Einsteigen - Verstehen - Beherrschen

DM 3,80 öS 30 sfr 3,80

computer kurs

Ein wöchentliches Sammelwerk

Der 16-Biter NIMBUS

Richtiger Start

Die Expertensysteme

Floppies für Sinclair

Heft 47

computer kurs

Heft 47

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

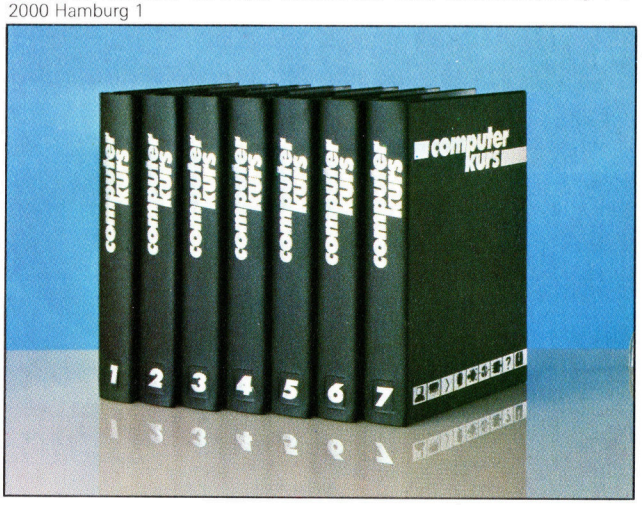
Schweiz: Der Sammelordner kostet sfr 15. Bitte wenden Sie sich an Ihren Kiosk; dort werden Sie jederzeit die gewünschten Exemplare erhalten.

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

Das Fachwissen

In dieser Folge unserer Serie über Künstliche Intelligenz beschäftigen wir uns mit Expertensystemen – hoch strukturierten Programmen, die diagnostische und beratende Aufgaben in bestimmten Wissensbereichen übernehmen. Dabei gehen wir auf die wichtigsten „Zutaten" eines Expertensystems ein.

Ein menschlicher Experte – sei er nun ein medizinischer Berater, ein Geologe oder ein Chemiker bzw. Analytiker – verbringt viel Zeit damit, seine Fähigkeiten zu verbessern. Er studiert, um seine Arbeit bestens verrichten zu können. Das Problem der menschlichen Experten jedoch ist: Es gibt nur wenige, sie stehen nicht immer zur Verfügung, sie verlangen Geld für ihre Leistungen, und wenn sie sterben, nehmen Sie viel von ihrem Wissen mit sich. Es ist verloren. Das sind die Gründe, warum viele Leute so erwartungsvoll auf Computerprogramme setzen und von der Idee des Speicherns von Expertenwissen so angetan sind: So erhält man viel Wissen in kompakter Form, ohne die mit dem menschlichen Experten verbundenen Nachteile. Auch das „menschliche Versagen" wird ausgeschlossen.

Dieses Konzept des „Expertensystems" entstand Anfang der siebziger Jahre, als die im Bereich der KI Forschenden ihre Suche nach intelligenten Maschinen abbrachen oder unterbrachen und sich statt dessen der Lösung von begrenzten Realwelt-Problemen zuwandten. Diese Expertensysteme sind somit eines der ersten Beispiele angewandter KI, und die daraus resultierenden Techniken haben sich viel weiter entwickelt, als man es in den Laboratorien für möglich hielt. Faktisch ist es so, daß Expertensysteme der KI erst den Einzug in die praktische Alltagsarbeit ermöglichten. Es gibt bereits Syteme, die beispielsweise im Bereich medizinischer Diagnosen Menschen weit übertreffen, die Spektrogramme lesen und auswerten können, Erkrankungen analysieren und vieles mehr.

Typisch für ein solches System ist eine weitreichende Informationsgrundlage über einen speziellen Problembereich. Dieses Wissen wird als Regelsammlung organisiert, die es dem System ermöglicht, Rückschlüsse aus gegebenen Daten oder Voraussetzungen zu ziehen. Damit kann es auf intelligente Art Rat geben oder gar intelligente Entscheidungen treffen. Dieser auf Wissen basierende Systemaufbau stellt einen grundlegenden Evolutions-Wechsel in der Computerwissenschaft dar. Die Konsequenzen sind revolutionär. Die Methode ersetzt die traditionelle Formel „Daten + Algorithmen = Programm" durch eine neue Architektur, konstruiert rund um eine „Wissens-(Informations-)Basis" und eine „Inference Engine" (Schlußfolgerungsmaschine). Ergebnis: Wissen + Schlußfolgerung = Expertensystem. Diese Formel scheint auf den ersten Blick sehr ähnlich zu sein. Die Unterschiede sind jedoch so wesentlich, daß die Formel mit Erfolg für komplizierte Zusammenhänge verwendet werden kann.

Systeme, die das Wissen von Experten aufgreifen und benutzen können, um Rat zu geben oder Diagnosen zu stellen, werden in vielen Bereichen immer populärer – ob in der Medizin, in der Landwirtschaft oder der Architektur. Expertensysteme, die zur Anwendung in einem relativ kleinen Forschungsbereich geschaffen wurden, sind eine wertvolle Unterstützung bei der täglichen Arbeit.

Kriterien eines Expertensystems

Was ist ein Expertensystem? Die folgende Checkliste typischer Merkmale ist bei der Definition nützlich:

- Ein Expertensystem ist auf einen bestimmten, relativ kleinen Wissensbereich beschränkt.
- Es sollte unsichere Daten und unzuverlässige Regeln „abarbeiten" können.
- Es muß imstande sein, seine Schlußfolgerungen nachvollziehbar zu erläutern.
- Fakten und Entscheidungs-Mechanismen lassen sich „trennen": Wissen ist also nicht „fest gespeichert".
- Es ist auf ständiges Wachstum (Erweiterungen) ausgerichtet.

Ein Expertensystem enthält mehrere Module, die das Wissen vom Experten zum Benutzer leiten. Zunächst muß Wissen vom Experten (oder von mehreren Experten) eingegeben werden und in eine Wissensbasis umgewandelt werden. Um Vorhersagen treffen zu können, Rat zu erteilen oder eine Diagnose zu erstellen, müssen dann Schlußfolgerungen innerhalb des Systems möglich sein. Die Erklärungs-Schnittstelle erlaubt schließlich dem Benutzer die Kommunikation mit dem System und „berät" ihn.

- Es basiert auf festen Regeln.
- Als Ausgabe wird ein Ratschlag oder Hinweis geliefert, nicht Zahlen oder Grafiken.

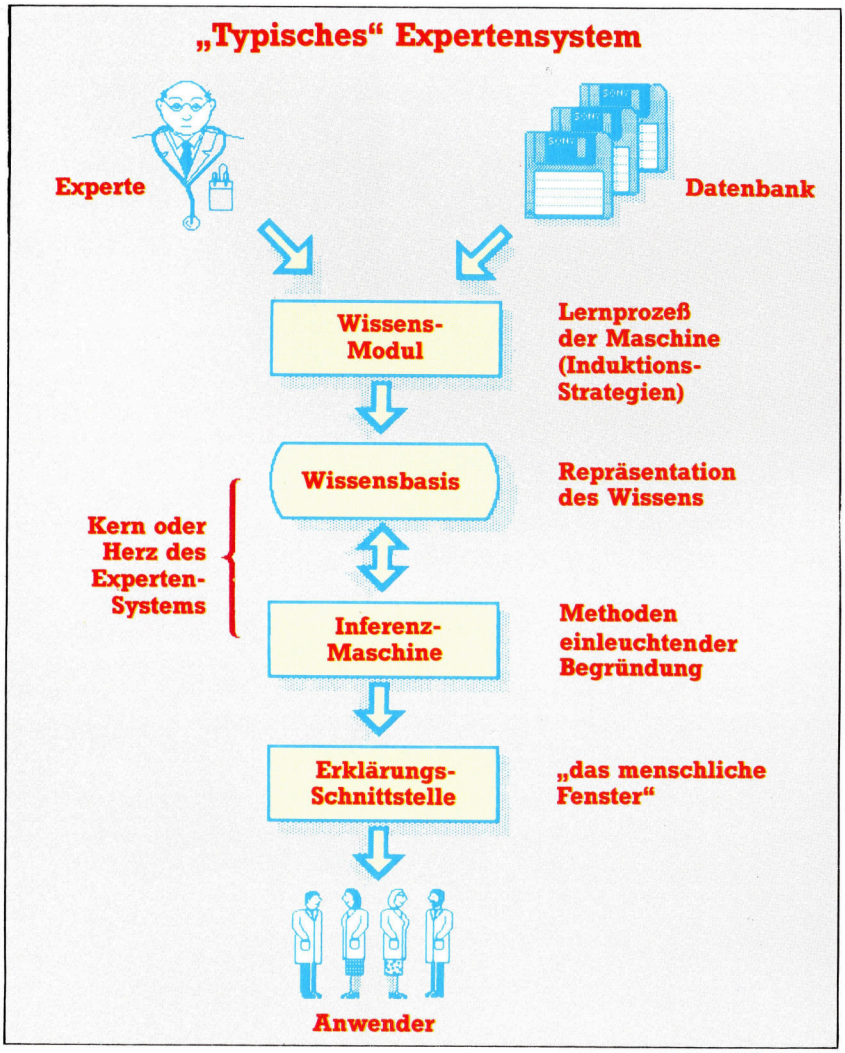
Das Schlüsselwort heißt Wissen. Ziel eines intelligenten Problemlösungssystems ist die Beseitigung von Blind- oder Zufalls-Suche. Ein Computersystem muß also über dieselben Vorteile verfügen wie ein menschlicher Experte gegenüber einem Laien – das sind Erfahrung oder organisiertes Wissen: Wissen über Fakten, über Schlußfolgerungsregeln und über Lösungsstrategien. Ein komplettes Expertensystem besteht aus vier Komponenten:

1. Die Wissensbasis
2. Die „Schlußfolgerungsmaschine" (Inferenzmaschine)
3. Das Wissens-Erweiterungs-Modul
4. Die Erklärungs-Schnittstelle

Alle vier Module sind problematisch. Ein allein auf Wissen basierendes System kann auf das eine oder andere davon verzichten, ein echtes Expertensystem aber nicht.

Die beiden fundamentalen Komponenten eines Expertensystems sind die Wissensbasis und die Inferenzmaschine. In der Wissensbasis werden die Informationen des Hauptthemenbereichs gespeichert. Allerdings handelt es sich dabei nicht um einen passiven Satz von Aufzeichnungen und Daten, wie sie in jeder Datenbank enthalten sind, sondern um symbolische Darstellungen von Experten-Regeln.

Die meisten Bestandteile einer Wissensbasis sind nicht mathematisch. Die Hauptschwierigkeiten bei der Entwicklung einer Wissensbasis sind Wissensdarstellung und Wissenszugriff. Grundsätzlich müssen die folgenden Elemente enthalten sein: Grundbegriffe (die von den jeweiligen Experten des Bereichs verwendete Terminologie), strukturelle Verwandtschaft (die Beziehungen der Komponenten untereinander) und ursächliche Verwandtschaften (die Ursachen-Wirkungs-Verhältnisse der Komponenten).



Vier Speichermöglichkeiten

Die Aufgabe des bearbeitenden Ingenieurs besteht darin, geeignete symbolische Speichermöglichkeiten für diese Informationen zu finden. Vier grundlegende Methoden haben sich herauskristallisiert:

- Regeln im IF. . .THEN-Format. Diese Bedingung schreibt einen Ablauf vor, und die Folgerung kann eine Aktion oder eine Aussage sein.
- Semantische Netze. Diese stellen Beziehungen zwischen Objekten durch Verbindungen zwischen Knoten dar.
- Rahmen. Hierbei handelt es sich um Aufzeichnungsstrukturen, die Grenzwerte enthalten können, aber auch Aktionen, die als Werte bestimmter Felder oder Sparten codiert sind.
- „Horn-Sätze". Eine Form von Aussage-Logik, auf der PROLOG basiert und mit der PROLOG Inferenzen durchführen kann.

Frühe Expertensysteme verwendeten fast ausschließlich den auf Gesetzen basierenden Formalismus. Ein Gesetz des Mycin-Systems für die Diagnose von Blutinfektionen ist typisch für die IF. . .THEN-Struktur:

WENN:
1. die eine Therapie erfordernde Infektion Meningitis ist und
2. die Art der Infektion ein Pilz ist (fungal) und
3. Organismen in der Kultur nicht zu sehen waren und
4. der Patient nicht als „Wirt" gefährdet ist und
5. der Patient in einer Gegend war, in der Coccidiomyzosen regional verbreitet (endemisch) sind und
6. der Patient Schwarzer oder Asiate ist und
7. das kryptococcale Antigen im CSF nicht positiv war

DANN
ist zu vermuten, daß ein Cryptococcus nicht zu den Organismen gehört, die die Infektion verursacht haben.

Aus diesem Beispiel geht hervor, daß in einem Expertensystem Fachausdrücke verwendet werden, die im betreffenden Bereich Anwendung finden – in diesem Fall in der Medizin. Die im Mycin verwendete IF. . .THEN-Struktur besteht aus einer Reihe von Aussa-

gen, die als wahr oder falsch gewertet werden können. Um die für die Erstellung der Diagnose erforderliche Information einholen zu können, muß Mycin in einen Dialog mit dem Anwender treten. Der Benutzer muß bei diesem System zumindest über gewisse Grundkenntnisse des betreffenden Bereichs verfügen, so daß die Fragen des Expertensystems verstanden und beantwortet werden können.

Der Inferenz-Mechanismus besteht aus Such- und Begründungs-Methoden, die das System befähigen, Lösungen und, falls erforderlich, auch Rechtfertigungen für seine Antworten zu geben. Dabei ist zwischen zwei wichtigen Begründungsstrategien zu unterscheiden – der Vorwärts-Kette und der Rückwärts-Kette.

Die Vorwärts-Kette führt vom Ereignis (oder den Symptomen) zu Schlußfolgerungen (der Diagnose). In einem auf Gesetzen basierenden System bedeutet das ein Vergleichen der IF-Konditionen mit den Fakten. Diese Art der Verkettung ist leicht zu codieren und für all jene Anwendungen geeignet, in denen ohnehin Daten gesammelt werden müssen.

Die rückwärtige Verknüpfung arbeitet von der Hypothese zum Ereignis. Das System stellt eine Hypothese (Behauptung) auf und sucht nach Daten, die sie stützen oder widerlegen. Sie kann in recursiver oder in konsultierender Form programmiert sein. Letztere führt dabei zu einem natürlichen Dialog. Das Problem, welche Hypothese in einer bestimmten Situation aufgestellt werden sollte, ist noch nicht völlig gelöst. In der Praxis werden daher meist Vor- und Rückwärts-Verknüpfungen miteinander kombiniert.

Wissenszugriff als Engpaß

Experten sind selten imstande zu erläutern, wie sie zu ihren Schlußfolgerungen kommen – nicht etwa, weil sie ihre Geheimnisse für sich behalten wollen, sondern weil ein Teil des Entscheidungsfindungsprozesses intuitiv abläuft, also nicht bewußt ist. Der Wissenszugriff wird deshalb als eigentlicher Engpaß bei der Entwicklung von Expertensystemen betrachtet. Experten sind aber gute Kritiker. Wenn ihnen ein Fall vorgelegt wird, könnten sie sagen, welche Entscheidung zu treffen wäre, und würden, falls nötig, den Lösungsvorschlag des Computers kritisieren und verbessern.

Einer der Vorteile von Mycin, Prospector und vielen klassischen Expertensystemen war, daß sie ihre Schlußfolgerungen auf Anfrage begründen konnten. Hier folgt ein Beispiel, wie Mycin auf eine WHY?-Frage antwortet, die von einem Arzt gestellt wurde. Der Dialog beginnt mit der Datenabfrage von Mycin.

Ist Meningitis eine Infektion, deren Übertragung im Krankenhaus erfolgt?
WARUM?
(2.0) Dies hilft bei der Bestimmung der Organismen (andere als die in Kulturen oder Schichten feststellbaren), die die Infektion verursacht haben könnten. Bestimmt wurde bereits, daß:
(2.1) die zu behandelnde Infektion Meningitis ist und daß
(2.2) Organismen in dieser Phase der CSF-Kultur nicht zu sehen waren und daß
(2.3) die Art der Infektion bakteriell ist und daß
(2.4) Neurochirurgie in der Krankengeschichte des Patienten keine Rolle spielte und daß
(2.5) der Patient älter als zehn Tage ist.
Wenn also:
(2.6) die Infektion während eines Krankenhausaufenthalts des Patienten erfolgte,
ist anzunehmen, daß die Organismen (andere als die in Kulturen oder Schichten feststellbaren), die die Infektion verursacht haben könnten, E. coli (0.75), Staphylococcen koag-pos (0.3), Pseudomonas aeruginosa (0.3) und Klebsiella pneumoniae (0.5) sind – laut (Regel 545).

Es ist zu beachten, daß die Schlußfolgerungen numerisch gewichtet sind. Es handelt sich hier aber nicht um echte Wahrscheinlichkeiten. Diese Gewichtungen erlauben dem System lediglich, Unsicherheiten zu berücksichtigen und in der Schlußanalyse eine geordnete Liste möglicher Diagnosen aufzustellen. Danach sind weitere Schlußfolgerungen möglich.

Seien Sie gut beraten . . .

Der Zugang zum „Counsellor" erfolgt in England über das „Prestel"-System . . .

Die Einführung erlaubt dem Benutzer die Wahl zwischen Sommer- und Herbst-Planungshilfen . . .

Daraufhin stellt das System dem Benutzer Fragen, deren Beantwortung Basis für die Beratung ist . . .

Nach Interpretation der Antworten des Landwirts erteilt der „Weizen-Counsellor" Rat in Form von Werten für die verschiedenen Düngemittel.

Der Weizen-Counsellor ist ein Expertensystem, das von der ICI entwickelt wurde und den englischen Landwirten über Prestel kostenlos zur Verfügung steht. Das System wurde entwickelt, um Landwirten bei der Schädlingsbekämpfung zu helfen, die dafür erforderlichen Chemikalien zu ermitteln und die möglichen Verluste zu errechnen. Wenngleich das System in Form eines Gespräches, wie es über den Zaun zum Nachbarn geführt werden würde, angelegt ist, basieren seine Kenntnisse auf wissenschaftlichen Ergebnissen. Um die gewünschte Information zu bekommen, stellt das System dem Landwirt einfache Fragen. Die Beratung erfolgt in Gestalt von Vorschlägen für bestimmte Maßnahmen mit Angaben über den zu erwartenden Erfolg.

Prolog-Variablen

Die Programmiersprache PROLOG löst Probleme, indem sie aus einer „Datenbank" sogenannte „Tatsachen" abruft und sie auf Behauptungen anwendet. In diesem Artikel analysieren wir unter anderem das „Backtracking".

Zur Lösung einer Aufgabe benötigt PROLOG eine Liste von Tatsachen, die die logische Struktur des Problems beschreiben, und entsprechende „if-then"-Regeln. Mit dieser „erklärenden" (deklarativen) Beschreibung können die Folgerungsmechanismen der Sprache alle Antworten auf mögliche Fragestellungen finden. Der Programmierer muß dem Computer nicht mehr bis in die letzten Einzelheiten vorschreiben, wie er die Sprache einsetzen soll (um ein Problem zu lösen), sondern gibt nur klar und logisch die Tatsachen an, die zur Lösung des Problems nötig sind. Den Rest erledigt PROLOG.

PROLOG hat zwar eine sehr einfache Syntax, verwendet jedoch recht obskure Fachbegriffe. Wichtigstes Sprachelement ist der „Ausdruck" (englisch: Term). Ausdrücke können „Konstanten" sein (wie baum, klaus oder 25), Variablen oder Strukturen. Oft wird auch das Strukturelement „Tatsache" (englisch: fact) verwandt. Tatsachen bestehen aus einer Aussage (muß eine Konstante sein), die entweder allein steht oder von einer in Klammern eingeschlossenen Argumentenliste (Konstanten oder Variablen) gefolgt wird.

```
aussage1.
aussage 2(argument1, argument2,
    argument3).
```

In der ersten Folge dieser Serie wurde erwähnt, daß sich eine Aussage als die Beziehung zwischen ihren Argumenten verstehen läßt. In dem Satz „Erdbewohner essen Brot" ist „essen" die Aussage, die die Beziehung zwischen Erdbewohnern und Brot beschreibt. In PROLOG sieht diese Tatsache folgendermaßen aus:

```
essen(erdbewohner, brot).
```

Aussagen dieser Art können wie die Datensätze einer Datei zu langen Listen zusammengestellt werden. Der folgende Kasten beschreibt mit einer sogenannten Tatsachendatei, wer welche Dinge ißt.

Wer ißt was?

```
essen (saturner, muesli).
essen (erdbewohner, brot).
essen (venusianer, steine).
essen (merkurianer, kaese).
essen (neptunianer, muesli).
essen (marsmenschen, marsmenschen).
```

„Tatsachen" werden in PROLOG als Aussage (in diesem Fall „essen") angegeben, die allein stehen kann oder von einer Liste von Argumenten gefolgt wird. Sie versehen das Programm mit einer „Tatsachendatei", die vom Anwender abgefragt werden kann.

