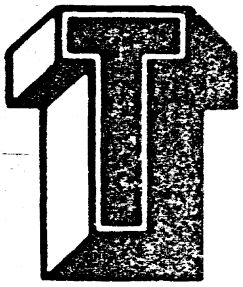
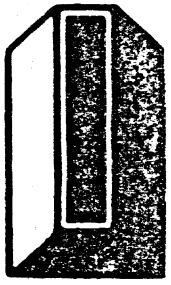
Kleinstrechner



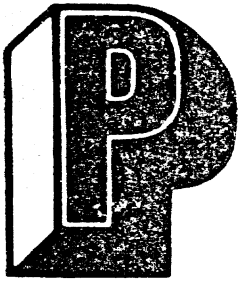
Tendenzen
und Theorien

FORTH



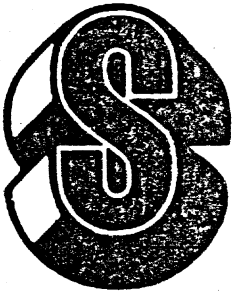
Informationen
und Ideen

Rechnerkopplung



Programme
und Projekte

Polynomgrafik



Spaß
und Spiel

Kleinstrechner-TIPS

Heft 10

Mit 38 Bildern

Herausgegeben von
Prof. Dr.-Ing. Hans Kreul
Prof. Dr. sc. techn. Thomas Horn
Doz. Dr.-Ing. Wilhelm Leupold

VEB Fachbuchverlag Leipzig

Inhalt

ISBN 3-343-00480-4

Zum Inhalt

Das vorliegende Heft 10 der Broschürenreihe „Kleinstrechner-TIPS" knüpft an das, wie zahlreiche Leserzuschriften zeigen, gute Resonanz gefundene Profil der letzten Hefte an und enthält eine, wie wir hoffen, gut ausgewogene Zusammenstellung von Beiträgen aus unterschiedlichen Gebieten der Anwendung der Kleincomputer.
Aus diesem Grunde dürfte es wohl für Schüler, Arbeitsgemeinschaften und Lehrer als auch für jeden KC-Anwender und an der Informatik interessierte Leser von Interesse sein.
Zu Beginn wird von KÜHNEL die Programmiersprache FORTH in einem thematisch orientierten Beitrag vorgestellt, der sich vor allem an den fortgeschrittenen Leser wendet.
Der folgende Beitrag von HÄNSEL und WAGENKNECHT befaßt sich mit der Anwendung des binären oder logarithmischen Suchverfahrens zum Suchen von Informationen in großen Datenbeständen. Die Implementierung der wichtigsten Routinen zur Anwendung dieses Verfahrens wird in PASCAL gezeigt.
Von LIPP und VIEWEG wird ein Programm zur Nullstellenbestimmung nichtlinearer Funktionen mit einer Variablen in BASIC vorgestellt. Dieser Aufsatz entstand im Rahmen einer Jahresarbeit in einer 12. Klasse. Es werden vier verschiedene Verfahren aus der numerischen Mathematik diskutiert und angewandt.
In der Arbeit von MAGERL und BORCHARDT wird das Ausbildungssystem TESY auf der Basis einer Rechnerkopplung zwischen Kleincomputern vorgestellt, das an der Militärtechnischen Schule der Luftstreitkräfte/Luftverteidigung in Bad Düben für die Informatikausbildung entwickelt wurde.

GIRLICH stellt ein BASIC-Programm zur grafischen Darstellung des Funktionsverlaufes von Polynomgleichungen auf dem Kleincomputer KC 85/3 einschließlich der Berechnung ihrer Nullstellen vor. Das Programm entstand, als er noch Schüler der 11. Klasse war und mit Kurvendiskussionen „geplagt" wurde.
Im abschließenden Teil werden von GÖRGENS einige Spielprogramme – von der Idee bis zum fertigen Spiel – vorgestellt („Siebzehn und Vier", „Rotation" und „Supermagische Quadrate"), die unterschiedliche Algorithmen, wie Pseudo-Zufallszahlengeneratoren, Indexrechnungen und Kombinationsalgorithmen, anwenden.
Schließlich hoffen wir auch bei diesem Heft auf eine gute Resonanz bei unseren Lesern und wünschen ihnen viele neue Ideen bei der Beschäftigung mit dem Kleincomputer.

Thomas Horn

FORTH – ein Softwarekonzept für Mikro- und Minicomputer

0. Einführung

FORTH geht in seinen Ursprüngen auf Arbeiten des Amerikaners CHARLES H. MOORE aus dem Jahre 1969 zurück. Ziel der Entwicklung war die Schaffung einer Programmierumgebung, die eine effektive anwendungsspezifische Programmierung von Echtzeitsystemen unterstützt. Herkömmliche Sprachkonzepte erwiesen sich für die genannte Problemstellung als zu starr.

FORTH weist in hohem Maße Betriebssystemeigenschaften auf und eignet sich wegen der Kompaktheit des erzeugten Codes, der maschinennahen Programmierung sowie der Laufzeiteigenschaften als Systemsprache und für Echtzeitanwendungen [1].

FORTH bietet in einer einfachen Struktur die Fähigkeiten einer interaktiven Programmiersprache, eines Macro-Assemblers und eines Betriebssystems mit Unterstützung durch Hilfsprogramme. Die folgenden Eigenschaften kennzeichnen FORTH weitgehend [2]:

- maschinennahe, iteraktive Hochsprache
- Umgekehrt Polnische Notation (UPN)
- Pusl-down-Pull-up-Datenstack
- strenge Strukturiertheit (weder GOTOs noch LABELs)
- Portabilität infolge Standardisierung gesichert (FORTH-83)
- gute Laufzeiteigenschaften (etwa zehnmal so schnell wie BASIC, vergleichbar mit Assembler)
- Erweiterbarkeit von Syntax und Semantik
- modulares Programmieren (Top-down Planen, Bottom-up Programmieren)
- schrittweises Testen der Moduln möglich
- FORTH und Assembler beliebig mischbar, daher ideal für Echtzeitanwendungen
- 16-bit- und 32-bit-Integerarithmetik
- Kompaktheit (FORTH-System, Assembler Editor und Massenspeicher-Interface meist innerhalb von 8 kByte)

Aufgrund der genannten Eigenschaften ist eine große Domäne von FORTH bei Einplatinencomputern, Einchipmikrorechnern und anderen kleinen, von größeren Computern abgesetzten Computersystemen zu suchen. Für diese Einrichtungen, die in sehr großen Stückzahlen für Aufgaben der Steuerungstechnik Verwendung finden, stehen nur in beschränktem Maße höhere Programmiersprachen unterstützende Compiler zur Verfügung. Oft dominiert an diesen Stellen auch heute noch die wenig effektive Assemblerprogrammierung.

Der Einsatz von FORTH ist aber bei weitem nicht auf die angegebene Klasse oft spartanisch ausgerüsteter Computer beschränkt. Zur Implementierung von Spezialsprachen, zur Arbeit mit umfangreichen Datenbanken und KI-Projekten werden auch komfortablere Computer herangezogen.

FORTH wird u.a. für die Mikroprozessoren 8080, Z 80, 8086, 68 000 sowie die Computer Apple II, Atari 520 ST, Commodore Amiga, IBM-PC, Mac Intosh und VAX angeboten. Verschiedene Versionen benutzen das jeweils vorhandene Betriebssystem. Dabei werden CP/M-80, CP/M-86, MS-DOS, CP/M-68k und auch UNIX benutzt. Für den Einsatz von FORTH sind international damit alle Voraussetzungen gegeben.
In der DDR beschäftigen sich einige Institutionen der Akademie der Wissenschaften und des Hochschulwesens mit der Entwicklung und Anwendung von FORTH. Bislang wurden im allgemeinen zum F.I.G.-FORTH der FORTH Interrested Group kompatible Systeme, wie comFORTH und pop-FORTH der Wilhelm-Pieck-Universität Rostock für Mikrocomputer unter dem Betriebssystem SCP, FORTH-System „Fichte" für die konfigurierbare Datenstation K 8915 von robotron Zella-Mehlis, KC-FORTH für die Kleincomputer KC 85/2 und KC 85/3 aus Mühlhausen sowie eine FORTH-Version für den Mikroprozessor U 8000, eingesetzt. Mit dem Einsatz der 16-Bit-Technik vollzieht sich gegenwärtig der Übergang zu dem 1983 vom FORTH-Standards-Team festgeschriebenen FORTH-83. Beide Versionen werden in naher Zukunft nebeneinander existieren, wobei aufgrund der möglichen Erweiterbarkeit die Portabilität (zumindest) prinzipiell als gesichert gelten kann.
FORTH ist weiterhin sehr gut für die Implementierung auf einem Chip geeignet. Der markanteste Schritt in Richtung FORTH-Prozessor war die Entwicklung des NC 4000 durch die Fa. Novix (USA) [3].

1. Der Aufbau von FORTH

Bevor die Programmierung mit FORTH betrachtet werden soll, erfolgt ein Blick hinter die Kulissen.
Ein Wort in FORTH ist das Äquivalent zu einer Subroutine in BASIC oder einer Procedure in PASCAL, um bei den verbreitetsten höheren Programmiersprachen für Mikrocomputer zu bleiben. Das FORTH-Wort kann als „black box" betrachtet werden. Es ist gleichgültig, was intern passiert. Eine klare Definition der Schnittstellen erlaubt auch die Nutzung dieses Wortes in einem übergeordneten. Eine so vorgenommene Erweiterung der Sprache bewirkt, daß das gesamte Programm seinerseits wiederum aus nur einem, andere Worte aufrufenden FORTH-Wort besteht.
Alle Worteintragungen werden im sog. Dictionary (Wörterbuch), einem dafür speziell reservierten Speicherbereich vorgenommen. Zu unterscheiden sind dabei sog. Primitive-Worte, die direkt im Maschinencode des betreffenden Mikroprozessors programmiert wurden, und die sog. Secondaries, die ihrerseits Aufruffolgen weiterer Moduln darstellen.
Ein Beispiel soll den Sachverhalt verdeutlichen. Es soll ein FORTH-Wort CLS (Clear Screen) geschaffen werden, welches das Löschen des gesamten Bildschirmes ermöglicht. Hierzu wird die später noch näher betrachtete Colon-Definition herangezogen.

```
: CLS 0C EMIT;
```

Der auf diese Weise neu definierte Befehl CLS bewirkt durch den Aufruf des Wortes EMIT die Ausgabe des ASCII-Zeichens FF (Form Feed) entsprechend der Codierung $0C_H$ an den Bildschirm. Durch die Wortfolge

```
'CLS. ENTER 20BF OK
```

(**ENTER** = Drücken der Enter-Taste)

gibt der Computer die Stelle im Speicher aus, an der das neue FORTH-Wort CLS zu finden ist. Der Speicherinhalt läßt sich durch das Wort DUMP untersuchen. Will man, beginnend mit der Adresse 20B7, sechzehn Adressen untersuchen, kann das mit Hilfe der Folge

20B7 10 DUMP ENTER

ausgelöst werden. Der Speicherinhalt wird dann in Form eines sog. HEX-DUMPs auf dem Bildschirm ausgegeben. An Hand der folgenden Bildschirmausgabe soll der Aufbau des neuen Wortes CLS erläutert werden.

```
20B7  83  43 4C D3 AC 20 75  07
                                .CLS, U.
20BF  D1  02 0C 00  52  04 B5 05
                               Q...R. 5.
```

Die Adresse 20B7 ist die sog. Namensfeldadresse NFA (name field address), deren Aufbau in Bild 1 gezeigt ist. Es ist ablesbar, daß das folgende Namensfeld 3 Zeichen (CLS) lang sein wird. Es folgt der Namenseintrag als ASCII-Zeichen codiert, wobei das MSB (höchstwertige Bit) des letzten Zeichens gesetzt wird (+ 80_H). An den Namenseintrag folgt das sog. Linkfeld. Im Linkfeld ist die Adresse 20AC eingetragen. Diese Adresse weist auf das nächste Wort im Dictionary. Über diese Linkfelder kann das gesamte Dictionary bei der Suche nach einem Wort durchgemustert werden. An das Linkfeld schließt sich das sog. Codefeld an. Der Inhalt des Codefeldes, die Code-Adresse (CA), definiert den Typ des Wortes (Primitive, Secondary, Constant, Variable u.a.) In unserem Fall liegt ein Secondary vor, weshalb die hier stehende Adresse (0775_H) auf die COLON-Definition (":" (DOCOL)) zeigt. Das Codefeld wird vom Parameterfeld gefolgt, welches durch die Wortadresse SEMIS, dem Abschluß der COLON-Definition, abgeschlossen wird. Die Adresse 02D1 weist auf das Primitive-Wort LIT, welches den Inhalt der darauffolgenden Speicherzelle als compilierte Konstante behandelt. Diese Konstante 000C steht damit für das über die Adresse 0452 erreichbare Wort EMIT bereit. Die Adresse 05B5 schließlich beendet die Abarbeitung durch den Aufruf von SEMIS.

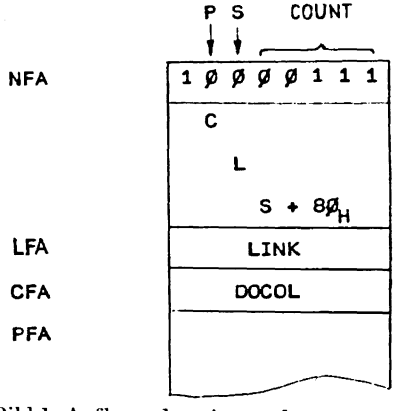


Bild 1. Aufbau des Anwenderwortes CLS intern

Zusammenfassend kann damit ein Dictionary-Eintrag durch den sog. Header, das ist das Namensfeld einschließlich des Führungsbytes, die Linkadresse zum vorangehenden Dictionary-Eintrag sowie das Codefeld mit dem sich anschließenden Parameterfeld, gekennzeichnet werden.

Die Übergabe der Daten erfolgt unter FORTH über den Parameterstack, einen Stapelspeicher nach dem LIFO-Prinzip (last in, first out). Ein weiterer Stack, der Returnstack, dient weitgehend Systemzwecken. Für die Zwischenspeicherung kann der Returnstack mit einer gewissen Vorsicht verwendet werden. Im Zusammenhang mit der Datenübergabe muß die gewöhnungsbedürftige Besonderheit von FORTH, die in der Umgekehrt Polnischen Notation besteht, erwähnt werden. Der Vorteil dieser Notation

liegt darin, daß komplexe Rechnungen klammerfrei vorgenommen werden können. Den Datenbewegungen auf dem Stack kommt beim Programmieren in FORTH ein hoher Stellenwert zu.
FORTH arbeitet im allgemeinen mit mindestens einem Diskettenspeicher. Modifikationen für Kassettenbetrieb wurden z.B. im FORTH-Modul M 026 für die Kleincomputer KC 85/2 und KC 85/3 vorgesehen. Für den Massenspeicher ist ein Buffer vorgesehen, der durch die Zeiger FIRST und LIMIT eingegrenzt wird. Die Speicheraufteilung ist installationsabhängig und kann mit dem kleinen Programm nach Bild 2 untersucht werden. Eine mögliche Aufteilung ist in Bild 2 enthalten.
Für einen ersten Einblick in die FORTH-Interna sollen die vorliegenden Bemerkungen genügen. An dieser Stelle soll auf die ausführliche Betrachtung dieses Problemkreises in [1] hingewiesen werden.

```
SUR # 256
  Ø ( ====== MEMORY - MAP ====== )
  1 HEX
  2 : .HEX Ø SWAP D.R ;
  3 : CLS ØC EMIT ;
  4 : LOOK DUP 7 .HEX @ 8 .HEX CR ;
  5 : MEM_MAP CLS
  6  ." *** MEMORY -- MAP ***" CR CR
  7  ."           ADRESSE INHALT" CR
  8  ." DP      " DP LOOK
  9  ." SØ      " SØ LOOK
 10  ." RØ      " RØ LOOK
 11  ." TIB     " TIB LOOK
 12  ." FIRST   " FIRST LOOK
 13  ." LIMIT   " LIMIT LOOK
 14  ." ORIGIN  " 7 SPACES
 15  Ø +ORIGIN 8 .HEX CR ;

*** MEMORY -- MAP ***

          ADRESSE INHALT
DP          212    248B
SØ          2Ø5     EØ
RØ          2Ø8    17E
TIB         2ØA     EØ
FIRST       216    2FFØ
LIMIT       218    4ØØØ
ORIGIN              26Ø
OK
```

Bild 2. Speichereinteilung (Listing und Kontrollausdruck)

2. Die Programmierung in FORTH

Nach einem ersten Einblick in die Interna von FORTH, der dem prinzipiellen Verständnis dienen sollte, beschäftigt sich dieser Abschnitt mit der praktischen Handhabung von FORTH.

FORTH-Arithmetik und Stackoperationen

Die zentrale Stelle zur Datenübernahme und -übergabe ist in FORTH der Datenstack. Die Operanden einer Operation werden auf dem Stack erwartet. Das Ergebnis der Operation wird auch auf dem Stack abgelegt. Der Sinn der Umgekehrt Polnischen Notation wird dadurch klar. Die Operanden gehen im allgemeinen verloren. Eine Eingabe in FORTH sieht dann folgendermaßen aus:

1 2 ENTER OK

Grundsätzlich gilt, daß Operanden und Worte in FORTH durch jeweils ein Leerzeichen (SPACE) voneinander getrennt werden. Wie gewohnt, wird eine Eingabe durch das Betätigen der ENTER-Taste (auch mitunter als CR, RETURN oder NEW LINE bezeichnet), das im folgenden durch **ENTER** gekennzeichnet wird, abgeschlossen. Durch die Eingabe der Operanden sind die beiden oberen Stackzellen (TOS – Top of Stack und SECOND) belegt. Auf diese beiden Operanden läßt sich z.B. eine Additionsoperation anwenden (+) und durch . (sprich: „dot") das Ergebnis auf dem Bildschirm darstellen.

+ . ENTER 3 OK

Ein Blick in die Zusammenstellung der häufigsten FORTH-Worte (siehe Tabelle 1, S. 8) zeigt, daß die Addition beide Operanden auf dem Stack erwartet, diese anschließend entfernt und nur die Summe auf dem TOS zurückläßt.

Tabelle 1. Die häufigsten FORTH-Worte

FORTH-Wort	Stackbewegungen vor der Operation	nach der Operation	Erläuterungen
Zahlensysteme:			
DECIMAL		—	deklariert Dezimalsystem
HEX		—	deklariert Hexadezimalsystem
BASE		— addr	übergibt die Adresse der Systemvariablen BASE, die die Zahlenbasis enthält
Stackmanipulationen:			
DUP	n	— n n	dupliziert den TOS
— DUP	n	— n ?	dupliziert den TOS, wenn ungleich Null
DROP	n	—	beseitigt den TOS
SWAP	n1 n2	— n2 n1	vertauscht die beiden oberen Einträge
OVER	n1 n2	— n1 n2 n1	kopiert SECOND nach TOS
ROT	n1 n2 n3	— n2 n3 n1	rotiert THIRD zum TOS
> R	n	—	bringt den TOS zum Return-Stack
R >		— n	holt Wert vom Return-Stack zum TOS
R		— n	kopiert den Return-Stack-Top zum TOS
Arithmetik:			
+	n1 n2	— n3	Addition (n3 = n2+n1)
D+	d1 d2	— d3	Addition (d3 = d2+d1)
—	n1 n2	— n3	Subtraktion (n3 = n1—n2)
*	n1 n2	— n3	Multiplikation (n3 = n1 *n2)
/	n1 n2	— n3	Division (n3 = n1/n2)
MOD	n1 n2	— Rest	Modulo-Division
/MOD	n1 n2	— Rest Quot.	Division mit Rest und Quotient
*/MOD	n1 n2 n3	— Rest Quot.	Multiplikation mit anschließender Division bei 32-bit-Zwischenergebnis
*/	n1 n2 n3	— Quot.	wie bei */MOD, aber ohne Rest
M/MOD	u d1 u2	— u3 ud4	Division einer vorzeichenlosen 32-bit-Zahl mit Übergabe des 16-bit-Restes und des 32-bit-Quotienten
MIN	n1 n2	— n3	kleinere Zahl nach TOS (n3 = min (n1, n2))
MAX	n1 n2	— n3	größere Zahl nach TOS (n3 = max (n1, n2))
ABS	n	— u	Absolutwert (16 bit)
DABS	d	— ud	Absolutwert (32 bit)
MINUS	n	— —n	Vorzeichenwechsel (16 bit)
DMINUS	d	— —d	Vorzeichenwechsel (32 bit)
1+	n	— n+1	inkrementiert TOS mit Eins
2+	n	— n+2	inkrementiert TOS mit Zwei

FORTH-Wort	Stackbewegungen vor der Operation	nach der Operation	Erläuterungen
Logische Befehle:			
AND	n1 n2	— n3	bitweises logisches AND (n3 = n1 · n2)
OR	n1 n2	— n3	bitweises logisches OR (n3 = n1 + n2)
XOR	n1 n2	— n3	bitweises logisches Exklusiv-OR (n3 = n1 ⊕ n2)
Vergleichsoperatoren:			
<	n1 n2	— f	Flag = true, wenn n1 kleiner n2
>	n1 n2	— f	Flag = true, wenn n1 größer n2
=	n1 n2	— f	Flag = true, wenn n1 gleich n2
Ø<	n1 n2	— f	Flag = true, wenn TOS negativ
Ø=	n	— f	Flag = true, wenn TOS gleich Null
Speicherbezogene Worte:			
@	addr	— n	ersetzt Adresse durch deren Inhalt
C @	addr	— b	wie @, jedoch nur auf ein Byte bezogen
!	n addr	—	speichere SECOND in die durch TOS vorgegebene Adresse
C !	b addr	—	wie !, jedoch nur ein Byte abspeichern
+ !	n addr	—	Erhöhung des durch TOS vorgegebenen Speicherinhaltes mit SECOND
CMOVE	from to u	—	verschiebe u Bytes von from nach to
FILL	addr u b	—	fülle u Bytes ab addr mit b
ERASE	addr u	—	fülle u Bytes ab addr mit Ø
BLANKS	addr u	—	fülle u Bytes ab addr mit SPACE (Leerz.)
TOGGLE	addr b	—	EXOR Byte in addr mit Maske b
SP @		— addr	übergibt Position des Stackpointers
strukturierte Worte:			
DO . . .LOOP	n1 n2	—	Schleife, Index läuft von n2 bis n1—1 mit Inkrement 1
DO . . .+LOOP	n1 n2	—	wie DO . . . LOOP, jedoch mit beliebigem Inkrement
I		— Index	Schleifenindex zum TOS
LEAVE		—	erzwingt Abbruch der Schleife bei nächster Gelegenheit
IF . . (wahr) . . ENDIF	f	—	wenn Flag = true, Ausführung des (wahr)-Zweiges
IF . . (wahr) . . ELSE . . . . (falsch) . . ENDIF	f	—	dto., jedoch bei Flag = false Ausführung des (falsch)-Zweiges
BEGIN . . . UNTIL		—	Schleife mit Abbruch, falls Flag für UNTIL gleich true
BEGIN . . WHILE . . . . REPEAT		—	wie BEGIN . . . UNTIL, Test auf Abbruch durch Flag für WHILE am Schleifenanfang
BEGIN . . . AGAIN		—	Endlos-Schleife

FORTH-Wort	Stackbewegungen vor der Operation	nach der Operation	Erläuterungen
Ein-/Ausgabe:			
.	n	—	druckt TOS aus (zerstörend!)
.R	n Feldweite	—	druckt n rechtsbündig in Feldweite
D.	d	—	druckt 32-bit-Zahl
D . R	d Feldweite	—	wie . R, nur 32-bit-Zahl
CR		—	Ausgabe CR/LF
SPACE		—	Ausgabe eines Leerzeichens
SPACES	n	—	Ausgabe von n Leerzeichen
TYPE	addr u	—	druckt u Zeichen ab addr
COUNT	addr	— addr+1 u	wandelt Zeichenkette in TYPE-Format um
?	addr	—	druckt den Inhalt von addr
? TERMINAL		— f	Flag=true, wenn Tastatur betätigt wurde
KEY		— c	wartet auf Tastaturbetätigung und legt den entsprechenden Code auf den TOS
EXPECT	addr n	—	erwartet n Zeichen und schafft diese nach addr
EMIT	c	—	gibt ASCII-Zeichen c aus
WORD	c	—	liest ein Wort bis zum Begrenzer c aus dem Eingabepuffer
Ein-/Ausgabe-Formatierung:			
NUMBER	addr	— d	wandelt die Zeichenkette von addr um in eine 32-bit-Zahl
< #		—	eröffnet Zahlenumwandlung
#	d	— d	wandelt nächste Stelle der Zahl und fügt der auszugebenden Zeichenkette ein Zeichen an
# S	d	— 00	wandelt alle signifikanten Stellen einer Zahl in eine Zeichenkette um
SIGN	n d	— d	fügt das Vorzeichen von n in Zeichenkette
# >	d	— addr u	beendet Umwandlung in Zeichenkette
HOLD	c	—	fügt das Zeichen c in die Zeichenkette
Massenspeicher (Disk):			
LIST	scr	—	Ausdrucken des Screens scr von Diskette
LOAD	scr	—	Laden des Screens scr von der Diskette
BLOCK	block	— addr	liest den Block nach addr
B/BUF		— n	Blockgröße in Bytes (Systemkonstante)
BLK		— addr	aktuelle Blocknummer (Systemvariable)
SCR		— addr	aktuelle Screennummer (Systemvariable)

FORTH-Wort	Stackbewegungen vor der Operation	Stackbewegungen nach der Operation	Erläuterungen
Massenspeicher (Disk) (Forts.)			
UPDATE	—		markiert letzten Buffer als „updated"
FLUSH	—		schreibt alle „updated"-Buffer auf Disk
EMPTY-BUFFERS	—		markiert alle Buffer als leer
Definitionsworte:			
: name	—		Beginn einer Colon-Definition mit dem Namen name
;	—		Abschluß der Colon-Definition
VARIABLE xxx	n	—	erzeugt eine Variable xxx, die mit dem Wert n initialisiert wird
CONSTANT yyy	n	—	erzeugt eine Konstante yyy mit dem Wert n
CODE zzz	—		erzeugt die Definition eines PRIMITIVEs mit dem Namen zzz
;CODE	—		Abschluß der Colon-Definition, wenn es sich um die Definition eines Definitionswortes handelt, bei der das Laufzeitverhalten in Assembler definiert werden soll
⟨BUILDS ... DOES⟩	does:	— addr	wird zur Definition von Definitionsworten benutzt, Laufzeitverhalten aber in High-Level
Vokabulare:			
CONTEXT	—	addr	übergibt die Adresse des Zeigers zum CONTEXT-Vokabular (Suchvokabular)
CURRENT	—	addr	übergibt die Adresse des Zeigers zum CURRENT-Vokabular (zu erweiterndes Vok.)
FORTH	—		Name des Hauptvokabulars (setzt CONTEXT)
EDITOR	—		Aufruf des Editors (setzt CONTEXT)
ASSEMBLER	—		Aufruf des Assemblers (setzt CONTEXT)
DEFINITIONS	—		macht das CONTEXT- zum CURRENT-Vokabular
VOCABULARY xyz	—		deklariert ein neues Vokabular mit dem Namen xyz
VLIST	—		druckt die Namen aller Worte im CONTEXT-Vokabular
System-Worte und Diverses:			
(Kommentar)	—		eröffnet einen Kommentar, der durch) abgeschlossen wird , hinter (muß stets ein Leerzeichen folgen !

