Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Heft 54

Ein wöchentliches Sammelwerk

Routine-Prüfung

Eigene Datenbanken

Der IBM PC/AT

Ermittlungen per Computer

Programmierkurs FORTH

computer kurs

Heft 54

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

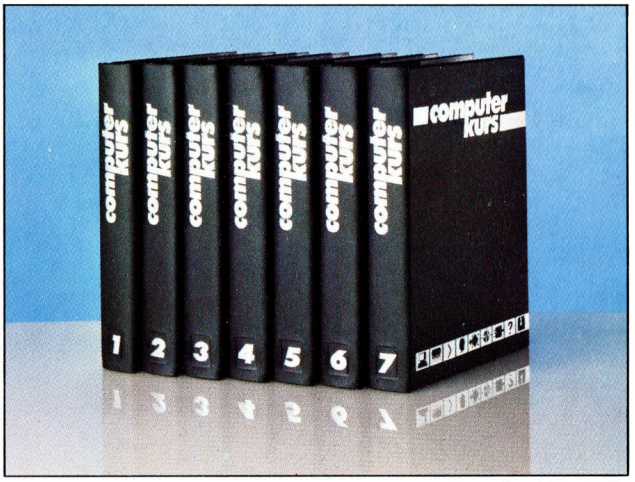
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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

Unter Tatverdacht

Einer der Hauptgründe für die Popularität von Abenteuerprogrammen ist die Interaktion zwischen dem Spieler und den Darstellern des Spielprogramms. Wir zeigen die vielfältigen Möglichkeiten am Beispiel von Infocoms „Suspect" auf.

Software, die einem Computer durch Analyse der Eingaben und Erzeugung offensichtlich bedeutungsvoller Antworten „Persönlichkeit" verleiht, erweckt fast immer Interesse und Spannung. Alle nur denkbaren psychologischen Elemente können integriert werden. Programme wie ELIZA beziehen den Anwender trotz ihrer relativen Einfachheit überproportional mit in das Geschehen ein. Aus der Sicht des Programmierers gibt es kaum eine größere Herausforderung und Befriedigung als einen Computer zum „Hören" und „Antworten" zu programmieren, als handele es sich um ein menschliches Wesen.

Ein einfaches Programm im ELIZA-Stil haben wir bereits vorgestellt und, im Rahmen unserer Serie über Künstliche Intelligenz, die Verarbeitungsprinzipien natürlicher Sprache erörtert. In dieser Serie untersuchen wir andere Anwendungsmöglichkeiten – nämlich, das Programmieren „interaktiver Charaktere" in Abenteuer-Spielen. Wir werden sehen, welche Rolle ein solcher Charakter in diesen Spielen einnehmen kann, und dann einen „Programm-Verwalter" programmieren, der sowohl als selbständiges Modul lauffähig ist als auch in Verbindung mit unseren Programmen „Haunted Forest" und „Digitaya" arbeitet.

Abenteuerprogramme haben sich, seit Crowther und Woods „Colossal Cave" in FORTRAN auf einem Großrechner mit rund 170 KByte Speicherkapazität schrieben, ungeheuer weiterentwickelt. Heute akzeptieren Programme wie „The Hitch Hiker's Guide to the Galaxy" nicht nur umfangreiche Eingaben, sondern zeichnen sich durch Charaktere aus, die sich „selbständig bewegen" und durch den Spieler angesprochen oder in einigen Fällen sogar kommandiert werden können.

Interaktive Fiktion

Die Einführung von Charakteren in „interaktive Fiktion", so wird Abenteuer-Software oftmals genannt, gab Programmierern die Möglichkeit, sich vom herkömmlichen Abenteuer-Szenario zu lösen. Anfänglich sollten fast nur Orte erforscht, Gegenstände gesammelt und Puzzles gelöst werden. Interaktive Charaktere brachten neue Möglichkeiten und Situationen, die vom Spieler ohne Hilfe eines anderen, computergesteuerten Charakters nicht gelöst werden können. Zudem, und dies ist weitaus wichtiger, lassen sich Spiele entwickeln, die bei jedem Start neu und anders sind.

Eine ermordete Märchenkönigin, ein Werwolf, ein Vampir und viele andere Charaktere warten auf Sie in Infocoms „Suspect", einem hervorragenden Abenteuerspiel mit interaktiven Akteuren und detaillierter Ortsbeschreibung. Sie sind zu einem Kostümball der „feinen Gesellschaft" eingeladen, geraten in das zuvor beschriebene Szenario und entdecken zu Ihrem Entsetzen, daß Ihre Gastgeberin ausgerechnet mit dem Lasso erdrosselt wurde, das Bestandteil Ihres Cowboy-Kostüms ist.

Es muß noch definiert werden, was unter einem interaktiven Charakter zu verstehen ist. Zumindest über eine der folgenden Eigenschaften sollte er verfügen:

- Er muß sich von einem Ort zum anderen bewegen können.
- Er muß Gegenstände aufnehmen und fallenlassen können, ohne daß eine entsprechende Anweisung des Spielers vorangegangen ist.
- Er muß vom Spieler ansprechbar sein und sinnvoll antworten können (selbst wenn auch nur: „Ich verstehe dich nicht.")
- Er muß den Spieler ansprechen können, ohne zuvor vom Spieler dazu aufgefordert zu sein.
- Er muß seine Umgebung wahrnehmen und auf sie reagieren können.

Das vielleicht beste Beispiel für interaktive Charakter-Programmierung ist „The Hobbit",

das Text/Grafik-Abenteuer von Melbourne House. In diesem Programm erscheinen viele Charaktere, und herausragende „Persönlichkeiten" sind Thorin, der Zwerg und der Zauberer Gandalf. Der Spieler kann diese Figuren durch Verwendung des SAY-TO-Befehls ansprechen. Etwa indem er SAY TO THORIN 'GO NORTH' eingibt. Das Programm greift dann auf eine Zufalls-Routine zurück, die entscheidet, ob der Charakter gehorcht oder nicht.

Was am bedeutendsten erscheint: Die Charaktere aus „The Hobbit" haben keine persönliche Geschichte, kein Erinnerungsvermögen. Vollzieht man eine Handlung, die Einfluß auf den Bezug zu ihnen haben könnte, „erinnern" sich die Charaktere nicht daran. Typisch für frühe Abenteuerprogramme ist, daß jeder Charakter auf eine bestimmte Verhaltensweise programmiert wurde, die er bis zum Spielende beibehielt, egal, was geschah.

Einer der ersten Versuche, Charaktere mit Erinnerungsvermögen auszustatten, wurde in „Valhalla" realisiert. In diesem Programm, das die Götter und Helden der nordischen Mythologie zum Inhalt hat, sind die „guten" Charaktere nur dann bereit, den Anweisungen des Spielers zu folgen, wenn man die bösen Charaktere tötet.

Damit bot sich unter anderem die interessante Möglichkeit der „Hirnwäsche" von Charakteren. So ist beispielsweise der nordische Gott Loki eigentlich ein bösartiges Wesen. Man kann sich aber mit ihm anfreunden, indem man beispielsweise mit Thor (einem

„Suspect" spielt im exklusiven Landhaus der Ashcrofts, das hier dargestellt wird. Einige der größeren Räume werden im Spiel aus verschiedenem Blickwinkel dargestellt – so zum Beispiel der Tanzsaal in Kapellennähe und der Tanzsaal in Kaminnähe sowie die nördliche große Halle. Die Karte zeigt die Position einiger Haupt-„Darsteller" kurz nach Ermordung von Veronica Ashford:

1 Harlekin, sieht fern
2 Märchenkönigin (tot)
3 Gorilla (der Butler Smythe)
4 Scheich, Gatte der Märchenkönigin
5 Harems-Dame, tanzt
6 Astronaut, geht nach Westen
7 Barmann
8 Vampir, verläßt den Tanzsaal
9 Der Spieler, als Cowboy kostümiert

guten Gott) kämpft, und dann Loki befiehlt, gute Taten auszuführen, wie etwa gegen seine (bösen) Weggefährten zu kämpfen.

Nun zu einem erheblich fortgeschritteneren interaktiven Abenteuerspiel – „Suspect" von Infocom.

Unter Tatverdacht

Suspect ist ein perfektes Beispiel für ein Programm, in dem die Charaktere dominieren. Und tatsächlich hängen Erfolg oder Nichterfolg im Spiel vom Beobachten, Befragen, Untersuchen und sogar dem Anklagen anderer, computergesteuerter „Leute" ab. Aufgrund des Zufallselements kann das Spiel auf verschiedene Arten gespielt werden.

Ort des Geschehens ist ein großes Landhaus (siehe dazu die Karte). Mindestens zwölf Hauptdarteller wirken mit. Das Spiel umfaßt einen Zeitraum von neun Uhr abends bis in die frühen Morgenstunden, in dem die Ashcrofts – ein hochangesehener Millionärsclan – ihren jährlichen Kostümball feiern. Während des Spielverlaufs entdeckt man bei einem Blick vors Haus, daß Veronica Ashcroft mit einem Lasso erdrosselt wurde, das Bestandteil Ihres Cowboy-Kostüms war. Der Verdacht (Suspect) fällt natürlich sofort auf Sie. Wenn es Ihnen nicht gelingt, den wahren Schuldigen zu finden, landen Sie im Gefängnis.

Während des gesamten Spiels laufen die Darsteller umher, reden miteinander und führen ein richtiggehendes „Leben". Das Programm hält den Spieler über ihren jeweiligen Aufenthaltsort auf dem Laufenden. Unsere Karte zeigt die Position einiger Akteure während eines bestimmten Spielstadiums. Sie befinden sich in der Nähe der Kapelle im Tanzsaal und erhalten folgende Mitteilung über den Bildschirm (im Programm wird diese natürlich in Englisch ausgegeben):

Tanzsaal, nahe der Kapelle

Dies ist die nördliche Tanzsaalbegrenzung. Eine Erhöhung stellt das Podium für die Kapelle dar. Sonst befindet sich dort eine Stereoanlage. In anderen Teilen des Saales halten sich andere Ballgäste auf, die mit allen nur vorstellbaren exotischen Kostümen bekleidet sind. Auf dem Tanzparkett bewegen sich ältere Paare. Die Kapelle spielt den „Tennessie Waltz".

Rundum sind Leute in kleinen Gruppen zu sehen, die über politische Ereignisse und lokale Sensationen sprechen. Michael befindet sich an der nördlichen Eingangstür. Alicia ist auf der Tanzfläche. Ostmann hält sich nahe des südlichen Eingangs auf. Der Astronaut steht in Kaminnähe. Johnson sitzt an der Bar. Der Astronaut hat inzwischen den südlichen Eingang erreicht.

Ihnen wird auffallen, daß der Astronaut in der Beschreibung zweimal erwähnt wurde – beim ersten Mal in Kaminnähe und anschließend in Nähe des südlichen Eingangs. Daraus können Sie schlußfolgern, daß er den Ballsaal verlassen will. Die Aufenthaltsorte anderer Charaktere, wie der des Butlers Smythe, bleiben unerwähnt. Sie halten sich in anderen Räumen auf, und Sie müßten diese aufsuchen, um ihre Anwesenheit wahrnehmen zu können.

Das Spielen eines Programms wie „Suspect" ist eine aufregende Erfahrung. Interaktive Charaktere bringen völlig neue Dimensionen in einen Spielbereich, der ohnehin schon außerordentlich populär ist. In den nächsten Folgen werden wir uns mit weiteren Beispielen von Interaktionen befassen und aufzeigen, wie man eigene Routinen schreibt.

. . . und so FORTH

FORTH ist eine Programmiersprache, die ursprünglich zur Steuerung eines Radio-Teleskops entwickelt wurde. Unsere Einführung betrachtet zwei wesentliche Eigenschaften dieser Sprache: Interaktivität und Erweiterbarkeit.

FORTH wurde Ende der sechziger Jahre von Charles Moore speziell zur Steuerung wissenschaftlicher Instrumente entwickelt, die eine sehr genaue Justierung benötigen. Die Leistungsfähigkeit dieser Sprache geht weit über reine Steuerungsanwendungen hinaus. Zur Einführung in die grundlegenden Eigenschaften von FORTH ist es jedoch sinnvoll, sich einmal zu überlegen, wie beispielsweise ein Teleskop durch FORTH gesteuert würde.

Es wäre doch ganz nett, wenn wir in der Lage wären, am Computerterminal in der Sternwarte einfach einzutippen:

30 GRAD ELEVATION

und die Einstellung des Teleskops würde sich sofort genau um diesen Wert ändern. Es wäre darüber hinaus sinnvoll, häufig verwendete Befehlssequenzen zusammenfassen und abspeichern zu können, um sich unnötige Tipparbeit zu ersparen. Wenn zum Beispiel die Einstellung des Teleskops 90 Grad Elevation und 0 Grad Azimut betragen würde, könnte man einen Befehl mit dem Namen „PARK" definieren, nämlich als

0 GRAD AZIMUT 90 GRAD ELEVATION

Der Jupiter Ace war ein mutiger Versuch, eine Maschine auf den Markt zu bringen, die standardmäßig mit FORTH ausgestattet war. Leider konnte sich der Jupiter Ace wegen seiner fehlenden Farbgrafik, seines zu kleinen Arbeitsspeichers und des zu geringen Bekanntheitsgrads von FORTH nicht auf dem Heimcomputermarkt durchsetzen.

Zusammen mit der hervorragenden Originalliteratur bietet er jedoch einen ausgezeichneten Einstieg in die Sprache.

Insgesamt benötigen wir also

- elementare Teleskop-Befehle,
- eine Möglichkeit, diese Befehle zu komplexeren Befehlen (zu einem Teleskop-Programm) zusammenfassen zu können.

Aus einem solchen System würde am Ende eine eigene, komplette Teleskop-Programmiersprache entstehen, die über besondere Eigenschaften verfügt.

Stellen Sie sich zunächst einmal vor, Sie müßten diese Aufgabe in BASIC lösen. Zwar lassen sich hier die fest eingebauten Befehle wie PRINT oder RUN direkt über die Tastatur eingeben, problematisch wird es aber mit den neu definierten Befehlen.

ELEVATION und AZIMUT

Nehmen wir an, sie haben Ihre Befehlserweiterungen ELEVATION und AZIMUT als Unterprogramme in den Zeilen 1000 bzw. 1100 abgelegt. Der Befehl PARK würde so aussehen:

```
1200 REM PARK
1205 LET GRAD = 0
1210 GOSUB 1000
1215 LET GRAD = 90
1220 GOSUB 1100
1225 RETURN
```

Eine andere Möglichkeit wäre, zunächst eine entsprechende Variable zu definieren:

```
LET PARK = 1200
```

und dann die Befehlssequenz aufzurufen, indem Sie die Variable im GOSUB angeben:

```
GOSUB PARK
```

Am einfachsten ist es jedoch, wenn Sie den Unterprogrammteil als eine Prozedur definieren (dies ist zum Beispiel im Acorn-BASIC oder in zahlreichen BASIC-Erweiterungen möglich). Der Aufruf würde dann etwa in der Form

```
PROCPARK
```

erfolgen. Eine Alternative wäre es, einen Interpreter zu schreiben, der die Befehle entgegennimmt und ausführt, wie beispielsweise

```
100 INPUT C$
110 IF C$ = "PARK" THEN GOSUB 1200 :
    GOTO 100
```

Allerdings ist es nun nicht mehr möglich, die eingebauten BASIC-Befehle direkt über die Tastatur einzugeben.

Aus dem Vorangegangenen war zu ersehen, daß wir zwei Forderungen an unsere Teleskop-Sprache stellen müssen:

● Erstens muß sie interaktiv sein. Das heißt, sie muß in der Lage sein, die Befehle direkt auszuführen, nachdem sie eingetippt wurden. Diese Eigenschaft besitzen Sprachen wie BASIC, LOGO, APL und PROLOG, nicht aber Sprachen wie PASCAL, ALGOL, FORTRAN oder COBOL. Auch die Kommandointerpreter von Betriebssystemen wie CP/M oder UNIX besitzen diese Eigenschaft.

● Zweitens muß sie erweiterbar sein. Das Eingabeformat für eingebaute Befehle muß dasselbe sein wie für selbstdefinierte Befehle. Nachdem wir einige Erweiterungen durchgeführt haben, können Sie so arbeiten, als hätten Sie es mit einer erweiterten, leistungsfähigeren Sprache, und nicht mit der um einige zusätzliche Routinen erweiterte Originalsprache zu tun. Das ist in LOGO oder den erwähnten Betriebssystemen möglich, aber nicht in den meisten anderen Sprachen.

Das Konzept der Erweiterbarkeit ist allerdings noch wesentlich leistungsfähiger. Wenn uns jemand eine Teleskop-Steuerungssprache verkauft und wir sie um einige nützliche Routinen für unser eigenes Teleskop ergänzen, so haben wir die ursprüngliche „allgemein einsetzbare Teleskop-Steuerungssprache" zu einer „persönlichen Teleskop-Steuerungssprache" erweitert.

Soll aber eine Erweiterung in vollem Umfang durchgeführt werden, so sollten wir mit einer „allgemein einsetzbaren erweiterbaren Computersprache" beginnen und diese zu einer „allgemein einsetzbaren erweiterbaren Teleskop-Steuerungssprache" erweitern können. Auf diese Art können nicht nur Teleskope, sondern auch Roboter, wissenschaftliche Experimentiergeräte, Produktionsanlagen, – kurz alles das, was sich an einen Computer anschließen läßt, gesteuert und kontrolliert werden.

FORTH-Dialekte

Von FORTH gibt es im wesentlichen drei Dialekte. Da wäre einmal das FIG (Forth Interest Group) FORTH, FORTH-79 und FORTH-83. Die meisten FORTH-Versionen beziehen sich auf einen dieser drei „Standards".

FORTH-79 und FORTH-83 sind aufeinanderfolgende Standards und definieren nur den grundlegenden Befehlssatz. Jede einzelne FORTH-Version wird sicher einen größeren Befehlssatz zur Verfügung stellen, der spezielle Eigenschaften des Computers (wie etwa Sound oder Grafik) unterstützt. Wenn Sie ein Programm schreiben, das sich strikt an den FORTH-83-Standard hält, so sollte es auf allen FORTH-83-Implementationen laufen. FORTH-83 ist der aktuelle Standard, und wir werden uns daran halten. FORTH-79 bietet zusätzlich eine Reihe von Befehlen, die die Implementierung des jeweiligen FORTH-Systems unterstützen. Viele FORTH-79- und FORTH-83-Implementationen gehen auf FIG-Forth zurück und werden erweitert, um dem jeweiligen Standard zu entsprechen.

Das Teleskop und die Turtle

Es lohnt sich, einen Vergleich zwischen LOGO und FORTH durchzuführen, da sich die beiden Sprachen sehr ähnlich sind. Beide wurden ursprünglich zur Kontrolle und Steuerung eines physikalischen Objekts entwickelt, eines Teleskops bzw. einer Schildkröte.

Beide Sprachen haben zwei wesentliche übereinstimmende Arbeitsprinzipien: Eine auszuführende Routine wird einfach durch ihren Namen aufgerufen, und neue Routinen werden auf bereits existierenden Routinen aufgebaut. Der eigentliche Unterschied liegt eher in den verschiedenen Erwartungen ihrer Benutzer. FORTH-Anwender erwarten eine hohe Arbeitsgeschwindigkeit (deshalb ist FORTH nicht nur interaktiv und erweiterbar, sondern auch sehr effizient) und nehmen dafür einige Unbequemlichkeiten in Kauf. LOGO dagegen ist konzipiert worden, um Kindern das Programmieren beizubringen. Es ist einfach zu erlernen und erlaubt daher keine Unbequemlichkeiten. Die Folge ist, daß LOGO wesentlich langsamer arbeitet. FORTH hat standardmäßig keine „Turtle-Grafik" eingebaut, verfügt aber über Möglichkeiten, den Grafik-Bildschirm zu steuern. Diese elementaren Befehle können Sie dazu verwenden, die LOGO-Befehle „FORWARD, RIGHT" usw. zu definieren. Vergleichen Sie die beiden Möglichkeiten, die Turtle zu steuern:

LOGO	FORTH
FORWARD 100	100 FORWARD

Bei diesem Beispiel könnte man den Eindruck gewinnen, FORTH würde sozusagen rückwärts arbeiten. Parameter in einer FORTH-Routine müssen zuerst berechnet und deshalb vor dem Namen der Routine eingegeben werden. Dieses Prinzip könnte man als „Rezeptbuch-Technik" beschreiben. Zuerst werden die Zutaten in bestimmten Mengen zusammengegeben und dann gegart.

Weitere Übereinstimmungen mit LOGO finden Sie bei der Definition neuer Routinen:

```
TO . . . END          : . . . ;
```

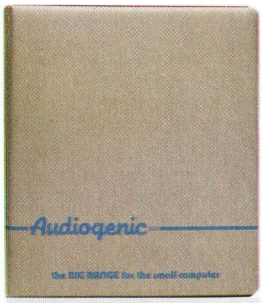
und bei der Schleifenkonstruktion:

```
REPEAT 4 [ . . . . ]      40 DO . . . . LOOP
```

Hier ein Beispiel, wie wir beides zu einer Routine kombinieren können, die ein Quadrat auf dem Bildschirm zeichnet.

```
TO QUADRAT SEITE          : QUADRAT
REPEAT 4 [                  40 DO
  FORWARD SEITE               DUP FORWARD
  RIGHT 90                      RIGHT
  ]                           LOOP
                              DROP
END                       ;
QUADRAT 100                 100 QUADRAT
```

DUP und DROP sind „Stackmanipulationen", die den Routinen ermöglichen, auf die Parameter der QUADRAT-Routine zuzugreifen. DUP kopiert die Zahl für die FORWARD-Routine, DROP löscht die Zahl.



Es existiert mittlerweile für fast jeden Heimcomputer eine FORTH-Implementation. FORTH lebt wie wohl kaum eine andere Computersprache von ihren Anwendern. Die Verbreitung von FORTH und die Unterstützung seiner Anwender ist auch das Ziel der als Verein arbeitenden FORTH Interest Group. In Deutschland hat diese Aufgabe die FORTH-Gesellschaft übernommen: FORTH GESELLSCHAFT DEUTSCHLAND e. V., Schanzenstr. 27, 2000 Hamburg 6.

Lückenschließer

Im letzten Teil des Selbstbau-Kurses haben wir ein aufsteckbares User Port-Interface für den Sinclair Spectrum geplant. Dieser Abschnitt zeigt den Zusammenbau des Interface, mit dem sich unser selbstgebauter Roboter auch vom Spectrum ansteuern läßt.