Für den Ablauf des Programms wird der Interpreter mit der Lösung der Aufgabe betraut. Wenn Sie wissen möchten, ob Venusianer Steine essen, geben Sie hinter dem Systemprompt (?—) folgende Frage ein:

```
?— essen(venusianer, steine).
```

PROLOG durchsucht nun seine Tatsachen. Wird eine Übereinstimmung gefunden, erscheint ein „ja": Das Programm kann beweisen, daß „essen (venusianer, steine)" wahr ist. Die Frage

```
?— essen (merkurianer, steine).
```

beantwortet PROLOG mit „nein".

Durch den Einsatz von Variablen werden die Programme flexibler. PROLOG hat folgendes Standardformat: Variablennamen fangen mit einem Großbuchstaben an, während normale PROLOG-Konstanten mit Kleinbuchstaben beginnen. Wenn Sie wissen wollen, wer Käse ißt, fragen Sie:

```
?— essen(Wesen, kaese).
```

PROLOG antwortet dann:

```
Wesen = merkurianer
```

und wartet auf eine neue Eingabe. PROLOG hat damit herausgefunden, daß die Behauptung bewiesen werden kann, wenn die Variable Wesen auf den Wert merkurianer gesetzt wird.

An dieser Stelle geben Sie entweder RETURN ein (wenn Ihnen die Antwort genügt) oder teilen PROLOG durch ein Semikolon mit, daß der Vorschlag auf andere Weise bewiesen werden soll. Da es in diesem Fall keinen anderen Weg gibt, erscheint nun das „nein". Die Frage, wer Müsli ißt, hat jedoch zwei mögliche Antworten: saturner und neptunianer. Auf die Frage:

```
?— essen (Wesen, muesli).
```

sagt PROLOG:

```
Wesen = saturner
```

und nach Eingabe eines Semikolons:

```
Wesen = neptunianer
```

Wenn es für einen Vorschlag mehrere Lösungen gibt, bewirkt das Semikolon, daß alle Lösungen ausgegeben werden. In der nächsten Folge gehen wir genauer auf diesen Ablauf ein. Hier wollen wir noch andere wichtige Konzepte von PROLOG vorstellen — „Regeln", „Folgerungsketten" und „Backtracking".

Ein oder mehrere Ausdrücke werden zu einem „Satz", der eine „Regel" beschreiben kann. Ein Satz besteht aus einem Kopf, gefolgt von einem Hauptteil, der einen oder mehrere Ausdrücke enthalten kann. Kopf und Hauptteil werden durch das Symbol :— dargestellt, das normalerweise als „wenn" (if) gelesen wird:

```
ausdruck1:— ausdruck2, ausdruck3,
    ausdruck4.
```

Die Kommas zwischen den Ausdrükken des Hauptteils lassen sich als logisches AND verstehen. Unser Beispiel zeigt, „daß ausdruck1 gültig ist, wenn ausdruck2 AND ausdruck3 AND ausdruck4 wahr sind".

Um dies zu verdeutlichen, wollen wir herausfinden, ob unsere Datei Kannibalen enthält. Wir schreiben daher einen Satz, der Kannibalen definiert:

```
kannibale(Wesen):—
essen (Wesen, Wesen).
```

Der Hauptteil dieses Satzes enthält einen Ausdruck, der folgende „Regel" beschreibt: Ein Kannibale ist ein Wesen, das die gleiche Art von Wesen ißt.

Fügen Sie diesen Satz in das Programm ein und fragen Sie:

```
?— kannibale(X).
```

PROLOG vergleicht diese Behauptung mit dem Kopf des neuen Satzes.

Da der neue Satz aber nicht automatisch wahr zu sein braucht, muß zunächst bewiesen werden, daß die Ausdrücke im Hauptteil des Satzes wahr sind. PROLOG nimmt daher jede Aussage von links nach rechts nacheinander als Behauptung. In diesem Fall enthält der Hauptteil jedoch nur einen Ausdruck

. . .,essen (Wesen, Wesen).

der mit der Tatsache essen (marsmenschen, marsmenschen) übereinstimmt. Nun wird Wesen = marsmenschen gesetzt, was bedeutet, daß kannibale(marsmenschen) bewiesen werden kann. Da dadurch X = marsmenschen gesetzt wird (X ist der Variablenname der ursprünglichen Frage), kann PROLOG daraufhin antworten:

X = marsmenschen

In diesem Fall wurde essen (marsmenschen, marsmenschen) als Tatsache in der Datei gefunden. Wenn PROLOG dort jedoch keine einfache Tatsache, sondern eine weitere Regel gefunden hätte, müßte die Sprache sich weitere (Unter-) Vorschläge setzen, um den Kopf der Regel beweisen zu können.

Der nächste Kasten zeigt, was in diesem Fall passiert. Bei der Frage:

?— farbe_von(martin,Farbe).

die die Farbe von martin herausfinden soll, wird zunächst die Frage als Behauptung angenommen. Nachdem PROLOG eine Regel gefunden hat, die entscheidet, ob farbe_von(martin, rosa) wahr ist, findet sie heraus, daß dies immer eintritt, „wenn" (:—) martin in einer glücklichen Stimmung ist. PROLOG nimmt dies nun als den nächsten Vorschlag und entdeckt eine Regel, die besagt, daß martin glücklich ist, wenn er in PROLOG programmieren kann. kann_programmieren_in(martin,prolog) wird daher die nächste zu beweisende Behauptung.

Wann sind Marsmenschen blau?

```
farbe_von(Marsmensch,rosa):—
    stimmung_von (Marsmensch,gluecklich).
farbe_von(Marsmensch,blau) .
stimmung_von(Marsmensch,gluecklich):—
    kann_programmieren_in(Marsmensch,prolog).
stimmung_von(Marsmensch,traurig).

kann_programmieren_in(martin,basic).
kann_programmieren_in(louise,prolog).
```

Beachten Sie, daß in dieser Liste von Ausdrücken alle Variablennamen mit Großbuchstaben anfangen. Das Symbol :— kann als logisches „wenn" (if) interpretiert werden.

Dieser Vorgang kann auf vielen Ebenen fortgeführt werden. In unserem Beispiel endet er hier, da es keine Regel oder Tatsache gibt, die beweist (oder zeigt, wie bewiesen werden kann), daß martin ein PROLOG-Programmierer ist.

An diesem Punkt gibt PROLOG jedoch nicht einfach auf. Die Sprache gesteht zwar ein, daß sie mit dem letzten Vorschlag nicht weiterkommt, kehrt aber auf den vorhergehenden Vorschlag zurück, um zu prüfen, ob er auf einem anderen Weg bewiesen werden kann. Da auch dieser Versuch fehlschlagen muß (es kann nicht festgestellt werden, daß die Stimmung von martin glücklich ist), geht PROLOG noch einen Schritt zurück („backtracks"), um weitere Alternativen zu testen. PROLOG findet die Tatsache farbe von(Marsmensch, blau), setzt Marsmensch = martin und schließt damit die Aufgabe erfolgreich ab.

Durch die Verfolgung einer langen Kette von Regeln kann PROLOG auf alle Fragen eine Antwort finden. Die Methode heißt „Backtracking", da PROLOG seinen Weg bis zu einem früheren Verzweigungspunkt zurückverfolgt, wenn ein bestimmter Pfad keine Lösung bringt.

Das Alvey-Programm

Bei dem Alvey-Forschungsprojekt zur Entwicklung der Fünften Computergeneration spielt PROLOG eine zentrale Rolle. Das Projekt wurde 1983 nach einem Bericht des Alvey-Komitees ins Leben gerufen und war die Antwort auf eine japanische Initiative. Das Komitee sollte erforschen, in welchen Bereichen bei der Entwicklung der Fünften Computergeneration eine Zusammenarbeit möglich wäre.

Das Alvey-Projekt war eng mit ESPRIT verbunden, dem europäischen Projekt zur Entwicklung der neuen Computergeneration, und unterhielt auch Kontakte zu den Japanern.

Etwa 100 Einzelprojekte wurden bewilligt. Sie mußten mit der Gesamtstrategie übereinstimmen und wurden jedes Jahr von neuem bewertet und aktualisiert.

Im Bereich der PROLOG-Forschung sollte mit der existierenden Technik eine PROLOG-Maschine konstruiert werden, deren Preis wesentlich unter dem eines typischen Microcomputers lag und die sich durch die Eigenschaften von PROLOGs „intelligenter Datenbank" direkt als Expertensystem einsetzen lassen würde.

Andere Anwendungen zielten auf den Einsatz von PROLOG als „natürliche Sprachschnittstelle". PROLOG eignet sich besonders gut für die Verarbeitung natürlicher Sprachelemente. Da gerade dieser Bereich einer der wesentlichen Aspekte der „benutzerfreundlichen" Maschinen der Fünften Computergeneration ist, scheint PROLOG seine zentrale Rolle in der Entwicklung dieser neuen Technologie auch weiterhin zu behaupten.

Der Vater des Gedankens
Professor Bob Kowalski wird oft als „Vater der logischen Programmierung" bezeichnet. Seine Ideen wurden von A. Colmerauer übernommen, der den ersten PROLOG-Interpreter zusammenstellte.

Micro-sensoren

Die vorige Folge unseres Selbstbau-Kurses führte bis zur Montage der ersten Ausbaustufe des Roboters. Jetzt soll das Robot-Auto vier Microtaster erhalten, deren Funktion sich einfach prüfen läßt.

Die bidirektionale Steuerung der beiden Schrittmotoren belegt vier der acht Datenleitungen am User Port. Also bleiben noch vier Leitungen für den Anschluß von Sensoren. Durch die Konstruktion eines Steckerfeldes können unterschiedliche Sensor-Kombinationen mit dem User Port verbunden werden, damit der Roboter möglichst vielseitig wird. Zu Beginn wollen wir vier Berührungs-Taster einbauen, zu denen später noch zwei Lichtsensoren hinzukommen sollen. Zum leichteren Umschalten – etwa auf zwei Licht- und zwei Berührungssensoren – bekommt jeder Sensor eine eigene Buchse auf dem Robot-Gehäuse. Die vier Buchsen (D4 bis D7) leiten die Taster-Werte zum User Port weiter. Zum Verändern der Sensor-Konfiguration muß also nur das Verbindungskabel umgesteckt werden.

Die Verdrahtung der vier Sensoren kann durch ein einfaches Programm geprüft werden, das die vier höherwertigen Bits im User-Port-Register liest und ihre dezimalen Werte anzeigt. Beim Programmablauf sollten alle vier Sensoren über das Steckerfeld mit den Datenleitungen D4–D7 verbunden sein. Beim Schließen eines Schalters ändert sich sofort die Anzeige auf dem Bildschirm.

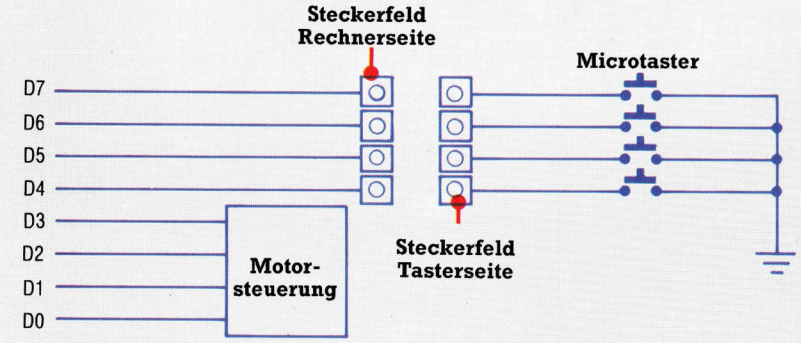
Einbau der Microtaster

Die verwendeten Microtaster sind in verschiedenen Ausführungen im Elektronik-Fachhandel erhältlich. Zur Montage müssen Sie an der Oberseite des Robot-Gehäuses acht Schlitze einschneiden. Die Anschlußfahnen der Microtaster sollten gerade hindurchpassen. Dann werden zehn Löcher mit einem Durchmesser von 5 mm gebohrt – sie nehmen die Buchsen für das Steckerfeld auf und liegen neben dem D-Anschluß. Die Microtaster müssen etwas verändert werden: Biegen sie dazu den langen Tasthebel mit zwei kleinen Zangen rechtwinklig ab. Aber Vorsicht – der Winkel darf nicht so nah am Tastergehäuse liegen, daß sich der Schaltkontakt nach dem Einbau nicht mehr schließen läßt! An der Rückseite des Microtasters befinden sich zwei Anschlußfahnen. Die obere ist der Ruhekontakt (NC=bei nicht gedrücktem Taster geschlossen). Er wird nicht gebraucht, Sie können ihn abbrechen oder absägen. Wir verwenden nur den Arbeitskontakt (NO=bei nicht gedrücktem Schalter offen). Die Anschlußfahne wird rechtwinklig abgebogen, so daß sie sich zusammen mit dem Mittelanschluß (COM) durch die Schlitze im Gehäuse stecken läßt. Die vorbereiteten Taster sollten so am Gehäuse montiert werden, daß die umgebogenen Tasthebel an der Vorder- und Rückseite des Robot-Gehäuses etwas vorstehen. Die Tastergehäuse lassen sich am besten mit einem „Superkleber" an den vier Ecken des Roboters befestigen. Damit das Gehäuse die Tasthebel der Schalter nach der Montage nicht blockiert, sollten Sie alle Einbauteile vor dem endgültigen Zusammenbau in einem Trockengang ohne Klebstoff provisorisch zusammensetzen.

```
10 REM **** BBC SENSOR TEST ****
20 MODE 7:OP=-1:DDR=&FE62:DATREG=&FE60:?DDR=15
30 PE=240-(?DATREG AND 240):IF PE=OP THEN 30
40 CLS:PRINT PE:OP=PE:GOTO 30
```

```
10 REM **** CBM SENSOR TEST ****
   OP=-1:DDR=56579:DATREG=56577:POKE DDR,15
   PE=240-(PEEK(DATREG) AND 240):IF PE=OP THEN 30
   PRINT CHR$(147):PRINT PE:OP=PE:GOTO 30
```

Schaltplan für Microtaster

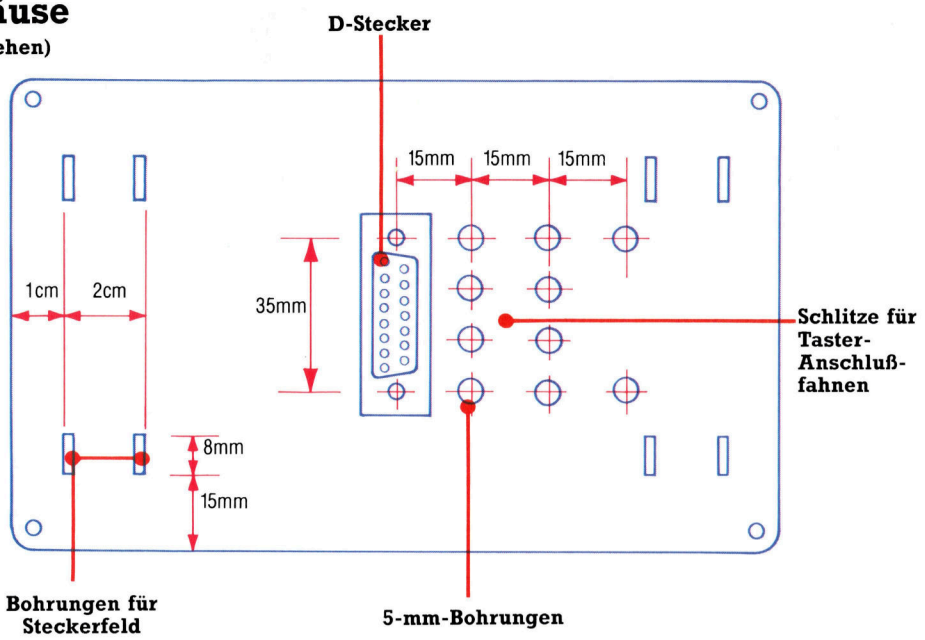


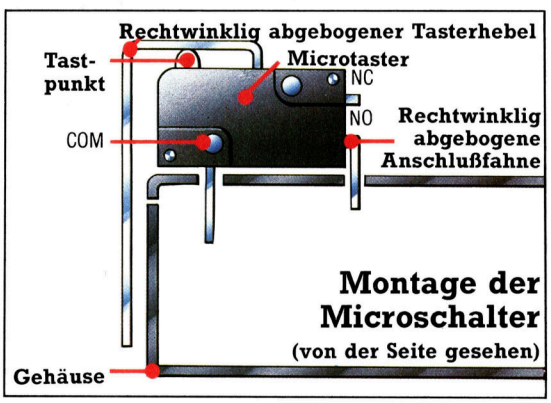
Anschluß der Microtaster

Die Verbindung der Microtaster mit dem Computer macht keine besonderen Schwierigkeiten: Die Datenleitungen D4 bis D7 sind mit den Buchsen am Steckerfeld verbunden, die Leitungen D0 bis D3 dienen zur Motorsteuerung. Je ein Taster-Kontakt ist an einer Buchse des Steckerfeldes angeschlossen, die anderen Kontakte werden mit einer gemeinsamen Masseleitung verbunden. Für den Einsatz aller vier Taster brauchen Sie vier kurze Verbindungskabel für das Steckerfeld. Wenn die vier oberen Bits im Datenregister auf Eingabe geschaltet sind, werden sie vom Rechner auf High (Eins) gehalten. Beim Schließen eines Tasters wird die dazugehörige Datenleitung mit Masse verbunden – das Bit geht auf Low (Null).

Schlitze und Bohrungen am Gehäuse

(von oben gesehen)



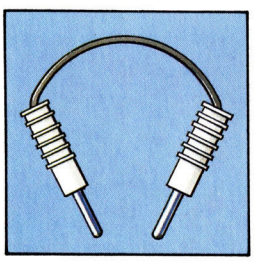


Liste der Bauteile

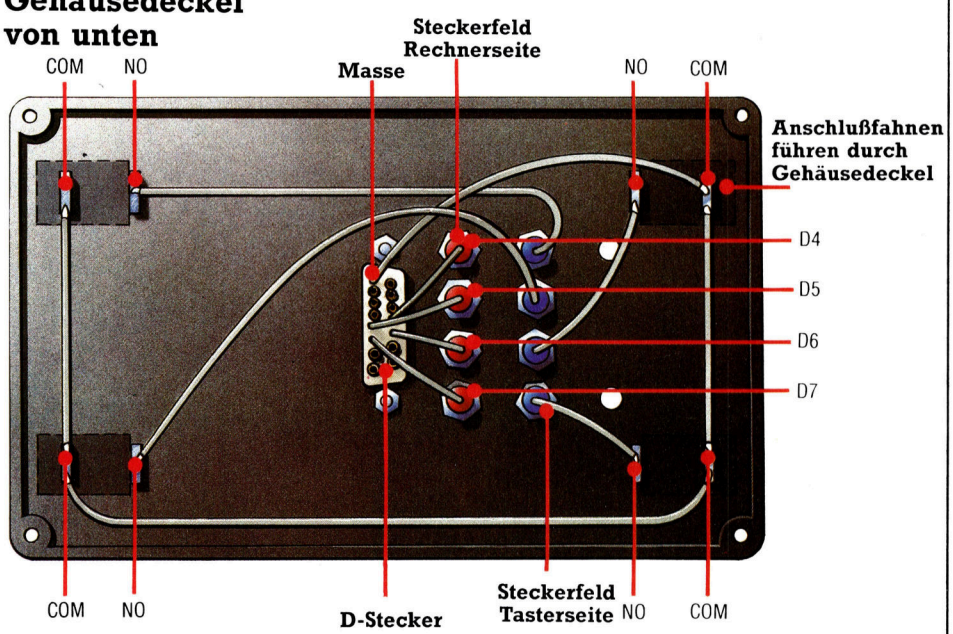
Anzahl	Bauteil
8	2-mm-Stecker
4	2-mm-Buchsen, rot
6	2-mm-Buchsen, blau
4	Microtaster
1 Meter	4poliges Flachkabel

Die Form der Tastergehäuse ist unwichtig. Achten Sie aber darauf, daß der bewegliche Tasthebel lang genug ist. Bei anderen Taster-Typen können Sie sich auch mit einem angelöteten Blechstreifen behelfen.

Isolierte Litze führt von den vier roten Buchsen zum D-Stecker. Die vier Mittelanschlüsse der Taster (COM) werden mit einer gemeinsamen Leitung verbunden, die zum Masseanschluß des D-Steckers führt. Die Arbeitskontakte der Taster werden an die blauen Buchsen des Steckerfeldes angeschlossen. Vier Kabelbrücken stellen die Verbindung her.



Gehäusedeckel von unten



Objekte ablegen

Im letzten Abschnitt unseres Abenteuerspiel-Projekts haben wir Routinen entwickelt, die das Aufnehmen von Objekten ermöglichen. Jetzt werden die Routinen entworfen, die das Ablegen dieser vielfältigen Gegenstände gestatten.

Die DROP-Unterroutine ist der TAKE-Routine in einigen Punkten ähnlich. Daher können wir die Prüfroutinen für die Objekte übernehmen. Insgesamt werden drei Tests in der TAKE-Routine durchgeführt. Beim ersten Test wird überprüft, ob der zweite Teil des Befehls ein gültiges Objekt enthält. Zu diesem Zweck wird jedes Wort der Eingabe systematisch mit den Objektnamen im Inhaltsverzeichnis IV$(,) verglichen. Trifft ein Vergleich zu, wird eine Variable F auf den Wert der Objektposition in IV$(,) gesetzt. Dieser Gültigkeitstest muß auch in der DROP-Routine verwendet werden, um festzustellen, daß das Objekt existiert und welche Position es im Inhaltsverzeichnis hat.