FORTH-Wort	Stackbewegungen vor der Operation	Stackbewegungen nach der Operation	Erläuterungen
System-Worte und Diverses (Forts.)			
FORGET abc		—	läßt alle neuen Definitionen einschließlich abc vergessen
ABORT		—	erzwingt einen Fehler-Abbruch einer Operation
' name		— addr	findet die Adresse (PFA) des Wortes name im Dictionary
HERE		— addr	übergibt die Adresse des nächsten freien Platzes im Dictionary
PAD		— addr	übergibt die Adresse eines Zwischenspeichers, meist 68 Bytes oberhalb von HERE
IN		— addr	Input-Buffer-Offset für WORD (Systemvar.)
ALLOT	n	—	reserviert n Bytes im Dictionary
,	n	—	compiliert eine Zahl in Dictionary an die durch HERE gekennzeichnete Stelle

Legende:

Operanden:	n, n1	16-bit-Komplementärzahl
	d, d1	32-bit-Komplementärzahl
	u, u1	16-bit-Zahl ohne Vorzeichen
	ud	32-bit-Zahl ohne Vorzeichen
	addr	Adresse
	b	Byte (8 bit)
	c	ASCII-Zeichen (7 bit)
	f	Boolesches Flag (16 bit)

Stackbewegungen:
- Ein- und Ausgaben immer von links nach rechts
- TOS (Top-Of-Stack) steht stets rechts außen
- SECOND bezeichnet die Stelle unter dem TOS
- THIRD bezeichnet die Stelle unter SECOND (SECOND und THIRD sind keine FORTH-Worte !)
- Kennzeichnung der Stackbewegung durch (Stack vor Aufruf eines Wortes — — Stack nach Aufruf des Wortes)

Das Wort . bringt den Inhalt des TOS auf dem Bildschirm zur Anzeige und entfernt ihn vom Stack.
Immer dann, wenn Operationen fehlerfrei absolviert wurden, antwortet FORTH mit dem schon mehrfach angegebenen OK. Die vorstehenden Aussagen befähigen im Zusammenwirken mit Tabelle 1 zum Gebrauch des FORTH-Rechners als „Taschenrechner".

Die Berechnung des Ausdruckes

$$\frac{2+3}{5} + \frac{4 \cdot 7}{3+8} = 3{,}\overline{54}$$

nimmt in FORTH dann folgende Form an:

2 3 + 5 / 4 7 * 3 8 + / . **ENTER** 3 OK

Die Umgekehrt Polnische Notation und das Stackprinzip machen einen Verzicht auf die Kenntnis der diversen

Rechenregeln, wie Klammerregeln und Punkt-Strich-Rechnung, durch die Maschine möglich. Sicher bedarf diese Art der Notation beim Ungeübten einiger Gewöhnung.
Vergleicht man das korrekte Ergebnis mit dem des FORTH-Rechners, zeigt sich ein deutlicher Unterschied. Die Ursache für das ungenaue Ergebnis ist die Tatsache, daß FORTH nur eine Festkomma-Arithmetik kennt. Der Verzicht auf eine Gleitkomma-Arithmetik bewirkt schnelle arithmetische Operationen mit geringem Speicherplatzbedarf. Der Datenstack hat eine Breite von 16 bit, so daß FORTH mit 16- bzw. 32-bit-Festkommazahlen arbeitet. An dieser Stelle muß betont werden, daß der Rechner selbst nur Bitmuster manipuliert. Ob eine Zahl als vorzeichenlos[1]) oder als Komplementärzahl[2]) aufgefaßt wird, entscheidet das auf diese Zahl angewendete FORTH-Wort. Das bereits verwendete Wort . druckt eine vorzeichenbehaftete 16-bit-Zahl aus, d.h. daß bei gesetztem MSB negative Zahlen ausgegeben werden.

```
32766 . ENTER 32766 OK
32767 . ENTER 32767 OK
32768 . ENTER - 32768 OK
32769 . ENTER - 32767 OK
```

An Hand der binären Zahlendarstellung kann man sich diesen Sachverhalt sehr schnell und einfach verdeutlichen.

Der Gültigkeitsbereich der 16-bit-Zahlen erstreckt sich bei vorzeichenlosen Zahlen von 0 bis $2^{16}-1$ und bei Komplementärzahlen von -2^{15} bis $2^{15}-1$. Äquivalente Ergebnisse erhält man natürlich für die 32-bit-Zahlen. Bereichsüberschreitungen werden durch FORTH nicht gemeldet. Der Programmierer muß hier sehr sorgfältig vorgehen.
Das Fehlen der Gleitkomma-Arithmetik kann an vielen Stellen überwunden werden. Die Skalierung ist dafür ein probates Mittel. FORTH stellt für die Skalierung ein spezielles Wort (*/) zur Verfügung. Ein Blick in Tabelle 1 verrät, daß das Multiplikationsergebnis 32 Bitstellen aufweist. Nach der Division steht dann wieder ein 16-bit-Ergebnis zur Verfügung. Eine Skalierung kann unter Zuhilfenahme dieser Operation sehr einfach durchgeführt werden.

$$\frac{\text{Teilstrecke}}{\text{Gesamtstrecke}} = \frac{\text{neue Teilstrecke}}{\text{neue Gesamtstrecke}}$$

$$\frac{777}{1555} = \frac{??}{2700}$$

In FORTH wird dieses Problem durch folgende Eingabe gelöst:

```
2700 777 1555 */ . ENTER 1349 OK
```

Abgeleitet aus dieser Art Aufgabenstellungen wird das Wort */ auch als Skalierungsoperator bezeichnet. Auf die vorgestellte Weise kann auch mit rationalen Näherungen für irrationale Konstanten gearbeitet werden. Die Zahl π sei hier stellvertretend genannt. Mit dem Bruch 355/113 wird die Zahl π bis auf die 6. Stelle nach dem Komma genau wiedergegeben.
Die angeführten Beispiele benutzten Dezimalzahlen. Bei der Arbeit mit FORTH ist man aber nicht auf eine dezimale Zahlenbasis beschränkt. Die Worte DECIMAL und HEX stellen die betreffende Zahlenbasis ein. Prinzipiell kann aber mit jeder beliebigen Zahlenbasis gearbeitet werden. An dieser Stelle sollen einige neue FORTH-Worte betrachtet werden, die nach Erläuterung der Colon-Definition und der Variablen sicher verstanden werden. Das Ausdrucken der gerade benutzten Zahlenbasis erfolgt durch das Wort . BASE .

---

[1]) natürliche Zahl
[2]) ganze Zahl in Komplementdarstellung bei negativen Werten

```
: . BASE BASE @ DUP DECIMAL
. BASE ! ;
```

Durch BINARY läßt sich eine binäre und durch SBASE eine beliebige Zahlenbasis einstellen.

```
: BINARY 2 BASE ! ;
: SBASE BASE ! ;
```

Die folgenden Beispiele sollen den Gebrauch dieser Worte verdeutlichen:

```
DECIMAL . BASE 255 DUP
  . ENTER 10 255 OK
HEX .BASE DUP . ENTER 16 FF OK
BINARY .BASE DUP .
  ENTER 2 11111111 OK
DECIMAL 8 SBASE . BASE
. ENTER 8 377 OK
```

Zum Verständnis der Beispiele ist es vorteilhaft, die Stackbewegungen zu verfolgen. Das in Bild 3 gezeigte Programmbeispiel ermöglicht das Lesen des Datenstacks, ohne irgendwelche Veränderungen vorzunehmen. Die ent-

```
SCR # 100
  0 ( === ANZEIGE DATENSTACK === )
  1
  2 : DEPTH S0 @ SP@ - 2 / 1 - ;
  3
  4 : .S CR DEPTH
  5       IF SP@ 2 - S0 @ 2 -
  6         DO
  7           I @ . -2
  8         +LOOP
  9       ELSE ." STACK EMPTY !" CR
 10       ENDIF ;
 11
 12
 13
 14
 15

11 22 33 44 55 ENTER OK
.S ENTER
11 22 33 44 55 OK
SWAP .S ENTER
11 22 33 55 44 OK
DROP .S ENTER
11 22 33 55 OK
OVER .S ENTER
11 22 33 55 33 OK
DUP .S ENTER
11 22 33 55 33 33 OK

DEPTH . ENTER 6 OK
```

Bild 3. Zerstörungsfreies Lesen des Datenstacks (.S) und Beispiele zur Stackmanipulation

haltenen Beispiele verdeutlichen die Anwendung. Beim Aufstellen neuer FORTH-Worte, d.h. beim Programmieren im eigentlichen Sinne, sollte man ein solches Wort zur Hand haben.

Die Definition von Anwenderworten und der Gebrauch des Editors

Die Programmierung in FORTH bedeutet eine Definition von Anwenderworten, die damit das FORTH-Vokabular erweitern. Zur Definition von Anwenderworten soll hier nur die einfach zu realisierende Colon-Definition betrachtet werden. Die Colon-Definition wurde bereits an einigen Stellen bemüht, wodurch ihr prinzipielles Aussehen schon geläufig ist. Im ersten Abschnitt wurde ein einfaches Wort zum Löschen des Bildschirmes CLS definiert. Nach dem Einstellen der hexadezimalen Zahlenbasis kann man folgende Zeile eintippen:

```
: CLS 0C EMIT ; ENTER OK
```

Die Colon-Definition wird durch den Doppelpunkt (: – Colon) eingeleitet und durch das Semikolon (;) abgeschlossen. Dem einleitenden Doppelpunkt folgt der Name des neuen FORTH-Wortes. Es schließen sich die das neue Wort bildenden FORTH-Worte bzw. Zahlen an. Hier wird die Hexadezimalzahl 0C auf den Stack gebracht und durch EMIT das entsprechende ASCII-Zeichen ausgegeben. Die Eingabe der Neudefinition wird wiederum durch das Betätigen der ENTER-Taste abgeschlossen, worauf sich sofort der Compilierungsvorgang anschließt. Ist die vorgenommene Anwenderdefinition gültig, antwortet das FORTH-System mit dem bekannten OK. Listet man anschließend mit VLIST das Dictionary auf, steht an erster Stelle als jüngste Eintragung CLS.
Kleine Anwenderdefinitionen lassen sich auf diese Weise in das Vokabular

aufnehmen. Schwieriger wird es bei umfangreicheren Definitionen, die im allgemeinen vor ihrem Funktionieren einiger Korrekturen bedürfen. Zur Speicherung des Programmtextes stellt FORTH sog. Screens zur Verfügung. Als Screen wird eine Informationsmenge verstanden, die sich auf dem Bildschirm darstellen läßt. Im allgemeinen umfaßt ein Screen in FORTH 16 Zeilen á 64 Zeichen wodurch 1024 Zeichen abgespeichert werden können. Computer, die nur einen 40-Zeichen-Bildschirm unterstützen, arbeiten häufig auch mit 512 Zeichen. Angeordnet ist dieser Pufferbereich in dem durch FIRST und LIMIT gekennzeichneten Speicherbereich. Mit dem Editor kann nun die Information im Screen manipuliert werden. Nach Aufruf des Editorvokabulars durch das Wort EDITOR können die Manipulationen vorgenommen werden. In den unterschiedlichen FORTH-Implementationen sind ebenso unterschiedliche Editoren vorzufinden. Hier soll eine einfache Editorversion betrachtet werden, die einen zeilenweisen Zugriff auf die Informationen eines beliebigen Screens erlaubt. Einige Änderungen am Screen SCR # 100 sollen die Handhabung dieses einfachen Editors erläutern. Das Auswählen eines zu editierenden Screens kann mit

```
100 LIST ENTER
```

vorgenommen werden. Besagter Screen wird zum aktuellen Screen deklariert und wie in Bild 3 aufgelistet angezeigt. Will man eine bestimmte Zeile ausdrucken, kann man das Wort T benutzen.

```
9 T ENTER ELSE ." STACK EMPTY !"
CR OK
```

Die vorstehende Eingabe bewirkt also das Ausdrucken der neunten Zeile des aktuellen Screens. Eine Änderung des Zeileninhaltes kann nun mit dem Wort P in der nachstehend gezeigten Weise gemacht werden.

```
9 P ELSE ." STACK LEER !"
CR ENTER OK
```

Durch die Eingabe des Wortes L erfolgt das Listen des aktuellen Screens, und die Änderung ist ablesbar. Wendet man nun das Wort .S an, ist man erstaunt, daß die Veränderung nicht wirksam ist. Der veränderte Screen enthält nur Quelltext, der erst durch 100 LOAD compiliert werden muß. Dann arbeitet das Wort .S mit der deutschen Ausschrift. Will man sich von einem Screeninhalt komplett trennen, dann geschieht das durch die Anwendung des Wortes CLEAR.

```
100 CLEAR ENTER OK
```

läßt den Quelltext vergessen. Wie das Wort VLIST aber zeigt, ist das Wort .S noch im Wörterbuch enthalten und kann angewandt werden.
Die prinzipielle Arbeitsweise eines einfachen, zeilenorientierten Editors ist damit beschrieben. Komfortablere Editoren, sog. Full-Screen-Editoren, zeigen den gesamten Screen auf dem Bildschirm an und ermöglichen durch Cursorsteuerung eine einfache Änderung jeder beliebigen Screenposition.

Konstanten, Variablen und Felder

Die Bewegung der Daten geschieht in FORTH über den Stack. Die FORTH-Worte für Stackmanipulationen, sowie die arithmetischen und logischen Operationen verdeutlichen das (s. Tabelle 1).
Ein gutes FORTH-Programm sollte deshalb das Stackprinzip effektiv nutzen. Trotzdem erfordern viele Anwendungen Konstanten und Variablen, die von FORTH natürlich unterstützt sind. Die Anwendung ist denkbar einfach.

```
123 CONSTANT ABC ENTER OK
456 VARIABLE DEF ENTER OK
```

Erzeugt werden durch die vorstehenden Anweisungen eine Konstante ABC, der der Wert 123 zugewiesen wird, und eine Variable DEF, deren Initialisierungswert 456 beträgt. Der Aufruf der Konstanten hinterläßt auf dem TOS deren Wert.

```
ABC .S ENTER
123 OK
```

Der Aufruf der Variablen hinterläßt auf dem Stack die Adresse des Speicherplatzes, der den Wert der Variablen hält. Mit den speicherbezogenen Worten @ (sprich: „fetch") und ! (sprich: „store") kann die Variable direkt erreicht werden.

```
DEF .S @ .S ENTER
8602
456 OK
```

Eine besondere Art von Variablen sind Feldvariablen oder kurz Felder. In Bild 4 ist ein Screen gezeigt, der ein Feld von 100 Byte erzeugt. Zwei Byte werden durch die Variablendefinition reserviert, weshalb durch das Reservierungswort ALLOT nur noch 98 Byte reserviert werden müssen. In diesen 100 Byte lassen sich dann 50 16-Bit-Zahlen abspeichern.
Das Wort INIT belegt den reservierten Speicherplatz mit dem auf dem Stack vorgefundenen Zahlenwert. Das Wort ZEIGE stellt den Inhalt fortlaufend auf dem Bildschirm dar. Auf beide Worte soll an dieser Stelle nicht weiter eingegangen werden. Sie sind nach der Behandlung der Schleifenkonstruktion lesbar. Durch den in Bild 4 enthaltenen HEX-DUMP ist die Belegung des Feldes nach einer Initialisierung mit dem Wert $\emptyset FFF_H$ angezeigt.
Zur Erzeugung einer Tabelle mit unterschiedlichen Zahlenwerten kann auch das Wort , herangezogen werden. Das Wort , compiliert einen auf dem Stack liegenden Zahlenwert in das Dictionary.

```
35 VARIABLE TEMP 36 , 37 , 38 , 39 ,
ENTER OK
TEMP 4 + @ . ENTER 37 OK
```

Mit dem Wort C, hingegen erfolgt ein byteweises Kopieren.

```
SCR # 101
  0 ( ======== FELDER ======== )
  1
  2 DECIMAL
  3 100 CONSTANT LAENGE
  4 0 VARIABLE FELD
  5 LAENGE 2 - ALLOT
  6
  7 : INIT LAENGE 0
  8    DO DUP FELD I + ! 2 +LOOP
  9    DROP ;
 10
 11 : ZEIGE CR LAENGE 0
 12    DO I DUP FELD + @ SWAP 2 /
 13    . . 2 +LOOP ;
 14
 15

HEX FFF INIT ENTER OK
' FELD NFA 70 DUMP ENTER

219E 84 46 45 4C 04 91 21 C6  .FELD.F.
21A6 07 FF 0F FF 0F FF 0F FF  ._._._._
21AE 0F FF 0F FF 0F FF 0F FF  ._._._._
21B6 0F FF 0F FF 0F FF 0F FF  ._._._._
21BE 0F FF 0F FF 0F FF 0F FF  ._._._._
21C6 0F FF 0F FF 0F FF 0F FF  ._._._._
21CE 0F FF 0F FF 0F FF 0F FF  ._._._._
21D6 0F FF 0F FF 0F FF 0F FF  ._._._._
21DE 0F FF 0F FF 0F FF 0F FF  ._._._._
21E6 0F FF 0F FF 0F FF 0F FF  ._._._._
21EE 0F FF 0F FF 0F FF 0F FF  ._._._._
21F6 0F FF 0F FF 0F FF 0F FF  ._._._._
21FE 0F FF 0F FF 0F FF 0F FF  ._._._._
2206 0F FF 0F FF 0F 84 49 4E  ._._._IN
```

Bild 4. Erzeugung und Initialisierung eines Variablenfeldes

Vergleiche, Entscheidungen, Schleifen

Die Steuerung des Programmlaufes wird in allen Programmiersprachen, so auch in FORTH, durch Vergleiche und Entscheidungen vorgenommen. Schleifenkonstruktionen erlauben eine wiederholte Abarbeitung.
Ein Blick in Tabelle 1 zeigt die zur Verfügung stehenden Vergleichsoperatoren. Zum einen lassen sich zwei auf dem TOS und dem SECOND liegende Zahlen miteinander vergleichen, zum anderen die auf dem TOS liegende Zahl testen. Die Entscheidung ist immer ein

Flag, welches gesetzt oder nicht gesetzt wird. Die Besonderheit von FORTH ist dabei, daß es kein Flagregister gibt, sondern der TOS als Flagregister fungiert. Der Vorteil dieser Vorgehensweise liegt darin begründet, daß auch Rechenergebnisse direkt als Flag gedeutet werden können.

```
1 Ø = . ENTER Ø OK
4 7 < . ENTER 1 OK
```

Vom vorstehenden Flag lassen sich Entscheidungen in Form bedingter Verzweigungen ableiten. Für diesen Fall stehen zwei Standardstrukturen zur Verfügung:

```
IF · · · ENDIF
IF · · · ELSE · · · ENDIF
```

Das Wort IF testet das Flag. Nimmt das Flag den Zustand „wahr" ein, erfolgt die Bearbeitung des sich an IF anschließenden Zweiges. Ist der Zustand des Flags „falsch", dann erfolgt im ersten Fall der Übergang nach ENDIF ohne eine Bearbeitung. Im zweiten Fall wird der sich an ELSE anschließende Zweig bearbeitet. Ein Beispiel soll die Aussagen verdeutlichen.

```
: DIV DUP IF / . ELSE "DIVISOR
= Ø" DROP DROP ENDIF CR;

62 DIV ENTER 3
OK
6 Ø DIV ENTER DIVISOR = Ø
OK
```

Ist der Divisor ungleich Null, dann findet das Wort IF ein wahres Flag vor, und der die Division und Ergebnisausgabe enthaltende Zweig wird durchlaufen. Im Fall eines Divisors mit dem Wert Ø wird von IF der Zustand „falsch" interpretiert und die Fehlermeldung „Divisor = Ø" ausgegeben. Die folgenden DROP-Operationen entfernen die Operanden vom Stack.
Für die Konstruktion von Programmschleifen stellt FORTH verschiedene Möglichkeiten zur Verfügung. Bei fester Schleifendurchlaufzahl wird die Konstruktion DO · · · LOOP verwendet. Die folgenden Anwenderworte zeigen die Funktion:

```
: SCHLEIFE1 10 Ø DO I . LOOP;
: SCHLEIFE2 10 Ø DO I . 2 + LOOP;
SCHLEIFE1 ENTER Ø 1 2 3 4 5 6 7 8 9 OK
SCHLEIFE2 ENTER Ø 2 4 6 8 OK
```

In beiden Schleifen ist der Bereich des Schleifenindex von Ø bis 9 (!). Durch das Wort I wird der jeweils aktuelle Index auf den TOS gebracht und anschließend durch . ausgedruckt. Im Fall von LOOP wird der Index um den Wert 1 erhöht. Im Fall +LOOP erfolgt die Erhöhung um den auf dem TOS liegenden Wert (hier 2). Nicht immer ist die Durchlaufzahl für eine Schleife von vornherein festlegbar. An diesen Stellen lassen sich die mit BEGIN eingeleiteten strukturierten Worte anwenden.

```
: STERNE BEGIN 42 EMIT SPACE
?TERMINAL UNTIL CR;
```

Das Anwenderwort STERNE besteht aus der von BEGIN und UNTIL eingeschlossenen Schleife. Das Wort UNTIL interpretiert den Inhalt des TOS als Flag und entscheidet, ob nach BEGIN zurückgesprungen wird („falsch") oder die Abarbeitung der Schleife abgebrochen und nach UNTIL fortgesetzt wird („wahr"). Das Wort ?TERMINAL legt bei Tastaturbetätigung ein wahres Flag auf den TOS, weshalb dann durch UNTIL die Abarbeitung beendet wird.

```
STERNE ENTER * * * * * * * *
OK
```

Soll das Flag bereits am Anfang der Schleife getestet werden, kann eine etwas abweichende Schleifenkonstruktion herangezogen werden. Das o.a. Beispiel soll etwas abgeändert werden.

```
: STERNE BEGIN ?TERMINAL
Ø = WHILE 42 EMIT SPACE
REPEAT CR ;
```

Zwischen BEGIN und WHILE wird ein Flag erzeugt, welches im Fall „wahr" die Abarbeitung der zwischen WHILE und REPEAT liegenden Worte und einen unbedingten Sprung nach BEGIN veranlaßt. Anderenfalls wird mit der Bearbeitung der Worte nach REPEAT fortgesetzt.

