X2,VE,HA,Y2,7
2310 NEXT I
2990 RETURN
```

Für die weiteren Unterprogramme sind hier nur noch die Überschriften und die Berechnungsteile angegeben:

```
3000 REM ===== KONZENTRISCHE QUADRATE =====
....
3200 PRINT AT(TZ,TS);BZ$
3210 FOR I=1 TO LS STEP S
3220    X1=XM-I : Y1=YM-I
3230    X2=XM+I : Y2=YM+I
3240    LINE X1,Y1,X2,Y1,7
3250    LINE X2,Y1,X2,Y2,7
3260    LINE X2,Y2,X1,Y2,7
3270    LINE X1,Y2,X1,Y1,7
3280 NEXT I
3990 RETURN

4000 REM =====  STRAHLENKREIS ==========
....
4200 PRINT AT(TZ,TS);BZ$
4210 FOR T=0 TO 360 STEP S
4220    X=X0+R*COS(PI*T/180)
4230    Y=Y0+R*SIN(PI*T/180)
4240    LINE X0,Y0,X,Y,7
4250 NEXT T
4990 RETURN

5000 REM ===== DREIECKSFOLGE ==========
....
5200 PRINT AT(TZ,TS);BZ$
5210 FOR T=0 TO E-XA STEP S
5220    X1=XA+XZ*T : Y1=YA+T
5230    LINE X1,Y1,X1+XL,Y1,7
5240    LINE X1+XL,Y1,X1+XL,Y1+YL,7
5250    LINE X1+XL,Y1+YL,X1,Y1,7
5260 NEXT T
5990 RETURN
```

Zum Unterprogramm SECHSECK werden noch einige Erläuterungen gegeben, die auch für viele andere Lösungen von Bedeutung sind!
Es ist ein regelmäßiges 6-Eck auf dem Bildschirm zu erzeugen. Wie aus Abbildung 2.5/8 zu entnehmen ist, kann das 6-Eck gezeichnet werden, wenn die Eckpunkte P1, P2, ..., P6 bekannt sind. Jeweils zwei benachbarte Eckpunkte bestimmen eindeutig eine Seite des regelmäßigen 6-Ecks. Verbindungsstrecken zwischen zwei Punkten sind mit dem LINE-Befehl sofort zeichenbar. Die Eckpunktkoordinaten im x-y-Koordinatensystem ergeben sich zu $x_n = r * \cos(n * \alpha)$ und $y_n = r * \sin(n * \alpha)$ mit $n = 1, 2, \ldots, 6$,

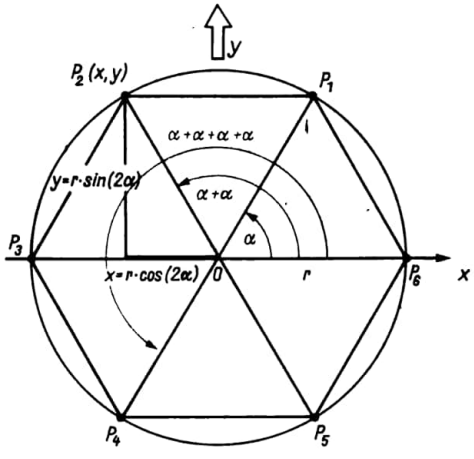


Abb. 2.5/8: Sechseck in Anwenderkoordinaten

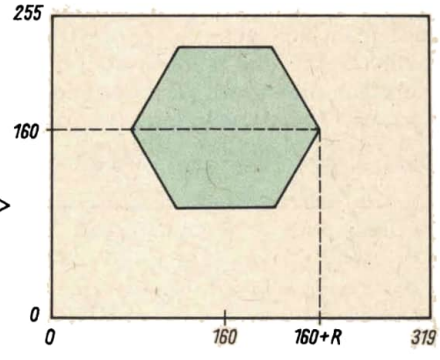


Abb. 2.5/9:
Sechseck auf dem Bildschirm

wobei α der 6. Teil eines Vollwinkels im Bogenmaß ist. Um das 6-Eck auf dem Bildschirm gut erkennbar darzustellen, werden den Koordinaten entsprechende Bildschirmkoordinaten im Punktraster zugeordnet. Alle x-Koordinaten werden um 160 Punkte nach rechts und alle y-Koordinaten um 60 Punkte nach oben verschoben (Abb. 2.5/9). Damit ergibt sich das folgende Programm:

```
7000 REM ===== SECHSECK ==========
7200 PRINT AT(TZ,TS);BZ$
7210 A0=2*PI/6
7220 XA=R+XM : YA=YM
7230 FOR A=A0 TO 2*PI STEP A0
7240    X=R*COS(A) : Y=R*SIN(A)
7250    LINE XA,YA,X+XM,Y+YM,7
7260    XA=X+XM : YA=Y+YM
7270 NEXT A
7990 RETURN
```

Ein Test zeigt, daß nur 5 Seiten gezeichnet werden. Vermutlich führt die wiederholte Summation des Winkels A0 im 6. Schritt über den Endparameter 2*PI hinaus. Mit

```
7235 PRINT "A=";A
7280 PRINT "A=";A
```

soll die Vermutung überprüft werden. Die Ausgabefolge

```
A = 1.0472
A = 2.0944
A = 3.14159
A = 4.18879
A = 5.23599
A = 6.28319
```

klärt noch nichts, da der Computer für 2*PI auch die Zahl 6.28319 ausgibt. Das letzte A stimmt im Rahmen der Genauigkeit von 6 Stellen mit dem Endparameter 2*PI überein. Läßt man sich aber noch die Differenz A–2*PI ausgeben, so bestätigt die Ausgabe 4.76837E–07, daß A im 6. Schritt größer als der Endparameter 2*PI wird. Wie bereits mehrfach praktiziert, wird der Endparameter um die Hälfte der Schrittweite A0/2 vergrößert, und das so geänderte Programm leistet das Verlangte.

Das gesamte Programm wird durch die folgende Datenliste abgeschlossen:

```
60000 REM ===== DATEN FUER DIE VERKLEINERUNGEN =====
60010 REM ------------ FUER MANDALA ---------------
60020 DATA 1,2,"",0,5,39,0,4,31
60030 REM --------- FUER STRAHLENKREIS ------------
60040 DATA 1,2,"",19,16,15,15
60050 REM ----- FUER KONZENTRISCHE QUADRATE --------
60060 DATA 1,2,"",15,2,99,16
60070 REM ---------- FUER DREIECKSFOLGE ------------
60080 DATA 1,2,"",123,7,2,5,10,138,2
60090 REM ------------- FUER SECHSECK -------------
60100 DATA 1,2,"",14,180,16
62000 REM ==== DATEN FUER GEOMETRISCHE OBJEKTE =====
62010 REM ------------ FUER MANDALA ---------------
62020 DATA 14,16,"MANDALA",20,14,299,40,10,239
62030 REM --------- FUER STRAHLENKREIS ------------
62040 DATA 1,13,"STRAHLENKREIS",160,140,90,2
62050 REM ---- FUER KONZENTRISCHE QUADRATE ---------
62060 DATA 1,9,"KONZENTRISCHE QUADRATE",90,4,160,150
62070 REM ----------- FUER DREIECKSFOLGE -----------
62080 DATA 1,13,"DREIECKSFOLGE",40,60,2,50,100,120,5
62090 REM ------------- FUER SECHSECK -------------
62100 DATA 1,16,"SECHSECK",80,160,160
65000 REM PHYSISCHES ENDE DES PROGRAMMS
```

2.6. Regeln zur Gestaltung von BASIC-Programmen

▶ Problemlösen mit dem Computer erfordert sprachliche Exaktheit in allen Stufen des Problemlösungsprozesses.

So kann die Lösung nur dann wie gewünscht ausfallen, wenn die Aufgabenstellung exakt formuliert ist. Beim Entwurf des Lösungsplanes ist eine exakte Sprache erforderlich. Bei der Erarbeitung der logischen Zusammenhänge aller Lösungskomponenten sind Verfahren wie die *schrittweise Verfeinerung* oder die *schrittweise Verallgemeinerung* hilfreich.
Ein Problem ist im Prinzip gelöst, wenn ein Algorithmus gefunden ist, der von einem Computer abgearbeitet werden kann, und aus den Eingabedaten auf eindeutige Weise die korrekten Ausgabedaten liefert.

▶ Die Algorithmierung läuft dann auf hohem Niveau ab, wenn sie den Grundprinzipien der strukturierten Programmierung genügt.

Jeder Algorithmus läßt sich aus vier Grundelementen aufbauen, die in der Übersicht (↗ innere Umschlagseiten) in den Darstellungsformen verbal formalisierte Notation, Sinnbilddarstellung, Leitliniendarstellung, Struktogramm und in BASIC der Kleincomputer KC 85/1 bis /3 zusammengestellt sind. Für die Darstellung gut strukturierter Algorithmen eignen sich Struktogramme besonders.

▶ Ein Struktogramm ist aus den Strukturelementen Folge, Auswahl, Wiederholung und Unteralgorithmusaufruf so aufzubauen, daß die Strukturelemente
entweder so aneinandergefügt werden, daß die gesamte Unterkante (Ausgang) eines Strukturelements mit der gesamten Oberkante (Eingang) eines anderen Strukturelements zusammenfällt,
oder indem in ein rechteckiges Feld eines Strukturelementes ein anderes Strukturelement (oder ein ganzes Struktogramm) hineingesetzt wird.

Bei der Algorithmierung sollte man bestrebt sein, stets nach leistungsfähigeren Algorithmen zu suchen. Das hat auch unter Berücksichtigung der wechselseitigen Beeinflussung zwischen Daten und Algorithmen zu erfolgen. Bei den Daten sind Datentypen (Zahlen und Zeichenketten) zu unterscheiden, die das Anwenden unterschiedlicher Sprachelemente, wie z. B. Standardfunktionen, ermöglichen.
Ausgehend vom erstellten Algorithmus ist die Strukturierung auch auf das entsprechende Programm zu übertragen. Dabei sind die Umsetzungen der algorithmischen Elemente Folge, Auswahl, Wiederholung und Unteralgorithmusaufruf in die gewählte Programmiersprache zu realisieren. Wie bereits gesagt, hat jedes Programm einer Grundstruktur zu genügen. Für BASIC-Programme sollte man sich an folgendem orientieren:

```
 a1 REM programmname ==========================
[a2        DIM bez(n[,m]...) ]
[a3        funktionsunterprogramme ]
an        folge
ae END
[au        unterprogramme ]
[az        DATA c1 [,c2] ... ]
```

Die Unterprogramme sollten nach dem END des Hauptprogramms angeordnet werden, da dadurch ein unbeabsichtigtes ‚Hineinlaufen' der Abarbeitung in ein Unterprogramm vermieden wird. Die Datenliste sollte auf festen Anweisungsnummernbereichen und am Ende des Gesamtprogramms liegen, weil dadurch ein unkomplizierter Datenaustausch gewährleistet wird.

Programme werden übersichtlich, verständlich und leicht korrigierbar, wenn auch die formale Programmgliederung nach festen Regeln erfolgt. Bewährt hat sich die folgende Programmgliederung (für umfangreiche Programme):

```
10  REM  Kurztitel des Programms
20  REM  Autorennamen, Schule, Schulort, Klassenstufe
30  REM  Datum der Fertigstellung des Programms
40  REM  Zusatzinformationen, z.B. Quelle der Programmidee,
..  ...  letzte Modifikation von Daten
```

Die Programmabarbeitung sollte mit einem Titelbild beginnen! Dies sollte einige der genannten Informationen enthalten und Hinweise auf Farb- und Tonausgabe geben, falls solche vorkommen. Das Titelbild sollte ästhetischen Ansprüchen genügen.

```
100 REM Hauptprogramm  ============================
... ...
990 END : REM logisches Ende des Programms
```

Das Hauptprogramm ist eine Folge von Prüfungen und Verzweigungen zu den Unterprogrammen. Es ist damit ein *Steuerprogramm*.

```
1000 REM Name des Unterprogramms 1 ==================
.... ...
1990 RETURN : REM Ende des Unterprogramms 1
```

Unterprogramme beginnen mit vollen Tausendern und enden i. a. mit genau einem RETURN.

```
2000 REM Name des Unterprogramms 2 ====================
.... ...
.... ...
2990 RETURN : REM Ende des Unterprogramms 2
.... ...
.... ...
50000 REM Daten
.... ...
65000 REM Physisches Ende des Programms
```

DATA-Zeilen beginnen grundsätzlich mit der Zeilennummer 50000 und enden mit 65000. Das erleichtert den Austausch von Datensätzen.

Die logisch zusammengehörigen Teilalgorithmen sind so zu gestalten, daß sie möglichst mit einem Blick überschaubar sind. Das ist im allgemeinen erreicht, wenn sie auf einer

Druckseite oder einer Bildschirmseite Platz finden. Jedes Unterprogramm hat im allgemeinen genau einen Eingang und genau einen Ausgang.

Für die gute Lesbarkeit der Programme sind ausreichend REM-Anweisungen einzusetzen. Aber auch Unterprogrammaufrufe und Sprünge sollten kommentiert werden, z. B.

```
340  GOSUB 4000 : REM → eingabe und zulaessigkeitspruefung
3380 GOTO 3470  : REM → ende der solange-schleife 1
```

Für Unterprogramme sollte der folgende Aufbau verwendet werden:

```
3000  REM Unterprogramm 3: unterprogrammname ===========
3010  REM zweck des unterprogramms
3020  REM liste der eingangsvariablen
....  REM liste der ausgangsvariablen
....  REM liste der 'lokalen ' variablen
....  REM ----------------
           folge von anweisungen
....
3990 RETURN
```

Die Variablenlisten sollten die Bezeichnung und eine Bedeutungsangabe umfassen. Die Fehlersuche wird begünstigt, wenn die Variablen nach Datentypen bzw. Datenstrukturen getrennt aufgeführt werden.

Als **Regeln** gelten:

- die Folge der Anweisungen grafisch durch Einrückungen zu gliedern,
- zwischen Schlüsselwörtern jeweils ein Leerzeichen zu lassen und
- auf jede Programmzeile nur eine Anweisung zu schreiben.

Allerdings ist es empfehlenswert, logisch sehr eng zusammenhängende Anweisungen auf eine Zeile – jeweils durch „:" getrennt – zu schreiben, solange die Programmzeilenlänge ausreicht. Das trifft z. B. für eine Parameterinitialisierung I=0 : S=0 : P=1 oder für den Dreieckstausch von Daten H=A(J) : A(J)=A(I) : A(J)=H zu.

So gestaltete Unterprogramme sind relativ leicht austauschbar. DELETE 3000,3990 schafft Platz für das neue Unterprogramm, das mit der Anweisungszeile 3000 beginnt und mit der Anweisungszeile 3990 endet.

Anweisungen eines Programmteils (z. B. eines Unterprogramms) werden mit

```
LOAD#1 "name"
```

in einen Programmtext im Hauptspeicher entsprechend ihrer Anweisungsnummern einsortiert. Allerdings muß ein solcher Programmteil mit

```
LIST#1 "name"
```

auf Magnetband gespeichert worden sein.

Ein Programm mit Dialogteilen muß dem Nutzer eindeutige Anweisungen anbieten! Diese sollten in verständlichem Deutsch und möglichst jedes Mal an der gleichen Stelle des Bildschirms erfolgen.

Eine oft genutzte Dialogform ist die Menütechnik. Man unterscheidet

- Auswahlmenüs,
- Erfassungsmenüs,
- Modifikationsmenüs,
- Befehlsmenüs und
- Ikonenmenüs.

Die Auswahlmenüs sollten aus 3 bis 7 Wahlmöglichkeiten bestehen. Es ist günstig, die Ziffer 0 grundsätzlich für den Übergang in das Hauptprogramm oder an das Ende des Programms festzulegen.

Wenn Wahlmöglichkeiten durch jeweils genau eine Ziffer bzw. durch genau einen Buchstaben realisiert sind, wird die nutzerfreundliche INKEY$-Abfragetechnik ermöglicht.

Bei den Wahlmöglichkeiten sollte zwischen ‚ungefährlichen' und ‚gefährlichen' unterschieden werden. Ungefährliche Programmfortsetzungen können sofort ausgeführt werden. Bei gefährlichen Programmfortsetzungen sollte der Programmnutzer die Möglichkeit einer Korrektur seiner Eingabe bekommen. Zu den ‚gefährlichen' Programmfortsetzungen ist z. B. das Löschen von Programmteilen zu zählen. Beispielsweise kann der Nutzer durch die Frage „Wollen Sie das wirklich löschen?" vor Fehlhandlungen gewarnt werden.

Ein nutzerfreundliches Programm zeichnet sich neben seiner guten Strukturierung auch durch eine gute Bildschirmgestaltung und unmißverständliche Dialogführung aus.

Bei der **Bildschirmgestaltung** sollte auf eine ästhetisch ansprechende Flächenaufteilung und Farbgestaltung geachtet werden. Aktuelle Bedienhinweise können durch Blinken unterstützt werden (nicht zu häufig einsetzen). Fehlbedienungen sollten akustisch signalisiert werden und dürfen nicht zum Programmabsturz führen. Wenn es möglich ist, sind bei Eingaben Zulässigkeitsprüfungen ins Programm einzufügen.

Ein Programm, das nach den oben zusammengestellten Regeln erarbeitet wurde, hat neben seinen Vorteilen aber auch Nachteile:
- Es kostet viel Speicherplatz.
- Seine Abarbeitungsgeschwindigkeit ist nicht besonders hoch.

Deshalb ist es günstig, von einem ausgereiften Programm eine Zweitfassung herzustellen, die von allen für die Programmabarbeitung unwichtigen Programmteilen befreit ist. Bei ihr können beispielsweise alle REM-Anweisungen und die Leerzeichen entfallen, die für die graphische Gliederung eingesetzt wurden.

● Aufgaben

1. Entwickeln Sie ein Programm, das das Schreiben einer Barverkaufsquittung nach dem Muster auf Seite 151 oben realisiert!
   Dabei ist zu berücksichtigen, daß im Falle einer Bezahlung der Ware mit Scheck in der linken unteren Ecke der Name des Käufers, seine Adresse, die Kontonummer und die Personalausweisnummer anstatt des Barzahlungsvermerks steht.

2. Schreiben Sie die Programme 1, 2, 4 des Untermenüs 3 „Zufallstexte" zum Programmbeispiel „DEMONSTRATION DES ZUFALLS"!

VEB BAUSTOFFVERSORGUNG BERLIN 24. 07. 85
FACHGESCHAEFT IDUNASTRASSE B*240785071 1

BARVERKAUFSQUITTUNG

ART.	BEZEICHNUNG	MENGE	ME	EVP	BETRAG
0101	ZEMENT ZZ 1/25	2	SAK	6,00	12,00
2103	HEISSKLEBER 11 KG	2	STK	8,80	17,60
9002	KEMPAFIL	10,00	M	0,95	9,50
				SUMME:	39,10

BAR: 50,10 RUECKZ.: 11,00 VERKAEUFER

Abb. 2.6/1: Barverkaufsquittung

Dabei soll beim Krokofantenspiel davon ausgegangen werden, daß neue Tiere die Natur bevölkern, die auf zufällige Weise kombiniert sind aus einem Namenvorderteil aus einem Feld von Vordernamen und aus einem Namenhinterteil aus dem Feld Hinternamen, wie z. B. aus
Ele fant und Kroko dil
ein Eledil oder ein Krokofant kurioser Weise entstehen können. Mutet das wie ein Spiel an, so gibt es durchaus seriöse Anwendungen solcher Zufallskombinationen, wenn man an Kombinationen chemischer Stoffe denkt. Dort sind allerdings die Kombinierfähigkeitsbedingungen wesentlich komplizierter als hier, wo nur die Zugehörigkeit zum Vorder- und Hinternamenfeld ausschlaggebend ist.

Beim Programm Eigenschaftenzuordnung soll der Zufall einem Namen aus einem Namenfeld eine Eigenschaft aus einem Eigenschaftsfeld zuordnen. Dabei sollen sowohl Namenfeld wie auch Eigenschaftsfeld auf Magnetband speicherbar und von dort auch wieder ladbar sein.

Beim Wörterraten ‚denkt' sich ein Spieler – der Computer – ein Wort aus. Der andere nennt Buchstaben. Kommt ein genannter Buchstabe im Wort vor, so wird er überall im Wort angezeigt. Kommt ein eingegebener Buchstabe im Wort nicht vor, so wird der Fehlversuchszähler um 1 erhöht. Schafft der Spieler es, das gesuchte Wort bei höchstens 10 Fehlversuchen zu erraten, so ist er Sieger, andernfalls gewinnt der Computer.
Folgende Teilaufgaben sind zu programmieren:
- (zufällige) Auswahl eines Wortes aus dem Feld der Ratewörter,
- Erzeugung des Formats dieses Wortes, damit der Spieler weiß, wieviel Buchstaben im Wort sind;
- Prüfen der Eingabe auf Zulässigkeit (Eingabe genau eines Buchstabens)
- Test, ob der eingegebene Buchstabe im Wort ist und Einschreiben des Buchstabens an alle Stellen seines Vorkommens.
- Kommt der Buchstabe im Wort nicht vor, so ist der Fehlversuchszähler um 1 zu erhöhen und ggf. die Fehlversuchsanzeige auszugeben;
- Bewertung des Spiels und
- Abfrage, ob ein neues Spiel gewünscht wird.

3. Entwickeln Sie ein Programm, das über ein Ikonenmenü unterschiedliche Arten von graphischen Darstellungen von Häufigkeitsverteilungen mit einer Zufallsgröße anbie-

tet! Das Ikonenmenü soll im unteren Bildschirmbereich angegeben werden und der in [5] S. 46 angegebenen Übersicht entsprechen. Im rechten Bildschirmbereich sind Dateneingabemöglichkeiten für die Anzahl k von Daten, für die Daten x_k und die dazugehörigen Häufigkeiten h_k zu schaffen.

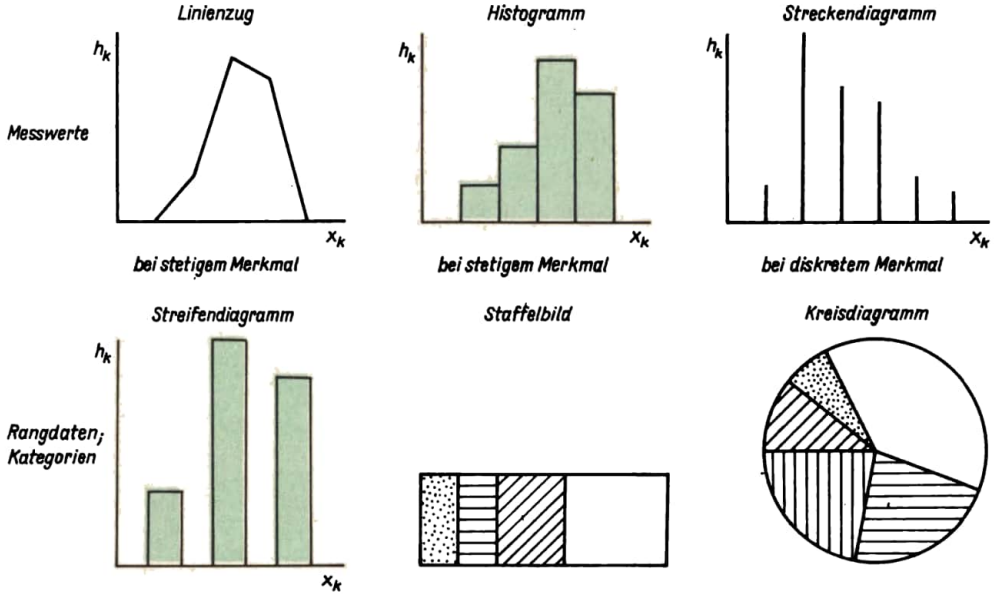


Abb. 2.6/2: Darstellungsmöglichkeiten von Häufigkeitsverteilungen

4. Entwickeln Sie zum Programm ‚Demonstration des Zufalls' die drei Programme des Untermenüs ‚6 Zufallsbewegungen'!

   0 zurueck ins Hauptmenue
   1 Fliegende Objekte
   2 Rennstrecke
   3 Galtonbrett

   Bei den ‚Fliegenden Objekten' sollen sich zufällig aus der Menge {*, 0, #, ., >, <} ausgewählte Grafikzeichen in zufälliger Weise bzgl. der Richtungen

   von links nach rechts,
   von rechts nach links,
   von oben nach unten,
   von unten nach oben,
   von links oben nach rechts unten und
   von rechts unten nach links oben

   über den Bildschirm bewegen.

   Die Rennstreckenbegrenzung sei durch Sterne * links und rechts markiert. Diese sollen sich von Zeile zu Zeile unabhängig voneinander jeweils ausgehend vom Wert der Funktion RND(1) um eine Kursorposition nach links oder nach rechts bewegen können oder an der Position verbleiben. Sicherzustellen ist dabei, daß einerseits ein Mindestabstand zwischen den beiden Begrenzungen nicht unterschritten wird und andererseits die linke bzw. rechte Bildschirmbegrenzung nicht überschritten wird. Eine so gestaltete ‚Rennstrecke' kann man für Reaktionsspiele nutzen.

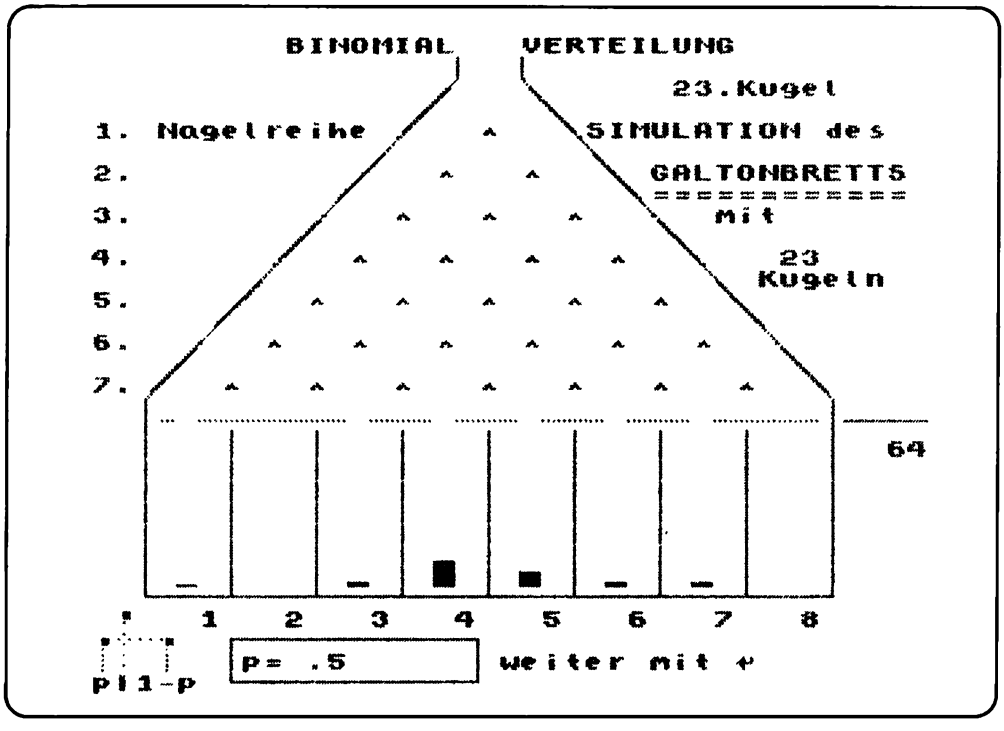
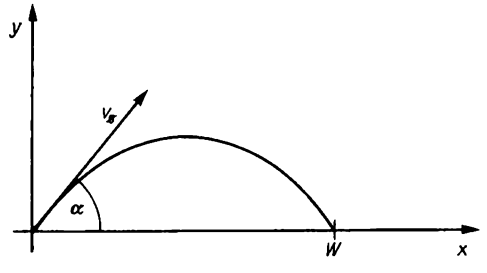


Abb. 2.6/3: Hardcopy vom Programm „Galtonbrett"

Entsprechend dem Hardcopy in Abbildung 2.6/3 ist der Kugellauf am Galtonbrett (↗ auch [6] S. 660) zu simulieren!
Der Nutzer des Simulationsprogramms soll neben der Anzahl der Kugeln die Zahl p mit $0 <= p < 1$ wählen können! Benutzen Sie für die Programme zum Untermenü 6 die Anweisungsnummern 12000 bis 14990!

5. Entwickeln Sie ein Programm zur Simulation des Zielwerfens mit einem Ball! Einzugeben sind:
   – die vorgegebene Entfernung eines Ziels in Metern,
   – die Anzahl der möglichen Würfe,
   – die geschätzte Anfangsgeschwindigkeit in m/s und
   – der Abwurfwinkel in Grad.

   Der Flug des Balls ist graphisch darzustellen. Die berechnete Wurfweite ist mit der vorgegebenen zu vergleichen. Ist die Abweichung kleiner als 10%, dann soll der Text „Ziel getroffen!" ausgegeben werden. Sonst erfolgt ein Hinweis darauf, daß die berechnete Wurfweite zu klein oder zu groß ist. Hat der Wurf das Ziel nicht mit der vorgegebenen Anzahl von Würfen getroffen, dann soll eine entsprechende Ausgabe erfolgen.



Die Formel zur Berechnung der Wurfweite des schrägen Wurfs ohne Berücksichtigung des Luftwiderstandes heißt:

$$W = \frac{v_0^2 * \sin(2 * \alpha)}{g},$$

wobei $g = 9{,}81\ m/s^2$ sei.

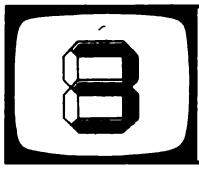
Die Bahn des Balls, der unter einem Winkel mit einer Anfangsgeschwindigkeit v_0 abgeworfen wurde, ergibt sich aus

$$y = y(x) = x * \tan(\alpha) - \left(\frac{g}{2v_0^2 \cos^2 \alpha}\right) x^2$$

Beachten Sie, daß der Computer Winkel im Bogenmaß bearbeitet! Vollziehen Sie die Umrechnung über eine benutzerdefinierte Funktion!

6. Entwickeln Sie einen Algorithmus und ein Programm
   - für die Eingabe einer Nukleotidfolge (Folge von A, C, G, T) einer DNS (Desoxyribonukleinsäure) bis zum vereinbarten Schlußzeichen über Tastatur,
   - für die Eingabe einer Nukleotidfolge von Magnetbandkassette, die vorher auf Kassette gespeichert wurde,
   - für die zeilenweise Ausgabe der Nukleotidfolge auf dem Bildschirm oder Drucker,
   - für die Korrektur und Abspeicherung auf Magnetbandkassette der eingegebenen oder bearbeiteten Nukleotidfolge und
   - für die Analyse der Nukleotidfolge, zum Beispiel in bezug auf die Auftretungshäufigkeit bestimmter Bestandteile!

   Beachten Sie, daß die DNS einfachster Organismen etwa 5000 bis 6000 Nukleotide umfaßt!



Die gesellschaftliche Bedeutung der Informatik

Rechnen ist das Band der Natur,
das uns im Forschen nach Wahrheit vor Irrtümern bewahrt.
(Johann Heinrich Pestalozzi)

Die Informatik ist noch eine junge Wissenschaft. Ihre heutige Bedeutung für die gesellschaftliche Entwicklung der DDR kann man aus folgendem Zitat erkennen, das dem Schlußwort des Genossen Honecker vom 3. Plenum des ZK der SED im November 1986 entnommen wurde.

„Mehr und mehr bestimmen, wie wir alle wissen, die Mikroelektronik, die moderne Rechentechnik, die rechnergestützte Konstruktion, Projektierung und Steuerung der Produktion das Leistungsvermögen einer modernen Volkswirtschaft. In enger Wechselwirkung damit breiten sich in unserer Volkswirtschaft auch solche Schlüsseltechnologien aus, wie flexible automatische Fertigungssysteme, neue Bearbeitungsverfahren und Werkstoffe, die Biotechnologie, die Kernenergie, die Lasertechnik.
Das sind gewaltige Herausforderungen und zugleich Chancen, die Produktion rasch zu erneuern, ihre Qualität zu erhöhen und den Aufwand in einem Maße zu senken, wie das bisher nicht möglich war. Darauf konzentrieren wir erfolgreich unsere Kräfte und stellen in Rechnung, daß sich international die Produktivkräfte zunehmend rascher entwickeln. Wir haben den Wettlauf mit der Zeit zu bestehen und dabei an wichtigen Punkten Vorsprung zu erzielen, um hohe ökonomische und soziale Ergebnisse zu erreichen."

Für die Einschätzung der Wirkung einer wissenschaftlichen Entwicklung ist es häufig wichtig, ihre Geschichte – zumindest in groben Zügen – zu kennen.

3.1. Die Entwicklung der Informationsverarbeitung

Die Entwicklung der informationsverarbeitenden Hilfsmittel und Maschinen kann man sehr gut mit der Entwicklung der Kraftmaschinen vergleichen. So wie der menschliche Erfindungsgeist mit den Kraftmaschinen (Wasserräder, Dampfmaschinen, Elektromotoren u. a.) die physischen Belastungen im Produktionsprozeß schrittweise verringerte, so werden die informationsverarbeitenden Maschinen den Menschen immer stärker bei geistigen Routinearbeiten entlasten. Diese Entwicklung begann in der Antike mit der Entwicklung des ersten mechanischen Rechenhilfsmittels, dem Abacus, und sie ist mit den heutigen Elektronenrechnern selbstverständlich nicht abgeschlossen. Die folgende Tabelle gibt einen Überblick über einige Meilensteine auf diesem Weg.

um 500 v. u. Z.	ABACUS als mechanisches Rechenhilfsmittel
um 325 v. u. Z.	Euklid konzipiert erstmals ein „universelles algorithmisches System" mit der Euklidischen Geometrie.

1641	Blaise Pascal konstruiert zur Addition und Subtraktion maximal sechsstelliger Zahlen eine Rechenmaschine, die sein Vater bei Steuerberechnungen nutzt.
um 1645	Wingate und Partridge erfinden den Rechenstab.
1660–1680	Gottfried Wilhelm Leibniz erfindet mit Staffelwalze und Zählwerkschlitten Elemente mechanischer Rechenmaschinen, die bis Mitte unseres Jahrhunderts verwendet werden. Leibniz skizzierte eine binär arbeitende Rechenmaschine.
1804	Josef Jacquard konstruiert einen automatischen Webstuhl mit Steuerung der Mustererstellung über genormte Lochstreifen.
um 1835	Charles Babbage entwirft die „Analytical Engine", die erste Konzeption einer programmgesteuerten Rechenanlage mit den drei noch heute anzutreffenden Komponenten – Steuerwerk – Rechenwerk – Speicher. Erst 40 Jahre später gelingt es, eine Vorstufe dieser Rechenanlage zu produzieren, die u.a. zur Berechnung von Sterndaten in Observatorien verwendet wurde.
1884	Hermann Hollerith erfindet eine elektromechanisch arbeitende Zähl- und Sortiermaschine, die 1890 erfolgreich zur Auswertung der Volkszählungsdaten in den USA eingesetzt wird. In der Folgezeit – bis Ende der vierziger Jahre unseres Jahrhunderts – wird die Technik der elektromechanischen Lochkartenmaschinen stark ausgebaut. Tabellier- und Sortiermaschinen werden zunehmend in der ökonomischen Datenverarbeitung eingesetzt. Ab 1930 werden Rechenlocher mit Multiplizierwerken ausgestattet.
1884	D. E. Felt erfindet eine tastaturbediente Rechenmaschine, die dann von W. W. Burroughs weiterentwickelt wird.
1920	Leonardo T. y Quevedo stellt einen Rechner mit elektromechanischen Bauelementen vor, der digital arbeitet.
1928	Leslie J. Comrie berechnet mit Hilfe eines elektromechanischen Rechners die Mondbahnen für die Jahre 1935 bis 2000.
1934–1942	Der Baustatiker Konrad Zuse entwickelt und baut verschiedene Varianten von Relaisrechnern – Z1 bis Z4 – und setzt sie für bautechnische Berechnungen ein. Wie bei vielen anderen technischen Entwicklungen, so wurde die Rechentechnik sehr schnell für militärische Zwecke mißbraucht. Im 2. Weltkrieg und danach beeinflussen militärische Forderungen wesentlich die Entwicklung der Rechentechnik. So werden höhere Rechengeschwindigkeiten gefordert als dies bei Relaisrechnern mit maximal 20 Additionen pro Sekunde möglich ist.
1944	In England wird der erste Rechner mit Elektronenröhren als elektronische Schalter gebaut. Der Rechner „Collossus" wird erfolgreich zum Entschlüsseln des Geheimcodes der Naziwehrmacht verwendet. Solche Röhrenrechner sind sehr groß und verbrauchen sehr viel Elektroenergie. Sie bestehen aus bis zu 18000 Röhren, wiegen ca. 30 t und erreichen bis ca. 5000 Additionen/s.
1946	In den USA wird der Rechner ENIAC fertiggestellt. Er enthält ca. 18000 Elektronenröhren und wiegt ca. 30 Tonnen.
1947	Erfindung des Transistors, der sehr schnell als elektronischer Schalter in Rechnern verwendet wird.

1951	Bau des ersten Rechners, in dem neben Röhren auch Halbleiterdioden und Magnetkernspeicher verwendet werden. Die Rechengeschwindigkeit liegt bei ca. 20000 Additionen/s.
1953	In der Sowjetunion wird der Röhrenrechner BESM und in Jena wird der Relaisrechner OPREMA gebaut. Letzterer wird in Jena zur Berechnung von Linsen eingesetzt. Etwa zur gleichen Zeit wird der erste volltransistorisierte Rechner TRADIC mit ca. 700 Transistoren gebaut und für Forschungen eingesetzt. Mit FORTRAN wird die erste problemorientierte Programmiersprache bereitgestellt. Sie findet vor allem bei Mathematikern und Naturwissenschaftlern eine weltweite Verbreitung. Mit COBOL folgt wenige Jahre später eine kaufmännisch orientierte höhere Programmiersprache.
1958	Kilby erfindet die integrierte Schaltung, die Grundlage für die weitere Miniaturisierung in der Elektronik ist. In Jena beginnt der Serienbau des Röhrenrechners ZRA1. Bei IBM beginnt der Serienbau eines volltransistorisierten Rechners.
1962	Mit der Anlage PDP10 wird der erste Minicomputer gebaut. Es beginnt die Entwicklung von Datenbanksystemen.
1965	IBM liefert die ersten Rechner der Modellreihe /360 aus und realisiert damit das Prinzip der Aufwärtsverträglichkeit von Programmen auf unterschiedlich großen Rechnern einer Rechnerfamilie.
1966	Die Serienproduktion des digitalen Kleinrechners D4a läuft an, der unter Leitung von Prof. Lehmann an der TU Dresden entwickelt wurde.
ab 1968	Die mittlere elektronische DVA R300 wird bei Robotron in Serie produziert. Im RGW werden Beschlüsse zur Entwicklung und dem Bau einer einheitlichen Rechnerfamilie – ESER I – gefaßt.
1970	Mit dem universellen Steuerbaustein 8008 wird der erste 8 Bit-Mikroprozessor auf den Markt gebracht.
1972	Erste Rechner der Familie ESER I werden ausgeliefert und kommen zum breiten Einsatz in Industrie, Forschung und Verwaltung.
1976	Der erste Mikroprozessor mit 16 Bit Verarbeitungsbreite wird herausgebracht.
1978	Die ersten industriell gefertigten Heimcomputer werden in den USA ausgeliefert.
1981	Die japanische Regierung beschließt das Forschungsprogramm „Fünfte Rechnergeneration". Andere Länder ziehen mit ähnlichen Programmen nach.
ab 1982	Erste Auslieferungen von Rechnern der Familie ESER II in den Ländern des RGW.
1983/1985	Entwicklung und Beginn der Produktion der Kleincomputer KC 85 in den Kombinaten Robotron und Mikroelektronik.
1986	Auslieferung der ersten Personalcomputer PC 1715 durch das Kombinat Robotron. Vorstellung des Arbeitsplatzcomputers A 7100 mit einem 16 Bit-Prozessor.

Tabellarischer Überblick über die einzelnen Rechnergenerationen

Generation Jahre	Erste 1946 bis 59	Zweite 1960 bis 68	Dritte 1969 bis 77	Vierte 1978 bis ?
Allgemeine Charakteristika	zentralisierte Großrechner mit Paketverarbeitung	zentralisierter Einsatz in großen RZ, kleinere Rechner entstehen	Prinzipiell neues Produkt: Minicomputer mit geringen Kosten und Anwendungen in Steuerungs-, Vermittlungs- und Datenverarbeitungssystemen. Ab 76 Superminirechner	Personalcomputer als arbeitsplatzorientierte Rechner, hochentwickelte Mensch-Rechner-Interaktion. Extrem dezentralisierte und vernetzte Supercomputer
Hardware	Vakuum-Röhren, Magnettrommel, Kathodenstrahlbildschirm	Transistoren, Magnetkernspeicher	Integrierte Schaltkreise, Halbleiterspeicher, Magnetplatte, Mini- und Mikrocomputer	VLSI-Schaltkreise, verteiltes Rechnen, optische Platten, Bubble-Speicher, Mikrorechner
Software	Gespeicherte Programme, Maschinensprache, Autocode	höhere Sprachen FORTRAN COBOL ALGOL	Höchstsprachen, Strukturierte Programmierung, Timesharing, Grafik, KI	Problemorientierte Programmpakete, objektorientierte Sprachen, Expertensysteme
Kommunikation	Telefon, Teletype	PCM-Übertragungssysteme	Paket-Vermittlungsnetze, Mikrowellen, Satelliten	Integrierte Systeme digitaler Netze
Typische Leistungsparameter	2-KByte-Speicher 10 Kilo-Instruktionen pro Sekunde (KIPS)	32 KByte 200 KIPS	2 MegaByte 3 MIPS	8 Mega-Byte 30 MIPS

3.2. Computer in Forschung, Produktion und Verwaltung

In zunehmendem Maße findet man in den Tageszeitungen oder den Berichten des Fernsehens Beiträge, die über den vielfältigen Einsatz von Computern berichten. So beispielsweise über

- ein rechnergestütztes System zur Herstellung der Schnittschablonen für die verschiedenen Größen in der Bekleidungsindustrie,
- intelligente Robotersteuerungen, die ihr Arbeitsprogramm „selbst lernen", wenn die einzelnen Arbeitsschritte im Rahmen des Teach-In-Verfahrens per Handsteuerung durchgeführt werden, u. ä,
- den Einsatz von Expertensystemen in der Geologie (Erdölsuche), Medizin (Unterstützung des Arztes bei der Diagnose) und Technik (Unterstützung bei der Fehlersuche in technischen Anlagen).

In letzter Zeit findet man aber auch Berichte, wo ganze Produktionslinien mit Rechnersteuerung arbeiten, so z. B.

- über einen vollautomatisierten, bedienarmen Fertigungsbereich für prismatische Teile im Drehmaschinenwerk Leipzig,
- über das flexible Fertigungssystem zur spanenden Bearbeitung von rotationssymmetrischen und gehäuseförmigen Einzelteilen im Elektromotorenwerk Dresden-Ost.

Diese unterschiedlichen Beispiele sind Varianten des Einsatzes des Computers, die alle unter dem Begriff CAM – Computer Aided Manufacturing (zu deutsch: rechnergestützte Produktion) zusammengefaßt werden. Ziel dieser Entwicklung ist der automatisierte Betrieb, in dem der Mensch mit einem wesentlich höheren Anteil an schöpferischer Tätigkeit auf einem hohen Qualifikationsniveau mehr Programmierer, Kontrolleur und Überwacher als Bediener mit körperlich schwerer Arbeit ist. Eine weitere Etappe auf diesem Weg wird mit CIM – Computer Integrated Manufacturing (zu deutsch: rechnerintegrierte Produktion) bezeichnet.

In vielen Berichten wird auf die rechnergestützte Konstruktion – CAD = Computer Aided Design – und ihre Nutzeffekte, aber auch auf die notwendigen Vorarbeiten verwiesen. Computergestützter Entwurf umfaßt den Computereinsatz zur Herstellung von Entwürfen in verschiedenen Ansichten und Schnitten, von Konstruktionsberechnungen, von technischen Zeichnungen des fertig Konstruierten und die Erstellung der Stücklisten aus den Zeichnungen. Grundelemente von CAD-Programmsystemen wurden in diesem Buch mit den verschiedenen Grafikprogrammen schon vorgestellt. Die breite Anwendung von CAD-Systemen ist ebenfalls ein Schritt zur automatisierten Fabrik.

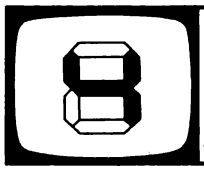
Das Anwendungsfeld des Computers in der Forschung ist sehr vielfältig. Wesentliche Aufgaben sind hierbei

- Berechnungen,
- Meßwerterfassung und -auswertung,
- Steuerung wissenschaftlicher Großgeräte,
- Simulation von Experimenten u. a.

Dabei werden die unterschiedlichsten Rechner – vom Kleincomputer bis zu Großrechenanlagen – genutzt. Zunehmend werden Rechner zur Simulation von Neuentwicklungen chemischer Verbindungen, von elektronischen Schaltungen, von Maschinenelementen und Maschinen genutzt, wobei dies auf der Grundlage komplexer mathematischer Modelle in Programmform erfolgt. Durch solche Simulationen, die wesentlich schneller als reale Experimente ablaufen, können Entwicklungszeiten, Kosten und Material eingespart werden. Auf heute verfügbaren Computern wird beispielsweise deren eigene Weiterent-

wicklung entworfen, werden die Schaltkreise entwickelt und mittels Simulation das Zusammenspiel aller Komponenten getestet. Ein gravierendes Beispiel dafür ist die Entwicklung eines Prozessors eines neuen Rechnersystems auf einem mittleren Personalcomputer. Dieser Prozessor verwendet als „Maschinensprache" die höhere Programmiersprache FORTH und arbeitet deshalb in einigen Aufgabengebieten schneller als heutige Prozessoren mit 32 Bit Verarbeitungsbreite.
Ein wichtiges Feld der Nutzung des Computers in der Forschung besteht in der Anlage von großen Datenbeständen, die über Datenübertragungseinrichtungen international abgefragt werden können. Der Zugriff zu speziellen Informationen ist so zeitgünstiger als mit Recherchen in Zeitschriften, Patentschriften oder der jeweiligen Fachliteratur. Ein Entwickler kann sich so schnell informieren, was international auf seinem Gebiet in letzter Zeit erarbeitet wurde.
Die Wirkungen der Anwendung der Datenverarbeitung in der Verwaltung spürt inzwischen jeder Bürger im täglichen Leben. Die Anmeldung für ein Zeitungsabonnement oder für die Teilnahme an Lotto und Toto im Dauerspiel mit Abbuchung vom Konto erfolgt auf datenverarbeitungsgerechten Belegen. Kontoauszüge der Sparkassen, Lohn- und Gehaltsabrechnung, Energie- und Postabonnementsabrechnungen werden auf Computerausdrucken dem Kunden übergeben. Auch die Personenkennzahl und die ELN – einheitliche Liefernummer in der Materialwirtschaft – sind Schlüsselsysteme, die zur breiten Nutzung der Informationsverarbeitung zur Rationalisierung der Verwaltung entwickelt wurden.
In der ersten Etappe der Anwendung der Datenverarbeitung in der Verwaltung wurden vor allem Einzelprojekte – z. B. Lohn-, Material- und Grundmittelrechnung – entwickelt und angewendet. Bei diesen Einzelprojekten wurden häufig gleiche Daten mehrfach erfaßt, so daß der erzielte Gesamteffekt bald nicht mehr den Anforderungen genügte.
Der zunehmende Einsatz der Computer in der Produktion und die weitere Rationalisierung der Verwaltung verlangen immer stärker, daß komplexe Systeme der Datenverarbeitung entwickelt und eingesetzt werden. Ein Vergleich der bisher in der Produktion und in der Verwaltung erzielten Produktivitätssteigerung zeigt ein Verhältnis von 3:1 und damit die Notwendigkeit einer drastischen Steigerung der Produktivität in der Verwaltung.
Durch den Aufbau und die Nutzung von Datenbanken als wichtige Grundlage komplexer Projektlösungen kann eine wesentliche Rationalisierung erzielt werden, da die Daten – einmal erfaßt – nach den verschiedensten Gesichtspunkten verknüpft und ausgewertet werden können. Durch die Verbindung mit der mechanisierten bzw. automatisierten Datenerfassung in der Fertigung und Fertigungsvorbereitung kann dieser Effekt noch gesteigert werden.
Die genannten Aufgaben sind Schritte zu Gesamtlösungen der Informationsverarbeitung in der Industrie in Richtung auf die automatisierte Fabrik.
Der Einsatz der Datenverarbeitung in der Verwaltung ist mit den oben dargestellten Fakten in der Industrie natürlich nicht vollständig erfaßt. Auch im Handel, Transportwesen und in staatlichen Verwaltungen werden schrittweise informationsverarbeitende Systeme aufgebaut, die zur Rationalisierung dieser Tätigkeiten beitragen.
In diesem Zusammenhang ist es bedeutsam, daß nur unter sozialistischen Produktionsverhältnissen eine Nutzung der informationsverarbeitenden Technik zum Wohle aller Menschen möglich ist. Ein Kapitalist nutzt diese Technik nur unter dem Aspekt der Rationalisierung von Produktion und Verwaltung zur Erzielung des Maximalprofits. Soziale Probleme spielen dabei eine untergeordnete Rolle; sie werden nur durch den Kampf starker Gewerkschaften gemildert. Das Streben nach Maximalprofit kommt auch in der verstärkten Rüstungsproduktion zum Ausdruck, die weder Arbeitsplätze schafft noch auf die Dauer erhält. Viele Waffen, wie die Cruise Missiles, Interkontinentalraketen mit gesteuer-

ten Mehrfachsprengköpfen oder SDI wären ohne den Mißbrauch der informationsverarbeitenden Technik überhaupt nicht realisierbar.
So besagen wissenschaftliche Schätzungen, daß ca. 15000 bis 75000 Mannjahre Entwicklungskapazität (d.h. 1500 Personen arbeiten 10 bis 50 Jahre an dieser Problematik) für die SDI-Software benötigt wird. Solche Systeme sind aber gar nicht mehr in allen Einzelheiten überprüfbar, so daß nicht gefundene Fehler die katastrophalsten Folgen haben können. Meldungen über solcherart verursachte Pannen in den verschiedensten Waffensystemen hat es ja schon reichlich gegeben. Aber die Eingreifzeit für den Menschen im Fehlerfalle und damit die Zeit zur Verhinderung einer nuklearen Katastrophe wird bei den neuentwickelten Systemen immer geringer.
Es gilt also für alle friedliebenden Menschen der Welt, diesen Entwicklungen den entschiedenen Kampf um Abrüstung und Erhaltung des Friedens entgegenzusetzen. Die friedliche Nutzung der informationsverarbeitenden Technik zum Wohle der Menschheit bietet ein vielfältiges Spektrum an interessanten Aufgaben, die es zu realisieren gilt.



Lösen nichtnumerischer Probleme mittels Kleincomputer

Das Interesse denkt nicht, es rechnet.
Die Motive sind reine Zahlen.
(Karl Marx)

Computer wurden entwickelt, um möglichst schnell und zuverlässig viel zu rechnen und den Menschen damit von lästiger Routinearbeit zu befreien. Die überwiegende Anzahl der heutigen Computernutzer löst mittels Computer aber Aufgaben und Probleme, die nicht zahlenmäßiger Natur sind, wenngleich der Computer intern alles in zahlenmäßiger Form verarbeitet.
Zu typischen nichtnumerischen Anwendungsbereichen von Computern zählen die Textverarbeitung, der computergestützte Entwurf (CAD), die computergestützte Produktion (CAM) und die Produktionsplanung, -kontrolle und -steuerung.
Die rationelle Lösung dieser Aufgaben und Probleme ist für die wirtschaftliche Entwicklung eines Landes so wichtig und im Kern durch vergleichbare Teilprobleme gekennzeichnet, daß es sogenannte Benutzersysteme gibt. Computerhersteller und Softwarebetriebe bieten sie den Benutzern von Computern als fertige, umfangreiche und leistungsstarke Programmsysteme an. Es gibt für Personal- oder Bürocomputer zwischenzeitlich unterschiedliche Benutzersysteme, beispielsweise für Textverarbeitung das System **TP** oder das Datenbanksystem **REDABAS.** Sie geben dem Benutzer so viele Möglichkeiten, wie sie durch eine Eigenentwicklung in einer problemorientierten Programmiersprache – z. B. BASIC – nur mit einem unvertretbaren Zeitaufwand erstellt werden könnten. Diese Systeme bieten leistungsfähige Kommandos bis hin zur „Programmierung in der Sprache des Benutzersystems" für die Lösung vielfältiger Aufgaben.
Die Programmierung in der Sprache eines Benutzersystems unterscheidet sich kaum von der in einer problemorientierten Programmiersprache, wie BASIC, PASCAL usw. Das Denken in und Arbeiten mit den Strukturelementen Folge, Auswahl, Wiederholung und Unteralgorithmenaufruf, die Differenzierung von Datentypen und Datenstrukturen sind das Verbindende; die Codierung in der Sprache des Benutzersystems ist das Spezifische.

4.1. Dateien und Sortierprogramme

4.1.1. Problembeschreibung

In Dateien werden Daten gespeichert, um sie für weitere Bearbeitungsschritte maschinenlesbar zu haben. Die Daten einer Datei werden nach gewissen Kriterien zusammengefaßt; ein Beispiel im folgenden Abschnitt verdeutlicht dies. Zur Lösung unterschiedlich-

ster Aufgaben müssen Dateien auf den aktuellen Stand gebracht und dann ausgewertet werden. Viele Aufgaben beziehen sich dabei auf Such-, Misch- und Sortiervorgänge in der Menge der Datensätze einer oder auch mehrerer Dateien. Es können Beziehungen zwischen den Daten herausgearbeitet und statistische Aussagen getroffen werden. Mehrere Dateien, die eine bestimmte Organisationseinheit umfassend beschreiben, können in einer Datenbank zusammengefaßt und rationell verarbeitet werden.
Die Darstellung der Verbindung einer zentralen Datenbank eines Betriebes zum Reproduktionsprozeß (nach T. Schmieder in [8]) unterstreicht in eindrucksvoller Weise die Bedeutung der computergestützten Datenverarbeitung.

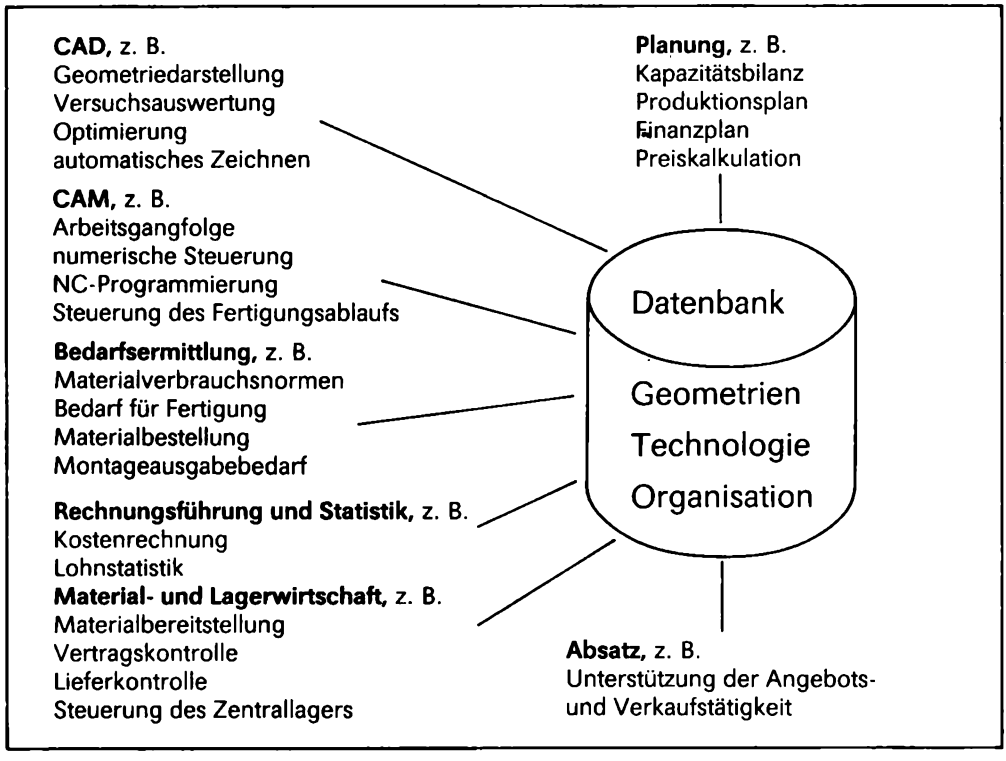


Abb. 4.1/1: Zentrale Datenbank im Reproduktionsprozeß

Eine Datenbank besteht aus mehreren Dateien. Eine **Datei** (engl. file) beinhaltet eine Menge gleichartiger Datensätze. Ein **Datensatz** (engl. record) ist aus Einzeldaten – so wie ein Satz der deutschen Sprache aus Satzgliedern – aufgebaut. Die Zusammenstellung von Einzeldaten zu einem Datensatz unterliegt im allgemeinen einer gewissen Vorschrift. Denkt man z. B. an eine Einwohnerdatei, so zeigt ein Meldeformular wichtige Daten auf. Das Meldeformular sei auf die Daten in Abbildung 4.1/2 reduziert.
Die Daten zu einer Person gehören zusammen – sie bilden einen Datensatz. Es soll möglich sein, auf einzelne Daten eines Datensatzes, z. B. auf die PKZ, den Namen oder auf die PLZ zugreifen zu können. Ein Datensatz EINWOHNER umfaßt im Beispiel Daten, die unterschiedlichen Datentypen zugeordnet werden können. Die Daten Name, Vorname, Ort und Straße müssen vom Datentyp Zeichenkette sein, während die Personenkennzahl, die Postleitzahl, die Hausnummer und die Telefonnummer vom Datentyp (ganze) Zahl

sein können. Dabei ist dies z. B. bei einer Hausnummer manchmal nicht möglich, wenn Aufgänge eines Gebäudes in 14a, 14b, 14c unterschieden werden.