Auch der zweite Test der TAKE-Routine wird in der DROP-Routine benötigt. Dabei wird überprüft, ob der Spieler das angegebene Objekt bei sich trägt oder nicht (in IC$()). Schließlich kann kein Objekt abgelegt werden, das man nicht besitzt. Der dritte Test der TAKE-Routine sollte gewährleisten, daß sich das entsprechende Objekt auch an der aktuellen Position des Spielers befand. Da ein abzulegendes Objekt jedoch vom Spieler getragen wird, somit seine Position nicht im Hauptinhaltsverzeichnis stehen kann, wird dieser Test in der DROP-Routine nicht benötigt.

Nimmt man an, daß beide Tests positiv verlaufen, müssen die folgenden Änderungen in den beiden Inhaltsverzeichnissen durchgeführt werden:

1) Die Position des abzulegenden Objekts wird durch F angegeben. Die aktuelle Position P ist somit in das Hauptinhaltsverzeichnis an der Position IV$(F,2) einzutragen.

2) Die Objektbeschreibung muß aus dem Inhaltsverzeichnis des Spielers, IC$(), gelöscht werden. Hierzu wird das Array durchsucht, und die entsprechenden Zeichen werden mit Leerstellen überschrieben.

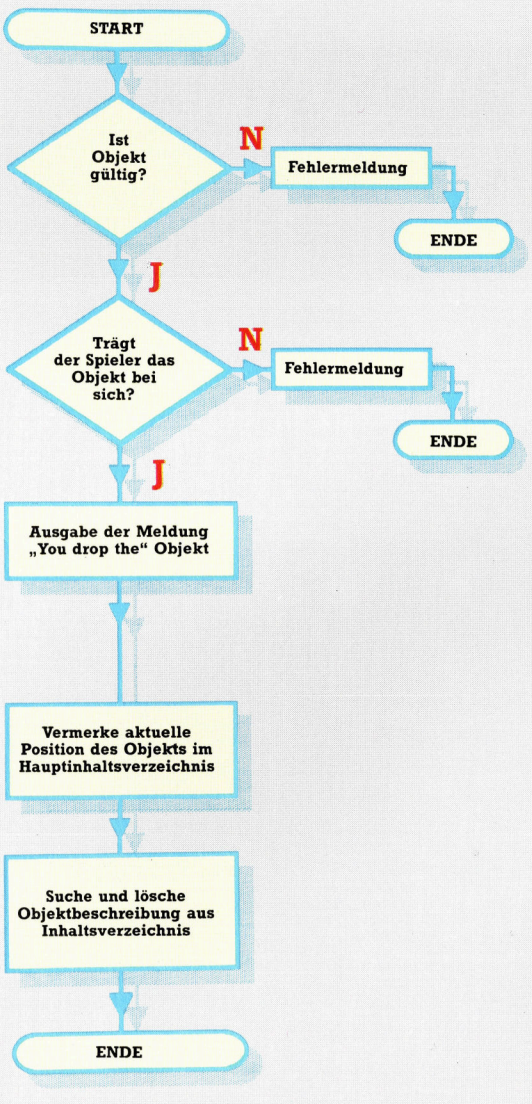
Der logische Ablauf der DROP-Routine ist aus dem Flußdiagramm ersichtlich. Im folgenden finden Sie das Listing für Haunted Forest:

```
3900 REM **** DROP S/R ****
3910 GOSUB5300:REM VALID OBJECT
3920 IF F=0 THEN SN$="THERE IS NO "+W$:GOSUB5500:
RETURN
3930 :
3940 REM ** IS OBJECT IN CARRIED INVENTORY **
3950 OV=F:GOSUB5450
3960 IF HF=0 THEN SN$="YOU DO NOT HAVE THE "+IV$
(F,1):GOSUB5500:RETURN
3970 :
3980 REM ** DROP OBJECT **
3990 SN$="YOU DROP THE "+IV$(F,1):GOSUB5500
4000 IV$(F,2)=STR$(P):REM MAKE ENTRY IN INVENTORY
4010 :
4020 REM ** DELETE OBJECT FROM CARRIED INVENTORY
**
4030 FOR J=1TO2
4040 IF IC$(J)=IV$(F,1) THEN IC$(J)="":J=2
4050 NEXT J
4060 RETURN
```

Hier wird wieder der Vorteil der Programmierung in Modulen deutlich. Eine Routine kann für verschiedene Zwecke eingesetzt werden. Verwendet man ein System mit Flags, können

Das Ablegen

Die Logik der DROP-Routine ähnelt der der TAKE-Routine. Vor der Befehlsausführung müssen jedoch nur der Gültigkeitstest sowie das Vorhandensein des Objekts im Inhaltsverzeichnis überprüft werden. Das Hauptinhaltsverzeichnis wird dann aktualisiert und der Objektname aus dem Inhaltsverzeichnis gelöscht.



innerhalb der Unterroutinen Entscheidungen getroffen werden, deren Ausführung jedoch erst nach Rückkehr in die Routine erfolgt, von der aus die Unterroutine aufgerufen wurde. Ein gutes Beispiel für diese Programmstruktur ist der Gültigkeitstest. Er wird von der TAKE- und der DROP-Routine aufgerufen. In beiden Fällen wird eine Entscheidung über die Gültigkeit eines Objekts getroffen. Der Programmlauf wird jedoch nicht unterbrochen, bis der Rücksprung zur TAKE- bzw. DROP-Routine erfolgt. Dort wird der Wert des Flags F überprüft und die entsprechende Verzweigung durchgeführt. Ein Nachteil ist die doppelte Abfrage derselben Kondition, wodurch sich die Verarbeitungsgeschwindigkeit leicht verringert. Doch dürfte die größere Flexibilität und einfachere Fehlersuche diesen Nachteil ohne weiteres ausgleichen.

Die „speziellen" Orte

Nun sind wir an dem Punkt angekommen, wo die Programmierung des Programmgerüsts beendet ist – der Spieler kann nun Objekte tragen und sich in der Abenteuerwelt bewegen. Jetzt befassen wir uns mit den „speziellen" Orten, an denen Objekte verwendet, Gefahren gemeistert und die Erfahrungen des Spielers auf die Probe gestellt werden.

Bevor wir uns im Detail mit der Programmierung dieser Routinen befassen, sind an der Hauptschleife noch Änderungen vorzunehmen, um spezielle Orte zu erkennen. Folgende zwei Programmzeilen müssen in das Programm eingefügt werden:

```
257 GOSUB2700:REM IS P SPECIAL ?
258 IF SF=1 THEN 300:REM NEXT INSTRUCTION
```

Zeile 257 ruft eine Unterroutine auf, in der überprüft wird, ob ein Ort speziell ist. Wenn ja, wird das Flag SF auf 1 gesetzt. Das heißt, daß der Befehlsteil umgangen werden kann, sobald wieder zur Hauptschleife zurückverzweigt wird. Die Unterroutine, die bestimmt, ob ein Ort speziell ist, sieht wie folgt aus:

```
2700 REM **** IS P SPECIAL S/R ****
2705 SF=0:REM UNSET SPECIAL FLAG
2716 REM ** OTHER SPECIAL LOCATIONS **
2720 ON P GOSUB4590,4690,4790,4590
2730 RETURN
```

Sie erinnern sich sicher, daß wir bei der Erstellung der Karte für Haunted Forest die speziellen Orte zuerst mit Nummern versehen haben. Dadurch kann die entsprechende Unterroutine einfach mittels des ON. . .GOSUB-Befehls aufgerufen werden. Wie Sie sehen, befinden sich hinter dem Befehl einige Zeilennummern, von denen entsprechend dem Wert von P eine ausgewählt wird. Hat P den Wert 1, wird zur ersten Zeilennummer verzweigt. Bei einem Wert von P = 2 wird die zweite Zeilennummer aufgerufen und so weiter.

Insgesamt gibt es vier Zeilennummern, jeweils eine für einen speziellen Ort in Haunted Forest. Ist P größer als vier, wird der Programmlauf einfach mit der folgenden Zeile fortgesetzt. Wenn in den vier Unterroutinen, die von Zeile 2720 aufgerufen werden können, das Flag SF gesetzt wird, läßt sich erkennen, ob P ein spezieller Ort war oder nicht. Wurde keine Routine aufgerufen, so ist SF=0, und P war nur ein normaler Ort. Sie sehen, daß der ON. . .GOSUB-Befehl eine bessere Methode zur Programmverzweigung darstellt als mehrere IF. . .THEN-Anweisungen.

Zwei der speziellen Orte in Haunted Forest sind die beiden Tunneleingänge (Ort 1 und 4). Damit der Spieler den Tunnel betreten kann, müssen wir eine Routine konstruieren, die normale Befehle verarbeitet sowie ein Betreten des Tunnels ermöglicht.

```
4590 REM **** TUNNEL ENTRANCE S/R ****
4600 SF=1
4605 SN$="YOU HAVE ARRIVED AT THE MOUTH OF A LARGE
TUNNEL":GOSUB5500
4610 SN$="YOU CAN ENTER THE TUNNEL OR RETREAT
ALONG THE PATH":GOSUB5500
4620 :
4625 PRINT:INPUT"INSTRUCTIONS";IS$
4630 GOSUB2500:REM SPLIT INSTRUCTION
4635 IF F=0 THEN 4625:REM INVALID INSTRUCTION
4637 GOSUB3000:REM NORMAL INSTRUCTIONS
4640 IF MF=1 THEN RETURN:REM PLAYER RETREATS
4645 IF VF=1 THEN 4625:REM INSTRUCTION OBEYED
4650 REM ** NEW INSTRUCTIONS **
4655 IF VB$="ENTER" THEN GOSUB 4700:RETURN
4660 IF VB$="RETREAT" AND P=4 THEN MF=1:P=6:RETURN
4665 IF VB$="RETREAT" AND P=1 THEN MF=1:P=9:RETURN
4667 SN$="I DON'T UNDERSTAND":GOSUB5500:GOTO 4625
```

Die Routine beginnt damit, daß sie SF auf 1 setzt, um zu kennzeichnen, daß ein spezieller Ort erreicht wurde. Nach Darstellung der Ortsbeschreibung und Bewegungsmöglichkeiten wird die Eingabe eines Befehls erwartet. Und wieder werden die Vorteile der modularen Programmstruktur deutlich: Die Unterroutinen zur Interpretation eines Befehls lassen sich separat aufrufen. Durch Untersuchung der verschiedenen, durch die beiden Routinen gesetzten Flags kann der Programmlauf in der neuen Routine entsprechend gesteuert werden. Betrachten wir die Flags im einzelnen.

Das F-Flag, das in der Routine zum Zerlegen eines Befehls gesetzt wird, gibt an, ob der Befehl ein gültiges Format hat. Besteht er nur aus einem Wort, hat F den Wert 0 – in diesem Fall muß ein neuer Befehl eingegeben werden.

Das MF-Flag wird durch die Routine zur Handhabung „normaler" Befehle gesetzt, wenn eine Ortsbeschreibung dargestellt werden soll – also bei einem GO- oder LOOK-Befehl. Nach Rücksprung in die Hauptschleife kann entweder zum neuen Ort gegangen werden, oder derselbe Ort wird neu beschrieben und erneut die „spezielle" Ortsroutine aufgerufen.

Das VF-Flag wird auch durch die Routine zur Handhabung „normaler" Befehle gesetzt. VF=1 gibt an, daß der Befehl verstanden und ausgeführt wurde. In diesem Fall verzweigt das Programm zur nächsten Befehlseingabe.

Ist der Wert von VF ungleich 1, handelt es sich um keinen normalen Befehl. Nachdem wir uns mit den normalen Befehlen befaßt haben, wollen wir nun zwei neue Befehle einfügen: ENTER, um in den Tunnel zu gehen, und RETREAT, um sich vom Tunneleingang zu entfernen. Da diese Routine für beide Eingänge des Tunnels verwendet werden soll, muß beim RETREAT-Befehl beachtet werden, an welchem Ende des Tunnels sich der Spieler befindet. Dies läßt sich durch P=1 oder P=4 kennzeichnen. P kann dann vor dem Rücksprung in die Hauptschleife entsprechend neu gesetzt werden, damit die Bewegung zum neuen Ort möglich ist.

Die speziellen Gefahren, die den Abenteuer-Spieler im Tunnel erwarten, sind Thema des nächsten Artikels.

Digitaya-Listing

```
1190 GOSUB2670:REM IS P SPECIAL
1200 IF SF=1 THEN 1250:REM NEXT LOOP

2360 REM ** DROP S/R **
2370 GOSUB5730:REM IS OBJECT VALID
2380 IF F=0 THEN PRINT"THERE IS NO ";W$:RETURN
2390 :
2400 REM ** IS OBJECT HELD ? **
2410 OV=F:GOSUB5830
2420 IFHF=0THENPRINT"YOU DO NOT HAVE THE ";IV$(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

2670 REM **** IS P SPECIAL S/R ****
2680 SF=0:REM UNSET SPECIAL FLAG

2710 ON P GOSUB 2850,2960,3450,3830,4180,4550,5150
2720 RETURN

2850 REM **** TV OUTLET S/R ****
2860 SF=1
2870 SN$="YOU HAVE ENTERED THE TV OUTLET AND THERE
 'S NO ESCAPE."
2880 SN$=SN$+"YOU ARE DOOMED FOREVER TO BE A TV CH
AT 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

3830 REM **** JOYSTICK PORT ****
3840 SF=1
3850 SN$="A USER WITH RED-RIMMED EYES ZAPS HIS LAS
ER 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":GO
TO3880
3930 IFVF=1THEN3880:REM NEXT INSTRUCTION
3940 IFVB$<>"USE"THENPRINT"I DON'T UNDERSTAND":GOT
O3880
3950 GOSUB5730:REM IS OBJECT VALID
3960 IFF=0THENPRINT"THERE IS NO ";NN$: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:G
OTO3880
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$="YOU USE THE LASER SHIELD TO PROTECT YOUR
SELF. A BLAST KNOCKS"
4070 SN$=SN$+" YOU OUT OF THE JOYSTICK PORT AND BA
CK INTO THE MACHINE."
4080 GOSUB5880:REM FORMAT
4090 P=INT(RND(TI)*40+7):MF=1:RETURN
4100 :
4110 REM ** HIT **
4120 SN$="YOU ARE HIT BY THE LASER AND YOU ARE ONL
Y 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

5150 REM **** GATEWAY TO MEMORY S/R ****
5160 SF=1
5170 SN$="AN USHER GREETS YOU BUT TELLS YOU THAT Y
OU CANNOT BE ADMITTED"
5180 SN$=SN$+" UNLESS YOU GIVE AN ADDRESS":GOSUB58
80
5190 REM ** INSTRUCTIONS **
5200 PRINT:INPUT"INSTRUCTIONS";IS$
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$<>"GIVE"THENPRINT"I DON'T UNDERSTAND":G
OTO 5200
5250 :
5260 GOSUB5730:REM IS OBJECT VALID
5270 IFF=0THENPRINT"THERE IS NO ";W$: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$="THE USHER LOOKS AT YOUR ADDRESS AND ALLO
WS YOU TO PASS"
5390 SN$=SN$+" THROUGH":GOSUB5880
5400 P=40:MF=1:RETURN
```

BASIC-Dialekte

Spectrum:
Verwenden Sie in beiden Programmen die folgenden Variablennamen: S$ anstelle von SN$, R$ für NN$, V$(,) für IV$(,), I$() anstelle von IC$(), T$ für IS$ und B$ für VB$. Ergänzen Sie außerdem die folgenden Zeilen im Listing von Haunted Forest:

```
2720 IF P=1 THEN GOSUB 4590
2722 IF P=2 THEN GOSUB 4690
2724 IF P=3 THEN GOSUB 4790
2726 IF P=4 THEN GOSUB 4590
```

Ergänzen Sie die folgenden Zeilen im Digitaya-Listing:

```
2710 IF P=1 THEN GOSUB 2850
2711 IF P=2 THEN GOSUB 2960
2712 IF P=3 THEN GOSUB 3450
2713 IF P=4 THEN GOSUB 3830
2714 IF P=5 THEN GOSUB 4180
2715 IF P=6 THEN GOSUB 4550
2716 IF P=7 THEN GOSUB 5150
3890 LET RD=RND(1)
4090 LET P=INT(RND(1)*40+7)
```

Acorn B:
Ergänzen Sie die folgenden Zeilen im Digitaya-Listing:

```
3890 RD= RND(1)
4090 P=RND(40)+7
```

Maschinengeister

Die Firma Ultimate ist bekannt für qualitativ hochwertige Computerspiele. Schon ihre ersten Veröffentlichungen setzten einen neuen Standard für die Bewegungsgrafik des Spectrum. Wir sehen uns Atic Atac an, das die Strategieelemente der Abenteuer mit der Schnelligkeit der Arcadespiele kombiniert.

Atic Atac ist eins der wenigen Spiele, in denen die Kombination zwischen komplexem Abenteuerspiel und schneller Action gelungen ist. Im klassischen Abenteuerstil werden Sie in ein verwunschenes Schloß versetzt, aus dem Sie nur entkommen können, wenn Sie den goldenen Schlüssel für das Hauptportal finden. Schreckliche Wesen bedrohen Ihr Leben: Spinnen, Dämonen, Hexen, Teufel, hungrige Monster und – nicht zu vergessen – Drakula, Frankensteins Monster, die Mumie und Fledermäuse. Zusammen mit den vielen Falltüren und Geheimgängen vermittelt das Spiel den Eindruck eines überbesetzten Horrorfilms.

Atic Atac sieht zwar auf den ersten Blick wie ein typisches Abenteuer aus, doch gibt die hervorragende Bewegungsgrafik dem Spiel eine Dynamik im besten Arcadestil. Alle Räume und Kerker sind mit Ritterrüstungen, Bücherregalen, Standuhren und Bildern ausgestattet und erscheinen farbig und dreidimensional mit Türen nach Norden, Süden, Osten und Westen. In der gewählten Verkleidung als Ritter, Zauberer oder Sklave und mit der entsprechenden Bewaffnung durchqueren Sie die Räume und treffen die rauchwolkenumhüllten Monster, deren Berührung die Kräfte schwächt. Glücklicherweise lassen sich die Angreifer mit Axt, Schwert oder Zauberspruch abwehren. Die meisten Monster stellen sich Ihnen nicht direkt entgegen, einige gehen jedoch genau auf Sie zu.

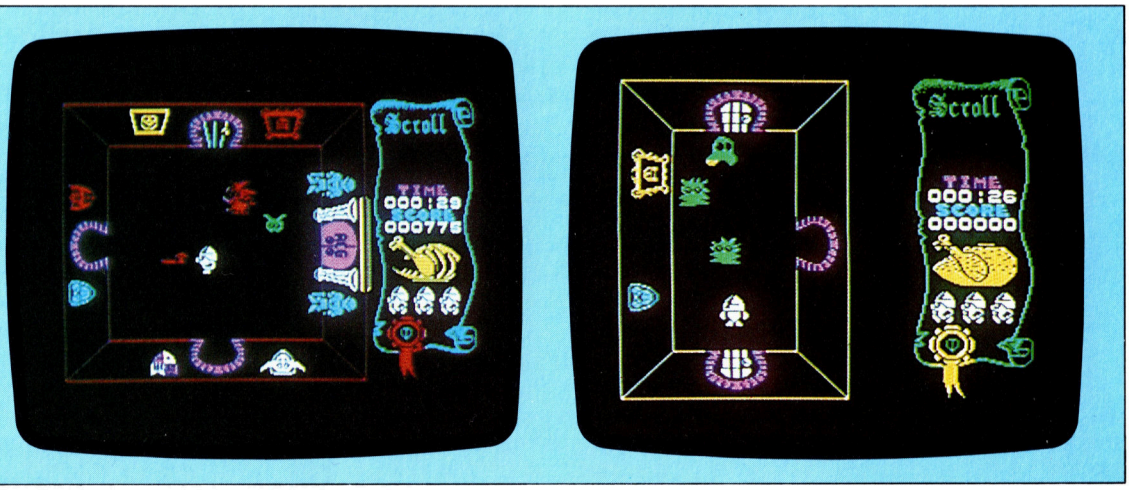
Die Bewegung wird mit einem Joystick oder per Tastatur gesteuert. Gäbe es nicht die wertvollen Objekte (beispielsweise den goldenen Schlüssel), würde das Abenteuer schnell zu einem reinen Ballerspiel werden. Manche Türen lassen sich jedoch nur öffnen, wenn Sie den Schlüssel in der richtigen Farbe bei sich haben, während andere Gegenstände bestimmte Arten von Monstern abschrecken. Wichtig sind auch die Nahrungsmittel. Auf der rechten Bildschirmseite zeigt ein Brathuhn an, wieviel Proviant Sie noch haben. Wenn Sie sich nicht rechtzeitig zum Essen entschließen, verändert es sich von einem knusprigen Hähnchen in einen Haufen Knochen, der schließlich Ihren Hungertod anzeigt.