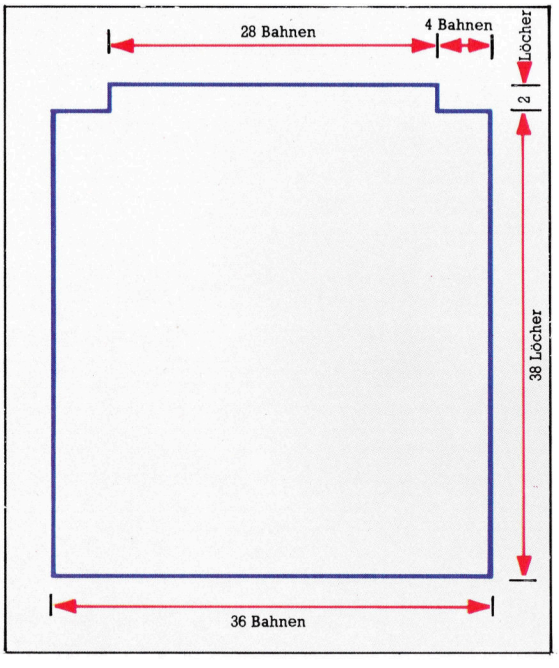
Der Platinenstecker für den Spectrum hat an Position fünf – von vorn gesehen – einen Blindkontakt. Wir müssen also unseren Verbindungsstecker ebenfalls mit einem Blindkontakt versehen. Er sorgt dafür, daß sich das Interface nicht verkehrtherum aufstecken läßt. Einen passenden Stecker werden Sie im Handel kaum finden, man kann ihn aber ohne große Schwierigkeiten selbst herstellen.

Nach dem Zuschnitt der Platine können Sie eine der abgeschnittenen Ecken für den Blindkontakt verwenden: Das Kupfer der Leiterbahnen wird abgekratzt und das Platinenstück hochkant bei Anschluß fünf in den Steckverbinder eingeschoben, wo Sie es mit Superkleber befestigen. Die beiden dazugehörigen Pins am Stecker schneiden Sie einfach ab. Danach werden die Leiterbahnen auf der Platine unterbrochen, entweder mit einem Spezialwerkzeug für diesen Zweck (Elektronik-Fachhandel) oder mit einem 5-mm-Spiralbohrer. Achten Sie darauf, daß die Leiterbahnen wirklich vollständig durchtrennt sind. Andernfalls würde nicht nur das neue Interface seinen Dienst versagen, auch der Spectrum selbst könnte zu Schaden kommen.

Nun geht es ans Bestücken der Platine. Wir fangen mit den Drahtbrücken an, die etwas länger als nötig zugeschnitten und erst einmal lose in die Platine eingesetzt werden. Eine Seite löten Sie fest, dann kann das Drahtende auf der anderen Seite straffgezogen und ebenfalls verlötet werden. Überstehende Enden entfernen Sie mit dem Seitenschneider. Die isolierte Drahtbrücke wird zuletzt eingesetzt, Sie können eine Ader des Flachkabels dafür verwenden.

Liste der Bauteile

Anzahl	Bauteil
1	IC 74LS27
1	IC 74LS75
1	IC 74LS126
4	Widerstände 15 kOhm, 0,3 Watt
1	Rolle verzinnte Litze
2	IC-Sockel, 14-polig
1	IC-Sockel, 16-polig
5 Meter	Flachbandkabel, 20-polig
1	Platinenstecker, 2 x 28 Kontakte, 2,54-mm-Raster
1	D-Stecker, 15-polig
1	Gehäuse für 15-poligen D-Stecker
1	Lochplatine (Veroboard), 36 Bahnen à 50 Löcher

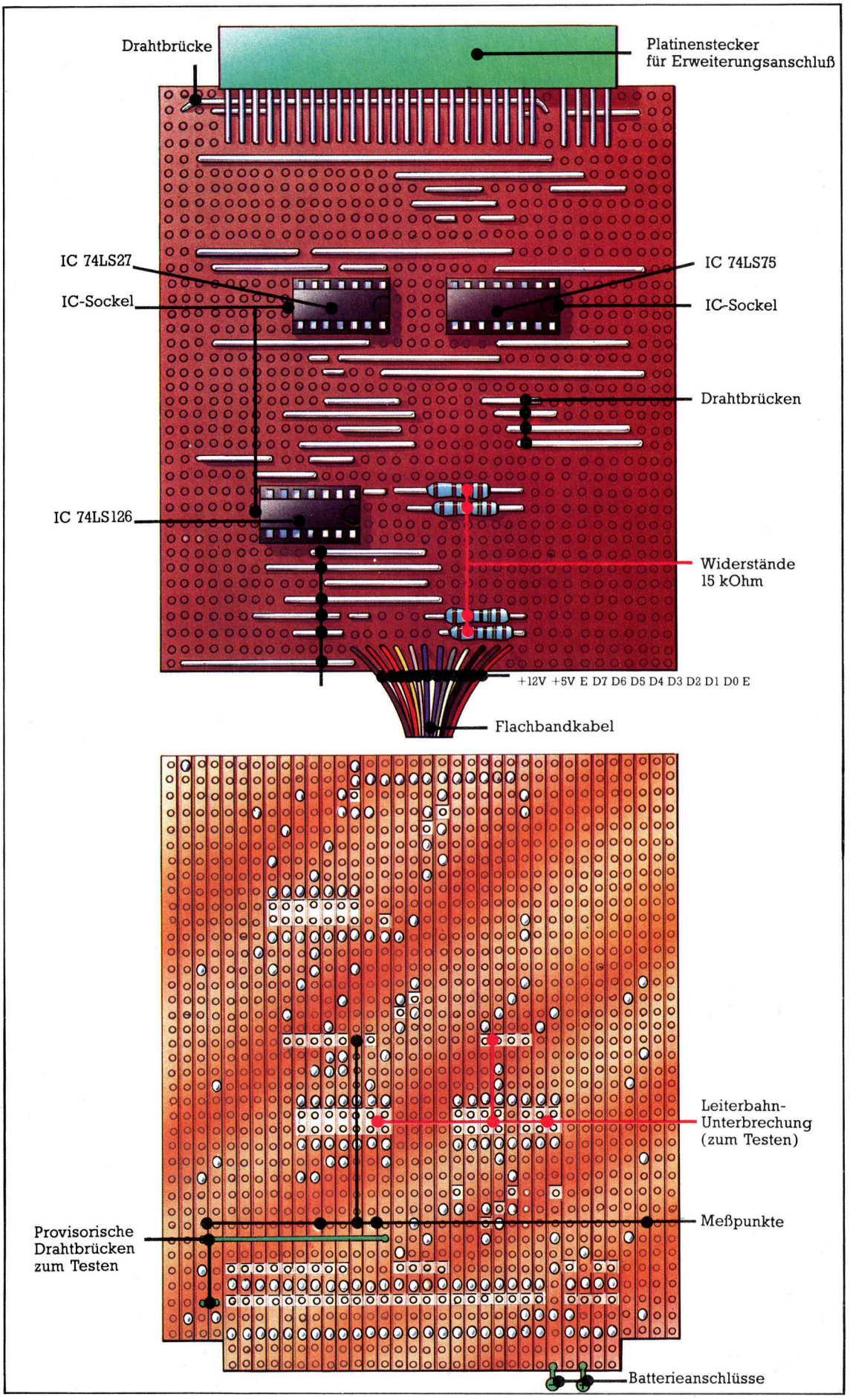


Für das Spectrum-Interface schneiden Sie die Lochplatine nach den nebenstehenden Angaben zu. Eine der abgetrennten Ecken dient als Blindkontakt für den Stecker, der am vorstehenden Teil der Platine festgelötet wird.

Kontaktfreudig

Sind alle Brücken an ihrem Platz, geht es mit der Montage des Platinensteckers weiter. Diese Stecker sind eigentlich für Platinen mit Leiterbahnen auf beiden Seiten vorgesehen – die Sache ist also nicht ganz einfach. Zuerst die untere Reihe der Anschlußpins wie im Bild abbiegen. Die oberen Pins verlängern Sie mit 3-cm-Stücken verzinnter Litze. Die abgebogenen Pins in die dritte Lochreihe von der Platinenkante aus einsetzen. Die Drahtverlängerungen werden nebeneinander durch die in der Zeichnung markierte Lochreihe gezogen. Stecker und Platine verbinden Sie mit etwas Superkleber. Nach dem Aushärten des Klebstoffs die Steckeranschlüsse sorgfältig mit den Leiterbahnen verlöten. Prüfen Sie die Platine genau – überschüssiges Lötzinn muß mit einem Messer weggeschnitten werden.

Nachdem Sie IC-Sockel und Widerstände eingelötet haben, können Sie die ICs einsetzen – aber aufgepaßt: Wenn Sie die Platine mit dem Stecker nach oben halten, müssen alle IC-Markierungskerben rechts liegen!



Platinen-layout

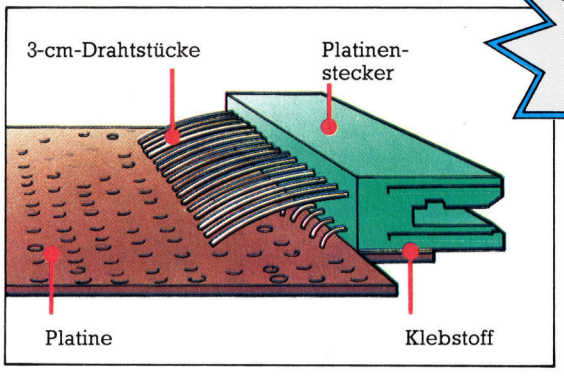
Auf der Bauteilseite werden zuerst die Drahtbrücken und Widerstände angelötet, danach sind der Stecker und die IC-Sockel an der Reihe. Beim Einsetzen der ICs auf korrekte Richtung achten – die Markierungen liegen jeweils rechts. Bevor Sie die Platine am Spectrum-Erweiterungsanschluß aufstecken, sollte sie noch einmal sehr sorgfältig mit dem Schaltungsplan verglichen werden!

Die Unterbrechungen der Leiterbahnen auf der Unterseite prüfen Sie am besten mit einem Meßgerät. Die grün gezeichneten Drahtbrücken dienen zum Testen der Schaltung – wenn alles klappt, müssen sie wieder entfernt werden. Eine gründliche Prüfung der fertigen Platine ist auch diesmal unabdingbar!

Das Interface ist jetzt zwar fertig, vor dem Anschließen müssen Sie es aber noch gründlich prüfen. Am Erweiterungsanschluß des Spectrum liegen neben den Adreß- und Datenleitungen auch Anschlüsse mit 12 und 9 Volt – sie dürfen nicht versehentlich mit den empfindlichen Schaltungen auf TTL-Niveau (0 bis maximal 5 Volt) verbunden werden. Also zuerst die Platine sehr genau inspizieren:

Können Kurzschlüsse durch überschüssiges Lötzinn entstehen? Befinden sich alle Drahtbrücken und Bauteile an der richtigen Stelle? Sind die ICs richtig eingesetzt?

Im nächsten Kursabschnitt gehen wir genauer auf die Verkabelung zwischen Interface und Roboter ein. Außerdem sollen einige der bereits bekannten Roboter-Steuerprogramme für den Spectrum abgewandelt werden.



Damit Sie einen Standard-Platinenstecker verwenden können, muß dieser ein wenig abgewandelt werden. Nach dem Abschneiden der Pins 5 wird die untere Pinreihe rechtwinklig abgebogen. Die oberen Anschlüsse bekommen Verlängerungen aus 3 cm langen Drahtstücken, die durch die entsprechenden Löcher in der sechsten Lochreihe der Platine gesteckt und dort festgelötet werden. Die unteren Pins sitzen in der dritten Lochreihe. Vor dem Anlöten wird der Stecker mit Superkleber auf der Platinenkante befestigt.

Achtung!

Der Erweiterungsanschluß des Spectrum ist direkt mit empfindlichen Bauteilen des Rechners verbunden. Ein Kurzschluß zwischen Daten- oder Adreßbusanschlüssen und den beiden Kontakten für externe Stromversorgung (9 V und 12 V) kann den Computer schwer beschädigen. Darum ALLE Verbindungen vor der ersten Inbetriebnahme GRÜNDLICH prüfen!

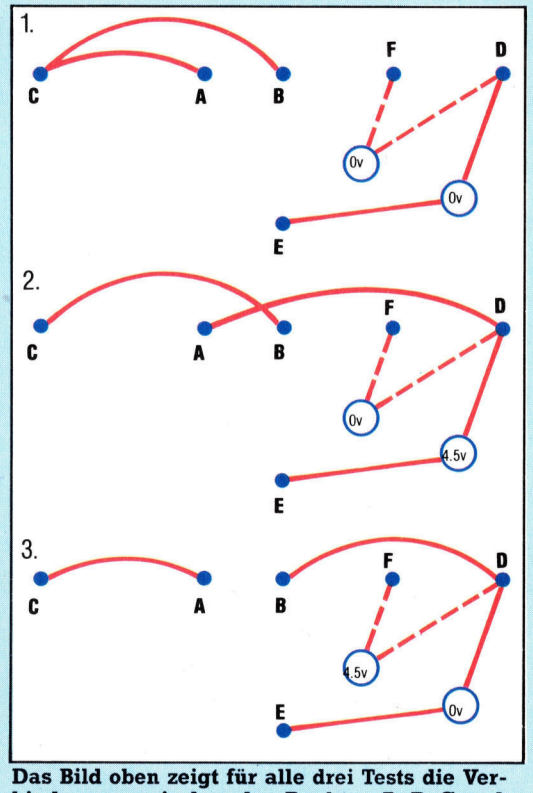
Schaltung testen

Vor dem Anschluß der Interface-Platine am Erweiterungsanschluß des Spectrum können wir mit einigen Tests prüfen, ob die Decodierlogik auch richtig arbeitet. Dazu benötigen wir ein Multimeter, eine 4,5-Volt-Batterie und einige isolierte Drahtstücke. Zuerst mit isolierter Litze die beiden in der Zeichnung grün dargestellten Verbindungen von A nach C und von B nach C herstellen und die Batterie an die Schaltung anschließen. Auch hier: Vorsicht bei überschüssigem Lötzinn zwischen einzelnen Leiterbahnen! Jetzt wird die Spannung an den Punkten E und F gemessen. Der negative (schwarze) Prüfanschluß kommt auf D, mit dem positiven Prüfanschluß wird nur bei E und F gemessen. Beide Spannungen sollten zwischen 0 und 1 Volt liegen, sonst steckt irgendwo ein Fehler.

Löten Sie den Draht zwischen A und C bei C ab, und verbinden Sie das abgelötete Ende mit Punkt D. Der negative Anschluß des Multimeters bleibt bei D. Wenn Sie jetzt erneut E und F nachmessen, sollten die Anzeige bei E 4,5 Volt und bei F immer noch 0 Volt sein.

Die Leitung von A nach D bei D und die Leitung B nach C bei C ablöten und jetzt B mit D und A mit C verbinden. Bei F sollte dann eine Spannung von 4,5 Volt anstehen, E liegt auf 0 Volt. Wenn alle Messungen das richtige Ergebnis hatten, können Sie die Batterieanschlüsse, die beiden Drahtbrücken (grün) und die provisorischen Testverbindungen zwischen A und C sowie B und D entfernen. Stimmte irgendeine Messung nicht mit unseren Angaben überein, sollten Sie die gesamte Platine noch einmal genau überprüfen. Keinesfalls vor einem erfolgreichen zweiten Test die Platine am Spectrum anschließen! Die Testverdrahtung mit den korrekten Meßwerten haben wir im Bild rechts zu einer Tabelle zusammengefaßt.

Meßpunkte	Anzeige
A-C, B-C	E=0v F=0v
A-D, B-C	E=4.5v F=0v
A-C, B-D	E=0v F=4.5v



Das Bild oben zeigt für alle drei Tests die Verbindungen zwischen den Punkten A, B, C und D, und die Tabelle nennt die richtigen Meßwerte an den Punkten E und F.

Action im Rechner

Hier wird das Adventure-Projekt abgeschlossen. Auf den nächsten Seiten finden Sie das vollständige Listing von Digitaya, einem umfangreicheren Spiel als Haunted Forest, das parallel erarbeitet wurde, aber doch einige Parallelen aufweist.

Die Struktur des Digitaya-Programms ist mit der von Haunted Forest vergleichbar, allerdings erheblich umfangreicher. Das Layout umfaßt eine Karte mit 100 Orten, in denen der interne Aufbau eines Computers, einschließlich Speicher, Datenbus, Prozessor und der wichtigsten Hardware, erfaßt wird. Der Spieler muß den mysteriösen Digitaya, der irgendwie im Computer gefangen sitzt, aufspüren. Dabei müssen vielschichtige Gefahren überwunden werden, um Digitaya zu retten und das Spiel erfolgreich zu beenden.

Digitaya verwendet ähnliche Routinen wie Haunted Forest für die wesentlichen Spielfunktionen wie etwa die Bewegung von einem Ort zum anderen, oder das Aufnehmen und Ablegen von Objekten. Zusätzlich wurden jedoch weitere Routinen eingearbeitet, die den Spielreiz durch besondere Schwierigkeiten und Hindernisse erhöhen.

Lösungsweg aufzeichnen

Um ein Adventure zu lösen, sollte man möglichst seinen Weg durch das Spiel mit Papier und Bleistift aufzeichnen. Die erste Aufgabe des Spielers ist, sich in den Programmierer hineinzuversetzen. Dazu ist das Nachvollziehen der Karte der erste Schritt. Digitaya ist auf einem Flächenraster entwickelt worden. Das bedeutet aber nicht, daß alle Adventure-Karten nur zweidimensional angelegt sind. Anhand der Programmstruktur von Digitaya sind auch dreidimensionale Karten leicht zu erstellen. Der Zeitfaktor kann für eine vierte Dimension verwendet werden. Ein derartiges, mit einer ständig wechselnden Karte erarbeitetes Spiel würde selbst eingefleischte Adventure-Spieler zur Verzweiflung bringen.

Das angegebene Listing gilt für den C64, gilt aber auch für die meisten Computer mit Microsoft-BASIC. Für den Acorn B werden besondere Hinweise gegeben.

Einige REM-Anweisungen können aus dem Listing entfernt werden, um Tipparbeit zu sparen. Entfernen Sie aber nur die REMs, die hinter anderen Codes am Zeilenende stehen. Solche Zeilen, die nur REMs enthalten, sind in der Regel Unterroutinen-Titel, die von GOSUBs aufgerufen werden. Ein Entfernen dieser REMs würde einen „UNDEFINED STATEMENT"-Fehler verursachen.

BASIC-Dialekte

Spectrum
Aufgrund der ungewöhnlichen Handhabung von String-Arrays und Variablen können hier nicht alle notwendigen Änderungen aufgeführt werden. Um Ihnen die Arbeit zu erleichtern, werden wir die wichtigsten Punkte nennen. Bitte lesen Sie auch die Spectrum-Hinweise in den vorhergehenden Artikeln.

Der Spectrum erlaubt nur String-Arrays mit fester Länge, wobei die Länge jedes Feldelements über die DIM-Anweisung definiert wird. Dabei können Probleme auftreten, wenn ein String-Array auf 20 Zeichen pro Element definiert wird. Ist dann ein Element nur fünfzehn Zeichen lang, werden die übrigen fünf Zeichen mit Leerstellen gefüllt. Die Leerstellen müssen vor dem Einfügen in einen Satz aus dem Feldelement entfernt werden. Die Variable A$ wird für die Übertragung des Feldelements in eine Unterroutine verwendet und sollte vor dem Aufruf der Unterroutine zugeordnet werden, zum Beispiel:

Microsoft-Version:
```
3650 SN$="YOUR"+IV$(F,1)+"IS
   USELESS,THE FORCE INCREASES"
```

Spectrum-Version:
```
3650 LET S$="YOUR ":A$=IV$(F,1):GOSUB
   8500: LETS$=S$+" IS USELESS,THE
   FORCE INCREASES"
```

Das einzige zusätzliche Problem ist das Löschen des Bildschirms. In der Commodore-Version wird PRINT CHR$(147) verwendet. Ersetzen Sie diese Anweisungen einfach durch CLS.

