Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 41

Banana-Interface

Maschinenroutinen

Wren Executive

Kalkulations-System

Tips: Erklärende Zeilen

Kommunikation mit BASICODE

Ein wöchentliches Sammelwerk

computer kurs

Heft 41

Inhalt

WIE SIE JEDE WOCHE IHR HEFT BEKOMMEN

Computer Kurs ist ein wöchentlich erscheinendes Sammelwerk. Die Gesamtzahl der Hefte ergibt ein vollständiges Computer-Nachschlagewerk. Damit Sie jede Woche Ihr Heft erhalten, bitten Sie Ihren Zeitschriftenhändler, Computer Kurs für Sie zu reservieren.

Zurückliegende Hefte
Ihr Zeitschriftenhändler besorgt Ihnen gerne zurückliegende Hefte. Sie können sie aber auch direkt beim Verlag bestellen.

Deutschland: Das einzelne Heft kostet DM 3,80. Bitte füllen Sie eine Postzahlkarte aus an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Computer Kurs

Österreich: Das einzelne Heft kostet öS 30. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs, Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Computer Kurs.

Schweiz: Das einzelne Heft kostet sfr 3,80. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

Abonnement

Sie können Computer Kurs auch alle 2 Wochen (je 2 Ausgaben) per Post zum gleichen Preis im Abonnement beziehen. Der Abopreis für 12 Ausgaben beträgt DM 45,60 inkl. Mwst., den wir Ihnen nach Eingang der Bestellung berechnen. Bitte senden Sie Ihre Bestellung an: Marshall Cavendish Int. Ltd. (MCI), Sammelwerk Service, Postgiroamt Hamburg 86853-201, Postfach 105703, 2000 Hamburg 1, Kennwort: Abo Computer Kurs. Bitte geben Sie an, ab welcher Nummer das Abo beginnen soll und ob Sie regelmäßig für jeweils 12 Folgen einen Sammelordner wünschen.

WICHTIG: Bei Ihren Bestellungen muß der linke Abschnitt der Zahlkarte Ihre vollständige Adresse enthalten, damit Sie die Hefte schnell und sicher erhalten. Überweisen Sie durch Ihre Bank, so muß die Überweisungskopie Ihre vollständige Anschrift gut leserlich enthalten.

SAMMELORDNER
Sie können die Sammelordner entweder direkt bei Ihrem Zeitschriftenhändler kaufen (falls nicht vorrätig, bestellt er sie gerne für Sie) oder aber Sie bestellen die Sammelordner für den gleichen Preis beim Verlag wie folgt:

Deutschland: Der Sammelordner kostet DM 12. Bitte füllen Sie eine Zahlkarte aus an: Marshall Cavendish International Ltd. (MCI), Sammelwerk-Service, Postgiroamt Hamburg 48064-202, Postfach 105703, 2000 Hamburg 1, Kennwort: Sammelordner Computer Kurs.

Österreich: Der Sammelordner kostet öS 98. Bitte füllen Sie eine Zahlkarte aus an: Computer Kurs Wollzeile 11, 1011 Wien, Postscheckkonto Wien 7857201 oder legen Sie Ihrer Bestellung einen Verrechnungsscheck bei. Kennwort: Sammelordner Computer Kurs

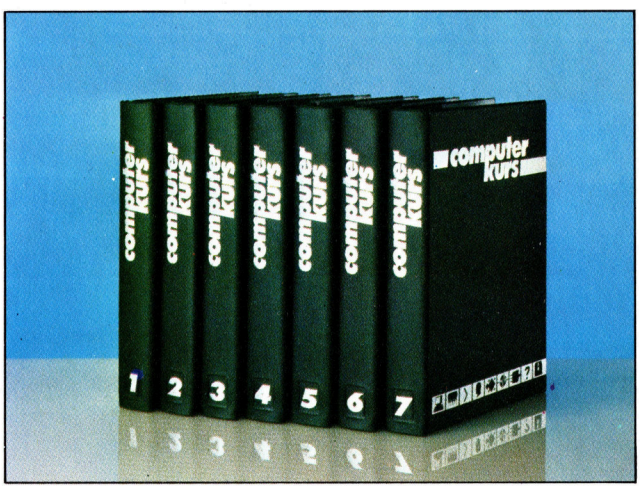
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

INHALTSVERZEICHNIS

Alle 12 Hefte erscheint ein Teilindex. Die letzte Ausgabe von Computer Kurs enthält den Gesamtindex — darin einbezogen sind Kreuzverweise auf die Artikel, die mit dem gesuchten Stichwort in Verbindung stehen.

Redaktion: Winfried Schmidt (verantw. f. d. Inhalt), Elke Leibinger, Susanne Brandt, Uta Brandl (Layout), Sammelwerk Redaktions-Service GmbH, Paulstraße 3, 2000 Hamburg 1

Vertrieb: Marshall Cavendish International Ltd., Heidenkampsweg 74, 2000 Hamburg 1



Druck: E. Schwend GmbH, Schmollerstraße 31, 7170 Schwäbisch Hall

Gemeinsamer Nenner

Eines der Haupthindernisse, dem sich Heimcomputerbesitzer, die Listings tauschen wollen, ausgesetzt sehen, ist der Mangel an Kompatibilität. BASICODE, eine in Holland entwickelte Programmiersprache, versucht dieses Problem zu lösen.

BASICODE erlaubt die Kommunikation von Rechnern untereinander durch einen gemeinsamen Standard. Dabei finden nur wenige BASIC-Befehle Verwendung sowie ein eigenes, auf Cassette befindliches Format, mit dem der Programmablauf auf zahlreichen Rechnern möglich wird. BASICODE-Programme werden auch von vielen Rundfunkanstalten ausgestrahlt. Besitzern unterschiedlicher Computer ist so die Benutzung ein und desselben Programms möglich.

BASIC selbst ist zur Standardsprache für Heimcomputer geworden. Doch wie jeder Computerbesitzer weiß, gibt es eine Fülle von BASIC-Dialekten. Selbst wenn Maschinen mit dem Microsoft-BASIC betrieben werden, bedeutet das noch längst keine Garantie dafür, daß ein auf einem Computer geschriebenes Programm auf einem anderen ebenfalls läuft.

BASICODE ist ein Versuch, das Kompatibilitätsproblem zu lösen. Es wurde zunächst in Holland für Hobbyscoop entwickelt, ein von Teleac produziertes Wissenschafts- und Technologie-Programm – dem holländischen Gegenstück zur „Volkshochschule". Als 1978 erstmals Hobbyscoop-Programme gesendet wurden, liefen diese auf den damals vier populären Rechnern – dem Apple, Exidy Sorcerer, Commodore PET und Tandy TRS-80. Pro Woche konnten nur zwei Übertragungen für einen Rechner stattfinden, und die Zuhörer mußten bis zu acht Minuten warten, da zwei dieser Computer extrem langsame Datenübertragungsraten hatten. Dies war natürlich unbefriedigend, und da für jeden neuen Rechner, der auf den Markt kam, eine eigene zusätzliche Sendung erforderlich wurde, erwies sich diese Art der Programmübertragung als unmöglich.

Der Radio-Amateur Claas Roberts, der die erste Version des BASICODE schrieb, befaßte sich mit diesem Problem. Seine Sprache basierte auf den gemeinsamen Befehlen, die alle BASIC-Dialekte enthielten und somit von allen Rechnern verstanden werden konnten. Dennoch gab es auch bei diesem Verfahren Probleme, da die verschiedenen Rechner zwar identische Befehle hatten, diese aber unterschiedlich ausführten. So entwickelte Claas Roberts gemeinsam mit Jochen Herrmann eine zweite, verbesserte Version der Sprache mit der Bezeichnung BASICODE-2.

Die erste Ausstrahlung eines BASICODE-2-Programms erfolgte zum Neujahrstag 1983 und war ein Riesenerfolg. Zuhörer, die in Belgien, Frankreich, England, Deutschland und Dänemark dabei waren, berichteten über den korrekten Empfang der ausgestrahlten Pro-

Das BASICODE-2-Handbuch mit dazugehöriger Cassette kann unter folgender Adresse angefordert werden:
NOS-Hobbyscoop
Postbus 1200
NL-1200 BE Hilversum
Niederlande

gramme. Die internationale Aufmerksamkeit nahm zu, als der niederländische Rundfunk BASICODE-2 auch über sein Auslandsprogramm ausstrahlte.

BASICODE-2 basiert auf den 42 Schlüsselwörtern und elf Symbolen, die die meisten Rechner gemeinsam haben. Ein BASIC-Schlüsselwort (oder Befehl) ist nicht aus Zeichen zusammengesetzt gespeichert, sondern in Form eines einzelnen Bytes, eines Symbols, das für einen Befehl steht. So wird beispielsweise der Befehl LEFT$ auf dem Commodore 64 durch ein einzelnes Byte, das den Wert 200 hat, dargestellt und nicht durch fünf Bytes, die die entsprechenden ASCII-Werte für L, E, F, T und $ enthalten. Dadurch wird die Arbeit des BASIC-Interpreters effizienter, und es wird weniger Speicherplatz benötigt.

Jedoch verwendet jeder Computer für das Speichern und die Interpretation der Programme eine unterschiedliche Codierung. Das Problem wurde durch zwei Übersetzungsprogramme gelöst, nämlich BASICODE-Save und BASICODE-Load. Nachdem ein Programm in BASIC geschrieben ist, wird es unter Verwendung des BASICODE-SAVE-Programms geSAVEd, das die spezielle BASICODE-Standard-Umwandlung für die Codierung des entsprechenden Rechners vornimmt und so auf Band ein Standard-BASICODE-Programm erzeugt.

Daraus resultiert die Frage, wie man sicherstellen kann, daß die unterschiedlichen Computer das Band auf dieselbe Art und Weise lesen bzw. beschreiben. Dabei ist nicht nur zu beachten, daß die Daten mit unterschiedlicher Übertragungsgeschwindigkeit gelesen oder geschrieben werden, es kann ebenso erhebliche Unterschiede bei den Start- und Stop-Bits geben und bei den Checksum-Verfahren. Die Schlußfolgerung war, für die Übertragung einen gemeinsamen Audiocode als Format zu schaffen.

In diesem Format werden Daten mit einer Geschwindigkeit von 1200 Bits pro Sekunde übertragen. Jedem Daten-Byte wird ein Startbit vorangestellt, danach folgen die acht Informations-Bits und zwei Stop-Bits.

Eine Markierung – bestehend aus einer Sequenz von Stop-Bits, die über fünf Sekunden übertragen wird, zeigt den Beginn eines BASICODE-Programms an. Darauf folgt die Codierung für „Textbeginn" (82 als Hexidezimalwert). Dem BASICODE-Programm folgt zwecks Prüfung der übertragenen Daten ein Prüf-Byte. Eine weitere fünf Sekunden dauernde Stop-Bit-Sequenz beendet die Datenübertragung.

Zwar sind fast alle Rechner allein durch die Software BASICODE-fähig, für die TRS-80-Modelle I und III sowie das Video Genie ist aber zusätzlich ein kleines Interface zum korrekten Lesen von Programmen auf Cassette erforderlich. Das mit der BASICODE-2-Cassette gelieferte Handbuch zeigt in allen Einzelheiten, wie dieses Interface gebaut wird. Wer weniger elektronisches Bastlergeschick hat, kann von der holländischen TRS-80-User-Group eine fertige Steckplatine beziehen.

Um ein BASICODE-Programm schreiben zu können, muß zuerst das BASICODE-Save-Programm geladen werden. Dieses Programm läßt nicht nur das SAVEn des neugeschriebenen Programms in einem Standard-Format zu, sondern lädt auch eine Reihe von Subroutinen in den Computer, die nur für diesen speziellen Rechner gelten. Diese Routinen sind in den Zeilen von 0 bis 999 gespeichert.

Diese Routinen sind Bestandteil des BASICODE-2-Übersetzungsprogramms, weil ein für mehrere Rechnertypen gültiger Befehl – wie etwa das Löschen (CLS) des Bildschirms – auf verschiedene Art ausgeführt wird. Statt den CLS-Befehl zu verwenden, benutzt der Programmierer GOSUB 100, womit auf die BASICODE-Routine zurückgegriffen wird, die diese Funktion ausführt.

Einheitlicher Start

Die erste Programmzeile sollte folgendermaßen geschrieben sein:

1000 A=(Wert):GOTO 20:REM Titel

wobei (Wert) die maximale Zeichenzahl darstellt, die von allen Strings gemeinsam verwendet werden kann. Danach hat der Anwender bei der Programmierung freie Hand. Dennoch gibt es eine Reihe von Beschränkungen, die durch das Format des Code bedingt sind. So müssen beispielsweise alle Variablen am Anfang des Programms initialisiert werden.

Ebenso gibt es einige Beschränkungen bei der Anwendung verschiedener BASIC-Befehle. Beispiel:

5000 INPUT "PASSWORD?";A$

ist in BASICODE-2 falsch. Das korrekte Format lautet:

5000 PRINT "PASSWORD?":INPUT A$

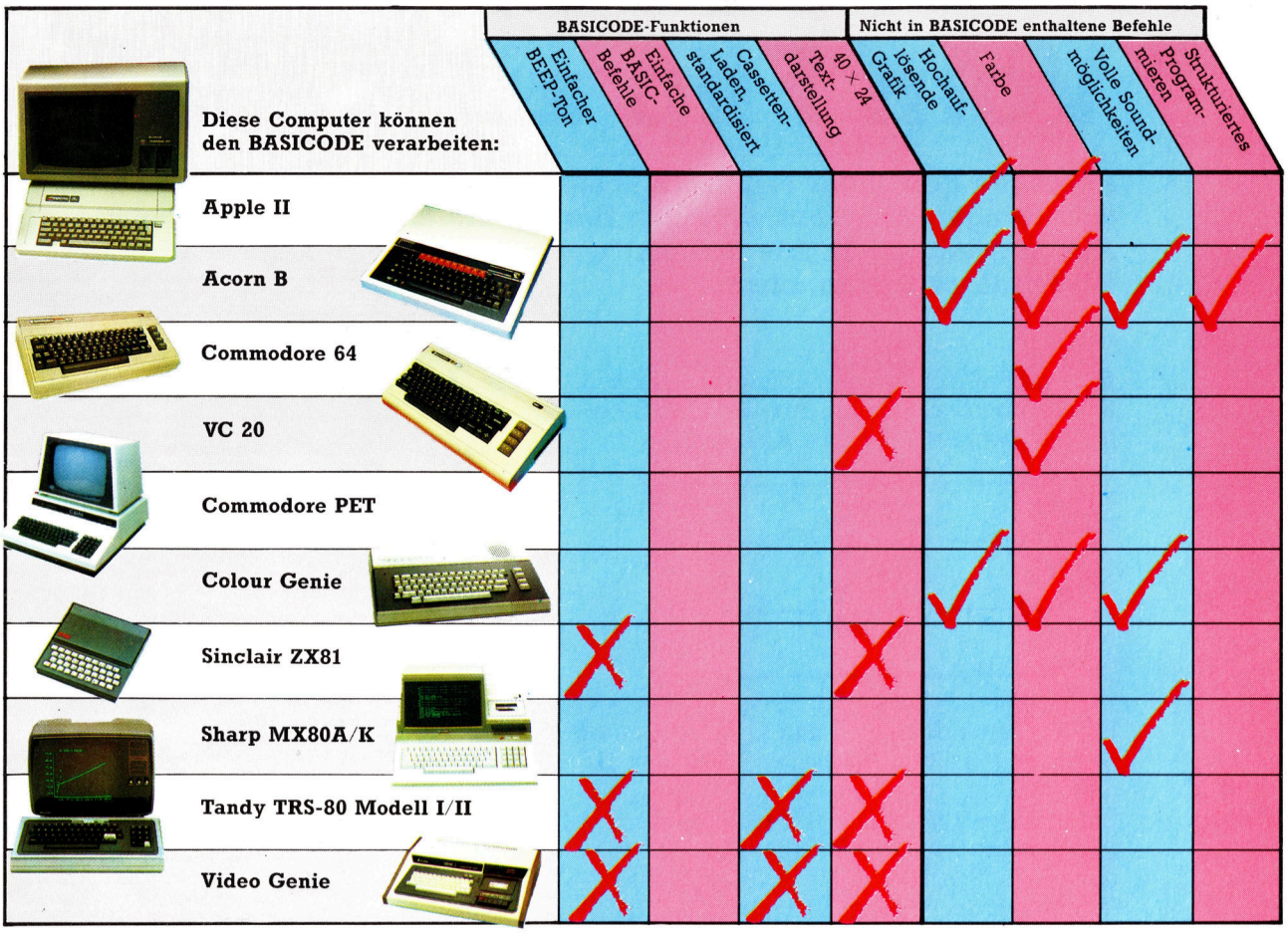
Darüber hinaus dürfen Programmzeilen nicht mehr als 60 Zeichen pro Zeile haben, da eine Bildschirmdarstellung von 24 Zeilen mit je 40 Zeichen zugrundegelegt ist.

Hier ist es interessant, darüber nachzudenken, warum all diese Beschränkungen erforderlich sind. Um soviele Rechner wie möglich BASICODE-lauffähig zu machen, mußte man versuchen, den „kleinsten gemeinsamen Nenner" zu finden. Daraus resultierte ein Kompromiß zwischen den eigentlichen Möglichkeiten des BASICODE und der Zahl der damit arbeitenden Computer. Das führte dazu, daß die Leistung schwächerer Computer sprachbestimmend war und weiterentwickelte Systeme nicht voll genutzt werden können.

Viele der Möglichkeiten, die ein Heimcomputer-Anwender bei der Auswahl für einen Computer zugrundelegt, finden im BASICODE-Format keine Anwendung. So gibt es zum Beispiel keine Möglichkeit, Höhe und Dauer von

Operations-Befehle
Nachstehend ist eine Liste von Befehlen und Operationen aufgeführt, die in BASICODE verwendet werden können. Dabei ist zu beachten, daß viele Rechner über weit mehr Befehlswörter verfügen, die aber im BASICODE nicht erkannt werden.

ABS	NEXT
AND	NOT
ASC	ON
ATN	OR
CHR$	PRINT
COS	READ
DATA	REM
DIM	RESTORE
END	RETURN
EXP	RIGHT$
FOR	RUN
GOSUB	SGN
GOTO	SIN
IF	SQR
INPUT	STEP
INT	STOP
LEFT$	TAB
LEN	TAN
LET	THEN
LOG	TO
MID$	VAL
+	<
-	>
*	<>
/	<=
^	>=



Die Übersicht zeigt einige der Computer, die das BASICODE-Format verarbeiten können. Rechner, die die Bedingungen des Standards nicht erfüllen, sind mit einem Kreuz versehen. Die mit einem Haken versehenen Rechner bieten zusätzliche Möglichkeiten, die mit BASICODE nicht ausgeschöpft werden können.

Klängen zu modifizieren. Es gibt nur den vergleichsweise einfachen BEEP-Befehl, mit dem man arbeiten kann. Ähnlich ist es mit den Grafikmöglichkeiten: BASICODE erlaubt das Programmieren nur im niedrig auflösenden Mode. Und hierbei kann auch nur in Schwarzweiß programmiert werden.

Es gibt noch ein weiteres Problem: Seit der Einführung von BASICODE hat es beträchtliche Weiterentwicklungen beim strukturierten Programmieren in BASIC gegeben. Beim BASICODE sind aber Statements wie WHILE . . . WEND nicht erlaubt. Die Strukturierung hängt weitgehend vom GOSUB-Befehl ab, womit die Sprache selbst den Programmablauf bestimmt.

Ein entschlossener oder begeisterter Programmierer wird indes das Programmieren in BASICODE als Herausforderung empfinden. Gerade wegen der gegebenen Beschränkungen muß sehr darauf geachtet werden, das Programm wirklich übertragbar bzw. lauffähig zu machen. Der Programmierer hat sich stets zu vergegenwärtigen, daß er lediglich circa 50 Befehle zur Verfügung hat. Ferner, daß er GOSUB-Befehle statt der nicht standardisierten Anweisungen wie etwa CLS verwendet. Ferner ist zu berücksichtigen, daß viele der BASICODE-geeigneten Rechner über eine sehr geringe Speicherkapazität verfügen. Es ist durchaus möglich, ein Programm zu erstellen, das auf dem eigenen Rechner läuft. Beim Test aber – nämlich beim SAVEn mit anschließendem Laufversuch auf einem anderen Rechner – würde das Programm an kleinsten Abweichungen scheitern.

Zusätzliche Features

Innerhalb des Hauptprogramms kann der Programmierer gewisse Features, die sonst nicht möglich sind, ergänzen. Das ist durch Hinzufügen von REM-Statements möglich, die genau erläutern, was der Programmierer beabsichtigt. Die BASICODE-Autoren empfehlen die Eingliederung dieser Statements in den Programmzeilen 20 000 bis 24 999.

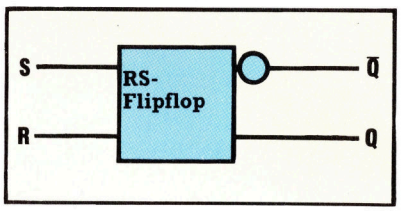
Detaillierte Anweisungen, wie BASICODE-2 zu benutzen ist, sind in dem Begleitbuch enthalten, das mit dem Programmpaket ausgeliefert wird. Der Benutzer erhält eine Cassette mit dem Übersetzungsprogramm für die unterschiedlichen Rechner auf der einen Seite. Obwohl für die meisten Computer nur ein einziges BASICODE-2-Übersetzungsprogramm erforderlich ist, gibt es für den Acorn B und den VC 20 verschiedene Load- und Save-Programme. Auf Seite zwei der Cassette sind 18 Demo-Programme enthalten, die verdeutlichen, was BASICODE alles zu leisten vermag.

Angesichts der schier unüberschaubaren Vielfalt von Versuchen, eine Standardsprache zu schaffen, ist es den BASICODE-Autoren gelungen, tatsächlich viele Rechner unter einen Hut zu bekommen. Ein beachtlicher Erfolg.

Vom Flip zum Flop

Entgegen den zuvor erklärten Schaltkreisen produzieren sequentielle Schaltungen auf einen einzigen Impuls hin eine stetige Kette von Ausgangssignalen. Als Beispiel wollen wir die Funktionsweise eines RS-Flipflops untersuchen.

Es gibt verschiedene Flipflop-Schaltungen, die jedoch alle ähnlich arbeiten. Ein RS-Flipflop hat zwei Ein- und zwei Ausgänge.

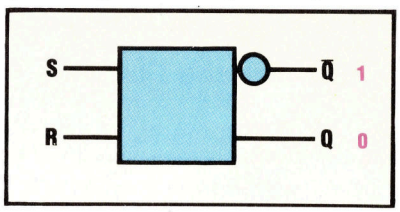


Die beiden Ausgänge Q und $\bar{Q}$ weisen immer einen gegensätzlichen Zustand auf, das bedeutet

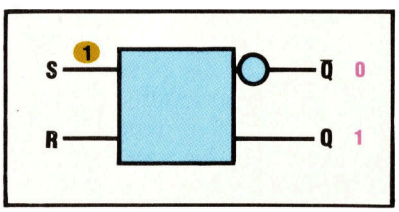
wenn Q = 1 dann $\bar{Q}$ = 0 (Set-Zustand)
wenn Q = 0 dann $\bar{Q}$ = 1 (Reset-Zustand)

Wenn sich der Flipflop zu Beginn im RESET-Zustand befindet, schaltet ihn ein Impuls am S-Eingang auf den SET-Zustand um.

