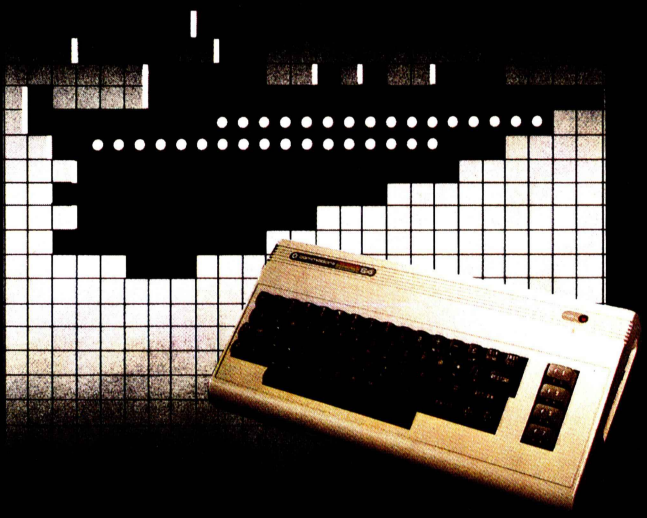
Computer Shop

Ian Stewart
Robin Jones

Commodore 64

Programmieren leichtgemacht



Birkhäuser

B

Computer Shop
Band 9

Ian Stewart / Robin Jones

Commodore 64

Programmieren leichtgemacht

Aus dem Englischen von
Tony Westermayr

Birkhäuser Verlag
Basel · Boston · Stuttgart

Die Originalausgabe erschien 1983 unter dem Titel:
"Easy Programming for the Commodore 64"
bei Shiva Publishing Ltd., Nantwich, England

Professor Ian Stewart
Mathematics Institute
University of Warwick
Coventry CA4 7AL
England, U.K.

Professor Robin Jones
Computer Unit
South Kent College of Technology
Ashford, Kent
England, U.K.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Stewart, Ian:
Commodore 64 : programmieren leicht gemacht /
Ian Stewart ; Robin Jones. Aus d. Engl. von
Tony Westermayr. – Basel ; Boston ; Stuttgart :
Birkhäuser, 1984.
(Computer-Shop ; Bd. 9)
Einheitssacht.: Easy programming for the
Commodore 64 ⟨dt.⟩
ISBN 3–7643–1588–1
NE: 28 Jones, Robin:; GT

Umschlaggestaltung: Bruckmann & Partner
Printed in Germany
ISBN 3–7643–1588–1

Inhalt

Einleitung

Dieses Buch ist gedacht für den Neuling, der sich einen Commodore 64-Mikrocomputer gekauft hat oder kaufen möchte. Für erfahrenere Aufsteiger, die von einem anderen Gerät kommen, sollte es ebenfalls von Nutzen sein. Der 64 ist ein außerordentlich leistungsstarker Computer mit modernen Einrichtungen wie einem Klangchip, Sprites-Grafik und ausgedehnten Möglichkeiten zur Erweiterung des Systems. Das hier gesteckte Ziel ist vergleichsweise bescheiden; es geht darum, in verständlichen Ausdrücken die Haupteigenschaften des Geräts und seiner Programmiersprache BASIC zu beschreiben.

Wenn Sie einen 64 kaufen, bekommen Sie gratis ein Handbuch dazu. Der Haken bei Handbüchern ist der, daß sie selten genug Platz haben, um mit Beispielen ins Einzelne zu gehen. Die richtige Informationsquelle für den 64 ist denn auch nicht das Handbuch, sondern ein dickes Nachschlagewerk von 486 Seiten mit dem Titel "Commodore 64 Programmer's Reference Guide", ein Führer also, der alles enthält bis hin zum Schaltplan und zur Anschlußbelegung der Chips . . . Eine hervorragende Sache für den erfahrenen Programmierer, aber den Anfänger erdrückt wohl eher die Masse gedrängt formulierter Information, die Fachsprache wird ihn verwirren. Unser Buch soll deshalb eine Brücke zwischen Handbuch und Reference Guide sein. Billiger als der Guide ist es auch noch!

Wir haben eine gründliche, aber verständliche Beschreibung der Grundlagen von BASIC aufgenommen, das ist die "höhere" Standard-Programmiersprache für den Commodore 64. Zugeschnitten ist das auf die besonderen Eigenheiten und Fähigkeiten des Commodore, und es enthält die wichtigsten Befehle, die für den Anfänger von Nutzen sein können. Sich mit allen Punkten auseinanderzusetzen, ist schon aus Platzgründen nicht möglich, aber wir möchten Ihnen eine tragfeste Grundlage liefern, auf der Sie mühelos weiterbauen können.

Damit man die vielseitigeren Merkmale des 64 nutzen kann, muß man ein gewisses Maß an "Theorie" im Kopf haben, vor allem die Verwendung der Binärzahlen und die Organisation des Hauptspeichers. Wir befassen uns mit den PEEK- und POKE-Befehlen näher, weil sie unentbehrlich sind, wenn Sie aus dem 64 alle die staunenswerten Leistungen herausholen wollen, die Anlaß für den Kauf gewesen sind. Der Commodore ist eine Spur anspruchsvoller als manches andere Gerät, aber wir haben uns bemüht, die Gedanken leicht faßlich darzustellen und ihre Anwendung zu erleichtern.

Der Text umfaßt eine große Zahl Programmbeispiele, von Testroutinen für neue Befehle bis hin zu längeren Programmen für Spiele, interessante Grafikdisplays, Umgang mit Daten auf Tonbandkassetten, Musik und Spritesteuerung. Die meisten Kapitel enthalten eine oder mehrere Aufgaben, die Ihre Fortschritte prüfen und zusätzliche Vorschläge liefern. Am Ende des jeweiligen Kapitels stehen die *Lösungen* der Aufgaben und können damit als Zusatzinformation genutzt werden.

Sehr wichtig sind die beiden Kapitel über *Musik* und *Sprites*. Der 64 besitzt einen eindrucksvollen Klangchip namens SID, der (neben Klangeffekten) drei Stimmen harmonisch auf verschiedenen Instrumenten erzeugen kann. Unsere Absicht ist, den SID jedem zugänglich zu machen, der klaren Kopf bewahren kann. Sprites sind vielseitig bewegliche Grafikblöcke, großartig ge-

eignet für Spiele, Animation und Grafikdisplays; gesteuert werden sie vom VIC-Chip, der so eindrucksvoll (und kompliziert) ist wie der SID. Wir erklären in vereinfachter Weise die VIC-Speicherregister, und wie man sie zur Steuerung von Sprites verwendet; dazu gibt es ein Programm für die automatische Produktion von Sprites auf dem Bildschirm und als Anhang eine Sammlung von Subroutinen für den Umgang mit Sprites.

Andere Anhänge betreffen nützliche Dinge, die in den Anhängen des Handbuchs nicht enthalten sind: so etwa Binärzahlen (können Sie sich vorstellen, daß Sie die brauchen, um Sprites zu konstruieren und Musik zu spielen?), den Speicheraufbau und nützliche Systemvariable.

Eine wichtige (und ziemlich ungewöhnliche) Eigenschaft dieses Buches ist die Kapitelfolge über *Debugging* (Fehlersuche): Wie man dahinterkommt, warum ein Programm nicht läuft, und den Fehler behebt. Jeder, der selbst Programme schreibt, wird diese Kapitel als besonders nützlich empfinden. Wir bieten auch ein paar Empfehlungen für das Verfassen gutstrukturierter Programme, vor allem für die sinnvolle Nutzung von *Subroutinen*, um das Programm in handliche Blöcke aufzuteilen.

Eine Frage, auf die das Handbuch nicht eingeht, ist die Möglichkeit hochauflösender Grafik, mit deren Hilfe man Kurven, Diagramme und Feinskizzen zeichnen kann. Wir erklären, wie man den Computer in den Hi-res-Modus (Hi-res = hochauflösend) versetzt und diese Fähigkeiten nutzt. Ein abschließendes, halbwegs schwieriges Kapitel erläutert, wie man mit *Files* Information auf Tonbandkassetten festhält.

Der Aufbau des Buches und die Auswahl des Inhalts beruhen auf unserer Erfahrung mit einer ganzen Reihe von Mikrocomputern verschiedener Hersteller. Die Programmlistings sollen jedoch keine Wunderexemplare verfeinerter Programmierung sein. Wir haben uns vielmehr bemüht, gut funktionierende Programme zu liefern, deren Eingabe in den Computer nicht eine Ewigkeit beansprucht. Wir wollen klüger machen, nicht eine Show aufziehen. Und es gibt natürlich viele, viele Eigenschaften des Geräts, die wir überhaupt nicht erwähnen, entweder weil sie eine große Wissensgrundlage erfordern, oder weil wir den Platz nicht haben. Das ist unvermeidlich (falls man nicht Unsummen für Riesenscharteken ausgeben will); aber was wir zu bieten haben, ist eine behutsame und doch gründliche Einführung in den Commodore 64, die *erklärt*, was vorgeht (Kochbuchrezepte mögen wir nicht) und sich mit den Hauptmerkmalen befaßt, die neue Anwender kennen sollten. Das eigentliche Problem ist ja doch immer, vom Boden hochzukommen.

Hinweis für Experten

Wie eben erwähnt, ist das ein Buch für Erstanwender; es wurde nicht für Fachleute geschrieben. Genauer: Die *einfachste* Weise, ein gewünschtes Ergebnis zu erzielen, ist nicht immer die *effektivste*. Wenn wir also manchmal den Eindruck erwecken, wir stellten uns ungeschickt an, trifft das wohl zu. Vor dem Laufen sollte man aber erst einmal das Gehen lernen, die einfachen Dinge zuerst, bevor man sich den komplizierteren und vielseitigeren zuwendet.

Erwarten Sie also auch nicht, daß die Programmlistings ausgefeilte Endprodukte mit der Grafikqualität von Spielhallengeräten sind, die mit atemberaubender Geschwindigkeit Wunder an Computerleistung bewirken. Hier geht es darum, einfache Listings zu liefern, die der Leser erstens in einer vernünftigen

Zeit eingeben und zweitens durcharbeiten und verstehen kann. Für das Komplizierte bleibt Zeit genug.

Hinweis für Pedanten und Grammatiker

Vor einem halben Dutzend Büchern sind wir zu dem Entschluß gekommen, wir sollten uns gemeinsam als "ich" bezeichnen. Das ist vielleicht ein bißchen ungewöhnlich, aber sehr praktisch, weil wir gelegentlich persönliche Erfahrungen mitzuteilen haben. "Wie wir heute früh zu unserer Frau sagten . . ." – also nein, das hört sich nicht gut an. "Ich" ist also eine Art Gesamtkomposition von uns beiden. "Wir" bedeutet in Hinkunft "der Leser und ich".

1 Auf los geht's los

Das Erste: Wie baut man die Hardware auf? Sie können das vermutlich schon, falls aber nicht, mögen ein paar Hinweise von Nutzen sein.

Wenn Sie an Ihren Fernseher schon einmal ein Videospiel oder einen Heimcomputer angeschlossen haben, fällt es Ihnen nicht schwer, den Commodore anzuschließen. Sollten Sie das aber noch nicht getan haben, stellen Sie vielleicht fest, daß eine genaue Schilderung Ihnen Mühe erspart.

Sie haben den Computer schon ausgepackt, weil kein normaler Mensch es fertigbrächte, ihn nach der Ankunft im Haus länger als eine halbe Minute im Paket zu lassen. Vorgefunden haben Sie demnach:

1. Den Computer.
2. Das Netzgerät, ein klobiges graues Kästchen mit zwei längeren Kabeln, an einem davon ein graues Kunststoffgebilde mit kleinen Stiften im Inneren.
3. Das Fernsehanschlußkabel, ungefähr zwei Meter lang, an beiden Enden mit Koaxialanschlüssen.
4. Das Handbuch.

Bevor Sie alles zusammenschließen, sollten Sie sich eine zivilisierte Anordnung überlegen. Falls Sie keinen Zweitfernseher verwenden, muß der Commodore

irgendwo in der Nähe Ihres Fernsehgeräts zu Hause sein. Ihn auf den Boden zu stellen, ist *keine* gute Idee; Sie laufen Gefahr, draufzutreten und den Computer zu beschädigen. Noch schlimmer: Vom verkrampften Sitzen schlafen Ihnen die Beine ein. Holen Sie also einen Tisch, vorzugsweise einen niedrigen, nehmen Sie einen Stuhl und stellen Sie beides vor den Fernseher.

Der Computer kommt auf den Tisch, das Netzgerät schließen Sie an der Buchse mit der Aufschrift POWER an. Daneben befindet sich ein schwarzer ON/OFF-Schalter (EIN/AUS); stellen Sie ihn auf OFF. Das Netzgerät schließen Sie ans Stromnetz an.

Nehmen Sie das TV-Anschlußkabel und schieben Sie das mit einem Mittelstift versehene Ende in die entsprechende Buchse an der Rückseite des Computers. Drücken Sie es fest hinein.

Das andere Ende stecken Sie in die Antennenbuchse Ihres Fernsehers.

Übrigens funktioniert der 64 bei einem Schwarzweiß-Fernseher perfekt, nur können Sie dann die Farbeinrichtungen nicht nutzen.

Manche (alten) Fernseher haben einen Drehknopf als Kanalwähler. Hier stellen Sie auf Kanal 36.

Sonst gibt es sechs oder acht Tasten für die Kanalwahl. Man nimmt in der Regel Taste 1 für das Erste Programm (ARD), Taste 2 für das Zweite (ZDF), und so weiter. Natürlich kann jede Taste auf jeden beliebigen Kanal eingestellt werden. Nehmen Sie am besten eine sonst nicht benutzte Taste (meinetwegen 6) und drücken Sie sie hinein.

Schalten Sie Fernseher und Computer ein.

Zu diesem Zeitpunkt werden Sie auf dem Bildschirm nur "Schnee" sehen, weil Sie die Feinabstimmung am Kanalwähler noch nicht vorgenommen haben. Irgendwo am Fernseher gibt es kleine Rädchen (oder Knöpfe oder sonst was), die dazu dienen. Oft sind sie hinter einer kleinen Klappe versteckt. Sollten sie Ihnen noch nie aufgefallen sein, dann suchen Sie einmal danach. Sie werden überrascht sein. Manchmal klappt der Herstellername herunter, manchmal ein ganzer Bauteil heraus, wenn man an der richtigen Stelle drückt. Sobald Sie das gefunden haben, stimmen Sie Kanal 6 (wie vorgewählt) ab, bis auf dem Bildschirm eine Mitteilung erscheint. Sie werden das Rädchen vermutlich mehrmals drehen oder die Drehrichtung ändern müssen, wenn es nicht auf Anhieb klappt. (Im übrigen hilft Ihnen sicher gern Ihr Fernsehtechniker.)

Möglicherweise müssen Sie auch Helligkeit und Kontrast und die Farbeinstellung abstimmen. Sie sind auf der Suche nach einem blauen Bildschirm mit der Meldung:

```
**** COMMODORE 64 BASIC V2 ****

64K RAM SYSTEM 38911 BASIC BYTES FREE
```

Sobald Sie das vor sich haben, ist der Computer betriebsbereit. Wenn Sie nichts herbekommen, prüfen Sie nach, ob alles richtig angeschlossen und eingeschaltet ist, und ob die Stecker fest in den Buchsen sitzen; versuchen Sie es dann erneut mit der Abstimmung. Wenn Sie an die richtige Stelle herankommen, sehen Sie, durch den Schnee schwach sichtbar, flackernde Linien. Können Sie, nachdem Sie den ganzen Abstimmungsbereich durchlaufen haben, die Miteilung immer noch nicht sehen, dann drehen Sie entweder am falschen Rädchen oder irgend etwas stimmt nicht. Falls Sie ein bißchen geschickt sind, sehen Sie sich die Anschlüsse der Kabel darauf an, ob Drähte sich gelockert haben, und/oder prüfen Sie mit Batterie und Taschenlampenbirne auf irgendwelche

Kurzschlüsse in den Kabeln. Hilft das immer noch nicht, dann setzen Sie sich mit dem Laden in Verbindung, wo Sie den Computer gekauft haben. Computer-Hardware ist sehr zuverlässig, aber ab und zu geht eben doch etwas schief.

Kassettenrecorder

Später werden Sie an Ihren 64 einen Kassettenrecorder anschließen wollen, um Programme auf Tonband zu sichern. Im Augenblick befassen wir uns damit noch nicht. Wenn Sie wissen wollen, wie das geht, schlagen Sie Kapitel 2Ø auf. (Für den Commodore gibt es auch Diskettenlaufwerke.)

Ein diskreter Rat

Es könnte vielleicht sein, daß bestimmte Mitglieder Ihres Haushalts vom 64 nicht so hellauf begeistert sind wie Sie. Man wird diese Gefühle erfolgreich verbergen bis zu dem Tag, an dem man den Fernseher einschaltet, um sich die neueste Folge von "Dallas" anzusehen, jedoch von einem mittleren Schneesturm in Farbe überrascht wird. Wenn Sie mit der Arbeit am Computer fertig sind, dann stellen Sie den Fernseher wieder auf seine Normaleinstellung zurück und vergewissern Sie sich, daß er richtig läuft. Auch ein Computer muß sich zu benehmen wissen.

2 Das Keyboard (vulgo Tastatur)

Was bewirken diese Tasten am Computer? Wenn wir sie wahllos betätigen, nichts Gescheites! Aber auf diese Weise bekommen Sie wenigstens ein Gefühl für das Keyboard.

Das Keyboard des 64 hat sehr viel Ähnlichkeit mit der Tastatur einer Schreibmaschine, die Buchstabenanordnung hält sich an die gewohnte "QWERTY"-Folge (die deutschen Tastaturen haben "QWERTZ"). Im Grunde ist das eine unsinnige Anordnung; sie stammt noch aus der Zeit, als Typenhebel sich oft verhedderten; aber die Sache hat sich so eingebürgert, daß man sich eben damit abfinden muß.

Neben Buchstaben und Ziffern gibt es Tasten mit den Aufschriften CTRL, RUN/STOP, RESTORE, RETURN und so weiter; zwei Tasten mit CRSR und Pfeilen; unten vorne einen langen Tastenbalken; vier größere abgesetzte Tasten auf der rechten Seite mit f1, f3, f5, f7; und vorne links eine Taste mit einem Symbol, das ungefähr so aussieht:

C=

Ich werde sie von jetzt an die COMMODORE-Taste nennen. Dieses Kapitel soll Sie einigermaßen mit dem Keyboard vertraut machen; wenn Sie sich auskennen, können Sie das Meiste hier überspringen.

Alphanumerische Tasten

Das sind Buchstaben und Ziffern. Wie bei einer Schreibmaschine erzeugen sie auf dem Bildschirm das entsprechende Zeichen. Experimentieren Sie eine Weile und füllen Sie den Bildschirm mit Kauderwelsch, drücken Sie aber, um sich das Dasein zu erleichtern, noch keine der anderen Tasten mit den merkwürdigen Abkürzungen.

Beachten Sie, daß das blinkende Quadrat, *Cursor* genannt, bestimmt, wo das jeweilige Zeichen hinkommt, und daß der Cursor beim Schreiben automatisch horizontal eine Stelle (und notfalls eine ganze Zeile) weiterrückt.

Hmmmm. Großartig, aber wie werden Sie den Zeichensalat wieder los?

CLR/HOME

Diese Taste befindet sich oben rechts. Wenn Sie sie drücken, stellen Sie fest, daß der Cursor nach oben links zurückgesprungen ist ("Home"-Position). Drücken Sie außerdem gleichzeitig SHIFT (Sie halten zuerst SHIFT nieder und drücken CLR/HOME), wird der Bildschirm von allen Zeichen gelöscht.

SHIFT

SHIFT kann aber viel mehr. SHIFT sorgt im Grunde dafür, daß die Wirkung anderer Tasten verändert wird.

Stehen auf einer Taste *oben* zwei Symbole, dann wählt SHIFT die obere. Beispiel:

SHIFT und Taste 3 erzeugen ein # -Zeichen

SHIFT und Taste 8 erzeugen ein (-Zeichen

Mit den alphabetischen Tasten zusammen liefert SHIFT jedoch das rechte Symbol von beiden an der *Vorderseite* der Taste. Das ist ein besonderes "PET-Grafik"-Symbol, von Taste zu Taste verschieden.

Es gibt, damit es praktischer ist, zwei SHIFT-Tasten, und mit SHIFT LOCK verhält sich der Computer so, als sei SHIFT ständig gedrückt. Um das aufzuheben, drückt man SHIFT LOCK ein zweitesmal.

COMMODORE (C=)

Das ist eigentlich eine zweite Shift-, also Umschalttaste. Damit wird das *linke* Grafikzeichen an der Tastenvorderseite gewählt. Dazu kommen noch ein paar andere, weniger klare Funktionen, die ich später erklären will, hier aber eine, auf die gleich hingewiesen werden soll.

Kleinbuchstaben

Der 64 kann neben den Großbuchstaben ABCD . . . ebensogut Kleinbuchstaben abcd . . . liefern. Um diese zu erreichen, müssen Sie die Tasten COMMODORE und SHIFT gleichzeitig drücken. Augenblicklich werden alle Großbuchstaben auf dem Schirm zu Kleinbuchstaben, alle neu getippten Buchstaben werden ebenfalls klein geschrieben. Die Grafikzeichen verändern sich ebenfalls auf recht willkürliche Weise. Es ist so: Der 64 besitzt einen doppelten Zeichenvorrat (die Menge an Symbolen, die er auf dem Bildschirm anzeigen kann), und Sie können bestimmen, welcher verwendet werden soll. Drückt man COMMODORE und SHIFT gleichzeitig ein zweitesmal, wird der alte Zustand wiederhergestellt (Großbuchstaben). Wenn Sie die genauen Veränderungen sehen wollen, schlagen Sie Anhang E des Handbuchs S. 132 nach.

Cursortasten

Die beiden CRS-Tasten mit Pfeilen bewegen den Cursor auf dem Bildschirm. Wenn Sie SHIFT nicht drücken, bewegt die linke CRSR-Taste ihn eine Stelle nach unten; die rechte bewegt ihn nach rechts. Wird gleichzeitig SHIFT gedrückt, dann bewegt die linke CRSR-Taste ihn nach oben, die rechte Taste nach links. Bei Links- oder Rechtsbewegungen springt er zu einer neuen Zeile, sobald er vom Bildschirmrand geht; bei Abwärtsbewegungen *scrollt* der Bildschirm (das heißt, er rollt als Ganzes nach oben und erzeugt neue Zeilen), wenn

Sie versuchen, über die unterste Zeile hinauszugehen. Bei Aufwärtsbewegungen läßt der Computer den Cursor nicht vom Bildschirm gehen, aber es wird nicht gescrollt.

Dadurch, daß der Cursor mit den Tasten an jede gewünschte Stelle zu bringen ist, können Sie Zeichen auf den Bildschirm schreiben, wohin Sie wollen, statt nur dort, wo die automatische Cursorbewegung es bestimmt.

RUN/STOP

Ohne SHIFT hat diese Taste die Wirkung STOP . . . was die Ausführung eines Programms zum Stehen bringt. Das ist dann nützlich, wenn Sie im Programm einen Fehler haben und nicht warten wollen, bis es abgeschlossen ist, bevor Sie ihn ausbessern können. (Bei manchen sehr häufig auftretenden Fehlern hört es vielleicht *nie* auf!)

Die Wirkung *RUN* wird erzielt durch gleichzeitiges Drücken von SHIFT, und betroffen ist das Kassettenrecordersystem, so daß Sie eine seltsame Meldung erhalten:

LOAD

PRESS PLAY ON TAPE

was uns im Augenblick nicht interessiert. Nur: Wenn Sie das aus Versehen drücken, sind Sie augenscheinlich in Schwierigkeiten, weil der Computer nicht mehr auf das Keyboard reagiert . . .

RESTORE

Aus der Schwierigkeit können Sie sich jedoch befreien, wenn Sie RUN/STOP und RESTORE gemeinsam drücken. (RESTORE allein genügt nicht.) Damit wird der Computer auf seinen normalen Ausgangszustand zurückgesetzt. Wenn Sie ein Programm im Speicher haben, geht es nicht verloren, aber anderes, wie die Bildschirmfarben, kehrt zu den Normalwerten zurück. Sobald Sie in Schwierigkeiten geraten, sollten RUN/STOP + RESTORE Ihnen heraushelfen, wobei möglichst wenig geleistete Arbeit zerstört wird. (Die Alternative, den Strom abzuschalten, löscht den gesamten Speicherinhalt.) Im gängigen Sprachgebrauch liefert RESTORE Ihnen einen "Warmstart".

INST/DEL

Ohne SHIFT ist das eine *Lösch*taste (*del*ete = löschen). Tippen Sie ein paar Buchstaben und drücken Sie DEL; Sie werden feststellen, daß der letzte Buchstabe verschwunden und der Cursor eine Stelle zurückgegangen ist. Das nächste DEL entfernt diesen Buchstaben, und so weiter. Sie können den Cursor mit den CSRS-Tasten an die gewünschte Stelle bringen und *dann* DEL benützen, um das Zeichen *unmittelbar links* neben dem Cursor loszuwerden. Zeichen rechts neben dem Cursor rücken automatisch auf, damit keine Leerstelle entsteht.

INST für *insert* (= einschieben) hat die entgegengesetzte Wirkung. Es fügt rechts neben dem Cursor Leerstellen ein und schiebt die Zeile weiter. Um das zu bewirken, drücken Sie gleichzeitig SHIFT und INST/DEL. Das wird vor allem bei der Programmbearbeitung gebraucht. Siehe dazu Kapitel 10.

CTRL

Das ist die *Control*-Taste (control = steuern). Sie ähnelt ein wenig SHIFT, weil sie die Bedeutung der gleichzeitig gedrückten Taste beeinflußt. Sie wirkt aber nur auf die oberste Tastenreihe und die Cursortasten.

Beispiel: Halten Sie CRTL nieder und drücken Sie Taste 9. Sie werden erkennen, daß die Schriftzeichen nun in *inverse video*, also in Negativschrift erscheinen (umgekehrte Farben für Vordergrund und Hintergrund). Das bedeutet das RVS ON auf Taste 9. Ebenso stellt RVS OFF auf Taste 0 die normale Farbgebung wieder her, vorausgesetzt, daß Sie 0 zusammen mit CRTL drücken.

CRTL mit den Tasten 1–8 verändert die Farbe der Symbole, wie sie auf der Tastenvorderseite angegeben ist: Schwarz, Weiß, Rot, Cyan (Hellblau), Dunkelrot, Grün, Blau, Gelb. Damit können Sie ein anders gefärbtes Display erzeugen. (Hintergrund- und Randfarben können Sie ebenfalls verändern, aber nicht so leicht. Siehe dazu Kapitel 13.)

RETURN

Diese Taste teilt dem Computer mit, "führe den eben geschriebenen Befehl aus". Bis Sie RETURN drücken, ist die Maschine im Grunde ein passiver Displayapparat, eine aufgemotzte Schreibmaschine ohne eigenen Willen. Wenn Sie ein Programm eintippen (Kapitel 5), müssen Sie nach jeder Programmzeile RETURN drücken, um sie im Speicher festzuhalten.

Probieren Sie Folgendes: Geben Sie eine Menge sinnloses Zeug ein und drücken Sie RETURN. Sie erhalten die Meldung:

```
?SYNTAX ERROR

READY
```

Das bedeutet, daß der Computer versucht hat, Ihren unsinnigen Befehl auszuführen, ihn nicht versteht und Ihnen das mitteilt. Kein Wunder! Wenn der Befehl Sinn hat, bringt die Maschine keine Syntaxfehlermeldung, sondern führt ihn aus. Probieren Sie beispielsweise den Befehl aus

```
PRINT "ETWAS" + RETURN
```

und das Wort "ETWAS" wird auch wirklich angezeigt.

Steht der Cursor auf irgendeiner Bildschirmzeile, und Sie drücken versehentlich RETURN, so wird der Cursor versuchen, diese Zeile auszuführen. In der Regel wird das Unsinn sein, und Sie erhalten einen Syntax Error – also wundern Sie sich nicht über die gelegentliche Fehlermeldung!

Andere Reaktionen sind undurchsichtiger. Tippen Sie beliebiges Zeug, drücken Sie RETURN und nehmen Sie die Fehlermeldung entgegen; dann

stellen Sie den Cursor eine Zeile hinauf, in gleiche Höhe mit dem READY, und geben erneut RETURN. Wollen Sie raten, was geschieht? Das:

```
?OUT OF DATA ERROR
READY
```

Das mag Ihnen Rätsel aufgeben. Das dumme Ding hat Ready als den (sinnvollen) Befehl READ Y gelesen, als Aufforderung, nach einer DATA-Anweisung zu suchen. Da sie nicht vorhanden ist . . . Sie sehen selbst. Ich erwähne diesen bizarren Vorgang nur, weil er viel Kopfzerbrechen bereiten kann, wenn Sie nicht zu erkennen vermögen, warum er auftritt.

Leertaste

Wie bei einer Schreibmaschine erzeugt der lange Tastenbalken unten eine Leerstelle. In diesem Buch verwende ich das Symbol:

□

für eine Leerstelle. (Suchen Sie auf dem Keyboard nicht nach einem □. Es ist nicht da. Genau deshalb habe ich es genommen!) Wenn völlig klar ist, wo Leerstellen hingehören, nehme ich mir vielleicht die Freiheit, die □-Zeichen wegzulassen.

Funktionstasten

Die großen Tasten im abgesetzten Feld rechts sind "programmierbare Funktionstasten". Wenn Sie diese drücken, kann das Gerät veranlaßt werden, auf geeignete Programmierung in der von Ihnen gewünschten Weise zu reagieren. In Kapitel 24 finden Sie nähere Einzelheiten.

Noch mehr?

Das ist natürlich bei weitem noch nicht alles. Das Keyboard des Commodore 64 kann noch eine Menge anderer Dinge. Zu Anfang wollen Sie sich aber nicht mit zu vielen Komplikationen belasten, also verschiebe ich die Feinheiten bis zu der Gelegenheit, wo sie gebraucht werden.

3 Appetitanreger

Sie möchten sich gewiß nicht durch einen ganzen Programmierkurs quälen müssen, bevor Sie erfahren; wozu der 64 fähig ist. Hier also drei kurze Programme:

Sie brauchen, um Programme zu schreiben, vom Programmieren nichts zu verstehen! Das heißt, vorausgesetzt, Sie klauen sie bei irgend jemand anderem. Eine völlig vernünftige Sache – für den Anfang das einzig Richtige, um mit dem Keyboard vertraut zu werden. Ich hoffe aber, daß Sie höhere Ambitionen haben, als nur anderer Leute Software zu fahren!

Bei jedem der drei unten aufgelisteten Programme gilt:

1. Tippen Sie NEW und drücken Sie dann die RETURN-Taste.
2. Schreiben Sie das Programmlisting sorgfältig ab. Drücken Sie nach jeder Zeile RETURN. Wenn Sie einen Fehler machen und RETURN noch nicht gedrückt haben, dann verwenden Sie die DEL-Taste, um die Zeile auszubessern; ist RETURN schon gedrückt, tippen Sie die ganze Zeile einfach noch einmal.
3. Tippen Sie LIST und RETURN, um sich zu vergewissern, daß das Listing des Programms mit dem meinen übereinstimmt.
4. Tippen Sie RUN und sehen Sie sich das Ganze an.

Da das für Sie nur gut ist, wurden die Programme als Aufgaben angelegt.

Aufgabe 1: Würmer

```
10  FOR N = 1 TO 1000
20  C = 105 + INT (4 * RND (0))
30  IF C = 108 THEN C = 117
40  PRINT CHR$ (C);
50  NEXT N
```

Aufgabe 2: Blinker

```
10  FOR N = 1 TO 100
20  X = INT (16 * RND (0))
30  POKE 53281, X
40  POKE 53280, 15 - X
```

```
5Ø  FOR T = 1 TO 1ØØ
6Ø  NEXT T
7Ø  NEXT N
```

Aufgabe 3: Kakutani

Das Kakutani-Problem, noch immer ungelöst, betrifft folgenden Vorgang. Nehmen Sie eine ganze Zahl (meinetwegen 48). Wenn sie gerade ist, wird sie halbiert. (Das ist hier der Fall, also erhalten wir 24.) Wenn ungerade, wird sie verdreifacht und 1 hinzugefügt. Wiederholen Sie das, solange Sie wollen – erhalten Sie am Ende immer 1? Beispielsweise haben wir hier:

48 → (halbieren) → 24 → (halbieren) → 12 → (halbieren) → 6
→ (halbieren) → 3 → (verdreifachen + 1) → 1Ø → (halbieren) → 5
→ (verdreifachen + 1) → 16 → (halbieren) → 8 → (halbieren) → 4
→ (halbieren) → 2 → (halbieren) → 1

und landen bei 1. Das nächste Programm ruft einen Zufallsstart hervor und rechnet die Folge aus. Es hört erst auf, wenn es bei 1 ist.

```
1Ø  N = 1ØØ + INT (1ØØØ * RND (Ø))
2Ø  PRINT N
3Ø  M = N: L = INT (M/2)
4Ø  IF M = 2 * L THEN N = L
5Ø  IF M > 1 AND M <> 2 * L THEN N = 3 * M + 1
6Ø  IF N > 1 THEN 2Ø
7Ø  PRINT 1
```

Eine schwache Ausrede

Sobald Sie mit dem Keyboard zurechtkommen, werden Sie einen Befehl oder ein ganzes Programm eintippen und fahren können, ohne irgendeine der Anweisungen darin verstanden zu haben.* Sie können sich sogar Listings aus Büchern und Zeitschriften "borgen". *Das ist völlig normal*, und Sie brauchen deshalb nicht schuldbewußt zu sein. Von Zeit zu Zeit übermannt jeden die Lust. Und es trägt dazu bei, daß Ihr Interesse erhalten bleibt und das Selbstvertrauen wächst. (Wie bei allen guten Dingen sollte man nicht übertreiben. Wenn das Jahr der Informationstechnologie nicht mehr bewirkt, als eine Generation von Menschen hervorzubringen, die anderer Leute Software abschreiben, aber

* Das haben Sie eben getan bei Aufgabe 1, 2 und 3.

keine eigene verfassen können, hätte man das Geld besser für ein Viertes Fernsehen ausgegeben.)

Manchmal, wenn ein bestimmter BASIC-Befehl vorgeführt wird, vor allem anfangs, wo wir noch nicht so viele kennen, wäre es hübsch, einen Befehl zu verwenden, der *noch nicht erklärt worden ist*. Und genau das werde ich tun, wo ich es für angebracht halte.

Reißen Sie also nicht vor Entsetzen die Arme hoch und fangen Sie nicht an zu jammern, der komische Kauz hätte das noch gar nicht erklärt. Beißen Sie die Zähne zusammen, geben Sie den Befehl trotzdem ein und beobachten Sie, was er bewirkt. Das ist der Sinn. Zu Ihrem eigenen Wohl.

Jedenfalls ist das meine Ausrede.

4 Direkte Befehle

Ein Programm ist nichts als eine Folge von Anweisungen, die gespeichert und später ausgeführt werden. Zunächst können Sie jedoch versuchen, die Befehle direkt vom Keyboard aus einzugeben. Hier spielt der 64 mit seinen Arithmetikmuskeln.

Wenn Sie dem Computer vom Keyboard aus einen Befehl erteilen, so daß er ihn nach Drücken der RETURN-Taste augenblicklich ausführt, benutzen Sie ihn im *Direktmodus*. ("Modus", zu deutsch Betriebsart, ist Computerjargon; gemeint wäre damit der "Gemütszustand" des Computers – falls er ein Gemüt hätte. Daher der Witz in einem Computerfachblatt, daß in einer bestimmten Serie von Fernsehprogrammen der Moderator die ganze Zeit im "Staunmodus" zu sein schien.) Es gibt einen zweiten, den *indirekten Modus*; hier wird der Befehl in den Speicher eingelesen und später ausgeführt. Tatsächlich wird in der Regel eine ganze Liste solcher Befehle, die ein *Programm* bilden, gespeichert. Programme besprechen wir im nächsten Kapitel; vorerst bleiben wir im Direktmodus.

Ein 64 im Direktmodus ist nicht nur eine Schreibmaschine, sondern auch ein eindrucksvoller Rechner.

Probieren Sie:

```
PRINT 2 + 2
```

(und dann RETURN). Sie sehen auf dem Bildschirm die Lösung 4. Na ja, so eindrucksvoll vielleicht auch wieder nicht, aber wenn Sie verlangen:

```
PRINT 12345678 + 87654321
```

erhalten Sie genauso schnell 99999999.

Aufgabe 1

Der Computer soll für Sie berechnen:

1. 7 + 4
2. 17 + 41
3. 5 + 16
4. 15123 + 97784

Der 64 kann auch subtrahieren:

```
PRINT 11 - 5
PRINT 77 - 3
PRINT 55555 - 22222
```

und so weiter. Zur Multiplikation müssen Sie statt des gewohnten Zeichens x das Sternchen * verwenden (weil Programmierer es sonst mit dem Buchstaben x verwechseln.) Probieren Sie:

```
PRINT 2 * 2
PRINT 2 * 3
PRINT 5 * 5
PRINT 99 * 77
```

Schließlich noch das Dividieren. Dafür wird statt ./. das Zeichen / verwendet. Wenn Sie also 24 durch 3 teilen wollen, müssen Sie verlangen:

```
PRINT 24/3
```

und um 777 durch 7 zu teilen:

```
PRINT 777/7
```

Der 64 kann nicht nur mit ganzen Zahlen wie diesen, sondern auch mit Dezimalzahlen wie 27.432 (statt des Dezimalkommas also stets der Punkt!) und negativen Zahlen wie −99 oder −27.342 umgehen. Ferner kann er außer den arithmetischen Operationen von Addition, Subtraktion, Multiplikation und Division eine ganze Reihe mathematischer Berechnungen ausführen. In *diesem* Buch hier benötigen aber alle Programme nur die Grundlagen der Arithmetik. Schließlich ist nicht jeder, der das Computern lernen möchte, ein begabter Mathematiker! Und es ist ganz erstaunlich, was man *ohne* komplizierte Mathematik alles leisten kann.

Potenzen

Für die mathematisch stärker Interessierten sollte hier jedoch noch ein Punkt erwähnt werden. Statt der üblichen Schreibweise

X^N

für die Nte Potenz von X, verwendet der 64 einen nach oben weisenden Pfeil:

X ↑ N

Beispielsweise wird die Kubikzahl von 5, also:

$5^3 = 5 * 5 * 5 = 125$

so geschrieben:

5 ↑ 3

In Kapitel 33 finden Sie einige Warnungen zu Rundungsfehlern bei Verwendung des Aufwärtspfeils.

Lösungen

Aufgabe 1

1. PRINT 7 + 4 (ergibt 11)
2. PRINT 17 + 41 (ergibt 58)
3. PRINT 5 + 16 (ergibt 21)
4. PRINT 15123 + 97784 (ergibt 112907)

5 Programme

Den Computer bewegt man dazu, das zu leisten, was Sie wünschen, indem man die erforderlichen Anweisungen in einer systematischen Liste zusammenfaßt.

Ein *Programm* ist eine Folge von Anweisungen, die der Computer ausführen soll. Es hat durchaus Ähnlichkeit mit einem Rezept aus dem Kochbuch:

Man nehme 2 Eier

schlage sie in einen Topf

rühre sie 2Ø Sekunden lang

gebe 2 Kilogramm Zucker dazu

sowie 4 Dosen braune Bohnen

und so fort. Ein Computerprogramm muß aber in einer sehr präzisen Sprache geschrieben sein.

Hier ein einfaches Programm:

```
1Ø  PRINT "HALLO!";
2Ø  GOTO 1Ø
```

Das können Sie in Ihren Computer eingeben. Zuerst tippen Sie NEW und drücken RETURN – damit werden alle Reste früherer Programme beseitigt. *Tun Sie das IMMER, bevor Sie ein neues Programm eingeben.* Dann schreiben Sie die erste Zeile:

```
1Ø  PRINT "HALLO!"
```

und drücken RETURN. Da die Anweisung mit einer *Zahl* beginnt, hier 1Ø, behandelt der Computer sie im indirekten Modus: Er speichert sie, um sie dann auszuführen, wann das verlangt wird. Nun schreiben Sie:

```
2Ø  GOTO 1Ø
```

plus RETURN. Zerbrechen Sie sich vorerst nicht den Kopf darüber, was dieses Gestammel zu *bedeuten* hat.

Damit ist es in den Speicher gelangt, aber wie weisen wir den Computer an, das ablaufen zu lassen, wie es im Jargon heißt. Wir geben den Befehl ein

```
RUN
```

(dazu RETURN, versteht sich – ich werde das künftig nicht mehr jedesmal erwähnen, erwarte aber, daß Sie nach dem Ende jeder "Zeile" eines Programms und nach jedem Befehl im Direktmodus RETURN drücken).

Unterstellt, Sie haben fehlerlos abgeschrieben, wird jetzt die Hallo-Hölle los sein. Das Wort "HALLO!" wird überall auf dem Bildschirm auftauchen und wild herumsausen. Wenn Sie nicht Einhalt gebieten, geht das ewig so weiter. Aber woher kommt das?

Die Maschine anhalten

Hier kommt die STOP-Taste zu ihrem Recht. Wenn Sie sie drücken, hält das Programm an und liefert eine Meldung, um anzugeben, wo es sich befindet. Falls Sie die Taste jetzt drücken, haben Sie den Bildschirm fast ganz voller "Hallos!" und am Ende die Meldung:

```
BREAK IN 1Ø
READY
```

oder bei Gelegenheit auch:

```
BREAK IN 2Ø
READY
```

Sie können mit STOP ein Programm immer anhalten, wenn es zu hängen scheint oder etwas falsch macht. Falls das nicht funktioniert, probieren Sie RUN/STOP plus RESTORE wie in Kapitel 2.

CONT

Um nach einem STOP weiterzumachen, können Sie den Befehl

```
CONT
```

verwenden (abgekürzt für "continue" = fahre fort). Machen Sie das, und es geht wieder los!

Was geht da vor?

In einem BASIC-Programm hat jede Anweisung eine Nummer, ihre *Zeilennummer*. Hier sind es die Zeilennummern 1Ø und 2Ø. Normalerweise führt das Gerät die Befehle in numerischer Reihenfolge der Zeilennummern aus – hier also zuerst Zeile 1Ø und dann Zeile 2Ø.

ABER: Manche Befehle haben die Wirkung, die als nächste auszuführende Zeilennummer zu verändern. Hier bedeutet der Befehl:

```
GOTO 1Ø
```

"beachte die nächste Zeile, falls vorhanden, nicht, und führe statt dessen Zeile 1Ø aus".

Um zu begreifen, was das Programm bewirkt, müssen wir noch eines wissen. Der Strichpunkt (;) nach der PRINT "HALLO!"-Anweisung teilt dem Computer mit, das *Nächste*, was anzuzeigen ist, soll unmittelbar nach dem Ende des HALLO! folgen. Wird der Strichpunkt weggelassen, dann zeigt der Computer statt dessen auf der nächsten Bildschirmzeile an.

Wird also RUN eingegeben, so tut die Maschine folgendes.

Sie sucht nach der ersten Zeile:

```
10  PRINT "HALLO!";
```

und führt sie aus, mit dem Bildschirmdisplay:

```
HALLO!
```

Da dieser Befehl *nicht* verlangt hat, der Computer solle den normalen Ablauf der Zeilennummern verändern, geht er in numerischer Reihenfolge zur *nächsten* Zeile:

```
20  GOTO 10
```

Sie führt das aus und kehrt wieder zu Zeile 10 zurück. Nun zeigt sie ein zweites HALLO! an, was so aussieht:

```
HALLO!HALLO!
```

und geht weiter zu Zeile 20, die sie aber zurückschickt zu Zeile 10.

```
HALLO!HALLO!HALLO!
```

und HALLO! ewig weiter (oder zumindest so lange, bis ein äußerer Eingriff, die STOP-Taste oder das Greisenalter oder das Ende der Welt Einhalt gebieten). Nach dem siebten Mal läuft sie vom Ende der Zeile und wickelt sich um die nächste, nach ungefähr 160 Abläufen (turns) erreicht sie die unterste Zeile, *der Bildschirm rollt nach oben*. Konkret geht das alles so schnell, daß Sie außer aufflackernden HALLOS! nichts sehen, während der Bildschirm vorbeisaust.

Um das zu verlangsamen, können Sie eine Zusatzzeile einfügen:

```
15  FOR I = 0 TO 200: NEXT I
```

(was ich in Kapitel 6 erkläre). Jetzt ist der Ablauf klar, nach dem die Anzeige stattfindet.

LIST

Wenn Sie ein Programm eingetippt haben, wollen Sie es vielleicht zu irgendeinem Zeitpunkt in ordentlicher Form auf dem Bildschirm dargeboten haben, um einen Überblick zu bekommen. (Vielleicht möchten Sie eine Zeile ändern, wissen aber die Nummer nicht mehr.) Um das zu erreichen, tippen Sie im Direktmodus den Befehl:

```
LIST
```

Probieren Sie das gleich aus.

Sie können ein Programm durch zusätzliche Zeilen ergänzen und brauchen Sie nur einzutippen. Zerbrechen Sie sich nicht den Kopf über die Reihenfolge der Zeilennummern; der Computer sortiert sie ganz automatisch in der richtigen Ordnung. Wenn Sie eingeben:

```
2Ø  PRINT "NICHT IN ORDNUNG"
1Ø  PRINT "DIESES PROGRAMM IST"
```

und dann LIST tippen, steht auf dem Schirm:

```
1Ø  PRINT "DIESES PROGRAMM IST"
2Ø  PRINT "NICHT IN ORDNUNG"
```

Ebenso können Sie eine Zeile einfach dadurch *verändern*, daß Sie eine neue mit derselben Nummer eingeben, also:

```
2Ø  PRINT "IN ORDNUNG"
```

und wieder LIST.

Um eine Zeile ganz zu löschen, schreiben Sie ihre Nummer, aber nichts sonst (außer natürlich RETURN). So löscht:

```
2Ø  (plus RETURN)
```

Zeile 2Ø. (Der Computer behandelt sie als Programmzeile ohne eigentlichen Befehl, nur als Zahl, und beachtet sie einfach nicht.) Es gibt raffiniertere Methoden, Zeilen zu verändern, siehe Kapitel 1Ø; was ich Ihnen eben erklärt habe, ist praktisch das Kleine ABC dessen, was man sich leisten kann. Gehen wir gleich weiter zu interessanteren Dingen.

Um bestimmte Zeilen aufzulisten (was bei einem langen Programm wegen Scrolling notwendig sein mag), verwenden Sie Befehle wie:

```
LIST 5Ø - 18Ø
```

worauf die Zeilen 5Ø bis 18Ø angezeigt werden.

NEW

Diesen Befehl habe ich schon in Kapitel 3 erwähnt, aber hier hat er auch seinen Platz. Um ein altes Programm aus dem Speicher zu entfernen und reinen Tisch zu machen, tippen Sie:

```
NEW
```

und, wie gewohnt, RETURN. Wenn Sie das nicht tun, lauern Reste des vorigen Programms irgendwo im Gerät und führen zu einer verblüffenden Vielfalt von unerklärlichen "bugs" (Fehlern) im Programm, von dem Sie sich froh eingebildet haben, es sei richtig eingegeben.

Sie können NEW auch *innerhalb* eines Programms verwenden, was aber lediglich dazu führt, daß das Programm sich selbst zerstört.

STOP und END

Die beiden Befehle:

```
STOP
END
```

bringen die Ausführung eines Programms zum Stillstand. Der Unterschied: Nach einem STOP können Sie mit CONT weitermachen, nach END aber nicht. Es gehört zum ordentlichen Programmieren, mit STOP oder END aufzuhören, und Sie müssen STOP vielleicht verwenden, um zu verhindern, daß ein Programm in den falschen Nummernbereich hineinläuft.

Zeilennummern in BASIC

Nicht alle höheren Sprachen verwenden Zeilennummern, um der Maschine mitzuteilen, in welcher Reihenfolge sie Befehle ausführen soll, aber *irgendeine* bestimmte Ordnung besteht immer. Manche Anweisungen dienen jedoch dazu, auf diese natürliche Ordnung einzuwirken, und schicken die Maschine damit zurück, um eine frühere Anweisung zu wiederholen (möglicherweise mit leichten Abänderungen). Genau dieses Gemisch von präzisen Instruktionen und variabler Ordnung, in der sie ausgeführt werden, macht den Computer so vielseitig.

In BASIC und in einigen anderen Sprachen ist jede Anweisung numeriert. Beim 64 können Sie Zahlen zwischen Ø und 63999 verwenden. Sie brauchen die Zeilen nicht 1, 2, 3 und so weiter zu numerieren; vielmehr ist es üblich, mit 1Ø, 2Ø, 3Ø . . . anzufangen, in Zehnerschritten. Der Sinn ist der, Platz zu lassen, wo zusätzliche Zeilen eingefügt werden können, falls Sie das später für erforderlich halten: es würde schwer halten, zwischen den Zeilen 2 und 3 eines Programms eine Zeile einzufügen! Es gibt aber keine Regel, wonach die Nummern *regelmäßig* sein müßten: Sie könnten die Zeilen (meinetwegen) 17, 18, 25, 356, 1ØØØ, 1ØØ3, 1Ø1Ø, 1Ø2Ø, 5Ø33 numerieren, wenn Sie wollen. Die meisten Programme beginnen mit ordentlich aussehenden Zeilennummern und hören auf mit ungeordneten, weil inzwischen Fehler ausgebessert worden sind.

Zeilen mit Mehrfachbefehlen

Sie können mehrere Befehle in eine einzige Programmzeile setzen, vorausgesetzt, Sie trennen sie durch Doppelpunkte (:). Beispiel:

```
1Ø  FOR T = 1 TO 2ØØ: PRINT T;: PRINT "□ MAL 999 ERGIBT □";:
    PRINT 999 * T: NEXT T
```

Das kann Platz und Zeit fürs Schreiben sparen, ist aber nicht immer eine so gute Idee, weil:

1. nur zum *Beginn* einer Zeile mit Mehrfachbefehlen gesprungen werden kann.
2. das Ändern einer Zeile bei einem Schreibfehler schwerer wird.

Es gibt jedoch Gelegenheiten, wo die Anwendung solcher Mehrfachbefehle viel Arbeit spart und keine Schwierigkeiten hervorruft, beispielsweise dann, wenn man eine ganze Reihe von Variablen zuteilt (siehe Kapitel 8):

```
1Ø  A = 1: B = 2: C = 3: D = 4: E = 5
```

In manchen meiner Listings finden Sie ab und zu Zeilen mit Mehrfachbefehlen, weshalb ich Sie darauf aufmerksam machen wollte. Man kommt aber sehr gut ohne sie aus, und sie sollten auf keinen Fall zu häufig verwendet werden, weil das Programm sonst unlesbar wird. Ein bekannter Verleger von Software für Unterrichtszwecke *untersagt* Zeilen mit Mehrfachbefehlen aus eben diesem Grund.

REM: Sag ihnen Bescheid über REM

Es gibt einen Befehl, der zuläßt, daß Sie Bemerkungen als Gedächtnishilfe (für sich oder andere) in ein Programm setzen, nämlich:

```
REM
```

Der Computer läßt in jeder Zeile unbeachtet, was nach einem REM kommt. Sie können also etwa schreiben:

```
15Ø  REM SATELLITENVERFOLGUNGSBERECHNUNG
16Ø  THETA = 52: REM BREITENGRAD
17Ø  PHI = 35: REM LÄNGENGRAD
```

Beachten Sie, daß in einer Zeile mit Mehrfachbefehlen Anweisungen nach dem REM ebenfalls nicht beachtet werden. Also:

```
2ØØ  X = 365: REM LÄNGE DES JAHRES:
     Y = 31: REM LÄNGE DES MONATS
```

setzt X auf 365, Y aber *nicht* auf 31.

Wenn Sie mit dem Programmieren anfangen, werden Sie vermutlich feststellen, daß REM-Anweisungen ein bißchen ablenken. Und sie kosten Zeit beim Schreiben. Sie können alle REM-Anweisungen *weglassen*, geraten aber in Schwierigkeiten, wenn es einen GOTO- oder GOSUB-Befehl gibt, der sich auf die REM-Zeile bezieht. Am einfachsten ist es, die REMs einzugeben und die Bemerkungen wegzulassen!

Wenn Ihr Selbstvertrauen zunimmt, werden Sie sehen, daß die REM-Anweisungen in Wirklichkeit sehr nützlich sind, und sie überall hineinsetzen. Ich habe in diesem Buch deshalb versucht, einen Kompromiß zu schließen, indem ich ein paar wichtige REMs eingefügt, sonst aber das Listing, soweit die Verständlichkeit nicht darunter litt, möglichst kurz gehalten habe. (Es gibt noch verschiedene andere Methoden der Codeverdichtung, um Listings kurz zu halten, aber sie führen zu Listings, die schwer zu lesen sind, deshalb verwende ich sie nicht.)

Abgekürzte Schlüsselwörter

Als konkretes Beispiel dafür: Die meisten BASIC-Schlüsselwörter (Befehlswörter wie PRINT, GOTO) können abgekürzt werden. So läßt sich GOTO ersetzen durch:

G SHIFT/O

Anhang D des Handbuchs, S. 13Ø, führt diese Abkürzungen auf. (*Vorsicht*: Er enthält mehrere Druckfehler: Das "G" bei GOTO und verschiedene Grafikzeichen in den Kästchen.) Da sie zu sonderbaren Listings führen, verwende ich sie hier nicht, aber Sie können es tun, wenn Sie das vorziehen: Sie sparen wirklich Zeit.

Die Aufgabe des Programmierers

Inzwischen sollte klar sein, worin die Aufgabe des Programmierers besteht. Um ein bestimmtes Ziel zu erreichen, muß der Programmierer eine Folge von Anweisungen zusammenfügen, die, wenn sie vom Computer *exakt* ausgeführt werden, das gewünschte Ergebnis erzielen.

Wichtig ist, sich darüber klarzuwerden, daß der Computer keine Ahnung hat, worin der "Zweck" eines Programms besteht. Er gehorcht einfach den Befehlen. Er ist ein blitzschnell denkender und absolut pedantischer Sklave, und wenn Sie ihm auftragen, etwas Dummes zu tun, dann gehorcht er.

Das werden Sie sicher sehr rasch entdecken, sobald Sie sich daran machen, Programme zu schreiben.

6 Schleifen

Eine grundlegende Programmiermethode veranlaßt den Computer, eine gestellte Aufgabe immer und immer wieder auszuführen. Noch besser, sie kann bei den Aufgaben auch regelmäßige Veränderungen vornehmen. Unter den Beispielen: Multiplikationstabellen.

In diesem Kapitel lernen wir den FOR . . . NEXT-Befehl kennen (der schon verwendet wurde, um den Computer bei seinen Überlegungen zu bremsen). Der Befehl beauftragt das Gerät, eine gestellte Aufgabe mehrmals hintereinander auszuführen. Das wäre nicht besonders aufregend, wenn die Aufgabe eindeutig festgelegt wäre, aber man kann dafür sorgen, daß ein paar Einzelheiten der Aufgabe sich bei jedem Schritt verändern. Die Folge ist eine ganz beachtliche Stärkung des Programmierers.

Da ich Lust habe, ein bißchen Bildung zu vermitteln, schlage ich vor, daß Sie das folgende Programm ausprobieren. Ich erkläre es, sobald Sie das getan haben.

Multiplikationstabellen

```
1Ø  PRINT CHR$ (147)          [Schirm löschen. Siehe Kapitel 7]
2Ø  PRINT "MAL SIEBEN"
3Ø  PRINT
4Ø  FOR N = 1 TO 12
5Ø  PRINT N; "X □ 7 □ = □"; 7 * N
6Ø  NEXT N
```

Fahren Sie das mit RUN. Wenn Sie keinen Fehler gemacht haben, sollten Sie sehr rasch erhalten:

```
MAL SIEBEN
 1 x 7 = 7
 2 x 7 = 14
 3 x 7 = 21
 4 x 7 = 28
 5 x 7 = 35
 6 x 7 = 42
 7 x 7 = 49
 8 x 7 = 56
```

9 x 7 = 63

1Ø x 7 = 7Ø

11 x 7 = 77

12 x 7 = 84

Wie die Schleife funktioniert

Das nennt man eine *Schleife*. Die Schleife beginnt mit dem FOR-Befehl in Zeile 4Ø und endet mit dem NEXT-Befehl in Zeile 6Ø. Dazu müssen wir noch eine Zahl N setzen, die als *Zähler* dient, und dem Computer sagen, wo er mit dem Zählen anfangen (1) und aufhören soll (12). Das geschieht alles in Zeile 4Ø:

FOR	(beginn hier mit der Schleife)
N	(verwende N als Zähler)
= 1	(Startwert für N)
TO	(geh weiter bis zu)
12	(Endwert für N)

Es geschieht folgendes: Wenn der Computer der Schleife erstmals begegnet, setzt er N gleich dem Startwert (1) und führt die Befehle aus, bis er auf NEXT stößt. Dann vergleicht er N mit dem Endwert (12), erhöht, falls N darunter liegt, N um 1 (erhält also 2) und kehrt zum Beginn der Schleife zurück, um die Befehle alle noch einmal auszuführen. Bei der nächsten Begegnung mit NEXT vergleicht er wieder und erhöht N auf 3, dann auf 4, 5, 6 . . ., bis N 12 wird. Wenn er auf NEXT stößt und feststellt, daß N den Endwert von 12 erreicht hat, verläßt er die Schleife und geht weiter zur nächsten Programmzeile (falls vorhanden) oder hört auf (falls nicht).

Ich gehe das gleich im Einzelnen durch, zuerst aber ein Wort zu Zeile 5Ø. Das ist einfach eine Reihe von aneinandergehängten PRINT-Anweisungen und führt zu Displays wie diesem:

1 x 7 = 7

Die Leerstellen □ sind nur dazu da, damit das am Ende auch hübsch aussieht. Dieses Display erscheint dann, wenn der Zähler N auf den Wert 1 gesetzt ist, und ergibt sich so:

PRINT N	PRINT 1	1
;	nicht weiter	
"X □ 7 □ = □"	PRINT "X □ 7 □ = □"	1 x 7 =
;	nicht weiter	
7 * N	PRINT 7 * 1 (also 7)	1 x 7 = 7
(kein Strichpunkt)	weiter zur nächsten Zeile	

Beachten Sie die Verwendung von PRINT N *ohne* Anführungszeichen. Wenn Sie PRINT "N" schreiben, dann zeigt er nur den einzelnen *Buchstaben* N an. Werden die Anführungszeichen weggelassen, zeigt er den *numerischen* Wert an, der N zugeteilt ist. Da N bei 1 beginnt und Schritt für Schritt auf 12 erhöht wird, hat der PRINT N-Befehl die Wirkung, je nach dem Stand in der Schleife die Zahlen 1, 2, 3 . . . 12 anzuzeigen.

Ebenso nimmt 7 * N die Werte 7 * 1 = 7, 7 * 2 = 14 . . . 7 * 12 = 84 an, so daß auch diese Zahlen der Reihe nach angezeigt werden.

Nun können wir das Programm im Ablauf durchgehen und feststellen, wie es sein Ergebnis erzielt.

1Ø	PRINT CHR$ (147)	Lösch den Schirm.
2Ø	PRINT "MAL SIEBEN"	MAL SIEBEN
3Ø	PRINT	Zeig eine Leerzeile an, damit unter der Überschrift Platz bleibt.
4Ø	FOR N = 1 TO 12	Setz eine Schleife mit N als Zähler, die von 1 (Start) bis 12 (Ende) reicht.
5Ø	PRINT N; "X □ 7 □ = □"; 7 * N	1 x 7 = 7
6Ø	NEXT N	Ist N = 12? Nein, es ist 1. Addiere 1, damit N 2 wird, und kehr zurück zu Zeile 5Ø.
5Ø	PRINT N; "X □ 7 □ = □"; 7 * N	2 x 7 = 14
6Ø	NEXT N	Ist N = 12? Nein, es ist 2. Erhöhe auf 3 und geh zurück zu 5Ø. Fahr so fort . . .
6Ø	NEXT N	N ist jetzt 12, also aus der Schleife heraus. Keine Befehle mehr: STOP.

Aufgabe 1

Verändern Sie Zeile 2Ø und 5Ø so, daß der Computer anzeigt:

1. Eine Tabelle mal fünf;
2. Eine Tabelle mal neun;

und für die Ehrgeizigen

3. Eine Tabelle mal neunundneunzig.

(Hinweis: Verändern Sie die 7 zu 5, 9 oder 99.)

Schrittgröße

Wenn Sie einen Befehl solcher Art schreiben:

```
FOR R = 3 TO 14
```

unterstellt der Computer, daß er in Einerschritten zählen soll:

3, 4, 5, 6, 7, 8, 9, 1Ø, 11, 12, 13, 14

Sie können diese Größe aber verändern, wenn Sie den Befehl STEP verwenden. Um in Zweierschritten zu zählen, würden Sie schreiben:

```
FOR R = 3 TO 14 STEP 2
```

was R der Reihe nach die Werte gäbe:

3, 5, 7, 9, 11, 13

(Da hört es auf, weil der nächste Wert 15 größer wäre als der begrenzende Wert 14 im Befehl. R wird überhaupt nicht gleich dem Wert 14 gesetzt. Hätten Sie jedoch geschrieben:

```
FOR R = 3 TO 15 STEP 2
```

dann bekäme R die Werte:

3, 5, 7, 9, 11, 13, 15

und der höchste Wert 15 im FOR-Befehl würde wirklich erreicht werden.)

Aufgabe 2

Verändern Sie Zeile 4Ø des Programms TABELLE MAL SIEBEN so, daß es nur Vielfache *gerader* Zahlen von 7 anzeigt.

Aufgabe 3

Wie Aufgabe 2, aber nur die *ungeraden* Zahlen.