Acorn B
Die folgenden Zeilen sollten im Digitaya-Listing ausgetauscht werden:
```
1400 A$=GET$
1410 CLS
2630 REPEAT:A$=GET$:UNTIL A$="Y" OR
      A$="N"
2750 RA=RND(1)
2820 P=RN(40)+7
3890 RD=RND(1):IF RD>.65 THEN 4110:REM
      HIT
4090 P=RND(40)+7
4520 IV$(4,2)=STR$(RND(40)+7):REM
      REALLOCATE TICKET POSITION
4570 RN=RND(3)+1
5560 RA=RND(1)
```

Das Digitaya-Listing

```
1030 REM **    'DIGITAYA'    **
1040 REM **    A COMPUTER    **
1050 REM ** ADVENTURE GAME **
1090 :
1100 GOSUB6090:REM READ ARRAY DATA
1110 GOSUB1290:REM STORY SO FAR
1120 P=47:REM START POINT
1130 :
1140 REM **** MAIN LOOP STARTS HERE ****
1150 :
1160 MF=0:PRINT
1170 GOSUB1440:REM DESCRIBE POSITION
1180 GOSUB1560: REM LIST EXITS
1190 GOSUB2670:REM IS P SPECIAL
1200 IF SF=1 THEN 1250:REM NEXT LOOP
1210 PRINT:INPUT"INSTRUCTIONS";IS$
1220 GOSUB1700:REM ANALYSE INSTRUCTIONS
1225 IF F=0 THEN 1210:REM INVALID INSTRUCTION
1230 GOSUB 1900:REM NORMAL INSTRUCTIONS
1240 IF VF=0 THENPRINT"I DON'T UNDERSTAND"
1250 IF MF=1 THEN 1160:REM NEW POSITION
1260 IF MF=0 THEN 1210:REM NEW INSTRUCTION
1270 END
1280 :
1290 REM **** STORY SO FAR ****
1300 SN$="WELCOME TO 'DIGITAYA'"
1310 GOSUB5880:REM FORMAT
1320 PRINT
1330 SN$="AS THE MACHINE HUMS QUIETLY. YOU LOOK AROUND."
1340 SN$=SN$+" TO THE NORTH AND SOUTH STRETCHES A WIDE HIGHWAY."
1350 SN$=SN$+" YOUR MISSION IS TO FIND THE MYSTERIOUS DIGITAYA"
1360 SN$=SN$+" AND CARRY IT TO SAFETY THROUGH ONE OF THE OUTPUT
     PORTS."
1370 SN$=SN$+".. BUT WHICH ONE ?"
1380 GOSUB5880
1390 PRINT:PRINT"PRESS A KEY TO START"
1400 GETA$:IFA$=""THEN1400
1410 PRINTCHR$(147):REM CLEAR SCREEN
1420 RETURN
1430 :
1440 REM **** DESCRIBE POSITION S/R ****
1450 SN$="YOU ARE "+LN$(P):GOSUB5880
1460 SN$="YOU SEE "
1470 REM ** SEARCH FOR OBJECT **
1480 F=0:SP$=""
1490 FOR I=1TO8
1500 IF VAL(IV$(I,2))=P THEN SN$=SN$+SP$+"A "+IV$(I,1):F=1:SP$="  "
1510 NEXTI
1520 IF F=0 THENSN$=SN$+"NO OBJECTS"
1530 GOSUB5880:REM FORMAT
1540 RETURN
1550 :
1560 REM **** LIST EXITS S/R ****
1570 EX$=EX$(P)
1580 NR=VAL(LEFT$(EX$,2))
1590 EA=VAL(MID$(EX$,3,2))
1600 SO=VAL(MID$(EX$,5,2))
1610 WE=VAL(RIGHT$(EX$,2))
1620 IF(NR OR EA OR SO OR WE)=0THEN RETURN
1630 PRINT:SN$="EXITS ARE TO THE "
1640 IF NR<>0 THEN SN$=SN$+"NORTH "
1650 IF EA<>0 THEN SN$=SN$+"EAST "
1660 IF SO<>0 THEN SN$=SN$+"SOUTH "
1670 IF WE<>0 THEN SN$=SN$+"WEST "
1675 GOSUB 5880:REM FORMAT
1680 PRINT:RETURN
1690 :
1700 REM **** ANALYSE INSTRUCTION S/R ****
1705 F=0:REM ZERO FLAG
1710 IFIS$="END" OR IS$="LIST" THEN VB$=IS$:F=1:RETURN
1720 IF IS$="LOOK" THEN VB$=IS$:F=1:RETURN
1730 :
1740 REM ** SPLIT INSTRUCTION **
1750 VB$="":NN$="":REM ZERO VERB AND NOUN
1770 LS=LEN(IS$)
1780 FOR C=1TO LS
1790 A$=MID$(IS$,C,1)
1800 IF A$=" "THEN VB$=LEFT$(IS$,C-1):NN$=RIGHT$(IS$,LS-C):F=1:C=LS
1810 NEXT
1830 IF F=0 THEN PRINT:PRINT"I NEED AT LEAST TWO WORDS"
1840 RETURN
1850 :
1900 REM **** NORMAL ACTIONS S/R ****
1910 VF=0
1920 PRINT
1930 IF VB$="GO"ORVB$="MOVE"THENVF=1:GOSUB2000
1940 IF VB$="TAKE"ORVB$="PICK"THEN VF=1:GOSUB2140
1950 IF VB$="DROP"ORVB$="PUT"THENVF=1:GOSUB2360
1960 IF VB$="LIST"ORVB$="INVENTORY"THENVF=1:GOSUB2540
1965 IF VB$="LOOK" THEN VF=1:MF=1:RETURN
1970 IF VB$="END"ORVB$="FINISH"THENVF=1:GOSUB2610
1980 RETURN
1990 :
2000 REM **** MOVE S/R ****
2010 MF=1:REM MOVE FLAG SET
2015 GOSUB8600:REM SEARCH FOR DIRECTION
2020 DR$= LEFT$(NN$,1)
2030 IFDR$<>"N"ANDDR$<>"E"ANDDR$<>"S"ANDDR$<>"W"THEN2100
2040 IF DR$="N" AND NR<>0 THEN P=NR:RETURN
2050 IF DR$="S" AND SO<>0 THEN P=SO:RETURN
2060 IF DR$="E" AND EA<>0 THEN P=EA:RETURN
2070 IF DR$="W" AND WE<>0 THEN P=WE:RETURN
2080 PRINT"YOU CANT ";IS$
2090 MF=0:RETURN
2100 REM NOUN NOT OK
2110 PRINT"WHAT IS ";NN$;" ?"
2120 MF=0:RETURN
2130 :
2140 REM **** TAKE S/R ****
2145 IV$(4,1)="TICKET TO TRI-STATE"
2150 GOSUB5730:REM IS OBJECT VALID
2160 IF F=0 THEN PRINT"THERE IS NO ";WB$:RETURN
2170 REM ** IS OBJECT ALREADY TAKEN ? ****
2180 OV=F:GOSUB5830
2190 IFHF=1 THEN SN$="YOU ALREADY HAVE THE "+IV$(F,1):GOSUB5880
     :RETURN
2200 :
2210 REM ** IS OBJECT HERE **
2220 IF VAL(IV$(F,2))<>P THENSN$=IV$(F,1)+" IS NOT HERE":GOSUB5880
     :RETURN
2230 :
2240 REM ** ADD OBJECT TO LIST **
2250 AF=0:FOR J=1TO4
2260 IFIC$(J)=""THENIC$(J)=IV$(F,1):AF=1:J=4
2270 NEXTJ
2280 :
2290 REM ** CHECK FOR FULL QUOTA **
2300 IF AF=0THENPRINT"YOU ALREADY HAVE 4 OBJECTS":RETURN
2310 :
2320 SN$="YOU TAKE THE "+IV$(F,1):GOSUB5880
2330 IV$(F,2)="-1":REM DELETE POSITION ENTRY
2340 RETURN
2350 :
2360 REM ** DROP S/R **
2370 GOSUB5730:REM IS OBJECT VALID
2380 IF F=0 THEN PRINT"THERE IS NO ";NN$:RETURN
2390 :
2400 REM ** IS OBJECT HELD ? **
2410 OV=F:GOSUB5830
2420 IFHF=0THENPRINT"YOU DO NOT HAVE THE ";IV$(F,1):RETURN
2430 :
2440 REM ** DROP OBJECT **
2450 SN$="YOU DROP THE "+IV$(F,1):GOSUB5880
2460 IV$(F,2)=STR$(P):REM UPDATE OBJ POSITION
2470 :
2480 REM ** DELETE FROM HELD OBJ LIST **
2490 FORJ=1TO4
2500 IF IC$(J)=IV$(F,1)THENIC$(J)="":J=4
2510 NEXTJ
2520 RETURN
2530 :
2540 REM **** LIST INVENTORY S/R ****
2550 PRINT"OBJECTS HELD:"
2560 FORI=1TO4
2570 PRINT"  ";IC$(I)
2580 NEXTI
2590 RETURN
2600 :
2610 REM **** END GAME S/R ****
2620 PRINT:PRINT"ARE YOU SURE (Y/N) ?"
2630 GETA$:IFA$<>"Y"AND A$<>"N"THEN2630
2640 IFA$="N"THEN RETURN
2650 END
2660 '
2670 REM **** IS P SPECIAL S/R ****
2680 SF=0:REM UNSET SPECIAL FLAG
2690 IF P=37 THEN2780:REM VECTOR TABLE
2700 IF P>7 THEN 2750:REM RANDOM BUG
2710 ON P GOSUB 2850,2960,3450,3830,4180,4550,5150
2720 RETURN
2730 :
2740 REM ** RANDOM BUG **
2750 RA=RND(TI)
2760 IF RA<0.05THEN GOSUB 5420:REM BUG
2770 RETURN
2780 REM ** VECTOR TABLE **
2790 SF=1
2800 SN$="YOU ARE MOVED AT HIGH SPEED TO A NEW LOCATION":GOSUB5880
2810 FORJ=1TO1000:NEXT:REM PAUSE
2820 P=INT(RND(TI)*40+7)
2830 MF=1:RETURN
2840 :
2850 REM **** TV OUTLET S/R ****
2860 SF=1
2870 SN$="YOU HAVE ENTERED THE TV OUTLET AND THERE IS NO ESCAPE."
2880 SN$=SN$+"YOU ARE DOOMED FOREVER TO BE A TV CHAT SHOW HOST"
2890 GOSUB 5880:REM FORMAT PRINT
2900 PRINT
2910 PRINT"WELCOME TO THE SHOW......
2920 FORJ=1TO500:NEXTJ
2930 GOTO 2910
2940 END
2950 :
2960 REM **** USER PORT S/R ****
2970 SF=1
2980 SN$="ESCAPE IS AT HAND BUT THE DDR BOOKING CLERK"
2990 SN$=SN$+" BARS YOUR WAY. HE TELLS YOU THAT HE HAS BEEN
     INSTRUCTED TO"
3000 SN$=SN$+" ACCEPT INPUTS ONLY. HOWEVER HE DOES TAKE ALL MAJOR"
3010 SN$=SN$+" CREDIT CARDS."
3020 GOSUB 5880:REM FORMAT PRINT
3030 :
```

```
3040 PRINT:INPUT"INSTRUCTIONS";IS$
3050 GOSUB1700:REM ANALYSE INSTRUCTIONS
3060 GOSUB1900:REM NORMAL ACTIONS
3070 IF MF=1 THEN RETURN:REM MOVE OUT
3080 IF VF=1 THEN3040:REM NEXT INSTRUCTION
3090 IF VB$<>"GIVE" THENPRINT"I DON'T UNDERSTAND":GOTO3040
3100 :
3110 REM ** INSTRUCTION IS GIVE **
3120 GOSUB5730:REM IS OBJECT VALID
3130 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO3040:REM NEXT INSTRUCTION
3140 :
3150 REM ** IS OBJECT CREDIT CARD **
3160 IF F<>5THENPRINT"HE ONLY ACCEPTS CREDIT CARDS":GOTO3040
3170 :
3180 REM ** IS CARD CARRIED **
3190 OV=5:GOSUB5830
3200 IFHF=0THENPRINT"YOU DO NOT HAVE THE ";IV$(5,1):GOTO 3040
3210 :
3220 SN$="THE CLERK TAKES THE CARD AND SAYS 'THAT WILL DO NICELY,
     SIR'"
3230 GOSUB5880:REM FORMAT PRINT
3240 SN$="YOU ARE ALLOWED TO PASS THE BARRIER AND ENTER THE USER
     PORT"
3250 GOSUB5880:REM FORMAT PRINT
3260 :
3270 REM ** IS DIGITAYA CARRIED **
3280 OV=6:GOSUB5830
3290 IF HF=1 THEN 3380:REM SUCCESS
3300 :
3310 REM ** FAILURE **
3320 SN$="WELL DONE YOU HAVE SUCCEEDED IN ESCAPING FROM THE
     CLUTCHES"
3330 SN$=SN$+" OF THE MACHINE, BUT HAVE FAILED IN YOUR MISSION"
3340 SN$=SN$+" TO BRING BACK THE MYSTERIOUS DIGITAYA"
3350 GOSUB5880:REM FORMAT PRINT
3360 END
3370 :
3380 REM ** SUCCESS **
3390 SN$="CONGRATULATIONS, YOU HAVE SUCCEEDED IN YOUR MISSION"
3400 SN$=SN$+" TO RESCUE THE WONDEROUS DIGITAYA FROM THE"
3410 SN$=SN$+" CLUTCHES OF THE MACHINE.
3420 GOSUB5880:REM FORMAT PRINT
3430 END
3440 :
3450 REM **** CASSETTE PORT S/R ****
3460 SF=1
3470 SN$="YOU FEEL AN IRRESTISTIBLE FORCE PULLING YOU TOWARDS"
3480 SN$=SN$+" PERMANANENT MAGNETIC SUSPENSION"
3490 GOSUB5880:REM FORMAT
3500 NS=0:REM START COUNTING INSTRUCTIONS
3510 REM ** INSTRUCTIONS **
3520 NS=NS+1:IFNS>3THEN3770:REM SUCKED OUT
3530 PRINT:INPUT"INSTRUCTIONS";IS$
3540 GOSUB1700:REM ANALYSE INSTRUCTIONS
3550 GOSUB1900:REM NORMAL ACTIONS
3560 IFMF=1THENMF=0:PRINT"YOU CAN'T MOVE...YET":GOTO3510
3570 IFVF=1THEN3510:REM NEXT INSTRUCTION
3580 IFVB$<>"USE"THENPRINT"I DON'T UNDERSTAND":GOTO3510
3590 REM ** INSTRUCTION IS USE **
3600 GOSUB5730:REM IS OBJECT VALID
3610 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO3510
3620 :
3630 REM ** IS OBJECT BUFFER ACTIVATOR **
3640 IF F=8 THEN3680:REM OK
3650 SN$="YOUR "+IV$(F,1)+" IS USELESS, THE FORCE INCREASES"
3660 GOSUB 5880:GOTO3510:REM NEXT INSTRUCTION
3670 :
3680 OV=8:GOSUB5830:REM IS BUFF ACT HELD
3690 IFHF=0THENSN$="YOU DON'T HAVE THE "+IV$(8,1):GOSUB5880:GOTO3510
3700 :
3710 REM ** SAVED **
3720 SN$="YOU USE THE BUFFER ACTIVATOR TO COUNTER THE PULL"
3730 SN$=SN$+" INTO MAGNETIC OBLIVION. THE FORCE SUBSIDES"
3740 GOSUB5880:REM FORMAT
3750 RETURN
3760 :
3770 REM ** SUCKED OUT **
3780 SN$="THE FORCE BECOMES TOO STRONG AND YOU ARE PULLED OUT"
3790 SN$=SN$+" THROUGH THE CASSETTE PORT INTO MAGNETIC NOTHINGNESS."
3800 GOSUB 5880:REM FORMAT
3810 END
3820 :
3830 REM **** JOYSTICK PORT ****
3840 SF=1
3850 SN$="A USER WITH RED-RIMMED EYES ZAPS HIS LASER AT YOU
     REPEATEDLY."
3860 GOSUB5880:REM FORMAT
3870 :
3880 REM ** INSTRUCTIONS **
3890 RD=RND(TI):IF RD>.65THEN 4110:REM HIT
3900 PRINT:INPUT"INSTRUCTIONS";IS$
3910 GOSUB1700:GOSUB1900:REM ANALYSE INSTRUCTION
3920 IFMF=1THENMF=0:PRINT"YOU CAN'T MOVE...YET":GOTO3880
3930 IFVF=1THEN3880:REM NEXT INSTRUCTION
3940 IFVB$<>"USE"THENPRINT"I DON'T UNDERSTAND":GOTO3880
3950 GOSUB5730:REM IS OBJECT VALID
3960 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO3880:REM NEXT INSTRUCTION
3970 :
3980 REM ** IS OBJECT LASER SHIELD **
3990 IF F=3 THEN4020:REM OK
4000 SN$="YOUR "+IV$(F,1)+" IS NO USE":GOSUB5880:GOTO3880
4010 :
4020 OV=3:GOSUB5830:REM IS LASER SHIELD CARRIED
4030 IFHF=0THENSN$="YOU DO NOT HAVE THE "+IV$(3,1):GOSUB5880
     :GOTO3880
4040 :
4050 REM ** SAVED **
4060 SN$="YOU USE THE LASER SHIELD TO PROTECT YOURSELF. A BLAST
     KNOCKS"
4070 SN$=SN$+" YOU OUT OF THE JOYSTICK PORT AND BACK INTO THE
     MACHINE."
4080 GOSUB5880:REM FORMAT
4090 P=INT(RND(TI)*40+7):MF=1:RETURN
4100 :
4110 REM ** HIT **
4120 SN$="YOU ARE HIT BY THE LASER AND YOU ARE ONLY DIMLY AWARE
     THAT"
4130 SN$=SN$+" YOUR ATOMS HAVE BEEN DISTRIBUTED TO THE FOUR CORNERS"
4140 SN$=SN$+" OF THE UNIVERSE"
4150 GOSUB5880:REM FORMAT
4160 END
4170 :
4180 REM **** TRI-STATE DEVICE S/R ****
4190 SF=1
4200 SN$="A LARGE SIGN SAYS 'I/O THIS WAY' BUT AS YOU MOVE
     TOWARDS IT"
4210 SN$=SN$+" A TICKET COLLECTOR SHOUTS 'TICKETS PLEASE'
4220 GOSUB5880:REM FORMAT
4230 :
4240 REM ** INSTRUCTIONS **
4250 PRINT:INPUT"INSTRUCTIONS";IS$
4260 GOSUB1700:GOSUB1900:REM ANALYSE
4270 IFMF=1 THEN RETURN
4280 IFVF=1THEN4240:REM NEXT INSTRUCTION
4290 IFVB$<>"GIVE"ANDVB$<>"OFFER"THENPRINT"I DON'T UNDERSTAND"
     :GOTO4240
4300 REM ** INSTRUCTION IS GIVE **
4310 GOSUB5730:REM IS OBJECT VALID
4320 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO4240:REM NEXT INSTRUCTION
4330 :
4340 REM ** IS OBJECT TICKET **
4350 IF F=4 THEN4400:REM OK
4360 SN$="THE TICKET COLLECTOR SHAKES HIS HEAD AND SAYS"
4370 SN$=SN$+" 'I CANNOT ACCEPT THIS "+IV$(F,1)
4380 GOSUB5880:GOTO4240:REM NEXT INSTRUCTION
4390 :
4400 OV=4:GOSUB5830:REM IS TICKET HELD
4410 IFHF=0THENPRINT"YOU DO NOT HAVE THE TICKET":GOTO4240
4420 :
4430 REM ** OK **
4440 SN$="THE TICKET COLLECTOR ACCEPTS YOUR TICKET AND ALLOWS YOU"
4450 SN$=SN$+" TO PASS THE BARRIER."
4460 GOSUB5880:REM FORMAT
4470 REM ** DEL TICKET FROM LIST **
4480 F=0
4490 FORJ=1TO4
4500 IF IC$(J)=IV$(4,1)THENIC$(J)=""":J=4
4510 NEXT J
4520 IV$(4,2)=STR$(INT(RND(TI)*40+8)):REM REALLOCATE TICKET
     POSITION
4530 P=15:MF=1:RETURN
4540 :
4550 REM **** ALU ****
4560 SF=1
4570 RN=INT(RND(TI)*3+1)
4580 IF RN=1 THEN CD$="AND"
4590 IF RN=2 THEN CD$="OR"
4600 IF RN=3 THEN CD$="NOT"
4610 SN$="MOUNTED ON THE WALL THERE ARE THREE BUTTONS MARKED"
4620 SN$=SN$+" 'AND', 'OR' AND 'NOT'. ACCESS CAN BE GAINED TO THE"
4630 SN$=SN$+" ACCUMULATOR BY PRESSING THE CORRECT BUTTON"
4640 GOSUB5880:REM FORMAT
4650 :
4660 REM ** INSTRUCTIONS **
4670 PRINT:INPUT"INSTRUCTIONS";IS$
4680 GOSUB1700:GOSUB1900:REM ANALYSE
4690 IF MF=1THEN RETURN:REM MOVE OUT
4700 IF VF=1THEN 4670:REM NEXT INSTRUCTION
4710 IFVB$="USE"OR VB$="PRESS"THEN4740
4720 PRINT"I DON'T UNDERSTAND":GOTO4670
4730 :
4740 REM ** VALID COMMAND **
4750 IF VB$="PRESS"THEN 4930
4760 REM ** COMMAND IS 'USE' **
4770 GOSUB5730:REM IS OBJECT VALID
4780 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO4670:REM NEXT INSTRUCTION
4790 :
4800 REM ** IS OBJECT CODE BOOK **
4810 IF F=7 THEN4850:REM OK
4820 SN$="YOUR "+IV$(F,1)+" IS OF NO USE":GOSUB5880
4830 GOTO4670:REM NEXT INSTRUCTION
4840 :
4850 OV=7:GOSUB5830:REM IS CODE BOOK HELD
4860 IFHF=1THEN4900:REM OK HELD
4870 SN$="YOU DO NOT HAVE THE "+IV$(7,1)
4880 GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4890 :
4900 SN$="YOU OPEN THE CODE BOOK AND FIND THE WORD '"+CD$+"'
     WRITTEN INSIDE"
4910 GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4920 :
4930 REM ** COMMAND IS PRESS **
4940 IFNN$="AND"OR NN$="OR"OR NN$="NOT"THEN4970
4950 SN$="THERE IS NO "+NN$:GOSUB5880:GOTO4670:REM NEXT INSTRUCTION
4960 :
4970 REM ** RIGHT OR WRONG **
```

```
4980 IFNN$=CD$ THEN GOSUB5100:RETURN
4990 GOSUB5010:RETURN
5000 :
5010 REM ** WRONG S/R **
5020 SN$="WRONG, A TRAP DOOR OPENS AND YOU FIND YOURSELF BACK"
5030 SN$=SN$+" BACK IN MAIN MEMORY"
5040 GOSUB5880:REM FORMAT
5050 IF RN=1 THEN P=39
5060 IF RN=2 THEN P=35
5070 IF RN=3 THEN P=29
5080 MF=1:RETURN
5090 :
5100 REM ** RIGHT S/R **
5110 SN$="THE GATEWAY TO THE ACCUMULATOR SWINGS OPEN AND"
5120 SN$=SN$+" YOU PASS THROUGH":GOSUB5880
5130 P=30:MF=1:RETURN
5140 :
5150 REM **** GATEWAY TO MEMORY S/R ****
5160 SF=1
5170 SN$="AN USHER GREETS YOU BUT TELLS YOU THAT YOU CANNOT BE
     ADMITTED"
5180 SN$=SN$+" UNLESS YOU GIVE AN ADDRESS":GOSUB5880
5190 REM ** INSTRUCTIONS **
5200 PRINT:INPUT"INSTRUCTIONS";IS$
5210 GOSUB1700:GOSUB1900:REM ANALYSE
5220 IF MF=1 THEN RETURN:REM MOVE OUT
5230 IF VF=1 THEN 5200:REM NEXT INSTRUCTION
5240 IF VB$<>"GIVE"THENPRINT"I DON'T UNDERSTAND":GOTO 5200
5250 :
5260 GOSUB5730:REM IS OBJECT VALID
5270 IFF=0THENPRINT"THERE IS NO ";NN$:GOTO5200:REM NEXT INSTRUCTION
5280 :
5290 REM ** IS OBJECT ADDRESS **
5300 IF F=1 THEN5330:REM OK
5310 PRINT"HE NEEDS YOUR ADDRESS":GOTO5200
5320 :
5330 OV=1:GOSUB5830:REM IS ADDRESS CARRIED
5340 IF HF=1 THEN 5370
5350 SN$="YOU DON'T HAVE THE "+IV$(1,1):GOSUB5880:GOTO5200
5360 :
5370 REM ** OK PASS THROUGH **
5380 SN$="THE USHER LOOKS AT YOUR ADDRESS AND ALLOWS YOU TO PASS"
5390 SN$=SN$+" THROUGH":GOSUB5880
5400 P=40:MF=1:RETURN
5410 :
5420 REM **** RANDOM BUG ****
5430 SF=1
5440 SN$="A LARGE AND UGLY BUG APPEARS FROM BEHIND A CHIP"
5450 SN$=SN$+" AND LUNGES TOWARDS YOU":GOSUB5880
5460 :
5470 REM ** INSTRUCTIONS **
5480 PRINT:INPUT"INSTRUCTIONS";IS$
5490 GOSUB1700:GOSUB1900:REM ANALYSE
5500 IFMF=1THENMF=0:PRINT"YOU CAN'T MOVE...YET":GOTO5480
5510 IF VF=1THEN5480:REM NEXT INSTRUCTION
5520 IF VB$="KILL"ORVB$="FIGHT"THEN5550
5530 PRINT"I DON'T UNDERSTAND":GOTO5480
5540 :
5550 REM ** COMAND IS FIGHT/KILL **
5560 RA=RND(TI)
5570 IFRA<0.5 THEN GOSUB5600
5580 GOSUB5670:RETURN
5590 :
5600 REM **** KILLED BY S/R ****
5610 SN$="YOU FIGHT WITH THE BUG. IT SHOWERS YOU WITH SPURIOUS"
5620 SN$=SN$+" ERRORS AND THEY EAT INTO YOUR BRAIN."
5630 SN$=SN$+" FINALLY YOU CAN TAKE NO MORE AND YOUR HEAD
     EXPLODES."
5640 GOSUB5880
5650 END
5660 :
5670 REM **** YOU KILL S/R ****
5680 SN$="YOU FIGHT WITH THE BUG AND THOUGH IT IS A HARD STRUGGLE"
5690 SN$=SN$+" YOU EVENTUALLY IRON IT OUT AND SURVIVE.":GOSUB5880
5700 RETURN
5710 :
5720 :
5730 REM **** VALID OJECT S/R ****
5740 NN$=NN$+" ":LN=LEN(NN$):F=0:C=1
5745 FOR K=1 TO LN
5750 IF MID$(NN$,K,1)<>" " THEN NEXTK:RETURN
5755 W$=MID$(NN$,C,K-C):C=K+1:LW=LEN(W$)
5760 FORJ=1 TO 8
5770 LI=LEN(IV$(J,1)):REM LENGTH OBJECT
5780 FORI=1TO LI
5790 IFMID$(IV$(J,I),I,LW)=W$THENF=J:I=LI:J=8:K=LN
5800 NEXT I,J,K
5810 RETURN
5820 :
5830 REM **** IS OBJECT HELD S/R ****
5840 HF=0
5850 IFIV$(OV,2)="-1"THEN HF=1
5860 PETURN
5870 :
5880 REM **** FORMAT PRINTING S/R ****
5890 LC=0:  REM CHAR/LINE COUNTER
5900 OC=1:  REM OLD COUNT
5910 OW$="":REM OLD WORD
5920 LL=40:REM SCREEN LINE LENGTH
5930 SN$=SN$+" DUMMY "
5940 PRINT
5950 FOR C=1 TO LEN(SN$)
5960 LC=LC+1
5970 IF MID$(SN$,C,1)=" " THENGOSUB6020
5980 NEXTC
5990 PRINT
6000 RETURN
6010 :
6020 REM **** END OF LINE CHECK S/R ****
6030 NW$=MID$(SN$,OC,C-OC+1)
6040 IF LC<LL THENPRINTOW$;:GOTO6060
6050 PRINTOW$:LC=LEN(NW$)
6060 OC=C+1:OW$=NW$
6070 RETURN
6080 :
6090 REM **** READ ARRAY DATA S/R ****
6100 REM ** READ INVENTORY **
6110 DIM IV$(8,2),IC$(4)
6120 FOR C =1TO8
6130 READ IV$(C,1),IV$(C,2)
6140 NEXT C
6150 :
6160 REM ** READ LOCATION & EXIT DATA **
6170 DIM LN$(55),EX$(55)
6180 C1=0:C2=0:REM INITIALISE CHECKSUMS
6190 FOR C=1TO54
6200 READ LN$(C),EX$(C)
6210 C1=C1+VAL(LEFT$(EX$(C),4))
6220 C2=C2+VAL(RIGHT$(EX$(C),4))
6230 NEXT C
6240 READ CA:IFCA<>C1THEN PRINT"CHECKSUM ERROR":STOP
6250 READ CB:IFCB<>C2THEN PRINT"CHECKSUM ERROR":STOP
6260 RETURN
6270 REM **** INVENTORY DATA ****
6280 DATA ADDRESS NUMBER,45,KEY,34,LASER SHIELD,25
6290 DATA TICKET TO TRI-STATE,26,DATA CREDIT CARD,28
6300 DATA DIGITAYA,30,CODE BOOK,19,BUFFER ACTIVATING DEVICE,13
6310 :
6320 REM **** LOCATION & EXIT DATA ****
6330 DATA IN THE TV OUTLET,00000000
6340 DATA IN THE USER PORT,00090100
6350 DATA IN THE CASSETTE PORT,00110000
6360 DATA IN THE JOYSTICK PORT,00130000
6370 DATA IN A TRI-STATE DEVICE,00170000
6380 DATA IN THE ARITHMETIC & LOGIC UNIT,00310016
6390 DATA AT THE GATEWAY TO MEMORY,00490000
6400 DATA ON THE I/O HIGHWAY,09000001
6410 DATA ON THE I/O HIGHWAY,10000802
6420 DATA ON THE I/O HIGHWAY,11000900
6430 DATA ON THE I/O HIGHWAY,12001003
6440 DATA ON THE I/O HIGHWAY,13531100
6450 DATA ON THE I/O HIGHWAY,14001204
6460 DATA ON THE I/O HIGHWAY,15001300
6470 DATA ON THE I/O HIGHWAY A SIGN SAYS 'S OUT H',00001400
6480 DATA IN THE DATA REGISTER,00061700
6490 DATA ON AN 8 LANE HIGHWAY,16001805
6500 DATA ON AN 8 LANE HIGHWAY,17001900
6510 DATA ON AN 8 LANE HIGHWAY,18002000
6520 DATA ON AN 8 LANE HIGHWAY,19292100
6530 DATA ON AN 8 LANE HIGHWAY,20282200
6540 DATA ON AN 8 LANE HIGHWAY,21272300
6550 DATA ON AN 8 LANE HIGHWAY,22262400
6560 DATA ON AN 8 LANE HIGHWAY,23250000
6570 DATA IN THE CHARACTER MATRIX,26360024
6580 DATA HIGH IN THE MEMORY,27352523
6590 DATA IN THE MIDDLE OF MEMORY,28342622
6600 DATA IN THE MIDDLE OF MEMORY,29332721
6610 DATA LOW IN THE MEMORY,00542820
6620 DATA IN THE ACCUMULATOR'S LAIR,00000600
6630 DATA IN A LONG CORRIDOR,00420006
6640 DATA IN AN INDEX REGISTER,31000000
6650 DATA LOW IN THE MEMORY,54403428
6660 DATA IN THE MIDDLE OF MEMORY,33393527
6670 DATA HIGH UP IN MEMORY,34383626
6680 DATA IN THE CHARACTER MATRIX,35370025
6690 DATA IN A RANDOM VECTOR TABLE,00000000
6700 DATA HIGH IN MEMORY OVERLOOKING A HIGHWAY,39003735
6710 DATA IN THE MIDDLE OF MEMORY,40003834
6720 DATA IN MEMORY - TO THE EAST IS A GATEWAY,41003933
6730 DATA LOW IN MEMORY,00004054
6740 DATA IN A CORRIDOR,00430031
6750 DATA IN A CORRIDOR,00440042
6760 DATA IN A CORRIDOR,00004543
6770 DATA IN THE ADDRESS REGISTER,00004600
6780 DATA ON A 16 LANE HIGHWAY,45004700
6790 DATA ON A 16 LANE HIGHWAY,46004800
6800 DATA ON A 16 LANE HIGHWAY,47004900
6810 DATA ON A 16 LANE HIGHWAY A LARGE GATE LOOMS TO THE WEST
     ,48005007
6820 DATA ON A 16 LANE HIGHWAY,49005100
6830 DATA ON A 16 LANE HIGHWAY,50005200
6840 DATA ON A 16 LANE HIGHWAY,51000000
6850 DATA IN A VECTOR TO MEMORY,00290012
6860 DATA LOW IN MEMORY,00413329
6870 REM ** CHECKSUM DATA **
6880 DATA 100169,103973
8599 :
8600 REM **** SEARCH FOR DIRECTION S/R ****
8610 NN$=NN$+" ":LN=LEN(NN$):C=1
8620 FOR1=1 TO LN
8630 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
8640 W$=MID$(NN$,C,I-C):C=I+1
8650 IF W$="NORTH" OR W$="EAST" THEN NN$=W$:I=LN
8660 IF W$="SOUTH" OR W$="WEST" THEN NN$=W$:I=LN
8670 NEXT I
8680 RETURN
```