1) Anfangszustand (RESET)

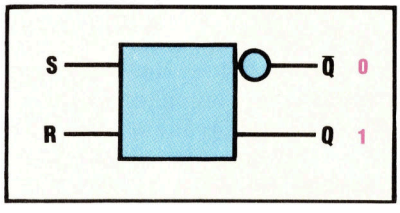


2) Impuls auf dem SET-Eingang



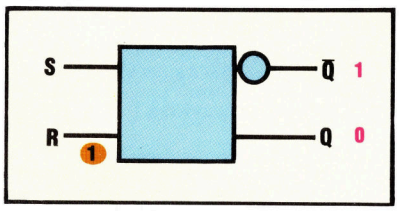
Die Schaltung bleibt auch nach dem Ende des Impulses auf Eingang S im Set-Zustand.

3) Schaltung bleibt im SET-Zustand

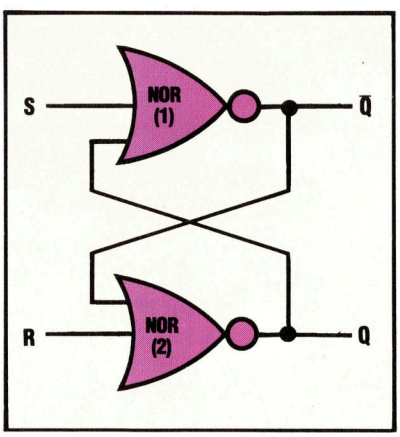


Ein weiterer Impuls auf Eingang R versetzt die Schaltung zurück in den alten RESET-Zustand.

4) Impuls auf dem R-Eingang

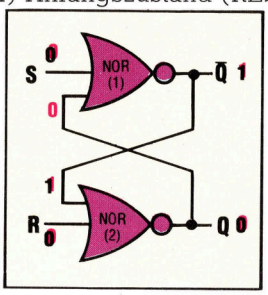


Wie sieht nun die Schaltungslogik aus, die das Flipflop-Verhalten ermöglicht? Es gibt mehrere Verfahren, eine Flipflop-Schaltung aufzubauen: die Verbindung zweier NAND-Gatter etwa oder, wie in dieser Zeichnung, das Zusammenschalten zweier NOR-Gatter, bei denen die beiden Ausgänge mit jeweils einem Eingang des anderen Gatters verbunden sind. Die „Speicherfähigkeit" der Schaltung beruht auf dieser Rückkopplung.

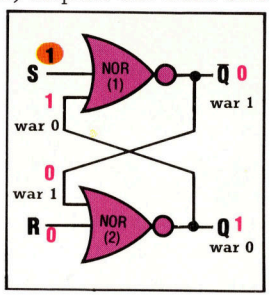


Versuchen wir, das SET- und RESET-Verhalten des Flipflops aus der Schaltung zu erklären: Angenommen, der Flipflop ist im RESET-Zustand. Ohne Eingangsimpulse bleibt er dann in diesem Zustand stabil (ein NOR-Gatter gibt nur dann eine 1 aus, wenn beide Eingänge auf 0 liegen). Ein Impuls auf Eingang S kippt den stabilen Zustand und schaltet den „NOT Q"-Ausgang auf 0. Der Ausgang ist mit dem Eingang des zweiten NOR-Gatters verkoppelt, und der Ausgang Q dieses Gatters schaltet daher auf 1. Ist der Impuls am ersten NOR-Gatter (1) noch vorhanden, liegen beide Eingänge auf 1, und der Ausgang des Gatters (1) bleibt weiter auf 0. Damit hat sich die Schaltung im SET-Zustand stabilisiert.

1) Anfangszustand (RESET)

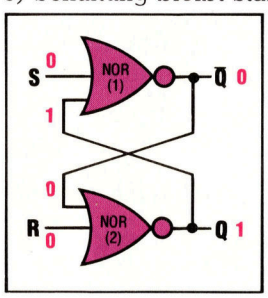


2) Impuls auf dem SET-Eingang

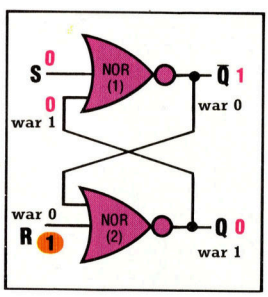


Die Schaltung bleibt auch dann im neuen Zustand, wenn der Eingangsimpuls vorbei ist. Erst ein Signal auf dem R-Eingang destabilisiert sie kurz, führt dann aber sofort wieder zu einem dauerhaften RESET-Status.

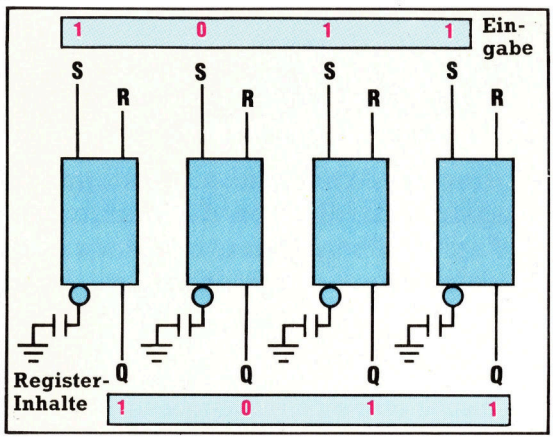
3) Schaltung bleibt stabil (SET)



4) Impuls auf dem RESET-Eingang

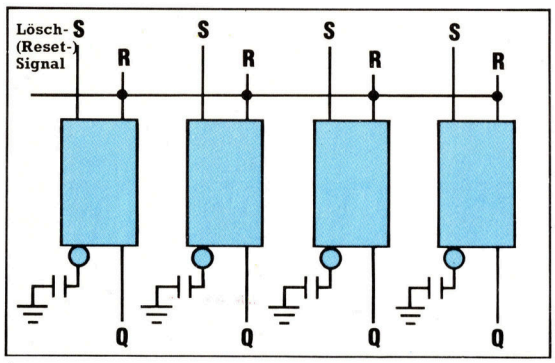


Der Microprozessor Ihres Computers besteht größtenteils aus einer Vielzahl von Speichern bzw. Registern wie dem Akkumulator, den Befehls- und Index-Registern. Die meisten Register können Acht-Bit-Wörter speichern – also achtstellige Binärzahlen mit dezimalen Werten von 0 bis 255. Diese Speicherung wird durch eine Kette aus acht Flipflops möglich. Um etwa die Binärzahl 1011 darin zu speichern, muß das entsprechende Bitmuster auf die S-Eingänge gegeben werden.



Bei dieser Anordnung ist der „NOT Q"-Ausgang unbenutzt. Da das Bitmuster über den S-Eingang zugeführt wird, steht es am Q-Ausgang zur Verfügung. Zum Überschreiben einer im Register gespeicherten Zahl mit einem neuen Wert – etwa 0110 – reicht es nicht aus, einfach nur das neue Muster über den S-Eingang einzugeben. Dabei würden in diesem Fall die beiden äußeren Einsen erhalten bleiben, und der falsche neue Wert wäre 1111.

Die Lösung des Problems liegt darin, alle Flipflops vor einer neuen Belegung zu löschen. Da dies bei allen Registerstellen gleichzeitig passieren muß, werden die Flipflops verbunden, so daß zum Löschen nur ein einziges Signal nötig ist.



Im nächsten Kursabschnitt werden Sie weitere sequentielle Schaltungen kennenlernen, unter anderen auch den D- und den JK-Flipflop.

Übung 7

1) Warum bezeichnet man Flipflops als „bistabil"?

2) Beim Einschalten eines Computers befindet sich ein Flipflop im folgenden Zustand:
$Q = 0, \bar{Q} = 0, S = 0, R = 0$

a) Ist dies ein stabiler Zustand?

b) Falls nicht, in welchen Zustand wird der Flipflop übergehen?

c) Kann der Flipflop auch in einen anderen Zustand als den in Ihrer Lösung zu b) übergehen? (Tip: Beginn beim anderen Gatter!)

d) Was muß unternommen werden, damit sich nach dem Einschalten eines Computers alle Register in einem vorhersehbaren Zustand befinden?

Per Tastendruck

In diesem Artikel über das Spectrum-BASIC wird erklärt, wie man Eingaben direkt von der Tastatur ablesen kann. Wir zeigen ein einfaches Programm, mit dem man einen Grafik-Cursor mittels der Tastatur über den Bildschirm bewegt.

Die Tastatur läßt sich mit den Anweisungen INPUT und INKEY$ abfragen. Die INPUT-Anweisung liest eine eingegebene Zahl oder eine Zeichenkette, die mit ENTER abgeschlossen wurde, in eine numerische oder eine alphanumerische Variable ein. Die INKEY$-Funktion fragt die Tastatur ab und gibt als Ergebnis einen String mit dem Zeichen der gedrückten Taste oder, wenn keine Taste gedrückt wurde, einen leeren String aus. Ferner kann die IN-Funktion verwendet werden, um die Tastatur direkt abzulesen.

Insgesamt gibt es beim Spectrum 65536 Ein-/Ausgabe-Kanäle, von denen jeder einzeln adressiert werden kann. In derselben Form, in der PEEK und POKE zum Lesen und Schreiben in den Speicher verwendet werden, werden IN und OUT zum Lesen und Schreiben in bezug auf die Ein-Ausgabe-Kanäle verwendet. Die Funktion IN(m) gibt als Ergebnis den Wert von Kanal m aus, wogegen die Anweisung OUT(m,n) den Wert n in Kanal m schreibt.

Die Tastatur besteht aus vier Reihen mit je zehn Tasten, wobei jede Reihe in zwei Hälften unterteilt ist. Jede Halbreihe wird auf einem Kanal dargestellt (siehe Bild). Beachten Sie, daß die Tasten der linken Halbreihen auf den niederwertigeren fünf Bits von rechts nach links dargestellt werden, wogegen die rechten Halbreihen entsprechend von links nach rechts zugeordnet sind.

Die rechten Bits eines Kanals sind jeweils einer Taste zugeordnet und enthalten den Wert 0, wenn die Taste gedrückt ist, und eine 1, wenn das nicht zutrifft. Die mit X gekennzeichneten Bits haben keine Bedeutung – sie können eine 1 oder eine 0 enthalten. Das bedeutet, daß der Wert von IN(m) nicht definiert ist. Dieses Problem kann gelöst werden, indem die oberen drei Bits mit Hilfe der folgenden Anweisung auf den Wert 0 gesetzt werden:

```
DEF FN a(p)=p-INT(p/32)*32
```

Dabei ist p der Wert innerhalb des Kanals. Die Division INT(p/32) ergibt eine Zahl zwischen 0

Die Spectrum-Tastatur besteht aus acht Blökken von je fünf Tasten, von denen jeder Block durch ein bestimmtes Byte, bzw. einen „Kanal", dargestellt wird. Jede Taste wird durch ein Bit seines Kanals repräsentiert. Wird eine Taste gedrückt, ändert sich der Wert des zugehörigen „Signal"-Bits von 1 (dem Normalzustand) auf 0. Im hier gezeigten Beispiel sind die Tasten Y, I und 0 gedrückt, wodurch der binäre Wert ihres Kanals, Byte 57342, XXX01001 entspricht –die Bits 5 bis 7 sind nicht definiert.

und 8. Multipliziert man diese Zahl mit 32, erhält man einen Wert, der die oberen drei Bits von p repräsentiert. Subtrahiert man diesen Wert von p selbst, werden diese drei Bits entfernt. Es bleibt eine ganze Zahl mit einem Wert von 0 bis 31, die die ersten fünf Bits von p repräsentiert.

Betrachten wir ein Beispiel. Wird die V-Taste gedrückt, so sieht der Inhalt von Kanal 65278 wie folgt aus:

X	X	X	0	1	1	1	1

FN a(IN(65278)) gibt somit als Ergebnis aus:
0 0 0 0 1 1 1 1 = 15
Werden Caps Shift und V zusammen gedrückt, enthält die Adresse folgende Werte:

X	X	X	0	1	1	1	0

Das Ergebnis von FN a(IN(65278)) wäre demnach der Wert 14.

Testen Sie einmal das folgende Programm:

```
 10 REM Gib die Werte der Kanaele aus
 20 REM Definiere die Masken-Funktion
 30 DEF FN a(p)=p-INT(p/32)*32
 40 REM Gib die Kanaele aus
 50 PRINT AT 1,1;"Kanal  Maskierter Wert"
 60 PRINT "32766",FN a(IN(32766))
 70 PRINT "49150",FN a(IN(49150))
 80 PRINT "57342",FN a(IN(57342))
 90 PRINT "61438",FN a(IN(61438))
100 PRINT "63486",FN a(IN(63486))
110 PRINT "64510",FN a(IN(64510))
120 PRINT "65022",FN a(IN(65022))
130 PRINT "65278",FN a(IN(65278))
140 FOR i=1 TO 250:NEXT i
150 CLS
160 GO TO 50
```

Geben Sie RUN ein. Wenn Sie nun verschiedene Tasten drücken, können Sie beobachten, wie sich die Werte ändern. Beim Drücken der Cursor-Tasten (5, 6, 7 und 8), sollten Sie auf den Kanälen 61438 und 63486 die folgenden Werte erhalten:

	Kanal A (Adresse 61438)	Kanal B (Adresse 63486)
←	31	15
↓	15	15
↑	23	15
→	27	15

Drücken Sie Caps Shift, sollten Sie den Wert 30 für Adresse 65278 erhalten (Adresse Kanal C).

Betrachten Sie das hier gezeigte einfache Grafik-Programm. Dieses Programm liest die Cursor-Tasten, um horizontale, vertikale und diagonale Linien zu zeichnen. Drückt man gleichzeitig die Shift-Taste, wird der „Grafik-Cursor" bewegt, ohne eine Linie zu zeichnen. Diagonale Linien entstehen, wenn man zwei Cursor-Tasten gleichzeitig drückt.

In Programmzeile 30 wird die Funktion FN(a) definiert, um die oberen drei Bits abzutrennen. Die horizontale Position des Grafik-Cursors wird durch x und die vertikale Koordinate durch y repräsentiert. In Zeile 50 wird die Startposition des Grafik-Cursors in der Mitte des Bildschirms festgelegt.

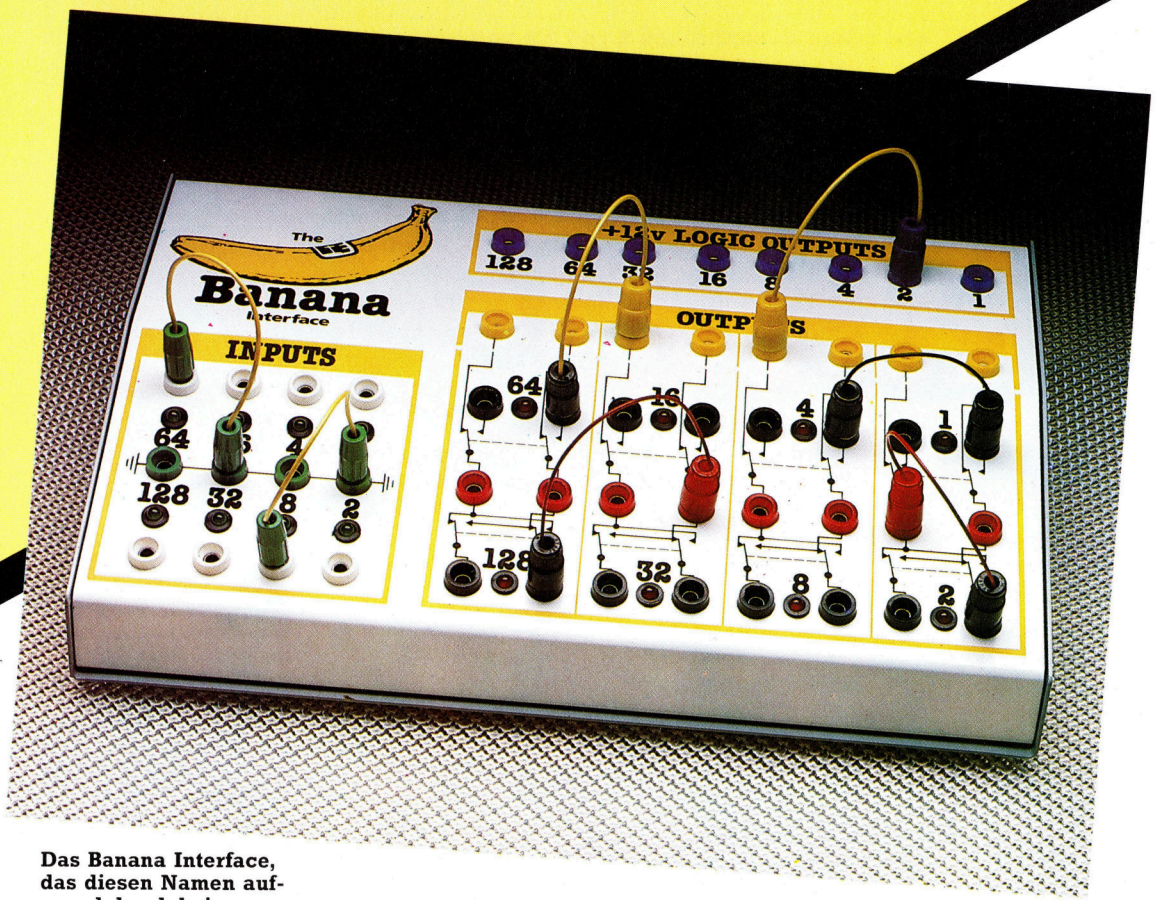
In den Zeilen 70 und 80 werden die Adressen der Cursor-Tasten eingelesen. Die Halbreihe von 6 bis 0 enthält drei der Cursor-Tasten und wird auf Kanal A (Adresse 61438) dargestellt. Die Halbreihe 5 bis 1 beinhaltet die Cursor-Taste mit dem Pfeil nach rechts und wird auf Kanal B (Adresse 63486) dargestellt. In Zeile 90 wird die Caps-Shift-Taste abgefragt – also die Halbreihe von V bis Caps Shift, die auf Kanal C (Adresse 65278) liegt.

In den Zeilen 110 bis 180 werden die acht legalen Kombinationen der Cursor-Tasten überprüft, und die x,y-Position des Grafik-Cursors wird entsprechend korrigiert:
Zeile 110 überprüft ↑
Zeile 120 überprüft ↑ und →
Zeile 130 überprüft →
Zeile 140 überprüft ↓ und →
Zeile 150 überprüft ↓
Zeile 160 überprüft ↓ und ←
Zeile 170 überprüft ←
Zeile 180 überprüft ↑ und ←
Mit den Zeilen 200 bis 240 wird sichergestellt, daß die Linien bei Erreichen des Bildschirmrandes unterbrochen werden. Wurde die Caps-Shift-Taste nicht gedrückt, dann erzeugt Zeile 250 an der neuen Position des „Stiftes" einen Punkt.

Das hier gezeigte Programm wendet die eben besprochene Technik zum Erkennen mehrerer gedrückter Tasten an. Werden die Cursor-Tasten ohne die Shift-Taste gedrückt, so können Sie horizontale, vertikale und diagonale Linien zeichnen (diagonal = horizontal + vertikal). Drückt man dieselben Tasten in Verbindung mit der Shift-Taste, so wird der Zeichenvorgang abgeschaltet.

Tastaturabfrage

```
 20 REM Set Up Function to Mask Top Three Bits
 30 DEF FN a(p)=p-INT (p/32)*32
 40 REM Initialise Pen Position
 50 LET x=127: LET y=35
 60 REM Read Ports and Mask Off Top Three Bits
 70 LET portA=FN a(IN 61438)
 80 LET portB=FN a(IN 63436)
 90 LET portC=FN a(IN 65278)
100 REM Alter Pen Position Depending on Which Key
       Has Been Pressed
110 IF portA=23 AND portB=31 THEN LET y=y+1
120 IF portA=19 AND portB=31 THEN LET x=x+1:
    LET y=y+1
130 IF portA=27 AND portB=31 THEN LET x=x+1
140 IF portA=11 AND portB=31 THEN LET x=x+1:
    LET y=y-1
150 IF portA=15 AND portB=31 THEN LET y=y-1
160 IF portA=15 AND portB=15 THEN LET x=x-1:
    LET y=y-1
170 IF portA=31 AND portB=15 THEN LET x=x-1
180 IF portA=23 AND portB=15 THEN LET y=y+1:
    LET x=x-1
190 REM Stop Pen Going Off The Screen
200 IF x<0 THEN LET x=0
210 IF x>255 THEN LET x=255
220 IF y<0 THEN LET y=0
230 IF y>175 THEN LET y=175
240 REM Plot Point if CAPS SHIFT Not Pressed
250 IF portC=31 THEN PLOT x,y
260 GO TO 70
```



Das Banana Interface findet für eine Reihe von Kontrollfunktionen im Labor wie zu Hause Anwendung. Die in diesem Raum dargestellten Einrichtungen lassen sich über das mit einem Computer verbundene Banana Interface regeln bzw. überwachen. Dazu sind zahlreiche Sensoren – u. a. Thermostate und Lichtsensoren – installiert worden. Die von diesen Sensoren empfangenen Informationen werden vom Computer analysiert und mit vorprogrammierten Optimalwerten verglichen. Regulierungen im Hinblick auf den Raumzustand (wie das Erhöhen der Temperatur oder Senken der Jalousien) erfolgen über die Output-Kanäle des Banana Interface.

CS Kontakt-Sensor
HC Temperatur-Sensor
DS Dimmer-Schalter
EM Elektromotor
SP Alarmsirene
SD Rauchmelder
TH Thermostat
PC Fotozelle

Das Banana Interface, das diesen Namen aufgrund der dabei verwendeten Bananenstekker erhielt, erlaubt die Verbindung analoger elektrischer Geräte mit einem Computer (der digitale Signale verarbeitet). So lassen sich durch Rückkopplung computergesteuerte Systeme konstruieren.

Das Banana Interface

Das von Castle Associates hergestellte und vornehmlich für den Schulbereich entwickelte Banana Interface ist eine solide gebaute Ergänzung für den Bereich computergesteuerter Geräte.

In unserem Bastelkurs haben wir bereits eine Reihe von Geräten zur Steuerung von externen Maschinen über Computer vorgestellt und ebenso gezeigt, wie viele Verbraucher angeschlossen werden können. Neu auf dem Markt ist das Banana Interface für den Acorn B und den C 64, mit dem sich viele Experimente durchführen lassen.