Aus- und Eingabeoperationen

Neben den Operationen, die im Rechner direkt ablaufen, sind die Befehle zur Ein- und Ausgabe über das Computerterminal von besonderem Interesse. In den Beispielen waren bereits einige der Ausgabeoperationen enthalten, die sich mit Tabelle 1 leicht deuten lassen (. .R CR SPACE EMIT). Als Eingabeoperation war das Wort ?TERMINAL bereits in der Anwendung.
Zur Auswertung einer einfachen Tastaturbetätigung existiert das Wort KEY, welches eine Tastaturbetätigung abwartet und den Tastencode auf dem TOS ablegt.

```
: .KEY KEY SPACE EMIT CR ;
```

Das Anwenderwort .KEY erwartet eine Tastaturbetätigung und gibt das betreffende ASCII-Zeichen auf dem Bildschirm aus.

```
.KEY ENTER Q Q
OK
```

Zur Eingabe einer Zahl auf den TOS existiert in FORTH keine eigenständige INPUT-Anweisung. Natürlich sind in FORTH alle Mittel für die Schaffung eines solchen Wortes enthalten.

DATUM	PROJEKT	REV.	AUTOR	SEITE
25. 9.1987			Kühnel	1 VON 1

BESCHREIBUNG	VOCABULARY	WORD
Eingabe einer Zahl von der Tastatur auf den TOS	FORTH	EING

STACK EFFECTS
Länge --- Zahl

STACK	TOP	WORDS
	N	: EING
N	PAD	PAD
PAD	N	SWAP
		EXPECT
		Zahleneing.
Ø	Ø	Ø.
Ø	PAD	PAD
PAD	1	1
Ø	PAD -1	-
Zahl	PAD	(NUMBER)
Zahl	Ø	DROP
	Zahl	DROP ;

Bild 5. Definition des Wortes EING im Programmierformular nach 4

Bild 5 enthält die Definition des Wortes EING. Das Programmierformular wurde [4] entnommen und ermöglicht eine übersichtliche Erarbeitung und Dokumentation eigener FORTH-Worte. Auf dem TOS wird eine Längenangabe der einzugebenden Zahl erwartet. Das Wort EXPECT interpretiert die Eingabe als String und benötigt für die Abspeicherung die Stringlänge. Kann die Länge nicht von vornherein angegeben werden, gibt man die größtmögliche Länge vor und schließt mit ENTER die Eingabe ab.
Für die Abspeicherung werden dann nur die Stellen bis zur Betätigung der ENTER-Taste herangezogen. PAD ist ein temporärer Speicher, der zur Zwischenspeicherung der eingegebenen Zeichenketten Verwendung findet. Damit EXPECT die Operatoren in der richtigen Reihenfolge auf dem Stack vorfindet, wird durch SWAP TOS und SECOND vertauscht. Ist nun die Eingabe abgeschlossen, dann befindet sich die Ziffern-Zeichenkette beginnend ab Adresse PAD bis PAD + Stellenzahl. Die Umwandlung der Zeichenkette in einen numerischen Wert besorgt das nicht in der Zusammenstellung enthaltene Wort (NUMBER). Durch Ø. wird eine 32-Bit-Zahl mit Ø belegt. Desweiteren benötigt (NUMBER) den Speicherplatz unterhalb der Zeichenkette, d.h. PAD-1, was durch PAD 1 – gesichert ist. Nach Abschluß der Umwandlungsprozedur steht die eingegebene Zahl auf dem THIRD, der drittobersten Stackzelle.
Die Inhalte von TOS und SECOND sind nicht mehr von Interesse und werden durch das zweifache DROP vom Stack entfernt. Die eingegebene Zahl steht nunmehr auf dem TOS zur Verfügung. Das in Bild 5 definierte Wort EING wird wie folgt gehandhabt.

```
8 EING ENTER 123 ENTER OK
.ENTER 123 OK
```

Das Gegenstück zum Wort EXPECT ist das Wort TYPE. Als Operanden werden wieder Adresse und Stellenzahl auf dem Stack erwartet.

```
PAD 5 EXPECT ENTER ABCDE OK
PAD 3 TYPE ENTER ABC OK
```

Das FORTH-Standard-Format einer Zeichenkette enthält stets ein führendes Count-Byte (Länge der Zeichenkette). Mit dem Wort COUNT werden die Voraussetzungen für TYPE geschaffen. Auf diese Weise lassen sich z.B. die Dictionary-Eintragungen ausgeben.

```
' VLIST NFA DUP . ENTER 6004 OK
COUNT 128 — TYPE ENTER VLISTOK
```

Die Namensfeldadresse des Wortes VLIST liegt bei 6004. Wie eingangs gezeigt, wird das erste Byte des Eintrages durch die Länge des Namens + 8Ø$_H$ gebildet. COUNT liefert die Adresse für TYPE und die Zeichenkettenlänge, die durch Subtraktion von 128 korrigiert wird.
Weitere, die Zeichenkettenarbeit direkt unterstützende Worte existieren in FORTH nicht. Ein String-Package, welches den von BASIC her bekannten Umfang überdeckt, ist in [1] enthalten.
Für die Formatierung der Ausgabe sind in FORTH Möglichkeiten enthalten, die aus Gründen des beschränkten Platzes nicht im einzelnen aufgeführt werden können.

3. Die Arbeit mit dem Massenspeicher

Als Massenspeicher wird heute in FORTH-Systemen im allgemeinen die Diskette verwendet. Weniger komfortable Systeme arbeiten auch mit dem Kassettenmagnetband.
FORTH betrachtet den Massenspeicher als einfache Erweiterung seines Arbeitsspeichers (RAM). Der Adreßraum des Massenspeichers ist in Blöcke segmen-

tiert, deren Größe durch die Systemkonstante B/BUF (Bytes/Buffer) ausgedrückt wird. Die Blöcke werden durch eine Blocknummer gekennzeichnet, die von Null an fortlaufend bis an die Grenze des Adreßraumes vergeben wird. Um den Datenaustausch zwischen RAM und Diskette zu realisieren, wird ein Pufferspeicher (Disk-Buffer-Area, begrenzt durch FIRST und LIMIT) eingerichtet. Betrachtet man Bild 2, dann kann ein Pufferspeicher von 4112 Byte abgelesen werden. In diesen Bereich werden dann

LIMIT @ FIRST @ — B/BUF /MOD..
ENTER 8 16 OK

acht Blöcke (á 512 Byte) abgelegt. Die restlichen 16 Byte fungieren als Kopf und Schwanz an jedem Block. Der Kopf beinhaltet dabei die Blocknummer, wobei das MSB noch die Funktion des sog. „update-Bit" hat. Ein Block, in dem Veränderungen vorgenommen wurden (update-Bit = 1), wird vor einem neuen Zugriff erst auf Diskette gerettet. Der Pufferspeicher wird wie ein Fenster über den Massenspeicher geschoben, wodurch ständig auf einen bestimmten Bereich zugegriffen werden kann. Die Organisation wird durch eine Reihe von Variablen und FORTH-Worten übernommen, die nach außen hin weitgehend verborgen bleiben. Von besonderer Wichtigkeit sind die Worte LIST und LOAD, die beide auf dem TOS eine Screennummer erwarten. Mit LIST erfolgt eine formatierte Ausgabe des Quelltextes, der im betreffenden Screen abgespeichert wurde. LOAD compiliert den Quelltext in das Dictionary. Bei Neuerstellung oder Editierung sollte zur Sicherung immer ein UPDATE vorgenommen werden, da nur so bei erneuter Compilierung der Screeninhalt auch auf Diskette gerettet werden kann.
FORTH-Systeme, die anstelle eines Diskettenspeichers eine Magnetbandkassette als Massenspeicher aufweisen, sind grundsätzlich ähnlich organisiert. Im allgemeinen existieren BASIC-ähnliche Befehle (CSAVE, CLOAD), um den Pufferbereich mit Quelltext oder Daten zu laden bzw. diese abzuspeichern. Diese computerabhängigen Erweiterungen des FORTH-Vokabulars sollen aufgrund ihrer Spezifik an dieser Stelle nicht weiter betrachtet werden und sind im betreffenden Handbuch nachzulesen.

4. Zusammenfassung

Es wurde versucht, die sicher anfangs recht eigenwillige Programmiersprache FORTH vorzustellen. Dem Umfang des Beitrages entsprechend, waren bestimmte Probleme nur anzureißen. Dem Leser wird empfohlen, die kleinen Programmbeispiele an Hand der Tabelle 1 zu analysieren. Komfortable FORTH-Versionen weisen einen Sprachumfang von etwa 250 Worten auf. Die Zusammenstellung zeigt einen wichtigen Bestandteil dieser Menge, doch konnten diese Worte nicht vollständig behandelt werden.
Die greifbare Literatur zu FORTH ist noch in starkem Maße beschränkt, weshalb im Literaturverzeichnis neben den zitierten Quellen einige weitere, die Arbeit mit FORTH unterstützende Stellen angezeigt wurden.

Literatur

[1] Zech, R.: Die Programmiersprache FORTH. – München: Franzis Verlag, 1984
[2] Kühnel, C.: Erste Erfahrungen mit FORTH. – In: radio fernsehen elektronik, Berlin 35 (1986) 9, S. 553—557
[3] FORTH-Systeme für die Industrie. – Katalog der Fa. Angelika Flesch, Titisee-Neustadt 1986, S. 27—35

[4] Briner, R. G.: Double Number Words in FORTH. – In: Elektroniker, Aarau 22 (1983) 18, S. 49—55
[5] Woitzel, E.: comFORTH – Programmierwerkzeug FORTH unter SCP. — In: edv-aspekte, Berlin (1986) 4, S. 47—52
[6] Brodie, L.: Programmieren in FORTH. 7 München, Wien: Carl Hanser Verlag, London: Prentice-Hall Internationa l Inc., 1984
[7] Vack, G.-U.: FORTH: Eine außergewöhnliche Softwarekonzeption. – In: Mikroprozessortechnik, Berlin 1 (1987) 6, S. 163—165
[8] Schiemann, B.: Ein fig-kompatibles FORTH für den U 8000. – In: Mikroprozessortechnik, Berlin 1 (1987) 6, S. 165—166
[9] Bachmann, B.: Gleitkomma in FORTH. – In: Mikroprozessortechnik, Berlin 1 (1987) 6, S. 166—168
[10] Bachmann, B.: Steuerung des Plotters K 6418 in FORTH. – In: Mikroprozessortechnik, Berlin 1 (1987) 6, S. 168—169
[11] Vack, G.-U.: Hardware-Realisierung von FORTH. – In: Mikroprozessortechnik, Berlin 1 (1987) 6, S. 169—170
[12] Horn, Th.: Programmiersprachen — ein Vergleich an Hand von Beispielen. – In: Kleinstrechner-TIPS, Heft 4. Leipzig: Fachbuchverlag, 1986
[13] Domschke, W.: Der Modul M 026 FORTH für die Kleincomputer KC 85/2 und KC 85/3. – In: Mikroprozessortechnik, Berlin 1 (1987) 8, S. 244—246

Autor:
Dr.-Ing. Claus Kühnel
Zschertnitzer Str. 52
Dresden, 8020

Hinweise für Autoren

Uns erreichen ständig Briefe mit Veröffentlichungsangeboten, für die wir sehr dankbar sind, und wir hoffen in Zukunft auch auf viele Zuschriften und Ideen, damit die Kleinstrechner-TIPS möglichst abwechslungsreich und vielseitig gestaltet werden können.

Für eine Veröffentlichung Ihrer Programme ist Voraussetzung, daß an den Programmen keine Rechte Dritter bestehen und das Programm möglichst auf einer originellen Idee oder Lösung aufbaut.

Weiterhin ist eine gute methodisch-didaktische Aufbereitung des Programms **und** der Veröffentlichung wichtig. Dabei sollte man zur Erklärung der Struktur des Programms, der Algorithmen und der Datenstrukturen verstärkt geeignete grafische Mittel, wie Struktogramme u. ä., verwenden. Die Praxis zeigt, daß wenn das Programm das erste Mal „gelaufen" ist, die Arbeit erst anfängt. Es muß nämlich das Programm für eine Veröffentlichung aufbereitet werden, wobei meistens unter Berücksichtigung der bei Entwurf und Testung gesammelten Erfahrungen eine Neuprogrammierung mit einer völlig veränderten Programmstruktur das Programm zu einer veröffentlichungsreifen originellen Lösung werden lassen. Scheuen Sie deshalb die „kleine" Mühe der Neuprogrammierung nicht, und betrachten Sie Ihre Programme unter diesem Aspekt einmal kritisch. Vielleicht ergibt sich daraus eine interessante Veröffentlichung.

Der binäre Suchbaum

Einleitung

Suchprozesse in Datenbanken gehören zu den wichtigsten Prozessen der Informationsverarbeitung. Man denke dabei z.B. an Bibliotheken, Lagerbestandsdateien oder auch Telefonverzeichnisse. Eine sicherlich sofort einsichtige Methode ist das sequentielle Durchblättern der Datei, bis der gewünschte Datensatz gefunden wurde. Allerdings stellt sich sehr bald heraus, daß bei großen Datenbeständen diese Methode zu keinem befriedigenden Ergebnis führt, da die Suchzeit in ungünstigen Fällen zu hoch ist. Aus diesem Grunde soll im folgenden ein Verfahren vorgestellt werden, welches Suchprozesse erstaunlich effektiviert. Es arbeitet mit sogenannten Suchbäumen. Als einfachster Vertreter sei hier der Binärbaum angeführt.

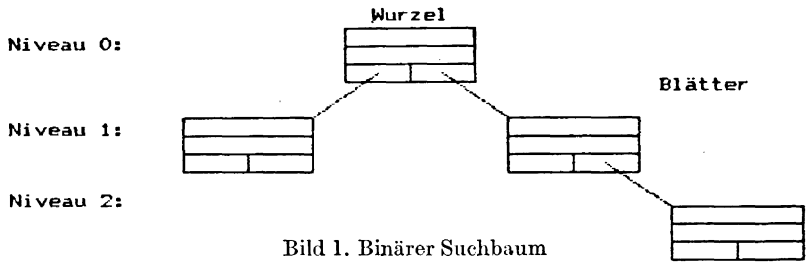


Bild 1. Binärer Suchbaum

1. Datenstruktur

Ein binärer Suchbaum (Bild 1) besteht aus einer Wurzel (Niveau 0) sowie einer Reihe von Datenblättern. Die Wurzel, wie auch jedes Datenblatt, besitzt maximal zwei Nachfolgeblätter.
Die Ordnung in einer solchen Baumstruktur ist als lexikografische Ordnung ($<\cdot$) bei alphanumerischen Schlüsselwörtern zu verstehen. Damit soll gelten:

LINKES_NACHFOLGEBLATT $<$-BLATT $<$-RECHTES_NACHFOLGEBLATT.

Ganz allgemein läßt sich die Struktur eines binären Baumes wie folgt definieren:

$\langle \text{baum}_i \rangle ::= [\langle \text{baum}_{i+1} \rangle]$ BLATT $[\langle \text{baum}_{i+1} \rangle]$

mit $i = 0, 1, 2, \ldots, n$. Der Index i soll hier nur zur Kennzeichnung des Niveaus dienen. Es läßt sich hieraus sofort erkennen, daß je Niveau maximal 2^i Blätter

vorkommen. Für einen sogenannten ausgeglichenen Baum, d.h. einen Baum, wo jedes Blatt genau zwei Nachfolgeblätter besitzt, bedeutet das, daß bei Niveau 10 bereits $2^0+2^1+2^2+\ldots+2^{10}=2^{11}-1=2047$ und bei Niveau 20 2097151 Blätter existieren.

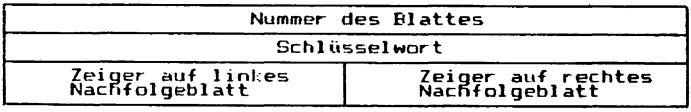


Bild 2. Datenstruktur eines Blattes des binären Suchbaumes

Die Struktur eines Blattes wird in Bild 2 verdeutlicht. Die Nummer des Blattes dient gleichzeitig (und in erster Linie) der Identifikation des Datensatzes in der zugrunde liegenden sequentiellen Datei, wodurch dieser durch Direktzugriff (z.B. Prozedur SEEK in TURBO-PASCAL) erreichbar wird. Die angegebenen Zeiger dienen der Durchführung der im folgenden darzustellenden Grundoperationen über Baumstrukturen. Mit Hilfe des Schlüsselwortes wird die Realisierung der weiter oben erwähnten lexikografischen Ordnung ermöglicht.
In PASCAL-Notation sind für diese Datenstruktur folgende Deklarationen notwendig:

```
TYPE ds = ^datensatz;
     datensatz = RECORD
                   nummer : INTEGER;
                   wort   : STRING[20];
                   links, rechts : ds
                 END;
```

Der Typ **ds** ist hier ein Pointer-Typ, der auf die Struktur **datensatz** zeigt. Dieser Record-Typ enthält die oben aufgezählten Elemente: Blattnummer, Schlüsselwort, Zeiger auf Links- und Rechtsnachfolgeblatt.
In BASIC ist die Datendefinition für unser Problem aufwendiger, da weder Pointer- noch Record-Typen zur Verfügung stehen. Diese müssen daher simuliert werden, z.B. mit Hilfe eines eindimensionalen Zeichenkettenfeldes D$ (I) mit I = 1 (1) N und $N < 1000$. Der Aufbau eines Elementes dieses Feldes könnte dann, wie in Bild 3 gezeigt, aussehen.

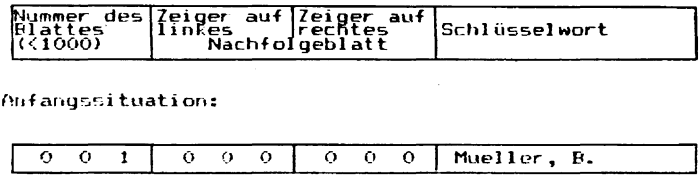


Bild 3. Umsetzung der Datenstruktur mit BASIC mit DS(I) und I := 1, 2, . . ., N Datensätzen und $N < 1000$ (keine Pointervariable)

2. Grundoperationen über Baumstrukturen

Wir wollen hier im wesentlichen vier solcher Grundoperationen betrachten:

1. Einfügen eines neuen Blattes in die Baumstruktur.
2. Ausgabe aller Schlüsselwörter entsprechend ihrer lexikografischen Ordnung..
3. Suchen eines Schlüsselwortes und Zugriff auf den zugeordneten Datensatz.
4. Löschen eines Blattes.

2.1. Einfügen

Diese Operation bildet die Grundlage für den Aufbau eines Suchbaumes, denn sie wird zum Neuaufbau genauso benötigt wie zur Erweiterung eines bereits bestehenden Baumes. Bild 4 veranschaulicht den für das Einfügen notwendigen Algorithmus in Form einer speziellen verbalen Beschreibung. Es ist zu beachten, daß diese, wie auch die folgenden Grundoperationen, in starkem Maße rekursiv realisiert werden können.

```
einfügen (schlüssel_neu);
  blattnr := blattnr + 1;
  vergleich (schlüssel_neu, wurzelschlüssel)
end;

vergleich (key1, key2);
  wenn key1 = key2
  dann fehler; blattnr := blattnr - 1
 sonst wenn key1 < key2
       dann wenn linker_nachfolger (key2) = nil
            dann blattlinks (key2)
            sonst vergleich (key1, linker_nachfolger (key2))
       sonst wenn rechter_nachfolger (key2) = nil
            dann blattrechts (key2)
            sonst vergleich (key1, rechter_nachfolger(key2))
end;
```

Bild 4. Algorithmus zur Operation „Einfügen"

2.2. Ausgabe

Diese Grundoperation läßt sich mittels eines backtracking-Algorithmus (backtrack svw. rückspuren) verwirklichen. Ausgangspunkt ist hierfür die äußerste linke Bahn des Suchbaumes. Das so gefundene, am weitesten links stehende Schlüsselwort ist in der lexikografischen Ordnung des Baumes das kleinste Element und wird somit als erstes ausgegeben. Das nächste auszugebende Blatt ist das Vorgängerblatt. Danach wird dessen rechter Nachfolger als Wurzel eines Teilbaumes betrachtet und die Ausgabe wieder mit dem äußersten linken Element fortgesetzt. Besteht ein solcher Teilbaum nur noch aus einer Wurzel, so wird diese ausgegeben und so lange zu Vorgängerblättern zurückgegangen, bis ein Schlüsselwort größer als das aktuelle ist. Dann wird dieses ausgegeben und rechts bzw., falls rechts kein Nachfolger existiert, beim Vorgänger fortgesetzt. Beendet ist der Algorithmus, wenn das äußerste rechte Blatt oder im backtracking die Wurzel von rechts erreicht wurde.

Der hier kurz beschriebene Algorithmus arbeitet iterativ. Es werden in erster Linie Zyklen mit Abbruchtest verwendet. Sehr zu empfehlen sind rekursive Algorithmen für backtracking, denn sie gestatten die unmittelbare Notation des o.g. Verfahrens: Voranschreiten auf den entsprechenden Ästen des Baumes bis keine Daten mehr vorhanden sind; dann „Rückzug" zum letzten Entscheidungspunkt (links/rechts) und erneutes Voranschreiten mit neuen aktuellen Parametern (Bild 5).

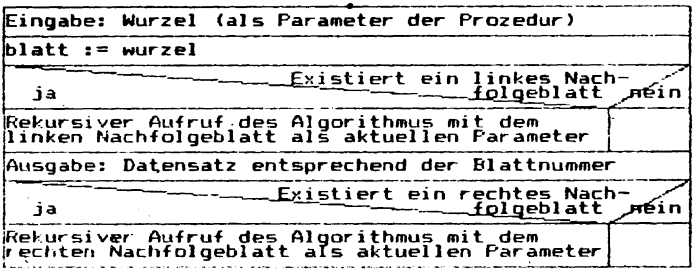


Bild 5. Algorithmus zur Operation „Ausgabe"

2.3. Suchen

Die Operation **Suchen** funktioniert entsprechend der unter 1. definierten lexikografischen Ordnung. Es wird auf Identität des Suchschlüssels mit dem aktuellen Schlüsselwort getestet. Erfolgte der Test positiv, dann ist der Suchvorgang abgeschlossen, ansonsten wird entsprechend der Ordnung im Baum weiter verzweigt und obiger Test wiederholt. Der Abbruch erfolgt ebenfalls bei erfolglosem Durchlaufen des Baumes (vgl. Bild 6).

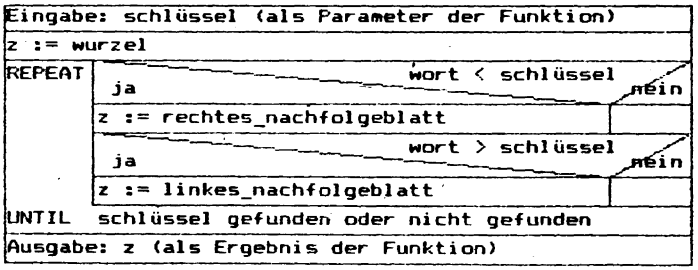


Bild 6. Algorithmus zur Operation „Suche"

Dieser Grundoperation wegen wurde die Datenstruktur **binärer Suchbaum** definiert. Dies erklärt sich, wie anfangs bereits erwähnt, aus der hohen Effizienz von Suchprozessen in solchen Strukturen, denn beim Vergleich kann der jeweils rechte/linke Teilbaum unberücksichtigt bleiben. Bei $63 = 2^6 - 1$ verschiedenen Such-

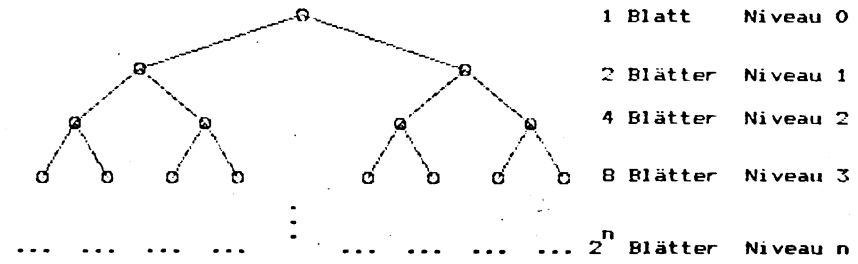


Bild 7. Komplexität eines binären Suchbaumes

schlüsseln (k) sind das im ungünstigsten Falle (worst case) nur sechs Vergleiche (v) nötig, bei sequentiellen Suchen in ungeordneten Listen können bis zu 63 Vergleiche erforderlich sein (vgl. Bild 7). Allgemein gilt $v \sim \ln(k+1)$, woraus sich die besondere Bedeutung der Suche auf Binärbäumen für eine große Anzahl von Suchschlüsseln ergibt. Natürlich verhält sich die Anzahl der notwendigen Vergleiche proportional zur dazu erforderlichen Rechenzeit. Dieser Fakt rechtfertigt das relativ aufwendige Einlesen und Löschen von Blättern.