Für den Anfänger kann Atic Atac zunächst recht frustrierend sein. Kennen Sie sich jedoch einmal mit der Bekämpfung der Monster aus, dann steht Ihnen die Abenteuerseite des Spiels offen, und Sie können das Schloß nach seinen Schätzen durchsuchen. Doch seien Sie gewarnt – es kann viele anstrengende Stunden dauern, bis der Schlüssel gefunden ist und die Türen geöffnet werden können.

Atic Atac: für den Spectrum (48K)
Herausgeber: Ashby Computer and Graphics Ltd., Ashby de la Zouch, Leicestershire, LE6 5JU
Autoren: Ultimate Play the Game
Joysticks: Kempston-Joystick
Format: Cassette

Atics Zauber

Die beiden Bildschirmfotos zeigen zwei Szenen aus Atic Atac. Während Sie in den Hallen und Geheimgängen nach dem goldenen Schlüssel suchen, begegnen Sie einer Vielzahl von Monstern und anderen unangenehmen Überraschungen.



Klassenbester

Die englische Firma Research Machines Ltd. konnte in den vergangenen Jahren mit ihren Acht-Bit-Rechnern 380Z und 480Z speziell im Markt der Schulcomputer Erfolge verzeichnen. Von ihrem zukunftsweisenden 16-Bit-Rechner „Nimbus" erhofft sich der britische Hersteller einen Durchbruch auch im kommerziellen Bereich.

Mit dem Nimbus wagt sich die englische Firma Research Machines auf das harte umkämpfte Feld kommerzieller Rechner. Der moderne Prozessor 80186 von Intel kann bis zu einem Megabyte Adressen verwalten. Es scheint, daß sich der Hersteller auf Dauer nicht mit seinem guten Ruf im Bereich Schul- und Forschungscomputer zufriedengeben möchte . . .

Abseits von Forschungsstätten und Schulen ist Research Machines Ltd. ein noch weitgehend unbeschriebenes Blatt, obwohl der ausbaufähige 380Z und der Netzwerk-Rechner 480Z für den Acorn B im Wettbewerb um das staatliche Schulrechner-Programm in England eine scharfe Konkurrenz darstellten.

Rechner wie der 380Z waren bei ihrer Präsentation sicherlich revolutionär – der Wunsch nach noch größerer Leistung führte dennoch bald auf den Weg zum 16-Bit-Prozessor. Research Machines erweiterte jetzt ihr Angebot mit dem neuen 16-Bit-Rechner Nimbus.

Seit der Entwicklung des 380Z hat sich der Rechnermarkt rapide weiterentwickelt – ganz oben steht der IBM PC, umgeben von einer kaum noch zu übersehenden Schar von Kompatiblen. Basis dieses sogenannten „Industriestandards" ist der Prozessor 8088, der – obwohl schneller als der Z80 – bereits nicht mehr die absolute Spitze der Prozessortechnik darstellt. Research Machines hat darauf verzichtet, einen weiteren IBM-Nachbau auf den Markt zu bringen – der Nimbus arbeitet mit dem sehr viel schnelleren Typ 80186. Dieser echte 16-Bit-Prozessor kann bis zu einer Million Befehle pro Sekunde verarbeiten und stellt damit fast alle Konkurrenten in den Schatten.

Obwohl der Nimbus nicht IBM-kompatibel ist, läuft er wegen der engen Verwandtschaft des 8088 und des 80186 auch unter dem Betriebssystem MS-DOS. Als Tastaturkonfiguration kann sowohl das IBM- wie auch das firmeneigene Research-Machines-Format gewählt werden. Dabei stehen zehn programmierbare Funktionstasten, eine numerische Tastatur und ein zweiter Zeichensatz zur Verfügung. Die Tasten selbst sind außerdem hervorragend zu bedienen.

Wir beziehen uns in diesem Bericht auf den Nimbus PC2 mit zwei 3½-Zoll-Sony-Laufwerken. Alternativ gibt es auch die PC1-Version mit einer Diskettenstation. Das 3½-Zoll-Format wird immer häufiger bei Spitzenrechnern eingesetzt – bei der hohen Aufzeichnungskapazität von 720 KByte ist das kein Wunder.

Speicher kontra Kompatibilität

Das Diskettenformat zeigt, daß es Research Machines eher um die Eroberung einer technischen Spitzenposition als um Kompatibilität ging. IBM verwendet das herkömmliche 5¼-Zoll-Format, das weniger Speicherplatz bietet als das Sony-Produkt. Offenbar wollte der britische Hersteller einer höheren Prozessorleistung auch die entsprechende Peripherie zur Seite stellen.

Es stellt sich die Frage, wie sich der Nimbus mit der 5¼-Zoll-Software seiner Vorgänger von Research Machines verträgt. Probleme sind nicht zu erwarten, der Nimbus ist für die Installation weiterer Laufwerke vorbereitet. Beim Anschluß von 5¼-Zoll-Laufwerken liest das Gerät Disketten im Format PC-DOS, MS-DOS oder auch CP/M. Für Umsteiger von Rechnern die-

ser Formate könnte die Vielseitigkeit zu einem entscheidenden Kaufanreiz werden, weil die schon erworbene Software verwendbar bleibt.

Für das – bisher noch begrenzte – Software-Angebot auf ROM-Cartridges befinden sich beim Nimbus unter den Laufwerken zwei Steckplätze, die mit den Anschlüssen beim 480Z kompatibel sind. Links daneben sitzt der „Dongle"-Steckplatz – eine Vorrichtung zum Schutz gegen unerlaubtes Benutzen der Software. Auch bei einer vorhandenen Kopie der Software läuft ohne Dongle kein Programm.

Das mit dem Nimbus gelieferte Software-Paket umfaßt neben der MS-DOS-Systemdiskette, Wordplan von Microsoft und Multiplan auch das RM-LOGO und RM-BASIC. Der Nimbus offenbart seine Stärken sofort nach dem Laden der Software. Eindrucksvoll sind speziell die Grafikmöglichkeiten im BASIC.

Strukturiertes RM-BASIC

Das mit dem Nimbus gelieferte BASIC ist aus der RM-BASIC-Version 5 entwickelt worden. Aufgrund der Aktivitäten der Firma im schulischen Bereich ist es klar, das das BASIC die strukturierte Programmierung besonders unterstützt. RM-BASIC verfügt über Befehlsstrukturen wie etwa REPEAT . . .UNTIL, PROC . . .ENDPROC und GLOBAL, allerdings nicht über das in der RM-BASIC-Version 6 vorhandene DO . . .WHILE.

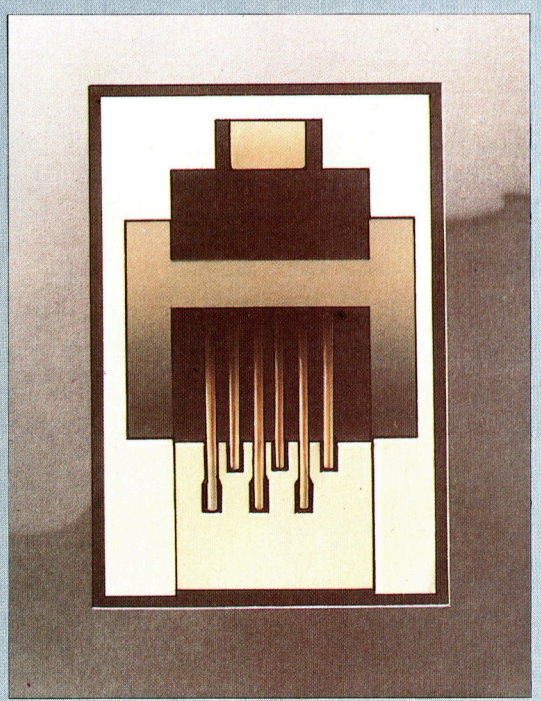
Dieser Mangel wird jedoch durch zusätzliche Befehle für die Spielprogrammierung ausgeglichen. Maus und Joystick (wahlweise) können mit den Befehlen JOYX, JOYY und MOUSE abgefragt und innerhalb des Programms durch den BUTTON-Befehl aktiviert werden.

Grafik und Sound werden im BASIC durch den von den entsprechenden Parametern gefolgten SET-Befehl angewählt. Damit können Vorder- und Hintergrundfarben (durch PAPER-, BRUSH- und PEN-Befehle) gewählt, Linien und Kreise gezeichnet, der Zeichensatz umgeschaltet oder Größen und Richtungen festgelegt werden. Auch die im Handbuch als „Styles" bzw. „Dithering Patterns" bezeichneten Möglichkeiten stehen zur Verfügung. Das sind Grundmuster zur Strukturierung des Vorder- oder Hintergrundes bzw. zur Mischtonbildung aus den einzelnen Farbpunkten eines bestimmten Bildschirmabschnittes.

Neben der Vielfalt beeindruckt die Geschwindigkeit, mit der Grafik im BASIC auf dem Bildschirm erscheint – hier sind sogar Vergleiche mit der maschinenprogrammierten Grafik anderer 16-Bit-Rechner erlaubt.

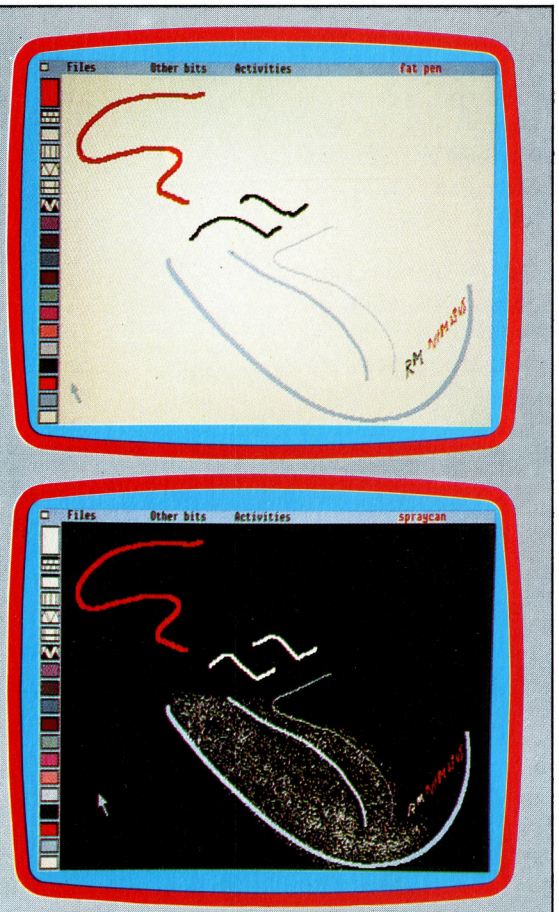
Die Diskettenverwaltung arbeitet mit Directories (Inhaltsverzeichnissen) und Sub-Directories. Beim Einstieg ins BASIC muß also erst das BASIC-Inhaltsverzeichnis aus dem Hauptinhaltsverzeichnis aufgerufen werden. Mit dem CD-Befehl (Change Directory) läßt sich die Arbeit mit dem MS-DOS und BASIC automatisieren – die Eingabe eines Namens führt dann durch die verschiedenen Directory-Ebenen. Das mag umständlich erscheinen, zwingt jedoch zu systematischer Anlage der Directories, was wiederum seine Vorteile hat. Mit dem Betriebssystem können auch neue Wege zwischen bereits vorhandenen Sub-Directories geschaffen werden.

Durch die „State of the art"-Philosophie des Nimbus-Herstellers findet man aber auch bei anderen Merkmalen Ungewöhnliches: Das Gerät hat auf der Gehäuserückseite drei verschiedene Spannungs-Ausgänge – einen



Ausbaufähig
Der Nimbus von RM hat eine Vielzahl von Anschlußmöglichkeiten. Der Rechner verfügt über zwei Monitoranschlüsse, einen Port für Joysticks bzw. Maus, einen „Piconet"-Anschluß und eine Druckerschnittstelle. Dazu kommen noch drei Anschlüsse zur Stromversorgung von Peripheriegeräten. Die wichtigsten Schnittstellen sind nach der englischen Norm ausgeführt.

Die neu eingeführten Telefon-Buchsen erlauben die serielle Datenein- und ausgabe. Die sechs Leitungen können unterschiedlich verwaltet werden, je nach dem, ob sie für die Datenübertragung oder für andere Zwecke verwendet werden sollen.



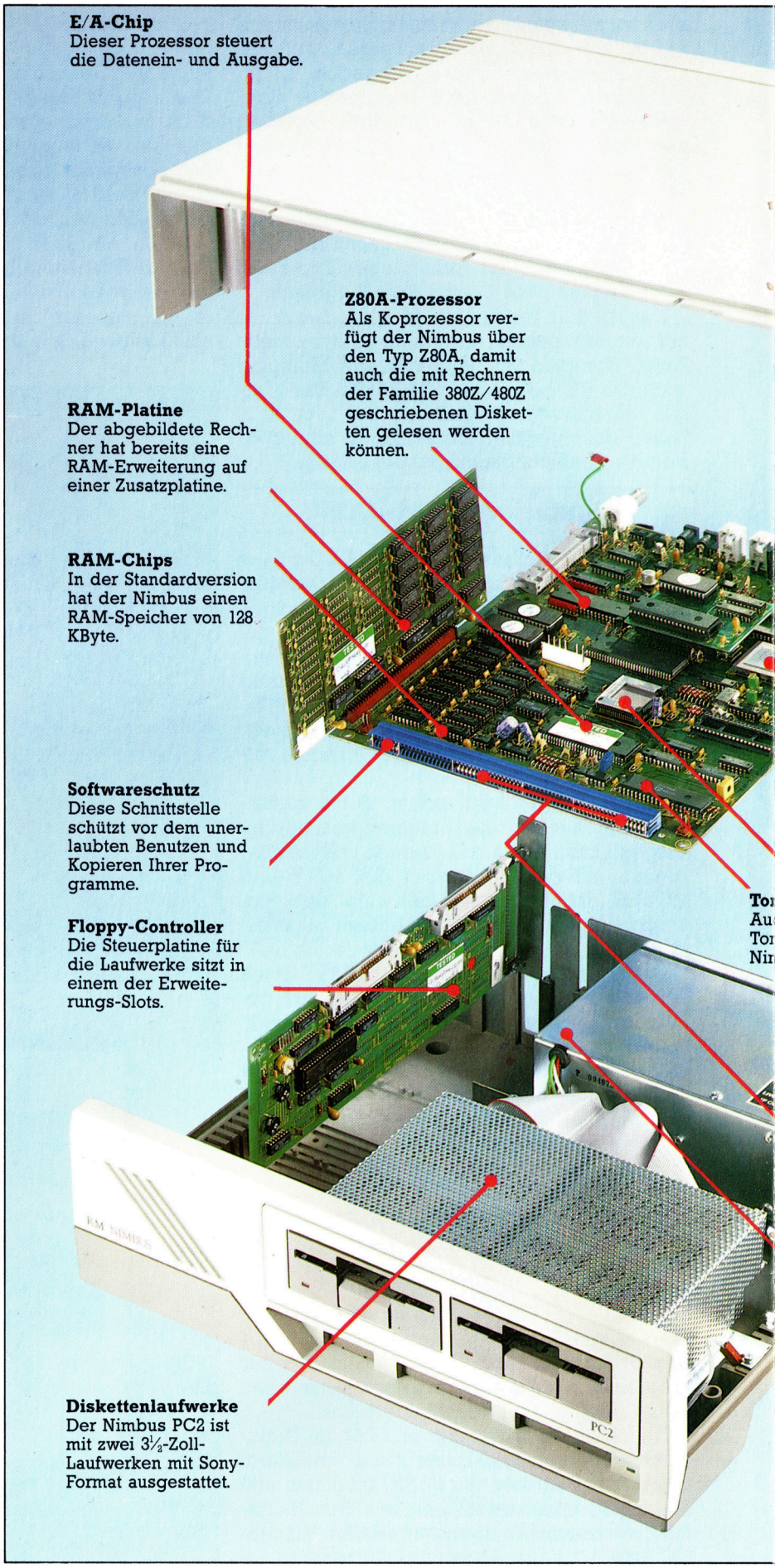
MacNimbus
Zu den Demoprogrammen des Nimbus gehört auch ein mausgesteuertes Grafikprogramm. Es hat starke Ähnlichkeit mit dem Programm des Apple Macintosh – auch hier können unterschiedliche Stifte bzw. „Pinsel" gewählt werden. Der Nimbus stellt bis zu 16 verschiedene Farben dar.

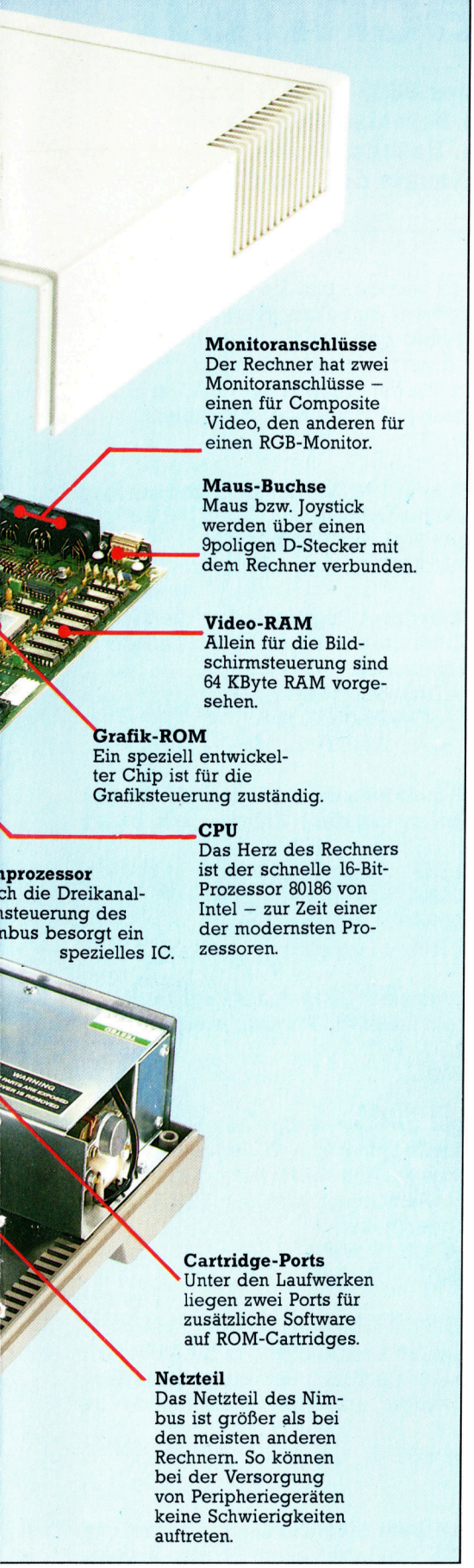
2-Ampere-Ausgang vom Netztrafo sowie einen 12- und einen 5-Volt-Ausgang.

Zusätzlich zu den anderen Schnittstellen gibt es einen „Piconet"-Anschluß, der dem Standard der englischen Telefonvorrichtung entspricht. Damit können über ein externes Modul bis zu 30 verschiedene Peripheriegeräte von einem einzigen Port angesprochen werden. Diese serielle Schnittstelle mit hoher Datenübertragungsgeschwindigkeit entspricht dem RS422-Standard.

Unglücklicherweise ist die RS422-Schnittstelle in den seltensten Fällen mit der verbreiteten RS232-Norm kompatibel – für den Anschluß bereits vorhandener Peripherie braucht man ein passendes Interface. Dafür gibt es allerdings einen Koaxial-Anschluß, mit dem der Nimbus und weitere Research-Machines-Computer zu einem Netzwerk zusammengeschaltet werden können.

Nicht weniger ungewöhnlich ist der Drukkeranschluß – ein Stecker nach Postnorm, ähnlich dem „Piconet"-Anschluß. Er entspricht ebenfalls einer englischen Telefonbuchse und arbeitet nach dem RS422-Standard, also seriell.





Die weiteren Schnittstellen richten sich eher nach der üblichen Norm, zum Beispiel ein 9poliger D-Stecker für Joystick bzw. Maus. Mit der Maus arbeitet der Nimbus übrigens beinahe wie ein Apple Macintosh – zur Demonstration gibt es ein Malprogramm, das in der Bildschirmdarstellung sehr an den Macintosh erinnert. Unterschiede gibt es bei der Farbgrafik. Der Nimbus bietet auch nicht alle MacPaint-Möglichkeiten.

Der Rechner hat zwei Monitor-Anschüsse: einen für den auch unter dem Markenzeichen von RM erhältlichen Cub-Monitor. Der zweite Anschluß liefert Composite Video für Monochrom-Bildschirme. Schließlich gibt es noch den Tastaturanschluß über eine DIN-Buchse.

Ganz links am Rechner (von hinten gesehen) liegen vier Ports zum weiteren Ausbau des Computers mit zusätzlichen Platinen, etwa für zusätzliches RAM oder Digital-Analogwandler. Research Machines folgt dabei nicht der Praxis anderer Hersteller, die den Ausbau nur ihren Fachwerkstätten übertragen möchten: Das Handbuch informiert sehr ausführlich, wie das Gerät durch den Anwender erweitert werden kann.

Gute Dokumentation

Auch bei der mitgelieferten Dokumentation hält RM seinem Qualitätsanspruch die Treue: Auf eine Einführung in MS-DOS und die Laufwerksbedienung folgt die genaue Beschreibung der einzelnen Ports. Als Mangel erscheint jedoch das Fehlen jeglicher Erklärungen zum 80186-Prozessor. Solche Informationen sind vielleicht für die Mehrzahl der Anwender von geringem Interesse, ein geübter Programmierer benötigt diese jedoch für die Entwicklung eigener Software unbedingt.