Sie können sogar rückwärts zählen, wenn Sie eine *negative* Schrittgröße verwenden:

```
1Ø  PRINT CHR$ (147)
2Ø  FOR I = 1Ø TO Ø STEP -1
```

```
3Ø  PRINT I
4Ø  NEXT I
5Ø  PRINT "WELTRAUMSTART GELUNGEN"
```

Pause

Wenn der Computer eine bestimmte Zeitspanne lang pausieren soll (etwa, damit Sie Zeit haben, den Bildschirm zu lesen), können Sie eine "Tunix"-Schleife dieser Art verwenden:

```
5ØØ  FOR I = 1 TO 1ØØ
51Ø  NEXT I
```

Das vergeudet einfach Zeit, während von 1 bis 1ØØ gezählt wird, bevor die Berechnungen weitergehen. Soll Platz gespart werden, können Sie auch diese Form wählen:

```
FOR I = 1 TO 1ØØ: NEXT I
```

Die genaue Dauer der Pause hängt von der Schleifengröße ab. Soll die Pause länger sein, verändern Sie 1ØØ zu einem größeren Wert, umgekehrt zu einem kleineren. Als grober Hinweis: Jede zusätzliche Sekunde erfordert eine Erhöhung der Schleifengröße um etwa 1ØØØ. Die Verzögerung mit einer Schleife der Größe 1ØØ, wie oben, beträgt also nur rund eine Zehntelsekunde.

Abkürzung

Der Schleifenzähler kann in jedem NEXT-Befehl *weggelassen* werden. Das heißt, Sie können einfach NEXT statt NEXT I oder NEXT R oder was auch immer schreiben. Das spart Platz, aber Sie verlieren vielleicht die Übersicht, welches Next nun kommt. (Der Computer aber nicht!)

Lösungen

Aufgabe 1

1.
```
2Ø  PRINT "MAL FÜNF"
5Ø  PRINT N; "X □ 5 □ = □"; 5 * N
```

2.
```
2Ø  PRINT "MAL NEUN"
5Ø  PRINT N; "X □ 9 □ = □"; 9 * N
```

3.
```
2Ø  PRINT "MAL NEUNUNDNEUNZIG"
5Ø  PRINT N; "X □ 99 □ = □"; 99 * N
```

Aufgabe 2

```
4Ø  FOR N = 2 TO 12 STEP 2
```

Aufgabe 3

```
4Ø  FOR N = 1 TO 11 STEP 2
```

oder

```
4Ø  FOR N = 1 TO 12 STEP 2
```

(Es geht mit beiden Zeilen. Vergessen Sie aber nicht, alle anderen Zeilen wie oben aufzunehmen!)

7 TV-Ausgabe

Mit zu den ersten Dingen, die man beherrschen muß, gehört, wie der Computer dazu zu bewegen ist, auf den Fernsehbildschirm das zu schreiben, was Sie wollen.

Ein Computer nützt nicht viel, wenn er einfach dasteht und fröhlich vor sich hinsummt, mit der Außenwelt aber nicht in Verbindung treten will. Ein paar Verbindungsmethoden haben wir schon in Aktion gesehen – das Keyboard (das als *Eingabe*gerät fungiert und Information in den Computer einspeist) und den TV-Bildschirm (ein *Ausgabe*gerät, das sie wieder herausfischt). Eingabe (Input) ist, wie Sie mit dem Computer reden, Ausgabe (Output), wie er mit Ihnen redet. In diesem Kapitel befasse ich mich näher mit TV-Ausgabe; zu Eingabe siehe Kapitel 9.

Zahlen und Strings

Der 64 befaßt sich mit zwei verschiedenen Arten von Information: *Zahlen* und *Strings*. Zahlen werden in ihrer üblichen Form angezeigt, gegebenenfalls mit negativem Vorzeichen und Dezimalpunkten, etwa so:

25

−999

76.332

und so weiter. Strings sind einfach Folgen von Symbolen oder *Zeichen*, als eine Einheit betrachtet. Um das hervorzuheben, werden sie normalerweise in Anführungsstriche gesetzt:

"MARMELADE"

"CATCH − 22"

"* * * ENDE * * *"

"%&4999BCXXX/* * UNSINN + + +)"

und ähnlich.

PRINT

Das ist der grundlegende Ausgabebefehl. Print (zu deutsch drucken) brauchen Sie aber nicht wörtlich zu nehmen, selbst wenn Sie über einen Drucker verfügen. Es liefert TV-Ausgabe, kein ausgedrucktes Listing. Der PRINT-Befehl wirkt je nach dem, was anzuzeigen ist, ein bißchen verschieden.

Wenn Sie eine *Zahl* mit PRINT anzeigen wollen, etwa 42, verwenden Sie eine Programmzeile dieser Art:

```
43Ø  PRINT 42
```

Um einen *String* anzuzeigen, setzen Sie in Anführungsstriche:

```
44Ø  PRINT "MARMELADE"
```

Wenn Sie diese "Programme" ausprobieren (RETURN nicht vergessen) und mit RUN fahren, werden Sie feststellen, daß das erste auf dem Bildschirm

```
42
```

liefert, das zweite, wie man erwarten möchte:

```
MARMELADE
```

Der Unterschied: *Die Anführungsstriche um einen String werden nicht angezeigt.* So kommt bei

```
PRINT 42
```

und

```
PRINT "42"
```

dasselbe heraus. Wenn Ihnen das Ganze sinnlos erscheint, versuchen Sie es mit:

Aufgabe 1

Vergleichen Sie die Ergebnisse der folgenden Einzeiler-Programme.

1. `1Ø  PRINT 6 * 7`

und

```
1Ø  PRINT "6 * 7"
```

2. `2Ø  PRINT 4Ø + 2`

und

```
2Ø  PRINT "4Ø + 2"
```

3. `3Ø  PRINT "THE ANSWER TO THE GREAT QUESTION OF LIFE, THE UNIVERSE, AND EVERYTHING"`

und

```
30 PRINT THE ANSWER TO THE GREAT QUESTION OF LIFE, THE
   UNIVERSE, AND EVERYTHING
```

Der Unterschied zwischen der Anzeige von Zahlen und Strings durch PRINT wird wichtiger, wenn Sie die Werte von *Variablen* anzeigen wollen. Siehe Kapitel 8.

Abkürzung

Statt das Wort

```
PRINT
```

zu tippen, können Sie auch ein einzelnes Fragezeichen verwenden:

```
?
```

STATT PRINT "MENUE" können Sie also auch schreiben:

```
? "MENUE"
```

Das spart Speicherplatz und Schreibzeit gegen den Preis, nicht auf Anhieb verständlich zu sein. Ich schlage vor, daß Sie sich zuerst an PRINT gewöhnen und dann zum Fragezeichen überwechseln. Der leichteren Verständlichkeit halber verwende ich in diesem Buch stets PRINT. Der 64 hat ohnehin Speicherplatz genug!

Abgesetzte Anzeige

Einem PRINT-Befehl kann eines von drei Satzzeichen angehängt sein:

1. Strichpunkt (;)
2. Komma (,)
3. gar keines

Entsprechende Beispiele sähen dann so aus:

1. `100 PRINT "DAS MACHT EINEN BRAVEN COMPUTER SEHR FROH";`
2. `110 PRINT "DAS MACHT EINEN BRAVEN COMPUTER SEHR FROH",`
3. `120 PRINT "DAS MACHT EINEN BRAVEN COMPUTER SEHR FROH"`

Die Satzzeichen schieben nach dem Angezeigten *Leerstellen* ein und passen dadurch die nächste PRINT-Position auf dem Schirm an. Sie wirken auf Zahlen anders als auf Strings, und zwar so (gedacht ist das für Ihre Bequemlichkeit, aber ich bin da ein bißchen skeptisch):

	Zahlen	Strings
Strichpunkt	Laß eine Leerstelle	Laß keine Leerstellen
Komma	Geh zur nächsten Spalte ab Ø, 1Ø, 2Ø, 3Ø	Geh zur nächsten Spalte ab 1, 11, 21, 31
Nichts	Geh zur nächsten Zeile	Geh zur nächsten Zeile

Weil wir schon dabei sind, möchte ich erwähnen, daß einfaches

```
PRINT
```

einfach eine leere Zeile anzeigt und zur nächsten weitergeht – wie bei der Zeilenumschaltung einer Schreibmaschine. In der Computersprache wird diese Operation NEWLINE genannt.

Der Grund für diese Absetzmöglichkeiten: Man kann verschiedene *Formate* von TV-Bildschirmausgabe herstellen. Die Hauptsache ist, daß man in einem einzigen PRINT-Befehl mehr als einen Posten unterbringen kann, vorausgesetzt, die Posten werden durch Satzzeichen *getrennt*. So könnten Sie etwa schreiben:

```
5ØØ  PRINT "FRED", "LAURA"; 77, 4.22; "ROGER AND OUT"
```

und erhalten als Display:

```
FRED □ □ □ LAURA □ 77 □ □ □ □ 4.22 ROGER □ AND □
OUT
```

Sonderlich hübsch ist das ja nun nicht, aber die nächste Aufgabe liefert Besseres.

Aufgabe 2

Vergleichen Sie die Ergebnisse der folgenden Programme:

1.
```
1Ø  PRINT 1
2Ø  PRINT 2
3Ø  PRINT 4
```
2.
```
1Ø  PRINT 1, 2, 3, 4
```
3.
```
1Ø  PRINT 1; 2; 3; 4
```
4.
```
1Ø  PRINT 1,, 2,, 3,,4
```

5. 1Ø PRINT 1,, 2,, 3
   2Ø PRINT 4,, 5,, 6
6. 1Ø PRINT 1,, 2,, 3,,
   2Ø PRINT 4,, 5,, 6
7. 1Ø PRINT 1,, 2,, 3
   2Ø PRINT
   3Ø PRINT 4,, 5,, 6

8. Probieren Sie noch einmal die Programme 1–7, ersetzen Sie 1, 2, . . . 6 aber durch die Strings "A", "B", . . . "F".

9. Wie Aufgabe 8, aber nun Strings mit unterschiedlicher Länge wie "MARMELADE", "JR", "RAT", "ROTKOHL" und so weiter.

TAB

Sie können diesen Befehl dazu benützen, die Ausgabe in säuberlichen Kolonnen zu *tabulieren*. Das Bildschirmdisplay des 64 besteht aus 25 *Reihen* von

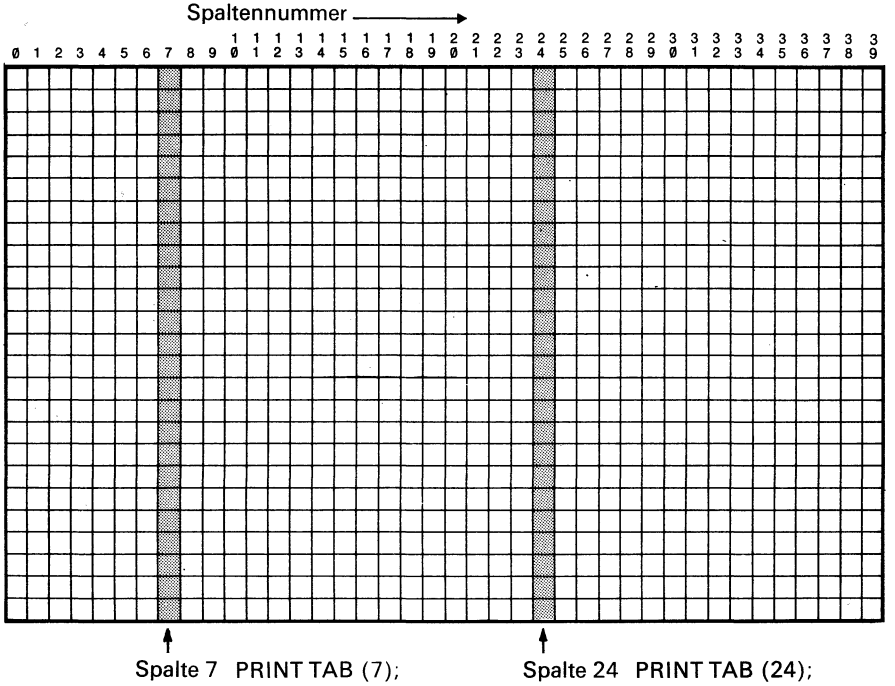


Abbildung 7.1
Das Bildschirmdisplay hat 25 Reihen zu je 40 Zeichen.
Der TAB-Befehl wählt eine Spalte für die Anzeige.

Zeichen, jede Reihe umfaßt 4Ø Zeichen. Sie können sich den Schirm also so vorstellen, daß er 4Ø *Spalten* besitzt (Abbildung 7.1). Sie sind, wie man sieht, von Ø–39 numeriert.

Die Anweisung

```
TAB (C);
```

in einem PRINT-Befehl führt dazu, daß die Anzeige an der nächsten verfügbaren Stelle in Spalte C beginnt. (Wenn Sie an Spalte C schon vorbei sind, geschieht das in der nächsten Zeile – nicht immer nach Ihren Erwartungen!) Beispielsweise führt

```
PRINT TAB (22); "WAS?"
```

zu dem Ergebnis, das Abbildung 7.2 zeigt.

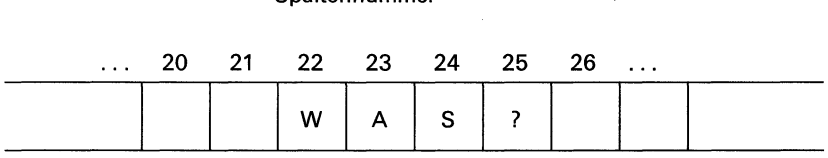


Abbildung 7.2
Wirkung von PRINT TAB (22); "WAS?"

Eine typische Verwendung von TAB zeigt nachstehendes Programm:

```
1Ø  PRINT "ZAHL"; TAB (1Ø); "QUADRAT"; TAB (2Ø;
    "WÜRFEL"; TAB (3Ø); "HOCH VIER"
2Ø  PRINT
3Ø  FOR T = 1 TO 2Ø
4Ø  PRINT T; TAB (1Ø); T * T; TAB (2Ø); T * T * T;
    TAB (3Ø); T * T * T * T
5Ø  NEXT T
```

Steuerzeichen und Zitiermodus

Probieren Sie dieses Programm aus und zerbrechen Sie sich noch nicht den Kopf über die Eigenheiten des Displays! Bringen Sie zuerst irgendeinen Zeichensalat auf den Schirm und tippen Sie dann

```
1Ø  PRINT"
```

drücken Sie gleichzeitig SHIFT und CLR/HOME und geben Sie das abschließende

"

ein, danach RETURN.

Jetzt RUN.

Sie werden sehen, daß der Bildschirm leer wird und der Cursor zur Home-Position geht – genauso, als hätten Sie SHIFT und CLR/HOME als direkten Befehl eingegeben.

Auf dieselbe Weise können mehrere direkte Befehle, in dieser Art vom Keyboard verfügar, innerhalb eines Programms eingegeben werden. Beispiel: Wenn Sie in dem obigen Beispiel statt SHIFT + CLR/HOME die Tasten CTRL + 6 drücken, werden Sie feststellen, daß nach RUN die Hintergrundfarbe grün geworden ist.

Diese Methode nennt man "Verwendung eines Steuerzeichens" oder Zitiermodus. Ein Steuerzeichen wird nicht angezeigt, es *bewirkt etwas*.

Um Sie daran zu erinnern, daß ein Steuerzeichen im Spiel ist, zeigt der 64 im Listing eines Programms trotzdem etwas an. Das ist aber ein willkürliches Symbol in Zusammenhang mit dem Steuerzeichen, zur Erinnerung gedacht. Bei CLEAR (also SHIFT + CLR/HOME) ist dieses Symbol ein Grafikzeichen in *Negativschrift*, das "Herz"-Symbol nach Spielkartenart. Bei CTRL + 6 ist es ein anderes Grafikzeichen in Negativschrift.

Das ist zwar eine nützliche Methode, die in Programmlistings aber Verwirrung stiftet (blättern Sie in den Fachzeitschriften – manche Listings sind aus diesem Grund beinahe unleserlich). Ich vermeide das also in diesem Buch möglichst und ersetze Steuerzeichen durch ihre codierte Form, nämlich

```
CHR$ (K)
```

wobei K eine Codezahl, der ASCII-Code, für das Zeichen ist (siehe Kapitel 2Ø). Die Codes sind aufgeführt in Anhang F des Handbuchs, S. 135–137. Wenn Sie diese Liste durchgehen, entdecken Sie beispielsweise CLR/HOME zweimal als Zeichencodes 19 und 147. Der erste ist die Version ohne Shift (Cursor HOME), die zweite mit Shift (CLEAR Bildschirm und Cursor HOME). So hat

```
PRINT CHR$ (147)
```

dieselbe Wirkung wie das oben erwähnte Herz im Zitiermodus. Ebenso kann mit

```
PRINT CHR$ (3Ø)
```

grüne Hintergrundfarbe erreicht werden. Ich empfehle, sich zu Anfang darüber keine großen Gedanken zu machen. Behalten Sie lediglich im Gedächtnis, daß PRINT CHR$ (K) oft verwendet wird, um aus einem Programm heraus solche Wirkungen zu erzielen.

Anzeigen in einer vorgewählten Reihe

Durch die Verwendung von Steuerzeichen können Sie vom Computer an jedem gewünschten Bildschirmpunkt ein Zeichen anzeigen lassen. Wie TAB eine Spalte auswählt, habe ich schon gezeigt, aber wie bestimmen wir die Reihe?

Hier ist ein Weg. Nehmen Sie zuerst CHR$ (19), um den Cursor nach oben zu bringen, ohne den Bildschirm zu löschen. Dann verwenden Sie eine Schleife, die mehrmals PRINT CHR$ (17) enthält, um den Cursor herunterzuführen. (Das ist der Code des Steuerzeichens für CURSOR ABWÄRTS.) Wollen Sie also zu Beginn von Reihe 15 anzeigen, schreiben Sie:

```
5ØØ  PRINT CHR$ (19);
51Ø  FOR T = 1 TO 15: PRINT CHR$ (17);: NEXT T
52Ø  PRINT "DAS IST IN REIHE 15"
```

Beachten Sie die Strichpunkte. Ohne sie werden viel mehr Reihen übersprungen. Wenn Sie in Spalte 8, Reihe 15, mit PRINT anzeigen wollen, verändern Sie obige Zeile 52Ø zu:

```
52Ø  PRINT TAB (8); "DAS IST IN REIHE 15 SPALTE 8"
```

Dieselbe Wirkung kann dadurch erreicht werden, daß man auf den entsprechenden Steuertasten im Zitiermodus schreibt, also innerhalb der Anführungsstriche eines PRINT ". . ."-Befehls wie oben. Es gibt noch einen völlig anderen Weg ohne Zitiermodus oder Steuerzeichen, den ich in Kapitel 19 erläutere. Ich persönlich ziehe ihn vor, aber Ihr Geschmack ist vielleicht ein anderer.

Lösungen

Aufgabe 1

1. 42

   6 * 7

2. 42

   4Ø + 2

3. THE ANSWER TO THE GREAT QUESTION OF LIFE, THE UNIVERSE AND EVERYTHING

   Ø

   ? SYNTAX ERROR

(Hier hat das TO einen Syntaxfehler hervorgerufen: THE ANSWER ist als Variable gesehen worden!)

Aufgabe 2

1. ☐ 1

   ☐ 2

□3
□4

2. □1□□□□□□□□□2□□□□□□□□□3
□□□□□□□□□4

3. □1□□2□□3□□4

4. □1□□□□□□□□□□□□□□□□□□□□2
□3□□□□□□□□□□□□□□□□□□□□4

5. □1□□□□□□□□□□□□□□□□□□□□2
□3
□4□□□□□□□□□□□□□□□□□□□□5
□6

6. □1□□□□□□□□□□□□□□□□□□□□2
□3□□□□□□□□□□□□□□□□□□□□4
□5□□□□□□□□□□□□□□□□□□□□6

7. □1□□□□□□□□□□□□□□□□□□□□2
□3

□4□□□□□□□□□□□□□□□□□□□□5
□6

8. Wie oben, aber 1 Stelle nach links verschoben.

9. Wie bei 8, was die Startpositionen für die Strings angeht.

8 Variable

Bei vielen Programmen müssen Sie mehrere Zahlen verwenden und sie auf eine bestimmte Art behandeln, aber die Zahlen selbst können sich verändern. Dann kommen die Variablen ins Spiel.

Viele Programme umfassen die regelmäßige Verarbeitung verschiedener Datenarten. Ein Programm, das die Stabilität von Hängebrücken analysiert, wird mit sehr vielen numerischen Daten umzugehen haben; ein Textautomat im Büro verarbeitet Textdaten, die aus Folgen von Buchstaben, Leerstellen und anderen Symbolen bestehen. In BASIC verwendet man für den Umgang mit solchen Größen *Variable*. Eine Variable wird mit irgendeinem Namen bezeichnet, den der Programmierer bestimmt, und dient als Platzhalter für die Daten; die eigentlichen Daten bestimmen die *Werte*, die die Variable annimmt. In Beispielen ist das alles, wie wir gleich sehen werden, viel einfacher als im Abstrakten. Hier ein typischer Fall:

```
10 FOR T = Ø TO 1ØØØ
20 PRINT T; "IM QUADRAT IST", T * T
30 NEXT T
```

Das hat das Quadrat T * T der Zahl T von Ø bis 1ØØØ zu berechnen. Wir sagen, T sei eine numerische Variable. Sie besitzt einen festen Namen, hier T, und einen Wert, der sich im Verlauf eines Programms ändern kann. Hier beginnt sie mit Wert Ø, nimmt dann Wert 1 und 2 und so weiter an, bis am Ende ihr Wert 1ØØØ ist.

Beim 64 gibt es zwei Hauptarten von Variablen:

1. *Numerische* Variable, die als ihre Werte Zahlen umfassen.
2. *String*variable, die Folgen von Zeichen umfassen.

Variablennamen müssen mit einem Buchstaben beginnen und aus Buchstaben und Ziffern aufgebaut sein. (Für diesen Zweck gelten Grafikzeichen als Buchstaben.) Typische numerische Variablennamen sind:

A	B	C	ALPHA	A5
GEWICHT	GROESSE	HARRY	MARIA	X233

und so weiter. Bei einer Stringvariablen muß am Ende das Dollarzeichen $ angehängt werden:

A$	B$	C$	ALPHA$	A5$
GEWICHT$	GROESSE$	HARRY$	MARIA$	X233$

Zu den Variablennamen muß man sich zwei Dinge merken:

1. Nur die beiden ersten Buchstaben oder Ziffern werden vom Computer erkannt (allerdings sind längere Namen bis zu rund 8Ø Zeichen erlaubt). Demnach werden

   EX EXPERTE EXIL EX55

alle mit der *gleichen* Bedeutung gelesen. Das kann nachteilig sein, wenn Sie Namen verwenden, die Sie an die Funktion der Variable erinnern sollen. Bei einem Firmenprogramm werden:

   BEITRAG

und

   BEZUEGE

vom Computer als gleich behandelt (BE), aber dem Programmierer entfällt das in der Hitze des Gefechts vielleicht, und er gerät in Verwirrung, wenn er das Ergebnis sieht . . .

2. Enthält ein Variablenname *irgendwo* ein BASIC-Schlüsselwort wie PRINT, LIST, TO, FOR und so weiter, so läßt das System ihn nicht zu. (Obwohl es sich nur für die beiden ersten Buchstaben interessiert! Der Grund: Es sucht zuerst nach den Schlüsselwörtern.) Die folgenden Namen können Sie also nicht verwenden, weil sie Schlüsselwörter (unterstrichen) enthalten:

   TOTAL FORDERUNG SORTEN$ BESTAND STABIL

und zu meinem Schrecken ist mein altes Lieblingswort

   FRED

auch nicht zulässig, weil es ein BASIC-Wort FRE gibt, das mitteilt, wieviel Speicherplatz noch vorhanden ist (siehe Kapitel 12).

In manchen Fällen kann das System mißverstehen, was Sie gemeint haben, statt einfach Einspruch zu erheben. Versuchen Sie zu erraten, was dieses Programm bewirkt.

```
1Ø  LETTER$ = "A"
2Ø  PRINT LETTER$
```

Sie erhalten eine Fehlermeldung. Verändern Sie Zeile 2Ø zu:

```
2Ø  PRINT TER$
```

und der Computer zeigt ein "A" an! Er hat Zeile 1Ø als einen Befehl aufgefaßt!

```
1Ø  LET TER$ = "A"
```

bezieht sich auf eine Variable namens TER$. Sehr spaßig. Aber Sie können Schwierigkeiten aus dem Weg gehen, solange Sie BASIC-Schlüsselwörter sorgfältig meiden.

Aufgabe 1

Zwei der folgenden Namen sind zugelassen, alle anderen nicht. Welche und warum? (Hinweis: Lassen Sie vom Computer mit PRINT alle anzeigen.)

```
#NEUN     SPENDE     RATLOS     FUNKTION     221B
BAKER     STREET     VERWENDUNG     SINN
GETRAGEN     POST     WAND     NOTE     VERDORBEN
BREMSE     FERNDIENST
```

Variable zuteilen

Der Programmierer muß dem Computer mitteilen, welchen Wert er einer Variablen zuteilen soll. (Wenn Sie das unterlassen, unterstellt er für jeden numerischen Wert Ø und für jede Stringvariable einen sogenannten *leeren String* ohne jedes Symbol.) Werte können Sie zuteilen mit dem Befehl:

```
LET
```

auf solche Weise:

```
1ØØ  LET GEWICHT = 77
11Ø  LET NAME$ = "CHARLIE"
```

Um Platz zu sparen, darf man das Wort LET weglassen (und tut das in der Regel auch):

```
1ØØ  GEWICHT = 77
11Ø  NAME$ = "CHARLIE"
```

Beachten Sie die Anführungsstriche um "CHARLIE". Sie brauchen Anführungsstriche stets dann, wenn Sie *String*variable zuteilen. Die Gänsefüßchen sind aber nicht Teil des Variablenwerts; sie geben nur an, wo die Folge von Symbolen anfängt und aufhört. Mehr über Strings in Kapitel 16.

Der Wert der Variablen kann auch indirekt dadurch zugeteilt werden, daß die neue Variable als eine Kombination schon definierter Variablen ausgedrückt wird:

```
1Ø  A = 36
2Ø  B = 5
3Ø  K = 1Ø * A + B
```

setzt K auf den Wert 1Ø * 36 + 5, also 365. Das nächste Programm multipliziert die beiden Zahlen 77 und 88:

```
1Ø  A = 77
2Ø  B = 88
3Ø  K = A * B
4Ø  PRINT K
```

Sie können, wenn Sie wollen, die Zeilen 1Ø und 2Ø verändern, um andere Zahlen zu multiplizieren. Natürlich gibt es kürzere Wege, um dasselbe zu erreichen, etwa:

```
1Ø  K = 77 * 88
2Ø  PRINT K
```

oder einfach:

```
1Ø  PRINT 77 * 88
```

aber das liegt daran, daß ich ein besonders einfaches Beispiel gewählt habe.

Aufgabe 2

In Commodore-Kaufhäusern kostet ein Helm mit Schmuckbesatz 135 Dollar, ein Zauberglas 32 Dollar und eine Packung Echsenschuppen 3 Dollar. Benützen Sie die drei Variablen HELM, GLAS und ECHS, um ein Programm aufzubauen, das die Kosten von sechs Helmen, fünf Gläsern und neunundzwanzig Packungen Echsenschuppen berechnet. Nennen Sie diese Variable PREIS.

Aufgabe 3

Ändern Sie nur eine einzige Programmzeile ab, um die Kosten von elf Helmen, vierzehn Gläsern und drei Packungen Echsenschuppen zu berechnen.

Aufgabe 4

Ändern Sie vier weitere Zeilen ab und lösen Sie Aufgabe 2, wenn die Preise sich geändert haben auf 147 Dollar für den Helm, 43 Dollar für ein Glas und 1 Dollar für eine Packung Echsenschuppen (die nicht mehr in Mode sind).

Lösungen

Aufgabe 1

Die zulässigen Namen sind BAKER und STREET. Die Namen #NEUN und 221B beginnen nicht mit Buchstaben. Die übrigen enthalten BASIC-Schlüsselwörter wie folgt:

SPENDE RATLOS FUNKTION VERWENDUNG

SINN GETRAGEN POST WAND NOTE

VERDORBEN BREMSE FERNDIENST

Aufgabe 2

```
1Ø  PRINT CHR$ (147
2Ø  HELM = 135
3Ø  GLAS = 32
4Ø  ECHS = 3
5Ø  PREIS = 6 * HELM + 5 * GLAS + 29 * ECHS
6Ø  PRINT "GESAMTPREIS BETRAEGT □$"; PREIS
```

(Ich habe einen Hinweis auf Leerstellen in der Meldung Zeile 6Ø bewußt unterlassen; von jetzt an mache ich das immer, wenn klar ist, wo sie hingehören.)

Aufgabe 3

Verändern Sie Zeile 5Ø zu:

```
5Ø  PREIS = 11 * HELM + 14 * GLAS + 3 * ECHS
```

Aufgabe 4

Verändern Sie die Zeilen 2Ø–4Ø zu:

```
2Ø  HELM = 146
3Ø  GLAS = 43
4Ø  ECHS = 1
```

9 Eingaben

Um Information in den Computer einzugeben, brauchen Sie ihm nur zu befehlen, er möge Sie erinnern, sobald er etwas wissen muß.

Die Aufgaben 2–4 im vorigen Kapitel haben ihre Arbeit in der Tat geleistet, aber es ist doch arg mühsam, jedesmal Programmzeilen verändern zu müssen, wenn Sie den Wert einer Variablen ändern wollen. Zum Glück ist das auch nicht notwendig, dank dem Befehl:

```
INPUT
```

mit dem Sie vom Keyboard aus den Wert setzen können. Und zwar so:

```
1Ø  PRINT "GIB EINE ZAHL EIN"
2Ø  INPUT N
3Ø  PRINT "DIE ZAHL WAR □"; N
```

Wenn Sie das fahren, erhalten Sie GIB EINE ZAHL EIN und dann ein Fragezeichen. Wenn Sie auf dem Keyboard jetzt eine Zahl tippen und RETURN drücken, geht das Programm weiter zu Zeile 3Ø und zeigt die Meldung samt der Zahl an.

Mehrfacheingaben

Wie bei PRINT-Befehlen können Sie mit INPUT mehrere verschiedene Variable eingeben, die Sie durch Kommata trennen. Probieren Sie

```
1Ø  INPUT A, B, C
2Ø  PRINT A, B, C
```

Sie erhalten ein ?-Zeichen. Drücken Sie eine Ziffer, meinetwegen 3. Nun erhalten Sie ?? für die zweite Zahl. Geben Sie 5 ein. Noch einmal erscheint ??, also drücken Sie 7. Der 64 zeigt prompt die Werte 3, 5, 7 an, die Sie für A, B, C eingegeben haben. Das Endergebnis sieht so aus:

```
RUN
?3
??5
??7
3          5          7
READY
```

Unrichtige Eingaben

Wenn Sie beim Input einen Fehler machen, können Sie mit der INST/DEL-Taste ausbessern, vorausgesetzt, Sie haben nicht auf RETURN gedrückt. Haben Sie das doch getan, gibt es Schwierigkeiten . . . außer in einem Fall, wo der 64 sich (und Sie) gegen einen häufig vorkommenden Fehler schützt.

Versuchen Sie für eine numerische Variable etwas einzugeben, das keine Zahl ist, erhalten Sie die Fehlermeldung

```
REDO FROM START
```

Sie sollen also die Eingabe wiederholen.

Aufforderung

Sie können eine Meldung anzeigen, die Sie daran erinnert, *wozu* eine Eingabe dient (ein ?-Zeichen ist nicht immer von Nutzen), nämlich so:

```
10 INPUT "GIB EINE ZAHL EIN"; N
```

was dieselbe Wirkung hat wie die Zeilen 10 und 20 des Programms, mit dem dieses Kapitel anfing. Sie können mit diesen *An-* oder *Aufforderungs*meldungen auch Mehrfacheingaben verbinden. Beachten Sie, wo Strichpunkt und Anführungsstriche stehen.

Hier ein Weg, die Aufgaben 2–4 des vorigen Kapitels ganz allgemein besser zu bewältigen.

```
10 PRINT CHR$ (147)
20 INPUT "PREISE: HELM, GLAS, ECHS?"; HE, GL, EC
30 INPUT "MENGEN?"; NH, NG, NE
40 PREIS = NH * HE + NG * GL + NE * EC
50 PRINT "GESAMTPREIS BETRAEGT $"; PREIS
```

Um das Listing abzukürzen, habe ich auf Variable mit zwei Zeichen abgekürzt. HE, GL und EC sind die alten HELM, GLAS und ECHS, und für die jeweiligen Mengen habe ich drei neue Variable namens NH, NG und NE.

Ein sauberes Display

Angenommen, Sie möchten, daß jemand, sagen wir, sein Alter und die Telefonnummer eingibt. Ihr erster Versuch könnte so aussehen:

```
10 PRINT CHR$ (147)
20 INPUT "ALTER"; A
30 PRINT A
```

```
40 INPUT "TELEFONNUMMER"; T
50 PRINT T
```

Das funktioniert zwar, aber am Ende sieht es auf dem Bildschirm so aus:

```
Alter? 17
17
TELEFONNUMMER? 361005
361005
READY
```

was nicht gerade ansehnlich ist. Es geht besser, wenn Sie die Reihenfolge ein bißchen verändern:

```
10 PRINT CHR$ (147)
20 INPUT "ALTER"; A
30 INPUT "TELEFONNUMMER?"; T
40 PRINT CHR$ (147)
50 PRINT A, T
```

Manchmal wollen Sie aber vielleicht nicht bis zum Ende warten, um alles anzuzeigen. Dann wäre es besser, dafür zu sorgen, daß die Meldungen oben auf dem Bildschirm angezeigt und gelöscht werden, bevor es weitergeht.

Hier ist der Ort für Steuerzeichen. Sie entsinnen sich, daß PRINT CHR$ (19) den Cursor zurückführt, den Bildschirm aber nicht antastet. Das können wir nutzen, um dafür zu sorgen, daß die Aufforderungszeichen in den obersten Zeilen angezeigt werden. Hier ein Beispiel mit Mehrfacheingaben.

```
10 PRINT CHR$ (147)
20 REIHE = 4
30 PRINT CHR$ (19)
40 PRINT "[30 x □]"
50 PRINT "[30 x □]"
60 INPUT "ALTER"; A
70 INPUT "TELEFONNUMMER"; T
80 PRINT CHR$ (19)
90 FOR R = 1 TO REIHE: PRINT CHR$ (17);: NEXT R
100 PRINT A, T
110 REIHE = REIHE + 1
120 GOTO 30
```

Hier löscht Zeile 1Ø den Schirm, Zeile 2Ø setzt eine Variable für die Reihennummer, wo der nächste Eintrag in der Liste angezeigt werden soll. Zeilen 3Ø–5Ø löschen die obersten beiden Reihen für neue Eingaben; 6Ø–7Ø nehmen die neuen Eingaben an. Zeilen 8Ø–9Ø führen zur Anzeige in der Reihe REIHE. Zeile 1ØØ schreibt die Einträge der Tabelle, 11Ø führt REIHE um 1 weiter zur nächsten Reihe, und 12Ø wiederholt den ganzen Vorgang.

Ein bißchen Jongleurkunst, aber nicht unmöglich, wenn man klaren Kopf behält . . .

Übrigens bedeutet Zeile 4Ø, daß Sie

PRINT "[3Ø x □]"

schreiben müssen, ebenso bei Zeile 5Ø.

Aufgabe 1

Schreiben Sie ein Programm, das als Eingaben bis zu 15 Namen und Rufnummern annimmt und sie nach der Eingabe anzeigt, wobei das Display ordentlich aussieht.

Lösungen

Aufgabe 1

```
1Ø  PRINT CHR$ (147)
2Ø  REIHE = 4
3Ø  FOR K = 1 TO 15
4Ø  PRINT CHR$ (19)
5Ø  PRINT "[3Ø x □]"
6Ø  PRINT "[3Ø x □]"
7Ø  INPUT "NAME"; NAME$
8Ø  INPUT "TELEFONNUMMER"; T
9Ø  PRINT CHR$ (19)
1ØØ FOR R = 1 TO REIHE: PRINT CHR$ (17);: NEXT R
11Ø PRINT NAME$, T
12Ø REIHE = REIHE + 1
13Ø NEXT K
```

Übrigens könnten Sie Zeilen 2Ø und 12Ø weglassen und 1ØØ ersetzen durch

```
1ØØ  FOR R = 1 TO K + 3: PRINT CHR$ (17);: NEXT R
```

weil Sie in Wahrheit keine *zwei* Zähler REIHE und K brauchen.

10 Debugging I

Es heißt, der bildkräftige Ausdruck "die Bugs (Käfer, Insekten) herauszuholen" sei in der Frühzeit der Computer entstanden, als Insekten in die Maschine krabbelten und Kurzschlüsse hervorriefen. Wenn heutzutage der Computer versagt, ist in der Regel der Programmierer schuld. Aber um das beheben zu können, müssen Sie sich mit dem Debugging (= Entfehlern) trotzdem auskennen.

Sie sollten sich nie entmutigen lassen, wenn ein Programm beim ersten Lauf nicht gleich fehlerlos läuft. Das kommt sogar bei Berufsprogrammierern ganz oft vor!

Sie müssen also über Methoden verfügen, die Ihnen helfen, die Fehler in einem Programm rasch und ohne Mühe zu finden. Das nennen wir *Debugging*.

Syntaxfehler

Fangen wir an mit den Fehlern, die erstmals dann auftreten, wenn Sie ein Programm fahren. Beispiel: Nehmen wir an, Sie schreiben irgendwo in einem Ihrer Programme

```
5Ø  FOR 1 = 1 - 7
```

weil Sie vergessen haben, daß man in einer FOR-Schleife die Werte durch ein "TO" trennen muß. Der 64 nimmt das nun geduldig an, wenn Sie es eingeben, aber beim Lauf des Programms erhalten Sie die Meldung:

```
SYNTAX ERROR IN 5Ø
```

Mit anderen Worten: Dem 64 behagt die grammatikalische Konstrukion der Anweisung in Zeile 5Ø nicht. Das ist ganz so, als würde ich sagen: "64, er nicht verstehen diese Anweisung!" Sie würden Einwände gegen meine Syntax erheben! Der einzige Unterschied ist der, daß *Sie* meine ungrammatikalische Mitteilung begreifen würden, der 64 sich aber gar nicht die Mühe macht, Zeile 5Ø zu verstehen. Er wirft sie, wie wir gesehen haben, einfach hinaus.

Fehler dieser Art unterlaufen besonders häufig, wenn Sie eine Programmiersprache erlernen, und es ist sehr ärgerlich, ein Programm von sechzig Zeilen mit zehn FOR-Anweisungen einzugeben, um dann beim Lauf zu erfahren, daß Sie den Aufbau einer FOR-Anweisung falsch in Erinnerung haben und sie nun alle ändern müssen. Sie sollten es sich also zur Gewohnheit machen, nach jeweils zwei, drei eingetippten Zeilen RUN zu geben. Unvollständige Programme verhalten sich freilich nicht sinnvoll, und Sie können sogar Fehlermeldungen erhalten, die wieder verschwinden, wenn der nächste Teil eingegeben wird, aber in diesem Stadium kommt es für Sie nur darauf an, die Syntaxfehler zu entdecken, bevor Ihnen zu viele davon unterlaufen.

Die fehlerhafte Zeile ändern

Sobald Sie den Fehler entdeckt haben, wollen Sie die Anweisung ändern. Am einfachsten geht das, wenn Sie die Zeile neu eingeben. (Wohlgemerkt: Wenn Sie eine Zeile ganz entfernen wollen, schreiben Sie einfach die Zeilennummer, gefolgt von RETURN.) Der Haken dabei ist natürlich der, daß die Zeile recht lang sein kann und es als Zeitvergeudung erscheint, die ganze Tipperei mit vielleicht nur einer einzigen Änderung wiederholen zu müssen.

Der 64 bedient Sie mit einem *Editor*, um dieses Problem zu bewältigen. Er ist sehr leistungstark und erfordert wie vieles in dieser Art ein wenig Übung, bevor Sie richtig damit umgehen können.

Sorgen Sie als erstes dafür, daß die Zeile, die Sie ändern wollen, irgendwo auf dem Bildschirm erscheint, an welcher Stelle, ist ganz gleichgültig. Wenn das nicht der Fall ist, schreiben Sie (in unserem Beispiel) LIST 5Ø.

Benützen Sie nun die beiden Cursortasten (am Keyboard unten rechts) dazu, den Cursor auf die "5" zu bewegen, also:

```
5Ø  FOR N = 1 - 7
```

Dazu halten Sie die SHIFT-Taste gedrückt und drücken gleichzeitig die *innere* Cursortaste. Wenn Sie die Taste nur für ein, zwei Sekunden drücken, sehen Sie den Cursor eine Zeile höherklettern. Drücken Sie die Taste längere Zeit nieder, dann wiederholt sich der Vorgang auf einmal sehr rasch, und Sie werden feststellen, daß Sie, bis Sie Übung bekommen haben, über das Ziel hinausschießen. Macht nichts; um am Bildschirm wieder herunterzukommen, drücken Sie einfach die Taste, diesmal aber ohne SHIFT.

Betätigen Sie nun die andere Cursortaste (ohne SHIFT), um den Cursor die Zeile entlangzubewegen, bis er auf dem − ist:

```
5Ø  FOR N = 1 - 7
```

Tippen Sie "T". Es tritt an die Stelle des −:

```
5Ø  FOR N = 1 T 7
```

Wenn Sie nun "O" tippen, ersetzt es die 7, was in diesem Fall keine große Rolle spielen würde, weil Sie die Ziffer einfach noch einmal tippen könnten, aber wie sähe es aus, wenn danach noch 2Ø weitere Zeichen kämen? Der 64 verschafft uns einen Weg, das Problem heimlich zu umgehen: Wir können eine Zeile dadurch *erweitern*, daß Stellen eingeschoben werden, die man dann überschreiben kann. An der Tastatur oben rechts befindet sich eine *Insert*-(Einschub)-Taste INST. Beachten Sie, daß das die obere Tastenhälfte betrifft, so daß Sie dazu SHIFT betätigen müssen, weil sonst mit DEL gelöscht wird.

Steht der Cursor so, wie oben angegeben, und Sie drücken INST, entsteht zwischen "T" und "7" eine freie Stelle:

```
5Ø  FOR N = 1 T □ 7
```

wo Sie das "O" jetzt hineinsetzen können:

```
5Ø  FOR N = 1 TO 7
```

Drücken Sie dann noch RETURN, und die Sache hat sich.

Das Ganze zu beschreiben, hat natürlich viel länger gedauert, als es zu tun, vor allem dann, wenn man ein bißchen Übung hat.

Aufgabe 1

Hier ein paar Beispiele für Sie zum Üben. In jedem findet sich ein Syntaxfehler. Versuchen Sie alle Anweisungen einzugeben, herauszufinden, worin der Fehler besteht, und mit dem Editor den Fehler dann zu entfernen. Schreiben Sie schließlich RUN, um sich zu vergewissern, daß keine ?SYNTAX ERROR-Meldung mehr kommt. (Sie erhalten vielleicht eine andere Meldung, aber hier befassen wir uns zunächst nur mit Syntaxfehlern.)

1. 2Ø INPUT A
2. 3Ø B * C = F
3. 4Ø INPUT "GIB EINEN WERT EIN", V
4. 5Ø IF P = 1: PRINT "P = 1"
5. 6Ø A = 3; B = 7; C = 1Ø
6. 7Ø TOP = 4

Lösungen

Aufgabe 1

1. Das müßte lauten:

   2Ø INPUT A

   Führen Sie, um es abzuändern, den Cursor auf "M", schreiben Sie "N" und drücken Sie RETURN. Einfach!

2. Das ist verkehrt herum. Es muß heißen:

   3Ø F = B * C

   In einem solchen Fall geht es vermutlich schneller, die Zeile neu zu schreiben, statt den Editor zu verwenden. Zur Übung trotzdem:
   Führen Sie den Cursor auf "B" und fügen Sie zwei Leerstellen ein. Tippen Sie "F=" hinein. Führen Sie anschließend den Cursor am "F" des Zeilenendes vorbei, entfernen Sie "F" und "=" mit DEL. Drücken Sie, wie gewohnt, RETURN.

3. Das Trennzeichen Komma muß ein Strichpunkt sein:

   4Ø INPUT "GIB EINEN WERT EIN"; V

   Also wird schlicht getauscht, wie in Beispiel 1.

4. Das muß lauten:

   5Ø IF P = 1 THEN PRINT "p = 1"

   (siehe Kapitel 11), so daß Sie das "=" durch "T" ersetzen, drei Leerstellen einfügen und "HEN" hineinschreiben müssen.

5. Die Strichpunkte müssen Doppelpunkte sein, also schlicht austauschen.

6. Da werden Sie vermutlich überfordert sein. Auf Anhieb ist nichts dagegen einzuwenden, "TOP" als Variablennamen zu verwenden, aber leider enthält er am Anfang das BASIC-Schlüsselwort "TO" (wie bei FOR N = 1 TO 2Ø) – siehe Kapitel 8. Der 64 gerät also in Verwirrung, weil er Sinn in einer Anweisung sucht, die mit "TO" beginnt und er als erstes ein "FOR" erwartet.
   Sie müssen den Variablennamen ändern, vielleicht zu "MAX".
   Das Problem kommt ziemlich oft vor, und es ist gut, darauf zu achten, weil es viel Verwirrung stiften kann. Sobald Sie ohne erkennbaren Grund einen Syntaxfehler zu sehen bekommen, versuchen Sie die Variablennamen zu ändern!

Hinweis: Wenn Sie ausgebessert haben, wird der Cursor vermutlich *nicht* in einer leeren Zeile stehen. Wenn Sie RUN schreiben, ersetzen Sie dann die ersten drei Zeichen in der Zeile, so daß, wenn die Zeile, zum Beispiel, lautete:

READY

dort nun

RUNDY

steht. Wenn Sie RETURN drücken, versucht BASIC zu verstehen, was RUNDY sein soll, was nicht geht, so daß Sie wieder eine Fehlermeldung einfangen. Am besten löscht man nach abgeschlossener Ausbesserung den Bildschirm.

11 Verzweigen

Manchmal muß der Computer verschiedene Aufgaben unter verschiedenen Bedingungen ausführen. Dazu nutzt man Verzweigungen.

Wir haben zwei Befehle mit Auswirkungen auf die Reihenfolge kennengelernt, in welcher der Computer Befehle ausführt, nämlich GOTO, der ihn einfach zu einer bestimmten Zeile, und FOR . . . NEXT, der ihn durch eine regelmäßige Schleife schickt. Die beiden sind für Programme, die unter verschiedenen Bedingungen verschieden reagieren müssen, ein bißchen *zu* regelmäßig.

Daß das Programm sich entsprechend den Bedingungen verzweigt, läßt sich unter anderem mit dem Befehl

IF . . . THEN . . .

erreichen.

Hier ein Beispiel:

Bankkonto

Das ist ein ganz einfaches Beispiel für ein "Praxis"-Programm. Es berechnet Ihren Kontostand, wenn Sie den vorherigen Stand angeben und Ihre Einkünfte und Ausgaben Posten für Posten aufführen. Damit es einfach bleibt, habe ich mir keine Gedanken über die Position der Eingabemeldungen gemacht; Sie können, wenn Sie Lust haben, hier Ordnung schaffen.

```
10  PRINT CHR$ (147)
20  PRINT "BISHERIGER KONTOSTAND";
30  INPUT B
40  PRINT "FUEHRE AUSGABEN AUF"
50  INPUT A
60  IF D < 0 THEN 90
70  LET B = B - A
80  GOTO 50
90  PRINT "FUEHRE EINNAHMEN AUF"
100 INPUT E
110 IF E < 0 THEN 140
120 LET B = B + E
```

```
13Ø  GOTO 1Ø
14Ø  PRINT "DERZEITIGER KONTOSTAND □"; B
```

Die entscheidenden Zeilen hier sind 6Ø und 11Ø. Um zu sehen, wie sie wirken, müssen Sie wissen, daß das Zeichen < *weniger als* bedeutet. Die *Bedingung* D < Ø bedeutet also: "D ist kleiner als Null". Das ist keine Zahl, hat also keinen numerischen Wert; es ist eine logische Feststellung, die entweder *wahr* oder *falsch* sein kann (je nachdem, ob D wirklich kleiner ist als Null oder nicht). Beispiel:

3 < Ø ist *falsch*

−7 < Ø ist *wahr*

Eine typische IF . . .-THEN-Anweisung hat die Form

IF Bedingung THEN Befehl

Ist die Bedingung wahr, wird der Befehl ausgeführt. Ist sie aber falsch, *geht das Programm zur nächsten Zeile*. Zeile 6Ø schickt das Programm also dann zu Zeile 9Ø, wenn D negativ, aber zur nächsten, Zeile 7Ø, wenn D positiv ist.

Sehen wir uns an, wie das geht. Bis Zeile 5Ø sollte alles klar sein. Nehmen wir an, Sie haben drei Ausgaben (Barzahlungen) von 15, 7 und 11 Pfund (oder Dollar, wenn Ihnen das lieber ist – beim heutigen Wechselkurs wird der Unterschied bald nicht mehr zu sehen sein). Wenn Sie das erstemal aufgefordert werden, D einzugeben, teilen Sie dem Computer den ersten Betrag mit.

```
? 15
```

Da D < Ø falsch ist, geht das Programm weiter zu den Zeilen 7Ø und 8Ø, die es zu einer neuen Eingabe in Zeile 5Ø zurückschicken. Sie liefern also den zweiten Betrag:

```
? 7
```

und dann den dritten:

```
? 5
```

aber der Computer giert immer noch nach einer Eingabe:

```
?
```

also müssen Sie etwas tun. Nun kommt D < Ø zu seinem Recht. Geben Sie einen negativen Wert ein (−1 bietet sich an). Jetzt geht das Programm zu Zeile 9Ø, weil D < Ø *wahr* ist. Der ganze Ablauf wiederholt sich, jetzt für Einkünfte (eingehende Beträge); und die Eingabe hört auf, wenn Sie wieder −1 beisteuern. Der Gebrauch eines sinnlosen Werts wie −1, um eine Verzweigung zu steuern, ist ein üblicher Kniff; wir sagen, −1 diene als *Begrenzer*, weil er dem Computer "Ende der Eingabeliste" signalisiert.

Inzwischen haben die Zeilen 7Ø und 12Ø Abhebungen abgezogen und Eingänge hinzuaddiert, so daß er endgültige Kontostand in Erscheinung tritt.

Aufgabe 1

Schreiben Sie ein Programm zur Addierung der Preise einer Einkaufsliste und verwenden Sie einen Begrenzer – 1, um das Ende der Eingaben anzuzeigen.

Aufgabe 2

Schreiben Sie ein Programm zur Eingabe einer Zahl, das anzeigen soll, ob sie kleiner als 1ØØ, gleich 1ØØ oder größer als 1ØØ ist. (Hinweis: > bedeutet "großer als".)

Größenrelationen von Zahlen

Es gibt eine ganze Reihe von Symbolen für die Feststellung, ob eine Zahl größer ist als eine andere oder gleich oder anders oder was auch immer. Hier eine vollständige Liste mit Beispielen.

= gleich	(3 = 3 wahr; 3 = 4 falsch)
> größer als	(3 > 2 wahr; 3 > 4 falsch)
< kleiner als	(3 < 4 wahr; 3 < 2 falsch)
< > nicht gleich	(3 < > 4 wahr; 3 < >3 falsch)
> = größer als oder gleich	(3 > = 3, 3 > = 2 wahr; 3 > = 4 falsch)
< = kleiner als oder gleich	(3 < = 3, 3 < = 4 wahr; 3 < = 2 falsch)

Logik

Die Befehlswörter

AND

OR

können dazu verwendet werden, Bedingungen zu verknüpfen. Wenn c und d Bedingungen sind, dann ist c AND d wahr, falls c und d *beide* wahr sind. Demnach ist

X > Ø AND X < 3

nur wahr, wenn X größer ist als Ø und auch kleiner als 3. Bei ganzen Zahlen X bedeutet das, X muß 1 oder 2 sein; bei Dezimalzahlen (über die ich noch nicht gesprochen habe), läßt das erheblich mehr Möglichkeiten zu.

Ebenso ist c OR d wahr, falls entweder c oder d wahr ist (oder möglicherweise beide). Also bedeutet

X < 5 OR X = 5

dasselbe wie X <= 5.

Schließlich sollte ich noch den logischen Befehl

NOT

erwähnen, der wahr in falsch verwandelt und umgekehrt.

Bedingte Sprünge

Ein gängiger Gebrauch von IF . . . THEN ist der, das Programm zu einer neuen Zeile umzuleiten, nämlich so:

```
83Ø  IF X = Y THEN 99Ø
84Ø  beliebig . .
```

Das nennt man einen bedingten Sprung.

Nach THEN folgt ein (stillschweigendes) GOTO. Eigentlich sollte es heißen IF X = Y THEN GOTO 99Ø. Der Kürze halber wird das GOTO aber weggelassen.

Bedingte Zuweisungen

Eine andere Möglichkeit ist die, einer Variablen verschiedene Werte je nach Wahrheit oder Falschheit einer Bedingung zu geben:

```
89Ø  IF X = Y THEN K = 77
9ØØ  IF X <> Y THEN LET K = 99
```

wonach K 77 ist, wenn X = Y, und 99, wenn nicht. Das LET kann, wie üblich, weggelassen werden.

Aufgabe 3

Die Verkaufssteuer für eine Ware hängt wie folgt von der steuerlichen Einstufung ab:

Tarif 1 (Erziehung)	2%
Tarif 2 (Kinderprodukte)	5%
Tarif 3 (Staatsverwendung)	Ø%
Tarif 4 (alles übrige)	15%

Schreiben Sie ein Programm, das den Steuertarif als Eingabe annimmt und den Prozentsatz der Steuer anzeigt.

Lösungen

Aufgabe 1

```
1Ø  PRINT CHR$ (147)
2Ø  PRINT "EINKAUFSLISTE"
3Ø  INPUT X
4Ø  IF X < Ø THEN 7Ø
5Ø  SUMME = SUMME + X
6Ø  GOTO 3Ø
7Ø  PRINT "GESAMTBETRAG"; SUMME
```

Aufgabe 2

```
1Ø  PRINT CHR$ (147)
2Ø  INPUT N
3Ø  IF N < 1ØØ THEN PRINT "KLEINER ALS 1ØØ"
4Ø  IF N = 1ØØ THEN PRINT "GLEICH 1ØØ"
5Ø  IF N > 1ØØ THEN PRINT "GROESSER ALS 1ØØ"
```

Aufgabe 3

```
1Ø  PRINT CHR$ (147)
2Ø  INPUT "STEUERTARIF"; S
3Ø  IF S < 1 OR S > 4 THEN 2Ø
4Ø  IF S = 1 THEN PRINT "2% ERZIEHUNGSTARIF"
5Ø  IF S = 2 THEN PRINT "5% KINDERTARIF"
6Ø  IF S = 3 THEN PRINT "Ø% STAATSTARIF"
7Ø  IF S = 4 THEN PRINT "15% – PECH GEHABT!"
```

Nur zum Spaß habe ich bei Zeile 3Ø eine *Narrensicherung* eingebaut, das heißt, gegen unsinnige Eingaben geschützt.

12 Binäre Zahlen

Computer verarbeiten Zahlen nicht nach unserer gewohnten Dezimalschreibweise, sondern auf eine Art, die für elektronische Schaltungen besser geeignet ist. Es ist ein bißchen so, als zähle man mit den Füßen, statt mit den Fingern.

Man braucht kein mathematischer Experte zu sein, um beim 64 Binärzahlen richtig zu nutzen, aber ein paar Grundgedanken sind unentbehrlich. Beispielsweise kann man ohne Binärzahlen keine Spritegrafik erzeugen. Wir müssen also ein bißchen früher, als uns eigentlich lieb ist, in den Computer hineinsteigen und uns damit befassen, was er in *Wahrheit* macht. Dann mal ran.

Normalerweise befassen wir uns mit Zahlen auf Zehnergrundlage. Wenn ich die Zahl 3814 schreibe, verstehen wir darunter alle die Zusammensetzung

$$3 \times 1000 + 8 \times 100 + 1 \times 10 + 4 \times 1$$

und wir können erkennen, daß wir, um einen "Stellenwert" von rechts weiterzukommen, einfach mit Zehn multiplizieren. Wir sagen, die Zahl beruhe auf der Basis *Zehn.*

Weil wir das schon so lange tun, wie wir denken können, fällt die Erkenntnis schwer, daß es noch andere, völlig vernünftige Wege gibt, dieselbe Aufgabe zu bewältigen. Die ersten Computerkonstrukteure kamen jedenfalls nicht darauf; sie verwendeten bei ihren Maschinen Zehnerdarstellungen und gerieten in böse Schwierigkeiten. Meistens rührten sie daher, daß elektronische Verstärker sich nicht bei allen Signalen, die man eingeben möchte, gleich verhalten. Beispiel: Ein Verstärker, der sein Eingabesignal verdoppeln soll, mag das bei Eingaben von 1, 2, 3 und 4 Einheiten durchaus tun, aber dann beginnt er "abzuflachen", so daß eine Eingabe von 5 zu einer Ausgabe von nur 9.6, 6 zu 10.8 führt und man den Unterschied in der Ausgabe bei Eingaben von 8 und 9 kaum noch unterscheiden kann.

Legen Sie eine Musikkassette in Ihren preiswert erworbenen Recorder und drehen Sie die Lautstärke ganz auf. Hören Sie die Verzerrung bei den lauten Stellen? Das ist dieselbe Wirkung.

Die Computerkonstrukteure der Anfangszeit hörten keine Verzerrung; sie stellten nur fest, daß die Maschinen manchmal nicht zwischen Ziffern unterscheiden konnten, bei einem Computer eine aussichtslose Sache. Sie mußten also ihre Zahlendarstellung neu überdenken, um sie dem anzupassen, was die Elektronikgeister am besten konnten.

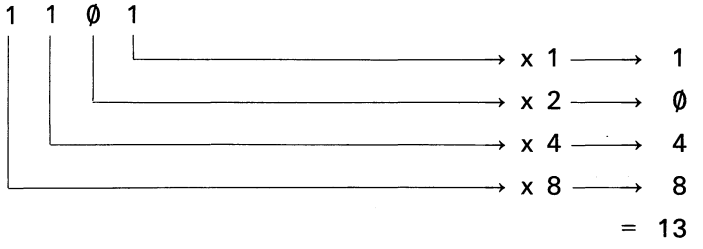
Das Einfachste, was man mit einem elektrischen Signal machen kann, ist, es an- oder abzuschalten; die Ziffern 0 (ein) und 1 (aus) lassen sich also befriedigend darstellen. Verzerrung spielt keine Rolle mehr. Alles bleibt klar, ob ein Signal vorhanden ist oder nicht, ohne Rücksicht auf irgendeine Verzerrung. Aber können wir denn ein Zahlensystem aufbauen, das sich nur auf 0 und 1 stützt?

Allerdings. Bei einer Zahl auf Zehnerbasis ist die größtmögliche Ziffer 9. Tut man 1 zu 9 hinzu, erhält man 10 – stattgefunden hat ein *Übertrag.* Wir können jede Zahl mit jeder anderen Basis schreiben, die uns behagt, und die

größtmögliche Ziffer wird stets um 1 niedriger sein als die Basis. Bei Basis 2 ist die größte Ziffer 1, also enthält eine Zahl auf Basis 2 (oder *binäre* Zahl) nur jeweils Ø und 1.

Und die Stellenwerte? Bei Basis Zehn erhielten wir sie, wenn wir (rechts) bei 1 anfingen und jedesmal, wenn wir eine Stelle nach links gingen, mit 1Ø multiplizierten. Bei einer Binärzahl beginnen wir auch mit 1, multiplizieren aber jedesmal, wenn wir nach links gehen, mit 2.

So kann beispielsweise die Binärzahl 11Ø1 auf folgende Weise in Basis Zehn umgewandelt werden.



Umgekehrt läßt sich ebenso leicht umwandeln; nehmen wir als Beispiel 25. Wenn wir die binären Stellenwerte

32 16 8 4 2 1

niederschreiben und links beginnen, ist klar, daß wir eine 16 brauchen, so daß 9 übrigbleibt, bestehend aus 8 und 1. 25 ist also:

Ø 1 1 Ø Ø 1

Jede binäre Ziffer (Ø oder 1) wird ein *Bit* (von binary digit) genannt.

Bytes

Ein *Byte* ist eine Binärzahl von 8 Ziffern, beispielsweise 11ØØØ1Ø1 (dezimal also 197). Die Dezimalwerte von Bytes gehen von Ø bis 255. Beim 64 können Speicherplätze genau ein Byte aufnehmen, daher ihre Bedeutung. (Das liegt daran, daß der 64 ein sogenannter *8 Bit-Mikroprozessor* ist; kompliziertere Geräte haben größere Speicherplätze, aber das kostet!) Anhang 1 liefert die Dezimal- und Binärwerte aller 256 möglichen Bytes. Mit dem nächsten Programm können Sie ein Byte eingeben und genau verfolgen, wie es in Dezimal umgewandelt wird.

```
1Ø DIM B (8)
2Ø SUMME = Ø
3Ø PRINT CHR$ (147)
4Ø INPUT "GIB EINE BINAERZAHL EIN"; B$
5Ø IF LEN (B$) <> 8 THEN 4Ø
```

```
 60 PRINT CHR$ (147)
 70 PRINT "UM DIE ZAHL □"; B$; "□ IN DEZIMAL"
 80 PRINT "UMZUWANDELN, WIRD SO VERFAHREN:"
 90 PRINT
100 FOR T = 1 TO 8
110 S = 2 ↑ (8 - T)
120 B (T) = VAL (MID$ (B$, T, 1))
130 PRINT B (T); TAB (3); "X"; S; TAB (10); "=";
140 IF B (T) = 0 THEN PRINT "□ □ □ □ □ 0"
150 IF B (T) = 1 AND T > 1 THEN PRINT "□";
160 IF B (T) = 1 AND T > 4 THEN PRINT "□";
170 IF B (T) = 1 THEN PRINT "□ □"; S
180 SUMME = SUMME + B (T) * S
190 NEXT T
200 PRINT "[17 x Grafik-D + SHIFT]"
210 PRINT "GESAMT □ □ □ □ = □ □";
220 IF SUMME < 100 THEN PRINT "□";
230 IF SUMME < 10 THEN PRINT "□";
240 PRINT SUMME
250 PRINT "[17 x Grafik-D + SHIFT]"
260 PRINT
270 PRINT
```

Bits in Bytes

Wir möchten uns manchmal auf, sagen wir, das fünfte Bit des Bytes 11010101 beziehen. Meinen wir damit aber das fünfte von links oder das fünfte von rechts? Darauf kommt es an; das eine ist 0, das andere 1. Um Verwirrung zu vermeiden, übernehmen wir eine Standardnumerierung von 0 bis 7, die von rechts nach links verläuft. (Sinnvoll ist das aus mathematischen Gründen, siehe unten.) Also so:

Bitnummer ⟶	7	6	5	4	3	2	1	0
Wert ⟶	1	1	0	1	0	1	0	1

Nun können wir von "Bit 4 des Bytes 11Ø1Ø1Ø1" sprechen, und es gibt keine Mißverständnisse.