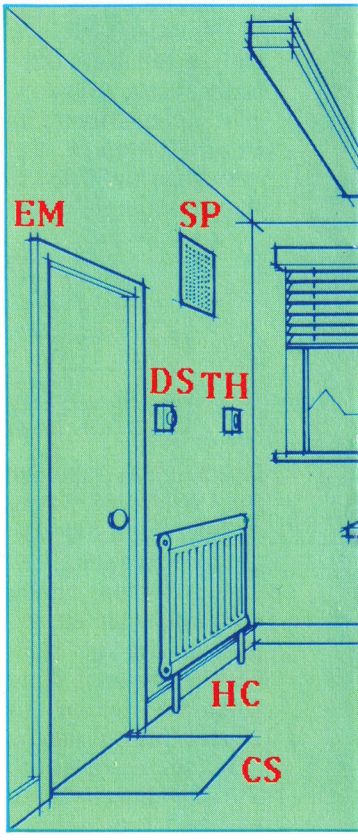
Der Aufbau des Banana Interface gleicht in etwa dem unserer selbstgebauten Buffer-Box. Allerdings ist das Banana erheblich besser. Durch Veränderung der in den Daten und Datenregistern enthaltenen binären Bitmuster können 5-V-Signale an externe Geräte übertragen werden. Da User- wie Drucker-Ports bidirektional arbeiten, kann der Computer periphere Signale empfangen, wodurch er den Status des Gerätes „kennt" – etwa seine Position – oder „weiß", ob ein Schalter aktiviert wurde. Durch diese Kombination von Input/Output-Signalen kann der Computer einen Roboter genau steuern und auf die sich verändernde Umgebung des Roboters reagieren.

Das Interface ist über ein Flachkabel, das in den User- oder Drucker-Port gesteckt wird, mit dem Computer verbunden. Das andere Ende paßt auf den 40poligen Stecker des Interface.

Die Stromversorgung

Für den Betrieb des Banana Interface sind zwei Gleichstromquellen erforderlich, von denen die eine mit einer Spannung von 5 V über das Flachkabel aus dem Computer kommt. Dieser Strom wird zum Betrieb der Interface-Logik benötigt.

Die zweite Stromquelle wird extern durch zwei Ministecker an der Rückseite des Interface direkt neben dem Flachkabelanschluß



angeschlossen. Leider hat sich der Hersteller Castle Associates nicht für eine 12-V-Spannung zur Versorgung entschieden. Grund dafür ist, daß das Banana für den Gebrauch in Schulen entwickelt wurde, wo man in den Labors über entsprechende Spannungen verfügt. Durch Weglassen des entsprechenden Trafos wurden die Kosten für das Gerät gesenkt.

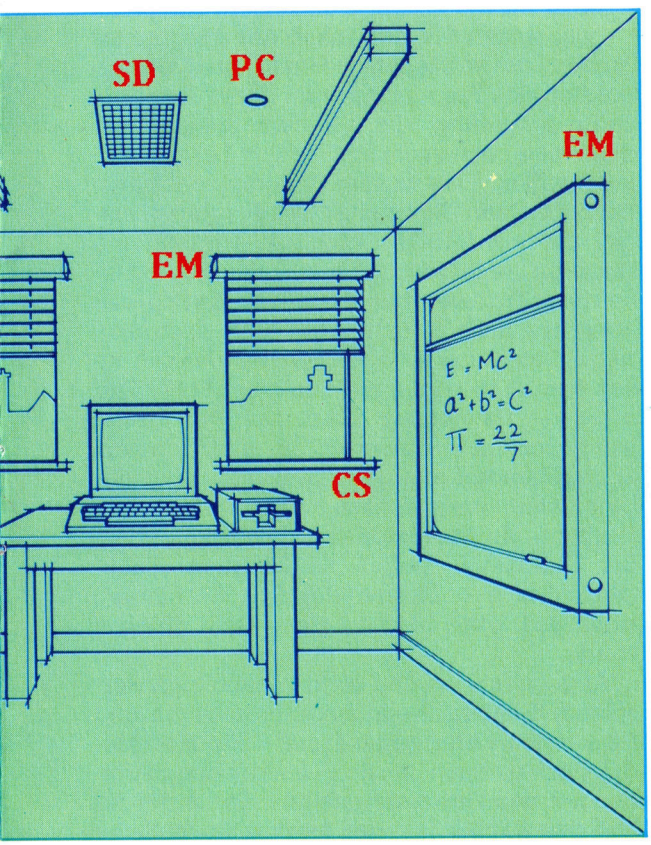
Wer sich also mit dem Gedanken trägt, ein Banana zu erwerben, sieht sich vor das Problem gestellt, eine geeignete Stromversorgung zu beschaffen. Da 12-V-Trafos mit kleinen Amperewerten nicht fertig erhältlich sind, läßt sich das Interface am ehesten über in Reihe geschaltete Batterien betreiben.

Stabiles Gehäuse

Im Planungsstadium wurde Castle Associates von Werk-, Kunst- und Physiklehrern beraten. Das wirkte sich vorteilhaft auf die Bauweise des Gerätes aus. Da das Gehäuse ganz aus Stahl gebaut ist, handelt es sich bei diesem Interface um eine der stabilsten Peripherien für Heimcomputer.

Die solide Bauweise gilt auch für das Innere des Interface. Die Platine ist fest mit den Stekkern verlötet. So hält das Gerät viel aus, und es gehört schon einige Kraftanstrengung dazu, es zu beschädigen.

Oben auf dem Gehäuse finden sich die Minibuchsen, die in drei Gruppen angeordnet sind. An der linken Seite liegen acht weiße Input-Eingänge, dazwischen jeweils — in Grün — die Erdung. Jede Buchse ist mit 1, 2, 4, 8 usw. entsprechend der Bit-Positionsnummer im Register beziffert. Die den Bit-Positionen zugeordneten LEDs zeigen an, welches Bit „high" ist (das bedeutet, daß ein 5-Volt-Strom über das entsprechende Datenkabel läuft).



Veränderungen an den Eingabeleitungen werden vom Computer registriert. Der einfachste Weg, dies zu demonstrieren, ist, alle Datenkanäle auf „high" zu setzen. Durch PEEKen des Registers wird ein Wert von 255 erzeugt. Verbinden wir aber einen Erdungsanschluß über Draht mit einer weißen Input-Buchse, fällt die Spannung auf Null. Der Registerwert sinkt entsprechend, abhängig von der verwendeten weißen Input-Buchse. Mit dieser einfachen Schaltung läßt sich beispielsweise eine Alarmanlage herstellen.

Rechts neben den Input-Buchsen befinden sich zahlreiche Output-Buchsen. Wie bei den Input-Anschlüssen gibt es acht Grundpositionen (von 1, 2, 4 usw. bis 128 beziffert) mit zugeordneten LEDs für jede Bitposition sowie vier Minibuchsen. Die Minibuchsen werden durch Relais paarig verbunden. Durch POKEn einer Zahl an die Registeradresse &FE61 werden die Relais geschaltet.

Verbindet man Elektromotoren und entsprechende Stromquellen mit dem Interface, lassen sich die Relais zur Steuerung und Positionierung der Motoren benutzen. Dabei können vier Elektromotoren gleichzeitig über das Banana Interface gesteuert werden. Der effektivere Weg zur Steuerung von Elektromotoren (die zum Beispiel mit einer Bodenschildkröte oder einem Roboterarm verbunden sind) ist der Betrieb über die Output-Leitungen und die Bewegungsüberwachung über die Leitungen des Input-Ports.

Anwendungsbereiche

Am oberen Rand des Interface ist eine weitere Reihe von acht Hochgeschwindigkeits-Logik-Ports angebracht, die jeweils Spannungen von 12 Volt erzeugen. Mit diesen Output-Kanälen lassen sich Schrittmotoren (bis zu sieben gleichzeitig) über das Interface betreiben.

Der Markt für das Banana Interface liegt hauptsächlich im Schul- und Forschungsbereich, wo es als Hilfsmittel für den Unterricht über computergesteuerte Geräte Grundlagen vermittelt. Es gibt eine Fülle von Anwendungsbeispielen. Für den privaten Benutzer scheint der Verwendungszweck des Gerätes weniger klar. Es kann sicher daheim für zahlreiche unterhaltsame Experimente verwendet werden und vermittelt Kenntnisse über Computersteuerungstechniken sowie über die Robotik. Benutzer eines Acorn B oder C 64 hingegen, die nur vorübergehend Interesse an diesem Bereich haben, werden wahrscheinlich vom Kauf eines Banana Interface wegen des hohen Preises (circa 800 Mark) Abstand nehmen.

Banana Interface

ABMESSUNGEN

300 × 200 × 63 mm

SCHNITTSTELLEN

40adrige Parallelverbindung von Drucker- und User-Ports des Acorn B und des Commodore 64, Output-Ports, acht 12-V-Ausgangsbuchsen.

STÄRKEN

Das Banana Interface ist solide gebaut und bietet zahlreiche Möglichkeiten für die Computersteuerung von Elektrogeräten.

SCHWÄCHEN

Das Fehlen einer geeigneten 12V-Stromversorgung limitiert seine Anwendungsmöglichkeiten, so daß hauptsächlich Hobby-Elektroniker sowie Ausbildungs- und Forschungsstätten als Interessenten in Frage kommen.

Eigene Abläufe

In PASCAL können Sie von der Sprache vorgegebene Prozeduren und Funktionen durch eigene Abläufe ergänzen. In diesem Artikel untersuchen wir, welche Unterschiede zwischen Funktionen und Prozeduren bestehen, und gehen auf ihre Gültigkeitsbereiche ein.

In PASCAL können Sie eigene Funktionen und Prozeduren definieren, wenn Ihnen die von der Sprache vorgegebenen Möglichkeiten wie „write" oder „sqrt" nicht genügen. Bevor wir uns jedoch diesen Mechanismen zuwenden, wollen wir zunächst untersuchen, warum beispielsweise „write" eine Prozedur ist, aber keine Funktion, und „sqrt" eine Funktion, aber keine Prozedur. Bei dem Befehl

```
write ('Hallo!')
```

erwarten wir, daß die als Parameter angegebene Zeichenfolge in dieser Reihenfolge ausgegeben wird. Mit anderen Worten: „write" ruft ein Unterprogramm auf, das die Daten als Zeichenfolge auf den Bildschirm bringt. Was aber passiert bei folgender Anweisung:

```
sqrt (256)
```

Ein Compilerfehler wird angezeigt – Anweisungen dieser Art sind nicht legal. Der Befehl

```
write(sqrt (256))
```

wäre jedoch völlig in Ordnung: Auf dem Bildschirm erscheint 1.60000E+01.

Da die write-Anweisung einen Ablauf auslöst (hier: Daten an ein Ausgabegerät sendet) wird sie als Prozedur per Namen aufgerufen. Eine weitere von PASCAL vorgegebene Prozedur ist beispielsweise „page(F)", (in der Textdatei F wird eine neue Seite angesprochen). Der Ausdruck „sqrt" dagegen liefert als Ergebnis nur eine reale Zahl – die Quadratwurzel des Argumentes. Er muß immer mit einem einfachen numerischen „Argument" versehen sein. Im Gegensatz zu Prozeduren sind Funktionen daher keine Anweisungen: Ebenso wie die Zahl 16 ohne weitere Information nur diesen Wert darstellt, haben Ausdrücke wie „sqrt(X/3)" oder „odd(N)" ebenfalls keine eigenständige Bedeutung. Die Namen der Funktionen können daher wie Variablen angesehen werden, die zwar nie initialisiert wurden, aber automatisch berechnet werden, wenn ihr Name im Programm auftaucht. Der Ergebniswert wird dabei immer vom aktuellen Wert des Argumentes abgeleitet, das heißt, er ist Funktion des Argumentes.

Prozeduren liefern jedoch keine Ergebniswerte und können daher auch nicht in Ausdrücke eingesetzt werden. Ein Befehl wie:

```
N:=WriteLn
```

ist ebensowenig möglich wie LET N = PRINT in BASIC.

PASCAL erleichtert das Schreiben umfangreicher Programme, indem es die Möglichkeit bietet, eigene Funktionen und Prozeduren anzulegen, deren Zugriff und Verbindung untereinander exakt gesteuert ist. Da die Namen hier wie bei den Variablenbezeichnungen nur wenigen Einschränkungen unterworfen sind, können Sie für Ihre eigenen Unterprogramme eindeutige und ausführliche Bezeichnungen angeben. Sehen Sie sich folgende Programmstruktur an:

```
PROGRAM Beispiel (input, output, Datei);
                        (* Deklarationen . . *)
BEGIN
  Oeffne (Datei);
  WHILE NOT EoF (Datei) DO
    BEGIN
      read (Datei, Inhalt);
      Bearbeite (Inhalt)
    END
END.
```

EoF und EoLn

Obwohl wir Dateizugriffe noch nicht behandelt haben, sollten Sie den Inhalt des Programms problemlos verstehen können. Im Deklarationsteil des Programms wäre dabei eine Prozedur aufgeführt, die auf dem Speichermedium eine Datei sucht und sie zum Lesen eröffnet (Oeffne). Eine weitere Prozedur verarbeitet dann die Datei (Bearbeite). Das Programm enthält die von PASCAL ebenfalls vordefinierte Prozedur „read", die wir bisher nur zum Einlesen von Texteingaben eingesetzt haben, die aber ebenfalls strukturierte oder unstrukturierte Daten aus entsprechenden Dateien lesen kann. Die boolesche Funktion „EoF" (End of File – Dateiende) liefert den Wert „true", wenn der letzte Datensatz der als Argument angegebenen Datei gelesen wurde. Wird der Dateiname weggelassen, dann nimmt die Funktion die Standarddatei als Argument an. Bei Textdateien liefert die Funktion „EoLn" (End of Line – Zeilenende) den Wert „true", wenn das letzte Zeichen einer Zeile gelesen wurde.

Abgesehen von den oben genannten Unterschieden haben Programme, Prozeduren und Funktionen vieles gemeinsam – alle sind Module mit eigenen lokalen Variablenbeschreibungen und Anweisungsfolgen. So ist der Inhalt des Kopfes fast der einzige wesentliche

Syntax-Unterschied. Ein Programmkopf besteht aus dem reservierten Wort „PROGRAM", einem vom Anwender vorgegebenen Namen und einer Parameter-Liste, die die Dateien angibt, mit denen das Programm Daten austauscht. Ein Prozedurkopf dagegen besteht aus dem reservierten Wort „PROCEDURE" und einer Parameter-Liste mit den Typennamen der Datenvariablen. Hier ein Beispiel:

```
PROCEDURE Zeilen (Anzahl Zeilen : integer);
  VAR
    N : integer;
  FOR N := 1 TO AnzahlZeilen DO
      WriteLn
END;
```

Der END-Anweisung einer Prozedur folgt immer ein Semikolon und kein Punkt wie am Ende eines Programms. Dem formalen Parameter-Namen „AnzahlZeilen" wird beim Aufruf der Prozedur der jeweils aktuelle Wert übergeben:

```
Zeilen (5)
```

Zugegeben, der Vorgang ist recht simpel, wenn Sie aber oft mehrere Leerzeilen nacheinander ausgeben müssen – in diesem Falle fünf – dann erspart Ihnen die Prozedur die lästigen Folgen von WriteLn-Anweisungen. Mit dieser Allzweckroutine können Sie soviel Leerzeilen ausgeben, wie Sie jeweils brauchen. „Zeilen (10)" erzeugt 10 Leerzeilen, „Zeilen (20)" 20 etc.

Unsere einfache Prozedur ist ebenfalls ein gutes Beispiel für die Datensicherheit, die PASCAL bietet. So muß die Steuervariable für die FOR-Schleife als echte lokale Variable deklariert sein. Es ist nicht möglich

```
FOR AnzahlZeilen := 1 To AnzahlZeilen DO . .
```

zu schreiben, da die Sicherheit einer Prozedur nicht garantiert werden kann, wenn die Steuervariable nicht lokal oder auch nur „relativ global" ist. Haben Sie schon einmal stundenlang nach einem Fehler im BASIC-Programm gesucht, das Programmzeilen dieser Art enthielt

```
300 FOR N = 1 TO T
400 GOSUB 2000
500 NEXT N
```

um schließlich herauszufinden, daß N von der Unterroutine mit einem anderen Wert belegt wurde? PASCAL fängt Fehler dieser Art von vornherein ab und spart Ihnen so viel Zeit bei der Programmentwicklung. Jeder Name, der im Inneren eines Blocks deklariert wird, kann sich nur auf diesen Block beziehen. Sein Gültigkeitsbereich (das heißt der Teil des Blocks, von dem aus auf diesen Namen zugegriffen werden kann) erstreckt sich zwar vom Punkt seiner Deklaration bis zum Blockende, kann aber durch Neudefinition weiter eingeschränkt werden. Die Variable N des vorigen Beispiels bezog sich auf das Innere der Prozedur „Zeilen" und ist damit eine lokal deklarierte Ganzzahl. Diese Deklaration hat bei jedem Prozeduraufruf automatisch Vorrang vor globalen Variablen gleichen Namens.

Bereichsweise

```
PROGRAM Bereich;

VAR
    N : integer;
    X : real;
PROCEDURE A ( Y : real );

  TYPE
    X = SET OF char;
  (* etc. *)

PROCEDURE B ( N : integer );
  (* etc. *)
BEGIN (* Hauptprogramm —
  Bereich *)
  . . .
  A ( succ (N) / 3 );
  B ( N );
  A ( X );

  (* etc. *)
END.
```

Die untenstehende Tabelle zeigt den Bezug und die Gültigkeitsbereiche der Variablen des nebenstehenden Kurzprogramms.

Prozedur/Var	Bezug	Gultigkeit
A	Hauptprog.	A,B,Hauptprog
B	Hauptprog.	B,Hauptprog.
N(global)	Hauptprog.	A,Hauptprog.
X(real)	Hauptprog.	B,Hauptprog.
X(in Proz. A)	A	A
N(in Proz. B)	B	B

Die globalen Variablen N und X im Hauptprogrammteil von „PROGRAM(Bereich)" beziehen sich auf das gesamte Programm. Der Gültigkeitsbereich von X reicht daher mit Ausnahme der Prozedur A von ihrer Deklaration bis an das Programmende. In A wird X als Typenname für „X(SET OF char)" verwendet, der aber nur für Prozedur A Gültigkeit hat. In ähnlicher Weise ist N der formale Parameter der Prozedur B und keine globale Variable.

A und B beziehen sich weiterhin zwar auf das gesamte Programm, doch fängt der Gültigkeitsbereich von B erst bei seiner Definition an. Während Sie also A durchaus in die Prozedur B einsetzen können, ist B für A unerreichbar. Hier zeigt sich die Logik von PASCAL: Kein Programm läßt sich aufrufen, bevor es geschrieben wurde. Es wird auch klar, warum alle Definitionen und Deklarationen immer in der festgelegten Reihenfolge erfolgen müssen: CONST, TYPE, VAR und dann die Definition der Prozeduren und Funktionen, geordnet nach den strukturellen Erfordernissen des Programms. Da viele PASCAL-Compiler den Quellentext nur einmal durchgehen, ist diese logische Anordnung unbedingt notwendig.

Modularer Programmaufbau

Der zusätzliche Aufwand mit lokal deklarierten Variablen zahlt sich aus. Wenn Programme alle ihre Daten als Parameter an Prozeduren übergeben, wird ein hoher Grad an modularem Aufbau erreicht. Zwar gibt es PASCAL-Programme, die Prozeduren ohne Parameter-Listen verwenden und nur globale Daten einsetzen, doch sollte diese Art der Programmierung möglichst vermieden werden. Modularer Programmaufbau hat außer erhöhter Datensicher-

heit noch den weiteren wesentlichen Vorteil, daß ein Programmiererteam an einem großen Programm arbeiten kann, ohne sich um doppelt vergebene Variablen-Namen kümmern zu müssen. Beim Aufruf von B wird der Wert der globalen Variablen N in dieser Prozedur zwar als aktueller Parameter an eine Variable gleichen Namens übergeben, doch ist N dort eine völlig andere Variable. Das zeigt folgende wichtige Punkte auf:

- Beim „Blockeintritt" wird der übergebene Wert in eine lokale Variable kopiert.
- Eine Veränderung dieses Wertes innerhalb des Blocks hat keine Auswirkungen auf die ursprüngliche globale Variable.
- Der übergebene Wert kann ein Ausdruck jedes (korrekten) Typs sein.

Natürlich muß jeder Prozeduraufruf die aktuellen Parameter enthalten, die mit der formalen Liste am Prozeduranfang übereinstimmen. Der Übergangsmechanismus kann daher als eine Art Initialisierung der formalen Parameter-Namen mit aktuellen Werten angesehen werden.

Zeilen (succ (N+gap)DIV 2)

führt innerhalb der Prozedur folgende Zuweisung aus:

AnzahlZeilen := succ(N+gap)DIV 2

Versuchen Sie einmal herauszufinden, was passiert, wenn die übergebenen Werte innerhalb der Prozedur verändert werden.

Grundwerte

Das Programm „Grundwert" zeigt, wie sich Wertparameter von Prozeduren aus einsetzen lassen. Bei Angabe einer Basis von 2 (binär) bis 16 (hexadezimal) werden per Tastatur eingebene Dezimalzahlen in der entsprechenden Schreibweise dargestellt. So würde 32767 in Hexadezimal 7FFF werden, in Binär 111111111111111 oder 77777 in Oktal.

Versuchen Sie, folgende Änderungen in das Programm einzufügen:

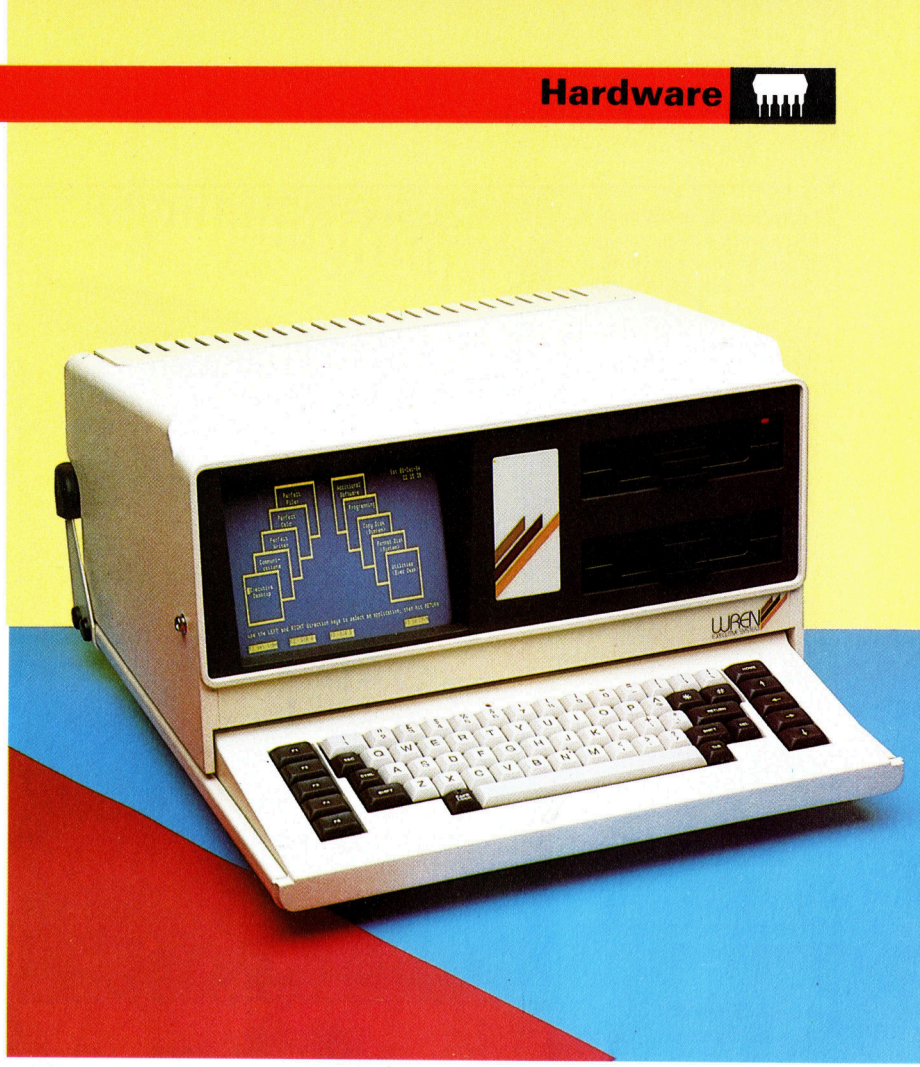
○ Um Negativdarstellungen zu ermöglichen, muß die Zahl in ihr Zweierkomplement umgewandelt werden. Auf Maschinenebene wird dafür die Zahl umgekehrt und Eins addiert. Finden Sie eine Methode, die diese Aufgabe in PASCAL erledigt?