Der große Bruder

Jede Produkteinführung von IBM wird allerorts mit großem Interesse und weltweiter Neugier betrachtet. So setzt auch der PC/AT einen neuen Standard für Microcomputer. Das Gerät arbeitet mit dem hochentwickelten Prozessor Intel 80286 und kann auf einen Arbeitsspeicher von drei MByte zugreifen.

Mit der Vorstellung des IBM PC im Jahre 1982 fand der Microcomputer seinen Eingang in die Geschäftswelt. Wie stark dieser Computer den Markt beeinflussen würde, war selbst für die Firma IBM nicht vorhersehbar. Der ursprüngliche IBM PC enthielt viele Bauteile anderer Hersteller, darunter der Prozessor Intel 8088, die Diskettenlaufwerke im 5 1/4-Zoll-Format und – als wichtigster Teil – das Betriebssystem PC-DOS. Trotz der Tatsache, daß zu diesem Zeitpunkt keine dieser Komponenten eine neue Technologie darstellte und der Maschine aus diesem Grunde starke Kritik entgegengebracht wurde, verkaufte sie sich in großen Stückzahlen. In der Folge entstand eine enorme Softwarebasis, die andere Hersteller veranlaßte, IBM-kompatible Geräte zu produzieren, auf denen die IBM-Software ebenfalls lief. Der IBM PC wurde dadurch zu einem inoffiziellen Industriestandard.

In der Zwischenzeit ist der Preis für Speicherchips rapide gefallen. Daraus resultierend verfügen die mit größeren Prozessoren ausgerüsteten kommerziellen Maschinen standardmäßig über 256 KByte RAM (der IBM PC hatte ursprünglich 64 KByte), und die Softwarehäuser stellen umfangreichere Programme her, die diesen erheblich erweiterten Speicherbereich nutzen.

In diesem Trend stellt der IBM PC/AT (Personal Computer / Advanced Technology) einen Höhepunkt dar. Das Gerät kommt der Kapazität eines Minicomputers (ein leistungsfähiges Mehrplatzsystem) nahe und ist mit einem der modernsten Prozessoren ausgerüstet: dem Intel 80286. Dieser Chip verfügt über einen 16-Bit-Datenbus und einen 24-Bit-Adreßbus und kann über 16 MByte direkt adressieren. Er ähnelt dem 8086-Prozessor, mit dem der Olivetti M-24 und die Apricot-Reihe arbeiten. Der Chip bietet aber auch die Möglichkeit der „virtuellen Speicherverwaltung", mit der er über ein GigaByte (1000 MegaByte) adressieren kann.

Bei der virtuellen Speicherverwaltung sieht der Prozessor den RAM-Bereich und das Speichermedium als „Hauptspeicher" an. Alle Daten und Programmteile, die beim Ablauf eines Programms nicht in den RAM-Speicher passen, sind auf Hochgeschwindigkeitsplatten untergebracht. Wenn der Computer Informationen braucht, die nicht im RAM liegen, überträgt er die entsprechenden Daten von der Platte in einen gerade nicht genutzten Speicherbereich und bearbeitet sie dort. Der Computer scheint damit über mehr Speicher zu verfügen, als er in Wirklichkeit besitzt.

Als IBM 1982 auf den Markt für Personal Computer vorstieß, wurde der Microcomputer plötzlich von der Geschäftswelt akzeptiert. Im Augenblick ist der neue IBM PC/AT einer der besten Microcomputer, die für den kommerziellen Einsatz zur Verfügung stehen. Der PC/AT ist dreimal so schnell wie der IBM PC und kann einen Arbeitsspeicher von bis zu drei MegaByte adressieren.

Gute Problemlösung

Die Technik der virtuellen Speicherverwaltung wurde ursprünglich für die Kombination teurer RAM-Kapazitäten mit preisgünstigeren Speichermedien eingesetzt. Inzwischen ist der Preis für RAM-Chips jedoch stark gefallen, und der AT verwendet diese Methode auch nicht aus Preisgründen, sondern wegen des Betriebssystems. Es wurde speziell für den PC/AT angepaßt, um die Lösung für ein wichtiges Problem liefern zu können.

Das PC-DOS und das eng damit verwandte MS-DOS können 640 KByte adressieren. Zum Zeitpunkt ihrer Entwicklung wurde angenommen, daß diese Speichergröße für jede nur denkbare Anwendung ausreicht. Schon kurze

Zeit später stießen jedoch viele der moderneren integrierten Softwarepakete bis an die Grenzen dieser Betriebssysteme vor, und es mußten innerhalb der Einschränkungen des PC-DOS weitere Speicherkapazitäten verfügbar gemacht werden.

Zur Lösung des Problems wurde der restliche Speicher als „RAM-Diskette" eingesetzt. RAM-Disketten sind Speicherbänke, deren Größe bis zum maximalen Adreßraum reichen kann (auf dem PC/AT liegt die Standardbegrenzung bei 64 KByte) und die vom Prozessor wie Diskettenlaufwerke behandelt werden. Der Prozessor spricht dabei die Daten nicht direkt an, sondern überträgt sie blockweise in den adressierbaren Speicher, um sie bearbeiten zu können. Nicht mehr benötigte Daten werden wieder auf die RAM-Diskette zurückgeschrieben. Diese RAM-Disketten haben extrem schnelle Zugriffszeiten, da sie nicht wie herkömmliche Disketten arbeiten, sondern aus RAM-Chips bestehen.

Die Diskettenlaufwerke des PC/AT haben zwar immer noch das 5 1/4-Zoll-Format, wurden aber gegenüber den langsamen und lauten Laufwerken des IBM PC wesentlich verbessert. Statt 160 KByte können die Laufwerke des PS/AT 1,2 MegaByte speichern. Die Aufstockung wurde teils durch die Verbesserung der Laufwerke und teils durch Veränderungen des DOS möglich. Die PC-DOS Version 3 des PC/AT schreibt pro Spur fünfzehn statt acht Sektoren (IBM PC). Der AT kann Disketten im PC-DOS-Format des IBM PC lesen, die PCs aber keine Disketten im PC/AT-Format.

Komfortabel

Außer des erweiterten Diskettenformats bietet das PC-DOS 3 aber auch einige neue Befehle. Durch ATTRIB können Dateien mit Schreibschutz versehen werden, während LABEL einer Diskette einen individuellen Namen zuordnet. SELECT bestimmt den Aufbau der Tastatur sowie das Format von Datum und Zeit, und COUNTRY schaltet diese Formate entsprechend der Schreibweise anderer Länder um. SHARE erlaubt den Mehrfachzugriff auf Dateien, FCBS legt die Anzahl der Dateisteuerblöcke fest, die gleichzeitig eröffnet sein können, DEVICE bestimmt die Größe und Zahl der virtuellen Disketten, und LASTDRIVE gibt die Anzahl der Laufwerke an, die das System einsetzen kann.

Es überrascht etwas, daß der PC/AT nicht völlig mit dem ursprünglichen PC kompatibel ist. Teilweise liegt es an den neuen Diskettenlaufwerken, die zwar verbessert wurden, aber mit dem Kopierschutz einiger hochentwickelter PC-Software nicht fertig werden. Auch wurden durch die Verbesserungen des BIOS-ROM die Einsprungadressen bestimmter Routinen verändert. Da einige Chips des AT sich außerdem von denen des IBM PC unterscheiden, ist alle Software, die diese Adresse direkt anspricht, nicht mehr kompatibel.

Auch der Prozessor hat ein geringfügig verändertes Format. Obwohl der Intel 80286 und der 8088 von der gleichen Firma hergestellt werden, sind einige Befehle nicht kompatibel. Software, die mit diesen Befehlen arbeitet, läßt sich daher nicht mit dem anderen Prozessor einsetzen.

Da außerdem die DIP-Schalter des IBM PC auf dem PC/AT durch ein CMOS-RAM ersetzt wurden, mußte viel Software erst umgestellt werden, damit ihr Kopierschutz auch mit dem neuen Format funktioniert. Selbst der Flugsimulator von Microsoft (FS1), der als idealer Test für Kompatibilität gilt, wurde in einer aktualisierten Fassung (FS2) herausgebracht.

Trotz dieser Inkompatibilität ist der IBM PC/AT eine ausgezeichnete Maschine, bei der fast alle Kritikpunkte des ursprünglichen IBM PC bereinigt wurden. Speziell die Tastatur ist eine der besten für einen Microcomputer.

Wie zu erwarten, ist die neue Maschine wesentlich schneller als der alte IBM PC – einige Anwendungen laufen in einem Drittel der Zeit. Mit den Erweiterungsmöglichkeiten, die der große Adreßraum bietet, gibt es kaum Zweifel an der Zukunft dieses Gerätes.

IBM PC/AT

ABMESSUNGEN
540 x 420 x 160 mm

ZENTRALEINHEIT
Intel 80286 mit 6 MHz

SPEICHER-KAPAZITÄT
Grundmodell 256 K RAM; Ausbaumodell 512 K RAM, erweiterbar auf 3 MByte.

BILDSCHIRM-DARSTELLUNG
25 Zeilen mit je 80 Zeichen; maximale Auflösung 640 x 200 Pixel.

SCHNITTSTELLEN
Acht Erweiterungssteckleisten stehen für eine breite Palette von Schnittstellen zur Verfügung.

PROGRAMMIER-SPRACHEN
Alle gebräuchlichen Sprachen.

HANDBÜCHER
Die Handbücher sind auf dem bei IBM zu erwartenden Niveau.

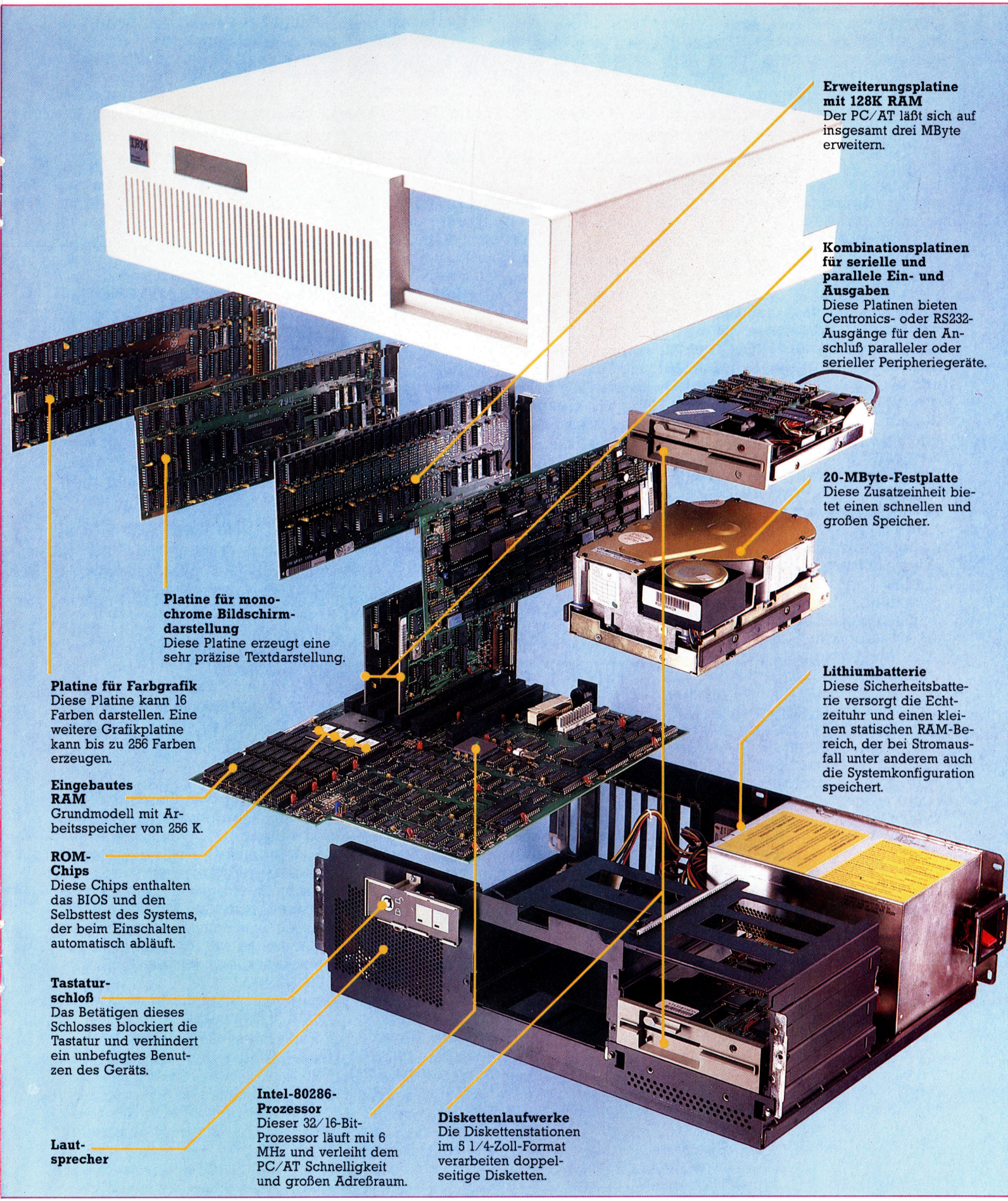
STÄRKEN
Der PC/AT ist im Augenblick einer der flexibelsten Microcomputer für den kommerziellen Einsatz. Er wird von dem größten Computerhersteller der Welt gebaut. Sein Erfolg ist vorgezeichnet.