```
          Meldeformular

   Name:    (max. 14 Zeichen)    --------------
Vorname:    (max. 14 Zeichen)    --------------
    PKZ:    (     12 Zeichen)    ------------
Strasse:    (max. 14 Zeichen)    --------------
Hausnr.:    (max.  4 Zeichen)    ----
    Ort:    (max. 14 Zeichen)    --------------
    PLZ:    (      4 Zeichen)    ----
Tel.Nr.:    (max.  7 Zeichen)    -------
```

Abb. 4.1/2: Meldeformular für Einwohnerdatei (vereinfacht)

Unter einem Verbund (record) versteht man einen strukturierten Datentyp (Datenstruktur), der aus einer festen Anzahl von Komponenten, möglicherweise unterschiedlicher Datentypen besteht.

Die Datenstruktur Verbund will man geschlossen handhaben. Beispielsweise soll ein Verbund vollständig auf dem Bildschirm angezeigt oder über Drucker ausgegeben oder aus einer Datei gelöscht werden. Deshalb erhält ein solcher Verbund auch einen Namen, unter dem die Bezugnahme innerhalb von Programmen realisiert wird. In BASIC ist die Datenstruktur Verbund nicht vorgesehen. Da es in BASIC möglich ist, numerische Ausdrücke in Zeichenketten umzuwandeln, wird der Verbund durch ein Feld von Zeichenketten simuliert.

1 Es ist ein Programmsystem zur Führung einer Einwohnerdatei zu erstellen.

4.1.2. Algorithmierung

Die Abbildung 4.1/3 zeigt, daß eine Vielzahl von Moduln zur Lösung der Aufgabenstellung zu entwickeln ist. In der Abbildung sind die schon bekannten Module umfangreicher Programme, wie Titelbild, Nutzungshinweise und Endebehandlung nicht angegeben. Sie sind aber trotzdem zu erstellen. Dazu kommen noch Module, wie Kennziffer- bzw. Kennzahleingabe und Zulässigkeitsprüfung, die Ausgabe eines Formularrahmens (entsprechend einer Karteikarte), der für die Eingabe, das Ändern und die Anzeige eines Datensatzes sowie für das ‚Blättern' in der Datei verwendet werden kann.

Außerdem können jetzt schon einige Ideen zur weiteren Verbesserung des Programms notiert werden, ohne diese Ideen in der ersten Programmversion realisieren zu wollen. Beim Suchprozeß muß man davon ausgehen, daß für den Computer z. B. die Zeichenketten „Berlin", „BERLIN" und „berlin" verschieden sind. Eine Lösungsverbesserung könnte eine Normierung der Zeichenketten auf eine Schreibweise bei der Eingabe beinhalten. Für bestimmte Benutzer des Programms kann es vorteilhaft sein, Suchbedingungen kombinieren zu können. Beispielsweise benötigt das Wehrkreiskommando die Daten aller Einwohner, die im laufenden Jahr 18 Jahre alt werden und männlichen Geschlechts sind, um ihnen Benachrichtigungen zur Musterung zuzusenden.

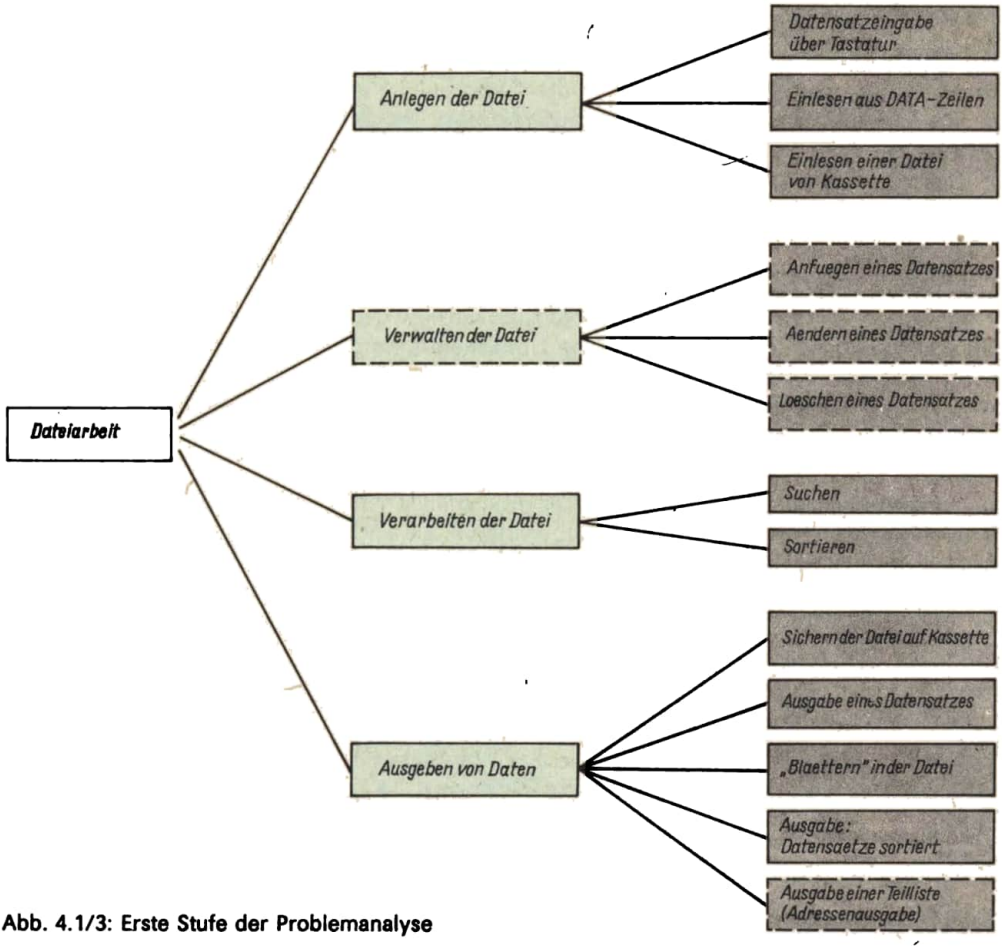


Abb. 4.1/3: Erste Stufe der Problemanalyse

Um das Programm systematisch erarbeiten zu können, ist es sinnvoll, Anweisungsnummernbereiche für die einzelnen Module festzulegen.

10 – 90	Kurzinformationen zur Programmerstellung
100 – 990	Hauptprogramm
1000 – 1990	Titelbild
2000 – 2990	Erläuternder Text – sog. Hilfeseiten, die von jeder Stelle des Programms aufrufbar sind
3000 – 3990	Initialisierung der Parameter
4000 – 4990	Kennzahleingabe und Zulässigkeitsprüfung
5000 – 5990	Kennziffereingabe und Zulässigkeitsprüfung
6000 – 6990	Formularrahmen
7000 – 7990	Behandlung des Programmendes
10000 – 19990	Untermenue 1: ‚Anlegen der Datei'
20000 – 29990	Untermenue 2: ‚Verwalten der Datei'
30000 – 39990	Untermenue 3: ‚Verarbeiten der Datei'
40000 – 49990	Untermenue 4: ‚Ausgeben von Daten'
50000 – 65000	Daten in DATA-Zeilen

Hauptalgorithmus

Für den Test eines entsprechenden Programms auf dieser recht allgemeinen Abstraktionsstufe werden die Übergänge zu den Untermenüs vorerst durch Entwürfe der Bildschirmdarstellungen dieser Untermenüs realisiert. Als Beispiel für die 5 Auswahlmenüs sei hier der Entwurf für das Hauptmenü angegeben (Abb. 4.1/5).
Wird bei der Eingabe der Kennziffer ein Fehler festgestellt, so wird je nach Art des Eingabefehlers in dem rechten unteren Bildbereich rot blinkend ‚Eingabefehler!' und einer der Hinweise

nur Ziffern eingeben bzw. 0 <= K <= 4

eingetragen, um zur korrekten Eingabe aufzufordern.
So wie die Bildschirmentwürfe für die Untermenüs dem des Hauptmenüs gleichen, so kann auch der Teil des Hauptprogramms in Analogie für die Untermenüs verwendet werden, der die Bildschirmentwürfe und die berechnete, mehrseitige Auswahl zu weiteren 3 oder 4 Unterprogrammen realisiert. Deshalb wird hier nur das Struktogramm (Abb. 4.1/4) für das Hauptprogramm (↗ S. 171) angegeben.

● Entwickeln Sie die Struktogramme zu den Untermenüs 1 bis 4 selbständig! Sie gleichen der großen Wiederholschleife im Hauptalgorithmus. Es sind nur wenige spezifische Änderungen zu machen. Um allein schon von der Farbe her zu wissen, in welchem Untermenü man sich gerade befindet, sind die Kombinationen für die Vordergrund- und Hintergrundfarbe neu zu bestimmen. Die Bezeichnungen für die Wahlmöglichkeiten und auch die Anzahl ist auf die jeweilige Untermenü-Anforderung zu konkretisieren. Die Kennzahl 0 wird nun stets für ‚zurueck ins Hauptmenue' verwendet. Die zu wählende Kennzahl soll die Variablenbezeichnung KZ erhalten, um den aktuellen Wert der Kennzahl im Hauptprogramm von dem im aktuellen Unterprogramm unterscheiden zu können. KZ ist auch die Ausgangsvariable des Unteralgorithmus ‚Kennzahleingabe und Zulässigkeitsprüfung'. Damit ist die Umspeicherung K:=KZ nicht notwendig.

Entwicklung der Hilfsalgorithmen

Die Initialisierung der Parameter erfolgt in Abhängigkeit der Art des Anlegens einer Datei. In jedem Fall ist die Anzahl P der Personen, deren Daten erfaßt werden sollen, mitzuteilen, und es ist die Dimensionierung des zweidimensionalen Zeichenkettenfeldes DA$, das den Verbund simuliert, vorzunehmen, damit der Computer genügend Speicherplatz für die zu verarbeitenden Daten reserviert (Abb. 4.1/6).
Die Eingabe von Zahlen und die Prüfung auf Zulässigkeit ist nach dem Algorithmus „KENNZAHLEINGABE UND ZULÄSSIGKEITSPRÜFUNG" (Abb. 4.1/7) realisierbar.
Bei den Auswahlmenüs sind nur Ziffern einzugeben. Um nicht zusätzlich zur Ziffereingabe noch die ENTER-Taste drücken zu müssen, kann die Eingabe über Tastaturabfrage mittels INKEY$ realisiert werden. Diese Änderung ergibt einen fast gleichen Algorithmus für das Problem „Kennziffereingabe und Zulässigkeitsprüfung" wie bei dem Teilproblem „Kennzahleingabe und Zulässigkeitsprüfung" in Abbildung 4.1/7. Auf eine Darstellung des Struktogramms wird hier verzichtet.
Es sei noch der Unteralgorithmus „DATEI SICHERN" entworfen:
Das einzige neue Element im Struktogramm DATEI SICHERN ist der Abspeicherungsbefehl FELD-AUF-KASSETTE-SICHERN. Das geschieht mit der BASIC-Anweisung

```
CSAVE*"DATEI";DA$
```

(EINWOHNERDATEI)

(Variable:
V, H – Vorder- bzw. Hintergrundfarbe
U, O – untere bzw. obere Grenze zuläss. Wertebereich
KZ – Ausgangsvariable des UA KENNZIFFEREINGABE
K – Kennziffer im Hauptprogramm
I$ – Eingabezeichenkette
Unterprogramme: TITELBILD
INITIALISIERUNG DER PARAMETER
KENNZIFFEREINGABE UND ZULÄSSIGKEITSPRÜFUNG
UM 1 ‚ANLEGEN DER DATEI'
UM 2 ‚VERWALTEN DER DATEI'
UM 3 ‚VERARBEITEN DER DATEI'
UM 4 ‚AUSGEBEN VON DATEN'
DATEI SICHERN
ENDEBEHANDLUNG)

RUFE TITELBILD

RUFE INITIALISIERUNG DER PARAMETER

WIEDER-
HOLE

Normalfenster : V := schwarz : H := gelb : Bildschirm löschen

Ausgabe „Hauptmenü:"
„0 Programm beenden"
„1 Anlegen der Datei"
„2 Verwalten der Datei"
„3 Verarbeiten der Datei"
„4 Ausgeben der Daten"
„Wählen Sie die Kennziffer K!"

U := 0 : O := 4

RUFE KENNZIFFEREINGABE UND ZULÄSSIGKEITSPRÜFUNG

K := KZ

K > 0
ja
n.

FALLS K =
1
RUFE UM 1
2
RUFE UM 2
3
RUFE UM 3
4
RUFE UM 4
%

BIS K = 0

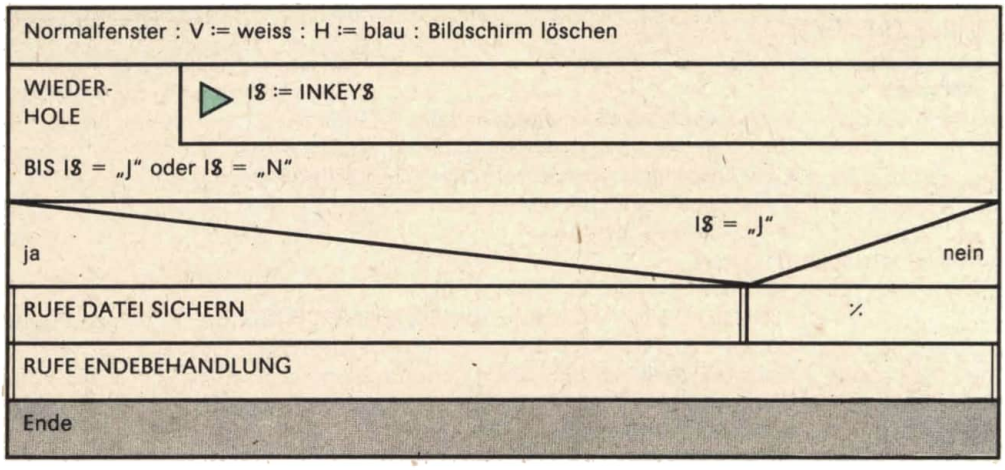


Abb. 4.1/4: Struktogramm des Hauptalgorithmus „Einwohnerdatei"

```
HAUPTMENUE:

   0 Programm beenden
   1 Anlegen der Datei
   2 Verwalten der Datei
   3 Verarbeiten der Datei
   4 Ausgeben von Daten



  Waehlen Sie
  die Kennziffer K
```

Abb. 4.1/5: Hauptmenü

(UA INITIALISIERUNG DER PARAMETER)
BEGINN
Normalfenster mit V = schwarz, H = grau : Bildschirm löschen

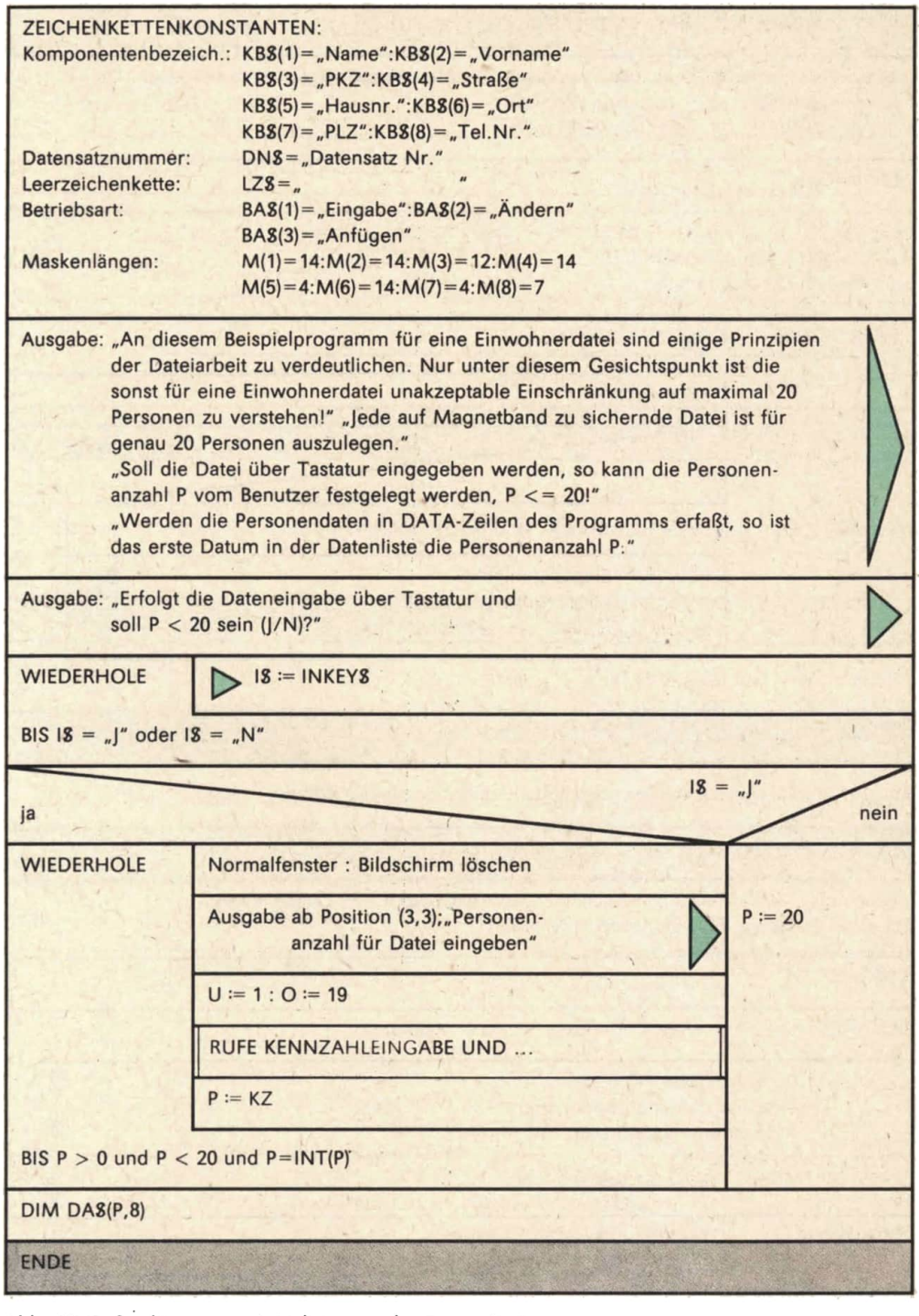


Abb. 4.1/6: Struktogramm „Initialisierung der Parameter"

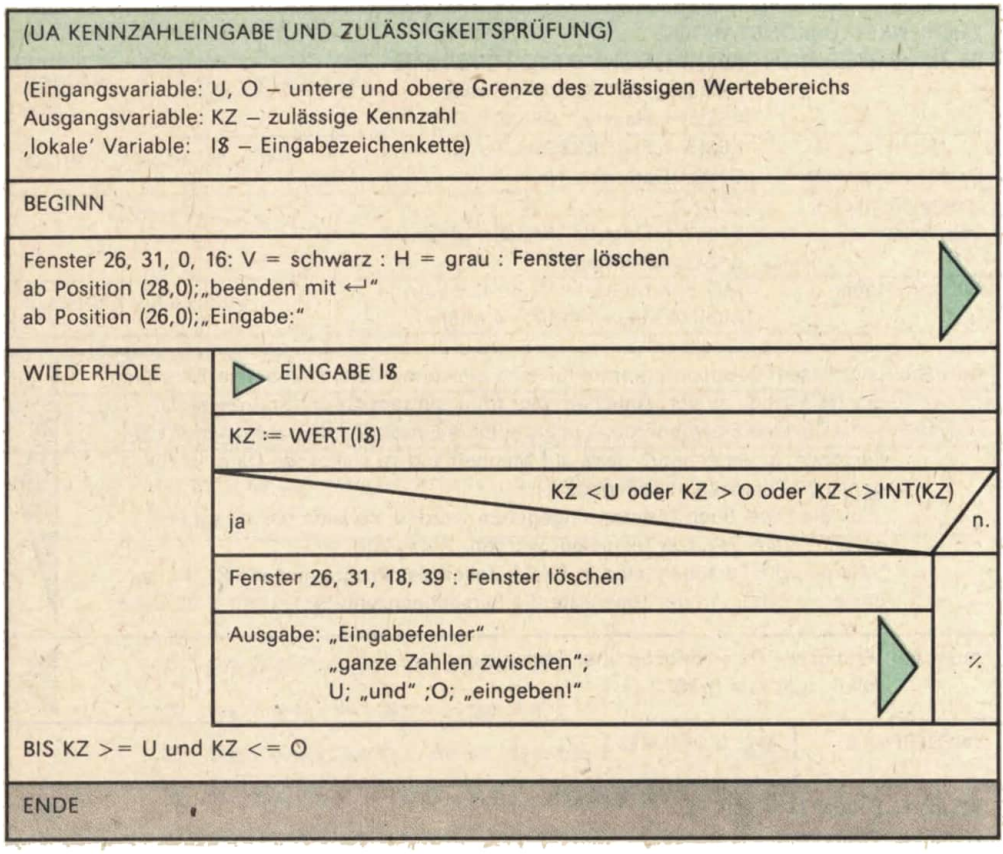
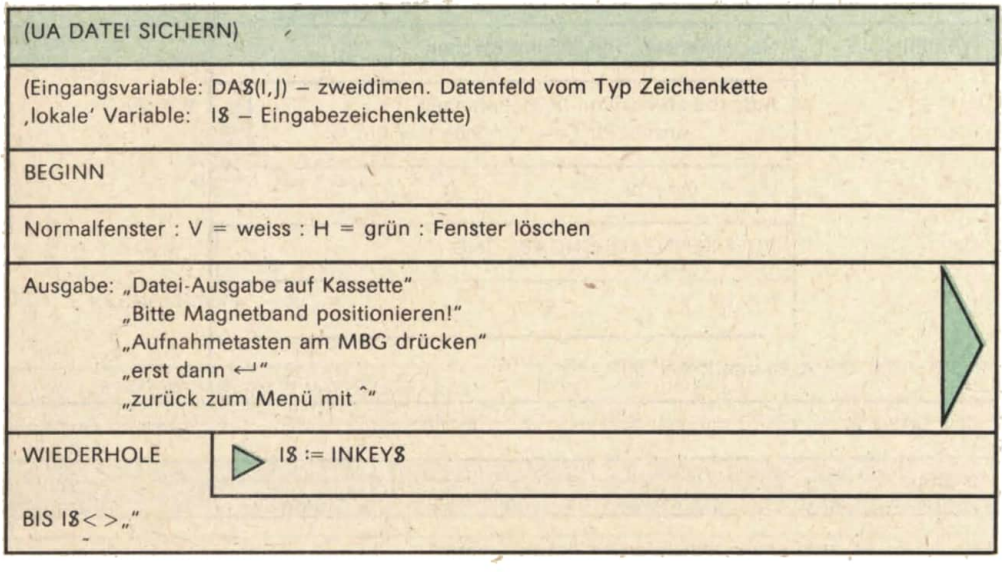


Abb. 4.1/7: Struktogramm „Kennzahleingabe und Zulässigkeitsprüfung"



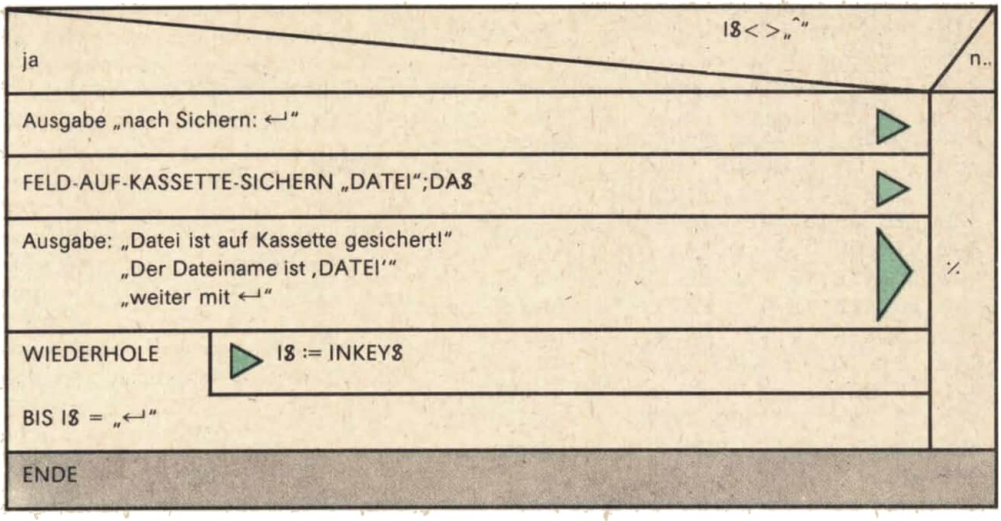


Abb. 4.1/8: Struktogramm zu „DATEI SICHERN"

Ein so abgespeichertes Datenfeld kann man von der Kassette mit der BASIC-Anweisung

```
CLOAD*"DATEI";DA$
```

wieder einlesen.

Codieren Sie die Unteralgorithmen INITIALISIERUNG DER PARAMETER, KENNZAHLEINGABE UND ZULÄSSIGKEITSPRÜFUNG, KENNZIFFEREINGABE UND ZULÄSSIGKEITSPRÜFUNG und DATEI SICHERN.

Entwickeln Sie einfache Varianten zum TITELBILD und zur ENDE BEHANDLUNG, um die Korrektheit des Hauptprogramms austesten zu können. In einer späteren Arbeitsphase sollten diese einfachen Varianten gegen anspruchsvollere – auf ästhetische Bildschirmgestaltung zielende – Unterprogramme ausgetauscht werden.

Das Hauptprogramm hat folgende Form:

```
100 REM HAUPTPROGRAMM ================================

190 CLEAR 1000
200 GOSUB 1000 : REM → Titelbild
210 GOSUB 3000 : REM → Initialisierung der Parameter
220 REM anfang der wiederholschleife
230   WINDOW 0,31,0,39 : COLOR 0,6 : CLS
240   LOCATE 2,3  : PRINT "HAUPTMENUE:"
250   LOCATE 5,5  : PRINT "0  Programm beenden"
260   LOCATE 7,5  : PRINT "1  Anlegen der Datei"
270   LOCATE 9,5  : PRINT "2  Verwalten der Datei"
280   LOCATE 11,5 : PRINT "3  Verarbeiten der Datei"
290   LOCATE 13,5 : PRINT "4  Ausgeben von Daten"
300   LOCATE 28,3 : PRINT "Waehlen Sie"
```

```
310   LOCATE 29,3 : PRINT "die Kennziffer K"
320   U=0 : O=4
330   GOSUB 5000 : REM → Kennziffereingabe und Zulaessig.
340   K=KZ
350   REM  anfang mehrseitige auswahl
360     ON K GOSUB 10000, 20000, 30000, 40000
370   IF NOT(K=0) THEN 230
380 REM ende der wiederholschleife
390 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
400 LOCATE 10,1 : PRINT "Datei auf Magnetband "
410 LOCATE 12,1 : PRINT "sichern, dann    < J >"
420 LOCATE 14,17: PRINT " andernfalls     < N >"
430 I$=INKEY$ : IF I$="" THEN 430
440 IF NOT(I$="J" OR I$="N") THEN 430
450 IF I$="J" THEN GOSUB 41000 : REM - > Datei sichern
460 GOSUB 7000 : REM - > Endebehandlung
990 END
```

Erstellen strukturell ähnlicher Programmteile am Beispiel des Untermenüs 1 ANLEGEN EINER DATEI

Die Codierung der Untermenüs gleichen sich untereinander und dem Kern des Hauptprogramms so stark, daß nur die Codierung des Untermenüs 1 stellvertretend für die anderen 3 hier angegeben wird.

```
10000 REM UP UNTERMENUE 1 === Anlegen einer Datei ===
10100 REM anfang der wiederholschleife
10110   WINDOW 0,31,0,39 : COLOR 7,1 : CLS
10120   LOCATE 2,3 : PRINT "UNTERMENUE 1: 'Anlegen einer Datei'"
10130   LOCATE 5,5 : PRINT "0  zurueck zum Hauptmenue"
10140   LOCATE 7,5 : PRINT "1  Datensatzeingabe ueber Tastatur"
10150   LOCATE 9,5 : PRINT "2  Einlesen aus DATA-Zeilen"
10160   LOCATE11,5 : PRINT "3  Einlesen von Kassette"
10170   LOCATE28,3 : PRINT "Waehlen Sie"
10180   LOCATE29,3 : PRINT "die Kennziffer KZ"
10190   U=0 : O=3
10200   GOSUB 5000 : REM - > Kennziffer/Pruefung
10210   REM  anfang mehrseitige auswahl
10220     ON KZ GOSUB 11000, 12000, 13000
10230   IF NOT(KZ=0) THEN 10100
10240 REM ende der wiederholschleife
10990 RETURN
```

Wenn Programmabschnitte sich so stark gleichen wie die Zeilen 220–380 mit den Zeilen 10100–10240 aus dem Untermenü und in den entsprechenden Zeilen der weiteren Untermenüs, dann bietet sich für die Rationalisierung der Codierung folgendes Vorgehen an.
Nach der Programmeingabe in den KC wird es auf Kassette gesichert, dann werden alle Anweisungszeilen außer den Zeilen 220–380 gelöscht. Mit RENUMBER 220, 380, 45000,

10 wird das Programmstück weit nach hinten verlagert. Auch dieses Programmstück wird auf Kassette gesichert. Nach einem NEW-Kommando steht kein Programm mehr im Nutzerbereich des Hauptspeichers. Dann wird das Hauptprogramm wieder geladen und im Anschluß das – der Form nach – auch im Untermenü 1 notwendige Programmstück. Dies kann mit RENUMBER 45000,45160,10100,10 genau an die richtige Position gesetzt werden, so daß nur noch die Kopf- und Schlußzeilen ergänzt werden müssen. Natürlich sind nun noch die auszugebenden Texte zu aktualisieren. Mit EDIT 10100 ergibt sich die Möglichkeit des Edierens. Diesen Vorgang kann man für die anderen Untermenüs in gleicher Weise durchführen. Das RENUMBER-Kommando für das jeweils neu von der Kassette eingelesene Programmstück ist dabei dem vorgegebenen Zeilenbereich anzupassen.
Diese Arbeitsweise setzt natürlich die oben dargestellten Vorarbeiten voraus. Dann spart sie nicht nur Zeit, sondern vermindert die Zahl der Schreibfehler bei der Programmeingabe. Bei leistungsfähigeren Computern gibt es die Möglichkeit, Teile von Programmen gleich im Hauptspeicher geeignet umzukopieren, so daß der Weg über externe Speicher nicht notwendig ist.

Programme zur Dateneingabe und Datenausgabe

Zuerst soll überlegt werden, wie die Datenein- und -ausgabe auf dem Bildschirm dargestellt werden kann. Es ist ein Formularrahmen (Abb. 4.1/9) zu entwerfen, der dem Meldeformular ähnelt und in dem zwar die Bezeichnungen der Datensatzkomponenten eingetragen werden, aber nicht der Vermerk, wieviel Zeichen für eine Komponente vorgesehen sind.
In der ASCII-Tabelle des KC 85/3 gibt es grafische Symbole für Steuerzeichen, so daß diese im Befehlsmenü angegeben werden können:

ASCII-Zahl	Steuerzeichen	grafisches Symbol	dazugehörige ASCII-Zahl
8	Cursor links	←	136
9	Cursor rechts	→	137
10	Cursor abwärts	↓	138
11	Cursor aufwärts	↑	139

Um z. B. die Bewegung des Cursors nach links bzw. rechts zu erreichen, ist die Anweisung PRINT CHR$(8) bzw. PRINT CHR$(9) auszuführen.
Oberhalb des Formularrahmens ist links der Text „Datensatz Nr.:" und die aktuelle Datensatz Nr. vermerkt und rechts die aktuelle Betriebsart, hier „Eingabe", eingetragen. Wird der Formularrahmen für die Dateneingabe genutzt, so werden in Form von Strichfolgen die Eingabemasken für die einzelnen Komponenten in der rechten Hälfte des Formulars vorgegeben. Ein blinkender Zeiger markiert die aktuelle Eingabeposition.

Soll der Rahmen beispielsweise bei der Ausgabe eines Datensatzes genutzt werden, sind die aktuellen Komponenteninhalte des Datensatzes in die dann leere rechte Hälfte des Formulars zu schreiben. Diese verschiedenen Anwendungszwecke verlangen eine flexible Gestaltung der Unteralgorithmen. Die 4 Unteralgorithmen FORMULARRAHMEN, KOMPONENTENBEZEICHNUNGEN, EINGABEMASKE und IN FORMULAR EINTRAGEN sichern die Verwendbarkeit für die unterschiedlichen Zwecke.

Für alle Anwendungsfälle, die die Formulardarstellung benutzen, werden die Teilalgorithmen FORMULARRAHMEN und KOMPONENTENBEZEICHNUNGEN verwendet. Für die Eingabe und für das Ein- bzw. Anfügen eines Datensatzes kommt noch der Teilalgorith-

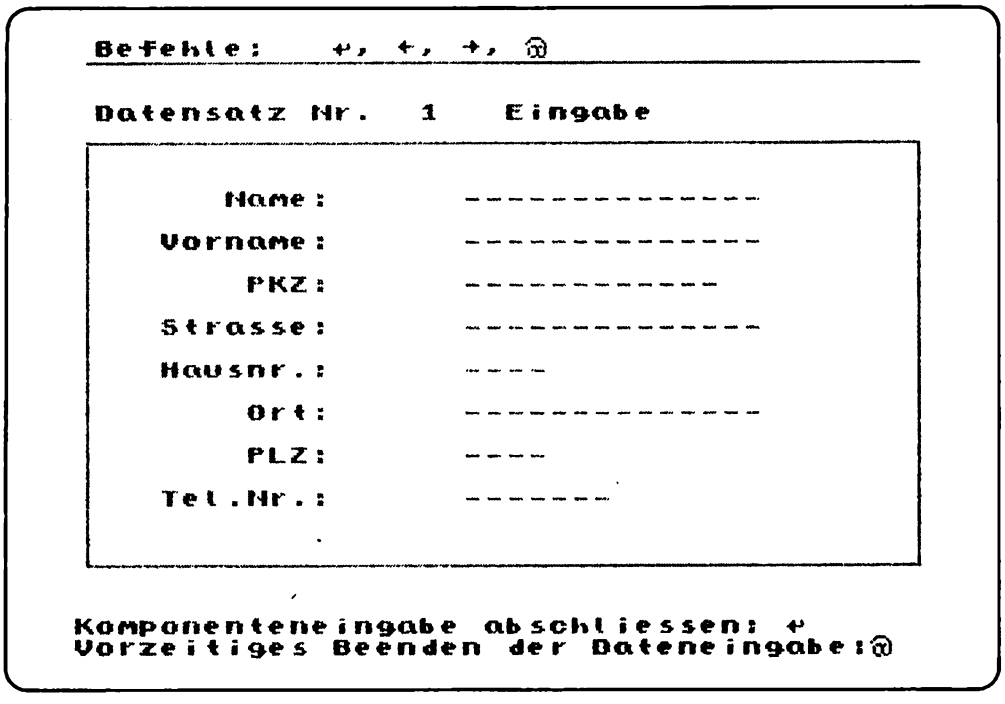


Abb. 4.1/9: Formularrahmen für Dateneingabe

mus EINGABEMASKE hinzu und bei der Ausgabe, beim ‚Blättern' und beim Löschen der Teilalgorithmus IN FORMULAR EINTRAGEN.

Die Algorithmen seien gleich in BASIC formuliert, wobei aus Platzgründen die Informationen über Variablen gelegentlich weggelassen werden.

```
6000 REM UP FORMULARRAHMEN ============================
6010 REM Eingangsvariablen:
6020 REM  BE$      - Befehlsmenuezeile
6030 REM  BA         - Betriebsart
6040 REM Ausgangsvariable:
6050 REM  BA$(BA)  - Betriebsartenbezeichnung
6060 REM Unterprogramme:
6070 REM  KOMPONENTENBEZEICHNUNGEN
6080 REM  IN FORMULAR EINTRAGEN
6090 REM  EINGABEMASKEN
6100 REM ----------------------------------------
6280 PAPER 7 : CLS
6290 LINE 5,50,314,50,0
6300 LINE 5,50,5,208,0
6310 LINE 5,208,314,208,0
6320 LINE 314,208,314,50,0
6330 REM
6340 PRINT AT(1,1);BE$
6350 LINE 5,238,314,238,0
```

```
6360 PRINT AT(4,1);DN$
6370 PRINT AT(4,20);BA$(BA)
6380 GOSUB 8500 : REM - > Komponentenbezeichnungen
6490 RETURN : REM ---------------------------------

6500 REM IN FORMULAR EINTRAGEN ======================
6510 REM Eingangsvariable:
6520 REM   PE       - Personennummer
6530 REM Ausgangsvariable:
6540 REM   DA$(PE,I) - Daten: Person PE
6550 REM 'lokale' Variable:
6560 REM   I        - Laufvariable
6570 REM ------------------------------------------
6600 CLS
6610 FOR I=1 TO 8
6620   PRINT DA$(PE,I)
6630   PRINT
6640 NEXT I
6990 RETURN : REM ----------------------------------

8000 REM UP EINGABEMASKEN ===========================

8090 FOR I=1 TO 8
8100   PRINT AT(6+I+I,18);STRING$(M(I),"-")
8110 NEXT I
8490 RETURN : REM ----------------------------------

8500 REM UP KOMPONENTENBEZEICHNUNGEN ================
8590 FOR I=1 TO 8
8600   PRINT AT(6+I+I,4);KB$(I)
8610 NEXT I
8990 RETURN : REM ----------------------------------
```

Für Tests des Programms ist es günstig, wenn eine Datei gleich nach dem Programmstart verfügbar ist. Deshalb wird eine Testdatei in den Bereich 50000–65000 geschrieben:

```
50000 REM DATENLISTE: 'EINWOHNERDATEI'  ============
50010 REM Anzahl der erfassten Personen:
50020 DATA 6
50030 REM Datensatz:
50040 REM Merker,Name,Vorname,PKZ,Str.,Nr.,Ort,PLZ,Tel.
50050 REM maximale Zeichenzahl der Komponenten:
50060 REM      1,  14,     14, 12,  14,  4, 14,  4,   7
50070 REM ------------------------------------------
51000 DATA 1,MEIER,GUSTAV,020214413031,W.-WOLFF-STR.,22,
          BERLIN,1110,4984875
51010 DATA 1,MUEHLMANN,HOLGER,211038427384,BOERNITZSTR.,
          7,BERLIN,1130,5549344
51020 DATA 1,WEBER,EMIL,151253431337,BERSARINSTR.,34,
          BERLIN,1034,4347889
```

```
51030 DATA 1,KAISERBERG,ROSA,101035529272,NEUE PROMENADE,
      3,BERLIN,1020,-
51040 DATA 1, SCHWARZ,INGE,030855510056,OPPERMANNSTR. ,7,
      BERLIN,1142,5418888
51050 DATA 1,WEBER,ELKE,140449523452,BERSARINSTR. ,34,
      BERLIN,1034,4347889
65000 REM Physisches Ende des Programms ================
```

Diese Daten lassen sich in das Datenfeld DA$(I,J) mit dem Unterprogramm DATEI AUS DATAZEILEN IN FELD LESEN einlesen.

```
12000 REM UP DATEI AUS DATAZEILEN IN FELD LESEN =========
12010 REM Eingangsvariable:
12020 REM  P        - Personenanzahl
12030 REM Ausgangsvariable:
12040 REM  DA$(I,J) - Daten zur i. Person
12050 REM 'lokale' Variable:
12060 REM  I,J      - Laufvariable
12070 REM ------------------------------------------------
12100 WINDOW 0,31,0,39 : PAPER 1 : CLS
12110 PRINT INK 23;AT(10,12); "Bitte warten !!"
12120 RESTORE 50020
12130 READ P
12140 FOR I=1 TO P
12150   FOR J=0 TO 8
12160     READ DA$(I,J)
12170   NEXT J
12180 NEXT I
12190 RETURN : REM  ------------------------------------
```

Sind die aktuellen Daten von einer Magnetbandkassette zu laden, so kann das mit folgendem Programm geschehen:

```
13000 REM UP DATEI LADEN VON KASSETTE ==============
13010 REM Ausgangsvariable:
13020 REM DA$ - Datenfeld mit DIM(20,8)
13030 REM ----------------------------------------
13100 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
13110 LOCATE 3,6 : PRINT "Datei-Eingabe von Kassette"
13120 LOCATE 6,0 : PRINT "(nur moeglich, falls auf MB"
13130 LOCATE 8,0 : PRINT ",DATEI' mit DIM(20,8) vorliegt)"
13140 LOCATE 12,3: PRINT "Kassette auf Start von ,DATEI'"
13150 LOCATE 14,3: PRINT "stellen u. MBG mit Wiedergabe"
13160 LOCATE 16,16: PRINT "starten !"
13170 LOCATE 28,3 : PRINT "Wenn Pilotton, dann ";CHR$(141)
13180 LOCATE 30,7 : PRINT "zurueck zum Menue mit^"
13190 I$=INKEY$ : IF I$="" THEN 13190
13200 IF I$=CHR$(94) THEN 13990
13210 CLOAD*"DATEI";DA$
13220 CLS
```

```
13230 LOCATE 3,7: PRINT "Datei wurde gelesen!"
13240 LOCATE 29,5 : PRINT "weiter mit ";CHR$(141)
13250 I$=INKEY$ : IF I$="" THEN 13250
13990 RETURN
```

In der bereitgestellten Datei kann man „blättern", wenn der folgende Unteralgorithmus in Form eines Struktogramms auch in BASIC realisiert ist (Abb. 4.1/10).
Im Unteralgorithmus BLAETTERN IN DER DATEI ist die Ausgabe eines Datensatzes im Prinzip enthalten, so daß hier auf die Darstellung in Form eines Struktogramms oder Unterprogramms verzichtet werden kann. Die datensatzweise Eingabe einer Datei über Tastatur ist in folgendem Struktogramm dargestellt (Abb. 4.1/11).
Das akustische Signal soll die Bereitschaft zur Eingabe eines nächsten Datensatzes signalisieren. Es dient der Unterstützung des optischen Ablaufs des Dialogs.

(UA BLÄTTERN IN DER DATEI)

(Zweck: Formulare vorwärts und rückwärts blättern
Eingangsvariable: PE – Anfangsposition Blättern
P – max. Formularanzahl
‚lokale' Variable: KZ – Kennzahl
U, O – untere, obere Grenze
Unterprogramme: FORMULARRAHMEN
IN FORMULAR EINTRAGEN
KENNZAHLEINGABE)

BEGINN

Normalfenster : Fenster löschen

Ausgabe
ab Position (10,0);„Mit welchem Datensatz beginnen"
ab Position (12,0);„Datensatznr. zwischen 1 und";P

U := 1 : O := P : BA := 6 (Betriebsartnummer)

RUFE KENNZAHLEINGABE

PE := KZ : Fenster löschen

BE$:=„Befehle: 0, ↓, ↑"

RUFE FORMULARRAHMEN

Ausgabe
ab Position (26,0); „Blättern beenden: 0"
ab Position (28,0); „Vorwärts blättern: ↓"
ab Position (29,0); „Rückwärts blättern: ↑"

Fenster 8, 24, 17, 38 : Fenster löschen

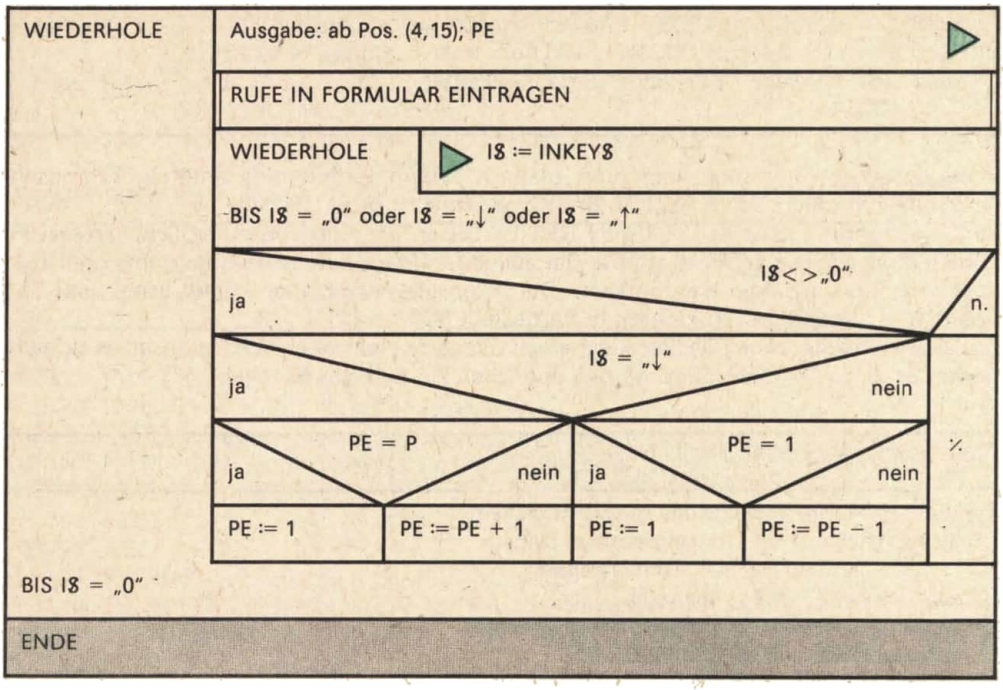


Abb. 4.1/10: Struktogramm zu „BLÄTTERN IN DATEI"

(UA DATEIEINGABE ÜBER TASTATUR)	
(ZWECK: Ersteingabe von Datensätzen einer Datei	
Eingangsvariable:	P – max. Personenzahl
Ausgangsvariable:	DA$(PE,I) – Datensätze; PE = 1,2,...,P I = 0, 1, ..., 8
‚lokale' Variable:	PE – Personenzähler I – Komponentenzähler
Unterprogramme:	FORMULARRAHMEN EINGABEMASKEN EINGABE EINES DATENSATZES ÜBER TASTATUR)
BEGINN	
BE$:= „Befehle: ↵, ←, →, @"	
RUFE FORMULARRAHMEN	
Ausgabe ab Pos. (28,0); „Komponenteneingabe abschließen: ↵" ab Pos. (29,0); „Vorzeit. Beenden der Eingabe: @"	
Fenster 7, 14, 17, 38	

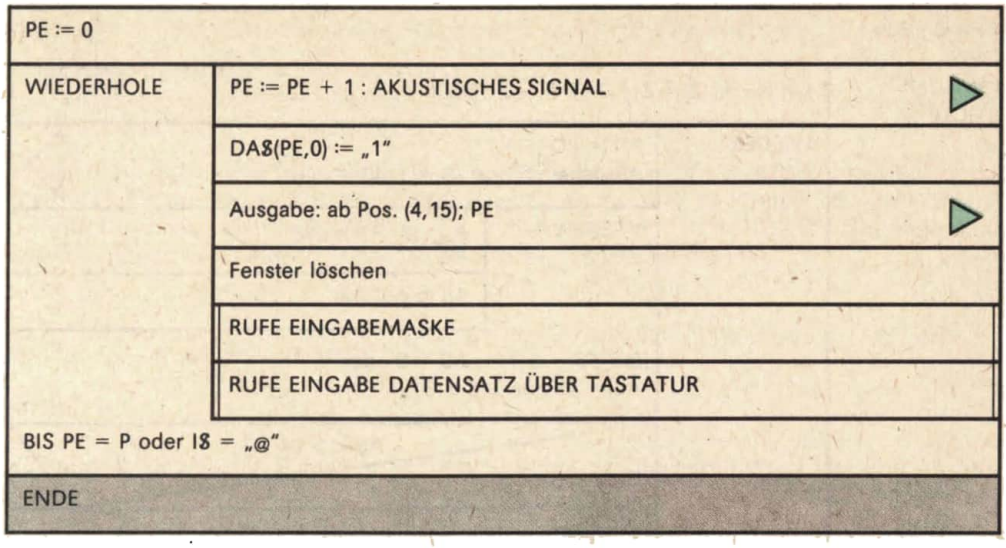


Abb. 4.1/11: Struktogramm „DATEIEINGABE ÜBER TASTATUR"

Der eigentliche, etwas kompliziertere Eingabevorgang eines Datensatzes in die vorgegebenen Masken ist in einem weiteren Unteralgorithmus ausgelagert. Damit kann dieser Unteralgorithmus gleich so konzipiert werden, daß er beispielsweise auch im Unteralgorithmus AENDERN EINES DATENSATZES verwendbar ist.
Damit sind alle nach der Problemanalyse vorgesehenen Ein- und Ausgabemöglichkeiten beschrieben.

Realisieren Sie die Umsetzung der Unteralgorithmen BLÄTTERN IN DER DATEI, DATEIEINGABE ÜBER TASTATUR und EINGABE EINES DATENSATZES ÜBER TASTATUR in BASIC!

(UA EINGABE EINES DATENSATZES ÜBER TASTATUR)

(ZWECK: Eingabe eines Datensatzes zur PE. Person
und Ausgabe des aktuellen Inhalts des Datensatzes
für die PE. Person in Form des Bildes eines
Formulars.

Eingangsvariable:	PE – Personennummer
	M(I) – Maskenlänge I. Komponente
Ausgangsvariable:	DA$(PE,I)– Komponenten PE. Person
‚lokale' Variable:	I – Komponentenzähler
	Z – Zeilenzähler
	PO – Positionszähler
	AS – ASCII-Zahl zu I$
	I$ – ZK Tastaturabfrage
	EI$ – Hilfs-ZK zur Eingabe
	L – Länge von EI$)

BEGINN

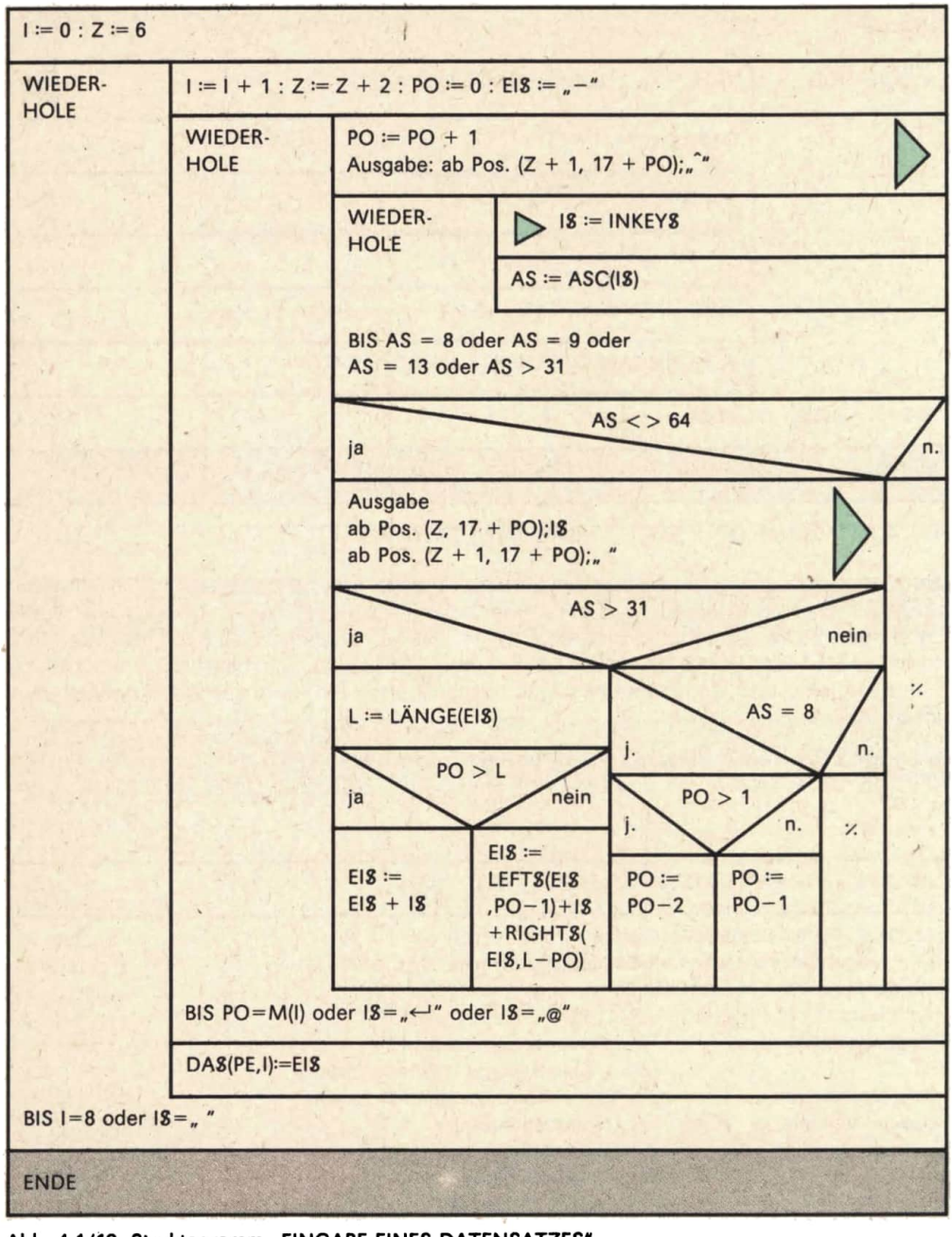


Abb. 4.1/12: Struktogramm „EINGABE EINES DATENSATZES"

Die Teilalgorithmen An- bzw. Einfügen, Ändern und Löschen eines Datensatzes, die innerhalb des Verwaltens der Datei vorgesehen sind, können im wesentlichen aus den bereits entwickelten Algorithmen aufgebaut werden, so daß folgender Auftrag formuliert werden kann:

Entwickeln Sie Struktogramme und Programme zu den Teilalgorithmen

- AN- UND EINFUEGEN EINES DATENSATZES,
- AENDERN EINES DATENSATZES und
- LOESCHEN EINES DATENSATZES.

Beachten Sie dabei, daß das Teilfeld DA$(PE,0) Merker beinhaltet, die auf 1 gestellt sind, falls Daten für die PE. Person erfaßt wurden und andernfalls auf 0 steht. Damit können z. B. die Datensätze markiert werden, die gelöscht wurden, so daß dieser Speicherplatz für ein Einfügen eines neuen Datensatzes wieder zur Verfügung steht.

4.1.3. Verarbeiten der Datei

Sortieren

Wichtige Aufgaben der Dateiverarbeitung bestehen im Sortieren der Datei und im Suchen in der Datei. Man kann die Datensammlung unter verschiedenen Gesichtspunkten sortieren. In der Einwohnerdatei könnte man z. B. nach den Datensatzkomponenten Name, PLZ oder Straße sortieren. Die Komponenten, nach denen sortiert wird, heißen auch Schlüsselbegriffe. Im Prinzip kann man nach jeder Komponente, nach jedem Schlüsselbegriff, sortieren. Es gibt unterschiedliche Sortierstrategien. Bei der folgenden einfachen Sortierstrategie wird davon ausgegangen, daß die Datenmenge nur aus Daten vom Typ Zahl besteht und *n* Elemente umfaßt, die in einem eindimensionalen Feld A abgelegt sind. Die Feldelemente sind der Größe nach vom kleinsten bis zum größten zu sortieren. Eine grobe Überlegung zur Sortierstrategie besteht im folgenden:
Die kleinste aller in A(1), ..., A(*n*) abgespeicherten Zahlen wird auf A(1) gebracht, die zweitkleinste auf A(2) usw. Das heißt, daß die *i*-t kleinste Zahl nach A(*i*) zu bringen ist, wobei durch das Umspeichern keine Informationen verlorengehen dürfen.

Zuerst wird geklärt, wie die kleinste Zahl auf A(1) zu bringen ist. In einem ersten Schritt werden die Werte A(1) und A(2) verglichen. Ist der Wert von A(2) kleiner als der von A(1), so sind die Werte zu vertauschen. Es steht nach diesem ersten Schritt die kleinere der beiden Zahlen in A(1).
Im zweiten Schritt werden die Werte von A(1) und A(3) verglichen. Nur im Falle, daß der Wert von A(3) kleiner als der von A(1) ist, wird der Austausch vorgenommen. Nach diesem zweiten Schritt steht die kleinste der drei Zahlen von A(1), A(2) und A(3) in A(1). Die folgenden Schritte stellen den Vergleich zwischen A(1) und A(*k*) mit *k*=4, 5, ..., *n* dar, so daß sich nach n Schritten die kleinste Zahl aller betrachteten Datenwerte in A(1) befindet. Dieses Vorgehen spiegelt der Ausschnitt eines Struktogramms wider.

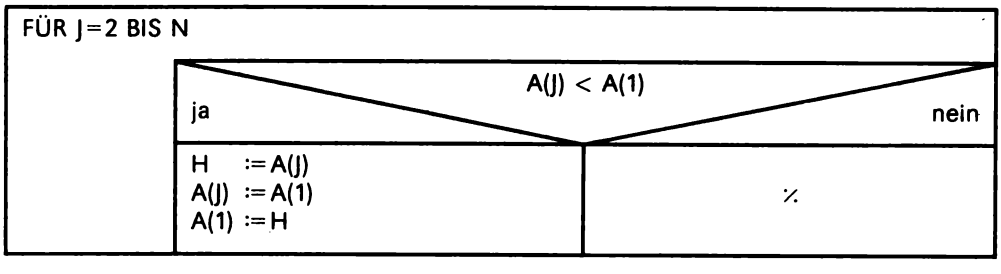


Abb. 4.1/13: Struktogrammausschnitt „Kleinster Wert nach A(1)"

Gesamtziel war aber, die *i*-t kleinste Zahl nach A(*i*) zu bringen. Wenn man erreicht hätte, daß die kleinste Zahl auf A(1), die zweitkleinste Zahl auf A(2), ... und schließlich die (*i*-1)-t kleinste Zahl auf A(*i*-1) abgespeichert sind, dann ist die *i*-t kleinste Zahl nur noch unter den Werten der Variablen A(*i*), ..., A(*n*) zu suchen. In diesem Bereich ist sie die kleinste Zahl, so daß obige Überlegung übertragen werden kann.

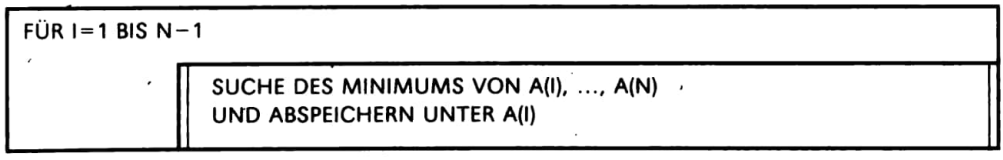


Abb. 4.1/14: Struktogrammausschnitt „Umlagern kleinster Wert"

Daraus erhält man das Struktogramm für das MINIMUM-Sortieren von Zahlen:

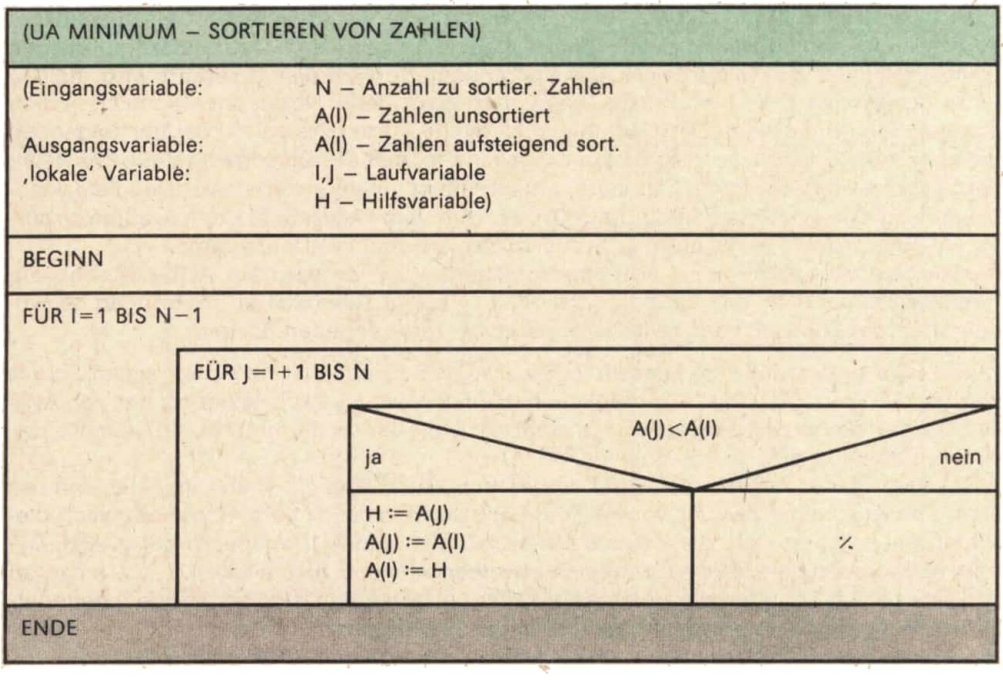


Abb. 4.1/15: Struktogramm „MINIMUM-SORTIEREN"

Leicht läßt sich dieser Algorithmus in BASIC umsetzen, wobei im folgenden die sonst üblichen Informationen über Eingangs-, Ausgangs- und ‚lokale' Variable nicht mit angegeben sind.