○ Möchten Sie die Umwandlung andersherum ausführen? Nehmen Sie dazu das Programm als Modell für die Eingabe einer Zahl mit Basis 2 bis 16 und für die Ausgabe im Dezimalformat. Die beiden vorgeschlagenen Programmteile lassen sich auch zu einem Programm zusammenfassen.

```
PROGRAM          GrundWert          ( Input, Output );
        (* wandelt Dezimalzahlen in Zahlenformate jeder
        Basis im Bereich binär bis hexadezimal um *)

CONST
        Spalte = 79; (* Bildschirm / Drucker *)

TYPE
        Byte     = 0 . . 255;
        Cardinal = 0 . . MaxInt;

VAR
        Zahl,
        Basis : integer;
        Ende,
        legal : boolean;

(* ------------------------------------------------------ *)

PROCEDURE          SchreibeZiffer          ( Ziffer : Byte );
    (* Wert von Liste [ Zaehler ] an Ziffer übergeben *)

  BEGIN
    IF Ziffer IN [ 0 . . 9]
      THEN
        write ( Ziffer : 1 )
      ELSE (* Darstellung als A . . F *)
        CASE Ziffer OF
          10      :  write    ( 'A' );
          11      :  write    ( 'B' );
          12      :  write    ( 'C' );
          13      :  write    ( 'D' );
          14      :  write    ( 'E' );
          15      :  write    ( 'F' )
        END; (* CASE *)
  END; (* SchreibeZiffer *)
(* ------------------------------------------------------ *)
PROCEDURE          Anzeigen          ( N          : Cardinal;
                                       Basis      : Byte );
  CONST  (* Alle diese Daten sind für die
            Prozedur lokale Daten *)
        MaxZiffer = 32;
  TYPE
        Grenze = 1 . . MaxZiffer;
  VAR
        Liste    : ARRAY [ Grenze ] OF Byte;
        Index    : Grenze;
        Zaehler : Byte

  BEGIN
    Zaehler := 0;

    REPEAT
        Zaehler := succ ( Zaehler );
        Liste [ Zaehler ] := N MOD Basis;
        N := N DIV Basis

    UNTIL N = 0;
              (* Anzeige beginnt bei MSB:*)
    FOR Zaehler := Zaehler DOWNTO 1 DO
        SchreibeZiffer ( Liste [ Zaehler ] )

  END; (* Anzeigen *)
(* ------------------------------------------------------ *)
BEGIN          (* Grundwert — Hauptprogramm *)

  REPEAT
    WriteLn ( 'Geben Sie eine Zahlenbasis:' );
    WriteLn ( ' von 2 . . 16 an' : Spalte );
    WriteLn ( '(Zahlen ausserhalb = Ende)' : Spalte );
    Write ( 'Basis ? ' );
    read (Basis );
    legal := Basis IN [2 . . 16 ];

    IF legal THEN
      BEGIN
        write ( 'Zahl zum Wandeln (0 wechselt Basis) ? ' );
        read ( Zahl );
        Ende := Zahl <= 0;

        WHILE NOT Ende DO
          BEGIN
            write ( Zahl : Spalte DIV 2,
                    ' zu Basis ', Basis : 1, ' ist ' );
            Anzeigen ( Zahl, Basis );
            WriteLn;

            write ( ' Neue Zahl ? ' );
            read ( Zahl );
            Ende := Zahl <= 0;
          END
      END

  UNTIL NOT legal

END.
```

Gut gerüstet

Der Wren Executive, ein tragbarer Computer mit zwei Diskettenstationen, eingebautem Selbstwähl-Modem und integriertem BASIC ausgestattet, bietet viele Anwendungsmöglichkeiten. Welche Chancen er hat, zeigen wir nachstehend auf.



Der Wren Executive Computer (im Vertrieb von Opus Supplies) ist ein preisgünstiger Portable, der mit dem CP/M-Betriebssystem ausgestattet ist. Er verfügt ferner über zwei Diskettenstationen, einen monochromen Monitor und ein eingebautes Modem.

Der Wren Executive Computer hat eine wechselvolle Geschichte. Die von Thorn EMI entwickelte Maschine wurde ursprünglich von Prism vertrieben. Als das Unternehmen zum Jahresanfang 1985 Konkurs anmeldete, schien es, daß der Wren ein weiteres Opfer der turbulenten Computerindustrie sein würde. Unlängst aber übernahm Opus Supplies den Vertrieb für dieses Gerät. Das Unternehmen will den Rechner, positioniert im Geschäftsbereich, wieder auf den Markt bringen.

Der Wren Executive wurde als „Portable" entwickelt. Das heißt, es handelt sich um einen Computer, der mit eigenem Monitor und Diskettenstationen ausgerüstet ist. Das Gehäuse des Rechners besteht aus strukturiertem Kunststoffmaterial. Nur der Rahmen und die Tastaturhalterung sind aus Metall gefertigt. An der Rückseite des Rechners befindet sich der Tragegriff. Wie viele andere tragbare Maschinen ist der Computer für einen längeren Transport etwas zu schwer und sicherlich besser als Desk Top geeignet.

Die Tastatur entspricht dem QWERTY-Standard mit der Ausnahme, daß das *- und das #-Symbol auf separaten Tasten liegen. Links neben der Schreibmaschinentastatur befinden sich fünf programmierbare Funktionstasten, die, in Verbindung mit der Control- oder Shift-Taste, bis zu 15 verschiedene Funktionen aufrufen können. Diese Tasten haben – abhängig von der verwendeten Applikation – unterschiedliche Bedeutungen. Sie werden vornehmlich aber für die Befehlseingabe beim CP/M verwendet, beispielsweise PIP oder RENAME. Auf der rechten Seite der Tastatur sind vier Cursor-Steuerungstasten und eine Home-Taste angeordnet.

Der Bildschirm mißt 150 × 105 mm und ist einer der üblichen monochromen Monitore, mit denen PCs ausgestattet sind. Seine Textauflösung beträgt 80 mal 25 Zeichen. Der einzige bemerkenswerte Unterschied ist, daß als Vordergrundfarbe Orange statt dem sonst üblichen Grün verwendet wurde. Die Bildschirmdarstellung ist dadurch sehr hell und wirkt auch bei längerem Arbeiten weniger ermüdend. Rechts vom Monitor sind zwei 5 1/4-Zoll-Diskettenstationen untergebracht.

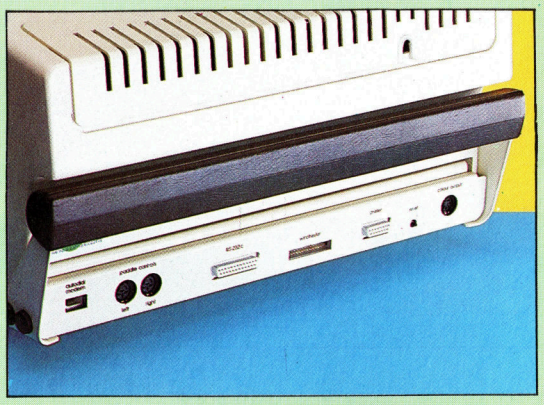
Schräger Blick

Ein Problem resultiert aus dem Design des Gehäuses. Da der Wren nicht mit Füßen ausgestattet ist, steht er vergleichsweise flach auf dem Tisch. Eine individuelle Schrägstellung des Bildschirms ist nicht möglich. Der Schirm befindet sich überdies tief im Gehäuse. Er ist zwar schräg integriert, jedoch für besonders große Leute oder Anwender, die gewöhnt sind, ganz dicht an der Tastatur zu sitzen, relativ schwer lesbar.

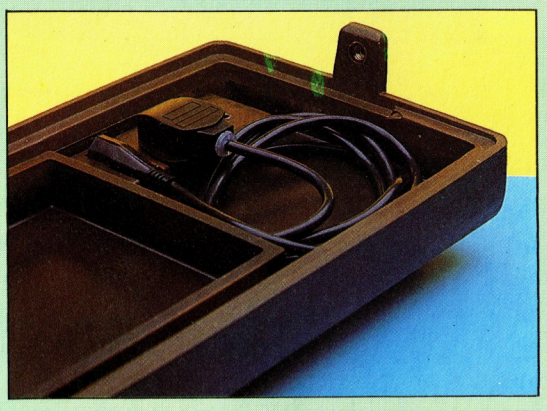
Der Wren ist mit zahlreichen Schnittstellen ausgestattet. Eine Reihe von Ports auf der Rückseite des Computers erlauben den Anschluß mehrerer Peripheriegeräte. Und anders als Business-Rechner derselben Preisklasse ist der Wren mit einem Selbstwähl-Modem ausgerüstet.

Sogar ein Kabel, das die Verbindung mit dem Telefonnetz erlaubt, ist im Lieferumfang des Wren enthalten. Es paßt in die Standardbuchse auf der Rückseite des Rechners. Die mitgelieferte Kommunikationssoftware bietet die sinnvolle Funktion, viele Telefonnummern zu speichern und automatisch anwählen zu lassen, bis sich jemand meldet.

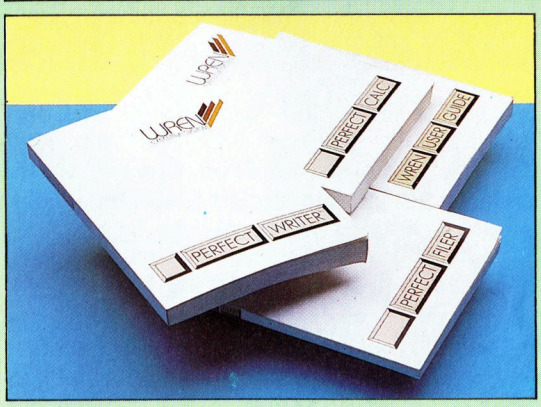
Schnittstellen reichlich
Eine Besonderheit des Wren sind die zahlreichen Schnittstellen, mit denen das System serienmäßig ausgestattet ist. Neben den üblichen Centronics- und RS232C-Schnittstellen gibt es auch einen Modem-Anschluß und sogar ein paralleles Interface für eine Winchester-Festplatte.



Einstecken und los
Monitor und Diskettenstationen sind beim Transport durch eine schwere Kunststoffhaube geschützt, die über die Gehäusevorderseite gestülpt wird. Hand in Hand gehend mit der Philosophie des „Einstecken und los!" haben die Hersteller das Stromkabel fest integriert. Es befindet sich während des Transports ebenfalls im Gehäuse.



Alles „Perfect"
Da der Wren von Thorn EMI gebaut wurde, ist naheliegend, daß das Unternehmen seine eigenen Programme mit dem Rechner ausliefert – Perfect Writer, Perfect Filer und Perfect Calc. Ferner gehören ein Service-Programm namens Executive Desktop dazu sowie Kommunikationssoftware.



Direkt neben dem Kommunikationsanschluß liegen zwei Paddle-Ports, an denen externe Steuereinheiten wie Joystick und Maus angeschlossen werden können. Dahinter befindet sich ein 25poliger D-Anschluß, der als Verbindung für das RS232C-Interface dient. Gewöhnlich werden RS232-Stecker bei Personal Computern für den Anschluß externer Modems verwendet. Da dies beim Wren nicht erforderlich ist, kann hier ein serieller Drucker angeschlossen werden.

Links neben dem RS232-Port befindet sich ein Parallel-Interface, über das man eine Festplatte ansprechen kann. Mit der Winchester ist eine Kapazitätserweiterung von 500 KByte auf 50 Mega-Byte möglich. Zudem steht eine Centronics-kompatible Parallel-Schnittstelle zur Verfügung, und ganz rechts außen liegt der Anschluß für einen RGB-Monitor. Zwischen Drucker- und Monitor-Interface ist ein winziger Reset-Knopf angebracht.

Monitor
Der Wren ist mit einem eingebauten monochromen Monitor ausgestattet, der gut lesbar ist.

RS232C-Port
Hierüber ist die direkte Kommunikation mit anderen Computern möglich.

Farb-Monitor-Anschluß

Lautsprecher
Die Klangwiedergabe erfolgt über den eingebauten Lautsprecher.

Winchester-Schnittstelle
Über diese Parallel-Schnittstelle kann eine Winchester-Festplatte, die die Speicherkapazität enorm erweitert, angeschlossen werden.

RAM-Chips
Wenngleich der Wren nur über einen Acht-Bit-Prozessor verfügt, lassen sich durch „Bank Switching" bis zu 400 K RAM adressieren.

Als Zentraleinheit des Wren dient der bekannte Z80B Acht-Bit-Prozessor. Dies ist vielleicht die größte Schwäche des Rechners. In einer Zeit, da selbst Heimcomputer-Hersteller 16-Bit-Rechner in Entwicklung haben, wirkt der so ausgestattete Wren antiquiert. Gemessen am Acht-Bit-Standard ist er zwar vergleichsweise schnell, kann aber in der Verarbeitungsgeschwindigkeit nicht mit Rechnern wie dem IBM PC oder dem Apple Macintosh verglichen werden.

Aufgrund des verwendeten Prozessors war naheliegend, daß man sich für CP/M als Betriebssystem entschied. Problem hierbei ist, daß CP/M nach neueren Standards trotz seiner Leistungsfähigkeit doch ein sehr benutzerunfreundliches Betriebssystem ist. Die Entwickler haben versucht, dieses Problem durch Integration einer menügesteuerten Prozedur zu überbrücken, die durch Cursoransteuerung und Druck auf die RETURN-Taste aktiviert wird. Das System fordert nun zum Einschieben der entsprechenden Diskette auf und startet das Programm automatisch.

Software als Zubehör

Mit dem Computer werden folgende Programme geliefert: die CP/M-Systemdiskette, Perfect Writer (ein Textverarbeitungsprogramm), Executive Desktop (ein Paket, mit dem Termine, Daten und ein Tagebuch verwaltet werden), das Datenbankprogramm Perfect Filer, das Perfect Calc Spreadsheet und eine Diskette mit dem Terminal/Kommunikationsprogramm sowie BASIC.

Das beim Wren verwendete BASIC entpuppt sich überraschenderweise als das Acorn-BASIC. Wenngleich diese BASIC-Version fast vollständig implementiert wurde – der Wren ist ja auf dem Z80 statt auf einem 6502 aufgebaut –, gibt es doch einige Unterschiede, namentlich bei den Input/Output-Routinen. Die Implementierung der Sprache scheint zudem einige Fehler verursacht zu haben. So verschwindet zum Beispiel in den Modi 0 und 1 der Cursor vom Bildschirm. Die Delete-Taste funktioniert in diesen Modi nicht, und das Zurückführen des Cursors beim Versuch, die Zeichen zu überschreiben, führt lediglich dazu, daß das neue Zeichen auf dem alten steht. Das macht ernsthaftes Programmieren in diesen Modi unmöglich – die anderen Modi arbeiten jedoch durchaus perfekt.

Der künftige Erfolg des Wren Executive hängt natürlich davon ab, ob es eine Marktlücke für den Rechner gibt. Mit eingebautem Modem, CP/M-Betriebssystem und gutem BASIC zu einem Preis von unter 4000 Mark bot der Wren bei seiner Einführung Anfang 1984 viel Leistung fürs Geld. Seitdem aber ist das Interesse und damit das Kaufpotential an Acht-Bit-Personal-Computern geschwunden. Für den Heimmarkt ist der Rechner zu teuer.

Wren Executive

ABMESSUNGEN
440 × 410 × 250 mm

ZENTRALEINHEIT
Z80B, 6MHz

BILDSCHIRMDARSTELLUNG
80 × 25 Textdarstellung,
640 × 256 Grafikdarstellung.

SCHNITTSTELLEN
Monitoreingang, Centronics-Schnittstelle, RS232C-Schnittstelle, doppelter Paddle-Anschluß, Winchester-Festplatten-Interface.

PROGRAMMIERSPRACHEN
BASIC

TASTATUR
57 Schreibmaschinentasten, fünf programmierbare Funktionstasten, fünf Cursor-Steuerungstasten.

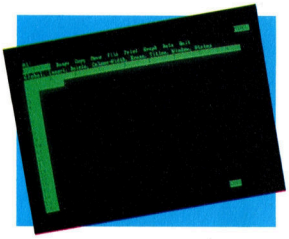
HANDBÜCHER
Neben dem allgemeinen Benutzerhandbuch verfügt jedes Programm über eine eigene Dokumentation. Diese sind sehr gut gestaltet und geben dem Anwender alle notwendigen Erläuterungen.

STÄRKEN
Der Computer wird mit vielen Schnittstellen, eingebautem Modem und einer Vielzahl von Programmen ausgeliefert.

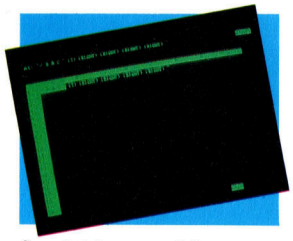
SCHWÄCHEN
Die im Wren verwendete 8-Bit-Technik ist deutlich überholt. Einige Applikationen enthalten Fehler.

Macros auf Micros

In unserer Serie über Kalkulationssysteme untersuchen wir den Einsatz von Tastatur-Macros anhand von Lotus 1–2–3, dem Tabellensystem mit integrierter Datenbank und Grafikfunktionen.



Das Hauptmenü von Lotus wird wie bei VisiCalc über die /-Taste aufgerufen.



So sieht unser Macro zur Kennzeichnung von Feldern in der Tabelle aus. Macro-Abläufe lassen sich auch in der Spalte nach unten fortsetzen.

Alle bisher behandelten Kalkulationssysteme waren speziell auf Geräte wie den Acorn B, Spectrum, Commodore 64 oder Sinclair QL ausgerichtet. Der geringe zur Verfügung stehende RAM-Bereich schränkt dabei die Funktionen und die Größe der Modelle stark ein. Ähnliche Programme, die für den IBM PC, den ACT Apricot oder andere kommerzielle Maschinen entwickelt wurden, haben den Vorteil von größeren Arbeitsspeichern und höherer Verarbeitungsgeschwindigkeit. Ein gutes Beispiel dafür ist Lotus 1–2–3, ein integriertes Tabellenkalkulationssystem mit Datenbank und Grafikfunktionen, das wir früher schon kurz gestreift hatten.

1–2–3 benötigt einen umfangreichen RAM-Bereich: Außer für den Programmcode der drei Anwendungen braucht das System so viel Platz, daß es eine Tabelle mit maximal 256 Spalten mal 1028 Zeilen verarbeiten kann. Bereits die ersten Versionen von 1–2–3 verlangten mindestens 128 KByte, um überhaupt funktionieren zu können – die neueren Versionen laufen nur auf Systemen mit wenigstens 256 KByte. Selbst wenn dem Anwender damit eine breite Palette von Funktionen zur Verfügung steht, sind die Kosten für Programme wie 1–2–3 durch den Preis eines geeigneten Computers außerordentlich hoch. In diesem Artikel zeigen wir, wie eine der interessantesten Eigenschaften von 1–2–3 eingesetzt wird – das Tastatur-Macro. Um Macros jedoch überhaupt verstehen zu können, müssen wir zunächst die Arbeitsweise untersuchen.

Beim Programmaufruf erscheint auf dem Bildschirm das Access-System von Lotus 1–2–3 – eine Reihe Bearbeitungsmöglichkeiten in Befehlsform. Setzt man den Cursor auf die erste Option und drückt Return, wird der Bildschirmaufbau vorbereitet. Bei 1–2–3 sind die Zeilen mit Buchstaben versehen und die Spalten numeriert. Wie Multiplan ist auch 1–2–3 menügesteuert. Beim Drücken der /-Taste wird das Hauptmenü dargestellt. Die Befehle lassen sich nun durch die Eingabe ihres ersten Buchstabens aufrufen oder durch die entsprechende Plazierung des Cursors und Return.

1–2–3 besitzt dermaßen viele Möglichkeiten, daß es verschiedene Ebenen von Untermenüs gibt. Auf der einen Seite stehen Hunderte von Bearbeitungsmöglichkeiten zur Verfügung, andererseits erfordert jeder Vorgang aber eine größere Anzahl Eingaben. Hier ein Beispiel: In 1–2–3 kann ein Feld oder ein Felderblock mit Namen belegt werden. Wenn Sie einen bestimmten Block ansprechen wollen – ihn beispielsweise in eine Formel einsetzen – brauchen Sie nur diesen Namen anzugeben:

A3 — B3 = C3
Feldbezug
VERKAUF — KOSTEN = GEWINN
Namensbezug

Die Namen erleichtern das Arbeiten mit großen Tabellensystemen sehr. Eine Gruppe von vier Feldern mit einem Namen zu versehen, erfordert folgende Eingaben:
/ — stellt das Hauptmenü dar;
R(ange) – teilt 1–2–3 mit, daß ein Felderblock und nicht die gesamte Tabelle angesprochen wird;
N(ame) – der angegebene Felderblock soll mit Namen versehen werden;
C(reate) – bereitet 1–2–3 vor, den Namen aufzunehmen und einzusetzen;
Sie geben einen Namen (z. B. "START") ein;
Return – kennzeichnet das aktive Feld als Anfang des Bereiches;
mit dem Cursor vier Felder nach rechts;
Return – kennzeichnet das jetzt aktive Feld als Ende des Bereiches.
Der gesamte Vorgang erfordert also außer der Angabe des Namens zehn weitere Eingaben.

Automatisierte Eingaben

Um dies auf eine akzeptable Zahl zu reduzieren, ermöglicht 1–2–3 die Einrichtung von Tastatur-Macros, die einfachen Programmen ähnlich sind und in der Betriebssprache von 1–2–3 geschrieben sind. Dabei werden die gewünschten Eingaben in einem kleinen Bereich der Tabelle gespeichert, mit Namen versehen und einer bestimmten Taste zugeordnet. 1–2–3 führt nun automatisch alle programmierten Eingaben aus, wenn diese Taste zusammen mit einer speziellen Funktionstaste betätigt wird, die auf dem IBM PC und kompatiblen Maschinen die Aufschrift „ALT" trägt.