Der Grund für diese Numerierung ist, daß die bedeutsamsten Bits, das heißt, jene, die zum Dezimalwert die größte Menge beitragen, die höchsten Nummern haben. Eine "1" in Bit 7 trägt zum Wert 128 bei, eine "1" in Bit 6 liefert 64, und so weiter bis hinunter zu Bit Ø, das dürftige 1 beisteuert. Bit K liefert also 2^K, das ist die Kte Potenz von 2. (In BASIC wird das 2↑K geschrieben.)

Hohe und niedrige Bytes

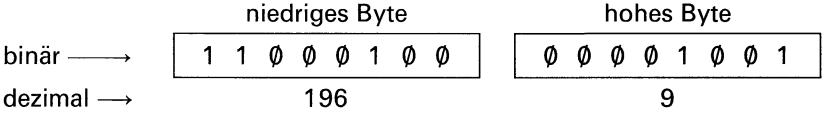
Ein Byte kann Zahlen nur bis zu 255 aufnehmen, aber viele wichtige Zahlen (beispielsweise Zeilennummern und Maschinenadressen) können größer sein. Wenn wir zwei Bytes kombinieren, können wir bis 65535 kommen. Angenommen, zwei aufeinanderfolgende Bytes enthalten die Zahlen N und H, dann wird der Wert behandelt als

N + 256 * H

Wir nennen N das *niedrige* Byte und H das *hohe* Byte. So würde die Zahl

25ØØ = 196 + 9 * 256

gespeichert werden als

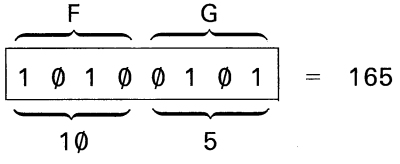


Im Grunde wird die Zahl damit als eine zweiziffrige Zahl mit Basis 256 behandelt! Diese Art von Zahlendarstellung wird etwa im SID-Klangchip verwendet (Kapitel 3Ø). Bei einer allgemeinen Zahl X zwischen Ø und 65535 sind die beiden Bytes:

hohes Byte H = INT (X/256)

niedriges Byte N = X − 256 * H = X − 256 * INT (X/256)

(INT bedeutet "Ganzzahliger- oder *Integerteil von*".) Ebenso kann ein Einzelbyte dazu verwendet werden, zwei 4 Bit-Zahlen zwischen F und G (zwischen Ø und 15) aufzunehmen, indem 16 * F + G gespeichert wird:



Ist der Bytewert X, dann haben wir:

F = INT (X/16)

G = X − 16 * F = X − 16 * INT (X/16)

Die Umhüllende Kurve (ASDR) im SID verwendet diese Methode, statt überall halbe Bytes zu vergeuden. (Ein halbes Byte wird im "Reference Guide" manchmal ein *Nybble* genannt.)

Speicherbelegung

Die Größe von Computerspeichern wird gewöhnlich in Kilobytes gemessen. Ein Kilobyte umfaßt 1Ø24 Bytes (was also nah genug bei 1ØØØ liegt). Ein "nackter" 64 hat 65536 Bytes oder 64 Kilobytes (64K) Speicher, aber rund 26 K sind reserviert vom System, so daß Ihnen zum Spielen 38K bleiben. Um genau zu sein, 38911 Bytes.

Während Sie ein Programm schreiben, wird Speicherplatz verbraucht. Auch Variablenwerte erfordern Platz. Es kann oft nützlich sein, zu wissen, wieviel Speicherplatz noch frei ist.

Um das festzustellen, verwenden Sie den Befehl

```
PRINT FRE (Ø)
```

Ist das Ergebnis positiv, wird die Zahl der freien Bytes genannt. Ist es negativ, versuchen Sie es noch einmal mit:

```
PRINT 65536 + FRE (Ø)
```

und erhalten jetzt den richtigen Wert. Es ist eine Eigenheit des FRE-Befehls, daß das vorkommt, noch dazu eine recht alberne, aber das läßt sich mühelos beheben. Wenn FRE einen negativen Wert meldet, haben Sie auf jeden Fall mindestens noch 32K freien Speicherplatz. In einem Programm könnten Sie verwenden:

```
1ØØØ  F = FRE (Ø): IF F < Ø THEN F = F + 65536
1Ø1Ø  PRINT F; " BYTES FREI"
```

Byte-Logik

Man kann mit Binärzahlen Arithmetik betreiben, entweder direkt oder durch Umwandlung in Dezimalzahlen und zurück. Man kann aber auch *logische* Operationen AND und OR ausführen. Vielleicht möchten Sie raten, wie die Lösung von

```
185 OR 148
```

lautet? Wären Sie auf 189 gekommen? Warten Sie auf die Erklärung.

Es gibt einen Zusammenhang zwischen Byte-Logik und gewöhnlicher Logik, gestützt auf eine Auffassung, wobei eine Ziffer einen *Wahrheitswert* erhält:

Ø bedeutet "falsch"

1 bedeutet "wahr"

Dann wirken bei einziffrigen Zahlen die logischen Operatoren OR und AND so:

Ø AND Ø = Ø	Ø OR Ø = Ø
Ø AND 1 = Ø	Ø OR 1 = 1
1 AND Ø = Ø	1 OR Ø = 1
1 AND 1 = 1	1 OR 1 = 1

Das heißt: p AND q ist wahr (Wert 1) nur dann, wenn p und q beide wahr sind; p OR q ist wahr, vorausgesetzt, p ist wahr oder q ist wahr oder beide sind es. Das stimmt mit den üblichen Regeln der Logik überein.

Dieser Gedanke gilt auch für Folgen von Binärziffern. Wir führen die Operation an jeder Stelle der Folge der Reihe nach aus. Um etwa

1Ø111ØØ1 OR 1ØØ1Ø1ØØ

zu berechnen, nehmen wir die Ziffern der Reihe nach so:

		erste Zahl		zweite Zahl		Lösung
Bit 7	→	1	OR	1	=	1
Bit 6	→	Ø	OR	Ø	=	Ø
Bit 5	→	1	OR	Ø	=	1
Bit 4	→	1	OR	1	=	1
Bit 3	→	1	OR	Ø	=	1
Bit 2	→	Ø	OR	1	=	1
Bit 1	→	Ø	OR	Ø	=	Ø
Bit Ø	→	1	OR	Ø	=	1

Dann lesen wir die Antwort ab:

1Ø111ØØ1 OR 11ØØ1Ø1ØØ = 1Ø1111Ø1

In der Dezimalschreibweise wird das, wie ich vorhin erklärt habe, zu

185 OR 148 = 189

Ebenso ergibt 185 AND 148:

Bit 7	→	1	AND	1	=	1
Bit 6	→	Ø	AND	Ø	=	Ø
Bit 5	→	1	AND	Ø	=	Ø
Bit 4	→	1	AND	1	=	1
Bit 3	→	1	AND	Ø	=	Ø
Bit 2	→	Ø	AND	1	=	Ø
Bit 1	→	Ø	AND	Ø	=	Ø
Bit Ø	→	1	AND	Ø	=	Ø

Hier ist die Antwort also 1ØØ1ØØØØ oder dezimal 144.

Das nächste Programm hilft Ihnen vielleicht, diese Berechnungen zu üben, damit der Grundgedanke sich einprägt.

```
1Ø  PRINT CHR$ (147)
2Ø  INPUT "ERSTE ZAHL SCHREIBEN"; A
3Ø  IF A < Ø OR A > 255 THEN 2Ø
4Ø  INPUT "'AND' ODER 'OR' SCHREIBEN"; X$
5Ø  IF X$ <> "AND" AND X$ <> "OR" THEN 4Ø
6Ø  INPUT "ZWEITE ZAHL SCHREIBEN"; B
7Ø  IF B < Ø OR B > 255 THEN 6Ø
8Ø  PRINT: PRINT
9Ø  PRINT TAB (5); A; TAB (25);
1ØØ  C = A: GOSUB 5ØØ: PRINT K$
11Ø  PRINT
12Ø  PRINT TAB (5); B; TAB (25)
13Ø  C = B: GOSUB 5ØØ: PRINT K$
14Ø  PRINT TAB (25); "[8 x Grafik-* + SHIFT]"
15Ø  IF X$ = "AND" THEN C = A AND B
16Ø  IF X$ = "OR" THEN C = A OR B
17Ø  GOSUB 5ØØ
18Ø  PRINT TAB (5); A; X$; B
```

```
19Ø  PRINT TAB (25); K$
2ØØ  PRINT TAB (25); "[8 x Grafik-* + SHIFT]"
21Ø  PRINT TAB (6);
22Ø  PRINT "IST IN DEZIMAL"; C
23Ø  END
5ØØ  REM BINAER-DEZIMAL-UMWANDLUNG C IN K$
51Ø  K$ = "" : Z = C
52Ø  FOR T = 1 TO 8
53Ø  CH = INT (C/2)
54Ø  IF 2 * CH = C THEN K$ = "Ø" + K$
55Ø  IF 2 * CH <> C THEN K$ = "1" + K$
56Ø  C = CH
57Ø  NEXT T
58Ø  C = Z : RETURN
```

Die Grafikzeichen in den Zeilen 14Ø und 2ØØ sind 8mal das Zeichen auf Taste *, wenn SHIFT gedrückt wird. Der GOSUB-Befehl ist noch nicht erklärt; siehe Kapitel 14. Er entspricht einem GOTO mit variablem Rückkehr-GOTO am Ende, verwendet, um irgendeine Unteraufgabe zu erfüllen – hier die Umwandlung von C in eine Binärzahl, die als String K$ gespeichert wird. Zerbrechen Sie sich darüber jetzt nicht den Kopf.

Das Programm hat drei Eingaben: die erste Zahl, die Wahl von AND oder OR, und die zweite Zahl.

Systemvariable setzen

Sie werden sich wohl fragen, was das alles nutzen soll. Das taucht auf, wenn wir den Computer in einen bestimmten "Gemütszustand" versetzen müssen, indem wir den Inhalt eines geeigneten Speicherplatzes verändern. Das System verwendet manchmal die Binärziffern an bestimmten Plätzen (Systemvariable genannt) als *Flaggen* – das heißt, als Signale, die bestimmen, ob gewisse Einträge in Gebrauch sind oder nicht. Man kann sich ein Byte als eine Folge von acht kleinen Schaltern vorstellen, wobei Ø "aus" oder "Flagge unten" und 1 "ein" oder "Flagge oben" bedeuten. Siehe Abbildung 12.1, die versucht, Flaggen und Schalter gleichzeitig darzustellen.

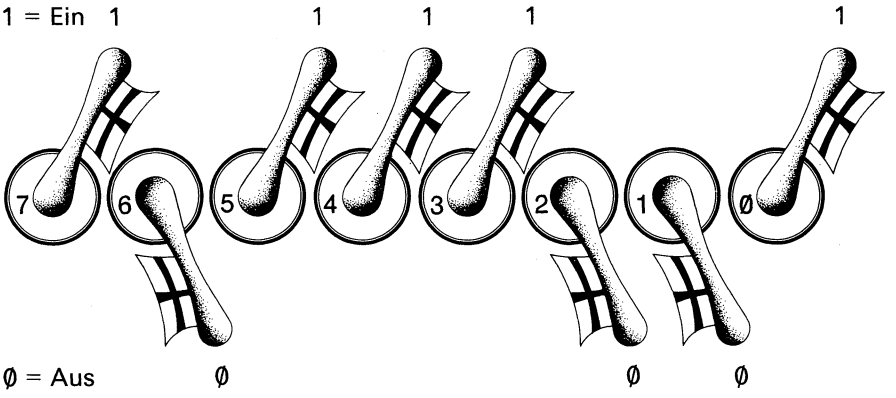


Abbildung 12.1
Ein Byte gleicht einer Reihe von acht Schaltern oder Flaggen, die sich jeweils in einem von zwei Zuständen befinden können.

Angenommen, wir möchten Bit 2 auf "Ein" schalten, gleichgültig, worauf es schon gesetzt ist, und die anderen Schalter in Ruhe lassen. Wenn wir wüßten, daß Schalter 2 *aus* ist, wäre es einfach: Man addiert zum Binärwert 4:

1ØØ11ØØ1 + ØØØØØ1ØØ = 1ØØ1111Ø1

Bit 2 *ab*geschaltet Bit 2 *ein*geschaltet

Fein! Aber nehmen wir an, wir wissen nicht, wie der Schalter gesetzt ist? 4 zu addieren, könnte schlimm werden, denn:

1ØØ1111Ø1 + ØØØØØ1ØØ = 11ØØØØØ1

Bit 2 *ein*geschaltet Hoppla Bit 2 *ab*geschaltet!

und was noch schlimmer ist, Bits 5, 4, 3 sind ebenfalls abgeschaltet, Bit 6 haben wir *ein*geschaltet. Es ist so, als wäre der Computer ein Auto, und wir hätten versucht, die Scheinwerfer einzuschalten, sie statt dessen aber ausgeschaltet, obwohl sie schon brannten, den Motor abgestellt, die Parkleuchten und die Richtungsblinker abgeschaltet – und die Hupe angestellt. Nicht das Ideale.

Das Problem sind die Übertragsziffern. Wir brauchen eine Operation, die *nur* Bit 2 betrifft und es ungeachtet seines momentanen Wertes auf 1 setzt. Wenn Sie eine Ziffer durch OR mit Ø verknüpfen, bleibt sie bekanntermaßen gleich, und wenn Sie durch OR mit 1 verknüpfen, wird daraus 1. (Prüfen Sie das anhand der obigen Tabelle nach.) Wenn wir also ein Byte durch OR mit ØØØØØ1ØØ verknüpfen, bleiben alle seine Bits gleich, bis auf Bit 2, das zu 1 wird. Kurz gesagt,

pqrstuvw OR ØØØØØ1ØØ = pqrstlvw

für alle Werte Ø, 1 von p, q, . . . w. Da wir diese Operation auf OR in Dezimal setzen können, heißt das, daß

X OR 4

dasselbe ist wie X, mit Bit 2 aber eingeschaltet.

Ebenso müssen wir, um Bit 2 auf Ø zu setzen, alle anderen aber unverändert zu lassen, die Zahl durch AND mit 11111Ø11 (251 dezimal) verbinden. Das liegt daran, daß p AND 1 = p, und zwar immer, dagegen p AND Ø = Ø. (Es ist kein Zufall, daß 251 = 255 − 4, aber nur Mathematiker sollten versuchen, hinter den Grund zu kommen.)

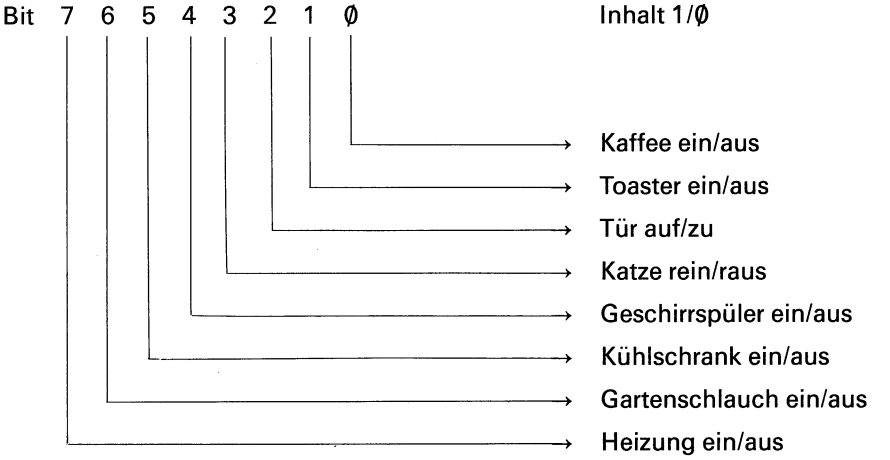
Ich werde diese Methode weiter unten gelegentlich verwenden müssen. Sie *brauchen* sie in diesen Einzelheiten nicht zu verstehen, um den Anweisungen folgen zu können (*jetzt* sagt er uns das! Stöhn, jaul), aber es trägt dazu bei, das klarzumachen, was sonst sinnloses Gefasel wäre. Vor allem sollen Sie sich einprägen, daß eine Programmzeile wie

Y = X OR 4

kein Anzeichen für schlechtes Korrekturlesen ist; sie ergibt durchaus Sinn. Und man darf wetten, daß sie an irgendeinem Speicherplatz Bit 2 einschaltet.

Aufgabe 1

(Nehmen Sie diese nicht allzu ernst, aber vielleicht trägt sie dazu bei, die Verwendung von Bits als Flaggen für die Art zu erklären, wie das System arbeitet.) Der (erfundene, aber berüchtigte) Mikro Storch 37 verwendet eine Variable namens SCHALTER, um einige seiner wahlweise käuflichen Peripheriegeräte wie folgt zu steuern:



Sie wissen nicht, worauf die Bits gesetzt sind, aber Sie wollen die Katze hinaustun. Welchen Befehl geben Sie?

Lösung

Aufgabe 1

Bit 3 steuert die Katze, und 1111Ø111 ist 247 dezimal, also bewirkt

```
SCHALTER = SCHALTER AND 247
```

das Gewünschte. Miaaauuuu!

13 PEEK und POKE

Jeder Platz im Commodore 64-Speicher hat seine eigene Adreßbenennung. Durch Veränderung des Inhalts geeigneter Speicherplätze können Sie den Computer veranlassen, eine ganze Reihe von Aufgaben zu erfüllen. Zwei Befehle zeigen Ihnen, was an einer bestimmten Adresse gespeichert ist, und wie Sie es ändern können.

Beim Programmieren in BASIC können Sie in der Regel dem Computer den Großteil der Arbeit überlassen, und Fragen wie: "Wo im Speicher ist das eigentlich abgelegt?" bekümmern den Programmierer nicht. Sobald Sie aber die fortgeschritteneren Einrichtungen des 64 nutzen wollen, etwa Farbe und Klang, die über Standard-BASIC hinausgehen, erlangen solche Fragen Bedeutung. Der 64 ist Anfängern gegenüber nicht so entgegenkommend wie manches andere Gerät, und verlangt vom Programmierer, daß er eine große Menge Organisationsdetails im Kopf behält. Wenn Sie sich aber die Mühe machen, ein paar einfache Grundideen zu beherrschen, reagiert der Computer als Ausgleich dafür viel schneller auf Ihre Befehle. Sie können sogar einfach die untenstehenden Befehle haargenau abschreiben und die erwähnten Einrichtungen nutzen, ohne eigentlich zu wissen, was Sie tun. Sollte Ihnen dieses Kapitel also zu anstrengend vorkommen, dann überblättern Sie es und kommen Sie später darauf zurück. Wie bei den meisten Aufgaben wird das Dasein aber erheblich erleichtert, wenn Sie nicht nur das *Wie* der Befehle verstanden haben, sondern auch das *Warum* der Gründe.

Speicheraufbau

Der Computerspeicher ist außerordentlich starr und systematisch aufgebaut. Jeder Speicherplatz erhält eine feste Nummer, seine *Adresse*. Beim 64 gehen die Adressen von Ø bis 63535. Jede Adresse enthält eine Zahl zwischen Ø und 255 (den Informationswert eines einzelnen Zeichens). Diese Zahl ist nun in *Binär*schreibweise als ein String von acht Nullen oder Einsen gespeichert, so daß jede Adresse ein einzelnes *Byte* enthält. Für manche Zwecke genügt es, sich die Zahl als eine gewöhnliche Dezimalgröße vorzustellen, und zunächst halte ich es hier so.

Der Computer verfügt über zwei Speicherarten:

1. ROM (Read Only Memory = Nur-Lese- oder Festspeicher), der sein Betriebssystem enthält, die eingebauten Arbeitsanweisungen, durch die er funktioniert.
2. RAM (Random Acces Memory – Speicher mit wahlfreiem Zugriff), der das speichert, was geändert werden kann: BASIC-Programm, Bildschirmdisplays, Notizblock für Berechnungen, und so weiter.

Verändern können Sie den Inhalt von RAM, aber nicht von ROM. Wenn Sie im Speicher herumwühlen, können Sie also *schlimmstenfalls* den Computer in einen Zustand versetzen, der ein Rücksetzen erfordert; an diesem Punkt müssen Sie ihn kurz abschalten. Sie verlieren den Inhalt von RAM, richten aber keinen dauernden Schaden an, so daß Sie unbesorgt experimentieren dürfen, ohne sich Sorgen darüber zu machen, was Sie dem Gerät antun könnten.

Beispielsweise sehen die letzten zehn Bytes von ROM aus wie Tabelle 13.1, und im 64 sind noch viele Tausend Bytes Information mehr untergebracht. Wenn der Computer arbeitet, sausen all die kleinen Nullen und Einsen im Inneren wie Ameisen durcheinander (na schön, die Einser sind Ameisen, die Nullen Nicht-Ameisen), bis sie da landen, wo Sie, der Programmierer (und die Leute, die das ROM-Programm geschrieben haben) das wünschen. Erfaßt Sie nicht Ehrfurcht und Demut bei dem Gedanken, Herrschaft über Leben und Tod so vieler Ameisen zu besitzen?

Tabelle 13.1

Adresse	Inhalt	
	binär	dezimal
65526	Ø1Ø1ØØ1Ø	82
65527	Ø1Ø1ØØ1Ø	82
65528	Ø1ØØØØ1Ø	66
65529	Ø1Ø11ØØ1	89
6553Ø	Ø1ØØØØ11	67
65531	11111111	254
65532	111ØØØ1Ø	226
65533	11111ØØØ	252
65534	Ø1ØØ1ØØØ	72
65535	11111111	255

PEEK

Der Befehl, mit dem Sie in einen Speicherplatz blicken und erkennen (aber nicht *verändern*) können, was sich dort befindet, heißt:

PEEK

Um etwa den Inhalt von Adresse 65535 zu finden, geben Sie

PRINT PEEK (65535)

ein. Auftauchen sollte die Zahl 255; wenn nicht, habe ich Sie angelogen.

Mit folgendem Programm (das im Verlauf des Kapitels noch ergänzt wird, daher die sonderbaren Zeilennummern) können Sie durch PEEK einen Blick in jeden beliebigen Speicherteil werfen:

```
10  REM PEEKING ROUTINE
20  INPUT "ADRESSE FUER START"; AD: IF AD < 0 OR AD >
    65535 THEN 20
30  H = 0
60  P = PEEK (AD)
90  PRINT AD; TAB (8); P; TAB (16);
100 IF P < 32 OR P > 127 AND P < 160 THEN PRINT "CTRL";
110 IF P > 31 AND P < 128 OR P > 159 THEN PRINT CHR$ (P);
120 PRINT TAB (24);
130 IF P > 191 THEN PRINT "+";
140 PRINT
230 AD = AD + 1: IF AD > 63535 THEN STOP
240 H = H + 1: IF H < 20 THEN 60
250 GET A$: IF A$ = "" THEN 250
260 IF A$ = "S" THEN STOP
270 GOTO 30
```

Fahren Sie das mit RUN und geben Sie 65526 ein, wenn die Startadresse verlangt wird. Sie sollten den Inhalt der letzten zehn Speicherplätze im ROM, wie oben aufgeführt, erhalten (ohne die Binärversion, aber mit einer Zusatzkolonne, die das Zeichen angibt, dessen Code die von PEEK erkannte Zahl ist). Da habe ich sie her.

Allgemein können Sie jeden beliebigen Startwert eingeben; das Programm ist mit einer Narrensicherung ausgestattet, um eine Eingabe zwischen 0 und 65535 sicherzustellen. Sie erhalten eine "Seite" Display. Drücken Sie Taste "S" zum Anhalten und irgendeine beliebige Taste für die nächste Seite. Konzentrieren Sie sich auf die Zahlen in den ersten beiden Kolonnen, die Adressen und ihren Inhalt; die dritte Kolonne erkläre ich unten.

Die konkreten Zahlen hier ergeben für Sie wohl nicht viel Sinn. Begreiflich – sie sind im eigenen privaten und persönlichen Maschinencode des 6510-Mikroprozessorchips geschrieben. Es würde ein weiteres Buch (etwa den "Reference Guide") erfordern, Sie hier auch nur auf den Weg zu bringen. Wenn Sie aber immer wieder eine Taste für eine neue Seite drücken, wird Ihnen immerhin klar werden, daß im Inneren Ihres Computers enorm viel Information steckt. Manches davon erkennen Sie vielleicht sogar:

Aufgabe 1

Fahren Sie das obige Programm mit RUN und geben Sie 41118 ein, dann wiederholen Sie das Ganze mit 41374. Welchen Bereich des Speichers haben Sie vor sich?

Narziß schlägt wieder zu

Der Legende nach verliebte Narziß sich in sein eigenes Spiegelbild in einer Quelle und wurde in eine Blume verwandelt (ein Racheakt von Artemis, als Narziß eine Nymphe namens Echo nicht zu schätzen wußte). Ich riskiere dasselbe Schicksal, wenn ich nun die PEEK-Routine mit PEEK auf sich selbst blicken lasse!

BASIC-Programme werden normalerweise beginnend bei Adresse 2Ø48 gespeichert (außer, Sie jonglieren mit den richtigen RAM-Bits, um das zu ändern, wobei ich davon ausgehe, daß Sie das nicht getan haben; wenn Sie wissen, wie man den BASIC-Zeiger verändert, brauchen Sie dieses Kapitel ohnehin nicht zu lesen). Wenn Sie die Routine mit RUN fahren und 2Ø48 eingeben, erhalten Sie auf der ersten Displayseite:

Adresse	Inhalt/Dezimal	Zeichen
2Ø48	Ø	CTRL
2Ø49	23	CTRL
2Ø5Ø	8	CTRL
2Ø51	1Ø	CTRL
2Ø52	Ø	CTRL
2Ø53	143	CTRL
2Ø54	32	(Leerstelle)
2Ø55	8Ø	P
2Ø56	69	E
2Ø57	69	E
2Ø58	75	K
2Ø59	73	I
2Ø6Ø	78	N
2Ø61	71	G
2Ø62	32	(Leerstelle)
2Ø63	82	R
2Ø64	79	O
2Ø65	85	U
2Ø66	84	T
2Ø67	73	I
2Ø68	78	N
2Ø69	69	E

(Streng genommen, stehen die beiden letzten Zeilen auf der zweiten Seite des Displays, und Sie müssen eine Taste drücken, um sie zu sehen – zusammen mit anderen Einträgen. Aber ich brauche sie der Ordnung halber.)

Die dritte Kolonne verwandelt die im Speicherplatz enthaltene Zahl in das entsprechende Zeichen (siehe Handbuch, Anhang F, S. 135–137). Manche Zeichen haben seltsame Auswirkungen auf das Display, wenn Sie versuchen, sie mit PRINT anzuzeigen, und aus diesem Grund habe ich dafür gesorgt, daß stattdessen CTRL ("control") angezeigt wird. Bei Codes von mehr als 191 steht ein "+"-Zeichen, weil sie dieselben Zeichen ergeben wie bestimmte kleinere Zahlen. Wenn Sie weiter nachforschen, wird Ihnen außerdem auffallen, daß Code 34 eine höchst eigenartige Anzeige ergibt, die zu erklären der Platz fehlt. Im übrigen ist sie harmlos.

Nicht alles in der Tabelle ergibt auf Anhieb Sinn, aber die Überschrift PEEKING ROUTINE können Sie ganz deutlich sehen. Wenn Sie weitergehen zu späteren Seiten des Displays, stoßen Sie auf andere Teile des Programms, die augenblicklich erkennbar sind, vermischt mit weniger einleuchtendem Material. Das Unklare codiert die Zeilennummern auf nicht gerade durchsichtige Weise oder verdichtet die BASIC-Schlüsselwörter zu 1 Byte-Codes, um Speicherplatz zu sparen. So steht etwa 143 an Speicherplatz 2Ø53 für REM. Jawohl, Herrschaften, das Programm ist im Computer *wirklich* gespeichert, und wenn Sie an der richtigen Stelle nachgucken, können Sie es *sehen*.

Hmmmm . . . bis jetzt noch nichts Blumiges zu merken. Dafür etwas anderes . . .

POKE

Sie können das Programm nicht nur sehen, sondern sogar verändern. Geben Sie diese direkten Befehle ein:

```
POKE 2Ø58,86
POKE 2Ø6Ø,83
POKE 2Ø61,72
```

Lassen Sie sich mit LIST das Programm zeigen. Sieht aus wie vorher? Nicht ganz; beim REM steht jetzt

```
PEEVISH ROUTINE
```

Wenn Sie die Routine mit RUN noch einmal fahren und auch hier bei 2Ø48 beginnen, können Sie sehen, was vorgegangen ist: Bei den Adressen 2Ø58, 2Ø6Ø und 2Ø61 ist der Inhalt der Codes für V, S, H, statt für K, N, G verändert worden. Mit dem POKE-Befehl können Sie also ein bestimmtes Byte an einer bestimmten Adresse in einen Speicherplatz setzen.

Aufgabe 2

Verändern Sie PEEKING ROUTINE mit zwei POKE-Befehlen in PARKING ROUTINE.

Jetzt habe ich Pandoras Büchse aber ganz aufgemacht. Wenn Sie nach Belieben mit POKE in Speicherplätze einzugreifen vermögen, können Sie alles verändern, was Sie wollen, vorausgesetzt, Sie wissen, wo es sich befindet, und es ist nicht sicher im ROM verwahrt. Die nützlichen Adressen können Sie aus dem "Reference Guide" erfahren – ich werde in späteren Kapiteln nicht wenige erwähnen. Jedenfalls können Sie nach Herzenslust mit PEEK in der ganzen Speicherkarte herumfuhrwerken, und wenn Sie die Lust nach einem gelegentlichen POKE anwandelt, tun Sie sich keinen Zwang an. Nur eine Warnung: Wahllose POKE-Versuche bringen selten viel ein; man muß vom System einiges verstehen, bevor man daran herumbasteln kann, ganz wie bei einem Auto.

Bildschirmfarben

Als eine Einführung in die POKErei, die von außen her das Betriebssystem nutzt, wollen wir uns die Bildschirmfarben ansehen. Ständig sind drei Farben vorhanden: Der Rand (*Border*) des Bildschirmbereichs, die Hintergrundfarbe auf dem Schirm (die ich *Paper* nennen werde), und die Farbe des angezeigten Zeichens (*Ink*). Sie können die Inkfarbe mit der CTRL-Taste verändern (siehe Handbuch, S. 1Ø) . . . aber wie ändern Sie Paper und Border?

Die Codes für die Steuerung dieser Farben sind in zwei Plätzen gespeichert:

53280 Borderfarbe

53281 Paperfarbe

und für Ink gibt es auch einen:

646 laufende Inkfarbe

den Sie statt der CTRL-Methode verwenden können. Soll etwas verändert werden, brauchen Sie in der Adresse mit POKE nur den erforderlichen Farbcode einzugeben. Die Farbcodes sind:

Ø Schwarz	8 Orange
1 Weiß	9 Braun
2 Rot	1Ø Hellrot
3 Cyan	11 Dunkelgrau
4 Purpur	12 Mittelgrau
5 Grün	13 Hellgrün
6 Blau	14 Hellblau
7 Gelb	15 Hellgrau

So liefert zum Beispiel

POKE 53280, 9 einen braunen Rand

POKE 53281, 4 purpurnen Hintergrund

POKE 646,11 dunkelgraue Inkfarbe

Die Paper- und Borderveränderungen wirken sich sofort auf den ganzen Bildschirm aus. Die Inkveränderung betrifft aber nur neu angezeigte Zeichen; die alten bleiben so, wie sie waren.

Der Zeichensatz

Als Schlußschnörkel will ich Ihnen zeigen, wo der 64 die Information speichert, um seine Zeichen hervorzubringen. Damit das Programm aber funktioniert, müssen zwei knifflige Punkte berücksichtigt werden.

Der erste: Die Zeicheninformation kann nur dann richtig gesehen werden, wenn die Zahlen *binär* geschrieben sind, weil jedes binäre "1" einen winzigen Punkt auf dem Bildschirm bestimmt und das Zeichen aus diesen Punkten aufgebaut ist. Ich muß also eine Routine zur Binärumwandlung aus Kapitel 12 einfügen.

Der zweite Haken: Die Adressen für den Zeichensatz sind zwar durchaus leicht erreichbar – laut "Reference Guide" gehen sie von 53248 bis 57343 – aber wenn Sie dort mit PEEK hineingehen, finden Sie die richtige Information nicht! Der 64 ist ein kompliziertes Ding und schiebt im Speicher Inhalte je nach seinen Bedürfnissen herum. Er jongliert sogar mit Adreßbezeichnungen, so daß eine gegebene Adresse mehr als eine Bedeutung haben kann, je nach dem "Gemütszustand", in dem die Maschine sich gerade befindet.

In ihrem normalen Zustand beziehen die obigen Adressen sich ganz und gar nicht auf den ROM-Zeichensatz, sondern auf einen RAM-Bereich, der für Input/Output verwendet wird. Der 64 kann aber durch ein bißchen POKErei dazu bewogen werden, diese Adressen dem ROM anzuhängen. Zusätzliches Poking ist nötig, um währenddessen das Keyboard abzuschalten, weil sonst die schlimmsten Dinge passieren können. Sobald man mit dem PEEKen fertig ist, muß die Maschine natürlich wieder in ihren Normalzustand versetzt werden, damit sie wie gewohnt arbeiten kann. Hört sich ein bißchen umständlich an, ist aber mein Problem, nicht Ihres, ja? Tun müssen Sie also folgendes:

Fügen Sie der PEEKING ROUTINE oben folgende Zeilen ein:

```
4Ø  POKE 56334, PEEK (56334) AND 254
5Ø  POKE 1, PEEK (1) AND 251
7Ø  POKE 1, PEEK (1) OR 4
8Ø  POKE 56334, PEEK (56334) OR 1
```

Das ist das Umschalten im Speicher. Zeile 4Ø unterbindet Unterbrechungen (das Keyboard); 5Ø holt die Information über den Zeichensatz an ihren Platz; 7Ø schiebt sie wieder hinaus; und 8Ø ermöglicht wieder Unterbrechungen. Dann die Umwandlung in Binär:

```
14Ø  PRINT TAB (28);
15Ø  B$ = ""
16Ø  FOR S = 1 TO 8
17Ø  PH = INT (P/2)
18Ø  IF P = 2 * PH THEN B$ = CHR$ (46) + B$
19Ø  IF P <> 2 * PH THEN B$ = CHRS (166) + B$
2ØØ  P = PH
21Ø  NEXT S
22Ø  PRINT B$
```

Beachten Sie, daß die neue Zeile 14Ø die vorherige überschreibt. Ändern Sie schließlich Zeile 24Ø so ab, daß sie lautet:

```
24Ø  H = H + 1: IF H < 2Ø THEN 4Ø
```

Fahren Sie mit RUN und geben Sie als Startadresse 53248 ein. Sie sehen die Formen der Zeichen, A, B, C, . . . an der rechten Randseite auftauchen. Statt Ø und 1 zu verwenden, habe ich versucht, die Formen dadurch deutlicher hervorzuheben, daß ich einen Punkt und ein kariertes Quadrat verwendet habe – Zeichen 46 und 166, daher die Zeilen 18Ø und 19Ø im Programm. Wenn Sie Ø und 1 vorziehen, ändern Sie diese Zeilen ab zu:

```
18Ø  IF P = 2 * PH THEN B$ = "Ø" + B$
19Ø  IF P <> 2 * PH THEN B$ = "1" + B$
```

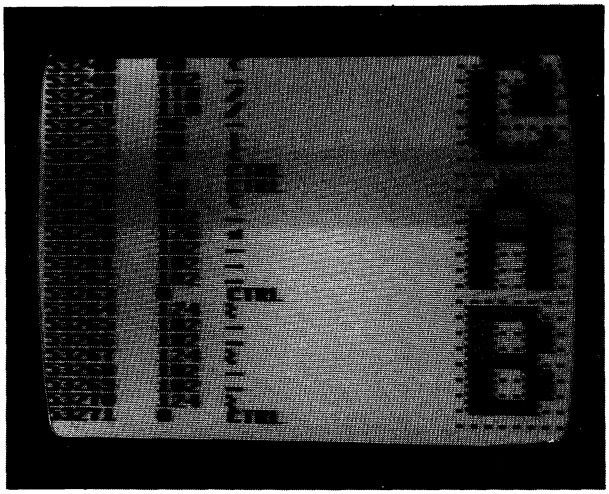


Abbildung 13.1
Zeichendaten, aus dem ROM entwendet.

Die verschiedenen Zeichenarten werden gespeichert, beginnend an den folgenden Adressen:

53248	Großbuchstaben
5376Ø	Grafik
54272	Großbuchstaben in Negativschrift (Paper und Ink vertauscht)
54784	Grafik in Negativschrift
55296	Kleinbuchstaben
558Ø8	Großbuchstaben und Grafik
563Ø2	Kleinbuchstaben in Negativschrift
56832	Großbuchstaben und Grafik in Negativschrift

Ein bestimmtes Maß an Wiederholung tritt wegen der Art auf, wie der 64 zwei verschiedene Zeichensätze verwendet (Handbuch, Anhang E, S. 132–134).

Sie können die Zeicheninformation in Programmen verwenden – beispielsweise, um Zeichen in Großform anzuzeigen. Verwenden Sie *jedesmal*, wenn Sie das tun, die Zeilen 4Ø, 5Ø, 7Ø, 8Ø aus dem obigen Programm. Sie können sogar selbst neue Zeichensätze entwerfen . . . aber das ist für dieses Buch zu hoch. Schlagen Sie also nach im "Reference Guide", S. 1Ø7–113.

Lösungen

Aufgabe 1

1. Eine Tabelle von BASIC-Schlüsselwörtern.
2. Eine Tabelle von Fehlermeldungen. Das letzte Zeichen in jedem Schlüsselwort und in jeder Meldung ist verändert worden, um eine Markierung zu liefern, die dem Computer zeigt, wo das Wort aufhört. Drücken Sie COMMODORE + SHIFT und fahren Sie das Programm erneut. Nun stehen die Wörter in Kleinschrift und die Merkzeichen in Großschrift. Der Gebrauch solcher Merkzeichen ist ein listiger Kniff, um Speicherplatz zu sparen.

Aufgabe 2

POKE 2Ø56,65

POKE 2Ø57,82

14 Subroutinen (Unterprogramme)

Wenn Sie feststellen, daß Sie immer wieder dieselben Codeteile mit kleinen Abwandlungen schreiben, oder wenn Ihnen zu entfallen beginnt, was Teile eines Programms eigentlich leisten sollen, dann könnten Subroutinen angebracht sein.

Oft wird innerhalb eines Programms eine bestimmte Folge von Befehlen mehrmals verwendet. Es *kann* Ihnen gelingen, durch überlegte Verwendung von GOTO-Befehlen das wiederholte Schreiben der Sequenz zu vermeiden, aber das genügt nicht immer. Der Befehl

```
GOSUB
```

ist wie ein GOTO, das "weiß, wo es herkommt". Bei der Begegnung mit einem anderen Befehl:

```
RETURN
```

springt das Programm zu der Zeile *nach* der jeweiligen GOSUB-Zeile zurück, von der es zunächst ausgegangen war.

Dem GOSUB muß eine Zeilennummer folgen, etwa

```
GOSUB 1000
```

Damit wird das Programm zu der Folge von Befehlen geschickt, die mit Zeile 1000 beginnen (genau wie bei GOTO 1000). Der Teil des Programms, der zwischen Zeile 1000 und dem RETURN-Befehl liegt, wird als *Subroutine* bezeichnet, und man spricht davon, daß sie vom GOSUB-Befehl *aufgerufen* wird. Eine Subroutine ist also eine Art "Miniprogramm", das innerhalb eines größeren Programms eine bestimmte Aufgabe erfüllen soll.

Für die Verwendung von Subroutinen gibt es mindestens zwei gute Gründe:

1. Sie vermeiden unnötige Wiederholung und führen zu kurzen und effektiven Programmen.
2. Sie erlauben dem Programmierer, ein Programm in leicht zu erfassenden Teilen zu strukturieren, wodurch Testen und Debugging einfacher wird.

Es lohnt, sich bald an Subroutinen zu gewöhnen, vor allem beim 64, wo viele Standardmöglichkeiten am besten durch die Entwicklung effektiver Subroutinen genutzt werden.

Beispiel: Die Veränderung der Hintergrundfarbe (Paper). Ein Miniprogramm dafür hat die Form:

```
3000  REM PAPERFARBE
3010  POKE 53281, PAPER
```

Hier ist PAPER eine Variable, die je nach der gewünschten Farbe im Bereich Ø–15 gesetzt wird (siehe Kapitel 13). Um das als Subroutine nutzen zu können, müssen Sie eine zusätzliche Zeile anfügen:

```
3Ø2Ø  RETURN
```

Jetzt können Sie statt eines Befehls POKE 53281 etc. einen GOSUB 3ØØØ-Befehl verwenden, was zu größerer Klarheit führt. Nehmen wir als Beispiel an, Sie wollen Farbveränderungen dazu verwenden, den Reiz eines Werbedisplays zu steigern. Dann könnten Sie ungefähr so verfahren:

```
1ØØ  REM HAUPTPROGRAMM
11Ø  PRINT CHR$ (147)          [Schirm löschen, Cursor home]
12Ø  PRINT: PRINT: PRINT       [3 leere Zeilen]
13Ø  PRINT TAB (1Ø); "BIRKHAEUSER VERLAG"
14Ø  PRINT
15Ø  PAPER = 2: GOSUB 3ØØØ    [Paper rot]
16Ø  PRINT TAB (5); "COMPUTERKATALOG NOVEMBER 1987"
17Ø  PRINT
18Ø  PAPER = 4: GOSUB 3ØØØ    [Paper purpur]
19Ø  PRINT "CRAY 1 – PROGRAMMIEREN LEICHT GEMACHT"
2ØØ  PRINT
21Ø  PAPER = 7: GOSUB 3ØØØ    [Paper gelb]
22Ø  PRINT "SPIELE AUF IHREM VAX"
23Ø  PRINT
24Ø  PAPER = 8: GOSUB 3ØØØ    [Paper orange]
25Ø  PRINT "COMPILERGENERATOREN
     FUER BLUTIGE ANFAENGER"
26Ø  PRINT
27Ø  PAPER = 13: GOSUB 3ØØØ   [Paper hellgrün]
28Ø  GOTO 1ØØ                 [fang von vorne an]
```

Der Hauptpunkt, der begriffen werden muß: Begegnet das Programm einem GOSUB, sagen wir, dem in Zeile 15Ø, dann springt es zur angesprochenen Zeile (hier 3ØØØ) und führt Befehle aus, bis es auf RETURN stößt, um anschließend zur nächsten Zeile *nach* 15Ø zu springen, das ist also Zeile 16Ø. Diese Wirkung *könnten* Sie nun auch erreichen, wenn Sie

```
15Ø  PAPER = 2: GOTO 3ØØØ
```

verwenden und das RETURN verändern zu:

```
3Ø2Ø  GOTO 16Ø
```

Aber . . . sobald Sie Zeile 18Ø erreichen, das nächste GOSUB 3ØØØ, schickt das RETURN Sie zurück zu Zeile 19Ø, so daß die abgeänderte Zeile (3Ø2Ø) nicht das Richtige leisten würde. Ebenso kehrt das GOSUB in Zeile 21Ø durch RETURN zu Zeile 22Ø zurück, das GOSUB in Zeile 24Ø zu Zeile 25Ø, und das GOSUB in Zeile 27Ø zu Zeile 28Ø. Das ist der entscheidende Unterschied zwischen GOSUB und GOTO, und dadurch ist GOSUB als Befehl viel leistungsfähiger.

Korrektur

Wenn Sie versuchen, das obige Programm wirklich zu fahren, werden Sie feststellen, daß alles ganz schnell abläuft, so daß die Werbung nahezu unleserlich ist. Das geht natürlich nicht; wir müssen den Ablauf dadurch verlangsamen, daß wir an den angemessenen Stellen Pausen einfügen. Das geht gut mit einer zusätzlichen Subroutine:

```
2ØØØ  REM PAUSE ETWA 2 SEKUNDEN
2Ø1Ø  FOR T = 1 TO 15ØØ
2Ø2Ø  NEXT T
2Ø3Ø  RETURN
```

Nun können wir diese Subroutine jedesmal aufrufen, wenn wir eine Pause wollen:

```
145  GOSUB 2ØØØ
175  GOSUB 2ØØØ
2Ø5  GOSUB 2ØØØ
235  GOSUB 2ØØØ
265  GOSUB 2ØØØ
```

Diese Art, eine Subroutine zu verwenden, um nachträglich zusätzliche Befehle einzufügen, wird als *Korrekturroutine* bezeichnet. Sie belegt eine weitere Methode, die Vielseitigkeit des GOSUB-Befehls zu nutzen.

Programmstruktur

Die Grundstruktur des Programms sieht jetzt aus wie Abbildung 14.1. Es gibt eine "Hauptrichtung" im Programmfluß mit wiederholten Abschweifungen zu den Subroutinen.

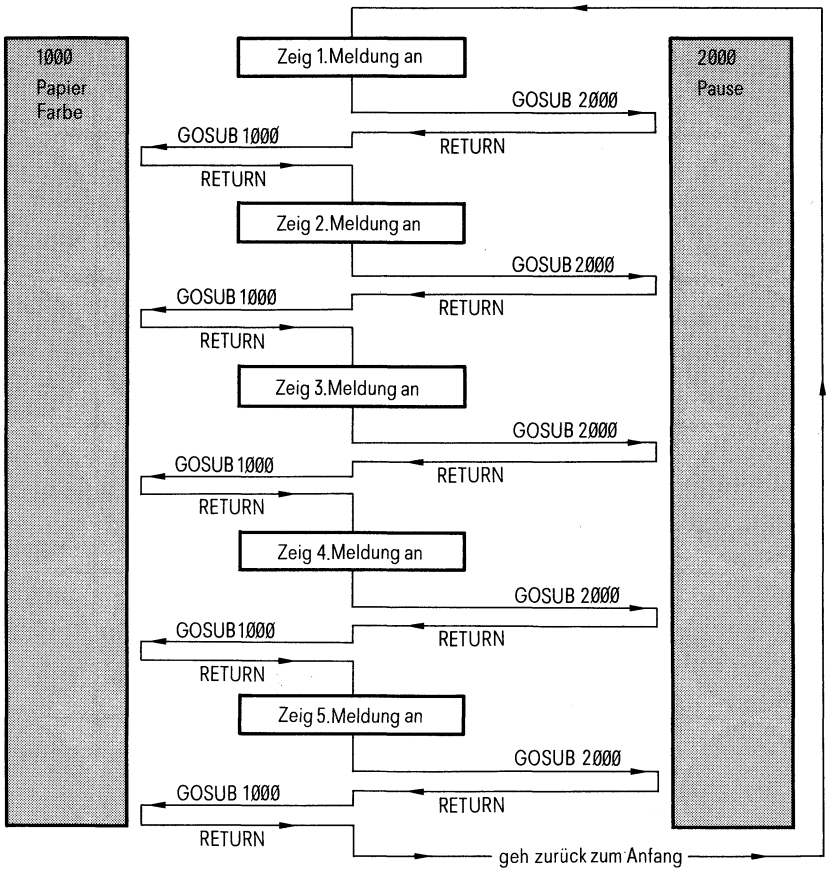


Abbildung 14.1
Steuerfluß im Werbeprogramm. Beachten Sie die wiederholten Aufrufe der beiden Subroutinen und die verschiedenen Rücksprungpositionen.

Das Vorrecht der Weiblichkeit . . .

. . . ist, ihren Sinn zu ändern (heißt es). Angenommen, Sie kommen zu der Meinung, die *Border*farbe sollte sich verändern und der Paperfarbe entsprechen. Wenn Sie jede Farbveränderung mit POKE 53281,2 und so weiter direkt geschrieben hätten, müßten Sie eine Menge POKE 5328Ø,2 einfügen, um das Programm hinzubringen. Mit Subroutinen leistet aber eine einzige Zeile dasselbe:

```
3Ø15  POKE 5328Ø, PAPER
```

Aufgabe 1

Damit es wirklich ordentlich wird, sollten Sie auch die Borderfarbe auf Dunkelblau setzen. Tun Sie das.

Top-down-Prinzip

Ein weiterer Vorteil von Subroutinen ist der, daß man mit ihnen ein Programm "top-down" (also von oben nach unten) entwerfen kann. Das heißt, grob gesprochen, man schiebt die Verfeinerungen für später auf. "Wer die Mark nicht ehrt, ist den Pfennig nicht wert . . .", wie es im Sprichwort nicht heißt . . . aber der Rat ist trotzdem besser. Nehmen wir das obige Beispiel, tun wir so, als hätten wir es nicht so angefangen, wie es der Fall war, und wenden wir das Top-down-Prinzip an. Sie werden sehen, daß das Ergebnis unauffällig subtile verbessert wird.

Erstes Anliegen: Die wesentlichen Unteraufgaben bestimmen. Mir fallen auf Anhieb drei ein:

1. Eine Meldung anzeigen.
2. Pause.
3. Farbe von Paper (und/oder) Border ändern.

Diese werden jeweils fünfmal verwendet. Wir beschließen also, drei Subroutinen zu schreiben, um sie zu bewältigen:

```
1ØØØ  REM ZEIG MELDUNG AN
(etwas, um eine Meldung M$ anzuzeigen)      1. Subroutine
????  RETURN

2ØØØ  REM PAUSE
(etwas für rund 2 Sekunden Pause)           2. Subroutine
????  RETURN

3ØØØ  REM PAPERFARBE
(etwas, um die Paperfarbe zu der von
einer Variablen PAPER bestimmten            3. Subroutine
zu verändern)
????  RETURN
```

Hier ist ???? eine Zeilennummer, die festzulegen ist, sobald wir die Subroutinen konkret *schreiben*. Vorerst brauchen wir aber nur darauf zu achten, welche Variablen sie verwenden werden.

Wir müssen außerdem den Computer in die richtigen Startbedingungen versetzen (Bildschirm löschen, Cursor home, etc.):

4. Initialisiere System.

Wir brauchen also eine vierte Subroutine:

```
4000  REM INITIALISIEREN
(etwas, um die erforderlichen Bedingungen
zu schaffen)
????  RETURN
```

4. Subroutine

Das Hauptprogramm

Version 1

Das Hauptprogramm nimmt nun die Form an:

```
100  REM HAUPTPROGRAMM
110  GOSUB 4000                    [initialisieren]
120  M$ = "BIRKHAEUSER VERLAG"
130  GOSUB 1000                    [1. Meldung anz.]
140  GOSUB 2000                    [Pause]
150  PAPER = 2: GOSUB 3000         [Paper rot]
160  M$ = "COMPUTERKATALOG NOVEMBER 1987"
170  GOSUB 1000                    [2. Meldung anz.]
180  GOSUB 2000                    [Pause]
190  PAPER = 4: GOSUB 3000         [Paper purpur]
200  M$ = "CRAY 1 – PROGRAMMIEREN LEICHT GEMACHT"
210  GOSUB 1000                    [3. Meldung anz.]
220  GOSUB 2000                    [Pause]
230  PAPER = 7: GOSUB 3000         [Paper gelb]
240  M$ = "SPIELE AUF IHREM VAX"
250  GOSUB 1000                    [4. Meldung anz.]
260  GOSUB 2000                    [Pause]
270  PAPER = 8: GOSUB 3000         [Paper orange]
```

```
28Ø  M$ = "COMPILERGENERATOREN FUER BLUTIGE
     ANFAENGER"
29Ø  GOSUB 1ØØØ                    [5. Meldung anz.]
3ØØ  GOSUB 2ØØØ                    [Pause]
31Ø  PAPER = 13: GOSUB 3ØØØ        [Paper hellgrün]
32Ø  GOTO 1ØØ                      [das Ganze wieder-
                                    holen]
```

Version 2

Wir haben diese Subroutinen noch nicht geschrieben . . . aber bevor wir es tun, ist jetzt ziemlich klar, daß das Hauptprogramm nicht vollkommen befriedigt. Es enthält viele Wiederholungen – derselbe Ablauf wird fünfmal wiederholt, es ändern sich nur Meldungen und Farbcodes. Wäre da eine Schleife nicht effektiver?

Allerdings – aber nur, wenn wir *Arrays* verwenden, von denen wir bis Kapitel 25 offiziell nichts wissen. Das heißt, fünf Variable

M$ (1) M$ (2) M$ (3) M$ (4) M$ (5)

enthalten die Meldungen, und

P (1) P (2) P (3) P (4) P (5)

die Paperfarben. Läuft der Schleifenzähler S von 1 bis 5, dann können wir uns auf die Variable S im Programm beziehen als M$ (S) oder P (S). Die tatsächlichen Werte dieser Variablen müssen natürlich irgendwo gesetzt werden – ein geeigneter Platz ist innerhalb der Subroutine *Initialisieren*. Jetzt haben wir also:

```
1ØØ  REM HAUPTPROGRAMM
11Ø  GOSUB 4ØØØ                    [initialisieren]
12Ø  FOR S = 1 TO 5
13Ø  M$ = M$ (S) GOSUB 1ØØØ        [zeig Meldung S an]
14Ø  GOSUB 2ØØØ                    [Pause]
15Ø  PAPER = P (S): GOSUB 3ØØØ     [Paper auf Farbe S]
16Ø  NEXT S                        [zurück zur Schleife]
17Ø  GOSUB 2ØØØ                    [Pause]
18Ø  PRINT CHR$ (147)              [Schirm leer/Cursor home]
19Ø  GOTO 12Ø                      [das Ganze wiederholen]
```

Beachten Sie, daß nach Zeile 19Ø nicht *erneut* initialisiert werden muß; wir verwenden GOTO 1ØØ nicht.

Nun haben wir eine sehr klare Vorstellung von der ganzen Programmstruktur – aber von den Subroutinen haben wir immer noch keine geschrieben! Das ist der große Vorteil des Top-down-Prinzips: Wir müßten im anderen Fall alle diese Programmteile ebenfalls schreiben, und das Ganze wäre wohl durcheinandergeraten. Schlimmer noch, die gesamte Aufgabe würde viel abschreckender wirken und unser Selbstvertrauen beschädigen. Wie lautet Jones' Erstes Computergesetz? "Verschiebe nie auf morgen, was du übermorgen kannst besorgen."

In diesem Geiste:

Aufgabe 2

Schreiben Sie die Subroutinen in den Zeilen 1000, 2000, 3000 und 4000, die noch benötigt werden, damit das Werbeprogramm läuft, und probieren Sie es anschließend aus.

Aufgabe 3

Erfinden Sie sieben zusätzliche naheliegende Buchtitel für das Birkhäuser-Angebot 1987 im Computershop, und wandeln Sie das Programm ab, um sie in den Katalog aufzunehmen.

Lösungen

Aufgabe 1

Fügen Sie eine Zeile ein:

```
115  POKE 53280,6
```

Aufgabe 2

Die Subroutinen lauten:

```
1000  REM DISPLAYMELDUNG
1010  PRINT                          [leere Zeile]
1020  PRINT M$                       [Meldung]
1030  RETURN

2000  REM PAUSE VON ETWA 2 SEKUNDEN
2010  FOR T = 0 TO 1500             ["Tunix"-
2020  NEXT T                         Schleife]
2030  RETURN
```

```
3000  REM PAPERFARBE
3010  POKE 53281, PAPER
3020  RETURN
4000  REM INITIALISIEREN
4010  DIM M$ (5): DIM P (5)      [Dimensionierungsarrays, Kap. 25]
4020  M$ (1) = "BIRKHAEUSER VERLAG"
4030  M$ (2) = "COMPUTERSHOP-KATALOG NOVEMBER 1987"
4040  M$ (3) = "CRAY 1 – PROGRAMMIEREN LEICHT GEMACHT"
4050  M$ (4) = "SPIELE FUER IHREN VAX-COMPUTER"
4060  M$ (5) = "COMPILERGENERATOREN FUER BLUTIGE
      ANFAENGER"
4200  P (1) = 2
4210  P (2) = 4
4220  P (3) = 7
4230  P (4) = 8
4240  P (5) = 13
4400  PRINT CHR$ (147)   [Schirm leer/Cursor home]
4410  POKE 646,0         [Ink schwarz]
4420  RETURN
```

Die fehlenden Zeilennummern in Subroutine 4000 sollen Platz lassen für die dritte Aufgabe und haben keine Bedeutung.

Aufgabe 3

Verändern Sie die Schleifengröße:

```
120  FOR S = 1 TO 12
```

Verändern Sie die Arraygrößen (Einzelheiten in Kapitel 25):

```
4010  DIM M$ (12): DIM P (12)
```

Fügen Sie zusätzliche Titel und Farben ein:

```
4070  M$ (6) = "REALTIME-STEUERUNG VON RAUMFAEHREN
      MIT DEM VIC-20"
```

```
4Ø8Ø  M$ (7) = "TIPS ZUM UMGANG MIT DEM UNIX-BETRIEBS-
      SYSTEM FUER DIE JUNGE BRAUT"
4Ø9Ø  M$ (8) = "WIE BAUT MAN VLSI-SCHALTUNGEN AUS ALTEN
      STREICHHOELZERN?"
41ØØ  M$ (9) = "COMMODORE 256 – PROGRAMMIEREN NICHT SO
      LEICHT GEMACHT"
411Ø  M$ (1Ø) = "MASCHINENCODEPROGRAMME FUER NUME-
      RISCH GESTEUERTE TOASTER"
412Ø  M$ (11) = "EINE EINFUEHRUNG IN DAS NETZSYSTEM SPIEL
      UND SPASS"
413Ø  M$ (12) = "LOGO FUER SYSTEMANALYTIKER"
425Ø  P (6) = 3
426Ø  P (7) = 1Ø
427Ø  P (8) = 5
428Ø  P (9) = 14
429Ø  P (1Ø) = 15
43ØØ  P (11) = 9
431Ø  P (12) = 6
```

Beachten Sie übrigens, wie leicht es wäre, die Farben in dem nach obiger Art strukturierten Programm zu ändern. Man müßte lediglich die Zuteilungen der Variablen P (1)–P (12) abändern. Ebenso könnten die Buchtitel geändert werden, ohne daß am Hauptprogramm etwas verändert werden müßte.

15 Debugging II

Debugging I befaßte sich mit Fehlern in der "Grammatik". Eine Anweisung kann aber grammatikalisch völlig in Ordnung sein und in einem Programm trotzdem Unsinn hervorrufen.

Laufzeitfehler

Der 64 weist Sie auf Syntaxfehler zwar erst dann hin, wenn Sie RUN tippen, aber im Prinzip könnte er es schon vorher tun, weil er eine Anweisung nur durchzugehen braucht, um zu erkennen, daß etwas nicht in Ordnung ist. Es gibt aber andere Arten von Fehlern, die auf keinen Fall zu erkennen sind, bevor das Programm gefahren wird. Man nennt sie Laufzeitfehler.

Hier ein simples Beispiel:

```
10  FOR P = 1 TO 20
20  N = 5/(5 - P)
30  PRINT N
40  NEXT P
```

Fahren Sie das. Sie werden feststellen, daß es ganz brav losgeht und die Werte

```
1.25
1.66666667
2.5
5
```

liefert, dann aber mit einer Fehlermeldung aufwartet:

```
? DIVISION BY ZERO ERROR IN 20
```

Was ist da passiert? Die Meldung teilt uns mit, daß dem 64 an Zeile 20 etwas unangenehm aufgefallen ist, die da lautet:

```
10  N = 5/(5 - P)
```

An der Anweisung selbst kann es allerdings nicht liegen, weil sie schon viermal ausgeführt wurde und die obigen vier Zahlen hervorbrachte. Es muß also mit dem Wert von P zusammenhängen, das einzige, was sich verändert. Schreiben Sie "PRINT P" (oder, damit es rascher geht, auch nur "?P"). Der Bildschirm zeigt "5".

Der Computer versucht also auszurechnen:

$$\frac{5}{5-5} = \frac{5}{\emptyset}$$

und kann das nicht, weil das Ergebnis eine Zahl wäre, größer, als Sie sich vorstellen mögen, und so groß der Speicher des 64 auch ist, sie könnte er doch nicht aufnehmen. Der 64 nimmt also sehr vernünftig Notiz, sobald Sie irgendeine Zahl durch Null teilen wollen, und fängt erst gar nicht damit an, sondern teilt Ihnen lieber mit, daß es hier krankt.

Dieser Fehler kann in viel weniger auffälliger Weise auftreten. Wie wäre es damit?

```
3Ø  INPUT P, Q, R
4Ø  A = (P + Q - R)/(5 + (P - R) * (P - R) - 2 * Q)
```

Probieren Sie 7, 15 und 2 als Werte für P, Q und R und warten Sie ab!

Aufgabe 1

Welche Werte in den folgenden Beispielen würden die Meldung "Teilung durch Null" hervorrufen?