```
500 REM UA=== MINIMUM - SORTIEREN VON N ZAHLEN ========
600 FOR I=1 TO N-1
610   FOR J=1 TO N
620     IF NOT(A(J) <A(I)) THEN 640
```

```
630         H=A(J) : A(J)=A(I) : A(I)=H
640    NEXT J
650 NEXT I
690 RETURN : REM ---------------------------------------
```

Natürlich kann man in der Zeile 620 das NOT vermeiden

```
620      IF A(J)  >= A(I) THEN 640
```

Um dieses Programm austesten zu können, wird ein Rahmenprogramm erstellt, das die Erzeugung von *n* unsortierten Zahlen und die Ausgabe der *n* sortierten Zahlen umfaßt. Eine lineare Anordnung von Datenelementen heißt auch Liste, so daß das Testprogramm folgende Form hat:

```
100 REM TESTRAHMEN FUER SORTIERPROGRAMME ================
110 WINDOW 0,31,0,39 : COLOR 1,7 : CLS
120 INPUT "ANZAHL DER ZAHLEN N= ";N
130 DIM A(N)
140 GOSUB 200 : REM - > Erzeugen einer Liste unsort. Zahlen
150 GOSUB 500 : REM - > Sortieren
160 GOSUB 700 : REM - > Ausgabe der sortierten Liste
190 END : REM -------------------------------------------

200 REM === ERZEUGEN EINER LISTE VON N UNSORT. ZAHLEN ===
210 REM Eingangsvariable: N    - Anzahl zu erstell. Zahlen
220 REM Ausgangsvariable: A(I) - Liste der unsort. Zahlen
230 REM 'lokale'Variable: I    - Laufvariable
290 REM -----------------------------------------------
300 FOR I=1 TO N
310    A(I)=INT(RND(1)*1000)
320    PRINT A(I)
330 NEXT I : BEEP
390 RETURN : REM ----------------------------------------

500 REM ====== MINIMUM - SORTIEREN VON N ZAHLEN =========
510 REM Eingangsvariablen:N    - Anzahl zu sortier. Zahlen
520 REM                   A(I) - Zahlen unsortiert
530 REM Ausgangsvariable: A(I) - sortierte Liste
540 REM 'lokale'Variablen:I,J  - Laufvariable
550 REM                   H    - Hilfsvariable
590 REM -----------------------------------------------
600 FOR I=1 TO N-1
610    FOR J=I+1 TO N
620       IF A(J)  >= A(I) THEN 640
630          H=A(J) : A(J)=A(I) : A(I)=H
640    NEXT J
650 NEXT I
690 RETURN : REM ----------------------------------------

700 REM === AUSGABE DER SORTIERTEN LISTE VON N ZAHLEN ===
```

```
710 REM Eingangsvariablen: N   - Anzahl auszugebender Zahlen
720 REM                    A(I)- sortierte Liste
730 REM 'lokale'Variable: I   - Laufvariable
790 REM ------------------------------------------------
800 CLS : PRINT : PRINT
810 FOR I=1 TO N
820    PRINT A(I),
830 NEXT I : BEEP
890 RETURN : REM ---------------------------------------
```

In den Programmzeilen 330 und 830 sind akustische Signale eingebaut, um eine Hilfe zu haben, wenn man die Sortierzeit mit einer Stoppuhr messen will. Die folgenden Testergebnisse zeigen, daß die Sortierzeit durchaus nicht proportional zur Anzahl der zu sortierenden Zahlen ist:

N	Sortierzeit (per Hand gestoppt) in s
10	3
30	11
40	17
50	25
80	59
200	346

Dieses Zeitverhalten ist erklärbar. Im ersten Schritt werden $n-1$ Vergleiche notwendig, im zweiten Schritt $n-2$ usw. so daß insgesamt

$$n + n - 1 + n - 2 + \ldots + 1 = n * \frac{n+1}{2}$$

Vergleiche notwendig sind. Da n^2 für wachsendes n wesentlich schneller wächst als n für $n > 1$, vernachlässigt man den linearen Anteil und sagt, daß die Rechenzeit in etwa proportional zu $n*n/2$ ist. Damit ist auch einsichtig, daß bei Verdoppelung der Anzahl der zu sortierenden Zahlen (falls n nur hinreichend groß ist, damit die Sortierzeit im wesentlichen durch die Anzahl der Vergleiche bestimmt wird) die Rechenzeit über das Vierfache anwächst.

Das Programm ist so reich an REM-Zeilen und Leerzeichen zur optisch günstigen Gliederung, daß man vermuten könnte, daß ein wesentlicher Zeitgewinn eintritt, wenn man alle REM-Zeilen und überflüssigen Leerzeichen eliminieren würde. Tests ergaben, daß nur unwesentliche Zeitverkürzungen dadurch erreichbar sind. Erst bei 200 zu sortierenden Zahlen trat eine Zeitverkürzung von 346 auf 341 Sekunden ein. Das sind etwa 1,5%. Schreibt man das Programm so kurz wie möglich in den entscheidenden Teilen, so erhält man eine etwas größere Zeitersparnis, im Testfall für 200 Zahlen ca. 23 Sekunden. Das sind etwa 6,6%. Aber das Programm ist dann kaum noch lesbar:

```
1 WINDOW0,31,0,39:COLOR1,7:CLS
2 INPUT"Anzahl der Zahlen n= ";N
3 DIMA(N)
4 GOSUB5:GOSUB6:GOSUB8:END
5 FORI=1TON:A(I)=INT(RND(1)*1000:PRINTA(I),:NEXT:BEEP:RETURN
6 FORI=1TON-1:FORJ=I+1TON:IFA(J) < A(I)THENH=A(J):A(J)=A(I):
  A(I)=H
7 NEXTJ,I:RETURN
8 CLS:PRINT:PRINT:FORI=1TON:PRINTA(I);:NEXT:BEEP:RETURN
```

Alle guten Regeln, die bisher der Übersichtlichkeit halber eingehalten wurden, sind in diesem Programm vergessen und doch ist der Rechenzeitgewinn nicht sehr groß. Solange man mit einem BASIC-Interpreter die Programmabarbeitung realisiert, werden keine nennenswerten Zeitreserven bei Beibehaltung der Sortierstrategie erreicht. Für den Nutzer ergab sich keine wesentliche Verbesserung durch das Schrumpfen des ordentlich lesbaren Programms auf diese 8 Zeilen-Variante. Will man schneller sortieren, so sollte man die interpretative Abarbeitung verlassen oder nach einer leistungsfähigeren Sortierstrategie suchen.

Da Sortieralgorithmen volkswirtschaftlich von großer Bedeutung sind, lohnt sich intensives Nachdenken über schnellere Sortieralgorithmen. Es gibt auch schon eine ganze Reihe unterschiedlich leistungsstarker Sortierverfahren, die hier aber nicht dargestellt werden können.

Alle Daten in der Einwohnerdatei sind als Zeichenketten erfaßt. Für Zeichenketten gibt es ebenfalls die Relationen <, <=, =, >=, > und <>. Eine Zeichenkette Z1$ „steht vor" einer Zeichenkette Z2$, wenn bezüglich der zu jedem Zeichen gehörenden ASCII-Zahl die Kleinerrelation gilt. Das heißt, ADAM „steht vor" EVA, weil beim zeichenweisen Vergleich bereits beim ersten Zeichen für die zugehörigen ASCII-Zahlen die Kleinerrelation erfüllt ist:
ASC("A")=65 ASC("E")=69.
Aber adam „steht nicht vor" EVA, weil ASC("a")=97 > ASC("E")=69 ist. Diese Vergleiche beziehen sich nicht nur auf das erste Zeichen einer Zeichenkette, sondern auf alle Zeichen. Der Vergleich wird solange durchgeführt, bis Zeichen auf gleicher Position unterschiedlich sind. Sind die zu vergleichenden Zeichenketten unterschiedlich lang, so wird die kürzere durch Anhängen von Leerzeichen auf gleiche Länge gebracht. Durch diese Anmerkungen ist geklärt, daß alphabetisch geordnet werden kann.

Das Sortierprogramm MINIMUM-SORTIEREN VON ZAHLEN ist für Zeichenketten leicht anpaßbar. Die Zahlenvariablen A(I) und H werden durch A$(I) und H$ ersetzt. Um für Testzwecke eine genügend große Anzahl zu sortierender Daten zu haben, werden in DATA-Zeilen z. B. 50 Vornamen eingetragen, die dann mit READ gelesen werden.

```
50000 REM ===== VORNAMEN-DATEI ======================

50010 DATA ADAM,OTTO,WOLFGANG,HANS,PETER,EVA,INGE,INGRID,
           KATJA,GISELA,HANSI,HANS-PETER
50020 DATA HANSPETER,KARLHEINZ,KARL,GABI,CATHRIN,PETRA,
           ERNA,GUENTER,DIETER,LOTHAR
50030 DATA ULF,ANNA,SUSI,IRMGARD,CHRISTA,EV,JUERGEN,WERNER
50040 DATA JOE,MARTIN,SANDRA,ANJA,MARIA,MIRIJAM,MAREN
50050 DATA KLAUS,HEINZ,SVEN,SIEGFRIED,FRANK,BIRGIT,JANA,
           INES,UTE,SABINE,ANDREA
65000 REM Physisches Ende des Programms ----------------
```

Nur das Programm zum Eingabemodul hat sich etwas verändert:

```
200 REM === ERZEUGEN EINER LISTE UNSORT. ZEICHENKETTEN ===
300 FOR I=1 TO N
310    R=INT(RND(1)*50)+1
320    RESTORE
330    FOR J=1 TO R
340       READ H$
```

```
350     NEXT J
360     A$(I)=H$ : PRINT H$,
370 NEXT I
380 BEEP
390 RETURN : REM ----------------------------------------
```

Nun soll das MINIMUM-SORTIEREN VON N ZEICHENKETTEN, das auch den Namen AUSWAHLSORT trägt, in das Dateiprogramm eingebaut werden. Dazu muß man wissen, nach welchem Schlüsselbegriff zu sortieren ist. Der Unteralgorithmus AUSWAHL DER SCHLUESSELBEGRIFFSNUMMER wird gleich so erstellt, daß er auch zur Schlüsselbegriffsfestlegung für Suchalgorithmen verwendet werden kann (Abb. 4.1/16).
Der eigentliche Sortiervorgang erfolgt dann bzgl. des Schlüsselbegriffs mit der Nummer SB, aber da i. a. der gesamte Datensatz ausgegeben werden soll, müssen alle Komponenten am Austauschvorgang beteiligt werden, sonst würde ein Durcheinander der Datensatzkomponenten eintreten. Der Sortiervorgang wird gleich als BASIC-Programm angegeben:

```
32700 REM ====== EIGENTLICHER SORTIERVORGANG ===========
32710 CLS : PRINT INK 23;AT(10,10);"Es wird sortiert !!"
32720 FOR I=1 TO P-1
32730    FOR J=I+1 TO P
32740       IF DA$(J,SB)  >= DA$(I,SB) THEN 32800
32750          FOR KO=1 TO 8
32760             H$       = DA$(J,KO)
32770             DA$(J,KO)= DA$(I,KO)
32780             DA$(I,KO)= H$
32790          NEXT KO
32800    NEXT J
32810 NEXT I
32900 RETURN : REM ------------------------------------
```

Den Programmtest innerhalb des bisher entwickelten Dateiprogramms kann man wie folgt durchführen.
Nach dem Programmstart wird zum Untermenü 1 ‚Anlegen der Datei' gegangen. Dort wird die Kennziffer 2 ‚Einlesen aus DATA-Zeilen' gewählt. Danach liegt im Feld DA$(I,J) eine zu sortierende Datei vor. Zunächst muß man in der Menüsteuerung zurück zum Hauptmenü, um von dort aus die Kennziffer 3 ‚Verarbeiten der Datei' zu starten. Im Untermenü 3 wird die Kennziffer 2 ‚Sortieren' gewählt. Der Computer bietet daraufhin das Menü der Schlüsselbegriffe an, woraus z. B. das Sortieren nach Postleitzahlen PLZ veranlaßt werden kann. Nach einem kurzen Augenblick ist der Sortiervorgang abgeschlossen, da in der Testdatei nur 6 Datensätze sind. Geht man über das Hauptmenü zum Untermenü 4 ‚Ausgeben von Daten', dann kann man sich z. B. mittels ‚Blaettern' in der Datei von dem erfolgten Sortiervorgang überzeugen.
Man kann immer wieder nach anderen Schlüsselbegriffen sortieren. Der Effekt, daß die gesamten Datensätze vollständig umgespeichert werden, ist dabei nicht rationell. Deshalb wird nach einer Verbesserung des Vorgehens beim Sortieren gestrebt. Das Umsortieren bedeutet eigentlich nur die Zuordnung der Datensätze zu der festen Datensatznummernfolge 1, 2, 3, ..., P. Aber es reicht für die Ausgabe der nach einem Schlüsselbegriff zu sortierenden Datensätze bereits aus, wenn der Computer gespeichert hat, in welcher Reihenfolge die Datensätze auszugeben sind. Das heißt, es müßte ein Zeiger vorliegen,

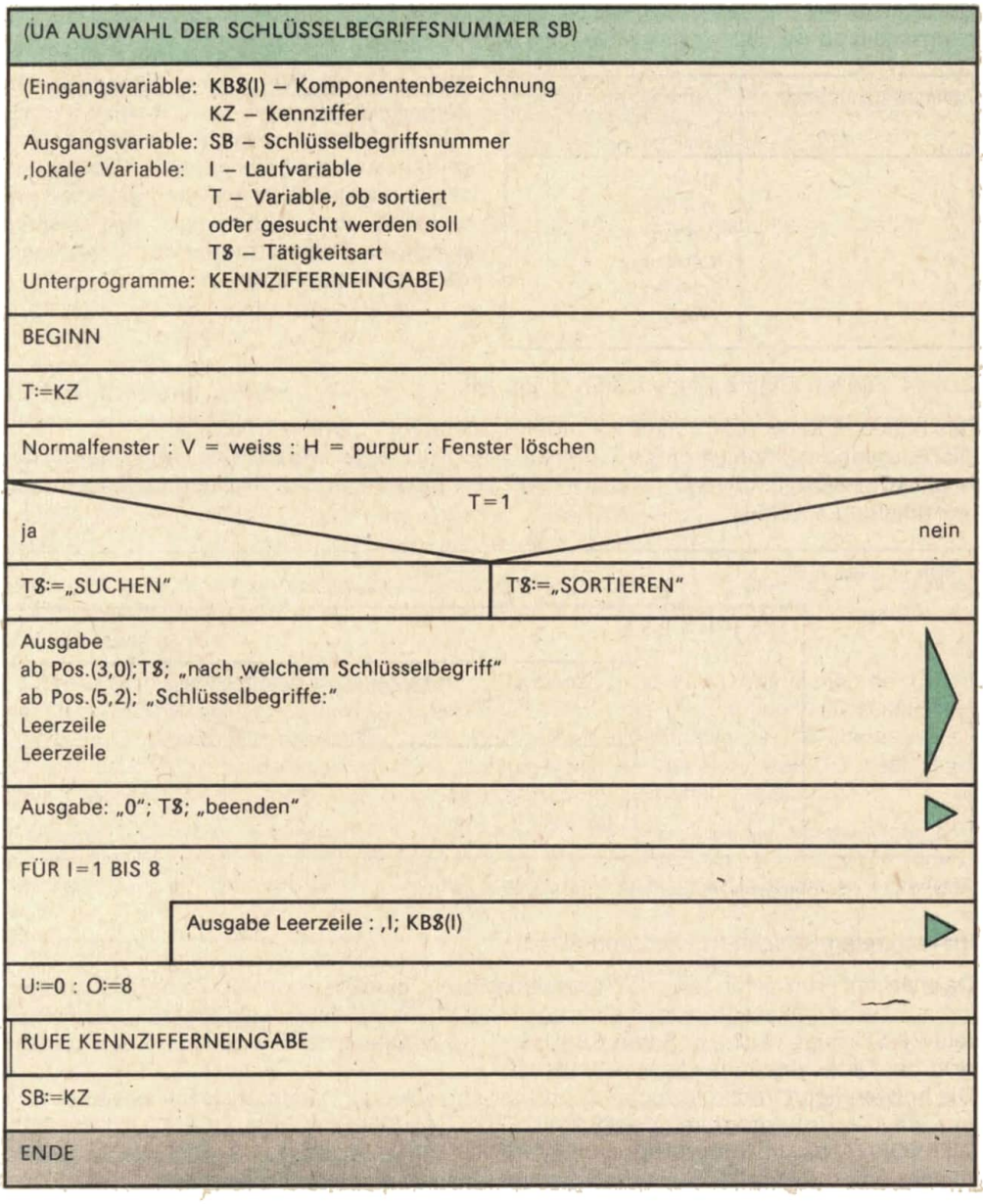


Abb. 4.1/16: Struktogramm „AUSWAHL SCHLÜSSELBEGRIFFSNUMMER"

der jeweils auf den in der sortierten Reihenfolge nächsten Datensatz verweist. Man sagt auch, daß man einen Index – eine Verweisliste – aufbaut, in der die Folge der Indizes der Datensätze entsprechend der Sortierfolge eingetragen ist.

Das Prinzip sei am Sortieren nach Namen bezogen auf die Testdatei erläutert. Die Ausgangssituation ist:

Datensatznummer	Name (1. Komponente)
1	Meier
2	Mühlmann
3	Weber
4	Kaiserberg
5	Schwarz
6	Weber

Ziel ist es, die Werte im eindimensionalen Zeigerfeld Z(*i*) mit *i*=1, ..., 6 so zu bestimmen, daß sich bezüglich des Schlüsselbegriffs Name eine alphabetische Sortierung ergibt. Letztendlich muß im Zeigerfeld die alphabetische Reihenfolge der Namen durch die Datensatznummer mit wachsendem Index markiert sein:

Z(1)=4 Z(2)=1 Z(3)=2 Z(4)=5 Z(5)=3 Z(6)=6.

Das bedeutet Kaiserberg „steht vor" Meier „steht vor" Mühlmann usw.
Die Ausgangssituation ist durch Z(*i*)=*i* für *i*=1, 2, ..., 6 gekennzeichnet. Ausgehend vom MINIMUM-SORTIEREN VON ZEICHENKETTEN erhält man den Struktogrammausschnitt in Abbildung 4.1/17.

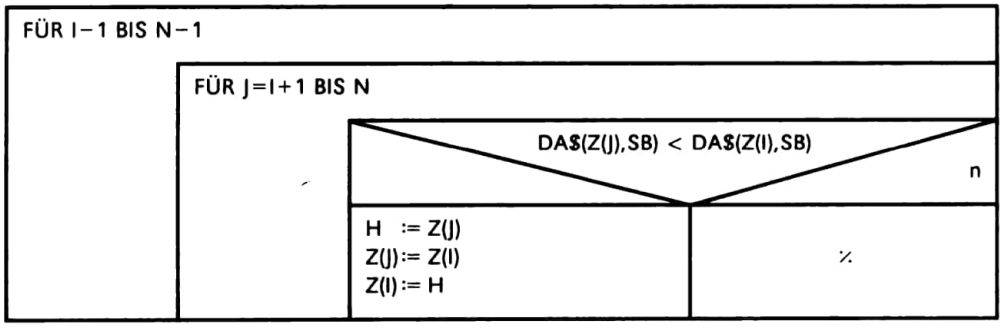


Abb. 4.1/17: Struktogrammausschnitt „Indexsortieren"

Im konkreten Beispiel ist N=6 und SB=1.

Da man im Prinzip für jeden Schlüsselbegriff ein eindimensionales Zeigerfeld anlegen kann, ist eine weitere Verallgemeinerung möglich. Es wird ein zweidimensionales Zeigerfeld Z(I,SB) mit I=1, 2, ..., P und SB=1, 2, ..., 8 angelegt, so daß zu jedem Schlüsselbegriff der Datei eine Indexdatei vorliegt.
Die notwendigen Veränderungen des bisher entwickelten Dateiprogramms beziehen sich nur auf die Unterprogramme PARAMETERINITIALISIERUNG und EIGENTLICHER SORTIERVORGANG. Im Unterprogramm PARAMETERINITIALISIERUNG wird das Zeigerfeld dimensioniert und jedes Element bekommt einen geeigneten Anfangswert:

```
3640 DIM DA$(P,8), Z(P,8)
3641 FOR I=1 TO P
3643    FOR J=1 TO 8
3645       Z(I,J)=I
3647    NEXT J
3649 NEXT I
```

Im Unterprogramm EIGENTLICHER SORTIERVORGANG sind wenige, aber bedeutsame Veränderungen vorzunehmen.

```
32700 REM ===== EIGENTLICHER SORTIERVORGANG ============
32710 CLS : PRINT INK 23;AT(10,10);" Es wird sortiert !!"
32720 FOR I=1 TO P-1
32730    FOR J=I+1 TO P
32740       IF DA$(Z(J,SB),SB) >= DA$(Z(I,SB),SB) THEN 32800
32760          H       = Z(J,SB)
32770          Z(J,SB) = Z(I,SB)
32780          Z(I,SB) = H
32800    NEXT J
32810 NEXT I
32990,RETURN : REM ------------------------------------
```

Zu vermerken ist, daß diese Art des Sortierens die Datensätze in der Datei an den bisherigen Positionen unverändert läßt, so daß man sich über das Ergebnis des Sortierens nicht mittels ‚Blaettern' in der Datei informieren kann. Deshalb wird im Untermenü 4 eine weitere Ausgabeform geschaffen: 4 Ausgabe: Datensaetze sortiert.

Entwickeln Sie einen Unteralgorithmus AUSGABE:DATENSAETZE SORTIERT. Schreiben Sie dazu ein BASIC-Programm und binden Sie es in das bisherige Dateiprogramm von 44000–44900 ein!

In der Testdatei kommt der Name Weber zweimal vor. Da in den DATA-Zeilen der Datensatz 3 WEBER, EMIL, ... vor dem Datensatz 6 WEBER, ELKE, ... steht, wird der Datensatz 3 vor dem Datensatz 6 ausgegeben, wenn nur nach dem Schlüsselbegriff Name sortiert wurde. Weil ELKE alphabetisch vor EMIL steht, würde man von Hand den Datensatz 6 vor dem Datensatz 3 einordnen. Wie kann man dies aber maschinell realisieren?

Eine erste Lösungsidee sei wieder am Beispiel erläutert. Beim Vergleich von Komponenten nach dem Schlüsselbegriff Name wird zusätzlich noch auf Gleichheit geachtet und falls diese vorliegt, wird noch nach dem Schlüsselbegriff Vorname verglichen. Im Struktogramm ist das wie folgt dargestellt:

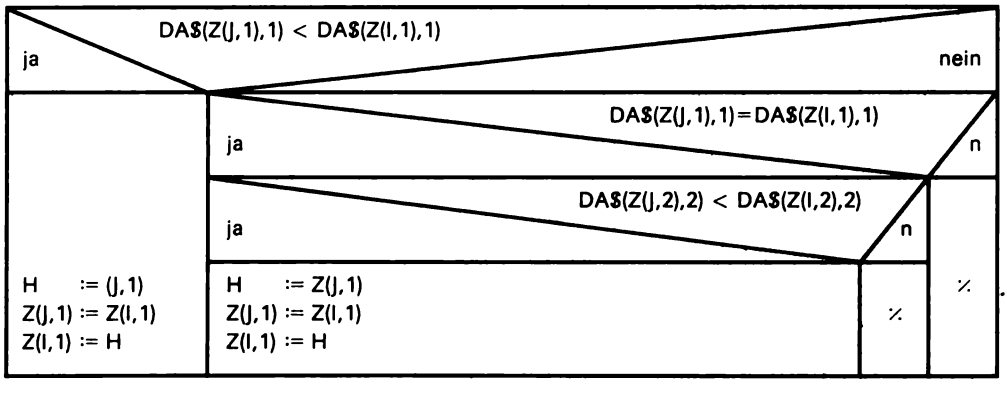


Abb. 4.1/18: Struktogrammausschnitt „Indexsort mit 2 Begriffen"

Aber es geht auch noch einfacher. Zeichenketten sind verkettbar und die alphabetische Reihenfolge ergibt sich aus dem zeichenweisen Vergleich von links nach rechts. Wenn

also die VORNAMEN durch Verkettung an die NAMEN angehängt werden, dann erfolgt die Sortierung zuerst nach den Namen und bei Namensgleichheit weiter nach den Vornamen. Die Programmänderung besteht nur im Ändern der Programmzeile 32740.

```
32740  IF NOT(DA$(Z(J,1),1)+DA$(Z(J,2),2 < DA$(Z(I,1),1)+
          DA$(Z(I,2),2)) THEN 32800
```

Suchen

Das Suchen – auch als Recherchieren bezeichnet – ist für die Dateiverarbeitung von großer Bedeutung. Dabei ist der Suchbegriff oft kein vollständiges Datum. Sucht man die Anschrift einer Person und weiß nur noch, daß der Name auf „mann" endet, dann wird man alle Datensätze haben wollen, in deren Namen die 4 Buchstaben „mann" vorkommen. Es ist unter einem solchen Gesichtspunkt wichtig, den Suchprozeß so zu organisieren, daß nach Festlegung des Schlüsselbegriffs, z. B. Name oder Vorname oder PKZ, die Möglichkeit besteht, eine Teilzeichenkette einzugeben, nach deren Vorkommen in allen Eintragungen zum Schlüsselbegriff gesucht wird. In BASIC ermöglicht die Funktion

```
INSTR(zeichenkette1,zeichenkette2)
```

die ‚zeichenkette1' in der ‚zeichenkette2' zu suchen. Die einfachste Art des Suchens besteht darin, alle Eintragungen zum Schlüsselbegriff von vorn bis hinten nach dem Suchbegriff zu durchmustern. Dieses lineare Suchen ist dann angebracht, wenn man keinerlei Vorinformationen über Eintragungen besitzt.
Das lineare Suchen kann mit dem Unteralgorithmus in Abbildung 4.1/19 erfolgen.

(UA LINEARES SUCHEN)	
(ZWECK: Alle Datensätze werden nach einem Suchbegriff in einer Datensatzkomponente durchmustert. Gefundene Datensätze werden ausgegeben.	
Eingangsvariable:	KZ – Kennzahl P – Personenanzahl DA$(PE,J) – Datenfeld KB$(J) – Komponentenbezeichnung
‚lokale' Variable:	T – Tätigkeitsart PE – Personenzähler F – Merker für Finden SU$ – Suchbegriff SB – Schlüsselbegriffsnummer
Unterprogramme:	AUSWAHL SCHLÜSSELBEGRIFFSNUMMER SB SUCHBEGRIFF SU$ EINGEBEN AUSGABE: PE. DATENSATZ)
BEGINN	
BA:=7 : T:=KZ	

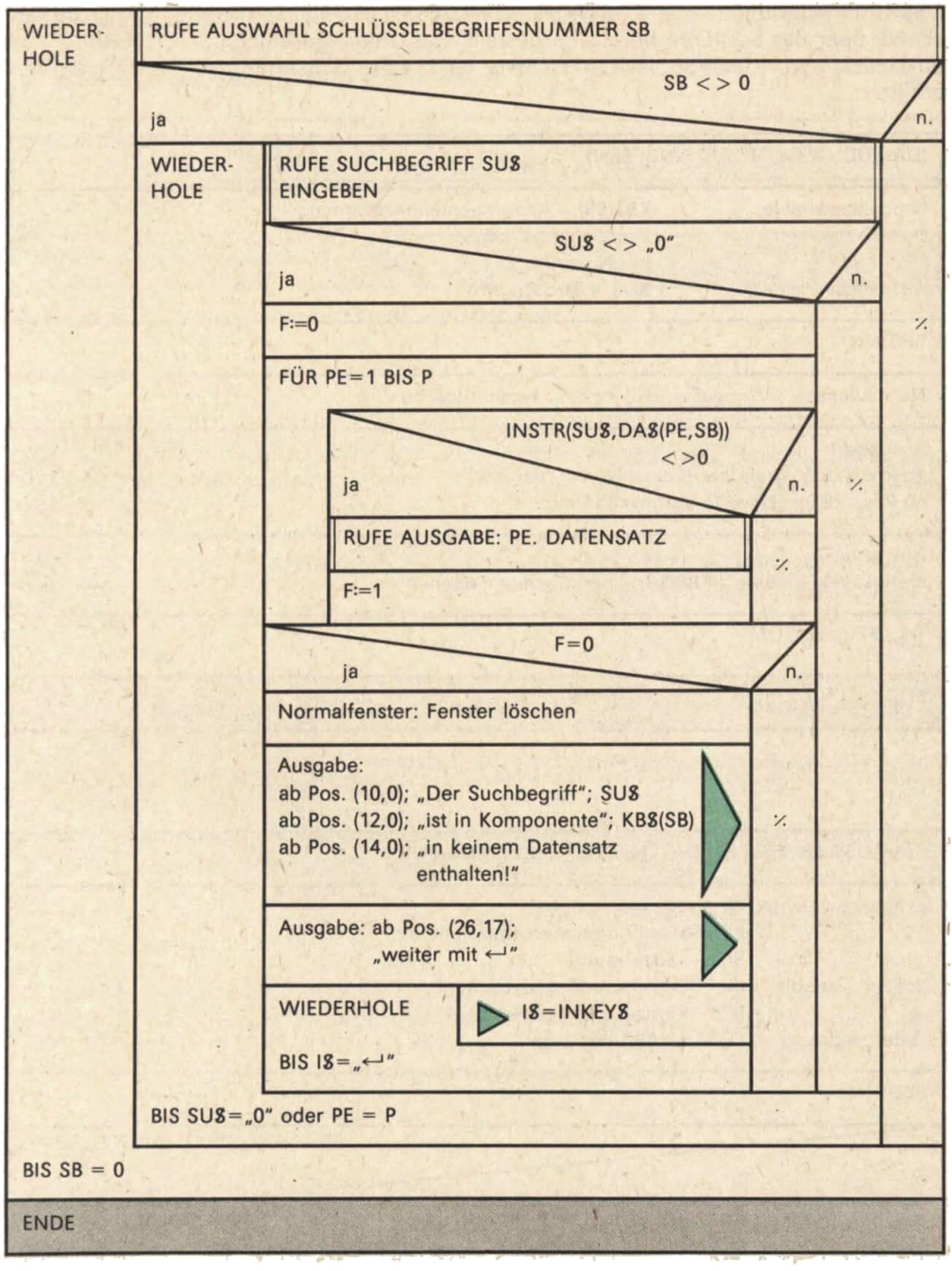


Abb. 4.1/19: Struktogramm „UA LINEARES SUCHEN"

Der Unteralgorithmus zur AUSWAHL SCHLUESSELBEGRIFFSNUMMER SB ist im Abschnitt über das Sortieren bereits dargestellt. Die Unteralgorithmen SUCHBEGRIFF SU$ EINGEBEN und AUSGABE: PE. DATENSATZ sind in den folgenden zwei Struktogrammen erfaßt.

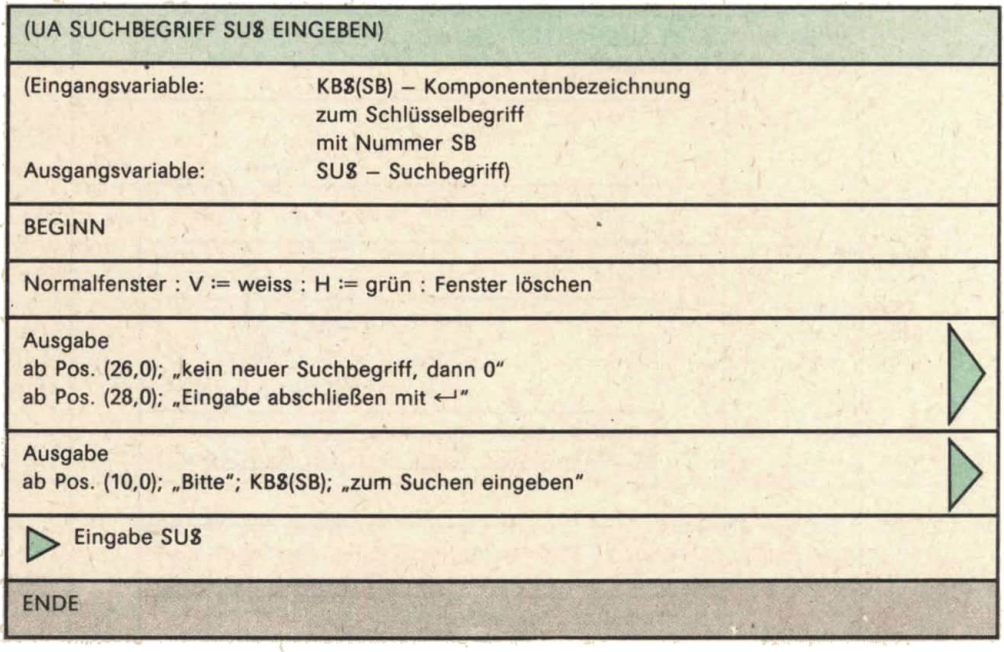
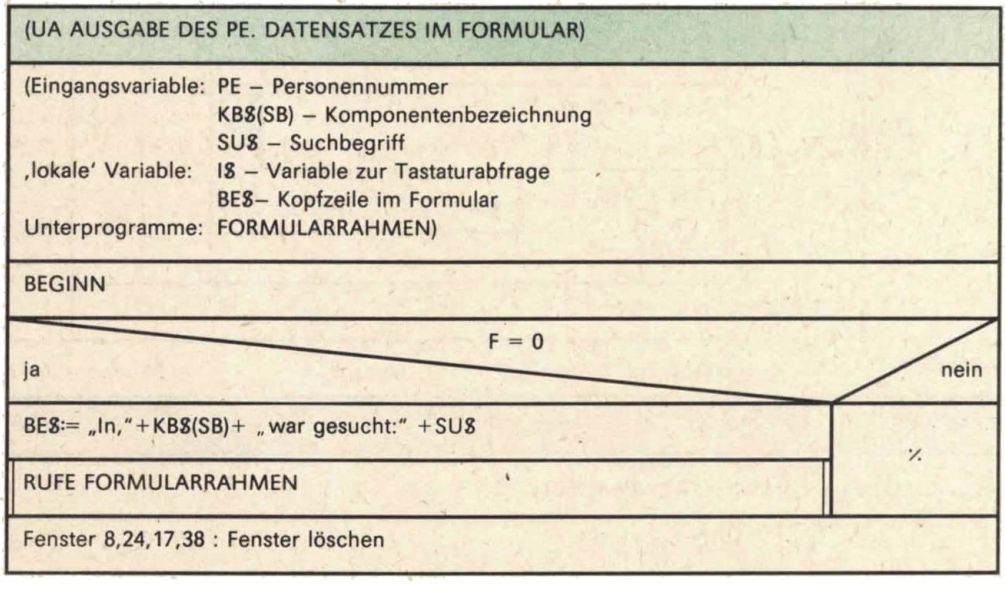


Abb. 4.1/20: Struktogramm „SUCHBEGRIFF SU$ EINGEBEN"



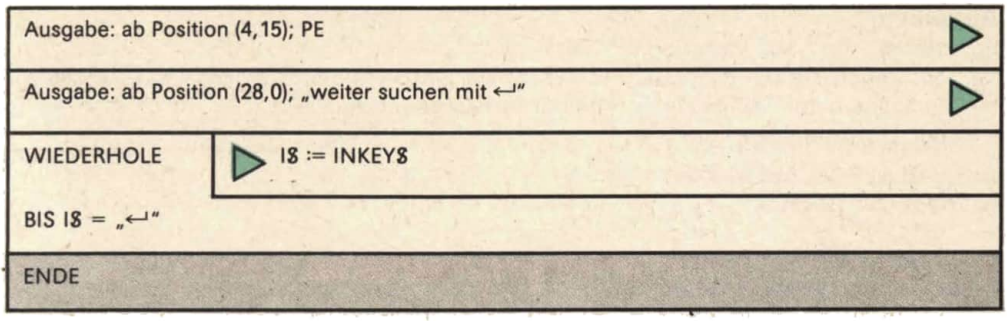


Abb. 4.1/21: Struktogramm „AUSGABE: PE. DATENSATZ"

Zusammenfassung

Computergestützte Dateiarbeit ist volkswirtschaftlich sehr wichtig. Deshalb gibt es leistungsfähige Benutzersysteme, wie z. B. das Relationale Datenbanksystem REDABAS.

In Abhängigkeit von speziellen Aufgaben gibt es in der Dateiarbeit eine große Anzahl unterschiedlicher Teilprobleme. Hier können nur wenige, prinzipielle Lösungsideen dargestellt werden. Die 4 Teilaufgaben

Anlegen,
Verwalten,
Verarbeiten und
Ausgeben

findet man in fast allen Dateiverarbeitungsprogrammen, wobei sie im allgemeinen unterschiedlich gelöst sind.
Für das Anlegen einer Datei sind beispielhaft dargestellt:

Eingabe von Datensätzen über Tastatur,
Einlesen einer (internen) Datei aus DATA-Zeilen und
Einlesen einer Datei von einem externen Speicher, z. B. Magnetbandkassette.

Die Art der Ausgabe der Daten hängt wesentlich vom Verwendungszweck der ausgegebenen Informationen und von den hardwaremäßigen Ausgabemöglichkeiten ab (z. B. Bildschirm, Kassette, Diskette, Drucker, elektronische Schreibmaschine).
Besonders wichtige Dateiverarbeitungsformen bestehen

im Suchen in Dateien und
im Sortieren von Daten.

Von der Effektivität der Such- und Sortieralgorithmen hängt die Effizienz von Dateiprogrammen entscheidend ab. Es gilt, nach immer leistungsfähigeren Such- und Sortieralgorithmen zu streben.

Aufgaben

1. Schreiben Sie ein Programm in den Anweisungsnummernbereich 46000–46990 zur Ausgabe einer Teilliste in Form einer fortlaufenden Ausgabe aller Anschriften der in der Einwohnerdatei erfaßten Personen nach dem Muster:

   HOLGER MUEHLMANN
   BOERNITZSTR. 7
   BERLIN
   1130

2. Schreiben Sie ein Programm im Anweisungsnummernbereich 47000–47990 zur Ausgabe von Datensätzen, wobei für jeden Datensatz jeweils nur eine Druckzeile von 80 Zeichen zur Verfügung steht! Da die Summe der maximal zulässigen Zeichen in der behandelten Einwohnerdatei 85 beträgt und mindestens jeweils ein Trennzeichen zwischen die 8 Komponenten zu setzen ist, entsteht der Zwang, Komponenten zu kürzen. Hierfür bietet sich z. B. der Vorname an, wenn man jeweils nur den Anfangsbuchstaben ausgibt. Die Ausgabe der einzelnen Komponenten erfolge in die Zeilenbereiche mit folgenden Spaltennummern:

Komponente	Spalten
Name	0 bis 13
Vorname	15 bis 16
PKZ	18 bis 29
Straße	31 bis 44
Hausnummer	46 bis 49
Ort	51 bis 64
PLZ	66 bis 69
Telefon-Nummer	71 bis 78

Nehmen Sie auch die notwendigen Veränderungen am Untermenü 4 ‚Ausgeben von Daten' zur Einbindung der Programme zu Aufgaben 1 und 2 vor!

3. Werten Sie den Rationalisierungseffekt, der beim Erzeugen von Adressenaufklebern nach dem Programm zu Aufgabe 1 im Büro des Rates einer Gemeinde entstehen kann!

   Werten Sie die Einsparung an Papier bei Wahrung guter Übersichtlichkeit der Datenausgabe über einen geeigneten Drucker nach dem Programm zu Aufgabe 2!

4. Überlegen Sie, an welchen Stellen Sie am Programm zur Einwohnerdatei Veränderungen vornehmen müßten, um ein Dateiprogramm z. B. für ein Telefonverzeichnis oder für einen Computerprogrammkatalog oder für ein Literaturverzeichnis oder für eine Lagerhaltungsdatei zu bekommen! Dabei seien folgende Datensätze zugrunde gelegt:

   Für ein *Telefonverzeichnis:*
   Name, Wohnort, Vorwahl, Tel.-Nr. (privater Anschluß), Dienststelle, Tel.-Nr. (dienstl. Anschluß);

   Für einen *Programmkatalog:*
   Programmname, Autorenname, Schule, Schulort, Klassenstufe, Datum der Fertigstellung, Zweck des Programms, Standort des Programms;

Für ein *Literaturverzeichnis:*
Autoren, Titel, Verlag, Erscheinungsort, Erscheinungsjahr, Standort, Signatur, Inhaltsstichpunkte;

Für eine *Lagerhaltungsdatei:*
Artikelnummer, Artikelbezeichnung, Standort im Lager, Stückzahl, Einzelpreis.
Wählen Sie aus den vier genannten Beispielen eins aus und schreiben Sie das Einwohnerdateiprogramm mit möglichst geringem Programmieraufwand für den neuen Anwendungszweck um.

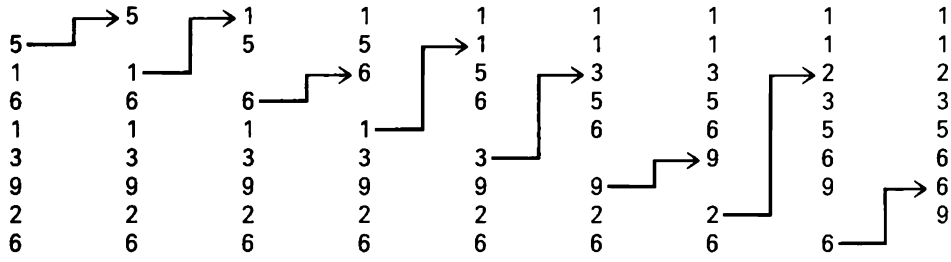
5. Entwickeln Sie ein Struktogramm für das Sortierverfahren Austauschsort! Codieren Sie anschließend Ihren Algorithmus in BASIC!
Die Sortierstrategie wird am Zahlenbeispiel erläutert. Beim Austauschsort (auch Bubble-Sort genannt) wird eine gegebene Zahlenfolge von oben nach unten durchmustert. Stehen benachbarte Elemente nicht der Größe nach geordnet, so werden sie vertauscht. Die erste Durchmusterung sichert, daß das größte Element die richtige Position hat.

Durchmusterungen				
1	2	3	4	5
5 1 6 1 3 9 2 6 → 1 5 6 1 3 9 2 6 → 1 5 1 6 3 9 2 6 → 1 5 1 3 6 9 2 6 → 1 5 1 3 6 2 6	1 5 1 3 6 2 6 → 1 1 5 3 6 2 6 → 1 1 3 5 6 2 6 → 1 1 3 5 2 6 → 1 1 3 2 5	1 1 3 5 2 6 → 1 1 3 2 5 6	1 1 3 2 5 → 1 1 2 3 5	1 1 2 3

Bei der 5. Durchmusterung ist kein Vertauschen mehr notwendig, so daß der Sortierprozeß abgebrochen werden kann. Wird ein Merker V zu Beginn jeder Durchmusterung auf Null gesetzt und nur dann innerhalb einer Durchmusterung auf 1, wenn mindestens eine Vertauschung erfolgte, so kann der Sortierprozeß bereits abgebrochen werden, wenn am Ende einer Durchmusterung V immer noch Null ist. Dann stehen alle benachbarten Elemente in der angestrebten Relation.

6. Schreiben Sie das Programm zur Aufgabe 5 so um, daß anstatt Zahlen Zeichenketten in lexikografischer Reihenfolge sortiert werden können! Vergleichen Sie die Sortiergeschwindigkeit zwischen Austauschsort und Auswahlsort in Experimenten mit unterschiedlich umfangreichen Zahlen- bzw. Zeichenkettenmengen!

7. Entwickeln Sie ein Struktogramm für das Einfüge-Sort! Codieren Sie anschließend den Algorithmus in BASIC!
Die Sortierstrategie gleicht dem Vorgehen eines Kartenspielers, der eine neuaufgenommene Karte durch Vergleich mit seinen bereits vorsortierten Karten an die richtige Stelle steckt. Allerdings soll hier sofort von der gesamten Zahlenfolge (5,1,6,1,3,9,2,6) ausgegangen werden. Die Folge kann man sich in zwei Teilfolgen zerlegt denken, in eine vorsortierte und in die der restlichen noch einzusortierenden Zahlen.



Das Verfahren endet sicher mit dem (n-1). Durchlauf, da dabei das letzte Element eingefügt wird.

8. Entwickeln Sie ein Programm zum Einfüge-Sort für Zeichenketten! Vergleichen Sie die Sortiergeschwindigkeit zwischen Austausch-, Auswahl- und Einfüge-Sort in Experimenten mit unterschiedlich umfangreichen Zahlen- bzw. Zeichenkettenmengen!

9. Mit Hilfe der Indexdateien kann man zu der eigentlichen Einwohnerdatei die Informationen über die Reihenfolge innerhalb der Schlüsselbegriffe bekommen. Überlegen Sie, wie mit der Idee des Einfüge-Sorts die Indexdateien gleich bei der Eingabe von Datensätzen über Tastatur angelegt werden können!

10. Überlegen Sie, wie ein Sortieren nach dem Geburtsdatum in die Einwohnerdatei realisiert werden kann! Das Geburtsdatum ist in der PKZ implizit in der Form ‚ttmmjj' erfaßt, wobei jeweils 2 Stellen für den Tag, den Monat und das Jahr vorgesehen sind. Folgendes Vorgehen liegt nahe: Zuerst wird nach dem Geburtsjahr sortiert. Liegt Gleichheit des Geburtsjahres vor, so ist nach dem Geburtsmonat zu sortieren. Liegt auch dabei noch Gleichheit vor, so ist nach dem Tag zu sortieren. Dieses algorithmisch recht umständliche Vorgehen kann umgangen werden, wenn man durch geeignete Zeichenkettenmanipulationen eine Zeichenkette schafft, in der das Geburtsdatum durch jjmmtt erfaßt ist. Realisieren Sie den Umbau des Geburtsdatums mittels Zeichenkettenfunktion MID$! Erweitern Sie das Auswahlmenü für die Schlüsselbegriffe um ‚9 Geburtsdatum' und modifizieren Sie das Programm zur Einwohnerdatei, so daß ein Sortieren nach dem Geburtsdatum möglich ist!

11. Entwickeln Sie ein Unterprogramm, das einen Zeichenketten-Suchbegriff SU$ so ‚umrechnet', daß er sowohl in lauter Großbuchstaben, als auch in lauter Kleinbuchstaben, als auch in gemischter Schreibweise mit einem Großbuchstaben am Wortanfang vorliegt!

12. Schreiben Sie ein Unterprogramm, das es ermöglicht, alle Männer im Alter von 18–65 Jahren und alle Frauen im Alter von 18–60 Jahren aus der Einwohnerdatei herauszusuchen! Bauen Sie dieses Programm im Untermenü ‚3 Verarbeiten der Datei' ein!

13. Entwickeln Sie ein Programm, das das Ausgeben von Skatkarten an drei Spieler (je 10 Karten) und in den Skat (2 Karten) und das anschließende Ordnen der 10 zufällig erhaltenen Karten eines Spielers simuliert! Dabei wird nach der Kartenfarbe in der Reihenfolge Karo – Herz – Pik – Kreuz und innerhalb einer Kartenfarbe nach den Bildern in der Reihenfolge Sieben – Acht – Neun – Zehn – Bube – Dame – König – As geordnet.

4.2. Textanalyse und Grafik

Im Kapitel 1.2. wurde ein fertiges Programm zur IGEL-Geometrie genutzt. Die wenigen zulässigen Befehle VORWAERTS x, RECHTS y, WIEDERHOLE z, BILDSCHIRM SAEUBERN, MITTE, HEBE, SENKE, IGEL, LERNE name und ENDE konnten zum Zeichnen geometrischer Figuren verwendet werden.

4.2.1. Problemanalyse

Hier geht es darum, zu erklären, wie derartige Programme entwickelt werden können. In einem solchen Programm steckt das Problem der Textanalyse und Sprachinterpretation in elementarer Form. Ist die syntaktische Richtigkeit nicht gegeben, so muß das Programm den Nutzer auf den Eingabefehler aufmerksam machen und ihm die Möglichkeit einer neuen Eingabe bieten. Im weiteren ist eine als syntaktisch korrekt erkannte Folge von Igelbefehlen durch eine Befehlsfolge in einer dem Computer bekannten Programmiersprache – hier BASIC – zu interpretieren, so daß anschließend die eingegebene Igel-Befehlsfolge realisiert wird.

4.2.2. Algorithmierung

Die im Kapitel 1.2. innerhalb des Programms zur Igel-Geometrie ‚erlernbaren' Befehlsfolgen wurden dort als Programme bezeichnet. Zur besseren textlichen Unterscheidung und zur Kennzeichnung ihrer Rolle werden sie nun Prozeduren genannt, so daß wir z. B. von „Eingabe einer Prozedur" und „Ausführung der Prozedur P$(J)" sprechen.

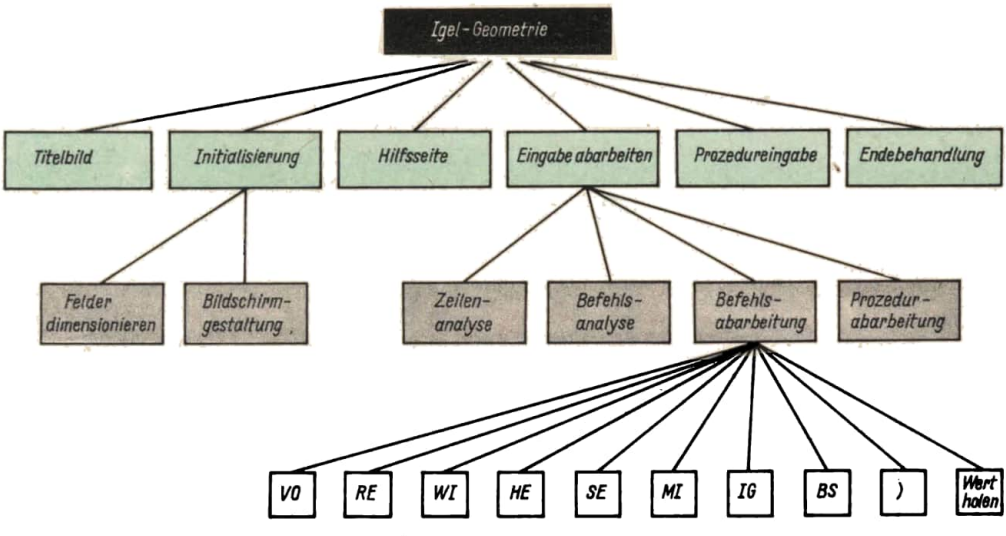


Abb. 4.2/1: Modulstruktur des Programms „Igel-Geometrie"

Die Umsetzung der Modulstruktur in ein Struktogramm zeigt Abb. 4.2/2.
Der Kern des Algorithmus besteht aus einer Wiederholschleife. Dort wird die Eingabe einer Zeichenkette, die Igel-Befehle oder Igel-Programmnamen enthalten sollte, wieder-

holt, bis die Eingabe das Zeichen „@" ist. Wenn E$ die Leerzeichenkette ist, so soll der Programmnutzer zur Eingabe eines Igel-Befehls aufgefordert werden. Ist E$=„?", so soll die auch am Anfang gezeigte Übersicht ‚HILFSSEITE' auf den Bildschirm geschrieben werden. Alle zulässigen Igel-Befehle werden gezeigt und kurz erläutert, damit sich der Nutzer über die Elemente der Dialogsprache des Programms informieren kann.

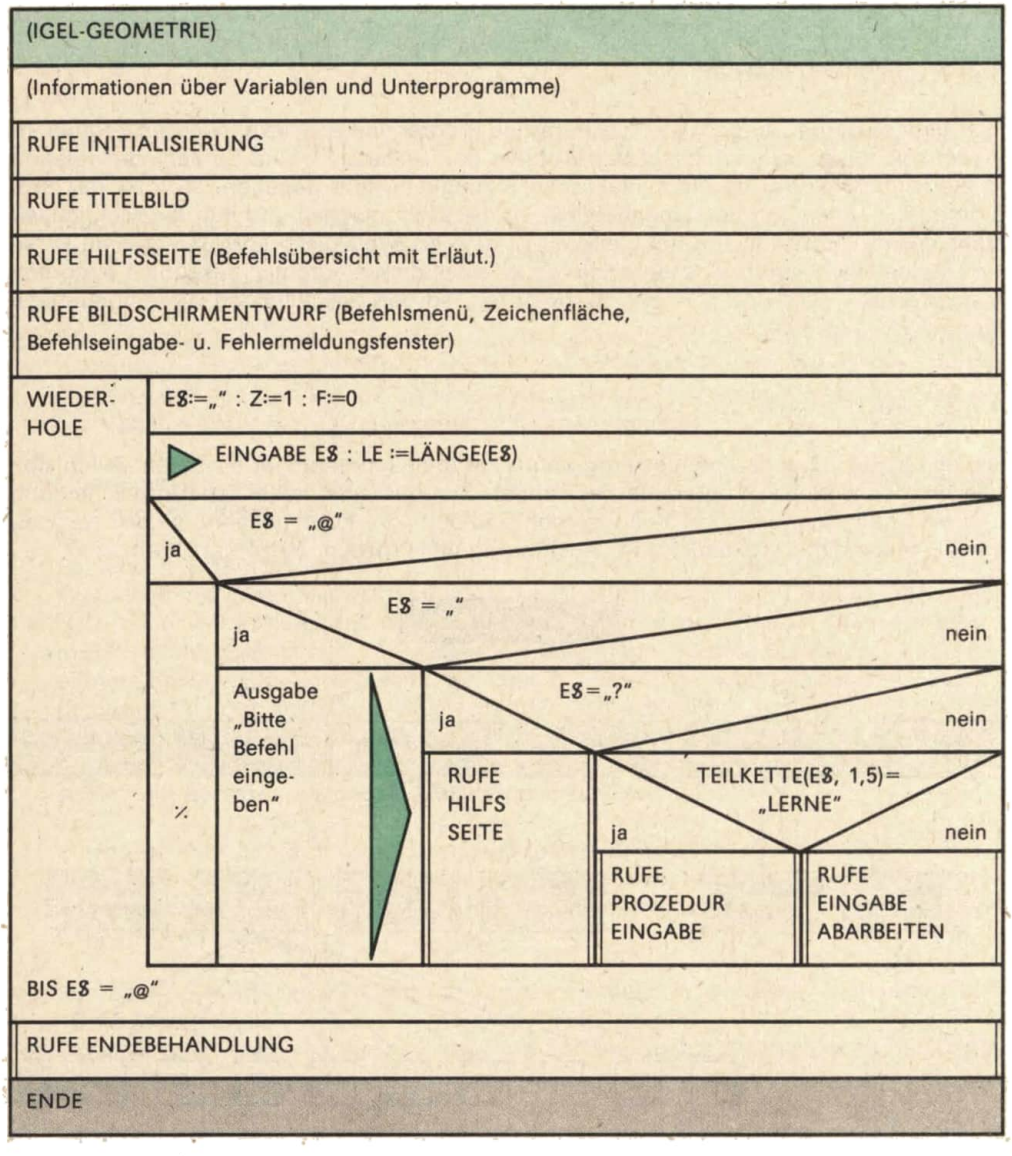


Abb. 4.2/2: Struktogramm „IGEL-GEOMETRIE"

Mit dem Schlüsselwort LERNE wird dem Nutzer die Möglichkeit eröffnet, Igel-Prozeduren zu schreiben, die mit dem Schlüsselwort ENDE abzuschließen sind. Sie werden zei-

lennumeriert eingegeben und in einem Prozedurfeld P$ gespeichert. Jede andere Zeichenkette ist daraufhin zu analysieren, ob sie entweder einen Igel-Befehl oder eine Igel-Prozedur oder keines von beiden beinhaltet.

(UA EINGABE ABARBEITEN)	
(Zweck: Aus der Eingabezeichenkette E$ baut der Unteralgorithmus ZEILENANALYSE aufgrund des Auftretens von Klammern oder Leerzeichen eine Ausgabezeichenkette A$ auf. Der Unteralgorithmus BEFEHLSANALYSE MIT ABARBEITUNG identifiziert A$ als einen Igel-Befehl oder eine Igel-Prozedur oder als unzulässige Eingabe. Igelbefehle oder -Prozeduren werden ausgeführt.)	
BEGINN	
WIEDERHOLE	RUFE ZEILENANALYSE
	RUFE BEFEHLSANALYSE MIT ABARBEITUNG
BIS Z>LE oder F = 1	
ENDE	

Abb. 4.2/3: Struktogramm „EINGABE ABARBEITEN"

Bei Igel-Befehlen ist zu unterscheiden zwischen parameterabhängigen Befehlen und solchen, die von keiner eingegebenen Größe abhängen. Bei den Befehlen VORWAERTS x, RECHTS y und WIEDERHOLE z(...) müssen nach der Schlüsselwortidentifizierung auch noch die Werte x bzw. y bzw. z identifiziert werden.

(UA ZEILENANALYSE)	
(Eingabevariable:	E$ – Eingangszeichenkette Z – aktuelle Zeichennr. in E$ LE – Länge der Zeichenkette E$
Ausgangsvariable:	A$ – Ausgangszeichenkette (Schlüsselwort, Wert, Prozedurnamen, unzulässige ZK)
‚lokale' Variable:	T$ – Teilzeichenkette ZA – Zeichennummer am Anfang der Ausgangszeichenkette)
BEGINN	
WIEDERHOLE	T$:= TEILKETTE(E$,Z,1)
	Z := Z+1
BIS T$<>„ " oder T$<>„(" oder Z>LE	

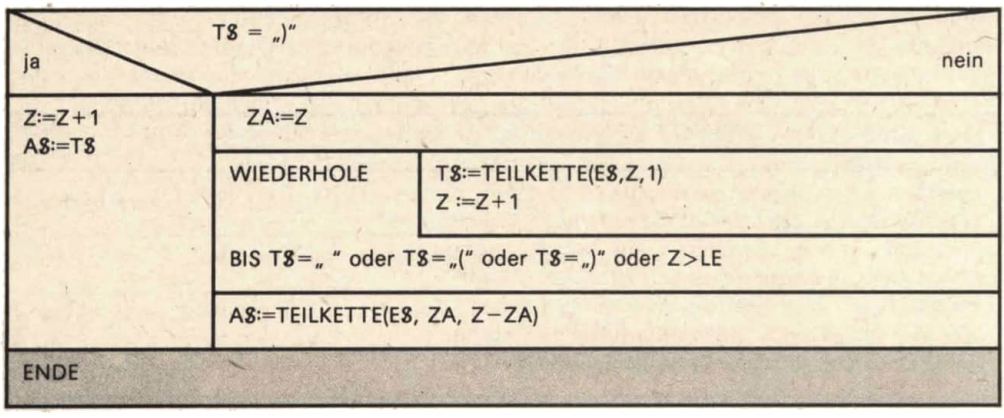


Abb. 4.2/4: Struktogramm „ZEILENANALYSE"

(UA BEFEHLSANALYSE MIT ABARBEITUNG)

(Eingangsvariable: A$ – Ausgangszeichenkette UA ZEILENANALYSE
B$ – Befehlswort der Igel-Geometrie
N – Anzahl der Igelbefehlsnamen
PN – Anzahl der definierten Prozeduren
PN$(I) – Prozedurnamenliste
P$(I) – definierte Prozedur (I=0, 1, ..., PN)
‚lokale' Variable: I, J – Laufvariable
G – Merkvariable; wenn Igelbefehl gefunden, dann G=1, sonst G=0
F – Merkvariable; wenn weder Igelbefehl noch Igelprozedur gefunden, dann F=1, sonst F=0
Unterprogramme: AUSFÜHRUNG DES I. BEFEHLS B$
AUSFÜHRUNG DER PROZEDUR P$(J))

BEGINN

I:=0 : G:=0

WIEDERHOLE

I:=I+1

Lies B$

A$ = B$

ja

nein

G:=1

BIS I=N oder G=1

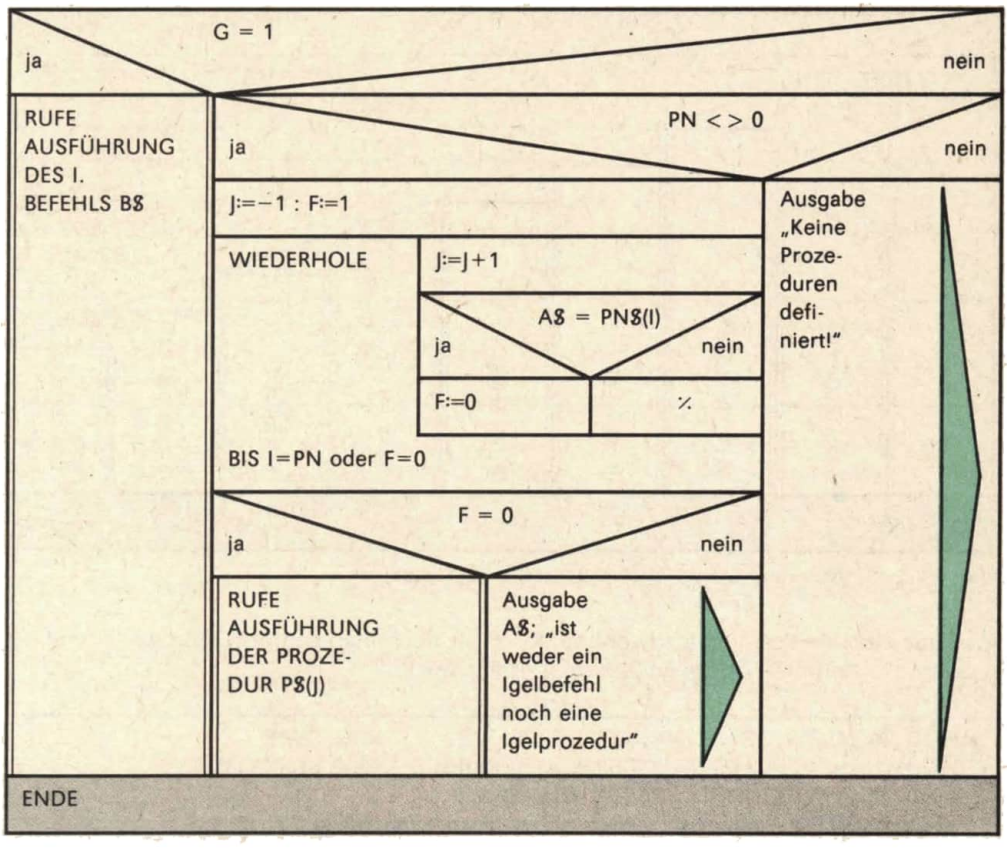


Abb. 4.2/5: Struktogramm „BEFEHLSANALYSE"

(UA AUSFÜHRUNG DES I. BEFEHLS B$)	
(Zweck: Der Unteralgorithmus vermittelt die Unteral-gorithmen, die zur Nummer I auszuführen sind. Eingangsvariable: I – Nummer des auszuführ. Befehls)	
BEGINN	
I>11	
ja	nein
I:=I−11	%

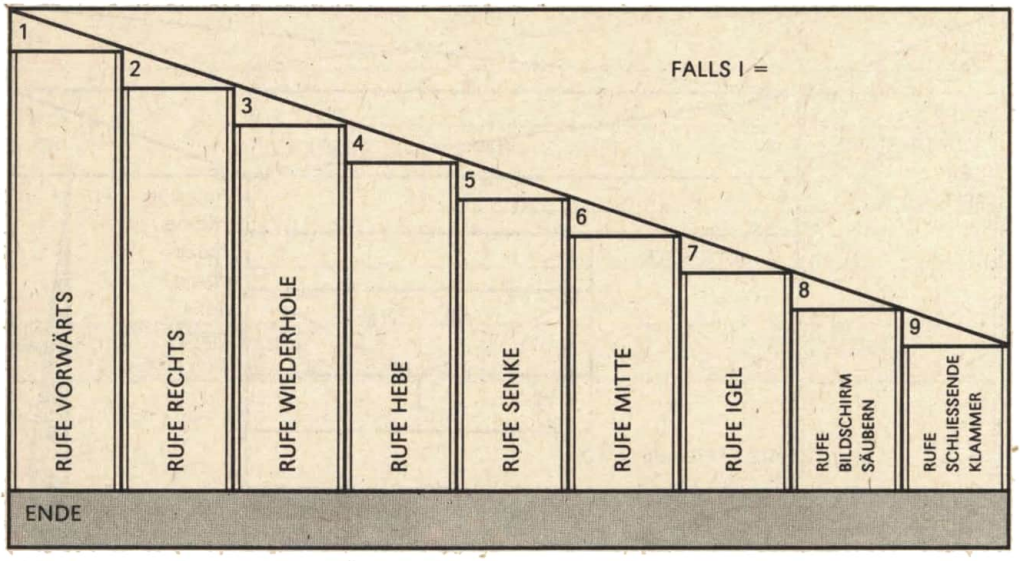


Abb. 4.2/6: Struktogramm „AUSFÜHRUNG DES I. BEFEHLS"

Weil zur Igelbefehlseingabe sowohl die Kurz- als auch die Langform möglich ist und die Igelbefehlsliste in DATA-Zeilen wie folgt erfaßt wird