2.4. Löschen

Der wohl komplizierteste Algorithmus ist für die Grundoperation **Löschen** notwendig. In ungünstigen Fällen wird sogar eine völlige Umstrukturierung des durch das zu löschende Blatt gekennzeichneten Teilbaumes notwendig werden. Dies muß genau dann geschehen, wenn das zu löschende Blatt einen linken und einen rechten Nachfolger besitzt.
Existiert zunächst kein Nachfolger, dann ist die Löschung vollkommen unproblematisch. Im Falle eines Nachfolgers wird dieser an die Stelle des zu löschenden Blattes gesetzt. Für den ungünstigsten Fall (zwei Nachfolger) gibt es unterschiedliche Methoden der Umstrukturierung des besagten Teilbaumes. Eine Methode ist die, das zu löschende Blatt mit dem Teilbaum eines Nachfolgers zu ersetzen und die Blätter des zweiten Teilbaumes mit Hilfe der **Einfüge-Prozedur** neu in den Baum einzugliedern. Der Aufwand hierfür kann allerdings unvertretbar hoch werden.
Ein einfacheres Verfahren ist folgendes: Wir ersetzen das zu löschende Blatt durch ein anderes Blatt mit höchstens einem Nachfolger. Dieses Blatt muß kleiner als der rechte und größer als der linke Nachfolger des zu löschenden Blattes sein. Der Einfachheit halber sollte hier ein Blatt gewählt werden, das diese Bedingung in jedem Fall erfüllt. Es gibt an jedem Suchbaum genau zwei solche Blätter. Im linken Teilbaum ist es das am weitesten rechts stehende Blatt und im rechten Teilbaum das am weitesten links stehende Blatt. Es handelt sich dabei also um das größte linke bzw. das kleinste rechte Blatt (Bild 8).

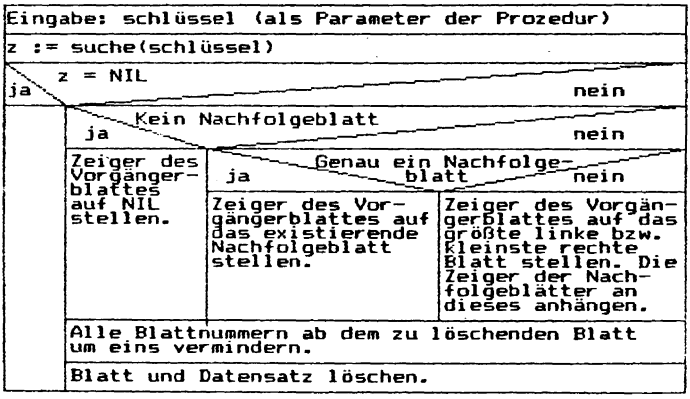


Bild 8. Algorithmus zur Operation „Löschen"

```
program BINAERER_SUCHBAUM;

type s=string[20];
     ds=^datensatz;
     datensatz=record
                 nummer      : integer;
                 wort        : s;
                 links,rechts: ds
               end;

var  wurzel, p: ds;
     blattnr  : integer;
     w        : s;

function suche (schluessel: s): ds;
var  z: ds;

begin
  z:=wurzel;
  repeat
    if z^.wort<schluessel
    then z:=z^.rechts;
    if  z^.wort>schluessel  then  z:=z^.links
    until  ((z^.wort=schluessel) or (z^.links=z^.rechts));
    suche:=z
end;


procedure vergleich (key1, key2: s);
var  z,z1: ds;

begin
  if  key1=key2  then
    begin
      writeln('Schluessel schon vorhanden!');
      blattnr:=blattnr-1
    end
  else
    begin
      z:=suche(key2);
      if  key1<key2  then
        if  z^.links=nil  then
          begin
            new(z1);
            z^.links:=z1;
            z1^.links:=nil;
            z1^.rechts:=nil;
            z1^.nummer:=blattnr;
            z1^.wort:=key1
          end
        else  vergleich(key1,z^.links^.wort)
      else
        if  z^.rechts=nil  then
          begin
            new(z1);
            z^.rechts:=z1;
            z1^.links:=nil;
            z1^.rechts:=nil;
            z1^.nummer:=blattnr;
            z1^.wort:=key1
          end
        else  vergleich(key1,z^.rechts^.wort)
    end;
end;
```

```
procedure einfuegen (schluessel_neu: s);

begin
  blattnr:=blattnr+1;
  vergleich(schluessel_neu,wurzel^.wort)
end;


procedure ausgabe(knoten: ds);

begin
  if  knoten<>nil  then
    with  knoten^  do  writeln(nummer:4,          ,wort)
  else  writeln('Schluessel nicht vorhanden!')
end;


procedure list(blatt: ds);

begin
  if  blatt^.links<>nil  then list(blatt^.links),
  ausgabe(blatt);
  if  blatt^.rechts<>nil  then list(blatt^.rechts)
end;


procedure menue;
var i: char;
    w: s;

begin
  repeat
    writeln;
    writeln('      B i n a e r b a u m');
    writeln('      ~~~~~~~~~~~~~~~~~~~');
    writeln('  1...Einlesen und Einfuegen');
    writeln('  2...Ausgabe der Schluessel in lexik. Anordnung');
    writeln('  3...Suchen eines Schluessels');
    writeln('  4...Ende des Programms');
    read(kbd,i);
    case i of
    '1': begin
           w:='';
           repeat
             write('Eingabe des naechsten Schluessels:  ');
             w:=''; readln(w);
             if  w<>''  then  einfuegen(w)
           until  w=''
         end;



    '2': begin
           clrscr;
           list(wurzel)
         end;
    '3': begin
           write('gesuchter Schluessel:  ');
           w:=''; readln(w);
           ausgabe(suche(w))
         end
    end
  until i='4'
end;
```

```
begin
  clrscr;
  new(wurzel);
  blattnr:=1;
  wurzel^.links:=nil;
  wurzel^.rechts:=nil;
  wurzel^.nummer:=blattnr;
  wurzel^.wort:='hans';
  menue;
  clrscr
end.
```

Bild 9. Programmlisting

Ein wesentlicher Aspekt beim Löschen eines Blattes darf allerdings nicht übersehen werden. Es ist notwendig, alle Blattnummern ab dem zu löschenden Blatt um eins zu vermindern. Bei der Umsetzung dieses Algorithmus mit BASIC kommt noch hinzu, daß auch die entsprechenden Zeiger geändert werden müssen.

3. Realisierung ausgewählter Grundoperationen mit PASCAL

Das Programmlisting (Bild 9) soll ausschließlich die beschriebenen Grundoperationen über einem binären Suchbaum realisieren. Die Prozedur **Löschen** wird allerdings dem Leser überlassen. Es ist weiterhin nicht Anliegen des Programmes, den eigentlichen Zweck eines solchen Baumes zu berücksichtigen, d.h. die dazugehörige Arbeit mit einer sequentiellen Datei zu steuern. Der interessierte Leser wird die dazu notwendigen Anweisungen sicherlich recht schnell selbst einfügen können.
Die Eingabe der Schlüsselwörter erfolgt über die Tastatur, alles andere ist automatisiert. Als PASCAL-Variante kommt TURBO-PASCAL (Version 3.0) zur Anwendung. Auf eine Eingabekontrolle sowie Feinheiten, z.B. die Bildschirmorganisation, wurde verzichtet.

Autoren:
Dr. Gert Hänsel
Akademie der Pädagogischen Wissenschaften
Forschungs- und Rechenzentrum Dresden

Dr. Christian Wagenknecht
Pädagogische Hochschule Dresden

Berichtigung zu Heft 8

Beitrag „Zur Ermittlung der exakten Lösungen algebraischer Gleichungen auf einem Rechner"

Die Programmzeilen 460 und 688 sind wie folgt zu korrigieren:

460 FI=−ATN (HY/SQR (1−HY* HY)) + PI/2

688 PRINT"X"; IN; "="; H1; "+I*"; − SQR (ABS(BI))

Prof. Dr. Dieter Oelschlägel

Nullstellenbestimmung nichtlinearer Funktionen mit einer Veränderlichen

1. Problemstellung

Bei der Bearbeitung einer Reihe mathematischer und technischer Probleme treten nichtlineare Gleichungen der Form $f(x) = 0$ auf. Die Lösungen dieser Gleichung sind die Nullstellen der Funktion $f(x)$. In den meisten Fällen gibt es aber keinen Algorithmus zur exakten Bestimmung dieser Nullstellen. Deshalb werden Näherungsverfahren angewandt. Gebräuchlich sind Iterationsverfahren, deren Ziel darin besteht, eine gewählte Anfangsnäherung schrittweise zu verbessern, bis eine gewünschte Genauigkeit für die Lösung erreicht ist.

Ziel dieses Beitrages ist es zu zeigen, wie diese numerischen Verfahren auf dem Kleincomputer KC 85/1 realisiert werden. Im vorliegenden Programm (Bild 1) werden folgende vier Verfahren implementiert:

- Intervallhalbierungsverfahren
- Regula falsi
- NEWTON-Verfahren
- Einfaches Iterationsverfahren.

Diese Verfahren werden im folgenden kurz beschrieben.

2. Beschreibung der Verfahren

2.1. Intervallhalbierung und Regula falsi

Das Intervallhalbierungsverfahren und die Regula falsi gehören zu den Intervallschachtelungsverfahren. Das bedeutet, es wird von einem Anfangsintervall (x_1, x_2) ausgegangen, das schrittweise verkleinert wird. Vorausgesetzt wird, daß die Funktion $f(x)$ in diesem Intervall stetig ist und $f(x_1)$ und $f(x_2)$ verschiedene Vorzeichen besitzen. Damit ist gewährleistet, daß sich im Intervall (x_1, x_2) mindestens eine Nullstelle befindet.

Beim **Intervallhalbierungsverfahren** wird dieses Anfangsintervall halbiert. Man erhält den Halbierungspunkt x_3, der zusammen mit der Intervallgrenze, deren Funktionswert das entgegengesetzte Vorzeichen wie $f(x_3)$ besitzt, ein neues Intervall (x_i, x_3) $(i = 1$ oder $i = 2)$ bildet, welches die gesuchte Lösung besser einschließt als das vorherige Intervall. Mit dem erhaltenen neuen Intervall wird genauso verfahren. Dies wird so oft wiederholt, bis das Intervall (x_k, x_n) die exakte Lösung mit der gewünschten Genauigkeit einschließt.

Bei der **Regula falsi** wird die Funktion im Anfangsintervall durch eine Sekante, die durch die Punkte $(x_1, f(x_1))$ und $(x_2, f(x_2))$ verläuft, angenähert. Der Schnittpunkt der Sekante mit der x-Achse ist die neue Intervallgrenze x_3, die mit x_1 bzw. x_2, je nach Vorzeichen der Funktionswerte, das neue Intervall (x_i, x_3) $(i = 1$ oder $i = 2)$ bildet.

```
10 GOTO260
20 DEF FNY(X)=  ---  61 LEERZEICHEN  ---
30 RETURN
40 DEF FNY1(X)= ---  60 LEERZEICHEN  ---
50 RETURN
60 DEF FNI(X)=  ---  61 LEERZEICHEN  ---
70 RETURN
80 ! * * * * * * * * * * * * * * * * * *
90 ! UP FUNKTIONSUEBERTRAGUNG
100 ! * * * * * * * * * * * * * * * * * *
110 X(40)=40:X(41)=41:X(42)=174:X(43)=172:X(45)=173:X(47)=175
120 X(61)=180:X(94)=176:X(32)=32:X(88)=88:X(255)=32:X(46)=46
130 X(76)=191:X(80)=199:X(84)=195:X(131)=184:X(146)=193:X(149)=196
140 X(151)=183:X(154)=182:X(156)=194:X(157)=192:X(164)=189
150 LET L=4478
160 LET L=L-1:LET M=M+1
170 IF L=4419 THEN RETURN
180 LET K=PEEK(-L)
190 IF K>=48 AND K<=57 THEN POKE M,K:GOTO160
200 IF K<64 OR K=94 OR K=88 OR K=255 THEN POKE M,X(K):GOTO160
210 IF K=76 OR K=80 THEN POKE M,X(K):M=M+1:L=L-2:GOTO180
220 IF K>64 AND K<84 THEN POKE M,X(K+PEEK(-L+1)):L=L-3:M=M+1:GOTO180
230 ! * * * * * * * * * * * * * * * * * *
240 ! HAUPTPROGRAMM
250 ! * * * * * * * * * * * * * * * * * *
260 WINDOW:CLS:LET Z=1:DIM X(260)
270 WINDOW:CLS:PRINTAT(0,7);"BERECHNUNG VON NULLSTELLEN"
280 PRINTAT(1,7);STRING$(26,""):PRINTAT(2,7);"NICHTLINEARER GLEICHUNGEN"
290 PRINTAT(3,7);STRING$(26,""):X$="FUNKTION ZU LANG!"
300 PRINTAT(5,0);STRING$(40,"6")
310 WINDOW 10,23,0,39:CLS:PRINT"Geben Sie bitte Ihre Funktion in der"
320 PRINT:PRINT" Form  Y=f(X)  ein !":M=1045
330 WINDOW16,23,0,39:CLS:INPUT"Y=";D$
340 IF LEN(D$)>58 THEN PRINTAT(19,11);X$:BEEP:PAUSE20:GOTO310
350 GOSUB110
360 ! WERTEBERECHNUNG
370 WINDOW 7,23,0,39:CLS:WINDOW10,23,0,39
380 PRINT"Wollen Sie Funktionswerte berechnet"
390 PRINT:PRINT"haben (5 Werte) ?"
400 PRINT:PRINTSPC(17);"(Y/N)":AB=0
410 LET B$=INKEY$
420 IF B$="N" THEN500
430 IF B$="Y" THEN440:ELSE GOTO410
440 WINDOW10,23,0,39:CLS
450 PRINTAT(7,10);"* WERTEBERECHNUNG *":GOSUB20
460 INPUT"X=";A:PRINTSPC(20);"Y=";FNY(A):AB=AB+1
470 PRINT:IF AB=5 THEN PAUSE20:GOTO370
480 GOTO460
490 ! VERFAHRENSAUSWAHL
500 WINDOW10,23,0,39:CLS:PRINT" Welches Verfahren wollen Sie"
510 PRINT:PRINT" verwenden?"
520 PRINT:PRINT:PRINT"   INTERVALLHALBIERUNG   1"
530 PRINT:PRINT"   REGULA FALSI          2"
540 PRINT:PRINT"   NEWTONVERFAHREN       3"
550 PRINT:PRINT"   EINFACHE ITERATION    4"
560 PRINT:INPUT"VERFAHREN:";A$:ON VAL(A$) GOSUB1000,1200,1400,1700
570 ! LOESUNGSAUSDRUCK
580 WINDOW 16,21,0,39:CLS
590 PRINTAT(23,3);"'CONT'"
600 IF CY=0 THEN PRINT SPC(10);"GENAUE LOESUNG:";C:PAUSE:GOTO710
610 PRINT SPC(10);"LOESUNG:";C;PAUSE
620 ! VERGLEICHSTBELLE
630 WINDOW 7,23,0,39:CLS:PRINTSPC(13);"* VERGLEICH *"
640 PRINT:PRINT:PRINT"METHODE","ITERATIONEN","LOESUNG"
```

```
650 PRINT"
660 PRINT:PRINT"INTERV.",Z1,C1
670 PRINT:PRINT"REGULA.",Z2,C2
680 PRINT:PRINT"NEWTON.",Z3,C3
690 PRINT:PRINT"ITERAT.",Z4,C4
700 PRINTAT(23,3);"'CONT'":PAUSE
710 ! NEUE RECNUNG?
720 WINDOW 7,23,0,39:CLS
730 PRINTAT(16,12);"NEUE RECHNUNG ?"
740 PRINTAT(18,16);"(Y/N)"
750 LET B$=INKEY$
760 IF B$="Y" THEN LET Z=1:X=0:GOTO790
770 IF B$="N" THEN WINDOW:CLS:END
780 GOTO750
790 PRINTAT(16,12);"NEUE FUNKTION ?"
800 LET B$=INKEY$
810 IF B$="Y" THEN H=0:G=0:C1=0:C2=0:C3=0:C4=0:Z1=0:Z2=0:Z3=0:Z4=0:GOTO310
820 IF B$="N" THEN500
830 GOTO800
1000 ! * * * * * * * * * * * * * * * *
1010 ! UP INTERVALLHALBIERUNG
1020 ! * * * * * * * * * * * * * * * *
1030 PRINTAT(7,8);"* INTERVALLHALBIERUNG *":Z=1
1040 GOSUB20
1050 GOSUB2030
1060 IF X=1 THEN RETURN
1070 LET C=(A+B)/2
1080 LET C1=C:LET Z1=Z
1090 IF ABS(FNY(C))<F THEN RETURN
1100 GOSUB2200
1110 GOTO1060
1200 ! * * * * * * * * * * * * * * * *
1210 ! UP REGULA FALSI
1220 ! * * * * * * * * * * * * * * * *
1230 PRINTAT(7,12);"* REGULA FALSI *":Z=1
1240 GOSUB20
1250 GOSUB2030
1260 IF X=1 THEN RETURN
1270 LET C=(A*BY-B*AY)/(BY-AY)
1280 LET C2=C:LETZ2=Z
1290 IF ABS(FNY(C))<F THEN RETURN
1300 GOSUB2200
1310 GOTO1260
1400 ! * * * * * * * * * * * * * * * *
1410 ! UP NEWTONVERFAHREN
1420 ! * * * * * * * * * * * * * * * *
1430 PRINTAT(7,10);"* NEWTONVERFAHREN *":LET D=1.7014E+38:Z=1
1440 IF H=1 THEN1500
1450 WINDOW 10,23,0,39:CLS:PRINT"Geben Sie die 1.Ableitung Ihrer Funk-"
1460 PRINT:PRINT" tion in der Form Y=f(X) ein !":H=1:M=1126
1470 WINDOW 16,23,0,39:CLS:INPUT"Y=";E$
1480 IF LEN(E$)>58 THEN PRINTAT(19,11);X$;BEEP:PAUSE20:GOTO1470
1490 GOSUB110
1500 GOSUB20
1510 WINDOW 10,23,0,39:CLS
1520 GOSUB40
1530 CLS:INPUT" 1.NAEHERUNG:";A:PRINT:D=1.7014E+38:Z=1
1540 INPUT" FEHLER:";F
1550 LET C=A-FNY(A)/FNY1(A)
1560 LET C3=C:LET Z3=Z
1570 LET CY=FNY(C)
1580 IF ABS(CY)<F THEN RETURN
1590 IF ABS(D)<ABS(A-C) THEN1600:ELSE GOTO1610
1600 CLS:PRINTAT(17,4);"DIE NAEHERUNG IST ZU SCHLECHT!":BEEP:PAUSE20:GOTO1510
```

```
1610 LET D=ABS(C-A)
1620 WINDOW 17,22,5,39:CLS
1630 PRINT"X=";C;"   Y=";CY:LET A=C
1640 LET Z=Z+1:PRINTAT(22,20);"ITERATION:",Z
1650 PRINTAT(23,3);"'CONT'":PAUSE
1660 GOTO1550
1700 ! * * * * * * * * * * * * * * * *
1710 ! UP EINFACHE ITERATION
1720 ! * * * * * * * * * * * * * * * *
1730 PRINTAT(7, 9);"* EINFACHE ITERATION *":IF G=1 THEN1790
1740 WINDOW 10,23,0,39:CLS:PRINT "Geben Sie Ihre nach X umgestellte Funk-"
1750 PRINT:PRINT" tion in der Form X=f(X) ein !":M=1205:G=1
1760 WINDOW 16,18,0,39:CLS:INPUT"X=";F$
1770 IF LEN(F$)>58 THEN PRINTAT(19,11);X$:BEEP:PAUSE20:GOTO1760
1780 DOSUB110
1790 WINDOW 10,23,0,39:CLS:INPUT" 1.NAEHERUNG: ";A
1800 PRINT:INPUT" FEHLER:";F
1810 D=1.70141E+38:Z=1
1820 CY=0:GOSUB20
1830 GOSUB60
1840 LET C=FNI(A)
1850 LET CY=FNY(C)
1860 LET C4=C:LETZ4=2
1870 IF ABS(CY)<F THEN RETURN
1880 IF ABS(D)<ABS(A-C) THEN1890:ELSE GOTO1900
1890 CLS:PRINTAT(17,4);"DIE NAEHERUNG IST ZU SCHLECHT!":BEEP:PAUSE20:GOTO1790
1900 D=A-C
1910 WINDOW 17,23,5,39:CLS
1920 PRINT"X=";C,"Y=";CY:A=C
1930 LET Z=Z+1:PRINTAT(22,20);"ITERATION:",Z
1940 PRINTAT(23,3);"'CONT'":PAUSE
1950 GOTO1820
2000 ! * * * * * * * * * * * * * * * *
2010 ! UPs ZU INTHALB UND REGULA-FALSI
2020 !'* * * * * * * * * * * * * * * *
2030 CLS:INPUT" 1.NAEHERUNG:";A:INPUT" 2.NAEHERUNG:";B:PRINT
2040 INPUT" FEHLER:";F
2050 LET AY=FNY(A):LET BY=FNY(B)
2060 IF AY=0 THEN LET CY=0:LET C=A:LET X=1
2070 IF BY=0 THEN LET CY=0:LET C=B:LET X=1
2080 IF AY*BY>0 THEN2090:ELSE GOTO2100
2090 PRINTAT(15,7);"KEINE BERECHNUNG MOEGLICH !":BEEP:PAUSE9:GOTO2030
2100 RETURN
2200 IF C=A OR C=B THEN2080
2210 AY=FNY(A):BY=FNY(B):CY=FNY(C)
2220 IF CY*AY<0 THEN LET B=C:LET BY=CY:GOTO2240
2230 LET AY=CY:LET A=C
2240 WINDOW 17,22,5,39:CLS
2250 PRINT"X(1)=";A;"   Y(1)=";AY:PRINT
2260 PRINT"X(2)=";B;"   Y(2)=";BY:LET Z=Z+1
2270 PRINTAT(22,20);"ITERATION",Z
2280 PRINTAT(23,3);"'CONT'":PAUSE
2290 RETURN
OK
```

Bild 1. Programm zur Nullstellenbestimmung

Der Wert von x_3 wird nach der Formel

$$x_3 = \frac{x_1 f(x_2) - x_2 f(x_1)}{f(x_2) - f(x_1)}$$

berechnet. Das Intervall (x_i, x_3) wird auf die gleiche Art verbessert. Das Verfahren wird so lange wiederholt, bis das Intervall (x_k, x_n) die exakte Lösung mit der gewünschten Genauigkeit einschließt. Das Anfangsintervall kann

bei diesen beiden Verfahren beliebig groß sein, wenn die Stetigkeit der Funktion in (x_1, x_2) gewährleistet ist und $f(x_1)$ und $f(x_2)$ verschiedene Vorzeichen besitzen. Unter diesen Voraussetzungen konvergiert das Verfahren immer gegen eine Nullstelle der Funktion.