1. A = 7/(B - C)
2. R = P + Q/(2 * P - Q)
3. M = R + 2/(R * R + R * R * R)

Lösung

Aufgabe 1

Sie können davon kommen, wenn Sie alle Variablen in diesen Beispielen auf Null setzen, aber andere Möglichkeiten sind:

1. B und C gleich setzen.
2. Q doppelt so groß setzen wie P, etwa Q = 7, p = 3.5
3. R = −1 setzen.

Sie *können* (und sollten) stets das Auftauchen der Meldung verhindern, wenn Sie einen eigenen Test einfügen. Bei Beispiel 1 oben könnten Sie beispielsweise schreiben:

```
2Ø  INPUT B, C
3Ø  D = B - C
4Ø  IF D = Ø THEN PRINT "GEHT NICHT. NOCH EINMAL"; GOTO 2Ø
5Ø  A = 7/D
```

16 Strings

Man kann Computer dazu bewegen, nicht nur Zahlen, sondern auch Wörter und andere Arten symbolischer Schreibweise zu verarbeiten.

Der Postbote klopft an die Tür. Ein Brief für Sie. Ein sehr persönlicher. "Lieber Herr Wagenschmalz", heißt es da, "Sie sind unter den Bewohnern von Hinterdorfhagen ausgewählt worden und erhalten völlig kostenlos . . ."

Sehr erfreulich. Die alte Frau Schnaufberger nebenan hat aber den gleichen Brief bekommen, wie auch jeder andere Einwohner von Hinterdorfhagen nebst dem gesamten Sprengel.

Das geht so.

```
10  INPUT "WIE IST IHR NAME"; N$
20  INPUT "IN WELCHER STADT LEBEN SIE"; T$
30  PRINT CHR$ (147)
40  PRINT "LIEBER HERR □"; N$
50  PRINT "□ □ □ SIE PERSOENLICH SIND HIER IN □"; T$
60  PRINT "AUSGEWAEHLT WORDEN UND ERHALTEN"
70  PRINT "VON UNS GRATIS UND OHNE KOSTEN (*)"
80  PRINT "EINEN GROSSARTIGEN NASEN-"
90  PRINT "BEFESTIGUNGSRING, VERGOLDET."
100 PRINT "WIR SIND SICHER, HERR □"; N$
110 PRINT "DASS SIE AM SELBIGEN BALDIGST"
120 PRINT "HERUMGEFUEHRT WERDEN KOENNEN."
130 PRINT "□ □ □ MIT DEN FREUNDLICHSTEN GRUESSEN"
140 PRINT "□ □ □ □ OTTOBALD K. BEUTELSCHNEIDER"
150 PRINT "□ □ □ VERTRIEB FUER LUKRATIVE
    NEUIGKEITEN"
160 PRINT
170 PRINT "* VERSANDGEBUEHREN DM 892,—"
```

Fahren Sie das und suchen Sie sich Eingaben aus:

WAGENSCHMALZ

HINTERDORFHAGEN

ZECKERWETTEL

SCHLAFHAUSEN

und so weiter. Probieren Sie andere Namen und Orte aus. Na . . .

Nun stellen Sie sich vor, daß dieses Programm automatisch mit Namen und Adressen aus einer Datenbank gefüttert wird und in der Stunde Tausende von Briefen hinausgehen.

Abgesehen davon, wie albern das Ganze wirkt, ist das Interessante daran, daß keinerlei *Rechenarbeit* stattfindet. Es wird nur gespeichert und ein bißchen mit geschriebenem Text herumgespielt. Der Computer bewältigt das, weil er nicht nur Zahlen, sondern auch *Strings* speichern kann. Das ist es, was diese Dollarzeichen $ signalisieren, obschon in diesem Zusammenhang noch eine naheliegendere Verbindung zu bestehen scheint.

In Kapitel 8 habe ich Strings und Stringvariable schon vorgestellt. Jetzt will ich Ihnen zeigen, wie man damit umgeht.

Verkettung

. . . ein hochgestochenes Wort für "aneinanderfügen". Wenn Sie zwei Strings aneinanderkleben wollen, schreiben Sie ein +-Zeichen dazwischen. Beispiel:

```
PRINT "HOT" + "DOG"
```

was ergibt:

```
HOTDOG
```

Beachten Sie die Reihenfolge: "DOG" + "HOT" ergibt "DOGHOT". Beachten Sie außerdem, daß die Anführungsstriche *nicht* Bestandteil des Strings sind. Wenn ein String angezeigt oder auf andere Weise verarbeitet wird, sind die Anführungsstriche nur da, um Anfang und Ende zu bezeichnen.

Man kann mehrere Strings auf folgende Weise kombinieren:

```
1Ø  INPUT B$, C$
2Ø  PRINT B$ + B$ + C$
```

Was geschieht, wenn Sie als Eingaben:

B$ = "B" C$ = "C"

B$ = "KO" C$ = "SNUSS"

B$ = "DU, □" C$ = "NUR DU ALLEIN"

nehmen? Warum?

Kommt eine bestimmte Folge von Zeichen (und dazu gehören *auch* Grafikzeichen) in einem Programm mehrmals vor, dann empfinden Sie es vielleicht als lohnend, sie einer Stringvariablen zuzuteilen.

Die Länge eines Strings

Der Befehl

```
LEN
```

liefert die Länge eines Strings – das heißt, die Anzahl der darin enthaltenen Zeichen. Beispiele:

```
LEN ("FIDO") = 4
LEN ("£££££££££") = 11
LEN ("2 + 2 = 5") = 5
LEN ("") = Ø
```

wobei "", der *leere* String, ein String *ohne* Zeichen ist. Um allgemein die Länge des Strings K$ zu finden, schreiben Sie:

```
LEN (K$)
```

Probieren Sie, um das zu testen, folgendes Programm aus:

```
1Ø  INPUT "STRING"; K$
2Ø  PRINT K$; "□ HAT LAENGE"; LEN (K$)
3Ø  GOTO 1Ø
```

Erklärt sich eigentlich von selbst.

Wortumkehrung

Das nächste Programm nimmt als Eingabe Buchstaben für Buchstaben ein Wort an und nutzt die Stringverkettung, um dieses Wort hervorzubringen, aber verkehrt herum. (Mit fortgeschritteneren Befehlen wie MID$, LEFT$, RIGHT$ könnte man das ganze Wort auf einmal eingeben, aber der Einfachheit halber gehe ich hier Buchstaben für Buchstaben vor. Im nächsten Kapitel finden Sie weitere Einzelheiten.)

```
1Ø  INPUT "ERSTER BUCHSTABE"; F$
2Ø  INPUT "NAECHSTER BUCHSTABE"; N$
3Ø  IF N$ = "Ø" THEN 6Ø
```

```
40  F$ = N$ + F$
50  GOTO 20
60  PRINT CHR$ (147)
70  PRINT F$
```

Um der Eingabe in Zeile 20 ein Ende zu machen, geben Sie keinen Buchstaben ein, sondern "0".

Um zu sehen, wie das funktioniert, nehmen Sie das Wort "GARTEN". In Zeile 10 geben wir den ersten Buchstaben "G" ein, also wird F$ auf "G" gesetzt. In Zeile 20 geben wir den nächsten Buchstaben "A" ein. Zeile 40 verwandelt F$ nun in:

N$ + F$ = "A" + "G" = "AG"

und Zeile 50 schickt uns zurück zu 20, um einen weiteren Buchstaben zu holen, diesmal "R". F$ wird damit zu

N$ + F$ = "R" + "AG" = "RAG"

und so weiter:

Eingabe "T": F$ = N$ + F$ = "T" + "RAG" = "TRAG"

Eingabe "E": F$ = N$ + F$ = "E" + "TRAG" = "ETRAG"

Eingabe "N": F$ = N$ + F$ = "N" + "ETRAG" = "NETRAG"

Eingabe "0": Programm springt zu Zeile 60 und zeigt "NETRAG" an.

Das Entscheidende ist die Reihenfolge, in der die Strings in Zeile 40 zusammengefügt werden. Was geschieht, wenn Sie stattdessen

```
40  F$ = F$ + N$
```

schreiben?

Aufgabe 1

Es gibt ein Spiel mit Wörtern, bei dem der erste Spieler einen einfachen Satz bildet wie:

GESTERN SAH ICH EINEN PAVIAN

Der nächste Spieler hängt dem Pavian ein beschreibendes Eigenschaftswort an:

GESTERN SAH ICH EINEN ROSA PAVIAN

Der nächste steuert wieder ein Adjektiv bei:

GESTERN SAH ICH EINEN WILDEN ROSA PAVIAN

und so weiter. Der Satz wird immer länger (bis jemand nicht mehr weiß, wo das Wort hineingehört), und sieht am Ende vielleicht so aus:

GESTERN SAH ICH EINEN BEDENKENREICHEN FAULEN LEICHT-GLAEUBIGEN ERFRISCHENDEN RIESIGEN BESONDERS VER-STAERKTEN MISSGELAUNTEN KUNSTSTOFFGEFÜTTERTEN HER-VORRAGENDEN WILDEN ROSA PAVIAN

oder so in der Art.

Schreiben Sie ein Programm, mit dem die Spieler solche Sätze dadurch aufbauen können, daß sie bei jedem Schritt ein zusätzliches Adjektiv einfügen.

Lösung

Aufgabe 1

```
10  Y$ = "GESTERN SAH ICH EINEN □"
20  B$ = "PAVIAN"
30  A$ = ""
40  PRINT Y$ + B$ + A$
50  INPUT "ADJEKTIV?"; I$
60  A$ = I$ + "□" + A$
70  GOTO 40
```

17 Substrings

Wenn Sie Teile eines Strings als Substrings herausnehmen, können Sie Wörter manipulieren. Sie finden hier Beispiele für Schüttelreime und ein Programm, das Ihnen mitteilt, auf welchen Wochentag Ihr Geburtstag fiel.

Im vorigen Kapitel habe ich den Gedanken eines *Strings* von Zeichen vorgestellt. Jetzt komme ich auf die Befehle

LEFT$

RIGHT$

MID$

mit denen Sie Teile eines Strings auswählen können – sogenannte *Substrings*. Für den Umgang mit Strings allgemein sind das sehr nützliche Befehle.

Links, rechts und Mitte

Um die linke Seite eines Strings zu wählen, verwenden Sie den Befehl

LEFT$ (X$; N)

was die linken N Zeichen des Strings X$ liefert. Beispiel:

```
1Ø  LET X$ = "LANDSCHAFTSMALEREI"
2Ø  LET Y$ = LEFT$ (X$, 4)
3Ø  PRINT Y$
```

erbringt LAND. Ebenso gibt es einen Befehl für die rechten N Zeichen:

RIGHT$ (X$, N)

und

```
1Ø  LET X$ = "LANDSCHAFTSMALEREI"
2Ø  LET Y$ = RIGHT$ (X$, 2)
3Ø  PRINT Y$
```

ergibt EI. (Übrigens sind diese LET-Anfänge eine Sache der freien Wahl und können genau wie bei Zuteilungen für numerische Variablen weggelassen werden.)

Schließlich kommt in diesem Zusammenhang noch der Befehl:

```
MID$ (X$, M, N)
```

der die N Zeichen von X$ ab Position M liefert. Für M gibt es eine Einschränkung: Es muß größer sein als Ø. Läßt man N weg, wird alles ab M erfaßt. Im obigen Programm heißt es dann:

```
2Ø  LET Y$ = MID$ (X$, 5, 6)
```

und Sie haben SCHAFT.

Eine typische Verwendungsart dieser Methode ist die, den Wochentag mit einer Ziffer von 1–7, beginnend ab Sonntag, anzuzeigen:

```
1Ø  W$ = "SOMODIMIDOFRSA"
2Ø  INPUT "WELCHER TAG"; T
3Ø  Y$ = MID$ (W$,2 * T – 1,2)
4Ø  PRINT Y$
```

Das 2 * T – 1 liefert die richtigen Startpositionen 1, 3, 5, 7, 9, 11 und 13 in W$.

Aufgabe 1

Verwenden Sie einen String "JANFEBMAR . . . DEZ", um ein ähnliches Programm zu schreiben, das mit einer Zahl von 1–12 den Monat anzeigt.

Strings und Zahlen

Der String "493" und die Zahl 493 werden vom Computer verschieden aufgefaßt. Das fällt Ihnen vielleicht nicht auf, wenn Sie sie nur anzeigen.

```
PRINT 493
PRINT "493"
```

liefern dasselbe Ergebnis. Nun probieren Sie:

```
PRINT 493 + 7
PRINT "493" + 7
PRINT "493" + "7"
```

Sie werden feststellen, daß Sie drei verschiedene Ergebnisse erhalten:

```
5ØØ
? TYPE MISMATCH ERROR
4937
```

Im ersten Fall werden die Zahlen einfach addiert.

Im zweiten wird versucht, einen String zu einer Zahl zu addieren, was einfach nicht geht. Einen String statt einer Zahl zu verwenden und umgekehrt, ruft jedesmal diese Fehlermeldung hervor.

Im dritten Fall werden die Strings "493" und "7" verkettet, zu "4937" zusammengesetzt und ohne Anführung angezeigt.

Das kann sehr nützlich sein, weil man mit Strings Dinge tun kann, die bei Zahlen nicht so leicht fallen. Um beispielsweise die erste Ziffer von 987654321 zu finden, brauchen Sie nur LEFT$ ("987654321",1). Mit arithmetischen Methoden geht das viel schwerer. Das Ergebnis wäre jedoch ein *String* "9" und nicht die Zahl 9. Vielleicht wollen Sie mit dieser 9 jetzt ein bißchen Arithmetik treiben. Und zwar wie?

Ein String, der die Form einer Zahl hat (in Anführungsstrichen) kann in eine echte Zahl verwandelt werden durch das Befehlswort

VAL

(für "value" = Wert). Demnach ist

VAL ("9")

die *Zahl* 9. Probieren Sie

PRINT VAL ("493") + VAL ("7")

dann können Sie sehen, daß es wirklich funktioniert.

Es gibt einen ähnlichen Befehl

STR$

der umgekehrt wirkt. Er verwandelt eine Zahl in einen String. Beispiel:

STR$ (7751) ist "7751"

Beispiele dazu finden Sie in Kapitel 34, Lösungen, Aufgabe 1.

Schüttelreime

Der Schüttelreim lebt davon, daß die Anfangskonsonanten eines Reimpaars vertauscht werden und dadurch einen überraschenden neuen Sinn ergeben.

Durch Stringmanipulation kann man das mit dem Computer bewältigen. Ob die Ergebnisse komisch sind oder nicht, ist erstens eine Geschmacksfrage, und zweitens hängt es vom Anwender ab. Das Programm hat dazu keine Meinung.

Verfassen wir ein Programm dafür.

```
10  INPUT "ERSTES WORT"; A$
20  INPUT "ZWEITES WORT"; B$
30  P$ = LEFT$ (A$,1)
```

```
40  Q$ = MID$ (A$,2)
50  R$ = LEFT$ (B$,1)
60  S$ = MID$ (B$,2)
70  PRINT A$ + "□" + B$
80  PRINT "WIRD ALS SCHUETTELREIM"
90  PRINT R$ + Q$ + "□" + P$ + S$
100 GOTO 10
```

Fahren Sie das und geben Sie in den Zeilen 10 und 20 "GELBER" und "SEHEN" ein.

Zeile 30 nimmt den ersten Buchstaben links von "GELBER", also P$ = "E". Zeile 40 nimmt vom Zeichen 2 an alles mit und macht Q$ zu "ELBER". Zeile 50 hängt R$ = "G" an, Zeile 60 liefert "EHEN".

Zeile 70 und 80 setzen das ursprüngliche Wort wieder zusammen, und Zeile 90 zeigt an:

"S" + "ELBER" + "□" + "G" + "EHEN" = SELBER GEHEN

was eigentlich gar nicht so schlecht ist, wenn man es genau bedenkt.

Tagfinder

Dieses Programm nimmt als Eingabe ein Datum an (Tag Nummer T, Monat M, Jahr J) und berechnet, welcher Wochentag es ist.

```
10  A$ = "033614625035"
20  B$ = "SOMODIMIDOFRSA"
30  INPUT "TAG"; T
40  INPUT "MONAT"; M
50  INPUT "JAHR"; J
60  PRINT "DER TAG IST □";
70  Z = J - 1
80  C = INT (Z/4) - INT (Z/100) + INT (Z/400)
90  X = J + T + C + VAL (MID$ (A$, M, 1)) - 1
100 IF M > 2 AND (J = 4 * INT (J/4)) AND
    J<> 100 * INT (J/100) OR J = 400 * INT (J/400)
    THEN X = X + 1
110 X = X - 7 * INT (X/7)
120 PRINT MID$ (B$,2 * X + 1,2)
```

Schreiben Sie das sehr genau ab, vor allem die Klammer in Zeile 1ØØ. Fahren Sie es und geben Sie (versuchsweise) 24 für T, 9 für M und 1945 für N ein. (Also den 24. September 1945). Sie *müssen* das vollständige Jahr eingeben, nicht bloß 45, sonst liefert das Programm die falsche Antwort. Sie sollten erhalten

DER TAG IST MO

für Montag. Probieren Sie es mit dem heutigen Datum. Mit Ihrem Geburtstag. Stellen Sie fest, an welchem Datum Goethe gestorben ist, und versuchen Sie es damit.

Zeile 1Ø speichert die "monatlichen Korrekturzahlen" in dichter Form als Einzelstring.

Zeile 2Ø baut den oben beschriebenen "Wochentags"-String auf.

Zeilen 3Ø–6Ø sind Eingabe/Ausgabe-Anweisungen, nichts Ungewöhnliches.

Zeilen 8Ø–1ØØ führen eine komplizierte Berechnung aus, die Schaltjahre und die "monatlichen Korrekturen" berücksichtigt. Beachten Sie die Verwendung von VAL und MID$ in Zeile 9Ø; hier wird die Ziffer M in A$ gefunden und in eine Zahl verwandelt.

Zeile 11Ø liefert eine Zahl im Bereich Ø–6 für den Wochentag (statt der oben zur Erläuterung verwendeten 1–7) und Zeile 12Ø zeigt den Tag an, wobei es MID$ bei B$ verwendet.

Die Uhr

Der 64 hat eine eingebaute Uhr – ja, sogar mehrere! Mit BASIC können Sie zwei Standardvariable nutzen:

TI

TI$

Die erste ist eine numerische Variable, und ihr Wert beträgt 6Ømal die Zahl der Sekunden, die abgelaufen sind, seitdem der Computer eingeschaltet wurde. Die zweite ist eine Stringvariable; sie gibt die Zeit in Stunden, Minuten und Sekunden an. Beispiel:

Ø21143 = 2 Stunden, 11 Minuten, 43 Sekunden

15Ø422 = 15 Stunden, 4 Minuten, 22 Sekunden

TI$ kann jederzeit auf Null zurückgesetzt werden mit

TI$ = "ØØØØØØ"

dann zählt die Uhr die von diesem Augenblick an abgelaufene Zeit. TI kann auf diese Weise nicht zurückgesetzt werden. Ich erwähne die beiden hier, weil TI$ eine Stringvariable ist und mit ein bißchen Stringmanipulation zu mehr Entgegenkommen veranlaßt werden kann.

```
10  PRINT CHR$ (147)
20  A$ = TI$
30  PRINT LEFT$ (A$,2); "□ STUNDEN □"; MID$ (A$, 3, 2);
    "□ MINUTEN □"; RIGHT$ (A$,2); "□ SEKUNDEN"
40  GOTO 20
```

Jetzt haben Sie eine Uhr auf dem Bildschirm. Sie rollt ab wie verrückt, also möchten Sie vielleicht einfügen:

```
25  PRINT CHR$ (19);
```

Beachten Sie, daß sie nicht regelmäßig im Sekundentakt läuft; das liegt am zeitlichen Ablauf des Zyklus. Ein paar Werte werden weggelassen, weil die Uhr nicht im richtigen Augenblick gelesen wird. Aber die Zeit stimmt trotzdem.

Eine Ergänzung, die sich bei jedem Programm empfiehlt, ist eine Routine, mit der die abgelaufene Zeit angezeigt wird. Schreiben Sie gleich zu Anfang:

```
1  TI$ = "000000"
```

Dann Ihr Programm. Aber statt STOP geben Sie (sagen wir) GOTO 10000 ein und fügen an:

```
10000  A$ = TI$
10010  GET B$: IF B$ = "" THEN 10010 [Pause für das Drücken einer
       Taste]
10020  PRINT "ABGELAUFENE GESAMTZEIT BETRAEGT"
10030  PRINT LEFT$ (A$, 2); "□ STUNDEN □"; MID$ (A$, 3, 2);
       "□ MINUTEN □"; RIGHT$ (A$, 2); "□ SEKUNDEN"
10040  STOP
```

Lösung

Aufgabe 1

```
10  M$ = "JANFEBMARAPRMAIJUNJULAUGSEPOKTNOVDEZ"
20  INPUT "WELCHER MONAT"; M
30  J$ = MID$ (M$, 3 * M − 2,3)
40  PRINT J$
```

18 ASCII-Codes

Jedes Zeichen hat seine eigene Codenummer. Damit können Sie prüfen, was für eine Art Zeichen es ist, oder von einem Code in den anderen verwandeln. Ein Anwendungsgebiet wäre das Morsen.

ASCII heißt "American Standard Code for Information Interchange" (amerikanischer Standardcode für Informationsaustausch) und entspricht seinem Namen ziemlich genau. Als eingebürgertes System für die Codierung von Zeichen als Zahlen ist es schon lange vorhanden.

Wenn Sie den Code eines Zeichens K$ finden wollen, verlangen Sie:

```
ASC (K$)
```

Hier ein Testprogramm:

```
10 INPUT K$
20 PRINT ASC (K$)
30 GOTO 10
```

Die ASCII-Codes sind aufgeführt im Handbuch auf den Seiten 135–37. Die Zeichen 0–31 sind *Steuerzeichen* und werden vom Betriebssystem verwendet. Die Zeichen 96–127 sind Grafikzeichen.

Um festzustellen, welches Zeichen einem gegebenen Code C entspricht, verwendet man einen Befehl, den wir schon oft gesehen haben:

```
CHR$ (C)
```

Hier ein Analog-Testprogramm:

```
10 INPUT C
20 PRINT CHR$ (C)
30 GOTO 10
```

Probieren Sie verschiedene Zahlen zwischen 0 und 255 aus. Von 0 bis 31 erhalten Sie seltsame Ergebnisse, weil das System sie nicht anzeigt, sondern ihnen *gehorcht*!

Eine der üblichen Anwendungen der ASCII-Codes ist die Prüfung, ob ein Zeichen zu einer bestimmten Art gehört. Nehmen wir an, das Zeichen sei K$. So ergibt sich aus

```
IF ASC (K$) > 47 AND ASC (K$) < 58 . . .
```

daß K$ eine Einzelziffer ist. Ebenso:

```
IF ASC (K$) > 64 AND ASC (K$) < 91 ...
```

dann ist K$ ein Großbuchstabe. Und

```
IF ASC (K$) > 95 AND ASC (K$) < 128 ...
```

dann ist es ein Grafikzeichen. Und so weiter.

Aufgabe 1

Schreiben Sie ein Programm, das nach Eingabe eines Strings die einzelnen ASCII-Codes ausgibt.

Morsecode-Generator

Das folgende Programm nimmt eine Eingabe an und zeigt sie in Morsecode an:

```
10  DIM A$ (26)
20  A$ (1) = ".-"
30  A$ (2) = "-..."
40  A$ (3) = "-.-."
50  A$ (4) = "-.."
60  A$ (5) = "."
70  A$ (6) = "..-."
80  A$ (7) = "--."
90  A$ (8) = "...."
100 A$ (9) = ".."
110 A$ (10) = ".---"
120 A$ (11) = "-.-"
130 A$ (12) = ".-.."
140 A$ (13) = "--"
150 A$ (14) = "-."
160 A$ (15) = "---"
170 A$ (16) = ".--."
180 A$ (17) = "--.-"
```

```
19Ø  A$ (18) = ". – ."
2ØØ  A$ (19) = ". . ."
21Ø  A$ (2Ø) = "–"
22Ø  A$ (21) = ". . –"
23Ø  A$ (22) = ". . . –"
24Ø  A$ (23) = ". – –"
25Ø  A$ (24) = "– . . –"
26Ø  A$ (25) = "– . – –"
27Ø  A$ (26) = "– – . ."
```

Das setzte nur die Codes für die Buchstaben A–Z hintereinander als Stringarray; vielleicht fallen Ihnen weniger langweilige Methoden dafür ein. Nun zum Programm selbst:

```
3ØØ  INPUT "MITTEILUNG EINGEBEN"; M$
31Ø  PRINT CHR$ (147)
32Ø  FOR I = 1 TO LEN (M$)
33Ø  C = ASC (MID$ (M$, I, 1))
34Ø  IF C < 32 OR C > 32 AND C < 65 OR C > 9Ø THEN 38Ø
35Ø  C = C – 64
36Ø  IF C < Ø THEN FOR J = 1 TO 2ØØ: NEXT J: PRINT
37Ø  IF C > Ø THEN PRINT CHR$ (C + 64), A$ (C)
38Ø  NEXT I
39Ø  STOP
```

Zeile 3ØØ erhält die Mitteilung zum Verschlüsseln. Zeilen 32Ø und 38Ø setzen eine Schleife, um die Mitteilung Zeichen für Zeichen abzusuchen; Zeile 33Ø findet das Zeichen I. Zeile 34Ø achtet darauf, daß es entweder ein Leerraum oder ein Buchstabe ist und beachtet es im anderen Fall nicht. Zeile 35Ø zieht von seinem ASCII-Code 64 ab, so daß A 1 wird, B 2, und so weiter. (Achtung: Leerraum wird jetzt zu 32–64, was negativ ist; siehe unten).

Zeile 36Ø fügt eine Verzögerung ein, wenn das Zeichen eine Leerstelle ist, um Wörter zu trennen, und zeigt eine Leere Zeile an.

Zeile 37Ø zeigt den Buchstaben der Mitteilung und seine Form in Morsezeichen an (entnommen aus dem Array A$).

Im Augenblick haben wir etwas leicht Ungewohntes: *stummes* Morsen. Verwendet man den SID-Klangchip im 64, kann man auch die zugehörigen Geräusche erzeugen; dazu Aufgabe 2 in Kapitel 3Ø.

Lösung

Aufgabe 1

```
1Ø  INPUT "STRING"; S$
2Ø  FOR I = 1 TO LEN (S$)
3Ø  K$ = MID $ (S$, I, 1)
4Ø  PRINT ASC (K$)
5Ø  NEXT I
```

19 Bildschirm- und Farbspeicher

Die Information, die erforderlich ist, um das TV-Bildschirmdisplay hervorzubringen, ist in zwei Speicherbereichen enthalten. Diese sind dem Programmierer direkt zugänglich und können genutzt werden, um das Display zu steuern.

In Kapitel 7 habe ich erwähnt, daß man sich das TV-Schirmdisplay als aus 25 Reihen zu je 40 Zeichen bestehend vorstellen kann, und daß die Reihen üblicherweise von 0–24 und die Spalten von 0–39 numeriert werden. Abbildung 19.1 zeigt diese Anordnung. Beachten Sie, daß sie genau 1000 (= 25 x 40) Zeichen umfaßt. Jedes Zeichen kann durch seinen Code als ein Einzelbyte Information bestimmt werden. Aus diesem Grund verwundert es nicht, einen Speicherbereich von 1000 Adressen Größe vorzufinden, der diese gesamte Information enthält.

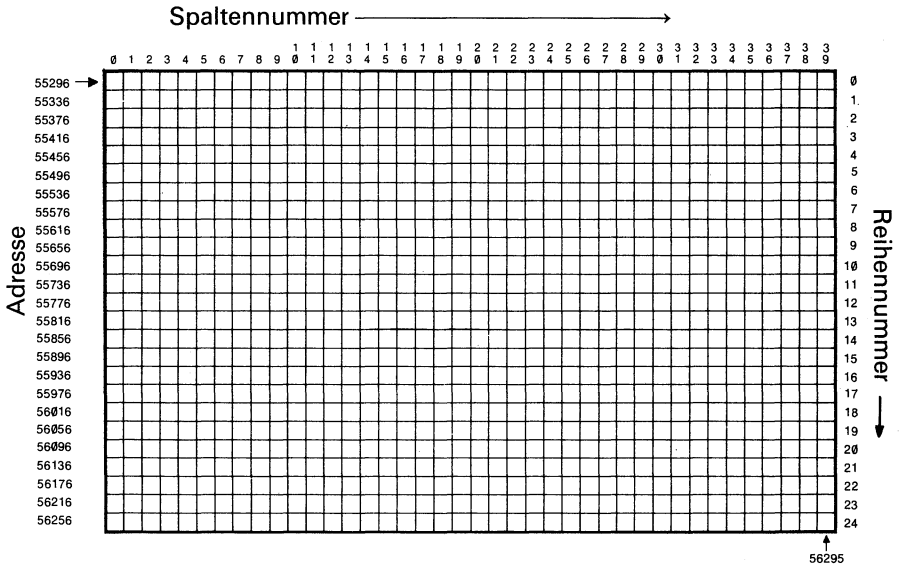


Abbildung 19.1
Aufbau des Bildschirmspeichers ahmt den des Bildschirms selbst nach.

Bildschirmspeicher

Der betreffende Speicherbereich beginnt bei Adresse 1Ø24 und endet bei 2Ø23. Er wird *Bildschirmspeicher* oder *Video-RAM* genannt. Nun hat ein Computerspeicher von Natur aus keine rechteckige Form, wie Abbildung 19.1 sie zeigt; praktisch ist alles in einer einzigen langen Reihe von Adressen angelegt. In diesem Fall verlaufen die Adressen den Reihen entlang und bewegen sich erst dann eine Spalte tiefer, wenn eine Reihe aufhört – so, als läse man in einem Buch. Anders ausgedrückt: Die oberste Reihe des Bildschirms ist gespeichert in den Adressen

1Ø24 1Ø25 1Ø26 1Ø27 ... ...

1Ø62 1Ø63

Die nächste Reihe (Reihe 1) in:

1Ø64 1Ø65 1Ø66 1Ø67 ... ...

11Ø2 11Ø3

und so weiter, mit dem Ende in Reihe 24 und:

1984 1985 1986 1987 ... ...

2Ø22 2Ø23

Die allgemeine Regel: Reihe R, Spalte C entspricht der Adresse:

1Ø24 + 4Ø * R + C

Wollen wir also auf dem Bildschirm in Reihe R, Spalte C ein bestimmtes Zeichen anzeigen, brauchen wir nur mit einem geeigneten Code über POKE in diese Adresse hineinzugehen:

POKE 1Ø24 + 4Ø * R + C, Code für Zeichen

Es gibt zwei kleine Haken.

1. Der erforderliche Code ist *nicht* der ASCII-Code, sondern der in Anhang E des Handbuchs, S. 132, aufgeführte. Grob gesprochen, handelt es sich um den ASCII-Code minus 64 für Buchstaben (und ein paar andere), minus 32 für Grafikzeichen, und unangetastet für Zahlen.
2. Es gibt konkret zwei Zeichensätze, die Sie anzeigen können. Satz 1 hat Großschrift, Satz 2 Kleinschrift. Um sie zu wählen, müssen Sie mit POKE in die Systemvariable 53272 heineingehen:

   Satz 1: POKE 53272,2Ø

   Satz 2: POKE 53272,22

Wenn Sie nicht versucht haben, den verwendeten Satz zu ändern, wird es Satz 1 mit Großbuchstaben sein. Ich schlage vor, daß Sie zunächst bei Satz 1 bleiben, bis Sie die Grundbegriffe verstanden haben.

POKE zum Bildschirm

Angenommen, wir möchten in Reihe 12, Spalte 2Ø, nah beim Mittelpunkt des Bildschirms, einen Ball anzeigen. Die Adresse lautet:

1Ø24 + 4Ø * 12 + 2Ø = 1524

und der Code für den Ball (Anhang E des Handbuchs):

81

Sie brauchen also die Anweisung

POKE 1524,81

Versuchen Sie das als direkten Befehl. Drücken Sie vorher RUN/STOP + RESTORE.

Hat sich irgend etwas getan?

Nein. Das ist ein bißchen merkwürdig, weil entsprechend dem Handbuch, S. 64 oben, ein weißer Ball auf dem Bildschirm erscheinen sollte. Dem Handbuch kann man sonst durchaus vertrauen, aber diesmal ist es ausgerutscht. Auf dem Schirm ist sogar *wirklich* ein Ball, aber Sie können ihn nicht sehen, weil er nicht weiß ist, sondern blau! Verändern Sie die PAPER-FARBE durch

POKE 53281,7

Der Bildschirm wird gelb, und da ist der Ball!

Jetzt können Sie experimentieren. Probieren Sie

POKE 1525,81

und Sie sehen einen zweiten (blauen) Ball.

Aufgabe 1

Welche POKE-Anweisungen benötigt man, um anzuzeigen:

1. Das Zeichen M in Reihe 7, Spalte 9?
2. Das Zeichen für Kreuz (Treff) aus einem Kartenspiel in Reihe 2Ø, Spalte 32?
3. Das Zeichen (pi) in Reihe 11, Spalte 8?

(Gehen Sie davon aus, daß Sie im Zeichensatz 1 sind.)

'PRINT AT'-Subroutine

Ich möchte ein bestimmtes Zeichen an einer bestimmten Position auf dem Bildschirm öfter anzeigen. Manche Versionen von BASIC haben einen PRINT AT-Befehl, der das mühelos bewältigt, wie

```
PRINT AT 1Ø, 15, "Z"          (*)
```

Die Version von BASIC im 64 dagegen (manchmal, falls ich das aussprechen darf, ein bißchen dürftig) läßt das nicht zu. Wie behilft man sich? Man schreibt eine Standard-Subroutine, die das leistet. Ich setze sie in Zeile 1ØØØØ, vor allem deshalb, weil das eine viel höhere Zahl ist, als Sie in der Regel verwenden, so daß nichts anderes gestört wird, was Sie schreiben. Genauso mache ich es bei allen anderen nützlichen Unterprogrammen – natürlich mit jeweils anderen Zeilennummern. Auf diese Weise baue ich eine *Bibliothek* von nützlichen Subroutinen auf. Das erweitert die Fähigkeiten des 64 ganz erheblich und erspart viel Routinearbeit.

Es ist ein guter Grundsatz, Subroutinen möglichst *allgemein* zu halten. So wäre es beispielsweise sinnlos, eine Subroutine zu verfassen, die in einer gewünschten Reihe vonSpalte 'X' anzeigt, wenn ein bißchen zusätzliche Überlegung Sie jedes gewünschte Zeichen anzeigen läßt. Diese Routine erfordert drei Datenposten:

REIHE = die Reihennummer

CLM = die Spaltennummer

CDE = den Zeichencode

Hier ist sie:

```
1ØØØØ  REM PRINT AT REIHE, CLM, CDE
1ØØ1Ø  POKE 1Ø24 + 4Ø * REIHE + CLM, CDE
1ØØ2Ø  RETURN
```

Ich habe ausgefallene Variablennamen gewählt, um mögliche Überschneidungen mit einem anderen Programm zu verhindern.

Sie können das genauso verwenden wie ein PRINT AT. Beispiel: Da der Code für 'Z' 26 ist, können Sie die Wirkung von (*) oben erzielen durch Verwendung von:

```
REIHE = 1Ø: CLM = 15: CDE = 26: GOSUB 1ØØØØ
```

Was zu schreiben nicht viel länger dauert.

Beachten Sie das Format: *Teilen* Sie die in der Subroutine verwendeten Variablen *zu* und *rufen* Sie mit GOSUB *auf.* Variable, die in einer Subroutine verwendet werden, bezeichnet man oft als *zugeteilte Parameter.* Sie müssen sie immer auf die richtigen Werte setzen, *bevor* Sie die Subroutine aufrufen. Oft berechnet eine Subroutine die Werte anderer Variablen; das sind die von der Subroutine *gelieferten* Parameter.

Beispiel: Der Umwandlungsroutine binär/dezimal in Zeile 5ØØ des Programms auf Seite 71 wird der Parameter C zugeteilt, sie liefert K$.

Achten Sie darauf, daß die Namen, die Sie für solche Variablen verwenden, nicht irgendwo in Ihrem Programm mit anderer Bedeutung verwendet werden; das kann Schwierigkeiten geben!

Das schnelle

Ich präsentiere jetzt einen Computerklassiker, die Grundlage vieler früher Spielhallen-Videogeräte, und eine ausgezeichnete Einführung für bewegte Grafik und die Verwendung von Verzweigungsbefehlen.

Geben Sie zuerst ab Zeile 1ØØØØ die obige PRINT AT-Subroutine ein. Dann fügen Sie an:

```
1Ø  PRINT CHR$ (147): POKE 53281,7
2Ø  C = 2: R = 3
3Ø  H = 1: V = 1
4Ø  REIHE = R: CLM = C: CDE = 35: GOSUB 1ØØØØ
5Ø  C = C + H: R = R + V
6Ø  IF C = Ø OR C = 39 THEN H = –H
7Ø  IF R = Ø OR R = 24 THEN V = –V
8Ø  GOTO 4Ø
```

Wenn Sie das fahren, zieht mit großer Geschwindigkeit eine Folge von '#'-Zeichen über den Bildschirm und prallt an den Rändern ab. Der Pralleffekt wird erzielt durch die Zeilen 6Ø und 7Ø. Die Grundlage: C, R sind die laufende Position des bewegten #, H, V die Veränderungen, die für die nächste Position in C und R gemacht werden müssen (H für horizontal, V für vertikal). Anfangs sind H und V 1, so daß das # abwärts und nach rechts läuft, aber jedesmal, wenn es an einen Rand stößt, kehren H oder V die Richtung um. Beachten Sie die Verwendung der Subroutine in Zeile 4Ø.

Die Bewegungsillusion ist ziemlich stark, wird aber verdorben durch den nachgezogenen #-Schwanz. Um der Illusion aufzuhelfen, können wir jedes # *löschen,* sobald das nächste geplottet ist. Zuerst müssen wir uns an seine Position *erinnern:*

```
45  CØ = C: RØ = R
```

Wenn C und R sich in Zeile 5Ø dann ändern, tun CØ und RØ es nicht. Nachdem wir in Zeile 4Ø das neue # geplottet haben, löschen wir das alte (durch Anzeige eines Leerraums, Code 32). Eine gute Stelle (das Löschen so lange wie möglich verzögern, damit das Bild nicht so flackert) ist in Zeile 75:

```
75  REIHE = RØ: CLM = CØ: CDE = 32: GOSUB 1ØØØØ
```

Und los geht es!

Statt eines einzelnen # wollen wir einen ganzen Wurm haben, der aus vier # hintereinander besteht. Das können Sie erreichen, wenn Sie eine Art 'Verzögerungszeile' einfügen, wo die PRINT-Position für die Löschung schrittweise in der Zeile weitergereicht und endlich danach gehandelt wird. Löschen Sie die obige Zeile 45 und ersetzen Sie sie durch:

```
42  CØ = C1: RØ = R1
44  C1 = C2: R1 = R2
46  C2 = C3: R2 = R3
48  C3 = C: R3 = R
```

Wenn Sie dahinterkommen, wie die Verzögerungen wirken, werden Sie überhaupt keine Schwierigkeiten haben mit

Aufgabe 2

Ersetzen Sie die Zeilen 42–48 durch sieben Zeilen, die einen 7gliedrigen #-Wurm erzeugen.

Farbspeicher

Kehren wir jetzt zurück zu dem Ausrutscher im Handbuch. Passiert war Folgendes: Das Zeichen wurde zwar angezeigt – aber in der falschen Farbe, die zufällig auch die PAPER-Farbe war! Das ist ein bedauerliches Merkmal (ein Merkmal ist ein Fehler, der sich nicht beseitigen läßt), aber man kann es umgehen, wenn wir die Farben selbst steuern. Oben habe ich geschwindelt, als ich zu Beginn die PAPER-Farbe veränderte, was ohnehin keine schlechte Idee ist. Sie können die Farbe von Zeichen aber dadurch steuern, daß Sie mit POKE geeignete Codes in einen anderen Speicherbereich eingeben, den *Farbspeicher* oder *Farb-RAM.* Der Speicherbereich beginnt jetzt bei Adresse 55296 und endet bei 56295 und entspricht dem Bildschirm genauso wie der Bildschirmspeicher: Reihe Ø ist enthalten in den Adressen:

```
55296   55297   55298   55299   ...   ...   ...
55334   55335
```

und so weiter, siehe Abbildung 19.2. Der Farbcode für Reihe R, Spalte C ist also enthalten in Adresse:

```
55296 + 4Ø * R + C
```

Leeren Sie den Schirm, geben Sie wie zuvor ein:

```
POKE 1524,81
```

und warten Sie, während nichts geschieht. Dann schreiben Sie:

```
POKE 55796,5
```

und bekommen einen grünen Ball zu sehen. Die Farbcodes entsprechen denen in Kapitel 13, also bedeutet 5 'grün'. Und die Farbadresse, die Reihe 12, Spalte 2Ø entspricht, ist:

```
55296 + 4Ø * 12 + 2Ø = 55796
```

also das, was wir verwendet haben.

Moral: Wenn man in den Bildschirmspeicher mit POKE ein Zeichen eingibt, dann auch *gleichzeitig* seine Farbe in den Farbspeicher.

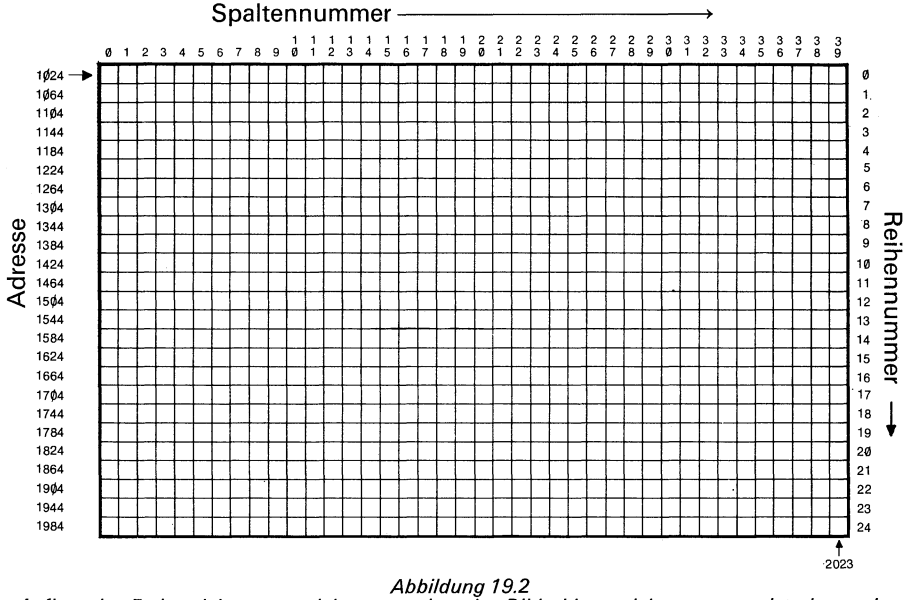


Abbildung 19.2
Aufbau des Farbspeichers entspricht genau dem des Bildschirmspeichers, verwendet aber andere Adressen.

Aufgabe 3

Wie Aufgabe 1, aber die Farben nun wie folgt:

1. Rot
2. Hellgrün
3. Weiß

Sie können auch die PRINT AT-Subroutine verändern, um die Farbe zu setzen: Verwenden Sie einfach eine Variable CR für den Farbcode und fügen Sie die Zeile ein:

```
1ØØ15 POKE 55296 + 4Ø * REIHE + CLM, CR
```

Wenn Sie vergessen, den Wert von CR zu setzen, erhalten Sie als *Vorgabe*wert Ø, also schwarz.

Hier ein Beispiel für eine verbesserte PRINT AT-Routine in Gebrauch:

```
1Ø  PRINT CHR$ (147)
2Ø  FOR T = Ø TO 15
3Ø  REIHE = T + 2: CLM = T + 2: CDE = T + 1: CR = T: GOSUB 1ØØØØ
4Ø  NEXT T
5Ø  GOTO 5Ø
```

Fügen Sie die Zeilen 1ØØØØ, 1ØØ1Ø, 1ØØ15 und 1ØØ2Ø aneinander und fahren Sie. Geben Sie STOP, um das Programm anzuhalten; Zeile 5Ø verhindert, daß eine Fehlermeldung das Display verdirbt.

Dynamische Wortumkehrung

Als ein längeres Beispiel hier ein Programm, das ein Wort von bis zu 21 Zeichen annimmt und es durch Bewegung der Buchstaben auf dem Bildschirm umkehrt:

```
1ØØ  DIM X$ (21)
11Ø  PRINT CHR$ (147)
12Ø  PRINT "SCHREIB DAS UMZUKEHRENDE WORT"
13Ø  PRINT
14Ø  INPUT W$
15Ø  WL = LEN (W$)
16Ø  IF WL = 2 * INT (WL/2) THEN WL = WL + 1: W$ = W$ + "□"
17Ø  WH = (WL–1)/2
18Ø  FOR T = 1 TO WL: X$ (T) = MID$ (W$, T, 1): NEXT T
2ØØ  REM WORT ANZEIGEN
21Ø  PRINT CHR$ (147)
22Ø  R = 12
23Ø  FOR T = 1 TO WL
24Ø  C = 19 – WH + T: C$ = X$ (T): GOSUB 2ØØØ
25Ø  NEXT T
3ØØ  REM ERSTER TEIL ROTATION
31Ø  FOR S = 1 TO WH
```

```
32Ø  FOR T = S TO WH
33Ø  C = 2Ø + T: R = S + 11: C$ = "□": GOSUB 2ØØØ
34Ø  R = R + 1: C$ = X$ (C + WH – 19): GOSUB 2ØØØ
35Ø  C = 2Ø – T: R = 13 – S: C$ = "□": GOSUB 2ØØØ
36Ø  R = R – 1: C$ = X$ (WH + C – 19): GOSUB 2ØØØ
37Ø  NEXT T: NEXT S
4ØØ  REM ZWEITER TEIL ROTATION
41Ø  FOR S = 1 TO WH
42Ø  FOR T = 1 TO S
43Ø  C = 19 – WH + S: R = 11 – WH + T: C$ = "□": GOSUB 2ØØØ
44Ø  C = C + 1: C$ = X$ (T): GOSUB 2ØØØ
45Ø  C = 21 + WH – S: R = 13 + WH – T: C$ = "□": GOSUB 2ØØØ
46Ø  C = C – 1: C$ = X$ (WL + 1 – T): GOSUB 2ØØØ
47Ø  NEXT T: NEXT S
5ØØ  REM DRITTER TEIL ROTATION
51Ø  FOR S = 1 TO WH
52Ø  FOR T = 1 TO WH + 1 – S
53Ø  R = 11 – WH + T: C = 19 + S: C$ = "□"; GOSUB 2ØØØ
54Ø  C = C + 1: C$ = X$ (T): GOSUB 2ØØØ
55Ø  C = 21 – S: R = 13 + WH – T: C$ = "□": GOSUB 2ØØØ
56Ø  C = C – 1: C$ = X$ (WL + 1 – T): GOSUB 2ØØØ
57Ø  NEXT T: NEXT S
6ØØ  REM VIERTER TEIL ROTATION
61Ø  FOR S = 1 TO WH
62Ø  FOR T = 1 TO S
63Ø  R = 13 + WH – S: C = 19 – WH + T: C$ = "□": GOSUB 2ØØØ
64Ø  R = R – 1: C$ = X$ (WL + 1 – T): GOSUB 2ØØØ
65Ø  R = 11 – WH + S: C = 21 + WH – T: C$ = "□": GOSUB 2ØØØ
66Ø  R = R + 1: C$ = X$ (T): GOSUB 2ØØØ
67Ø  NEXT T: NEXT S
68Ø  STOP
```

```
2ØØØ  REM PRINT AT R, C, C$
2Ø1Ø  CD = ASC (C$)
2Ø2Ø  IF CD > 64 THEN CD = CD – 64
2Ø3Ø  POKE 1Ø24 + 4Ø * R + C, CD
2Ø4Ø  POKE 55296 + 4Ø * R + C, 7
2Ø5Ø  RETURN
```

Ich werde nicht erklären, wie das im einzelnen funktioniert, weil das umständliche Berechnungen darüber betrifft, wo jedes einzelne Zeichen hinkommt. Beachten Sie aber die wiederholte Verwendung der Subroutine. Das ist eine Abwandlung unseres gewöhnlichen PRINT AT-Unterprogramms; das Zeichen wird direkt angenommen, der Code umgewandelt. Mehr über ASC in Kapitel 18.

Lösungen

Aufgabe 1

1. POKE 1Ø24 + 4Ø * 7 + 9, 13 oder POKE 1313, 13
2. POKE 1Ø24 + 4Ø * 2Ø + 32, 88 oder POKE 1696, 88
3. POKE 1Ø24 + 4Ø * 11 + 8, 94 oder POKE 1472, 94

Aufgabe 2

Löschen Sie die Zeilen 42–48 und fügen Sie ein:

```
41  CØ = C1: RØ = R1
42  C1 = C2: R1 = R2
43  C2 = C3: R2 = R3
44  C3 = C4: R3 = R4
45  C4 = C5: R4 = R5
46  C5 = C6: R5 = R6
47  C6 = C: R6 = R
```

Offensichtlich muß es Methoden geben, das auf flottere Weise zu erreichen, indem man dem Computer die ganze Arbeit überläßt. Lesen Sie über Arrays nach (Kapitel 25) und versuchen Sie das Programm zu verbessern.

Aufgabe 3

Ergänzen Sie die POKE-Anweisungen in Aufgabe 1 um

1. POKE 55296 + 4Ø * 7 + 9, 2 oder POKE 55585, 2
2. POKE 55296 + 4Ø * 2Ø + 32, 13 oder POKE 55968, 13
3. POKE 55296 + 4Ø * 11 + 8, 1 oder POKE 55744, 1

20 Tonbandkassetten

Mit einem Spezial-Kassettenrecorder können Sie Programme auf Band sichern und sie später neu laden, wenn Sie sie wieder verwenden wollen. Das geht so:

Wenn Sie ein Programm geschrieben haben, das etwas Hübsches leistet, und Sie besonders zufrieden mit ihm (und sich selbst) sind, wollen Sie es nicht jedesmal wieder eintippen müssen, falls Sie es jemandem zeigen oder es selbst benützen wollen. Sobald Sie den 64 aber abschalten, verliert er sein Gedächtnis. (Das tun alle Heimcomputer. Sie verwenden ein Speichersystem namens Dynamite RAM, das Strom braucht, um zu laufen.)

Oder falls Ihnen beim Schreiben eines längeren Programms die Zeit ausgeht oder Sie bei der Entwicklung vorübergehend steckenbleiben, wollen Sie beim nächstenmal nicht noch einmal alles eingeben müssen.

Sie können den Computer nicht gut ununterbrochen eingeschaltet lassen. Stattdessen übertragen Sie das Programm auf Tonband, und zwar mit dem Befehl:

```
SAVE
```

Ein zweiter Befehl

```
LOAD
```

holt das Zeug wieder vom Band und bringt es zurück in den 64. Freilich ist ein bißchen mehr dran, daher dieses Kapitel.

Das Kassettengerät

Sie brauchen ein Spezial-Kassettengerät. Normale Kassettenrecorder, wie sie überall angeboten werden, funktionieren *nicht* – ihnen fehlen die erforderlichen Anschlüsse und Bedienungsknöpfe. Das Gerät ist ein C2N Cassette Unit, erhältlich bei Ihrem Commodore-Händler. Es läuft bei den CIV-, PET-, und CBM-Modellen ebenso wie beim 64. Angeschlossen wird es an der Rückseite des 64. Ein Schlitz im entsprechenden Stecker sorgt dafür, daß er nicht falsch eingeschoben werden kann.

> *Warnung:* Stellen Sie den Stromanschluß auf OFF, bevor Sie den Kassettenrecorder einschalten, weil Sie sonst den Computer beschädigen. Lösen Sie den Recorderanschluß auch nicht, solange er unter Strom steht.

Mein Kassettenrecorder war mit einem Erdanschluß ausgestattet. Zu benötigen scheint man den nicht, aber erkundigen Sie sich, wenn Sie unsicher sind, bei Ihrem Händler.

Der 64 übernimmt den größten Teil der Steuerung für den Recorder selbst, aber ein paar Knöpfe müssen Sie drücken. Es lohnt vielleicht, im Umriß zu erklären, was sich abspielen wird, weil einige der Befehle dann mehr Sinn ergeben.

Es geht darum, daß der Computer die Programmzeichen in eine Folge von akustischen Signalen umwandelt, die sich anhören wie Morsezeichen in Hochgeschwindigkeit. Diese werden auf Tonband aufgezeichnet, so, wie Sie es mit einem Musikstück tun würden. Wenn Sie dann wieder laden möchten, werden diese Signale in den Computer zurückgeladen.

Auf dem Vorspannband der Kassette aus Kunststoff können Sie kein Programm speichern; sie können keines zurückladen, das nicht aufgezeichnet wurde oder sich an der falschen Stelle befindet; und Sie können versehentlich ein aufgezeichnetes Programm löschen, wenn Sie auf demselben Bandabschnitt ein anderes sichern. Also aufpassen!

Vorsichtsmaßnahmen

Um die besten Ergebnisse zu erzielen und das Dasein zu erleichtern:

1. Achten Sie darauf, daß Sie alles richtig zusammengeschlossen haben.
2. Sorgen Sie dafür, daß alles an den Strom angeschlossen und eingeschaltet ist. Vergewissern Sie sich, daß der Kassettenrecorder läuft.
3. *Reinigen* und *entmagnetisieren* Sie Ton- und Löschkopf Ihres Recorders. Das sind die beiden kleinen Metallblöcke, an denen das Tonband entlangläuft. Magnetische Teilchen an diesen Stellen können die Aufzeichnungsqualität beeinträchtigen und zu Schwierigkeiten führen.
4. Verwenden Sie eine Bandkassette guter Qualität. Eine Computerkassette C12 oder C15 ist ideal, eine C30 mittlerer Qualität müßte aber genügen. Spottbillige Bänder funktionieren *vielleicht,* aber sie halten nicht lange und hinterlassen beim Lauf häufig Magnetteilchen, so daß oft gereinigt werden muß.
5. Verwenden Sie kein Band von *mehr* als 3Ø Minuten Länge, weil das dem Kassettengerät schaden kann.
6. Spulen Sie die Kassette zum Anfang zurück, damit Sie wissen, wo Sie sind. Verwenden Sie ein Band, auf dem sonst nichts aufgezeichnet ist (damit *Sie* leicht hören können, was vorgeht).
7. Lassen Sie das Band vorlaufen, bis der Vorspann durch ist. Dort aufzuzeichnen, hat keinen Sinn; es geht nicht.

SAVE testen

Geben Sie ein einfaches Testprogramm ein, meinetwegen:

```
1Ø PRINT "EINFACHES TESTPROGRAMM"
2Ø GOTO 1Ø
```

oder was Ihnen sonst einfällt. Dann schreiben Sie:

```
SAVE "FRED"
```

und drücken RETURN. Auf dem Bildschirm erscheint die Meldung:

```
PRESS RECORD & PLAY ON TAPE
```

Drücken Sie also Aufnahme- und Wiedergabetasten des Recorders. (Ganz fest: Achten Sie darauf, daß die Tasten richtig eingerastet sind, bevor Sie loslassen, sonst müssen Sie noch einmal anfangen.)

Der Bildschirm wird nun für einige Sekunden leer. Der Kassettenrecorder fängt von selbst an zu laufen, tut das eine Weile und bleibt stehen. Es erscheint die Meldung:

```
SAVING FRED
READY
```

Das war's! Ihr Programm befindet sich jetzt auf dem Magnetband, und zwar unter dem Namen FRED.

LOAD testen

Spulen Sie das Band zum Anfang zurück und schreiben Sie NEW, um das Programm aus dem Speicher zu entfernen. Das geschieht nur, damit wir sicher sein können, daß das Band wirklich das Programm enthält.

Tippen Sie:

```
LOAD "FRED"
```

und drücken Sie RETURN. Die Meldung lautet nun:

```
PRESS PLAY ON TAPE
```

Drücken Sie die Taste Wiedergabe am Recorder. Das Band läuft wieder und hält nach einer Weile an (nur Geduld!).

```
OK
SEARCHING FOR FRED
FOUND FRED
```

Nach einer Pause (die Sie durch Druck auf die COMMODORE-Taste abkürzen können) läuft das Band weiter, und wenn alles gut ist,bleibt es mit der Schlußmeldung stehen:

```
LOADING
READY
```

Sie können das Programm jetzt mit LIST anzeigen lassen, um nachzuprüfen, ob es wirklich wieder im Speicher ist, und es wie gewohnt mit RUN fahren.

Programmnamen

Warum SAVE "FRED" und LOAD "FRED"? Was hat der arme alte Fred damit zu tun?

Ein Programm auf Band soll einen Namen erhalten, damit es von irgendwelchen anderen auf demselben Band zu unterscheiden ist. Der Name kann beliebig sein und bis zu 16 Zeichen umfassen. Erlaubte Programmnamen sind also:

```
FRED
MARTHA
PROG1
+ + # # # # A < > DD7.
```

SAVE ohne Namen sichert das Programm einfach auf Band; SAVE, gefolgt von einem Namen in Anführungszeichen, sichert zusätzlich einen 'Kopfblock', der den Namen enthält und später gefunden werden kann. LOAD ohne Namen lädt einfach das erste Programm, das auf Band gefunden wird. LOAD, gefolgt von einem Namen in Anführungszeichen, sucht, bis es den entsprechenden Kopfblock findet, und lädt dann das Programm.

Der Grund, warum man Namen auf diese Weise verwendet, ist der, daß Sie mehrere Programme mit verschiedenen Namen auf einem Band aufzeichnen und dann jenes laden können, das Sie ausgewählt haben. Sichern Sie, um das zu beobachten, mehrere Testprogramme unter verschiedenen Namen mit SAVE hinteinander (achten Sie aber darauf, nicht eines auf das vorherige aufzuspielen):

```
SAVE "TEST1"
SAVE "TEST2"
SAVE "TEST3"
```

(Die Programme können alle gleich sein, wenn Sie wollen; aber Sie sehen vielleicht eher ein, daß es klappt, wenn sie unterscheidbar sind.)

Spulen Sie das Band zurück und schreiben Sie dann:

```
LOAD "TEST3"
```

Die übliche Meldung

```
PRESS PLAY ON TAPE
```

erscheint; halten Sie sich daran. Warten Sie. Das Band beginnt zu laufen, der Bildschirm wird leer, das Band bleibt stehen, und die Meldung

SEARCHING FOR TEST3

FOUND TEST1

erscheint. Es gibt eine lange Pause (abzukürzen durch Druck auf die COMMODORE-Taste), dann geht es weiter . . . und diesmal kommt:

SEARCHING FOR TEST3

FOUND TEST1

FOUND TEST2

Endlich wird TEST3 gefunden und wie gewohnt geladen.

VERIFY

Dieser Befehl wirkt genau wie ein LOAD, nur lädt er das Programm nicht in den Computer, sondern vergleicht es mit dem, was sich schon im Speicher befindet. Die Meldungen und der Ablauf der Ereignisse bleiben im Grunde gleich. Der Hauptzweck von VERIFY ist die Prüfung, ob ein Programm wirklich gesichert worden ist. Also SAVE, zurückspulen, VERIFY.

Der Kassettenspieler hat einen eingebauten Bandzähler; mit ihm können Sie feststellen, wo auf dem Band Sie sich befinden. Es ist eine ausgezeichnete Idee, auf der Karte in der Kassettenbox eine Liste der Programme und Bandzahlen zu führen.

Aber was ist, wenn Sie das vergessen haben und herausfinden wollen, was sich auf einem Band befindet . . . ohne jedoch ein Programm zu stören, das sich schon im Computer befindet? Ein Versuch mit LOAD ist sinnlos; Sie könnten Erfolg haben und das verlieren, was schon im Computer ist.

Der Kniff: Man verwendet

VERIFY "NICHTS"

wobei NICHTS für irgendeinen Namen steht, den Sie nicht für ein Programm verwenden. Der Computer sucht dann nach NICHTS und zeigt dabei auf dem Bildschirm eine Liste aller Programme, die sich auf dem Band befinden:

SEARCHING FOR NICHTS

FOUND FRED

FOUND MARTHA

FOUND CRAYSIMULATOR

FOUND SCHACHPROGRAMM

etc.

Sie können sich bei jeder Stufe die Bandzahlen notieren.

Software im Handel

Sie können Programme kaufen, die schon auf Band aufgezeichnet sind. Der Ladevorgang bei ihnen ist derselbe wie bei den von Ihnen gesicherten Programmen. Spulen Sie die Kassette zum Anfang zurück und schreiben Sie LOAD. (Wenn das Programm einen Namen hat, sollten Sie ihn aus der beigegebenen Dokumentation erfahren, zusammen mit irgendwelchen besonderen Ladeinstruktionen; aber einfaches LOAD müßte immer genügen.) Haben Sie Geduld, weil die Hersteller oft lange Bandabschnitte vor dem Programm leerlassen.

Schutz gegen Löschen

Hi-Fi-Experten kennen sich da aus, aber Leute, die einen Kassettenspieler eigens für den Computer gekauft haben, vielleicht nicht. Sie können eine Kassette gegen Löschen (durch versehentliches Drücken von RECORD) schützen, wenn Sie am Kassettenrand der Bandseite gegenüber ein Plastikplättchen entfernen. Bei einer beliebigen Seite nach oben schützt das linke Plättchen diese Seite. Brechen Sie es mit einem kleinen Schraubenzieher oder einem ähnlichen Instrument heraus.

Wenn Sie auf einem geschützten Band aufnehmen wollen, kleben Sie einfach ein Stückchen Klebeband über das Loch, wo das Plättchen war.

Im Handel angebotene Kassetten werden in der Regel mit Löchern anstelle der Plättchen geliefert.