Für eine Automatisierung der Namensgebung müssen wir dem Macro zunächst einen Felderblock der Tabelle zuordnen. Bei der Auswahl dieser Felder muß auf zwei Dinge geachtet werden. Zunächst müssen die Felder in einem Tabellenbereich liegen, der von Daten nicht überschrieben wird. Nun stellt 1–2–3

Speicherplatz allerdings nur für aktivierte Felder zur Verfügung. Wenn aber außen liegende Felder durch den Cursor aktiviert werden, wird auch für dazwischenliegende leere Felder Speicherplatz reserviert. Für Macros, die rechts außen in der Tabelle stehen, also mit weitem Abstand zu den Daten, werden mehrere KByte Speicherplatz auf die dazwischenliegenden leeren Felder verschwendet. Aus diesem Grund bringen Sie die Macros am besten an der linken Tabellenseite unter – beispielsweise in Spalte A, B und C.

Macros in die erste Spalte

Wir werden unser Macro in die Spalte A legen. Wenn dabei die Eingaben nicht mehr in ein einzelnes Feld passen, können sie auf den darunterliegenden Zeilen der gleichen Spalte fortgesetzt werden. Wir positionieren also den Cursor auf Feld A1 und geben folgendes ein:

' / R N C

Hier soll 1–2–3 auf die Eingabe des Namens warten. Ein in Klammern stehendes Fragezeichen (?) setzt die gewünschte Pause an diese Stelle. 1–2–3 wartet nun, bis der Anwender Return drückt und setzt erst dann die Ausführung des Macros fort. Die Klammern werden für alle Funktionen verwandt, die sich nicht durch eine bestimmte Taste ausführen lassen. Darunter fallen auch die Cursorbewegungen, die durch in Klammern gesetzte Richtungsangaben bezeichnet werden. Return wird durch eine Schlangenlinie (~) angezeigt. Das gesamte Macro sieht also folgendermaßen aus:

' / R N C (?) ~ (right) (right) (right) (right) ~

Als nächstes muß eine Taste – zum Beispiel „N" für Name – mit dem Macro belegt werden. Wir setzen dafür den Cursor auf Feld A1 und geben ein:

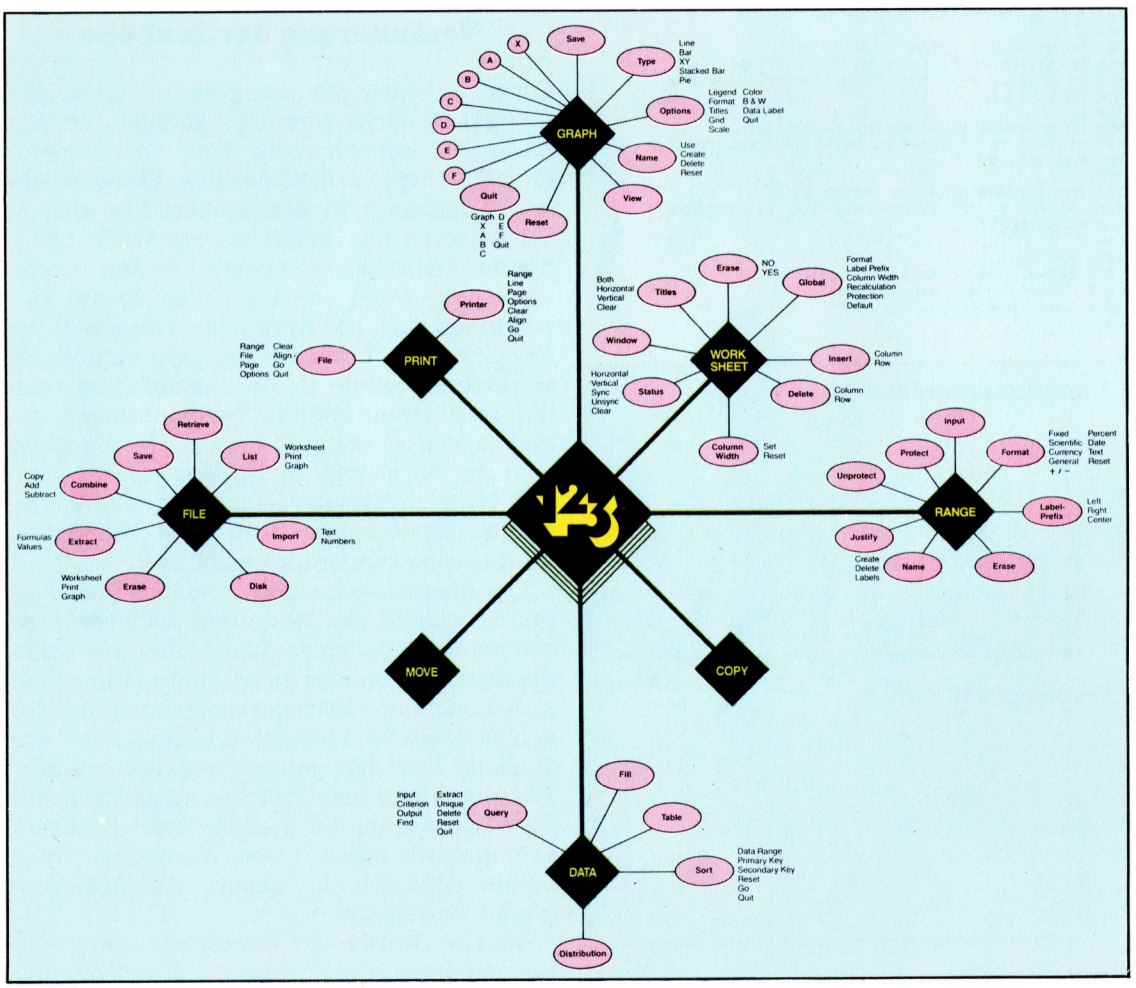
' / R N C \ N Return Return

Das \-Zeichen zeigt an, daß die ALT-Taste gedrückt werden muß. Die Zeichenkombination „\ N" (der Name unseres Macros) bedeutet daher für 1–2–3 „ALT N". Nachdem das Macro nun einen Namen hat, aktiviert die gleichzeitige Betätigung der Tasten ALT und N automatisch die gespeicherten Eingaben. Von nun an können wir Blöcke durch

ALT — N NAME Return

kennzeichnen.

Dieses Macro ist nur ein kleines Beispiel für das Potential der Tastatur-Macros. Es gibt grundsätzlich keine Grenze für die Anzahl der Tasten oder die Anzahl der Typen, die sich in Macros speichern lassen.



Lotus hat mehrere Menüebenen. Das Bild zeigt die eingebauten Kommandos. Andere Befehle können über die Tastatur-Macros definiert werden.

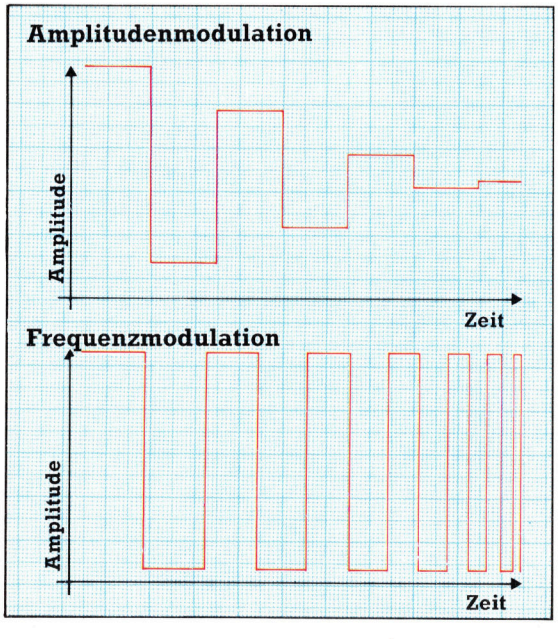
Voller Sound

Im letzten Abschnitt wurde erklärt, wie man mit Hilfe eines Digital/Analog-Wandlers unterschiedliche Wellenformen erzeugt. Neben der Wellenform wird jetzt die Steuerung von Lautstärke und Tonhöhe behandelt.

Die Lautstärke eines Tones ergibt sich aus der Differenz zwischen dem Maximal- und dem Minimalwert einer Welle. Diese Differenz bezeichnet man als „Amplitude".

Das folgende BASIC-Programm verändert die Werte im Datenregister des User Ports und zeigt, wie eine Amplitudenänderung bei einem digitalen Ton zu erzeugen ist.

```
10 REM **** CBM BAS FREQ/AMPLITUDE ****
20 :
25 DDR=56579:DATREG=56577
30 POKE DDR,255:REM ALL OUTPUT
40 FOR I=255 TO 0 STEP-15
50 FOR J=1 TO 100
60 POKE DATREG,I:POKE DATREG,0
70 NEXT J,I

800 REM **** SAMPLE PROGRAM ****
805 :
806 UP$=CHR$(145)
810 DIV=49798: REM AMPLITUDE FACTOR LOCATION
820 DEL=49799: REM DELAY FACTOR LOCATION
830 TME=49800: REM DURATION FACTOR LOCATION
840 CALL=49801:REM PROGRAM START ADDRESS
850 :
860 DDR=56577:POKEDDR,255:REM ALL OUTPUT
870 :
880 PRINTCHR$(147):REM CLEAR SCREEN
890 PRINT:INPUT"AMPLITUDE FACTOR 0-7";AF
900 IF AF<0 OR AF>7 THEN PRINT UP$;UP$;:GOTO890
910 POKE DIV,AF
920 :
930 PRINT:INPUT"DELAY FACTOR 1-101";DF
940 IF DF<1 OR DF>101 THEN PRINT UP$;UP$;:GOTO930
945 POKE DEL,DF
950 :
960 PRINT:INPUT"DURATION FACTOR 0-15";TF
970 IF TF<0 OR TF>15 THEN PRINTUP$;UP$;:GOTO960
980 POKE TME,TF
990 :
1000 SYS CALL
1010 GETA$:IFA$="" THEN 1010
1020 IFA$="X" THEN 880:REM RESTART
1030 GOTO 1000REM ANOTHER BEEP
```



Eine Wellenform läßt sich durch die Frequenz und die Amplitude verändern. Die Frequenz, also die Anzahl der ausgegebenen Wellenzyklen pro Sekunde, bestimmt die Tonhöhe. Die Amplitude ist die Differenz zwischen dem höchsten und dem niedrigsten Punkt einer Schwingung – sie entscheidet über die Lautstärke. Die erste Zeichnung zeigt eine abnehmende Amplitude – der Ton wird schrittweise leiser. Die zweite Zeichnung stellt das Diagramm eines allmählich höher werdenden Klanges dar.

Zu Beginn des Programms wird eine einfache Rechteckschwingung zwischen den Werten 0 und 255 erzeugt; die Amplitude dieser Schwingung beträgt also 255. Danach wird der Maximalwert schrittweise um jeweils 15 vermindert, und die Schwingung wird geringer. Beim Abhören des erzeugten Tons über die Stereoanlage oder mit Kopfhörern werden Sie feststellen, daß die Lautstärke entsprechend abnimmt, bis zum Schluß nichts mehr zu hören ist. Anhand dieser Methode läßt sich die Lautstärke digitaler Klänge über das Datenregister des User Ports verändern.

Die Tonhöhe eines Klanges wird von der Frequenz der erzeugenden Welle bestimmt, also der Anzahl der Schwingungszyklen pro Sekunde. Ein Ton ist um so höher, je größer die Frequenz ist.

Veränderung der Tonhöhe

Für eine digitale Steuerung der Frequenz gibt es zwei Möglichkeiten: Eine Methode besteht darin, bei zunehmender Frequenz entsprechend weniger Abschnitte bzw. Einzelpunkte der Wellenform wiederzugeben. Für eine in 100 einzelne Abschnitte geteilte Welle benötigt das Programm eine bestimmte Zeit, bis die einzelnen Werte in das Datenregister geschrieben sind. Die Anzahl der Werte legt die Frequenz des Tons fest. Um die Frequenz zu verdoppeln, dürfte das Maschinenprogramm nur noch jeden zweiten Wellenabschnitt berücksichtigen, und für eine Verdreifachung wird nur noch jeder dritte Wert gelesen usw. Der Nachteil dieser Methode ist, daß kleine Frequenzänderungen nicht ohne Verzerrung der Schwingungsform möglich sind.

Die Alternative besteht darin, mit einer Programmschleife die Werte der Wellenform so schnell wie möglich zu durchlaufen und damit die Maximalfrequenz zu erzeugen. Diese läßt sich durch kurze Verzögerungen innerhalb der Schleife wieder reduzieren. Damit kann die Tonhöhe sehr viel genauer reguliert werden. Allerdings muß zum Erreichen möglichst hoher Töne die Anzahl der Wellen-Einzelabschnitte so gering wie möglich sein. Wir werden diese zweite Methode in einem Maschinenprogramm verwenden.

Mit der Division der Einzelwerte einer Wellenform durch eine Konstante läßt sich die Am-

plitude variieren, ohne die Schwingungsform zu verändern. Dabei wird entweder jeder Einzelwert nach dem Laden in den Akkumulator dividiert oder die Werte werden vor der Schleife des Hauptprogramms dividiert. Besser ist die zweite Methode: Sie verzögert den Programmablauf nicht und verursacht deshalb keine Verminderung der maximalen Frequenz. Die ursprünglichen Werte einer Wellenform werden einzeln durch eine Konstante dividiert und zu einer neuen Tabelle zusammengesetzt, die dann vom Hauptprogramm als Datenbasis für die Wellenform verwendet wird. Die Konstante bestimmt, wie oft nacheinander der jeweilige Zahlenwert nach rechts verschoben wird. Jede Verschiebung entspricht einer Division durch Zwei, Konstante „n" bewirkt also die Teilung der Wellenwerte durch 2n.

Setzt man eine Null als Konstante ein, so werden die ursprünglichen Werte der Schwingungsform verwendet, wobei das Programm selbsttätig zur Adresse der ursprünglichen Werteliste springt.

Die Ausführung der Hauptprogramm-Schleife wird durch verzögernde Befehle künstlich verlangsamt. Dies läßt sich durch das Dekrementieren einer 8-Bit-Zahl in einem der Indexregister oder durch Dekrementieren eines 16-Bit-Wertes im Speicher bewerkstelligen. Für unsere Anwendungen ist jedoch ein 8-Bit-Wert ausreichend.

Probleme treten weniger bei der Erzeugung einer großen Verzögerungszeit (niedrige Frequenz) auf, sondern bei den kurzen Verzögerungszeiten (hohe Frequenzen). Bisher haben wir unsere Schwingung in 80 Einzelwerte aufgeteilt – wegen der zusätzlichen Programmteile für die Verzögerung müssen wir jetzt mit einer kleineren Zahl von Werten auskommen.

So sieht der Code für die Verzögerungsschleife aus:

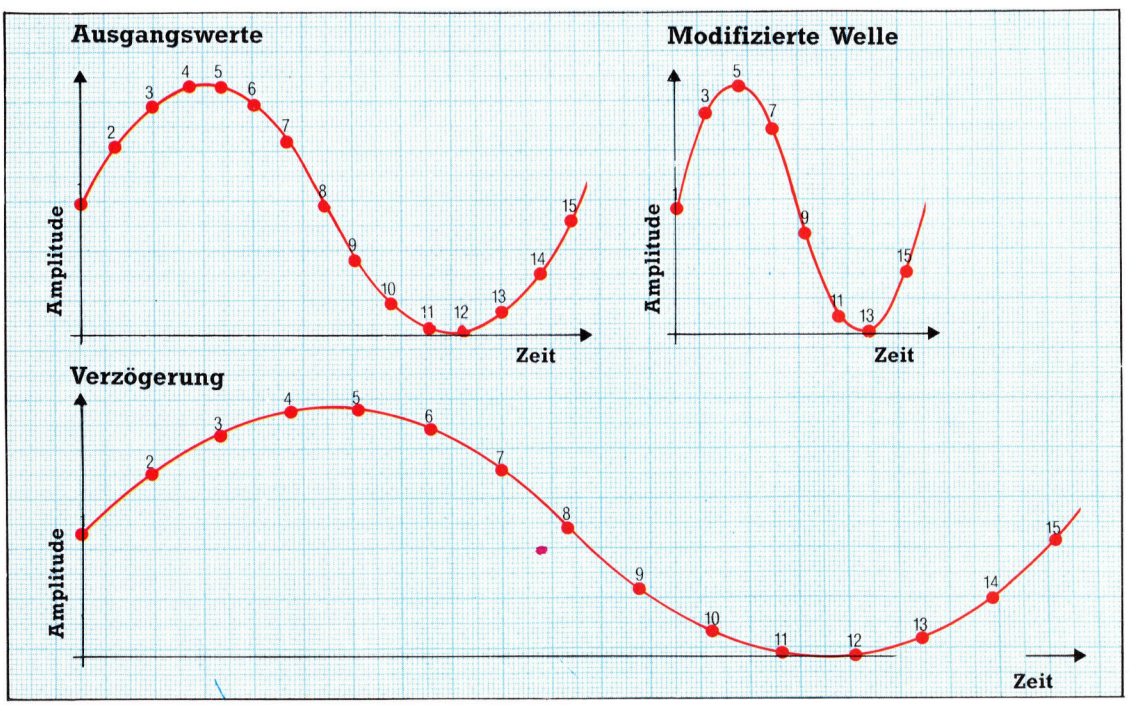
```
MAIN                                    Zeit in Rechnerzyklen
        LDX #$00                        2
NEXVAL
        LDA AMPTAB,X                    4
        LDY DELAY                       4
MORDEL
        DEY                             2
        BNE MORDEL      3 (2 bei Versagen der Schleife)
        STA PORT                        4
        INX                             2
        CPX # STEPS                     2
        BNE NEXVAL      3 (2 bei Versagen der Schleife)
```

Die Gesamtzahl der nötigen Rechnerzyklen berechnet sich folgendermaßen: 2+(4+4+(2+3)×Verzögerung−1+4+2+2+3)× Steps−1=1+(18+5×Verzögerung)×Steps. Mit 1 als kleinstem Verzögerungswert ergibt sich die größte Frequenz : Höchste Frequenz = 1000000/(1+23×Steps).

Bei einer Maximalfrequenz von ca. 3000 Hz darf die Wellenform also nur in fünfzehn Schritten gelesen werden. Bei dieser Anzahl ergibt sich aus der ursprünglichen Formel: die Anzahl der Rechnerzyklen=271+75×Verzögerung.

Wenn wir als niedrigste Frequenz 128 Hz wählen (zwei Oktaven unter dem mittleren C), ergibt sich als Verzögerungswert 101. Dieser Wert läßt sich in einem Indexregister speichern und dekrementieren.

Die Veränderung der Frequenz erzeugt allerdings noch eine weitere Schwierigkeit: Eine vorgegebene Zahl von Wiederholungen der Schleife benötigt bei höheren Tönen weniger Zeit als bei tieferen. Um dies auszugleichen, muß die Tondauer mit einer zusätzlichen Formel berechnet werden, unabhängig von der



Die Frequenz einer digital „gesampelten" Welle kann durch Beschränkung auf eine kleinere Anzahl von Werten oder durch Zeitverzögerung bei der Wiedergabe der Werte verändert werden. Wenn sich die ursprüngliche Werteliste aus fünfzehn Daten zusammensetzt, kann durch Überspringen der Zwischenwerte die Frequenz verdoppelt werden. Umgekehrt läßt sich aber auch durch Verzögerung die Zeit für einen Wellendurchlauf verdoppeln – der Ton erklingt mit halber Frequenz. Die erste Methode erlaubt zwar eine genaue Abtastung der Wellenform, ist aber für eine exakte Bestimmung der Tonhöhe schlecht geeignet. Das geht mit der zweiten Methode besser, dafür kann die Schwingungsform nur mit begrenzter Wertezahl festgelegt werden.

Frequenz des Tons. Die Einheit der Tondauer soll eine Fünfzigstelsekunde sein, die Anzahl der Schleifendurchläufe ergibt dann 1000000/(50×(271+75×Verzögerung). Eine Berechnung in Maschinensprache wäre schwierig und zeitraubend, daher werden die entsprechenden Werte in einer BASIC-Tabelle untergebracht, die für alle Verzögerungswerte (von 1 bis 109) die Anzahl der Schleifendurchläufe enthält. Das Maschinenprogramm kann nun einfach die Zahl der Wiederholungen in einem Zähler unterbringen, dessen Wert bei jedem Durchlauf um Eins vermindert wird.

Besitzer eines Commodore 64, die über einen Assembler verfügen, können das Quellprogramm eingeben und es zu einem Zielprogramm assemblieren. Mit einem Aufruf-Programm kann es jederzeit wieder von Cassette bzw. Diskette geladen werden, wobei auch die BASIC-Wertelisten neu aufgestellt werden. Geben Sie diese ein, und lassen Sie das Programm laufen. Danach NEW eingeben und das BASIC-Programm mit den vom Maschinenprogramm benötigten Daten eintippen. Zum Testen müssen Ausgangsbuffer und D/A-Wandler angeschlossen werden.