```
50010 DATA VO,RE,WI,HE,SE, MI,IG,BS,),LERNE,ENDE
50020 DATA VORWAERTS,RECHTS,WIEDERHOLE,HEBE,SENKE,MITTE,
          IGEL,BILDSCHIRM SAEUBERN
```

muß die Steuervariable um 11 vermindert werden, wenn sie größer als 11 ist.

Um den kompliziertesten Teil des Programms zur Igel-Geometrie, die AUSFUEHRUNG DER PROZEDUR P$(J), zu entwickeln, wird nochmals die im Kapitel 1.2. beschriebene Prozedur HAUS betrachtet:

```
PROZEDUR HAUS
1 ? RECHTECK DREIECK
2 ? HEBE
3 ? VORWAERTS 20 RECHTS 90 VORWAERTS 20
4 ? SENKE FENSTER RECHTS 270
5 ? HEBE VORWAERTS 40
6 ? SENKE FENSTER RECHTS 180
7 ? HEBE VORWAERTS 70 RECHTS 90
8 ? VORWAERTS 20 RECHTS 90
9 ? ENDE
```

Innerhalb der AUSFUEHRUNG der Prozedur gibt es Stellen, an denen sich die AUSFUEHRUNG der Prozedur selbst aufruft, nämlich z. B. dann, wenn die AUSFUEHRUNG der Prozedur HAUS auf eine Prozedur DREIECK oder auf eine Prozedur FENSTER stößt.

Wäre darüber hinaus die Prozedur FENSTER wie folgt erfaßt:

```
PROZEDUR FENSTER
   1 ? WIEDERHOLE 4 (QUADRAT RECHTS 90)
   2 ? ENDE
```

dann würde bei der AUSFUEHRUNG der Prozedur FENSTER sogar eine weitere AUSFUEHRUNG der Prozedur QUADRAT aufgerufen werden. QUADRAT kann in 2 Zeilen definiert werden:

```
PROZEDUR QUADRAT
   1 ? WIEDERHOLE 4 (VORWAERTS 10 RECHTS 90)
   2 ? ENDE
```

Dann tritt in den Zeilen 4 und 6 der PROZEDUR HAUS ein zweifach in sich geschachtelter Aufruf der AUSFUEHRUNG EINER PROZEDUR auf. Es wird nämlich von der AUSFUEHRUNG von HAUS die AUSFUEHRUNG von FENSTER, von dieser die AUSFUEHRUNG von QUADRAT aufgerufen. Ist die AUSFUEHRUNG von QUADRAT erfolgt, wird das Ergebnis an die aufrufende Prozedur FENSTER zurückgegeben. Ist die AUSFUEHRUNG von FENSTER erfolgt, kann das Ergebnis an die aufrufende Prozedur HAUS zurückgegeben werden.

Eine solche geschachtelte Programmabarbeitung, bei der sich ein Programm selbst aufruft, wird rekursiv genannt. In der Mathematik gibt es auch rekursiv definierte Funktionen. Im Kapitel 2.3. (↗ S. 90) wurde eine rekursive Definition von N! gegeben:

1! = 1

N! = (N−1)! * N für N>1

Das dort angegebene Programm zur Berechnung von N! arbeitet allerdings iterativ, denn die Berechnung von Fakultät(N) wird nicht auf die Berechnung von Fakultät (N−1) zurückgeführt. Mit folgendem Programm kann N! auch rekursiv berechnet werden.

```
10 REM ========= REKURSIVE BERECHNUNG VON N! ========
20 WINDOW 0,31,0,39: COLOR 7,1: CLS
30 PRINT AT(2,5); "REKURSIVE Berechnung von N!"
100 WINDOW 4,26,0,39: COLOR 0,6: CLS
110 K=0
120 F=1
130 INPUT "N="; N
140 GOSUB 200
150 PRINT ,N;"!= "; F
190 END

200 REM ====== UP ======
210 IF K=N THEN 280
220 K=K+1
240 GOSUB 200:! UP ruft sich selbst auf
250 F=F*K
270 K=K-1
280 RETURN
```

Soll die Verschachtelung mitprotokolliert werden, sind noch zwei Anweisungszeilen einzufügen:

```
230 PRRINT K;". Aufruf"
260 PRINT "Rekursionstiefe ";K,"F= ";F
```

Ein *‚Schreibtischprotokoll'* soll die Rekursivität am Beispiel 3! verdeutlichen.

Zeilennummern von interessierenden Befehlen in der Reihenfolge der Abarbeitung	K	F	K = 3 erfüllt? (ja/nein)	Kommentare	Rückkehradressenspeicher
110 120 130 140	0	1		Die Rückkehradresse, die direkt auf 140 GOSUB 200 folgt, wird im Kellerspeicher – STACK genannt – aufbewahrt.	150
200 210 220 240	1		nein	Rückkehradresse im STACK speichern.	250 150
200 210 220 240	2		nein	Rückkehradresse im STACK speichern.	250 250 150
200 210 220 240	3		nein	Rückkehradresse im STACK speichern.	250 250 250 150
200 210 280			ja	Sprung nach 280, wo RETURN steht, das das Aufsuchen der zuletzt im STACK gespeicherten Adresse bewirkt.	250 250 150
250 260 280	2	3 := 3 * 1		RETURN bewirkt das Aufsuchen der obersten STACK-Adresse.	250 150

250 260 280		1	6 := 3 * 2		RETURN bewirkt das Aufsuchen der obersten STACK-Adresse.	150
250 260 280		0	6 := 6 * 1		RETURN bewirkt das Aufsuchen der obersten STACK-Adresse.	STACK ist leer
150 190					Ergebnis wird auf Schirm ausgegeben. Programmende	

Abb. 4.2/7: Protokoll zum ‚Schreibtischtest'

Diese Vorgehensweise ist auch auf die AUSFUEHRUNG DER PROZEDUR P$(J) übertragbar. Allerdings muß zum Anfang des Unteralgorithmus gespeichert werden, was abgearbeitet werden soll und bis zu welcher Stelle die Zeichenkette, die die Prozedur beinhaltet, abgearbeitet wurde. Diese Informationen werden beim Austritt aus dem Unteralgorithmus zur weiteren Programmabarbeitung wieder zur Verfügung gestellt.

4.2.3. Prozeduren und Unteralgorithmen zur Igelgeometrie

```
(UA AUSFÜHRUNG DER PROZEDUR P$(J))

(ZWECK: In der Eingabezeichenkette E$ wurde A$ gefunden u. Vergleich mit
Prozedurnamenliste ergab A$=PN$(J), so daß defin. Prozedur P$(J) auszuführen ist.

Eingangsvariable: E$ – Eingabezeichenkette
                  Z – aktuelle Zeichennummer
Ausgangsvariable: F – Fehlmeldungsvariable
‚lokale' Variable: ER$ – Ersatzfeld zur Aufnahme
                  der Eingabezeichenkette
                  ZR – Merkvariable Zeichennr.
                  AN – Anfangszeile der J. Proz.
                  EE – Merkvariable für Ende
Unterprogramme: ZEILENANALYSE
                BEFEHLSANALYSE MIT ABARBEITUNG)

BEGINN

ER$(RR) := E$          :   ZR(RR) := Z
RR      := RR+1        :   AN     := P(J)
EE      := 0

E$ := P$(AN)           :   LE := LÄNGE(E$)
Z := 1
```

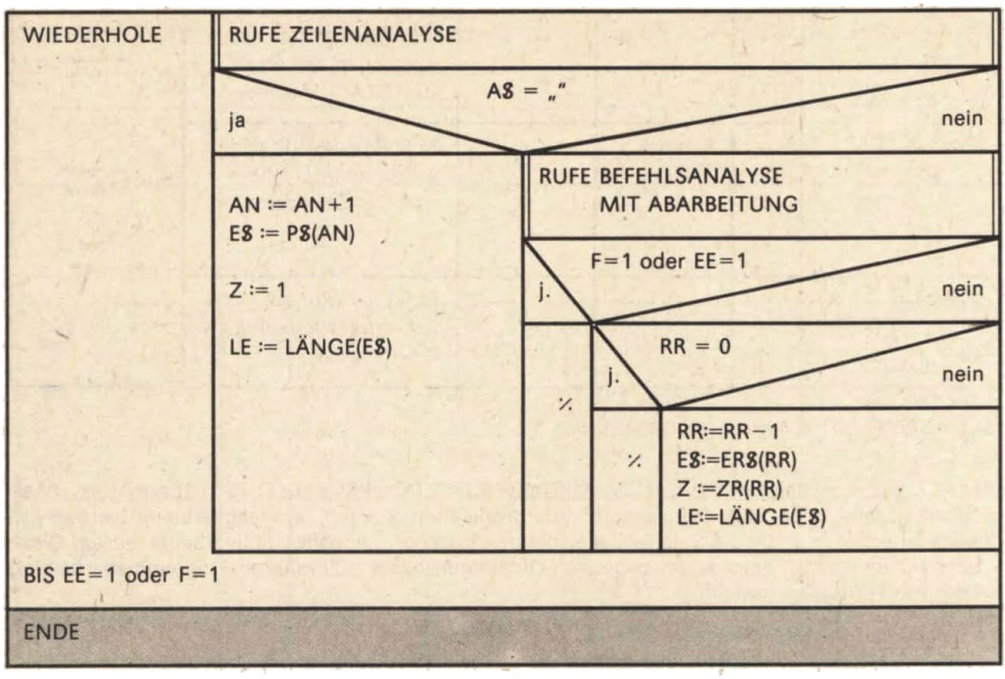
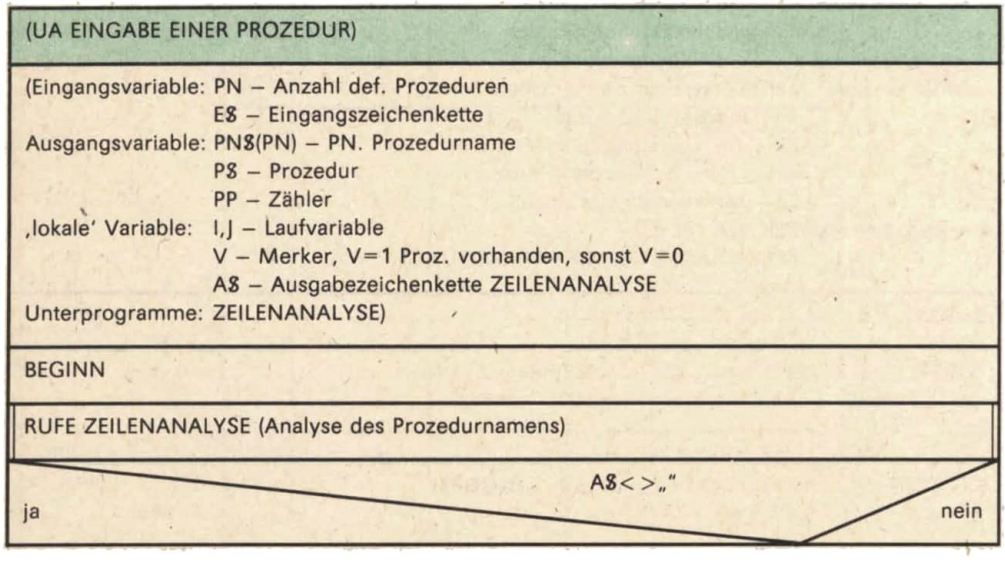


Abb. 4.2/8: Struktogramm „AUSFÜHRUNG DER PROZEDUR P$(J)"

Soll eine Prozedur abgearbeitet werden, so muß sie vorher in den Computer eingegeben sein. Das nächste Struktogramm zeigt den Algorithmus zur EINGABE EINER PROZEDUR.



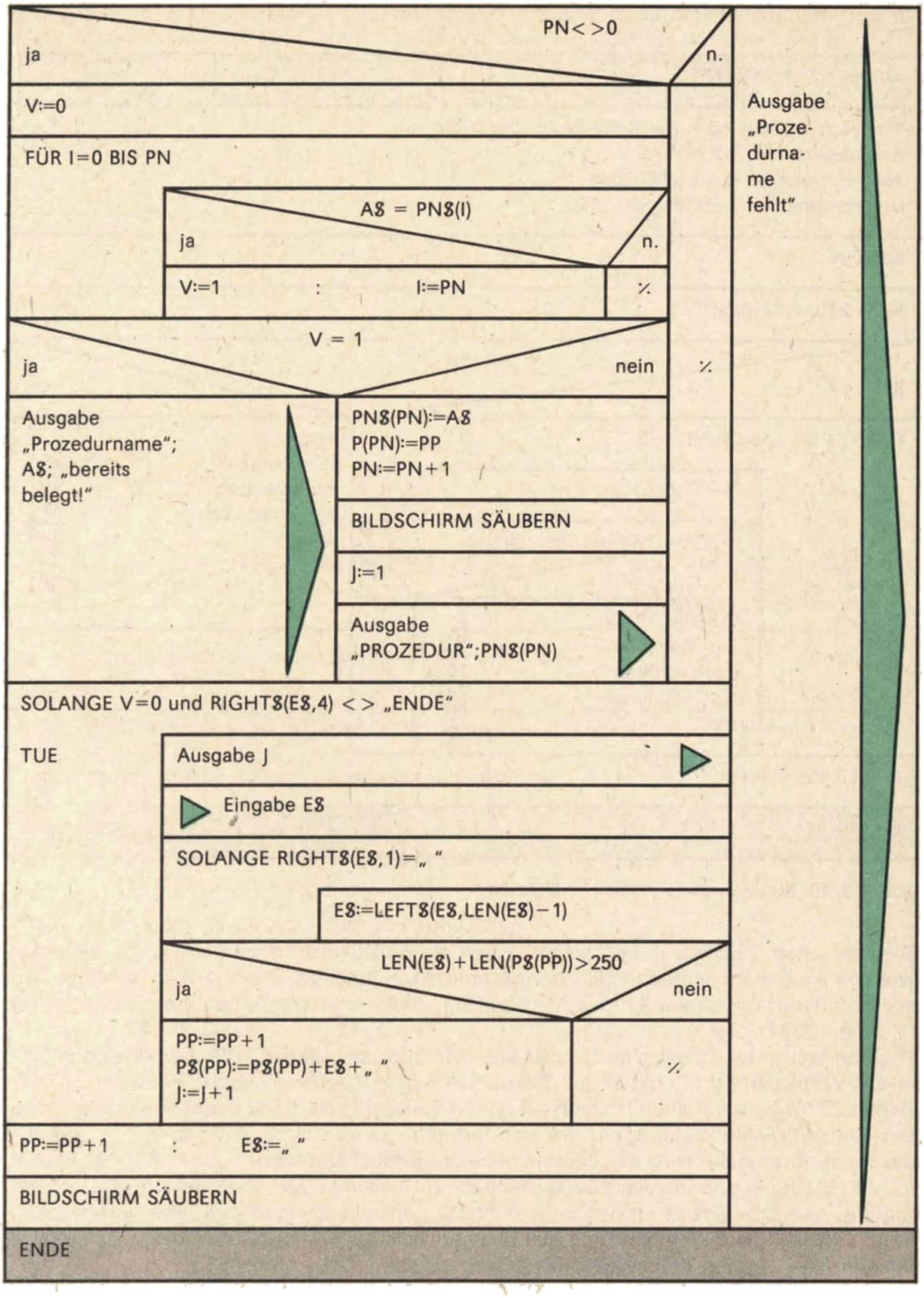


Abb. 4.2/9: Struktogramm „EINGABE EINER PROZEDUR"

Für die Befehle mit Parametern ist ein Unteralgorithmus zum Wert holen zu entwickeln:

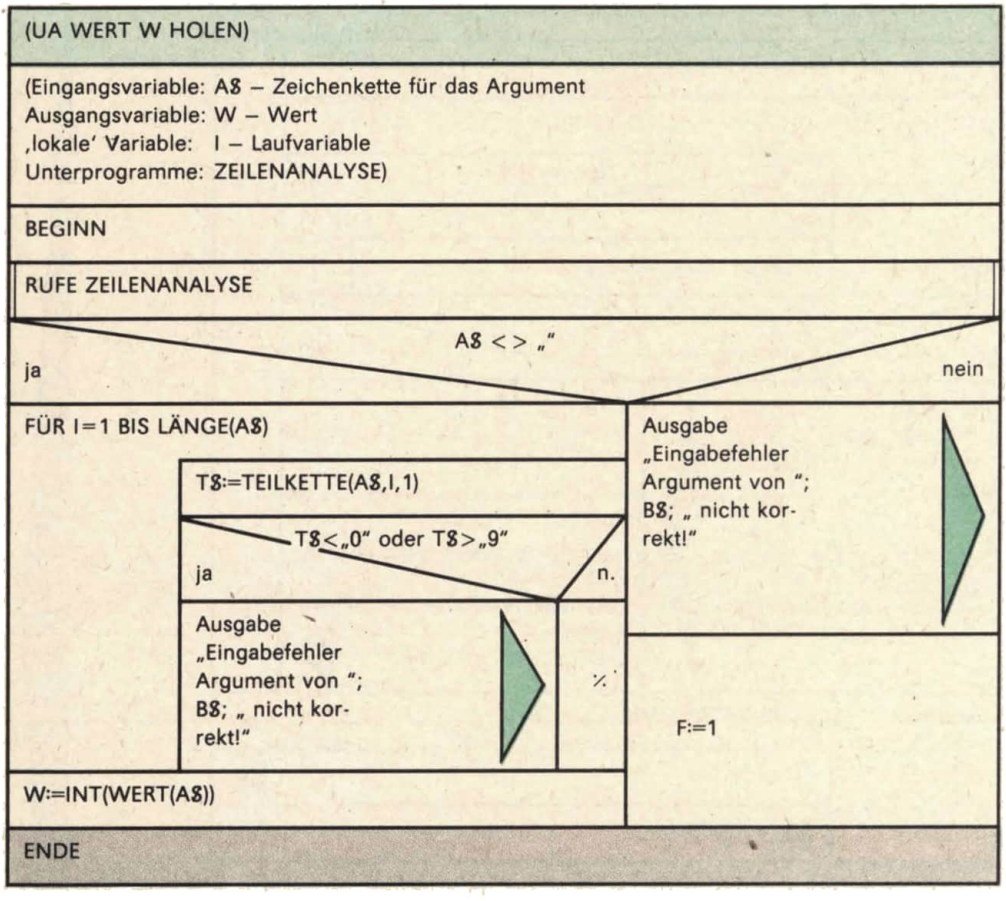


Abb. 4.2/10: Struktogramm „WERT W HOLEN"

Die restlichen Teilalgorithmen sollen gleich als BASIC-Programme angegeben werden, weil sie von recht einfacher algorithmischer Struktur sind. Doch zuvor wird ein Bildschirmentwurf mit einem Befehlsmenü zur Eingabe von Befehlsfolgen erarbeitet, den Abbildung 4.2/11 zeigt.
Die obersten zwei Zeilen sind für das Befehlsmenü vorgesehen. Die Zeichenfläche geht in der Vertikalen von Punkt 56 bis 239 und in der Horizontalen von Punkt 0 bis 319. Die Zeilen 25 bis 31 beinhalten im linken Teil die Zeilen für die Befehlseingabe und im rechten Teil ggf. Fehlermeldungen, die sich farblich von der Umgebung abheben. Im Falle der Prozedureingabe wird das Befehlsmenü ausgetauscht gegen VO, RE, WI, HE, SE, MI, IG, BS, ENDE, @ und die Zeichenfläche steht zur zeilenweisen Prozedureingabe zur Verfügung. Mit dem Erreichen des Befehls ENDE geht das Programm in den vorhergehenden Zustand – nach dem Löschen des Zeichenschirms und nach dem Rücktausch des Befehlsmenüs – in den Befehlseingabezustand zurück.
Um die richtige Einordnung der BASIC-Programmteile zu ermöglichen, wird das Hauptprogramm in den wichtigsten Passagen dargestellt:

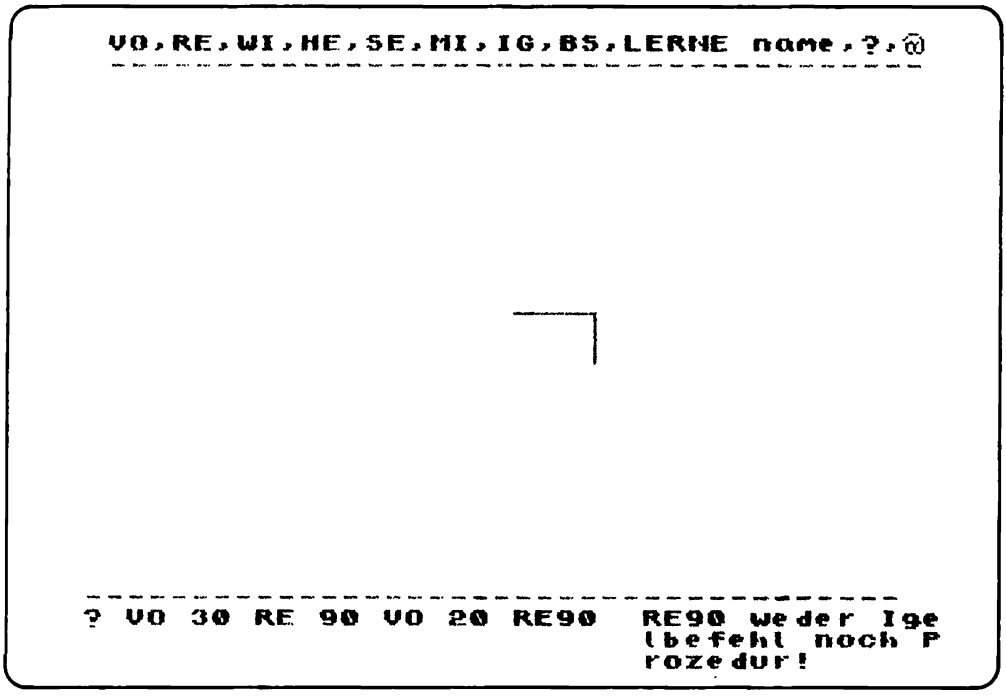


Abb. 4.2/11: Bildschirmentwurf zur Igel-Geometrie

```
100  REM ===== HAUPTPROGRAMM ======================
...  ...
300  GOSUB 1000 : REM → INITIALISIERUNG
310  GOSUB 9000 : REM → TITELBILD
320  GOSUB 2000 : REM → HILFSSEITE
330  GOSUB 3000 : REM → BILDSCHIRMENTWURF
...
400  REM anfang der wiederholschleife 1
410  E$="" : Z=1 : F=0
420  INPUT E$ : LE=LEN(E$)
430    IF E$="@" THEN 900
440      IF NOT(E$="") THEN 490
450        PRINT INK 23;AT(26,30);"Bitte"
460        PRINT INK 23;AT(27,30);"Befehl"
470        PRINT INK 23;AT(28,30);"eingeben!"
480      GOTO 900
490        IF NOT(E$="?") THEN 520
500          GOSUB 2000 : REM → HILFSSEITE
510        GOTO 900
520          IF NOT(MID$(E$,1,5)="LERNE") THEN 550
530            GOSUB 5000 : REM → PROZEDUREINGABE
540          GOTO 900
550            REM anfang wiederholschleife 2
```

```
560              GOSUB 6000 : REM →ZEILENANALYSE
570              GOSUB 7000 : REM →BEFEHLSANALYSE
580            IF NOT(Z>LE OR F=1) THEN 560
590            REM ende wiederholschleife 2
900  IF NOT(E$="@") THEN 410
910  REM ende wiederholschleife 1
920  GOSUB 8000 : REM → ENDEBEHANDLUNG
990  END : REM ----------------------------------------

1000 REM ===== UA INITIALISIERUNG ==========
1100 DIM PN$(20),P$(20),S(20,1),P(10),ZR(10),ER$(10)
1110 GOSUB 4700 : REM → IGEL IN DIE MITTE
1120 GOSUB 4900 : REM → BILDSCHIRM SAEUBERN
1130 GOSUB 4600 : REM → IGEL SENKEN
1140 BE$="VO,RE,WI,HE,SE,MI,IG,BS,LERNE name,?,@"
1150 BP$="VO,RE,WI,HE,SE,MI,IG,BS,ENDE,@"
1990 RETURN : REM ------------------------------------
```

Die Programmteile zum Titelbild und zur Endebehandlung werden hier nicht angesprochen. Sie sind entsprechend der individuellen Note des Programmierers sinnvoll zu gestalten. Die textlichen Ausführungen zum Unteralgorithmus HILFSSEITE bzw. BILDSCHIRMENTWURF sind ausführlich, so daß auch das nicht dargestellt werden braucht. Somit bleiben noch die Programmteile für die Igelbefehle.

```
4100 REM ===== VORWAERTS ============================
4110 REM
4120 GOSUB 4000 : REM → WERT W HOLEN
4130 IF F=1 THEN 4190
4140 R=W
4150 XA=X : YA=Y
4160 X=X+R*COS(PH) : Y=Y+R*SIN(PH)
4170 IF ZE=0 THEN 4190
4180 LINE XA,YA,X,Y,15 : REM Linie zeichnen (XA,YA)→(X,Y)
4190 RETURN : REM ------------------------------------

4200 REM ===== RECHTS ==============================
4210 REM
4220 GOSUB 4000 : REM → WERT W HOLEN
4230 IF F=1 THEN 4290
4240 WI=WI-W
4250 IF WI>=360 THEN WI=WI-360 : GOTO 4250
4260 IF WI<0 THEN WI=WI+360 : GOTO 4260
4270 PH=WI*PI/180 : IF WI=90 OR WI=270 THEN M=0 : GOTO 4290
4280 M=TAN(PH)
4290 RETURN : REM ------------------------------------

4300 REM ===== WIEDERHOLE ============================

4310 REM
4320 GOSUB 4000 : REM → WERT W HOLEN
4330 IF F=1 THEN 4390
```

```
4340 S=S+1
4350 S(S,0)=Z
4360 S(S,1)=W-1
4390 RETURN : REM ------------------------------------

4400 REM ===== ) schliessende Klammer ==========
4410 REM
4420 IF S=0 THEN PRINT PAPER 0;" ) zuviel!"; : PRINT :
     GOTO 4490
4430 IF S(S,1)=0 THEN S=S-1 : GOTO 4490
4440 S(S,1)=S(S,1)-1
4450 Z=S(S,0)
4490 RETURN : REM ------------------------------------
```

- Entwickeln Sie BASIC-Programme für die Igelbefehle HEBE, SENKE, MITTE, IGEL, BILDSCHIRM SAEUBERN! Beachten Sie dabei, daß das Unterprogramm IGEL die aktuelle Position und Richtung der ‚Igelschnauze' ▷ zeigen soll. Der Befehl IG hat wie ein Schalter zu funktionieren:

  – ist ▷ nicht zu sehen, bewirkt IG oder die Langform IGEL, daß ▷ gezeichnet wird,
  – ist ▷ zu sehen, wird mit IG das Zeichen ▷ ‚radiert'.

  Ausgehend vom erreichten Punkt und der bisherigen Richtung werden die 3 Eckpunkte berechnet, die das Dreieck eindeutig festlegen. Es kann punktweise mit PSET gezeichnet werden, so daß mit PRESET ein ‚Radieren' möglich wird.

- Codieren Sie alle bisher entwickelten Unteralgorithmen des Igel-Programms!

- Ergänzen Sie das so entstandene BASIC-Programm durch Unterprogramme zur Realisierung des Bildschirmentwurfs, einer Hilfsseite, eines Titelbildes und der Endebehandlung! Testen Sie das durch diese 3 Aufträge realisierte Programm zur Igel-Geometrie zumindest mit den Igelprogrammen des Kapitels 1.2.!

  Das entstandene Programm zur Igel-Geometrie ist in mehreren Richtungen verbesserungswürdig:

  1. Neue Igelbefehle können den Sprachumfang erweitern, wie z. B. ZURUECK x, LINKS y, KREIS r.
  2. Geschriebene Igelprozeduren sollen auf Magnetband gespeichert und von dort wieder eingelesen werden können.
  3. Es ist günstig, wenn geeignete Igelbefehle zum Ausgeben der Prozedurnamenliste auf dem Bildschirm, zum Löschen oder zum Korrigieren einer Prozedur existieren. Diese Aufgaben ähneln den entsprechenden im Dateiprogramm.
  4. Unteralgorithmen zur Umbenennung von Prozedurnamen können auch von Interesse sein.
  5. Zu Bildschirmgrafiken, die mit Igelprozeduren erzeugt wurden, sollten Hardkopien mittels grafikfähigem Drucker oder Plotter realisierbar sein.
  6. Ein weites Feld von Anwendungen eröffnet sich, wenn Prozeduren nicht nur mit festen Parametern geschrieben werden können. Aber die Realisierung von Parameterübergabemechanismen stellt hohe Anforderungen an die Algorithmierfähigkeiten, so daß diese Aufgabenstellung zum Gegenstand eines Projektes gemacht werden kann.

Zusammenfassung

Um die vielen Computernutzer nicht alle zu Programmierern ausbilden zu müssen, werden Programme geschaffen, die den Nutzern eine sehr problemnahe Sprache zum Formulieren und Lösen ihrer Aufgaben bzw. Probleme anbieten. Ein übergreifendes Prinzip solcher problemnaher Sprachen besteht im Übersetzen der Aufgaben- bzw. Problemlösungsformulierung in eine dem Computer verständliche Sprache. Der Text einer Befehlsfolge wird im allgemeinen in zweifacher Hinsicht analysiert:

1. Es ist ein eingegebener ‚Befehlstext' auf syntaktische Korrektheit bezüglich der Befehlssprache zu untersuchen (Zeilenanalyse).

2. Es erfolgt die Übersetzung (Befehlsanalyse und ev. Abarbeitung) in eine dem Computer verständliche Sprache, so daß die Befehle ausgeführt werden können.

Was mit den Ausführungen am Beispiel eines elementaren Grafikprogramms erläutert wurde, kann im Prinzip beispielsweise in Programmen für die Steuerung von Robotermodellen stecken, so daß eine sehr elementare Robotersteuersprache aus {LINKS DREHEN x, RECHTS DREHEN y, AUF z, AB t, MAGNET AN, MAGNET AUS, ENDE, START} bestehen kann. Auch Übersetzungsprogramme z. B. aus der verbal formalisierten Notation oder aus den Struktogrammen in eine problemorientierte Programmiersprache wie z. B. BASIC oder PASCAL können nach dem dargestellten Prinzip gestaltet werden.
Die problemorientierten Programmiersprachen selbst werden nach ähnlichen Prinzipien in eine maschinenorientierte Programmiersprache wie z. B. Assembler übertragen. Die Vorgänge sind allerdings wesentlich komplizierter, denkt man nur daran, daß ihr Sprachumfang unter Umständen mehrere hundert Schlüsselworte umfassen kann und Namen und Werte von Variablen und Sprungadressen zu speichern sind. Außerdem sind bei dieser Übersetzung die Spezifika des jeweiligen Rechnertyps zwecks Laufzeit- und Speicherplatzoptimierung zu beachten.
Beim Grafikprogramm kam auch das in der Informatik bedeutsame und verbreitete Prinzip der Rekursion zum Tragen. Allerdings läßt sich die Rekursivität in BASIC nur recht umständlich realisieren.

● Aufgaben

1. Begründen Sie, warum die Abarbeitung von Igelbefehlsfolgen, die in der Langform geschrieben sind, länger dauert als die Abarbeitung der gleichen Befehlsfolgen, die in Kurzform geschrieben sind!

2. Die Elemente der im Programm realisierten Menge von Igelbefehlen {VO, RE, WI, HE, SE, MI, IG, BS, LERNE, ENDE, ?, @} beginnen mit paarweise verschiedenen Anfangsbuchstaben. Modifizieren Sie das Programm so, daß die Igelbefehle bereits aufgrund der Angabe ihres Anfangsbuchstabens identifiziert werden können!

3. Modifizieren Sie das Programm so, daß z. B. auch jedes Teilschlüsselwort der Form V, VO, VOR, VORW, VORWA, VORWAE, VORWAER, VORWAERT als Igelbefehl VORWAERTS erkannt und der Befehl ausgeführt wird!

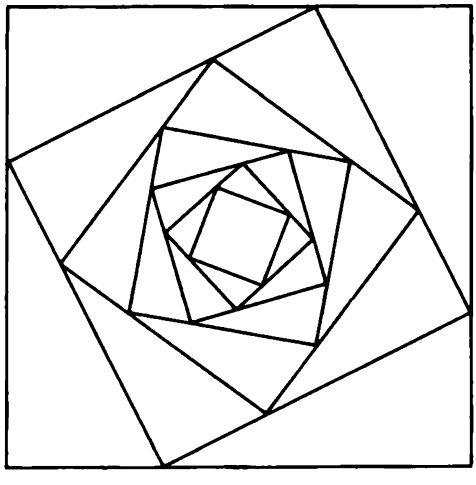
4. Belegen Sie die frei definierbaren Funktionstasten F1, F2, ..., F12 des KC 85/2 oder 3 mit der Igelbefehlsmenge, um die Eingabe von Igelbefehlsfolgen zu erleichtern!

5. Was bewirkt das folgende rekursive Programm?

```
10 CLS:X=160:Y=120
20 DIM L(100)
30 L(0)=150:N=0:PH=0
40 GOSUB 100
50 END

100 REM ==== UP ==============
110 IF L(N)<6 THEN 200
120 L(N+1)=L(N)-2
130 N=N+1
140 GOSUB 100
150 XA=X:YA=Y
160 X=X+L(N)*COS(PH) : Y=Y+L(N)*SIN(PH)
170 LINE XA,YA,X,Y,7
180 L(N-1)=L(N)*1.05
190 PH=PH+1.25 : N=N-1
200 RETURN
```

6. Entwickeln Sie ein Programm, das auf rekursive Weise folgendes Bild erzeugt:



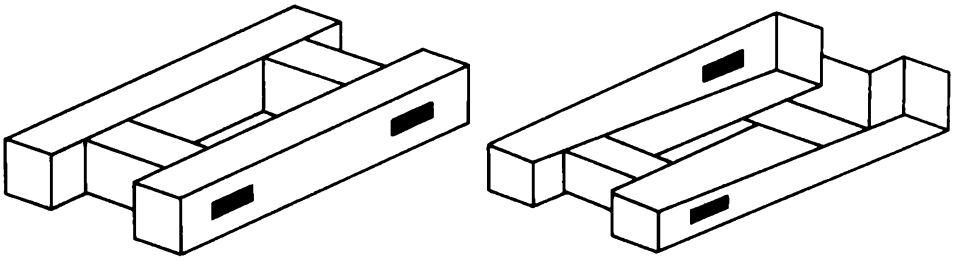
7. Schreiben Sie rekursive Igelprozeduren, wie z. B.

```
PROZEDUR REKU1      oder      PROZEDUR REKU2
1 ? WI 6 (VO 20 RE 60 )       1 ? VO 40 RE 100
2 ? RE 30                     2 ? REKU2
3 ? REKU1                     3 ? ENDE
4 ? ENDE
```

Erklären Sie, warum es zu einem BS ERROR kommt?

8. Schreiben Sie ein Programm, mit dessen Hilfe nach der Eingabe der Koordinaten von n Punkten der verbindende Polygonzug gezeichnet wird! Anschließend soll dieser Polygonzug folgenden Abbildungen unterzogen werden können:
   a) Spiegeln an einer Vertikalen, Horizontalen oder einem Punkt des Zeichenschirms;
   b) Verschieben;
   c) Drehen um den festen Punkt (159,147) mit einem Winkel alpha;
   d) Strecken bzgl. des festen Punkts (159,147) mit dem Streckungsfaktor k ($k>1$) bzw. Stauchen mit dem Stauchungsfaktor k ($0<k<1$).

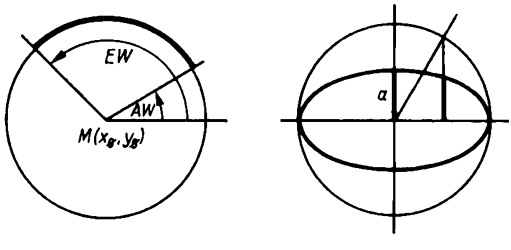
9. Entwickeln Sie ein Programm zum Zeichnen von ‚unmöglichen Figuren'! Eine Vielzahl von gezeichneten Beispielen sind in der Schülerzeitschrift „alpha" zu finden. Eine besonders originelle Idee geht auf den Schüler Thomas Fröhlich der Spezialschule „Carl Zeiss" Jena zurück. Er entwickelte ein Programm, das die Wandlung des Abbilds eines real existierenden Körpers in einen unmöglichen Körper auf dem Bildschirm ablaufen läßt, indem bestimmte Verbindungen zwischen Punkten ‚radiert' werden und andere neu eingezeichnet werden. In der folgenden Abbildung sind Ausgangsbild und Endbild der ‚Metamorphose einer Leiter' dargestellt.



10. Schreiben Sie ein Programm, das römische Zahlen in das dezimale Positionssystem umrechnet und umgekehrt! Dabei ist u. a. ein nach den Regeln eines Additionssystems gebildeter ‚text', der aus der Zeichenmenge {I, V, X, L, C, D, M} aufgebaut ist, umzurechnen in einen ‚Text', dem die Ziffernmenge {0, 1, ..., 9} zugrunde liegt und dessen Aufbau den Regeln eines Positionssystems genügt.

11. Schreiben Sie ein Programm, das folgende Kurven auf den Bildschirm zeichnet:
   a) eine pascalsche Schnecke nach

$$x = 80 + 90\cos^2\alpha + 70\cos\alpha$$
$$y = 130 + 90\cos\alpha\sin\alpha + 70\sin\alpha\ ;$$



   b) Kreisbögen, wobei Anfangs- und Endwinkel einzugeben sind;
   c) Ellipsen, wobei die kleine Halbachse a mit $0 < a < 1$ zu wählen ist!



Lösen numerischer Probleme mittels Computer

Der Computer ist ein Magellansches Schiff,
das uns zu neuen mathematischen Welten trägt.
(D. Hofstadter, Gödel-Escher-Bach)

Die Numerik als Teildisziplin der Mathematik war zu Beginn unseres Jahrhunderts in den Hintergrund getreten, da der zeitliche Aufwand für rein numerische Problemlösungen sehr groß geworden war. Für viele Probleme wurden andere Lösungswege gefunden. Erst seit der größeren Verfügbarkeit des schnellen Hilfsmittels Computer wurden numerische Verfahren wieder verstärkt genutzt.

5.1. Elementare Näherungsverfahren

Mit der verstärkten Nutzung numerischer Näherungsverfahren traten aber neue Aspekte auf, da sich bald herausstellte, daß diese neue Technik auch ihre Eigenheiten hat. Diese gilt es jedesmal neu zu überdenken.

Es wurde bisher schon mehrfach darauf hingewiesen, daß die Computer-Zahlenmenge auf der binären Darstellung beruht, und sich nur auf eine kleine, endliche Anzahl von Ziffernstellen beschränkt. Vom ETR kennen Sie Darstellungen mit 8 oder 10 dezimalen Ziffern. Aus der Beschreibung des Heimcomputers KC 85/n können Sie ersehen, daß diese Rechner ganze Zahlen nur im Bereich von
−999999 bis +999999
und die reellen Zahlen nur im Bereich von
9.40396E-39 bis 1.70141E+38
– mit 6 Ziffernstellen in der Mantisse – darstellen können.

Die Computerzahlen stellen eine endliche Teilmenge der rationalen Zahlen dar. Für die Menge der darstellbaren Computerzahlen sind nicht mehr alle arithmetischen Operationen uneingeschränkt ausführbar.

Durch die Umwandlung der Dezimalzahlen in die internen Binärzahlen treten in vielen Fällen Ungenauigkeiten auf, da Brüche im Dualsystem nur richtig darstellbar sind, wenn im Nenner Zweierpotenzen stehen. Alle anderen periodischen Brüche müssen bei dieser Umwandlung künstlich abgebrochen werden, so daß ein Fehler entsteht. Ein ganz einfaches Beispiel ist schon die Umwandlung des Dezimalbruches 0,1 in eine Binärzahl. Im Dualsystem ist dies ein periodischer Bruch, der aber aufgrund der begrenzten internen Ziffernanzahl abgebrochen wird. Die Rechengenauigkeit eines Computers ergibt sich zu $0.5 * 10^{1-t}$.
Dabei ist ‚t' die Anzahl der Ziffernstellen, die im Rechner verwendet wird. Die Kleincomputer KC 85 arbeiten intern mit 7 Stellen und zeigen extern 6 Stellen an. Deshalb liegt die

Maschinengenauigkeit bei $0.5 * 10^{-6}$. Im Zusammenhang mit eigentlich unbeschränkten Algorithmen ist es unsinnig, Abbruchkriterien kleiner als die Maschinengenauigkeit zu verlangen!

1 Der oben genannte Rundungsfehler hat bei numerischen Rechnungen natürlich Auswirkungen. Das zeigt folgendes kleines Programm:

```
10 REM Beispiel
20 FOR I=-1 TO 1
30    FOR J=I-.3 TO I+.3 STEP .1
40       K=J*J
50       PRINT I,J,K
60    NEXT J
70 NEXT I
80 END
```

Das Progamm ist schnell eingegeben und gestartet. Auf dem Bildschirm erscheinen in drei Spalten Zahlenkolonnen. Betrachten wir diese etwas näher, so fällt besonders die mittlere Zeile auf, wo nach dezimaler Rechnung eigentlich der Wert 0 in allen drei Spalten stehen müßte. Der Rechner bringt uns aber hier von 0 verschiedene Werte in Gleitkommaschreibweise. In der zweiten und dritten Spalte stehen hier
−1.49012E−08 bzw. −2.22045E−16.
Die relativ kleine negative Zahl
$-1.49012 * 10^{-8}$
ist aber eben nicht die Zahl Null. Auch wenn der Fehler als vernachlässigbar klein erscheint, so werden doch numerische Gesetzmäßigkeiten verletzt.

Worin liegen die Ursachen für diese Fehler? Eine liegt darin, daß der endliche Dezimalbruch 1/10 kein endliches Äquivalent im Dualsystem hat. Der eigentlich unendliche binäre Bruch wird abgebrochen, wenn die Anzahl der technisch möglichen Binärstellen ausgeschöpft ist. Damit entsteht ein sehr kleiner Fehler, der sich bei wiederholter Verwendung dieser ‚ungenauen' Zahl vergrößern kann. In unserem Beispiel wird der Fehler nach zehnfacher Subtraktion sichtbar.

In Beispiel 1 ist aber auch eine weitere Fehlerquelle wirksam geworden, die in der Art der Konstruktion der Laufanweisung begründet ist. Nach dem neunten Durchlauf hat der Laufindex der inneren Schleife den Wert J=−0.1, wie es der Bildschirm anzeigt. Zu diesem Wert wird nun die Schrittweite 0.1 addiert. Damit entsteht die Rechnung

$$J = -0.1 + 0.1.$$

Es werden – durch die Umwandlungsungenauigkeiten und die Fehlerfortpflanzung – annähernd gleiche Werte voneinander subtrahiert.

Folgen, Reihen und ihre Eigenschaften spielen in der numerischen Mathematik eine große Rolle. Die harmonische Reihe haben wir schon in einer Aufgabe untersucht. Sie ist ein Beispiel, für eine als „gutmütig" zu bezeichnende Folge. Gutmütig, weil ihr Verhalten jederzeit eindeutig definiert ist.

Folgen eignen sich gut zur Demonstration weiterer „numerischer Fallen", die die kritiklose Nutzung von Computern immer wieder bereithält.

2 Die Eulersche Zahl e ist durch

$$e = \lim_{n \to \infty} (1 + 1/n)^n$$

definiert. Eine ganz grobe Näherung sagt, daß der Wert für e ungefähr 2,7 ist.
Ein kleines Programm soll uns einen genaueren Wert von e liefern.

Es sollen nur einige Werte ausgedruckt werden, deshalb wird *n* nach jeder Rechnung mit 10 multipliziert. Der Nutzer hat die Möglichkeit, mit der Eingabe von m den Maximalwert von *n* festzulegen.