Bei welcher Käuferschicht der Nimbus ankommt, läßt sich heute noch nicht sagen. Hohe Qualität und ein sehr fortschrittliches Konstruktionsprinzip kann man ihm guten Gewissens bescheinigen. Zusätzlich fällt ins Gewicht, daß der Nimbus nur halb soviel kostet wie sein einziger ernsthafter Konkurrent, der IBM PC. Trotzdem hängt sein Erfolg maßgeblich von einer Marketing-Strategie des Herstellers ab.

Universitäten und Forschungsinstitute speziell in England haben bereits gute Erfahrungen mit RM gemacht. Sicherlich werden hier auch neue Programme entstehen, so daß die Software-Unterstützung in Zukunft stärker wird. Auch die Vielfalt der anschließbaren Peripherie und die leichte Programmierbarkeit des Rechners machen das Gerät attraktiv.

Zweifel bleiben allerdings, ob sich der Nimbus auch auf dem kommerziellen Markt durchsetzen wird. Denn dazu gehört neben der reinen Rechnerleistung ein umfangreiches Software-Angebot. Auch die Schnittstellen sollte die Herstellerfirma an die im Büro gängigen Standards Centronics und RS232 anpassen.

RM Nimbus

ABMESSUNGEN
400 x 350 x 110 mm

CPU
Intel 80186 mit 8 MHz

BILDSCHIRMAUFLÖSUNG
Maximal 80 x 25 Zeichen (Text), 640 x 250 Pixel im Vierfarb-Modus, 320 x 125 Pixel im Achtfarb-Modus.

SCHNITTSTELLEN
Netzwerkanschluß, Piconet-Anschluß, Druckeranschluß, 12V-, 5V- und 2A-Spannungsversorgung, Composite- und RGB-Monitorbuchsen.

PROGRAMMIERSPRACHEN
RM-BASIC, RM-LOGO, RM-PASCAL.

TASTATUR
83 Tasten einschl. numerischem Tastenfeld und zehn programmierbare Funktionstasten.

HANDBUCH
Gutes, ausführliches Handbuch. Besser als frühere RM-Handbücher, mehr Informationen, die leichter lesbar geworden sind.

STÄRKEN
Im Vergleich mit Konkurrenten sehr günstiger Preis. Durch die moderne Technik, leichte Bedienung, hervorragende Grafik und gute Dokumentation könnte das Gerät ein Renner werden.

SCHWÄCHEN
Zur Zeit kann es noch zu Lieferproblemen bei der passenden Peripherie kommen.

Einfaches Rechnen

Nachdem in der letzten Folge der Aufbau des 6809 erklärt wurde, sehen wir uns nun an, wie sich mit seinem Befehlssatz einfache mathematische Aufgaben ausführen lassen. Besonders interessant sind dabei die Vorzeichen-Arithmetik und der Einsatz des Condition Code Registers in den Programmen.

In dieser Folge setzen wir erstmals Befehle zu einem Programm zusammen. Dazu führen wir zunächst einige neue Kommandos ein und sehen uns die Darstellung von Daten genauer an. Unser Beispielprogramm wandelt eine „Binär Codierte Dezimalzahl" (BCD) in das Binärformat um.

Dezimalzahlen im BCD-Format eignen sich ausgezeichnet für die Acht-Bit-Verarbeitung. Die einzelnen Ziffern der Dezimalzahlen werden dabei als vierstellige Binärzahlen geschrieben. So wird aus der Dezimalzahl 69 die BCD-Zahl %01101001. Die vier Bits auf der linken Seite (0110) sind die binäre Entsprechung der Zahl 6, während die vier rechten Bits (1001) die 9 darstellen. Das BCD-Format ergibt einen völlig anderen Dezimalwert als die Umwandlung einer Binärzahl (%01101001 ergibt dezimal 105).

Für unser Umwandlungsprogramm benötigen wir eine Reihe neuer Befehle:

● LSR (Logische Rechtsverschiebung): Alle Bits des Operanden werden um eine Position nach rechts geschoben. Das rechtsaußen stehende Bit wandert in das Übertragsbit des Condition Code Registers, und das linksaußen stehende Bit wird auf Null gesetzt.

● AND: Für jedes Bit eines Registers wird eine logische Verknüpfung mit den entsprechenden Bits des Operanden durchgeführt und das Ergebnis wieder im Register gespeichert. Mit diesem Vorgang lassen sich Bits „maskieren". Steht ein Bit des ersten Registers auf Eins, dann kopiert AND den Bitwert des zweiten Registers in das Ergebnisregister. Steht das Bit des ersten Registers dagegen auf Null, dann ist unabhängig vom Inhalt des zweiten Registers auch das Ergebnisbit Null. Eine logische AND-Verknüpfung mit %00001111 kopiert daher nur die vier rechten Bits in das Ergebnisregister:

```
%00001111 Registerwert
%10110110 AND-Wert des Operanden
%00000110 Ergebnis im Register
```

● MUL: Multipliziert den Inhalt der Register A und B und speichert das Ergebnis im D-Register (das aus A und B zusammengesetzte 16-Bit-Register). Nur wenige Acht-Bit-Prozessoren unterstützen einen Op-Code für Multiplikation.

● SWI (SoftWare Interrupt): Dieser Befehl bietet eine Möglichkeit, Maschinencodeprogramme zu beenden und die Steuerung an das Betriebssystem zurückzugeben. Wir werden diesen Befehl genauer behandeln, wenn wir auf das Interruptsystem eingehen. Hier das Programm für die Umwandlung BCD in Binär:

● Der Anfangswert des Speicherzählers:

```
ORG $1000
```

● BCD 58 in BCDNUM speichern und ein Byte in BINNUM reservieren:

```
BCDNUM FCB  %01011000
BINNUM  RMB 1
```

● BCD 58 in das A-Register laden, die niederwertige Ziffer maskieren und anschließend in BINNUM speichern:

```
START LDA   BCDNUM
      ANDA #%00001111
      STA   BINNUM
```

● BCD 58 in Akkumulator A laden und die vier hohen (linksstehenden) Ziffern nach rechts schieben:

```
SHIFT LSRA
      LSRA
      LSRA
      LSRA
```

● Die Dezimalzahl 10 in das B-Register laden und mit dem Inhalt von A multiplizieren:

```
MULT LDB  #10
      MUL
```

● Ergebnis sind die 16 Bits des D-Registers. Da das Resultat nicht über 90 liegen kann (10 * 9 = 90), wird nur das niederwertige Byte von D gebraucht. Sein Inhalt wird auf BINNUM addiert und gespeichert.

```
ADDIT ADDB BINNUM
      STB   BINNUM
```

● Die BCD-Zahl befindet sich nun in BCDNUM und ihre binäre Entsprechung in BINNUM. Wir brauchen nur noch auf Betriebssystemebene zurückzukehren und den Quellencode zu beenden:

```
RETURN SWI
END
```

Es gibt mehrere Möglichkeiten, positive und negative Zahlen darzustellen. Für den Arbeits-

speicher und die Register eines Computers ist das **Zweierkomplement** die einfachste Methode, mit negativen Zahlen umzugehen. Das Zweierkomplement einer Binärzahl entsteht, wenn alle Ziffern in ihr Gegenteil verkehrt (alle Nullen in Einsen und umgekehrt) und die Zahl Eins addiert wird. So ist das Zweierkomplement von 0101 die Binärzahl 1011.

Wie läßt sich diese Zahl nun für die Vorzeichen-Mathematik verwenden? Sehen wir uns zunächst an, welcher Zahlenbereich überhaupt möglich ist. Ein Acht-Bit-Register hat 256 verschiedene Bitmuster, die positive Zahlen von 0 bis 255 **oder** negative und positive Zahlen von −128 bis +127 darstellen können. Die Tabelle zeigt die Binärdarstellung von −7 bis +7 im Vier-Bit-Format.

Sie werden bemerken, daß alle negativen Zahlen der Tabelle als höchstwertiges Bit (linksaußen) eine Eins haben und alle positiven Zahlen eine Null.

Aus dieser Vier-Bit-Tabelle lassen sich nun einige grundlegende Eigenschaften der Vorzeichen-Mathematik mit dem Zweierkomplement ableiten:

- Aus dem Zweierkomplement einer negativen Zahl ergibt sich das positive Äquivalent (und umgekehrt).
- Das höchstwertige Bit ist bei positiven Zahlen immer Null und bei negativen Zahlen immer Eins. Damit lassen sich positive und negative Zahlen leicht auseinanderhalten.
- Das Zweierkomplement von Null ist Null (1111 plus 1).
- Addition und Subtraktion werden wie üblich ausgeführt, wobei das Ergebnis immer das richtige Vorzeichen erhält.

Steuermechanismen

Überprüfen Sie die letzte Aussage einmal mit einigen einfachen Additionen und Subtraktionen. Die Multiplikation von Zahlen mit Vorzeichen ist komplizierter, da der Befehl MUL, den wir schon bei der Umwandlung BCD-in-Binär eingesetzt hatten, den Inhalt der Register A und B nur als Zahlen ohne Vorzeichen behandelt. Die Multiplikation zweier Zahlen mit Vorzeichen muß extra programmiert werden.

Die engen Grenzen unseres bisherigen „linearen" Programmaufbaus können wir nur mit folgenden Steuermechanismen erweitern:

- Auswahl: die Wahl zwischen zwei unterschiedlichen Lösungswegen (wie der IF-Befehl in BASIC).
- Wiederholung: der wiederholte Ablauf einer Befehlsfolge:

1) wenn eine bestimmte Bedingung „wahr" bleibt (die WHILE. . .WEND-Struktur);
2) bis eine bestimmte Bedingung „wahr" wird (REPEAT. . .UNTIL); oder
3) für eine festgelegte Anzahl von Durchläufen (FOR. . .NEXT).

All diese Strukturen sind nur möglich, wenn wir auch testen können, ob eine Bedingung wahr oder falsch ist. Dabei wird zumeist überprüft, ob eine Variable einen bestimmten Wert hat oder nicht. Doch obwohl die Assemblersprache leicht zwei Registerwerte miteinander vergleichen kann, würden damit im Normalfall nur zwei Möglichkeiten zur Verfügung stehen: Ein Wert ist Null oder nicht Null, und ein Wert ist positiv oder negativ. Mit Zusatzbefehlen lassen sich jedoch auch andere Arten von Tests durchführen.

Viele Tests werden erst durch den Einsatz des Condition Code (CC) Registers möglich. Das CC-Register hat ein Acht-Bit-Format, bei dem uns jedoch nicht der dort gespeicherte Wert interessiert, sondern der Status (1 oder 0) jedes einzelnen der acht Bits. Fünf der acht Bits zeigen Bedingungen an, die wir für die oben erwähnten Verzweigungen einsetzen können, die anderen drei beziehen sich auf

Die folgenden Beispiele zeigen, wie sich Assembleranweisungen und Befehle auf den Speicherzähler des Assemblers und die Speicherstellen auswirken.

Assembleranweisungen

```
Label-   Op-Code-   Operand-    Speicher-   Speicher-
Feld     Feld       Feld        zähler      inhalt
* ——— BEISPIEL ———————————————————————————
RESET    EQU        $F100       $0400       ??
INDEX    EQU        16          $0400       ??
MASK1    EQU        %01101010   $0400       ??

         ORG        $1000       $1000       ??

CR       FCB        16          $1000       $10

MEMTOP   FDB        $7FFF       $1001       $7F
                                $1002       $FF

TABLE1   RMB        7           $1003       ??
                                $1004       ??
                                $1005       ??
                                $1006       ??
                                $1007       ??
                                $1008       ??
                                $1009       ??

ERRMSG   FCC        'ERROR      $100A       $45
                                $100B       $52
                                $100C       $52
                                $100D       $4F
                                $100E       $52

         CLRA                   $100F       $4F

         END                    $100F       ??

* ——— SYMBOLTABELLE ———————————————————————
RESET   F100   INDEX   0010   MASK1   006A
CR      1000   MEMTOP  1001   TABLE1  1003
ERRMSG  100A
```

Da kein ORG angegeben ist, wird die Speicheradresse auf die Standardanfangsadresse des Assemblers gesetzt. EQU wird nicht beeinflußt.

Setzt die Anfangsadresse, wie von ORG angegeben.

FCB speichert den Operanden in dem Byte, das der Speicherzähler angibt.

Zwei Bytes werden mit FDB initialisiert.

RMB reserviert sieben Bytes (mit undefiniertem Inhalt), indem er den Speicherzähler um sieben inkrementiert.

Der ASCII-Code des Operanden-Strings wird mit der FCC-Anweisung im Speicher abgelegt.

Schließlich ein Assemblerbefehl. Da er keinen Operanden hat, besteht der Op-Code nur aus einem Byte.

Eine Anweisung, die den Speicherzähler nicht beeinflußt.

So sind die Symbole in dem Arbeitsspeicher des Assemblers untergebracht.

Dezimal	Binär
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

die Behandlung von Interrupts, auf die wir später genauer eingehen. Eins der fünf Bits – das interne Übertragsflag H – wird nur für die BCD-Arithmetik gebraucht und ist für uns daher im Augenblick nicht wichtig. Die anderen Flags haben folgende Aufgaben:

- C: Das Übertragsbit zeigt an, ob bei einer Addition oder Subtraktion ein Übertrag aufgetreten ist. Bei Verschiebungen des Akkumulatorinhalts kann es außerdem das „herausgefallene" Bit speichern. Für die Feststellung, ob eine Zahl gerade oder ungerade ist, wird beispielsweise das niederwertigste Bit nach C verschoben und dort getestet. Flag C ist Bit 0 (das niederwertigste Bit) des CC.
- V: Das Überlaufbit wird auf Eins gesetzt, wenn das Ergebnis eines arithmetischen Vorgangs zu groß für das Ergebnisregister ist. Flag V ist Bit 1 des CC.
- Z: Das Null-Flag wird auf Eins gesetzt, wenn der Inhalt eines Registers Null ist. Flag Z ist Bit 2 des CC.
- N: Das Negativ-Flag ist eine Kopie des höchstwertigen (Vorzeichen-)Bits eines Registers. Steht das Flag auf Eins, dann ist die Zahl negativ. Flag N ist Bit 3 des CC.

Verzweigungsbefehle

Eine der schwierigsten Aufgaben der Assemblersprache ist der richtige Einsatz der Status-Flags. Nicht jeder Befehl ändert die Flags. Einige Flags werden nur in Abhängigkeit vom Akkumulatorinhalt gesetzt, andere dagegen auch von anderen Registern. Am einfachsten ist es, nur den Speicherinhalt des Akkumulators zu testen und den Test an einem Punkt auszuführen, an dem die verschiedenen Flags noch nicht von dazwischenliegenden Befehlen verändert wurden.

Flags lassen sich mit Verzweigungsbefehlen testen, die eine maschinennahe Version des BASIC-Befehls GOTO sind. Der 6809 setzt fast nur relative (und kaum absolute) Verzweigungsbefehle ein. **Relative Verzweigungen** überspringen von der aktuellen Befehlsadresse an vorwärts oder rückwärts eine programmierte Anzahl Bytes, **absolute Verzweigungen** dagegen verzweigen direkt zu einer angegebenen Adresse. Beide Befehlsarten haben jedoch die gleiche Wirkung. Es gibt kurze Verzweigungen, deren Bereich in einem Byte angegeben wird (−128 bis 127), und sogenannte **Weitverzweigungen,** denen jede Adresse des Speichers zugänglich ist. Wir werden nur kurze Verzweigungen verwenden.

Der 6809 besitzt viele Verzweigungsbefehle, die wir nach und nach vorstellen werden. Die folgenden Beispiele zeigen, wie die Verzweigungsbefehle durch das Testen und Vergleichen von Akkumulatorwerten Auswahl und Wiederholung ermöglichen.

- ANDCC: Vor Einsatz des Condition Code Registers sollten routinemäßig alle Flags auf Null gesetzt werden. Zwar kann dieses Register nicht direkt mit Werten geladen werden, doch erzielt der ANDCC-Befehl die gleiche Wirkung. Wie bei dem AND-Befehl werden die Bits mit Nullen maskiert.
- SUB (SUBtrahieren): Vom Akkumulatorinhalt wird der Operand abgezogen und je nach Ergebnis das Flag C, V, Z und N gesetzt. (Bei Acht-Bit-Subtraktionen wird ebenfalls das H-Flag gesetzt.)
- CMP: Dieser Befehl arbeitet wie SUB, verändert aber nicht den Registerinhalt. Wie bei SUB werden die Flags C, V, Z und N (möglicherweise auch H) gesetzt.
- BRA (unbedingte Verzweigung): entspricht dem BASIC-Befehl GOTO.
- BGT (verzweige, falls größer als Null-Vorzeichentest): Die Verzweigung wird ausgeführt, wenn Z gleich Null ist (der Registerinhalt ist ungleich Null) und N und V entweder beide auf Null oder beide auf Eins stehen. Diese Bedingung ist erfüllt, wenn nach einer Subtraktion oder einem Vergleich der Registerinhalt größer ist als der Speicheroperand. Ähnliche Tests für Zahlen mit Vorzeichen sind BGE, BLT und BLE.
- BLO (verzweige, falls niedriger als Null): Dieser Test prüft nicht auf Vorzeichen, da es sinnlos wäre, bei Zahlen ohne Vorzeichen Flag N abzufragen. Die Verzweigung tritt ein, wenn das C-Flag gesetzt ist und damit ein negativer Übertrag nach einer Subtraktion angezeigt wird. BLS, BHI und BHS arbeiten auf ähnliche Weise.
- Das folgende Programm soll die größere von zwei Acht-Bit-Zahlen mit Vorzeichen finden. Die Zahlen sind in $3000 und $3001 gespeichert. Zunächst werden die Zahlen mit Labeln versehen:

```
NUM1 EQU $3000
NUM2 EQU $3001
ANS  EQU $3002
     ORG $1000
```

- Am Anfang des Programms werden die Flags des Condition Code Registers auf Null gesetzt, die erste Zahl geladen und mit der zweiten verglichen:

```
ANDCC #%11110000
LDA   NUM1
CMPA  NUM2
```

- Wenn NUM1 größer ist, verzweigt das Programm auf FINISH. Ist dies nicht der Fall, wird die zweite Zahl in das A-Register geladen. Die Zahl, die sich beim Sprung auf FINISH in diesem Register befindet, wird dann in ANS gespeichert. Das Programm kehrt zum Betriebssystem zurück und ENDet:

```
      BGT FINISH
      LDA NUM2
FINISH STA ANS
      SWI
      END
```

Entdeckungen

Unter den Namen Discovery 1 und 2 vertreibt der britische Hersteller Opus Supplies in Konkurrenz zum Sinclair Microdrive und zum Wafadrive von Rotronics seine Diskettenlaufwerke. Bietet sich dem Spectrum-Besitzer damit eine universelle, zuverlässige Lösung des Massenspeicherproblems?

In einem früheren Artikel stand das Wafadrive von Rotronics als Alternative zum Sinclair Microdrive zur Diskussion. Das Wafadrive schien etwas zuverlässiger, jedoch langsamer als die Microdrives und arbeitete ebenfalls mit dem Endlosbandsystem. Opus Supplies will dem Spectrum dagegen mit einem konventionellen Diskettenlaufwerk zu dem fehlenden Massenspeicher verhelfen.

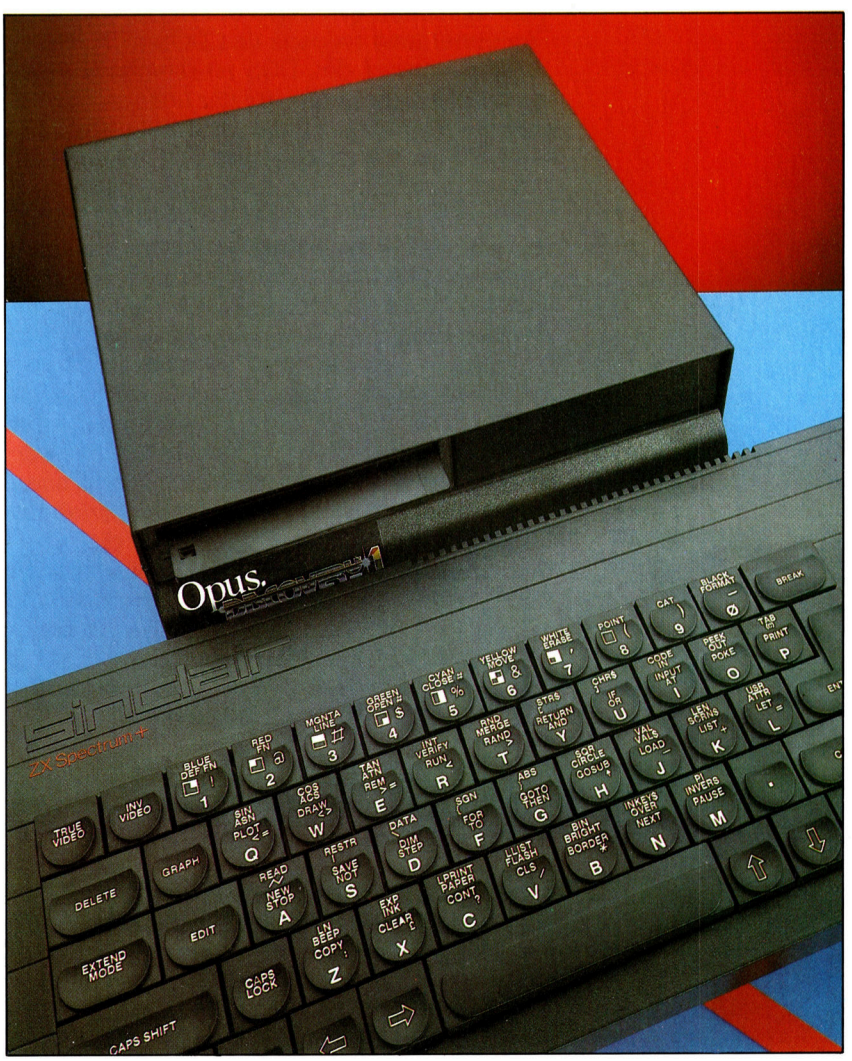
Der Einfall ist nicht neu, denn in den letzten Jahren sind etliche Diskettensysteme mit Spectrum-Interface auf den Markt gekommen, ohne sich jedoch wirklich durchsetzen zu können. Ursache dafür ist zum Teil wohl, daß diese Geräte ausschließlich in der Fachpresse erwähnt wurden, so daß viele potentielle Käufer davon nichts wußten. Erschwerend kam hinzu, daß die Produkte nur über den Versandhandel vertrieben wurden und somit keine größeren Stückzahlen abgesetzt werden konnten.