SCHWÄCHEN
Einige PC-Software ist nicht mit dem AT kompatibel.

Virtuelle Speicher

Die Technik der virtuellen Speicherverwaltung wurde ursprünglich für Großcomputer entwickelt. Die Mechanik der virtuellen Speicherverwaltung ist einfach: Beim Ablauf eines umfangreichen Programms oder einer Programmfolge können oft nicht alle Module im Arbeitsspeicher untergebracht werden. Die Programme werden daher auf einem schnellen Speichermedium – zum Beispiel einer Festplatte – untergebracht. Während des Programmlaufs lädt der Computer nur die Programmteile, die gerade benötigt werden. Nach Beendigung eines Teilprogramms wird der entsprechendes Code gelöscht und ein anderer Teil nachgeladen. Wenn dieser Austauschvorgang schnell genug stattfindet, entsteht der Eindruck, daß sich alle Programmteile im Speicher befinden und gleichzeitig laufen.

Die Vorteile der virtuellen Speicherverwaltung treten am deutlichsten zutage, wenn im Computer mehrere Aufgaben gleichzeitig ablaufen. Das Gerät kann beispielsweise Daten laden, formatieren und drucken, während parallel dazu ein Netzwerk betrieben wird und eine Textverarbeitung läuft. Obwohl es aussieht, als ob alle Programme gleichzeitig ablaufen, gibt es Perioden, während der die Programme nichts ausführen, das heißt auf den Empfang von Datenblöcken oder auf Eingaben oder Signale von Peripheriegeräten warten. In dieser Zeit kann der Computer seinen Speicher und seine Rechenzeit für andere Aufgaben einsetzen und beispielsweise die benötigten Programmblöcke nachladen.



Erweiterungsplatine mit 128K RAM
Der PC/AT läßt sich auf insgesamt drei MByte erweitern.

Kombinationsplatinen für serielle und parallele Ein- und Ausgaben
Diese Platinen bieten Centronics- oder RS232-Ausgänge für den Anschluß paralleler oder serieller Peripheriegeräte.

20-MByte-Festplatte
Diese Zusatzeinheit bietet einen schnellen und großen Speicher.

Platine für monochrome Bildschirmdarstellung
Diese Platine erzeugt eine sehr präzise Textdarstellung.

Platine für Farbgrafik
Diese Platine kann 16 Farben darstellen. Eine weitere Grafikplatine kann bis zu 256 Farben erzeugen.

Lithiumbatterie
Diese Sicherheitsbatterie versorgt die Echtzeituhr und einen kleinen statischen RAM-Bereich, der bei Stromausfall unter anderem auch die Systemkonfiguration speichert.

Eingebautes RAM
Grundmodell mit Arbeitsspeicher von 256 K.

ROM-Chips
Diese Chips enthalten das BIOS und den Selbsttest des Systems, der beim Einschalten automatisch abläuft.

Tastaturschloß
Das Betätigen dieses Schlosses blockiert die Tastatur und verhindert ein unbefugtes Benutzen des Geräts.

Lautsprecher

Intel-80286-Prozessor
Dieser 32/16-Bit-Prozessor läuft mit 6 MHz und verleiht dem PC/AT Schnelligkeit und großen Adreßraum.

Diskettenlaufwerke
Die Diskettenstationen im 5 1/4-Zoll-Format verarbeiten doppelseitige Disketten.

Sprachprobleme

Im Bereich der Künstlichen Intelligenz hängt viel von der Wahl der richtigen Sprache ab. Hier beschäftigen wir uns mit unterschiedlichen Programmiertechniken, die sich bei der KI-Forschung als wertvoll erwiesen haben.

Da die meisten KI-Forscher LISP oder PROLOG für die Programmierung benutzen, werden diese Sprachen häufig als „KI-Sprachen" bezeichnet. Dieser Ausdruck ist jedoch aus zwei Gründen irreführend: Zunächst wird der Eindruck vermittelt, als seien LISP wie PROLOG für die herkömmliche Datenverarbeitung nicht geeignet. Zum zweiten – und das scheint noch wichtiger – läßt es die Schlußfolgerung zu, daß Programme in LISP oder PROLOG automatisch mit Künstlicher Intelligenz ausgestattet sind. Weiterhin scheint es, als seien andere Symbolsprachen wie POP-11 oder SNOBOL4, um nur zwei zu nennen, für ernsthafte KI-Arbeit unbrauchbar.

LISP ist ein „Veteran" unter den Programmiersprachen. Ende der fünfziger Jahre wurde sie von John McCarthy am MIT entwickelt und ist somit ebenso alt wie COBOL. Zur Vervollkommnung gelangte die Sprache 1961, und ihre Form wurde seither kaum verändert. Eine kurze LISP-Einführung haben wir bereits gebracht. Ein Konzept aber, das bisher keine Erwähnung fand, für KI-Anwendungen jedoch von erheblicher Bedeutung ist, blieb unerwähnt: die „Property-List" (Zuordnungs-Liste).

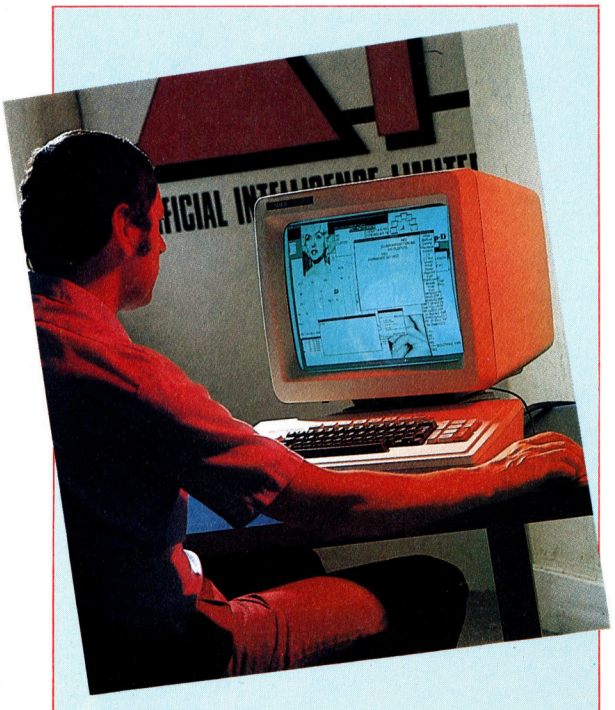
Jedes Atom verfügt über eine solche „Property-List", die aus „Attribut-Wert"-Paaren besteht. Die Property-List ist nichts weiter als eine Beschreibung des Atoms und erlaubt leichten Zugang zu Datenbank-Strukturen. Property-Lists werden durch die Befehle GET und PUTPROP manipuliert. Beispielsweise so:

```
(PUTPROP 'GESTERN 0.1 'NIEDERSCHLAG)
(PUTPROP 'MORGEN 'BEWOELKT
    'VORHERSAGE)
```

Damit wird 0.1 als Wert des Attributs NIEDERSCHLAG des Atoms GESTERN eingefügt. BEWOELKT ist der Wert von VORHERSAGE als Attribut für das Atom MORGEN. Die Attribute NIEDERSCHLAG und VORHERSAGE sind Feldern in einer Datenbank vergleichbar. Der Wert jedes gegebenen Attributs kann durch Verwendung von GET erhalten werden. So gelangte man beispielsweise durch

```
(SET Q REGEN (GET 'GESTERN'
    NIEDERSCHLAG))
```

an den NIEDERSCHLAG-Wert der GESTERN-Property-List und würde es als neuen Wert des Atoms REGEN anfügen.



KI-Entwicklungssystem
Das Interlisp-D-Netzwerk von Rank Xerox beinhaltet ein ideales Programmierumfeld mit interaktiven Grafikelementen, Fehlersuchroutinen und externen Speichergeräten mit 29 Megabyte Kapazität. Zusammen mit dem LOOPS-Programmiersystem bietet es dem KI-Forscher zahlreiche Möglichkeiten, unterschiedliche Programmiertechniken einzusetzen.

KI-Programmieren

Beim KI-Programmieren ist die Computersprache PROLOG die hauptsächliche Alternative zu LISP. Da es sich bei PROLOG um eine Erklärungssprache handelt, hat der Programmierer die Möglichkeit, Fakten und Regeln über irgendwelche Gegenstände und Verwandtschaften zu spezifizieren.

PROLOG beantwortet Fragen durch eine Theorem-Beweismethode, die auf dem „Entschluß-Prinzip" basiert. Die zu beweisende Behauptung wird dabei negiert, und es wird versucht, sie zu widerlegen – eine hochentwickelte Version der „reductio ad absurdum-Technik".

PROLOG kann wie LISP komplizierte Datenstrukturen erstellen (einschließlich Listen) und

verfügt über Eingabe-, Ausgabe-Möglichkeiten, File-Verwaltung und dergleichen mehr. Es ist eine leistungsfähige und flexible Sprache, entspricht aber in vielen Punkten nicht den Idealen logischer Programmierung.

LISP und PROLOG sind nicht die einzigen für KI-Programmierung geeigneten Sprachen. POP-11 gilt ebenso als KI-Kandidat. Ferner kann man auch mit traditionellen Sprachen wie C und PASCAL – oder sogar BASIC – effektive KI-Programmierarbeiten vornehmen. Das erfordert aber mehr Arbeitsaufwand.

Aufgrund der Komplexität der KI sind Programmierhilfen unbedingt wünschenswert. Zweifellos wird sich der neue Trend von KI-Programmen, die auf speziellen KI-Stationen laufen, fortsetzen.

LOOPS und POPLOG

Beispiele dafür sind LOOPS und POPLOG. LOOPS (LISP Object-Oriented Programming System) wurde im Forschungszentrum von Xerox in Palo Alto entwickelt. Es basiert auf einem LISP-Dialekt namens INTERLISP-D, bietet aber viel mehr Möglichkeiten als ein LISP-System. Es ist mit einer Vielzahl von Software-Tools ausgestattet – einigen konventionellen (wie Fenster- und Piktogramm-Verwaltung) und einigen weniger konventionellen (so etwa vorbereiteten Interferenz-Prozeduren für die Verwendung in Expertensytemen). Das System läuft auf einem speziellen KI-Rechner, der mit einem für LISP modifizierten und optimierten Microprozessor ausgestattet ist. POPLOG läuft auf VAX-Rechnern, benötigt also keine spezielle Software. Softwaremäßig ist es aber LOOPS ähnlich. Bildschirm-Edition und andere nützliche Standardpakete sind darin enthalten. Ferner gibt es die Möglichkeit, aus POP-11 (der eigentlichen Sprache) sowohl LISP als auch PROLOG aufzurufen. Ergänzend ist eine Bibliothek von Mustererkennungs- und anderen maßgeschneiderten Prozeduren für spezielle KI-Anwendung vorhanden.

Die damit zur Verfügung stehende Software steigert die Produktivität des Programmierens erheblich. Ihr eigentlicher Nachteil besteht jedoch darin, daß diese umfangreichen Applikationen nicht auf normalen Personal Computern lauffähig sind.

Bei Beginn einer Programmentwicklung wird ein KI-Problem nur sehr selten völlig erfaßt. KI-Forscher schreiben häufig nur Programme, um tieferen Einblick in die gestellten Probleme zu gewinnen. Die Betonung beim KI-Programmieren liegt deshalb auf „wachsenden Prototypen" – einer Programmiertechnik, in der Systeme durch viele kleine Zutaten und Veränderungen allmählich wachsen. Herkömmliche Software-Hilfen und Software-Ingenieurmethoden erfüllen die Anforderungen der KI-Programmierung nicht.

Arbeitshilfen

Im Lauf der Jahre sind viele Software-Hilfen entwickelt worden, die die Aufgaben des KI-Programmierers erleichtern. Somit können sich KI-Programmierer an anspruchsvolleren Projekten versuchen. Derartige „Werkzeuge" haben eine Reihe wichtiger Merkmale gemeinsam:

1. Die Sprache muß mit Recursionen arbeiten.

2. Die Sprache sollte über gute String-Verarbeitungs- und Symbol-Manipulierungsmöglichkeiten verfügen, womit flexible Datenstrukturen unbegrenzter Komplexität erstellt werden können.

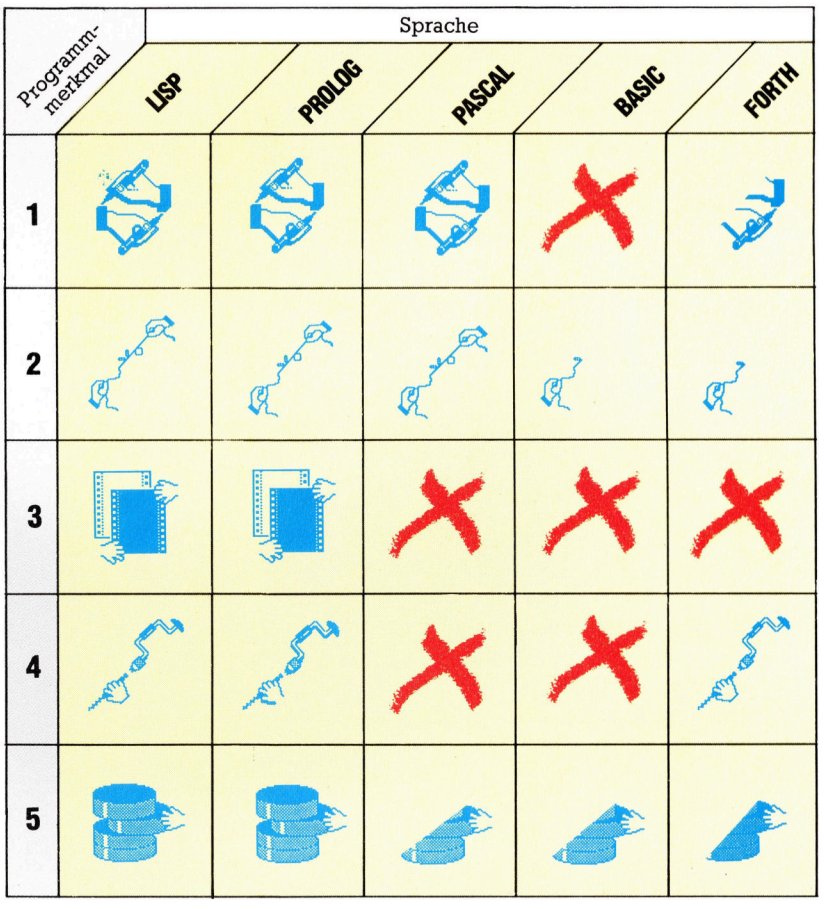
3. Wichtig ist die Übereinstimmung von Programm und Daten: Sowohl LISP als auch PROLOG stellen Programm und Daten in ein und demselben Format dar.

4. Die Syntax sollte erweiterbar sein. Wie bei LISP und PROLOG sollte es die Möglichkeit geben, in die Originalsprache eine neue einzubauen.

5. Der Zugriff zu integrierten Datenbänken ist wichtig: PROLOG ist mit einer Datenbank ausgestattet, LISP (Property-Lists) verfügt aber über vergleichbare Möglichkeiten.

Ergänzend werden für die Erleichterung der KI-Programmierarbeit andere Werkzeuge benötigt (Editoren, Grafikroutinen, Syntax-Prüfer, Debugger, Compiler, Dokumentations-Erzeuger u. a. m.).

Unser Diagramm zeigt, in welchem Umfang diese Merkmale bei vier verschiedenen Hochsprachen verfügbar sind.



Routine-Prüfung

Nachdem in unserer Artikelserie über die Technik des Programmierens bisher Anregungen zum Programmentwurf und zu den Details der Ausführung im Vordergrund standen, geht es hier abschließend um das Testen des fertigen Programms.

Ein Testdatensatz einschließlich der handgerechneten Ergebnisse für das Beispiel aus den Abbildungen 2 und 3 könnte folgendermaßen aussehen:

LEVEL	INPUT HITS	BONUS	OUTPUT BONUS
6	10	200	1300
4	10	550	2300
7	10	550	3950
4	10	200	800
7	10	200	1400
1	20	2500	2600
1	20	550	550
6	5	200	300
6	50	200	300
4	5	2500	2600
7	50	2500	2600
4	50	550	550
7	5	550	550

Ein Hauptvorteil von Interpretersprachen wie BASIC besteht darin, daß schon während des Programmeditierens ein Funktionstest möglich ist – man kann jederzeit RUN eintippen und verfolgen, was daraufhin passiert. Bei den meisten Rechnern können Sie den Probelauf auch mit BREAK unterbrechen, mit PRINT die aktuellen Werte von Schlüsselvariablen ausgeben lassen, diese verändern und das Programm dann mit CONTinue fortsetzen. So lassen sich formale und teilweise auch logische Fehler weitgehend, schon während der Programmeingabe, finden und korrigieren. Ein Test des fertigen Programms in seiner endgültigen Form bleibt aber trotzdem unerläßlich.

Diese letzte Prüfung soll sicherstellen, daß das Programm genauso arbeitet wie vorgesehen. Für jeden zulässigen Eingangsdatensatz muß das richtige Ergebnis herauskommen, und für jede unzulässige Eingabe die entsprechende Fehlermeldung. Selbst wenn Ihr Programm nur zwei Integer-Zahlen addiert und die Summe ausdruckt, müßten Sie etliche Milliarden von Wertepaaren durchspielen.

Testmethoden

Sie könnten auch daran denken, jeden möglichen Weg durch das Flußdiagramm vom Anfangs- zum Endpunkt zu prüfen. Jede neue Verzweigung eröffnet einen Alternativweg, und jeder neue Schleifendurchlauf wieder zusätzliche Kombinationen. In Abbildung 1 wird eine einfache Schleife mit Bedingungsabfragen (IF. . .THEN) für vier interne Wege dargestellt, die zehnmal durchlaufen wird. Dabei gibt es $4^{10}=1\,048\,576$ verschiedene Wegvarianten vom Start bis zum Ende – verblüffend für ein Programm, das sich in wenigen Zeilen niederschreiben läßt. Also scheidet auch das Testen der Wegvarianten aus. Welche Möglichkeit bleibt dann noch?

Überraschenderweise gar keine – Sie können ein komplexes Programm in realistischer Zeit nicht im genannten Sinn erschöpfend prüfen. Deshalb verläßt man sich beim Testen auf das Gesetz von der abnehmenden Effizienz jeder Fehlersuche: Die Anzahl der Fehler, die mit jedem neuen Durchgang gefunden werden, nimmt kontinuierlich ab. Man hört auf, wenn die weitere Fehlersuche kostspieliger würde als die Schadenbeseitigung bei eventuell verbliebenen Restfehlern.

Unabhängig davon lohnt es sich, über rationelle Testmethoden nachzudenken. Eine vernünftige Annahme ist zum Beispiel, daß ein Programm, wenn es einen bestimmten Satz von Eingangsdaten richtig verarbeitet, auch mit gleichgearteten neuen Zahlenwerten korrekt verfahren wird. Wenn ein Unterprogramm mit einer positiven Integer aus seinem Definitionsbereich einwandfrei läuft, sollte es auch mit jeder anderen zugelassenen positiven Integer funktionieren. Das führt zum „Äquivalenzklassen"-Verfahren: Sie denken sich Testdatensätze aus, die jeweils für eine ganze Klasse von gleichartigen Werten repräsentativ sind. Wird etwa irgendwo abgefragt, ob ein Parameter X zwischen 1 und 100 liegt, dann braucht man drei Testgrößen X_1, X_2 und X_3 mit den Werten $X_1<1$, $X_2>100$ und $1\leqslant X_3\leqslant 100$.

Vierergespann

Selbst ein relativ einfaches Gebilde wie diese Schleife mit vier internen Parallelpfaden läßt sich nicht erschöpfend austesten, weil es zu viele Durchlauf-Varianten gibt: Bei zehn Zyklen sind über eine Million unterschiedliche Streckenzusammenstellungen möglich.

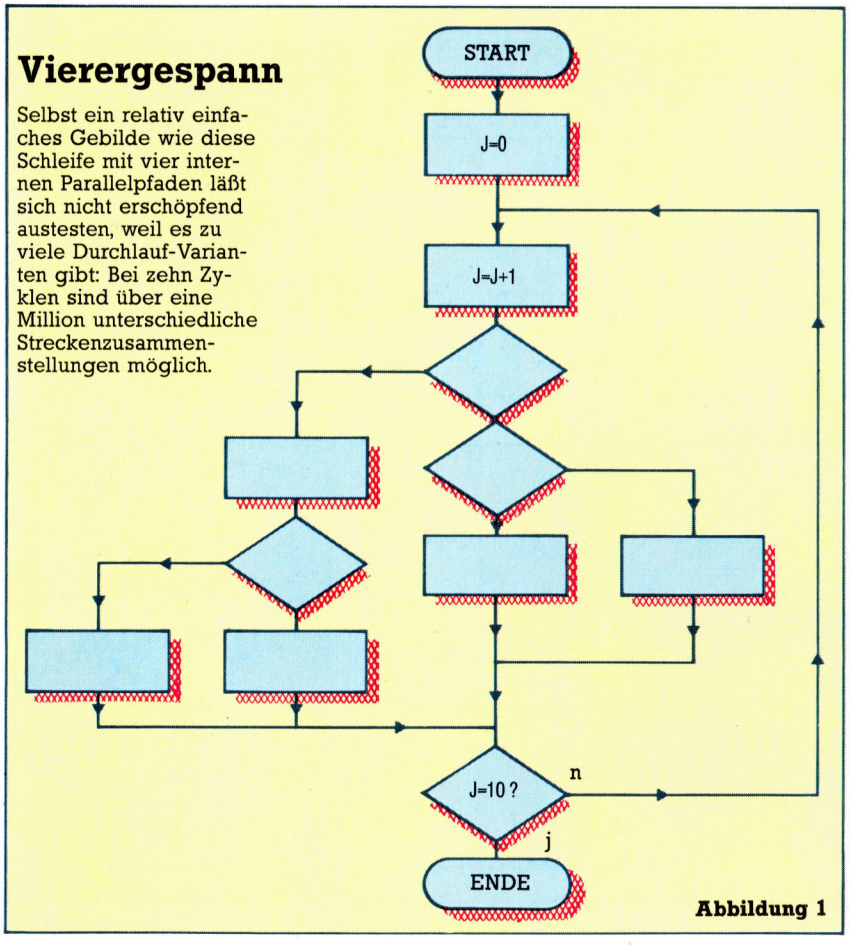


Abbildung 1

Die Überprüfung der Programmwege läßt sich so vereinfachen, daß auf die Einstiegsadresse jeder Routine verzweigt wird und die Testdaten so gewählt werden, daß jeder interne Ast einmal durchlaufen wird. In Abbildung 2 ermittelt ein Programm für ein Computerspiel aus der alten Punktzahl BONUS, dem Schwierigkeitsgrad LEVEL und der Trefferanzahl HITS den neuen BONUS. Der Text dazu könnte lauten:

```
6030 IF LEVEL>2 AND HITS=10 THEN
        BONUS = BONUS*LEVEL
6040 IF LEVEL = 6 OR BONUS> 2000 THEN
        BONUS = BONUS + 100
```

Um die möglichen Ergebnisse jeder Bedingungsabfrage abzudecken, ist zu überlegen, welche Eingangswerte die Erfüllung bzw. Nichterfüllung der Bedingungen bedeuten. Die Abfragen in Abbildung 2 betreffen je zwei durch AND/OR verknüpfte Variablen. Entscheidend ist, wie sich die Kombination der Variablen auswirkt. Wenn man zum Test der ersten Abfrage beispielsweise für LEVEL einmal 4 und einmal 1 und für HITS jedesmal 10, 5 und 20 vorsähe, würden mit LEVEL=4 zwar die drei Möglichkeiten für HITS ausprobiert, mit LEVEL=1 aber nicht. Hier ist ein Teil der Entscheidung durch einen anderen „maskiert".