```
10  REM Berechnung von e
20  WINDOW 0,31,0,3 : CLS : LOCATE 7,0
30  PRINT "Naeherungsberechnung der"
40  PRINT "Eulerschen Zahl e"
50  PRINT "e = (1+1/n)^n"
60  N=10
70  LOCATE 13,5 : INPUT "Anzahl Multiplikationen (<12):";M
80  CLS
90  PRINT "      n","   Wert"
95  PRINT "============================" : PRINT
100 FOR I=1 TO M
110   W=(1+1/N)^N
120   N$=RIGHT$("         "+STR$(N),9)
130   PRINT N$,W
140   N=10*N
150 NEXT I
160 PRINT
170 PRINT "e mit EXP(1) errechnet: ";EXP(1)
180 END
```

Im Programm wird die Anzeige des Exponenten aufbereitet, um ein übersichtlicheres Bild zu erhalten. Ergänzt wurde das Programm durch die Ausgabe des Wertes der Funktion EXP(x) für x=1, also der Ausgabe des ‚echten' Rechnerwertes für die Eulersche Zahl *e*.

```
      Naeherungsberechnung der
          Eulerschen Zahl e
           mit e=(1+1/n)^n

Anzahl Multiplikationen (<12): 8

          N                 Wert
 ============================

           10          2.59374
          100          2.70478
         1000          2.71689
        10000          2.71779
       100000          2.74034
        1E+06          2.69542
        1E+07          1
        1E+08          1


e mit EXP(1) errechnet:  2.71828
```

Abb. 5.1/1: Bildschirmkopie von „Eulersche Zahl"

Das Programm wird einem ersten Test unterzogen und M=8 eingegeben. Abbildung 5.1/1 zeigt uns die beiden Bildschirmanzeigen in Form von Ausdrucken mit Matrixdrucker (Hardcopy). Die Ergebnisse nähern sich erst dem untenstehenden ‚echten' Wert, um dann wieder davon weglaufend bei 1 zu landen. Das ist aber falsch! Wo liegt die Falle? Es ist auch hier die begrenzte interne Stellenzahl. Das führt bei dieser Rechnung dazu, daß der Ausdruck $1+1/n$ bei sehr großem n – beispielsweise $n=10000000$ – intern durch Runden nur noch mit

$1+0 = 1$

berechnet wird.

Im Ergebnis könnte man nun schlußfolgern, bei näherungsweiser Rechnung muß ich mich vor sehr großen n hüten, dann vermeide ich diese Fehlerart prinzipiell. Leider reicht diese Schlußfolgerung aber nicht aus, wie uns die nächste Aufgabenstellung zeigen wird!

3 Sie haben die Sinusfunktion schon kennnengelernt und nutzen das Tafelwerk oder den ETR zur Ermittlung der Sinuswerte. Das Tafelwerk ist auf der Grundlage einer Reihenentwicklung berechnet worden. Die unendliche Reihe

$$f_n(x) = x - \frac{x^3}{1*2*3} + \frac{x^5}{1*2*3*4*5} - \ldots + (-1)^n \frac{x^{2n+1}}{1*2*\ldots*(2n+1)}$$

ergibt für alle x bei hinreichend großem n eine Näherung an $\sin(x)$.
Erstellen Sie Algorithmus und Programm zur Berechnung des Wertes $\sin(x)$ über obige Näherungsrechnung! Die Summmation soll bei Unterschreiten einer Schranke von 0,00001 für einen Summanden abgebrochen werden.

Wenn Sie das Programm erstellt haben, dann starten Sie es und geben Sie nacheinander für x die Werte 1, 5, 10, 20, 30, 50 und 100 ein! Vergleichen Sie jeweils die Ergebniswerte für s und $\sin(x)$! Abbildung 5.1/2 zeigt eine Hardcopy einer Programmversion.

```
        Naeherung fuer sin(x)
      ueber Berechnung der Folge
f(n)=(-1)^n*x^(2n+1)/(1*2*...*(2*n+1))


     X= 5

       x=  5
       5= -.958926
 sin(x)=-.958924


                    Weitere Werte (J/N)
```

Abb. 5.1/2: Ausgabe von „SINUS"

Bei kleinerem x sehen Sie eine hervorragende Annäherung zwischen beiden Werten, die sich aber mit wachsendem x verschlechtert. Dem Fehler kommen wir auf die Spur, wenn wir uns beispielsweise für $x=10$ die einzelnen Summanden ansehen. Die PRINT-Anweisung ist schnell in den Zyklus eingesetzt, und die ausgelisteten Werte zeigen uns, nach einer kurzen Anlaufphase eine sehr schnelle Annäherung an den Wert von sin(10).

Auch hier liegt das Problem wieder – allerdings in einer anderen Weise als vorher – an der geringen Stellenzahl im Rechner. Bei höheren Werten für x erhalten wir in der Anlaufphase relativ große Werte, so daß nur noch wenige Nachkommastellen berücksichtigt werden können. Die Ziffern, die uns hier verloren gehen – zum Beispiel gegenüber der Zahlendarstellung auf dem ETR – bewirken obige Fehler.
Diese Fehlerquelle ist eine wirklich böse Falle numerischer Rechnungen, da sie völlig unvermutet zuschnappen kann!
Die aufgezeigten Fehlerquellen sollen uns ein Hinweis sein, Ergebnisse von Computerberechnungen mit einer gewissen Skepsis zu betrachten. „Computergläubigkeit" ist völlig fehl am Platze. Bei der Erstellung von Algorithmen und Programmen sollte man sich sehr genau überlegen, wie die Wertebereiche der einzelnen Größen im Laufe der Rechnung liegen. Nur so kann man das Fehlverhalten erkennen und grobe Fehler vermeiden.

Seit einigen Jahren gewinnt in der numerischen Mathematik zunehmend eine Klasse von Folgen ein reges Interesse, da sie sich nicht „gutmütig" verhalten. Dies sind Folgen, die in der sogenannten „Chaos-Theorie" im Zusammenhang mit hochkomplexen dynamischen Systemen in der Mechanik, Hydraulik, Elektronik, Wirtschaftswissenschaften, Soziologie und Ökologie wesentliche Bedeutung gefunden haben.
Die Grundlagen der Chaos-Theorie sollen und können hier nicht betrachtet werden. Näheres ist nachlesbar z. B. in [9] Heft 5, S. 4–10.
Hier sollen Folgen diskutiert werden, die die Rekursionsgleichung
$w_{i+1} = k * w_i * (1 - w_i) \qquad i \in N$
erfüllen. Dabei ist k ein Parameter, der einen beliebigen Wert annehmen kann. Für manche Werte von k und dem Anfangswert w_0 liegen alle Folgeglieder w_i zwischen 0 und 1, so daß wir sie als Wahrscheinlichkeiten interpretieren können.

Das Verhalten der obigen Rekursionsfolge ist mittels eines Programms zu untersuchen! Dies ist so zu gestalten, daß entweder
– der Parameter k variiert und der Anfangswert w_0 fest gehalten wird
oder
– w_0 variiert und k festgehalten wird!
Im weiteren Ausbau des Programms sollte die Ausgabe von Grafiken zur Veranschaulichung der Effekte möglich sein!

Der Algorithmus ist eigentlich unproblematisch und schnell skizziert, siehe Abbildung 5.1/3.
Im Algorithmus ist enthalten, daß zuerst nur 10 Werte berechnet und angezeigt werden. Dann kann der Nutzer entscheiden, ob er weitere Werte benötigt oder die Berechnung mit anderen Startwerten durchführen will. Die Erstellung des BASIC-Programm sollte keine Schwierigkeiten mehr bereiten!
Nach der Programmeingabe starten wir es und geben bei den einzelnen Tests folgende Werte ein:
$w_0 = 0.5$
$k \in \{-.5, .5, .7, 1.5, 2.5, 3.3, 3.8, 10\}$.
Werten wir die Ergebnisse, so fallen $k=-.5$ und $k=10$ völlig aus dem Rahmen, da hier keine Werte zwischen Null und Eins entstehen. Bei den anderen Werten kann man vier unterschiedliche Muster feststellen.

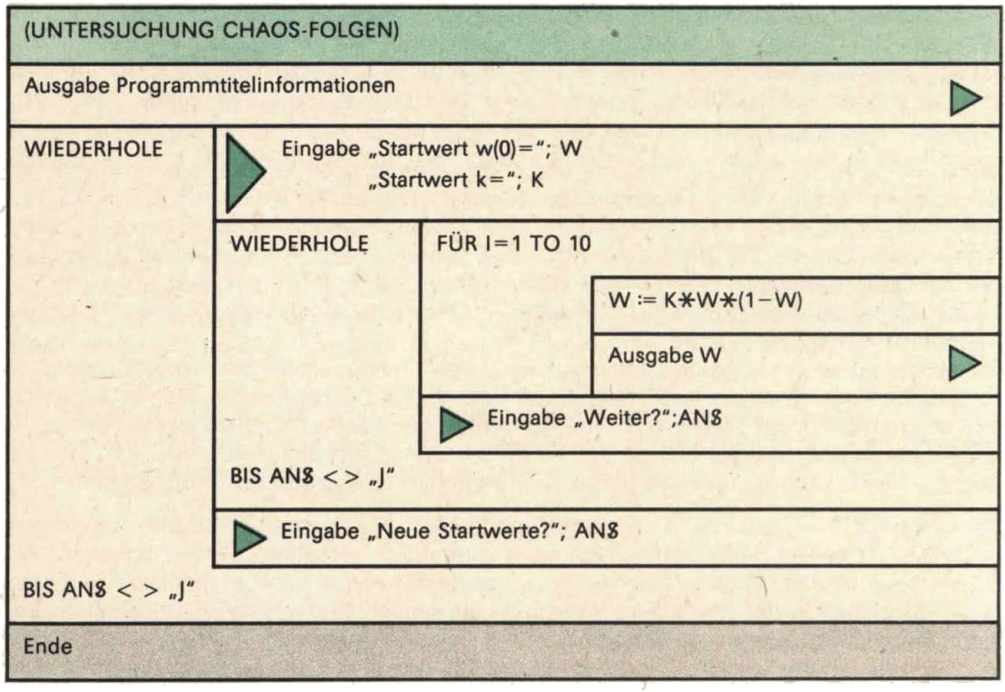


Abb. 5.1/3: Struktogramm „CHAOS 1"

1. k=.5 und k=.7 ergeben für w_i Nullfolgen
2. k=1.5 und k=2.5 ergeben konvergente Folgen mit dem Grenzwert 1/3 bzw. 3/5
3. k=3.3 ergibt eine divergente Folge mit zwei Häufungspunkten
4. k=3.8 ergibt gleichfalls eine divergente Folge, die aber keine Tendenz erkennen läßt.

Gleiche Muster erhalten wir, wenn für k ein fester Wert vorgegeben und w_i zwischen 0 und 1 variiert wird.
Offensichtlicher wird das Verhalten der Folge, wenn wir unser Programm so abändern, daß der Kurvenverlauf in einer Grafik dargestellt wird. Dazu gehen wir von den (diskreten) Folgen zur (kontinuierlichen) Funktion mit der Funktionsgleichung $y(x) = k * x * (1 - x)$ über. Der Graph dieser Funktion 2. Grades wird eine parabelförmige Gestalt haben. Nach der Modifikation könnte das Programm folgendes Aussehen haben:

```
10  REM CHAOS-Folgen mit GRAFIK
20  REM Darstellung der Funktion w(i+1)=k*w*(1-w)
30  WINDOW 0,31,0,39 : COLOR 1,7 : CLS
40  PRINT AT(0,5);"Untersuchung der rekursiven"
50  PRINT AT(1,5);"Funktion  w(i+1)=k*w*(1-w)"
60  WINDOW 2,31,0,39 : CLS
70  LOCATE 1,5 : INPUT "Startwert w(0)=";W
80  LOCATE 2,5 : INPUT "Startwert k= ";K
90  LINE 10,5,10,180 : REM Koordinatensystem
100 LINE 5,30,300,30
```

```
110 PRINT AT(7,0);"K"
120 PRINT AT(30,38);"W"
170 FOR I= 1 TO 10 STEP .01
180    W=K*W*(1-W)
190    PSET I*25+10,W*150+30
200 NEXT I
210 WINDOW 31,31,20,38 : CLS
220 INPUT "Weiter (J/N)";AN$
230 IF AN$="J" THEN 170
240 INPUT "Neue Startwerte";AN$
250 IF AN$="J" THEN 60
260 WINDOW 0,31,0,39 : CLS
270 END
```

```
        Untersuchung der rekursiven
        Folge     w(i+1)=k*W*(1-W)

        Startwert w(0)= .5
        Startwert k=  2.375
.59375
.572876
.581137
.578115
.579258
.578831
.578991
.578931
.578954
.578945
.578948
.578947
.578948
.578947
.578947
.578947
.578947
.578947
.578947
.578947

                              Weiter (J/N)
```

Abb. 5.1/4: Hardcopy des Programms „CHAOS-Folgen"

Tests des Programms zeigen, daß einige Verbesserungen notwendig sind, um den Anforderungen bezüglich Nutzerfreundlichkeit besser zu entsprechen. Solche Verbesserungen betreffen beispielsweise die Einteilung und Beschriftung der Koordinatenachsen, die Eingabe in den Zeilen 220 und 250 ist mit INKEY$ günstiger realisiert und anderes (Abb. 5.1/5).
Die grafische Untersuchung kann weiter ausgebaut werden, wenn die Winkelhalbierende y=x in das Diagramm eingesetzt wird. Durch grafische Iteration kann man dann, von w_0 ausgehend, die Folgewerte grafisch bestimmen. Wird dies beispielsweise mit den Werten k=2.4 und w_0=.8 durchgeführt, so erhält man eine Schnittstelle zwischen Winkelhalbierender und Parabel, die eine sogenannte Fixstelle der Iteration ist. Diese Fixstelle stellt

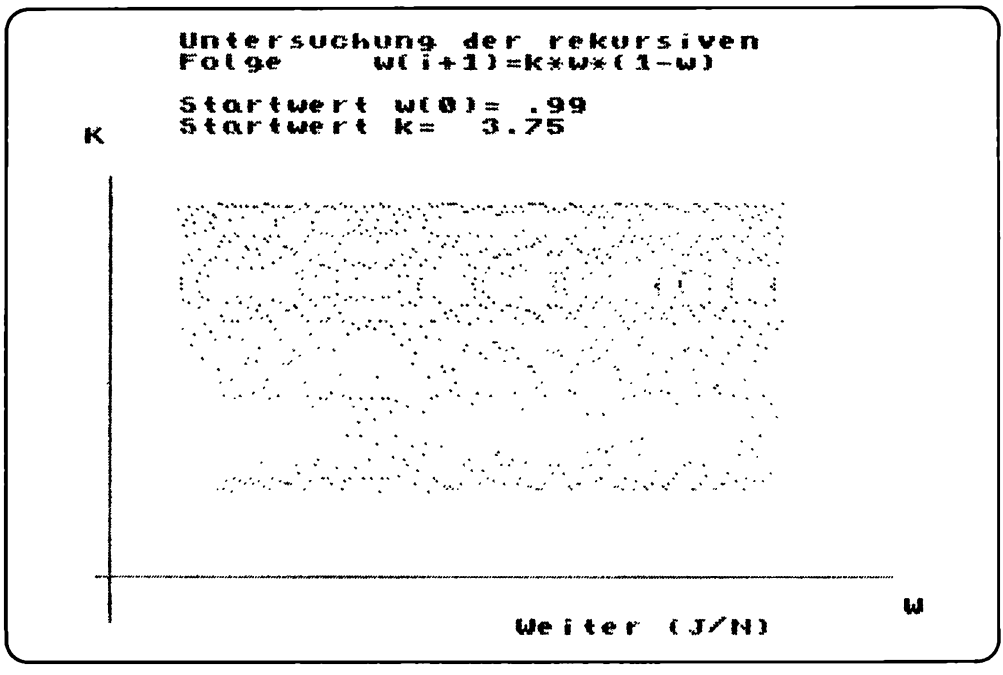


Abb. 5.1/5: Hardcopy des Programms „CHAOS-Folgen Grafik"

einen Grenzwert der Folge w_i dar. Die Konvergenzbedingungen für die Folge im Bereich $1<k<3$ können nun leicht ermittelt werden. Das Ergebnis besagt, daß die Folge in diesem Bereich streng monoton fallend und konvergent ist. Im Falle $k=1$ ist 0 der Grenzwert der Folge.
Aber nicht dieser eindeutig bestimmbare Bereich interessiert bei der Untersuchung oben genannter Systeme. Man beschäftigt sich mit dem Bereich von k, für den die Folge „chaotisches Verhalten" aufweist. Dies trifft bei obiger Folge beispielsweise auf ein $k=3.56994...$ zu. Zuerst hat sich der amerikanische Physiker Feigenbaum bei der Untersuchung dynamischer Systeme mit diesem Phänomen beschäftigt. Er hat dazu zwei Methoden angewendet, die wir auch auf unsere Folge ansetzen wollen:

1. Variation von k im Bereich von 0 bis 4 in relativ kleinen Schritten. Die Folgeglieder w_{101} bis w_{200} (bei einem festen Anfangswert w_0) trägt man in ein k-w_i-Koordinatensystem ein. Im Falle der Oszillation erhält man für jedes k mehrere Punkte im Diagramm; im Falle des Chaos sind die Punkte scheinbar regellos verteilt.
2. Festes k und Variation des Anfangswertes w_0 von 0 bis 1 in sehr kleinen Schritten, z. B. 0.01. Der Punkt (k,w_{100}) wird jeweils in das Diagramm eingetragen. Das Ergebnis sieht ähnlich wie bei der 1. Methode aus. Sehr interessant ist dabei die starke Änderung von w_{100} bei geringsten Änderungen von w_0 im Falle des Chaos.

Abbildung 5.1/6 zeigt ein solches Diagramm, das auch als „Feigenbaum"-Diagramm bezeichnet wird. Das BASIC-Programm hat dabei die erste Methode als Grundlage und braucht zur Erstellung dieses Diagramms etwa 1 Stunde und 30 Minuten.

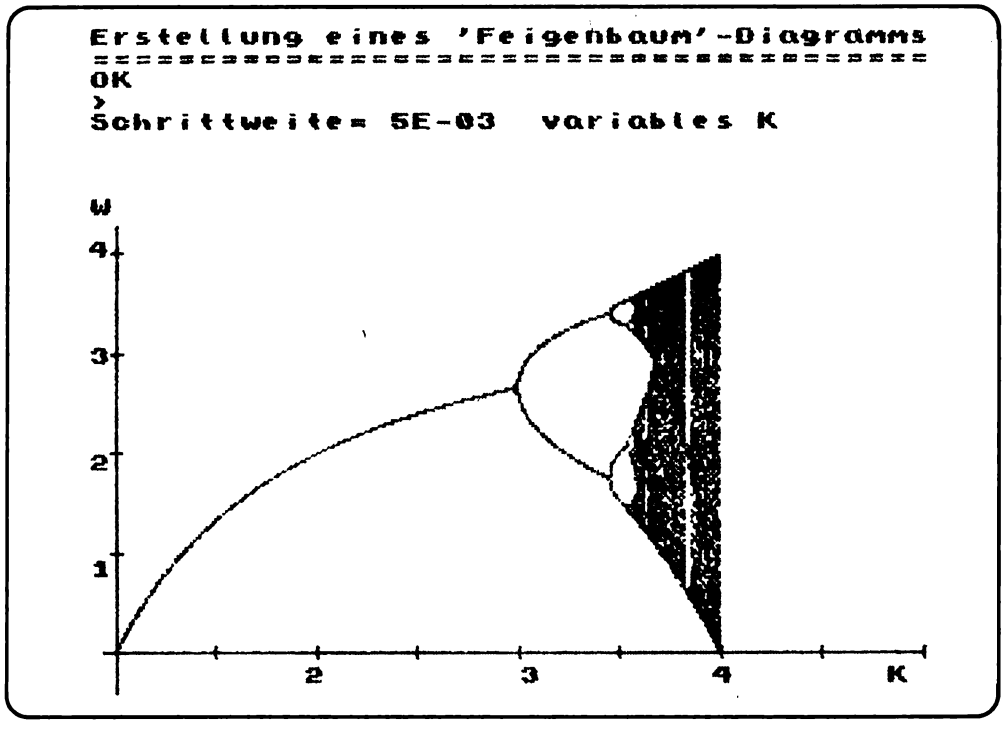


Abb. 5.1/6: Feigenbaum-Diagramm (w_0=.5)

Auch bei diesen Untersuchungen spielen die genannten numerischen Effekte natürlich eine wesentliche Rolle. Gerade im Grenzbereich zum Chaos wirken sich Rundungserscheinungen so aus, daß gleiche Programme auf unterschiedlichen Rechnertypen bzw. bei BASIC-Versionen mit unterschiedlicher Genauigkeit zu gravierend unterschiedlichen Ergebnissen kommen können. Auf der Computerzahlenmenge gelten die uns aus der Mathematik bekannten Rechengesetze nicht uneingeschränkt. Alle bisherigen Beispiele unterstreichen nachhaltig, daß mathematisches Beweisen weiterhin seinen Wert behält, ja sogar noch einen höheren Stellenwert erhält. Trotz dieser Einschränkung bietet die Nutzung des Computers bei der Bearbeitung numerischer Probleme noch viele Möglichkeiten.

Es gibt auch Beispiele, in denen erst der Computer einen Beweis einer mathematischen Behauptung realisierte. Das überzeugendste Beispiel ist wohl der Beweis des Vierfarben-Satzes durch W. Haken und K. Appel aus dem Jahre 1976. Dabei wurden Computer zur Durchmusterung einer endlichen Figuren-Menge eingesetzt. Immer wenn es darum geht, endlich viele Fälle durchzuprobieren, kann der Computer sehr hilfreich sein.

Im Jahre 1949 hat der indische Mathematiker D. R. Kaprekar folgendes herausgefunden:
Wenn *n* eine beliebige natürliche Zahl ist, die in der Dezimalschreibweise vierstellig ist und nicht lauter gleiche Ziffern aufweist, dann führt folgendes Vorgehen nach endlich vielen Wiederholungen auf die Zahl 6174.

Die Ziffern der gewählten vierstelligen Zahl werden so umgestellt, daß einerseits die größtmögliche Zahl und andererseits die kleinstmögliche Zahl entsteht. Die Differenz aus der größtmöglichen und kleinstmöglichen Zahl ist wieder eine vierstellige natürliche

Zahl, die nun als die gewählte Zahl betrachtet wird. Dieser Vorgang wird solange wiederholt, bis die Zahl 6174 entstanden ist. Wird das Verfahren auf 6174 angewendet, so entsteht wieder die Zahl 6174:

$$n = 6174 \begin{array}{l} \nearrow g = 7641 \\ \searrow k = 1467 \\ \hline g - k = 6174 \end{array}$$

Drei-, zwei- und einstellige Dezimalzahlen werden mit ‚führenden' Nullen aufgefüllt, so daß sie auch als vierstellig angesehen werden können.
Der Beweis der Behauptung von Kaprekar ist erbracht, wenn alle möglichen 9990 Startwerte durchprobiert wurden und das beschriebene Vorgehen stets zu 6174 führte.
Es ist ein Algorithmus zu entwickeln, mit dessen Hilfe der mathematische Beweis zur Kaprekar-Zahl 6174 geführt werden kann! Nach der Methode der schrittweisen Verfeinerung läßt sich der Lösungsweg entsprechend Abbildung 5.1/7 vorzeichnen.

Kaprekar-Beweis

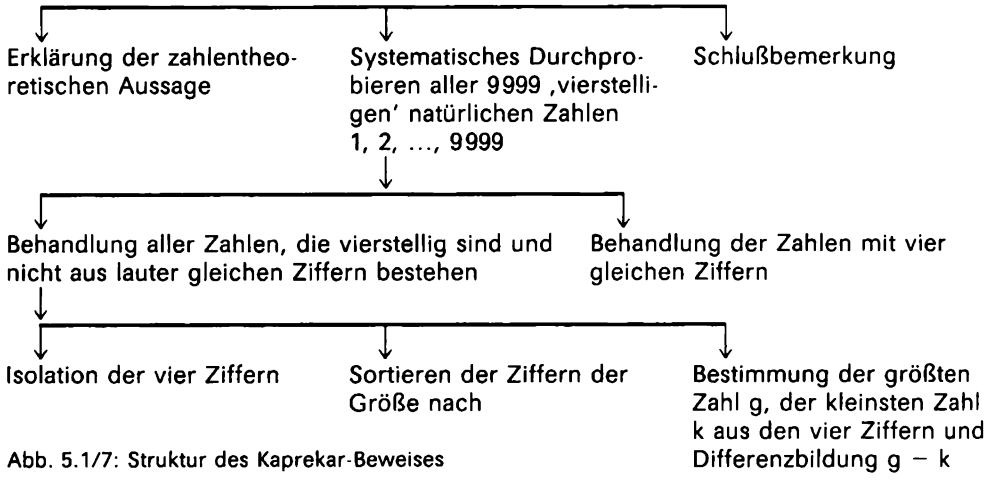


Abb. 5.1/7: Struktur des Kaprekar-Beweises

Auf der Grundlage dieser Modulstruktur werden zuerst die Module auf der untersten Ebene entwickelt.
Zur Isolation von Ziffern aus einer Zahl ist es günstig, die Zahl als Zeichenkette umzuformulieren, da mit der Zeichenkettenfunktion TEILKETTE sehr einfach die Ziffern isolierbar werden. Durch Hinzufügen von ‚führenden' Nullen kann für jede Zahl von 1 bis 9999 die Vierstelligkeit erreicht werden. Der Modul wird gleich in BASIC formuliert:

```
1000 REM Modul 1: Isolation der vier Ziffern==============
1010 REM Eingangsvariable: D - natuerliche Zahl
1020 REM Ausgangsvariablen: Z(1),..,Z(4) - 4 Ziffern von D
1030 REM 'lokale' Variablen: I - Laufvariable
1040 REM        D$ - Zeichenkette zu D
1050 REM  Z$(1),...,Z$(4) - Ziffern als Zeichenkette
1060 REM ------------------------------------------------
1100 D$=RIGHT$("000"+STR$(D),4)
1110 FOR I=1 TO 4
1120     Z$(I)=MID$(D$,I,1)
```

```
1130     Z(I) =VAL(Z$(I))
1140 NEXT I
1990 RETURN
```

Wie im Kapitel 4 dargestellt wurde, gibt es unterschiedliche Sortierstrategien. Hier soll das sogenannte Austausch- oder Bubble-Sort (↗ S. 195) verwendet werden.
Es erfolgt ein Austausch von benachbarten Elemente, bis sie in der richtigen Reihenfolge stehen. Wenn also kein Austausch von benachbarten Elementen mehr notwendig ist, ist der Sortiervorgang abgeschlossen. Deshalb wird eine Merkvariable V eingeführt, die auf 1 gesetzt wird, wenn eine Vertauschung von Elementen stattfand. Bleibt sie – wie zu Anfang – auf 0 gesetzt, so ist der Sortiervorgang beendet.

```
2000 REM Modul 2: Sortieren durch Austausch (Bubble-Sort)=
2010 REM Eingangsvariable: Z(1),...,Z(4) unsortiert
2020 REM Ausgangsvariable: Z(1),...,Z(4) sortiert
2030 REM   d. h.: Z(1)<=Z(2)<=Z(3)<=Z(4)
2040 REM 'lokale' Variable:
2050 REM  V - Merkvariable
2060 REM  I - Laufvariable
2070 REM ------------------------------------------------
2100 REM Anfang der Wiederholschleife
2110 V=0
2120 FOR I=1 TO 3
2130   IF Z(I)>Z(I+1) THEN H=Z(I):Z(I)=Z(I+1):Z(I+1)=H:V=1
2140 NEXT I
2150 IF NOT(V=0) THEN 2110
2990 RETURN
```

Die Bestimmung der größten und der kleinsten Zahl aus den der Größe nach geordneten Ziffern und die Bildung der Differenz dieser beiden Zahlen sind so eng miteinander verbunden, daß sie in einem Modul realisiert werden sollten.

```
3000 REM Modul 3:Bestimmung der groessten und kleinsten Zahl
3010 REM         aus vier Ziffern und Differenzbildung
3020 REM Eingangsvariable: Z(1),...,Z(4) - Ziffern sortiert
3030 REM Ausgangsvariablen:
3040 REM   G   - groesste vierstellige Zahl aus den Ziffern
3050 REM   K   - kleinste vierstellige Zahl aus den Ziffern
3060 REM   D,D$- Differenz G-K
3070 REM 'lokale' Variable: I - Laufvariable
3080 REM ------------------------------------------------
3100 G=0 : K=0
3110 FOR I=1 TO 4
3120     G=G+Z(I)*10^(I-1)
3130     K=K+Z(5-I)* 10^(I-1)
3140 NEXT I
3150 D=G-K
3160 D$=STR$(D) : PRINT RIGHT$("     "+D$,5);
3990 RETURN
```

Die Bestimmung von G und K in den Programmzeilen 3100 bis 3140 liegt zwar aufgrund der Positionsschreibweise
$G = Z(4) * 10^3 + Z(3) * 10^2 + Z(2) * 10 + Z(1)$
nahe, aber sie widerspricht der Regel, daß möglichst Operationen höherer Stufe durch Operationen niederer Stufe ersetzt werden sollen. G und K sind numerisch günstiger berechenbar nach

$G = ((Z(4) * 10 + Z(3)) * 10 + Z(2)) * 10 + Z(1)$ bzw.
$K = ((Z(1) * 10 + Z(2)) * 10 + Z(3)) * 10 + Z(4)$.

Damit können die Zeilen 3100 bis 3140 des 3. Moduls umgeschrieben werden.

```
3100 G=Z(4):K=Z(1)
3110 FOR I=1 TO 3
3120      G=G*10+Z(4-I)
3130      K=K*10+Z(I+1)
3140 NEXT I
```

Beide Realisierungen des Moduls 3 sind zu testen.

Die zu allen Moduln geschriebenen Bemerkungen über Eingangs-, Ausgangs- und ‚lokale' Variable erleichtern das Schreiben von Rahmenprogrammen für einen Test der Module 1, 2 und 3:

```
10 WINDOW 0,31,0,39 : CLS
20 INPUT "D  (vierstellige nat. Zahl):"; D
30 GOSUB 1000 : REM → Isolation der Ziffern
40 FOR I=1 TO 4
50     PRINT ,I,Z(I)
60 NEXT I
70 END
```

Der Test des ersten Moduls zeigt, daß dieser Modul korrekt ist. Da die Ausgangsvariablen des ersten Moduls die Eingangsvariablen des zweiten Moduls sind, wird das Testrahmenprogramm weitergeschrieben:

```
70  GOSUB 2000 : REM → Ziffernsortieren
80  PRINT "sortiert:"
90  FOR I=1 TO 4
100    PRINT Z(I);
110 NEXT I
120 END
```

Auch der zweite Modul ist korrekt. Die Ausgangsvariablen des Moduls 2, die nach der Größe sortierten Ziffern, sind die Eingangsvariablen für den 3. Modul, so daß für den Test des Moduls 3 das Testrahmenprogramm fortgesetzt werden kann!

```
120 PRINT : PRINT : GOSUB 3000 : REM → G,K,D Bestimmung
130 PRINT : PRINT : PRINT " G,K,D:" : PRINT G;K;D;D$
140 END
```

Der Modul erweist sich sowohl in der ersten wie auch in der zweiten Variante als korrekt.
Nach dem Test der Einzelmodule wird das Testrahmenprogramm mit DELETE 10,140 gelöscht, und es kann das Hauptprogramm für den Kaprekar-Beweis geschrieben werden. Der für die Problemlösung nebensächliche Modul „Erklärung der zahlentheoretischen Aussage von Kaprekar" wird vorerst nicht durchprogrammiert, sondern nur durch die Programmzeilen

```
4000 REM Modul 4: Erklaerung der Aussage von Kaprekar
4100 WINDOW 0,31,0,39 : COLOR 0,6 : CLS
4110 PRINT AT(1,10); "Kaprekar-Beweis"
4990 RETURN
```

erfaßt, um den logischen Ablauf des Gesamtprogramms testen zu können. Analoges gilt für die Module ‚Titelbild' und ‚Schlußbemerkung'.
Das Hauptprogramm beginnt mit der Anweisungszeile 100 und beinhaltet bis 190 die Informationen über Variablen. Der Anweisungsteil beginnt mit 200:

```
200 GOSUB 6000 : REM → Titelbild
210 WINDOW 0,31,0,39 : COLOR 7,1 : CLS : LOCATE 14,0
220 PRINT "Ist eine Erklaerung der Kaprekaraussage"
230 PRINT : PRINT TAB (10);"gewuenscht (J/N)?"
240 I$=INKEY$ : IF I$="" THEN 240
250 IF I$="J" THEN GOSUB 4000 : REM→ Text
260 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
270 PRINT AT(0,12); "Kaprekar-Beweis"
280 PRINT : PRINT : PRINT "Zahl N", "Anschlussfolge"
290 WINDOW 3,31,0,39 : COLOR 7,1 : CLS
300 FOR N=1 TO 9998
310   N$=RIGHT$("    "+STR$(N),4)
320   PRINT INK 7;N$;
330   D=N
340   MG=0
350   REM Anfang der Wiederholschleife
360     GOSUB 1000 : REM → Ziffernisol.
370     IF Z$(1)=Z$(2) AND Z$(1)=Z$(3) AND Z$(1)=Z$(4)
THEN MG=1:GOTO 400
380     GOSUB  2000 : REM → Ziffernsort.
390     GOSUB 3000 : REM → G,K,D Bestimm.
400     IF NOT(MG=1 OR D=6174) THEN 360
410     REM Ende der Wiederholschleife
420     PRINT
430 NEXT N
440 GOSUB 5000 : REM → Schlussbemerkung
990 END
```

Der Test des Gesamtprogramms ergibt ein unerwartetes Ergebnis. Für N=1 wird die Anschlußfolge 999, 8991, 8082, 8532, 6174, 6174, ..., 6174, ... produziert, obwohl beim Errei-

chen der Zahl 6174 die Behandlung der Zahl N=1 abgeschlossen werden soll und der Fall N=2 behandelt werden müßte.

Der Modul 3 liegt beim ersten Test des Gesamtprogramms in der Variante 1 vor, bei der G bzw. K nach

```
3100 G=0 : K=0
3110 FOR I=1 TO 4
3120      G=G+Z(I)*10^(I-1)
3130      K=K+Z(5-1)*10^(I-1)
3140 NEXT I
```

berechnet werden. Vermutlich ist D=G–K nicht exakt 6174, obwohl der Computer dies ausgibt. Es wird vorübergehend die Zeile 3160 ergänzt durch:

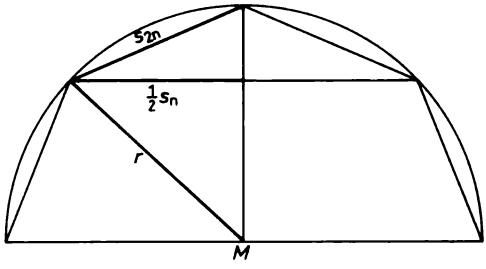
```
PRINT D-6174;
```

Da im Anschluß an die erste Ausgabe von D=6174 für die Differenz D–6174 die Zahl 9.76563E–04 ausgegeben wird und bei den folgenden Gliedern 4.88281E–04, ist klar, warum sich das Programm in einer Endlosschleife befindet. Die notwendigerweise zu erfüllende Forderung der Endlichkeit der Algorithmen, wäre nicht gewährleistet. Damit würde die wohl durchdachte Computer-Beweisführung scheitern.

Die Zahl D hat aufgrund des Algorithmus eine natürliche Zahl zu sein, so daß es hier gerechtfertigt ist, D auf ganze Zahlen zu runden, bevor auf Gleichheit von D mit der Zahl 6174 getestet wird. D=INT(D+0.5) genügt den Rundungsregeln. Diese Wertzuweisung kommt in die Programmzeile 3155 und damit funktioniert das Programm korrekt. Nach etwa 13 Stunden und 30 Minuten ist die Kaprekar-Behauptung unter Computernutzung bewiesen. Nimmt man die numerisch günstigere Variante des Moduls 3, so ist keine Rundung notwendig und der Beweis der Behauptung dauert etwa 8 Stunden und 45 Minuten.
Ein Blick auf den Bildschirm bei der Programmbearbeitung regt zu weiteren zahlentheoretischen Überlegungen an, da offensichtlich jede Anschlußfolge spätestens beim 7. Glied die Zahl 6174 erreicht hat. Das Programm könnte z. B. um eine Registrierung erweitert werden, um zu ermitteln, wieviele von den 9990 Anschlußfolgen nach dem 1., 2., ..., 7. Glied erstmalig die Zahl 6174 erreichen.
Darüber hinaus stellt sich die Frage, ob es auch in anderen B-alsystemen (B>=2) – außer im Dezimalsystem – solche ‚Kaprekarzahlen' gibt.
Wir haben in den bisherigen Beispielen aus dem Gebiet Numerik schon einige Fallen kennengelernt, die der Computer – genauer gesagt, der BASIC-Interpreter – besonders hier bereithält. Das waren aber bisher leider noch nicht alle. Neben der begrenzten Stellenzahl und dem damit verbundenen Rundungsfehler gibt es auch *Verfahrensfehler.* Diese treten auf, wenn wir einen Algorithmus zur Berechnung einer Irrationalzahl nach einer bestimmten Anzahl von Schritten abbrechen.

6 Erstellen Sie Algorithmus und Programm zur näherungsweisen Berechnung der irrationalen Zahl PI nach dem Verfahren von Archimedes!
Archimedes von Syrakus (um 250 v. u. Z.) hat mit folgendem Verfahren den Wert von PI schon recht genau bestimmt:
Ausgangspunkt war ein Kreis, das einbeschriebene und das umschriebene *n*-Eck. Durch schrittweise Verdopplung der Eckenanzahl und jeweilige Berechnung des Umfangs der

n-Ecke nähert sich der berechnete Wert schrittweise 2*PI an. Das Verfahren wurde mit einem Sechseck begonnen.



$$s_{2n} = \sqrt{2 * r^2 - r * \sqrt{4r^2 - s_n^2}}$$

Abb. 5.1/8: Ableitung des mathematischen Modells

Ausgehend vom im Kreis einbeschriebenen n-Eck kann man das in Abbildung 5.1/8 gezeigte mathematische Modell erstellen.
Verwendet man den Einheitskreis mit r = 1, so vereinfacht sich das Modell zu

$$s_{2n} = \sqrt{2 - \sqrt{4 - s_n^2}}$$

Gehen Sie bei der Erstellung des Algorithmus von einem einbeschriebenen Viereck aus!

```
10  REM  PI nach ARCHIMEDES
100 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
110 PRINT "N-Eck","Wert"
120 S = SQR(2) : ZN = 4 : REM Anfangswerte
130 FOR N=3 TO 20
140    S=SQR(2-SQR(4-S*S))
150    PRINT ZN,ZN*S
160    ZN=ZN*2
170 NEXT N
180 END
```

Starten Sie dann das Programm! Die Hardcopy der Ergebnisse der recht einfachen Programmversion zeigt Abbildung 5.1/9.

```
                N-Eck          PI

                 4             3.06147
                 8             3.12144
                16             3.13654
                32             3.14028
                64             3.14105
               128             3.1409
               256             3.13996
               512             3.13748
              1024             3.1225
              2048             3.08221
              4096             2.44949
?FC ERROR IN 140
OK
>
```

Abb. 5.1/9: Bildschirmausgabe von „Archimedes"

Bei den ersten Zeilen können wir eine schrittweise Annäherung feststellen, die im 7. Schritt mit dem Wert 3.14105 am dichtesten an der Zahl PI=3.141592... liegt, dann aber wieder schlechter wird. Im 14. Iterationsschritt bricht das Programm mit der Ausschrift

```
?FC ERROR IN 140
```

plötzlich ab.
Bei der Suche nach der Fehlerquelle lassen wir uns im Direktmodus den aktuellen Wert von S anzeigen und berechnen dann die Werte folgender Terme.

```
PRINT S
5.9802E-04
T1 = 4 - S * S
T2 = SQR(T1)
T3 = 2 - T2
PRINT T1, T2, T3
```

Auf dem Bildschirm wird ausgegeben: 4 2 –2.38419E–07.
Aufgrund der ersten beiden Ergebnisse müßte der 3. Wert Null sein. Er ist aber negativ! Die Anwendung der SQR-Funktion führt deshalb zum FC-ERROR (illegal function call – fehlerhafter Funktionsaufruf).

Die Fehlerursache ist also gefunden! Die Ursache liegt in der Rechenvorschrift in Zeile 140, da dort zwei annähernd gleich große Werte voneinander subtrahiert werden. Die Rundungsfehler summieren sich bei jedem Durchlauf und führen schließlich zum obigen Fehler. Davon können Sie sich ganz einfach überzeugen, wenn Sie die Zeile 140 durch

```
140     S=SQR(ABS(2*SQR(4-S*S)))
```

ersetzen. Die ausgedruckte Folge divergiert gegen Unendlich.

Ein Algorithmus, bei dessen Abarbeitung es zu einer starken Aufsummierung von *Rundungsfehlern* kommt, wird als *numerisch instabil* bezeichnet, andernfalls als *numerisch stabil.*

Da unser Algorithmus grundsätzlich mathematisch richtig ist, muß ein Weg zur Vermeidung dieses Verfahrensfehlers gesucht werden. Der Ausdruck in Zeile 140 wird umgeformt.

$$s_{2n} = \sqrt{2 - \sqrt{4 - s_n^2}} * \frac{\sqrt{2 + \sqrt{4 - s_n^2}}}{\sqrt{2 + \sqrt{4 - s_n^2}}}$$

und dann ergibt sich unter Beachtung der Rechenregeln

$$s_{2n} = \frac{s_n}{\sqrt{2 + \sqrt{4 - s_n^2}}}$$

Dieser Ausdruck wird als neue Zeile 140 in unserem Programm eingesetzt und dieses erneut mit RUN gestartet. Ein Blick auf den Bildschirm zeigt eine ständige Annäherung an den Wert PI, bis ab 10. Durchlauf keine Veränderung in den angezeigten Ziffernstellen mehr auftritt.
Wir wollen dieses Programm noch einmal modifizieren, um es zur Ermittlung des Einflus-

ses von Rundungen auf den Ergebniswert zu nutzen. Dazu muß in jedem Ausdruck, in dem *s* ermittelt bzw. mit *s* gerechnet wird, eine Rundungsvorschrift eingebaut werden. Um den Änderungsaufwand möglichst gering zu halten, definieren wir die Rundungsvorschrift als Nutzerfunktion und rufen diese nur in den entsprechenden Ausdrücken auf.

```
20 DEF FNR(P)=INT(P*K+.5)/K
30 INPUT "Stellenzahl (10,100,1000 usw.)";K
...
120 P=SQR(2) : S=FNR(P) : ZN=4
...
140     P=SQR(4-S*S) : P=SQR(2+FNR(P)) : P=S/FNR(P)
145     S=FNR(P) : P=ZN*S
150     PRINT N,ZN,FNR(P)
...
```

Lassen wir das Programm laufen und geben nacheinander für K die Werte 0, 1, ..., 7 ein, so können wir eine Tabelle folgender Art notieren:

	k						
n	0	1	2	3	...	6	7
3	3	3.1	3.06	3.061	...	3.06147	3.06147
4	3	3.1	3.12	3.121	...	3.12145	3.12145
5	.	.	3.14	3.136	...	3.13655	3.13655
6	.	.	3.14	3.14	...	3.14033	3.14033
7	.	.	.	3.141	...	3.14128	3.14128
8	.	.	.	3.141	...	3.14151	3.14151
9	.	.	.	.	...	3.14157	3.14157
10	.	.	.	.	...	3.14159	3.14159
11	.	.	.	.	...	3.14159	3.14159
12	.	.	.	.	...	.	.

(In der Tabelle bedeutet ein einzelner Punkt in einer Spalte, daß sich der vorhergehende Wert wiederholt.)
Mit dieser Rundungsfunktion in der Zeile 20 haben wir eine allgemeingültige Rundungsvorschrift, die in allen Programmen – im Normalfall mit einem problemabhängigen festen Rundungsfaktor K – verwendet werden kann. Dabei sollte man bei mehr als 3 Stellen, nach denen gerundet werden soll, den Faktor K nicht als Zehnerpotenz verwenden. Sie können im obigen Programm probehalber statt K=100000 einmal K=10∧5 eingeben und die Werte mit obiger Tabelle vergleichen. Wieder ist der Rundungsfehler wirksam geworden! Geben Sie dazu im Direktmodus

```
PRINT 100000-10∧5    <ENTER>
```

ein. Auf dem Bildschirm erscheint nicht die erwartete 0, sondern