„Discovery 1" steckt in einem Blechgehäuse, dessen Bodenplatte nach vorn verlängert ist. Der Rechner wird über den Buserweiterungsstecker in eine Buchsenleiste der Diskettenstation geschoben. Das klingt recht einfach, aber bei der Installation können durchaus Schwierigkeiten auftreten. Die Kabel für Fernseher und Cassettenrecorder verhindern unter Umständen den korrekten Sitz der Steckverbindung, und das kann zu ernsthaften Schäden am Rechner führen. Bei der alten Spectrum-Version ist das Zusammenfügen der Komponenten einfacher (dafür wurde das Laufwerk auch ursprünglich entwickelt), bei dem Spectrum+ mit seinem größeren Gehäuse kann die Installation problematisch werden. Sobald der Stecker endlich richtigen Kontakt hat, übernimmt die Diskettenstation die gesamte Spannungsversorgung für Rechner und Laufwerk und macht das externe Spectrum-Netzteil überflüssig.

Auch als Doppellaufwerk

In der linken Hälfte des Discovery-1-Gehäuses ist ein 3½-Zoll-Einzellaufwerk untergebracht, während rechts Platz für den Einbau eines zweiten Laufwerks ist. Eine Doppellaufwerk-Version mit der Bezeichnung Discovery 2 ist in Vorbereitung. Wer das Discovery 1 nachträglich in ein Doppellaufwerk verwandeln möchte, kann demnächst mit einem Umrüstsatz (Discovery+) rechnen. Laut Aussage von Opus lassen sich aber ebenso 5¼-Zoll-Standardlaufwerke einbauen.

Wie Rotronics hat auch Opus sein Gerät so konzipiert, daß es nicht nur einen Massenspeicher, sondern gleichzeitig diverse Schnittstellen für den Betrieb von Druckern und weiteren Peripheriegeräten zur Verfügung stellt. Das Discovery 1 stellt an einer rückwärtigen Buchse ein Composite-Video-Signal bereit, das laut Hersteller für den Anschluß eines Monochrom-Monitors (beispielsweise bei Textverarbeitungsaufgaben) gedacht ist. Mit dem



Die Diskettenstation „Discovery 1" ist als universelle Erweiterungseinheit für den Sinclair Spectrum gedacht. Das System umfaßt nicht nur das Laufwerk mit der zugehörigen Betriebssoftware, sondern auch die erforderlichen Schnittstellen für den Anschluß von Drucker, Joystick, Farbmonitor und anderer Peripherie.

Composite-Video-Signal läßt sich natürlich auch ein Farbmonitor betreiben. Leider hat Opus keinen RGB-Ausgang eingebaut, der ein brillanteres Bild liefern würde. Das ist eigentlich schade bei einem Rechner, der für seine farbenfrohen Spiele bekannt ist.

Rechts hinten am Gehäuse befinden sich ein Kempston-kompatibler Eingang für einen Joystick nach Atari-Norm und daneben eine bidirektionale Centronics-Druckerschnittstelle. Für den Betrieb weiterer Spectrum-kompatibler Peripherie steht zudem der Buserweiterungsstecker des Rechners zur Verfügung, an den sich unter anderem ein RGB-Interface anschließen läßt.

Wie beim Wafadrive ist auch hier das Betriebssystem eng an die Interface-1-Software angelehnt, beispielsweise sind Kommandos in der Form <BEFEHL>* einzugeben. Sobald der BASIC-Interpreter auf das Zeichen * stößt, verarbeitet er den Befehl nicht als BASIC-Kommando und versucht, eine Fehlermeldung zu erzeugen. Diese wird jedoch vom DOS abgefangen (das Betriebssystem der Discovery-Diskettenstation überlagert die untersten acht KByte des Spectrum-ROM), und der Befehl wird übersetzt. Natürlich werden Kommandos mit „echten" Syntaxfehlern auch nicht vom DOS akzeptiert und lösen eine entsprechende Fehlermeldung (seitens des DOS-ROM) aus.

Beim Entwurf des DOS ist die Firma Opus hinsichtlich der Kompatibilität noch weiter gegangen als Rotronics und hat wirklich alle Microdrive-Befehle beibehalten. Dafür gibt es gute Gründe. Da die Spectrum-Eingabe für vorgegebene Einzeltasten-Befehle ausgelegt ist, läßt sich neue Betriebssoftware einfacher erstellen, wenn man die originären Kommandos des Rechners nutzt. Deshalb kann auch jeder Benutzer, der mit dem Microdrive vertraut ist, ohne großes Umlernen sofort mit dem Discovery-Laufwerk umgehen, da die gesamte Syntax identisch ist. Durch Verwendung der Rechner-spezifischen Kommandos wurde weiterhin sichergestellt, daß es nicht zu unerwarteten Kollisionen in der Speichertabelle kommt. Daher laufen Programme, die Interface-1-kompatibel sind, keineswegs immer unter einem „verbesserten" Betriebssystem – ein lästiges Problem, das bei vielen Peripheriegeräten von Drittlieferanten auftritt.

Ein/Ausgabe-Kanäle

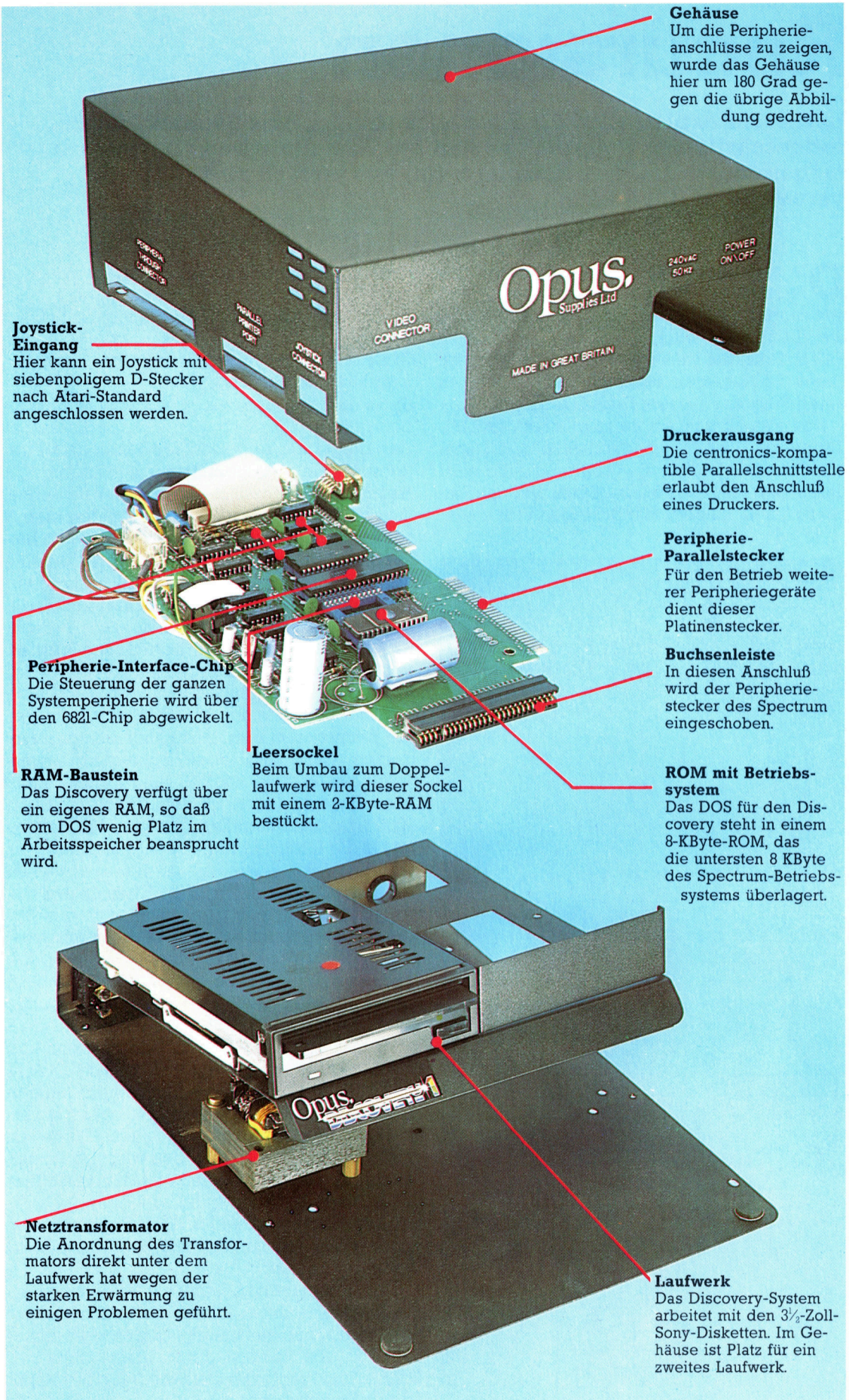
Am deutlichsten wird die enge Anlehnung an das Microdrive-System in der Organisation der Kanäle für die Ein/Ausgabe. Der Spectrum kennt 16 derartige Kanäle (von 0 bis 15 durchnumeriert). Drei davon sind für Bildschirm, Tastatur und Drucker reserviert, die übrigen für die sonstige Peripherie frei verfügbar. Die Interface-1-Befehlsliste von Sinclair spezifiziert eine Reihe von Einzelbuchstaben zur Eröffnung von Kanälen für bestimmte Geräte, zum Beispiel ‚m' für die Microdrives, was beim Discovery-System einfach übernommen worden ist. Daher arbeitet der Befehl LOAD*‚m'; 1;‚name' beim Discovery 1 genau wie beim Microdrive – ‚m' kennzeichnet aber jetzt die Floppy. Man kann den Buchstaben m beim Opus-System auch weglassen, das ist eine Erleichterung gegenüber der etwas umständlichen Sinclair-Schreibweise. Andere Befehle sind ebenfalls leicht modifiziert. So bezieht sich der Buchstabe ‚t' beim Microdrive-System auf das RS232-Interface, beim Discovery 1 dagegen auf die Drucker-Parallelschnittstelle.

Direkter Dateizugriff

Das Discovery-Laufwerk macht von den 3½-Zoll-Sony-Disketten mit doppelter Dichte (einseitig) Gebrauch, die sich bei Microcomputern zunehmend durchsetzen. Die Speicherkapazität beträgt „brutto" 250 KByte. Nach dem Formatieren bleiben 180 KByte nutzbar. Das DOS arbeitet beim Aufsuchen einer Datei mit direktem Zugriff, also erheblich schneller als andere Diskettensysteme mit seriellen Suchverfahren. Außerdem setzt die Directory keine Grenze für die Anzahl der Dateien je Diskette. Das ist sehr vorteilhaft, wenn Sie sehr viele kurze Files zu speichern haben. Der Nachteil dabei ist, daß bei kurzen Dateien viel Diskettenspeicherplatz ungenutzt bleibt.

Vergleicht man den Zeitbedarf beim Abspeichern und Laden von Dateien, braucht das Discovery 1 weniger Zeit zum Suchen, aber deutlich mehr zum Übertragen als das Microdrive. Das Aufsuchen geht bei der Diskette natürlich schneller, weil die Speicherorganisation den Direktzugriff erlaubt – bei den Microdrives ist ja systembedingt nur ein serieller Zugriff möglich. Daß die Übertragung selbst so viel langsamer geht, läßt sich mit der niedrigeren Datenrate des Opus-Systems allein (15 kBaud gegenüber 19,2 kBaud beim Microdrive) kaum erklären. Der eigentliche Vorteil des Discovery 1 liegt darin, daß es weniger störanfällig ist als das Microdrive und ein Speichermedium benutzt, das von zahlreichen Herstellern angeboten wird.

Die Einführung der Discovery-Serie ist offensichtlich sehr sorgfältig geplant worden, um dem neuen System eine möglichst große Marktchance mitzugeben. Für den Hersteller ist jetzt das Wichtigste, die Spectrum-Besitzer davon zu überzeugen, daß das Discovery-Laufwerk eine lohnendere Anschaffung für ihre Rechner ist als das Micro- oder das Wafadrive. Wenn der Zeitpunkt richtig gewählt ist und die Spectrum-Anhänger sich an ein Diskettensystem heranwagen, könnte Opus mit seiner Discovery-Serie durchaus erfolgreich sein. Die wichtigste Voraussetzung für Erfolg der Diskettenstation ist jedoch, daß sich die Software-Häuser dazu entschließen, Spectrum-Programme auf Diskette zu produzieren.



Gehäuse
Um die Peripherieanschlüsse zu zeigen, wurde das Gehäuse hier um 180 Grad gegen die übrige Abbildung gedreht.

Joystick-Eingang
Hier kann ein Joystick mit siebenpoligem D-Stecker nach Atari-Standard angeschlossen werden.

Druckerausgang
Die centronics-kompatible Parallelschnittstelle erlaubt den Anschluß eines Druckers.

Peripherie-Parallelstecker
Für den Betrieb weiterer Peripheriegeräte dient dieser Platinenstecker.

Buchsenleiste
In diesen Anschluß wird der Peripheriestecker des Spectrum eingeschoben.

Peripherie-Interface-Chip
Die Steuerung der ganzen Systemperipherie wird über den 6821-Chip abgewickelt.

Leersockel
Beim Umbau zum Doppellaufwerk wird dieser Sockel mit einem 2-KByte-RAM bestückt.

RAM-Baustein
Das Discovery verfügt über ein eigenes RAM, so daß vom DOS wenig Platz im Arbeitsspeicher beansprucht wird.

ROM mit Betriebssystem
Das DOS für den Discovery steht in einem 8-KByte-ROM, das die untersten 8 KByte des Spectrum-Betriebssystems überlagert.

Netztransformator
Die Anordnung des Transformators direkt unter dem Laufwerk hat wegen der starken Erwärmung zu einigen Problemen geführt.

Laufwerk
Das Discovery-System arbeitet mit den 3½-Zoll-Sony-Disketten. Im Gehäuse ist Platz für ein zweites Laufwerk.

Discovery 1
ABMESSUNGEN
300 x 210 x 75 mm
SCHNITTSTELLEN
Peripherie-Parallelstecker, Centronics-Parallelschnittstelle, Joystick-Eingang, Composite-Video-Buchse.
DISKETTENFORMAT
3½-Zoll-Sony-Disketten mit doppelter Dichte, einseitig.
SPEICHER-KAPAZITÄT
Insgesamt 250 KByte, formatiert 180 KByte.
Geschwindigkeit
Übertragungsrate 15 kBaud, Kopf-Positionierungszeit 3 Millisekunden/Spur.

Recursionen

In diesem Artikel wird die Recursion erklärt, eine Technik, die bei fortgeschrittenen Programmen und bei Compilern und Assemblern eingesetzt wird. Unser Beispiel zeigt, wie einfach diese Technik verwendet werden kann.

Die einfachste Definition für Recursion ist: Recursion: siehe Recursion. Diese Definition zeigt ein wichtiges Detail der Recursion – die Definition eines Ausdrucks durch sich selbst. Doch ein wichtiger Punkt wird ignoriert: Damit die Recursion funktionsfähig ist, muß es einen Weg aus diesem Kreislauf geben.

Das Rätsel, das als Beispiel dient, ist „Die Türme von Hanoi". Es beginnt mit einem Stapel von Scheiben, wobei die größte Scheibe ganz unten liegt und die kleinste oben. Um das Rätsel zu lösen, müssen alle Scheiben vom ersten Stapel auf einen zweiten Stapel verschoben werden. Dabei gelten folgende Regeln:

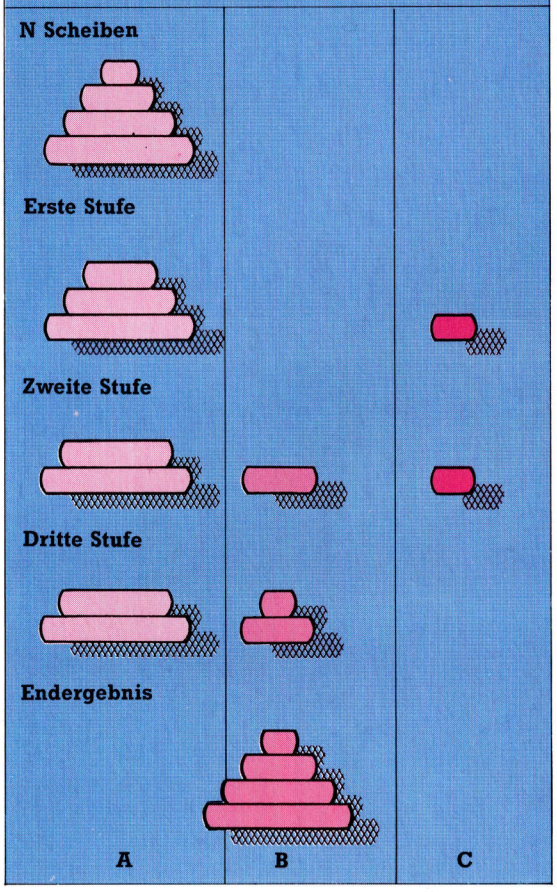
1) Es darf jeweils nur eine Scheibe bewegt werden;
2) Eine Scheibe darf nicht auf einer kleineren abgelegt werden;
3) Es darf nie mehr als drei Stapel mit Scheiben geben.

Das Diagramm zeigt, wie wir das Konzept der Recursion zur Problemlösung anwenden. Wir beginnen mit einem Stapel von vier Scheiben. Durch Zuordnen des Wertes 4 zur Variablen N geben wir die Gesamtanzahl der zu verschiebenden Scheiben an. Da nicht mehr als eine Scheibe auf einmal bewegt werden darf, verwenden wir eine recursive Formel zur Reduzierung von N um 1 und setzen die Berechnung fort, bis N=1 ist. Sobald dieser Wert erreicht ist, werden die Berechnungen beendet, und die entsprechenden Scheiben bewegt.

Wenn Sie mit einer BASIC-Version arbeiten, die Recursionen ermöglicht, läßt sich das Programm nach den eben gezeigten Regeln schreiben. In dem Acorn-Listing werden sämtliche Berechnungen in den Zeilen 1000 bis 1050 vorgenommen. Der Rest des Programms ist für die Bildschirmdarstellung zuständig.

Die Türme

Das Programm fordert den Spieler auf, die Anzahl der zu verschiebenden Scheiben einzugeben. Die zugeordnete Anzahl an Scheiben wird auf Stapel A abgelegt. Der recursive Algorithmus verschiebt dann Scheibe für Scheibe über die Stapel A, B und C. Irgendwann ist der Originalstapel auf Null reduziert, und alle Scheiben befinden sich in sequentieller Folge auf Stapel B.



Anpassung für den Spectrum

Um das Programm für den Spectrum anzupassen, muß die recursive Programmroutine gegen eine Unterroutine ab Zeile 1000 ersetzt werden. Jedesmal, wenn die Unterroutine einen recursiven Aufruf zu den Arrays M, A, B oder C macht, wird die Variable J erhöht, und die neuen Variablenwerte werden in M(J), A(J), B(J) und C(J) abgelegt. Anschließend können diese neuen Werte beim nächsten Aufruf verwendet werden, ohne die alten Werte zu löschen. Am Ende der Routine wird J reduziert, wodurch die alten Werte wieder verfügbar sind. Diese Methode kann zum Schreiben recursiver Unterroutinen in BASIC immer verwendet werden, gleichgültig, wie kompliziert die Recursion ist.

Der Programmabschnitt zur Bildschirmdarstellung verläuft geradlinig, das heißt, ein Objekt wird an seiner neuen Position gezeichnet und die alte Position durch Leerzeichen gelöscht. Das Programm stellt die Scheibenstapel in der Seitenansicht dar.