Beim Acorn B geht es einfacher – nur die Acorn-Version eingeben und mit RUN starten.

```
 10 REM BASIC LOADER FOR FREQ/AMP PROGRAM
 20 :
 30 FOR I=49801 TO 49911
 40 READ A:POKEI,A
 50 CC=CC+A
 60 NEXT I
 70 READ CS:IF CC<>CS THENPRINT
    "CHECKSUM ERROR":STOP
100 DATA172,134,194,208,8,169,0,141
110 DATA204,194,76,174,194,169,15,141
120 DATA204,194,162,15,172,134,194,189
130 DATA0,194,24,106,157,15,194,136
140 DATA208,248,202,16,239,120,169,0
150 DATA141,133,194,174,135,194,189,30
160 DATA194,172,136,194,240,7,10,46
170 DATA133,194,136,208,249,141,132
180 DATA194,162,0,189,15,194,172,135
190 DATA194,136,208,253,141,1,221,232
200 DATA224,15,208,239,173,132,194,56
210 DATA233,1,141,132,194,173,133,194
220 DATA233,0,141,133,194,208,218,169
230 DATA0,205,132,194,208,211,88,96
240 DATA16186:REM*CHECKSUM*
```

```
 10 REM **** CALLING PROGRAM ****
 20 REM ****        AND        ****
 30 REM ****   TABLE SET UP    ****
 40 :
 45 DN=8:REM IF CASSETTE DN=1
 50 IF A=0 THEN A=1:LOAD"FREQ.HEX",DN,1
 60 :
 70 REM **** SET UP SHAPE TABLE ****
 75 :
 80 S=15:TB=12*4096+2*256
 90 FOR I=0 TO S-1
100 Y=127*SIN(X)+127
110 POKE TB+I,Y
120 X=X+2/S
130 NEXT I
140 :
150 REM **** SET UP FREQ/DELAY TABLE ****
160 :
170 TB=TB+2*S
180 FOR D=0 TO 101
190 TV=10^6/(50*(271+75*D))
200 POKETB+D,TV
210 NEXT D
```

```
;++++++++++++++++++++
;++++++++++++++++++++
;++                  ++
;++     CBM 64       ++
;++   FREQUENCY      ++
;++ AND AMPLITUDE    ++
;++  GENERATION      ++
;++                  ++
;++++++++++++++++++++
;++++++++++++++++++++
;
STEPS  = 15      ;NO. OF STEPS PER WAVE CYCLE
PORT   = 56577
;
*=     $C200
SHAPTB *=*+STEPS         ;WAVE SHAPE TABLE
AMPTAB *=*+STEPS         ;AMPLITUDE TABLE
LOOPTB *=*+102           ;FREQ/DELAY TABLE
COUNT  *=*+2             ;LOOP COUNTER
DIVISN *=*+1             ;DIV OF WAVE FACTOR
DELAY  *=*+1             ;DELAY FACTOR
TIME   *=*+1             ;NOTE DURATION FACTOR
;
;++++ SET AMPLITUDE TABLE ++++
;
       LDY DIVISN
       BNE CONT
       LDA #<SHAPTB      ;MODIFY PROGRAM
       STA NEXVAL+1      ;TO LOAD SHAPTB
       JMP INITC
;
CONT
       LDA #<AMPTAB
       STA NEXVAL+1
       LDX #STEPS
NEXT
       LDY DIVISN
       LDA SHAPTB,X
MORE
       CLC
       ROR A
       STA AMPTAB,X
       DEY
       BNE MORE
       DEX
       BPL NEXT
;
;++++ SET COUNT VALUE ++++
;
INITC
       SEI
       LDA #$00
       STA COUNT+1       ;INIT COUNT HIBYTE
       LDX DELAY
       LDA LOOPTB,X
       LDY TIME
       BEQ NOMULT
MULT
       ASL A
       ROL COUNT+1
       DEY
       BNE MULT
NOMULT
       STA COUNT
;
;++++ MAIN PROGRAM LOOP ++++
;
MAIN
       LDX #$00
NEXVAL
       LDA AMPTAB,X
       LDY DELAY
MORDEL                   ;DELAY LOOP
       DEY
       BNE MORDEL
;
       STA PORT
       INX
       CPX #STEPS
       BNE NEXVAL
;
;++++ DECREMENT COUNT ++++
;
       LDA COUNT
       SEC
       SBC #$01
       STA COUNT
       LDA COUNT+1
       SBC #$00
       STA COUNT+1
       BNE MAIN
       LDA #$00
       CMP COUNT
       BNE MAIN
       CLI
       RTS
```

```
 15REM ****************************
 20REM **                        **
 25REM **          BBC           **
 30REM ** FREQUENCY & AMPLITUDE **
 40REM **        GENERATION      **
 50REM **                        **
 60REM ****************************
 90MODE 7
 95steps=15:port=&FE60
 97ddr=&FE62:? ddr=255:REM ALL OUTPUT
100HIMEM=HIMEM-&101:REM RESERVE TABLE SPACE
110shape_table=HIMEM+1
112amplitude_table=shape_table+steps
114loop_table=amplitude_table+steps
115PROCset_tables
120PROCassemble code
140REM **** BASIC TEST PROGRAM ****
160CLS
170PRINT:INPUT"AMPLITUDE FACTOR 0-7";AF
180IF AF<0 OR AF>7 THEN 170
190?div_factor=AF
200PRINT:INPUT"DELAY FACTOR 1-101";DF
210IF DF<1 OR DF>101 THEN 200
220?delay_factor=DF
230PRINT:INPUT"DURATION FACTOR 0-15";TF
240IF TF<0 OR TF>15 THEN 230
250?time_factor=TF
265REPEAT
270CALL freq
280A$=GET$
290UNTIL A$="X"
300GOTO 160:REM RESTART
900END
999:
1000DEF PROCassemble_code
1005DIM MC% &FF
1010FOR opt%=0 TO 3 STEP 3
1020P%=MC%
1030count=P%:P%=P%+2
1040div_factor=P%:P%=P%+1
1050delay_factor=P%:P%=P%+1
1060time_factor=P%:P%=P%+1
1070[
1075OPT opt%
1080\**** SET AMPLITUDE TABLE ****
1090\
1095.freq
1100        LDY div_factor
1110        BNE cont
1120        LDA #shape_table MOD 256
1130        STA nexval+1
1140        JMP initc
1150\
1160.cont
1170        LDA # amplitude_table MOD 256
1180        STA nexval+1
1190        LDX #steps
1195.next
1200        LDY div_factor
1210        LDA shape_table,X
1220.more
1230        CLC
1240        ROR A
1250        STA amplitude_table
1260        DEY
1270        BNE more
1280        DEX
1290        BPL next
1300\
1310\**** SET COUNT VALUE ****
1320\
1330.initc
1340        SEI
1350        LDA #0
1360        STA count+1
1370        LDX delay_factor
1380        LDA loop_table,X
1390        LDY time_factor
1400        BEQ nomult
1410.mult
1420        ASL A
1430        ROL count+1
1440        DEY
1450        BNE mult
1460.nomult
1470        STA count
1480\
1490\**** MAIN PROGRAM LOOP ****
1500\
1510.main
1520        LDX #0
1530.nexval
1540        LDA amplitude_table,X
1550        LDY delay_factor
1560.mordel
1570        DEY
1580        BNE mordel
1590\
1600        STA port
1610        INX
1620        CPX #steps
1630        BNE nexval
1640\
1650\**** DECREMENT COUNT ****
1660\
1670        LDA count
1680        SEC
1690        SBC #1
1700        STA count
1710        LDA count+1
1720        SBC #0
1730        STA count+1
1740        BNE main
1750        LDA #0
1760        CMP count
1770        BNE main
1780        CLI
1790        RTS
1800]
1810NEXT opt%
1820ENDPROC
1999:
2000DEF PROCset_tables
2005x=0
2010FOR I=shape_table TO shape_table+steps-1
2020y=127*SIN(x)+127
2030? I=y
2040x=x+2*PI/steps
2050NEXT I
2060:
2070FOR delay=0 TO 101
2080loop_val=10^6/(50*(271+75*delay))
2090loop_table?delay=loop_val
2100NEXT delay
2120ENDPROC
```

Schwarze Magie

„Necromancer" von Synapse Software, geschrieben von Bill Williams, kann mit einem Schauspiel verglichen werden. Die Handlung ist in drei Akte geteilt. Im ersten wird die Grundlage geschaffen, und danach führt das Geschehen unaufhaltsam auf den Höhepunkt zu: die Auseinandersetzung zwischen Gut und Böse.

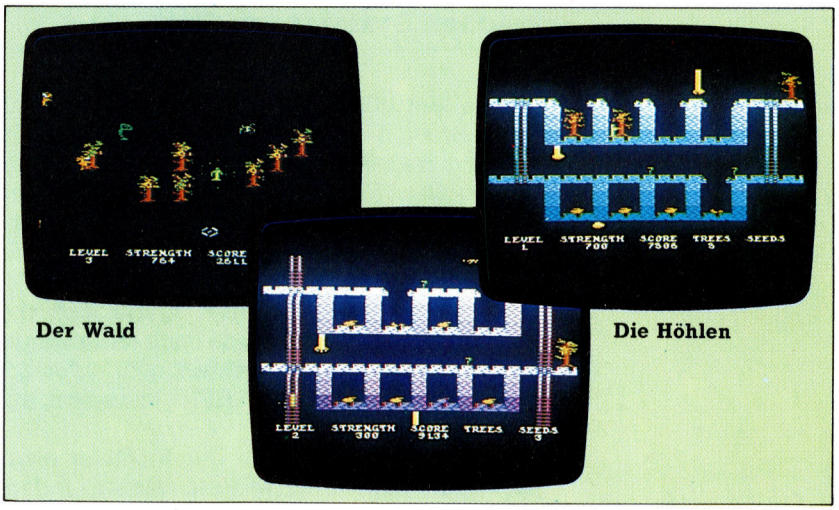
Die Ouvertüre wird sowohl beim Atari als auch beim C 64 leicht geladen. Die Auftaktgrafiken sind von einleitender Musik begleitet, die die Klangmöglichkeiten beider Rechner voll ausnutzen.

Akt Eins/ Der Wald: Das Spiel beginnt im „Zeitalter der Finsternis", in dem „Tetragorn, der teuflische Zauberer, herrscht". Der Spieler übernimmt die Rolle des Illuminar, eines Druiden, der als „Verteidiger der Wahrheit und Beschützer der menschlichen Rasse" beschrieben wird. Es ist klar, daß dies keine einfache Aufgabe ist. Zu Spielbeginn erscheint der Druide in einem dunklen, offenen Raum. Eine Sternenaura schützt vor Hunderten kleiner Menschenfresser, die unermüdlich über den Bildschirm marschieren und gewaltige Säbel im Gleichklang zur Musik schwingen. Mit dem Joystick steuert man einen magischen „Besen", der über den Bildschirm gleitend die Menschenfresser eliminiert und somit zum Punktesammeln beiträgt.

Bäume pflanzen

Die Plazierung des „Besens" an gewünschter Stelle bei gleichzeitigem Drücken des Joystick-Actionknopfs erlaubt das Anpflanzen von Bäumen. So schafft man einen Wald, der sich im späteren Spielverlauf als hilfreich erweisen kann. Die Setzlinge müssen vor den Säbeln der Menschenfresser und vor den Spinnen Tetragorns beschützt werden. Gleichzeitig ist das eigene Kräftepotential im Auge zu behalten, das durch den Biß einer Spinne schwinden und eventuell sogar zum Tode führen kann. Nach fünf Spielebenen endet der erste Akt mit einem Angriff der Spinnen, und die Handlung wird eingefroren. Nach einer Zählung der gepflanzten Bäume befindet sich der Druide im nächsten Akt.

Akt Zwei/ Die Höhlen: Hier hausen die Spinnen. Es gibt fünf verschiedene Ebenen, die sich aus jeweils zwei Schichten von vier Höhlen zusammensetzen. In den Kammern liegen Spinneneier, die beim Schlüpfen in unterschiedlichen Farben blitzen. Auf dem Schirm wird ebenfalls ein „Baumbehälter" gezeigt (darin befinden sich die Bäume, die man im ersten Akt gepflanzt hat). Mittels des „Besens" wird nun ein Baum über einer Höhle positioniert. Ist man schnell genug, schlägt der Baum Wurzeln und dringt durch die Höhlendecke, wobei er gleichzeitig die Eier zerstört, bevor sie aufbrechen.



In Necromancer sind Sprite-Grafiken mit hochauflösenden Hintergrunddarstellungen kombiniert, die in allen drei Akten beeindruckende Szenarios erzeugen. Die funkelnden Baumwipfel, der pfeilschnell huschende Druiden-„Besen", die Bewegung der Spinnen und Menschenfresser sowie das Feuer des Necromancer machen das Programm zu einem „optischen Vergnügen" und tragen zur Spielspannung bei.

Zusätzliche Gefahr bringen „die Hände des Schicksals", die aus der Decke herausgreifen und alles zu fassen versuchen, was sich unter ihnen befindet – Druiden, Bäume oder Fragezeichen. Letztere stellen rätselhafte Schätze oder Boni dar.

Akt Drei: The Necromancer's Lair. Der Höhepunkt spielt auf einem unheimlichen, dunklen Friedhof. Die Grabsteine markieren die zahlreichen Gräber des Necromancer (Geisterbeschwörers) und müssen zerstört werden, um seine Reinkarnation zu verhindern. Doch so einfach ist der Kampf zwischen Gut und Böse nicht, da alle Spinnen, die im zweiten Akt entkommen konnten, sich nun in „Zombies" verwandeln und antreten, um den Necromancer zu verteidigen.

Es ist wichtig, reichlich Energie für diesen Teil der Handlung in Reserve zu haben, da es hier um alles geht.

Necromancer: Für Atari und C 64
Hersteller: Synapse Software
Autoren: Bill Williams (Atari), Scott und Steve Coleman (C 64)
Joysticks: Standard-Atari/Commodore-Joysticks
Format: Diskette und Cassette

Computer-Analyse

Die Computer der „fünften Generation“ werden nicht mehr im üblichen Sinne programmiert, sondern „verstehen“ „natürliche Sprachen“. Das folgende Programm soll die Prinzipien eines solchen Systems verdeutlichen.

Das hier gezeigte Programm simuliert das Verhalten eines Gesprächspartners in einem therapeutischen Gespräch. Mit anderen Worten: Der Therapeut „diktiert“ das Gespräch nicht, sondern führt es nur mit knappen Kommentaren wie „ERZAEHLEN SIE MIR MEHR . . .“ oder „WARUM IST DAS WICHTIG . .?“ fort. Das Programm merkt sich die Konversation und stellt diese nach einer vorgegebenen Anzahl von Wortwechseln, oder sobald Sie „AUF WIEDERSEHEN“ eingeben, auf dem Bildschirm dar.

Das Programm kann in der Richtung einer Pseudo-Intelligenz entwickelt werden, indem es eine Analyse der Eingaben des Anwenders vornimmt und so entsprechende Antworten auswählt. In eingeschränktem Rahmen geschieht dies bereits durch die Programmzeile 3100, in der die Eingabe auf „AUF WIEDERSE-

Programm

```
100 GOSUB 2000: REM INITIALISIERUNG
200 FOR L=1 TO 2*LT STEP 2
300 PRINT O$: INPUT I$
400 GOSUB 3000: REM ANALYSE
500 NEXT L
1000 PRINT TAB(5);"------ DIE SPRECHSTUNDE IST
     BEENDET ------"
1100 PRINT TAB(4);"- DRUECKE EINE TASTE, UM FORT-
     ZUFAHREN -"
1200 A$=INKEY$: IF A$="" THEN GOTO 1200
1300 GOSUB 5000: REM REPORT
1900 END
1999 REM ****************************
2000 REM ** INITIALISIERUNG U/R **
2001 REM ****************************
2050 LT=10:AN=10:T9=500:EX=2*LT
2100 DIM R$(AN):DIM H$(2*LT)
2200 DATA "JA. .","ERZAEHLE MIR MEHR. .", "FAHRE
     FORT. .","UND. .","SO. ."
2250 DATA "IST DAS WICHTIG. .","WARUM IST DAS VON
     BEDEUTUNG. ."
2300 DATA "BITTE ERKLAERE DAS. .","WARUM SAGST
     DU DAS. ."
2350 DATA "INWIEWEIT BERUEHRT DICH DAS. ."
2500 FOR K=1 TO AN:READ R$(K):NEXT K
2600 CLS:0$="HALLO — WO LIEGT DAS PROBLEM. ."
2950 RETURN
2999 REM ********************
3000 REM ** ANALYSE U/R **
3001 REM ********************
3100 IF I$="AUF WIEDERSEHEN" THEN EX=L-1:
     L=2*LT:RETURN
3200 H$(L)=O$:H$(L+1)=I$
3300 R9 = INT(AN*RND+1):IF R9=R0 THEN GOTO 3300
3400 O$=R$(R9):R0=R9
3950 RETURN
4999 REM ******************
5000 REM ** REPORT U/R **
5001 REM ******************
5050 CLS:PRINT TAB(10);"**REPORT**"
5100 FOR K=1 TO EX STEP 2
5150 PRINT TAB(5);H$(K)
5200 PRINT H$(K+1)
5300 FOR D=1 TO T9:NEXT D
5400 NEXT K
5900 RETURN
```

HEN" überprüft wird. Man kann diese Analyse durch Überprüfung auf die Wörter „JA" oder „NEIN" erweitern, und auf diese Art spezifische Antworten erzielen – beispielsweise durch einfaches „WARUM?" oder „WARUM NICHT?". Ferner könnte man überprüfen, ob der Anwender sich wiederholt und, falls dies der Fall ist, entsprechend antworten oder die Sprechstunde abbrechen. Außerdem kann man überprüfen, ob eine Antwort mit einem Fragezeichen oder einem Ausrufungszeichen endet und entsprechend mit „WARUM FRAGST DU DAS . .?" oder „WARUM REGT DICH DAS AUF . .?" antworten.

Die Textanalyse

Man könnte nun eine Tabelle mit Schlüsselwörtern erstellen und diese nach bestimmten Wörtern durchsuchen, damit man aus einer anderen Tabelle eine spezifische Antwort auswählen kann – vorausgesetzt, man findet ein passendes Wort. Die Auswahl der Schlüsselwörter und der Antworten ist vom Thema der erwarteten Konversation abhängig. Das Problem dieser Methode – und aller Methoden, die Texte durchsuchen müssen – ist die Zeitspanne, die zur Analyse selbst eines kurzen Satzes notwendig ist. Die Verarbeitungsgeschwindigkeit von BASIC ist dabei der einschränkendste Faktor. Ein wirklich gutes Programm muß daher in Maschinensprache geschrieben werden. Für unseren Zweck können wir jedoch eventuell entstehende Verzögerungen akzeptieren. Sogar das hier gezeigte einfache Programm kann überraschende Ergebnisse bringen und zeigt einen kleinen Teil der Problematik auf, die die Computer der „fünften Generation" zu lösen haben.

Wenn wir beabsichtigen, die Wörter des Anwenders zu analysieren, müssen mehrere Angleichungen erfolgen: Am interessantesten ist dabei die syntaktische Analyse – die Aufgliederung eines Satzes in seine einzelnen Sprach-Komponenten, wie Pronomen, Verb und Substantiv. Hierzu sind ein Grundgerüst syntaktischer und grammatikalischer Regeln sowie Tabellen mit Pronomen, Präpositionen, Konjunktionen, Wortumwandlungen usw. erforderlich. Wie Sie sehen, ist diese Analyse nicht gerade einfach auszuführen. Leichter wäre es dagegen, das längste Wort innerhalb einer Antwort auszuwählen und den Anwender nach seinen Gedanken zu diesem Wort zu fragen. Dabei wird darauf aufgebaut, daß in einem kurzen Satz das längste Wort auch am wichtigsten ist. Diese Analyse könnte dadurch noch verbessert werden, daß zufällig zwischen den Wörtern, die länger als beispielsweise fünf Buchstaben sind, gewählt wird oder daß das Wort verwendet wird, das dem Wort „Ich", „Mir", „Du" oder „Dir" folgt.

Eine Auswahl bzw. eine Verbesserung dieser Methoden ist eine gute Übung für Programmierer. Sie bekommen einen optimalen Einblick in die Komplexität der Sprache und ihre Analyse. Zudem haben Sie die Möglichkeit, sich beliebig lange mit einem „perfekten Gesprächspartner" zu unterhalten.

```
3420 IF LEFT$(I$,3)="JA" THEN O$="WAS MACHT
     DICH SO SICHER. .":RETURN
3450 IF LEFT$(I$,2)="NEIN" THEN O$="WARUM
     NICHT. .":RETURN
3500 Z$=RIGHT$(I$,1)
3520 IF Z$="?" THEN O$="WARUM FRAGST DU
     MICH. ."
3550 IF Z$="!" THEN O$="WARUM REGT DICH DAS
     AUF . ."
3600 IF R9 < AN/4 THEN GOSUB 4000
3950 RETURN
3999 REM **************************
4000 REM ** LAENGSTES WORT U/R **
4001 REM **************************
4050 W=1:H=1:WL=1:S=1
4100 I$=I$+" ":L9=LEN(I$)
4120 FOR C=1 TO L9:FOR P=C TO L9
4150 Z$=MID$(I$,P,1)
4170 IF Z$=" " THEN W=P-C:H=C:C=P:P=L9
4200 NEXT P
4220 IF W > WL THEN WL=W:S=H
4250 NEXT C
4270 O$="WAS BEDEUTET "+MID$(I$,S,WL)+"
     FUER DICH. ."
4450 RETURN
```

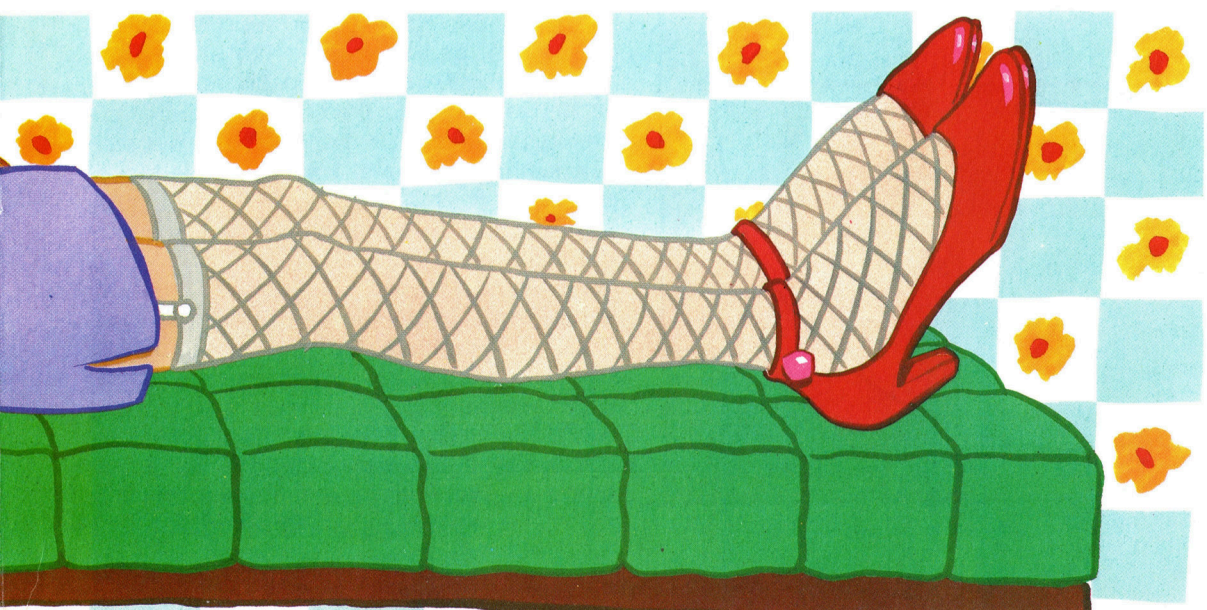
Fügen Sie diese Zeilen in das erste Programm ein, um die Routinen „JA/NEIN" zu integrieren und um gleichzeitig das längste Wort erkennen zu können.

BASIC-Dialekte
Dieses Programm ist in Microsoft-BASIC geschrieben. Besitzer eines Spectrums müssen bei allen Zuordnungs-Anweisungen LET einfügen sowie die TAB-Werte anpassen.

Spectrum
Ersetzen Sie DIM R$(AN): DIM H$(2*LT) durch DIM R$(AN,30): DIM H$(2*LT,100). Ersetzen Sie ferner LEFT$(I$,3) und LEFT$(I$,2) durch I$(TO 3) und I$(TO 2). Ersetzen Sie außerdem RIGHT$(I$,1) durch I$(LEN(I$)). Ersetzen Sie MID$(I$,P,1) durch I$(P) und MID$(I$,S,WL) durch I$(S TO S+WL-1).

C 64 und VC 20
Ersetzen Sie CLS durch PRINT CHR$(147). Ersetzen Sie INT(AN*RND+1) durch INT(RND(1)*AN+1). Ersetzen Sie A$=INKEY$ durch GET A$.

Acorn B
Ersetzen Sie A$=INKEY$ durch A$=INKEY$(0). Ersetzen Sie INT (AN*RND+1) durch RND (AN).