```
-,015625
```

Mit dem Verfahren von Archimedes haben wir einen Iterationsalgorithmus kennengelernt, der aber sehr speziell auf dieses Problem zugeschnitten ist. Ein Algorithmus, der die wesentlichen Eigenschaften des allgemeinen Iterationsalgorithmus

- variierbarer Startwert als erste Näherung,
- Verwendung dieses ersten Näherungswertes zur Berechnung des nächsten Näherungswertes,
- Algorithmus ist auf mehrere gleichartige Probleme anwendbar,

besitzt, ist der des Heron von Alexandria (1. Jahrhundert) zur Berechnung von Wurzeln.

7 Es ist die Quadratwurzel der Zahl A durch ein Programm unter Verwendung des Heronschen Verfahrens zu berechnen.

Der griechische Mathematiker hatte herausgefunden, daß die 3. Wurzel von *a* der Kantenlänge eines 3-dimensionalen Würfels vom Inhalt *a* entspricht. Diese Kantenlänge wird mittels eines Ausgleichprozesses ermittelt, der in verkürzter Form durch die rekursive Folge

$x_{i+1} = 1/n((n-1)x_i + a/x_i)$

beschrieben wird.
Auf die Berechnung der Quadratwurzel reduziert, ergibt sich mit $n=2$

$x_{i+1} = 1/2(x_i + a/x_i)$

$x(i+1)$ ist also das arithmetische Mittel von $x(i)$ und $a/x(i)$.
Die Quadratwurzel aus der Zahl *a* ist als Seitenlänge eines Quadrates vom Flächeninhalt *a* interpretierbar. Das Heron-Verfahren ermittelt das gesuchte Quadrat über eine Folge von Rechtecken, die alle den Flächeninhalt *a* besitzen. Das Ausgangsrechteck ist im Prinzip beliebig vorgebbar.

Im zu erstellenden Algorithmus wählen wir die Seitenlängen mit $x=a$ und $y=1$. Zur Beschränkung der Zahl der Näherungsschritte geben wir eine Genauigkeitsschranke *E* vor. Unter diesen Voraussetzungen wird der folgende Näherungsprozeß abgearbeitet:

1. die Seite *x*(neu) wird durch das arithmetische Mittel der Seiten *x*(alt) und *y*(alt) errechnet,
2. die Seite *y*(neu) wird so festgelegt, daß der Flächeninhalt des neuen Rechtecks gleich *a* ist: $y=a/x$,
3. solange ABS(x*x–a)>E wieder (1), sonst Ende.

Zur Veranschaulichung sollte die Näherung nicht nur wertmäßig, sondern auch grafisch verfolgbar sein!

Eingabe:	Zahl *a* und Genauigkeitsschranke *E*
Verarbeitung:	Abarbeitung obiger Schrittfolge
Ausgabe:	Seitenlängen der Rechtecke
	Endwert des Verfahrens
	grafische Darstellung

Eine erste Version des nach diesem Struktogramm erstellten Programms wird mit A=3 und der Schranke E=.00001 getestet. Das Ergebnis ist aus der Bildschirmkopie in Abbildung 5.1/11 ersichtlich. Sowohl Tabelle als auch die Grafik sehen gut aus. Ein weiterer Test mit A=8 und gleichem E bringt auch richtige Ergebnisse. Geben wir aber beispielsweise A=8 und E=.0000001 ein, so ändert sich das Bild, das Programm schreibt fortlaufend die gleichen Werte in die 2 Spalten – nachdem der schon bekannte Wert für die

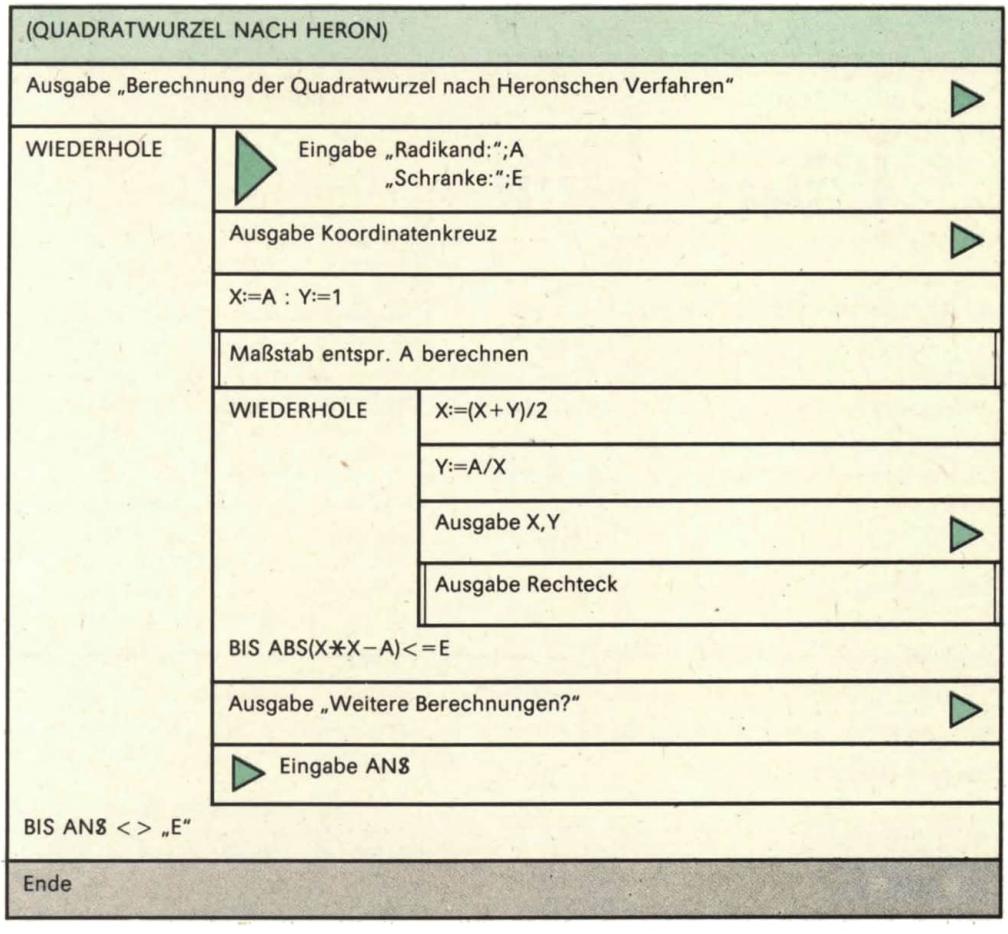


Abb. 5.1/10: Struktogramm „Heronsches Verfahren"

Quadratwurzel aus 8 ermittelt wurde – und kommt nicht zum Ende. Die Ursache liegt in einem Eingabefehler, der nicht vom Programm abgefangen wird. Die Schranke .0000001 unterschreitet die kleinste darstellbare 6stellige Zahl, so daß der Rechner auf Null für E rundet. Durch Rundungsfehler bei der Differenz annähernd gleich großer Werte im Testausdruck entsteht auf dessen linker Seite aber nicht immer Null, so daß ein Abbruch des Programms von außen erfolgen muß.

Der Algorithmus sollte durch einen Test der Schranke auf >0 und eventueller Anforderung eines neuen Wertes ergänzt werden. Eine weitere notwendige Änderung betrifft die Grafik. Ein Wert A>10 erbringt in der ersten Programmversion eine sehr dichte Aufeinanderfolge der Einheitenstriche auf *x*- und *y*-Achse. Man könnte beispielsweise eine Staffelung für den Maßstab vorsehen. Etwa in der Art:

A <= 10	jeder Teilstrich wird gezeichnet,
10 < A <= 20	jeder 2. Teilstrich wird gezeichnet,
20 < A <= 50	jeder 5. Teilstrich wird gezeichnet und
A > 50	unterdrücken der Grafik, da sie unübersichtlich wird, oder Tabelle und Grafik in zwei Bildern bringen.

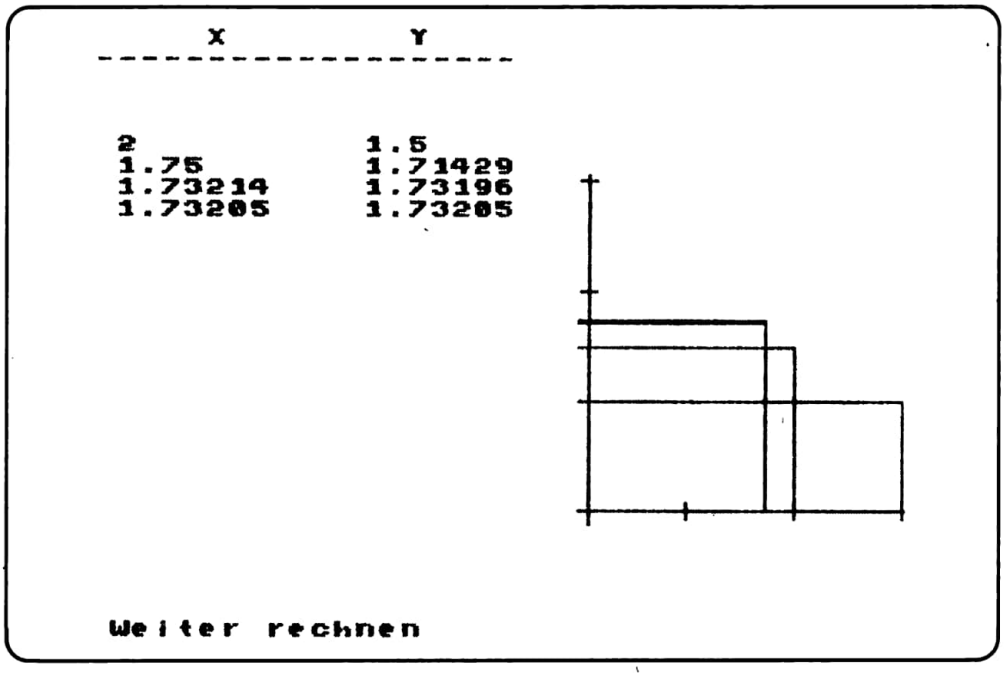


Abb. 5.1/11: Ergebnis des ersten Tests von „HERON"

Zusammenfassung

Wir haben folgende Fehlerarten bei der Anwendung des Computers zum Lösen numerischer Probleme kennengelernt:

- **Eingabefehler** — Es werden Werte an den Algorithmus übergeben, für die dieser nicht gültig ist, die zu unvorhersehbaren Folgen führen können. Vermeidung ist durch entsprechende Prüfungen der Eingabedaten möglich.
- **Rundungsfehler** — Sie beruhen auf der begrenzten Ziffernzahl des Computers, die ihm zur Zahlendarstellung zur Verfügung stehen. Die Computerzahlenmenge stellt nur eine Teilmenge aller Zahlen dar, so daß nicht alle mathematischen Gesetze uneingeschränkt gelten!
- **Verfahrensfehler** — Entstehen dann, wenn ein Algorithmus zur Berechnung einer Irrationalzahl nach einer bestimmten Anzahl von Schritten abgebrochen wird.

Die Rechengenauigkeit eines Computers ergibt sich zu
$0.5 * 10^{1-t}$,
wobei t die Anzahl der Ziffernstellen zur Darstellung einer Zahl ist. Diese maximal erreichbare Genauigkeit muß bei der Festlegung von Abbruchkriterien iterativer Algorithmen beachtet werden.

● Aufgaben

1. Es ist ein Programm zu erstellen, das einen Näherungswert der Eulerschen Zahl e mit Hilfe der Reihenentwicklung

   $$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

   errechnet. Bis zu welchem Wert von n ist es sinnvoll zu rechnen?

2. Es ist ein Programm zu erstellen, das das „Feigenbaum"-Diagramm nach der im Text (↗ S. 222) genannten 1. Methode erzeugt!

3. Erarbeiten Sie ein Programm zur Erstellung des „Feigenbaum"-Diagramms nach der im Kapitel 5. (↗ S. 222) genannten 2. Methode!

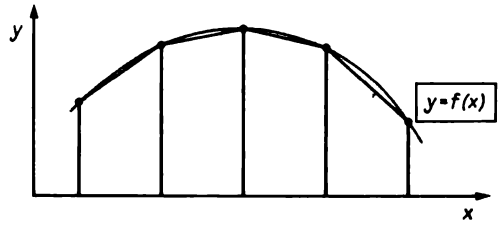
4. Programmieren Sie den Algorithmus von Heron mit den im Kapitel 5. (↗ S. 233) angegebenen Modifikationen bezüglich Maßstab und Beschriftung des Koordinatensystems!

5. Modifizieren Sie das Programm zur vierstelligen Kaprekarzahl 6174 so, daß untersucht werden kann, ob es
   a) für dreistellige Dezimalzahlen,
   b) für fünfstellige Dezimalzahlen
   auch jeweils eine Kaprekarzahl gibt! Ermitteln Sie diese gegebenenfalls mit Ihrem modifizierten Programm!

6. Erstellen Sie einen Algorithmus und ein Programm zur rechentechnischen Realisierung des „Regula falsi"-Verfahrens (↗ z. B. in [10] S. 73)!

7. Erstellen Sie ein Programm zur Realisierung der Bisektionsmethode zur Lösung nichtlinearer Gleichungen (↗ in [10] S. 86)!

8. Im Kapitel 2 wurde das Beispiel zur Berechnung des Flächeninhalts zwischen dem Graph einer Funktion und der x-Achse im Intervall $<a,b>$ mittels Ober- und Untersummen von Rechtecken – die sogenannte Rechteckregel verwendet.

   In gleicher Weise ist ein Programm zu erstellen, das aber die Trapezregel zur näherungsweisen Flächenberechnung verwendet (↗ in [10] S. 128)!
   Testen Sie Ihr Programm mit den Funktionen $y = x^2$ und $y = 1/SQR(1 + x^4)$ im Bereich von 0 bis 1 und vergleichen Sie die Ergebnisse mit denen bei Verwendung der Rechteckregel!

9. Erstellen Sie ein Programm, das dem Anwender die Möglichkeit der Berechnung des Wertes von π nach verschiedenen Methoden bietet!
   a) Berechnung nach J. Machin
      PI = 16 arctan (1/5) − 4 arctan (1/239)
   b) Berechnung nach C. F. Gauss
      PI = 48 arctan (1/18) + 32 arctan (1/57) − 20 arctan (1/239)
   c) Berechnung nach C. Störmer
      PI = 24 arctan (1/8) + 8 arctan (1/57) + 4 arctan (1/239).
   Vergleichen Sie die Verfahren bezüglich Genauigkeit und Schnelligkeit untereinander und mit dem Verfahren von Archimedes!

5.2. Probleme der elementaren Stochastik

Jeden Sonntagabend beobachten Hunderttausende die Ziehung der Gewinnzahlen in den unterschiedlichen Spielarten von Toto und Lotto. Viele sind sich bewußt, daß ihre Gewinnchance verschwindend klein ist, trotzdem hoffen sie im Stillen, daß das Glück – besser der Zufall – ihnen den großen Gewinn ins Haus bringt. Der Reiz liegt darin, daß es weder abschätzbar noch berechenbar ist, welche Zahlen wirklich gezogen werden.
Aber nicht nur bei Toto und Lotto spielt der Zufall eine große Rolle. Der Zufall ist immer dann im Spiel, wenn für einen betrachteten Vorgang die Bedingungen und Voraussetzungen nur teilweise bekannt oder nicht eindeutig bestimmbar sind, wenn also mehrere Möglichkeiten für den Ausgang des Geschehens bestehen. Denken wir nur an den täglichen Wetterbericht. Es liegt nicht am Unvermögen der Meteorologen, wenn es entgegen der Vorhersagen doch plötzlich regnet. Es sind noch nicht alle Einflüsse auf die Wetterentwicklung bekannt bzw. berechenbar, so daß es immer wieder zu fehlerhaften Vorhersagen kommt.
Auch in anderen Bereichen des täglichen Lebens wird häufig vom Zufall gesprochen. So wurde zum Beispiel nach einer Panne ein neuer Reifen auf das Vorderrad des Fahrrades aufgezogen, der normalerweise hunderte Kilometer halten müßte. Trotzdem können wir ‚zufällig' nach wenigen Kilometern schon wieder einen Defekt haben, weil ein Splitter in den Reifen eingedrungen ist.
Es muß uns aber bewußt sein, daß es den absoluten Zufall nicht gibt, auch das Zufällige hat seine Ursache. Beispielsweise den Splitter, der in den Reifen eingedrungen ist.
Die Kleincomputer bieten uns mit der Funktion RND die Möglichkeit, uns mit dem Zufall etwas näher zu beschäftigen. Es wurde schon darauf hingewiesen, daß RND Werte zwischen 0 und 1 liefert, die mit einem besonderen Verfahren berechnet werden.

1 Es ist ein Programm zu erstellen, mit dessen Hilfe die Zufallszahlenerzeugung für einen Würfel mittels

- RND
- Lehmer-Folge (z=k*z–INT(k*z)) mit k=997
- Weyl-Folge (z=i*a–b*INT(i*a/907))/907 mit a=811, b=907

durchgeführt wird. Es soll ein Startwert vorgegeben werden. Der Vergleich soll mit einer Stichprobe < 250 Würfe erfolgen.
Es wurde schon einmal erläutert (↗ S. 116), wie unter Verwendung von RND die Augenzahlen eines Würfels erzeugt werden können. Diese Vorschrift lautete
Z = INT(RND(1) * 6) + 1.
Damit sind alle notwendigen Angaben vorhanden, und der Algorithmus aus der Aufgabenstellung ist klar erkennbar (Abb. 5.2/1).
Der Unteralgorithmus HAEUFZAHLEN bringt nur die Ausgabe der angefallenen Häufigkeitswerte der einzelnen Würfelanzeigen und stellt somit keine neue Teilaufgabe dar. Bei der Darstellung der Häufigkeit in einem Streifendiagramm sind aber einige Dinge zu beachten, die hier kurz genannt seien:

- Der größte Wert bestimmt den Darstellungsmaßstab im Diagramm,
- Auswahl des Darstellungsmittels in Form von Zeichen- oder Pixelfolgen,
- senkrechte oder waagerechte Darstellung der Streifen des Diagramms.

Die im Struktogramm angegebene Höchstzahl für die Würfe ist willkürlich festgelegt worden. Bei einer wesentlichen Erhöhung sollte man das Programm um die Ausgabe eines Hinweises über die Arbeit des Rechners erweitern, z. B. um die Anzeige des Laufindex und dessen zu erreichenden Endwert.

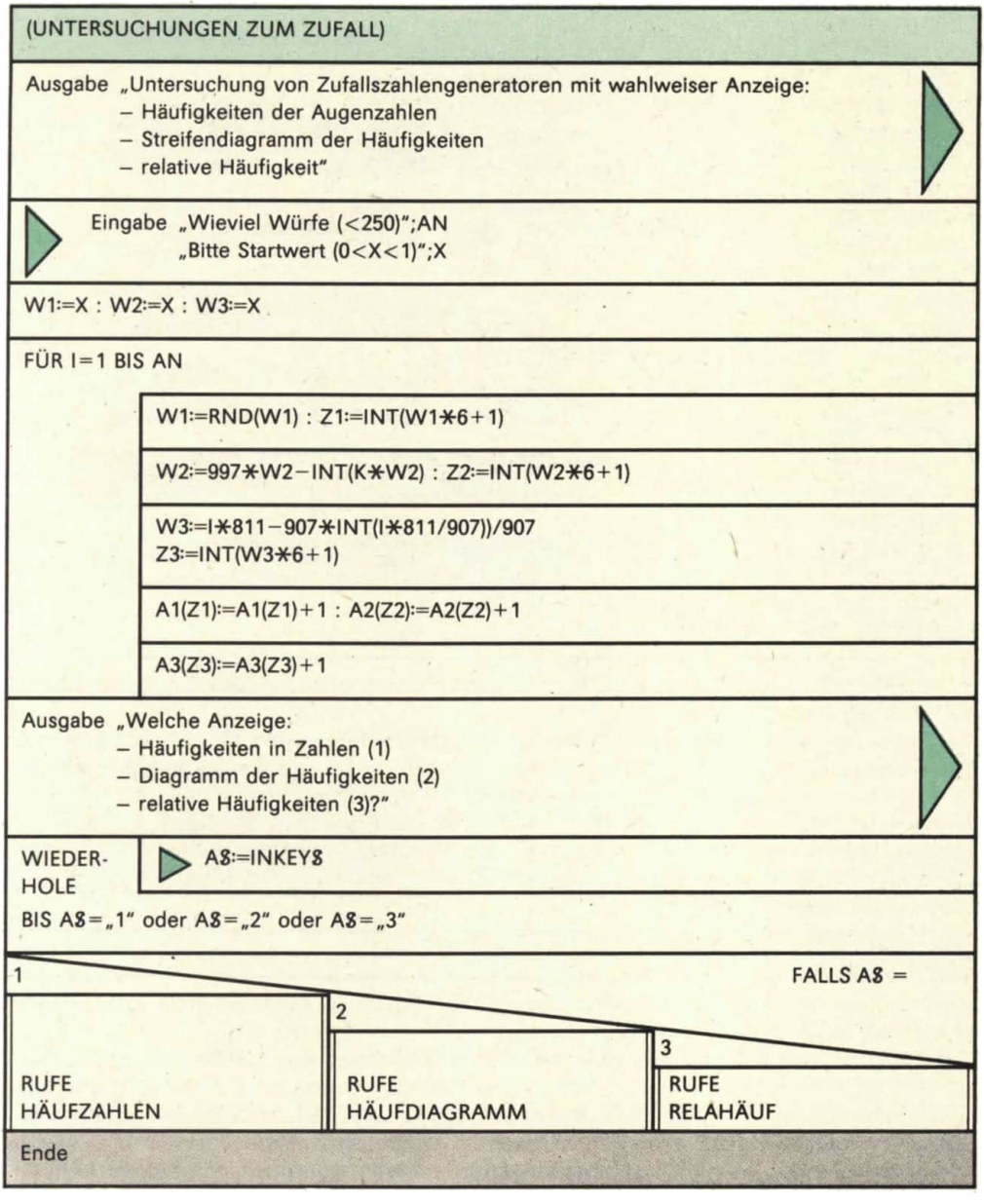


Abb. 5.2/1: Struktogramm „Zufall"

Geht man bei der Untersuchung der angegebenen Fragen von dem oberen Grenzwert 250 und einer etwa Gleichverteilung der möglichen Würfelwerte aus, so kann man das Streifendiagramm ohne Schwierigkeiten sowohl waagerecht als auch senkrecht (Balkendiagramm) zeichnen lassen. Unter Verwendung des LINE-Befehls kann man die Streifen zeichnen lassen.

```
2000 REM ===== UP HAEUFDIAG =====
2010 REM Zeichnen des Streifendiagramms
2020 REM 5 Bildpunkte -> 1 Einheit
2030 WINDOW 0,31,0,39 : COLOR 1,7 : CLS
2040 PRINT AT(1,3);"Untersuchung Zufallszahlengeneratoren"
2050 PRINT AT(3,12);" STREIFENDIAGRAMM"
2060 PRINT AT(4,3);"-----------------------------------"
2070 LINE 15,38,15,215
2080 YH=215
2090 FOR I=1 TO 6
2100   FOR J=1 TO 9
2110     LINE 15,YH-J,A1(I)*5,YH-J,F1
2120     LINE 15,YH-(8+J),A2(I)*5,YH-(8+J),F2
2130     LINE 15,YH-(16+J),A3(I)*5,YH-(16+J),F3
2140   NEXT J
2150   YH=YH-32
2160 NEXT
2170 PRINT AT(6,1);"1"       :  PRINT AT(10,1);"2"
2180 PRINT AT(14,1);"3"      :  PRINT AT(18,1);"4"
2190 PRINT AT(22,1);"5"      :  PRINT AT(26,1);"6"
2200 PRINT AT(30,15);"Weiter (J/N)
2210 AN$=INKEY$ : IF AN$="" THEN 2210
2290 RETURN : REM --------------------------------
```

In der Abbildung 5.2/2 wird eine Bildschirmkopie der Ausgabe des Unteralgorithmus HAEUFDIAG gezeigt.

Die Realisierung des Unteralgorithmus RELAHAEUF bereitet keine Probleme, wenn man sich daran erinnert, daß die relative Häufigkeit das Verhältnis der Einzelhäufigkeit zur Gesamtzahl der Versuche ist und auch in Prozent angegeben werden kann.

Wenn mehrere Tests des Programms mit immer dem gleichen Startwert durchgeführt wurden, wird aufgefallen sein, daß immer die gleichen Zahlenfolgen entstehen. Dieser Effekt kann bei der Nutzung der Funktion RND vermieden werden, wenn man nach dem Start des Programms die Funktion

```
RANDOMIZE
```

aufruft. Sie sorgt dafür, daß bei jedem neuen Start durch die Funktion RND unterschiedliche Zahlenfolgen geliefert werden.

Zufallszahlen sind aber nicht nur für Würfelspiele wichtig! Auch in bestimmten numerischen Verfahren werden Zufallszahlen verwendet. Ein klassisches Beispiel ist die Monte-Carlo-Methode, die beispielsweise zur näherungsweisen Flächenberechnung unter komplizierten Funktionsgraphen oder auch zur näherungsweisen Bestimmung von π verwendet werden kann. Für das erstgenannte Problem haben wir schon einen Algorithmus – die Unter- und Obersummenberechnung – kennengelernt. Damit bietet sich ein Vergleich zwischen beiden Verfahren an.

2 Mittels der Monte-Carlo-Methode ist die Fläche zwischen dem Graph der Funktion $y=f(x)$ im Intervall $<a, b>$ und der x-Achse zu bestimmen. Die Funktion ist in diesem Intervall stetig, und die Funktionswerte sind dort nicht negativ (Abb. 5.2/3).

Man legt einen Wert m fest, der mindestens so groß ist, wie alle im Intervall $<a, b>$ auftretenden y-Werte. F sei der Flächeninhalt des Rechtecks $a, b, c, d,$ und I sei der Flächeninhalt des schraffierten Teilstücks.

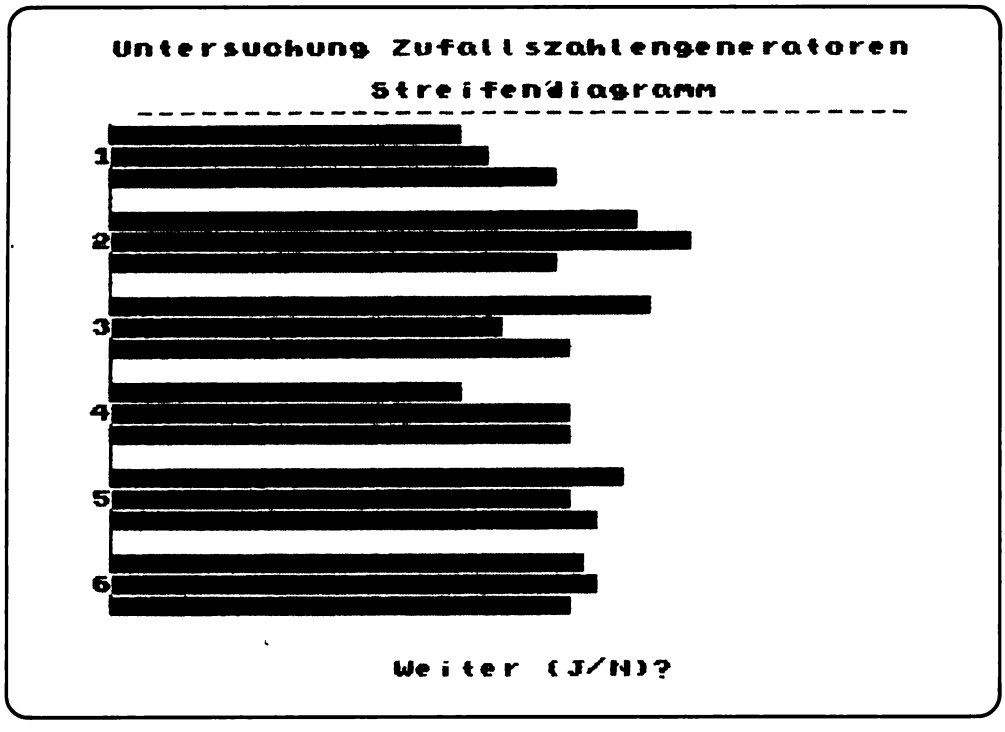


Abb. 5.2/2: Bildschirmkopie von „HAEUFDIAG"

Man stelle sich nun vor, daß das Rechteck wahllos von Pfeilen getroffen wird. Setzt man die Anzahl der Pfeile A_F, die die Rechteckfläche treffen, zu der Anzahl der Pfeile A_I, die die schraffierte Fläche einschließlich Begrenzung treffen, ins Verhältnis, so gilt näherungsweise

$$\frac{I}{F} = \frac{A_I}{A_F} .$$

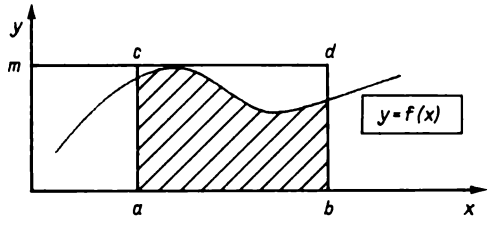


Abb. 5.2/3: Ansatz für Monte-Carlo-Methode

Daraus läßt sich I berechnen, wenn die anderen Werte bekannt sind. Die Rechteckfläche kann exakt berechnet werden.
Zur Bestimmung der Werte A_F und A_I wird der Zufallszahlengenerator verwendet. Dabei liefert der Zufallszahlengenerator die Koordinaten x und y der Auftreffpunkte der Pfeile. Der Wertebereich wird dabei so eingestellt, daß nur Koordinatenwerte der Rechteckfläche erzeugt werden.
Unter diesen Voraussetzungen trifft ein Pfeil die schraffierte Fläche, wenn gilt $y <= f(x)$.

Die Treffer A_F und A_I der geworfenen Pfeile werden durch zwei Zähler festgehalten. Die Anzahl der gewünschten Pfeilwürfe soll über eine Nutzereingabe bestimmbar sein.

Der Algorithmus in Abbildung 5.2/4 ist in der Weise vereinfacht, daß sowohl eine bestimmte Funktion – $y=x*x$ – und das Einheitsquadrat als Rechteckfläche im Programm fest verankert wurden.

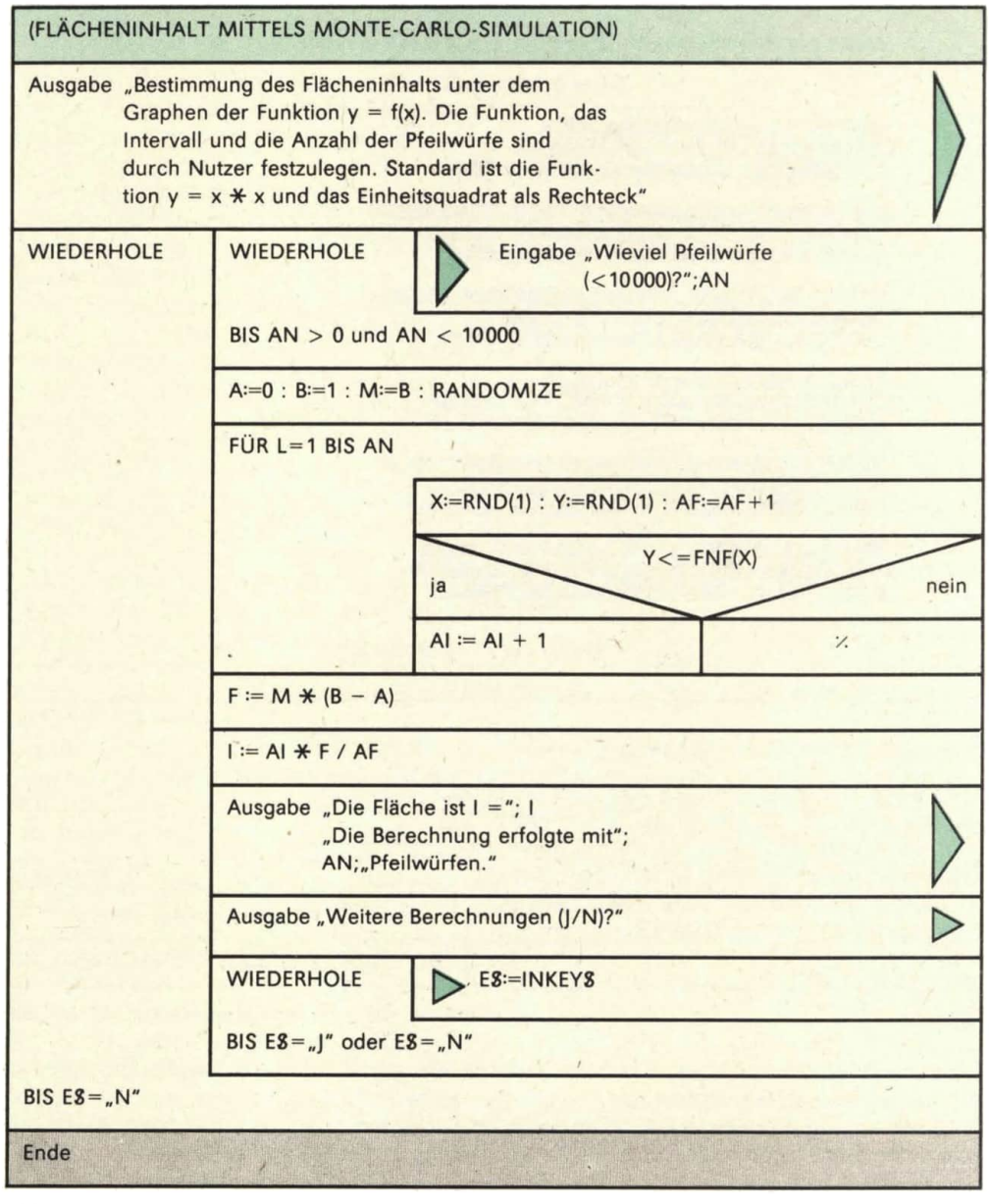


Abb. 5.2/4: Struktogramm zu „Fläche mit Monte-Carlo"

Die im Algorithmus angegebene Grenze von <10000 Würfen ist festgelegt worden, um die Wartezeit auf das Ergebnis nicht unnötig zu verlängern. Wenn man solch große Werte wirklich verwenden will, dann sollte man den Algorithmus und das Programm um die Ausgabe eines Hinweises an den Nutzer erweitern, damit dieser erkennt, daß der Rechner noch für ihn arbeitet. Im Programm, das die Vorlage für die Hardcopy in Abbildung 5.2/5 erzeugte, ist dies berücksichtigt worden.

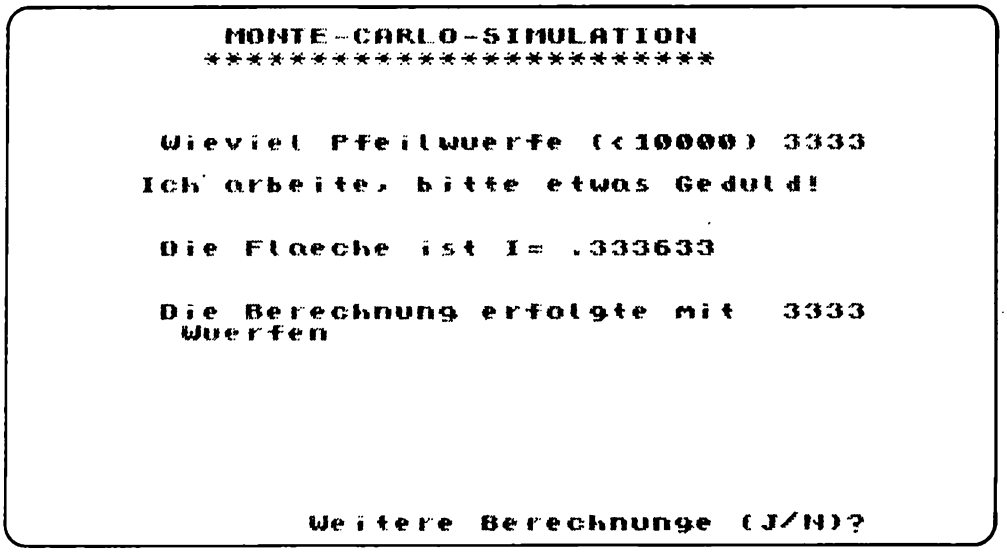


Abb. 5.2/5: Hardcopy des Programms „Fläche mit Monte-Carlo"

Soll der Algorithmus allgemeiner gefaßt sein, so muß die Definition der Funktion und des Intervalls dem Nutzer angeboten werden. Dazu gibt es beispielsweise folgende Möglichkeit:

- *Ausgabe:* Geben Sie die Funktion in folgender Weise ein:
  zzz DEF FNF(X)=X*X
  und starten Sie das Programm wieder mit
  GOTO zzz.
- *Unterbrechen* des Programmlaufs mit Abwarten auf Nutzereingabe. Nach Eingabe der Funktionszeile und Start des Programms
- *Ausgabe:* Bitte Intervallgrenzen eingeben!

Eine weitere Verbesserung könnte darin bestehen, eine grafische Ausgabe zu gestalten. Dabei könnte das zu untersuchende Intervall des Graphen mit einem gewissen Randbereich zuerst gezeichnet werden. Während der Berechnung der einzelnen Pfeilwürfe werden die Zielpunkte mit PSET in die Grafik gesetzt, so daß im Endeffekt auch ein grafischer Eindruck der Verteilung entsteht.

Zufallszahlen haben aber nicht nur in mathematischen Verfahren große Bedeutung erlangt. Auch bei vielen praktischen Problemen treten Ereignisse nicht immer zu bestimmten, berechenbaren Zeitpunkten ein, sondern zufällig.
In größeren Industriebetrieben werden vorbeugende Instandhaltungsmaßnahmen an hochproduktiven Anlagen regelmäßig in bestimmten Abständen durchgeführt. Beispiele können wir häufig in der Tagespresse lesen, wenn von planmäßigen Reparaturen in Kraftwerken oder Tagebauen berichtet wird. Die Zeitabstände zwischen solchen planmäßigen Wartungs- und Reparaturarbeiten wurden früher allein aus Erfahrungswerten heraus festgelegt. Heute verwendet man dazu häufig Ergebnisse von Simulationen zu sogenannten Warteschlangen.
So wie es bei der Problematik der vorbeugenden Wartung nicht möglich ist, einen Störfall grundsätzlich zu vermeiden, so gibt es auch viele andere Beispiele für gleichgelagerte Probleme. Mittels Wahrscheinlichkeitsmodellen kann man diese Probleme mit einer gewissen Sicherheit beschreiben, niemals mit deterministischen Modellen. Aufgrund von

Erfahrungswerten kann man gewisse Wahrscheinlichkeiten dafür angeben, wann beispielsweise ein Auto an eine Kreuzung kommt, wann ein Telefonanruf in einer Vermittlung eintrifft oder wie der Kundenstrom an einer Kasse, in einer Sparkasse oder einer Kaufhalle verläuft. Ein Hilfsmittel, um auf der Grundlage solcher Erfahrungswerte doch wahrscheinlich zutreffende Aussagen zu erhalten, sind probabilistische Simulationen (Simulationen unter Verwendung von Wahrscheinlichkeitswerten), in denen mittels Zufallszahlengenerator die notwendigen Werte erzeugt werden.
Auch Probleme des täglichen Lebens können mit solchen Methoden untersucht und daraus eventuell Verbesserungsmaßnahmen abgeleitet werden.

3 In Vorbereitung auf eine vorgesehene Verbesserung der Situation in Kaufhallen soll mittels Simulation festgestellt werden, welches die optimale Anzahl besetzter Kassen ist. Nach Beendigung der Simulation ist auszugeben:

- die untersuchte Zeitspanne,
- die Gesamtzahl der Kunden,
- die mittlere Wartezeit an jeder Kasse und
- die mittlere Wartezeit je Kunde.

In Untersuchungen war festgestellt worden, daß sich die Bedienungszeiten pro Kunde und Kasse wie folgt verteilen:

Bedienungszeit in s	30	45	60	75	90	105	120
relative Häufigkeit	0,1	0,1	0,2	0,2	0,2	0,1	0,1

Zur Vereinfachung legen wir fest, daß ein Kunde immer zu Beginn einer definierten Zeiteinheit an eine Kasse tritt und diese nur am Ende einer Zeiteinheit verläßt. In der betrachteten Zeiteinheit betritt immer nur ein Kunde den Kassenbereich. Wählen wir 5 s als Zeiteinheit (ZE), so soll gelten, daß durchschnittlich alle 3 ZE ein Kunde den Kassenbereich betritt, also in 1 Minute 4 Kunden. Damit beträgt die Wahrscheinlichkeit, daß in einer bestimmten Zeiteinheit ein Kunde an eine Kasse tritt, 1/4. Dieser Kunde stellt sich an der Kasse an, an der die wenigsten Kunden warten. Stellt er keinen Belegungsunterschied fest, so wählt der Kunde die Kasse aus, die am dichtesten zum Ausgang angeordnet ist.

Die jeweilige Zeiteinheit innerhalb einer Minute – 12 haben wir aufgrund obiger Festlegung – wird über Zufallszahlengenerator bestimmt. Als letzte Vorgabe muß noch festgelegt werden, wie die Zuordnung des Kunden zu den einzelnen Bedienzeiten erfolgen soll. Auch hier wird der Zufallszahlengenerator uns behilflich sein. Wir legen dazu folgende Zuordnungsvorschrift fest:

Bedienungszeit in Sekunden	30	45	...	105	120
Bedienungszeit in ZE	6	9	...	21	24
Ereigniswahrscheinlichkeit	<0.0, 0.1)	<0.1, 0.2)	...	<0.8, 0.9)	<0.9, 1.0)

Immer, wenn ein Kunde den Kassenbereich betritt, muß eine Zufallszahl w erzeugt werden. Eine Umrechnung ist nicht notwendig, da wir w als Wahrscheinlichkeit deuten. Wobei ein w im Bereich zwischen 0.2 und 0.4 bedeutet, daß die Kassiererin 12 ZE zur Kassierung benötigt.

Führt man die Simulation in mehreren Durchläufen mit unterschiedlichen Anzahlen besetzter Kassen durch, so erhält man Auskunft über die optimale Anzahl besetzter Kassen.
Mit den bisherigen Festlegungen kann der in Abbildung 5.2/6 gezeigte Grundalgorithmus erstellt werden.

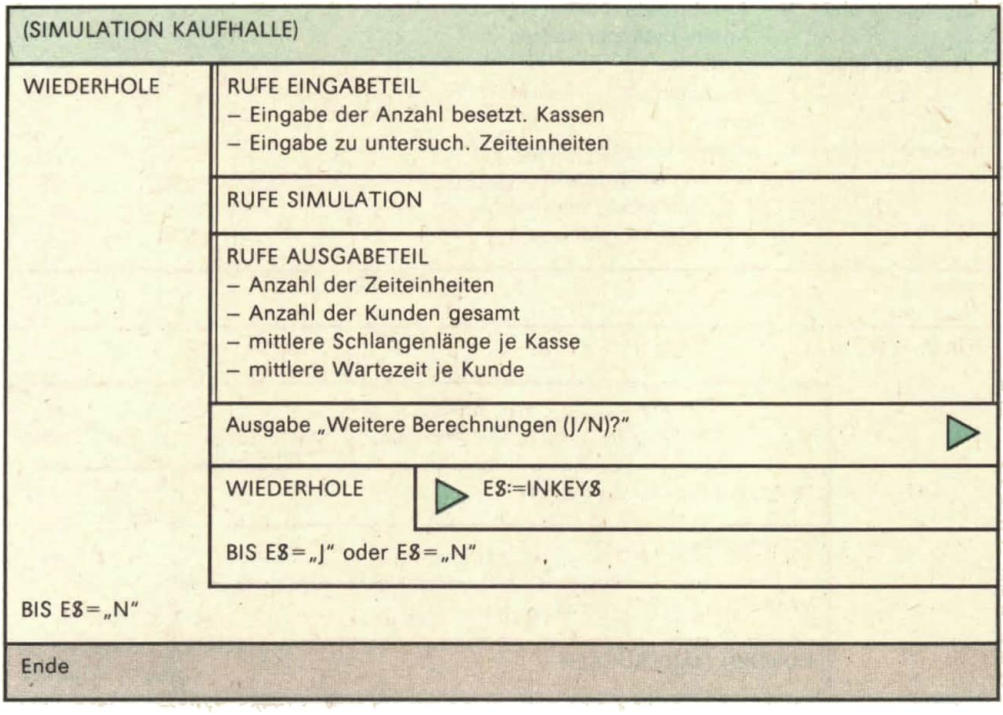


Abb. 5.2/6: Struktogramm Grundalgorithmus „Kaufhalle"

Programmtechnisch neue Aspekte sind nur im Teilalgorithmus „SIMULATION" enthalten. Deshalb ist die schrittweise Verfeinerung nur dafür dargestellt. Eingabeteil, Berechnungen und Ausgabeteil können auf der Grundlage des Struktogramms und obiger Festlegungen in BASIC programmiert werden. Der Unteralgorithmus „SIMULATION" besteht aus folgenden Bausteinen:

- Ermittlung der Zeiteinheit, wenn ein Kunde den Kassenbereich betritt und zählen dieses Kunden,
- Ermittlung der Bedienungszeit für diesen Kunden,
- Einordnen in eine Warteschlange,
- Zählen des neuen Kunden in Kassenwarteschlange,
- Bedienungszeit des aktuellen Kunden verringern,
- wenn Kasse frei wird, Kunden nachrücken lassen.

Die Bausteine stehen sowohl in Abhängigkeit von der ablaufenden Zeit als auch der Anzahl der besetzten Kassen. Die nun noch enthaltenen Unteralgorithmen werden hier nicht aufgegliedert, sondern gleich als BASIC-Programmteile angegeben, wobei der Vorspann und das Hauptprogramm den Anweisungsnummernbereich bis 990 belegen. Der

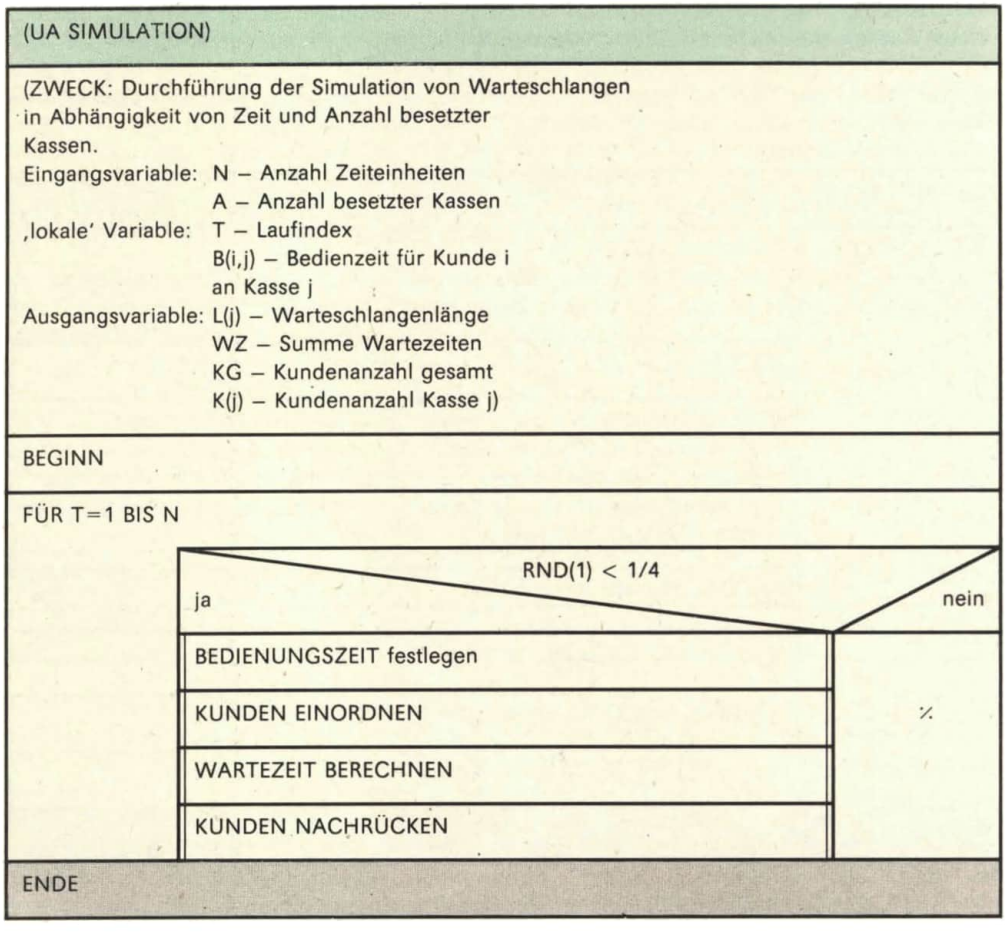


Abb. 5.2/7: Struktogramm „UA SIMULATION"

Eingabeteil beginnt mit Zeile 1000, SIMULATION mit 2000 und der Ausgabeteil mit 3000. Den in SIMULATION enthaltenen Unteralgorithmen werden folgende Nummernbereiche zugeordnet:

- BEDIENUNGSZEIT ERMITTELN 2500 bis 2690
- KUNDEN EINORDNEN 2700 bis 2790
- WARTEZEIT BERECHNEN 2800 bis 2890
- KUNDEN NACHRÜCKEN 2900 bis 2990.

```
2000 REM ===== UA SIMULATION ====================
2010 REM Realisierung der Simulation der Warteschlangen
2020 REM Eingangsvariablen:
2030 REM   N      - Anzahl Zeiteinheiten
2040 REM   A      - Anzahl besetzter Kassen
2050 REM Ausgangsvariablen:
2060 REM   L(j)   - Warteschlangenlaenge je Kasse
2070 REM   WZ     - Summe Wartezeiten
```

```
2080 REM   KG      - Kundenanzahl gesamt
2090 REM   K(j)    - Kundenanzahl an Kasse j
2100 REM 'lokale'Variablen:
2110 REM   B(i,j) - Bedienungszeit je Kunde i, Kasse j
2120 REM   T,S     - Laufindex
2130 REM ------------------------------------------
2190 RANDOMIZE
2200 FOR T=1 TO N
2210    IF RND(1) >= 1/4 THEN 2270
2220    KG = KG + 1
2230    GOSUB 2500 : REM → BEDIENUNGSZEIT
2240    GOSUB 2700 : REM → KUNDEN ZUORDNEN
2250    GOSUB 2800 : REM → WARTEZEIT
2260    PRINT T;"   ";M;"   ";W1,B
2270 REM Betrachtung einzelner Kassen
2280    FOR S=1 TO A
2290       IF K(S) = 0 THEN 2340
2300       L(S)=L(S)+K(S)
2310       B(1,S) = B(1,S) - 1
2320       IF B(1,S) > 0 THEN 2340
2330       GOSUB 2900 : REM → KUNDEN NACHRUECKEN
2340    NEXT S
2350 NEXT T
2360 RETURN : REM ----- Ende UA SIMULATION ----------
2500 REM ===== UP BEDIENUNGSZEIT ====================
2510 REM Bedienungszeit zufaellig festlegen
2520 REM Ausgangsvariable: BZ - Bedienungszeit
2530 REM 'lokale'Variable: X  - Zufallszahl
2540 X=RND(1)
2550 IF X < .1 THEN BZ=6  : GOTO 2690
2560 IF X < .2 THEN BZ=9  : GOTO 2690
2570 IF X < .4 THEN BZ=12 : GOTO 2690
2580 IF X < .6 THEN BZ=15 : GOTO 2690
2590 IF X < .8 THEN BZ=18 : GOTO 2690
2600 IF X < .9 THEN BZ=21 : GOTO 2690
2610 IF X < 1  THEN BZ=24
2690 RETURN : REM -------- Ende UP BEDIENUNGSZEIT ---
2700 REM ===== UP KUNDEN EINORDNEN ==================
2710 REM Kuerzeste Schlange suchen
2720 M = 1
2730 FOR S=2 TO A
2740    IF K(M) > K(S) THEN M = S
2750 NEXT S
2760 K(M) = K(M) + 1
2770 B(K(M),M) = BZ
2790 RETURN : REM ----- Ende UP KUNDEN EINORDNEN ----
2800 REM ===== UP WARTEZEIT BERECHNEN ===============
2810 IF K(M) = 1 THEN 2890
2820 W1 = 0
2830 FOR Z=1 TO K(M)-1
```

```
2840     W1 = W1 + B(Z,M)
2850 NEXT Z
2860 WZ = WZ + W1
2890 RETURN : REM ----- Ende UP WARTEZEIT BERECHNEN -
2900 REM ===== UP KUNDEN NACHRUECKEN ================
2910 K(S) = K(S) - 1
2920 IF K(S)=0 THEN 2990
2930 FOR I=1 TO K(S)
2940     B(I,S) = B(I+1,S)
2950 NEXT I
2960 B(K(S)+1,S) = 0
2990 RETURN : REM ----- Ende UP KUNDEN NACHRUECKEN --
```

Wenn neben diesen Unterprogrammen auch die anderen Algorithmenteile in BASIC erstellt wurden, kann der Abschlußtest des gesamten Programms durchgeführt werden.
Mit dem letzten Problem und seiner Lösung wurde die Möglichkeit der Simulation von Zufallsprozessen demonstriert. Der Ausgangspunkt dafür war, daß eine exakte Beschreibung des Problems nicht möglich war.
Es gibt aber viele andere Prozesse und Probleme, die sich exakt beschreiben lassen und deren Verhalten genau berechenbar ist. So kann man beispielsweise die Bewegung eines Körpers der Masse *m*, auf den die Kraft *F* ausgeübt wird, mit dem Newtonschen Gesetz

$F = m * a$

ganz exakt berechnen. Dieses Newtonsche Gesetz ist ein mathematisches Modell für diesen Vorgang, der dadurch eindeutig definiert bzw. determiniert ist. Deshalb spricht man in diesem Fall auch von deterministischen Modellen. Solche Modelle gibt es nicht nur in der Physik sondern auch in der Chemie und Biologie sowie in anderen Wissenschaftsgebieten.

```
Simulierte Zeitspanne: 345 ZE

Gesamtzahl der Kunden: 92

Mittlere Schlangenlaenge:

    Kasse 1 :                .9 Kunden
    Kasse 2 :                .8 Kunden
    Kasse 3 :                .6 Kunden
    Kasse 4 :                .5 Kunden
    Kasse 5 :                .4 Kunden
    Kasse 6 :                .3 Kunden

Mittlere Wartezeit:  .1 ZE je Kunde!


          Weitere Berechnunge (J/N)? >
```

Abb. 5.2/8: Hardcopy des Programms „Warteschlange"

Die Simulation auf der Grundlage solcher deterministischer Modelle kann man mit relativ geringem Aufwand – Erstellung des Modells und dessen Programmierung – unter Verwendung von Computern realisieren. Der wesentliche Vorteil besteht dabei in der Zeit- und Kostenersparnis. Solche Simulationen dürfen aber niemals die alleinige Grundlage für Entwicklungen in Wissenschaft und Technik sein. Auch eine sehr umfassend angelegte Simulation kann nicht völlig die Beobachtung des Realexperiments ersetzen. Ein Modell spiegelt immer nur einen Teil der Realität wider.

Ein Beispiel für eine deterministische Simulation ist die Wachstumssimulation, die nicht nur für die Biologen interessant ist. Probleme des Wachstums treten beispielsweise auf bei Betrachtungen von:

- Länge der Menschen,
- Wissen des Menschen,
- Sparguthaben,
- Bevölkerungsanzahl eines Landes,
- Wirtschaftskraft eines Landes u. v. a.

Die deterministische Simulation soll in Form einer Räuber-Beute-Simulation kurz in ihren wesentlichen Zügen gezeigt werden.
Ausgangspunkt ist ein als abgeschlossen betrachtetes Biosystem – ein Gebiet der nördlichen Taiga –, wo Rentiere (R) und Wölfe (W) als einzige Tiere existieren. Durch den Ausschluß anderer Störfaktoren könnte man exponentielles Wachstum ansetzen, wenn die Rentiere nicht die Beutetiere der Wölfe wären. Die gegenseitigen Abhängigkeiten werden durch folgende von der Zeit abhängige Beziehungen beschrieben.

$R_{i+1} = R_i + GR * R_i - SR * R_i * W_i$
$W_{i+1} = W_i - SW * W_i + ZW * R_i * W_i$

Dabei bedeuten:

GR → Geburtenrate der Rentiere
SR → Sterberate der Rentiere
SW → Sterberate der Wölfe
ZW → Zuwachsrate der Wölfe.