Programme ketten

Wenn ein Programm geladen ist, müssen Sie immer noch RUN schreiben, um es zu fahren. Sie können LOAD aber auch von *innerhalb* eines Programms verwenden. In diesem Fall läuft das geladene Programm automatisch.

Sie können also auf Band eine Kette von Programmen aufbauen, vielleicht so:

```
PROG1:  1Ø
        2Ø
        ...
        5ØØØ LOAD "PROG2"
PROG2:  1Ø
        2Ø
        ...
        7ØØØ LOAD "PROG3"
```

```
PROG3   1Ø
        2Ø
        . . .
        8962 LOAD "PROG4"
```

und so weiter. Laden Sie PROG1 mit LOAD per Hand und fahren Sie mit RUN; der Rest läuft automatisch. Beispielsweise könnten Sie ein Videospiel 'Golf' betreiben. PROG1 betrifft das erste Loch, PROG2 das zweite, und so weiter; Sie erreichen die Zeile, die das nächste Programm lädt, nur, wenn Sie den Ball ins Loch befördern. Bei komplizierter Grafik, die zuviel Platz erfordert, als daß die ganze Folge von Programmen gleichzeitig aufgenommen werden könnte, erweitert das den Bereich des Möglichen.

Für andere Verwendungen von Bandkassetten siehe Kapitel 34 zu Files.

Hinweis: Für den Commodore 64 wie auch für den VC 2Ø gibt es jetzt auch Diskettenlaufwerke.

21 Debugging III

Bleistift und Papier haben immer noch ihren Nutzen . . .

Schreibtischtests

Häufig sind Laufzeitfehler nicht so leicht zu finden wie die in "Debugging II". Wenn Sie nicht gleich erkennen können, was nicht stimmt, müssen Sie den Code sorgfältig und systematisch durchgehen. Das kann eine sehr mühsame Sache sein, aber wenn Sie sich eisern daran halten, ist der Erfolg beinahe garantiert.

Ich will die Methode für den "Schreibtischtest" eines Programms an einem Beispiel erläutern. Wir stellen uns vor, Sie hätten den folgenden Programmcode in einem Zeitschriftenartikel gefunden:

```
10  INPUT "NAECHSTE ZAHL EINGEBEN"; N
20  IF N > 0 THEN S = S + N: C = C + I: GOTO 10
30  PRINT "MITTEL IST"; S/C
40  INPUT "WEITERE DATEN (J/N)"; Q$
50  IF Q$ = "J" THEN 10
```

und der dazugehörige Text teilt Ihnen mit, Sie würden das Mittel einer Reihe positiver Zahlen erhalten, beendet mit Null. Mit anderen Worten: Sie könnten eingeben

```
NAECHSTE ZAHL EINGEBEN? 3
NAECHSTE ZAHL EINGEBEN? 4
NAECHSTE ZAHL EINGEBEN? 5
NAECHSTE ZAHL EINGEBEN? 0
```

und die Meldung erhalten

```
MITTEL IST 4
```

(Die Null gehört nicht zu den Daten; sie ist der Begrenzer, siehe Kapitel 11.) Das Programm wird den Anwender dann fragen, ob er es mit einer neuen Datenmenge versuchen will:

```
WEITERE DATEN (J/N)?
```

Wenn Sie auf diese Frage "J" eingeben, fängt das Programm von vorne an und verlangt erneut eine Folge von Zahlen.

Wenn Sie das Programm eingeben und mit den Zahlen 3, 4, 5 und Ø fahren, werden Sie feststellen, daß es nicht funktioniert. Es nimmt die Zahlen wie erwartet an, aber sobald Sie die Null eingegeben haben, stürzt es mit der Meldung ab:

?DIVISION BY ZERO ERROR IN 3Ø

Der Nullteiler-Dämon hat wieder zugeschlagen!

Um nun festzustellen, was der dumme Computer angestellt hat, tun Sie so, als wären *Sie* die Maschine, und gehorchen sklavisch den Programmanweisungen, wie der Computer es täte. Sie weisen nach, wie die verschiedenen Werte, die das Programm verarbeitet, sich in einer Tabelle folgender Art verändern:

Zeile Nr.	S	N	C	I	Verzweigung

Dieses Programm enthält nur vier Variable: S, N, C und I. Während jede Zeile ausgeführt wird, tragen Sie die Zeilennummer ein, die neuen Werte von S, N, C und I (falls sie in dieser Zeile verändert werden) und, wenn die Anweisung ein IF . . . THEN ist, in die Spalte "Verzweigung" einen Haken, falls ein Sprung stattfindet, sonst ein Kreuz.

Bevor wir aber anfangen, tut der 64 noch eines, sobald Sie RUN tippen, nämlich, alle Variablen auf Null zu setzen, so daß die Tabelle zu Beginn so aussieht:

Zeile Nr.	S	N	C	I	Verzweigung
	Ø	Ø	Ø	Ø	

Wir wollen das Programm mit den Zahlen 3, 4, 5 und Ø testen. Die erste Zeile 1Ø ist ausgeführt, 3 wird an N weitergegeben, also:

Zeile Nr.	S	N	C	I	Verzweigung
1Ø	Ø	Ø 3	Ø	Ø	

Bei Zeile 2Ø ist N größer als Ø, also wird die 3 zu S addiert, I(Ø) zu C, und die Verzweigung zu Zeile 1Ø findet statt:

Zeile Nr.	S	N	C	I	Verzweigung
	Ø	Ø	Ø	Ø	
1Ø		3			
2Ø	3		Ø		V

Wenn wir das fortsetzen, erhalten wir:

Zeile Nr.	S	N	C	I	Verzweigung
	Ø	Ø	Ø	Ø	
1Ø		3			
2Ø	3		Ø		
1Ø		4			V
2Ø	7		Ø		
1Ø		5			V
2Ø	12		Ø		
1Ø		Ø			V
2Ø					
3Ø					x

Wir können also erkennen, warum der Fehler aufgetreten ist. Aus der Tabelle ergeben sich Hinweise darauf, wie jede Variable sich verändert. sehen Sie sich zuerst Spalte N an. Wir sehen die eingegebenen 3, 4, 5 und Ø der Reihe nach in Erscheinung treten, was den Erwartungen entspricht. S ist schon interessanter. Hier sehen wir, wie die Summe der Zahlen in N langsam entsteht. Zuerst erscheint die 3, dann 7 (3 + 4), dann 12 (3 + 4 + 5). Und was ist mit C los? Dort ist absolut nichts los! Da drei Zahlen eingegeben wurden, sollte das Mittel 12/3 = 4 betragen, also müßte C 3 sein. Der Grund, weshalb C bei Null bleibt, ist in Wahrheit der, daß ihm immer wieder I zuaddiert wird und I auf Null festgelegt ist. Angenommen, I wäre ein Schreibfehler für 1? Dann sähe die Tabelle so aus:

Zeile Nr.	S	N	C	Verzweigung
	Ø	Ø	Ø	
1Ø		3		
2Ø	3		1	V
1Ø		4		
2Ø	7		2	V
1Ø		5		
2Ø	12		3	V
1Ø		Ø		
2Ø				x
3Ø				

Print 12/3=4

Jetzt sehen die Dinge vielversprechend aus. Zeile 2Ø wird redigiert, aus dem "l" eine "1" gemacht, und mit RUN gefahren. Geben Sie wie vorher 3, 4, 5, Ø ein, angezeigt wird 4. Nun geben Sie "J", wenn Sie gefragt werden, ob weitere Daten kommen, und versuchen Sie es mit 1Ø, 2Ø, 3Ø, Ø. Das Programm zeigt die Lösung 12 an, was seltsam ist, weil das Mittel von 1Ø, 2Ø und 3Ø nämlich 2Ø ist!

Logikfehler

Das ist nun ein Beispiel für eine neue Art von Fehler – einen logischen Fehler. Das Programm leistet etwas und schließt seine Aufgabe ab, ohne daß dem 64 etwas auffiele, aber was immer das Programm auch gemacht haben mag, es hat *nicht* das Mittel aus 1Ø, 2Ø und 3Ø genommmen.

Fahren wir mit dem Schreibtischtest also fort, um festzustellen, was schiefgegangen ist. Die bisherige Arbeit zu wiederholen, hat wenig Sinn, also brauchen wir nur festzuhalten, daß, wenn wir Zeile 4Ø erreichen, S = 12, N = Ø und C = 3. Außerdem brauchen wir nun die Stringvariable Q$:

Zeile Nr.	S	N	C	QS	Verzweigung
4Ø	12	Ø	3	J	
5Ø					V
1Ø		1Ø			
2Ø	22		4		V

Weiter, glaube ich, brauchen wir nicht zu gehen, um das Problem zu erkennen. Da S zu Beginn 12 ist, wird es 22, obwohl es nur 1Ø, und C zu 4, wo es 1 sein sollte. Anders ausgedrückt: Der 64 hat die erste Folge von Werten nicht vergessen, als er die zweite verarbeitete, wird also ausrechnen:

(3 + 4 + 5 + 1Ø + 2Ø + 3Ø)/6 = 12

Jetzt ist das leicht zu beheben. Wir fügen einfach eine Zeile 5 ein:

```
5  S = Ø: C = Ø
```

und verändern Zeile 5Ø zu:

```
5Ø  IF Q$ = "J" THEN 5
```

Mit welcher Wahrscheinlichkeit treten Fehler dieser Art auf? Nun ja, Schreibfehler von der Art I für 1, 2 für Z sind ziemlich häufig (wenn auch nicht in diesem Buch, wie ich hoffe), und selbst wenn der Fehler nicht auf der gedruckten Seite steht, ist die Wahrscheinlichkeit groß, daß Ihnen bei der Eingabe eines Programms gelegentlich unbemerkt ein Übertragungsfehler unterläuft. Selbst wenn Sie ein eigenes Programm schreiben, werden Sie sich wundern, wie oft Sie den Namen einer Variablen vergessen und die falsche verwenden oder sogar dieselbe Variable für zwei verschiedene Dinge verwenden. Sie sollten vorsichtig sein und sich die Namen aller Variablen und ihrer Funktion gleich zu Beginn aufschreiben; aber so, wie der Mensch nun einmal gebaut ist . . .

Der zweite Fehler zeigt zwei wichtige Dinge. Erstens: Obwohl BASIC dadurch von Nutzen zu sein versucht, daß es Variable auf Null initialisiert, ohne daß Sie das verlangen müssen, ist das doch kein reiner Segen. Obwohl man es vielleicht nicht muß, sollte man zu Beginn eines Programms alle Variablen festlegen, weil damit die Möglichkeit eines solchen Fehlers vermieden wird. Ich meine nicht, daß Ihnen ein derart banaler Fehler unterlaufen könnte, aber wenn man ein Programm verfaßt, das sich mit einer bestimmten Datenmenge befaßt, und es erst später ändert, damit es viele Datenmengen verarbeiten kann, läßt sich schon begreifen, daß die Initialisierung übersehen wird.

Zweitens zeigt er, wie sorgfältig Sie beim Testen eines Programms vorgehen müssen. Wenn ein Programm bei einer bestimmten Datenmenge läuft, dürfen Sie sich noch nicht zu der Schlußfolgerung verleiten lassen, das würde auch bei jeder anderen geschehen. Zu diesem Thema habe ich in den Kapiteln 26 und 29 noch mehr zu sagen.

Aufgabe 1

Hier ein Beispiel für Sie, an dem Sie Ihr Können erproben können. Das folgende Programm soll eine Folge von positiven Werten (Null bis 1ØØØ) annehmen und am Ende den größten und kleinsten gelesenen Wert anzeigen. Als Begrenzer wird eine negative Zahl verwendet. Beispiel: Wenn die Daten 8, 4, 3, 9, −1 lauten, sollte das Programm demnach anzeigen

```
9          3
```

Einen Hinweis gebe ich Ihnen – man muß nach zwei Fehlern suchen.

```
1Ø  MAX = Ø: MIN = Ø
2Ø  INPUT N
3Ø  IF N < MIN THEN MIN = N
4Ø  IF N > MAX THEN MAX = N
5Ø  IF N < Ø THEN PRINT MAX, MIN: END
6Ø  GOTO 2Ø
```

Lösung

Aufgabe 1

Bei meinem Beispiel sieht der Schreibtischtest so aus:

Zeile Nr.	MAX	MIN	N	Verzweigung
	Ø	Ø	Ø	
1Ø	Ø	Ø		
2Ø			8	
3Ø				
4Ø	8			
5Ø				
6Ø				V
2Ø			4	
3Ø				
4Ø				
5Ø				
6Ø				V
2Ø			3	
3Ø				
4Ø				
5Ø				
6Ø				V
2Ø			9	
3Ø				
4Ø	9			
5Ø				
6Ø				V
2Ø			–1	
3Ø		–1		
4Ø				
5Ø				

Print 9, –1

Der *größte* Wert wird also richtig angezeigt, der kleinste dagegen nicht. Wenn wir uns die MAX-Spalte ansehen, können wir erkennen, wie das Programm arbeiten soll. Jedesmal, wenn ein Wert größer als der in MAX erscheint, ersetzt er den laufenden Wert in MAX. MIN sollte sich natürlich ebenso verhalten, so daß dort zuerst 8, dann 4, dann 3 auftauchen sollten. In MIN geht aber keine Veränderung vor, bis –1 eingegeben wird, was aber gar nichts bewirken soll, weil das ja nur ein Begrenzer ist!

Der Grund, warum keine neuen Werte in MIN gesetzt werden: Der kleinste mögliche Wert (Null) ist dort schon enthalten. MIN sollte vielmehr mit einem sehr großen Wert beginnen, damit der erste erscheinende Wert ihn ersetzt, denn der neue Wert *muß* kleiner sein. Da als der größtmögliche Wert 1ØØØ bekannt ist, geht *alles*, was größer ist. Zeile 1Ø muß also lauten:

```
1Ø  MAX = Ø: MIN = 1ØØ1
```

Nun müssen wir verhindern, daß das begrenzende −1 nach MIN gelangt. Kein Problem. Wir müssen fragen, ob die Dateneingabe beendet ist, bevor wir versuchen, den Wert nach MAX oder MIN zu übertragen. Zeile 5Ø sollte also gelöscht und als Zeile 25 neu geschrieben werden.

22 Zufallszahlen

Sogar ein Computer kann unberechenbar sein!

Bei manchen Programmen möchte man, daß der Computer sich unberechenbar verhält. Der 64 verfügt über einen Befehl, der "Zufalls"-Zahlen hervorbringt, und diese können Sie verwenden, wenn der Computer etwas tun soll, Sie aber vorher nicht wissen wollen, was er macht. Besonders nützlich ist das bei Spielen. Wieviele Spiele kennen Sie, bei denen gewürfelt oder eine Karte gezogen wird?

Der Befehl für Zufallszahlen ist

RND

gefolgt von einer Zahl in Klammern.

Damit Sie einen Begriff davon bekommen, was dieser Befehl leistet, schreiben Sie ab und fahren Sie:

```
1Ø INPUT N
2Ø FOR T = 1 TO 1Ø
3Ø PRINT RND (N)
4Ø NEXT T
5Ø GOTO 1Ø
```

Geben Sie für N zuerst 1 ein. Sie erhalten zehn Dezimalzahlen zwischen Ø und 1 ohne erkennbare Anordnung. Nehmen Sie noch einmal 1. Sie erhalten wieder zehn Zahlen, noch immer ohne jede Ordnung.

Nehmen Sie jetzt Ø. Wieder erhalten Sie zehn beliebige Zahlen zwischen Ø und 1.

Jetzt nehmen Sie −2. Sie erhalten eine Zahl, die sich zehnmal wiederholt. Die allgemeine Folge von RND (N) ist nämlich:

1. Ist N positiv, erzeugt RND (N) Zufallszahlen zwischen Ø und 1 (Ø kann vorkommen, 1 nicht.) Das geschieht allerdings auf "wiederholbare" Weise.
2. Ist N Null, so liefert RND (Ø) eine Zufallszahl, bestimmt durch die Gesamtzeit, die seit dem Einschalten des Computers abgelaufen ist. In einem gewissen Sinn geht es nicht zufälliger!
3. Ist N negativ, erzeugt RND (N) eine *bestimmte* "Zufalls"-zahl, die von N abhängt.

Ich schlage vor, daß Sie Punkt 3 ganz beiseite lassen. Er wird meist für Debugging verwendet und steht in Verbindung mit dem konkreten Prozeß, durch den die "Zufalls"-Zahlen erzeugt werden. (Sie sind nicht wirklich zufällig, unterstellt, daß das Wort überhaupt etwas zu bedeuten hat, aber der Prozeß, der sie

hervorbringt, gesteuert von einer Zahl, die *Keim* genannt wird, ist darauf ausgerichtet, für die meisten praktischen Zwecke ungeordnete Ergebnisse zu erzeugen.) Der einzige Befehl, den Sie wohl je brauchen werden, ist:

RND (Ø)

und bei dem bleibe ich von jetzt an.

Würfel, Spielkarten und Glücksspieleinrichtungen

Halt mal, da klopft es an die Tür . . . Nein, Herr Inspektor, ich habe keine Spielklublizenz . . . Na gut, wenn Sie darauf bestehen, ändere ich die Überschrift des Abschnitts ab zu

Zufallsereignisse

Mann! Aber nun zur Sache. Wenn Sie einen Würfel rollen, liefert er zufällige Zahlen zwischen 1 und 6. Der "Würfel" im 64 liefert Zahlen zwischen Ø und 1, die Dezimalzahlen sein können. Ein bißchen mathematische Jongleurkunst ist also angebracht.

Hier eine typische Liste von zufälligen Zahlen des 64 (in der ersten Spalte) zusammen mit dem, was geschieht, wenn Sie mit 6 multiplizieren.

RND(Ø)	6 * RND(Ø)
.131137465	.78682479
.80924873	4.85549238
.846447204	5.07868323
.591965711	5.04921935
.26800113	3.55179427

Multiplikation mit 6 dehnt also den Zahlenbereich von Ø–1 zu Ø–6 aus. Ein Schritt in die richtige Richtung. Der nächste besteht darin, die Dezimalpunkte loszuwerden. Der Befehl

INT

ersetzt eine Zahl durch ihren *Integerteil*, die größte ganze Zahl, die nicht größer ist als sie. Bei positiven Zahlen ist dies der Teil vor dem Dezimalpunkt, bei negativen Zahlen um 1 kleiner. Beispiel:

INT (4.85549238) = 4

INT (−3.131592) = −4

Steht vor dem Dezimalpunkt nichts, gibt INT Ø (bei positiven Zahlen). Wenn wir aus der rechten Spalte also INT-Werte nehmen, erhalten wir die Zahlenfolge:

Ø 4 5 5 3

was für Würfelspiele fast schon richtig ist. Der einzige Haken: Sie gehen von Ø bis 5, statt von 1 bis 6. Wir fügen also 1 hinzu:

1 5 6 6 4

was genau richtig ist. Alles zusammengenommen, erzeugt

INT (6 * RND (Ø)) + 1

wie ein Würfel im Bereich 1–6 zufällige ganze Zahlen.

Aufgabe 1

Welche Befehle würden Sie verwenden, um zufällige ganze Zahlen hervorzubringen entsprechend:

1. Einem Paket von 52 Spielkarten?
2. Einer einzelnen Farbe von 13 Spielkarten?
3. Einem Domino aus einem vollständigen Satz von 28 Steinen?
4. Einem Geburtstag in einem Nichtschaltjahr?
5. Einer Zahl zwischen 1Ø und 99 (beide eingeschlossen)?

Frequenzdiagramm

Hier ein Programm von halbwegs ernster Art. Es wirft 15Ø Würfel und hält fest, wie oft jede Zahl erscheint.

```
1Ø  PRINT CHR$ (147)
2Ø  POKE 53281,Ø
3Ø  POKE 5328Ø,Ø
4Ø  POKE 646,1
5Ø  FOR R = 1 TO 6
6Ø  REIHE = 3 * R: CLM = Ø: CDE = 48 + R: CR = 1: GOSUB 1ØØØØ
7Ø  NEXT R
8Ø  FOR T = 1 TO 15Ø
9Ø  W = INT (6 * RND (Ø) + 1)
1ØØ  C (W) = C (W) + 1
11Ø  IF C (W) > 38 THEN STOP
12Ø  REIHE = 3 * W: CLM = C (W) + 2: CDE = 16Ø: CR = W + 1:
     GOSUB 1ØØØØ
```

```
13Ø  NEXT T
14Ø  GOTO 13Ø

1ØØØØ  REM PRINT AT
1ØØ1Ø  POKE 1Ø24 + 4Ø * REIHE + CLM, CDE
1ØØ15  POKE 55296 + 4Ø * REIHE + CLM, CR
1ØØ2Ø  RETURN
```

Zeilen 1Ø–4Ø leeren den Bildschirm und setzen die Farben. Die Anzahl, wie oft der Würfel die Zahl 1, 2, 3, 4, 5, 6 gezeigt hat, wird in sechs Variablen C(1), C(2), C(3), C(4), C(5), C(6) festgehalten. Ich habe ein bißchen geschwindelt, weil diese Variablenliste das ist, was man ein *Array* nennt, ich aber erst in Kapitel 25 zu Arrays komme. Egal, was gemeint ist, wird deutlich.

Zeilen 5Ø–7Ø zeigen die Zahlen 1–6 in einer Kolonne an. Das 48+R erzeugt lediglich die Zeichen für 1–6, deren Bildschirmcodes (siehe Anhang E des Handbuchs) 49–54 sind. Beachten Sie den Gebrauch der PRINT-AT-Subroutine ab Zeile 1ØØØØ.

Zeilen 8Ø und 13Ø werfen den Würfel 15Ømal.

Zeile 1ØØ erhöht die Zählung für den jeweiligen Wurf.

Zeile 12Ø zeigt in der Zeile mit der Zahl W auf dem Bildschirm einen Block Farbe an. Zeile 11Ø verhindert lediglich Probleme, wenn die Anzeigeposition vom rechten Bildschirmrand wandert.

Beachten Sie, wie die Balken wachsen; auch wenn bei einem Lauf eine Zahl im Vorsprung sein mag, auf die Dauer gleicht sich das aus. Der Würfel ist *fair*.

Lösung

Aufgabe 1

1. INT (52 * RND (Ø)) + 1
2. INT (13 * RND (Ø)) + 1
3. INT (28 * RND (Ø)) + 1
4. INT (365 * RND (Ø)) + 1
5. INT (9Ø * RND (Ø)) + 1Ø, weil 9Ø * RND (Ø) von Ø bis 89 geht, die Addition von 1Ø also den Bereich 1Ø–99 liefert. die 9Ø habe ich erhalten durch Abzählen des Bereichs 99 − 1Ø = 89 plus eins am Ende, dann mit der 1Ø eingeregelt.

Es *könnte* sein, daß Sie erwogen haben, das Dominoproblem (Nr. 3) durch INT (7 * RND (Ø)) zu bewältigen, um Punkte Ø–6 zu erzeugen, mit Ø für einen leeren (halben) Dominostein. Tun Sie das zweimal, behalten Sie aber nur Paare M, N, für die gilt M < = N, sonst bekommen Sie die nicht-doppelten Dominosteine wie [3|6] zweimal, einmal so und einmal als [6|3], die doppelten [5|5] aber nur einmal, was die Wahrscheinlichkeit verändert.

23 PET-Grafik

Der 64 ist ausgestattet mit einem enormen Bereich von Grafikzeichen, die man dazu benützen kann, auf dem Schirm Bilder aufzubauen.

Es kann Ihnen kaum entgangen sein, daß der 64 neben Buchstaben und Zahlen auch Grafikzeichen anzeigen kann. Diese sind vom Keyboard aus zugänglich und stehen auf der *Vorderseite* der Tasten. Das rechte Zeichen wird bedient durch SHIFT, das linke durch die COMMODORE-Taste. Die Zeichen, ein wilder, aber vielseitiger Haufen, stammen noch vom Commodore PET-Computer, daher diese Kapitelüberschrift.

Was den Computer angeht, ist ein PET-Grafikzeichen wie jedes andere auch, auf dieselbe Weise mit PRINT anzuzeigen, über seinen ASCII-Code zugänglich und mit POKE zum Bildschirm zu bringen.

Für Verfasser von Büchern stellen die Grafikzeichen ein Problem dar: Im Satzbestand der Normaldruckerei sind sie nicht zu finden. Außerdem sind sie oft schwer zu unterscheiden (wie Sie feststellen werden, wenn Sie sich gedruckte Programme in Zeitschriften ansehen). Wir brauchen also eine Übereinkunft, um sie besprechen zu können.

Nehmen Sie Taste A. Links befindet sich ein Grafikzeichen in der Art eines kleinen rechten Winkels, rechts das Symbol einer Pik. Ich bezeichne sie der Reihe nach als

gAc (Grafik-A + COMMODORE-Taste)

gAs (Grafik-A + SHIFT-Taste)

und ebenso bei anderen Tasten.

Gitterkonstruktion

Sie können auf einem Gitter aus PET-Grafik Bilder aufbauen. Beispielsweise zeigt Abbildung 23.1 den Versuch, ein Auto zu zeichnen.

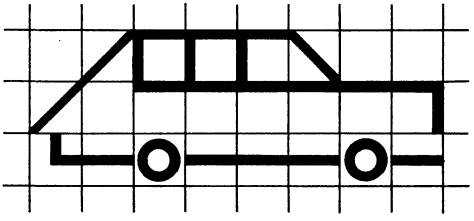


Abbildung 23.1
Pet-Grafikentwurf für ein Auto

Die verwendeten Zeichen sind:

□ gNs gOs gOs gOs gMs □ □
gNs □ gYc gYc gYc gYc gYc gPs
gZc gCs gWs gCs gCs gCs gWs gCs

Wir könnten also den Wagen etwa so auf den Bildschirm zeichnen:

```
1Ø PRINT CHR$(147)
2Ø PRINT: PRINT: PRINT
3Ø PRINT TAB(15); "□ gNs gOs gOs gOs gMs □ □"
4Ø PRINT TAB(15); "gNs □ gYc gYc gYc gYc gYc gPs"
5Ø PRINT TAB(15); "gZc gCs GWs gCs gCs gCs gWs gCs"
6Ø GOTO 6Ø
```

Aufgabe 1

Entwerfen Sie eine Lok mit einem angehängten Waggon. Zeigen Sie den Zug auf dem Bildschirm.

Bewegte Grafik

Vielleicht ist Ihnen der Gedanke gekommen, man könnte das Auto in Bewegung versetzen, wenn bei der Anzeige verschiedene TAB-Werte eingegeben werden. Ungefähr so:

```
1Ø PRINT CHR$(147)
2Ø FOR C = Ø TO 3Ø
3Ø PRINT CHR$(19)
4Ø PRINT: PRINT: PRINT
5Ø PRINT TAB(C); "□ gNs gOs gOs gOs gMs □ □"
6Ø PRINT TAB(C); "gNs □ gYc gYc gYc gYc gYc gPs"
7Ø PRINT TAB(C); "gZc gCs gWs gCs gCs gCs gWs gCs"
8Ø NEXT C
```

Versuchen Sie es. Irgend etwas stimmt nicht ganz! Wir hinterlassen einen Rattenschwanz von Autos.

Das Problem: Wir haben vergessen, das vorige Auto zu löschen, wenn wir das neue anzeigen. Sie *könnten* Zeile 3Ø verändern zu:

```
3Ø  PRINT CHR$(147)
```

aber das empfehle ich nicht, weil der Bildschirm zu stark flackert.

Eine bessere Methode ist die, das Auto mit einem "unsichtbaren Schwanz" von Leerstellen auszustatten, der vorige Autos auslöscht. Sie brauchen nur die Zeilen 5Ø–7Ø umzuschreiben und vor jeden String von Grafikzeichen eine zusätzliche Leerstelle zu setzen:

```
5Ø  PRINT TAB(C); "□ □ gNs gOs gOs gOs gMs □ □"
6Ø  PRINT TAB(C); "□ gNs □ gYc gYc gYc gYc gYc gPs"
7Ø  PRINT TAB(C); "□ gZc gCs gWs gCs gCs gCs gWs gCs"
```

Jetzt ist alles in Ordnung.

Aufgabe 2

Das Auto soll von Spalte 3Ø zu Spalte Ø zurückstoßen. (Hinweis: Sie brauchen einen "unsichtbaren Bug" statt einen Schwanz.)

Verwendung von Stringvariablen

Wenn Sie auf diese Weise eine Folge von Grafikzeichen anzeigen müssen, ist es viel besser, sie als Stringvariable zu setzen. Hier etwa ein Programm, bei dem zwei Autos ein Rennen gegeneinander austragen.

```
1Ø  PRINT CHR$(147)
2Ø  C1 = Ø: C2 = Ø: C = Ø
3Ø  A$ = "□ □ gNs gOs gOs gOs gMs"
4Ø  B$ = "□ gNs □ gYc gYc gYc gYc gYc gPs"
5Ø  C$ = "□ gZc gCs gWs gCs gCs gCs gWs gCs"
6Ø  R = 8: GOSUB 1ØØØ
7Ø  R = 16: GOSUB 1ØØØ
8Ø  N = INT (1 + 2 * RND (Ø))
9Ø  IF N = 1 THEN C1 = C1 + 1: R = 8: C = C1
1ØØ IF N = 2 THEN C2 = C2 + 1: R = 16: C = C2
11Ø GOSUB 1ØØØ
12Ø IF C1 = 3Ø THEN PRINT CHR$(19); "AUTO 1 SIEGT": STOP
```

```
13Ø  IF C2 = 3Ø THEN PRINT CHR$(19); "AUTO 2 SIEGT": STOP
14Ø  GOTO 8Ø

1ØØØ  PRINT CHR$(19);
1Ø1Ø  FOR T = 1 TO R: PRINT: NEXT T
1Ø2Ø  PRINT TAB(C); A$
1Ø3Ø  PRINT TAB(C); B$
1Ø4Ø  PRINT TAB(C); C$
1Ø5Ø  RETURN
```

In Zeile 1ØØØ beginnt eine Subroutine, die in Reihe R, Spalte C ein Auto anzeigt. Anfangs sind die Spaltennummern C1 und C2 für Autos 1 und 2 noch Ø, die Reihen 8, 16. Zeile 8Ø entscheidet zufällig, welches Auto sich bewegt: 9Ø–11Ø bewegen es; 12Ø–13Ø prüfen, welches gewonnen hat.

Aufgabe 3

Schreiben Sie das Programm so um, daß Auto 1 rot und Auto 2 grün angezeigt wird. (Hinweis: PRINT CHR$(28) erzeugt Ink rot, PRINT CHR$(3Ø) Ink grün.)

Lösungen

Aufgabe 1

```
1Ø  PRINT CHR$(147)
2Ø  A$ = "gAc gCs gCs gCs gSc □ gOs gYc g–s □ □ □ gNc g–s"
3Ø  B$ = "g–s g+c □ g+c g–s □ gPs □ gEc g–s g–s g–s gZc gIs"
4Ø  C$ = "g–s G W R gQc gRc gOs A M T R A K g–s"
5Ø  D$ = "gZc gWs gCs gWs gXc □ gCs gQs gCs gCs gQs gCs gQs
    gXc"
6Ø  FOR T = 1 TO 7: PRINT: NEXT T
7Ø  PRINT TAB(8); A$
8Ø  PRINT TAB(8); B$
9Ø  PRINT TAB(8); C$
1ØØ  PRINT TAB(8); D$
11Ø  GOTO 11Ø
```

Beachten Sie, wie in Zeile 4Ø Textzeichen (G W R, etc.) und Grafik gemischt werden.

Aufgabe 2

```
1Ø  PRINT CHR$(147)
2Ø  FOR C = 3Ø TO Ø STEP -1
3Ø  PRINT CHR$(19)
4Ø  PRINT: PRINT: PRINT
5Ø  PRINT TAB(C); "□ gNs gOs gOs gOs gMs □ □ □"
6Ø  PRINT TAB(C); "gNs □ gYc gYc gYc gYc gYc gPs □"
7Ø  PRINT TAB(C); "gZc gCs gWs gCs gCs gCs gWs gCs □"
8Ø  NEXT C
```

In Zeile 5Ø brauchen Sie am rechten Ende nur einen Leerraum; warum?

Aufgabe 3

Fügen Sie eine Zeile ein:

```
1Ø15  PRINT CHR$(26 + 2 * N)
```

und schreiben Sie diese Zeilen um:

```
6Ø  R = 8: N = 1: GOSUB 1ØØØ
7Ø  R = 16: N = 2: GOSUB 1ØØØ
```

24 Keyboardsteuerung

Der Computer kann angewiesen werden, das Keyboard abzusuchen und je nachdem, was er vorfindet, auf verschiedene Weise zu reagieren. Ein Anwendungsgebiet: Die Steuerung bewegter Grafik.

Der Haken bei einem INPUT-Befehl ist der, daß er alles aufhält, bis die Eingabe erfolgt ist. Stellen Sie sich ein Videospiel vor, bei dem alles stehenbleibt, bis Sie die Tasten drücken . . . Würde nicht viel Spaß machen! Der Befehl

```
GET
```

weist den Computer an, die Tastatur abzusuchen und festzustellen, was gedrückt wird. Das Ergebnis wird in irgendeiner von Ihnen gewählten Stringvariablen gespeichert. Wenn Sie A$ verwenden wollen, sähe das Format so aus:

```
GET A$
```

(hübsch zum Merken: hol nen String, nicht? Na ja, schon gut.)

Hier ein einfaches Testprogramm:

```
1Ø  PRINT CHR$(147)
2Ø  GET A$
3Ø  PRINT "SIE DRUECKTEN TASTE □"; A$
4Ø  GOTO 2Ø
```

Eine endlose Linie von Mitteilungen saust vorbei, aber wenn Sie eine Taste drücken, etwa "Z", sehen Sie auch das Z vorbeirasen, ebenso bei den anderen Tasten.

Der Grund für soviel Aktion: Wenn Sie keine Taste drücken, setzt der Computer A$ auf den leeren String "" und macht trotzdem weiter. Um das zu umgehen, fügen Sie ein:

```
25  IF A$ = "" THEN 2Ø
```

Jetzt sitzt er in einer kleinen Schleife und wartet darauf, daß Sie eine Taste drücken; schon viel friedlicher! Experimentieren Sie. Probieren Sie die RETURN-Taste (durch die das Display um eine Zeile weiterspringt: RETURN wirkt wie NEWLINE). Versuchen Sie es mit CTRL-Tasten, etwa mit CTRL + 5.

Bewegte Grafik

Führen wir die Idee einen Schritt weiter und verwenden wir die Tasten dazu, Grafik zu steuern. Kehren wir zu unserem Auto aus Kapitel 23 zurück. Nehmen

wir an, wir wollen es, wenn wir Taste "V" drücken, eine Stelle vorwärts-, wenn wir "R" drücken, eine Stelle rückwärtsbewegen. Das geht so:

```
10  PRINT CHR$(147)
20  C = 15
30  A$ = "□ □ gNs gOs gOs gOs gMs □"
40  B$ = "□ gNs □ gYc gYc gYc gYc gYc gPs □"
50  C$ = "□ gZc gCs gWs gCs gCs gCs gWs gCs □"
60  R = 12: GOSUB 1000
70  GET G$                                  A$ ist schon verwendet!
80  IF G$ = "V" THEN C = C + 1
90  IF G$ = "R" THEN C = C - 1
100 IF C < 0 THEN C = 0
110 IF C > 29 THEN C = 29
120 GOTO 60

1000 PRINT CHR$(19);
1010 FOR T = 1 TO R: PRINT: NEXT T
1020 PRINT TAB(C); A$
1030 PRINT TAB(C); B$
1040 PRINT TAB(C); C$
1050 RETURN
```

Die Spalte, in der das Auto angezeigt wird, ist in Variable C enthalten. Zeilen 80 und 90 passen C je nach der gedrückten Taste an. Zeilen 100 und 110 schützen vor dem Verlassen des Bildschirms.

Beachten Sie den unsichtbaren Bug *ebenso* wie den unsichtbaren Schwanz in den Zeilen 30–50. Das Auto kann sich ebensogut in der einen wie in der anderen Richtung bewegen.

Aufgabe 1

Schreiben Sie das Programm so um, daß das Auto sich nicht nur seitlich, sondern auf dem Bildschirm auch auf- und abwärts bewegt, und zwar mit den Taste "O" und "U". (Bilden Sie, wenn Sie wollen, die Grafik zu einem Hubschrauber um.)

Dummer Puffer

GET hat einen Haken, der damit zusammenhängt, wie das Keyboard gelesen wird, wenn es in Gebrauch ist. Es gibt einen *Keyboardpuffer* von bis zu 10 Zeichen, und auf Druck einer Taste wird das Ergebnis (falls Platz ist) im Puffer gespeichert. Die Folge: Wenn Sie *mehrere* Tasten drücken (wie das in der Hitze des Gefechts schon vorkommen kann), werden Sie alle gespeichert und *befolgt*.

Wenn Sie das sehen wollen, verlangsamen Sie das fahrende Auto durch Anfügen einer Verzögerung:

```
115  FOR T = 1 TO 1000: NEXT T
```

Fahren Sie das Programm jetzt und drücken Sie mehrmals *schnell* auf "V". Sie werden sehen, wie das Auto sich bewegt, erneut bewegt und weiter bewegt, bis der Pufferspeicher leer ist. Tippen Sie *schnell* VRVRVR und warten Sie: das Auto bewegt sich hin und her.

Um diese Eigenheit des Computers zu überwinden, können Sie die Länge des Keyboardpuffers steuern mit einem Befehl:

```
POKE 649,1
```

was ihn so setzt, daß er nur noch ein Zeichen aufnehmen kann. (Für N Zeichen nehmen Sie POKE 649,N). Probieren Sie das durch Hinzufügen einer Zeile:

```
5  POKE 649,1
```

Jetzt ruft Mehrfachdruck auf Tasten nur noch *eine* Bewegung hervor – die der nach dem vorherigen GET erstmals gedrückten Taste.

Automatische Wiederholung

Sie werden außerdem feststellen, daß, sobald Sie eine Taste gedrückt halten, der GET-Befehl das nicht mehr zur Kenntnis nimmt. Er liest eine Taste und beachtet sie dann nicht mehr, bis diese Taste losgelassen wird. Sie können das Auto also nicht einfach dadurch schnell vorwärtsbewegen, daß Sie "V" ständig gedrückt halten. Oft wollen Sie aber genau das.

Das Geheimnis ist die Systemvariable in Adresse 197, die eine (sonderbar codierte) Version der derzeit niedergedrückten Taste enthält. Geben Sie dieses kleine Programm ein:

```
2000  PRINT PEEK (197)
2010  GOTO 2000
```

Und jetzt RUN 2000. Eine Kolonne 64 läuft den Bildschirm hinunter. Drücken Sie eine Taste, meinetwegen "A", und halten Sie sie unten; aus der Zahl wird 10. Versuchen Sie es mit anderen Tasten. Sie werden sehen, daß jede ein anderes Ergebnis bringt. Die richtigen Zahlen sind in Anhang 7 aufgeführt; es sind weder ASCII-, noch Bildschirmcodes, sondern etwas völlig anderes!

Der Code für "V" ist 31, für "R" 17. Wenn wir unser Automobil also automatisch wiederholen lassen wollen, können wir so vorgehen:

Löschen Sie Zeile 115. Schreiben Sie 7Ø–9Ø so um:

```
7Ø  A = PEEK (197)
8Ø  IF A = 31 THEN C = C + 1
9Ø  IF A = 17 THEN C = C – 1
```

Geben Sie nun wieder RUN, halten Sie die Tasten V oder R gedrückt und beobachten Sie die sofortige Reaktion.

Die Funktionstasten

Sie haben sich vielleicht gefragt, wozu die vier großen, auffälligen Tasten auf der rechten Keyboardseite dienen mögen. Das Handbuch nennt sie "programmierbare Funktionstasten" und erwähnt sie anschließend mit keinem Wort mehr.

Es sind in Wahrheit acht Tasten, ohne Umschaltung:

f1 f3 f5 f7

und mit SHIFT:

f2 f4 f6 f8

In Wirklichkeit sind sie nur "Attrappen"-Tasten, die von einem GET-Befehl gelesen werden können. Eine echte "programmierbare" Taste würde sofort reagieren, sobald sie mit einer vom Anwender eingegebenen Standardroutine gedrückt wird; aber diese Tasten reagieren nur, wenn Sie selbst eine GET-Anweisung programmieren. Innerhalb des Commodore-BASIC werden Sie echte Programmierbarkeit leider nicht erreichen können. Mit den Funktionstasten haben Sie aber wenigstens Standardpositionen zur Verfügung, die den ganzen Kram "drückt man Taste "Z" oder "K", um die fremde Raumflotte zu zerblasen" überflüssig machen.

Die ASCII-Codes für die Funktionstasten (siehe Kapitel 18) sind:

f1	133
f2	137
f3	134
f4	138
f5	135
f6	139
f7	136
f8	14Ø

Eine Art, etwa Taste f5 zu verwenden, ist also diese:

```
15Ø  GET A$: IF A$ = CHR$(135) THEN und so weiter . . .
```

Eine andere Methode ist die, den Zitiermodus zu benützen (Kapitel 7):

```
15Ø  GET A$: IF A$ = "[Taste f5 drücken]" THEN . . .
```

Nehmen wir zum Beispiel an, Sie möchten, wenn Sie Taste f5 drücken, den Bildschirm löschen. Dann könnten Sie verwenden:

```
15Ø  GET A$: IF A$ = "[Taste f5]" THEN PRINT CHR$(147)
```

Geben Sie das ein und fahren Sie mit RUN, wenn Sie wollen; drücken Sie f5. Geschieht etwas? Nur, wenn Sie sehr schnell sind, weil f5 *nur* in Betrieb ist, solange der GET-Befehl ausgeführt wird. Fügen Sie ein

```
16Ø  GOTO 15Ø
```

dann wird es besser . . .

Der zivilisierte Weg, mit diesen Tasten umzugehen, ist der, Subroutinen zu schreiben, etwa:

```
15ØØØ  GET A$: IF A$ = "[Taste f5]" THEN PRINT CHR$(147)
15Ø1Ø  RETURN
```

An strategischen Stellen verstreuen Sie dann GOSUB 15ØØØ-Befehle in Ihrem Programm, so daß es ständig Taste f5 prüft. Beispielsweise können Sie ein solches GOSUB in die innerste Schleife eines Mehrschleifensystems setzen.

Insoweit, als die Funktionstasten groß und leicht zu finden sind, ist das eine ausgezeichnete Methode. Denken Sie aber daran, daß Taste f5 nichts leistet, was Sie auf dieselbe Art nicht auch mit irgendeiner anderen Taste, etwa "Z", tun könnten.

Aufgabe 2

Schreiben Sie eine Routine, die eine Tabelle von Quadraten zwischen 1 und 1 Million anzeigt und, wenn Sie Taste f7 drücken, die PAPER-Farbe schrittweise von Ø–15 wandern läßt.

Lösungen

Aufgabe 1

Geben Sie dem Auto zunächst oben und unten einen unsichtbaren Rand:

```
25  D$ = "[1Ø x □]"
55  E$ = "[1Ø x □]"
```

Nun lassen Sie sie anzeigen:

```
1Ø15  PRINT TAB (C); D$
1Ø45  PRINT TAB (C); E$
```

Dann erweitern Sie den Bereich der Keyboardabfrage:

```
92  IF G$ = "O" THEN R = R - 1
93  IF G$ = "U" THEN R = R + 1
```

und schützen Sie:

```
95  IF R < Ø THEN R = Ø
96  IF R > 18 THEN R = 18
```

Schließlich teilen Sie Zeile 6Ø auf in:

```
55  R = 12
6Ø  GOSUB 1ØØØ
```

Für die Hubschraubergrafik zeigt Abbildung 24.1 einen Vorschlag. Das führt zu:

```
3Ø  A$ = "□ gCs gCs gRc gCs gCs □ □"
4Ø  B$ = "□ gUs gOs gOs gYc gMs gPc gQs □"
5Ø  C$ = "□ gJs gCs gWs gCs gCs gCs gKs □"
```

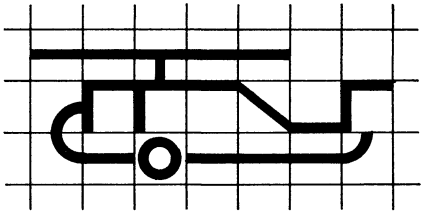


Abbildung 24.1
PET-Grafik-Entwurf für einen Hubschrauber

Aufgabe 2

```
10  PRINT CHR$(147)
20  CL = 0
30  FOR T = 1 TO 1000000
40  PRINT T, T * T
50  GOSUB 5000
60  NEXT T
70  STOP

5000  GET A$
5010  IF A$ <> "[Taste f7]" THEN RETURN
5020  POKE 53281, CL
5030  CL = CL + 1: IF CL = 16 THEN CL = 0
5040  RETURN
```

25 Arrays

Computerprogramme erfordern oft Informationslisten: Zahlen oder Wörter. Eine Art, solche Listen zu speichern, nutzt Arrays.

Ein *Array* ist im Grunde eine numerierte Liste von Einzelposten, wie eine Wäscheliste:

1. Socken
2. Hemd
3. Kissenbezug
4. Pullover

Wenn jemand Posten 3 braucht, kann er/sie in der Liste nachsehen, was das sei (hier also ein Kissenbezug). Es gibt zwei Arten von Arrays: *numerische* Arrays und *String*arrays.

Nehmen wir an, wir wollen im Computer eine Liste der zwölf Monate im Jahr in Reihenfolge speichern. Monatsnamen sind Strings, also verwenden wir ein Stringarray.

(Die Regeln für Arraynamen entsprechen denen für Variablennamen. Nur die beiden ersten Buchstaben zählen.)

Als erstes müssen wir dem Computer mitteilen, *wie lang die Liste sein wird*. Das nennt man "das Array dimensionieren", und es geschieht durch einen Befehl

```
DIM
```

in dieser Art:

```
10  DIM MONAT$ (12)
```

Streng genommen, kämen Sie auch mit 11 statt 12 durch, weil (aus Gründen, auf die ich hier nicht eingehen will) Arrays im 64 ab Ø gezählt werden: Ein wie oben auf 12 dimensioniertes Array hat die Einträge Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1Ø, 11, 12 – insgesamt dreizehn! Der Verwirrung entgeht man am besten dadurch, daß man den Ø-Eintrag nicht beachtet.

Als nächstes müssen Sie dem Computer mitteilen, was die 12 Eintragungen sind:

```
20  MONAT$ (1) = "JANUAR"
30  MONAT$ (2) = "FEBRUAR"
    ...
130 MONAT$ (12) = "DEZEMBER"
```

Ich habe Punkte verwendet, aber *Sie* müssen alle zwölf Monate eingeben. Pech für Sie. Sie können sich jetzt auf jeden beliebigen Eintrag im Array beziehen,

wenn Sie den Namen des Arrays verwenden, gefolgt von der Postennummer in Klammern. Beispiel:

```
MONAT$ (7)
```

erweist sich als "JULI", und so weiter.

Um also einen beliebigen Monat anzeigen zu lassen, könnten Sie die Zeilen anfügen:

```
14Ø  INPUT "ZAHL DES MONATS"; N
15Ø  PRINT MONAT$ (N)
```

Ebenso können Sie numerische Arrays aufbauen:

```
1Ø  DIM ALICE (5ØØ)
```

mit Eintragungen:

ALICE (Ø)
ALICE (1)
ALICE (2)
. . .
ALICE (5ØØ)

oft am besten ignoriert, aber nicht immer

Arrays nehmen rasch viel Speicherplatz in Anspruch. Wenn Sie ein großes Array haben, dessen Einträge *Integer* sind (also keine Dezimal-, sondern ganze Zahlen) können Sie Platz sparen, indem Sie *Integervariable* verwenden; nähere Einzelheiten in Kapitel 3Ø. Eine Integervariable ist wie eine gewöhnliche numerische Variable, aber dem Namen muß ein %-Zeichen folgen:

```
DIM ALICE% (5ØØ)
```

Multidimensionale Arrays

Multidimensionale Arrays funktionieren ganz ähnlich, aber die Einträge bilden eine Art Tabelle. Bei, sagen wir, zwei Dimensionen:

```
DIM ALICE (7, 4)
```

besteht die Tabelle aus sieben Reihen und vier Spalten. Das heißt, eigentlich aus *acht* Reihen und *fünf* Spalten, weil eine zusätzliche Reihe Ø und Spalte Ø eingeschlossen sind, genau wie im eindimensionalen Fall:

ALICE (Ø, Ø)	ALICE (Ø, 1)	ALICE (Ø, 2)	ALICE (Ø, 3)	ALICE (Ø, 4)
ALICE (1, Ø)	ALICE (1, 1)	ALICE (1, 2)	ALICE (1, 3)	ALICE (1, 4)
ALICE (2, Ø)	ALICE (2, 1)	ALICE (2, 2)	ALICE (2, 3)	ALICE (2, 4)
ALICE (3, Ø)	ALICE (3, 1)	ALICE (3, 2)	ALICE (3, 3)	ALICE (3, 4)
ALICE (4, Ø)	ALICE (4, 1)	ALICE (4, 2)	ALICE (4, 3)	ALICE (4, 4)
ALICE (5, Ø)	ALICE (5, 1)	ALICE (5, 2)	ALICE (5, 3)	ALICE (5, 4)
ALICE (6, Ø)	ALICE (6, 1)	ALICE (6, 2)	ALICE (6, 3)	ALICE (6, 4)
ALICE (7, Ø)	ALICE (7, 1)	ALICE (7, 2)	ALICE (7, 3)	ALICE (7, 4)

Die Zeilen teilen das Material bei Reihe oder Spalte Ø ab, was man am besten oft unbeachtet läßt... je nachdem. Der Eintrag in Reihe REIHE und Spalte SPALTE ist:

```
ALICE (REIHE, SPALTE)
```

Sie können auch ein zweidimensionales Stringarray verwenden:

```
DIM ALICE$ (7, 4)
```

dessen Einträge Strings sind, nicht Zahlen.

Arrays sind sehr nützliche Einrichtungen, weil enorm viel Information auf natürliche Weise in tabellarischer Form vorkommt. Sie brauchen nur an ein Telefonbuch zu denken:

Name	Adresse	Telefonamt	Nummer
Franz Bauer	Marktgasse 23	Basel	555555
Karl Huber	Hauptplatz 8	Dorfen	4242
Otto Schenk	Goethestraße 46	München	2323712
Marie Murks	Odenwaldplatz 9	Frankfurt	1Ø4767

Auf diese Weise aufgeteilt, können Sie sich das als Vierspalten-Stringarray vorstellen. (Die Telefonnummern müßten als Strings betrachtet werden, weil man Strings und Zahlen in einem Array nicht mischen darf.)

Spielkarten zeichnen

Ich will vorführen, wie nützlich Arrays sein können, wenn man ein Programm entwickelt, das auf dem Bildschirm ein Pokerblatt (5 Karten) zeigen soll. Um das Programmlisting in Grenzen zu halten, nehme ich ein paar Vereinfachungen vor:

1. Nur Karten von As bis 1Ø sind erlaubt; keine Figuren – oder Bilderkarten. (Im Prinzip ist es nicht schwierig, diese herzustellen; entwerfen Sie sie einfach mit PET-Grafik. Das Abfassen ist aber langwierig, so daß ich es hier unterlasse, wo es das Grundsätzliche überdeckt.)
2. Jede Karte wird zufällig ausgewählt und kein Versuch unternommen, festzustellen, ob dieselbe Karte zweimal vorkommt. Siehe Aufgabe 1, wie das umgangen werden kann.

Na gut, sieht schwierig aus, also gehen wir "top down" vor. Die Grundschritte werden sein:

Zufällige Karte auswählen

Zufällige Farbe auswählen

Karte an geeigneter Stelle auf den Bildschirm zeichnen

Für 5 Karten der Reihe nach wiederholen

Das Hauptproblem wird offenkundig sein, eine Karte zu ziehen, also konzentriere ich mich zunächst darauf.

Wir werden Symbole für Herz, Treff, Karo und Pik brauchen; kein Problem – PET-Grafik. Auch der Kartenrand kann mit PET-Grafik gezeichnet werden. Schwierig wird es wohl bei der *Anordnung* der Symbole. Ich strebe ein befriedigendes Ergebnis an, aber keines, das so gut aussieht wie eine echte Spielkarte.

Nachdem ich ein paar Vorstellungen auf Papier ausprobiert hatte, beschloß ich, für die Symbole, angeordnet wie in Abbildung 25.1, ein Normalgitter zu verwenden.

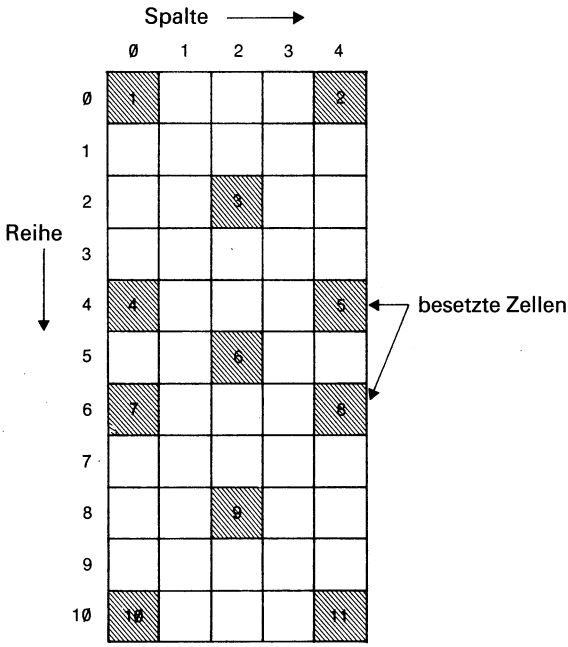


Abbildung 25.1
Normalgitter für Spielkartenpunkte

Es gibt 11 mögliche Stellen, wo Symbole hinkommen können, im Bild numeriert. Beispielsweise können Sie eine gute Pik-Drei erzielen, wenn Sie an die Stellen 3, 6, 9 je ein Pikzeichen setzen.

Für jeden Wert der Kartennummer 1–1Ø muß ich also festlegen, *welche* dieser Zellen besetzt werden soll. Ich verwende für die Kartennummer eine Variable KARTE, die von 1 bis 1Ø reicht. Wenn ich auf die Byte-Logik zurückgreife, kann ich jedem Wert von KARTE einen String von 11 Bits aus Ø oder 1 zuteilen – eine Ø am Platz K bedeutet, daß Zelle K in Abbildung 25.1 leergelassen wird, eine 1, daß das Farbensymbol dort vorhanden ist. KARTE = 3 entspricht also der Folge ØØ1ØØ1ØØ1ØØ, wo 1 nur an den Stellen 3, 6, 9 auftaucht. Das ist eine gute Idee, weil ich alle zehn Folgen in einem Stringarray C$ unterbringen kann.

Da das Gesamtlisting ziemlich lang ist, liefere ich es Ihnen Stück für Stück, und Sie können das Ganze eingeben, wie Sie wollen – ich schlage vor, ebenfalls Stück für Stück. Bis jetzt führt meine Vorstellung zu:

```
1ØØ  DIM C$ (1Ø)
11Ø  C$(1) = "ØØØØØ1ØØØØØ"
12Ø  C$(2) = "ØØ1ØØØØØ1ØØ"
13Ø  C$(3) = "ØØ1ØØ1ØØ1ØØ"
14Ø  C$(4) = "11ØØØØØØØ11"
15Ø  C$(5) = "11ØØØ1ØØØ11"
16Ø  C$(6) = "11Ø11ØØØ11"
17Ø  C$(7) = "111ØØØ11Ø11"
18Ø  C$(8) = "111ØØØ11111"
19Ø  C$(9) = "11Ø11111Ø11"
2ØØ  C$(1Ø) = "11111Ø11111"
```

Anzeigepositionen

Bis jetzt schön und gut, aber diese Zellen besetzen recht unregelmäßig angeordnete Positionen. Ich kann das umgehen, wenn ich Reihe und Spalte für eine gegebene Zelle K als den Eintrag K von numerischen Arrays R und C speichere. Abgelesen von Abbildung 25.1 ergibt das:

```
25Ø  DIM R(11): DIM C(11)
26Ø  R(1) = Ø: C(1) = Ø
27Ø  R(2) = Ø: C(2) = 4
28Ø  R(3) = 2: C(3) = 2
29Ø  R(4) = 4: C(4) = Ø
3ØØ  R(5) = 4: C(5) = 4
31Ø  R(6) = 5: C(6) = 2
```

```
32Ø  R(7) = 6: C(7) = Ø
33Ø  R(8) = 6: C(8) = 4
34Ø  R(9) = 8: C(9) = 2
35Ø  R(1Ø) = 1Ø: C(1Ø) = Ø
36Ø  R(11) = 1Ø: C(11) = 4
```

Farben

Ich numeriere die vier Kartenfarben so:

1. Herz
2. Treff
3. Karo
4. Pik

Also: Die Bildschirm-POKE-Werte für die vier entsprechenden Grafiksymbole sind 83, 88, 89, 65. Wieder unregelmäßig. Wir verwenden deshalb ein anderes numerisches Array S, um sie aufzunehmen:

```
4ØØ  S(1) = 83: S(2) = 88: S(3) = 9Ø: S(4) = 65
```

(S hat nur vier Einträge, also spare ich mir die Mühe des Dimensionierens.)

Herz und Karo sind *rote* Farben, die anderen schwarz. Die Farbcodes Ø (schwarz) und 2 (rot) leisten das, und wir können sie in einem Array K unterbringen:

```
41Ø  K(1) = 2: K(2) = Ø: K(3) = 2: K(4) = Ø
```

Hauptprogramm

Nun zur Sache. Um das Dasein zu vereinfachen, wollen wir damit anfangen, auszuarbeiten, wie *eine* Karte entsteht, deren Symbolgitter (Abbildung 25.1) angezeigt wird mit ihrer oberen Ecke an Reihe REIHE und Spalte CLM (allgemein denken, ja?). Im Augenblick setze ich REIHE auf 5 und CLM auf 7, aber das wird sich später ändern, wenn wir das ganze Programm in eine Schleife setzen, um ein Blatt von *fünf* Karten zu erzielen.

Als erstes wählen wir Zufallswerte für KARTE und FARBE:

```
5ØØ  KARTE = 1 + INT (1Ø * RND (Ø))
51Ø  FARBE = 1 + INT (4 * RND (Ø))
```

Dann setzen wir REIHE und CLM:

```
52Ø  REIHE = 5
53Ø  CLM = 7
```

Nun finden wir Symbolcode und Farbcode für die Kartenfarbe:

```
54Ø  SCD = S (FARBE): KR = K (FARBE)
```

Als nächstes zeichnen wir die Karte. Da wir noch nicht wissen, wie wir das machen, verwende ich eine Subroutine:

```
55Ø  GOSUB 2ØØØ: REM ZEICHNE KARTE
```

und für den Augenblick:

```
56Ø  STOP
```

Die Subroutine Zeichne-eine-Karte

Wir haben in Abbildung 25.1 11 Zellen, jede entsprechend einem möglichen "Punkt" auf der Karte. Wir müssen also alle 11 Positionen im String C$ (KARTE) absuchen und, wenn wir eine 1 sehen, an der richtigen Stelle einen Punkt zeichnen.

```
2ØØØ  REM PUNKT SUCHEN
2Ø1Ø  FOR X = 1 TO 11
2Ø2Ø  U$ = MID$ (C$ (KARTE), X, 1)
2Ø3Ø  IF U$ = "Ø" THEN 2Ø5Ø
2Ø4Ø  GOSUB 3ØØØ: REM PUNKT ZEICHNEN
2Ø5Ø  NEXT X
2Ø6Ø  RETURN
```

Wird schon . . . Nun zum Punktezeichnen:

```
3ØØØ  REM PUNKT ZEICHNEN
3Ø1Ø  RC = REIHE + R (X): CC = CLM + C (X)
3Ø2Ø  POKE 1Ø24 + 4Ø * RC + CC, SCD
3Ø3Ø  POKE 55296 + 4Ø * RC + CC, KR
3Ø4Ø  RETURN
```

Wenn Sie das mit RUN fahren, werden Sie erstens feststellen, daß wir den Bildschirm leeren und die Farbe wählen sollten, und außerdem, daß die Karte keinen Rand hat. Die Initialisierung ist einfach:

```
1Ø  POKE 53281,7
2Ø  PRINT CHR$ (147)
```

Darstellung verbessern

Um der Karte einen Rand zu geben sowie Farbe und Wert dafür zu schreiben, brauchen wir noch eine weitere Subroutine:

```
545  GOSUB 4ØØØ: REM UMRANDUNG KARTE
```

Ich kann das Schlimme nicht länger vor mir herschieben – also los.

```
4ØØØ  REM UMRANDUNG KARTE
4Ø1Ø  PRINT CHR$ (19);
4Ø2Ø  FOR Y = 1 TO REIHE -3: PRINT: NEXT Y
4Ø3Ø  CØ = CLM - 4
4Ø4Ø  PRINT TAB (CØ); "gUs [11 mal g * s] gIs"
4Ø5Ø  FOR Y = 1 TO 15
4Ø6Ø  PRINT TAB (CØ); "g - s [11 x □] g - s"
4Ø7Ø  NEXT Y
4Ø8Ø  PRINT TAB (CØ); "gJs [11 mal g * s] gKs"
```

Das liefert die Umrandungen. Beachten Sie die Grafikzeichen: g * s ist Grafik * plus SHIFT, g - s ist Grafik – plus SHIFT.

Nun Farbe und Wert:

```
41ØØ  PRINT CHR$(19);
411Ø  FOR Y = 1 TO REIHE -2: PRINT: NEXT Y
412Ø  PRINT TAB (CLM -3); KARTE
413Ø  PRINT TAB (CLM -3); CHR$ (SCD + 32)
414Ø  RETURN
```

Das SCD + 32 dient dazu, von Bildschirm-POKE-Codes in ASCII umzuwandeln. Die verschiedenen CLM -4, CLM -3, etc. wurden durch Herumprobieren gefunden (was lange dauerte); bilden Sie sich nicht ein, es ginge immer so glatt, wie es im Buch steht!