2.2. Newton-Verfahren

Das NEWTON-Verfahren und das einfache Iterationsverfahren gehen von einer Anfangsnäherung x_1 aus, die schrittweise verbessert wird.
Beim NEWTON-Verfahren wird die Funktion durch die Tangente im Punkt x_1 approximiert. Der Schnittpunkt der Tangente mit der x-Achse liefert die verbesserte Iterationslösung. Die Iterationsvorschrift dazu liefert folgende Gleichung

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (i = 1, 2, \ldots).$$

Aus dieser Gleichung geht hervor, daß von der Funktion $f(x)$ die 1. Ableitung $f'(x)$ existieren muß, und in der Umgebung der exakten Nullstelle x_0 muß $f'(x_i)$ verschieden von Null sein. Die Konvergenzbedingung für dieses Verfahren liefert die folgende Ungleichung

$$\left| \frac{f(x_0)\, f''(x_0)}{f'(x_0)^2} \right| < 1.$$

Hierbei ist $f'(x)$ die erste Ableitung und $f''(x)$ die zweite Ableitung von $f(x)$ mit $f'(x_0)$ verschieden von Null. Die Konvergenz des Verfahrens ist abhängig vom Startwert. Ist dieser gut, konvergiert das Verfahren schnell.

2.3. Einfaches Iterationsverfahren

Beim einfachen Iterationsverfahren wird die Gleichung $f(x) = 0$ in eine Fixpunktform $x = g(x)$ gebracht. Hieraus ergibt sich eine Iterationsvorschrift

$$x_{i+1} = g(x_i) \quad (i = 1, 2, \ldots),$$

wobei x_1 eine gegebene Anfangsnäherung ist.
Die nach der Iteration berechneten x_i konvergieren nur dann gegen die Nullstelle x_0 von $f(x)$, wenn in dem betrachteten Näherungsintervall für alle x einer Umgebung von x_0

$$g'(x) < 1$$

gilt. Die Konvergenz des Verfahrens hängt somit von der gewählten Iterationsvorschrift und vom Startpunkt ab. Das Erfülltsein der Konvergenzbedingung für die Anfangsnäherung ist hinreichend für die Konvergenz der Iteration. Wird sie für den Anfangswert nicht erfüllt, ist zu testen, ob die Intervalle (x_i, x_{i+1}) nach jeder Iteration kleiner werden. Ist das nicht der Fall, ist das Verfahren abzubrechen und gegebenenfalls mit einem neuen Startwert zu beginnen.

Für alle Verfahren gilt:
Es wird immer nur eine Nullstelle von $f(x)$ berechnet. Besitzt die Funktion mehrere Nullstellen im vorgegebenen Intervall, so können diese mit geänderter Anfangsnäherung erhalten werden. Einen Überblick darüber, ob mehrere Nullstellen vorliegen, kann man sich durch Funktionswerteberechnung verschaffen. Auch das wird dem Nutzer zu Beginn des Programms angeboten.

3. Programmbeschreibung und Nutzerhinweise

Die vier beschriebenen Näherungsverfahren wurden im nachfolgend abgedruckten Programm realisiert. Der Nutzer wird an Hand eines Dialogs durch das Programm geführt. Die Eingabe der Funktionen erfolgt ohne Unterbrechung des Programms. Die Funktionen werden als Zeichenketten eingegeben und zeichenweise in den Programmspeicher übertragen. Dies er-

klärt die geforderten Leerzeichen der Zeilen 20, 40 und 60. Die Funktion ist in der üblichen Syntax wie eine normale Nutzerfunktion einzugeben. Als Veränderliche darf sie nur x enthalten. Fehlerhafte Eingaben werden nicht erkannt und führen zu falschen Funktionen im Programmspeicher. Die Funktion darf nicht mehr als 58 Zeichen umfassen, sonst erscheint ein Fehlerausdruck. Das Programm ist nur auf einem Rechner mit BASIC-Interpretermodul verwendbar. Es benötigt einen bestimmten Speicherbereich im Arbeitsspeicher, deshalb ist vor dem Einlesen des Programms der Befehl NEW auszuführen. Die ersten 7 Programmzeilen sind exakt mit den angegebenen Leerzeichen einzugeben. Vor dem Start ist es günstig zu überprüfen, ob die Speicherzellen 1045, 1126 und 1205 mit 180 belegt sind. Das erfolgt durch PEEK(Adresse) mit den Adressen 1045, 1126 und 1205. Bei falscher Belegung ist das Programm nicht lauffähig. Änderungen des Programms sind erst ab Zeile 80 möglich, da sich sonst der Speicherplatz der Nutzerfunktionen verschiebt.

Nach dem Start des Programms erscheint die Aufforderung zur Eingabe der Funktion. Die Eingabe wird mit ENTER beendet. Auf Wunsch kann sich der Nutzer eine Wertetabelle für diese Funktion aufstellen.

Das gewünschte Verfahren wird über ein Menü ausgewählt.

Nach Auswahl des Verfahrens werden die vom jeweiligen Verfahren benötigten weiteren Eingaben gefordert.

Beim Intervallhalbierungsverfahren und der Regula falsi sind das die Intervallgrenzen und die zugelassene Abweichung des Funktionswertes von Null (Fehler).

Das NEWTON-Verfahren fordert die Eingabe der ersten Ableitung der Funktion, der Anfangsnäherung und des Fehlers.

Das einfache Iterationsverfahren fordert die Eingabe der nach x umgestellten Nutzerfunktion, der Anfangsnäherung und des Fehlers. Bei allen Verfahren werden die Ergebnisse jedes Iterationsschrittes und die Anzahl der ausgeführten Iterationen ausgegeben. Das Programm kann jeweils mit CONT fortgesetzt werden. Falls das Verfahren zu langsam konvergiert, kann es mit STOP unterbrochen werden und durch GOTO 490 neu gestartet werden.

Ist der Funktionswert der laufenden Iteration betragsmäßig kleiner als der Fehler, wird die Rechnung beendet und das Ergebnis ausgegeben. Das Programm wird mit CONT fortgesetzt. Es erscheint dann eine Tabelle, die die Ergebnisse und die Anzahl der Iterationen der vier Verfahren zum Vergleich nebeneinander ausgibt. Dieser Vergleich ist natürlich nur sinnvoll bei gleichen Anfangsnäherungen und gleichem Fehler für alle Verfahren.

```
150 LET L=13442
160 LET L=L+1:LET M=M+1
170 IF L=13501 THEN RETURN
180 LET K=VPEEK(L)
190 IF K>=48 AND K<=57 THEN POKE M,K:GOTO160
200 IF (K>31ANDK<64)OR K=94 OR K=88 OR K=255 THEN POKE M,X(K):GOTO160
210 IF K=76 OR K=80 THEN POKE M,X(K):M=M+1:L=L+2:GOTO180
220 IF K>64 AND K<84 THEN POKE M,X(K+VPEEK(L+1)):L=L+3:M=M+1:GOTO180
225 POKE M,32:GOTO160
```

Bild 2. Änderung für KC 85/2/3

Die weitere Fortsetzung erfolgt wieder mit CONT. Danach besteht die Möglichkeit, eine neue Rechnung durchzuführen, wobei auf Wunsch eine neue Funktion eingegeben bzw. die Anfangsnäherung geändert werden kann. Wird die Funktion beibehalten, werden die jeweils letzten Ergebnisse jedes Verfahrens gespeichert und beim Vergleich ausgegeben.
Beim Einlesen einer neuen Funktion werden diese gelöscht. Mit geringen Änderungen, die sich auf die Abspeicherung der Funktionen beziehen, ist das Programm auf dem KC 85/2/3 lauffähig (Bild 2).

TESTBEISPIEL:

$$f(x) = x^2 + x \cdot \ln(x + 2) - 3 = 0$$
$$f'(x) = 2x + \ln(x + 2) + x/(x + 2)$$
$$x = \varphi(x) = \sqrt{3 - x \cdot \ln(x + 2)}$$

1. Intervallhalbierung:
Fehler 0.01, Anfangsintervall (1, 2)

Iteration	Einschließung der Nullstelle	Intervall der Funktionswerte
1	(1, 2)	(— 0.9014, 3.7726)
2	(1, 1.5)	(— 0.9014, 1.129)
3	(1, 1.25)	(— 0.9014, 0.03582)
4	(1.125, 1.25)	(— 0.4525, 0.03582)
5	(1.1875, 1.25)	(— 0.21325, 0.03582)
6	(1.21875, 1.25)	(— 0.08994, 0.03582)
7	(1.23438, 1.25)	(— 0.02736, 0.03582)

Lösung: 1.24219

2. Regula falsi:
Fehler 0.01, Anfangsintervall (1, 2)

Iteration	Einschließung der Nullstelle	Intervall der Funktionswerte
1	(1, 2)	(— 0.9014, 3.7726)
2	(1.19285, 2)	(— 0.19230, 3.7726)
3	(1.232, 2)	(— 0.036914, 3.7726)

Lösung: 1.24944

3. Newton-Verfahren:
Fehler 0.01, Anfangsnäherung 1

Iteration	Näherungslösung	Funktionswert
1	1	— 0.9014
2	1.26265	0.0874

Lösung: 1.2413

4. Gewöhnliche Iteration:
Fehler 0.01; Anfangsnäherung 1

Iteration	Näherungslösung	Funktionswert
1	1	— 0.9014
2	1.37891	0.580282
3	1.14939	— 0.360297
4	1.29669	0.228249
5	1.20547	— 0.142644
6	1.26325	0.089876
7	1.22716	— 0.056329
8	1.2499	0.035419
9	1.23565	— 0.022225
10	1.24461	0.013965

Lösung: 1.23899

4. Schlußwort

Vorliegende Arbeit entstand im Rahmen einer Jahresarbeit an den Spezialklassen der Sektion Chemie der Technischen Hochschule Leuna-Merseburg. Das Programm zur Nullstellenbestimmung ist sehr dialogfreundlich gestaltet. Beim Nutzer werden geringe fachliche Kenntnisse vorausgesetzt. Das Programm ist gut zum Einsatz in der studentischen Ausbildung geeignet.
Die vorliegende Fassung des Programms läuft auf dem KC 85/1 und KC 87 sowie mit angegebenen Änderungen auf dem KC 85/2/3.

Autoren:

Dr. Christine Lipp
Steffen Vieweg
Sektion Mathematik der Technischen Hochschule Leuna-Merseburg

TESY – eine Rechnerkopplung mit dem Kleincomputer KC 85/2

Seit einiger Zeit entstehen in fast allen Ausbildungsbereichen unserer Republik Computerkabinette für die Aus- und Weiterbildung auf dem Gebiet der Informatik und der Mikrorechentechnik. Die Mehrzahl dieser Kabinette werden vorrangig mit Kleincomputern ausgerüstet.
Zur Verbesserung der Fachmethodik und zur Effektivierung des Unterrichtes vor allem im Fach Informatik wurde an der Militärtechnischen Schule der Luftstreitkräfte/Luftverteidigung in Bad Düben durch ein Jugendneuererkollektiv eine Rechnerkopplung entwickelt. Diese Kopplung wurde auf der 29. Zentralen Messe der Meister von Morgen in Leipzig vorgestellt.

Was ist TESY?

TESY (teaching-system) ist ein System miteinander gekoppelter Kleincomputer KC 85/2. Alle in unserem Lehrkabinett eingesetzten Kleincomputer sind durch eine modulare Hardware-Erweiterung in Sternstruktur miteinander gekoppelt (Bild 1).
Das System wird an unserer Lehreinrichtung mit einem Lehrerrechner (M) und 9 Schülerrechnern (S) betrieben. Es ist auf insgesamt 16 Schülerrechner erweiterbar.

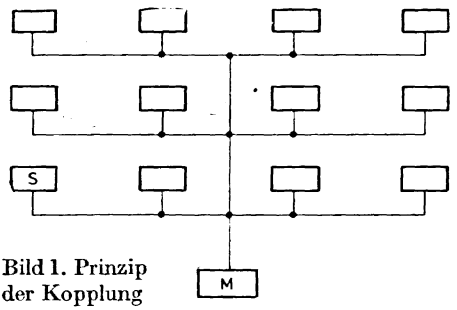


Bild 1. Prinzip der Kopplung

Beschreibung des Systems

Mit diesem System wurde die Möglichkeit des Datentransfers im Rechnerverbund des Lehrkabinettes geschaffen. Veränderungen an der Hardware der Kleincomputer waren nicht notwendig. Die Funktion des Systems basiert auf dem Master-Slave-Prinzip. Das bedeutet in unserem Fall, daß nur der Fachlehrer die Möglichkeit hat, den Datentransfer auszulösen. Auf die einzelnen Arbeitsweisen wird im weiteren noch näher eingegangen.
TESY besteht im wesentlichen aus zwei verschiedenen Steuermodulen, dem Modul für den Lehrerrechner (Master) und den Modulen für die Schülerarbeitsplätze (Slave). Kernstück beider Modulvarianten bilden der programmierbare Parallel-Ein-/Ausgabe-Schaltkreis UA 855 D sowie die elektrisch programmierbaren Festwertspeicher-Schaltkreise U 2716 C für den

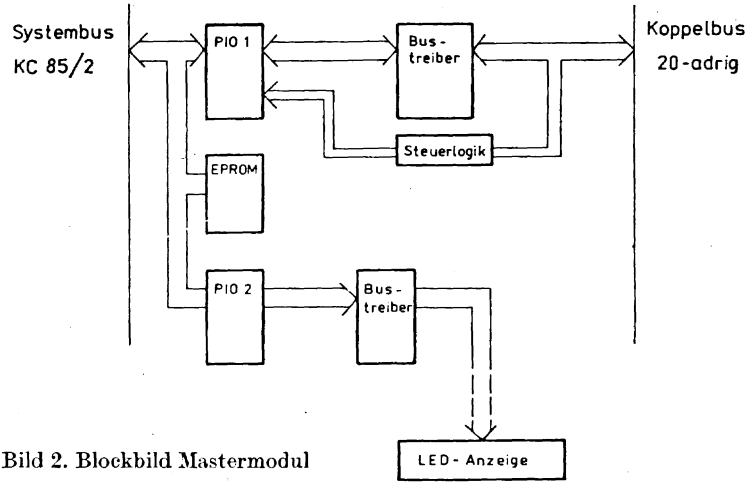


Bild 2. Blockbild Mastermodul

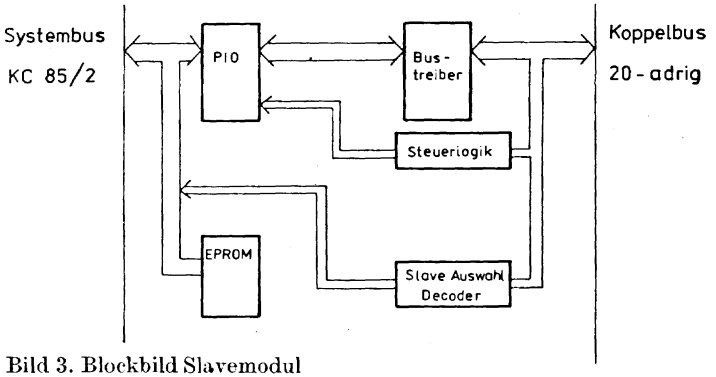


Bild 3. Blockbild Slavemodul

Master und die U 555 C für die Slaves. In Bild 2 und 3 sind die Blockbilder beider Modulvarianten ersichtlich. Die Ankopplung der Module an die Rechner erfolgt über den jeweiligen Modulschacht C. Dabei wird für den Master der Systembus des KC 85/2 in ein sogenanntes Mastergehäuse geführt, in dem neben dem Mastermodul auch die LED-Anzeige zur Kontrolle der Aktivierung der angeschlossenen Schülerrechner untergebracht ist (Bild 4). Die Steuerung des Systems erfolgt über vier der freiprogrammierbaren Funktionstasten (F1 bis F4) der Lehrerrechnertastatur.

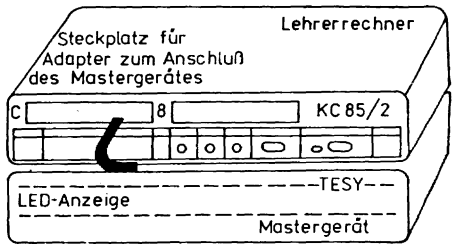


Bild 4. Ansicht Lehrerrechner

Arbeitsweisen des Systems

Wie bereits beschrieben, wird mit TESY ein Datentransfer zwischen Lehrer- und Schülerrechner realisiert.

Es sind zwei prinzipielle Arbeitsweisen möglich.

1. Kontrolle der Schülerarbeit (Datentransfer Schülerrechner zum Lehrerrechner) und
2. Lehrerdemonstration (Datentransfer Lehrerrechner zu einem, mehreren oder allen Schülerrechnern)

Durch eine Kombination dieser Arbeitsweisen ist ein Datentransfer zwischen den Schülerrechnern möglich.
Beim erstmaligen Start des Systems (Aufruf von TESY aus dem Menübild des Lehrerrechners) werden zunächst die vorher, entsprechend einer Steuerroutine von TESY für die Schülerrechner, eingegebenen Bedienernamen (Schülernamen) der angeschlossenen Schülerrechner abgefragt. Sind alle Rechner abgefragt, kann die eigentliche Arbeit mit der Rechnerkopplung beginnen.
Beispiel des Startmenüs am Lehrerrechner:

```
Rechnerkopplung TESY
SLAVE 1  Müller     F1: CLS
SLAVE 2  Schulze    F2: Kontrolle
SLAVE 3  Lehmann    F3: DEMO-MODUS
   :                F4: Demonstration
   :                   — ENTER —
SLAVE 9  Anton
```

Arbeitsweise Schülerkontrolle

Im Verlauf der Ausbildung macht es sich oft notwendig, den Stand der momentanen Schülerarbeit zu überprüfen. Ohne Rechnerkopplung kann bekanntlich der Fachlehrer die Schülerarbeit nur am jeweiligen Arbeitsplatz überprüfen. Mit Hilfe von TESY ist es nun dem Fachlehrer möglich, die Schülerarbeit von seinem Arbeitsplatz aus zu kontrollieren. Ausgelöst wird dieser Vorgang durch Betätigung der Funktionstaste F2 der Lehrertastatur. Der Fachlehrer kann sich einen Schülerplatz durch Eingabe der Slavenummer auswählen und anschließend die gewünschte Arbeitsweise einstellen.
Prinzipiell handelt es sich bei beiden Arbeitsweisen um einen Speichertransfer. In der Arbeitsweise Bildtransfer wird der Datenaustausch nach Betätigung der ENTER-Taste ausgelöst und der Bildinhalt des Schülerrechners zum Lehrerrechner übertragen.
Bei Benutzung der Arbeitsweise Speichertransfer ist es erforderlich, die Anfangs-, End- und Zieladresse des Transfers einzutragen. In dieser Arbeitsweise können damit der gesamte oder Teile des Speicherbereichs des jeweiligen Schülerrechners zum Lehrerrechner übertragen und die erarbeiteten Programme der Schüler durch den Lehrer anschließend geprüft, korrigiert bzw. getestet werden. In beiden Arbeitsweisen wird die Arbeit der Schüler für die Zeitdauer der Datenübertragung unterbrochen. Laufende Programme setzen ihre Arbeit anschließend fort.

Beispiel für die Arbeitsweise KONTROLLE:

```
Kontrollmodus
SLAVE 1
Arbeitsweise: S      B: Bildtransfer
      AAT 0200       S: Speichertransfer
      EAT 3FFF
      ZAT 0200
```

Arbeitsweise Lehrerdemonstration

In dieser Arbeitsweise hat der Fachlehrer die Möglichkeit, Bildinhalte oder Programmteile an einzelne, mehrere oder alle zugeschalteten Rechner zu senden. Dieser Vorgang wird durch Betätigung der Funktionstaste F3 ausgelöst. Der Fachlehrer gibt wiederum die Nummer(n) der(s) jeweiligen Schülerplatzes ein. Anschließend hat er auch hier die Wahl zwischen den Arbeitsweisen Bild- bzw. Speichertransfer. In der Arbeitsweise Speichertransfer ist durch den Fachlehrer wieder die

Anfangs-, End- und Zieladresse des Transfers einzugeben. Nach Betätigung der ENTER-Taste wird der Bildschirm des Lehrerrechners wieder in den Ausgangszustand versetzt. Ausgelöst wird der Transfer erst durch Betätigung der Funktionstaste F4. Der oder die angewählten Schülerrechner erhalten nun das momentane Bild bzw. das Programm des Lehrerrechners.
Laufende Schülerprogramme werden in der Arbeitsweise Bildtransfer nicht gestört, sondern wiederum nur unterbrochen. In der Arbeitsweise Speichertransfer können Schülerprogramme vom Lehrer überschrieben und damit verändert oder gar zerstört werden. Bei der Arbeit mit TESY muß das entsprechend berücksichtigt werden.

Beispiel Arbeitsweise Lehrerdemonstration:

```
DEMO-MODUS
SLAVE   01 02 04 07
MODUS   S          B: Bildtransfer
  AAT   0200       S: Speichertransfer
  EAT   3FFF
  ZAT   0200
```

Schlußbemerkungen

Das System TESY wird seit September 1986 fast täglich in der Informatikausbildung eingesetzt. Nach einer kurzen Einarbeitungszeit sind Lehrer und Schüler auf die Arbeitsweisen des Systems eingestellt. Es kann eingeschätzt werden, daß bei guter organisatorisch-methodischer Vorbereitung des Fachlehrers auf den Unterricht sowie bei störungsfreier Funktion des Systems eine hohe Effektivität der Ausbildung zu verzeichnen ist. Auch in anderen Fächern ist der Einsatz von TESY denkbar.
Da zum Zeitpunkt der Entwicklung dieser Rechnerkopplung nur der KC 85/2 zur Verfügung stand, wurde die Kopplung hard- und softwaremäßig auf diesen Typ festgelegt. Die Anwendung für den KC 85/3 erfordert Veränderungen in der Modulhardware und Steuersoftware.
Der Bau der hier vorgestellten Rechnerkopplung wurde in weniger als einem halben Jahr realisiert. Die Entwickler besaßen die notwendigen Kenntnisse auf dem Gebiet der Informatik und hatten die entsprechenden materiellen Voraussetzungen zur Verfügung.

Autoren:
Dipl.-Ing. Thomas Magerl
VEB Nachrichtenelektronik
„Ernst Thälmann"
Arnstadt

Ing. Dieter Borchardt
Militärtechnische Schule der LSK/LV
„Harry Kuhn"

Polynomgrafik

Jeder Schüler der 11. Klasse wird verstärkt mit Kurvendiskussionen geplagt. Um meinen Leidensgefährten in dieser mißlichen Situation zu helfen, habe ich den KC 85/3 bemüht und ein Programm geschrieben, das jeden Mathematiklehrer befriedigen müßte, wenn er sich mit Polynomen zufrieden gibt. Die gesuchte Kurve und alle Extras erscheinen auf dem Bildschirm!

In der Kurvendiskussion dieser Polynome spielt die Ermittlung der Nullstellen eine wichtige Rolle. Mit erheblichem Programmieraufwand lassen sich noch Gleichungen 5. Grades exakt lösen; darüber hinaus führen nur noch Näherungsmethoden zu einer Nullstellenbestimmung. In der Regel wird eine solche Gleichung nicht nur reelle, sondern auch komplexe Nullstellen besitzen. Ein Großteil der bekannten Näherungsverfahren (Newton, Regula falsi) scheitert an dieser Klippe. Doch das Verfahren von BAIRSTOW (und HICHCOCK) berechnet sowohl konjugiert komplexe, als auch reelle Nullstellen eines Polynoms iterativ. Es basiert auf der Abspaltung jeweils eines quadratischen Terms von dem ursprünglichen Polynom. Dadurch werden stets 2 Nullstellen mit einem Mal bestimmt.

In meinem Programm verwende ich eine spezielle Version des oben genannten BAIRSTOW-Verfahrens. Aber wie jede iterative Rechenvorschrift, ist auch die jetzige nicht unfehlbar. Das heißt, es kann durchaus geschehen, daß der Algorithmus nicht konvergiert. Deshalb habe ich eine Abbruchmöglichkeit eingebaut, um den Algorithmus vorzeitig zu beenden.

Neben diesem Hauptteil des Programmes besteht noch die Möglichkeit, die Extremstellen und die Wendepunkte zu berechnen, wobei wieder der Nullstellenalgorithmus angewendet wird (nur auf die entsprechende Ableitung, die gleichfalls ein Polynom ist). Alle Ableitungen können auch extra berechnet werden, ebenso wie einzelne Funktionswerte, um sich einen Überblick über den Funktionsverlauf zu verschaffen. Der Rechner gibt auf Wunsch die Tangentengleichung für einen beliebigen Abszissenwert aus.

Als krönender Abschluß kann das Bild der nun weitgehend analysierten Funktion auch noch gezeichnet werden.