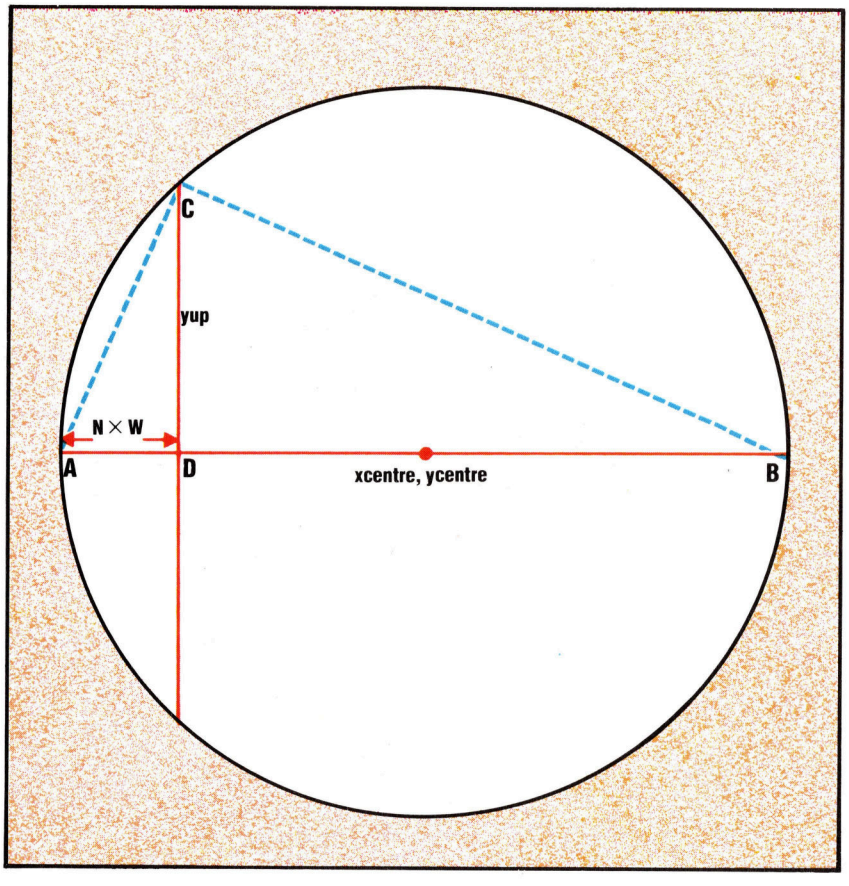


Computerkreise

Mit den Maschinenroutinen Plotsub und Linesub haben wir die hochauflösenden Fähigkeiten des Commodore 64 für unsere Zwecke eingesetzt. In diesem Artikel wird eine Routine gezeigt, die bei Angabe der Mittelpunktkoordinaten und des Radius Kreise zeichnet.

Heimcomputer können keine exakten Kreise erzeugen. Wie nahe die Zeichnung einem perfekten Kreis kommt, bestimmen hauptsächlich die dafür eingesetzte Methode sowie die Länge und Komplexität der Berechnungsroutine. Unter BASIC werden die Koordinaten der Kreispunkte normalerweise mit Sinus- und Cosinusfunktionen und Quadratwurzeln berechnet. Da sich diese beiden Methoden jedoch nur schwer in Maschinencode umsetzen lassen, stellen wir hier eine Methode vor, die speziell für diesen Code geeignet ist.

Dabei wird der Kreisdurchmesser in eine Anzahl gleicher Abschnitte mit der Länge W unterteilt. Von jedem Teilungspunkt reicht eine Linie hinauf bis zu Punkt C der Kreislinie. Die Abbildung zeigt eine dieser Linien, die N Unterteilungen vom linken Ende des Durchmessers AB entfernt ist. Bei der Verbindung der Punkte A und B mit C entstehen die beiden rechtwinkligen Dreiecke ACD und BCD.



Aus der Darstellung lassen sich nach dem Satz des Pythagoras folgende Formeln ableiten:

$AC^2 = AD^2 + CD^2$

$CB^2 = DB^2 + CD^2$

Die Addition der beiden Gleichungen ergibt:

$AC^2 + CB^2 = AD^2 + DB^2 + 2CD^2$

Da bei Kreisen das Dreieck ABC jedoch ebenfalls rechtwinklig ist, können wir sagen:

$AC^2 + CB^2 = AB^2$

Die modifizierte Gleichung setzt sich wie folgt zusammen:

$AB^2 = AD^2 + DB^2 + 2CD^2$

CD entspricht „yup" – dem Abstand des Durchmessers zu der Kreislinie. AD ist N×W und AB ist 2×R (R = Kreisradius). Nachdem wir die Werte in die Gleichung eingesetzt und diese umgeformt haben, erhalten wir:

$2yup^2 = (2R)^2 - (NW)^2 - (2R-NW)^2$

$yup^2 = 2RNW - N^2 W^2$

Wenn wir nun den Durchmesser in 64 gleiche Abschnitte unterteilen, dann ist $W = 2*R/64$, was sich auch als $W = R/32$ schreiben läßt. In die Gleichung eingesetzt ergibt sich:

$yup^2 = 2RNR/32 - N^2 R^2/32^2$

$yup^2 = R^2/32^2 \times (64N-N^2)$

Nachdem auf beiden Seiten die Quadratwurzel gezogen wurde, haben wir:

$yup = R/32*SQR(64N-N^2)$

Wenn wir mit x = xcentre – R anfangen und x in 64 gleichen Schritten inkrementieren, dann liefert uns die Gleichung bei jedem Schritt die Länge der Strecke vom Durchmesser bis zur Kreislinie, wobei N die Anzahl der Schritte angibt. Beachten Sie, daß dieses Ergebnis unabhängig von den Koordinaten des Kreismittelpunktes oder dem Radius ist. Wir können daher eine Tabelle zusammenstellen, die für jeden N-Wert von 0 bis 64 die Lösung der Wurzelfunktion enthält. Da diese Tabelle nur einmal erstellt werden muß, können wir sie fest in das Programm einbauen.

Die absolute Y-Koordinate für jedes Inkrement von X ist:

ya = ycentre – yup

Aufgrund der Symmetrie des Kreises können wir die Y-Koordinaten für die untere Kreishälfte folgendermaßen berechnen:

yb = ycentre + yup

Das Flußdiagramm zeigt den Ablauf bei der Berechnung der Kreispunkte. Da für jeden Punkt nur eine Multiplikation ausgeführt wird,

ist die Routine relativ schnell. Dabei ergeben sich jedoch zwei Probleme. Zunächst entsteht kein kontinuierlicher Kreis, sondern nur eine Anzahl Punkte auf der Kreislinie. Weiterhin erzeugt diese Methode im Maschinencode einige Ungenauigkeiten, die unter BASIC nicht auftreten.

Das erste Problem läßt sich lösen, indem man die Punkte über Linesub mit kurzen Linien verbindet und so eine fortlaufende Kreislinie erzeugt. Das zweite Problem entsteht durch Ungenauigkeiten der Quadratwurzeltabelle. Wenn wir die in BASIC erzeugten Werte in die dafür reservierten Adreßbytes POKEn, werden nur die Ganzzahlen gespeichert. Wir müssen daher die Genauigkeit der Tabelle erhöhen, indem wir jede Zahl mit Acht multiplizieren. Da 32 die höchste Zahl der Tabelle ist, lassen sich die multiplizierten Werte immer noch in einem einzigen Byte unterbringen. Um die Routine einfach zu halten, haben wir allerdings nur Näherungswerte eingesetzt. Die Maschinencoderoutine teilt den Tabellenwert mit drei LSR-Befehlen (logical shifts right) durch Acht. Wichtig ist dabei, daß wir den Teilungsrest für unsere weiteren Berechnungen zur Verfügung haben.

Das Circsub-Programm

Der Quellentext reserviert am Anfang 65 Bytes für die Tabelle. Dabei wird die erste Byte-Adresse als Label angegeben, und alle folgenden Tabelleneinträge werden über die indizierte Adressierung angesprochen. Bei der Eingabe und Assemblierung des Quellentextes ist nichts Besonderes zu beachten. Bevor der Code gespeichert wird, sollte jedoch das folgende BASIC-Programm eingegeben und mit RUN aufgerufen werden. Der Maschinencode wird dadurch nicht beeinflußt, da er sich oben im Speicher befindet. Das BASIC-Programm erzeugt die 65 Tabellenwerte für unser Circsub-Programm, multipliziert die Werte mit Acht und POKEt die Ergebnisse in den dafür vorgesehenen Speicherbereich. Nach Generierung der Tabelle kann der Maschinencode mit SAVE gespeichert werden. Vergewissern Sie sich aber, daß dabei die Tabelle in den Objektcode mit eingeschlossen wird (die Tabelle beginnt bei der Adresse $C500). Bei jedem Aufruf von Circsub wird die Tabelle dann automatisch geladen.

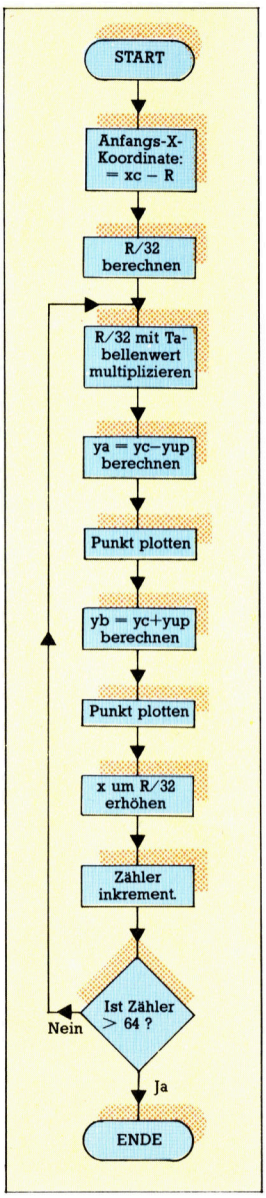
```
5 REM**** CREATE CIRCSUB TABLE ****
10 FORN=0TO64
20 X=SQR(64*N-N↑2)*8
25 X%=X:IF=X-X%
27 IFRE>=.5THENX%=X%+1
28 IFX%>255THENX%=255
29 POKE50432+N,X%
30 NEXT
```

Das folgende Programm zeigt, wie Circsub von einem BASIC-Programm aus aufgerufen wird. Dabei müssen die Koordinaten des Kreismittelpunktes und der Radius angegeben werden. Die Unterroutine in Zeile 2000 wandelt die X-Koordinate in das Format LO-Byte/HI-Byte um, übergibt diese Werte mit POKE an Circsub und ruft die Routine mit SYS auf. Beachten Sie, daß Circsub die Linesub-Routine einsetzt und diese wiederum Plotsub. Alle drei Unterroutinen müssen daher am Anfang des Programms geladen werden. Das BASIC-Programm zeichnet Kreise mit sich ständig vergrößernden Radien auf den Bildschirm.

```
3 REM**** CIRCSUB TEST PROGRAM ****
5 DN=8:REM FOR CASSETTE DN=1
10 IFA=0THENA=1:LOAD"PLOTSUB.HEX",DN,1
15 IFA=1THENA=2:LOAD"LINESUB.HEX",DN,1
20 IFA=2THENA=3:LOAD"CIRCSUB.HEX",DN,1
100 GOSUB1000
110 YC=100:R=2
120 FORXC=30TO210STEP7
130 R=R+3
140 GOSUB2000
150 NEXT
160 GETA$:IFA$=""THEN160
170 GOSUB3000
180 END
1000 REM **** SET HIRES ****
1010 POKE49408,1:POKE49409,1
1020 POKE49410,3
1030 SYS49422
1040 RETURN
1050 :
2000 REM **** ENTER CIRCSUB ****
2010 CHI=INT(XC/256):CLO=XC-256*CHI
2020 POKE50497,CLO:POKE50498,CHI
2030 POKE50499,YC
2040 POKE50500,R
2050 SYS50521
2060 RETURN
2070 :
3000 REM **** CLEAR HIRES ****
3005 RESTORE
3007 PRINTCHR$(147):REM CLEAR SCREEN
3008 PRINTTAB(10)"CIRCSUB VARIABLES"
3009 PRINT
3010 POKE49408,0:SYS49422
3030 FORI=50497TO50497+23STEP2
3035 READA$
3040 PRINTTAB(2),A$,PEEK(I),
3045 READA$
3047 PRINTA$,PEEK(I+1)
3050 NEXT
3060 RETURN
3070 :
5000 DATAXCTRLO,XCTRHI,YCTR,RADIUS,INTR
5010 DATAREMR,TOTX,RESREM,RESLO,RESHI,OLDXLO
5020 DATAOLDXHI,NEWXLO,NEWXHI,OLDYA
5030 DATAOLDYB,NEWYA,NEWYB,XFLAG,YFLAG,OLDY,NEWY
5040 DATAINTTAB,REMTAB
```

Wenn Sie keinen Assembler haben, können Sie den Maschinencode – wie auch die anderen hochauflösenden Grafikroutinen für den Commodore 64 – in Form von Data-Befehlen eingeben. In diesem Fall sollte der Maschinencode mit dem abgebildeten BASIC-Ladeprogramm in den Arbeitsspeicher geladen werden, sowie Linesub und Plotsub mit ihren jeweiligen Programmen. Die Ladeprogramme müssen vor dem Aufruf des BASIC-Beispielprogramms einmal abgelaufen sein. Da alle drei Routinen sich nun bereits im Speicher befinden, können die Zeilen 10, 15 und 20 des Beispielprogramms gelöscht werden. Beachten Sie weiterhin, daß das BASIC-Ladeprogramm für Circsub die Quadratwurzeltabelle bereits enthält und das Generierungsprogramm somit nicht aufgerufen werden muß.

In den nächsten Folgen werden wir uns mit dem 6809-Prozessor beschäftigen, der auch bei den Dragon-Computern verwendet wird.



BASIC-Ladeprogramm

```
10 REM**** BASIC LOADER FOR CIRCSUB ****
20 FORI=50432TO50432+498
30 READA:POKEI,A
40 CC=CC+A
50 NEXT I
60 READA:IFA<>CCTHENPRINT"CHECKSUM ERROR"
100 DATA0,63,89,108,123,137,149,159
110 DATA169,177,185,193,199,205,211
120 DATA216,221,226,230,233,237,240
130 DATA243,245,247,249,251,252,253
140 DATA254,255,255,255,255,255,254
150 DATA253,252,251,249,247,245,243
160 DATA240,237,233,230,226,221,216
170 DATA211,205,199,193,185,177,169
180 DATA159,149,137,123,108,89,63,0
190 DATA205,0,100,80,2,16,16,0,0,0,29
200 DATA1,31,1,100,100,100,100,0,0,120
210 DATA100,0,0,72,138,72,152,72,173
220 DATA68,197,41,31,141,70,197,173,68
230 DATA197,160,5,74,136,208,252,141
240 DATA69,197,173,65,197,56,237,68
250 DATA197,141,77,197,141,75,197,173
260 DATA66,197,233,0,141,78,197,141,76
270 DATA197,173,67,197,141,79,197,141
280 DATA80,197,169,0,170,141,71,197
290 DATA189,0,197,160,3,74,136,208,252
300 DATA141,87,197,189,0,197,41,7,141
310 DATA88,197,172,68,197,169,0,141,72
320 DATA197,141,73,197,141,74,197,173
330 DATA72,197,24,109,88,197,141,72
340 DATA197,201,8,144,23,56,233,8,141
350 DATA72,197,173,73,197,24,105,1,141
360 DATA73,197,173,74,197,105,0,141,74
370 DATA197,173,73,197,24,109,87,197
380 DATA141,73,197,173,74,197,105,0
390 DATA141,74,197,136,208,198,160,5
400 DATA78,74,197,110,73,197,110,72
410 DATA197,136,208,244,173,72,197,201
420 DATA16,144,9,173,73,197,24,105,1
430 DATA141,73,197,173,67,197,56,237
440 DATA73,197,141,81,197,173,81,197
450 DATA141,86,197,173,79,197,141,85
460 DATA197,32,165,198,173,67,197,24
470 DATA109,73,197,141,82,197,173,82
480 DATA197,141,86,197,173,80,197,141
490 DATA85,197,32,165,198,173,77,197
500 DATA141,75,197,173,78,197,141,76
510 DATA197,173,81,197,141,79,197,173
520 DATA82,197,141,80,197,173,71,197
530 DATA24,109,70,197,141,71,197,201
540 DATA32,144,26,173,71,197,56,233,32
550 DATA141,71,197,173,77,197,24,105,1
560 DATA141,77,197,173,76,197,105,0
570 DATA141,78,197,173,77,197,24,109
580 DATA69,197,141,77,197,173,78,197
590 DATA105,0,141,78,197,232,224,65
600 DATA240,3,76,153,197,104,168,104
610 DATA170,104,96,169,0,141,83,197
620 DATA141,84,197,173,75,197,205,77
630 DATA197,208,3,238,83,197,173,85
640 DATA197,205,86,197,208,3,238,84
650 DATA197,173,83,197,45,84,197,208
660 DATA39,173,75,197,141,0,195,173,76
670 DATA197,141,1,195,173,85,197,141,4
680 DATA195,173,77,197,141,2,195,173
690 DATA78,197,141,3,195,173,86,197
700 DATA141,5,195,32,14,195,96
710 DATA67223:REM*CHECKSUM*
```

Circsub-Programm

```
;++++++++++++++++++
;++++++++++++++++++
;++                ++
;++  CIRCSUB 64    ++
;++                ++
;++++++++++++++++++
;++++++++++++++++++
;++++ LINESUB VARIABLES ++++
;
LINSUB = $C30E
X1LO   = $C300
X1HI   = $C301
X2LO   = $C302
X2HI   = $C303
Y1     = $C304
Y2     = $C305
*      = $C500
;
;++++ CIRCSUB VARIABLES ++++
;
TABLE  *=*+65          ; LOOK UP TABLE
XCTRLO *=*+1
XCTRHI *=*+1
YCTR   *=*+1
RADIUS *=*+1
INTR   *=*+1
REMR   *=*+1
TOTX   *=*+1
RESREM *=*+1
RESLO  *=*+1
RESHI  *=*+1
OLDXLO *=*+1
OLDXHI *=*+1
NEWXLO *=*+1
NEWXHI *=*+1
OLDYA  *=*+1
OLDYB  *=*+1
NEWYA  *=*+1
NEWYB  *=*+1
XFLAG  *=*+1
YFLAG  *=*+1
OLDY   *=*+1
NEWY   *=*+1
INTTAB *=*+1
REMTAB *=*+1
;
;++++ PUSH REGISTERS ONTO STACK ++++
;
       PHA
       TXA
       PHA
       TYA
       PHA
;
;++++ CALC INT(R/32) AND REMR ++++
;
AD 44 C5       LDA RADIUS
29 1F          AND #$1F
8D 46 C5       STA REMR
AD 44 C5       LDA RADIUS
A0 05          LDY #$05
4A       LOOP1 LSR A
88             DEY
D0 FC          BNE LOOP1
8D 45 C5       STA INTR
;
;++++ CALC INITIAL X COORDINATE++++
;
AD 41 C5       LDA XCTRLO
38             SEC
ED 44 C5       SBC RADIUS
8D 4D C5       STA NEWXLO
8D 4B C5       STA OLDXLO
AD 42 C5       LDA XCTRHI
E9 00          SBC #$00
8D 4E C5       STA NEWXHI
8D 4C C5       STA OLDXHI
;
;++++ SET INITIAL OLD Y VALUES ++++
;
AD 43 C5       LDA YCTR
8D 4F C5       STA OLDYA
8D 50 C5       STA OLDYB
;
;++++ ZERO COUNT AND TOTX ++++
;
A9 00          LDA #$00
AA             TAX
8D 47 C5       STA TOTX
;
;++++ CALC INTTAB AND REMTAB ++++
;
NEXTPT
BD 00 C5       LDA TABLE,X
A0 03          LDY #$03
LOOP2
4A             LSR A
88             DEY
D0 FC          BNE LOOP2
8D 57 C5       STA INTTAB
;
BD 00 C5       LDA TABLE,X
29 07          AND #$07
8D 58 C5       STA REMTAB
;
;++++ MULTIPLY TABLE BY RADIUS ++++
;
AC 44 C5       LDY RADIUS
A9 00          LDA #$00
8D 48 C5       STA RESREM
8D 49 C5       STA RESLO       ;ZERO ANSWER
8D 4A C5       STA RESHI
;
AD 48 C5 AGAIN LDA RESREM
18             CLC
6D 58 C5       ADC REMTAB
8D 48 C5       STA RESREM
C9 08          CMP #$08        ;>= 8?
90 17          BCC NOCARR
38             SEC
E9 08          SBC #$08        ;RESET REM
8D 48 C5       STA RESREM
AD 49 C5       LDA RESLO
18             CLC
69 01          ADC #$01        ;INC RESULT
8D 49 C5       STA RESLO
AD 4A C5       LDA RESHI
69 00          ADC #$00
8D 4A C5       STA RESHI
NOCARR
AD 49 C5       LDA RESLO
18             CLC
6D 57 C5       ADC INTTAB      ADD INTEGER PART
8D 49 C5       STA RESLO
AD 4A C5       LDA RESHI
69 00          ADC #$00
8D 4A C5       STA RESHI
88             DEY
D0 C6          BNE AGAIN
;
;++++ DIVIDE RESULT BY 32 ++++
;
A0 05          LDY #$05
LOOP3
4E 4A C5       LSR RESHI
6E 49 C5       ROR RESLO
6E 48 C5       ROR RESREM
88             DEY
D0 F4          BNE LOOP3
;
AD 48 C5       LDA RESREM
C9 10          CMP #$10        ; REM >=16?
90 09          BCC NOCRRY
AD 49 C5       LDA RESLO
18             CLC
69 01          ADC #$01        ; ADD 1
8D 49 C5       STA RESLO
;
;++++ CALC 1ST Y COORDINATE ++++
;
NOCRRY
AD 43 C5       LDA YCTR
38             SEC
ED 49 C5       SBC RESLO
8D 51 C5       STA NEWYA
;
;++++ DRAW LINE ++++
;
AD 51 C5       LDA NEWYA
8D 56 C5       STA NEWY
AD 4F C5       LDA OLDYA
8D 55 C5       STA OLDY
20 A5 C6       JSR DRAW
;
;++++ CALC 2ND Y COORDINATE ++++
;
AD 43 C5       LDA YCTR
18             CLC
6D 49 C5       ADC RESLO
8D 52 C5       STA NEWYB
;
;++++ DRAW LINE ++++
;
AD 52 C5       LDA NEWYB
8D 56 C5       STA NEWY
AD 50 C5       LDA OLDYB
8D 55 C5       STA OLDY
20 A5 C6       JSR DRAW
;
;++++ SWOP OLD FOR NEW ++++
;
AD 4D C5       LDA NEWXLO
8D 4B C5       STA OLDXLO
AD 4E C5       LDA NEWXHI
8D 4C C5       STA OLDXHI
AD 51 C5       LDA NEWYA
8D 4F C5       STA OLDYA
AD 52 C5       LDA NEWYB
8D 50 C5       STA OLDYB
;
;++++ ALTER X COORDINATE ++++
;
AD 47 C5       LDA TOTX
18             CLC
6D 46 C5       ADC REMR
8D 47 C5       STA TOTX        ;ADD REMR TO TOTAL
C9 20          CMP #$20        ;>=32?
90 1A          BCC NOINC
AD 47 C5       LDA TOTX
38             SEC
E9 20          SBC #$20        ;RESET TOTX
8D 47 C5       STA TOTX
AD 4D C5       LDA NEWXLO
18             CLC
69 01          ADC #$01          INC X COORD
8D 4D C5       STA NEWXLO
AD 4C C5       LDA OLDXHI
69 00          ADC #$00
8D 4E C5       STA NEWXHI
NOINC
AD 4D C5       LDA NEWXLO
18             CLC
6D 45 C5       ADC INTR
8D 4D C5       STA NEWXLO
AD 4E C5       LDA NEWXHI
69 00          ADC #$00
8D 4E C5       STA NEWXHI
;
;++++ INCREMENT COUNTER ++++
;
E8             INX
E0 41          CPX #$41
F0 03          BEQ FINISH
4C 99 C5       JMP NEXTPT
;
;++++ PULL REGISTERS OFF STACK ++++
;
FINISH
       PLA
       TAY
       PLA
       TAX
       PLA
       RTS
;
;++++ CHECK FOR SAME POINT ++++
;
DRAW
A9 00          LDA #$00
8D 53 C5       STA XFLAG       ;ZERO FLAGS
8D 54 C5       STA YFLAG
;
AD 4B C5       LDA OLDXLO
CD 4D C5       CMP NEWXLO
D0 03          BNE NOXFLG
NOXFLG
AD 55 C5       LDA OLDY
CD 56 C5       CMP NEWY
D0 03          BNE NOYFLG
EE 54 C5       INC YFLAG
NOYFLG
AD 53 C5       LDA XFLAG
2D 54 C5       AND YFLAG
D0 27          BNE NODRAW
;
;++++ DRAW LINE ++++
;
AD 4B C5       LDA OLDXLO
8D 00 C3       STA X1LO
AD 4C C5       LDA OLDXHI
8D 01 C3       STA X1HI
AD 55 C5       LDA OLDY
8D 04 C3       STA Y1
AD 4D C5       LDA NEWXLO
8D 02 C3       STA X2LO
AD 4E C5       LDA NEWXHI
8D 03 C3       STA X2HI
AD 56 C5       LDA NEWY
8D 05 C3       STA Y2
;
20 0E C3       JSR LINSUB
;
NODRAW
60             RTS
```

Erklärende Zeilen

Zur Dokumentation eines Programms gehört weit mehr als das Hinzufügen von Kommentaren und Anweisungen für den Benutzer. Hier wird gezeigt, wie ein einfaches Programm in BASIC und PASCAL mit einer passenden Dokumentation ausgestattet wird.