Ein ganzes Blatt

Nachdem das läuft und entfehlert ist – sieht recht hübsch aus, nicht wahr? – können wir jetzt in die Schleife setzen, um fünf Karten zu zeichnen.

Löschen Sie Zeilen 52Ø, 53Ø und 56Ø. Fügen Sie ein:

```
49Ø  FOR Z = 1 TO 5
52Ø  REIHE = 5 + Z: CLM = 5 + 4 * Z
56Ø  NEXT Z
57Ø  GOTO 57Ø
```

Das wär's.

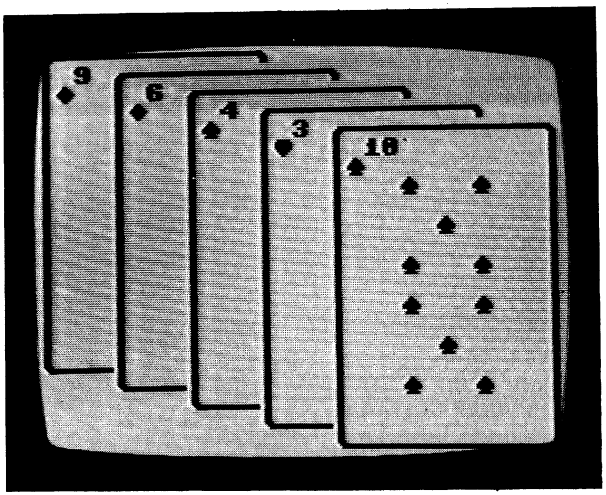


Abbildung 25.2
Das Endergebnis: Ein Pokerblatt.

Aufgabe 1

Fügen Sie eine Routine ein, die jede Zufallsauswahl von Karte und Farbe überprüft, um sich zu vergewissern, daß sie nicht schon einmal vorgekommen ist, und es in diesem Fall erneut versucht. (Ein Blatt mit zwei Pik-As in einer Pokerrunde würde den Gegenspielern kaum behagen . . .)

Die nächste Aufgabe bekommt von mir keine Nummer, weil ich die Zeit nicht aufwenden will, die für eine gute Erklärung erforderlich wäre. Aber trotzdem: Überlegen Sie sich, wie Sie Figurenkarten zeichnen würden.

Lösung

Wir müssen ein Array aufbauen und jede neue Karte samt Farbe damit vergleichen. Hier ist eine Methode.

```
511  FOR W = 1 TO Z
512  IF F (W) = KARTE AND G (W) = FARBE THEN 5ØØ
513  NEXT W
514  F (Z) = KARTE: G (Z) = FARBE
```

Das speichert die Werte für KARTE in Array F, FARBE in G. Da sie nur 5 Posten lang sind, brauchen F und G nicht dimensioniert zu werden.

Streng genommen ist es ungut, aus einer Schleife zu springen. Wir sollten also lieber eine Flagge setzen, auf die nach Schleifenende reagiert wird:

```
512  IF F (W) = KARTE AND G (W) = FARBE THEN FLAGGE = 1
515  IF FLAGGE = 1 THEN FLAGGE = Ø: GOTO 5ØØØ
516  FLAGGE = Ø
```

Aber das ist schon ein bißchen pedantisch.

26 Debugging IV

Wie man Computer veranlaßt, Fehler selbst zu beseitigen!

In Debugging III sprach ich vom 'Schreibtischtest' eines Programms – auf dem Papier den Fluß der Befehle nachverfolgen. Der Hauptnachteil bei dieser Methode ist, daß der Anwender – also Sie – die ganze Arbeit allein tun muß. Der 64 hockt behaglich da und läßt Sie wissen, er habe keine Ahnung, wovon Sie reden, oder, noch schlimmer, liefert nur falsche Antworten. Es wird Zeit, ihn an die Arbeit zu schicken, dafür ist er schließlich da.

Man kann ihn dazu bringen, selbst einen 'Trockenlauf' zu veranstalten. Sehen wir uns zum Beispiel noch einmal Aufgabe 1 aus Debugging III an:

```
10  MAX = 0: MIN = 0
20  INPUT N
30  IF N < MIN THEN MIN = N
40  IF N > MAX THEN MAX = N
50  IF N < 0 THEN PRINT MAX, MIN: END
60  GOTO 20
```

Das soll Höchst- und Mindestwert einer Reihe von Eingaben finden. Tut es aber nicht! Den (die) Fehler eruiert man nun so: Fügen Sie eine Zeile 5 ein, um die Tabellenüberschriften und in jeder Zeile die Zeilennummer und jeden Wert, der sich in dieser Zeile ändert, in den entsprechenden Spalten anzuzeigen:

```
 5  PRINT "LN □ □ □ MAX □□ MIN □ □ N □ □ □ □
    VERZWEIGUNG"
10  MAX = 0: MIN = 0: PRINT "10"; TAB (5) ; MAX ; TAB (10); MIN
20  INPUT N: PRINT "20"; TAB (15); N
30  IF N < MIN THEN MIN = N: PRINT "30"; TAB (10); MIN
40  IF N > MAX THEN MAX = N: PRINT "40"; TAB (5); MAX
50  IF N < 0 THEN PRINT MAX, MIN: END
60  PRINT "60"; TAB (20); "V": GOTO 20
```

Der TAB-Befehl sorgt lediglich dafür, daß in der entsprechenden Bildschirmspalte angezeigt wird. Die Form der daraus entstehenden Tabelle ist von der auf dem Papier ein bißchen verschieden. Zeilen, in denen keine Veränderung eintritt, werden nicht aufgeführt, aber dadurch wird das Ganze eher übersichtlicher. Ein ernsteres Problem ist, daß die Eingabe-Anforderungen und -werte die

Tabelle auseinanderreißen. Im Idealfall sollten die beiden Arten von Ausgabe an verschiedene Stellen geschickt werden. Beispielsweise könnte die Tabelle über einen Drucker, oder, wenn Sie keinen haben, über Kassettenband ausgegeben werden. Das erfordert eine Kenntnis des File-Systems im 64, und ich komme auf dieses Problem zurück, wenn ich in Kapitel 34 Dateien besprochen habe. Falls Sie keinen Drucker besitzen, haben Sie vielleicht den Wunsch, bei der Zahl der angezeigten Werte sparsam zu verfahren. Beispiel: In diesem Fall wird es vermutlich schon genügen, die Werte in MAX und MIN am Ende jeder Schleife zu kennen. Es wäre also ausreichend, dem Programm nur eine Zeile 55 einzufügen:

```
55 PRINT MAX, MIN
```

Sie möchten die Variablen vielleicht verändern, um sie bei verschiedenen Testläufen zu untersuchen. Ein guter Kniff ist der, in eine PRINT-Anweisung, die Sie lahmlegen wollen, ein REM einzufügen, also:

```
55 REM PRINT MAX, MIN
```

Da Zeile 55 jetzt eine Bemerkung ist, beachtet der Computer sie nicht. Wenn Sie sie später wieder brauchen, erspart Ihnen das, die Zeile neu schreiben zu müssen – Sie brauchen nur das REM zu löschen.

Der 64 kann uns also mitteilen, wie seine Variablen sich verändern. Er könnte noch zwei nützliche Dinge tun:

1. Er könnte uns sagen, welche Zeilen er ausführt und in welcher Reihenfolge.
2. Er könnte uns sagen, wie oft er bestimmte Zeilen oder Codeblöcke ausführt.

Ablaufüberwachung

Die einfachste Methode, den Weg zu verfolgen, den ein Programm nimmt, ist die, in jede Zeile eine PRINT-Anweisung einzufügen, die einfach die ausgeführten Zeilennummern anzeigt. Das Maximum/Minimum-Programm sähe dann so aus:

```
1Ø PRINT "< 1Ø >": MAX = Ø: MIN = Ø
2Ø PRINT "< 2Ø >": INPUT N
3Ø PRINT "< 3Ø >": IF N < MIN THEN MIN = N
4Ø PRINT "< 4Ø >": IF N > MAX THEN MAX = N
5Ø PRINT "< 5Ø >": IF N < Ø THEN PRINT MAX, MIN: END
6Ø PRINT "< 6Ø >": GOTO 2Ø
```

Ich habe die angezeigten Zeilennummern in Winkelklammern gesetzt, damit man sie nicht mit den vom Programm ausgegebenen Zahlen verwechseln kann. Sie sollen nicht den Kopf verlieren.

Beachten Sie, daß der verfolgte Teil der Zeile stets zuerst auftaucht. Normalerweise ist die Reihenfolge unwichtig, aber wenn ich Zeile 3Ø so abgefaßt hätte:

```
3Ø  IF N < MIN THEN MIN = N: PRINT "< 3Ø >"
```

würde das PRINT nur dann ausgeführt werden, wenn N *kleiner* wäre als MIN, die Verfolgung wäre also unvollständig.

Ablaufüberwachungen können verwirrender sein als nutzvoll, wenn man sie unbekümmert und wahllos anwendet. Man muß bei jeder Debuggingmethode darauf achten, vernünftige Fragen zu stellen, und den Computer dann veranlassen, die Antworten auf genau *diese* Fragen zu liefern. Es hat keinen Sinn, einen kompletten Test anzustellen, wenn Sie eigentlich nur wissen wollen, ob eine bestimmte Variable jemals den Wert 25 erreicht. Je mehr Daten Sie sich vom Gerät anzeigen lassen, desto länger brauchen Sie für die Analyse. Bei Ablaufüberwachungen ist es genauso. Wenn Sie sicher sind, daß ein bestimmter Codeblock richtig ausgeführt wird, warum ihn protokollieren? Im großen und ganzen werden wir uns damit befassen, ein Programm in der Umgebung von Verzweigungen zu überwachen.

Hier ein Beispiel. Wir schreiben einen Codeteil, der einen Tag des Monats für Verwendung an anderer Stelle in einem Programm annimmt. Es empfiehlt sich, den eingegebenen Wert zu prüfen, um dafür zu sorgen, daß er im Bereich 1–31 liegt. Wir *könnten* kompliziertere Tests anstellen, um zu prüfen, daß der Wert nicht größer ist als 29, wenn es sich um den Februar handelt, und so weiter, bleiben hier aber bei den einfachsten Verfahren.

Unser erster Versuch könnte lauten:

```
5Ø  INPUT "GIB TAG DES MONATS EIN"; T
6Ø  IF T > Ø OR T < 32 THEN 2ØØ
7Ø  PRINT "GIBT ES NICHT"
8Ø  GOTO 5Ø
    . . .
2ØØ  REM HIER FUER GUELTIGEN TAG
```

Das Programm verhält sich nicht richtig, also könnten wir so umschreiben:

```
6Ø  PRINT "< 6Ø >": IF T > Ø OR T < 32 THEN 2ØØ
8Ø  PRINT "< 8Ø >": GOTO 5Ø
2ØØ  PRINT "< 2ØØ >": REM HIER FUER GUELTIGEN TAG
```

um den verdächtigen Code aufzuspüren.

Wenn wir das fahren, stellen wir fest, daß, gleichgültig, welcher Wert für T eingegeben wird, der Ablauf stets so erscheint:

```
6Ø
2ØØ
```

Zeile 7Ø kann nicht erreicht werden. Zeile 6Ø hätte *so* lauten müssen:

```
6Ø  IF T > Ø AND T < 32 THEN 2ØØ
```

Wenn Sie es sich genau überlegen, ist nämlich *jede* Zeile entweder größer als Ø oder kleiner als 32!

Verwechslungen von AND und OR kommen nicht selten vor, wenn man nicht genau nachdenkt. Schlampereien verzeiht der Computer aber nicht.

Programmprofile

Ein Programmprofil zeigt, wie oft jede Zeile eines Programms ausgeführt worden ist. Wie üblich ist das des Guten zuviel, und wir sollten uns genau überlegen, welche Teile eines Programms im Profil dargestellt werden müssen. Nehmen wir an, wir wollen in Erfahrung bringen, wie oft Zeile 42Ø eines bestimmten Programms ausgeführt wird. Wir setzen zu Beginn des Programms eine Zählung auf Ø und inkrementieren jedesmal um 1, wenn Zeile 42Ø durchlaufen wird:

```
  5  PZ = Ø
     . . .
42Ø  A = A * (P-1)
421  PZ = PZ + 1
     . . .
8Ø9  PRINT PZ
81Ø  STOP
```

Sehen wir uns ein konkretes Beispiel an. Das folgende Programm soll im Höchstfall 2Ø Werte, begrenzt durch Null, annehmen und sie in aufsteigender Reihenfolge ordnen. Lautet die Eingabefolge also:

```
3
8
1
4
2
Ø
```

sollte herauskommen:

```
1
2
3
```

4

8

Die Null sollte nicht erscheinen, weil sie nur ein Begrenzer ist.

```
10  DIM A(20)
20  FOR P = 1 TO 20
30  INPUT A(P)
40  IF A(P) = 0 THEN 60
50  NEXT P
60  N = P
65  F = 0
70  FOR P = 1 TO N
80  IF A(P) < A(P + 1) THEN 130
90  T = A(P)
100 A(P) = A(P + 1)
110 A(P + 1) = T
120 F = 1
130 NEXT P
140 IF F = 1 THEN 65
150 FOR P = 1 TO N
160 PRINT A(P)
170 NEXT P
```

Das Programm leistet nicht ganz, was es sollte. (Geben Sie ein und probieren Sie es aus.) Vielmehr gerät es in eine Endlosschleife.

Wo also mit dem Suchen anfangen? Die erste Schleife (20–50) sieht ganz harmlos aus, die letzte (150–170) zeigt lediglich den Arrayinhalt an. Demnach erscheint es angebracht, sich mit der Schleife von 70 bis 130 zu befassen. Aus Zeile 80 wird klar, daß manchmal alle Anweisungen in der Schleife ausgeführt werden, manchmal die von 90 bis 120 nicht beachtet werden. Wir nehmen also zwei Profilzählungen vor, Z1 und Z2. Sie zählen, wie oft in die Schleife eingetreten und wie oft der letzte Teil der Schleife ausgeführt wird.

Das können wir erreichen mit:

```
67  Z1 = 0
68  Z2 = 0
75  Z1 = Z1 + 1
```

```
125  Z2 = Z2 + 1
132  PRINT Z1, Z2
```

Weil wir schon dabei sind, können wir ebensogut den Inhalt des Arrays am Ende jeder Schleife anzeigen lassen; es liegt nahe, daß im Inneren Zahlen umgeschaufelt und nur sehr wenige andere Variable verwendet werden. Demnach:

```
134  FOR Q = 1 TO N                [weil 1 TO N der betroffene Arrayteil
                                    zu sein scheint)
135  PRINT A(Q);
136  NEXT Q
137  PRINT
138  FOR X = 1 TO 2ØØ: NEXT X      [Pause]
```

Probieren wir ein paar Datensätze aus, um zu sehen, was passiert. Wenn wir eingeben:

```
3
6
1
8
5
Ø
```

erhalten wir:

```
6     4
3165ØØ
6     4
13ØØ56
6     2
1ØØ356
6     2
ØØ1356
6     2
ØØ1356
```

```
6     1
ØØ1356
6     1
ØØ1356
```

und so weiter, bis unendlich.

Die Werte scheinen zwar in die Reihenfolge zu kommen, aber die '8' haben wir verloren, und wo kommt plötzlich das Nullenpaar her? Außerdem geht das Problem beharrlich sechsmal durch die Hauptschleife, aber die Durchläufe in der Unterschleife nehmen ständig ab, bis sie 1 erreichen, wo sie bleiben.

Eine der Nullen ist offenkundig der Begrenzer, die andere ein Element des Arrays, das nicht während des Laufs gesetzt, sondern durch das System auf Null initialisiert wird. Anders ausgedrückt: Das Programm behandelt zwei Werte zuviel. Schreiben wir die Zeile 6Ø also um:

```
6Ø  N = P-2
```

und versuchen wir es noch einmal. Die Hoffnung währet ewiglich . . . Wir erhalten:

```
4          2
3165
4          2
1356
4          Ø
1356
1
3
5
6
```

Immerhin ein gewisser Fortschritt – wir sind die Nullen losgeworden. Aber unsere 8 haben wir immer noch nicht wieder.

Schwer zu erkennen, wo sie verlorengegangen sein kann. Vielleicht ist sie noch da und wird nur nicht angezeigt. Wo zeigen wir sie mit PRINT an? In den Zeilen 15Ø–17Ø. Der Bereich 1 TO N muß zu klein sein. Erhöhen wir ihn um 1:

```
15Ø  FOR P = 1 TO N + 1
```

Und weil wir schon dabei sind: Die Überwachung in Zeile 134 wird wohl dieselben Probleme haben. Das beheben wir auch gleich:

```
134  FOR Q = 1 TO N + 1
```

Je nun, Freunde, laßt es uns noch einmal versuchen . . .

Diesmal erhalten wir (bei denselben Daten):

```
4    2
31658
4    2
13568
4    ∅
13568
1
3
5
6
8
```

Prima! Wir haben es geschafft. Jetzt klappt alles. Wirklich? Probieren wir:

```
3
5
2
1
5
∅
```

Diesmal erhalten wir:

```
4    3
32155
4    3
21355
4    2
12355
4    1
12355
4    1
12355
```

```
4    1
12355
etc.
```

Das Programm liefert die richtige Antwort, kommt aber nie aus der Schleife heraus. Uns fällt auf, daß Z2 in diesem Fall nie auf Null kommt. Man darf also davon ausgehen, daß dadurch das Programm abgeschlossen wird.

Was entscheidet, ob das Programm in die Unterschleife eintritt oder nicht? Zeile 8Ø:

```
8Ø  IF A(P) <A (P + 1) THEN 13Ø
```

Der Unterschied zwischen den beiden Datensätzen ist der, daß der zweite zwei identische Werte enthält. Da 5 nicht kleiner ist als 5, wird die Unterschleife jedesmal ausgeführt, sobald die beiden 5 auftauchen. Deshalb geht das Programm stets einmal durch die Unterschleife. Vielleicht sollte die Frage lauten:

```
8Ø  IF A(P) < = A(P + 1) THEN 13Ø
```

Diesmal funktioniert alles.

```
4    2
32155
4    2
21355
4    1
12355
4    Ø
12355
1
2
3
5
5
```

Jetzt läuft es wie geschmiert, und wir können die Testzeichen herausnehmen.

Ich hoffe, ich habe hier ein paar wichtige Punkte veranschaulichen können. Erstens mußten wir nicht genau wissen, wie die Prozedur verläuft. Wenn Sie diesen Abschnitt sorgfältig durchgearbeitet haben, wird sie inzwischen ziemlich klar sein, und ein paar Schreibtischtests würden Sie vermutlich davon überzeugen, daß Sie sie begriffen haben. (Schreibtischtests tragen sehr dazu bei, Computerprozeduren zu begreifen. Ich habe bei unklarem Code – von

fremder Hand, versteht sich – oft bis zu einem Dutzend Tests angestellt, bis mir richtig klar wurde, was sich abspielte.) Zweitens neigt man, wenn ein Programm das erstemal erfolgreich ist, stets zu dem Glauben, nun sei die Arbeit getan und man könne sich in der Eckkneipe rasch ein Glas gönnen. Wir wir gesehen haben, ist die Arbeit *nicht* getan, weil es andere Datensätze geben kann, bei denen das Programm versagt; abgesehen davon, hat die Kneipe schon vor eineinhalb Stunden zugemacht, falls Ihnen die Zeit beim Codeverfassen so schnell vergeht wie mir.

27 Datenlisten

Es gibt zweckmäßigere Methoden, eine Vielzahl von Variablen zu definieren, als sehr ähnlich geartete Programmzeilen in Mengen einzugeben.

Wenn Sie dieses Buch durchblättern, wird Ihnen auffallen, daß ein großer Teil der Programme lange Folgen von Zahlen oder Strings zu betreffen scheint. Oft sind diese in Arrays gespeichert. Kapitel 25 ist ein extremes Beispiel. Soweit die angebliche Stärke eines Computers darin besteht, ähnlich geartete Befehle *ohne* menschlichen Eingriff oft auszuführen, muß es gewiß einen besseren Weg geben, als diese fast gleich lautenden Zeilen vom Programmierer mühsam eintippen zu lassen. Die Zahlen und Strings freilich müssen auf *irgend*eine Weise eingegeben werden, weil der 64 keine Gedanken lesen kann.

Es gibt einen Weg. Er verwendet den Befehl

```
DATA
```

um die Zahlenliste aufzubauen, und

```
READ
```

um Zahlen gebrauchsbereit von der Liste zu holen.

Die Posten in einer DATA-Anweisung können sowohl Zahlen als auch Strings sein, und Sie dürfen die beiden nach Belieben mischen. Allerdings müssen Sie dafür sorgen, daß in jeder READ-Anweisung die richtige *Art* Variable verwendet wird. Wenn

```
READ X
```

auf einen String stößt, bewirkt es nichts; Sie müßten den String mit

```
READ X$
```

einer Stringvariablen zuteilen. Beispiel: Hier ein Programm, das den Beginn einer Liste bundesdeutscher Länder und ihrer Bevölkerungszahlen (in 1ØØØ) anzeigt:

```
1Ø  PRINT CHR$(147)
2Ø  DATA BAYERN, 1Ø644, BREMEN, 757, BERLIN, 213Ø,
    HESSEN, 5461
3Ø  FOR T = 1 TO 4
4Ø  READ L$, BEV
5Ø  PRINT L$, BEV
6Ø  NEXT T
```

Sie können, wenn Sie wollen, Strings in Anführungszeichen setzen oder auch nicht – in einer DATA-Liste spielt das keine Rolle. Man kann also auch "BREMEN" etc. nehmen. Ein String, der Kommas enthält, *muß* in Anführungsstrichen stehen.

Aufgabe 1

Hier ist eine Liste der ersten fünf Präsidenten der Vereinigten Staaten mit ihren Amtszeiten. Schreiben Sie ein Programm, das DATA-Listen verwendet, um die Information anzuzeigen.

George Washington	1789–1797
John Adams	1797–18Ø1
Thomas Jefferson	18Ø1–18Ø9
James Madison	18Ø9–1817
James Monroe	1817–1825

Planetensuche

Mit einer DATA-Liste können Sie mehr tun, als sie nur anzuzeigen! Beispielsweise können sie darin nach einem bestimmten Eintrag suchen:

```
1Ø  DATA MERKUR, 58, VENUS, 1Ø8
2Ø  DATA ERDE, 150, MARS, 228
3Ø  DATA JUPITER, 778, SATURN, 1427
4Ø  DATA URANUS, 287Ø, NEPTUN, 4997, PLUTO, 59ØØ
5Ø  INPUT "WELCHER PLANET"; PLANET$
6Ø  FOR G = 1 TO 9
7Ø  READ X$, Y
8Ø  IF X$ = PLANET$ THEN 1ØØ
9Ø  NEXT G
1ØØ  PRINT X$; "□IST VON DER SONNE"
11Ø  PRINT Y; "□MILLIONEN KILOMETER"
12Ø  PRINT "□ENTFERNT."
```

Zeilen 1Ø–4Ø führen die Daten auf. Auf den Namen des Planeten folgt seine Entfernung (in Millionen Kilometer) zur Sonne. Die Schleife von Zeile 5Ø bis 9Ø sucht die Datenliste nach einem bestimmten Planeten ab, dessen Namen sie in Zeile 5Ø eingeben. Wenn er gefunden ist, geht das Programm zu Zeilen 1ØØ–12Ø, die das Gewünschte an Information anzeigen.

(Zur Beachtung: Sie müssen PLANET$ *genauso* eingeben, wie aufgeführt, also keine zusätzlichen Leerstellen etc., sonst verläßt das Programm die Schleife, ohne Ihre Eingabe entdeckt zu haben, und zeigt die Daten für Pluto an. (Warum gerade für Pluto?)

Wenn Sie ins Reisebüro oder zum Flugschalter gehen und man Ihren Namen eingibt, um zu prüfen, ob Ihr Flug gebucht ist, verwendet man eine (kompliziertere) Version dieser Art von Datensuche. Diese Datenliste ist natürlich *riesengroß,* und der Hauptaufwand an Programmierkönnen dient der narrensicheren Verarbeitung enormer Datenmengen.

RESTORE

Manchmal wollen Sie vielleicht in einem Programm dieselbe Datenliste mehrmals verwenden. Hier ein sehr simpler Fall. Was geht vor?

```
10 DATA FRED
20 READ X$
30 PRINT X$
40 GOTO 20
```

Das erste FRED wird angezeigt – dann kommt die Meldung OUT OF DATA. Der Computer weiß nicht, daß er die ursprüngliche Datenliste wiederverwenden soll.

Um ihm das zu sagen, verwenden Sie das Wort

```
RESTORE
```

Fügen Sie eine Zeile

```
35 RESTORE
```

an, dann können sie es sehen. RESTORE schickt den 'Zeiger' zur laufenden Position in der Datenliste zurück, an den Anfang.

Lösung

Aufgabe 1

```
10 DATA GEORGE WASHINGTON, 1789, 1797,
   JOHN ADAMS, 1797, 1801
20 DATA THOMAS JEFFERSON, 1801, 1809,
   JAMES MADISON, 1809, 1817
30 DATA JAMES MONROE, 1817, 1825
```

```
4Ø  PRINT CHR$(147)
5Ø  FOR N = 1 TO 5
6Ø  READ NAME$, D1, D2
7Ø  PRINT NAME$, D1; "BIS"; D2
8Ø  NEXT N
```

28 Sprites

Eine anregende und ungewöhnliche Eigenschaft des Commodore 64 ist seine Fähigkeit, überall auf den Bildschirm große Grafikblöcke zu zeichnen und zu bewegen. Sie können sich überlappen, Zusammenstöße lassen sich feststellen.

Sprites oder MOBs (Moveable Object Blocks, dt. etwa Bewegliche Blockobjekte) sind mittelgroße Grafikgebilde, die von einem eigenen VIP-Chip verarbeitet werden. Man kann sie nach Wunsch des Programmierers auf dem Bildschirm bewegen. Sie können zur Grundlage vieler hübscher Spiele und Displays gemacht werden. Ganz einfach ist der Umgang mit ihnen aber nicht. Dieses Kapitel hat zum Ziel, einige der Grundgedanken darzustellen, jedenfalls soviel, daß Sie selbst Sprites einsetzen können.

Spriteaufbau

Die Information, die ein Sprite definiert, besteht aus einem 21 x 24 Punkte großen Gitter, dessen Zellen entweder leer oder geschwärzt sind. Beispielsweise zeigt Abbildung 28.1 ein Sprite in Form eines 'Raumkreuzers'.

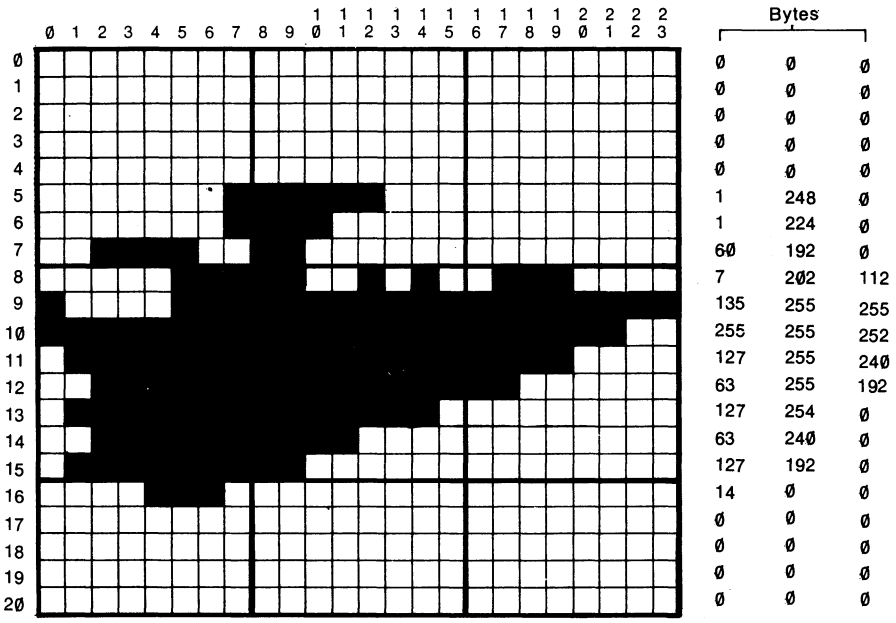


Abbildung 28.1
Raumkreuzer auf einem Spritegitter, und die entsprechenden Daten, von Binär in Dezimal umgewandelt.

Diese leeren oder ausgefüllten Zellen müssen in eine Folge von Zahlen umgewandelt und an der entsprechenden Stelle gespeichert werden (siehe unten). Dazu ersetzt man jede leere Zelle durch eine Ø, jede volle Zelle durch eine 1, wie in Abbildung 28.1. Teilen Sie jede Reihe von 24 Ziffern in drei Abschnitte zu je 8 Ziffern auf. Beispielsweise zerfällt Reihe 8 der Abbildung folgendermaßen:

ØØØØØ111 11ØØ1Ø1Ø Ø111ØØØØ

Sie sehen aus wie Binärbytes . . . und genau das ist der Sinn. Umgewandelt in Dezimalzahlen werden daraus

7 2Ø2 112

Man kann sich jede Reihe des Sprites demnach als eine Folge von drei Dezimalzahlen (zwischen Ø und 255) vorstellen. Die Zahlen für das gesamte Sprite sind neben Abbildung 28.1 aufgeführt, und gelesen werden sie der Reihe nach von oben links nach unten rechts, also die drei Bytes für Reihe Ø, dann die drei für Reihe 1 und so weiter bis Reihe 2Ø. Das ergibt insgesamt 63 Zahlen.

Sie *können* Ihr Sprite auf kariertem Papier entwerfen und die Zahlen aus der Tabelle in Anhang 1 entnehmen, aber wäre es nicht viel hübscher, wenn man die ganze Arbeit dem Computer aufhalsen könnte?

Computerunterstützte Spritekonstruktionen

Hier ein ziemlich einfaches Programm. Damit können Sie auf dem Bildschirm ein Sprite zeichnen und die Datenliste generieren. Um das Listing in Grenzen zu halten, sind verschiedene mögliche Verbesserungen weggelassen worden. Falls Sie den Wunsch haben, das aufzumöbeln, bitte gern!

```
1Ø  POKE 5328Ø, 4
2Ø  PRINT CHR$(147);
3Ø  FOR S = Ø TO 2Ø
4Ø  IF S = 8 * INT(S/8) THEN PRINT
    "------------------------"      [24 – Zeichen]
5Ø  IF S < > 8 * INT(S/8) THEN PRINT
    ".......:.......:.......:"
6Ø  NEXT S

1ØØ DIM S(2Ø, 23)
11Ø FOR R = Ø TO 2Ø
12Ø FOR C = Ø TO 23
13Ø CDE = 63: GOSUB 1ØØØ
14Ø GET A$
```

```
15Ø  IF A$ < > "Ø" AND A$ < > "1" THEN 14Ø
16Ø  IF A$ = "Ø" THEN S(R,C) = Ø: CDE = 32: GOSUB 1ØØØ
17Ø  IF A$ = "1" THEN S(R,C) = 1: CDE = 1Ø2: GOSUB 1ØØØ
18Ø  NEXT C
19Ø  NEXT R
2ØØ  POKE 5328Ø, 3
21Ø  GET A$: IF A$ < > "N" AND A$ < > "J" THEN 21Ø
22Ø  IF A$ = "N" THEN POKE 5328Ø, 4: GOTO 11Ø
25Ø  PRINT CHR$(19);
26Ø  FOR R = Ø TO 2Ø
27Ø  FOR X = Ø TO 16 STEP 8
28Ø  V = Ø
29Ø  FOR C = Ø TO 7
3ØØ  IF S(R, X + C) = 1 THEN V = V + 2 ↑ (7-C)
31Ø  NEXT C
32Ø  PRINT TAB(24 + X/2); V;
33Ø  NEXT X
34Ø  PRINT
35Ø  NEXT R
36Ø  GOTO 36Ø
1ØØØ  REM PRINT AT
1Ø1Ø  POKE 1Ø24 + 4Ø * R + C, CDE
1Ø2Ø  POKE 55296 + 4Ø * R + C, 3
1Ø3Ø  RETURN
```

Fahren Sie das mit RUN. Der Rand wird pupurrot, aus Gründen, die gleich klarwerden. Sie erhalten ein 21 x 24 großes Gitter aus Punkten und Strichen, in Abschnitte 8 x 8 aufgeteilt, damit es praktischer ist. Oben links steht ein ?-Zeichen. Wenn Sie '1' drücken, wird es ersetzt durch ein kariertes Muster, bei 'Ø' durch eine Leerstelle. Dann bewegt es sich um eine Stelle vorwärts. So können sie weitermachen und einen Block oder eine Leerstelle plotten, bis das ganze Gitter ausgefüllt ist.

An diesem Punkt wird der Rand cyanfarben, um Sie daran zu erinnern, daß Sie eine Taste drücken müssen. (Für eine Meldung bleibt nicht viel Platz, deshalb ist das ein günstiger Ausweg.) 'J' für 'Ja' teilt dem Programm mit, daß es fortfahren soll; 'N' für 'Nein' bedeutet, daß Sie einen Fehler gemacht haben

und es noch einmal versuchen wollen. (Beim zweiten Lauf müssen sie alle Ø und 1 noch einmal eingaben; hier wäre eine Verbesserung möglich.)

Der Computer listet dann automatisch an der rechten Seite die Daten für die Reihen auf. Notieren Sie sich die auf einem Blatt Papier. (Oder lassen Sie sie vom Drucker ausdrucken oder kopieren Sie in eine Datei auf Kassette, siehe Kapitel 34.)

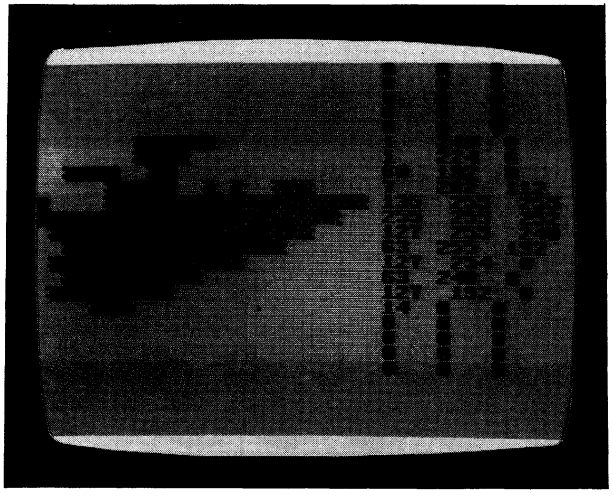


Abbildung 28.2
Das Sprite-Aufbauprogramm in Aktion.

Die Sprite-Register

Für die Sprites sind eigene Speicherbereiche reserviert. Die Adressen beginnen bei 53248 (was ich aus praktischen Erwägungen von jetzt an V nenne) und enden bei 53294. Nicht alle sind für einen Anfänger von Nutzen, und ich lasse die ausgefalleneren weg. Zusätzlich gibt es in den Adressen 2Ø4Ø–2Ø47 noch mehrere *Zeiger,* die dem Computer mitteilen, wo er die 63 Bytes Grafikdaten suchen soll, die zur Definierung jedes Sprites gebraucht werden. Ich beschreibe sie gleich ausführlicher, hier zunächst ein rascher Überblick.

Spritepositionen: Adressen V bis V + 15 enthalten Spaltennummer (oder x-Koordinate in hochauflösender Grafik) und Reihennummer (Y-Koordinate) für jedes der acht Sprites. Diese Zahlen gehen von Ø–255. Jede ist als ein Byte in einer Einzeladresse gespeichert.

Versetzungsflagge: Die acht Bits eines Einzelbytes in Adresse V + 16 definieren eine Versetzung an der X-Koordinate (Spaltennummer) nach rechts. Ist Bit K auf 1 gesetzt, wird die Spaltennummer um 256 erhöht. Das ist erforderlich, um Sprites zur rechten Bildschirmseite zu befördern.

Aktivieren/Sperren: Die acht Bits eines Einzelbytes in Adresse V + 21 aktivieren das Sprite K (schalten es ein), wenn Bit K auf 1 gesetzt ist, und sperren (schalten es ab), wenn Bit K Ø ist.

Vertikal erweitern: Die acht Bits eines Einzelbytes in Adresse V + 23 erweitern das Sprite K auf doppelte Höhe, wenn Bit K 1 ist.

Horizontal erweitern: Ebenso erweitern die acht Bits in Adresse V + 29 Sprite K auf doppelte Breite, wenn Bit K 1 ist.

Kollisionsflagge: Wenn zwei Sprites 'zusammenstoßen', werden die entsprechenden Bits in diesem Register auf 1 gesetzt.

Farben: Jede Adresse V + 39 bis V + 46 enthält den Farbcode (wie in Kapitel 13 von Ø bis 15) für ein Sprite.

Datenzeiger: Adressen 2Ø4Ø bis 2Ø47 (oberes Ende des Farb-RAM) enthalten Zeiger zu den Startadressen der Daten für die Sprites Ø bis 7 der Reihe nach. Wenn der Zeiger K den Wert PTR hat, beginnt die Adresse für die Daten bei 64*PTR. Wir wollen das den PTR-Speicher*block* nennen, von 64*PTR bis 64*PTR+63. Damit können Sie überall in den ersten 16 348 Bytes RAM Sprites definieren. Es gibt Methoden, auch die anderen 49 512 Bytes zu verwenden, aber sie sind ziemlich umständlich, siehe 'Reference Guide', S. 1Ø1 und 133. Sie können aber nicht einfach Sprites in jede beliebige Adresse setzen; das BASIC-System demoliert die Daten. Siehe unten die empfohlenen Adressen.)

Die Adressen für die Steuerung von Sprites sind zusammengefaßt in den Tabellen 28.1 und 28.2, zur besseren Bequemlichkeit als Anhang 2 wiederholt. Zur Bedeutung der weggelassenen Adressen siehe 'Reference Guide', S. 131–181. Das sind 5Ø Seiten; ich sage ja, mit Sprites ist es nicht ganz so einfach!

Tabelle 28.1 Sprite-Datenträger

Adesse	Inhalt
2Ø4Ø	Datenzeiger Sprite Ø
2Ø41	Datenzeiger Sprite 1
2Ø42	Datenzeiger Sprite 2
2Ø43	Datenzeiger Sprite 3
2Ø44	Datenzeiger Sprite 4
2Ø45	Datenzeiger Sprite 5
2Ø46	Datenzeiger Sprite 6
2Ø47	Datenzeiger Sprite 7

Tabelle 28.2 Sprite-Register

V = 53248 Ø = Startadresse des Registerbereichs

Adresse	Inhalt								Funktion
V + Ø	Spaltennummer Sprite Ø								Spritepositionen
V + 1	Reihennummer Sprite Ø								
V + 2	Spaltennummer Sprite 1								
V + 3	Reihennummer Sprite 1								
V + 4	Spaltennummer Sprite 2								
V + 5	Reihennummer Sprite 2								
V + 6	Spaltennummer Sprite 3								
V + 7	Reihennummer Sprite 3								
V + 8	Spaltennummer Sprite 4								
V + 9	Reihennummer Sprite 4								
V + 1Ø	Spaltennummer Sprite 5								
V + 11	Reihennummer Sprite 5								
V + 12	Spaltennummer Sprite 6								
V + 13	Reihennummer Sprite 6								
V + 14	Spaltennummer Sprite 7								
V + 15	Reihennummer Sprite 7								
V + 16	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	Versetzungsflagge
V + 21	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	Aktivieren/Sperren
V + 23	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	erweitern vertikal
V + 29	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	erweitern horizontal
V + 3Ø	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	Kollisionsflagge
V + 39	Farbcode Sprite Ø								Farben
V + 4Ø	Farbcode Sprite 1								
V + 41	Farbcode Sprite 2								
V + 42	Farbcode Sprite 3								
V + 43	Farbcode Sprite 4								
V + 44	Farbcode Sprite 5								
V + 45	Farbcode Sprite 6								
V + 46	Farbcode Sprite 7								

Sprite-Koordinaten

Sprite-Reihen- und Spaltennummern können in den folgenden Bereichen liegen:

Reihe: Ø–255

Spalte: Ø–511 (Versetzungsflagge eingeschlossen)

Das ist viel größer als der Bildschirmbereich, der 2ØØ Reihen und 32Ø Spalten umfaßt. Der 64 nutzt das und läßt zu, daß sie Sprites außerhalb des Bildschirms in Position bringen und unbehindert weiterbewegen. Die Beziehung zwischen Sprite-Koordinaten und Bildschirm ist Abbildung 28.3 zu entnehmen.

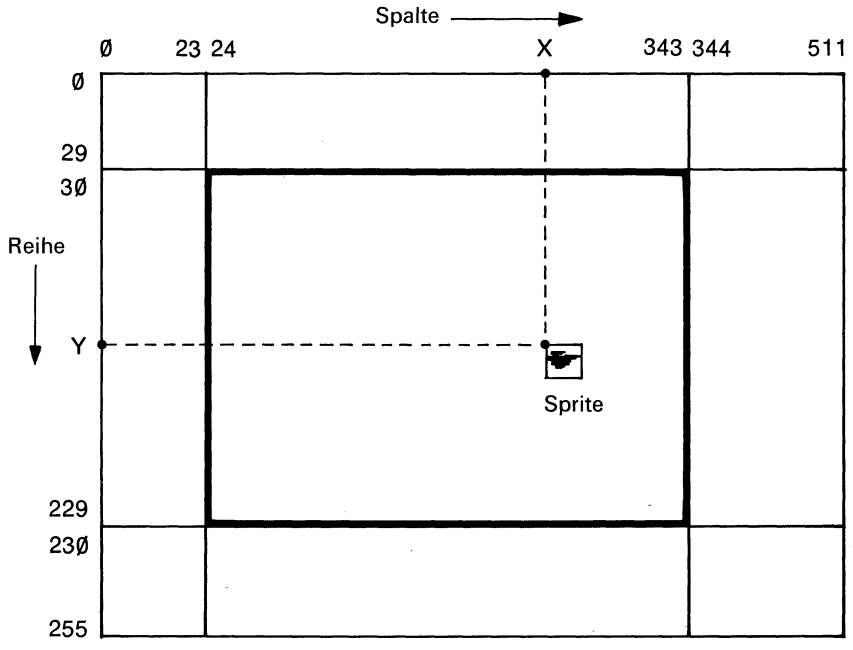


Abbildung 28.3
Koordinatengitter für Sprite.

Beachten Sie, daß Reihen- und Spaltennummern eines Sprites die Position der oberen linken Ecke des 21 x 24-*Gitters* sind, das es definiert. In Programmen müssen Sie das berücksichtigen. Bei erweiterten Sprites bleibt es die linke obere Gitterecke.

Ein Sprite aufbauen

Um ein Sprite auf den Bildschirm zu bekommen, müssen wir:

1. Die Daten für seine Form definieren.
2. Den Zeiger dorthin setzen, wo die Daten gespeichert sind.
3. Die Daten mit POKE an Ort und Stelle bringen.
4. Das Sprite aktivieren.
5. Die Farbe des Sprites bestimmen.
6. Reihen- und Spaltennummern für das Sprite einsetzen

Nehmen wir das Raumkreuzer-Sprite von weiter oben und bauen wir es als Sprite 1 auf. Dazu brauchen wir:

```
 1Ø  V = 53248
1ØØ  DATA Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø
11Ø  DATA 1, 248, Ø, 1, 224, Ø, 6Ø, 192, Ø, 7, 2Ø2, 112, 135, 255, 255
12Ø  DATA 255, 255, 252, 127, 255, 24Ø, 63, 255, 192, 127, 254, Ø
13Ø  DATA 63, 24Ø, Ø, 127, 192, Ø, 14, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø
14Ø  DATA Ø, Ø, Ø, Ø, Ø, Ø
2ØØ  POKE 2Ø41, 13          [Sprite 1-Zeiger auf 13. Block]
21Ø  FOR G = Ø TO 62
22Ø  READ H                 [lies Daten]
23Ø  POKE 832 + G, H        [mit POKE in Block:
                            beachten 832 = 64*13]
24Ø  NEXT G
25Ø  POKE V + 21, 2         [Sprite 1 aktivieren]
26Ø  POKE V + 4Ø, 7         [Sprite 1 gelb]
27Ø  POKE V + 2, 1ØØ        [Sprite in Spalte 1ØØ]
28Ø  POKE V + 3, 1ØØ        [Sprite 1 in Reihe 1ØØ]
```

Geben Sie das *sorgfältig* ein und fahren sie mit RUN. Sie sollten, wie gewünscht, den Raumkreuzer in Gelb sehen.

Sie können mit der Veränderung der Positionen durch direkte Befehle experimentieren.

```
POKE V + 2, 11Ø
```

bewegt es nach rechts,

```
POKE V + 3, 9Ø
```

nach oben; durch

```
POKE V + 4Ø, 5
```

wird es grün. Versuchen Sie andere Werte und sehen Sie sich an, was geschieht.

Bewegung

Um das Sprite zu bewegen, brauchen wir nur laufend die Reihen- und Spaltennummern zu verändern. Beispiel: *Löschen* Sie 27Ø und 28Ø oben und fügen Sie die Schleife an:

```
1ØØØ  X = Ø: Y = 7Ø
1Ø1Ø  POKE V + 2, X: POKE V + 3, Y
1Ø2Ø  X = X + 3: IF X > 255 THEN X = Ø
11ØØ  GOTO 1Ø1Ø
```

Der Raumkreuzer bewegt sich jetzt wiederholt von links nach rechts über den Bildschirm. Beachten Sie, daß er nicht die ganze Strecke zurücklegt; das liegt daran, daß wir dazu die Spaltennummer größer setzen müssen als 255. Siehe weiter unten, weil hier eine kleine Komplikation auftritt.

Erweiterung

Für einen Raumkreuzer ist das Ding ja wirklich ein bißchen kurz und plump. Keine Sorge, wir können ihn horizontal dehnen durch die zusätzliche Zeile:

```
29Ø  POKE V + 29, 2
```

Das erweitert Sprite 1, aber keine anderen, weil 2 in Binär = ØØØØØØ1Ø ist, also nur Bit 1 auf 1 gesetzt wird. Sieht schon besser aus, nicht?

Um ihn auch noch vertikal auszudehnen, fügen Sie hinzu:

```
3ØØ  POKE V + 23, 2
```

Die Erweiterungen verdoppeln die Größe in der jeweiligen Richtung. Beachten Sie, wie die Grafik gröber wird, wenn Sie das tun. Sie wird zusammengesetzt aus denselben Daten, aber mit Pixelblöcken 2 x 1, 1 x 2 oder 2 x 2.

Ich mag *große,* dicke Raumkreuzer nicht, also löschen wir 3ØØ wieder.

Nur zum Spaß, um Ihnen die Art von Effekt zu zeigen, den Sie anstreben können, fügen Sie das der Bewegungsschleife an:

```
1Ø3Ø  POKE V + 29,2*INT (2*RND(Ø))
1Ø4Ø  POKE V + 23,2*INT (2*RND(Ø))
```

Nun verändert er die Form in zufälliger Weise ganz rasch. Wenn Sie auch noch die Farbe verändern wollen, schreiben Sie dazu:

```
1Ø5Ø  POKE V + 4Ø, 16*RND(Ø)
```

Lustig . . . aber es lenkt ab, also löschen wir 1Ø3Ø–1Ø5Ø wieder.

Keyboard-Steuerung

Wenn wir innerhalb der Schleife ein GET verwenden, können wir die Bewegung vom Keyboard aus beeinflussen.

```
1Ø3Ø  GET A$
1Ø4Ø  IF A$ = "Ø" THEN Y = Y-4
1Ø5Ø  IF A$ = "U" THEN Y = Y + 4
```

Nun geht der Raumkreuzer am Bildschirm nach oben, wenn Sie 'O', und nach unten, wenn Sie 'U' drücken. Offensichtlich könnten viel kompliziertere Positionsveränderungen bewirkt werden, aber das veranschaulicht die Möglichkeiten.

Um die Vorgänge zu beschleunigen, ersetzen Sie das X + 3 in 1Ø2Ø durch X + 6 oder X + 1Ø (schnellere Bewegung verläuft auch ruckhafter):

```
1Ø2Ø  X = X + 6: IF X > 255 THEN X = Ø
```

Bewegung über den ganzen Bildschirm

Es wird Zeit, sich mit dem Problem zu befassen, wie das Sprite auf der rechten Seite des Bildschirms in Position gebracht wird, über Spalte 255 hinaus.

Hier wird die Versetzungsflagge in V + 16 genutzt. Wenn Bit K davon 1 ist, wird die Spaltennummer von Sprite K um 256 erhöht. Wir schreiben 1Ø1Ø also folgendermaßen um:

```
1Ø1Ø  POKE V + 3, Y
1Ø11  IF X > 255 THEN OF = 1
1Ø12  IF OF = Ø THEN POKE V + 16, (PEEK V + 16)) AND 253
1Ø13  IF OF = 1 THEN POKE V + 16, (PEEK (V + 16)) OR 2
1Ø14  POKE V + 2, X-OF*256
1Ø15  OF = Ø
```

Nehmen Sie diese Veränderung vor und schreiben Sie Zeile 1Ø2Ø um:

```
1Ø2Ø  X = X + 6: IF X > 35Ø THEN X = Ø
```

Nun befährt der Raumkreuzer auf seinen Reisen den ganzen Bildschirm.

Sprite-Priorität

Wenn sich zwei Sprites überschneiden, scheint das mit der kleinsten Nummer dem anderen aufzuliegen. Das Sprite 'darunter' ist aber durch alle 'Löcher' im oberen sichtbar, so, wie man es auch im richtigen leben erwarten würde.

Um das auszuprobieren, will ich ein zweites Sprite aufbauen. Dieselbe Routine (zunächst ist das gut für die Übung, aber später will ich einen besseren Weg empfehlen, wenn Sie *viele* Sprites verwenden möchten).

```
400  DATA 0, 255, 0, 3, 255, 192, 15, 195, 240, 63, 0, 252, 255, 0, 255
410  DATA 63, 0, 252, 127, 195, 254, 31, 255, 248, 3, 255, 192
420  DATA 0, 255, 0, 0, 195, 0, 1, 129, 128, 3, 0, 192, 6, 0, 96
430  DATA 15, 0, 240, 15, 0, 240, 7, 129, 224, 3, 195, 192
440  DATA 1, 231, 128, 0, 0, 0, 31, 255, 248

500  POKE 2040, 14          [Sprite 0-Zeiger zum 14. Block]
510  FOR G = 0 TO 62
520  READ H
530  POKE 896 + G, H        [Block 14: 896 = 64*14]
540  NEXT G
```

Um Sprite 0 ebenso zu aktivieren wie Sprite 1, müssen wir Zeile 250 oben umschreiben zu:

```
250  POKE V + 21, 3
```

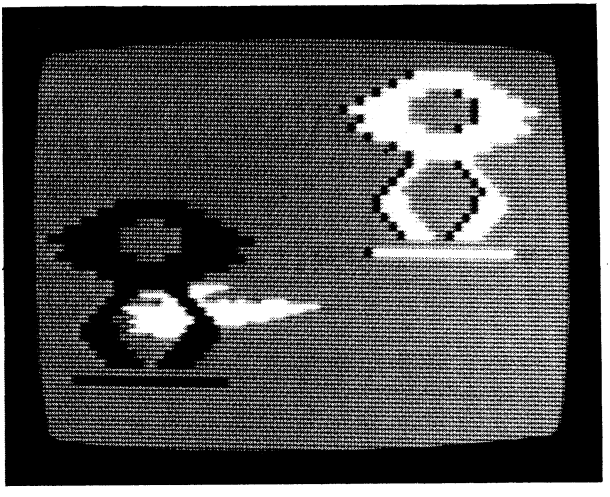


Abbildung 28.4
Sprite-Priorität: Eines liegt 'auf' dem anderen.

weil 3 in Binär ØØØØØØ11 ist, die Bits 1 und Ø also auf 1 gesetzt sind. Nun fahren wir fort:

```
56Ø  POKE V + 39, 5      [Sprite grün]
57Ø  POKE V, 120         [Sprite Ø in Spalte 12Ø]
58Ø  POKE V + 1, 95      [Sprite Ø in Reihe 95]
59Ø  POKE V + 29, 3      [Sprites Ø, 1 horizontal erweitern]
6ØØ  POKE V + 23, 1      [Sprite Ø vertikal erweitern]
```

Nun RUN und mit den Tasten 'O' und 'U' den Raumkreuzer fliegen lassen im – na ja, All. Sehen Sie, wie er sich dahinterzuschieben scheint? Das liegt daran, das das grüne Gebilde (Sprite Ø) Priorität gegenüber dem Kreuzer hat (Sprite 1).

Angenommen, wir möchten diese Priorität ändern. Dann können wir das Grüne Gebilde zu Sprite 2, nicht Ø machen. Das erfordert folgende Änderung:

```
5ØØ  POKE 2Ø42, 14
25Ø  POKE V + 21, 6      [6 = ØØØØØ11Ø]
56Ø  POKE V + 41, 5
57Ø  POKE V + 4, 12Ø
58Ø  POKE V + 5, 95
59Ø  POKE V + 29, 6
6ØØ  POKE V + 23, 4
```

Probieren Sie es – der Raumkreuzer schiebt sich jetzt davor, nicht dahinter.

Verwendung derselben Daten für mehrere Sprites

Wir können mehr als ein Sprite dadurch auf dieselben Daten setzen, daß wir zwei oder mehr Zeiger auf gleichen Wert festlegen. Nehmen wir an, wir haben Sprite 1 und 2 wie oben aufgebaut, wollen aber nun, daß Sprite Ø ein Schwarzes Gebilde (ebenfalls doppelter Größe) an einer anderen Stelle sei. Das können wir. Wir aktivieren alle drei Sprites, indem wir 25Ø erneut umschreiben:

```
25Ø  POKE V + 21, 7      [7 = ØØØØØ111]
```

Nun Sprite Ø aufbauen:

```
7ØØ  POKE 2Ø4Ø, 14       [Daten für Sprite Ø vom selben Block, 14]
                         [Sprite Ø schwarz]
77Ø  POKE V, 7Ø          Sprite Ø in Spalte 7Ø
```

```
78Ø  POKE V + 1, 124      Sprite Ø in Reihe 124
79Ø  POKE V + 29, 7       alle 3 Sprites horizontal gedehnt
8ØØ  POKE V + 23, 5       nur Ø und 2 horizontal
```

Wenn Sie jetzt mit RUN fahren, finden Sie zwei Gebilde und einen Raumkreuzer.

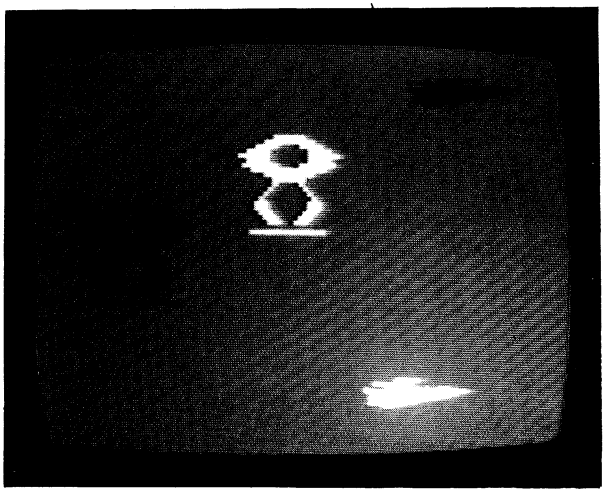


Abbildung 28.5
Dieselben Daten für ein zusätzliches Sprite

Aufgabe 1

Schreiben Sie das Programm so um, daß Sprite 3 als zweiter Raumkreuzer von derselben Form, aber in Rot, erscheint; er soll mit dem gelben auf gleicher Höhe fliegen, aber 2Ø Reihen tiefer auf dem Bildschirm

Kollisionserkennung

Jetzt wollen wir herauszufinden versuchen, ob es einen Zusammenstoß mit Sprite 2 gibt. Die Zusammenstöße werden erkannt durch die Flaggen im Register V + 3Ø. Wenn zwei Sprites zusammenstoßen, werden diese beiden Bits auf 1 gesetzt. Beispiel: Wenn Sprite 1 und 2 zusammenstoßen, enthält V + 3Ø

ØØØØ11Ø = 6

Der Wert wird bei jedem Zusammenstoß aktualisiert. Wenn Sie mit PEEK in diese Adresse hineingehen, wird der Inhalt automatisch auf Ø zurückgesetzt.

Um zu sehen, ob Sprite 2 ein anderes Sprite gerammt hat, müssen wir Bit 2 des Registers prüfen; das geschieht durch den Befehl

IF (PEEK (V + 3Ø) AND 4) = 4 THEN hat es gekracht . . .

Wir können einen Kollisionstest bei Zeile 1ØØØ so in die Schleife setzen:

```
1Ø25  GOSUB 2ØØØ
2ØØØ  REM KOLLISIONSTEST FUER SPRITE 2
2Ø1Ø  IF (PEEK (V+3Ø) AND 4) = 4 THEN POKE 5328Ø, X-7*INT (X/7)
2Ø2Ø  RETURN
```

Dadurch blinkt der Rand (wieso?), wenn der Zusammenstoß eintritt.

Fahren Sie das Programm und überprüfen Sie, daß der Rand blinkt, wenn Sprite 1 mit Sprite 2 zusammenstößt.

Sie können sich sogar damit amüsieren, dauernd mit PEEK in V + 3Ø hineinzusehen, um festzustellen, was dort los ist:

```
2Ø15 PRINT PEEK (V + 3Ø)
```

Das trägt dazu bei, noch einen wichtigen Punkt darzulegen: Das Sprite-Display ist von dem üblichen Text auf dem Bildschirm ganz unabhängig, und Sie können beides gleichzeitig haben. Sie werden eine Kolonne von Zahlen sehen, in der Regel Ø, aber die Zahlen wechseln, so oft zwei Sprites sich berühren. Vergewissern Sie sich, daß ihr Bitmuster dem richtigen Spritepaar entspricht:

3 = ØØØØØØ11, wenn Sprite 1 mit Sprite Ø zusammenstößt

6 = ØØØØ11Ø, wenn Sprite 1 mit Sprite 2 zusammenstößt

Aufgabe 2

Welche Zahl erscheint in V + 3Ø, wenn:

1. Sprite 5 mit Sprite 7 zusammenstößt?
2. Sprite 2 mit Sprite 4 zusammenstößt?
3. Sprite 6 mit Sprite 3 zusammenstößt?

Aufgabe 3

Versuchen Sie das V + 3Ø in Zeilen 2Ø1Ø und 2Ø15 zu V + 31 abzuändern. Der Rand blinkt ab und zu immer noch. Wann? Was geht da vor?

Das ist erst der Anfang

Das ist ein langes Kapitel gewesen, dabei haben wir die Oberfläche kaum angekratzt. Sie können beispielsweise vielfarbige Sprites erzeugen. Der Platz geht aber zu Ende, und ich hoffe, Sie haben genug Anregungen erhalten, um auch so genug beschäftigt zu sein. Sobald Sie das beherrschen, was ich Ihnen über den Umgang mit Sprites erzählt habe, können Sie zusätzliche Möglichkeiten im 'Reference Guide' erkunden.

Verbesserungen

Statt jedes Sprite 'von Hand' aufzubauen, erscheint es Ihnen vielleicht besser, eine Folge von Subroutinen für den Umgang mit Sprites zu schreiben und diese zu verwenden. Anhang 3 führt ein paar Grundprogramme auf.

Wo man Sprite-Daten speichert

Für drei oder weniger Sprites können Sie Blocks 13, 14 und 15 verwenden. Sie liegen im *Kassettenpuffer,* einem Speicherbereich, der nur benützt wird, wenn der Kassettenrecorder angeschlossen ist. Dort kann man Sprites also ungefährdet speichern. Leider ist er nicht lang genug, um alle acht 64 Byte-Blocks aufzunehmen. Wenn Sie nicht ein sehr langes BASIC-Programm haben, empfiehlt der 'Reference Guide' die Blöcke 192–199. Wenn Sie mehr wissen möchten, sollten Sie sich auch hier an den 'Reference Guide' halten.