```
        RAEUBER-BEUTE-SIMULATION
        ========================
Zeiteinh.      Rentiere      Woelfe
-----------------------------------
    0            275           21
    5            118           22
   10             87           13
   15            138            7
   20            293            7
   25            496           17
   30            110           37
   35             20           17
   40             27            6
   45             71            3
   50            223            2
   55            708            4
   60            707           63
   65              0           40
   70              0           13
   75              0            4
   80              0            1
   85              0            0
```

Abb. 5.2/9: Hardcopy „RÄUBER-BEUTE-SIMULATION"

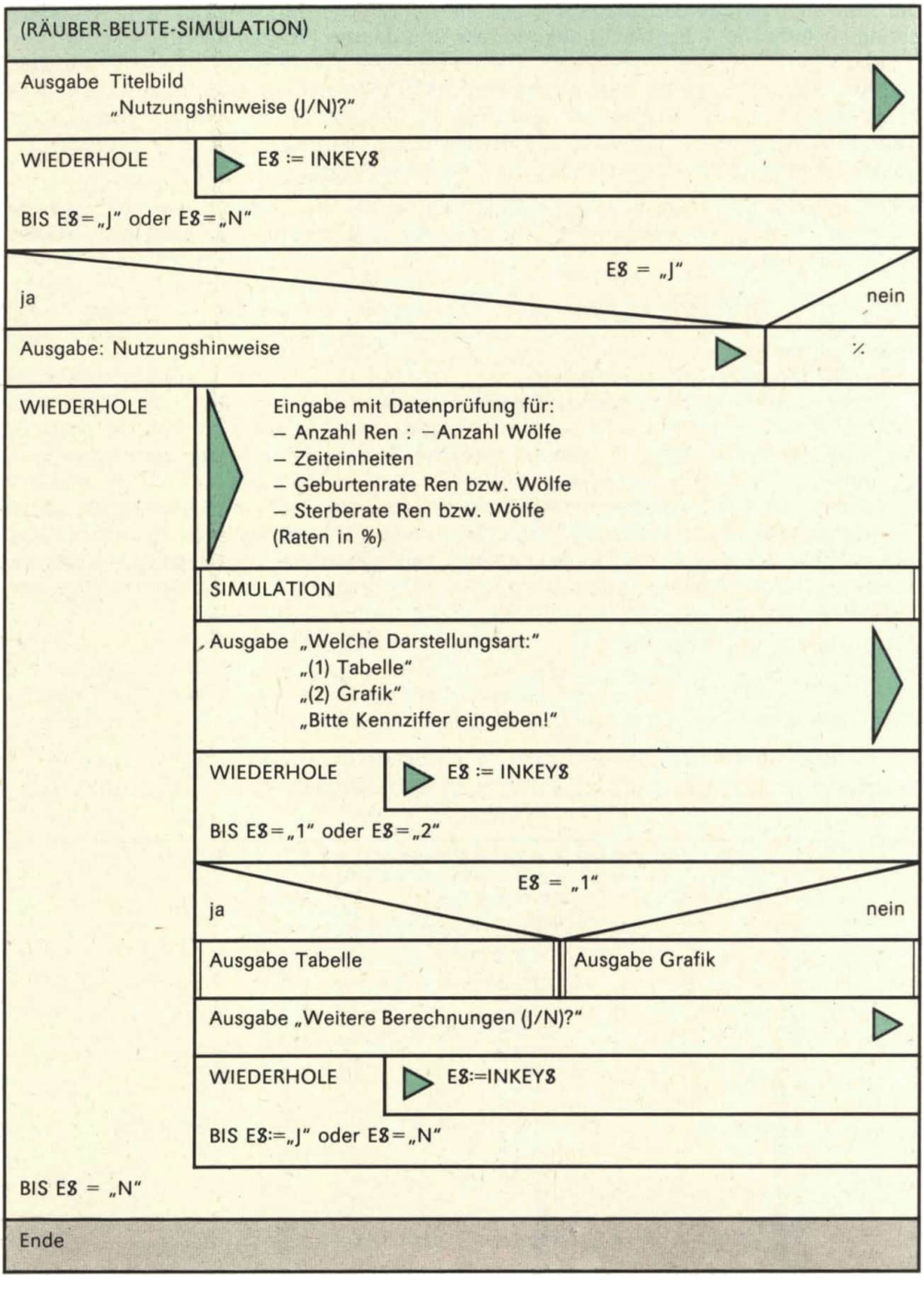


Abb. 5.2/10: Struktogramm „RÄUBER-BEUTE-SIMULATION"

Es ist ein Programm zu erstellen, mit dessen Hilfe diese Räuber-Beute-Simulation durchgeführt werden kann. Der Nutzer des Programms soll in einem Eingabedialog die Startanzahl für Ren und Wölfe sowie die Parameter GR, SR, SW und ZW eingeben können, um so verschiedene Varianten zu berechnen. Die Ergebnisse sollen wahlweise als Tabelle bzw. Grafik ausgegeben werden, wobei die Entwicklung der Individuen über der Zeit dargestellt werden soll. Die Umsetzung des Algorithmus (Abb. 5.2/10) in ein BASIC-Programm bereitet keine Probleme, deshalb wird hier auf eine Darstellung des Programmtextes verzichtet. Eine Hardcopy eines Programmlaufs (Abb. 5.2/9) soll die Funktionsfähigkeit des Algorithmus demonstrieren.

Zusammenfassung

Anwendungsmöglichkeiten des Computers bei der Lösung numerischer Probleme sind (Auswahl):

- statistische Berechnungen,
- Näherungsverfahren,
- Wahrscheinlichkeitsrechnung,
- Kombinatorik,
- Simulationen.

Bei Simulationen ist zu beachten, daß stets das Realexperiment bzw. die Beobachtung in der Natur zur weiteren Konkretisierung der Simulationsergebnisse mit herangezogen werden muß.

● Aufgaben

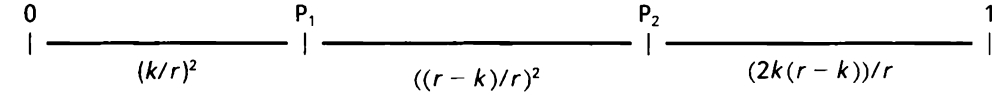
1. Erstellen Sie das vollständige Programm zur Untersuchung von Zufallszahlengeneratoren entsprechend Abbildung 5.2/1! Experimentieren Sie mit diesem Programm, in dem Sie die Zufallszahlengeneratoren durch andere ersetzen (z. B. aus [11] Hefte 5 und 6)!
2. Erstellen Sie ein vollständiges Programm zur Flächenberechnung mittels Monte-Carlo-Simulation! Realisieren Sie den allgemeinen Fall der Nutzereingabe der zu untersuchenden Funktion und des Intervalls!
3. Erarbeiten Sie ein Programm zur grafischen Darstellung der Alterspyramide der Bevölkerung der DDR in einem Streifendiagramm auf der Grundlage des aktuellen Statistischen Jahrbuchs der DDR! Bieten Sie dabei dem Nutzer die Wahl zwischen einer Gesamtpyramide und Teilpyramiden für Frauen und Männer!
4. Erstellen Sie für das Würfelspiel „Die Böse 1" ein Programm! Ein Spieler ist der Nutzer und ein zweiter Spieler der Computer. Die Spielregeln lauten:
   - 2 Spieler würfeln abwechselnd jeweils solange, bis eine 1 kommt oder mindestens 11 Punkte erreicht wurden. Dann entscheidet der Spieler, ob er weiter würfelt oder nicht.
   - Wenn eine 1 geworfen wird, erhält der Spieler für diese Runde 0 Punkte. Die Punkte der einzelnen Runden werden addiert.
   - Gewonnen hat der Spieler, der zuerst 100 Punkte erreicht hat. Es gibt also kein Unentschieden!

Es ist sinnvoll, zur Erstellung des Algorithmus selbst ein paar Runden mit dem Würfel nach den Spielregeln zu spielen und den Verlauf zu protokollieren. Erstellen Sie zuerst einen Minimalalgorithmus und bauen sie diesen dann entsprechend den bekannten Forderungen nach Nutzerfreundlichkeit schrittweise aus!

5. Erstellen Sie Algorithmus und Programm für das Spiel „Drei gewinnt"! 2 Spieler – der Programmnutzer und der Computer – besitzen zu Beginn des Spiels gleichviel Kugeln, der eine weiße und der andere schwarze. Nach Festlegung des Spielers mit dem ersten Zug dürfen die Spieler abwechselnd jeweils eine Kugel in einen beliebigen der m Schächte des Spielgerätes werfen. Die Maximalanzahl pro Schacht ist n. Die Werte von m und n werden zu Beginn des Spiels festgelegt. Ziel einer Spielrunde ist es, nach dem Spielen aller Kugeln möglichst viele Reihen aus mindestens drei Kugeln der eigenen Farbe senkrecht übereinander oder waagerecht nebeneinander liegend zu haben. Der Spieler mit der höchsten Reihenanzahl in jeder Runde erhält zwei Punkte, bei Gleichstand erfolgt Punkteteilung. Es werden mehrere Runden (zu Beginn festgelegt) gespielt.
   Stellen Sie das Spielfeld im Programm grafisch dar!

6. Der Computer soll eine fünfstellige ganze Zahl mittels Zufallszahlengenerator bilden und sich merken. Der Nutzer soll diese Zahl mit möglichst wenigen Versuchen erraten. Dazu gibt er eine gedachte Zahl ein, und der Rechner untersucht und bewertet ziffernweise in der Art, daß er anzeigt
   # für jede Ziffer auf der richtigen Position,
   $ für jede richtige Ziffer, die aber in der Computerzahl an anderer Position steht.
   Unter Verwendung dieser Bewertung errät der Spieler die nächste Zahl.
   Erstellen Sie Algorithmus und vollständiges Programm für dieses Spiel!

7. Mittels eines Simulationsprogramms soll das Diffusionsverhalten zweier inkompressibler Flüssigkeiten untersucht werden. Die Flüssigkeiten befinden sich in zwei durch eine durchlässige Membran verbundene Gefäße. Im rechten sollen sich rote (r), im linken blaue (b) Moleküle befinden. Nach Bernoulli und Laplace verwenden wir das vereinfachte Modell, daß ein Teilchenaustausch stets paarweise erfolgt. Für qualitative Betrachtungen genügt es damit, nach jedem Austauschschritt die Anzahl k der roten Moleküle im linken Gefäß zu wissen. Offenbar gibt es drei Fälle:
   a) Anzahl der roten Moleküle nimmt um 1 ab, Wahrscheinlichkeit dafür liegt bei $(k/r)^2$.
   b) Anzahl der roten Moleküle nimmt um 1 zu, Wahrscheinlichkeit $((r-k)/r)^2$.
   c) Anzahl der roten Moleküle bleibt konstant, Wahrscheinlichkeit $2k(r-k)/r^2$.
   Ausgangspunkt soll $r=10000$ sein. Nach jeweils 100 Austauschschritten ist auszudrucken, wieviel rote Moleküle links anzutreffen sind. Die Fälle a bis c werden dabei durch Ziehung einer Zufallszahl simuliert, wobei gelten soll



8. Auf vielen Jahrmärkten sind Schießbuden zu sehen. Ein Besucher bezahlt an einer solchen Bude 1 M für 5 Schuß. Wenn er mit diesen 5 Schuß mindestens 3 Tonröhrchen vollständig zerstört, erhält er einen Preis von durchschnittlich 5 M. 5 dieser Tonröhrchen kosten den Budenbesitzer 0,20 M. Die Wahrscheinlichkeit, ein Tonröhrchen mit einem Schuß zu zerstören, beträgt $^1/_2$.
   Simulieren Sie mit dem Programm das Schießen von 1000 Personen und berechnen Sie dabei, was der Budenbesitzer durchschnittlich an einem Schützen verdient!

9. Ein Ball wird unter einem Winkel von 45° zur Vertikalen mit einer Anfangsgeschwindigkeit von 9 m/s geworfen. Wenn der Ball wieder auf den Boden fällt, hüpft er weiter, wobei sich bei jedem Aufprall die Horizontalkomponente der Geschwindigkeit nicht ändert. Die Vertikalkomponente der Geschwindigkeit kehrt sich um und ist um 10% kleiner als die vorhergehende.
   Simulieren Sie die ersten 10 Sprünge des Balls und erstellen Sie eine Grafik der Bahnkurve! Verwenden Sie Zeitintervalle von jeweils $^1/_{10}$ bzw. $^1/_{100}$ s! Erstellen Sie den Algorithmus und ein vollständiges Programm, das in einem Anfangsdialog auch die Eingabe für Winkel und Geschwindigkeit ermöglicht!

10. Auf den Ecken eines Quadrates mit der Seitenlänge 100 cm befindet sich je eine Schnecke. Jede Schnecke beginnt mit einer Geschwindigkeit von 1 cm/s auf die rechts von ihr befindliche Schnecke zuzukriechen. Strebt sie immer direkt auf die jeweils rechts befindliche Schnecke zu, so ergeben sich bestimmte Bahnen, bis sich die Schnecken nahe dem Mittelpunkt des Quadrates treffen.
    Erstellen Sie ein Programm zur Simulation der Bahn der Schnecken und bestimmen Sie die Zeit, bis sie sich getroffen haben! Die Bahnen sollen auf dem Bildschirm gezeichnet werden. Experimentieren Sie mit diesem Simulationsprogramm durch Verwendung unterschiedlicher Zeitintervalle, beispielsweise 1, $^1/_2$ und $^1/_5$ Sekunde!

11. Ein Behälter enthält 750 l Wasser, in dem 50 kg Salz gelöst sind. In diesen Behälter fließt Salzwasser (Salzgehalt 0,25 g/l) mit einer Geschwindigkeit von 24 l/min. Ein im Behälter befindliches Rührwerk sorgt für eine gute Durchmischung. Aus einem Ventil des Behälters fließen pro Minute 16 Liter Salzwasser ab.
    Simulieren Sie diesen Vorgang und berechnen Sie die Salzmenge im Behälter nach 30 Minuten! Eine grafische Darstellung ist ebenfalls zu realisieren.

12. Erstellen Sie das vollständige Programm zur Räuber-Beute-Simulation entsprechend Abbildung 5.2/10! Experimentieren Sie mit verschiedenen Wertesätzen für die Parameter GR, SR, ZW und SW! Bewerten Sie die Ergebnisse dieser Simulationen!

8 Vorprojekt: Meßwerterfassung und Meßwertauswertung

Zwischen Maß und dem Gemessenen
wird bei der größten Gleichheit
noch eine Differenz übrig bleiben.
(Nicolaus von Cusa)

Mit diesem Vorprojekt soll das Themengebiet **„Projektarbeit"** anhand eines konkreten Beispiels in seinen Anforderungen und in seiner Realisierung demonstriert werden.

Im Physikunterricht wurde der freie Fall behandelt. In Verbindung mit den Möglichkeiten des Computers soll dieser Vorgang noch einmal behandelt werden. Auf die physikalischen Grundlagen wird nur soweit eingegangen, wie es die Problemanalyse erfordert.

Aufgabenstellung: Es ist eine Versuchsanordnung zu erstellen, die die Untersuchung des Falls eines Körpers unter Verwendung des Kleincomters KC 85/2 bzw. KC 85/3 zur Meßwerterfassung und zur Meßwertauswertung ermöglicht.

6.1. Problemanalyse und schrittweise Verfeinerung

6.1.1. Fachlicher Hintergrund

Die Geschwindigkeit eines fallenden Körpers wird durch zwei auf ihn wirkende Kräfte bestimmt. Das ist einmal die Gravitationskraft F_G und die entgegengesetzt wirkende Reibungskraft F_R. Die Differenz beider Kräfte bestimmt die Fallgeschwindigkeit des Körpers. Bei größerer Gravitationskraft wird der Fall beschleunigt. Bei Gleichgewicht der Kräfte ist die Fallgeschwindigkeit konstant und im dritten Fall – größere Reibungskräfte – wird die Fallgeschwindigkeit verringert.
Die Erfahrungen aus dem Alltag lassen uns die Einflußfaktoren bezüglich des Luftwiderstandes schnell erkennen. Diese sind

- die Angriffsfläche q,
- die Geschwindigkeit v,
- die aerodynamische Form (Widerstandsbeiwert) c_w und
- die Dichte des Mediums ϱ.

Ein Blick ins Physikbuch zeigt uns die exakte Gleichung für die *Reibungskraft:*
$F_R = 0{,}5 * \varrho * q * c_w * v^2$.
Die *Gravitationskraft* ergibt sich aus der Masse m des fallenden Körpers und der in Erdnähe als konstant anzunehmenden Fallbeschleunigung g. Damit gilt
$F_g = m * g$.
Durch Umformung dieser Gleichung kann man als Versuchsergebnis die Berechnung der Fallbeschleunigung durchführen.

6.1.2. Experimentgestaltung

Unter Verwendung der genannten Kleincomputer ist eine Versuchsanordnung aufzubauen, die die *Zeitmessung für verschiedene Fallhöhen* und bei Verwendung verschieden geformter Körper aus verschiedenen Materialien realisieren kann.

Die Grundausstattung des Kleincomputers muß dazu ergänzt werden, da folgende Aufgaben zu lösen sind:

- Steuerung des Experimentablaufs,
- Festlegung der Fallhöhe,
- Zeitmessung.

Die erste Aufgabe wird unter Verwendung eines Elektromagneten, der vom Computer aus angesteuert wird und den Fall des Körpers auslöst, realisiert. Für die zweite Aufgabe werden zwei Lichtschranken, die in festlegbarem Abstand montiert werden, an den Computer angeschlossen. Die Zeit zwischen dem Unterbrechen des ersten und des zweiten Lichtstrahls wird vom Computer erfaßt und im weiteren ausgewertet.
Um die genannten Aufgaben realisieren zu können, muß der Modul M001 – Digital INOUT – in einen der Modulschächte des Kleincomputers gesteckt und eine Schaltung mit den Lichtschranken gebaut und angeschlossen werden. Unter Verwendung des Bausatzes „Lichtschranke" ist diese Schaltung leicht zu realisieren.

6.1.3. Programmtechnische Problemanalyse

Unter Beachtung der erstellten Versuchsanordnung ergeben sich folgende *vom Programm zu lösende Aufgaben:*

- Kontrolle, ob Fallkörper vorhanden ist,
- Steuerung der Auslösung des Magneten,
- Erfassung des Eingangsimpulses,
- Erfassung des Ausgangsimpulses,
- Zeitmessung zwischen beiden Impulsen,
- Auswertung des Versuchsergebnisses unter Verwendung weiterer einzugebender Werte (Masse, Fläche und Widerstandsbeiwert des Körpers, Länge der Meßstrecke)

Das Programm sollte die Möglichkeit offen lassen, über weitere Programmbausteine Simulationen zum physikalischen Problemkreis ‚Fall – Reibung – Beschleunigung' – mit entsprechenden grafischen Bausteinen – durchführen zu können.
Damit sind als Ergebnis der ersten Verfeinerungsstufe folgende Bausteine erkenntlich:
- Rahmenprogramm mit Auswahlmenü,
- Versuchsbaustein mit:
    - Steuerung des Elektromagneten,
    - Steuerung der Lichtschranken,
    - Zeiterfassung,
    - Auswertung

[– Simulationsbaustein].

6.2. Modulprogrammierung

Die spezielle Art dieses Projektes verlangt in diesem Kapitel nicht nur die programmtechnische Erarbeitung sondern auch die Erarbeitung der notwendigen Hardwareergänzungen, soweit diese nicht als fertige Baugruppe erhältlich ist. Da die Bereitstellung der Hardware eine Voraussetzung für die Programmerprobung ist, werden diese Arbeiten zuerst im unbedingt notwendigen Maß dargelegt.

6.2.1. Hardwareergänzungen

Der prinzipielle Aufbau des Fallversuchs ist aus Abbildung 6.2/1 ersichtlich. An einem Elektromagneten, der über den PIO A0 (PIO – programmierbarer Ein- und Ausgang) angesteuert wird, kann der Fallkörper befestigt werden. Das Signal 0 am Ausgang A0 bedeutet keinen Stromfluß in der Spule des Elektromagneten, so daß der Fallkörper nicht mehr gehalten wird. Mittels Lichtschranke LS0 wird kontrolliert, ob ein Fallkörper vorhanden ist. Wenn nicht, dann wird ein entsprechendes Signal über den Modul M 001 an das Programm gesendet.

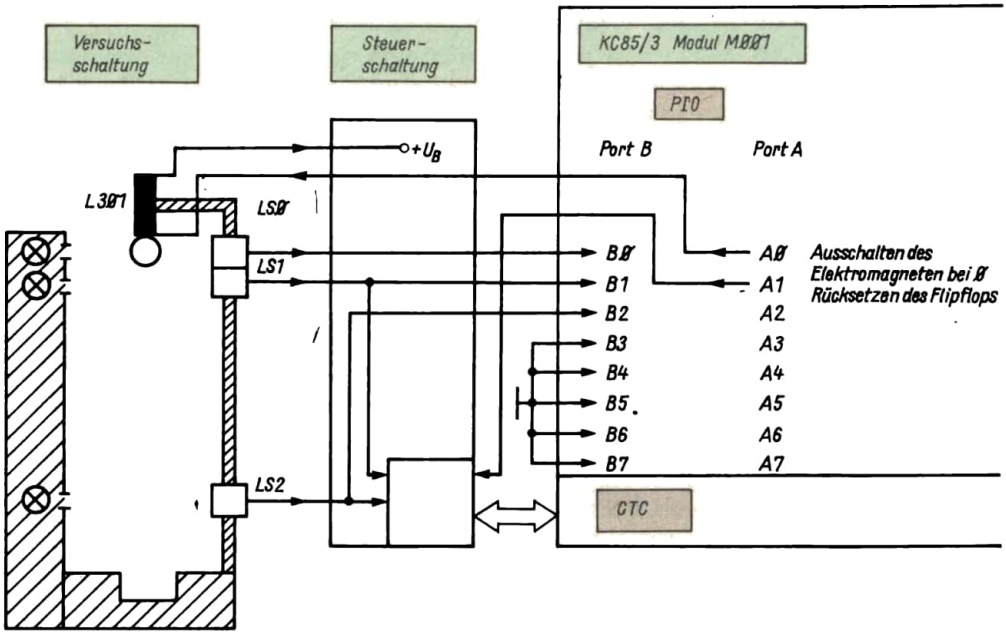


Abb. 6.2/1: Prinzip des Fallversuches

Bei Durchgang des Fallkörpers durch die Lichtschranke LS1 wird der Zeitzählbaustein CTC im Modul M 001 eingeschaltet und bei Durchgang durch die Lichtschranke LS2 wieder ausgeschaltet. Die Lichtschranken sind dazu mit den Anschlüssen B0, B1 und B2 des PIO-Port B verbunden, der auf Eingabe programmiert werden muß. Der Port A der PIO ist auf Ausgabe programmiert. Der Anschluß A1 dient dem Rücksetzen der Steuerschaltung der Lichtschranken. Die Anschlüsse A2 ... A7 können für weitere Steuerungen, z. B. Ein- und Ausschalten der Lampen, genutzt werden. Gleiches gilt für die Anschlüsse B3...B7.

Nicht genutzte Eingänge müssen aber über einen Widerstand an Masse gelegt werden, da sonst eine Verfälschung der Meßergebnisse eintreten kann.

Die Schaltungen für Lichtschrankenbaustein (ist für jede Lichtschranke zu bauen) und Steuerschaltung sollen hier nicht weiter erläutert werden. Die Schaltpläne sind in den Abbildungen 6.2/2 und 6.2/3 dargestellt.

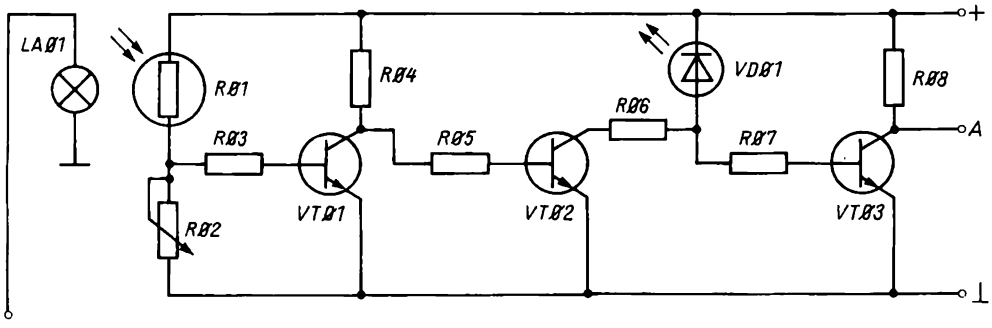


Abb. 6.2/2: Schaltbild des Lichtschrankenbausteins

Die Übersicht zeigt die Bauelemente.

- Bauelemente aus dem Elektronikbausatz „Lichtschranke" des VEB Halbleiterwerk Frankfurt/Oder:

R01, R11, R21	Fotowiderstand
R02, R12, R22	Widerstand 1 kΩ
R03, R13, R23	Widerstand 1 kΩ
R04, R14, R24	Widerstand 3,3 kΩ
R05, R15, R25	Widerstand 1 kΩ
VT01, VT11, VT21	Miniplasttransistor SC236D
VT02, VT12, VT22	Transistor 600 mW SF126D
VD301	SY350/05 (in Steuerschalt.)
L301	Relais (ohne Schaltmechanismus in Steuerschaltung)

- weitere Bauelemente:

R06, R16, R26	Widerstand 82 Ω
R07, R17, R27	Widerstand 750 Ω
R08, R18, R28	Widerstand 270 Ω
VT03, VT13, VT23	Transistor SF126D
VD01, VD02, VD03	VQA

- Steuerelektronik:

R301 ... R303	Widerstand 2 kΩ
R304 ... R306	Widerstand 1 kΩ
R307 ... R311	Widerstand 270 Ω
D301 ... D306	1/6 D204 (P204) oder 1/4 D200
D307 ... D308	1/4 D200 (P200)
D309	1/2 D274 (P274)
D310	1/4 D200 (P200)
VT301	SF126D

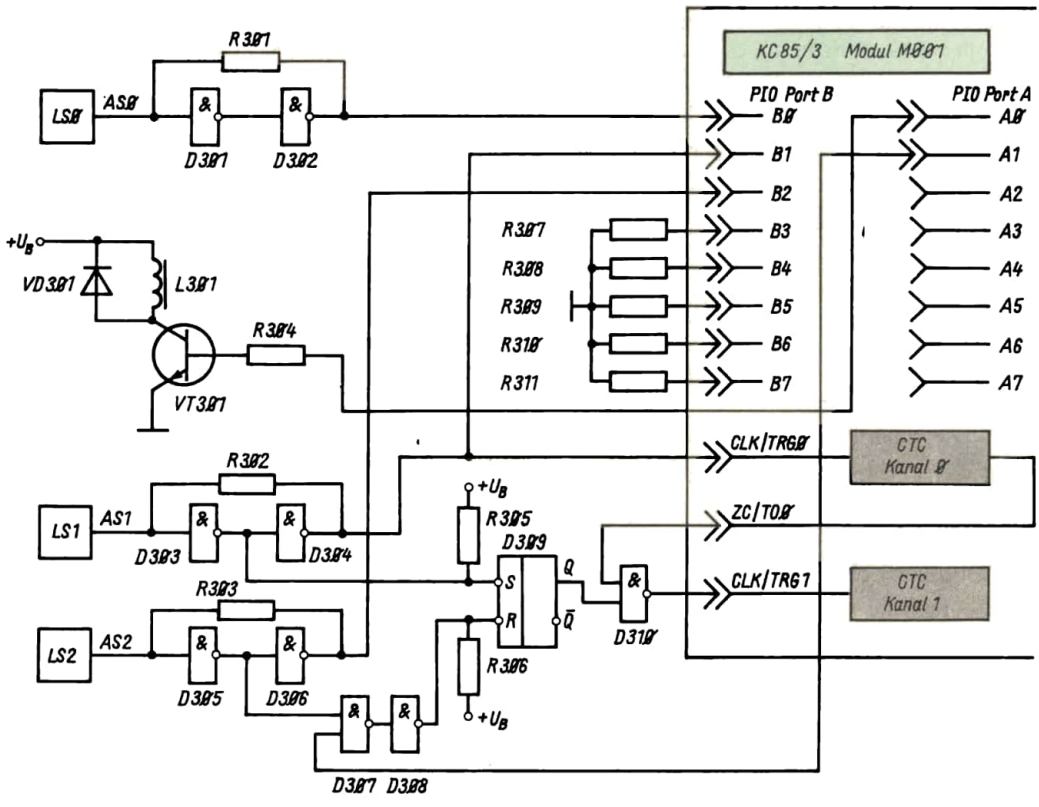


Abb. 6.2/3: Schaltbild der Steuerschaltung

6.2.2. Modulprogrammierung

Ausgehend von den einleitenden Bemerkungen sind in einer ersten Ausbaustufe im wesentlichen 2 Module vollständig zu programmieren und für weitere Module die Integration vorzusehen. Dabei handelt es sich um das Rahmenprogramm mit dem Auswahlmenü, wo auch die Integration noch nicht erstellter Module zu beachten ist, und der Versuchsbaustein. Auch der angedeutete Simulationsbaustein sollte einer weiteren Ausbaustufe vorbehalten bleiben.

Die *Hauptteile des Rahmenprogramms* sind:

- Titelbild,
- Hinweise zur Nutzung des Programms (wahlweise),
- Auswahlmenü,
- Aufruf des gewählten Unteralgorithmus,
- Endebehandlung.

Diese Aufgaben wurden in verschiedenen Beispielen schon in ähnlicher Weise vorgestellt, so daß hier nur das Struktogramm gezeigt wird.

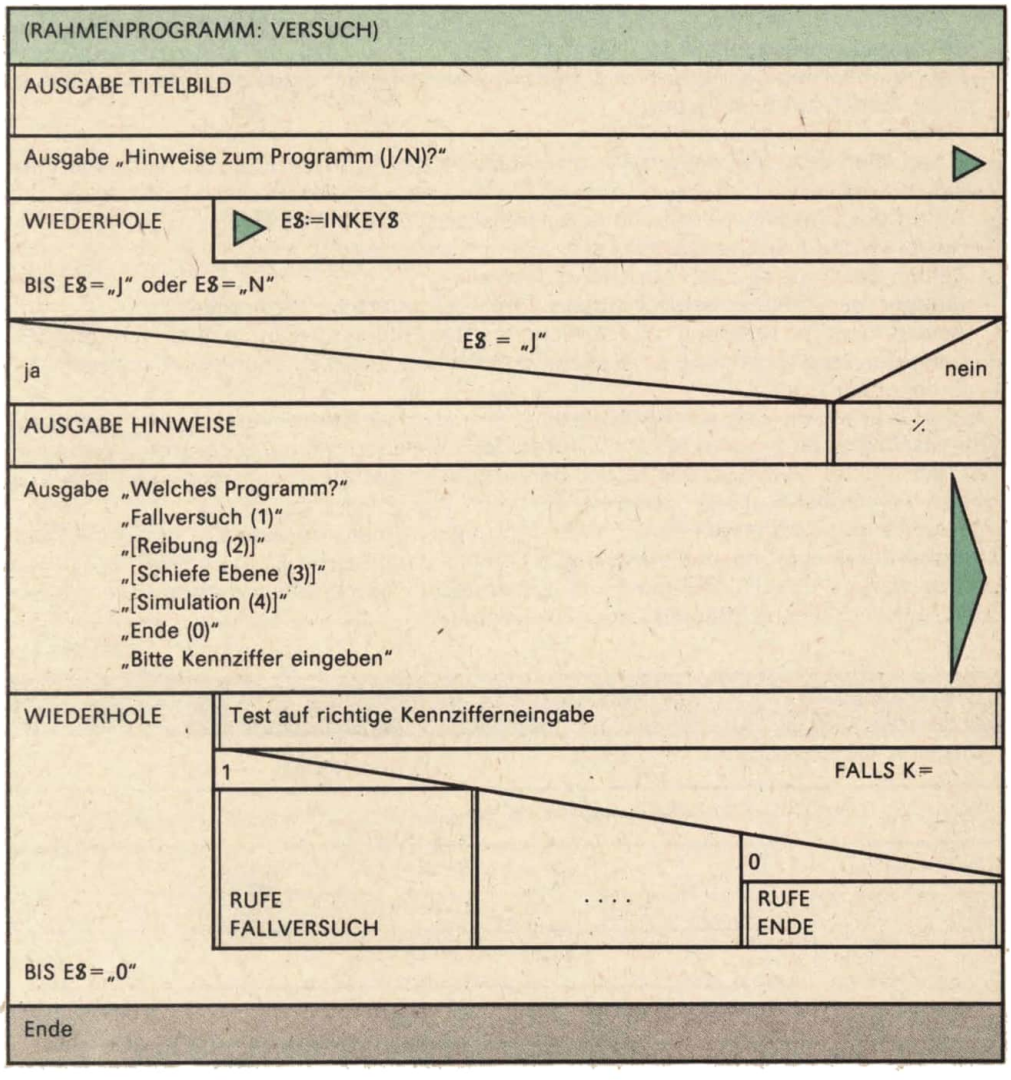


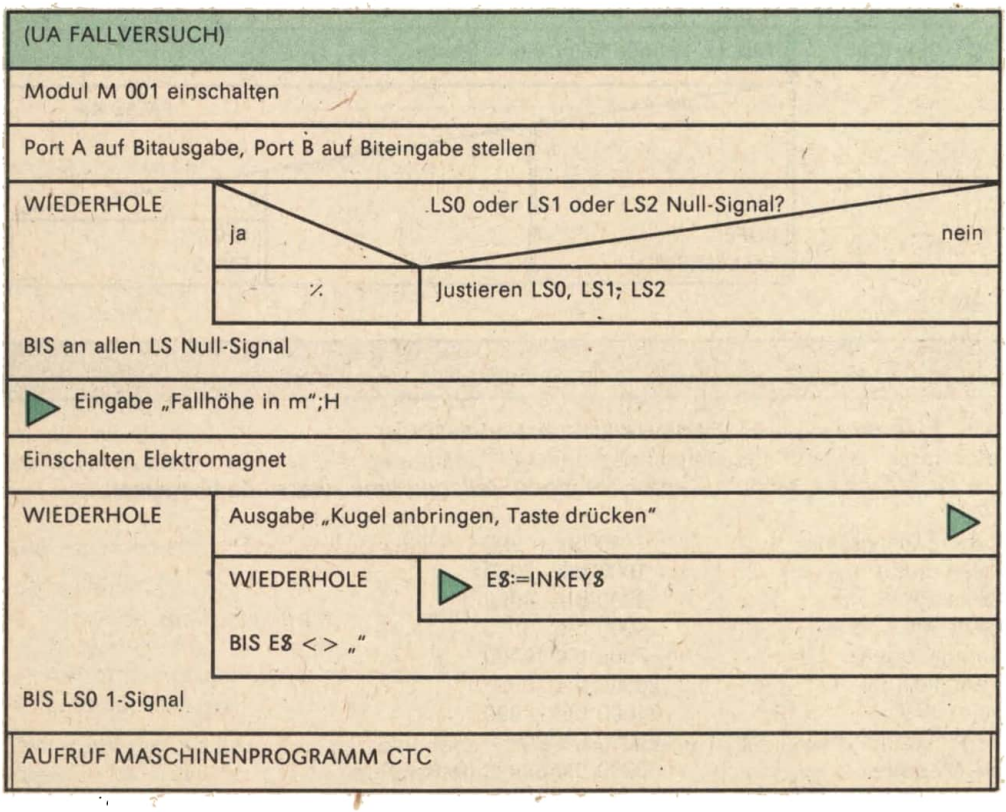
Abb. 6.2/4: Struktogramm „RAHMENPROGRAMM VERSUCH"

Für die einzelnen Module werden folgende *Zeilennummernbereiche* festgelegt:

Rahmenprogramm	10 bis 990
Fallversuch	1000 bis 2990
Reibung	3000 bis 4990
Geneigte Ebene	5000 bis 6990
Simulationen	7000 bis 19990
Titelbild	20000 bis 20990
Hinweise	21000 bis 21990
Endebehandlung	29000 bis 29990
DATA-Zeilen	50000 bis 64990 (falls nötig).

Der *Algorithmus* zum Modul FALLVERSUCH hat folgende wesentlichen Aufgaben:
- Aktivieren des Moduls M 001 (Digital In/Out),
- Programmierung der Ports A und B des Moduls auf Anfangszustand (Port A auf Bitausgabe, Port B auf Biteingabe),
- Abfrage der eingestellten Fallhöhe,
- Einschalten des Elektromagneten und Ausgabe der Aufforderung zum Anbringen des Fallkörpers,
- Aufruf des Maschinenprogramms zur Initialisierung des Zeitzählers,
- Aufforderung zum Starten des Falls,
- Zählen der Impulse nach Auslösung des Falls,
- Stoppen der Zählung bei Durchgang durch die untere Lichtschranke,
- Auswertung der Messung mit Umrechnung der Zählimpulse in Zeiteinheiten, Berechnung der Beschleunigung *g*, Ausgabe von Fallhöhe, Fallzeit und Beschleunigung,
- Endebehandlung.

Versuche ergaben, daß ein BASIC-Programm für die erstellte Versuchsanordnung zur Zeitmessung zu langsam ist, deshalb wurde das Unterprogramm des Zählvorgangs in Maschinensprache realisiert. Das Maschinenprogramm wurde so ausgelegt, daß eine vielseitige Verwendung gewährleistet ist.
Im Algorithmus sind wegen der Verwendung eines Unterprogramms in Maschinensprache und der Steuerung der Versuchsanordnung *einige neue Elemente der BASIC-Programmierung* enthalten, die ganz kurz genannt und beschrieben werden sollen, bevor das Programm hier vollständig angegeben wird.



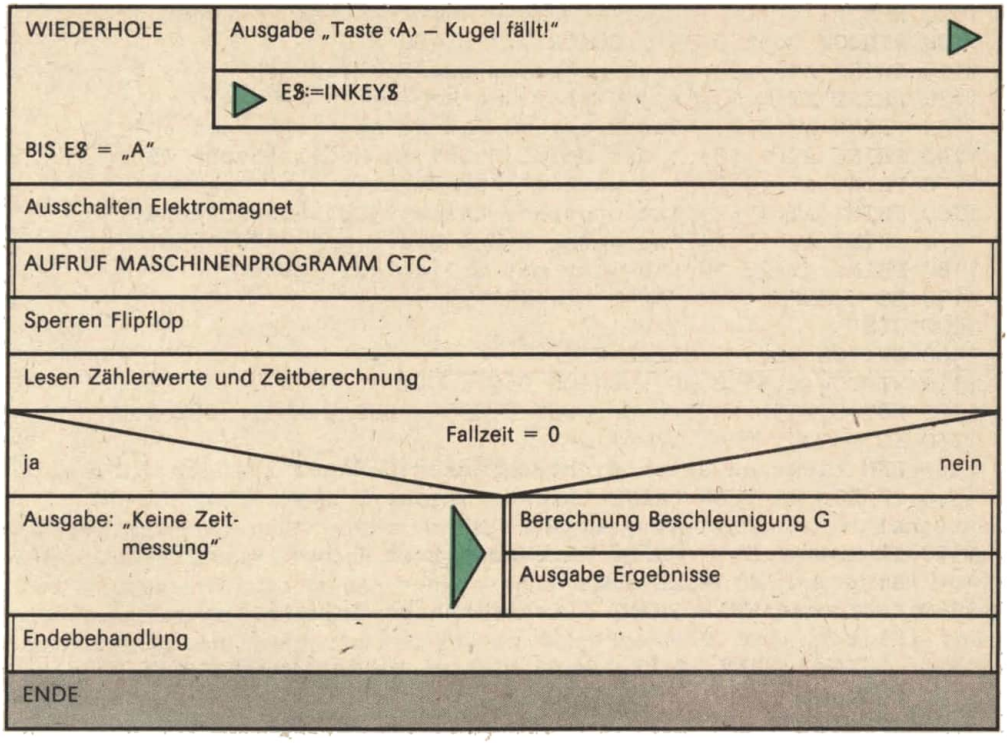


Abb. 6.2/5: Struktogramm „FALLVERSUCH"

Die Verbindung zwischen Kleincomputer und eingeschobenem Modul wird im BASIC-Programm durch die Anweisung SWITCH hergestellt. Diese Anweisung hat das Format

```
SWITCH mm,kk
```

wobei in mm die Adresse des Modulsteckplatzes und in kk der Schaltzustand des Moduls festgelegt wird.

Durch die Anweisungen **INP** und **OUT** wird jeweils ein Byte von der Peripherie gelesen bzw. nach dort ausgegeben.

Der Aufruf eines Unterprogramms in Maschinensprache erfolgt durch die Anweisung **CALL *n**, wobei in *n* die Startadresse des UP in hexadezimaler Form programmiert wird.
Das Maschinensprachprogramm stellt die Zählimpulse in zwei Bytes bereit, die vom BASIC-Programm unter Verwendung der Anweisung **PEEK(n)** gelesen werden können. Weitere Informationen zu den verwendeten Anweisungen sind dem Handbuch zu entnehmen.

```
1000 REM ===== UP FALLVERSUCH ======================

1010 REM Modul zur Steuerung des Fallversuchs
1020 REM und seiner Auswertung
```

```
1030 REM ------------------------------------------
1100 WINDOW 0,31,0,39 : COLOR 7,1 : CLS
1110 PRINT AT(5,2);"Unterprogramm zum Fallversuch"
1120 PRINT AT(7,1);"********************************"
1130 PRINT AT(9,2);"Das Programm ist so angelegt, daß es"
1140 PRINT AT(11,2);" den Modul M 001 im Modulschacht 08"
1150 PRINT AT(13,2);" erwartet! Andernfalls ist Programm"
1160 PRINT AT(15,2);"zu stoppen, Zeile 1300 entsprechend"
1170 PRINT AT(17,2);"zu aendern und Start mit GOTO 1200!"
1180 PRINT AT(28,10);"Weiter mit beliebiger Taste!"
1190 E$=INKEY$ : IF E$="" THEN 1190
1200 CLS
1300 SWITCH 08,1 : CLEAR 256
1310 WINDOW 0,31,0,39 : COLOR 0,7 : CLS
1320 OUT 6,255 : OUT 6,0 : OUT 7,255 : OUT 7,255 : OUT 4,0
1330 K0 = INP(05) : K1 = 0
1335 REM Liegt an allen Lichtschranken 0-Pegel ?
1340 IF K0>=4 THEN PRINT "Lichtschranke 2 justieren!" :
     K1 = 1 : K0 = K0 - 4
1350 IF K0>=2 THEN PRINT "Lichtschranke 1 justieren!" :
     K1 = 1 : K0 = K0 - 2
1360 IF K0>=1 THEN PRINT "Lichtschranke 0 justieren!" :
     K1 =1
1370 IF K1=1 THEN PRINT "Nach Justage beliebige Taste";
     " druecken!" : ELSE 1390
1380 E$=INKEY$ : IF E$="" THEN 1380 : ELSE 1330
1390 CLS : LOCATE 10,2
1400 INPUT "Fallhoehe in m  H=";H
1405 REM  Magnet einschalten
1410 OUT 4,1
1420 PRINT AT(13,2);"Kugel anbringen, dann beliebige Taste"
1430 E$=INKEY$ : IF E$="" THEN 1430
1440 K0 = INP(05)
1445 REM Kugel am Magnet ?
1450 IF K0.<> 1 THEN 1420
1460 REM Initialisierung des CTC-Bausteins
1470 CALL *0030
1480 OUT 4,3
1490 PRINT AT(16,5);"Kugel kann mit der Taste <A> "
1500 PRINT AT(17,9);"ausgeloest werden!"
1510 E$=INKEY$ : IF E$ <> "A" THEN 1510
1520 OUT 4,2
1530 REM CTC-Abfrage
1540 CALL *0060
1550 OUT 4,0
1560 TØ = PEEK(32) : REM Niederwertiges Byte lesen
1570 T1 = PEEK(33) : REM Hoeherwertiges Byte lesen
1580 REM Zeitberechnung aus Zaehlimpulsen
1590 T = (T1/256 + T0) * 1000
1600 IF T = 0 THEN PRINT "Es erfolgte keine Zeitmessung!" :
     GOTO 1690
```

```
1610 REM Berechnungen
1620 G = 2 * H / (T * T)
1630 CLS
1640 PRINT AT(5,6);" Ergebnisse des Versuchs"
1650 PRINT AT(7,2);"Fallhoehe H =";H;"m"
1660 PRINT AT(9,2);"Falldauer T =";T;"sec."
1670 PRINT AT(11,2);"Beschleunigung G=";G;"m/sec^2"
1690 REM Ende des Versuchs
1700 INPUT "Erneuter Versuch erwuenscht (J/N)";E$
1710 IF E$="J" THEN 1200
2970 CLS : PRINT AT(12,5);"Ende des Fallversuchs!"
2980 RETURN
2990 REM ----- Ende des UP FALLVERSUCH ---------------
```

6.3. Einzel- und Gesamttest

Auf die Teststrategie wurde schon mehrfach eingegangen. Noch einmal zur Erinnerung. Jeder einzelne Modul ist möglichst umfassend auf seine Funktionsfähigkeit zu testen. Dazu ist es unter Umständen notwendig, Programmstücke zusätzlich nur für den Test zu schreiben. Beispielsweise sollte man für den Test des Rahmenprogramms alle konzipierten Ausgänge echt programmieren und die aufgerufenen Unterprogramme – außer FALLVERSUCH – durch eine Druckausgabe, als einzigen auszuführenden Befehl des jeweiligen UPs, erstellen.
Der Modul FALLVERSUCH kann durch Austausch der Zeile 2980 RETURN gegen 2980 END als unabhängiges Programm getestet werden. Dabei sollte der Test alle möglichen Programmzweige nachweislich abarbeiten! Es sind also entsprechende Vorüberlegungen zu machen. Am besten ließe sich die vollständige Testung des Programms über ein gedrucktes Testprotokoll nachweisen. Dazu kann ein Drucker über einen Modul V24 im zweiten Modulschacht der KC 85/2 und /3 angeschlossen werden. (In Abbildung 6.3/1 ist ein Hardcopy eines Programmtests dargestellt.) Besteht diese Möglichkeit nicht, so muß dieses Protokoll handschriftlich in allen Einzelheiten notiert werden!
Dem Einzeltest schließt sich der Gesamttest aller Programmkomponenten des Projektes an, der in gleicher Weise zu dokumentieren ist.

```
Lichtschranke 2 justieren
Lichtschranke 1 justieren
Lichtschranke 0 justieren
Fallhoehe in m H= 0.5
Bitte Kugel anbringen und eine Taste
druecken
Die Kugel kann mit der Taste <A> fallen-
gelassen werden
Fallhoehe H=   .5 m
Falldauer T=   .31933 sec.
Beschleunigung G= 9.80665 m/sec^2
Erneuter Versuch erwuenscht ? (J/N) N
Ende des Versuches
OK
>
```

Abb. 6.3/1: Hardcopy eines Testlaufes von „FALLVERSUCH"

6.4. Dokumentation

Die Projektdokumentation ist die Grundlage für die Anwendung der im Projekt erstellten Programme im Routinebetrieb und durch andere Anwender. In ihr müssen alle Aspekte der Projektlösung dokumentiert sein!
Es gehören folgende Teile in eine vollständige Projektdokumentation:

1. vollständige Aufgabenstellung,
2. Darstellung möglicher Lösungsvarianten,
3. realisierte Lösung mit allen Randbedingungen, unter denen eine Anwendung möglich ist,
4. Algorithmenbeschreibung der Lösung,
5. Programmlisting,
6. Programmbeschreibung,
7. Testprotokolle,
8. notwendige Hardware, einschließlich Erweiterungen und selbstgebauter Bausteine mit allen Unterlagen zum Nachbau.

Ein Beispiel für vollständige Projektdokumentation ist in Kapitel 7 enthalten.

Ein Teil der obigen Forderung wurde mit den Darstellungen in diesem Kapitel schon realisiert. Es fehlt eigentlich nur noch die Programmbeschreibung. In dieser werden die Programmzeilen näher erläutert, die für die Problemlösung entscheidend sind oder bei denen Reaktionen des Nutzers erwartet werden, die hier beispielhaft gezeigt werden.

Programmbeschreibung

Zeile	Bemerkungen
1100–1180	Hinweise an den Nutzer über erwartete Belegung der Modulschächte
1190	Eingabeanforderung zur Lösung der Hinweise
1300	Aktivieren des Moduls M 001 in Modulschacht 08, Reservieren und Löschen des Zeichenkettenspeicherbereichs
1320	OUT 6,255 schaltet PORT A des Moduls auf Bit-Ein/Ausgabe und mit OUT 6,0 nur auf Ausgabe. PORT B wird durch OUT 7,255 und OUT 7,255 auf Biteingabe programmiert. OUT 4,0 setzt die Ausgänge AO und A1 auf Nullpegel, also Ausschalten des Elektromagneten und Sperren der Zeitzählung.
1330–1380	Überprüfung der Pegel an den Ausgängen B0...B2. Falls nicht 0-Pegel, erfolgt Ausgabe der Forderung nach Justage der einzelnen Lichtschranken.
1400	Aufforderung zur Eingabe der Fallhöhe, die vorher zu messen war (möglichst genau!)
1410	OUT 4,1 schaltet den Elektromagneten ein.
1420–1430	Aufforderung zum Anbringen des Fallkörpers. Danach ist eine beliebige Taste zu betätigen.
1440–1460	prüfen, ob Fallkörper vorhanden ist, sonst erneute Aufforderung.
1470	Aufruf des Maschinenprogramms zur Initialisierung des Zeitzählerbausteins
1480	OUT 4,3 läßt Magneten eingeschaltet und gibt Zeitzähler frei.

Zeile	Bemerkungen
1490–1510	fordern Nutzer zur Freigabe des Fallkörpers auf, was durch Drücken von ⟨A⟩ erfolgt.
1520	schaltet Magneten ab, Körper fällt.
1540	Aufruf des Maschinenprogramms zur Zeitmessung, die durch ein positives Signal am Ausgang von LS2 beendet wird.
1550	OUT 4,0 sperrt das Flipflop (A1 = 0).
1560–1570	Umspeichern der Zeitimpulse nach T0 und T1
1580	Umrechnung der Zeitimpulse
1610–1620	Berechnung
1630–1680	Ausgabe der Ergebnisse
1700–1710	Anfrage nach Versuchswiederholung und eventuell Rücksprung zum Versuchsbeginn
2970–2990	Programmendebehandlung

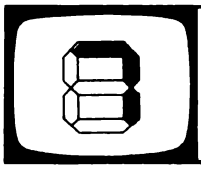
Die hier dargestellte Art der Programmbeschreibung ist eine Kurzform. Bei anderen Programmen muß sie eventuell ausführlicher sein.

6.5. Pflege und Wartung von Programmen

Für die Übergabe eines Programms in den Routinebetrieb eines Rechenzentrums bzw. zur Nachnutzung ist es notwendig, daß der Programmautor eine entsprechende Anzahl von Programmkopien auf weiteren Datenträgern erstellt.
Im Laufe der Nutzung eines Programms wird es sich ergeben, daß aus den unterschiedlichsten Gründen Modifikationen zum Programm gewünscht werden. Häufige Gründe sind der Wunsch

- nach Erweiterung, da die Nutzer weitere Einsatzmöglichkeiten erkannt haben,
- nach Anpassung auf veränderte Gerätetechnik,
- nach Änderung, weil neue gesetzliche Bestimmungen erlassen wurden.

Bei großen Projekten finden sich meist im Laufe der Nutzung noch Fehler bzw. treten Datenkonstellationen auf, die im Rahmen der Datenprüfung nicht gefunden werden und somit zu Programmfehlern führen.
Aus diesen Gründen gilt bei Softwareproduzenten, daß die Arbeiten an größeren Programmsystemen erst zu Ende gehen, wenn die Programme nicht mehr genutzt werden. Der Aufwand für Pflege und Wartung von größeren Programmen wird oft unterschätzt. Die gezielte Anwendung der Methode der schrittweisen Verfeinerung und der strukturierten Programmierung kann den notwendigen Aufwand verringern helfen.



Projektarbeit

Die Genauigkeit einer wissenschaftlichen Arbeit sollte ihre Zuverlässigkeit nicht überschreiten.
(Dimitri Iwanowitsch Mendelejew)

Die Projektarbeit ist auf das Ziel gerichtet, in einer genau festgelegten Zeit ein Problem unter Ausnutzung eines Computers lösen zu lassen. Dabei sind viele der erworbenen Fähigkeiten und Fertigkeiten gefordert. Die *Projektthemenstellung* sollte genügend Spielraum für schöpferische, originelle Lösungen geben. Am Ende der Projektphase hat eine *Dokumentation* in aussagekräftiger Form vorzuliegen. Nicht die Anzahl der Seiten der Dokumentation ist entscheidend, sondern die Qualität der entwickelten Algorithmen und Programme, die gute Verständlichkeit der Beschreibung und eine hohe Nutzerfreundlichkeit. Einerseits ist klar, daß jedes Programm verbesserungsfähig ist, andererseits muß das vorgelegte Programm eine korrekte Lösung der Aufgaben- oder Problemstellung sein.

7.1. Allgemeines

Das erlernte strukturierte Programmieren ist auch in der Projektphase die grundlegende Arbeitstechnik, weil es eine sehr geeignete Lösungsstrategie ist und insbesondere die Übersichtlichkeit fördert und die Übertragbarkeit von Problemlösungen in andere Programmiersprachen oder auf andere Computer unterstützt.

Aber es gibt bei größeren Projekten auch einige Faktoren, wie Speicherplatz- und Laufzeitprobleme, die dazu zwingen können, im Laufe der Fertigstellung die Regeln der strukturierten Programmierung stellenweise bewußt zu verlassen. Es wurde schon einmal darauf hingewiesen, daß man sich in solchen Fällen für eventuell notwendige Korrekturen eine Programmversion aufbewahren sollte, die vor der Optimierung eine ebenfalls korrekte, strukturierte Lösung realisierte.

Folgende Regeln, die zum Teil nicht der strukturierten Programmierung entsprechen, sind für speicherplatzarme und schnelle BASIC-Programme bedeutsam:

1. Leerstellen zwischen Schlüsselwörtern und Variablen weglassen, solange dies nicht zu syntaktischen Fehlern führt.
2. Kommentare nicht in REM-Zeilen, sondern nur in der schriftlichen Dokumentation erfassen.
3. Möglichst viele Befehle auf eine Anweisungszeile schreiben, um die Zeilenanzahl zu reduzieren.
4. Statt PRINT bzw. REM sind die Abkürzungen ? bzw. ! zu verwenden.
5. Variablennamen sollten möglichst kurz sein, wodurch die Interpretationszeit verringert wird.

6. Unterprogramme sind am Anfang des Programms zu notieren. Das hat allerdings zur Folge, daß beim Programmstart über sie hinweg zum Anfang des Hauptprogramms zu springen ist, damit die Programmabarbeitung nicht in ein Unterprogramm ‚hineinläuft'.
7. Programmteile, die am häufigsten auszuführen sind, sollten am Anfang des Programms stehen.
8. Möglichst selten FOR-NEXT, GOTO und GOSUB verwenden. Alle Sprünge erhöhen die Abarbeitungszeit.
9. Stellen die Bedingungen in IF-THEN-Anweisungen logische Ausdrücke dar, in denen einzelne Bedingungen durch AND oder OR verknüpft sind, so ist es für die Laufzeit günstiger, diese zahlreichen Bedingungen in einzelne IF-THEN-Bedingungen hintereinander zu setzen. Dabei ist diejenige als erste auszuführen, die mit größter Wahrscheinlichkeit am häufigsten eintritt, so daß die anderen Prüfungen erst gar nicht mehr durchgeführt werden müssen.
10. Potenzieren sollte möglichst durch Mehrfach-Multiplikation ersetzt werden, da im allgemeinen das Potenzieren länger dauert und oft Ungenauigkeiten bringt. Die Division dauert länger als die Multiplikation, so daß anstelle einer Division mit dem Kehrwert zu multiplizieren ist.
11. Es ist günstiger, nur ein – nämlich das abzuarbeitende – Programm in den Hauptspeicher zu laden, da der BASIC-Interpreter bis zur letzten Programmzeile sucht und übersetzt.
12. Programmteile, die besonders schnell laufen sollen, sind als Maschinenprogramme zu schreiben und von BASIC mit CALL aufzurufen. Eine maschinenorientierte Programmierung setzt spezielle Kenntnisse von Abläufen im Mikroprozessor voraus. Auf deren Darstellung wurde im Rahmen dieses Buches bewußt verzichtet.

Diese *Gesichtspunkte der Programmoptimierung* wurden erst abschließend genannt, da es nicht Aufgabe eines Buches zu Grundlagen der Informatik sein kann, den Lesern trickreiches ‚Optimieren' zu vermitteln.

Im sechsten Kapitel wurde auf Art und Gliederung der Dokumentation zu Programmen hingewiesen. Dies ist nicht als verbindlich zu betrachten. Vielmehr ist es eine auf Erfahrung mit größeren Softwareprojekten beruhende Empfehlung. In einer Dokumentation zu einem bearbeiteten Projektthema kann auch die Darstellung der individuell erlebten Schwierigkeiten bei der Problemlösung wichtig sein sowie die Beschreibung, wie diese überwunden werden konnten. Über noch offene Fragestellungen und ungelöste Probleme, die unmittelbar dem Projektthema erwachsen, sollte berichtet werden. Das ist nicht nur für den Projektbearbeitenden selbst bedeutsam, sondern auch gerade für nachfolgende Nutzer, die die Arbeit zum Projektthema erneut aufgreifen oder sie unter neuem Aspekt fortsetzen wollen.

7.2. Projekt „Vokabeltrainer"

Als ein Beispiel für die Anfertigung eines Projekts wird die Entwicklung des Programmpakets VOKABELTRAINER (DEUTSCH-ENGLISCH; DEUTSCH-RUSSISCH) durch den Schüler Torsten Reigber der Spezialschule „Heinrich-Hertz" Berlin in Auszügen dargestellt.
Das Projekt reifte im Verlauf von etwa 3 Monaten ausgehend von der ersten (nicht besonders eingrenzenden) Projektthemenstellung: „Angeregt durch den Artikel von Dr. G. Keller in [12] Heft 9 (1984) zum Thema ‚Basic für Heimcomputer Z9001' ist ein Programm zum Lernen von Vokabeln zu entwickeln."

Es ging dabei nicht darum, Software für einen computerunterstützten Unterricht zu entwickeln, sondern an der konkreten Aufgabe zu klären, welche *Programmiertechniken, Datenstrukturen* und *Bildschirmgestaltungsmöglichkeiten* für derartige Programme verwendet werden können.

Erste Arbeitsschritte waren:

1. Lesen des Artikels
2. Entwurf eines Algorithmus zur Abfrage von Vokabeln, die in einer DATA-Zeilen-Liste erfaßt sind. Dabei soll die Richtung
   Deutsch → Englisch bzw. Englisch → Deutsch
   durch den Programmnutzer bestimmt werden können.
3. Codierung des Algorithmus in BASIC für den Z9001.

In der weiteren Arbeit wirkten Gespräche zwischen Betreuer und Schüler auf den Ausbau eines relativ einfachen Programms bis zum vorliegenden Programmpaket. Dabei ergänzten folgende Denkanstöße die Aufgabenstellung:

- Vokabeln werden nicht nur in einer festen Abfolge gelernt. Das Abfragen von Vokabeln sollte auf zufällige Weise erfolgen.
- Vokabeln, die nicht korrekt beherrscht werden, sind häufiger abzufragen. Das hat das Programm zu realisieren.
- Bei einer fehlerhaften Eingabe des Nutzers sollten durch das Programm mehrere Fortsetzungsmöglichkeiten bestehen. Eine zweite Antwort wird durch die Wahl von „1 = Wiederholen" möglich. Glaubt der Nutzer, daß er die Vokabel eigentlich weiß, sie aber vielleicht nicht orthographisch richtig geschrieben hat, so soll er mit „2 = Korrektur" die Möglichkeit der orthographischen Korrektur erhalten. Dabei hat der Computer alle an der richtigen Stelle stehenden Buchstaben der Vokabel anzuzeigen und die fehlerhaften Stellen im Wort durch „-" zu markieren. Fällt dem Nutzer absolut nicht ein, wie die gesuchte Vokabel heißt, hat ihm „3 = RICHTIGE UEBERSETZUNG" zu helfen. Soll die gesuchte Vokabel erst einmal zurückgestellt werden, wählt der Nutzer „4 = FORTSETZUNG".
- Um vom Nutzer keine Programmierfähigkeiten verlangen zu müssen, ist ein separates Programm zur Erstellung von Vokabelsätzen zu entwickeln.
- Da das Lernen von Englisch-Vokabeln auf den Kleincomputern KC 85/1 und KC 85/2 bzw. /3 realisiert werden soll, ist das anfangs für den KC 85/1 entwickelte Programm für die anderen Rechner anzupassen.
- Ein Programm soll das Lernen von Russisch-Vokabeln ermöglichen. Dazu werden kyrillische Buchstaben benötigt. Ausgehend von einem Ausdruck eines russischen Textes über einen Matrixdrucker sind alle kyrillischen Großbuchstaben zu entwerfen und im Computer KC 85/2 zu generieren.
- Die Tastenzahl für die deutschen Buchstaben reicht für das kyrillische Alphabet nicht aus, so daß Sonderzeichentasten mit zu nutzen sind. Dabei müssen Umrechnungen in der ASCII-Codierung erfolgen.

Am Ende der angestrengten Arbeitsphase legte der Schüler ein Programmpaket ordentlich dokumentiert vor, das Auskunft darüber gibt, daß er es versteht, eine Problemstellung algorithmisch aufzubereiten, die Algorithmen in die BASIC-Versionen beider Kleincomputer zu codieren, die Spezifika der Computer – wie z. B. die Zeichengenerierbarkeit des KC 85/2 – vorteilhaft zu nutzen, kleine Maschinenprogramme zu schreiben und Programme anderer Schüler aus der Programmbibliothek der Schule zu nutzen. Das beweisen auch folgende Auszüge aus seiner Dokumentation, wobei die Seiteneinteilung nicht original übernommen wurde:

VOKABELTRAINER
(DEUTSCH – ENGLISCH; DEUTSCH – RUSSISCH)

TORSTEN REIGBER BERLIN, JUNI 1985

INHALT

1.	Theoretische Beschreibung des Programms	1-1
1.1.	Allgemeine Programmbeschreibung	1-1
1.2.	Nutzerinstruktionen	1-2
1.2.1.	Z 9001	1-2
1.2.2.	HC 900	1-3
1.3.	Erstellung eigener Vokabelsätze	1-3
1.3.1.	Z 9001	1-3
1.3.2.	HC 900	1-4
1.4.	Funktionswese des Programms	1-4
1.4.1.	Variablenliste	1-4
1.4.1.1.	Felder	1-4
1.4.1.2.	Einfache Variable	1-4
1.4.2.	Änderungen für die Anpassung an HC 900	1-5
2.	Listings der Programme	2-1
2.1.	Englisch für Z 9001	2-1
2.2.	Abfrageprogramm für englische Vokabeln	2-6
2.3.	Abfrageprogramm für russische Vokabeln	2-10
2.4.	Programm zur Erstellung eigener Vokabelsätze	2-14

1. Theoretische Beschreibung des Programms

1.1. Allgemeine Programmbeschreibung

Das vorliegende Programm dient dem beim Erlernen einer Fremdsprache notwendigen Training der Vokabeln. Es wurde speziell für die englische Sprache geschrieben, läßt sich aber auch für jede beliebige Sprache nutzen, sofern der auf dem Rechner vorhandene Zeichensatz alle notwendigen Buchstaben bereithält. Das Programm ist auf dem Z 9001 vom VEB Robotron und auf dem HC 900 aus Mühlhausen lauffähig.
Der Nutzer hat die Möglichkeit, sich den jeweiligen Vokabelsatz auf dem Bildschirm eines Fernsehgerätes oder Monitors ausgeben zu lassen. Die Abfrage der Vokabeln erfolgt wahlweise in Reihenfolge oder zufällig. Die Richtung der Übersetzung wird vom Nutzer festgelegt. Bei der Abfrage in Reihenfolge ist es zufällig, ob der Rechner mit der ersten oder mit der letzten Vokabel beginnt. Bei zufälliger Reihenfolge fragt der Rechner falsch übersetzte Vokabeln immer wieder ab, bis diese beherrscht werden. Die Zahl der Fehler,

sowie die Anzahl der Übersetzungsversuche wird ständig im unteren Teil des Bildes angezeigt. Bei einem Fehler kann man die falsch übersetzte Vokabel noch einmal übersetzen. Es besteht die Möglichkeit, sich dazu die Anzahl der Buchstaben ausgeben zu lassen. Dabei werden die Buchstaben, die sich an der richtigen Stelle befinden, geschrieben, fehlende oder unrichtige werden durch ‚–' ersetzt. Außerdem kann man sich die richtige Übersetzung ansehen oder die Abfrage mit der nächsten Vokabel fortsetzen. Die Abfrage der Vokabeln läßt sich jederzeit durch Drücken der Taste <ESC> (für HC 900 <HOME>) beenden. Es erfolgt die Anzeige des prozentualen Anteils der richtig übersetzten Vokabeln und nach ca. 10 Sekunden der Rücksprung ins Hauptmenü.
Bei der Abfrage der Vokabeln wird eine Schreibmaschinentastatur simuliert. Für große Buchstaben ist also die Taste <SHIFT> zu betätigen. Das Einschieben von Buchstaben ist mit <INS>, das Löschen mit <SHIFT>+<DEL> (für HC 900 <DEL>) möglich. Die gesamte Eingabezeile ist mit <SHIFT>+<CL LN> (für HC 900 <CLR>) löschbar. Ist die eingegebene Vokabel länger als der in der Zeile verbleibende Platz, wird die Eingabe vom Rechner in die darunterliegende kopiert. Danach ist die weitere Eingabe von Buchstaben möglich (Die Gesamtlänge darf aber 24 Zeichen nicht überschreiten). Nach dem Löschen der Eingabezeile wird die eventuell vom Rechner vorgenommene Verschiebung rückgängig gemacht. Jede Eingabe ist mit <ENTER> abzuschließen. Die Anzahl der Vokabeln ist nur durch die im Rechner zur Verfügung stehende Speicherkapazität beschränkt. Die Erstellung eigener Vokabelsätze ist unter Punkt 1.3. beschrieben.

1.2. Nutzerinstruktionen

1.2.1. Z 9001

(01) Einschalten des Fernsehgerätes und des Kassettenrecorders
(02) Einschalten des Computers
(03) Herstellen der Verbindung zwischen Recorder und Rechner
(04) Eingabe von ‚BASIC' <ENTER>
Erscheint die Meldung ‚START TAPE' so muß der Interpreter von der Systemkassette geladen werden. Dazu muß die Kassette mit der bedruckten Seite nach oben in den Recorder eingelegt werden. Nach dem Rückspulen ist das Zählwerk auf ‚000' zurückzustellen. Der BASIC-Interpreter beginnt ab ca. ‚016'. Während des Vortones ist die Taste <ENTER> zu drücken. Tritt ein ‚BOS'-Error auf, so ist das Band einige Blöcke zurückzuspulen und <ENTER> zu drücken.
(05) Meldung ‚MEMORY SIZE:' mit <ENTER> quittieren
(06) Einlegen der Kassette ‚LEHRPROGRAMME' Seite 1
(07) Rückspulen der Kassette
(08) Bandzählwerk auf ‚000' einstellen
(09) Vorspulen bis ‚002'
(10) Eingabe von ‚CLOAD"ENGLISCH"'
(11) Starten des Recorders
(12) Während des Vortones drücken der Taste <ENTER>
(13) Nach dem Einladen mit <RUN> starten

1.2.2. HC 900

(01) Einschalten des Fernsehgerätes und des Recorders
(02) Einschalten des Heimcomputers
(03) Herstellung der Verbindung zwischen Recorder und Rechner
(04) Eingabe von ‚LOAD'

Dann muß die Systemkassette mit der bedruckten Seite nach oben in den Recorder eingelegt werden. Nach dem Rückspulen ist das Zählwerk auf ‚000' zurückzustellen. Der BASIC-Interpreter beginnt ab ca. ‚016'. Während des Vortones ist die Taste <RETURN> zu drücken. Tritt ein ‚BOS'-Error auf, so ist das Band einige Blöcke zurückzuspulen und <RETURN> zu drücken.

(05) Meldung ‚MEMORY ENDE:' mit <RETURN> quittieren
(06) Einlegen der Kassette ‚HERTZARBEITEN BAND 3' Seite 1
(07) Rückspulen der Kassette
(08) Bandzählwerk auf ‚000' einstellen
(09) Vorspulen bis zum Beginn des Programms ‚ALPHA'
Drücken der Taste <RESET> und Einlesen des Buchstabensatzes wie beim BASIC-Interpreter beschrieben. Danach Eingabe von ‚REBASIC' + <RETURN>
(10) Eingabe von ‚CLOAD' + Programmname
(11) Starten des Recorders
(12) Während des Vortones drücken der Taste <RETURN>
(13) Nach dem Einladen mit 'RUN' + <RETURN> starten

ANMERKUNG
Nach der Änderung des Vokabelsatzes muß das Programm erneut mit <RUN> gestartet werden.

1.3. Erstellung eigener Vokabelsätze

1.3.1. Z 9001

Vor der Eingabe der Vokabeln muß das eventuell vorhandene BASIC-Programm gelöscht werden. Die Vokabelsätze müssen im Bereich der Zeilen 1000 bis 5000 abgelegt werden. Das Vorhandensein der Zeilen 1000 und 5000 ist Grundvoraussetzung für die Funktion des Programmes. Die Eingabe erfolgt in DATA-Zeilen. Durch Komma getrennt wird erst das englische, dann das deutsche Wort eingegeben. In einer Zeile können auch mehrere Vokabelpaare stehen, die dann wieder untereinander durch Komma getrennt werden. Dabei ist aber auf die maximale BASIC-Zeilenlänge von 72 Zeichen zu achten. In der ersten Zeile muß hinter dem Wort ‚DATA' die Anzahl der Vokabelpaare angegeben werden. Dahinter ist gleichfalls ein Komma zu setzen. Am Ende einer DATA-Zeile darf aber kein Komma stehen! Der so erzeugte Vokabelsatz kann dann unter dem Namen ‚VOK' auf der Kassette abgespeichert und für das Programm genutzt werden.

1.3.2. HC 900

Für den HC 900 bestand die Notwendigkeit, ein Programm für die Erstellung der Vokabelsätze zu schreiben, da nicht alle verwendeten Buchstaben auf der Tastatur erreichbar sind. Dieses Programm ermöglicht es, Vokabelsätze sowohl für die englische, als auch für die russische Sprache zu erstellen. Für die Funktion dieses Programms ist aber das Vorhandensein des Maschinenunterprogramms ‚BODOBAS' notwendig. Es befindet sich gleichfalls auf der Hertzarbeitenkassette und wird wie der Zeichensatz eingeladen. Das Programm wird mit ‚RUN' gestartet. Danach wird die Sprache für den Vokabelsatz ausgewählt.
Nun erfolgt die Eingabe der Vokabelpaare. Eventuelles Überschreiben der Vokabeln nach der Betätigung der Taste <RETURN> hat keinerlei Auswirkungen auf die Abspeicherung. Die Eingabe wird durch einfache Betätigung der Taste <RETURN> bei der Abfrage der fremdsprachigen Vokabel beendet, worauf der Rechner dazu auffordert, den erstellten Datensatz auf Magnetband abzuspeichern.

1.4. Funktionsweise des Programms

1.4.1. Variablenliste

1.4.1.1. Felder

A – ASCII-Codes der richtigen Übersetzung
B – ASCII-Code der eingegebenen Vokabel
VO$ – (zweidimensional) (0,X) – englische Vokabel
(1,X) – deutsche Vokabel
VO – Merkfeld, ob Vokabel falsch übersetzt
– VO(0) Zähler, wenn VO(0)=3, dann Suche nach falsch übersetzten Vokabeln

1.4.1.2. Einfache Variablen

N1 – Anzahl der Vokabeln im Vokabelsatz
I – Allgemeine Laufvariable – Nummer abgefragter Vokabel
A$ – Allgemeine Variable für Eingaben,
Richtige Übersetzung bei Abfrage
KE – (Kennziffer) Nummer des aufgerufenen Programmteils
RE$ – (Reihenfolge)
J – Allgemeine Zählvariable,
Nummer des aktuellen Buchstabens bei der Eingabe
FE – (Fehler) Anzahl der gemachten Fehler
ZA – (Zahl) Zähler der Übersetzungsversuche
RI – (Richtung) der Übersetzung
N2 – Hilfsvariable
A1$ – zu übersetzende Vokabel
F – Fehler bei der aktuellen Übersetzung
W$ – Hilfsvariable für Eingaben
EN – (Entscheidung) Fortsetzung nach Fehler
ER – Prozentualer Anteil der richtig übersetzten Vokabeln
V$ – Vorsatzzeichen bei Abfrage
K – Allgemeine Zählvariable
B$ – Aktuelles Eingabezeichen
B – ASCII-Code des aktuellen Eingabezeichens
P – Merkvariable für Zeilenanfang

1.4.2. Änderungen für die Anpassung an den HC 900

Das Programm wurde auf den beim HC 900 größeren Bildschirm erweitert. Gleichzeitig wurde es notwendig, die Ausgaberoutinen des Programms auf die im HC 900 vorhandene BASIC-Version anzupassen. Hier ist besonders wichtig, darauf zu achten, daß bei ‚PRINTAT'-Befehlen für die Trennung der einzelnen Ausgabedaten ohne Tabulator ein Semikolon, nicht ein Komma wie beim Z 9001 verlangt wird. Auch erfolgt bei ‚PRINTAT'-Befehlen außerhalb des WINDOWS ein Rollen des gesamten Bildschirms, wenn die Zeile nicht mit einem Semikolon abgeschlossen wird. Solche, wenn auch nur minimalen Unterschiede, müssen für die Übertragung von BASIC-Programmen zwischen den beiden Rechnermodellen unbedingt beachtet werden.

Ein anderer sehr wesentlicher Unterschied ist die Tatsache, daß der HC 900 bei der Abfrage der Tastatur mittels ‚INKEY$' keinen Cursor auf dem Bildschirm darstellt. Um diesen Mangel zu umgehen, wird am Anfang des Programms ein kleines Maschinenprogramm in einen freien Speicherplatz geschrieben.