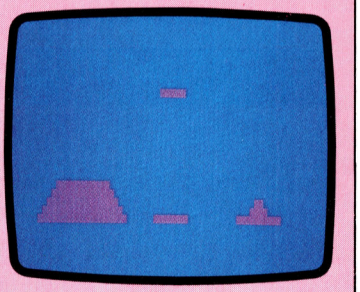
Spectrum

```
 10 DIM M(10): DIM A(10): DIM B(10): DIM C(10)
 20 DIM D$(10,10): DIM H(3): DIM P(3,10)
 30 GO SUB 3000
 90 DIM M(100): DIM A(100): DIM B(100):
  DIM C(100)
100 INPUT "HOW MANY DISCS? ";N
110 IF N<1 OR N>10 THEN  GO TO 100
120 GO SUB 3100
130 LET J=1: LET M(J)=N: LET A(J)=1: LET B(J)
  =2: LET C(J)=3
140 GO SUB 1000
200 STOP
1000 IF M(J)=1 THEN  GO SUB 1500: RETURN
1010 LET J=J+1
1020 LET M(J)=M(J-1)-1
1030 LET A(J)=A(J-1)
1040 LET B(J)=C(J-1)
1050 LET C(J)=B(J-1)
1060 GO SUB 1000
1100 LET M(J)=1
1110 LET A(J)=A(J-1)
1120 LET B(J)=B(J-1)
1130 LET C(J)=C(J-1)
1140 GO SUB 1000
1200 LET M(J)=M(J-1)-1
1210  LET A(J)=C(J-1)
1220 LET B(J)=B(J-1)
1230 LET C(J)=A(J-1)
1240 GO SUB 1000
1300 LET J=J-1
1310 RETURN
1500 LET PA=A(J): LET PB=B(J)
1510 LET M$=D$(P(PA,N+1-H(PA)))
1520 FOR I=22-H(PA) TO 7 STEP -1
1530 PRINT AT I-1,10*(PA-1);M$;
1540 PRINT AT I,10*(PA-1);B$;
1550 NEXT I
1560 FOR I=10*(PA-1) TO 10*(PB-1) STEP SGN (PB-PA)
1570 PRINT AT 6,I;M$;
1575 PRINT AT 6,I;B$;
1580 NEXT I
1590 FOR I=6 TO 20-H(PB)
1600 PRINT AT I,10*(PB-1);B$;
1610 PRINT AT I+1,10*(PB-1);M$;
1620 NEXT I
1640 LET H(PB)=H(PB)+1: LET P(PB,N+1-H(PB))=P
   (PA,N+1-H(PA))
1650 LET P(PA,N+1-H(PA))=0: LET H(PA)=H(PA)-1
1660 RETURN
3000 LET B$="           ": LET C$=CHR$ 143+CHR$
   143+CHR$ 143+CHR$ 143
3010 LET C$="": FOR I=1 TO 10: LET C$=C$+CHR$
   143: NEXT I
3020 FOR I=1 TO 9 STEP 2
3030 LET D$(I)=B$( TO 4-INT (I/2))+CHR$ 133+C$
   ( TO 2*INT (I/2))+CHR$ 138+B$( TO 4-INT (I/2))
3040 LET D$(I+1)=B$( TO 4-INT (I/2))+C$( TO
   I+1)+B$( TO 4-INT (I/2))
3050 NEXT I
3060 RETURN
3100 INK 3: PAPER 6: BORDER 6: CLS
3120 FOR I=1 TO N
3130 PRINT AT 21-N+I,0;D$(I);
3135 LET P(1,I)=I: LET P(2,I)=0: LET P(3,I)=0
3140 NEXT I
3150 LET H(1)=N: LET H(2)=0: LET H(3)=0
3160 RETURN
```

Acorn B

```
  10DIM D$(12),H(3),P(3,12)
  20PROCINIT
 100INPUT"HOW MANY DISCS (1-12)";N
 110IF N<1 OR N>12 THEN 100
 120PROCDISPLAY(N)
 130PROCHANOI(N,1,2,3)
 200END
1000DEFPROCHANOI(M,PA,PB,PC)
1010IF M=1 THEN PROCMOVE(PA,PB):
   ENDPROC
1020PROCHANOI(M-1,PA,PC,PB)
1030PROCHANOI(1,PA,PB,PC)
1040PROCHANOI(M-1,PC,PB,PA)
1050ENDPROC
1100DEFPROCMOVE(PA,PB)
1110D$=D$(P(PA,N+1-H(PA)))
1120FORI=24-H(PA) TO 10 STEP -1
1130PRINT TAB(13*(PA-1),I);B$;
1140PRINT TAB(13*(PA-1),I-1);D$;
1150NEXTI
1160FORI=13*(PA-1) TO 13*(PB-1)
         (PB-PA)
1170PRINT TAB(I,9);D$;
1180NEXT I
1190FORI=9 TO 22-H(PB)
1200PRINT TAB(13*(PB-1),I);B$
1210PRINT TAB(13*(PB-1),I+1);D$;
1220NEXT I
1240H(PB)=H(PB)+1:P(PB,N+1-H(PB))
   =P(PA,N+1-H(PA))
1250P(PA,N+1-H(PA))=0:H(PA)=H(PA)-1
1260ENDPROC
3000DEFPROCINIT
3020FOR I%=1 TO 11 STEP 2
3030D$(I%)=CHR$150+STRING$(5-
   I%DIV2," ")+CHR$234+STRING$(2*
   (I%DIV2),CHR$255)+CH R$53+STRING$
   (5-I%DIV2," ")
3040D$(I%+1)=CHR$150+STRING$(5-
   I%DIV2," ")+STRING$(I%+1,CHR$
   255)+STRING$(5-I%DI V2," ")
3050NEXT I%
3060B$=CHR$150+STRING$(12," ")
3070VDU 23,1,0;0;0;0;
3080ENDPROC
3100DEFPROCDISPLAY(N)
3110CLS
3120FORI=1 TO N
3130PRINT TAB(0,23-N+I);D$(I);
3135P(1,I)=I:P(2,I)=0:P(3,I)=0
3140NEXT I
3150H(1)=N:H(2)=0:H(3)=0
3160ENDPROC
```

Wiederkehrende Probleme
Dies ist ein Bildschirmfoto des Programms in der Spectrum-Version. Die Farbe der Scheiben läßt sich durch zusätzliche Befehle schnell ändern.



Richtiger Start

In diesem Beitrag beschäftigen wir uns mit der Palette von Lehr-Software, die für Kinder geschrieben wurde. Zunächst setzen wir uns mit den speziellen Kriterien auseinander, die ein Programmierer berücksichtigen muß, wenn er Software für Kinder im Vorschulalter und für Erstkläßler schreibt.

Die erzieherischen Ziele der folgenden Programmpakete reichen von einfacher Umriß- und Farb-Erkennung über grundlegende Zähl- und Lese-Fertigkeiten bis hin zu intensiven Versuchen, die künstlerischen Fähigkeiten eines Kindes zu fördern.

Alle Programme sind unter dem Gesichtspunkt „Bedienungsanleitung" gut. Dies ist eine der Prioritäten jedes Ausbildungsprogramms: Das Kind muß klar verstehen, worum es geht, und es muß eindeutig definierte Belohnungen geben, wenn eine Aufgabe gelöst wurde.

Ein Ausbildungsprogramm muß darüber hinaus das Interesse des Kindes wecken. Gleich, wie wichtig oder bedeutend die beabsichtigten Lernziele sind: Es wird seinen Zweck nicht erfüllen, wenn es die Fähigkeiten des Kindes überfordert oder sich wiederholt und damit langweilig wird.

Schließlich ist die Güte eines Lernprogramms daran zu messen, ob es den Stoff wirklich vermittelt. Das mag selbstverständlich klingen, doch schon oft haben Software-Häuser den Lernzweck eines Programms vergessen, weil sie dem Unterhaltungswert mehr Raum gaben.

Das faszinierendste der von uns getesteten Programme ist vielleicht „Dance Fantasy" von Fisher-Price, das für vier- bis achtjährige Kinder gedacht ist. Auf dem Bildschirm wird eine Bühne gezeigt, auf der zwei Gestalten stehen. Der Benutzer wird aufgefordert, deren Bewegungen zu choreografieren, also die Figuren nach seinen Wünschen tanzen zu lassen.

Am unteren Bildschirmrand werden die Tänzer in verschiedenen Positionen gezeigt. Jede davon stellt eine bestimmte Tanzfigur dar – einen Sprung, einen Schwung usw. Durch Führen eines der Tänzer mittels Joystick (wie einen Cursor) wählt das Kind die gewünschte Routine aus, positioniert sie dann auf der Bühne und läßt das Gewählte durch Druck auf den Action-Knopf ausführen. Ein Tanz wird so durch Auswahl einer Reihe von Bewegungen und deren Ausführung an verschiedenen Bühnenpunkten gestaltet. Dabei werden die im Programm enthaltenen Bewegungsbestandteile miteinander verbunden. Ist ein Tanz vollständig, kann das Kind ihn SAVEn und die Gesamtabfolge betrachten.

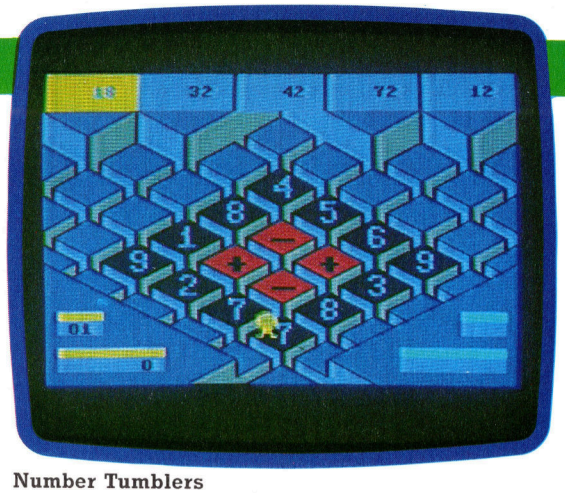
Vermittlung von Grundbegriffen

Die Stärke des Programms liegt in der lehrreichen Analogie zu einem Computer-Programm: Das Kind ist in der Lage, seinen eigenen Tanz (Programm) zu schaffen – unter Verwendung von Grund-Routinen (einer Reihe von Prozeduren). Diese werden dann auf Cassette gespeichert und anschließend dann bei Bedarf wieder geladen. Damit wird das Kind auf einfache Art und Weise mit den Begriffen des Programmierens vertraut gemacht.

„Aegean Voyage" (Reise durch die Ägäis) von Spinnaker wendet sich an eine etwas ältere Zielgruppe und verwendet Charaktere

Dance Fantasy

Aegean Voyage



Number Tumblers

und Örtlichkeiten der griechischen Mythologie als Spielelemente. Es geht darum, ein Schiff von Athen zu den verschiedenen ägäischen Inseln zu segeln und dabei Felsen und Stürmen auszuweichen. Hat man einen sicheren Hafen erreicht, wird der Name der Insel gezeigt, und am unteren Bildschirmrand erscheint eine rätselhafte Botschaft. Der Spieler muß dann entscheiden, ob er die Insel erforschen will: Eine richtige Entscheidung wird mit einem Schatz belohnt, wie etwa dem Schild des Achill. Entscheidet man sich falsch, wird das Schiff durch eine mythische Kreatur versenkt, beispielsweise durch einen Gorgonen.

Kenner der klassischen Sagen werden einige Fehler entdecken: So befindet sich beispielsweise der Minotaurus im Programm sowohl auf Kreta wie auf Delos. Im Spiel wird nicht versucht, die Bedeutung von Namen oder Orten zu erläutern. Die Kinder bekommen nicht einmal Grundkenntnisse der Sagengeschichten mit diesem Programm vermittelt. Es ist unwahrscheinlich, daß ein Kind an einem solchen Programm lange interessiert sein wird, da Grafik wie Abfolge uninteressant und langweilig sind.

„Number Tumblers" von Fisher-Price will die Kopfrechenfähigkeiten von acht- bis zwölfjährigen Kindern schulen. Am oberen Bildschirmrand befinden sich Darstellungen von Zahlenfolgen. Der Spieler muß numerische wie arithmetische Symbole auf die Oberflächen einiger Würfel bringen, um so einen mathematischen Ausdruck zu schaffen, der einer der Zahlen entspricht. Das Spiel ist schnell und unterhaltsam, verfügt über klare, gut gestaltete Grafiken und stellt einen echten Beitrag zur Verbesserung der Rechenfähigkeiten dar.

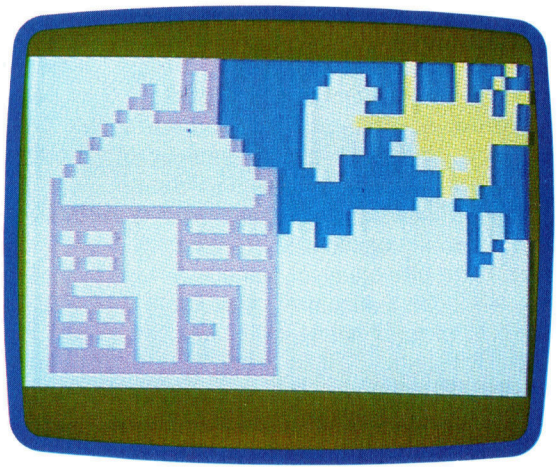
„Kindercomp" von Spinnaker wendet sich an Kinder zwischen drei und acht Jahren. Ziel des Programms ist es, Kinder an Computer heranzuführen und ihre künstlerischen Fähigkeiten zu entwickeln.

Das Originalprogramm wurde von Dr. Doug Davis für seine Tochter geschrieben, vermutlich zur Unterhaltung. Leider vermittelt Kindercomp den Eindruck, als enthalte es vor allem Computer-Tricks – eben den Stoff, den Programmierer beim Erlernen von BASIC durcharbeiten, wenn sie die Fähigkeiten des Rechners entdecken. So lautet zum Beispiel eine der Optionen „Namen". Der Anwender wird aufgefordert, einen Namen oder eine maximal 15 Charaktere umfassende Sequenz einzugeben, die anschließend in verschiedenen Farben und Größen auf dem Bildschirm dargestellt wird. Der Effekt ist attraktiv und wird für ein mit Computergrafiken nicht vertrautes Kind sehr beeindruckend sein. Der erzieherische Wert des Programms indes scheint fragwürdig, da jede Buchstabenkombination denselben Effekt bewirkt.

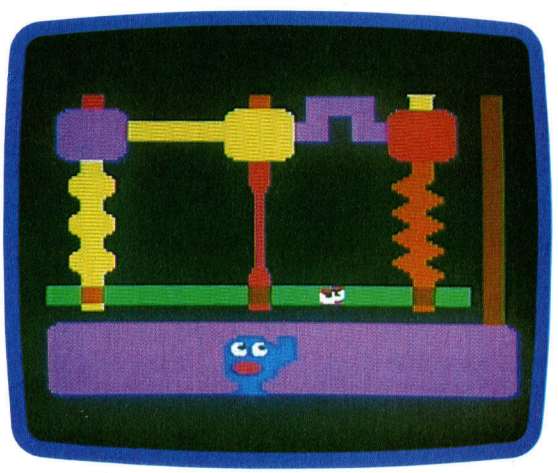
Zudem wird Kindercomp kein Kind lange beschäftigen. Die sogenannten Programmier-„Tricks" sind zwar witzig, doch bald langweilig. Es handelt sich dabei um Programme, die maximal drei Tage lang benutzt und dann wahrscheinlich nie wieder angefaßt werden.

Das letzte Programm ist für sehr kleine Kinder entwickelt worden. In „Alf In The Color Caves" (Spinnaker) agiert ein witziger kleiner Typ als Hauptdarsteller, der durch vielfarbige, unterschiedlich geformte Röhren in eine Höhle zu führen ist. Das Kind muß ihn mittels Joystick oder Tastatur sicher durch die Röhren und Höhlen bringen. Anschließend wird das Kind mit einem Tanz von Alf belohnt. Danach verschwindet er in einer Röhre, um einen neuerlichen „Abstieg" zu beginnen.

Kindercomp



Alf In The Color Caves



Streng geheim

Unsere kleine Serie über Programmiertechniken hat bisher gezeigt, wie Programme aus kurzen, größtenteils unabhängigen Teilen konstruiert werden. Wir haben uns mit der Entwicklung dieser Module befaßt und wollen nun zeigen, wie man sie bei der Entwicklung eines vollständigen Programms einsetzt.

Eine Programmstruktur besteht aus einer Basis allgemeiner Routinen, die von anderen, spezielleren Routinen verwendet und von einem Hauptmodul an der Spitze kontrolliert werden. Diese „Pyramiden"-Struktur ermöglicht den Einsatz einer Entwicklungs-Methode mit dem Namen „Top-Down-Entwurf".

Nach dieser Methode wird das oberste Kontrollprogramm zuerst entwickelt. Wir beschreiben seine Funktionen mit Aufrufen von Unterroutinen, mit deren Arbeitsweise wir uns allerdings erst später befassen. Ist diese Arbeit erledigt, gehen wir zur nächsten Stufe und beschreiben jede Routine, die vom obersten Modul aufgerufen wird. Auch hier wird jeder Teil mit Unterroutinen beschrieben. Dieser Vorgang wird so oft wiederholt, bis die unterste Stufe erreicht ist.

Als Beispiel soll das „Hangman"-Spiel dienen. Bei dieser Version muß das Programm ein Wort erraten, das sich der Spieler zuvor ausgedacht hat.

```
100 REM Initialise variables and arrays
                    '
                    '
                    '
500 REM *****Control Routine***********
510 REM
520 GOSUB 1000:REM Title & Help screens
530 GOSUB 2000:REM Set up Board
540 GOSUB 4000:REM Find word length
from player
550 GOSUB 8000:REM Select data set and
load it
560 GOSUB 3000:REM Guess a letter
570 GOSUB 4500:REM Check guess with
player
580 GOSUB 5000:REM Update the board
590 IF GAME_NOT_OVER THEN 560: REM
guess again until game is over
600 IF WIN THEN GOSUB 10000 ELSE
GOSUB 11000:REM Give appropriate
ending for win or lose
610 GOSUB 6000:REM ask the player for
another game
620 IF ANOTHER THEN 530:REM if
another then start again
630 GOSUB 7000:REM say goodbye and stop
640 END
```

Wir wissen vor dem Programmstart, daß bestimmte Dinge gemacht werden müssen: Variablen initialisieren, Arrays dimensionieren, den Bildschirmaufbau erstellen und Routinen schreiben, die die Punktzahl speichern, Wörter raten und das Spiel beenden.

Beim ersten Versuch zur Entwicklung der Kontrollroutine steht an dieser Stelle nur eine REM-Anweisung – alle Details werden später eingefügt. Die Kontrollroutine selbst besteht nur aus einigen Schleifen. Die äußere Schleife (Zeile 620) überprüft, ob der Spieler aufhören will, wogegen die innere Schleife (Zeile 590) überprüft, ob das Spiel beendet wurde.

Um die Kontrollroutine zu testen, müssen Test-Unterroutinen geschrieben werden. Jedes GOSUB in der Kontrollroutine sollte eine beschreibende REM-Anweisung enthalten. Bei Routinen mit ähnlichen Funktionen sollten einheitliche Zeilennummern (etwa 1000, 2000 usw.) verwendet werden. Dadurch wird das Austauschen von Routinen zwischen verschiedenen Programmen erheblich einfacher. So sollten Spielanweisungen beispielsweise in einer Unterroutine ab Zeile 1000 stehen, wogegen ein GOSUB 7000 die Hauptroutine aufruft und damit das Spiel beendet.

Überprüfung des Programms

Die Kontrollroutine ist sehr kurz und einfach und ist somit, im Gegensatz zu einem seitenlangen Programm, einfach zu bearbeiten. Die drei Variablen GAME NOT OVER, WIN und ANOTHER sind Flags, die in den einzelnen Unterroutinen gesetzt werden. Sie werden hier als Testwerte im Kontrollprogramm verwendet. Logikfehler sollten in dieser einfachen Routine leicht zu finden sein.

An diesem Punkt sollte man einen kritischen Blick auf die Programmstruktur werfen – wir müssen uns vergewissern, daß das Programm sich unter allen Umständen so verhält, wie es sollte. Außerdem kann man jetzt auch Verbesserungen vornehmen. So könnten Sie beispielsweise Anweisungen einbauen, mit denen sich jederzeit die Spielanweisungen oder eine Ergebnistabelle abrufen lassen.

Der nächste Schritt ist die Ausarbeitung aller vom Kontrollprogramm aufgerufenen Unterroutinen. Das Listing zeigt, wie zwei dieser Routinen aussehen könnten. Die erste (ab Zeile 4000) fragt nach einer Zahl zwischen 1 und 20 (die Wortlänge). Dazu wird eine allgemeine Unterroutine verwendet, von der wir annehmen, daß sie bei Zeile 51000 beginnt. Diese Routine nimmt eine Zeichenkette in PROMPT$,

stellt sie auf dem Bildschirm dar und akzeptiert eine numerische Eingabe.

```
4000 REM Discover word length from
player
4010 REM
4020 PROMPT$="How many letters are
there in your word ?"
4030 MIN%=1
4040 MAX%=20
4050 GOSUB 51000:REM input an integer
between MIN% & MAX%
4060 WORDLEN%=RESP%:REM RESP% is used by
the subroutine at 51000 to pass back
the response
4070 RETURN

8000 REM select data set and load it
8010 REM
8020 IF WORDLEN%>7 THEN FILE_L%=8
                   ELSE FILE_L%=WORDLEN%
8030 FILENO_L$=STR$(FILE_L%)
8040 FILENAME$="TABLE"+FILENO_L$
8050 GOSUB 9000:REM OPEN, READ & CLOSE
the file with the likelihood data for
the appropriate word length.
8060 RETURN
```

Liegt die Zahl nicht im durch MIN% und MAX% eingeschränkten Bereich, erscheinen eine Fehlermeldung und die Aufforderung zur Neueingabe. Eine solche Routine kann natürlich abgewandelt und somit in vielen Programmen eingesetzt werden.
Die andere Routine (ab Zeile 8000) verwendet lokale Variablen (FILE L% und FILENO L$). Wir nehmen an, daß die Daten zum Raten eines Buchstabens in acht Tabellen abgelegt sind. Da jedoch jeweils nur ein Datensatz im RAM sein soll, muß ein FILENAME$ generiert werden, der den Namen der Daten-Datei enthält. Danach wird Zeile 9000 aufgerufen, um die Datei einzulesen.