Grand Prix

Das ist alles graue Theorie. Aus diesem Grund hier ein verhältnismäßig einfaches, aber vollständiges Spielprogramm, das Umgang mit Sprites beinhaltet. Es gehört zu den gängigen Erscheinungen des Marktes: Auf einer Rennbahn steht ein Auto, die Rennbahn rollt vorbei. Hier das Listing – es nutzt mehrere Kniffe, die wir uns in vorangegangenen Kapiteln angeeignet haben.

```
1Ø  PRINT CHR$(147)
2Ø  PRINT "GRAND PRIX"
3Ø  GOSUB 1ØØØ: REM SPRITE AUFBAUEN
4Ø  PRINT
5Ø  INPUT "STUFE: 1 BIS 5"; D
6Ø  IF D < 1 OR D > 5 THEN 5Ø
7Ø  L = 9 + D: R = 31-D
8Ø  PRINT CHR$(147)
9Ø  FOR T = 1 TO 15
```

```
100  PRINT TAB (L); "gWc g+c";
     TAB (R);   "g+c gQc"              [Rand der Straße]
110  NEXT T
120  TI$ = "000000"                    [Uhr stellen]
130  POKE V + 31, 0
200  PRINT TAB (L); "gWc g + c";
     TAB (R); "g + c gQc"              [neuer Straßenrand]
210  Q = INT (3*RND(0))-1
220  IF L + Q < 0 OR R + Q > 38 THEN Q = 0
230  L = L + Q: R = R + Q
240  GOSUB 2000: REM LIES KEYBOARD
250  GOTO 200
1000 REM SPRITE
1010 V = 53248
1020 DATA 0, 126, 0, 0, 255, 0, 49, 255, 140, 49, 255, 140
1030 DATA 63, 255, 252, 49, 255, 204, 49, 255, 204, 3, 255, 192
1040 DATA 3, 255, 192, 3, 255, 192, 3, 255, 192, 3, 255, 192
1050 DATA 3, 255, 192, 3, 255, 192, 49, 255, 140, 49, 255, 140
1060 DATA 63, 255, 252, 49, 255, 140, 48, 255, 12, 0, 126, 0
1070 DATA 0, 24, 0
1080 POKE 2041, 13                     [Sprite 1 in Block 13]
1090 FOR G = 0 TO 62
1100 READ H
1110 POKE 832 + G, H                   [Sprite in Block 13 setzen]
1120 NEXT G
1130 POKE V + 21, 2                    [Sprite 1 aktivieren]
1140 POKE V + 40, 7                    [gelb färben]
1150 POKE V + 23, 2                    [vertikal erweitern]
1160 POKE V + 29, 2                    [horizontal erweitern]
1170 X = 168                           [horizontale Koordinate]
1180 POKE V + 2, X: POKE V + 3, 100    [in Position bringen]
1190 RETURN
```

```
2000  REM LIES KEYBOARD
2010  P = PEEK (197)                    [zuletzt gedrückte Taste]
2020  IF P = 47 THEN X = X-3            [nach links, wenn < gedr.]
2030  IF P = 44 THEN X = X + 3          [nach rechts, wenn > gedr.]
2040  GOSUB 3000: REM SPRITE BEWEGEN
2050  IF (PEEK (V + 31) AND 2) =
      2 THEN 4000                       [Unfalltest]
2060  RETURN

3000  REM SPRITE SEITWAERTS BEWEGEN
3010  IF X > 255 THEN OF = 1
3020  POKE V + 2, X-256*OF
3030  IF OF = 0 THEN POKE V + 16, 0
3040  IF OF = 1 THEN POKE V + 16, 2
3050  OF = 0
3060  RETURN
4000  REM FINISH
4010  M$ = TI$                          [Uhr ablesen]
4020  FOR N = 1 TO 25
4030  POKE 53281, 15*RND (0)            [Bildschirm blinken]
4040  NEXT N
4050  POKE 53281, 6                     [Schirm wieder blau]
4060  PRINT CHR$(19)
4070  PRINT: PRINT
4080  PRINT "gTc[2mal]"
4090  PRINT "UNFALL NACH"
4100  PRINT VAL (LEFT$ (M$, 2));
      "STUNDEN□□□□□□"
4110  PRINT VAL (MID$ (M$, 3, 2));
      "MINUTEN□□□□"                     [Gesamtfahrzeit]
4120  PRINT VAL (RIGHT$ (M$,2));
      "SEKUNDEN□□□□"
```

```
413Ø  PRINT "g@c [12mal]"
414Ø  STOP
```

Wenn Sie das fahren, werden Sie aufgefordert, die Schwierigkeitsstufe zu wählen. 1 ist leicht, 5 schwer. Im Zweifel beginnen Sie bei 1 und arbeiten sich hoch.

Das Spiel beginnt, sobald Sie nach dieser Eingabe RETURN drücken. (Falls Sie das überrascht, fügen sie ein:

```
85  FOR T = 1 TO 2ØØØ: NEXT T
```

damit Sie Zeit zum Überlegen haben.)

Drücken Sie Taste □ für Bewegung nach links und Taste □ für Bewegung nach rechts. Versuchen Sie den Straßenrand zu meiden. Das Programm verwendet Kollisionserkennung von Hintergrundsprites (siehe Lösung zu Aufgabe 3).

Lösungen

Aufgabe 1

Fügen Sie dem Programm diese Zeilen an:

```
85Ø  POKE 2Ø43, 13
86Ø  POKE V + 42, 2
```

um Zeiger und Farbe zu setzen. Schreiben Sie vorangehende Zeilen um, um alle vier Sprites zu aktivieren und die Erweiterungen richtig zu erzielen:

```
25Ø  POKE V + 21, 15
79Ø  POKE V + 29, 15
```

und nun die Schleife abändern:

```
1Ø1Ø  POKE V + 3, Y: POKE V + 7, Y + 2Ø
1Ø12  IF OF = 1 THEN POKE V + 16, (PEEK (V + 16)) OR 1Ø
1Ø13  IF OF = Ø THEN POKE V + 16, (PEEK (V + 16)) AND 245
1Ø14  POKE V + 2, X-OF*256: POKE V + 6, X-OF*256
```

Aufgabe 2

1. 1Ø1ØØØØØ = 16Ø
2. ØØØ1Ø1ØØ = 2Ø
3. Ø1ØØ1ØØØ = 72

Aufgabe 3

Der Rand blinkt, wenn das Sprite mit *Text* zusammenstößt. Register V + 31 setzt bei Kollisionen zwischen Sprites und Text Bit K auf 1, sobald Sprite K mit Text zusammenstößt. Seite 154 des Handbuchs beschreibt das als 'Sprite-Hintergrund-Kollision', ein eher verwirrender Begriff; es muß heißen 'Sprite-*Vorder*grund-Kollision'. Ebenso muß es bei Register V + 27 im Handbuch statt 'Hintergrund-Sprite-Priorität' 'Sprite-Vordergrund-Priorität' heißen. Bit K entscheidet dann, ob Sprite K auf oder unter Text dargestellt wird.

29 Debugging V

Läuft das Programm wirklich?

Wie beweisen wir schlüssig, daß ein Programm genau das leistet, wozu es geschrieben worden ist? Ich möchte hier nicht philosophisch werden (wir sind auf dem besten Weg dazu), aber grob gesprochen ist das so ähnlich, als wolle man von einem Astronomen wissen, ob morgen die Sonne aufgehen wird. Wenn er sehr pedantisch ist, erwidert er vielleicht, die Erde flöge nun schon sehr lange Zeit um die Sonne, und wir wären mit einer Reihe von physikalischen Gesetzen vertraut, die darauf hindeuten, daß sie das regelmäßig weiterhin zu tun gedenke, weshalb man einiges darauf verwetten dürfe, es werde auch morgen noch der Fall sein; er würde aber hinzufügen, daß er nicht wissen könne, ob unsere physikalischen Gesetze auch richtig seien und das, was wir Jahrtausende lang beobachtet haben, nicht vielleicht die äußere Erscheinung eines viel komplexeren Gesetzes sei, dessen Wirkung morgen darin bestehen könne, die Richtung der Erddrehung umzukehren oder den Erdball ganz aus seiner Bahn zu führen.

Versteckte Fehler

Analog gibt es, weil ein Programm sich bei den ersten tausend Sätzen von Dateneingaben richtig verhält, keine absolute Garantie dafür, daß es beim tausendunderstenmal auch klappt. Vielmehr treten Fehler oft monate- oder sogar jahrelang nicht in Erscheinung, nachdem ein Programm erfolgreich abgeschlossen und bei Dutzenden oder sogar Hunderten von Anlässen problemlos gefahren worden ist. Eigentlich kein Wunder; schließlich sind es gerade die am seltensten auftretenden Bedingungen, die ein Programmierer am leichtesten übersieht.

Hier ein Beispiel:
Wir schreiben eine Folge von Programmen für die E-Werke Magerhausen zur Verwaltung ihrer Kundenkonten. Wir erfahren, daß es zwei Tarife gibt, A und B. Bei Tarif A bezahlt der Kunde eine Vierteljahrespauschale von 150 Mark und pro Verbrauchseinheit 4 Pfennig. Bei Tarif B fällt die Pauschale weg, die Einheit kostet 7 Pfennig. Wir schreiben also einen Code solcher Art:

```
1ØØ  INPUT T$
1Ø5  INPUT EINHEITEN
11Ø  IF T$ = "A" THEN 3ØØ
12Ø  IF T$ = "B" THEN 14Ø
13Ø  GOTO 5ØØØ
14Ø  RECHNUNG = 7 * EINHEITEN/1ØØ
```

```
15Ø  PRINT RECHNUNG
16Ø  GOTO 1ØØ
3ØØ  RECHNUNG = 15Ø + 4 * EINHEITEN/1ØØ
31Ø  PRINT RECHNUNG
32Ø  GOTO 1ØØ
...
5ØØØ  PRINT  "UNGUELTIGER TARIF"
5Ø1Ø  STOP
```

Schön. Ich weiß, der Code könnte effektiver sein, und wir würden mehr Informationen benötigen, etwa Namen und Kontonummer des Kunden, aber der Sinn wird schon klar.

Wir testen also das Programm, es läuft glatt, und wir entfernen uns mit den gemurmelten Worten, eigentlich sei es doch eine Schande, daß wir mit unseren glänzenden Gaben derart lächerliche Programme verfassen müßten.

Und es läuft auch wunderbar, jahrelang, bis eines schönen Tages eine Rechnung über DM Ø.ØØ ausgedruckt wird. Das fällt natürlich keinem Menschen auf, weil es eine Rechnung unter Tausenden ist, die vermutlich ohnehin automatisch kuvertiert werden. Der Empfänger wundert sich über die Rechnung und ist wahrscheinlich belustigt, weil sie wieder einmal beweist, wie blöd Computer doch sind; er sieht aber keinen Anlaß, etwas zu unternehmen und wirft die Rechnung in den Papierkorb. Leider haben wir in derselben Folge ein Programm verfaßt, mit dem das Absendedatum jeder Rechnung gespeichert wird. Falls keine Bestätigung eingeht, daß die Rechnung innerhalb von 28 Tagen beglichen worden ist, geht eine letzte Mahnung hinaus. Diesmal ist der Empfänger eher verärgert als belustigt, wirft aber auch diese Rechnung weg. Von nun an geht's bergab. Die Routine, mit der die Frist zwischen Rechnungsdatum und Zahlungseingang überwacht wird, weist die Betriebsabteilung an, dem Kunden den Strom abzuschalten, falls er nach 60 Tagen immer noch nicht bezahlt hat.

Was ist passiert? Ganz einfach! Der Kunde ist ein Rentner, der sich die billigen Touristikangebote zunutzegemacht und den ganzen Winter in Mallorca verbracht hat. Er war knapp über drei Monate außer Landes und hat für einen ganzen Berechnungszeitraum keinen Strom verbraucht. Außerdem ist er ein sparsamer Stromverbraucher, zahlt also nach Tarif B. Deshalb hat das System eine Aufforderung abgeschickt, Null Mark zu entrichten. Freilich kann das nicht oft vorkommen, weil nur wenige Menschen so lange von zu Hause fort sind und auch nicht sehr viele Leute Tarif B wählen dürften. Damit das Problem auftreten kann, muß der Kunde beiden Bedingungen entsprechen.

Einmal entdeckt, ist der Fehler leicht zu beheben:

```
145  IF RECHNUNG = Ø THEN 1ØØ
```

so daß die PRINT-Anweisung umgangen wird. Dieses Problem soll bei einem frühen Computersystem aufgetaucht sein, aber es kann sich auch um eine Legende handeln. Auf jeden Fall scheint es mir deutlich zu zeigen, daß ein Fehler fast ewig schlummern kann.

Die Moral: Wenn Sie Daten erfinden, um ein Programm zu testen, tun Sie das nicht aufs Geratewohl. Wählen Sie Werte in und bei Verzweigungswerten im Programm. Wenn es in einer Anweisung heißt

```
305  IF U < 30 THEN 400
```

fahren Sie einen Test mit U bei 29.999, einen mit U = 30 und einen mit U = 30.001. Sie könnten auch gemeint haben:

```
305  IF U < = 30 THEN 400
```

Wenn Sie nur mit U = 15 und U = 160 testen, fällt Ihnen der Fehler nicht auf.

Wählen Sie Testdaten so aus, daß zu irgendeinem Zeitpunkt jeder Abschnitt des Programms ausgeführt wird. Und natürlich müssen Sie genau wissen, wie die Antwort für jeden Satz Testdaten auszufallen hat.

30 Klang und Musik

Laut Commodore bedeutet die Abkürzung SID für den Klangchip 'Sound Interface Device'. Nach meinem Gefühl steht sie für 'Sydney Opera House' und ist nur falsch geschrieben.

Sie haben VIC kennengelernt, der, wie wir sehen konnten, bemerkenswerte Dinge beim Bildschirmdisplay bewirkt. Jetzt möchte ich Ihnen SID vorstellen, der sich nicht lumpen läßt, sondern den 64 in ein Musikinstrument oder sogar in einen annehmbaren Synthesizer verwandelt.

SID mitzuteilen, was Sie tun wollen, besitzt viel Ähnlichkeit mit den Anweisungen für VIC. Das heißt, SID hat eine Anzahl von Registern, in die Sie mit POKE hineingehen können. Jedes Register bestimmt eine andere Eigenschaft des Tones, der erzeugt werden soll.

Wir sollten uns zu Beginn überlegen, welche Eigenschaften eines Tons ihn für ein bestimmtes Instrument charakterisieren. Beispielsweise können wir den Unterschied zwischen dem eingestrichenen C auf eine Gitarre und demselben Ton auf einer Orgel leicht erkennen. Warum?

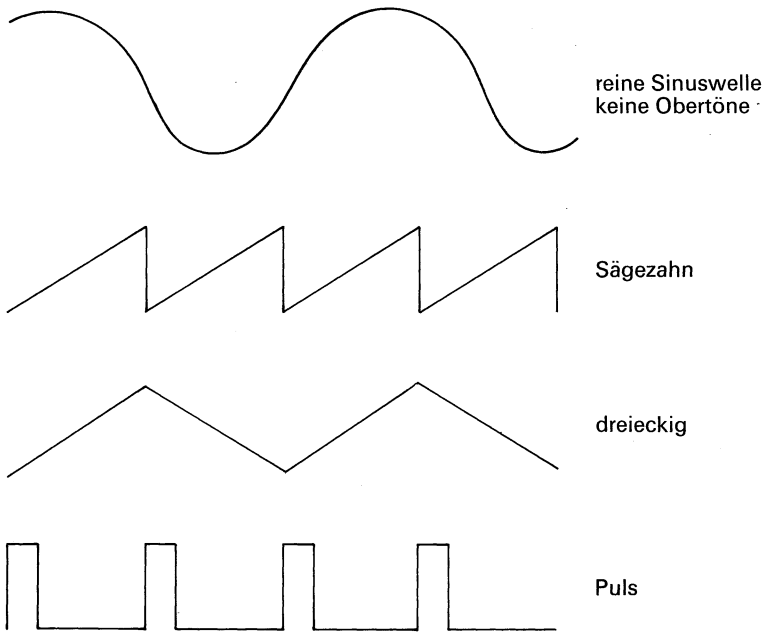


Abbildung 30.1
Die vier Wellengrundformen.

Nun, erstens gibt es die Zusammensetzung der Frequenzen innerhalb der Note. Ich will hier nicht auf die Physik der Tonerzeugung eingehen, aber ganz kurz: Bei jedem Ton wird eine Ton*grundfrequenz* zusammen mit *Obertönen* erzeugt, die Vielfache der Grundfrequenz sind. Zahl und relative Lautstärke dieser Obertöne (verglichen mit dem Grundton) verleihen dem Ton einen Klang, der für das Instrument charakteristisch ist. Wenn wir uns diese Frequenzen kombiniert vorstellen, ergibt sich eine bestimmte Wellenform, die auf einem Oszillographen wie Abbildung 3Ø.1 aussehen könnte.

Würden Sie SID aber nur mitteilen, wie er die *Frequenzen* eines Gitarrentons erzeugen soll, würde das Ergebnis kaum nach einer Gitarre klingen. Das liegt daran, daß eine zweite Eigenschaft zu berücksichtigen ist. Die Lautstärke des Tones verändert sich beim Spiel. Beispielsweise beginnt bei einer Orgel der Ton leise, weil in der Pfeife nicht viel Luft ist, aber wenn sie sich verstärkt, nimmt der Ton an Lautstärke zu. Bei einer Gitarre ist es umgekehrt; der Ton ist unmittelbar nach dem Zupfen der Saite am lautesten, dann verklingt er langsam, falls er vom Gitarristen nicht bewußt gedämpft wird. Ein Saxophonist kann einen Ton, den zu spielen er beginnt, durchhalten, solange sein Atem reicht. Und so weiter.

Aus alledem ergibt sich deutlich, daß die Produktion der richtigen Eigenschaften (oder *Hüllkurve*) eines Tons aus mehreren Stufen besteht.

SID kennt vier davon:

1. *Anschlag* (*A*tack): Die Geschwindigkeit, mit der vom Beginn des Tones an die höchste Lautstärke erreicht wird.
2. *Abschwellen* (*D*ecay): Das Tempo, mit dem die Lautstärke von diesem Scheitelpunkt hinabfällt zum:
3. *Halten* (*S*ustain): Der Lautstärkepegel, der dann gehalten wird, bis der Ton nicht mehr gespielt wird.
4. *Ausklingen* (*R*elease): Das Tempo, mit dem der Ton verklingt, wenn er nicht mehr gespielt wird.

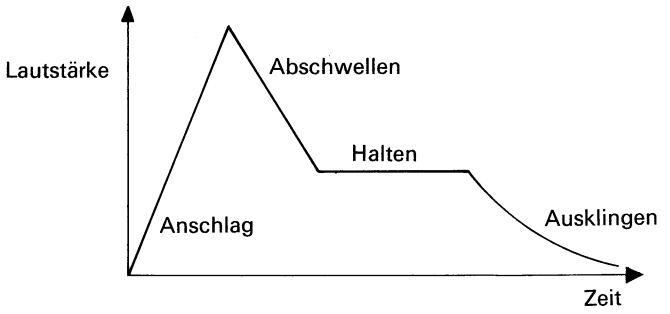


Abbildung 30.2
Die vier Stufen einer ADSR-Hüllkurve.
(ADSR = Attack, Decay, Sustain, Release)

All das erlaubt Ihnen allerhand Raffinessen, aber wie gewohnt, je leistungsstärker etwas ist, desto vorsichtiger muß man damit umgehen, und desto mehr muß man wissen.

Wir fangen also mit sehr begrenzten Zielen an und erweitern unseren Horizont nur langsam.

Die erste Aufgabe ist die, einen musikalischen Ton zu erzeugen und dann Töne aneinanderzureihen, damit sie eine Melodie bilden. Ob das dann nach Klavier oder Orgel klingt, ist im Augenblick belanglos.

Offensichtlich wäre es praktisch, musikalisch gesprochen (einigermaßen) konventionell zu denken. Ich möchte vom 'eingestrichenen C' oder 'F#' und so reden. Aus Rücksicht auf die numerische Beschaffenheit von BASIC definieren wir jeden Ton als Zahlenpaar, das den Ton und seine Oktave bezeichnet. Wir nennen die mittlere Oktave Ø und 'C' Ton Ø. Die mittlere Oktave wird also geschrieben wie in Tabelle 3Ø.1.

Tabelle 30.1

Ton	Oktave	Notenschreibweise
Ø	Ø	C
1	Ø	C# (oder D*b*)
2	Ø	D
3	Ø	D# (oder E*b*)
4	Ø	E
5	Ø	F
6	Ø	F# (oder G*b*)
7	Ø	G
8	Ø	G# (oder A*b*)
9	Ø	A
1Ø	Ø	A# (oder B oder H*b*)
11	Ø	H
Ø	1	C (eine Oktave höher)

Auf diese Weise reichen die verfügbaren Oktaven von –4 (der niedrigsten) bis 3 (der höchsten).

Jetzt brauchen wir einen Zusammenhang zwischen dieser Schreibweise und den Frequenzen, die mit POKE in die entsprechenden Register von SID eingegeben werden müssen.

Die Frequenzintervalle zwischen Noten in der Musik des Abendlandes beruhen auf der sogenannten *diatonischen* Skala. (Diatonisch heißt 'in Tonintervallen' und hat nichts zu tun mit der Vertonung von Diapositiven.) Leider sind die Verhältnisse zwischen aufeinanderfolgenden Tönen nicht immer gleich. Die Komponisten im 18. Jahrhundert kamen dahinter, daß das hieß, sie könnten die Tonart nicht wechseln, und erfanden deshalb die 'gleichschwebende Temperatur', die nah herankommt.

Das geht so: Um von einem Ton in einer Oktave zum gleichen in einer anderen zu gelangen, multipliziert man mit zwei. (Das gilt für beide Tonleitern.) Da es in jeder Oktave zwölf Töne gibt, muß man, um die Verhältnisse zwischen aufeinanderfolgenden Tönen gleichzuhalten, mit 2 ↑ (1/12) multiplizieren, um von einem Ton zum nächsten zu gelangen.

Nun ergibt sich, daß die Frequenz für das eingestrichene C 4291 ist. Wir brauchen also eine Subroutine, die einen Noten- und Oktavwert annimmt (NTE und OCT), 4291 mal 2 hoch irgendwas multipliziert und einen angemessenen

Wert liefert, der in die Frequenzregister von SID gesteckt werden kann. Register Mehrzahl? Wieso mehr als eines? Nun, es gibt mehr als 256 verschiedene Töne, also bestimmen zwei Bytes (die ich H% und L%, für High (hohes) und Low (niedriges) Byte nenne), die Frequenz.

Also los:

```
1000  F = 4291 * 2 ↑ OCT                [für die richtige Oktave]
1010  IF NTE > 0 THEN                   [mit 2 ↑ (1/12) für jeden Ton
      F = F * 2 ↑ .0833333333:          über C multiplizieren]
      NTE = NTE - 1: GOTO 1010
1020  FI = INT (F + 0.5)                [F aufrunden und Integerteil
                                        nehmen]
1030  H% = FI/256: L% = FI - 256 * H%   die zwei Bytes erzeugen
1040  RETURN
```

Die %-Zeichen an H% und L% verwundern Sie vielleicht. Es gibt zwei Möglichkeiten, im Computer Zahlen zu speichern: *Fließpunkt* (Dezimalzahlen wie 7.443) und *Integer* (ganze Zahlen wie 7). Integervariable erfordern im Speicher weniger Platz. Wenn Sie dem Namen einer Variablen oder einen Array das %-Zeichen anhängen, weisen Sie den Computer an, sie als ganze Zahlen zu behandeln. Das müssen Sie aber im ganzen Programm tun, denn der Computer behandelt H und H% als verschiedene Variable.

Jetzt zum Hauptprogramm. Ich kann mir die Registeradressen von SID nie merken, beginne also damit, daß ich diejenigen, die ich brauche, Variablen zuteile:

```
10  VOL = 54296: FH = 54273: FL = 54272: WFM = 54276: AD =
    54277: SR = 54278
```

'VOL' setzt die Lautstärke für den ganzen Chip auf einen Wert zwischen 0 und 15. FH und FL sind die hohen und niedrigen Bytes der Frequenz. WFM ist die Wellenform. AD und SR sind die Anschlag/Abschwell- und Halten/Ausklang-Teile der Hüllkurve.

Die Berechnung anzustellen, während die Musik spielt, ist nicht praktisch; das dauert zu lange. Wir setzen also Arrays, berechnen die verlangten Werte, speichern sie und spielen die Melodie erst am Schluß.

```
20  DIM H%(200), L%(200), D(200)
```

H% und L% sind die hohen und niedrigen Bytes der Frequenzwerte, D ist die Dauer des Tons. Ich lasse maximal 201 Töne zu, aber Sie könnten natürlich Platz für mehr vorsehen.

Wir gehen davon aus, daß wir eine DATA-Liste von Noten haben, mit denen wir arbeiten können, jede bestehend aus Noten-, Oktav- und Dauerwerten. Eine Dauer von 0 soll als Begrenzer dienen.

Wir können also eine Schleife setzen, die Noten liest, notwendige Umwandlungen vornimmt und die Arrays lädt:

```
40  P = 0
50  READ NTE, OCT, D
55  IF D = 0 THEN 100
60  GOSUB 1000: REM GET H%, L%
70  H%(P) = H%: L%(P) = L%: D(P) = D
80  P = P + 1
90  GOTO 50
```

Jetzt die Melodie spielen. Zuerst setzen wir einen Tempowert, damit die Geschwindigkeit, mit der die Melodie gespielt wird, verändert werden kann. Dann setzen wir die anderen Register:

```
100  INPUT "TEMPO"; TE
102  INPUT "AD, SR"; A, S
104  POKE AD, A: POKE SR, S
106  INPUT "WELLENFORM"; W
108  POKE WFM, W
110  POKE VOL, 15
```

Die Reihenfolge der Ereignisse hier ist bedeutsam. Besonders WFM muß aus Gründen, auf die ich später komme, *nach* AD und SR gesetzt werden.

Jetzt gehen wir durch eine Schleife, geben die Arraywerte mit POKE in die Frequenzregister ein und verzögern das Abspielen des nächsten Tons entsprechend:

```
115  P = 0
120  POKE FH, H%(P): POKE FL, L%(P)
130  FOR N = 1 TO D(P) * TE: NEXT
140  P = P + 1
150  IF D(P) = 0 THEN POKE VOL, 0: END
160  GOTO 120
```

Beachten Sie, daß die Verzögerung in 130 von TE abhängt. Je größer TE, desto langsamer wird die Melodie gespielt. Die in D konkret verwendeten Zahlen spielen deshalb keine Rolle, solange das Verhältnis zwischen ihnen richtig ist. TE können Sie jederzeit auf angemessene Weise anpassen. Beachten Sie außerdem, daß VOL am Ende auf Null gesetzt wird, weil der letzte Ton sonst endlos lange ausgehalten wird.

Probieren Sie es jetzt. Schreiben Sie:

```
2ØØØ  DATA 1, Ø, 2
2Ø1Ø  DATA Ø, Ø, Ø
```

und RUN.

Versuchen Sie es mit folgenden Parametern:

```
TEMPO = 2ØØ
AD, SR = Ø,24Ø
WELLENFORM = 17
```

Sie sollten in der mittleren Oktave ein C# erhalten, das orgelähnlich klingt.

Lassen Sie Zeile 2ØØØ, wie sie dasteht, und fügen Sie an:

```
2Ø1Ø  DATA 1, Ø, 2
2Ø2Ø  DATA 5, Ø, 2
2Ø3Ø  DATA 5, Ø, 2
2Ø4Ø  DATA 3, Ø, 2
2Ø5Ø  DATA 1, Ø, 2
2Ø6Ø  DATA 6, Ø, 2
2Ø7Ø  DATA 6, Ø, 2
2Ø8Ø  DATA 5, Ø, 2
2Ø9Ø  DATA 3, Ø, 2
21ØØ  DATA 5, Ø, 2
211Ø  DATA 8, Ø, 2
212Ø  DATA 8, Ø, 2
213Ø  DATA 7, Ø, 2
214Ø  DATA 8, Ø, 4
215Ø  DATA 5, Ø, 2
216Ø  DATA 1Ø, Ø, 3
217Ø  DATA 8, Ø, 1
218Ø  DATA 6, Ø, 2
219Ø  DATA 5, Ø, 2
22ØØ  DATA 3, Ø, 2
221Ø  DATA 1, Ø, 2
```

```
220  DATA 0, 0, 2
230  DATA 5, 0, 2
240  DATA 3, 0, 2
250  DATA 1, 0, 2
260  DATA 1, 0, 2
270  DATA 0, 0, 2
280  DATA 1, 0, 4
290  DATA 0, 0, 0
```

RUN, und dieselben Werte wie zuvor. Das paßt vielleicht ganz gut, wenn Sie Ihren 64 zu Weihnachten bekommen haben!

Mein eigentlicher Grund für diese Wahl ist aber der, daß die meisten Töne von gleicher Länge und in derselben Oktave sind, so daß das leicht zu bewältigen ist. Man kann das ('While Shepherds Watched Their Flocks' heißt das schöne Lied) auch von Notenblättern abschreiben, aber ich habe es durch Herumprobieren herausgebracht. (Was heißt da, das merkt man?)

Aufgabe 1

Verändern Sie die DATA-Liste in diesem Programm, damit es den Refrain von 'Don't Cry for Me, Argentina' spielt.

Harmonik

SID ist viel schlauer, als ich bisher zugegeben habe.

Als erstes können wir mehr als eine Stimme spielen (sogar bis zu drei), die sich voneinander völlig unabhängig entfalten können.

Um das vorzuführen, probieren wir ein bißchen mit Zweiklängen. Da gleichzeitig zwei Töne gespielt werden, bauen Sie Zeile 20 um:

```
20  DIM H%(200,2), L%(200,2), D(200)
```

(Der Einfachtheit halber unterstellen wir, daß beide Töne dieselbe Dauer haben.) Jede DATA-Anweisung enthält nun also fünf Werte:

Ton (Stimme 1), Oktave (Stimme 1), Dauer, Ton (Stimme 2),

Oktave (Stimme 2)

In Zeile 50 wollen wir die ersten drei davon haben, also bleibt sie gleich, wie auch die Zeilen 55 und 60, aber in Zeile 70 müssen die Ergebnisse in die *erste* Spalte der Arrays gesetzt werden:

```
70  H%(P,1) = H%: L%(P,1) = D(P) = D
```

Nun müssen wir den zweiten Ton holen und die Berechnungen bei ihm wiederholen, aber das Ergebnis gehört in die zweite Spalte der Arrays:

```
75  READ NTE, OCT
76  GOSUB 1ØØØ
77  H%(P,2) = H%: L%(P,2) = L%
```

Alle vorher für Stimme 1 gesetzten Register müssen jetzt auch für Stimme 2 gesetzt werden. Praktischerweise befinden sich alle Register für die zweite Stimme 7 über den entsprechenden Registern für Stimme 1. Übrigens sind die Register für die dritte Stimme 14 über den Werten für Stimme 1. Siehe Anhang 4, der eine kurze Zusammenfassung von SID-Registern enthält.

Der Codeblock, der die Register setzt, sieht also nun etwa so aus:

```
1ØØ  INPUT "TEMPO"; TE                    wie vorher
1Ø2  INPUT "AD, SR(1)"; A, S
1Ø4  POKE AD, A: POKE SR, S
1Ø5  INPUT "AD, SR(2)"; A, S: POKE AD + 7, A: POKE SR + 7, S
1Ø6  INPUT "WELLENFORM(1)"; W1: POKE WFM, W1
1Ø8  INPUT "WELLENFORM(2)"; W2: POKE WFM + 7, W2
```

Zeile 12Ø wird zu:

```
12Ø  POKE FH, H%(P, 1): POKE FL, L%(P, 1)
```

und wir brauchen eine Zeile 125 für die Harmonik:

```
125  POKE FH + 7, H%(P, 2): POKE FL + 7, L%(P, 2)
```

Im übrigen bleibt das Programm, wie es war. Sämtliche DATA-Anweisungen verändern sich natürlich:

```
2ØØØ  DATA 5, Ø, 1, 5, Ø
2Ø1Ø  DATA 1, Ø, 1, 5, Ø
2Ø2Ø  DATA 3, Ø, 1, 5, Ø
2Ø3Ø  DATA 5, Ø, 1, 5, Ø
2Ø4Ø  DATA 8, Ø, 1, 5, Ø
2Ø5Ø  DATA 6, Ø, 1, 5, Ø
2Ø6Ø  DATA 6, Ø, 1, 6, Ø
2Ø7Ø  DATA 1Ø, Ø, 1, 6, Ø
2Ø8Ø  DATA 8, Ø, 1, 6, Ø
```

```
2Ø9Ø  DATA 8, Ø, 1, 8, Ø
21ØØ  DATA 1, 1, 1, 8, Ø
211Ø  DATA Ø, 1, 1, 8, Ø
212Ø  DATA 1, 1, 1, 8, Ø
213Ø  DATA 8, Ø, 1, 8, Ø
214Ø  DATA 5, Ø, 1, 8, Ø
215Ø  DATA 1, Ø, 1, 6, Ø
216Ø  DATA 3, Ø, 1, 6, Ø
217Ø  DATA 5, Ø, 1, 6, Ø
218Ø  DATA 6, Ø, 1, 5, Ø
219Ø  DATA 8, Ø, 1, 5, Ø
22ØØ  DATA 1Ø, Ø, 1, 5, Ø
221Ø  DATA 8, Ø, 1, 5, Ø
222Ø  DATA 6, Ø, 1, 5, Ø
223Ø  DATA 5, Ø, 1, 5, Ø
224Ø  DATA 3, Ø, 1, 1, Ø
225Ø  DATA 5, Ø, 1, 1, Ø
226Ø  DATA 1, Ø, 1, 1, Ø
227Ø  DATA Ø, Ø, 1, Ø, Ø
228Ø  DATA 1, Ø, 1, Ø, Ø
229Ø  DATA 3, Ø, 1, Ø, Ø
23ØØ  DATA 8, -1, 1, Ø, Ø
231Ø  DATA Ø, Ø, 1, Ø, Ø
232Ø  DATA 3, Ø, 1, Ø, Ø
233Ø  DATA 6, Ø, 1, Ø, Ø
234Ø  DATA 5, Ø, 1, Ø, Ø
235Ø  DATA 3, Ø, 1, Ø, Ø
236Ø  DATA 5, Ø, 1, 5, Ø
237Ø  DATA 1, Ø, 1, 5, Ø
238Ø  DATA 3, Ø, 1, 5, Ø
239Ø  DATA 5, Ø, 1, 5, Ø
```

```
2400  DATA 8, 0, 1, 5, 0
2410  DATA 6, 0, 1, 5, 0
2420  DATA 6, 0, 1, 6, 0
2430  DATA 10, 0, 1, 6, 0
2440  DATA 8, 0, 1, 6, 0
2450  DATA 8, 0, 1, 8, 0
2460  DATA 1, 1, 1, 8, 0
2470  DATA 0, 1, 1, 8, 0
2480  DATA 1, 1, 1, 8, 0
2490  DATA 8, 0, 1, 8, 0
2500  DATA 5, 0, 1, 8, 0
2510  DATA 1, 0, 1, 6, 0
2520  DATA 3, 0, 1, 6, 0
2530  DATA 5, 0, 1, 6, 0
2540  DATA 10, -1, 1, 5, 0
2550  DATA 8, 0, 1, 5, 0
2560  DATA 6, 0, 1, 5, 0
2570  DATA 5, 0, 1, 5, 0
2580  DATA 3, 0, 1, 5, 0
2590  DATA 1, 0, 1, 5, 0
2600  DATA 8, -1, 1, 0, 0
2610  DATA 1, 0, 1, 0, 0
2620  DATA 0, 0, 1, 0, 0
2630  DATA 1, 0, 6, 1, 0
2640  DATA 0, 0, 0, 0, 0
```

Fahren Sie das mit RUN und nehmen Sie für den Anfang dieselben Parameter wie vorher für beide Stimmen. Sie sollten hören 'Jesus, du des Menschen Freude' von J. S. Bach. (Nicht Johann Sebastian, sondern sein wenig bekannter Sohn Jones Stewart Bach, der dafür bekannt war, daß ihm seine Harmonik immer ein bißchen durcheinandergeriet.)

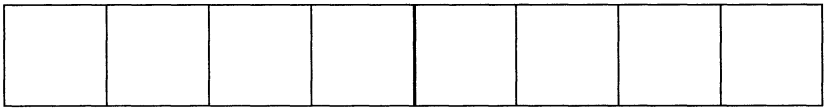
Um den Klang ein bißchen zu 'verbreitern', schreiben Sie Zeile 75 um:

```
75  READ NTE, OCT: OCT = OCT 1
```

Nun läuft die Harmonik eine Oktave unter der Melodie. Klingt doch recht hübsch, oder?

Verändern Sie als nächstes die Wellenform für die Melodie (Stimme 1). Verwenden Sie 33 statt 17. Der Unterschied im Klangcharakter ist auffällig.

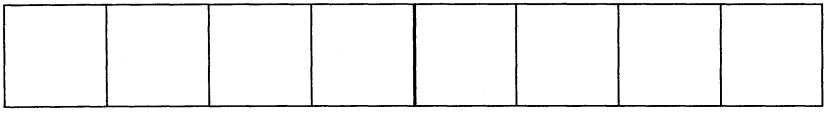
Bis jetzt bin ich der Aufgabe ausgewichen, an den ASDR-Werten herumzubasteln, aber nun haben wir eine ausreichende Grundlage, um zuversichtlich mit ihnen umgehen zu können. Die Anschlag- und Abschwellwerte sind in den beiden Hälften eines 8 Bit-Reigsters (siehe Kapitel 12) so enthalten:



Anschlag Abschwellen

Je größer die Zahl in beiden, desto länger die Anschlags-(oder Abschwell)periode. Der höchste Anschlag ist also 1111 (= 15 Dezimal). Da sich das in der linken Hälfte des Bytes befindet, ist es in Wirklichkeit das 16fache (= 24Ø). Ein ziemlich rascher Abschwellwert wäre ØØ11, die Dezimalentsprechung für diese Kombination 243.

Die Werte für Halten und Ausklingen sind ähnlich organisiert:



Halten Ausklingen

Der höchste Wert für das Halten ist somit 24Ø (dezimal).

Die Werte, die ich bisher verwendet habe, liefern schnellsten Anschlag, schnellstes Abschwellen für den lautesten Haltewert, gefolgt vom raschesten Ausklingen. In Wahrheit gibt es kein Abschwellen, weil der Haltewert dafür sorgt, daß die höchste Lautstärke beibehalten wird. Wenn man hinhört, merkt man das.

Verwirrender ist, daß man nicht hören kann, wie die Töne abgeschaltet werden. Vom einen zum anderen findet ein glatter Übergang statt.

Der Grund: Der Wert für das Halten wird endlos beibehalten, bis Bit Ø des Wellenform-Steuerregisters (WFM) auf Null zurückgesetzt wird. Die Bitmuster der Wellenform, die wir bis jetzt benützt haben, sind 18 (binär ØØØ1ØØ1) und 33 (binär ØØ1ØØØØ1), die beide Bit Ø auf 1 setzen. *Deshalb* ist es wichtig, ADSR vor WFM zu setzen. Sobald Bit Ø von WFM gesetzt ist, können Sie mit ADSR ohne Auswirkungen anstellen, was Sie wollen. Dies alles bedeutet, daß der einzige Weg, den Ausklingteil des Zyklus auszulösen, der ist, mit POKE in WFM eine Ø einzugeben. Dann müssen Sie den gewünschten Wellenform-Wert mit POKE neu eingeben, bevor Sie versuchen, den nächsten Ton zu spielen.

Um das also auszuprobieren, lassen Sie die Unterharmonik in Ruhe und schreiben Zeile 12Ø so um:

```
12Ø  POKE WFM, W1: POKE FH, H%(P, 1): POKE FL, L%(P, 1)
```

Wir wollen den entsprechenden Harmonieton im selben Augenblick beginnen (oder so nah wie möglich daran), also bleibt Zeile 125 unverändert, aber nun müssen wir nach einer passenden Verzögerung die Melodie abschalten:

```
127  FOR N = 1 TO D(P) * MD: NEXT: POKE WFM, Ø
```

Das unterstellt natürlich, daß für MD (Melodieverzögerung) ein Wert beigesteuert worden ist. Da wir vermutlich die Verzögerung in der Melodie und das Tempo getrennt ändern wollen, wäre es vernünftig, beide einzugeben; dann sähe Zeile 1ØØ so aus:

```
1ØØ  INPUT "TEMPO"; TE: INPUT "MELODIEVERZOEGERUNG"; MD
```

Die Verzögerung in Zeile 13Ø ist jetzt zu lang; die Schleife sollte TE-MDmal ausgeführt werden:

```
13Ø  FOR N = 1 TO D(P) * (TE-ND): NEXT
```

Fahren Sie das und probieren Sie die folgenden Werte aus:

```
TEMPO? 18Ø
MELODIE? 18Ø          [damit beide Töne gleich lang dauern]
AD SR (1)? 9, Ø
AD, SR (2)? Ø,24Ø
WELLENFORM (1)? 33
WELLENFORM (2)? 17
```

Nun lassen Sie die Melodie von einem Cembalo spielen, eine Orgel steuert die Harmonik bei! (Na ja, ein *bißchen* Phantasie muß man schon mitbringen!)

Andere Instrumente

Sie können mit Wellenform- und ADSR-Werten in zufälliger Weise spielen und recht erstaunliche Ergebnisse erzielen. Wenn Sie aber ein bestimmtes Instrument nachahmen wollen, müssen Sie ein paar Dinge im Gedächtnis behalten.

Wellenform 17 ('Dreieck') besitzt den ziemlich reinen Klang, den man mit Orgel, Flöte oder Glocke verbindet.

Wellenform 33 ('Sägezahn') hat etwas Schwirrendes an sich. Daher die Verwendung für Cembalo.

Es gibt noch eine andere Wellenform, von der wir noch gar nicht gesprochen haben. Ihr Wert ist 65, und sie erzeugt *Pulse*, die durch Setzen eines 2 Byte-Registers in der Schwingung verändert werden können. Für Stimme 1 ist das niedrige Byte 54274, das hohe Byte 54275. Probieren Sie aus:

```
POKE 54274,255: POKE 54275,Ø
```

Fahren Sie das Programm erneut und setzen Sie Wellenform 1 auf 65, ADSR 1 auf 9,Ø.

Das schwirrt noch mehr, nicht? Das Handbuch behauptet, es sei ein Klavier, aber ich meine, man braucht schon eine Riesenphantasie, um *darauf* zu kommen. Ein Instrument, das ebenfalls schwirrt, ist jedoch die Gitarre. Können wir berechnen, wie die Hüllkurve für einen Gitarrenton aussehen müßte?

Geht ganz leicht, wenn man überlegt. Die Saite wird geschlagen (oder gezupft) und erzeugt unmittelbar danach den lautesten Klang. Der Anschlag ist also schnell (Ø). Der Ton wird nun mit der Zeit leiser, so daß das Abschwellen ziemlich lange dauert, sagen wir 1Ø. Der AD-Wert ist also Ø * 16 + 1Ø = 1Ø. Halten gibt es nicht, und das Ausklingen können wir rasch besorgen (weil es keinen Unterschied macht), also SR = Ø.

Probieren Sie das. Ich denke, Sie werden Gitarrenartiges hören. Verringern Sie den AD-Wert ein bißchen (6? 4?). Sie werden feststellen, daß der Ton, wie erwartet, 'gedämpft' ist und Sie einen Banjo- oder Ukuleleklang erhalten.

Wie wäre es mit einer Glocke? In Begriffen der Hüllkurve besteht viel Ähnlichkeit mit einer Gitarre, weil der Ton von seinem Scheitelpunkt, der sofort nach dem Schlagen auftritt, langsam abschwillt. Der Klang ist aber reiner. Versuchen Sie also Wellenform 17 mit AD = 1Ø, SR = Ø. Das Ergebnis läßt sehr an eine Glocke denken, aber das fällt Ihnen vielleicht nicht auf, bis Sie ein sehr langsames Tempo wählen, um den Ton ausklingen zu lassen.

Aufgabe 2

Das Morseprogramm in Kapitel 18 zeigte nur die Punkte und Striche an. Es wäre viel interessanter, zusätzlich die angemessenen Geräusche zu erzeugen. Schreiben Sie das Programm entsprechend um.

(Verwenden Sie für Punkte und Striche denselben Ton, aber die Strichtöne sollen dreimal so lang sein wie die Punkttöne.)

Lösungen

Aufgabe 1

Die folgenden DATA-Anweisungen genügen (andere sind natürlich möglich, wenn Sie eine andere Tonart bevorzugen):

```
2ØØØ  DATA 7, Ø, 4
2Ø1Ø  DATA 7, Ø, 2
2Ø2Ø  DATA 7, Ø, 2
2Ø3Ø  DATA 7, Ø, 4
2Ø4Ø  DATA 8, Ø, 2
2Ø5Ø  DATA 1Ø, Ø, 2
2Ø6Ø  DATA Ø, 1, 2
2Ø7Ø  DATA 1Ø, Ø, 8
```

```
2Ø8Ø  DATA 1Ø, Ø, 2
2Ø9Ø  DATA Ø, 1, 4
21ØØ  DATA Ø, 1, 2
211Ø  DATA 1Ø, Ø, 2
212Ø  DATA 3, 1, 6
213Ø  DATA 1Ø, Ø, 2
214Ø  DATA 8, Ø, 4
215Ø  DATA 7, Ø, 4
216Ø  DATA 7, Ø, 3
217Ø  DATA 8, Ø, 3
218Ø  DATA 1Ø, Ø, 2
219Ø  DATA 5, Ø, 8
22ØØ  DATA 5, Ø, 3
221Ø  DATA 7, Ø, 3
222Ø  DATA 8, Ø, 3
223Ø  DATA 3, Ø, 1Ø
224Ø  DATA 3, Ø, 2
225Ø  DATA 5, Ø, 2
226Ø  DATA 3, Ø, 2
227Ø  DATA 7, Ø, 4
228Ø  DATA 1Ø, Ø, 6
229Ø  DATA 1Ø, –1, 2
23ØØ  DATA 1Ø, –1, 2
231Ø  DATA 1Ø, –1, 2
232Ø  DATA Ø, Ø, 4
233Ø  DATA 3, Ø, 6
234Ø  DATA Ø, Ø, Ø
```

Hier taucht ein kleines Problem auf. Da manche der aufeinanderfolgenden Töne gleich sind, können Sie das Ende des einen vom Anfang des nächsten Tones nicht unterscheiden – er wird einfach durchgehalten.

Wenn Sie den ganzen Abschnitt über Harmonik gelesen haben, werden Sie einen Weg sehen, jeden Ton bewußt abzuschalten, damit man die Töne getrennt hört. (Sie müssen die Zeilen 12Ø und 13Ø ändern.)

Aufgabe 2

Zuerst die SID-Register wie gewohnt initialisieren:

```
5 VOL = 54296: FH = 54273: FL = 54272: WFM = 54276.
  AD = 54277: SR = 54278
```

Das Morseprogramm selbst bleibt unverändert, mit zwei Ausnahmen:

1. Fügen Sie Zeilen 28Ø, 29Ø und 295 ein:

```
280 INPUT "GIB NOTE, OKTAVE EIN"; NTE, OCT
29Ø GOSUB 1ØØØ
295 POKE FH, H%: POKE FL, L%: POKE VOL, 15:
    POKE AD, Ø: POKE SR, 24Ø
```

   Die Subroutine ab Zeile 1ØØØ ist natürlich jene, die eine Frequenz nach den eingegebenen musikalischen Werten berechnet. Schreiben Sie ab von Seite 2Ø1.

2. Rufen Sie in Zeile 37Ø eine Subroutine auf, um den Klang zu erzeugen:

```
37Ø IF C > Ø THEN PRINT CHR$ (C + 64), A$ (C): GOSUB 5ØØØ
```

   Das bringt eine neue Subroutine ins Spiel, die so aussieht:

```
5ØØØ K$ = A$ (C)
5Ø1Ø T = 1
5Ø2Ø L$ = MID$ (K$, T, 1)
5Ø3Ø IF L$ = "." THEN POKE WFM, 17: FOR D = Ø
     TO 1ØØ: NEXT D: POKE WFM, Ø
5Ø4Ø IF L$ = "–" THEN POKE WFM, 17: FOR D = Ø
     TO 3ØØ: NEXT D: POKE WFM, Ø
5Ø5Ø IF L$ = " " OR T = 4 THEN RETURN
5Ø6Ø T = T + 1: GOTO 5Ø2Ø
```

31 Programmplanung

Beim Entwurf eines Programms erleichtert man sich das Dasein oft, wenn man den richtigen Weg wählt, Information im Computer darzustellen. Als Beispiel hier ein Programm, das 'Nullen und Kreuze' spielt.

In Kapitel 14 haben wir uns ein Programm angesehen, wo es von Nutzen war, die beteiligten Subroutinen regelmäßig zu strukturieren. Befassen wir uns nun mit einem, in dem das Programm nicht nur strukturierte Subroutinen, sondern auch strukturierte Daten besitzt.

Es geht darum, den Computer zu veranlassen, daß er auf eine vernünftige Weise mit einem menschlichen Gegner 'Nullen und Kreuze' spielt (eine Art 'Mühlespiel' auf Papier). Ich wähle das, weil die Regeln sehr einfach sind: In ein Gitter von 3 x 3 Feldern setzen die Spieler abwechselnd 0 oder X. Wer als erster drei Zeichen in einer Reihe hat, ist Sieger. Es fällt leicht, sich von Fragen der Spielstrategie ablenken zu lassen und komplizierte Routinen erdenken zu wollen, um die besten Züge zu bestimmen, aber ein bißchen Nachdenken vorher zahlt sich später aus. Zu Beginn muß klar entschieden werden, wie Spielfeld und Spielstand dargestellt werden sollen.

Das Spielfeld

Eine typische Spielsituation sieht etwa so aus:

X	O	
X		X
O		

Darstellen kann man sie etwa dadurch, daß man die Zellen so numeriert:

1	2	3
4	5	6
7	8	9

und dann eine Liste davon anfertigt, was jede Zelle enthält:

1 X
2 O
3 –

4 X

5 –

6 X

7 O

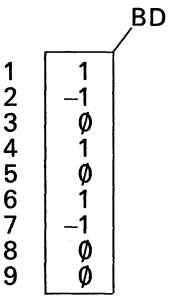
8 –

9 –

Ein Gedankenstrich bedeutet hier eine leere Zelle. Die Symbole X, O, – ersetzt man durch einen numerierten Code:

Ersetze X durch 1
Ersetze O durch –1
Ersetze – durch Ø

(Sie werden später sehen, warum ich gerade diese Zahlen wähle. Im Augenblick ist es nur eine Meinung, ausgelöst durch ein Gefühl für Symmetrie.) Der Spielstand auf dem Feld wird also zu einem *Array* BD:



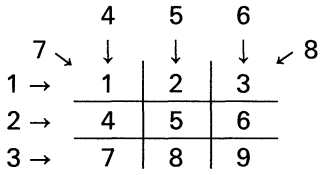
So oft der Spielstand sich ändert, verändern sich auch die Einträge in BD, so daß beispielsweise BD (5) stets den Inhalt der mittleren Zelle des Feldes speichert.

Sie fragen sich vielleicht, warum das besser sein soll als ein Array BD$, das die O- und X-Zeichen als Strings enthält. Nur Geduld, alles kommt an den Tag.

Eine Position bewerten

Das nächste Problem: Wie können wir den Computer zu vernünftigen Zügen zwingen? Es hat keinen Sinn, ihn zufällig spielen zu lassen – stehen beispielsweise in einer Reihe zwei X, so muß er sperren. (Ich gehe durchwegs davon aus, daß der 64 den O-Part spielt.) Wir brauchen also eine Methode, den Spiel*stand* auf irgendeine Weise zu *bewerten,* das heißt, mögliche Züge abzuwägen.

Wir schaffen ein zweites Array namens EV, das Information über den laufenden Stand jeder Reihe, Spalte und Diagonale enthält. Insgesamt gibt es davon acht:



Wenn wir diese Methode, Reihen, Spalten und Diagonale zu numerieren, verwenden, ermitteln wir EV so:

	EV		
1	Ø	= BD(1) + BD(2) + BD(3)	oberste Reihe
2	2	= BD(4) + BD(5) + BD(6)	zweite Reihe
3	–1	= BD(7) + BD(8) + BD(9)	dritte Reihe
4	1	= BD(1) + BD(4) + BD(7)	linke Spalte
5	–1	= BD(2) + BD(5) + BD(8)	mittlere Spalte
6	1	= BD(3) + BD(6) + BD(9)	rechte Spalte
7	1	= BD(1) + BD(5) + BD(9)	vordere Diagonale
8	–1	= BD(3) + BD(5) + BD(7)	hintere Diagonale

Die in EV gezeigten Werte sind die für den konkreten Spielstand oben, und wir sehen, daß sie uns Hinweise auf vernünftige Züge geben. Beachten Sie, daß jeder Wert in EV einfach die *Summe* der Einträge in BD für die betreffende Reihe etc. ist. Das ist ein Grund, die Position nicht in einem *String*array festzuhalten. Aber warum die Summe verwenden?

Sehen Sie sich als erstes EV (2) an, das 2 enthält. Das kann nur geschehen, wenn in dieser Reihe zwei X stehen und *kein* O die Reihe sperrt. Zwei X *und* ein O werden nämlich eine Summe 1 + 1 + (–1) = 1 ergeben, nicht 2. Siehe EV (4), wo genau das stattfindet.

Der springende Punkt ist also der: Wenn irgendwo in EV eine +2 steht, könnte X einen Gewinnzug machen (falls der betreffende Spieler an der Reihe ist). Ebenso bedeutet ein Eintrag –2 in EV, daß O einen Gewinnzug machen könnte. Eintrag –3 bedeutet, daß O soeben gewonnen hat, Eintrag +3, daß X der Sieger ist.

Übrig bleiben die kniffligeren Fälle, wo EV 1, –1 oder Ø enthält, aber wir sehen, daß wir uns darüber den Kopf nicht zu zerbrechen brauchen, weil sie 'entscheidenden' Punkten im taktischen Spiel nicht entsprechen.

Programmstruktur

Die Gesamtstruktur des Programms wird so aussehen:

Funktion	Subroutine Beginn Zeile
Feld initialisieren	1ØØØ
Feld anzeigen	2ØØØ
┌→ Zug des menschlichen Spielers holen	3ØØØ
│ Feld anzeigen	2ØØØ [erneut]
│ Auf Spielende prüfen	4ØØØ
│ Computerzug	5ØØØ
│ Feld anzeigen	2ØØØ [erneut]
└─ Auf Spielende prüfen	4ØØØ [erneut]

Ein paar davon müssen weiter aufgegliedert werden. Beispielsweise muß die 'Spielende'-Prüfroutine das EV-Array erzeugen und dann nach Ergebnissen von 3 und –3 suchen. Sind keine vorhanden, muß sie feststellen, ob das Spiel unentschieden ausgegangen ist, indem sie prüft, ob in BD noch Nullen vorhanden sind. Wenn nicht, sind alle Quadrate verbraucht worden.

Auf Spielende prüfen (4000)

Spielfeld bewerten (6ØØØ)

Wenn in EV –3 vorhanden, siegt der Computer: END

Wenn in EV 3 vorhanden, siegt der Mensch: END

Wenn in BD keine Nullen, Spiel unentschieden: END RETURN

(Rufen Sie sich in Erinnerung, daß der 64 stets O spielt. Es ist nicht schwer, das so abzuändern, daß er je nach Laune X oder O spielt, aber das führt Komplikationen ein, die von der Hauptsache ablenken.)

Wie steht es nun mit der Routine 'Computerzug'? Nun, um vernünftig zu spielen, muß er den Spielstand kennen; er wird also damit beginnen, daß er die Routine 'Spielfeld bewerten' aufruft. Dann sucht er in EV nach –2. Findet er das, weiß er, daß er mit dem nächsten Zug gewinnen kann und es keinen Sinn hat, weiterzusuchen. Im anderen Fall sucht er nach 2, weil das eine Bedrohung darstellt, der begegnet werden kann. Es sei denn, daß 2 *zweimal* vorhanden wäre, weil X dann im nächsten Durchgang gewinnen kann, gleichgültig, was O tut. (In Wirklichkeit stimmt das nicht ganz; es *kann* sein, daß die beiden X-Reihen sich überschneiden und O beide mit einem Zug zu sperren vermag. Das kann allerdings nur vorkommen, wenn bei einem vorherigen Zug X schon zwei Zeichen in einer Reihe hatte . . . so daß der Computer gezwungen gewesen wäre, zu sperren, weil er das Spiel andernfalls einen Zug früher ohnehin schon verloren hätte. Anders ausgedrückt: Die Strategie des Computers sorgt dafür, daß diese Möglichkeit in Wahrheit nie eintritt!)

Wenn es keinen Gewinnzug gibt und keine Drohung abgeblockt werden muß, spielt der Computer einen *Zufalls*zug. (Mit anderen Worten: Er blickt nur

einen Zug voraus. Man könnte weitsichtigere Strategien entwickeln, aber für dieses Spiel sind sie kaum erforderlich.)

Computerzug (5000)

Spielfeld bewerten (6ØØØ)

Wenn in EV –2 vorhanden, Siegeszug (7ØØØ): RETURN

Wenn in E zweimal 2 vorhanden, dann aufgeben: END

Wenn in EV eine 2 vorhanden, dann sperren (8ØØØ): RETURN

Zufallszug spielen (9ØØØ)

RETURN

Wir haben jetzt in der nächstniedrigen Schicht mehrere Routinen erfunden. 'Spielfeld bewerten' ist sehr leicht zu schreiben, und ich sage dazu nichts mehr, bis ich den Code dafür liefere. Siegzug sieht so aus:

Siegzug (7000)

Zufallszug spielen (9ØØØ)

Spielfeld bewerten (6ØØØ)

Wenn EV –3 enthält, RETURN

Zurücknehmen (1ØØØØ)

Der Sinn: Da wir Siegzug überhaupt erreicht haben, *muß* es einen solchen geben. Wir *könnten* eine Routine schreiben, um ihn direkt zu finden, aber es ist einfacher und geht in der Praxis genauso schnell, bloß einen Zufallszug zu spielen und das Spielfeld erneut zu bewerten. Wenn EV –3 enthält, hat der Computer gewonnen. Wenn nicht, war es der falsche Zug; also nehmen wir ihn zurück (das heißt, wir geben BD seinen vorherigen Zustand zurück) und versuchen es noch einmal.

'Sperren' ist ganz ähnlich.

Sperren (8000)

Zufallszug spielen (9ØØØ)

Spielfeld bewerten (6ØØØ)

Wenn EV nicht 2 enthält, RETURN

Zurücknehmen (1ØØØØ)

Alle übrigen Routinen sind ohne die Erfindung weiterer Subroutinen direkt codiert. Jetzt können wir also endlich mit BASIC loslegen!

Der Code

Das Hauptprogramm schreibt sich von selbst:

```
 5  DIM BD(9): DIM EV(8)
1Ø  GOSUB 1ØØØ: REM SPIELFELD INITIALISIEREN
2Ø  GOSUB 2ØØØ: REM ANZEIGEN
3Ø  GOSUB 3ØØØ: REM ZUG HOLEN
4Ø  GOSUB 2ØØØ: REM ANZEIGEN
5Ø  GOSUB 4ØØØ: REM AUF SPIELENDE PRUEFEN
6Ø  GOSUB 5ØØØ: REM COMPUTERZUG
7Ø  GOSUB 2ØØØ: REM ANZEIGEN
8Ø  GOSUB 4ØØØ: REM AUF SPIELENDE PRUEFEN
9Ø  GOTO 3Ø
```

Die *Initialisierungs*routine ist einfach:

```
1ØØØ  FOR P = 1 TO 9
1Ø1Ø  BD(P) = Ø
1Ø2Ø  NEXT P
1Ø3Ø  RETURN
```

Sie fragen sich vielleicht: "Warum die ganze Mühe, wenn das DIM doch alles auf Null setzt?", und das ist gewiß eine gute Frage. Allerdings gibt es eine Anwort darauf: *Sicherheit.* Unter den Aufgaben sind solche, die mehrere Spielpartien hintereinander betreffen, und Sie müßten zwischen den Spielen BD auf Null zurücksetzen. So etwas vergißt sich sehr leicht, wenn man an einem Programm herumbastelt, und man initialisiert üblicherweise Variable, *ohne* zu früheren Werten irgend etwas zu unterstellen.

Hier die *Anzeige*routine:

```
2ØØØ  FOR P = 1 TO 9
2Ø1Ø  IF BD(P) = 1 THEN PRINT "X";
2Ø2Ø  IF BD(P) = -1 THEN PRINT "O";
2Ø3Ø  IF BD(P) = Ø THEN PRINT ".";
2Ø4Ø  IF INT(P/3) = P/3 THEN PRINT
2Ø5Ø  NEXT P
2Ø6Ø  PRINT: PRINT
2Ø7Ø  RETURN
```

Auch das verträgt einen Kommentar. Das BD-Array wird abgesucht nach 1, –1 und Ø, und diese werden in X, O und Punkte umgewandelt. Wir möchten aber, daß die drei ersten Werte in derselben Zeile stehen; aus diesem Grund folgen auf jedes PRINT Strichpunkte, um zu verhindern, daß auf neue Zeilen umgesprungen wird. Wir brauchen aber eine Methode, eine Newline zu erzwingen, wenn O = 3, 6 oder 9 ist, weil sonst das Spielfeld angezeigt werden würde als

```
X X . X . X O . .
```

und nicht als

```
X O .
X . X
O . .
```

Das ist die Funktion von Zeile 2Ø4Ø. Wenn P 3 ist, dann INT(P/3) = 1, was dasselbe ist wie P/3. Ebenso sind beide, wenn P 6 oder 9 ist, 2 beziehungsweise 3. Ist aber etwa P = 4, dann INT(P/3) = 1, aber P/3 ist 1.3333. Nur wenn P ein Vielfaches von 3 ist, sind P/3 und INT(P/3) gleich. (Siehe Kapitel 22). Wir erhalten neue Zeilen also nur für P = 3, 6 und 9.

Ich hätte natürlich den Drückeberger spielen und schreiben können:

```
2Ø4Ø  IF P = 3 OR P = 6 OR P = 9 THEN PRINT
```

aber der Kniff mit INT ist vielseitiger anwendbar.

Es wird Ihnen aufgefallen sein, daß diese Routine ein bißchen primitiv ist. Sie erhalten Anzeigen wie:

```
X   .   .
O   X   .
O   .   O
```

statt:

```
 X |   |
---+---+---
 O | X |
---+---+---
 X |   | O
```

Und das Display ist an die linke Bildschirmseite gequetscht.

Wenn das ganze Programm zufriedenstellend läuft, möchten Sie diese Routine vielleicht durch eine andere ersetzen, die das vertrautere Format liefert. PET-Grafik würde da viel nützen. Das ist übrigens eine der schönen Seiten dabei, Programme auf diese Weise in Subroutinen zu strukturieren. Sie können eine Subroutine herausziehen und durch eine verbesserte oder andersartige Version ersetzen – und Sie können (ziemlich) sicher sein, daß sich das auf andere Programme nicht nachteilig auswirkt, weil es keine GOTO-Befehle gibt, die mehr als eine Subroutine beeinflussen. (Freilich muß man weiterhin darauf achten, keine Variablennamen zu gebrauchen, die sich gegenseitig ins Gehege kommen können.)