```
0000 CD03F0   CALL F003    ;A:=NUMMER DER AKTUELLEN TASTE
0003 16       DEFB 16
0004 B7       OR A         ;WENN A=0 DANN Z-FLAG SETZEN
0005 28F9     JR NZ 0000   ;RUECKSPRUNG FALLS KEINE TASTE
0007 47       LD B,A       ;B:=A
0008 AF       XOR A        ;A:=0
0009 CDB412   CALL 12B4    ;UEBERGABE DER TASTENNUMMER AN BASIC
000C C9       RET          ;RUECKSPRUNG
```

Dieses Hilfsprogramm wird von BASIC aus mit Hilfe der USR-Funktion aufgerufen und ermöglicht die Abfrage der Tastatur mit sichtbarem Cursor.

Aufgrund der grafischen Fähigkeiten des HC 900 konnte der Zeichensatz für deutsche Vokabeln durch die Umlaute und das <SZ> erweitert werden. Auch der kyrillische Zeichensatz wurde aufgenommen.
(Es folgen in der Dokumentation die Programmlisten entsprechend der Gliederung, die hier nicht dargestellt werden.)

8 Weitere Problemklassen und andere Programmiersprachen

Im großen Buch der Natur kann nur der lesen,
der die Sprache kennt,
in welcher dieses Buch geschrieben ist, ...
(Galileo Galilei)

In den bisherigen Kapiteln haben wir uns auf die Erstellung von Algorithmen und Programmen – unter Verwendung von BASIC – konzentriert. Aber nicht alle Menschen müssen programmieren können und BASIC ist nur eine von vielen möglichen Programmiersprachen.

8.1. Weitere Problemklassen

Der volkswirtschaftlich unbedingt notwendige breite Einsatz der Informations- und Kommunikationstechnik erfordert, daß viele Berufstätige diese Technik anwenden lernen. Dabei liegt der Schwerpunkt auf der *Nutzung dieser Technik als Hilfsmittel oder Werkzeug bei der täglichen Arbeit.* Dazu werden in ständig zunehmenden Maße nutzerfreundliche Programmsysteme bereitgestellt.
Programme bzw. Programmsysteme, die den Computer zum Werkzeug beim Problemlösen machen, nennt man *Software-Werkzeuge* (engl.: software tools). Dabei handelt es sich um Software, welche es dem naiven Nutzer ohne bzw. mit nur geringer Programmiererfahrung und mit minimalen Gerätekenntnissen gestattet, den Computer zur Unterstützung in der täglichen Arbeit selbständig einzusetzen. Der Unterschied zwischen einem Anwenderprogramm und einem Software-Werkzeug besteht darin, daß das Anwenderprogramm nur für einen ganz speziellen Zweck eingesetzt werden kann, während das Software-Werkzeug für eine ganze Klasse gleichgelagerter Probleme genutzt werden kann. Ein Textverarbeitungssystem kann vom Nutzer zum Briefe-, Zeitschriftenartikel-, Bücher- oder Promotionsarbeiten-Schreiben verwendet werden. Demgegenüber kann man das Programm „KAPREKAR-BEWEIS" aber nur zum Beweis der Hypothese von Kaprekar nutzen.

Solche Werkzeuge gibt es gegenwärtig für folgende Aufgaben:

- Bearbeitung von Texten,
- Bearbeitung von Datenbeständen,
- Bearbeitung von Tabellen und
- Bearbeitung von Grafiken.

Zu den heute am häufigsten genutzten Software-Werkzeugen gehören *Textverarbeitungssysteme.* Vertreter dieser Gruppe sind:

- TP für die PC 1715 oder BC 5120,
- TEXOR und WORDPRO für die KC 85/2 und /3,
- TEXT 1 für den KC 85/1 und KC 87 oder
- WORDSTAR als einer der Stammväter dieser Gruppe.

Textverarbeitungssysteme unterstützen alle vier Phasen des Verarbeitungsprozesses, den ein Text durchläuft:

- in der *Entwurfsphase* durch Bereitstellen von Standardtexten und Textbausteinen,
- in der *Phase der Textfixierung* durch einen einfach zu bedienenden, aber leistungsfähigen Editor,
- in der *Phase der Textumformung* durch den genannten Editor und eine reichhaltige Auswahl an Funktionen zur Gestaltung des Textes (Formatieren, Tabulatorfunktionen, Einfügen oder Löschen, Unterstreichen, automatische Seitennumerierung u. a.),
- bei der *Textverwendung* durch Unterstützung der unterschiedlichsten Druckertypen, beim *Versenden* auf Datenträgern, beim *Archivieren*.

Die meisten Textverarbeitungssysteme kann man ohne jegliche Programmierkenntnisse nutzen, da diese Systeme im wesentlichen durch Einzelkommandos gesteuert werden können.
Mit den meisten Textverarbeitungssystemen kann man auch Programmtexte in den unterschiedlichsten Programmiersprachen erfassen und bearbeiten.

In die zweite genannte Gruppe von Software-Werkzeugen gehören solche Programmsysteme wie
- REDABAS für die PC 1715 und BC 5120 oder
- TOPAS für ESER-Rechner.

Es handelt sich dabei um *Datenbanksysteme* als leistungsfähigste Vertreter der Software-Werkzeuge zur Bearbeitung von Datenbeständen.

Das Erfassen, Katalogisieren, Ändern, Sortieren und Wiedergewinnen von Daten ist die Hauptaufgabe der Programmsysteme dieser Gruppe. Der Nutzer legt die Struktur seiner Daten mittels einer Beschreibungssprache fest, speichert die Daten ein und kann sie dann nach seinem Belieben sortieren, mischen und auswerten. Dazu kann er die vielfältigen Funktionen solcher Systeme nutzen. Bestandteile solcher Systeme sind eine Beschreibungssprache für die Daten und ihre Struktur, die notwendigen Programme zur Umsetzung dieser Beschreibungssprache in die interne Form und ein Datenverwaltungssystem.
Durch die genannten Datenbanksysteme hat der Nutzer ein mächtiges Werkzeug zur *Erstellung, Bearbeitung* und *Verwaltung seiner Datenbestände* zur Verfügung, das er aber nur sinnvoll nutzen kann, wenn entsprechende Algorithmen erarbeitet wurden. Unter Steuerung des Systems können diese Algorithmen im Direktmodus oder in Form von Programmen abgearbeitet werden. Die Algorithmen bestehen aus den bekannten Bausteinen (Folge, Auswahl und Wiederholung bzw. Unteralgorithmen) und die Programmiersprache hat die Eigenschaften einer höheren Programmiersprache. Da diese Sprachen einen kleinen Wortschatz haben, sind sie relativ leicht zu erlernen und anzuwenden. Auf der Grundlage einer bestehenden Datenbasis kann ein Nutzer sehr bald Auswertungsprogramme nach eigenen Vorstellungen erstellen. Es besteht natürlich auch die Möglichkeit der Programmierung durch entsprechende Fachleute.

Viele Aufgabenstellungen in der Ökonomie, der Wissenschaft und der Technik lassen sich als *Tabellen* darstellen. Es ist deshalb nicht erstaunlich, daß die ersten kommerziellen Software-Werkzeuge gerade für diesen Problemkreis erstellt wurden. Diese *Tabellenkalkulationssysteme* erlauben das Edieren, Berechnen, Formatieren, Speichern und Drukken von Tabelleninhalten. Dabei können in den Zeilen oder Spalten solcher Tabellen nicht nur Text- oder Zahlkonstanten enthalten sein, vielmehr können es auch Aufrufe von Unterprogrammen, Berechnungsformeln oder Verweise auf andere Zeilen sein. Dadurch sind Tabellenkalkulationssysteme schon einfache, interaktive Programmierumgebungen.

Im Gegensatz zur Programmierung in einer höheren Programmiersprache findet in diesen Systemen die Programmierung in einer Tabelle zur Ermittlung einzelner Tabellenelemente statt. Die Reihenfolge, in der man die Relationen zwischen den Tabellenelementen definiert, ist beliebig. Unter der Voraussetzung, daß alle Eingangsgrößen zur Ermittlung des Tabellenelements bekannt sind, kann die Definition sofort nach der Programmierung getestet werden.
Tabellenkalkulationssysteme stellen recht leistungsfähige Software-Werkzeuge dar, die nicht nur in der Ökonomie genutzt werden. Solche Systeme können auch in den Naturwissenschaften oder zur wertmäßigen Simulation eingesetzt werden. So könnte man die Berechnung und die Zahlenausgabe zur Räuber-Beute-Simulation auch in einem Rechenblatt eines Tabellenkalkulationssystems realisieren.

Die letzte Gruppe der Software-Werkzeuge betrifft die *Grafik-Systeme*. Solche Grafik-Systeme unterstützen die Herstellung grafischer Darstellungen, ihre Änderung und Verarbeitung in unterschiedlichster Weise. Sie bieten aber auch die Möglichkeit, komplexe Zeichnungen aus einfachen Grundelementen zu erstellen.
Für Kleincomputer gibt es hier nur einfache Systeme, die die Erstellung und Modifizierung von Zeichnungen nicht zu komplexer Art unterstützen. Grafik-Systeme für den Konstrukteur, den Architekten, den Designer oder den Zeichentrickfilmgestalter erfordern eine sehr leistungsfähige Gerätetechnik mit sehr hohen Rechengeschwindigkeiten und großen Hauptspeichern.

Die Entwicklung geht zwischenzeitlich von der Erstellung spezieller Software-Werkzeuge immer stärker zur Entwicklung sogenannter *integrierter Systeme,* die die Funktionen der einzelnen genannten Werkzeuge vereinigen. Die wesentlichen Vorteile bestehen darin, daß der Nutzer eines solchen komplexen Werkzeugs nur eine Sprache zu erlernen hat und daß Daten, Texte und Tabellen zwischen den Einzelkomponenten problemlos ausgetauscht werden können. Nachteilig ist bei solchen Systemen, daß die Voraussetzungen an Speicherplatz und Verarbeitungsgeschwindigkeit natürlich erheblich sind. Gegenwärtig bestehende integrierte Systeme setzen Computer mit mindestens 256 KByte Hauptspeicher und mindestens einem Disketten- oder Festplattenlaufwerk voraus.

8.2. Vergleich einiger Programmiersprachen

Es wurde schon darauf hingewiesen, daß es eine Vielzahl von Programmiersprachen – gegenwärtig über 1000 – gibt. Auch der Informatikanfänger sollte wesentliche Programmiersprachen nicht nur dem Namen nach kennen. Deshalb werden in diesem Kapitel einige vorgestellt, die gegenwärtig oder in naher Zukunft eine Rolle in der Informatikausbildung spielen.
Es werden die Sprachen

- **BASIC,**
- **PASCAL,**
- **FORTH,**
- **LOGO** und
- **PROLOG**

in ihren wichtigsten Charakteristika vorgestellt. Neben verbalen Ausführungen zu wünschenswerten Spracheigenschaften wird ein Beispielprogramm (außer bei PROLOG) als Beschreibungsmittel verwendet. Das Beispielprogramm soll die Aufgabe des Heraussuchens aller pythagoräischen Zahlentripel aus dem Bereich von 1 bis 50 realisieren.

BASIC

BASIC ist zeilenorientiert und jede Programmzeile muß am Zeilenanfang eine Nummer besitzen. Die aufsteigende Folge der Zeilennummern bestimmt im allgemeinen die Abarbeitungsfolge. BASIC-Programme werden in den meisten Fällen interpretativ abgearbeitet. Werden dabei Fehler gefunden, so stoppt die Abarbeitung beim ersten Fehler mit einem Hinweis. Direktmodus ist möglich, ebenso wie der Einzelschrittmodus zur Fehlersuche. Wenn es die Hardware erlaubt, sind Befehle für Punktgrafik (Pixel-Grafik) verfügbar.
Programme können meistens nur mit größeren Änderungen auf unterschiedliche Rechner übertragen (portiert) werden, da es sehr viele unterschiedliche Dialekte gibt. BASIC stellt nur geringe Hardwareanforderungen (>=12 KByte Speicherplatz, Kassette als externes Speichermedium).

Wesentliche Sprachmittel:

- IF bed. THEN anw1 [ELSE an2],
- ON natürl. zahl GOTO (oder GOSUB) zeilennummer,
- FOR variable=anfang TO ende [STEP sw] : ... : NEXT,
- als Datentypen nur Zahlen und Zeichenketten als skalare Variable oder Felder,
- Funktionen mit Parameter in einer Zeile, sonst keine modularen Strukturen und keine lokalen Variablen.

Programmbeispiel:

```
10  REM ===== Pythagoraeische Zahlen =====
20  REM Heraussuchen pythag. Zahlen zwischen 1 und 50
30  REM Variablen:
40  REM   A,B,C - ganze Zahlen
50  CLS
60  PRINT "  Pythagoraeische Zahlen"
70  FOR C=5 TO 50
80     FOR B=INT(C/2) TO C-1
90        FOR A=3 TO B
100          IF A*A+B*B=C*C THEN PRINT A,B,C
110       NEXT A
120    NEXT B
130 NEXT C
140 END
```

PASCAL

PASCAL ist eine prozedurorientierte Programmiersprache mit einem strengen Typkonzept. Es kennt keine Zeilennumerierung und verlangt die Vereinbarung aller im Programm verwendeten Variablen. PASCAL ist eine höhere Compilersprache, von der es trotz Standardisierung einige unterschiedliche Dialekte gibt. Auf der Klein- und Arbeitsplatzcomputerhardware hat sich in großem Umfang das System Turbo-PASCAL durchgesetzt, das über eine Programmierumgebung in Form eines leistungsfähigen Editors, eines schnellen Compilers und gewisser Testunterstützungen verfügt. Die Handhabung dieses Systems ist relativ einfach. Syntaxfehler sind meist einfach zu korrigieren, da in diesem Fall durch das System sofort der Editor aufgerufen und die Fehlerstelle im Quelltext ange-

zeigt wird. Ein Programm kann erst abgearbeitet werden, wenn es keine Syntaxfehler mehr enthält. Einige PASCAL-Dialekte kennen nur interne Prozeduren, die textlicher Bestandteil des Hauptprogramms sein müssen. Es wird zwischen lokalen und globalen Variablen unterschieden. In Turbo-PASCAL gibt es auch Grafik-Befehle für Punktgrafik.

Hardware-Voraussetzungen sind ein Hauptspeicher mit mindestens 48 KByte, bei Turbo-PASCAL 64 KByte und Diskette.

Sowohl PASCAL als auch BASIC sind Vertreter der Klasse der *imperativen* (befehlsorientierten) *Programmiersprachen.*

Wesentliche Sprachmittel:

- IF bedingung THEN anw1 ELSE anw2,
- CASE ... DO ... (Mehrfachverzweigung),
- WHILE bed. DO anweisung (kopfgest. Schleife),
- REPEAT anweisung UNTIL bed. (fußgest. Schleife),
- FOR var:=anf TO ende DO anw (Zählschleife),
- Prozeduren mit Parameterübergabe,
- vielfältige Datentypen, z. B. Feld, Menge, Verbund, Aufzählung, Zeiger, Datei.

Programmbeispiel:

```
PROGRAM Pythzahl;
(* Berechnung von pythag. Zahlentripel zwischen 1 u. 50 *)

VAR Kathete1, Kathete2, Hypotenuse : INTEGER;
PROCEDURE Ueberschrift;
  BEGIN
    CLRSCR;     (* Bildschirmloeschen *)
    WRITELN ('Pythagoraeische Zahlentripel':45);
    WRITELN
  END;
PROCEDURE Verarbeitung;
 BEGIN
  FOR Hypotenuse:=5 TO 50 DO
   BEGIN
    FOR Kathede2:=TRUNC(Hypotenuse/2) TO Hypotenus-1 DO
     BEGIN
      FOR Kathete1:=3 TO Kathete2 DO
       IF Kathete1*Kathete1 + Kathete2*Kathete2=
            Hypotenuse*Hypotenuse THEN WRITELN
            (Kathete1:8,Kathete2:8,Hypothenuse:8)
     END
    END;
    WRITELN
 END;

BEGIN
 Ueberschrift;
 Verarbeitung
END.
```

FORTH

FORTH ist nicht nur eine höhere Programmiersprache sondern eine Programmierumgebung und ein Betriebssystem. FORTH wurde zur Steuerung eines Radioteleskops als eine offene Sprache entworfen. Jeder Programmierer kann sich – auf dem standardisierten Grundwortschatz aufbauend – sein eigenes FORTH definieren. Zwischenzeitlich wurde ein Prozessor entwickelt, der diese Sprache als „Maschinensprache" verwendet.
FORTH kann sowohl im Direkt- als auch im Programm-Modus abgearbeitet werden. Ein leistungsfähiger Editor ist Bestandteil des FORTH-Systems. Die Sprache unterstützt sehr stark das strukturierte Programmieren, da jedes neue Wort ein Modul ist und vereinbart wird. Die zentrale Datenstruktur in FORTH ist der Stack – ein Kellerspeicher, der nach dem LIFO-Prinzip (last In first out) arbeitet. Jedes FORTH-Programm besteht im wesentlichen aus Anweisungen zur Manipulation dieses Kellerspeichers.
In der Schreibweise unterscheidet sich ein FORTH-Programm wesentlich von den bisher bekannten Sprachen, da die umgekehrte polnische Notation (UPN) verwendet wird. UPN bedeutet, daß z. B. die Multiplikation 3 * 5 in Verbindung mit der Ergebnisanzeige als FORTH-Befehlsfolge lautet
3 5 * .
Dabei bewirkt der Punkt die Ausgabe des obersten Kellerelements. Ohne Übung sind FORTH-Programme relativ schwer lesbar, wie auch das Beispielprogramm zeigen wird.
FORTH ist eine höhere Programmiersprache mit sehr leichtem Zugriff zu den Routinen auf Maschinensprachebene und wesentlich schneller als BASIC. Aufgrund seiner Eigenschaften wird FORTH nur in Form von Implementierungen komplexer Programme im Bildungsbereich eine Rolle spielen können. Ein Einsatz als Unterrichtssprache ist nicht sinnvoll, es sei denn im Rahmen der Berufsausbildung für Steuerungsaufgaben im Echtzeitbetrieb.
FORTH stellt nur geringe Hardwareforderungen. Das System mit dem Grundwortschatz benötigt mindestens 8 KByte Speicher und wenigstens Kassettenspeicher.

Wesentliche Sprachmittel.

- Befehle zur Veränderung des Kellerspeichers,
- IF ... ENDIF,
- DO ... LOOP (Zählschleife),
- WHILE – Schleife,
- keine Sprunganweisung.

Programmbeispiel:

```
0 VARIABLE A
: PYTRIP  ."Pythagoraeische Zahlen"
    50 3 DO I DUP SQ
       50 I DO DUP I DUP A !
          SQ +
          71 I DO DUP I SQ - DUP
             0= IF I A @ 6 PICK
                CR PT PT PT ENDIF
             0< IF LEAVE ENDIF
          LOOP DROP
       LOOP DROP DROP
    LOOP CR ;  PYTRIP
```

LOGO

LOGO ist eine interaktive Programmiersprache, die einen Editor und ein einfaches Dateisystem beinhaltet. Sie ist besonders gut für den Programmieranfänger – bis ins Vorschulalter zurück – geeignet und zugleich aber auch eine sehr leistungsfähige Sprache für anspruchsvolle Probleme. LOGO gehört in die Klasse der *funktionalen* und *applikativen Programmiersprachen* und basiert auf einem Listenkonzept. Rekursivität ist in vollem Umfange möglich. Der Sprachumfang ist durch den Nutzer erweiterbar.
Ein LOGO-Programm wird interpretativ abgearbeitet und jeder Befehl bzw. jeder Modul kann im Direktmodus abgearbeitet oder getestet werden. Wesentlicher Bestandteil der Sprache ist ein Grafikteil, die sogenannte Turtle- (Schildkröten-) Grafik, bei dessen Verwendung selbst durch Vorschulkinder der Einstieg in die Programmierung erlernt werden kann. Mit wenigen Grundbefehlen ist selbständiges Arbeiten möglich. Durch die volle Rekursivität sind Module als vielseitige Werkzeuge erlebbar.

Das LOGO-System benötigt 64 KByte Speicher und möglichst Diskettenlaufwerk.
Die Programmiersprache LOGO wird u. a. in der Sowjetunion und in der VR Bulgarien in verschiedenen Projekten der Volksbildung erprobt.

Wesentliche Sprachmittel:

- IF bedingung [anweisung1] [anweisung2]
- REPEAT anzahl [anweisung] (Zählschleife),
- andere Schleifen können durch Rekursivität realisiert werden,
- andere Kontrollstrukturen können selbst definiert werden,
- Module haben lokale Variable,
- beliebige Daten können zu Listen zusammengefaßt werden,
- einfache Listenbearbeitungsbefehle.

Programmbeispiel:

```
TO PYTHA.ZAHLEN
 CLESRTEXT
 PRINT [Heraussuchen pythagoraeischer Zahlen]
 PRINT "
 (LOCAL "SQRT2 "AMAX "BMAX "CMAX)
 MAKE "SQRT2 SQRT2
 MAKE "CMAX 50
 PYTH.C 5
 PRINT "
END

TO PYTH.C :C
 IF :C > :CMAX [STOP]
 MAKE "BMAX :C - 1
 PYTH.B 1 + INT :C / SQRT2
 PYTH.C :C + 1
END

TO PYTH.B :B
 IF :B > :BMAX [STOP]
 MAKE "AMAX :B - 1
 PYTH.A 3
```

```
 PYTH.B :B + 1
END

TO PYTH.A :
 IF :A > :AMAX [STOP]
 TEST :A * :A + :B * :B = :C * :C
 IFTRUE [TAB 10 TYPE :A TAB 15 TYPE :B TAB 20 PRINT :C]
 PYTH.A :A + 1
END
```

PROLOG

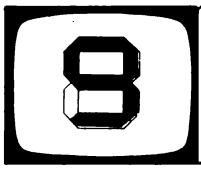
Bei PROLOG handelt es sich um eine prädikative Programmiersprache, die im Zusammenhang mit der fünften Rechnergeneration an Bedeutung gewinnen wird. Bei diesen Sprachen wird Programmieren als Beweisen in einem System von Tatsachen und Schlußfolgerungen aufgefaßt. Durch den Nutzer wird eine Menge von Fakten (gültige Prädikate) und Regeln vorgegeben und das Programm hat dann die Aufgabe zu prüfen, ob eine gestellte Frage als richtig oder falsch zu beantworten ist. Die Aussage

Franz liest das Informatikbuch

wird in PROLOG-Notation formuliert als

„liest—das—Informatikbuch (Franz)".

Das heißt, das Subjekt „Franz" hat die Eigenschaft „liest—das—Informatikbuch".
Programmieren in PROLOG heißt, dem Computer alle Fakten und Regeln vorzugeben, die für die Lösung eines Problems wesentlich sind. Die eigentliche Problemlösung wird nicht explizit als Algorithmus programmiert, sie ist implizit in der Menge der gegebenen Fakten und Regeln bereits vorhanden und wird vom PROLOG-System selbständig gefunden.
Die Sprache PROLOG ist ein ziemlich mächtiges Werkzeug, das einen Personalcomputer mit mindestens 384 KByte Arbeitsspeicher und mindestens 2 Diskettenlaufwerken erfordert. In jedem PROLOG-System ist eine Datenbank integriert.
In letzter Zeit wird ein Turbo-PROLOG-System mit leistungsfähigem Editor und einem schnellen Compiler angeboten, der die Arbeit mit der Sprache wesentlich vereinfacht.
Die Sprache PROLOG wird in der weiteren Entwicklung und Anwendung der Informatik in der Schule eventuell mal im Zusammenhang mit komplexen Lehr- und Lernsystemen und großen Wissens- und Methodenbanken eine Rolle spielen können. Als Programmiersprache für den täglichen Gebrauch ist sie nicht geeignet.



Anhang

Im Text behandelte BASIC-Sprachelemente

ABS
AND
ASC
ATN
AUTO

BEEP
BREAK

CALL*
CHR$
CIRCLE
CLEAR
CLOAD
CLOAD*
CLS
COLOR
CONT
COS
CSAVE
CSAVE*

DATA
DEF FN
DELETE
DIM

EDIT
END
ENTER
EXP

FOR...TO...STEP...NEXT

GOSUB
GOTO

INKEY$
INT
IF...THEN...
IF...THEN...:ELSE...
INK
INP
INPUT
INSTR

JOYST

LEFT$
LEN
LET
LINE
LIST
LIST#
LOAD
LOAD#
LOCATE

MID$
MODIFY
NEW
NOT

OR
ON...GOSUB...
ON...GOTO...
OUT

PAPER
PAUSE
PEEK
PRESET
PRINT
PRINT AT
PRINT TAB
PSET

RANDOMIZE
READ
REM
RENUMBER
RESTORE
RETURN
RIGHT$
RND
RUN

SGN
SIN
SOUND
SQR
STOP
STRING$
STR$
SWITCH

TAN

VAL

WINDOW

Die vollständige Liste der BASIC-Sprachelemente ist den Handbüchern zu den Computern zu entnehmen!

Im Text behandelte BASIC-Fehlermeldungen

BS – Feldelement außerhalb dimensionierten Bereichs
DD – doppelt definiertes Feld
FC – unzulässiger Funktionsaufruf
OD – zu wenig Daten in DATA-Zeilen
OS – Speicherplatz für Zeichenketten reicht nicht
OV – numerischer Überlauf
RG – RETURN trat vor GOSUB auf
SN – syntaktischer Fehler

Die vollständige Liste der Fehlermeldungen sind den Handbüchern zu den benutzten Computern zu entnehmen!

Literaturverzeichnis

[1] KC 85/3-BASIC-HANDBUCH. VEB Mikroelektronik Mühlhausen, 1986
[2] Busch, Haase, Keller, Nowottne: Heimcomputer robotron Z 9001, Programmierhandbuch. VEB Robotron-Meßelektronik „Otto Schön" Dresden
[3] Rücke, Schmoll: Lexikon für Information und Dokumentation, VEB Bibliographisches Institut, Leipzig 1984
[4] Möller, G.: Dein Weg zum guten Stil, Volk und Wissen Volkseigener Verlag Berlin
[5] Lohse, H.: Elementare Statistik. Volk und Wissen Volkseigener Verlag, Berlin 1983
[6] Autorenkollektiv: Kleine Enzyklopädie Mathematik. VEB Bibliographisches Institut Leipzig
[7] ZK der SED, Drittes Plenum, Dietz Verlag, Berlin 1986
[8] Schmieder, T. in Rechentechnik – Datenverarbeitung. Verlag Die Wirtschaft, Berlin 1985
[9] Kleinstrechner – TIPS, VEB Fachbuchverlag Leipzig, 1986 (Reihe)
[10] Kerner, I.: Numerische Mathematik mit Kleinstrechnern, VEB Deutscher Verlag der Wissenschaften, Berlin 1985
[11] alpha, Volk und Wissen Volkseigener Verlag Berlin, Heft 5 und 6 (1986)
[12] radio fernsehen elektronik, Verlag Technik, Berlin, 33. Jahrgang (1984)
[13] Dejkstra, E. W.: Notes on Structured Programming. Academic Press, New York 1972
[14] Fischer, P.: BASIC für Anfänger. Verlag Die Wirtschaft, Berlin 1988
[15] Grote, U.; H. Völz: BASIC Einmaleins des Programmierens. Urania-Verlag Leipzig Jena · Berlin 1987
[16] Heblik, P.: Wissensspeicher BASIC. Volkseigener Verlag Volk und Wissen, Berlin 1986
[17] Groh, J.: Kleincomputer-Fibel. Akademie-Verlag, Berlin 1986
[18] Rothhardt, G.: Praxis der Softwareentwicklung. VEB Verlag Technik, Berlin 1987
[19] Löthe, H.; W. Quehl: Systematisches Arbeiten mit BASIC. B. G. Teubner, Stuttgart 1982
[20] Harbeck, G. u. a.: Metzler Informatik. J. B. Metzlersche Verlagsbuchhandlung, Stuttgart 1984

Sachwortregister

A
Abakus 157
Adreßbus 10
Algorithmus
–, sequentieller 9
–, nicht-sequentieller 9
Alltagsalgorithmen 25
Alphabet 9
Alternative 57
Anfangswert 33, 76
Anlegen einer Datei 165
Anweisung 9
Anweisungszeile 38
Anwenderkoordinaten 145
Ahwenderprogramm 272
Anwendersoftware 12
ASCII 86
ASCII-Zahlen 86
Aufruf von Unteralgorithmen 121, 147
Aufruf von benutzerdefinierten Funktionen 126
Ausdruck 32
Ausgabe
–, tabellarische 27
–, stellengerechte 69
Ausgabeeinheit 14
Ausgabegerät 11
Ausgabeliste 56
Ausgeben einer Datei 171
Austauschsort 195
Auswahl
–, einseitige 57, 61
–, zweiseitige 57f.
–, mehrseitige 57, 65
Auswahlmenü 137
Auswahlsort 186

B
BASIC-Interpreter 27
Bedingung 32
Befehl 9
Befehlsanalyse 200
Befehlseingabebereich 14
Befehlsfolge 17
Befehlsmenü 138
Befehlsvorrat 18
Bemerkung 34
Benutzersysteme 162
Betriebssystem 12
Bevölkerungspyramide 248
Bezeichner 30
Bildpunkt 14, 70
Bildpunktraster 70
Bildschirm 11
Bildschirmgestaltung 93
Bildschirmkoordinaten 145
Bildschirmprotokoll 95
Bildschirmzeile 27
Binärwort 8
Binomialkoeffizient 122
Biosystem 247
Bisektionsmethode 235
Bit 8
Biteingabe 262
Bottom-up-Verfahren 44
Bubble-Sort 225
Bus 10
Byte 8

C
CAD 21, 159
CAM 159
Chaos-Theorie 219
charakter (engl.) 87
CIM 159
Code 9
Codierung des Algorithmus 45
Computer 9
Computer-Zahlenmenge 215
CPU 10
Cursor (auch Kursor) 141
Cursortasten 141
CTC (Zeitgeber-Schaltkreis)

D
Darstellung, binäre 215
Darstellungsmöglichkeiten für Algorithmen 46
DATA-Zeilen 82
Datei 162
Daten
–, alphabetische 8
–, alphanumerische 8
–, numerische 8
Datenbank 162
Datenbanksystem 162
Datenbus 10
Datenelement 8
Datenliste 146
Datensatz 163

Datensatzkomponente 181
Datenstruktur Feld 128
Datenzeiger 82
determiniert 9
Dezimalzahlen 33
Dialog 16
Dienstprogramme 13
DIM-Anweisung 128
Dimension 128
Direktmodus 14
Diskette (Floppy disk) 12
Diskettengerät 11
Dokumentation 46, 262
Drucker 11
Dualarithmetik 96

E
e (Eulersche Zahl) 216
Echtzeitbetrieb 247
Edier-Modus 31
Editor 31
Ein-/Ausgabeverwaltung 13
Einfüge-Sort 195
Eingabe 14
Eingabebefehl 55
Eingabeeinheit 14
Eingabegerät 11
Eingabeliste 55
Eingabemaske 174
Eingangsvariable 130
Elementarbaustein, algorithmischer 44
Elementaroperation 9, 46
Ellipse 214
Empfänger 7
Endwert 33, 76
Entwurf, computergestützter 159
– eines Lösungsplans 43
Erfassungsmenü 138
Ergibtanweisung 32
ergibt sich aus 33
Ergibtzeichen 32
ETR 215, 219
EVA-Prinzip 14
Experimentgestaltung 253
Expertensystem 43
Externspeicherverwaltung 13

F
Fakultät 203
Fallauswahl 65
Farbcodetabelle 79
Fehlereffekt 33
Fehlerfortpflanzung 216
Feld
–, eindimensionales 128
–, zweidimensionales 128
Feldelemente 128
Feldname 128
file (engl.) 163
Firmware 13
Folge 51, 216
–, divergente 220
–, konvergente 220
FORTH 277
Funktion, benutzerdefinierte 126
Funktionsaufruf, unzulässiger 5?
Funktionstasten 213

G
Galtonbrett 153
Genauigkeitsbedingung 232
Gleichheitszeichen 32
Gleichung, diophantische 101
Gleichverteilung 237
Grafik-System 274
Grobgrafik 107
Grundstrukturen 51

H
Hardcopy 153
Hardware 9
Hardwareergänzung 254
Häufigkeit, relative 238
Hauptmenü 116, 168
Hauptspeicher 10
Heron-Verfahren 232
Hilfsprogramme 13
Hilfsseite 198
Hilfsvariable 62
Histogramm 152

I
Informatik 7
–, angewandte 7
Informationsbits 8
Informationseinheit 9
IGELGEOMETRIE 14, 197
Igel-Befehl 198
Igel-Prozedur 198
Ikonenmenü 138, 140
Index 128
Index-Sort 188

Initialisierung der Parameter 169
instruction (engl.) 9
Internationale Standardbuchnummer (ISBN) 134
Iteration 221
–, grafische 221

K
Kanäle 10
Kaprekar-Zahl 224
Kellerspeicher 204
Kerninformatik 7
Kilobyte 9
Klasse von Aufgaben 43
Kleincomputer 10
Kommando 14
Kommando-Modus 14
Kommunikation 7
Komponente 181
Konstruktionsprinzip
für Struktogramme 147
konvergent 220
Konvergenzbedingung 222
Körper, unmögliche 214
Korrektheit des Lösungsweges
Korrektur 31
Kreisbögen 214
Kreisdiagramm 152
Kursor (Cursor) 141
Kursor-Tasten 141

L
Lagerhaltungsdatei 195
Laufanweisung 76
Laufvariable 38, 76
Laufzeitorganisation 13
Leerzeichen 16
Leerzeichenkette 198
Leitliniendarstellung 147
length (engl.) 38
Lern-Modus 18
Lese-/Schreibkopf 12
Lese-/Schreibspeicher 10
LIFO-Prinzip 272
linksbündig 100
Linienzug 152
Lochstreifenleser 11
LOGO 278
Lohnrechnungsproblem 160

M
Magnetbandkassette 12
Megabyte 9
Melodien 108
Menü 106
Menütechnik 106
Meßwertauswertung 252
Meßwerterfassung 252
middle (engl.) 63
Minimum-Sortieren 182
Modell
–, deterministisches 241
–, mathematisches 229
Modifikationsmenü 138
modifizieren 89
Modul 164
Modul (Digital In/Out) 253
Modulprogrammierung 256
Monte-Carlo-Methode 238

N
Nachricht 7
Näherungsverfahren, numerische 215
Neunumerierung 34
Notation
–, umgekehrte polnische 277
–, verbal formalisierte 46
Numerik 215

O
Operation 58
–, arithmetische 58
Operator 58
–, arithmetischer 74
–, logischer 58
–, Vergleichs- 58
Ordnungszahl 128

P
PAGE-Parkettgestaltung 14
Palindrom 100
Parameterübergabe 126
PASCAL 275
Peripherigeräte 10
Personalcomputer 10
Pictogramme 6
PIO 254
PIO-Port 254
Pixel 88
Plotter 11
Population 106

Positionssystem, dezimales 214
Printliste 56
Priorität 58
Problemlösen 42, 147
Programm 9, 150
–, anwendungsspezifisches 13
Programmablaufplan (PAP) 46
Programmbibliothek 130
Programmgliederung, formale 148
Programmiersprache 34, 274
–, applikative 278
–, funktionale 278
–, imperative (befehlsorientierte) 276
–, interaktive 278
–, logische 279
–, prädikative 279
Programmierung 42
–, strukturierte 42
Programmkatalog 194
Programmkopf 35
Programmliniendarstellung 46
Programmlisting 262
Programmname 35
Programm-Modus 14
Programm-Rumpf 35
Programmsysteme 272
Programmzeile 27
Programmzeilenlänge 27
Projektarbeit 264
Projektdokumentation 262
PROLOG 279
Prozeduren 197
Prozedurfeld 200
Prozedurnamenliste 200
Prozessor 10
Prüfbit 8
Pseudozufallszahl 39, 113
Punktraster 70

Q
Quadrat, magisches 129
Quasigrafiksymbole 120
Quelltext 275

R
RAM 10
Räuber-Beute-Simulation 247
Rechenwerk 10
Recherchieren 190
Rechnergeneration 158
Rechteckregel 93, 235
rechtsbündig 100
record (engl.) 163
regula falsi 235
Reihe, harmonische 216
Reihenfolge, lexikografische 195
Rekursionsfolge 219
rekursiv 203
Rekursivität 204
remark (engl.) 34
Repeat-Schleife 76
Roboter 22
ROM 10
Rundungseffekte 96
Rundungsfehler 230, 233

S
Schachtelungstiefe 37
Schleife 102
–, zählergesteuerte 76
Schleifenendwert 76
Schleifenkörper 37
Schlüsselbegriff 186
Schlüsselwort 30
Schnecke, pascalsche 214
Schreibtischprotokoll 204
Schrittweite 37, 76
Semantik 8
Sender 7
Sequenz 51
Simulation 153, 242, 246
–, deterministische 246
–, probabilistische 242
Sinnbilddarstellung 46
Sinus-Reihe 218
Software 9, 12
Software-Werkzeuge 272
Solange-Schleife 76, 101
Sonderzeichen 8
Sortieren 181
Sortierstrategie 181
Speichergeräte, externe 11
Speichermodul, steckbarer 13
Sprachausgabegerät 11
Sprache 8
–, formale 8
Spur 12
square root (engl.) 26
Stack 204
Staffelbild 152
Standardfunktion 26
–, Zeichenketten 38

Startwert 232
statement (engl.) 9
Steuerbus 10
Steuerstruktur 57
Steuerwerk 10
Störfaktoren 247
Streckendiagramm 152
Streifendiagramm 237
Strichmarkierungsleser 11
String (engl.) 36
Struktogramm 46
Suchbegriff 191
Suchen 190
–, lineares 191
Syntax 8
Syntax-Fehler 30
System, hochkomplexes dynamisches 219
Systemsoftware 12
Systemunterstützung 13

T
Tabellenkalkulationssysteme 274
Tabellenkopf 28
Tastatur 11
Teach-in-Prinzip 23
Teilalgorithmus 113, 130
Teilkette 63
Telefonverzeichnis 194
Testdatei 175
Testprogramm 183
Testprotokoll 261
Testrahmenprogramm 183
Testwerte 95
Textanalyse 197
Textgerüst 135
Textverarbeitung 162
Textverarbeitungssysteme 162
Titelbild 148
Tonwiedergabe 109
Tool (engl.) 272
top-down-Verfahren 44
Trapezregel 235
TURBO-PASCAL 275
TURBO-PROLOG 279
Typkonzept 275

U
Übersetzungsprogramme 13
Überprüfen der Problemstellung 43
Unteralgorithmus 121, 147
Unterprogramm 132
UPN – umgekehrte polnische Notation 277

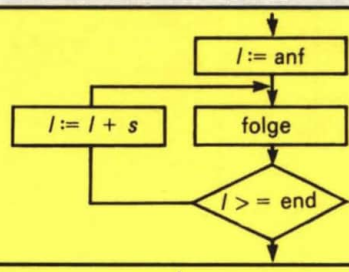
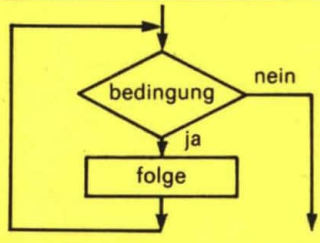
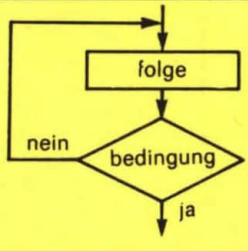
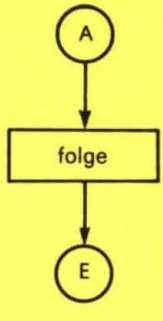
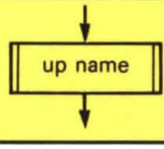
V
value (engl.) 66
Variable 33
–, lokale 130
–, numerische 72
–, Zeichenketten 40
Variablenname 33
–, zulässiger 33
Variablenliste 149
Verallgemeinerung, schrittweise 147
Verarbeiten einer Datei 181
Verarbeitungseinheit, zentrale 14
Verarbeitungsgeschwindigkeit 10
Verarbeitungsvorschrift 9
Verbund 164
Vereinbarung 9
– von Unteralgorithmen 121
– von benutzerdefinierten Funktionen 126
Vereinbarungsteil 34
Verfahren, Heron'sches 232
Verfahrensfehler 228
Verfeinerung, schrittweise 44, 147
Vergleichsoperator 58
Verkettungsoperator 59
Verschachtelung 204
Verwalten einer Datei 165
Verweisliste 287
Vorteiler 108

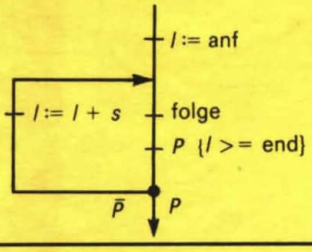
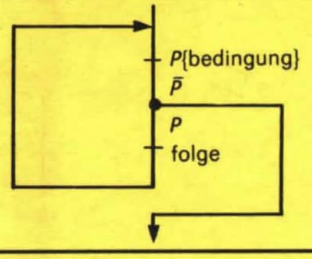
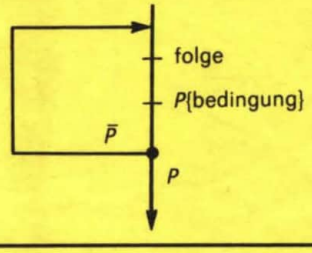
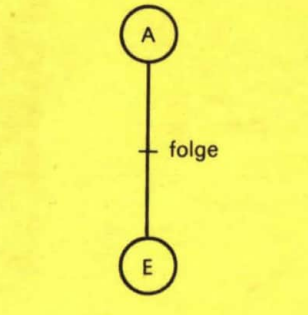
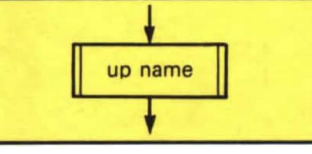
W
Wachstumssimulation 247
Wahrscheinlichkeit 242
Warteschlange 243
Wertzuweisung 32
While-Schleife 76
Wiederholschleife 76, 101
Wiederholung 76
–, mit Abbruchbedingung 101
–, mit Eingangsbedingung 102
Wissensbasis 43
Wortspiegelung 100

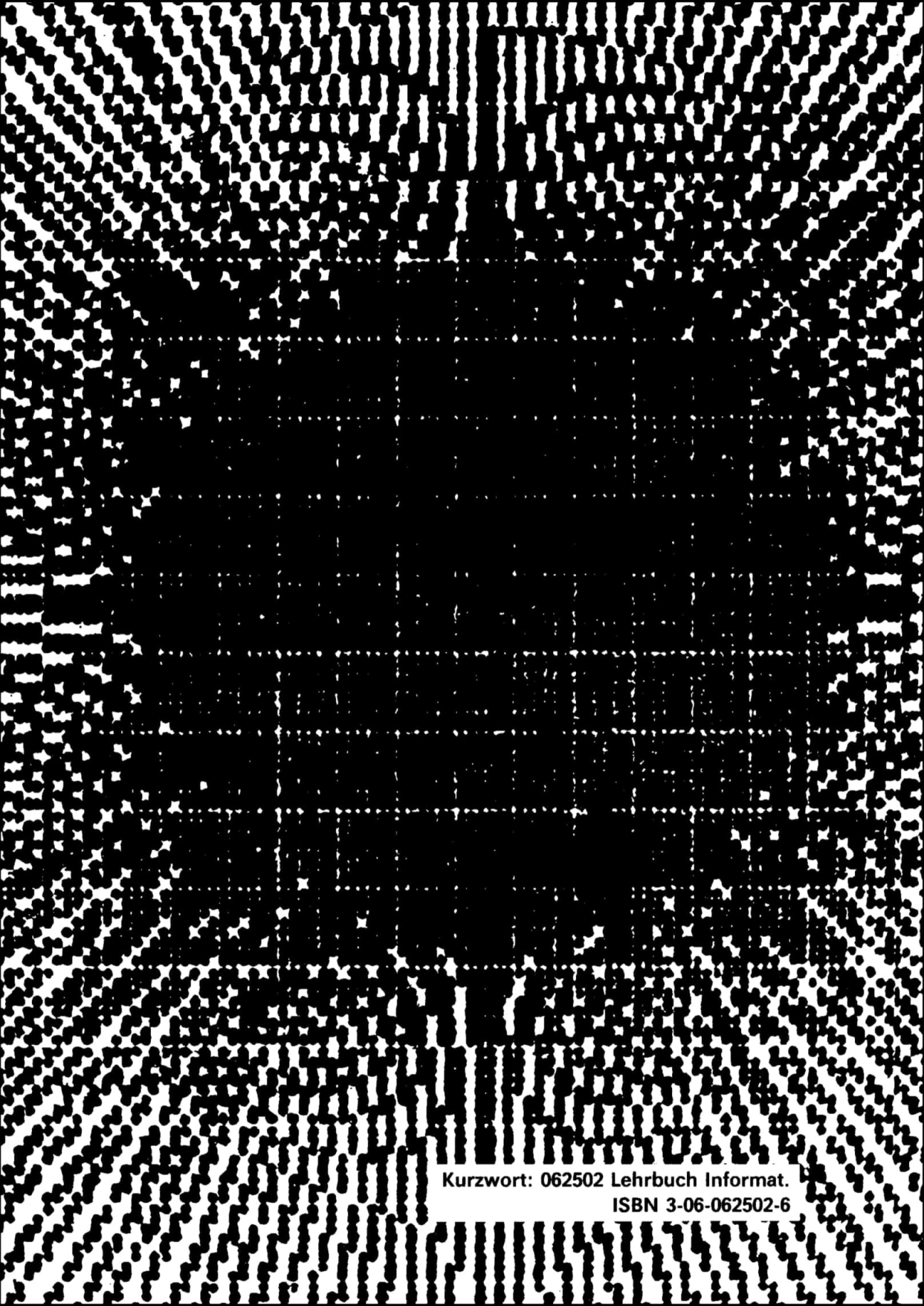
Z
Zählschleife 76
Zeichenbildtabelle 89
Zeichenbildtabellen-Zeiger 90
Zeichenentwürfe 83
Zeichenkette 36

Zeichenkettenfeld 128
Zeichenkettenkonstante 40
Zeichenkettenoperationen 54
Zeichenkettenspeicherbereich 54
Zeichenkettenstandardfunktion 40
Zeichenkettenvariable 40
Zeichenraster 70, 106
Zeilenanalyse 199
Zeilennummer 28
Zeitmessung 253
Zeitzählbaustein (CTC) 254
Zentraleinheit 10
zentriert 99
Zerlegungsbaum 44
Ziffern 8
Zufallsgrafik 120
Zufallszahlenerzeugung 116
Zufallszahlengenerator 249

Strukturelemente der Algorithmierung in verschiedenen Darstellungsformen

	verbal formalisierte Notation	Sinnbilddarstellung
3. Wiederholung (Repetition) 3.1. Zähl-schleife	FÜR *I* = anf BIS end SCHRITT *s* folge NÄCHSTES *I*	
3.2. Solange-schleife	SOLANGE bedingung TUE folge	
3.3. Wieder-hol-schleife	WIEDERHOLE folge BIS bedingung	
4. Unteralgorithmus (Abstraktion) 4.1. Unter-algorithmus-vereinbarung	UA unteralgorithmus-name vereinbarungsteil BEGINN folge ENDE	
4.2. Unter-algorithmus-aufruf	RUFE unteralgo-rithmusname	

Leitlinien-darstellung	Struktogramme	BASIC der Kleincomputer
	FÜR I = anf BIS end SCHRITT s folge	ak FOR I = anf TO end STEP s al folge am NEXT I
	SOLANGE bedingung TUE folge	ak REM anfang der solangeschleife al IF NOT (bedingung) THEN ap am folge an GOTO al ap REM ende der solangeschleife
	WIEDERHOLE folge BIS bedingung	ak REM anfang der wiederholschleife al folge an IF NOT (bedingung) THEN ak am REM ende der wiederholschleife
	(UA unteralgorithmusname) (vereinbarungsteil) BEGINN folge ENDE	ak REM up name al REM bemerkungen zu eingangs-, ausgangs- und ‚lokale' variablen am folge an RETURN
	RUFE unteralgorithmusname	ap GOSUB ak : REM →up name



Kurzwort: 062502 Lehrbuch Informat.
ISBN 3-06-062502-6