Sehen wir uns die erste Version des Programms (Listing 1) einmal an. Es ist rätselhaft, da sein Zweck nur vermutet werden kann. Abgesehen von der Aussage, daß „zwei Zahlen eingegeben werden, diese mit zwei anderen Zahlen multipliziert, die beiden Resultate addiert und das Ergebnis ausgedruckt wird" gibt es keinen Hinweis auf das, was überhaupt geschieht. Wenden wir uns Listing 2 zu. In dieser Version ist alles klar. Aber es fehlen noch Anmerkungen, es gibt keinen Programmtitel und keine erklärenden REM-Zeilen.

Eine intensive Beschäftigung mit den Unterschieden der beiden Versionen ist lohnend. Zunächst einmal sind die verwirrenden Zahlen des ersten Listings durch Namen ersetzt worden (AYEAR und AMONTH). Zahlen, deren Werte sich beim Programmablauf nicht verändern, nennt man „Konstante". In manchen Sprachen wie PASCAL gibt es für Konstante eine spezielle Schreibweise. Die Definition von Konstanten ist nur dann sinnvoll, wenn sie regelmäßig verwendet werden. Ansonsten erfüllt ein entsprechender Hinweis im Programm denselben Zweck.

Der zweite wesentliche Unterschied ist, daß die verwirrenden Variablen-Namen durch längere verständliche Namen ersetzt wurden. Die hier gezeigten wurden deshalb gewählt, weil sie kürzer als zehn Zeichen sind und die beiden ersten Zeichen sich voneinander unterscheiden.

Es ist grundsätzlich empfehlenswert, die Variablen mit solchen Namen zu versehen, die einen Bezug zu ihrer Funktion haben. So sollte ein Loop-Zähler (Schleifenzähler) LOOP (statt des üblichen J oder L) benannt werden, und die ersten und letzten Werte des Zählers könnten in Konstante oder Variable mit entsprechenden Namen gelegt werden. Folglich könnte die Schleife

FOR J = 1 TO 10. . .NEXT J

so aussehen:

FOR LOOP=FIRST TO TENTH. . .NEXT LOOP

Lange Variablennamen erfordern natürlich mehr Zeit für die Eingabe und mehr Speicherplatz. Sie machen jedoch Programme leichter verständlich und erleichtern die Fehlersuche. Werden in der von Ihnen verwendeten Sprache nur die ersten beiden Zeichen der Namen zu deren Unterscheidung verwendet, vergewissern Sie sich, daß sich diese Zeichen voneinander unterscheiden.

Ein weiterer wesentlicher Unterschied der beiden Listings besteht darin, daß im zweiten lange und sinnvolle Prompts für die Eingabe verwendet wurden sowie verständliche Erläuterungen der Ausgabe.

PASCAL-Benutzer kennen die Vorteile klar strukturierter Programme. Ganz einfache Maßnahmen – so zum Beispiel das Einfügen von Leerzeilen oder die Verwendung gemischter Groß- und Kleinschreibung – verwandeln eine undurchdringliche Anhäufung von Zeichen in übersichtliche und lesbare Logikblöcke. Das Formatieren eines Programms auf dem Bildschirm oder für den Drucker ist erst dann sinnvoll, wenn darin Schleifen (FOR. . .NEXT, WHILE. . . .WEND, REPEAT. . . .UNTIL) enthalten sind, und besonders dann, wenn die Schleifen verschachtelt sind.

Übersichtlich formatieren

Bedauerlich ist in diesem Zusammenhang, daß die meisten BASIC-Versionen nur sehr wenige Möglichkeiten für eine übersichtliche Programmdarstellung bieten. Unter diesem Gesichtspunkt sind kompilierte Sprachen wie PASCAL weitaus flexibler, da sie normalerweise mit einem Texteditor geschrieben werden. Das Editieren eines BASIC-Programms ist meist komplizierter (es sei denn – wie bei Microsofts MBASIC – daß der Interpreter die ASCII-Version des Programms annimmt und sie in ein lauffähiges Programm umwandelt). Schlimmer aber ist, daß einige BASIC-Dialekte die geschriebenen Programme umformatieren und Leerzeilen entfernen! Andere hingegen fügen Leerzeilen ein. Die meisten PASCAL-Dialekte beinhalten diese Formatierfunktion, die prinzipiell sehr nützlich ist. Es ist dennoch empfehlenswert, selbst Formatierfunktionen zu entwickeln.

Anmerkungen oder Kommentare werden natürlich hauptsächlich zur Dokumentation eines Programms im Programm selbst verwendet. Die Art der Aufbereitung ist von Sprache zu Sprache unterschiedlich. Im BASIC wird ein REM-Statement verwendet. Das Wort REM muß vor der Anmerkung stehen. Der Interpreter wird alles ignorieren, was darauf folgt – bis zum Zeilenende bzw. bis zum Beginn des nächsten Befehls. In anderen Sprachen (PASCAL, PL/1,PROLOG, usw.) werden Anmerkungen durch /* und */ gekennzeichnet. Der

Gut dokumentiert

Listing 1

BASIC

```
(a) 10 INPUT A,B
    20 C=A*31536000
    30 D=B*2592000
    40 E=C+D
    50 PRINT E
```

PASCAL

```
(b) program abcde (input,output);
    var a,b,c,d,e:integer;
    begin
    read(a,b);
    c:=a*31536000;
    d:=b*2592000;
    e:=c+d;
    writeln(e);
    end.
```

Listing 2

BASIC

```
(a) 10 AYEAR=31536000
    20 AMONTH=2592000
    30 PRINT"Enter your age (years then months separated by a comma) ";
    40 INPUT NYEARS,NMONTHS
    50 YSECS=NYEARS*AYEAR
    60 MSECS=NMONTHS*AMONTH
    70 AGEINSECS=YSECS+MSECS
    80 PRINT"Your age in seconds is (approximately) ";AGEINSECS
```

PASCAL

```
(b) program ageinseconds (input,output);
    const
      ayear=31536000;
      amonth=2592000;
    var
      nyears,nmonths,ysecs,msecs,ageinsecs:integer;
    begin
      write('Enter your age (years then months separated by a comma) ');
      read(nyears,nmonths);
      ysecs:=nyears*ayear;
      msecs:=nmonths*amonth;
      ageinsecs:=ysecs+msecs;
      writeln('Your age in seconds is (approximately) ',ageinsecs);
    end.
```

Listing 3

BASIC

```
(a) 10 REM "AGEINSECONDS"   June 1984
    20 REM INPUTs age in years and months (y,m) and
    30 REM uses an approximate conversion (month = 30 days)
    40 REM to give age in seconds.
    50 REM
    60 AYEAR=31536000:REM          seconds in 365 days
    70 AMONTH=2592000:REM          seconds in 30 days
    80 PRINT"Enter your age (years then months separated by a comma) ";
    90 INPUT NYEARS,NMONTHS
   100 REM age in secs is (age in years * secs in year) plus
         (months since last birthday * secs in month)
   110 YSECS=NYEARS*AYEAR
   120 MSECS=NMONTHS*AMONTH
   130 AGEINSECS=YSECS+MSECS
   140 PRINT"Your age in seconds is (approximately) ";AGEINSECS
```

PASCAL

```
(b) program ageinseconds (input,output);
    /*   June 1984
         reads age in years and months (y,m) and uses an
         approximate conversion (month = 30 days) to give
         age in seconds.                                  */

    const
      ayear=31536000;      /* seconds in 365 days */
      amonth=2592000;      /* seconds in 30 days  */
    var
      nyears,nmonths,ysecs,msecs,ageinsecs:integer;
    begin
      write('Enter your age(years then months separated by a comma) ');
      read(nyears,nmonths);
      /* age in secs is (age in years * secs in year) plus
         (months since last birthday * secs in month)     */
      ysecs:=nyears*ayear;
      msecs:=nmonths*amonth;
      ageinsecs:=ysecs+msecs;
      writeln('Your age in seconds is (approximately) ',ageinsecs);
    end.
```

Compiler ignoriert alles, was zwischen den Zeichen steht. Vorteil dieses Systems ist, daß Anmerkungen länger als eine Zeile sein können. Nachteil aber ist, daß der Rest des Programms als Anmerkung betrachtet und ignoriert wird, wenn man das zweite */ vergißt.

Anmerkungen sollten immer dann, wenn man es für nötig erachtet, eingefügt werden: ob für die Definition von Konstanten, das Initialisieren von Variablen, den Programmbeginn, Beginn einer neuen Prozedur (Subroutine), die Definition einer Funktion oder das Schreiben eines Codes, der wegen seines Umfangs schwer verständlich ist. Anmerkungen müssen nicht lang oder gar wortreich sein. Oft reicht eine „Erinnerungs"-REM-Zeile. Versucht man die Logik einiger Abenteuerspiele zu verstehen, wird deutlich, daß große Blocks allgemeiner Kommentare die Codierung unterbrechen und aufgrund zu geringer Details eher hinderlich sind. Anmerkungen sollten also kurz und präzise sein. Sie sollten vor komplizierten Codierungsteilen eingefügt werden und nur dann in die Daten selbst, wenn sie das Lesen der logischen Struktur des Programms nicht stören.

Schwerster und langwierigster Teil der Arbeit ist das Erstellen von Dokumentationen in Form von Handbüchern und anderen geschriebenen Spezifikationen. Untersuchungen bei Programmierern haben ergeben, daß geschriebene Dokumentationen nur als letzte Möglichkeit zu Rate gezogen werden. Benutzt man sie aber, können sie viel Arbeit sparen. Ist Ihr Programm nicht sehr lang und intern gut dokumentiert, erübrigt sich eine externe Programm-Dokumentation. Die Dokumentation für den Anwender ist eine ganz andere Sache. Darauf gehen wir an anderer Stelle dieser Serie noch ein. Dennoch ist es oft sinnvoll, geschriebene Dokumentationen zur Hand zu haben, beispielsweise dann, wenn ein altes Programm überarbeitet werden soll. Eine Möglichkeit, mit der die Sprachen der „fünften Generation" entwickelt werden, ist die automatische Dokumenations-Generierung. Das wird durch Verwendung von Informationen aus der Gestaltungsphase bei der Programmentwicklung erreicht.

Beim Schreiben Ihres Programms sollten Sie sich Notizen machen, und diese in einem File ablegen. Dieses sollte alle Details zum Programm enthalten, einschließlich der Algorithmen und Flußdiagramme. Am wichtigsten ist die Aufbewahrung der letzten Version des Flußdiagramms, das dem eigentlichen Programm zugrunde liegt. Besitzen Sie einen Drucker, sollten Sie ein Listing des fertigen Programms erstellen. Beachten Sie, daß in unserer vervollständigten Programmversion in der ersten Anmerkung Programmname und Entstehungsdatum enthalten sind. Immer dann, wenn Sie ein Programm ändern, sollten Sie auch das Datum ändern, damit Sie auch wissen, welches die neueste Version ist.

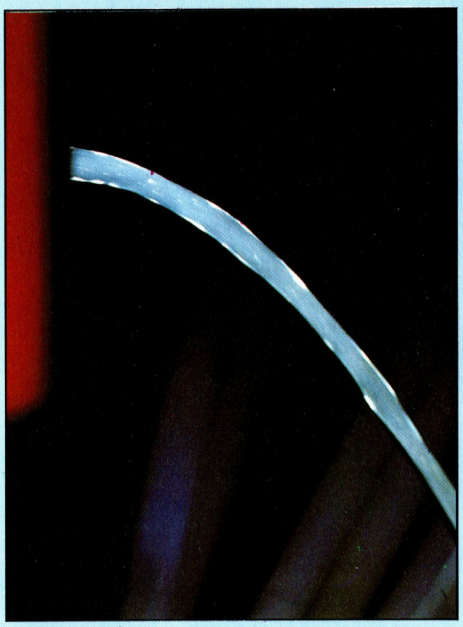
Fachwörter von A bis Z

Hier werden einzelne Fachausdrücke eingehend behandelt. Da bei der Kommunikation mit dem Computer meist die englische Sprache verwendet wird, werden hier zunächst die englischen Begriffe genannt, dann die deutsche Übersetzung. In den Gesamtindex werden sowohl deutsche als auch englische Stichwörter aufgenommen, damit Sie es leichter haben, das von Ihnen Gesuchte zu finden.

Feedback = Rückkopplung
Unter Rückkopplung versteht man die Einflußnahme auf ein Steuerungsorgan durch Information über den eingetretenen Zustand – der Wirkungsablauf wird dadurch zum „Regelkreis" geschlossen. Ein einfaches Beispiel ist die thermostatische Regelung eines Warmwasserspeichers. Der Thermostat erfaßt die Wassertemperatur und schaltet die Heizung ein, wenn eine vorgegebene Mindesttemperatur unterschritten wird, und schaltet sie wieder aus, wenn eine bestimmte Höchsttemperatur erreicht ist. Bei digitaler Regelung mit Microprozessoren wird dem Rechner das Rüchkopplungssignal des Meßfühlers über einen Analog/Digital-Wandler zugeführt.

Fibonacci Sequence = Fibonacci-Reihe
Im 13. Jahrhundert entwickelte der Mathematiker Leonardo Fibonacci die Fibonacci-Reihe, eine unendliche Zahlenfolge, in der jede Zahl die Summe der beiden vorangehenden Zahlen darstellt. Die Sequenz beginnt bei 0,1 und setzt sich fort:
1 (die Summe von 0 und 1),
2,3,5,8,13 . . .
Die formale mathematische Definition der Fibonacci-Reihe lautet:
$F_0=0$, $F_1=1$, $F_{n+2}=F_{n+1}+F_n$, $n\geq 0$

Fibre Optics = Glasfaseroptik
Optische Fasern werden aus extrem dünnem Glas oder Plastik hergestellt und lassen sich anstelle von Kupferkabeln zur Übermittlung von Sprache oder Daten über große Distanzen einsetzen. Eine einzige Glasfaser kann mehrere tausend Signale gleichzeitig übertragen. Glasfaserkabel arbeiten nach dem Prinzip der internen Spiegelung: Das Licht bleibt in der Faser, da die äußeren Oberflächen es wieder in das Innere reflektieren. Die Übertragung funktioniert auch bei gedrehten oder stark gekrümmten Fasern. Ein einfacher Versuch mit einem Wasserbehälter macht das Prinzip deutlich. Von einer Lichtquelle im Inneren des Behälters dringt normalerweise kein Licht nach außen. Kann das Wasser



Demonstration des Glasfaserprinzips

jedoch durch ein Loch abfließen, dann folgt das Licht dem gebeugten Wasserstrahl auf die gleiche Weise wie in der Glasfaser.

Die Informationsmenge, die eine Faser übertragen kann, und die Qualität des Signals hängen von der optischen Dichte des Glases ab. Da optische Kabel keine Elektrizität leiten, können sie gefahrlos in Umgebungen eingesetzt werden, in denen normale Stromkabel ein Sicherheitsrisiko darstellen.

Field = Feld
Felder stellen Informationen dar, die mit einer gemeinsamen Bezeichnung angesprochen werden. In einem Telefonverzeichnis ist normalerweise der Nachname das erste Feld, gefolgt von Vornamen, Adresse und Telefonnummer. Eine Ansammlung von Feldern wird als Datensatz bezeichnet. Die einzelnen darin enthaltenen Daten werden Einträge genannt. Oft dürfen Einträge nur eine bestimmte Anzahl Zeichen enthalten, da ihre Länge in der Felddefinition festgelegt ist.

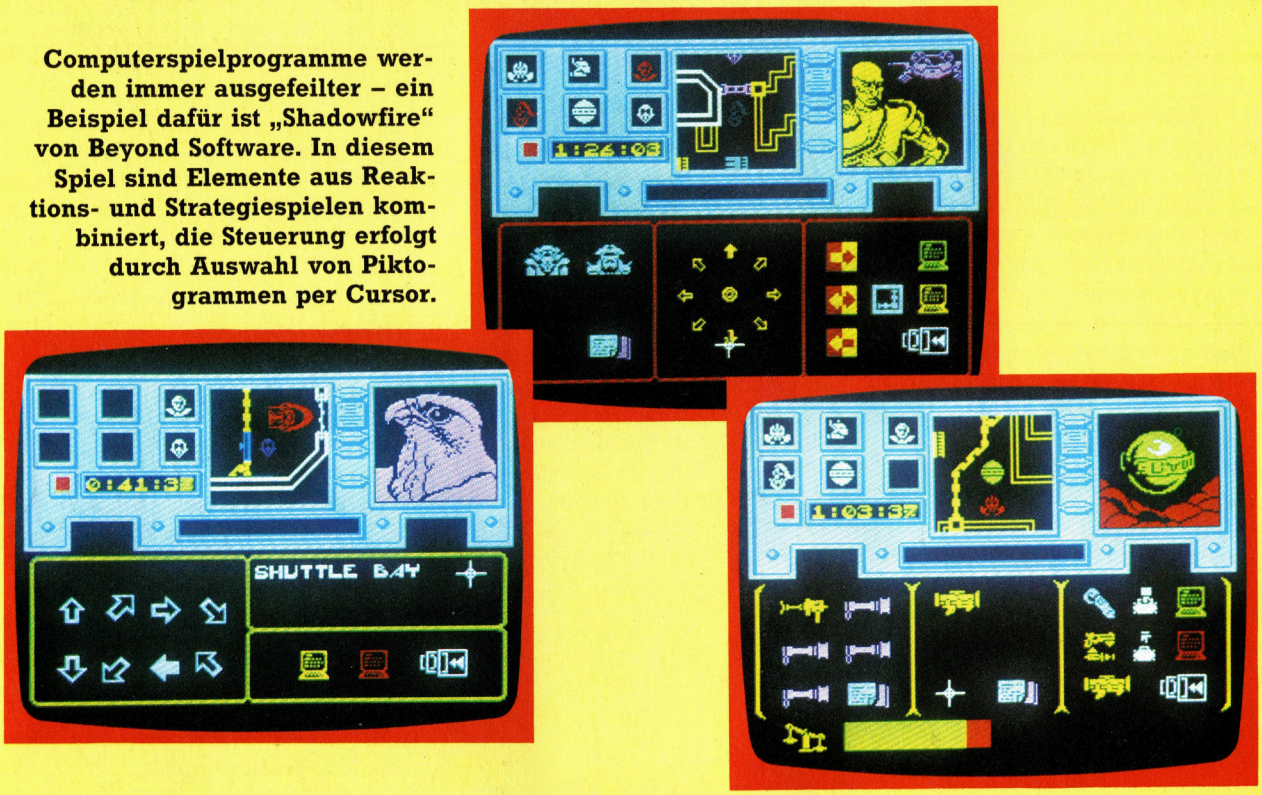
FIFO = FIFO
FIFO ist die Abkürzung für „First In First Out" und beschreibt eine Methode, die Daten eines Stacks zu bearbeiten. Dabei wird das erste auf dem Stack abgelegte Element auch wieder als erstes heruntergezogen. Stacks können auch in umgekehrter Reihenfolge bearbeitet werden. Dieser Fall heißt LIFO „Last In First Out". Eine FIFO-Liste wird auch Queue List (Warteschlange) oder Pushup-Stack genannt.

File = Datei
Eine Datei enthält gleichartige Informationen, die nach ihrer Erstellung gespeichert, geändert und wiederverwendet werden können. Computerdateien sind normalerweise auf Cassette oder Diskette gespeichert. Sie können vollständige Programme, Programmteile (oft eingesetzte Unterroutinen, die als Modul gespeichert und in anderen Programmen wiederverwendet werden), Daten (die von Programmen geladen werden), Texte (alle Arten von Schriftstücken) oder grafische Daten für visuelle Darstellungen enthalten.

Datenbanken fassen gleichartige Informationen normalerweise in einer Datei zusammen, die als Einheit bearbeitet wird. So würde beispielsweise die Datenbank einer Firma alle Personaldaten in einer separaten „Personaldatei" ablegen.

Bildnachweise
1121, 1123: Ian McKinnell
1124, 1125, 1138, 1139, 1144: Liz Dixon
1126, 1129: Kevin Jones
1128: Crispin Thomas
1133, 1134, 1135: Chris Stevens
1137: Lotus Development Corp.
1141: Liz Heaney
1142, 1143: Michael Brownlow
U3: Nick Daly

Computerspielprogramme werden immer ausgefeilter – ein Beispiel dafür ist „Shadowfire" von Beyond Software. In diesem Spiel sind Elemente aus Reaktions- und Strategiespielen kombiniert, die Steuerung erfolgt durch Auswahl von Piktogrammen per Cursor.



computer kurs

Heft 42

Gedanken-Steuerung

Was wie Science-Fiction anmutet, ist bereits machbar: Computer-Steuerung durch Gedanken. Das „Mindlink"-Verfahren ist bereits im Versuchs-Stadium.

Robuster Rechner

Der Colour Genie von Eaca ist eine robuste, für den Heimgebrauch konstruierte Maschine. Das Gerät hat zahlreiche integrierte Schnittstellen.

Flipflop-Schalter

In diesem Kursabschnitt werden drei Schaltungen zur Zeitsteuerung vorgestellt.

Im Labyrinth

Unser Selbstbau-Kurs beschäftigt sich mit einem Programm, das den kürzesten Weg durch ein Labyrinth findet.

Auf ins Abenteuer

Adventure-Games selbst gemacht! Im BASIC-Kurs diesmal alle dazu notwendigen Informationen und Tips.