Testbeispiele

Bevor wir uns dem Programm selbst zuwenden, soll erst einmal ein Eindruck von dessen Leistungsfähigkeit vermittelt werden.

Beispiel 1

$$f(x) = x^5 - 4{,}25\,x^4 + 4{,}125\,x^3 + 1{,}0625\,x^2 - 1{,}09375\,x$$

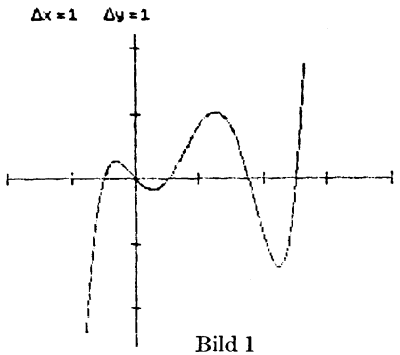


Bild 1

NULLSTELLEN (Bild 1)

$x(1) = 0$
$x(2) = -0{,}5$
$x(3) = 1{,}75$
$x(4) = 0{,}5$
$x(5) = 2{,}5$

EXTREMWERTE

$f(\ 0{,}256455) = -0{,}158317$ – min
$f(-0{,}310961) = 0{,}276173$ – max
$f(\ 2{,}217520) = -1{,}366880$ – min
$f(\ 1{,}236980) = 1{,}026020$ – max

WENDEPUNKTE

$f(\ 1{,}8500600) = -0{,}3817290$
$f(-0{,}0741876) = 0{,}0851752$
$f(\ 0{,}7741270) = 0{,}4553860$

Beispiel 2

$f(x) = x^7 + 3x^6 - 14x^5 - 40x^4 + 39x^3 + 157x^2 - 26x - 120$

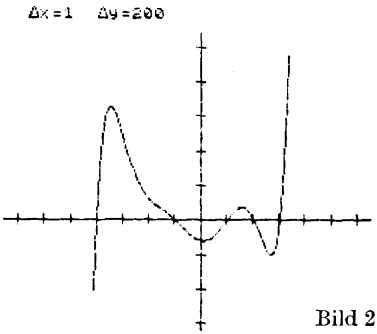


Bild 2

NULLSTELLEN (Bild 2)

$x(1) = 1$
$x(2) = -1$
$x(3) = 3$
$x(4) = -4$
$x(5) = -2 + i*1$
$x(6) = -2 - i*1$
$x(7) = 2$

EXTREMWERTE

$f(\ 2{,}6608700) = -206{,}259$ – min
$f(-3{,}4722700) = 654{,}617$ – max
$f(\ 1{,}5500700) = 69{,}0868$ – max
$f(\ 0{,}0806559) = -121{,}057$ – min

WENDEPUNKTE

$f(\ 0{,}916910) = -16{,}8005$
$f(-0{,}763086) = -35{,}5670$
$f(\ 2{,}264890) = -96{,}6735$
$f(-2{,}976660) = 467{,}477$
$f(-1{,}584910) = 70{,}3585$

BASIC-Programm

Mein Programm meldet sich nach einer kurzen Initialisierungsphase zur Generierung der Sonderzeichen mit der ÜBERSICHT (Bild 3).

```
POLYNOMGRAFIK (ÜBERSICHT)

Wählen Sie durch Druck der entsprechen-
den Zifferntaste eine der folgenden zehn
Möglichkeiten aus:

1 ALLGEMEINE INSTRUKTIONEN

2 EINGABE DES POLYNOMS

3 EINZELNE FUNKTIONSWERTE

4 NULLSTELLEN

5 ABLEITUNGEN

6 EXTREMWERTE

7 WENDEPUNKTE

8 GRAFISCHE DARSTELLUNG

9 TANGENTENGLEICHUNGEN

0 ENDE
```

Bild 3

```
ALLGEMEINE INSTRUKTIONEN

Durch dieses Programm werden Polynome
bis einschließlich 10. Grades analysiert
und grafisch dargestellt.

Einige Hinweise zur Benutzung:

Alle Eingaben sind durch 'ENTER' (↵)
abzuschließen.

Wenn man nur 'ENTER' drückt, gelangt man
stets zur ÜBERSICHT zurück.

In den Punkten 4,6 und 7 werden Itera-
tionsverfahren verwendet. Um diese vor-
zeitig abzubrechen (keine Konvergenz),
ist 'A' (Abbruch) - bis Beep ertönt - zu
drücken.
Die Genauigkeit des (Abbruch)Ergebnisses
liegt in der Größenordnung der Zahl, die
unten gerade angezeigt wird. Sie sollte
gegen Null gehen.

Bevor die Funktion grafisch dargestellt
werden kann (Pkt. 8), müssen die Null-
stellen (Pkt. 4) und die Extremwerte
(Pkt. 6) bestimmt werden, da Sie bei der
Skalierung erforderlich sind.
Ruft man dennoch Pkt. 8 zuerst auf, wird
automatisch zum Pkt. 4 bzw. 6 umge-
schaltet.
```

Bild 4

Der Nutzer wird zunächst einmal Hinweise erwarten, wie er mit dem Programm arbeiten soll. Diese können unter Punkt 1 abgerufen werden (vgl. Bild 4).

Das BASIC-Programm besitzt folgende Struktur:

Programmzeile	Programmteil
0 — 2	Name, Version
10 — 70	Initialisierung
100 — 290	ÜBERSICHT
1000 — 1320	Allgemeine Instruktionen
2000 — 2310	Eingabe des Polynoms
3000 — 3130	Einzelne Funktionswerte
4000 — 4190	Nullstellen
5000 — 5120	Ableitungen
6000 — 6280	Extremwerte
7000 — 7260	Wendepunkte
8000 — 8660	Grafische Darstellung
9000 — 9270	Tangentengleichungen
10000 — 10060	Ende
20000 — 20250	Grafikzeichen
21000 — 21040	Zurück zur ÜBERSICHT
22000 — 22080	Eingabetest
23000 — 23180	Polynome ausgeben
24000 — 24090	HORNER-Schema
25000 — 25540	BAIRSTOW

Ich verzichte hier darauf, die einzelnen Programmteile zu erläutern, obwohl es z. B. sehr schwierig ist, einen günstigen Ausschnitt für die grafische Darstellung zu finden und das Achsenkreuz mit einem passenden Maßstab zu versehen.
Dafür gebe ich nun das vollständige – 10 KBytes umfassende – Listing an (S. 44 ff.).

Autor:
Uwe Girlich
Leipzig

```
0 REM **Polynomgrafik****(C) U.Girlich**
1 REM VERSION 1.0
2 REM 20.3.87
10 !INITIALISIERUNG
20 WINDOW0,31,0,39:COLOR7,0:CLS
30 IFPEEK(0)<>102THENGOSUB20000
40 F1=0:F2=0:F3=0:F4=0:F5=0
50 E0=1E-7:E1=1E-10:E2=1E-37
60 DIML(2,10):DIMP(320):DIMW(3,9):DIMZ(2,10)
70 DIMA(11,11):DIMB(14):DIMC(14):DIME(3,10):DIMF(11)
100 !UEBERSICHT
110 WINDOW0,31,0,39:COLOR7,O:CLS
120 PRINTAT(2,7);"POLYNOMGRAFIK  (";CHR$(163);"BERSICHT)"
130 PRINTAT(5,0);"W";CHR$(160);"hlen Sie durch Druck der  entspr
echen-"
140 PRINTAT(6,0);"den Zifferntaste eine der folgenden zehn"
150 PRINTAT(7,0);"M";CHR$(162);"glichkeiten aus:"
160 PRINTAT(10,0);"1 ALLGEMEINE INSTRUKTIONEN"
170 PRINTAT(12,0);"2 EINGABE DES POLYNOMS"
180 PRINTAT(14,0);"3 EINZELNE FUNKTIONSWERTE"
190 PRINTAT(16,0);"4 NULLSTELLEN"
200 PRINTAT(18,0);"5 ABLEITUNGEN"
210 PRINTAT(20,0);"6 EXTREMWERTE"
220 PRINTAT(22,0);"7 WENDEPUNKTE"
230 PRINTAT(24,0);"8 GRAFISCHE DARSTELLUNG"
240 PRINTAT(26,0);"9 TANGENTENGLEICHUNGEN"
250 PRINTAT(28,0);"0 ENDE"
260 T$=INKEY$
270 IF T$<"0"ORT$>"9"THEN260
280 ONVAL(T$)GOTO1000,2000,3000,4000,5000,6000,7000,8000,9000
290 GOTO10000
1000 !ALLGEMEINE INSTRUKTIONEN
1010 CLS:PRINTAT(2,8);"ALLGEMEINE INSTRUKTIONEN"
1020 PRINTAT(3,8);"_________________________"
1030 PRINTAT(5,0);"Durch dieses  Programm  werden  Polynome"
1040 PRINTAT(6,0);"bis einschlie";CHR$(164);"lich 10. Grades ana
lysiert"
1050 PRINTAT(7,0);"und grafisch dargestellt."
1060 PRINTAT(10,0);"Einige Hinweise zur Benutzung:"
1070 PRINTAT(13,0);"Alle  Eingaben  sind  durch  'ENTER' (";CHR$
(141);")"
1080 PRINTAT(14,0);"abzuschlie";CHR$(164);"en."
1090 PRINTAT(16,0);"Wenn man nur 'ENTER' dr";CHR$(163);"ckt, gel
angt man"
1100 PRINTAT(17,0);"stets zur ";CHR$(163);"BERSICHT zur";CHR$(16
3);"ck."
1110 PRINTAT(20,0);"In den Punkten 4,6 und 7  werden  Itera-"
1120 PRINTAT(21,0);"tionsverfahren verwendet. Um diese  vor-"
1130 PRINTAT(22,0);"zeitig abzubrechen  (keine  Konvergenz),"
1140 PRINTAT(23,0);"ist 'A' (Abbruch) - bis Beep ert";CHR$(162);
"nt - zu"
1150 PRINTAT(24,0);"dr";CHR$(163);"cken."
1160 PRINTAT(25,0);"Die Genauigkeit des (Abbruch)Ergebnisses"
1170 PRINTAT(26,0);"liegt in der Gr";CHR$(162);CHR$(164);"enordn
ung der Zahl"
1180 PRINTAT(26,35);", die"
1190 PRINTAT(27,0);"unten gerade angezeigt wird. Sie  sollte"
1200 PRINTAT(28,0);"gegen Null gehen."
1210 PRINtAT(30,32);"››ENTER"
1220 IFINKEY$<>CHR$(13)THEN1220
1230 WINDOW5,31,0,39:CLS
1240 PRINTAT(10,0);"Bevor die Funktion grafisch  dargestellt"
1250 PRINTAT(11,0);"werden kann (Pkt. 8), m";CHR$(163);"ssen  di
e  Null-"
```

```
1260 PRINTAT(12,0);"stellen  (Pkt. 4)  und  die  Extremwerte"
1270 PRINTAT(13,0);"(Pkt. 6) bestimmt werden, da sie bei der"
1280 PRINTAT(14,0);"Skalierung erforderlich sind."
1290 PRINTAT(15,0);"Ruft man dennoch Pkt. 8 zuerst auf, wird"
1300 PRINTAT(16,0);"automatisch zum  Pkt.  4  bzw.  6  umge-"
1310 PRINTAT(17,0);"schaltet."
1320 GOTO21000
2000 !EINGABE DES POLYNOMS
2010 CLS
2020 PRINTAT(2,9);"EINGABE DES POLYNOMS"
2030 PRINTAT(3,9);"_____________________"
2040 LOCATE6,0:T$=" ":INPUT"Grad des Polynoms:";T$
2050 GOSUB22000
2060 IFFE=1THEN2040
2070 IFFE=2THEN100
2080 N=INT(ABS(VAL(T$)))
2090 IFN=0ORN>10THEN2040
2100 PRINTAT(8,0);"Der H";CHR$(162);"chstkoeffizient ist a(";N;"
)."
2110 LOCATE10,0
2120 I=N
2130 PRINT"a(";I;")";:T$=" ":INPUT"=";T$
2140 GOSUB22000
2150 IFFE=1THENPRINTCHR$(11);:GOTO2130
2160 IFFE=2THENF1=0:GOTO100
2170 A(1,I+1)=VAL(T$)
2180 IFI=0THEN2220
2190 FORJ=2TON+1
2200 A(J,I)=I*A(J-1,I+1)
2210 NEXT
2220 IFI>0THENI=I-1:GOTO2130
2230 I=1:J=1
2240 IFABS(A(1,I))>E0THENJ=I
2250 IFI<N+1THENI=I+1:GOTO2240
2260 N=J-1
2270 IF N=0 THEN PRINT:PRINT:PRINT:PRINT"f(x)=";A(1,1):F1=0:GOTO
2300
2280 PRINT:PRINT:J=0:GOSUB23000
2290 F1=1
2300 F2=0:F3=0:F4=0:F5=0
2310 GOTO21000
3000 !EINZELNE FUNKTIONSWERTE
3010 IFF1=0THEN2000
3020 CLS:PRINTAT(2,8);"EINZELNE FUNKTIONSWERTE"
3030 PRINTAT(3,8);"________________________"
3040 LOCATE6,0:J=0:GOSUB23000:PRINT:PRINT:PRINT
3050 T$=" ":INPUT"x=";T$:GOSUB22000
3060 IFFE=1THENPRINTCHR$(11);:GOTO3050
3070 IFFE=2THEN100
3080 X=VAL(T$)
3090 PRINTCHR$(11);"f(";X;")=";
3100 J=0
3110 GOSUB24000
3120 PRINTY
3130 GOTO3050
4000 !NULLSTELLEN
4010 IFF1=0THEN2000
4020 CLS:PRINTAT(2,10);"NULLSTELLEN":PRINTAT(3,10);"__________"
4030 LOCATE6,0:J=0:GOSUB23000:PRINT:PRINT:PRINT
4040 IFF2=1THEN4090
4050 FORI=1TON+1:F(I)=A(1,I):NEXT
4060 M=N
4070 GOSUB25000
4080 FORI=1TON:Z(1,I)=L(1,I):Z(2,I)=L(2,I):NEXT
```

```
4090 FORI=1TON
4100 PRINT"x<";I;">=";Z(1,I);
4110 IFZ(2,I)=0THEN4160
4120 IFZ(2,I)>0THENPRINT"+";
4130 IFZ(2,I)<0THENPRINT"-";
4140 PRINT"1";
4150 IFABS(ABS(Z(2,I))-1)>E0THENPRINT"*";ABS(Z(2,I));
4160 PRINT
4170 NEXT
4180 F2=1
4190 GOTO21000
5000 !ABLEITUNGEN
5010 IFF1=0THEN2000
5020 CLS:PRINTAT(2,14);"ABLEITUNGEN"
5030 PRINTAT(3,14);"___________"
5040 LOCATE6,0:J=0:GOSUB23000:PRINT:PRINT:PRINT
5050 T$=" ":PRINT"Ordnung der Ableitung:";:INPUT"";T$
5060 GOSUB22000
5070 IFFE=1THENPRINTCHR$(11);:GOTO5050
5080 IFFE=2THEN100
5090 J=INT(ABS(VAL(T$))):IFJ=0THENPRINTCHR$(11);:GOTO5050
5100 GOSUB23000:PRINT:PRINT
5120 GOTO5050
6000 !EXTREMWERTE
6010 IFF1=0THEN2000
6020 CLS:PRINTAT(2,14);"EXTREMWERTE"
6030 PRINTAT(3,14);"___________"
6040 LOCATE6,0:J=0:GOSUB23000:PRINT:PRINT:PRINT
6050 IFF3=1THEN6210
6060 M=N-1
6070 FORI=1TON:F(I)=A(2,I):NEXT
6080 GOSUB25000
6090 EX=NU:IFEX=0THEN6270
6100 FORI=1TON-1:E(1,I)=L(1,I):E(2,I)=L(2,I):E(3,I)=0:NEXT
6110 EX=0
6120 FORI=1TON-1
6130 IFE(2,I)<>0THEN6200
6140 J=2
6150 X=E(1,I):GOSUB24000
6160 IFY>E0THENE(3,I)=-1:EX=EX+1:GOTO6200
6170 IFY<-E0THENE(3,I)=1:EX=EX+1:GOTO6200
6180 J=J+2
6190 IFJ<=NTHEN6150
6200 NEXT
6210 FORI=1TON-1
6220 IFE(3,I)=0THEN6260
6230 PRINT"f(";E(1,I);")=";
6240 J=0:X=E(1,I):GOSUB24000:PRINTY;" - ";
6250 IFE(3,I)=1THENPRINT"max":ELSEPRINT"min"
6260 NEXT
6270 IFEX=0THENPRINT"    Es existieren keine Extremwerte."
6280 F3=1:GOTO21000
7000 !WENDEPUNKTE
7010 IFF1=0THEN2000
7020 CLS:PRINTAT(2,14);"WENDEPUNKTE"
7030 PRINTAT(3,14);"____________"
7040 LOCATE6,0:J=0:GOSUB23000:PRINT:PRINT:PRINT
7050 IFF4=1THEN7180
7060 WE=0
7070 IFN<3THEN7240
7080 M=N-2:FORI=1TON-1:F(I)=A(3,I):NEXT
7090 GOSUB25000
7100 WE=NU:IFWE=0THENGOTO7240
7110 FORI=1TON-2:W(1,I)=L(1,I):W(2,I)=L(2,I):W(3,I)=0:NEXT
```

```
7120 WE=0
7130 FORI=1TON-2
7140 IFW(2,I)<>0THEN7170
7150 X=W(1,I):J=3:GOSUB24000
7160 IFY<>0THENW(3,I)=1:WE=WE+1
7170 NEXT
7180 IFWE=0THENGOTO7240
7190 FORI=1TON-2
7200 IFW(3,I)<>1THENGOTO7230
7210 X=W(1,I):J=0:GOSUB24000
7220 PRINT"f(";W(1,I);")=";Y
7230 NEXT
7240 IFWE=0THENPRINT"    Es existieren keine Wendepunkte."
7250 F4=1
7260 GOTO21000
8000 !GRAFISCHE DARSTELLUNG
8010 IFF1=0THEN2000
8020 IFF2=0THEN4000
8030 IFF3=0THEN6000
8040 CLS
8050 A=1E38:B=-1E38
8060 FORI=1TON
8070 IFZ(1,I)<A THEN A=Z(1,I)
8080 IFZ(1,I)>BTHENB=Z(1,I)
8090 IFI=NTHEN8120
8100 IFE(1,I)<A THENA=E(1,I)
8110 IFE(1,I)>BTHENB=E(1,I)
8120 NEXT
8130 DX=B-A
8140 IFDX=0THENA=A-5:B=B+5:GOTO8130
8150 A=A-DX/2:B=B+DX/2:DX=B-A
8160 C=0:D=0
8170 FORW=1TON-1
8180 IFE(3,W)=0THEN8220
8190 J=0:X=E(1,W):GOSUB24000
8200 IFY<CTHENC=Y
8210 IFY>DTHEND=Y
8220 NEXT
8230 IFC=0ANDD=0THENC=-3.875:D=3.875
8240 DY=D-C
8250 C=C-DY/2:D=D+DY/2:DY=D-C
8260 F=239/DY:G=319/DX
8270 X0=-C*F:LINE0,X0,319,X0,7
8280 SX=10^INT(LN(DX)/LN(10)):IFABS(DX-SX)<E0THENSX=DX/10
8290 V=SX*INT(A/SX):IFV<A THENV=V+SX
8300 IFV+SX>BTHENSX=SX/10:GOTO8290
8310 IFV+3*SX>BTHENSX=SX/5:GOTO8290
8320 FORI=VTOBSTEPSX
8330 IFABS(I)<E0THENI=0
8340 LINE(I-A)*G,X0-3,(I-A)*G,X0+3,7
8350 NEXT
8360 Y0=-A*G:IFY0<0ORY0>319THEN8380
8370 LINEY0,0,Y0,239,7
8380 IFY0<3THENY0=3
8390 IFY0>316THENY0=316
8400 SY=10^INT(LN(DY)/LN(10)):IFABS(DY-SY)<E0THENSY=DY/10
8410 V=SY*INT(C/SY):IFV<CTHENV=V+SY
8420 IFV+SY>DTHENSY=SY/10:GOTO8410
8430 IFV+3*SY>DTHENSY=SY/5:GOTO8410
8440 FORI=VTODSTEPSY
8450 IFABS(I)<E0THENI=0
8460 LINEY0-3,(I-C)*F,Y0+3,(I-C)*F,7
8470 NEXT
8480 IFF5=1THEN8560
```

```
8490 K=30+N*7:!Punktanzahl
8500 FORI=0TOK-1:P(I)=0:NEXT
8510 X=A:J=0
8520 FORW=0TOK-1:GOSUB24000
8530 P(W)=(Y-C)*F
8540 X=X+DX/(K-1)
8550 NEXT
8560 FORI=0TOK-2
8570 IFP(I)<0ORP(I)>239ORP(I+1)<0ORP(I+1)>239THEN8590
8580 LINEI*319/(K-1),P(I),(I+1)*319/(K+1),P(I+1),7
8590 NEXT I
8600 WINDOW 0,1,0,39:COLOR0,7:CLS
8610 PRINTAT(1,4);CHR$(166);"x=";SX;"   ";CHR$(166);"y=";SY
8620 F5=1
8630 PRINTAT(0,4);"Mit 'ENTER' zur";CHR$(163);"ck "
8640 PRINTAT(0,23);"zur ";CHR$(163);"BERSICHT"
8650 T$=INKEY$:IFT$<>CHR$(13)THEN8650
8660 GOTO100
9000 !TANGENTENGLEICHUNGEN
9010 IFF1=0THEN2000
9020 CLS:PRINTAT(2,10);"TANGENTENGLEICHUNGEN"
9030 PRINTAT(3,10);"____________________"
9040 LOCATE6,0:J=0:GOSUB23000:PRINT:PRINT
9050 J=1:GOSUB23000:PRINT:PRINT:PRINT
9060 T$=" ":INPUT"x=";T$:GOSUB22000
9070 IFFE=1THENPRINTCHR$(11);:GOTO9060
9080 IFFE=2THEN100
9090 X=VAL(T$)
9100 J=0:GOSUB24000:YF=Y
9110 J=1:GOSUB24000:YA=Y
9120 NT=YF-YA*X
9130 PRINT"y=";
9140 FL=0
9150 IFYA=0THENGOTO9190
9160 IFYA=1THENPRINT"x";:FL=1:GOTO9190
9170 IFYA=-1THENPRINT"-x";:FL=1:GOTO9190
9180 PRINTSTR$(YA);"x";:FL=1
9190 IFNT=0THENGOTO9220
9200 IFLEFT$(STR$(NT),1)=" "ANDFL=1THENPRINT"+";
9210 PRINTSTR$(NT);:FL=1
9220 IFFL=0THENPRINT"0";
9230 AL=ATN(YA)*57.295778
9240 IFAL<0THENAL=AL+180
9250 AL=INT(AL*100+.5)/100
9260 PRINTTAB(25);CHR$(167);"=";STR$(AL);CHR(168):PRINT
9270 GOTO9060
10000 !ENDE
10010 WINDOW0,31,0,39:COLOR7,0:CLS
10020 PRINTAT(12,2);"EMPFEHLEN SIE DIESES PROGRAMM WEITER!"
10030 PRINTAT(15,14);"UWE GIRLICH"
10040 PRINTAT(18,17);CHR$(165);"1986"
10050 WINDOW0,0,0,0:COLOR0,0
10060 END
20000 !GRAFIKZEICHEN
20010 RESTORE20080
20020 FORI=0TO17
20030 FORI=0TO7
20040 READK:POKE 8*I+J,K
20050 NEXTJ,I
20060 VPOKE14250,0:VPOKE14251,0
20070 RETURN
20080 DATA 102,0,60,102,102,102,59,0
20090 DATA 102,0,60,102,126,102,102,0
20100 DATA 102,0,60,102,102,102,60,0
```

```
20110 DATA 102,0,102,102,102,102,60,0
20120 DATA 0,56,108,120,108,120,96,96
20130 DATA 60,66,153,161,161,153,66.60
20140 DATA 8,20,20,34,34,65,127,0
20150 DATA 0,0,49,74,68,74,49,0
20160 DATA 96,144,144,96,0,0,0,0
20170 DATA 96,144,32,64,240,0,0,0
20180 DATA 224,16,96,16,224,0,0,0
20190 DATA 128,160,240,32,32,0,0,0
20200 DATA 240,128,224,16,224,0,0,0
20210 DATA 96,128,224,144,96,0,0,0
20220 DATA 240,16,32,64,64,0,0,0
20230 DATA 96,144,96,144,96,0,0,0
20240 DATA 96,144,112,16,96,0,0,0
20250 DATA 46,106,170,42,46,0,0,0
21000 !ZURUECK ZUR UEBERSICHT
21010 PRINTAT(30,0);"    Mit 'ENTER' zur";CHR$(163);"ck "
21020 PRINTAT(30,23);"zur ";CHR$(163);"BERSICHT"
21030 T$=INKEY$:IFT$<>CHR$(13)THEN21030
21040 GOTO100
22000 !EINGABETEST
22010 IFASC(T$)=32THENFE=2:RETURN
22020 FE=0
22030 IFLEN(T$)>12THENFE=1:RETURN
22040 FORH=1TOLEN(T$)
22050 T=ASC(MID$(T$,H,1))
22060 IFT<>45ANDT<>46ANDT<>69ANDT<>101AND(T<48ORT>57)THENFE=1
22070 NEXT
22080 RETURN
23000 !POLYNOME AUSGEBEN
23010 PRINT"f";
23020 IFJ=0THEN23040
23030 IFJ<4THENPRINTSTR$(J,"'");:ELSEPRINT"<";J;">";
23040 PRINT"(x)=";
23050 IFJ>NTHENPRINT"0";:GOTO23180
23060 FORI=N-JTO0STEP-1
23070 IF ABS(A(J+1,I+1))<E0THEN23170
23080 IFA(J+1,I+1))>0ANDI<N-JTHENPRINT"+";
23090 IFABS(A(J+1,I+1))<>1ORI=0THEN23120
23100 IFA(J+1,I+1)=-1THENPRINT"-";
23110 GOTO23150
23120 A$=STR$(A(J+1,I+1))
23130 IFLEFT$(A$,1)=" "THENA$=RIGHT$(A$,LEN(A$)-1)
23140 PRINTA$;
23150 IFI>0THENPRINT"x";
23160 IFI>1THENPRINTCHR$(I+167);
23170 NEXTI
23180 RETURN
24000 !HORNERSCHEMA
24010 J=J+1
24020 Y=A(J,N-J+2)*X
24030 IF N-J=0 THEN24070
24040 FORE=N-J+1TO2STEP-1
24050 Y=(Y+A(J,E))*X
24060 NEXT
24070 Y=Y+A(J,1)
24080 J=J-1
24090 RETURN
25000 !NULLSTELLEN
25010 NU=0
25020 FORI=1TOM+3:B(I)=0:C(I)=0:NEXT
25030 FORI=1TON:L(1,I)=0:L(2,I)=0:NEXT
25040 IFM=1THEN25530
25050 IFM=0THENNU=0:RETURN
```

```
25060 PRINTAT(30,0);"Genauigkeit:"
25070 P=1:Q=1
25080 FORI=M+1TO1STEP-1
25090 B(I)=F(I)-P*B(I+1)-Q*B(I+2)
25100 NEXT
25110 FORI=M+1TO1STEP-1
25120 C(I)=B(I)-P*C(I+1)-Q*C(I+2)
25130 NEXT
25140 P1=P
25150 DET=C(3)*C(3)-C(4)*C(2)
25160 IFABS(DET)<E2THENP=P+1:Q=Q+1:GOTO25080
25170 P=P+(B(2)*C(3)-B(1)*C(4))/DET
25180 Q=Q+(B(1)*C(3)-B(2)*C(2))/DET
25190 PRINTAT(30,12);B(2);"              "
25200 IFPEEK(509)=65THENSOUND54,0,54,0,31,0:GOTO25220
25210 IFABS(P1-P)>=E1THEN25080
25220 IFPEEK(509)=65THEN25220
25230 SOUND0,0,0,0,0,0
25240 D=P*P/4-Q
25250 IFABS(D)<E0THEND=0
25260 IFABS(P)<E0THENP=0
25270 IFD<0THEN25350
25280 NU=NU+1
25290 L(1,NU)=-P/2+SQR(D)
25300 IFABS(L(1,NU))<E0THENL(1,NU)=0
25310 NU=NU+1
25320 L(1,NU)=-P/2-SQR(D)
25330 IFABS(L(1,NU))<E0THENL(1,NU)=0
25340 GOTO25460
25350 D=SQR(ABS(D))
25360 NU=NU+1
25370 L(1,NU)=-P/2
25380 L(2,NU)=D
25390 IFABS(L(1,NU))<E0THENL(1,NU)=0
25400 IFABS(L(2,NU))<E0THENL(2,NU)=0
25410 NU=NU+1
25420 L(1,NU)=-P/2
25430 L(2,NU)=-D
25440 IFABS(L(1,NU))<E0THENL(1,NU)=0
25450 IFABS(L(2,NU))<E0THENL(2,NU)=0
25460 M=M-2
25470 IFM=0THENRETURN
25480 FORI=1TOM+1
25490 F(I)=B(I+2)
25500 NEXT
25510 FORI=1TOM+3:B(I)=0:C(I)=0:NEXT
25520 IFM<>1THEN25070
25530 NU=NU+1:L(1,NU)=-F(1)/F(2):IFABS(L(1,NU))<E0THENL(1,NU)=0
25540 RETURN
```