Solche Abfragen sollten Sie zum Testen logisch zerlegen, wie es in Abbildung 3 dargestellt wird. Die vier einfachen Entscheidungen führen hier oben und unten auf je drei verschiedene Wege, je nach den Eingangswerten. Nützlich ist eine Zusammenstellung der Ja/Nein-Bedingungen für jede Abfrage:

Abfrage	1	2	3	4
Ja	LEVEL>2	HITS=10	LEVEL=6	BONUS>20000
Nein	LEVEL=2 LEVEL<2	HITS<10 HITS>10	LEVEL<6 LEVEL>6	BONUS=2000 BONUS<2000

Damit lassen sich repräsentative Testdaten herleiten. Zum Beispiel muß für die Route a/d/f/i der LEVEL>2, nämlich =6 sein, die HITS müssen <>10 sein und der (alte) BONUS ist beliebig, weil er nicht abgefragt wird. LEVEL=6, HITS=20 und BONUS=150 wäre ein möglicher Wertesatz (natürlich nicht der einzige). Der Pfad a/b/e/h/j würde mit LEVEL=4, HITS=10 und BONUS=600 (vor Multiplikation mit LEVEL!) angesteuert.

Unerläßlich ist eine Handrechnung mit jedem Datensatz als Gegenprobe zum Computer. In der Tabelle links wird der ganze Testdatensatz durchgerechnet.

Wie kann man nun mit ähnlichen Prüfmethoden großen Programmen zuleibe rücken, ohne von ihrer Komplexität erdrückt zu werden? Erste Voraussetzung ist eine übersichtliche Programmstruktur. Ist das Programm nach dem Prinzip der schrittweisen Verfeinerung aus unabhängigen Modulen aufgebaut, können Sie alle einzeln von oben nach unten testen (siehe Abbildung 4).

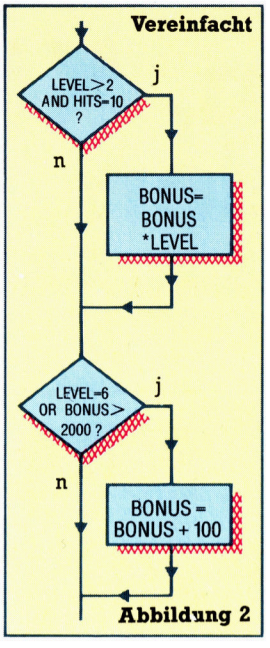


Abbildung 2

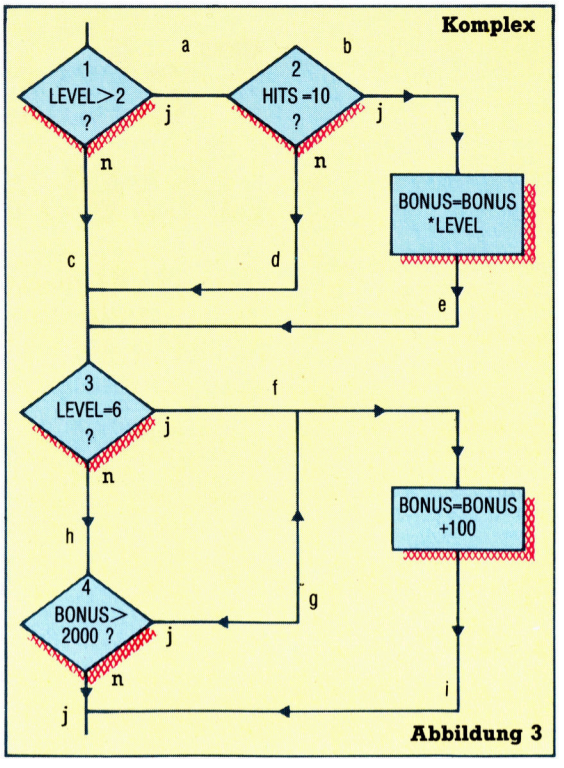


Abbildung 3

Am besten zerlegen Sie komplexe Entscheidungen und beschriften die Flußlinien, um die Tests zu vereinfachen.

Bei diesem Vorgehen kommt jedes Segment erst an die Reihe, wenn die darüberliegenden „Treibermodule" ausgetestet sind, so ist auch die Parameterversorgung gesichert. Die tiefer stehenden, ungetesteten und daher unzuverlässigen Module werden zunächst durch Scheinroutinen („Dummies") ersetzt – ein paar Programmzeilen, die bei Aufruf vorher eingespeicherte Testdaten abliefern. Das Ganze hat den Charakter eines „Testrahmens" (Test Harness), in den die Segmente zur Erprobung eingespannt werden. In Abbildung 4 sind die Module 1, 2 und 3 bereits ausgetestet, während 5, 6 und 7 durch Dummies simuliert werden. Die Nummer 4 wird gerade untersucht.

Wenn Sie die Testdatensätze und Ergebnisse aufheben, muß bei später auftretenden Fehlern nicht alles wiederholt werden, und Sie können anhand der Dokumentation die erforderlichen Ergänzungstests festlegen.

Wenn Ihr Programm eine „umgekehrte Baumstruktur" (Top-Down) aufweist, also vorschriftsmäßig nach dem Prinzip der schrittweisen Verfeinerung aufgebaut ist, können Sie jedes Segment unmittelbar nach Fertigstellung sowohl isoliert wie im Zusammenspiel mit anderen prüfen.

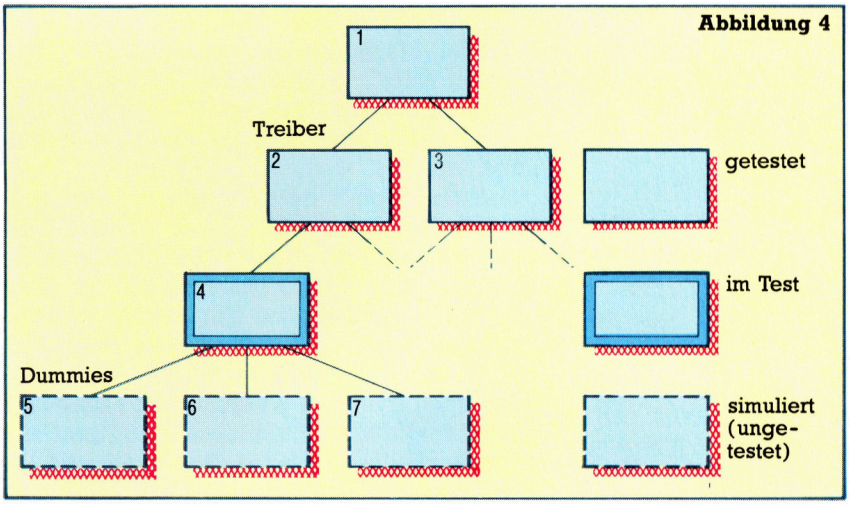


Abbildung 4

Felsen, Käfer und Juwelen

Schon kurze Zeit nach der Veröffentlichung des sehr erfolgreichen „Boulder Dash" kam „Rockford's Riot" auf den Markt. Bei der Fortsetzung legten die Programmierer Wert darauf, die einfache Form des Originals beizubehalten. Ziel beider Spiele ist es, durch Graben unterirdischer Gänge Juwelen zu finden und dabei Hindernissen wie Leuchtkäfern und herabfallenden Felsen auszuweichen.

Wie die Film- und Verlagsindustrie haben auch Software-Häuser festgestellt, daß Fortsetzungen populärer Produkte Garanten großer Umsätze sind. Wir stellen die beiden Programme „Boulder Dash" und das Nachfolgeprogramm „Rockford's Riot" vor.

Boulder Dash

Rockford's Riot

Wenngleich sich die Software-Industrie zu einem Millionen-Markt entwickelt hat, auf dem jährlich Tausende von Programmen angeboten werden, so hinterlassen doch die wenigsten einen nachhaltigen Eindruck. Nur allzuoft sind selbst sehr erfolgreiche Spiele wenige Monate nach ihrer Veröffentlichung bereits wieder vergessen. Es gibt jedoch einige Programme, die qualitativ gleichwertige Folgeprogramme nach sich ziehen. So etwa das Spiel „Manic Miner" und seine Fortsetzung „Jet Set Willy". Letzteres war so erfolgreich, daß der Hersteller das Spiel in einer neuen Version mit zusätzlichen „Räumen" veröffentlichte.

Rockford spielt den Helden in dem Programm, das in England als „Rockford's Riot" und in den USA als „Boulder Dash II" bezeichnet wird. Beide Spiele basieren auf der Idee, unterirdisch verborgene Edelsteine zu sammeln. Rockford muß sich durch die Erde wühlen, um an sie heranzukommen.

Natürlich gibt es – wie in jedem Computerspiel – zahlreiche Hindernisse. Hauptproblem in diesem Fall sind die auf dem Bildschirm angeordneten Felsen, die Rockford daran hindern, direkt auf die Juwelen zuzugraben.

Eine weitere Gefahr kommt in Gestalt der Leuchtkäfer, die selbst zwar keine Tunnel graben können, aber durch vorhandene oder von Rockford geschaffene fliegen. Die Berührung mit ihnen ist tödlich. Sie sind jedoch ebenso verwundbar wie Rockford, wenn sie unter fallendes Gestein geraten. Man kann also am Ende eines Schachts auf den Käfer warten und bei seinem Auftauchen einen der gefährlichen Felsen auf ihn werfen.

Um in den nächsten Schwierigkeitsgrad zu gelangen, muß eine bestimmte Anzahl von Edelsteinen gesammelt sein, durch die irgendwo auf dem Bildschirm eine Tür geöffnet wird, die Rockford den Durchgang erlaubt. Zunächst kann das verwirrend sein, da der Eindruck entsteht, als seien gar nicht genug Edelsteine zum Sammeln vorhanden. In beiden Spielen ist aber eine grüne Masse, „Amöbe" benannt, vorhanden, die langsam an Volumen zunimmt. Wird sie durch Felsen an ihrer Ausdehnung behindert, wächst sie zur „kritischen Masse" und kristallisiert zu Edelsteinen.

Der Erfolg bei diesen Programmen hängt sehr von schnellen Reaktionen an Joysticks und Tastatur ab. Doch ein gutes Spiel ist dadurch gekennzeichnet, daß die Bewegungen durchdacht sein müssen. Unter diesem Gesichtspunkt verdienen beide Programme großes Lob. „Boulder Dash" wie „Rockford's Riot" fordern im wesentlichen strategisches Können und Denkvermögen des Spielers.

Die Grafiken der Spiele sind ausgezeichnet. Die Bildschirme wurden mehrfarbig gestaltet, und Rollbewegung wie Kontrolle bzw. Steuerung durch den Spieler werden einwandfrei umgesetzt. Ferner zeichnen sich die Spiele dadurch aus, daß selbst kleinsten Details größte Aufmerksamkeit geschenkt wurde.

„Boulder Dash" und „Rockford's Riot":
Für C 64, Atari und ZX Spectrum
Hersteller: Beyond Software, Competition House, Farndon Road, Market Harborough, LE16 9NR
Autoren: Peter Liepa und Chris Gray
Programm: Cassette
Joystick: Empfehlenswert

Rangiermethoden

Wir gehen auf den verschiebbaren (relocatable) Code, die Befehlslängen und verschiedene Ausführungsgeschwindigkeiten ein.

Ein Programm mit verschiebbarem oder positionsunabhängigem Code kann in jeden beliebigen Speicherbereich geladen werden und dort ohne Änderungen funktionieren. Besonders bei Mehrplatzsystemen und Hintergrundprozessen hat dies große Bedeutung, da dabei mehrere Programme gleichzeitig in den Speicher geladen werden können und das Betriebssystem den Code oft an verschiedenen Positionen unterbringen muß. Doch selbst bei Einplatzsystemen sollten Unterroutinen als selbständige Module angelegt sein, da sie sich dann ohne Änderungen auch in anderen Programmen einsetzen lassen.

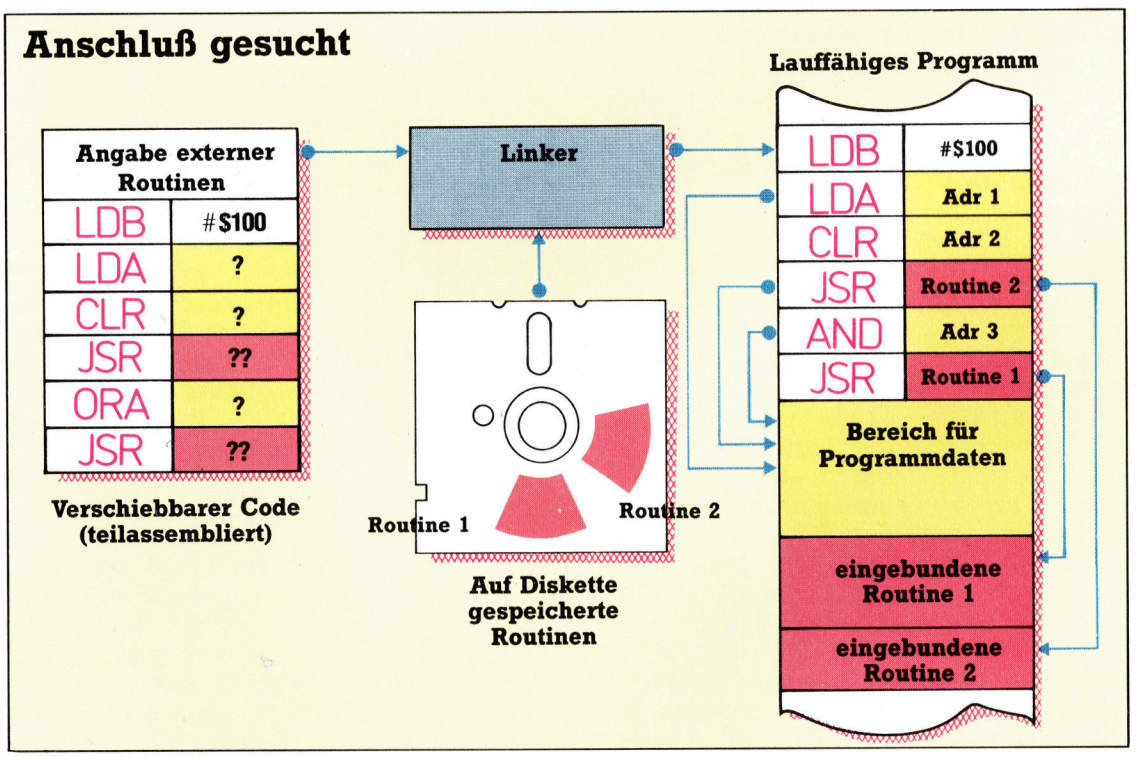
Die meisten Prozessoren verwenden einen sogenannten „Linker", um Adressen in den vom Assembler erstellten verschiebbaren Code (der nur relative Speicherstellen angibt) einzusetzen. Der Linker stellt dabei sicher, daß die Steuerung zwischen den einzelnen Modulen korrekt übergeben wird. Auf diese Weise ist es sogar möglich, die Codes unterschiedlicher Sprachen in den gleichen verschiebbaren Code zu compilieren oder zu assemblieren. PASCAL-Programme können so beispielsweise mit FORTRAN-Modulen arbeiten. Diese Technik ist auch auf dem 6809 möglich. Der 6809 bietet zusätzlich den Vorteil, daß ein verschiebbarer Code direkt eingegeben werden kann und die Adressen nicht erst durch einen zusätzlichen und zeitraubenden Arbeitsgang erstellt werden müssen.

Der Schlüssel zum verschiebbaren Code ist das „Offset", das alle Adressen mit Bezug auf den Befehlszähler (PC) angibt. Programme setzen Adressen auf zwei Arten ein: als Daten oder als Adressen, an die die Steuerung übergeben wird. Verzweigungsbefehle wie BRA oder BSR berechnen ihre Sprungadressen als Offsets vom PC und sollten daher für alle Sprünge innerhalb eines Anwenderprogramms verwandt werden. Absolute Sprungbefehle wie JMP und JSR sollten nur Adressen ansprechen, die sich im Speicher immer auf der gleichen Position befinden, beispielsweise die Routinen des Betriebssystems.

Weitaus schwieriger ist der unabhängige Bezug auf Datenpositionen. Der 6809 kann dafür den PC als Index einsetzen. Der Befehl

```
LDA OFFSET,PC
```

addiert das Offset (mit Vorzeichen) zu dem aktuellen Wert des PC und erhält so die „effektive Adresse". Allerdings ist diese Adressierungsart nicht ganz einfach, da bei der Berechnung der Differenz zwischen Datenadresse und dem Wert des PC berücksichtigt werden



Bei größeren Systemen werden die Maschinencodeprogramme vom „Linker" in den Speicher geladen. Dieses Dienstprogramm des Betriebssystems übernimmt vom Assembler den teilassemblierten Code (der keine absoluten Adressen enthält) und bestimmt aus dem aktuellen Status des Systems die geeignetste ORG-Adresse. Mit dieser „Schlüsseladresse" tauscht der Linker die vom Assembler eingesetzten symbolischen Programmadressen gegen absolute Adressen aus und bindet die vom Programmierer angegebenen externen Module in den Code ein. Die absoluten (CALL) Adressen der externen Module werden dabei gegen die symbolischen Adressen des Programms ausgetauscht. Schließlich übergibt der Linker das vollständige Programm zur Ausführung an das Betriebssystem.

muß, daß der PC unmittelbar nach dem Laden eines Befehls inkrementiert wird. Nach der Ausführung eines Befehls zeigt der PC daher immer auf die darauffolgende Instruktion.

Die vielen möglichen Befehlslängen des 6809 (ein bis fünf Byte) verkomplizieren diese Methode. So belegt

```
LDX OFFSET,Y
```

beispielsweise ein Byte für den Op-Code und ein Byte für das nachgestellte Byte, das bei jedem Indexvorgang angibt, welches Indexregister eingesetzt wird und ob die Adressierung indirekt erfolgen soll. Das Offset kann Null, ein oder zwei Byte belegen. Außerdem können Null-Offsets und Offsets, die sich in fünf Bits ausdrücken lassen, in das nachgestellte Byte integriert werden (einige Assembler haben damit jedoch Probleme). Längere Offsets benötigen entweder ein zusätzliches Byte (bis acht Bits Länge) oder zwei Byte. Spezielle Offsets für Null oder im Fünf-Bit-Format sind jedoch nicht möglich, wenn der PC als Index eingesetzt wird.

```
LDY OFFSET,X
```

würde ein zusätzliches Byte belegen, da der Op-Code für LDY zwei Byte lang ist.

Wenn Sie gern in der Assemblersprache programmieren und mit Datenadressen und Subroutinen umgehen können, wird Ihnen die Berechnung der Op-Code-Längen und der Offsets schon bald zur Selbstverständlichkeit werden. Sie können sich aber auch einen Assembler kaufen, der Ihnen diese Arbeit abnimmt. Die meisten Assembler verfügen über die Anweisung PCR (Programm Counter Relative – relativer Bezug zum Befehlszähler), die den Assembler anweist, den PC als Indexregister einzusetzen und den Offset selbst zu berechnen. Hier ein Beispiel:

```
DATITM FCB 0
.....
       LDA DATITM,PCR
```

Escape-Sequenzen

In unserem Programmbeispiel setzen wir die Offset-Technik ein, um Terminals zu emulieren (imitieren). Damit funktionieren Programme, die auf ein bestimmtes Terminal ausgerichtet sind, auch auf anderen Geräten. Die Unterschiede zwischen einzelnen Terminals zeigen sich besonders in den Codes, die die Bildschirmfunktionen (die Cursorpositionierung oder das Löschen des Schirms) steuern. Dafür werden entweder Steuercodes (Zeichen mit einem ASCII-Code unter 32) oder „Escape-Sequenzen" eingesetzt, die aus dem Escape-Zeichen (ASCII 27) gefolgt von einem oder mehreren Zeichen bestehen. Unser Programm ist recht einfach und kann nur jeweils ein Steuerzeichen durch ein anderes oder eine Escape-Sequenz durch eine andere ersetzen. Es zeigt aber, wie eine Emulation funktioniert. Die Routine legt zwei Tabellen an: eine für Steuerzeichen und die zweite für Escape-Sequenzen. Das Originalzeichen dient dabei als Tabellen-Offset für das Austauschzeichen.

Der Code ist frei verschiebbar und kann problemlos an jedes Programm angehängt werden. Wir setzen voraus, daß die Betriebssystemroutine OUTCH das im Akkumulator A gespeicherte Zeichen an den Bildschirm sendet. Da wir den Befehl JMP einsetzen, sollte sich diese Routine an einer festen Speicheradresse befinden. Beachten Sie, daß die ORG-Anweisung gegeben werden muß, obwohl sie hier keine direkte Funktion hat. Das angezeigte Zeichen sollte ein A sein.