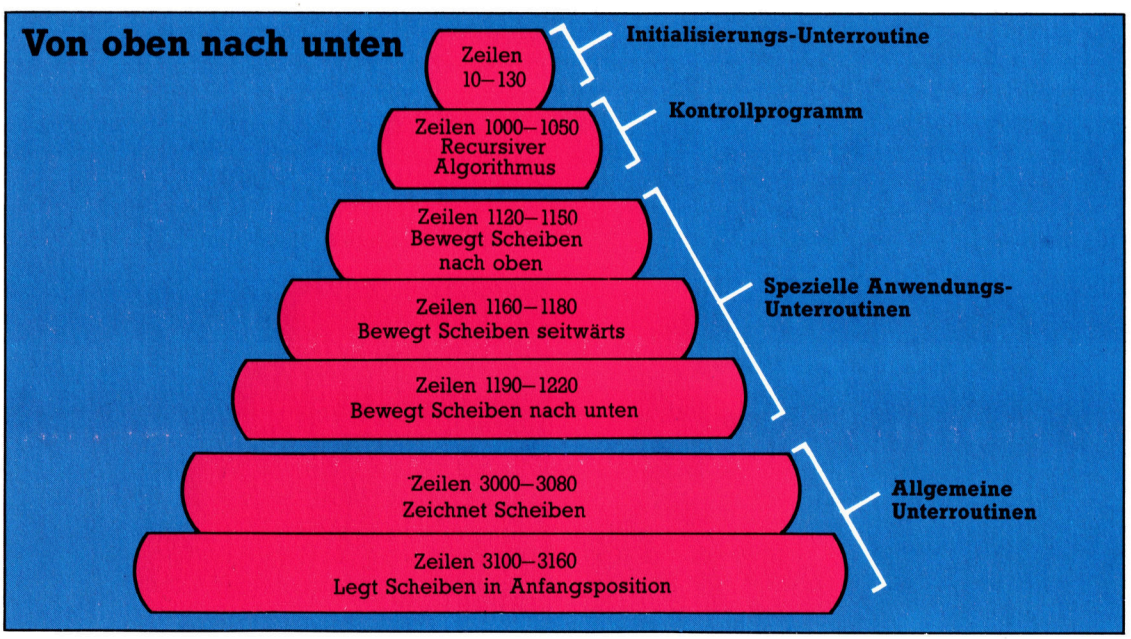
Dabei werden Sie bemerken, daß das Programm von einer zur anderen Routine verzweigt. Daher sollte eine zusätzliche Routine entwickelt werden, die die Programmteile einzeln aufruft. Dies mag als unnötige Komplikation erscheinen, doch es ermöglicht eine bessere Kontrolle über den Programmfluß. Außerdem bleiben die Programmteile getrennt und lassen sich so leichter in andere Programme integrieren.

Die Verwendung von Unterroutinen, die zwischen verschiedenen Programmen transferierbar sind, erfordert zusätzliche Arbeit, damit sie unabhängig voneinander eingesetzt werden können. Oft reicht das Austauschen von Konstanten gegen Variablen. Es ist wichtig, daß alle Routinen ausführlich beschrieben werden. Die Dokumentation sollte genaue Angaben über die Funktionsweise, die verwendeten Variablen, die erwarteten Ein- und Ausgabewerte sowie jegliche Nebeneffekte (Cursor-Positionierung, Schließen und Öffnen von Dateien usw.) enthalten.

Der Merge-Befehl

Ferner ist ein Standardformat sehr hilfreich. Sie sollten sicherstellen, daß alle Zeilennummern einen festgelegten Abstand haben, Titel und Kommentare nur in bestimmten Zeilennummern abgelegt werden, und RETURNs immer in der letzten Zeile erfolgen. Notieren Sie die erste und letzte Zeilennummer jeder Routine. Wird eine Library-Routine benötigt, stellen Sie sicher, daß entsprechende Zeilennummern unbenutzt sind, und verwenden Sie MERGE zum Integrieren der Routine in das Programm. Verfügt Ihr Computer nicht über den MERGE-Befehl, ist es eventuell möglich, einen Text-Editor zu verwenden, um im ASCII-Format gespeicherte Programme zusammenzufügen. Ist auch dies nicht möglich, muß die entsprechende Unterroutine jeweils neu eingegeben werden.

Im BASIC-Kurs wird dieses Prinzip anhand des Programms „Türme von Hanoi" (für Acorn B und Sinclair Spectrum) erläutert.



Das Diagramm zeigt das Prinzip der TOP-DOWN-Programmierung. Wir haben das Programm „Türme von Hanoi" als Beispiel verwendet. Die Zeilennummern entsprechen dem in diesem Heft gezeigten Listing für den Acorn B.

Die erste Ebene der Struktur zeigt die Initialisierung des Programms, die ausgeführt werden muß, bevor das Programm gestartet werden kann. Das Kontrollprogramm beinhaltet den recursiven Algorithmus, der die Berechnungen ausführt und bei Bedarf die Unterroutinen aufruft. Die spezifischen Anwendungs-Unterroutinen (Zeilen 1120 bis 1220) dienen zum Bewegen der Scheiben von Turm zu Turm. Die letzten zwei Abschnitte des Diagramms repräsentieren die letzten zwei Programmteile, die zur Darstellung der Grafik verwendet werden.

Bug-Byte

Der Aufstieg der Firma Bug-Byte ist eng an die Entwicklung des englischen Heimcomputermarkts gebunden. Aus dem Hersteller von Sinclair-Programmen wurde einer der führenden Spielelieferanten für die populärsten Maschinen. Zu seinen Bestsellern gehören Manic Miner und Twin Kingdom Valley.

Bug-Byte wurde im Frühjahr 1980 gegründet, als die beiden Chemiestudenten Tony Baden und Tony Milner ihre ersten Programme für den neuerstandenen ZX80 schrieben. Bald merkten sie, daß es für dieses Gerät kaum Software gab, und entschieden sich, ihre eigenen Programme anzubieten. Sie kauften 40 Leercassetten und annoncierten per Computermagazin ein Softwarepaket mit fünf Spielen. Anfangs gingen pro Woche fünfzehn Bestellungen ein. Die beiden Partner investierten den Gewinn in weitere Anzeigen und die Erstellung neuer ZX80-Programme.

Mit dem Erscheinen des ZX81 Anfang 1981 ging die Nachfrage für ZX80-Programme rapide zurück. Baden und Milner wandten sich nun der Erstellung von ZX81-Software zu. Nach ihrem Studienabschluß im Juni 1981 verlegten sie Bug-Byte in Tony Badens Heimatstadt Liverpool. Sie betrieben die Firma nun auf Vollzeitbasis, und die Anzahl der verkauften Programme verdoppelte sich in kurzer Zeit.

Weihnachten 1981 wurde der Konkurrenzkampf auf dem Spielemarkt härter. Zur Sicherung ihrer Verkaufszahlen beauftragte Bug-Byte eine Werbeagentur und gestaltete die Cassetteneinlagen und Anzeigen von nun an farbig. Eine professionelle Kopierfirma sorgte für die hohe Cassettenqualität.

Mit der neuen Aufmachung ihrer Produkte, die inzwischen über ein landesweites Händlernetz vertrieben wurden, stiegen die Verkaufszahlen sprunghaft an, und die Firma stellte weitere Mitarbeiter ein. Auch durch die zahlreichen neuen Computermodelle wuchs die Nachfrage. Bug-Byte beschäftigte daher freie Mitarbeiter, um den Bedarf decken zu können. Viele dieser Mitarbeiter gründeten später Konkurrenzunternehmen wie Quicksilva und Software Projects. Pro Cassette zahlt Bug-Byte seinen Programmierern einen festen Anteil. Spiele, die in der Bestseller-Hitparade unter den ersten zwanzig sind, bringen dem Autor nach Aussage der Firma im ersten Jahr zwischen 40 000 und 150 000 Mark ein.

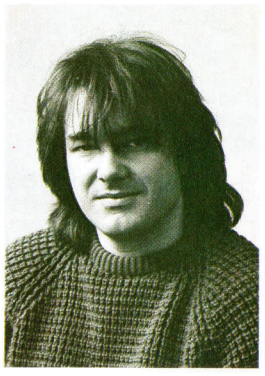
Eigene Kopierfirma

Ende 1982 verkaufte Bug-Byte seine Programme außer über die großen Ladenketten auch über 200 unabhängige Läden und stellte nach und nach seinen Versandhandel ein. Da die Kopierfirmen nicht genügend Cassetten für das Weihnachtsgeschäft herstellen konnten, gründete Bug-Byte eine eigene Kopierfirma mit dem Namen Spool. 1983 bezog Bug-Byte dann neue Räume in Liverpool.

Zu den Bestsellern von Bug-Byte zählen das Abenteuerspiel Twin Kingdom Valley (für den Acorn B und Commodore 64) und Manic Miner für den ZX Spectrum und den Commodore 64. Matthew Smith, der Autor von Manic Miner, hat inzwischen die Firma verlassen, um Software Projects zu gründen, und nahm das Copyright seines Spiels mit.

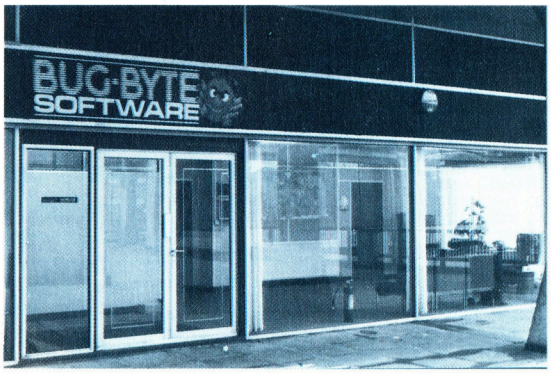
Das Wachstum von Bug-Byte setzt sich auch weiterhin fort. Inzwischen wird seine Software in fast allen westeuropäischen Ländern, Australien, Neuseeland und Südafrika verkauft. Ein kürzlich mit CBS UK geschlossener Vertrag könnte eine Ausweitung in den lukrativen amerikanischen Markt bedeuten.

Tony Milner



Tony Baden

Bug-Bytes Zentrale in Liverpool



Fachwörter von A bis Z

Gate = Gatter
Das Innenleben eines Computers besteht im wesentlichen aus einer Vielfalt elektronischer Schalter. Zur Darstellung der Bitmuster dienen Spannungspegel bzw. sehr kleine elektrische Ströme, und die Verarbeitung der Daten erfolgt, indem für diese Signale durch Öffnen und Schließen der Schalter unterschiedliche Wege freigegeben werden. Wegen dieser Funktion der digitalen Schaltkreise bezeichnet man sie als Gatter.

Vom Typ des jeweiligen Gatters hängt es ab, wie die Informationen verknüpft werden. Die Grundformen sind das AND- und das OR-Gatter, mit denen die entsprechenden logischen Operationen der Booleschen Algebra realisiert werden. Bei der Integration vieler Stufen ist es allerdings einfacher, mit NAND- und NOR-Gattern zu arbeiten.

global = global
Bei einer vergleichsweise einfachen Sprache wie BASIC sind in einem Programmteil verwendete Variablen im allgemeinen auch überall sonst im Programm zugänglich. Wenn etwa zu Beginn X der Wert 134 zugewiesen wird, dann wird bei jedem späteren Zugriff auf X derselbe eingangs mit 134 belegte Speicherplatz angesprochen. Wegen der universellen Gültigkeit spricht man dabei von „globalen" Variablen. Bei umfangreichen Programmen kann es damit aber Probleme geben, besonders bei Teamarbeit oder beim Einbeziehen von Bibliotheksprogrammen. Häufig werden dann einmal eingeführte Variablen irrtümlich für andere Zwecke nochmals verwendet – vor allem, wenn der Compiler nur

Bildnachweise
1289: Marcus Wilson-Smith
1290, 1296, 1302, 1311, 1312, 1313: Ian McKinnell
1291: Ici Plant Protection UK Division
1293: Kawalski, Computer Weekly
1295, 1301: Kevin Jones
1299: Liz Heaney
1300, 1303: Chris Stevens
1310, 1315, U3: Liz Dixon
1316: Roger Sanders

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.

Schleifenzähler mit einem Buchstaben zuläßt. Einen entscheidenden Nachteil erzeugt die globale Vereinbarung jedoch für die Programmstruktur. Beim Aufruf von unabhängigen Programmteilen wie Subroutinen und Prozeduren sollten nur die als Ein- und Ausgangsparameter angeführten Variablen aktiviert werden. Wenn aber alle Größen global sind, kann natürlich auch jede andere Variable in Mitleidenschaft gezogen werden.

Einige BASIC-Dialekte und viele andere Sprachen wie PASCAL kennen „lokale" Variablen, die nur in dem Programmsegment gelten, für das sie vereinbart wurden. Auch das hat seine Vorteile.

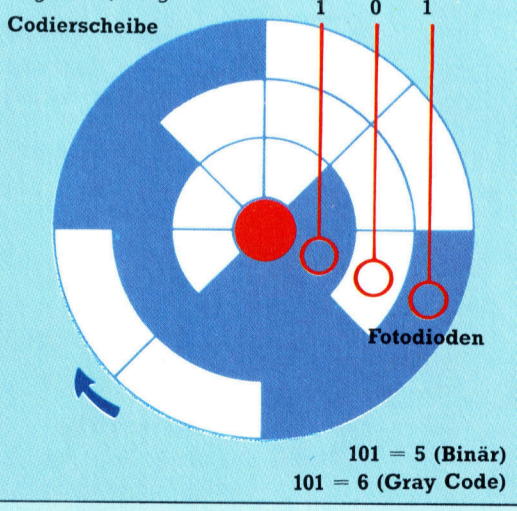
Grandfathering = Generationsprinzip
Das „Grandfathering" ist ein Sicherungsverfahren bei der Aktualisierung von Dateien. Bis zur Fertigstellung der jeweils neuesten Version (Sohn-Datei) wird nicht nur der unmittelbare Vorgänger (Vater), sondern auch der Vor-Vorgänger (Großvater = Grandfather) aufbewahrt. Sollte vor oder bei der Erzeugung der Sohn-Datei durch einen Fehler die Vater-Datei zerstört werden, läßt sich mit Hilfe des noch „lebenden" Großvaters der Vater rekonstruieren und daraus in einem neuen Anlauf der Sohn. Erst wenn der Sohn „fertiggestellt" ist, dürfen Sie die Großvaterdatei überschreiben – ihre Rolle übernimmt nun die Vaterdatei, während die Sohndatei zum Vater aufrückt.

Gray Code = Gray Code
Bei Steuerungsaufgaben muß der Rechner häufig anhand einer mitgeführten Skala die Position bewegter Objekte feststellen, beispielsweise mittels Fotodioden. Erfolgt deren Abfrage gerade dann, wenn der Markierungscode von einem Wert auf den nächsten wechselt, entsteht ein Zufallsergebnis, das unter Umständen Fehler enthält. Beim „Gray Code" sind die Skalenwerte so verschlüsselt, daß die Übergangsfehler minimal bleiben. Das Prinzip besteht darin, bei jedem Einzelschritt nur ein einziges Bit zu ändern, und dessen Stellenwert soll möglichst niedrig sein. Daraus ergibt sich folgende Verschlüsselung:

Dezimal	Binär	Gray Code
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111

Wenn der Sensor zwischen zwei benachbarte Gray-Code-Werte gerät, bleibt der Fehler – anders als beim Binärcode – stets auf eine Einheit der letzten Stelle begrenzt.

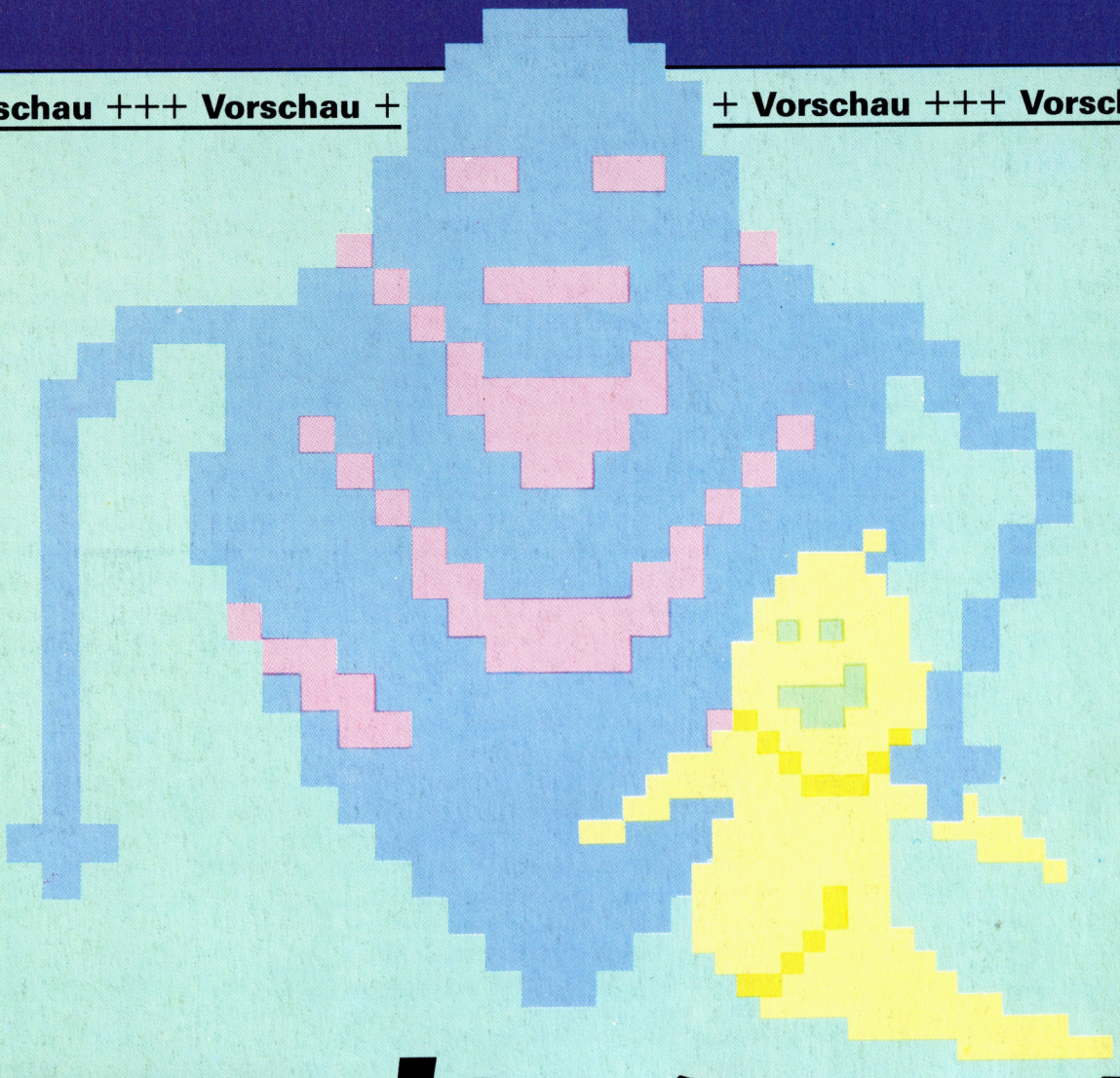
Gray-Code-Scheibe
Auf einer rotierenden Welle angebracht, ermöglicht die Scheibe, den Drehwinkel abzulesen. Die drei Ringzonen sind in jedem der acht Sektoren zur Darstellung der Gray-Code-Werte 0–7 unterschiedlich gefärbt; sie werden durch Fotodioden abgetastet. Jeder Sektor unterscheidet sich vom benachbarten nur in einem (möglichst weit außen liegenden) Ring.



101 = 5 (Binär)
101 = 6 (Gray Code)

+ Vorschau +++ Vorschau +

+ Vorschau +++ Vorschau +



computer kurs

Heft 48

Geisterstunde
Weiter geht es mit dem Abenteuerspiel in BASIC: Nun erscheinen „Geister".

Günstiger Farbcomputer
Obwohl der Tandy MC-10 zur unteren Preisklasse gehört, bietet er viel Interessantes.

Rechner-Kunst
Diese Folge über Computer-Grafik zeigt „realistische" Bilder.

Suchprozedur
In PROLOG erfolgt die Programmsteuerung mit einer Suchprozedur.

Kalibrierung
Für unser Robot-Auto muß nun das Impuls/Streckenverhältnis festgelegt werden.