Spielprogrammierung

— Von der Idee zum fertigen Spiel —

(Teil 1)

Dieser Beitrag richtet sich an einen Leserkreis mit BASIC-Grundkenntnissen und hat das Ziel, den Weg von der Idee für ein Spielprogramm über eine mathematische Betrachtung und Modellierung sowie stufenweiser Umsetzung und Testung bis hin zur »druckreifen« Fassung darzustellen.
Alle Programme wurden für den KC 85/2 in seiner Grundausstattung geschrieben, kommen also mit 5 KByte Arbeitsspeicher aus. Die Programme widmen sich im einzelnen folgenden Schwerpunkten:

Siebzehn und Vier	– Zufallszahlen
Rotation	– Einfache Indexrechnung, Zeichendarstellung im ASCII-Code
Supermagische Quadrate	– Mathematische Analyse, Indexrechnung
Regelmäßige Vielecke	– Geometrische Betrachtung, Vollgrafik
Schiebefax	– Untergliederung einer komplexen Aufgabe in Teilschritte
Barrikade	– Arbeit mit dem Bildwiederholspeicher, Pseudografik

Alle Programme sollten entsprechend den angegebenen Hinweisen schrittweise aufgebaut und getestet werden. Dies fördert das Verständnis, garantiert zugleich eine einwandfreie Funktion und legt die Grundlage für eigene Verbesserungen und Erweiterungen des Lesers.

Siebzehn und Vier

Dieses einfache Spiel wird mit Skatkarten gespielt. Es dürfen beliebig viele Karten aufgenommen werden, wer aber dabei 21 Punkte überschreitet, hat verloren. Sieger ist derjenige mit den meisten Punkten.

1. Nachbilden des Kartenstromes

Es gibt im Skatblatt folgende Kartenwerte:

2 3 4 7 8 9 10 11.

Damit diese Karten in ungeordneter Folge erscheinen, wird eine gleichverteilte Zufallszahlenreihe entsprechend transformiert. Eine Zufallszahlenreihe erhält man mit RND (1), diese liefert Zahlen im Bereich von Null bis fast eins. Folglich liefert 8*RND (1) Zufallszahlen im Bereich von Null bis fast acht und somit INT (8*RND (1)) die Zahlen 0, 1, 2, 3, 4, 5, 6 und 7 in zufälliger Reihenfolge. Durch

```
10 W= INT (8*RND (1)) +2
20 IF W>4 THEN W=W+2
```

erhält man schließlich die gewünschte Reihe, wovon man sich leicht überzeugen kann, indem man hinzufügt

```
30 PRINT W
40 GOTO 10
```

Nach RUN erscheinen die erwarteten Zahlen, die Darstellung des Kartenstromes auf dem Rechner ist damit erfolgreich verwirklicht. Auf den realen Kartenstoß wurde hierbei nicht eingegangen, bei dem ja jeder Wert nur viermal auftreten kann. Unter Beachtung, daß nach jedem Spiel erneut alle Karten gemischt werden, spielt das hier auch keine wesentliche Rolle und kann bei der Darstellung auf dem Rechner vernachlässigt werden.

2. Programmeinführung

Im Programm wird nach Löschen des Bildschirms und eines Summenbereichs S die erste »Karte« erzeugt. Danach ist der Spieler zu befragen, ob er weiterspielen will. Wie bei jedem guten Programm werden mögliche Antworten in die Fragestellung hineinformuliert und bei der Auswertung dieser Antwort im Rechner auch Tippfehler berücksichtigt, d.h., beide mögliche Antworten werden abgetestet.
Vor der Abfrage wurde die Endekontrolle eingefügt, damit das Programm im ordentlichen Zyklus laufen kann. Bei Überschreitung des Höchstwertes oder durch Entscheidung des Spielers wird das Spiel beendet, die Abschlußbehandlung ist daher anzufügen (vgl. Bild 1).

```
10 CLS
20 S=0
30 PRINT:PRINT" DAS SPIEL BEGINNT"
40 W=INT(8*RND(1))+2
50 IF W>4 THEN W=W+2
60 S=S+W
70 PRINT"KARTE:";W;"  MACHT";S;"PUNKTE"
80 IF S>21 THEN130
90 INPUT"NOCH EINE KARTE (J/N)?";A$
100 IF A$="J" THEN 40
110 IF A$<>"N" THEN 90
120 GOTO 140
130 PRINT"        VERLOREN"
140 INPUT"EIN NEUES SPIEL (J/N)?";A$
150 IF A$="J" THEN 20
160 IF A$<>"N" THEN 140
```

Bild 1

3. Erweiterung

Interessanter wird das Ganze, wenn die Spiele gezählt und der jeweils erreichte Durchschnittsgewinn ausgegeben werden. Dazu ist das Programm entsprechend zu ergänzen.

4. Komplettierung

Vollständig wird das Programm nach Ergänzung um Endemeldung, Spielanleitung und Kommentar. Nun ist alles nur noch neu zu numerieren mit RENUMBER, und das Programm ist hoffähig und kann Freunden, Verwandten, Bekannten, dem Lehrer usw. vorgeführt werden (vgl. Bild 2).

Rotation

Hinter diesem Begriff verbergen sich zwei einfache Spiele, die mit wenig Aufwand und fast ohne mathematische Kenntnisse realisierbar sind. Eine Folge von zehn Buchstaben ist alphabetisch zu ordnen, indem man eine vorgegebene Anzahl von Buchstaben in sich verdreht bzw. das gleiche mit allen Buchstaben jeweils links und rechts eines vorgegeben Buchstabens durchführt.

1. Einfache Rotation

Zuerst wird eine ungeordnete Folge von Zahlen gebildet, wobei jede Zahl nur einmal vorkommen darf. Deshalb muß jede zufällig erzeugte Zahl mit allen vorherigen Zahlen auf Ungleichheit

```
10 REM  SIEBZEHN UND VIER
20 CLS
30 PRINT"DAS SPIEL  17 + 4"
40 PRINT"ES KOENNEN BELIEBIG VIELE KARTEN"
50 PRINT"GENOMMEN WERDEN. WER 21 UEBERSCHRTET"
60 PRINT"VERLIERT, SIEGER IST DER MIT DEM"
70 PRINT"HOECHSTEN WERT."
80 N=0:G=0
90 S=0
100 PRINT:PRINT" DAS SPIEL BEGINNT"
110 REM  K A R T E
120 W=INT(8*RND(1))+2
130 IF W>4 THEN W=W+2
140 S=S+W
150 REM  A N Z E I G E
160 PRINT"KARTE:";W;"  MACHT";S;"PUNKTE"
170 IF S>21 THEN 220
180 INPUT"NOCH EINE KARTE (J/N)?";A$
190 IF A$="J" THEN 120
200 IF A$<>"N" THEN 180
210 GOTO 250
220 PRINT"         VERLOREN"
230 S=0
240 REM  S T A T I S T I K
250 N=N+1:G=G+S
260 PRINT"AKTUELLER DURCHSCHNITT:";G/N
270 INPUT"EIN NEUES SPIEL (J/N)?";A$
280 IF A$="J" THEN 90
290 IF A$<>"N" THEN 270
300 PRINT"DANKE, ES HAT MIR SPASZ GEMACHT"
310 PRINT:PRINT"AUF WIEDERSEHEN ."
```

Bild 2

überprüft werden. Beim Druck werden diese Zahlen in Buchstaben umgewandelt, da jene optisch besser in Erscheinung treten. Der erste Schritt ist somit die Eingabe der Zeilen 100 bis 190 sowie deren Erprobung mittels RUN.
An die Zeilen 170 bis 190 zur Anzeige der aktuellen Stellung schließt sich die Endeabfrage an. Das Ordnen der Buchstaben wird in den Zeilen bis 280 vorbereitet durch die Eingabe und die Eingabekontrolle, die Rotation selbst erfolgt in den Zeilen 300 bis 320. Zu beachten ist hier, daß die Zahl der Vertauschungen nur halb so groß ist, wie die der zu vertauschenden Buchstaben, weil an jedem Tausch zwei Buchstaben beteiligt sind, z. B.

CFADGJHBIE WIEVIEL STELLEN: 5

bedeutet, daß C mit G und F mit D die Plätze tauschen zu

GDAFCJHBIE.

Nach Eingabe des Rücksprunges in Zeile 330 ist bereits eine Erprobung des Algorithmus möglich. Statistik (Zeilen 90, 290), Endemeldung (Zeilen 230 und 340 bis 370) sowie eine kurze Spielanleitung (Zeilen 10 bis 80) komplettieren das Programm.
Das Spiel kann nun beginnen. Eine Spielstrategie ist leicht selbst herauszufinden, dazu nur soviel – man kann immer mit weniger als 20 Zügen die geforderte Ordnung erreichen (vgl. Bild 3).

2. Doppelte Rotation

Etwas anspruchsvoller ist dieses Spielchen. Hierbei rotieren gleichzeitig die Buchstaben links und rechts von einem vorgegebenen, z. B. aus

CFADGJHBIE
FESTER BUCHSTABE: D

folgt

AFCDEIBHJG.

```
10 CLS
20 PRINT" ROTATION"
30 PRINT
40 PRINT" DIE ANGEGEBENE BUCHSTABENFOLGE IST"
50 PRINT" DURCH UMDREHEN EINER VORGEBBAREN"
60 PRINT" ANZAHL VON BUCHSTABEN ZU ORDNEN."
70 PRINT" ES WERDEN STETS DIE LINKEN BUCHSTABEN"
80 PRINT" IN BEFOHLENER ANZAHL VERTAUSCHT."
90 PRINT:N=0
100 A(1)=INT(10*RND(1))
110 FOR I=2 TO 10
120 A(I)=INT(10*RND(1))
130 FOR J=1 TO I-1
140 IF A(I)=A(J) THEN 120
150 NEXT J
160 NEXT I
170 FOR I=1 TO 10
180 PRINT CHR$(A(I)+65);
190 NEXT I
200 FOR I=0 TO 9
210 IF I<>A(I+1) THEN 240
220 NEXT I
230 PRINT:PRINT"MIT";N;"VERSUCHEN GELOEST":GOTO 340
240 PRINT"   WIEVIEL STELLEN";
250 INPUT K
260 IF K<2 THEN 240
270 IF INT(K)<K THEN 240
280 IF K>10 THEN 240
290 N=N+1
300 FOR I=1 TO INT(K/2)
310 Z=A(I):A(I)=A(K-I+1):A(K-I+1)=Z
320 NEXT I
330 GOTO 170
340 PRINT:PRINT"AUF EIN NEUES(J/N)?"
350 B$=INKEY$
360 IF B$="J" THEN 90
370 IF B$<>"N" THEN 350
```

Bild 3

Bei jedem Zug werden stets 4 Buchstabenpaare getauscht, wie man leicht ausprobieren kann. Deshalb sind bei diesem Spiel auch nur Aufgabenstellungen aus geraden Permutationen lösbar, das sind solche Anordnungen der Buchstaben, die durch eine gerade Anzahl von Vertauschungen aus der geordneten Folge erzeugt werden können. Im Unterschied zum vorstehenden Programm wird hier die Anfangsreihenfolge durch 10 willkürliche Vertauschungen erzeugt. Wichtig ist dabei, daß keine Zahl mit sich selbst vertauscht werden darf, da dann eine ungerade Permutation die Folge wäre.
Das Programmieren sollte wieder in Etappen erfolgen, also zunächst nur mit den Zeilen 90 bis 180 beginnend. Damit erscheint beim ersten Abarbeitungsversuch die ungeordnete Buchstabenfolge auf dem Bildschirm. Als nächstes sind die Zeilen

190 bis 220 – Endekontrolle,
230 bis 310 – Eingabe und Eingabekontrolle sowie
330 bis 410 – Vertauschungen vor und nach dem angegebenen Buchstaben

an der Reihe.

Wenn dieses Programm soweit funktioniert, dann braucht das Programm nur noch mit den restlichen Zeilen entsprechend obigem Verfahrensweg komplettiert zu werden. Ein Vergleich mit der gesamten, hier folgenden Vorlage zeigt alle betreffenden Stellen auf.
In diesem Spiel ist die Ordnung der Buchstaben (bei entsprechender

```
10 CLS
20 PRINT"DOPPELTE ROTATION"
30 PRINT:PRINT"DIE FOLGENDEN BUCHSTABEN SIND ZU"
40 PRINT"ORDNEN. NACH DER VORGABE EINES BUCH-"
50 PRINT"STABENS WERDEN DIE DAVOR- UND DIE"
60 PRINT"DANACHSTEHENDEN JEWEILS IN IHRER"
70 PRINT"REIHENFOLGE UMGEDREHT."
80 PRINT:N=0
90 FOR I=1 TO 10
100 A(I)=I-1
110 NEXT I
120 FOR I=1 TO 10
130 J=INT(10*RND(1))+1:IF J=I THEN 130
140 Z=A(I):A(I)=A(J):A(J)=Z
150 NEXT I
160 FOR I=1 TO 10
170 PRINT CHR$(A(I)+65);
180 NEXT I
190 FOR I=0 TO 9
200 IF I<>A(I+1) THEN 230
210 NEXT I
220 PRINT:PRINT"MIT";N;"VERSUCHEN GELOEST":GOTO 420
230 PRINT"   FESTER BUCHSTABE ";
240 B$=INKEY$
250 IF B$<"A" THEN 240
260 IF B$>"J" THEN 240
270 PRINT B$
280 L=ASC(B$)-65
290 FOR K=1 TO 10
300 IF A(K)=L THEN 320
310 NEXT K
320 N=N+1
330 IF K<3 THEN 370
340 FOR I=1 TO INT(K/2)
350 Z=A(I):A(I)=A(K-I):A(K-I)=Z
360 NEXT I
370 IF K>8 THEN 410
380 FOR I=INT((13+K)/2) TO 10
390 Z=A(I):A(I)=A(11+K-I):A(11+K-I)=Z
400 NEXT I
410 GOTO 160
420 PRINT:PRINT"AUF EIN NEUES(J/N)? "
430 B$=INKEY$
440 IF B$="J" THEN 80
450 IF B$<>"N" THEN 430
```

Bild 4

Übung) noch schneller als bei dem ersteren zu erreichen (vgl. Bild 4).

Supermagische Quadrate

Magische Quadrate beschäftigen die Menschheit schon seit langem, erinnert sei hier nur an Dürers Kupferstich »Melancholie« von 1514 sowie an den Zauberspruch aus Goethes Faust. Laut [6] erwähnt sogar ein 6000 bis 7000 Jahre altes Buch aus dem Orient solche Gebilde. Sie bestehen aus aufeinanderfolgenden Zahlen in quadratischer Anordnung, deren Zeilen- und Spaltensummen sowie meist auch Diagonalensummen alle übereinstimmen sollen.

Supermagisch sei ein solches vierzeiliges Quadrat, in welchem außer den oben genannten noch die Summe der Ecken, die Summen aller 2×2-Teilquadrate, die Summen der Ecken in allen 3×3-Teilquadraten, ja sogar die Summen aller gebrochenen Diagonalen und die Summen der Ecken in allen

2×4-Teilrechtecken – in welchem diese alle die gleiche Summe ergeben. Dies leistet z.B. mit der supermagischen Summe 34

13	12	6	3
2	7	9	16
11	14	4	5
8	1	15	10

Bild 5

Zum besseren Verständnis seien obige und weitere Summenbildungen nochmals als Skizzen dargestellt (Bild 6).

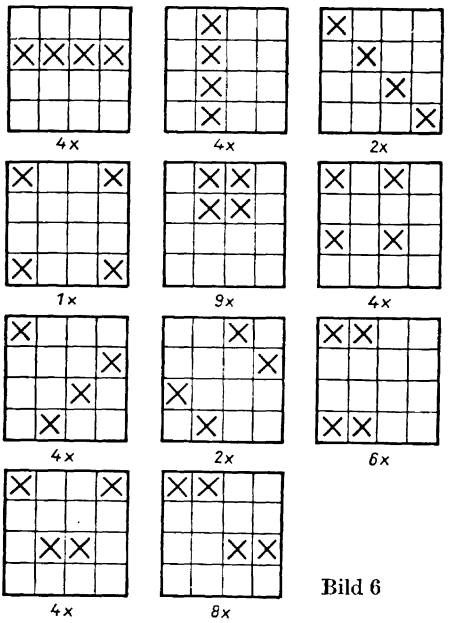


Bild 6

Das sind also 48 verschiedene Summationen mit dem selben Wert! Die Aufgabe besteht nun darin, für jede beliebige Summe ein (möglichst super-) magisches Quadrat zu erzeugen.

1. Lösungsalgorithmus

Ein Elementarquadrat sei eine Anordnung von je zwei Nullen und zwei Einsen in jeder Zeile und jeder Spalte eines 4reihigen Quadrates. Durch Fallunterscheidung läßt sich beweisen, daß es genau 4 voneinander unabhängige Elementarquadrate gibt, die an einer fest vorgegebenen Stelle eine Eins haben und zugleich in allen oben angeführten Summen den Wert 2 ergeben. Für die Fixstelle unten rechts sind dies:

$S_1 =$

1	1	0	0
0	0	1	1
1	1	0	0
0	0	1	1

$S_2 =$

1	0	1	0
1	0	1	0
0	1	0	1
0	1	0	1

$S_3 =$

0	1	1	0
1	0	0	1
0	1	1	0
1	0	0	1

$S_4 =$

0	1	0	1
1	0	1	0
1	0	1	0
0	1	0	1

Bild 7. Beispiel Quadrat – Teilungsverhältnis 1:3

Analoges ist trivialerweise (es liegt auf der Hand, ist offensichtlich) auch für ein Quadrat erfüllt, welches nur mit Einsen besetzt ist.

$E =$

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

Bild 8

Hier ist sogar jede beliebige Summe aus vier Elementen gleich 4. Jede Linearkombination der $S(i)$ und E hat zwangsläufig auch die gleichen Summationseigenschaften. Speziell also auch

$\bar{S}_3 = E - S_3 =$

1	0	0	1
0	1	1	0
1	0	0	1
0	1	1	0

Bild 9

welches mit S1, S2 und S4 die Fixstelle in Zeile 2 und Spalte 3 gemeinsam hat. Auf diese Weise ist jedes $S(i)$

durch $E - S(i)$ austauschbar, der oben verwendete Begriff Fixstelle ist daher entbehrlich.
Jede Linearkombination der $S(i)$ mit den Koeffizienten 1, 2, 4 und 8 bzw. ersatzweise der $E - S(i)$ ergibt ein supermagisches Quadrat mit der magischen Summe 30. Hierbei kann man die Elementarquadrate $S(i)$ bzw. ihre Gegenstücke auch als Ziffernstellen im Dualsystem auffassen. Ganzzahlige Vielfache von E hinzugefügt, erhält man für alle Summen der Form $4k + 2$ supermagische Quadrate.

$M = k * E \pm 8 * S(a) \pm 4 * S(b) \pm 2 * S(c) \pm S(d)$
mit a, b, c, d = 1, 2, 3, 4 in beliebiger Reihenfolge.