Vier Regeln

Nicht nur bei der PCR-Adressierung müssen Befehlslängen errechnet werden. Oft wird die Gesamtlänge einer Routine gebraucht, damit sie in einen begrenzten Speicherbereich eingepaßt werden kann. Fast jedes Buch über den 6809-Assembler enthält eine Tabelle der mnemotischen Befehle und ihrer Daten. Bei jedem mnemotischen Kürzel steht der zugehörige Op-Code, die Gesamtlänge des Befehls, die Zahl der Taktzyklen, die der Befehl zur Ausführung benötigt, und die Auswirkungen auf die Flags des Condition Code Registers.

Es gibt vier allgemeine Regeln für die Berechnung von Befehlslängen:

1) Die meisten Op-Codes sind ein Byte lang. Op-Codes, die den Inhalt von S und Y ansprechen (außer LEA) und einige, die U einsetzen (LDY und STS), haben dagegen die Länge von zwei Byte.

2) Jede indizierte Adressierung braucht ein nachgestelltes Byte (Post-Byte) und möglicherweise je nach Länge des Offsets weitere ein oder zwei Byte.

3) Daten, die den Op-Codes für unmittelbare Adressierung folgen, sind je nach eingesetztem Register ein oder zwei Byte lang.

4) Direkte Adressen sind ein Byte lang, alle anderen zwei Byte. Nicht alle Assembler verwenden die direkte Adressierung optimal, so daß Adressen oft zwei Byte haben, die nur ein Byte brauchen.

Die vielen möglichen Befehlslängen verkomplizieren auch die Berechnung der Ausführungszeit: Sie ist von den Byte abhängig, die geladen werden müssen. Diese Information ist besonders für Echtzeitanwendungen und einige Peripherietreiber wichtig. Da die Ausführungszeit in Taktzyklen angegeben wird – oder zumindest in der geringstmöglichen Anzahl Taktzyklen –, hängt die reale Ablaufzeit von der Taktfrequenz ab. Auf dem 6809 wird oft ein MHz eingesetzt – eine Million

Zyklen pro Sekunde. Damit dauert jeder Taktzyklus eine Millionstelsekunde. Der Befehl

```
LDA DATITM
```

verwendet eine 16-Bit-Adresse und braucht fünf Zyklen – er dauert daher fünf Millionstelsekunden.

```
PSHS PC, B, CC
```

braucht fünf Zyklen plus einen Zyklus für jedes Byte, das auf den Stack geschoben wird – insgesamt also neun Zyklen, da der PC aus zwei Byte besteht.

Wenn ein System keine Echtzeituhr enthält, kann die Zeit nur durch eine Verzögerungsroutine gemessen werden. Dabei wird eine Befehlsfolge ausgeführt, deren Ablaufzeit die Länge der gewünschten Routine simuliert. Bei einer Taktfrequenz von einem MHz ruft unsere Routine Verzögerungen im Bereich von 1 bis 255 Millisekunden hervor. Die Anzahl der Millisekunden (ms) wird als Parameter an A (den Akkumulator A) übergeben.

Die Berechnung der Konstanten COUNT läßt sich folgendermaßen darstellen:

Befehl	Anzahl der Taktzyklen	Anzahl der Wiederholungen	Benötigte Zeit (Taktzyklen
PSHS B,CC	7	1	7
LDB #COUNT	2	(A)	(A) * 2
DECB	2	(A) * COUNT	(A) * COUNT * 2
BNE LOOP2	3	(A) * COUNT	(A) * COUNT * 3
DECA	2	(A)	(A) * 2
BNE LOOP1	3	(A)	(A) * 3
PULS PC,B,CC	9	1	9

Die Ablaufzeit errechnet sich aus dem Gesamtwert von (A) * (7 + 5 * COUNT) + 16 Taktzyklen. Damit die Berechnung einfacher wird, ignorieren wir die 16 Taktzyklen. (Da Intervalle dieser Art üblicherweise in Millisekunden gemessen werden, sind die 16 Millionstelsekunden unerheblich.) Bei einer Taktfrequenz von einem MHz enthält eine Millisekunde 1000 Taktzyklen. Die Gesamtzahl ist daher (A) * 1000 Taktzyklen.

(A) * (7 + 5 * COUNT) = (A) * 1000
(7 + 5 * COUNT) = 1000
5 * COUNT = 973
COUNT = 195
(nächste Ganzzahl)

Zwar können andere Routinen exaktere Verzögerungszeiten errechnen oder mit 16-Bit-Adressen einen größeren Bereich umfasssen, alle dekrementieren jedoch ein Register um eine feste Zahl.

Emulations-Routine

```
ESCAPE  EQU   27
SPACE   EQU   32          (Leerzeichen ist ASCII 32)
OUTCH   EQU               Hier die Betriebssystemadresse eintragen
        ORG   $1000
CTABLE  RMB   32          Tabelle der Steuerzeichen
ETABLE  RMB   128         Tabelle der Escape-Zeichen
EFLAG   FCB   0           Dieses Flag zeigt an, ob das letzte Zeichen ein Escape war
DISPCH  PSHS  X           X speichern
        TST   EFLAG, PCR  Testen, ob das letzte Zeichen ein Escape war
        BEQ   DISP1       Falls kein Escape, auf DISP1 verzweigen
        LEAX  ETABLE, PCR Sonst die Adresse von CTABLE nach X
        LDA   A,X         Mit dem Originalzeichen als Offset das Austauschzeichen in A laden
        CLR   EFLAG,PCR   EFLAG löschen
        BRA   FINISH
DISP1   CMPA  SPACE       Testen, ob Steuerzeichen
        BGE   FINISH      Falls kein Steuerzeichen, zu FINISH verzweigen
        CMPA  ESCAPE      Sonst auf Escape testen
        BEQ   ESCCH       Falls ja, auf ESCCH verzweigen
        LEAX  CTABLE,PCR  Adresse von CTABLE in X laden
        LDA   A,X         Mit dem Originalzeichen als Offset das Austauschzeichen in A laden
        BRA   FINISH
ESCCH   INC   EFLAG,PCR   EFLAG setzen, um anzuzeigen, daß das Zeichen ein Escape war
FINISH  PULS  X           X wiederherstellen
        JMP   OUTCH       Zeichen in A anzeigen
        END               Hier gibt RTS am Ende von OUTCH die Steuerung an das aufrufende Programm zurück
```

Verzögerungs-Routine

```
                          Subroutine für Verzögerungen um (A) Millisekunden
COUNT   EQU   195         Siehe Berechnung
        ORG   $1000
DELAY   PSHS  B,CC        Die anderen beiden eingesetzten Register sichern
LOOP1   LDB   #COUNT      Eine Millisekunde zählen
LOOP2   DECB              Weiter dekrementieren
        BNE   LOOP2       Bis B Null erreicht
        DECA              Jede ms A dekrementieren
        BNE   LOOP1       Bis A Null erreicht
        PULS  PC,B,CC     Zurückspringen
```

Drachensterben

Das englische Unternehmen Dragon Data erzielte bis vor einigen Monaten gute Umsatzzahlen, doch finanzielle Probleme bereiteten weiteren Zukunftsplänen ein jähes Ende.

Dragon Data wurde 1981 als Tochtergesellschaft des englischen Spielzeugherstellers Mettoy gegründet, der sich in den Boom der Heimcomputer-Branche einkaufen wollte. Die walisische Gesellschaft für regionale Wirtschaftsentwicklung beteiligte sich finanziell am Bau einer Fabrik in Swansea, die 1982 mit der Produktion des Dragon 32 begann.

Dragon zog damals den Microprozessor Motorola 6809 seinen bei anderen Herstellern favorisierten Konkurrenten Z80 und 6502 vor. Bei der Schaltungsauslegung hielt man sich an die Empfehlungen des Chipherstellers Motorola, was dem Dragon den Ruf als angebliche Kopie des Tandy-Color-Computers einbrachte. Computerfreunde fanden bald einen willkommenen Nebeneffekt der Identität der Microprozessoren: Einige Tandy-Programme laufen auch auf dem Dragon 32.

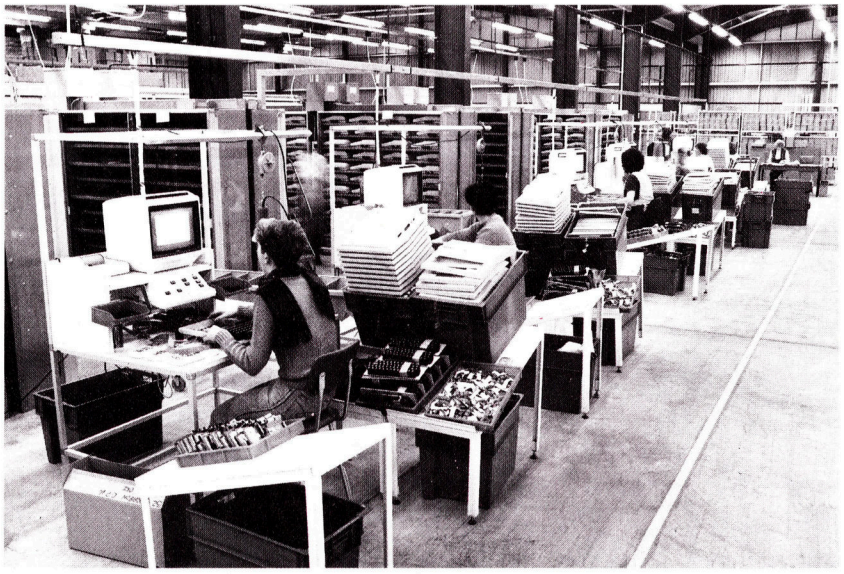
Ein starkes Argument für den Dragon war neben seinem Microsoft-BASIC (der meistverwendete BASIC-Dialekt) auch die vollwertige Schreibmaschinentastatur – 1982/83 gab es zu vergleichbaren Preisen nur noch den VC 20 mit einem ähnlich komfortablen Keyboard. Aber auch die Verkaufsstrategie von Dragon Data zeigte – zumindest bei unserem EG-Nachbarn England – große Wirkung: Weihnachten 1982 gab es den Commodore 64 noch nicht, andere attraktive Rechner (Acorn, Sinclair) waren nicht in ausreichenden Stückzahlen lieferbar. Dadurch wurden in weniger als einem halben Jahr über 32 000 Dragons verkauft. Wichtig war auch die Verbindung der Muttergesellschaft zum Spielzeugmarkt, durch die Dragon über die Warenhausketten einen einfachen Weg zum Käufer fand.

Eine erste Krise

Trotzdem geriet Dragon bereits im Erfolgsjahr 1983 in finanzielle Bedrängnis. Mitten in einer Expansionsphase meldete das Mutterhaus Mettoy Konkurs an, was die Kreditwürdigkeit der Tochter entscheidend schwächte. Rettung kam durch ein Firmenkonsortium unter Führung der Prudec, welche für einen der größten englischen Versicherungskonzerne die Investitionen im Technologiebereich koordiniert. Neben einer Finanzspritze von fast neun Millionen Mark bekam das Unternehmen mit Brian Moore auch einen neuen Direktor. Der Liquiditätsmangel war behoben, Dragon Data konnte bald danach in Port Talbot eine zweite Fertigungsstätte errichten. Auch mit der Entwicklung des Dragon 64 und der geplanten Diskettenlaufwerke ging es wieder voran.

Der Dragon 64 hat 64 KByte RAM, eine verbesserte Tastatur und eine serielle RS232-Schnittstelle. Dragons 5 1/4-Zoll-Diskettenlaufwerke laufen unter Dragon-DOS und können daher sowohl vom Modell 32 wie vom Dragon 64 genutzt werden.

Für die ehrgeizigen Pläne des Unternehmens gab es jedoch bereits 1984 ein vorläufiges „Aus": Prutec und die walisische Gesellschaft für regionale Entwicklung verweigerten zusätzliches Kapital, Dragon meldete Konkurs an. Neben Peripheriegeräten waren allein für 1984 drei neue Computer in der Planung, von denen einer nach dem japanischen MSX-Standard arbeiten sollte. Hochfliegende Pläne – es scheint jedoch, daß auch walisische Drachen nicht sicher vor dem Aussterben sind.



Die meisten europäischen Computer-Anbieter lassen ihre Geräte von Vertragsfirmen herstellen. Das war bei Dragon Data anders. Die Rechner entstanden ausschließlich in firmeneigenen Produktionsstätten wie dieser Fabrik in Port Talbot, West Glamorgan.

Bis zum Konkurs ging Dragon-Data-Vertriebsdirektor Richard Wadman von der Entwicklung und dem erfolgreichen Verkauf einer ganzen Serie neuer Dragon-Computer aus. Die plötzliche Zahlungsunfähigkeit machte seinen Plänen ein vorschnelles Ende.

Galgenvögel

Wir erarbeiten die Struktur einer einfachen Version des überaus beliebten „Galgenspiels“ und zeigen Ihnen die Listings für den Acorn B und den Sinclair Spectrum.

Ziel des Galgenspiels ist das Erraten eines Wortes. Die einzige Information, die der Spieler erhält, ist die Buchstabenanzahl pro Wort, die durch Striche angezeigt wird. Ein korrekt erratener Buchstabe wird auf den Bildschirm gebracht. Kommt der genannte Buchstabe im Wort nicht vor, wird ein Teil einer Figur an einem Galgen gezeichnet. In diesem Programm wird das Bild nach zehn Fehlschlägen vollendet – das Strichmännchen wird gehängt, und das Spiel ist verloren.

Die Grundstruktur des Programms ist einfach. Es muß lediglich geprüft werden, ob ein über Tastatur eingegebener Buchstabe in dem gewählten Wort enthalten ist. (Unser Listing enthält elf Wörter, die am Ende des Programms in DATA-Anweisungen aufgeführt sind). Ist der eingegebene Buchstabe in dem entsprechenden Wort vorhanden, wird er an der korrekten Bildschirmposition dargestellt. Trifft dies nicht zu, muß er an einer anderen Stelle angezeigt werden, um zweimaliges falsches Raten zu verhindern. Zusätzlich soll das Programm bei einem Fehler in eine Unterroutine springen, um einen Teil des Galgens zu zeichnen.

Größere Auswahl

In beiden Versionen sind die verwendeten Wörter in Zeile 1020 und 1030 abgelegt. Die Anzahl der Wörter kann über weitere DATA-Anweisungen erhöht werden. Sie sollten aber darauf achten, daß die Wörter nicht länger als zehn Buchstaben sind. Diese Einschränkung läßt sich allerdings durch Ändern der Zeilen 30 und 50 aufheben. Zudem muß die Gesamtzahl der Wörter in den DATA-Anweisungen addiert sowie der Wert von N in Zeile 20 auf die aktuelle Größe geändert werden.

Am Programmanfang werden alle Wörter in ein Feld eingelesen. Eines der Feldelemente wird zufällig ausgewählt, und es erscheint ein Strich für jeden Buchstaben des Wortes auf dem Bildschirm. Das restliche Programm besteht aus einer Wiederholungsschleife. Bei ungültigen Eingaben erklingt ein Warnton, und das Programm springt an den Schleifenanfang, um auf eine neue Eingabe zu warten. Zudem wird geprüft, ob der Buchstabe bereits verwendet wurde. In diesem Fall erscheint eine Warnung und die Bitte um erneute Eingabe.

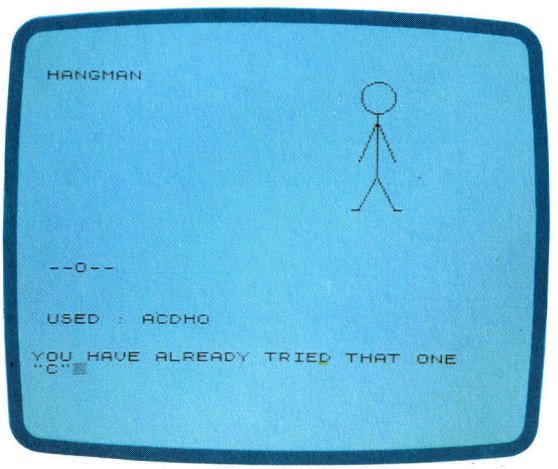
Akzeptiert das Programm das eingegebene Zeichen, wird dieses in die Liste der verwendeten Buchstaben aufgenommen und gleichzeitig mit allen Buchstaben des Wortes verglichen. Gibt es eine Übereinstimmung, wird der entsprechende Strich auf dem Schirm gegen den Buchstaben ausgetauscht. Andernfalls springt das Programm in eine Unterroutine, um einen Teil des Galgens zu zeichnen.

Aber bitte mit Musik

Hat der Spieler zehnmal falsch geraten, hängt die Figur vollständig am Galgen, ein Trauermarsch wird gespielt und ein neues Wort auf dem Datenfeld abgerufen. Wurde jedoch das Wort rechtzeitig erraten, erklingt eine Siegesfanfare.

Die beiden Versionen können leicht auf andere Heimcomputer übertragen werden. Die Unterroutinen für das Zeichnen des Galgens müssen dabei an die Grafikfähigkeiten der jeweiligen Maschine angepaßt werden. Programmierer, die eine eindrucksvolle Grafik erstellen wollen, können die Zeichnung optisch verbessern.

Das Spiel kann auch anderweitig verändert werden, zum Beispiel durch einen Test, der überprüft, ob das ausgewählte Wort bereits verwendet wurde. Nützlich wäre auch eine Routine, die nur Großbuchstaben als Eingabe erlaubt. Kleinbuchstaben, Zahlen und Symbole könnten auf ihre ASCII-Codes hin überprüft und gegebenenfalls abgelehnt werden. Das hier erarbeitete Programm bietet umfangreiche Möglichkeiten für engagierte Programmierer und leidenschaftliche Spieler.



Auf dem Bildschirm werden die Figur, der Galgen und die bisher erratenen Wortbestandteile dargestellt.

Acorn B

```
                    Hangman
10 REM
20 N=11
30 DIM W$(N)
40 REM initialise
50 D$="----------"
60 FOR I=1 TO N
70   READ W$
80   W$(I)=W$
90   NEXT I
100 MODE 1
110 REM start new word
120 CLS
130 PRINT TAB(5,1);"HANGMAN"
140 W$=W$(RND(N))
150 C=0:W=0:D=0
160 G$=""
170 PRINT TAB(5,23);"Letters tried :"
180 L=LEN(W$)
190 PRINT TAB(5,20);LEFT$(D$,L)
200 REM input the trial letter
210 PRINT TAB(5,30);L$
220 INPUT TAB(5,30)L$
230 REM check input letter
240 IF L$="" THEN SOUND 1,-15,50,5:GOTO 210
250 IF LEN(L$)>1 THEN SOUND 1,-15,50,5:GOTO 210
260 F=0
270 FOR I=1 TO LEN(G$)
280   IF L$<>MID$(G$,I,1) THEN GOTO 330
290   PRINT TAB(5,30)"YOU HAVE ALREADY TRIED THAT LETTER"
300   SOUND 1,-15,50,5
310   T%=TIME:REPEAT UNTIL TIME>T%+50
320   F=1
330   NEXT :
340 IF F=1 THEN 210
350 G$=G$+L$
360 PRINT TAB(5,25);G$
370 REM test letter against word
380 F=0
390 FOR I=1 TO L
400   IF L$=MID$(W$,I,1) THEN GOSUB 480
410   NEXT I
420 IF F<>1 THEN GOSUB 670
430 IF D=1 THEN GOSUB 590:GOTO 120
440 IF C=L THEN GOSUB 530:GOTO 120
450 GOTO 210
460 END
470 REM letter matches
480 PRINT TAB(4+I,20);L$
490 C=C+1
500 F=1
510 SOUND 1,-15,200,5:SOUND 1,0,0,2
520 RETURN
530 REM success!
540 FOR I=1 TO 200 STEP 10
550   SOUND 1,-15,I,2
560   NEXT I
570 RETURN
580 REM failure!
590 FOR I=200 TO 1 STEP -10
600   SOUND 1,-15,I,2
610   NEXT I
620 CLS
630 PRINT TAB(5,20);"THE WORD WAS : ";W$
640 T%=TIME:REPEAT UNTIL TIME>T%+200
650 RETURN
660 REM letter doesn't match
670 W=W+1
680 ON W GOTO 690,760,780,810,830,870,890,910,930,960
690 VDU29,800;800;
700 MOVE 50,0
710 FOR A=0 TO 7 STEP 0.4
720   DRAW 50*COS(A),50*SIN(A)
730   NEXT A
740 VDU29,0;0;
750 RETURN
760 MOVE 800,750:DRAW 800,550
770 RETURN
780 MOVE 740,450:DRAW 750,450:DRAW 800,550
790 DRAW 850,450:DRAW 860,450
800 RETURN
810 MOVE 750,630:DRAW 800,730:DRAW 850,630
820 RETURN
830 PLOT69,800,800
840 PLOT69,820,820
850 PLOT69,780,820
860 RETURN
870 MOVE 780,790:DRAW 800,770:DRAW 820,790
880 RETURN
890 MOVE 600,400:DRAW 1100,400
900 RETURN
910 MOVE 1000,400:DRAW 1000,900
920 RETURN
930 MOVE 1000,900:DRAW 800,900
940 MOVE 1000,850:DRAW 950,900
950 RETURN
960 MOVE 800,900:DRAW 800,850
970 MOVE 780,790:PLOT 14,800,770:PLOT 6,820,790
980 MOVE 780,770:DRAW 800,780:DRAW 820,770
990 D=1
1000 RETURN
1010 REM the words
1020 DATA "MUTATION","REBATE","SERAPH","HANGMAN","BBC"
1030 DATA "SPECTRUM","GEOFF","ELEPHANT","XENOPHOBE",
     "WRINKLE","GRENADE"
```

Spectrum

```
                    HANGMAN
10 REM
20 LET N=11
30 DIM X$(N,10)
35 DIM Y(N)
40 REM initialise
50 LET D$="----------"
60 FOR I=1 TO N
70 READ W$
75 LET Y(I)=LEN W$
80 LET X$(I)=W$
90 NEXT I
110 REM start a new word
120 CLS
130 PRINT AT 1,1;"HANGMAN"
140 LET I=(1+INT (RND*N))
145 LET W$=X$(I)
150 LET C=0: LET W=0: LET D=0
160 LET G$=""
170 PRINT AT 20,1;"USED : "
180 LET L=Y(I)
190 PRINT AT 16,1;D$(1 TO L)
200 REM input the trial letter
220 INPUT L$
230 REM check input letter
240 IF L$="" THEN  BEEP 0.25,-10: GO TO 210
250 IF LEN L$>1 THEN  BEEP 0.25,-10: GO TO 210
260 LET F=0
270 FOR I=1 TO LEN G$
280 IF L$<>G$(I) THEN  GO TO 330
290 PRINT AT 21,1;"YOU HAVE ALREADY TRIED THAT ONE"
300 BEEP 0.25,-10
310 PAUSE 25: PRINT AT 21,1,,
320 LET F=1
330 NEXT I
340 IF F=1 THEN  GO TO 210
350 LET G$=G$+L$
360 PRINT AT 20,8;G$
370 REM test letter against word
380 LET F=0
390 FOR I=1 TO L
400 IF L$=W$(I) THEN  GO SUB 480
410 NEXT I
420 IF F<>1 THEN  GO SUB 670
430 IF D=1 THEN  GO SUB 590: GO TO 120
440 IF C=L THEN  GO SUB 530: GO TO 120
450 GO TO 210
460 STOP
470 REM letter matches
480 PRINT AT 16,I;L$
490 LET C=C+1
500 LET F=1
510 BEEP 0.25,+16
520 RETURN
530 REM success!
540 FOR I=-10 TO 10
550 BEEP 0.1,I
560 NEXT I
570 RETURN
580 REM failure!
590 FOR I=10 TO -10 STEP -1
600 BEEP 0.1,I
610 NEXT I
620 CLS
630 PRINT AT 10,3;"THE WORD WAS : ";W$
640 PAUSE 50
650 RETURN
660 REM letter doesn't match
670 LET W=W+1
680 IF W=1 THEN  GO TO 690
681 IF W=2 THEN  GO TO 760
682 IF W=3 THEN  GO TO 780
683 IF W=4 THEN  GO TO 810
684 IF W=5 THEN  GO TO 830
685 IF W=6 THEN  GO TO 870
686 IF W=7 THEN  GO TO 890
687 IF W=8 THEN  GO TO 910
688 IF W=9 THEN  GO TO 930
689 IF W=10 THEN  GO TO 960
690 CIRCLE 200,150,10
700 RETURN
760 PLOT 200,140: DRAW 0,-40
770 RETURN
780 DRAW -10,-20: DRAW -4,0
790 PLOT 200,100: DRAW 10,-20: DRAW 4,0
800 RETURN
810 PLOT 190,110: DRAW 10,25: DRAW 10,-25
820 RETURN
830 PLOT 200,150
840 PLOT 196,155
850 PLOT 204,155
860 RETURN
870 PLOT 196,148: DRAW 4,-4: DRAW 4,4
880 RETURN
890 PLOT 170,60: DRAW 80,0
900 RETURN
910 PLOT 240,60: DRAW 0,115
920 RETURN
930 DRAW -40,0
940 PLOT 240,165: DRAW -10,10
950 RETURN
960 PLOT 200,175: DRAW 0,-15
970 OVER 1: PLOT 196,148: DRAW 4,-4: DRAW 4,4: OVER 0
980 PLOT 196,144: DRAW 4,4: DRAW 4,-4
990 LET D=1
1000 RETURN
1010 REM the words
1020 DATA "MUTATION","REBATE","SERAPH","HANGMAN","BBC"
1030 DATA "SPECTRUM","GEOFF","ELEPHANT","XENOPHOBE",
     "WRINKLE","GRENADE"
```

Datenbanken

In Datenbanken können Informationen problemlos organisiert werden. Sie ermöglichen unabhängig von der Datenmenge einen schnellen Zugriff auf Einzeldaten oder Datenkombinationen. In dieser Serie untersuchen wir den Aufbau von Datenbanken und strukturieren ein eigenes, vielseitig einsetzbares System.

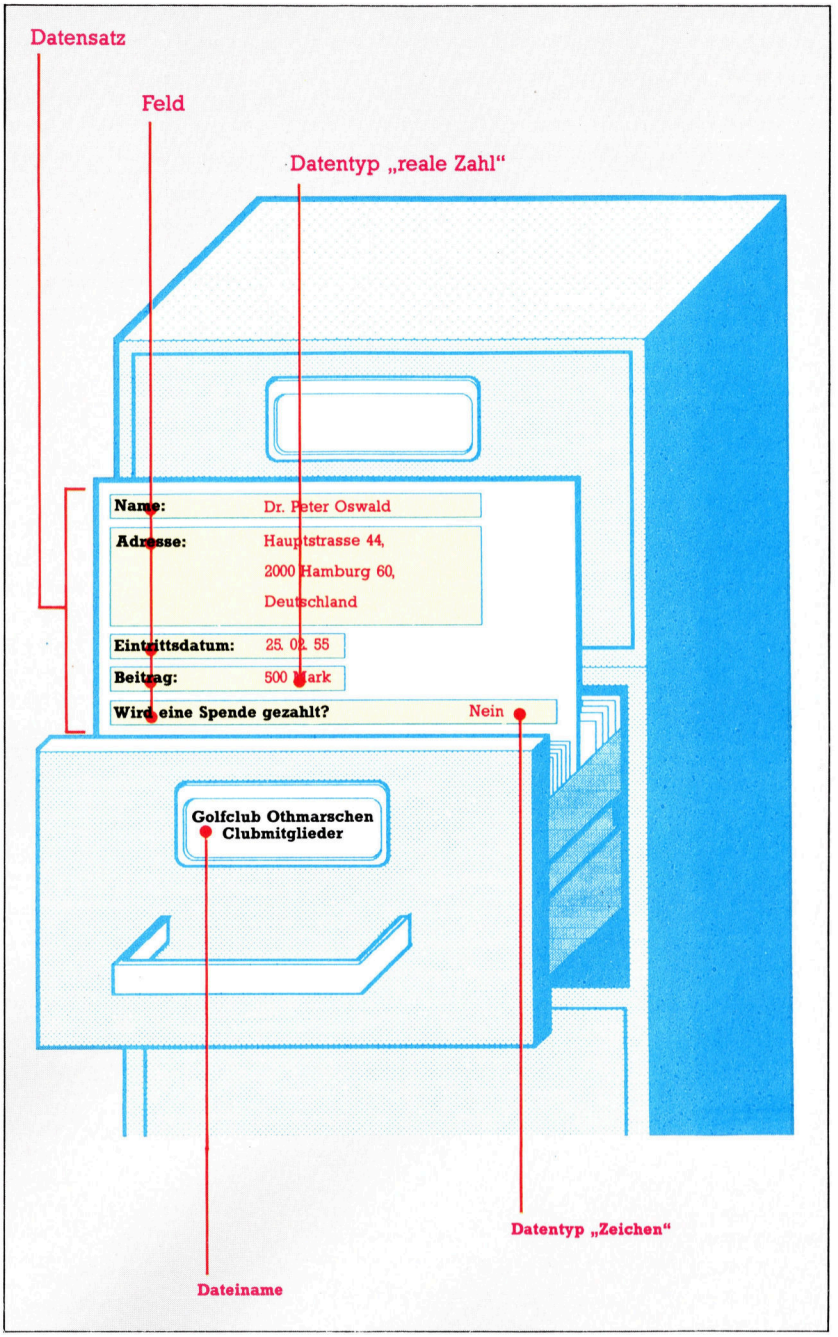
Man unterscheidet zwischen strukturierten und unstrukturierten Daten. Unstrukturierte Daten sind Einzelwerte, die normalerweise in Variablen oder Konstanten gespeichert sind. Hier zwei Beispiele: ANZAHL := ANZAHL + 1 oder IF WERT = 1 THEN PROCEDURE INKREMENTIEREN. In diesen beiden Beispielen sind „ANZAHL" und „WERT" die Variablen. Im ersten Fall wird der Wert einer Variablen verändert und im zweiten getestet (mit einer möglichen Verzweigung zu einer Prozedur). „ANZAHL" und „WERT" können dabei immer nur einen Wert annehmen.

Viele Programmierer – und besonders die BASIC-Programmierer – sind es gewohnt, mit unstrukturierten Daten umzugehen. Es gibt jedoch im täglichen Leben zahlreiche Beispiele für strukturierte Daten. So haben Sie beispielsweise in bezug auf Personen eine eigene „Datenstruktur" im Kopf.

Sätze und Felder

Sie kennen die Namen, Geschlecht, das Alter und vermutlich auch Beruf, Haarfarbe und weitere Informationen. Einen derartigen Satz zusammenhängender Daten (über eine Person, ein Konzept oder Objekt) nennen wir „Datensatz". Jeder Datensatz besteht aus einem oder mehr „Feldern" (mit gleichem oder unterschiedlichem Format). Eine Anzahl von Datensätzen wird als Datei bezeichnet.

Nehmen wir als Beispiel die Sekretärin eines Golf-Clubs, die Daten von 500 Mitgliedern zu verwalten hat. Um eine Struktur in die Daten bringen zu können, legt sie für jedes Mitglied eine Karteikarte an. Jede Karte enthält verschiedene Arten (Typen) von Informationen (Daten) über ein Mitglied: NAME, VORNAME, GESCHLECHT, ALTER, EINTRITTSJAHR, BEITRAG BEZAHLT?, RÜCKSTÄNDE, MITGLIEDSNUMMER etc. In der Fachsprache werden diese Einzeldaten als Felder eines Datensatzes bezeichnet, wobei die verschiedenen



Felder unterschiedliche Datentypen enthalten, NAME und VORNAME sind Zeichenketten, und die Zahl „1374662" wäre an dieser Stelle nicht richtig. GESCHLECHT ist eine Boolesche Variable, die nur einen von zwei binären Werten annehmen kann. ALTER ist vom Typ Ganzzahl (vorausgesetzt, wir müssen nicht wissen, ob ein Mitglied 37,624 Jahre alt ist). EINTRITTSJAHR ist ein Ganzzahl mit eingeschränktem Bereich. Selbst wenn wir die Daten aller früheren Mitglieder speichern würden, wäre die Zahl 1066 hier unangebracht. Auch

das Jahr 2001 wäre nicht korrekt, es sei denn, daß Kinder ihr Eintrittsdatum schon für die Zukunft eintragen könnten. RÜCKSTÄNDE ist eine reale Zahl – 62 345,67 ist durchaus möglich bei bestimmten Mitgliedern.

Professionelle Programmierer interessieren sich bei Programmiersprachen besonders für die Datentypen. Einige Sprachen, darunter BASIC und C, enthalten kaum fest definierte Typen, während PASCAL beispielsweise darauf besteht, daß alle Werte exakt mit den definierten Datentypen übereinstimmen.

Datentypen

Ein Vergleich zwischen BASIC und PASCAL zeigt, warum exakte Typendefinitionen wichtig sind. BASIC kennt nur zwei Datenstrukturen: Variablen und Dateien. Jede weitere Strukturierung muß vom Programmierer selbst vorgenommen werden. Es gibt unterschiedliche Typen von BASIC-Variablen: Ganzzahlenwerte (ZAHL% = 7), reale Zahlen mit einfacher Genauigkeit (RÜCKSTÄNDE! = 1234,56) und doppelter Genauigkeit (DISTANZ =123456,66666666666) und Stringvariablen (INTERESSANT$ = „Der Computer Kurs"). In BASIC gibt es keine Werte vom Typ Konstante. PI läßt sich zwar als PI = 3.141592 definieren, das Programm kann diesen Wert jedoch jederzeit umdefinieren (zum Beispiel als PI=6,2).

Andere Programmiersprachen – PASCAL ist das bekannteste Beispiel – bieten verschiedene vordefinierte Typen und erlauben außerdem die Definition neuer Typen. PASCAL hat folgende vordefinierte Typen: echte Konstante (nicht umdefinierbar) und Variablen vom Typ „char" (Zeichen), „integer" (Ganzzahlen), „real" (reale Zahlen wie 12,31) und „boolean" (Binärdaten, die entweder wahr oder falsch anzeigen). Die Sprache bietet weiterhin Sets, Arrays, Records und Files. Folgende Anweisung legt den neu definierten Datentyp TAG an:

```
TYPE
  TAG = (MONTAG,DIENSTAG,MITTWOCH,
    DONNERSTAG,FREITAG,SAMSTAG,
    SONNTAG);
```

In einem Programm mit diesem Befehl kann jede Variable vom Typ TAG nur einen der definierten Werte annehmen. Die Variable TAG_URLAUB vom Typ TAG, wird bei Zuordnungen wie TAG_URLAUB := NOVEMBER einen Fehler anzeigen, während TAG_URLAUB := SONNTAG korrekt ist.

Über exakte Typendefinitionen gibt es geteilte Meinungen, da sie umständlich sein können, aber auch helfen, Fehler zu vermeiden. Sehen wir uns nun an, welchen Wert Datentypen für Datenbanken haben.

Eine Datenbank ist eine Sammlung von strukturierten Daten, die miteinander in Beziehung stehen. Im allgemeinen werden Daten in Dateien (mit eigenem Dateinamen) gespeichert. Die Dateien bestehen aus Datensätzen (jeder mit einer eigener Satznummer), die Datensätze werden aus einem oder mehreren Feldern zusammengesetzt, und die Felder enthalten einen oder mehrere Datentypen.

Wenn wir uns eine Datenbank als den Index eines Karteikastens vorstellen, dann sind die alphabetisch angeordneten Namen fast immer der wichtigste Schlüssel. Um das Mitglied „Svenson" zu finden, durchblättern Sie die Karten, bis Sie die Namen mit S finden, suchen dann langsamer, bis die Namen mit SV auftauchen und stoßen schließlich auf die Einträge für Svenson. Wenn Sie jedoch herausfinden sollten, wie groß die Zahl der weiblichen Mitglieder ist, wäre die Aufgabe schon schwieriger, und noch mehr Zeit wäre notwendig, um festzustellen, wie viele Mitglieder älter als 36 Jahre sind und mit über 1000 Mark bei ihren Beitragszahlungen im Rückstand sind.

Für diese Art Aufgaben ist ein Datenbankprogramm jedoch ausgezeichnet geeignet. Genauer gesagt: Die Datenbank enthält die Informationen, und das Programm hilft bei der Eingabe und Bearbeitung der Daten. Um besser zwischen der Ansammlung von Daten und dem Bearbeitungsprogramm unterscheiden zu können, nennen wir das Programm „Datenbankverwaltung" oder kurz „DBV".

Es gibt große Unterschiede zwischen den einzelnen DBVs. Die einfachsten sind kaum mehr als elektronische Adreßbücher, die bei Eingabe einfacher Parameter (etwa NAME=?) einen Datensatz abrufen. Höher entwickelte DBVs arbeiten mit eigenen Programmiersprachen, mit denen die Informationen der Datenbank bearbeitet werden können und die die Daten auch in andere Formate umwandeln (zum Beispiel für Lohnprogramme, Rechnungserstellung oder Textverarbeitung). In dieser Serie untersuchen wir das Spektrum der DBVs von einfachen bis zu hochentwickelten Versionen und sehen uns ihre Funktionsweise an. Zuvor jedoch noch ein Wort über Datentypen: Bei komfortablen DBVs kann der Anwender für jedes Feld Datentypen definieren, die Einträge wie NAME = 123456 oder EINTRITTSJAHR = 1066 zurückweisen. Auch Abfragen wie „IF NOT (MAENNLICH OR WEIBLICH) AND GEHALT < 0" würden eine Fehlermeldung hervorrufen. In DBVs müssen exakte Typenprüfungen eingebaut werden, um Fehler schon bei der Eingabe oder Änderung abfangen zu können.

In den nächsten Folgen sehen wir uns an, wie verschiedene DBVs ihre Daten organisieren. Wir untersuchen Datenbanken, die nach der relationalen und der Mehrdateimethode aufgebaut sind, stellen die Vor- und Nachteile von festen Feld- und Datensatzlängen den variablen Längen gegenüber, richten Datenbanken ein und rufen Daten ab. Wir konzentrieren uns dabei auf drei DBVs – Psion's Archive, dBase II von Ashton Tate und Caxtons Software's Card Box.

Fachwörter von A bis Z

Information storage and retrieval = Speicherung und Wiedergewinnung von Informationen
„Storage and Retrieval" ist ein fester Begriff für die Handhabung von Daten in Informations- (speziell Datenbank-) Systemen. Die Information kann in Form von Programm- und Datenfiles oder als Grafik vorliegen. Als Datenträger kommen nur nichtflüchtige (das heißt gegen Stromabschaltung unempfindliche) Speicher in Frage, und selbstverständlich muß der Rechner geeignete Schnittstellen für die Datenübertragung aufweisen. Softwareseitig kann das selektive Suchen anhand unterschiedlicher Kriterien durch ausgetüftelte Retrieval-Programme unterstützt werden.

Information Technology = Informationstechnologie
Informationstechnologie gibt es einerseits schon seit Tausenden von Jahren: Die Hieroglyphen, der Abakus, der Federkiel und die Druckerpresse sind Beispiele dafür. Die Entwicklung der Microelektronik und ihre Verbreitung bei rapide sinkenden Preisen im letzten Jahrzehnt hat dazu geführt, daß man bei Informationstechnologie heute primär an die elektronische Informationsdarstellung, Übertragung und Verarbeitung denkt, insbesondere für Computer, Videosysteme und Telekommunikation. Die Mehrzahl der großen Unternehmen ist heute mit Rechnern ausgerüstet. Die weltweite Kommunikation erfolgt über Satelliten wie COMSAT, und militärische Systeme halten ständig jeden Quadratmeter der Erdoberfläche unter Beobachtung. Heute werden gewaltige Forschungsanstrengungen und Geldmittel in die Informationstechnologie gesteckt.

Bildnachweise
1488, 1494, 1495, 1496, 1501, 1511: Ian McKinnell
1489: Marcus Wilson-Smith
1491: Kevin Jones
1499: Chris Stevens
1500: Artificial Intelligenz Ltd.
1502, 1503, 1505: Liz Dixon
1509: Kevin Jones, Liz Heaney
U3: Steve Cross

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.

Information Theory = Informationstheorie
Die Informationstheorie (begründet um 1948 von Claude Shannon am Bell-Laboratorium in New Jersey/USA) beschäftigt sich mit dem Informationsgehalt und der Übertragung von Nachrichten. Das schließt auch die Analyse der Natur von Informationen und der Eigenschaften von Nachrichtenkanälen ein.

Zur Informationstheorie gehört auch die „Codiertheorie", die sich mit der Verschlüsselung (Codierung) von Daten befaßt und auch mit der Umsetzung in „redundante Codes", die eine Übertragung von Nachrichten ohne Verlust an Informationsgehalt sicherstellen.

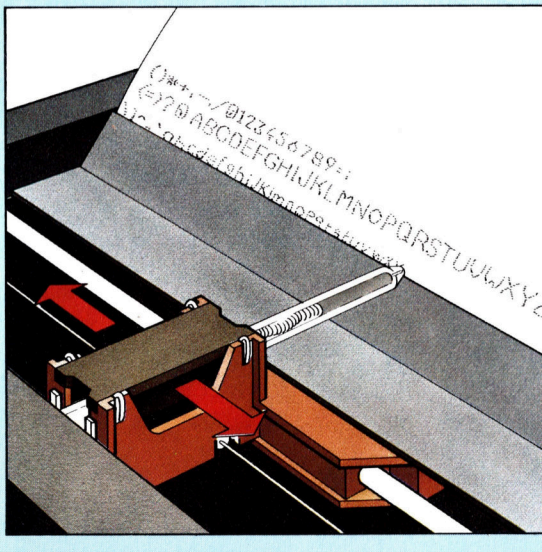
Initialisation = Initialisierung
Unter Initialisierung sind in erster Linie die Vorgänge zu verstehen, die nach dem Einschalten des Rechners ablaufen müssen, um die Betriebsbereitschaft zu gewährleisten. Dafür sind ROM-gespeicherte Routinen zuständig, die bestimmte Standardwerte in Register und diverse Speicherplätze eintragen, den Stackpointer (Stapelzeiger) auf Null setzen und die verschiedenen Ein- und Ausgabegeräte startklar machen. Die oberen und unteren Grenzen des RAM-Bereichs werden festgelegt, und die Zero Page wird erstellt. Schließlich beinhaltet die Initialisierung noch den Aufruf des Begrüßungsschirms und die Einstellung der Bildschirmeditor-Variablen.

Manchmal ist auch das Formatieren von Disketten gemeint.

Ink Jet Printer = Tintenstrahldrucker
Beim Tintenstrahlverfahren wird die Schrift auf dem Papier durch Versprühen feiner Tintentröpfchen aus einer Düse erzeugt.

Zwei Bauformen sind zu unterscheiden. Der „gepulste" Ink Jet arbeitet wie ein Nadeldrucker nach der Punktmatrix-Methode: Der Tröpfchenstrahl (aus einer Düse oder mehreren übereinander) wird bei der zeilenweisen Bewegung des Druckkopfs über das Papier abwechselnd gesperrt und freigegeben. Mit mehreren Röhrchen für verschiedenfarbige Tinten sind auch Farbausdrucke möglich.

Im Gegensatz dazu ist beim „kontinuierlichen" Ink Jet ein Tröpfchenstrahl während der Erzeugung eines Zeichens ständig aktiv, wobei die (geladenen) Tröpfchen hinter der Düse durch elektrische Felder abgelenkt werden. – Jedes Zeichen wird wie mit der Feder in einem Zug geschrieben. Die Tintenstrahldrucker arbeiten lautlos mit Geschwindigkeiten bis zu einigen hundert Zeichen/s. Überdies zeichnen sie sich durch ein hervorragendes Schriftbild und hohe Variabilität aus.



Der Tintenstrahl wird beim Austritt aus der Düse in winzige elektrisch geladene Tröpfchen zerteilt, die durch ein Ablenkfeld so gesteuert werden können, wie es das jeweilige Schriftzeichen erfordert. Dabei ist ohne weiteres Schönschrift erreichbar.

+ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +++ Vorschau +

computer kurs

Heft 55

Erziehung
Wir beginnen mit einer Serie, die den Einfluß von Computern in Schule und Erziehung analysiert und Tendenzen beschreibt.

Namentlich
Strukturierung von Datenbankprogrammen – eine faszinierende Aufgabe. Wir bringen Beispiele.

Übersicht
Klares und strukturiertes Programmieren in Assembler erleichtert den Umgang mit Ihrem Rechner, wenn es auf Schnelligkeit ankommt.

Schiff ahoi
Wir stechen in See und suchen Handel mit der Neuen Welt – BASIC-Simulationsspiel.