Das oben angeführte Beispielquadrat besteht aus

$$E + 8 * S1 + 4 * \overline{S4} + 2 * \overline{S2} + S3.$$

Man kann beweisen, daß mit dieser Rechenvorschrift alle nur möglichen supermagischen Quadrate erzeugt werden können. Es gibt übrigens genau 48 verschiedene supermagische Quadrate zu einer vorgegebenen Summe, die sich nicht durch Drehung oder Spiegelung ineinander überführen lassen. Sie sind im Anschluß an das Programm aufgeführt.
Die Summe der Zahlen von 0 bis 15 ist 120, die Summe von 16 ab A dicht aufeinander folgenden Zahlen ist demnach $16A + 120$. Dies ergibt als Zeilen- oder Spaltensumme eines magischen Quadrates $4A + 30$ oder, allgemeiner ausgedrückt, stets die Form $4k + 2$ (k ganz). Um auch zu anderen Zahlen magische Quadrate aufbauen zu können, muß eine Lücke in der Zahlenreihe zugelassen werden. Bei Verwendung der Gewichte 1, 2, 4 und 9 entsteht genau in der Mitte der Zahlenfolge eine einschrittige Lücke. Dies ist für Summen der Form $4k$ der einzige Makel an der Supermagik.

$M' = k * E \pm 9 * S(a) \pm 4 * S(b) \pm 2 * S(c) \pm S(d)$
mit a, b, c, d = 1, 2, 3, 4 in beliebiger Reihenfolge

Zur Realisierung von magischen Quadraten mit ungerader Zeilensumme betrachtet man die Elementarquadrate mit den Mitteln der Boolеschen Algebra. Wie nähere Untersuchungen zeigen, ergeben nur die Kombinationen von S2 mit S3 und von S1 mit S4 brauchbare Ergebnisse, alle in der Gestalt wie

$S_1 \wedge S_4 =$

0	1	0	0
0	0	1	0
1	0	0	0
0	0	0	1

$S_1 \vee S_4 =$

1	1	0	1
1	0	1	1
1	1	1	0
0	1	1	1

Bild 10

Hierbei bleiben immerhin noch 34 der oben angegebenen Summen bestehen. Die Erzeugungsformel für fast supermagische Quadrate mit ungerader Zeilen- und Spaltensumme lautet somit:

$M'' = k * E \pm 8 * S(a) \pm 4 * S(b) \pm 2 * S(c) \pm S(d) \pm S(a) \wedge_{\vee} S(b)$
mit a, b, c, d = 1, 2, 3, 4 in solcher Reihenfolge, daß a + b = 5 gilt

Eine einfache Realisierung ergibt sich somit durch Bezug auf ein supermagisches Quadrat mit der magischen Summe 30 und der Beziehung $a + b = 5$. Mittels $k * E$ erfolgt eine Korrektur, so daß die geforderte Summe höchstens um 3 höher, aber nicht niedriger liegt. Anschließend werden je nach Bedarf die größten 0, 4, 8 oder 12 Zahlen um jeweils 1 erhöht.

2. Programmeinführung

In einer einfachen Programmvariante wird von einem festliegenden magischen Quadrat mit der Summe 30 ausgegangen. Aus dem Eingabewert werden der Erhöhungsbetrag A und die Position einer eventuell notwendigen Lücke R berechnet. Daraus wird das gewünschte magische Quadrat in der Reihenfolge der Zahlen des zugrunde gelegten magischen Quadrates aufgebaut und anschließend komplett angezeigt. Bei jedem Programmaufruf entsteht hier zur selben Eingabezahl auch stets das selbe magische Quadrat.

Bei der komfortablen Programmvariante wird von den binären Elementarquadraten ausgegangen. Über einen Zufallszahlengenerator erfolgt die Auswahl einer geeigneten Linearkombination daraus, welche zur Grundlage der Berechnung gemacht wird.

Die Programmeingabe erfolgt zweckmäßigerweise in drei Etappen, welche jeweils für sich getestet werden, bevor die jeweils nächste Etappe in Angriff genommen wird. Zuerst sind die Programmzeilen 1Ø bis 14Ø einzugeben und dazu als Testhilfe noch Bild 11.

```
150 FOR J=1 TO 4
160 FOR I=1 TO 4
170 FOR K=1 TO 4
180 PRINT CHR$(48+S(I,J,K));
190 NEXT K
200 PRINT" ";
210 NEXT I
220 PRINT
230 NEXT J
```

Bild 11

Wenn alles richtig eingegeben wurde, dann erscheinen hierauf die vier Elementarquadrate auf dem Bildschirm. Die zweite Etappe geht bis zur Zeile 31Ø. Durch

32Ø PRINT F (1); F (2); F (3); F (4)

möge man sich von der sachgemäßen Bildung der Koeffizienten überzeugen. Danach kann das Programm fertig aufgebaut und ausgetestet werden.

Ein einfaches Programm zur Berechnung magischer Quadrate zeigt Bild 12.

Ein komfortables Programm zur Berechnung aller supermagischen Quadrate in lexikografischer Anordnung (ohne Drehungen und Spiegelungen) stellt Bild 13 dar.

Bild 14 weist die Ergebnisse des Programms für alle supermagischen Quadrate der Summe 30 aus.

```
10 REM MAGISCHES QUADRAT
20 DIM F(4,4)
30 INPUT"MAGISCHE SUMME";S
40 D=S-30:A=INT(D/4):R=D-4*A
50 F(4,2)=A:F(2,1)=A+1:F(1,4)=A+2:F(3,3)=A+3
60 IF R=3 THEN A=A+1
70 F(3,4)=A+4:F(1,3)=A+5:F(2,2)=A+6:F(4,1)=A+7
80 IF R=2 THEN A=A+1
90 F(2,3)=A+8:F(4,4)=A+9:F(3,1)=A+10:F(1,2)=A+11
100 IF R=1 THEN A=A+1
110 F(1,1)=A+12:F(3,2)=A+13:F(4,3)=A+14:F(2,4)=A+15
120 CLS
130 FOR I=1 TO 5
140 PRINT AT(4*I,5);STRING$(25,"■")
150 FOR J=5 TO 19
160 PRINT AT(J,6*I-1);"■"
170 NEXT J,I
180 FOR I=1 TO 4
190 FOR J=1 TO 4
200 PRINT AT(4*I+2,6*J);F(I,J)
210 NEXT J,I
```

Bild 12

```
10 REM MAGISCHE QUADRATE
20 DIM S(4,4,4)
30 B=1
40 FOR I=1 TO 4
50 B=1-B
60 FOR J=1 TO 4
70 C=B
80 IF J>2 THEN C=1-C
90 S(1,I,J)=C
100 S(2,J,I)=C
110 K=J-1:IF K=0 THEN K=4
120 S(3,I,K)=C
130 S(4,K,I)=C
140 NEXT J,I
150 INPUT"MAGISCHE SUMME:";M
160 D=M-30:A=INT(D/4):R=D-4*A
170 H=0.22+0.2*R:L=0
180 F(1)=INT(RND(1)*4) :REM PERMUTATION
190 F(4)=INT(RND(1)*4)
200 IF F(4)=F(1) THEN 190
210 IF R=2 OR R=0 THEN 240
220 IF F(1)>1 THEN F(4)=5-F(1)
230 IF F(1)<2 THEN F(4)=1-F(1)
240 F(2)=INT(RND(1)*4)
250 IF F(2)=F(1) OR F(2)=F(4) THEN 240
260 F(3)=6-F(1)-F(2)-F(4)
270 FOR J=1 TO 4
280 V=1:IF RND(1)<.5 THEN V=-1
290 F(J)=V*2^F(J)
300 IF V<0 THEN L=L-F(J)
310 NEXT J
320 REM RAHMEN
330 CLS
340 FOR I=1 TO 5
350 PRINT AT(4*I,5);STRING$(25,"■")
360 FOR J=5 TO 19
370 PRINT AT(J,6*I-1);"■"
380 NEXT J,I
390 REM BERECHNUNG UND AUSGABE
400 FOR I=1 TO 4
410 FOR J=1 TO 4
420 E=L
430 FOR K=1 TO 4
440 E=E+S(K,I,J)*F(K)
450 NEXT K
460 PRINT AT(4*I+2,6*J);INT(E*1.05+H)+A
470 NEXT J,I
480 GOTO 150
```

Bild 13

```
 0  7  9 14     0  7  9 14     0  7 10 13     0  7 10 13
11 12  2  5    13 10  4  3    11 12  1  6    14  9  4  3
 6  1 15  8     6  1 15  8     5  2 15  8     5  2 15  8
13 10  4  3    11 12  2  5    14  9  4  3    11 12  1  6

 0  7 12 11     0  7 12 11     0 11  5 14     0 11  6 13
13 10  1  6    14  9  2  5    13  6  8  3    14  5  8  3
 3  4 15  8     3  4 15  8    10  1 15  4     9  2 15  4
14  9  2  5    13 10  1  6     7 12  2  9     7 12  1 10

 0 11 12  7     0 11 12  7     0 13  6 11     0 13 10  7
13  6  1 10    14  5  2  9    14  3  8  5    14  3  4  9
 3  8 15  4     3  8 15  4     9  4 15  2     5  8 15  2
14  5  2  9    13  6  1 10     7 10  1 12    11  6  1 12
```

1	6	8	15	1	6	8	15	1	6	11	12	1	6	11	12
10	13	3	4	12	11	5	2	10	13	0	7	15	8	5	2
7	0	14	9	7	0	14	9	4	3	14	9	4	3	14	9
12	11	5	2	10	13	3	4	15	8	5	2	10	13	0	7
1	6	13	10	1	6	13	10	1	10	4	15	1	10	7	12
12	11	0	7	15	8	3	4	12	7	9	2	15	4	9	2
2	5	14	9	2	5	14	9	11	0	14	5	8	3	14	5
15	8	3	4	12	11	0	7	6	13	3	8	6	13	0	11
1	10	13	6	1	10	13	6	1	12	7	10	1	12	11	6
12	7	0	11	15	4	3	8	15	2	9	4	15	2	5	8
2	9	14	5	2	9	14	5	8	5	14	3	4	9	14	3
15	4	3	8	12	7	0	11	6	11	0	13	10	7	0	13
2	5	8	15	2	5	11	12	2	5	14	9	2	5	14	9
12	11	6	1	15	8	6	1	12	11	0	7	15	8	3	4
7	0	13	10	4	3	13	10	1	6	13	10	1	6	13	10
9	14	3	4	9	14	0	7	15	8	3	4	12	11	0	7
2	9	4	15	2	9	7	12	2	9	14	5	2	9	14	5
12	7	10	1	15	4	10	1	12	7	0	11	15	4	3	8
11	0	13	6	8	3	13	6	1	10	13	6	1	10	13	6
5	14	3	8	5	14	0	11	15	4	3	8	12	7	0	11
2	12	7	9	2	12	11	5	3	4	9	14	3	4	10	13
15	1	10	4	15	1	6	8	13	10	7	0	14	9	7	0
8	6	13	3	4	10	13	3	6	1	12	11	5	2	12	11
5	11	0	14	9	7	0	14	8	15	2	5	8	15	1	6
3	4	15	8	3	4	15	8	3	8	5	14	3	8	6	13
13	10	1	6	14	9	2	5	13	6	11	0	14	5	11	0
0	7	12	11	0	7	12	11	10	1	12	7	9	2	12	7
14	9	2	5	13	10	1	6	4	15	2	9	4	15	1	10
3	8	15	4	3	8	15	4	3	13	6	8	3	13	10	4
13	6	1	10	14	5	2	9	14	0	11	5	14	0	7	9
0	11	12	7	0	11	12	7	9	7	12	2	5	11	12	2
14	5	2	9	13	6	1	10	4	10	1	15	8	6	1	15
4	3	13	10	4	3	14	9	5	2	12	11	5	2	15	8
15	8	6	1	15	8	5	2	14	9	7	0	14	9	4	3
2	5	11	12	1	6	11	12	3	4	10	13	0	7	10	13
9	14	0	7	10	13	0	7	8	15	1	6	11	12	1	6

Bild 14

Literatur

[1] Jahn, F.: Alte deutsche Spiele. – Dresden, 1923

[2] Schubert, H.; Fitting, F.: Mathematische Mußestunden. – Berlin, 1943

[3] Kührt, H.: Alte und neue Brettspiele.— Weimar 1951

[4] Domorâd, A. P.: Matematičeskie igry i razvlečeniâ. – Moskau, 1961

[5] Kleine Enzyklopädie Mathematik. – Leipzig, 1965

[6] Perelman, J. I.: Unterhaltsame Aufgaben und Versuche. – Moskau, 1977

[7] Hirte, W.: Unsere Spiele. – Leipzig, 1982

Autor:

Dipl.-Math. Walter Görgens

Technische Universität

„Otto von Guericke"

Magdeburg, Sektion Informatik

Workstation

1. Was versteht man unter einer Workstation?

Der Begriff „Workstation" wurde Anfang der achtziger Jahre von der amerikanischen Firma Apollo Domain geprägt. Man verstand darunter einen Arbeitsplatzrechner für den professionellen Einsatz, der in seinen technischen Parametern den Heim- und Personalcomputern weit überlegen war. Ein charakteristisches Merkmal der Workstation ist vor allem ein grafischer Bildschirm mit Einzelpunktansteuerung, auf dessen Basis ein fortgeschrittener Mensch-Maschine-Dialog geführt werden kann. Weitere Geräte der Grafikperipherie wie Digitalisiertabletts und Mäuse vervollständigen oft eine Workstation.

Auf Grund des weltweiten Übergangs zum breiten Einsatz von Methoden des rechnergestützten Konstruierens (CAD) erhielt der Workstation-Markt in den letzten Jahren eine stürmische Entwicklung.

Workstations werden in der Regel als „intelligente" Datenendplätze von Klein- oder Großrechnern verwendet, wodurch der Workstationbenutzer bei Bedarf auf vom Zentralrechner verwaltete, gemeinsam genutzte Datenbanken, eine größere Verarbeitungsleistung und spezielle Peripherie, wie Laserdrucker, Magnetbandarchive usw., zugreifen kann.

Der Ausgangspunkt der Workstationentwicklung war die Entlastung der teuren Zentralrechner von den zeit- und speicherplatzaufwendigen elementaren Grafikoperationen durch Verlagerung auf einen preiswerten Mikrorechner in das Grafikterminal, wodurch die Qualität des Mensch-Maschine-Dialogs gleichzeitig verbessert werden konnte. Mit der weiteren Entwicklung der Workstation-Technologie vergrößerten sich ständig die Forderungen nach immer mehr Computerleistung, nach größeren Bildschirmen und immer höherer Auflösung der Bildschirme am Arbeitsplatz des Ingenieurs.

Betrachtet man heute das Angebot an Workstations auf dem Weltmarkt, muß man feststellen, daß die Produktpalette sehr breit ist. An der unteren Leistungsgrenze fließen verstärkt die leistungsstarken Personalcomputer und an der oberen Leistungsgrenze Workstations auf der Basis von Kleinrechnern ein.

Die Beantwortung der Frage nach einer Definition des Begriffes „Workstation" erweist sich bei dem gegenwärtigen enormen Entwicklungstempo als recht kompliziert. In den USA wurde vor etwa 5 Jahren der Begriff einer 5M-Maschine geprägt, worunter man die Mindestanforderungen verstand:

- 1 MIPS (Million Befehle/Sekunde) Rechenleistung,
- 1 MByte Arbeitsspeicher,

- 1 Million Bildpunkte (Pixel) auf dem Grafik-Bildschirm,
- 1 MBit/s Datenübertragungsrate,
- Mehrfenstertechnik (Multiwindowing).

Während diese Anforderungen vor einigen Jahren für viele Systeme noch unüberwindbar waren, werden sie heute schon von vielen Personalcomputern erreicht.
Bei den Workstations der oberen Leistungsklasse spricht man heute oft schon von der 10M-Maschine, deren Rechnerleistung bis zu 10 MIPS, der Hauptspeicher 8–16 MByte und die Bildschirme bis zu 4096 × 4096 Pixel haben. Als Quasi-Industrie-Standard für die Kommunikation hat sich Ethernet mit 10 MBit/s herausgestellt. In dieser Klasse haben sich eindeutig die 32-Bit-Mikroprozessoren wie MC 68020, MC 68030 und Intel 80386 durchgesetzt, die jetzt zunehmend durch RISC-Prozessoren mit einer vereinfachten Prozessorarchitektur und höherer Verarbeitungsleistung verdrängt werden.

2. Basissoftware für Workstations

Mit dem zunehmenden Einsatz von 32-Bit-Mikroprozessoren bzw. RISC-Prozessoren in Workstations hat sich eindeutig als Basisbetriebssystem das Betriebssystem UNIX wegen seiner höheren Flexibilität in der Mensch-Maschine-Kommunikation durchgesetzt. Die meisten Hersteller orientierten dabei auf Systeme, die zum Berkeley-UNIX BSD 4.3 kompatibel sind. Als zweite Hauptrichtung haben sich verstärkt UNIX-Systeme entsprechend der System-V-Interface-Definition von AT&T bzw. den Empfehlungen der X/OPEN-Gruppe durchgesetzt.
Auf den leistungsschwächeren Workstations wird für technische Anwendungen vorrangig die 2D-GKS-Software und auf den leistungsstärkeren Workstations die 3D-GKS-Software angeboten. Für die Windowtechnik wird allgemein die X-Window-Software auf der Basis des X11-Window-Standards bereitgestellt.

3. Haupteinsatzgebiete von Workstations

Das Haupteinsatzgebiet von Workstations war in der Vergangenheit vor allem das rechnergestützte Konstruieren (CAD – Computer Aided Design) in fast allen ingenieurtechnischen Gebieten.
Zu den neueren Anwendungsgebieten gehören vor allem:

- rechnergestützte Softwareentwicklung (CASE – Computer Aided Software Engineering),
- Expertensysteme (KI-Workstations) für fast alle Wissensgebiete,
- Büroautomatisierung (Dokumentenbearbeitung, Informations- und Archivierungssysteme),
- Desk-Top-Publishing (Textverarbeitung, Schriftsatzaufbereitung, Layoutgestaltung für Verlage und Druckereien).

Zweifellos sind damit auch noch lange nicht alle Anwendungsgebiete erschlossen.

Prof. Dr. sc. techn. Thomas Horn

BASIC-Tricks und -Kniffe

Nehmen wir an, ein Verkaufsstellenleiter summiert allabendlich seine Tagesumsätze auf, die zwischen acht- und neuntausend Mark liegen sollen. Bei rund 250 Öffnungstagen im Jahr ergibt das einen Jahresumsatz von über 2 Millionen Mark. Wenn er dazu einen 6stelligen BASIC-Interpreter benutzt, dann wird ihm z. B. anstelle von 2182858,65 M der Betrag 2.18286E6 angezeigt. Ein Revisor würde sich mit dieser Zahlenangabe sowohl von der Darstellungsform als auch von der Genauigkeit her wohl nicht zufrieden geben.
Wie können wir den Interpreter dazu bringen, auch zwei zehnstellige Zahlen exakt zu addieren? Wir können dazu ein Programm in Maschinensprache schreiben und es über BASIC aufrufen. Oder wir nutzen die Anweisungen zur Textverarbeitung, die BASIC bietet. Im folgenden Programm haben wir den letzten Fall gewählt. Die Gültigkeitsbedingungen gehen aus dem Programmtext hervor.

```
10 REM ADD. ZEHN9T. ZAHLEN
20 PRINT "ADDITION VON 2 ZAHLEN!"
30 PRINT "MAX. 10 STELLEN, POSITIV"
40 PRINT "BEIDE ZAHLEN MIT 2 KOMMASTELLEN"
50 PRINT "EINGEBEN!"
60 INPUT "1. ZAHL"; A$: LET C$= A$;GOSUB 300
70 IF FE=1 THEN GOTO 60
80 INPUT"2.ZAHL="; B$ : LET C$=B$ : GOSUB 300
90 IF FE=1 THEN GOTO 80
100 LET C = 0 : LET E$= ""
110 FOR I=0 TO 11
120 IF I=2 THEN LET D$="." : GOTO 210
130 IF LEN(AS)—I<1 THEN LET A=0:GOTO 150
140 LET A=VAL(MID$(A$,LEN(A$)—I,1))
150 IF LEN(B$)-I<1 THEN LET B=0;GOTO 170
160 LET B=VAL(MID$(B$,LEN(B$)—I,1))
170 LET D=A+B+C
180 IF D>9 THEN LET C=1; LET D=D—10; GOTO 200
190 LET C=0
200 LET D$=STR$(D)
210 LET E$=RIGHT$ (D$,1)+E$
220 NEXT I
230 REM AUSGABE
240 CLS
250 PRINT TAB(12—LEN(A$)); A$
260 PRINT TAB(12—LEN(B$,); B$
270 PRINT STRING$(12,"—")
280 PRINT TAB(12—LEN(E$)); E$
290 STOP
300 REM UP EINGANGSPRUEFUNG
310 LET FE=0
320 IF LEN(C$)>11 OR LEN(C$)<3 THEN LET FE=1:RETURN
330 IF MID$(C$,LEN(C$)—2,1)<>"." THEN LET FE=1:RETURN
340 RETURN
```

Das Programm kann natürlich auch auf eine Mehrfachaddition erweitert oder in bestehende Programme eingebunden werden. Zu beachten ist dabei, daß Ein- und Ausgabewerte in Wirklichkeit Zeichenketten sind.

Dr. Hannes Gutzer, Dirk Jahn

Hinweise für Autoren

Herausgeber und Verlag danken den Lesern für das Interesse an den »Kleinstrechner-TIPS«, das sich in zahlreichen Zuschriften und Veröffentlichungsangeboten äußert. Beim Einsenden von Artikeln bitten sie folgende Hinweise zu beachten:

- In den »Kleinstrechner-TIPS« werden Artikel aus den auf der 4. Umschlagseite angegebenen Gebieten veröffentlicht.
- Manuskripte sind zweizeilig mit schwarzem Farbband mit Schreibmaschine zu schreiben, ä, ö und ü dürfen nicht durch ae, oe und ue ersetzt werden.
- Bilder sollten auf getrennten Blättern gezeichnet sein. Eine Bildunterschriftenliste ist beizufügen.
- Rechnerausdrucke (z.B. Programme) sollen tiefschwarz mit guter Ausnutzung des Formats (engzeilig, kein zu breites Kommentarfeld usw.) gedruckt sein.
- Die Autorenangabe soll enthalten: akadem. Grad, Vorname, Name, Tätigkeit, Arbeitsstelle, Privatanschrift.

Manuskripte mit einem Umfang von nicht mehr als 15 Schreibmaschinenseiten (einschließlich Bilder und Programme) sind in zwei Exemplaren an einen der Herausgeber zu senden (Anschrift s. unten).

ISBN 3-343-00480-4

1. Auflage
Lizenznummer 114—210/2/89
LSV 1083
Verlagslektor: Helga Fago
Printed in GDR
Satz und Druck:
Messedruck Leipzig, Bereich Borsdorf
III-18-328
Redaktionsschluß: 15. 8. 1989
Bestellnummer: 547 5085
00780

Anschrift des Verlages:
VEB Fachbuchverlag Leipzig
PSF 67
Leipzig, DDR – 7031

Kleinstrechner-TIPS/Hrsg. von
Hans Kreul u.a. – Leipzig: Fachbuchverl.,
H. 10. – 1. Aufl. – 1989. – 64 S.: 38 Bild.

Herausgeber:
Prof. Dr.-Ing. Hans Kreul, Technische Hochschule Zittau, Abt. EDV und Rechentechnik, Theodor-Körner-Allee 16,
Zittau, 8800;
Prof. Dr. sc. techn. Thomas Horn,
Doz. Dr.-Ing. Wilhelm Leupold,
Informatik-Zentrum des Hochschulwesens an der Technischen Universität Dresden,
Mommsenstr. 13, Dresden, DDR – 8027

Vorschau auf die nächsten Hefte

Lorenz/Schulze: Simulation auf Mikrorechnern (Teil 3)

Bockhacker: Programm zum chemischen Rechnen

Löhr: Elemtare Behandlung von Schwingungen mit dem Kleincomputer KC 85/2

Köhler: Elektronische Schreibmaschine S 6005 als Drucker

Kreul: Beherrschen Sie BASIC?

Görgens: Spielprogrammierung

Die Broschürenreihe

KLEINSTRECHNER-TIPS

behandelt

- Tendenzen und Theorien
- Informationen und Ideen
- Programme und Projekte
- Spaß und Spiel

und stellt sich das Ziel

- den Nutzer der Mikrorechentechnik aus allen Bereichen der Volkswirtschaft und dem Bildungswesen bei der Einarbeitung in die Informatik und Computertechnik zu unterstützen
- Entwicklungstendenzen der Informatik und Computertechnik vorzustellen und zur Erweiterung des Grundwissens beizutragen
- Anregungen für den Computereinsatz zu geben und Beispielprogramme für Kleincomputer zu veröffentlichen

um somit einem großen Kreis von Freunden der Informatik und Computertechnik zu helfen, sich moderner Hilfsmittel und Methoden zu bedienen.

ISBN 3-343-00480-4