Der Zug des menschlichen Spielers ist ebenfalls leicht zu codieren:

```
3000  INPUT "GIB ZUG EIN"; P
3010  IF BD(P) < > 0 THEN PRINT
      "DIESES QUADRAT IST BESETZT": GOTO 3000
3020  BD (P) = 1
3030  RETURN
```

Der Zug wird eingegeben als eine Zahl 1–9. Als erstes testen wir das angemessene Element von BD. Ist es nicht Null, dann steht dort also schon etwas, wir sagen das und springen zurück, um einen gültigen Zug zu verlangen. Im anderen Fall setzen wir das Element von BD auf 1 (für eine X), weil der Mensch an der Reihe ist. 'Auf Spielende prüfen' hält sich ziemlich genau an sein obiges Gerüst:

```
4000  GOSUB 6000
4010  FOR P = 1 TO 8
4020  IF EV(P) = -3 THEN PRINT "ICH GEWINNE": END
4030  IF EV(P) = 3 THEN PRINT "DU GEWINNST": END
4040  NEXT P
4050  FOR P = 1 TO 9
4060  IF BD(P) = 0 THEN RETURN
4070  NEXT P
4080  PRINT "UNENTSCHIEDEN": END
```

Ebenso die Routine 'Computerzug':

```
5000  GOSUB 6000
5010  MT = 0
5020  FOR P = 1 TO 8
5030  IF EV(P) = -2 THEN GOSUB 7000: RETURN
5040  IF EV(P) = 2 THEN MT = MT + 1
5050  NEXT P
5060  IF MT = 2 THEN PRINT "AUCH GUT. ANGEBER!": END
5070  IF MT = 1 THEN GOSUB 8000: RETURN
5080  GOSUB 9000
5090  RETURN
```

Ebenfalls leicht fällt 'Spielfeld bewerten':

```
6000  EV(1) = BD(1) + BD(2) + BD(3)
6010  EV(2) = BD(4) + BD(5) + BD(6)
6020  EV(3) = BD(7) + BD(8) + BD(9)
6030  EV(4) = BD(1) + BD(4) + BD(7)
6040  EV(5) = BD(2) + BD(5) + BD(8)
6050  EV(6) = BD(3) + BD(6) + BD(9)
6060  EV(7) = BD(1) + BD(5) + BD(9)
6070  EV(8) = BD(3) + BD(5) + BD(7)
6080  RETURN
```

Offenkundig sind hier Muster, die man dazu verwenden könnte, in Schleifen zu berechnen, oder Sie verwenden geeignete DATA-Listen . . . aber die Mühe lohnt eigentlich nicht.

Über 'Siegzug' ist nicht viel zu sagen:

```
7000  GOSUB 9000
7010  GOSUB 6000
7020  FOR P = 1 TO 8
7030  IF EV(P) = -3 THEN RETURN
7040  NEXT P
7050  GOSUB 10000
7060  GOTO 7000
```

Und 'Sperren' ist ziemlich ähnlich, sieht man davon ab, daß eine Flagge F für die Entscheidung verwendet wird, ob noch eine 2 vorhanden ist:

```
8000  GOSUB 9000
8010  GOSUB 6000
8020  F = 0
8030  FOR P = 1 TO 8
8040  IF EV(P) = 2 THEN F = 1
8050  NEXT P
8060  IF F = 0 THEN RETURN
8070  GOSUB 10000
8080  GOTO 8000
```

Beachten Sie, daß das RETURN von dieser Routine in Zeile 8Ø6Ø steht. Da wir wissen, daß es einen Sperrzug gibt, wird der Computer ihn schließlich finden: Die Flagge bleibt bei Ø, und der Rücksprung von der Subroutine erfolgt, wie es sich gehört.

Hier 'Zufallszug':

```
9ØØØ  CM = 8 * RND (Ø) + 1
9Ø1Ø  IF BD(CM) < > Ø THEN 9ØØØ
9Ø2Ø  BD(CM) = –1
9Ø3Ø  RETURN
```

Warum ein Zufallszug? Warum nicht einfach P von 1 bis 9 laufen lassen? Das könnten Sie auch tun, aber die Reaktion des Computers wäre dann völlig voraussehbar. Bei einem Spiel vermeidet man das besser.

'Zurücknehmen' erweist sich als das Einfachste überhaupt:

```
1ØØØØ  BD(CM) = Ø
1ØØ1Ø  RETURN
```

Und damit hat sich die Sache! Das Programm hat sich beinahe von selbst geschrieben – nach der harten, langen Arbeit, die für den Aufbau der Struktur aufgewendet worden ist. Was ein kompliziertes Durcheinander von unleserlichen Hieroglyphen hätte werden können, entpuppt sich als lediglich eine Handvoll kurzer, klarer Subroutinen, ergänzt durch eine noch übersichtlichere Steuerroutine, die sie miteinander verbindet. Und Sie haben jetzt einen vernünftigen Gegenspieler.

Aber . . . das ließe sich noch verbessern. (Wie immer. Das ist der Grund, warum fast jedes Programm und fast jeder Computer sich als Ausgabe 3.7 lb vorstellt. Das ursprüngliche Modell wird verbessert, weil Sie, nachdem es läuft, endlich begriffen haben, worum es bei dem Problem in Wirklichkeit geht – und warum Ihre Lösung nicht so gut war, wie Sie dachten.) Ich habe schon empfohlen, das Display zu verbessern. Hier ein paar andere Vorschläge.

Aufgabe 1

Spieler O (der Computer) beginnt stets im Nachteil, weil er erst an zweiter Stelle zieht. (Man kann bei *diesem* Spiel nachweisen, daß das wirklich ein Nachteil ist; bei manchen Spielen trifft das nicht zu!) Seine Verteidigung sollte deshalb möglichst hieb- und stichfest gemacht werden. Vor allem sollte er, wenn er kann, jedesmal das Mittelquadrat (Nummer 5) besetzen. 'Computerzug' sollte deshalb so umgeschrieben werden, daß das geschieht (vorausgesetzt, es besteht keine Bedrohung oder eine Gewinnposition.) Tun Sie das.

Aufgabe 2

Im Augenblick wird nur eine Partie gespielt, dann müssen Sie wieder RUN geben. Ändern Sie das Programm so ab, daß eine Folge von Partien gespielt und Buch geführt wird.

Aufgabe 3

Veranlassen Sie, daß X und O abwechselnd beginnen, aber, wenn O gewinnt, X das nächste Spiel beginnt, und umgekehrt.

Lösungen

Aufgabe 1

So, wie die Routine 'Computerzug' dasteht (5ØØØ), wird, sobald entschieden ist, daß keine Gewinn- oder Sperrzüge zu machen sind, ein Zufallszug getan mit:

```
5Ø8Ø  GOSUB 9ØØØ
```

Kurz *zuvor* sollten Sie deshalb einen Test einschieben, um zu prüfen, ob Quadrat 5 unbesetzt ist, und falls ja, dort spielen:

```
5Ø75  IF BD(5) = Ø THEN BD(5) = –1: RETURN
```

Aufgabe 2

Sie brauchen zwei neue Variable XSCRE und OSCRE (und DR, um die Unentschieden zu zählen, wenn Sie das wollen), die zu Anfang auf Ø gesetzt werden. Gleichzeitig kann eine Spielende-Flagge gesetzt werden:

```
4  XSCRE = Ø: OSCRE = Ø: DR = Ø: EOG = Ø
```

(Auch hier zur Sicherheit: Sie könnten diese Initialisierung weglassen, aber . . .)

Nun müssen Sie alle END-Anweisungen ändern. Bei einem Sieg für den Computer geht OSCRE um 1 höher, EOG wird auf 1 gesetzt. Bei einem Sieg des menschlichen Spielers wird XSCRE um 1 inkrementiert und EOG auf 1 gesetzt. Bei einem Unentschieden wird DR inkrementiert und EOG auf 1 gesetzt.

Aus 4Ø2Ø, 4Ø3Ø und 4Ø8Ø werden also:

```
4Ø2Ø  IF EV(P) = –3 THEN PRINT "ICH GEWINNE":
      OSCRE = OSCRE + 1: EOG = 1: RETURN
4Ø3Ø  IF EV(P) = 3 THEN PRINT "DU GEWINNST":
      XSCRE = XSCRE + 1: EOG = 1: RETURN
4Ø8Ø  PRINT "UNENTSCHIEDEN": DR = DR + 1:
      EOG = 1: RETURN
```

Übrigens wird Zeile 4Ø3Ø nie ausgeführt, weder im Original noch in dieser Abwandlung! Warum nicht?

Es gibt noch eine Stelle, wo das Spiel zu Ende gehen kann, in der Routine 'Sperren', wo der Computer eingesehen hat, daß er unterlegen ist und aufgibt (nicht gerade begeistert). Zeile 5Ø6Ø verändert sich deshalb zu:

```
5Ø6Ø  IF MT = 2 THEN PRINT "AUCH GUT. ANGEBER!":
      XSCRE = XSCRE + 1: EOG = 1: RETURN
```

Nun wird EOG im Hauptprogramm getestet:

```
55  IF EOG = 1 THEN GOSUB 11ØØØ
85  IF EOG = 1 THEN GOSUB 11ØØØ
```

und dann bei 11ØØØ eine Subroutine geschrieben, die sich erkundigt, ob der Spieler noch eine Partie wünscht, und, falls nicht, den Endstand anzeigt.

```
11ØØØ  INPUT "NOCH EIN SPIEL? (J/N)"; QS
11Ø1Ø  IF Q$ = "J" THEN EOG = Ø: GOSUB 1ØØØ: RETURN
11Ø2Ø  PRINT "ENDSTAND"
11Ø3Ø  PRINT "□ X □: "; XSCRE; "□ O □: "; OSCRE;
       "□ UNENTSCHIEDEN □ : □"; DR
11Ø4Ø  END
```

Vergessen Sie nicht, die Spielende-Flagge zurückzusetzen und das Spielfeld auf Zeile 11Ø1Ø neu zu initialisieren!

Wenn Sie das fahren, werden Sie feststellen, daß es einen heimlichen (nun, halbwegs heimlichen) Fehler enthält. *Alle* Spielende-Bedingungen hätten in der Subroutine ab Zeile 4ØØØ behandelt werden müssen. So hat sich in der nächsten Routine in Zeile 5Ø6Ø, eine Macke eingeschlichen. Sie hätte eigentlich so verfaßt werden müssen, daß sie eine Flagge zurückbringt, die von der Routine ab 4ØØØ aufgenommen wird. Nachdem ich Sie aber in diese Klemme gebracht habe, kommen wir am schnellsten wieder heraus, wenn wir die *Display*-Routine abschalten, falls EOG = 1:

```
2ØØØ  IF EOG = 1 THEN RETURN
2ØØ5  FOR P = 1 TO 9
```

Aufgabe 3

Es wird Ihnen aufgefallen sein, daß sie schon gelöst ist! Da X und O abwechselnd spielen und ein neues Spiel in die Schleife dort wieder eintritt, wo sie verlassen wurde, macht, wenn X den letzten Zug einer Partie getan hat, O den ersten der nächsten. Das heißt, die Spielanfänge wechseln in einer Folge von Unentschieden ab (weil ein Unentschieden 9 Züge umfaßt), aber O fängt an, wenn X das letzte Spiel gewonnen hat, und umgekehrt.

32 Hochauflösende Grafik

Das Handbuch erwähnt davon zwar nichts, aber Sie können neben der groben PET-Grafik mit dem 64 auch sehr feine Grafik zeichnen. Das kostet zwar ein bißchen Mühe, aber sie lohnt sich!

Jede Zeichenzelle des TV-Displays besteht in Wahrheit aus einem Quadrat von 8 x 8 winzigen Zellen oder *Pixeln,* mit denen das Zeichen (tief innen in der Elektronik) aufgebaut wird. Dadurch, daß Sie direkten Zugang zu diesen Zellen gewinnen, können Sie auf dem *Hi-res*-Bildschirm (hochauflösend) grafische Displays zeichnen. Das heißt, Sie haben ein Display von 25 x 8 = 2ØØ Reihen und 4Ø x 8 = 32Ø Spalten zur Verfügung. Das ist fast dasselbe Numerierungssystem wie bei den Sprites, aber beschränkt auf den Bildschirmbereich (siehe Abbildung 32.1).

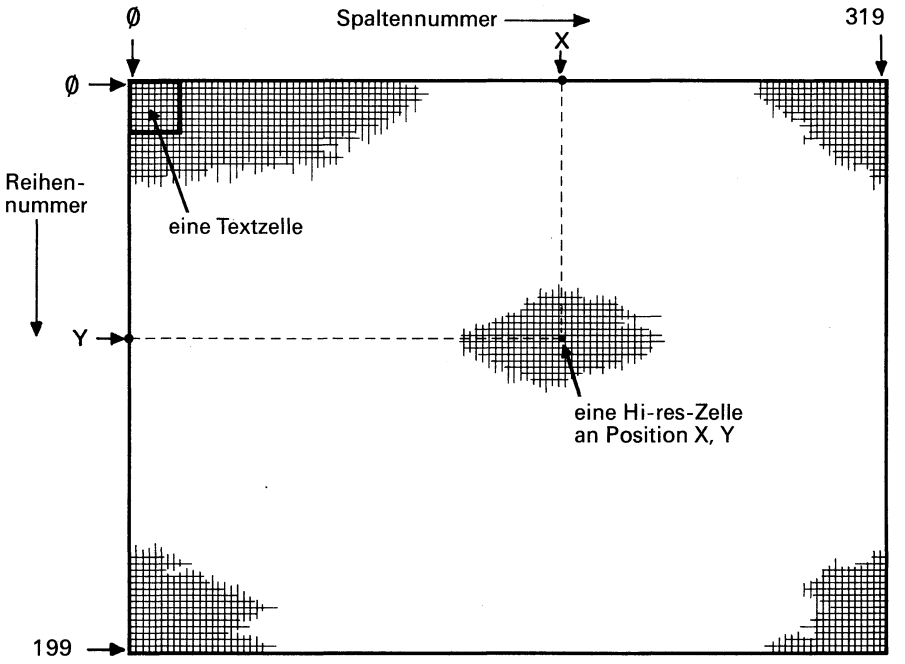


Abbildung 32.1
Struktur des hochauflösenden Bildschirms.

Hochauflösender Modus

Weil Hi-res für Anfänger ein bißchen schwierig ist, gebe ich Ihnen drei wichtige Subroutinen, die Sie abschreiben und in BASIC-Programmen verwenden können.

1. schalten auf hochauflösenden Modus und setzen eines Bildschirmspeicherbereichs;
2. plotten eines Pixels an einer gewählten Position;
3. ziehen einer Linie zwischen zwei gewählten Pixeln.

Um den Computer in hochauflösenden Modus umzuschalten, müssen wir Bit 5 von Adresse 53265 auf 1 setzen (siehe den 'Reference Guide', S. 123). Das geschieht durch den Befehl:

```
POKE 53265, PEEK (53265) OR 32
```

der so funktioniert, wie Kapitel 12 das erklärt. Außerdem muß Speicherplatz für die Bildschirmdisplaydaten reserviert, dieser Platz freigemacht und es müssen Farben zugeteilt werden. Die folgende Routine setzt den Bildschirmspeicherbereich auf Adresse 8192:

```
11ØØØ  REM HI-RES INITIALISIERUNG
11Ø1Ø  POKE 53265, PEEK(53265) OR 32
11Ø2Ø  POKE 53272, PEEK(53272) OR 8
11Ø3Ø  BM = 8192
11Ø4Ø  FOR U = BM TO BM + 7999
11Ø5Ø  POKE U, Ø
11Ø6Ø  NEXT U
11Ø7Ø  FOR U = 1Ø24 TO 2Ø23
11Ø8Ø  POKE U, 13
11Ø9Ø  NEXT U
111ØØ  RETURN
```

Fahren Sie das. Zuerst erhalten Sie ein Durcheinander, dann wird der Schirm leer mit vorwiegend schwarzem Hintergrund, aber mit einigen farbigen Klecksen, wo der Text gewesen ist; zuletzt wird alles hellgrün. (Verändern Sie das 13 in Zeile 11Ø8Ø zu 16 * INK + PAPER, wo INK und PAPER die Farbcodes sind, die Sie wünschen. Dieses Programm liefert INK schwarz auf PAPER hellgrün.)

Beachten Sie, daß das Freimachen des Bildschirmspeichers ziemlich langsam vor sich geht. Bei BASIC dauert es etwa 20 Sekunden. Eine Maschinencode-Routine würde das im Nu bewältigen, aber das ist eine andere Geschichte!

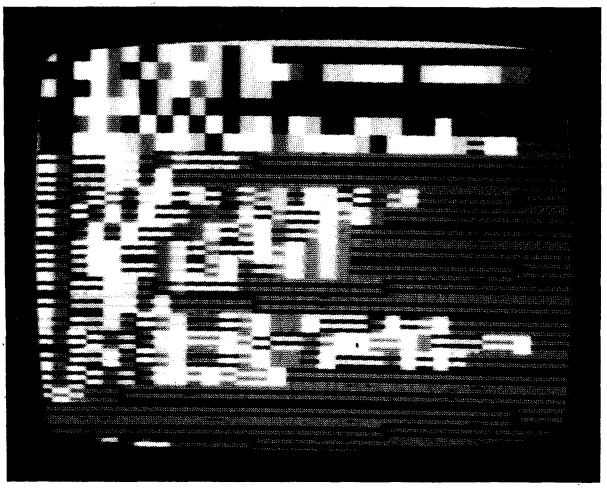


Abbildung 32.2
Wenn der hochauflösende Bildschirm erstmals gesetzt wird, sieht er so aus. Er muß von Daten freigemacht werden, bevor er für Grafik verwendet werden kann.

PLOT

Die Hi-res Spalten und Reihen bestimmen ein System von Koordinaten auf dem Bildschirm, wie Abbildung 32.1 es zeigt. Die Hauptaufgabe ist die, eine Routine zu schreiben, die ein Einzelpixel an Position (X, Y) *PLOT,* das heißt, in Spalte X, Reihe Y. Wenn wir solche Plots kombinieren, können wir Linien, Kurven oder sogar ganze Regionen zeichnen.

Die nächste Routine plottet ein Einzelpixel in Reihe Y, Spalte X. Sie geht davon aus, daß Y zwischen Ø und 199, X zwischen Ø und 32Ø liegt.

```
12ØØØ  REM PLOT X, Y
12Ø1Ø  BY = BM + 32Ø * INT (Y/8) + 8 * INT (X/8) + (Y AND 7)
12Ø2Ø  BT = 7 – (X AND 7)
12Ø3Ø  POKE BY, PEEK (BY) OR (2 ↑ BT)
12Ø4Ø  RETURN
```

Aufgabe 1

Schützen Sie die obige Subroutine so, daß nur zulässige Eingaben für X und Y angenommen werden.

Wie funktioniert das?

Dieser Abschnitt gerät ein bißchen theoretisch. Sie können ihn also überblättern und später darauf zurückkommen.

Jedes Byte im Hi-res-Bildschirmspeicher enthält Daten für eine Reihe von 8 x 1-Pixel auf dem hochauflösenden Bildschirm. Eine binäre Ø bedeutet 'kein Punkt hier', eine 1 'setze hier einen Punkt'. Beispielsweise liefert das Byte 1Ø11Ø1Ø1 somit die in Abbildung 32.3 gezeigte Wirkung.

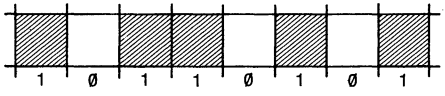


Abbildung 32.2
Pixel mit Biteinteilung.

Wenn Sie die Systemvariable in Adresse 53265 auf hochauflösenden Modus setzen, wird der Computer vom Betriebssystem angewiesen, die Daten auf diese Weise zu interpretieren. Das nennt man *biteingeteilte* Grafik.

die Adressen für unseren hochauflösenden Bildschirmspeicher entsprechen den tatsächlichen Bildschirmpositionen in Tabelle 32.1.

Tabelle 32.1

Reihennummer hochauflösend ↓	Ø	1	2	...	39 ←	Spaltennummer niedrig auflösend
Ø	8182	82ØØ	82Ø8	...	85Ø4	Reihennummer niedrig-auflösend ↓
1	8193	82Ø1	82Ø9	...	85Ø5	
2	8194	82Ø2	821Ø	...	85Ø6	
3	8195	82Ø3	8211	...	85Ø7	Ø
4	8196	82Ø4	8212	...	85Ø8	
5	8197	82Ø5	8213	...	85Ø9	
6	8198	82Ø6	8214	...	851Ø	
7	8199	82Ø7	8215	...	8511	
8	8512	852Ø	...	...	...	1
9	8513	8521	...	...	...	
1Ø	8514	8522	...	...	...	
11	8515	8523	...	...	...	
12	8516	8524	...	...	...	
13	8517	8525	...	...	...	
14	8518	8526	...	...	...	
15	8519	8527	...	...	...	
.	...	...	...	...	...	.
.	...	...	...	...	...	.
.	...	...	...	...	...	.

Jede Zeichenzelle, die im Bildschirmspeicher *einer* Adresse zu entsprechen pflegte, entspricht also nun *acht* Adressen, einem Speicherblock von acht Bytes Länge. Die Blöcke sind in derselben Reihenfolge angeordnet wie die Zellen im Bildschirmspeicher – zuerst die niedrigauflösenden Reihen entlanggehen und nach Spalte 39 eine Reihe hinunterspringen.

Angenommen, wir möchten in die linke obere Ecke eine Diagonallinie von 5 Pixel Länge setzen. Die Adressen und Inhalte nehmen die Form von Abbildung 32.4 an.

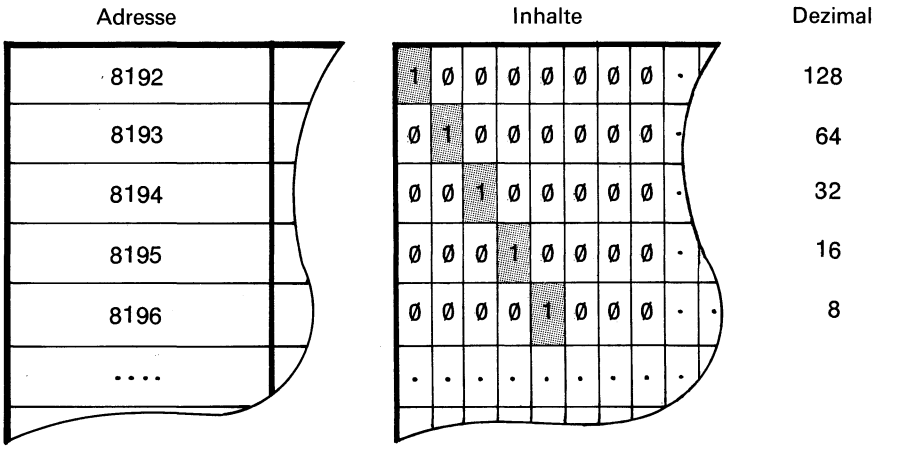


Abbildung 32.4
Pixel für eine Diagonallinie zusammensetzen.

Dieses Programm sollte das also bewältigen:

```
1Ø  GOSUB 11ØØØ          [Hi-res-Subroutine eingeben]
2Ø  POKE 8192, 128
3Ø  POKE 8193, 64
4Ø  POKE 8194, 32
5Ø  POKE 8195, 16
6Ø  POKE 8196, 8
7Ø  GOTO 7Ø
```

Probieren Sie es aus.

Dieselbe Methode allgemein:

1. Die in Frage kommende Adresse finden.
2. Mit POKE den erforderlichen Wert eingeben, um das gewünschte Bildschirmdisplay zu erzielen.

Da wir nichts löschen wollen, was sich schon auf dem Bildschirm befindet, müssen wir unterstellen, daß die Adresse einen Wert enthalten kann, der nicht

Null ist. Dazu müssen wir den Inhalt mit dem neuen Wert in OR-Verbindung bringen, wie in Kapitel 12.

Zeile 12Ø1Ø berechnet die richtige Adresse.

Zeile 12Ø2Ø berechnet den Wert, der mit POKE eingegeben werden muß, um ein neues Pixel zu plotten.

Zeile 12Ø3Ø stellt die OR-Verbindung mit dem vorhandenen Inhalt wieder her und setzt das Ergebnis mit POKE wieder hinein.

Weitere Einzelheiten im 'Reference Guide', S. 125.

Spirale

Hier ein Beispiel dafür, wie man diese Routinen nutzt.

```
1Ø  GOSUB 11ØØØ
2Ø  FOR T = 1 TO 1ØØØ
3Ø  X = 16Ø + T * SIN (T/1Ø)/1Ø
4Ø  Y = 1ØØ + T * COS (T/1Ø)/1Ø
5Ø  GOSUB 12ØØØ
6Ø  NEXT T
7Ø  GOTO 7Ø
```

Vergessen Sie die beiden obigen Subroutinen nicht! Dieses Programm hier nutzt die trigonometrischen Funktionen SIN und COS, um die Spirale zu zeichnen.

Wenn Sie sich am Ergebnis lange genug ergötzt haben, drücken Sie RUN/STOP + RESTORE.

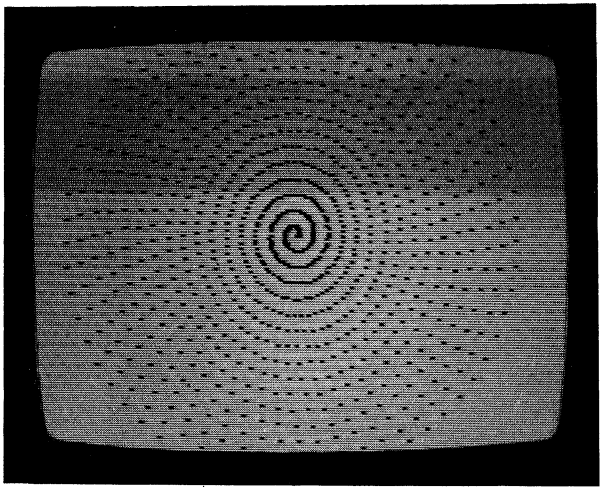


Abbildung 32.5
Eine Spirale aus 1000 geplotteten Bits.

DRAW

Die nächste Aufgabe ist die, eine Routine zu entwickeln, die eine (ziemlich) gerade Linie zwischen Punkten mit den Koordinaten (A, B) und (C, D) wie in Abbildung 32.6 zeichnet.

Das wird ein bißchen mathematisch, so daß ich auf die Feinheiten nicht eingehe. Auf jeden Fall kommt dabei heraus:

```
13ØØØ  REM DRAW (A, B) TO (C, D)
13Ø1Ø  IF A > C THEN AØ = C: C = A: A = AØ: BØ = D: D = B: B = BØ
13Ø2Ø  CA = C – A: DB = D – B
13Ø3Ø  IF CA = Ø THEN 131ØØ
13Ø4Ø  IF DB < = CA AND DB > = –CA THEN 132ØØ
13Ø5Ø  S = SGN(DB)
13Ø6Ø  FOR V = Ø TO DB STEP S
13Ø7Ø  X = V * CA/DB + A: Y = V + B: GOSUB 12ØØØ
13Ø8Ø  NEXT V
13Ø9Ø  RETURN
131ØØ  IF DB = Ø THEN RETURN
1311Ø  S = SGN(DB)
1312Ø  FOR V = B TO D STEP S
1313Ø  X = A: Y = V: GOSUB 12ØØØ
1314Ø  NEXT V
1315Ø  RETURN
132ØØ  FOR V = Ø TO CA
1321Ø  Y = V * DB/CA + B: X = V + A: GOSUB 12ØØØ
1322Ø  NEXT V
1323Ø  RETURN
```

Das SGN in 13Ø5Ø und 1311Ø ist die *Vorzeichen*funktion. SGN (J) ist 1, wenn J > Ø, Ø, wenn J = Ø, und –1, wenn J < Ø. Der Rest ist Koordinationsgeometrie.

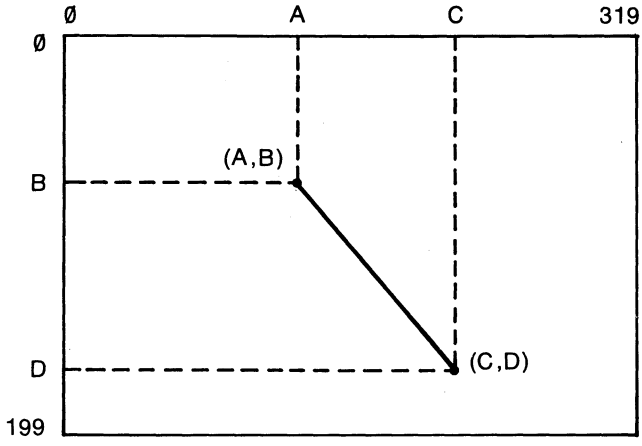


Abbildung 32.6
Eine Linie zwischen zwei Punkten zeichnen.

Speichen

Dieses Programm zeichnet Radiallinien in Speichenart.

```
1Ø  GOSUB 11ØØØ
2Ø  FOR T = 1 TO 3Ø
3Ø  A = 16Ø: B = 1ØØ
4Ø  C = 16Ø + 9Ø * COS(π * T/15)
5Ø  D = 1ØØ + 9Ø * SIN(π * T/15)
6Ø  GOSUB 13ØØØ
7Ø  NEXT T
8Ø  GOTO 8Ø
```

Verändern Sie das 3Ø in Zeile 2Ø zu (meinetwegen) 2Ø oder 4Ø, wenn es weniger oder mehr Speichen werden sollen. Vergessen sie nicht, daß auch die Standardroutinen bei 11ØØØ, 12ØØØ und 13ØØØ eingegeben werden müssen!

Vielecke

Hier noch ein Programm:

```
10  INPUT "ZAHL DER SEITEN"; N
20  GOSUB 11000
30  FOR T = 0 TO N - 1
40  U = T + 1
50  A = 160 + 90 * COS(2 * π * T/N)
60  B = 100 + 90 * SIN(2 * π * T/N)
70  C = 160 + 90 * COS(2 * π * U/N)
80  D = 100 + 90 * SIN(2 * π * U/N)
90  GOSUB 13000
100 NEXT T
110 GOTO 110
```

Versuchen Sie das der Reihe nach mit N = 5, 6, 7 . . . zu fahren. Nach jedem Lauf RUN/STOP + RESTORE.

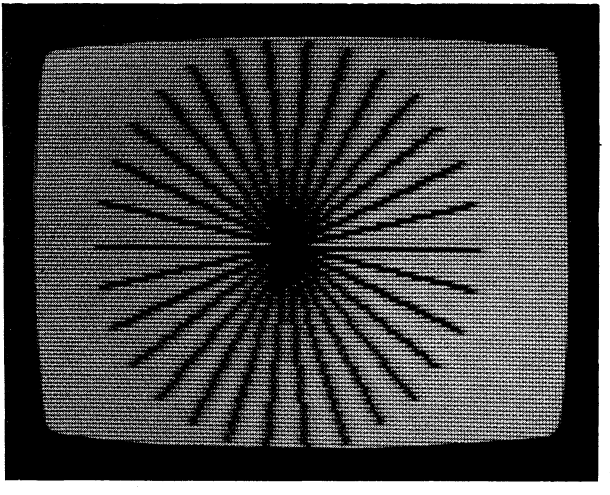


Abbildung 32.7
Speichen.

Computerkunst

Zum Schluß ein Programm, das Zufallsvielecke zeichnet, bis Sie es durch Druck auf Taste 'S' anhalten

```
10  GOSUB 11000
20  CX = 300 + 260 * RND(0): CY = 300 + 140 * RND(0)
30  RAD = 200 * RND(0) + 5
50  N = INT(4 + 6 * RND(0))
60  FOR T = 0 TO N - 1
70  U = T + 1
80  A = CX + RAD * COS(2 * π *T/N)
90  B = CY + RAD * SIN(2 * π * T/N)
100 C = CX + RAD *COS(2 * π * U/N)
110 D = CY + RAD * SIN(2 * π * U/N)
120 GOSUB 13000
130 NEXT T
140 GET A$: IF A$ < > "S" THEN 20
150 GOTO 150
```

Sie brauchen trotzdem RUN/STOP + RESTORE, um ganz aufzuhören.

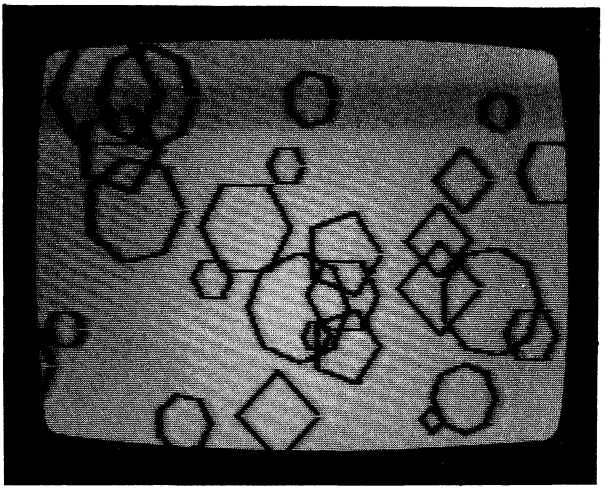


Abbildung 32.8
Computerkunst mit Zufallsvielecken.

Aufgabe 2

Schreiben Sie ein Programm, um in hochauflösendem Modus ein Rechteck mit den Spaltennummern XL und XR für die linken und rechten Seiten und den beiden Zahlen YT, YB für Ober- und Unterseite zu zeichnen.

Lösungen

Aufgabe 1

Fügen Sie die Zeilen an:

```
12ØØ2  IF X < Ø OR X > 319 THEN FLAGGE = 1: RETURN
12ØØ4  IF Y < Ø OR Y > 199 THEN FLAGGE = 1: RETURN
```

Fügen Sie dann in Ihr Hauptprogramm einige Zeilen ein, um die Flagge aufzunehmen, (vielleicht) eine Fehlermeldung anzuzeigen und X und Y in dieser Richtung neu zu definieren:

```
764  IF FLAGGE = 1 THEN PRINT "HOPPLA, VOM BILDSCHIRM
     GERUTSCHT": FLAGGE = Ø:
     GOTO (dorthin, wo X und Y gesetzt sind)
     etc.
```

Es hängt vom Programm ab.

Aufgabe 2

```
 1Ø  INPUT "LINKE SPALTE"; XL
 2Ø  INPUT "RECHTE SPALTE"; XR
 3Ø  INPUT "OBERE REIHE"; YO
 4Ø  INPUT "UNTERE REIHE"; YU
 5Ø  GOSUB 11ØØØ
 6Ø  A = XL: B = YO; C = XR: D = YO: GOSUB 13ØØØ
 7Ø  A = XR: B = YU: C = XR: D = YO: GOSUB 13ØØØ
 8Ø  A = XR: B = YU: C = XL: D = YU: GOSUB 13ØØØ
 9Ø  A = XL: B = YO: C = XL: D = YU: GOSUB 13ØØØ
1ØØ  GOTO 1ØØ
```

Ich habe mir nicht die Mühe gemacht, die INPUT-Anweisungen zu schützen; Sie können das, wenn Sie wollen, leicht nachholen.

33 Debugging VI

Manchmal sind Zahlen, die gleich aussehen, gar nicht gleich!

Abrundungsfehler

Die Art von Fehlern, nach denen wir bisher suchten, stammten von uns selbst und waren, einmal erkannt, verhältnismäßig leicht zu beheben. Es gibt einen anderen Fehler, der durch den Aufbau des Computers selbst entsteht. Das ist kein Konstruktionsfehler, sondern eine Folge dessen, wie alle Computer angelegt sind. Er wird bedingt durch die Art der Genauigkeit, mit der Computer Zahlen speichern. Wenn wir uns überlegen, wie Zahlen normalerweise festgehalten werden, liegt nahe, daß es für die Anzahl der Ziffern, die zu bewältigen sind, eine Grenze gibt. Ein Auto-Tachometer, beispielsweise, kann nur 5 Ziffern bewältigen, weil er nur fünf 'Fenster' hat. Beim Computer ist es genauso. Jede Zahl kann nicht mehr besetzen als eine festgelegte Zahl von 'Fenstern'. Nicht jedes Fenster steht aber für eine Dezimalziffer. Der interne Maschinencode für Computer ist ganz anders angelegt, als wir uns Zahlen vorstellen; mit den grausigen Einzelheiten will ich Sie nicht langweilen. Die Tatsache, daß von Natur aus Ungenauigkeit besteht *und* eine Codeumwandlung angewendet wird, bedeutet, daß die äußere Darstellung einer Zahl (wie sie auf dem Bildschirm erscheint) durchaus nicht dieselbe sein muß wie die innere Darstellung. Ich will das, was ich meine, durch ein Beispiel mit Logarithmen der Schulzeit erläutern. Wenn Sie 2 mit 2 logarithmisch multiplizieren, erhalten Sie:

Zahl	log
2	Ø.3Ø1Ø
2	Ø.3Ø1Ø
3.999	Ø.6Ø2Ø +

also 2 x 2 = 3.999!!

Die Kombination der Tatsache, daß Logarithmen nur auf 4 Stellen genau sind (das heißt, nur 4 Fenster besetzen dürfen), und daß eine Codeumwandlung stattfindet (Zahl in Logarithmus, Logarithmus zurück in Zahl), erzeugt die Ungenauigkeit.

Hier ein Programm, das dieselbe Art von Problem hervorruft:

```
1Ø  FOR P = 1 TO 1Ø
2Ø  S = SQR(P)
3Ø  Q = S ↑ 2
4Ø  IF P < > Q THEN PRINT P, Q
5Ø  NEXT P
```

Nehmen Sie den Fall, wo P = 9. In Zeile 2Ø wird die Wurzel gezogen, also S = 3. Wir erheben S in Zeile 3Ø ins Quadrat, also Q = 9, wie P. P wird Q natürlich

immer gleich sein, weil der Rechenvorgang des Wurzelziehens aus dem Quadrat gefolgt vom Erheben ins Quadrat dorthin zurückführen muß, wo man hergekommen ist. Zeile 4Ø ist demnach unsinnig, weil P sich von Q nie unterscheidet. Es wird also nichts angezeigt.

Oder doch? Fahren Sie das Programm einmal. Sie erhalten:

```
3    3
5    5.ØØØØØØØ1
6    6.ØØØØØØØ1
7    7.ØØØØØØØ1
9    9.ØØØØØØØ1
1Ø   1Ø
```

Das ist allerdings ein überaus seltsames Ergebnis, weil der Computer nicht nur Werte anzeigt und damit behauptet, sie wären verschieden, sondern sie auch noch so anzeigt, als wären sie gleich! Was passiert ist? Die verwickelten mathematischen Prozesse haben zu kleinen Ungenauigkeiten in der internen Zahlendarstellung geführt, die verantwortlich sind für die Unterschiede zwischen P und Q. Ungenauigkeiten treten aber *auch* auf, wenn das interne Format entschlüsselt wird zu den auf dem Bildschirm angezeigten Dezimalzahlen, so daß sie identisch zu sein *scheinen,* obwohl der Computer steif und fest behauptet, sie wären es nicht. Bei manchen Werten sind die internen Codes in der Tat identisch – etwa bei 8.

Diese Art von Fehler kann außerordentlich verwirrend sein, und manchmal bleibt nur der Ausweg, in die IF-Anweisung einen kleinen Fehler einzubauen, also:

```
IF ABS(P – Q) < Ø.ØØØ1 THEN . . .
```

Die ABS-Funktion ist notwendig, weil Q größer sein könnte als P und das Resultat dann negativ wäre, demnach kleiner als Ø.ØØØ1, obwohl sein Wert sehr groß sein könnte (–3Ø ist kleiner als Ø.ØØØ1!) Das ABS schneidet das Minuszeichen ab.

Potenzen

Rundungsfehler treten besonders häufig auf, wenn Sie für die Potenz N einer Zahl

```
T ↑ N
```

verwenden. Probieren Sie Folgendes:

```
1Ø  FOR T = 1 TO 1Ø
2Ø  PRINT T * T, T ↑ 2
3Ø  NEXT T
```

Seltsames findet statt bei T = 7 und 9, nämlich Rundung. Wenn Sie (sagen wir) die fünfte Potenz einer *ganzen* Zahl berechnen wollen, verwenden Sie zutreffender:

```
T * T * T * T *T
```

statt

```
T ↑ 5
```

T ↑ N wird nämlich berechnet als

```
EXP (N * LOG (T))
```

(für diejenigen, die mit Exponenten und Logarithmen vertraut sind).

Beispiel: Sie suchen nach ganzzahligen Lösungen A, B, C der Gleichung:

$$A^2 + B^2 = C^2$$

Sie könnten in Versuchung geraten, es so zu probieren:

```
1Ø  FOR A = 1 TO 1Ø
2Ø  FOR B = 1 TO 1Ø
3Ø  C = SQR (A ↑ 2 + B ↑ 2)
4Ø  IF C = INT (C) THEN PRINT A, B, C
5Ø  NEXT B
6Ø  NEXT C
```

Das scheitert kläglich, weil es keine Lösungen findet, obwohl (beispielsweise) 3, 4, 5 eine Lösung in dem abgesuchten Bereich wäre. Der Sünder ist der Rundungsfehler. Der Fehler mit ↑ kann beseitigt werden, wenn man A ↑ 2 und B ↑ 2 durch A * A und B * B ersetzt. Das SQR schafft schon mehr Probleme, aber ein Ausweg wäre:

```
1Ø  FOR A = 1 TO 1Ø
2Ø  FOR B = 1 TO 1Ø
3Ø  C = INT (SQR (A * A + B * B))
4Ø  IF A * A + B * B = C THEN PRINT A, B, C
5Ø  NEXT B
6Ø  NEXT A
```

Das funktioniert, zum Teil deswegen, weil SQR zu hoch zu schätzen scheint. Als zusätzliche Sicherheit fügen Sie ein:

```
45  IF A * A + B * B = (C + 1) * (C + 1) THEN PRINT A, B, C + 1
```

für den Fall, daß INT etwas abgerundet hat, das ein bißchen kleiner ist als eine ganze Zahl.

(Das ist, mathematisch gesehen, ohnehin eine höchst unzweckmäßige Weise, sich mit dem Problem auseinanderzusetzen, aber der Fehler kommt so häufig vor, daß er Erwähnung verdient.)

Integervariable A%, B%, C% können die Erfolgsaussicht ebenfalls verbessern – aber nicht bei Fehlern von zu *niedriger* Schätzung.

34 Files (Dateien)

Der Speicher des 64 mag für die meisten Zwecke recht groß erscheinen, aber er reicht nicht immer aus. Wenn man sehr große Datenmengen oder kleine Mengen für Spezialzwecke zu verarbeiten hat, sind zusätzliche Speicherarten nötig.

Zu den Dingen, die Computer besonders gut können, gehört, große Datenmengen zu speichern und durchzugehen. Aber wie machen sie das? Schließlich hat der Speicher des Commodore 64 nur um die 39ØØØ Bytes (Zeichen) Speicherplatz, und das reicht nicht weit, wenn Sie die Werke Goethes speichern oder einen Katalog für ein großes Kunstmuseum aufstellen wollen.

Diese Seite enthält rund 18ØØ Zeichen (3Ø Zeilen zu je 6Ø Anschlägen, Leerräume mitgerechnet). Der 64 könnte auf einmal also rund 22 Seiten dieses Buches aufnehmen. Und das, *bevor* wir den Speicherplatz abgezogen haben, den das Programm für die Analyse der Daten besetzt – es ist sogar noch schlimmer, als es den Anschein hat!

Dazu kommt noch ein Problem: Die Daten werden nur gespeichert, solange der Computer eingeschaltet ist, also müßten wir den 64 für 22 Seiten 'Faust' (oder sonst irgend etwas) reservieren und nie mehr abschalten.

Das ist ohne Zweifel unsinnig; es muß einen besseren Weg geben. Was wir brauchen, ist ein Datenspeicher, der *nichtflüchtig* ist (soll heißen, der nicht die Sinne verliert, wenn man ihm den Strom abschaltet). Mit einem solchen Speichersystem sind wir bereits gut vertraut: Tonbandkassetten. Der einzige Unterschied: Wir möchten es dazu verwenden, Daten zu speichern, während wir ihn vorher nur für die Aufzeichnung von Programmen genutzt haben.

Kassettendateien

Ein Datensatz, der als Einheit auf Band festgehalten wird, heißt *File* oder Datei. Sie können sich, wenn Sie wollen, auch ein Programm als File vorstellen, und so, wie wir Programmen Namen geben, benennen wir auch Datenfiles.

Wenn wir ein Programm sichern, schreiben wir etwa:

```
SAVE "FRED"
```

Das hat zwei Wirkungen. Zuerst wird der Dateiname 'FRED' auf das Band geschrieben, dann das Programm.

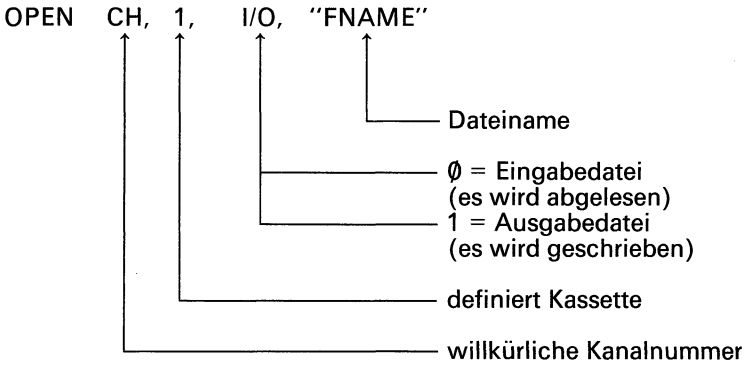
Wenn Sie Daten auf ein Kassettenfile ausgeben möchten, sind diese beiden Schritte ebenfalls notwendig, aber sie geschehen gleichzeitig. Mit anderen Worten: Sie müssen zuerst einen Befehl geben, der die Datei benennt, und dann einen zweiten, der ihr Daten einliest.

Um eine Datei zu benennen, schreiben Sie:

```
OPEN 1, 1, 1, "FRED"
```

Das '1, 1, 1' bedarf einer Erklärung. Das erste '1' ist eine Filenummer (oder Kanalnummer), und Sie können dafür jeden beliebigen Wert nehmen. Ich habe sie '1' genannt, weil sie die erste ist, die ich vergebe. Die zweite '1' bezeichnet das verwendete Gerät. Das ist '1' für einen Kasettenrecorder, '4' für einen Drucker, '8' für ein Diskettenlaufwerk, und so weiter. Diese Werte können Sie *nicht* ändern. Sie sind vom System festgelegt. Das dritte '1' teilt dem System mit, ob vom Gerät gelesen oder in dieses etwas geschrieben werden soll. (Es hat für verschiedene Geräte verschiedene Bedeutung, aber da wir hier nur von Kasettenrecordern sprechen, will ich die Dinge nicht dadurch komplizieren, daß ich seine Bedeutung für andere Peripherie mitteile.)

Die allgemeine Form einer OPEN-Anweisung somit *nur* bei Kassetten lautet:



Gut. Probieren Sie zum Anfang:

```
1Ø  CH = 5
2Ø  OPEN CH, 1, 1, "FRED"
3Ø  FOR N = 1 TO 1Ø
4Ø  PRINT# CH, N
5Ø  PRINT N
6Ø  NEXT N
7Ø  CLOSE CH
```

Zeile 2Ø benennt die Datei 'FRED' und bestimmt einen Ausgabekanal (5) dafür. Zeile 4Ø schreibt die Zahlen 1 bis 1Ø in den Kanal und damit in die Datei (FRED), die damit in Verbindung gebracht wird. Beachten Sie die Form dieser Anweisung. Sie ist wie eine gewöhnliche PRINT-Anweisung, bis auf das '# CH', das die Wirkung hat, die Ausgabe zu Kanal 5 zu steuern. Als nächstes (Zeile 5Ø) werden die Zahlen auf dem Bildschirm angezeigt. Nach dem Ende der Schleife wird die Datei *geschlossen* (Zeile 7Ø). Sie bemerken, daß der Dateiname nur in der OPEN-Anweisung erwähnt wird. Überall sonst wird lediglich die Kanalnummer angegeben.

Wenn sie das fahren, erhalten Sie die Aufforderung:

PRESS RECORD & PLAY ON TAPE

Sorgen Sie dafür, daß eine Kassette eingelegt ist und drücken Sie RECORD und PLAY gleichzeitig. Achten Sie genau auf den Bildschirm. Sie sehen, daß er leer wird und die Kassette automatisch läuft. Während dieser Zeit gehorcht der Computer Zeile 2Ø und schreibt den Dateinamen auf einen 'Kopfblock' des Tonbands. Dann leuchtet kurz ein Bildschirmdisplay auf, während die Zahlen 1 bis 1Ø angezeigt werden. Während dieser kurzen Zeit bleibt das Band stehen! Der Schirm wird wieder leer, das Band läuft weiter, das Display wird schließlich wiederhergestellt und die Kassette bleibt stehen.

Wenn Sie es sich recht überlegen, ist das ja merkwürdig. Die Anweisungen für die Ausgabe zum Band sind schließlich mit denen verwoben, die das Display anzeigen, so daß man annehmen möchte, das Band liefe weiter, *während* die Zahlen auf dem Sichtgerät angezeigt werden. Der Grund für dieses seltsame Verhalten ist der, daß der Computer Ihnen nicht wirklich gehorcht, wenn Sie verlangen, er solle einen Wert an eine Datei ausgeben. Er speichert ihn vorübergehend in einem Speicherbereich, der *Puffer* genannt wird, bis er soviel Daten hat, daß es lohnt, sie auf das Kassettenband zu übertragen (das heißt, wenn der Puffer voll ist). Nun füllen die Zahlen 1 bis 1Ø den Pufferspeicher nicht völlig. Sollten Sie also nichts weiter unternehmen, würden auf das Band überhaupt keine Daten gelangen! Dazu ist die CLOSE-Anweisung in Zeile 7Ø da. Sie teilt dem 64 mit, daß in die mit Kanal CH verbundene Datei keine weiteren Daten mehr geschrieben werden sollen, er also ausgeben muß, was er gerade im Puffer hat. Das nennt man den 'Puffer leeren'.

SchreibenSie Zeile 3Ø um zu

```
3Ø  FOR N = 1 TO 2ØØ
```

und fahren Sie wieder mit RUN. Jetzt werden Sie die Pufferorganisation sehr deutlich sehen, weil mehrere Puffer voll Daten zu übertragen sind und Sie zwischen den Übertragungen der einzelnen Blöcke Displays geliefert bekommen.

Das Band sieht also so ähnlich aus, wie Abbildung 34.1 es zeigt:

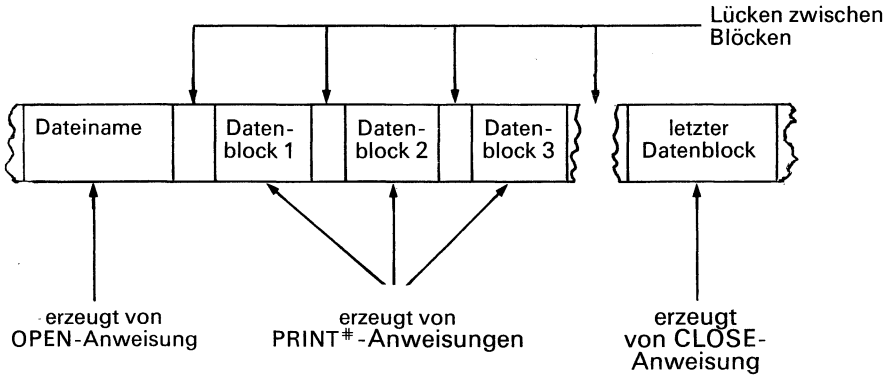


Abbildung 34.1
Struktur einer Datei auf Kassette.

Für Sie, den Anwender, spielt im einzelnen nichts davon eine Rolle, solange Sie nicht vergessen, nach Beendigung einer Ausgabe in eine Datei eine CLOSE-Anweisung anzufügen; rechnen Sie einfach nicht damit, daß die Kassette sofort nach jeder PRINT#-Anweisung zu laufen beginnt. Wenn Sie wissen, was wirklich vorgeht, werden Ihnen weniger Fehler unterlaufen – oder schlimmstenfalls werden Sie Ihnen erklärlich sein, wenn sie passieren. (Das ist ein bißchen wie beim Autofahren: Sie *brauchen* nicht zu wissen, wie ein Getriebe funktioniert, um den Gang zu wechseln – aber es erklärt, warum das Schalten lohnt, und es erklärt auch das gräßliche Knirschen, das manchmal auftritt, wenn Sie das Kupplungspedal nicht richtig durchgedrückt haben.)

Die Datei lesen

Nun wollen wir sehen, ob wir die Daten auch wieder von der Kassette holen können. Fügen Sie diesen Zeilen an:

```
80 END
100 OPEN CH, 1, 0, "FRED"  [Öffne Kanal 5 für Ablesen aus 'FRED']
110 INPUT# CH, A
120 PRINT A
130 IF A < > 200 THEN 110
```

Spulen Sie die Kassette zurück und fahren Sie mit GOTO 100 (*nicht* RUN 100 – Sie verlieren sonst den Wert in CH). Sie werden aufgefordert, das Band mit PLAY laufen zu lassen, und das System sucht nach dem Kopfblock mit 'FRED'. Sobald es Erfolg gehabt hat, sieht es sich sofort einer Aufforderung gegenüber, vom Band abzulesen (Zeile 110), so daß auf der Stelle ein zweiter Block gelesen wird. Nun wird der erste Block Zahlen angezeigt, dann der nächste Block gelesen, und so weiter, bis die Zahl 200 gelesen wird, wo das Programm in den Abgrund kippt und stehenbleibt.

Ich habe vorgeschlagen, Programme im Speicher gleichzeitig zu 'lesen' und 'schreiben', damit Sie mit diesen Vorstellungen mühelos experimentieren können und irgendwelche Anfangsprobleme nicht zu katastrophal werden. Beispielsweise kommen Sie vielleicht dahinter, daß Sie versuchen, auf Vorspannband zu schreiben, wenn Sie vergessen haben, die Kassette vor dem Einlegen in den Recorder vorzuspulen. In diesem Fall ist der Kopfblock nicht gesichert, und die Leseroutine kann ihn nicht finden, also müssen Sie den ganzen Ablauf wiederholen.

Anwendung: Geburtstage

Das ist ja nun alles hochinteressant, aber ist es auch nützlich? Doch, ja, und das in mehr als einer Beziehung. Ein Beispiel: Nehmen wir an, der Computer soll einige verwickelte mathematische Aufgaben bewältigen, die mehrere Stunden in Anspruch nehmen. Klar, daß Sie nicht vor dem Ding sitzenbleiben wollen, bis endlich die Antwort geliefert wird, aber wenn Sie nicht gerade einen Drucker

haben, müßten Sie das tun, zumindest dann, wenn die Gefahr besteht, daß die Werte, die Sie brauchen, durch spätere Ausgaben vom Bildschirm gescrollt werden. Die Alternative ist die, die Ausgabe in eine Datei zu bewirken, damit Sie lesen können, wann sie Zeit haben. Und selbst wenn Sie einen Drucker *besitzen,* kann sich die Methode lohnen, wenn ein langes Programm über Nacht laufen soll. Es kann höchst ärgerlich sein, wenn zwischen Mitternacht und 6 Uhr morgens ein lauter Drucker alle 20 Minuten losrattert.

Eine viel alltäglichere Verwendung von Dateien ist die, einen organisierten Vorrat an alphabetischen wie numerischen Daten anzulegen. Zum Beispiel möchten Sie vielleicht eine Adressliste aller Ihrer Freunde und Bekannten haben – ein computerisiertes Adreßbuch, wenn Sie so wollen. Offensichtlich müssen Sie Namen, Adressen und Rufnummern festhalten, aber es wäre hübsch, auch noch zusätzliche Informationen zu besitzen, etwa die, ob Sie normalerweise einer bestimmten Person zum Geburtstag oder zu Weihnachten eine Glückwunschkarte oder auch ein Geschenk schicken. Wenn Sie Geburtstagsgeschenke schicken, wird es nützlich sein, auch Geburtsdaten in die Datei aufzunehmen. Auf diese Weise sollten wir Programme schreiben können, die in der Datei nach Personen suchen, denen Sie beispielsweise im März irgendeinen Geburtstagsgruß senden. Nie mehr kann es vorkommen, daß sie irgendeinen Geburtstag vergessen!

Aber das kommt später. Unsere erste Aufgabe ist die, für die Datei zu sorgen. Jede Person hat einen Eintrag, der besteht aus:

Name

Adresse

Geburtsdatum

Einzelheiten zu Geschenk und Glückwunschkarte

Wir nennen das einen *Datensatz.* (Um die Dinge zu vereinfachen, wollen wir Rufnummern und dergleichen hier vergessen.) Jeden der darin enthaltenen Posten, etwa die Adresse, nennen wir ein *Datenfeld.*

Die beiden ersten Felder stellen keine Probleme dar. Wir müssen aber ein Standardformat für das Geburtsdatum festlegen und dabei bleiben. Schließlich hat es keinen Sinn, dem Computer aufzutragen, er möge nach NOV suchen, wenn die Monate als Zahlen abgespeichert sind, so daß er eigentlich nach 11 suchen müßte.

Überhaupt wird es am einfachsten sein, Zahlen zu verwenden, so daß

Ø2Ø96Ø

bedeutet:

Ø2 2.

Ø9 September (Monat 9)

6Ø 196Ø (das 19 lassen Sie weg)

Außerdem brauchen wir eine geeignete Methode, die Glückwunschinformation zu codieren. Dafür gibt es viele Möglichkeiten. Wie wäre es, sie zu numerieren, etwa so:

Karte zum Geburtstag	1
Geschenk zum Geburtstag	2
Karte zu Weihnachten	3
Geschenk zu Weihnachten	4

so daß der Code 13 lautet, wenn Sie jemandem zum Geburtstag *und* zu Weihnachten eine Karte schicken, aber keine Geschenke.

Ein Programm folgender Art würde die Datei erzeugen:

```
 5  CH = 1: KAS = 1: IN = Ø: AUS = 1
 1Ø  OPEN CH, KAS, AUS, "MLIST"
 2Ø  INPUT "NAME";N$
 3Ø  PRINT# CH, N$
 4Ø  INPUT "ADRESSE"; AD$
 5Ø  PRINT# CH, AD$
 6Ø  INPUT "GEBURTSDATUM"; BD$
 7Ø  PRINT# CH, BD$
 8Ø  INPUT "GLUECKWUNSCHCODES", G$
 9Ø  PRINT# CH, G$
1ØØ  INPUT "WEITERE DATEN (J/N)"; Q$
11Ø  IF QS = "J" THEN 2Ø
115  PRINT# CH, "* * * *"
12Ø  CLOSE KAS
13Ø  END
```

Obwohl der Code sehr einfach erscheint, sind ein paar Bemerkungen angebracht.

Erstens ist es verlockend, *alle* Felder einzugeben und dann zu schreiben:

```
PRINT# CH, N$, AD$, BD$, G$
```

Der Haken dabei: PRINT# ruft *genau* dieselbe Wirkung bei der Datei hervor wie PRINT beim Bildschirm. Wenn also N$ = "HUGO", AD$ = HAUBENGASSE 48 OSTHOFEN", BD$ = "11Ø737" und G$ = "3", haben Sie in der Datei:

```
HUGO      HAUBENGASSE 48 OSTHOFEN      11Ø737      3
```

(dazu diverse Leerstellen zwischendrin). Wenn Sie das zurückzuholen versuchen mit

```
INPUT# CH, N$, AD$, BD$, G$
```

geht es nicht. Der *ganze* Datensatz wird nach N$ übertragen, weil die Felder nicht durch Kommas getrennt sind, was INPUT erwartet.

Sie könnten die Kommas bewußt einstreuen:

```
PRINT#, CH, N$; ","; AD$; ","; BD$; ","; G$
```

aber ich finde das häßlich und ziehe getrennte PRINT#-Anweisungen vor.

Zweitens dürfen aus denselben Gründen keine Kommas in den Adreßfeldern stehen.

Drittens: Beachten Sie, daß die letzten beiden Felder zwar Zahlen sind, ich sie aber wie Strings behandelt habe. Den Grund werden Sie später erkennen.

Viertens: Sehen Sie sich Zeile 115 an. Sie schreibt an das Ende der Datei einen Begrenzer (* * * *), und zwar in das 'Namens'-Feld des Satzes nach dem letzten eingegebenen. Wir brauchen das, um zu bestimmen, ob schon die ganze Datei gelesen worden ist, wenn wir sie absuchen.

Sehen Sie sich schließlich Zeile 5 an. Ich habe sie dazu benützt, sinnvolle Abkürzungen für die Dateioperationen zu definieren, die normalerweise von Zahlen angezeigt werden. Beispielsweise kann ich jetzt in einer OPEN-Anweisung statt '1' 'AUS' schreiben, um anzuzeigen, daß in die Datei geschrieben werden soll. Das kommt sprachlich der Anweisung 'Öffne einen Kanal für Kassette zur Ausgabe' näher als 'OPEN 1, 1, 1 . . .'

Aufgabe 1

Zur Zeit muß der Anwender sich die Glückwunschcodes noch merken. Ersetzen Sie Zeile 8Ø durch:

```
8Ø  GOSUB 1ØØØ
```

und schreiben Sie in Zeile 1ØØØ eine Subroutine, die dem Anwender eine Reihe von Fragen stellt wie:

```
HAST DU EINE GEBURTSTAGSKARTE GESCHICKT?
```

und G$ automatisch erzeugt.

Die Datei absuchen

Wir möchten Fragen beantworten können wie: "An welche Geburtstage muß ich mich im Juni erinnern und was muß ich den Leuten schicken?" Schreiben wir also zunächst ein Programm, das genau dies leistet. Es wird uns einige Hinweise darauf liefern, wie wir es später allgemeiner fassen können.

Als erstes müssen wir die Datei öffnen:

```
5  CH = 1: KAS = 1: IN = Ø: AUS = 1
1Ø  OPEN CH, KAS, IN, "MLIST"
```

und dann einen Datensatz lesen:

```
2Ø  INPUT# CH, N$
25  IF N$ = "* * * *" THEN END          [auf Begrenzer prüfen]
3Ø  INPUT# AD$
4Ø  INPUT# CH, BD$
5Ø  INPUT# CH, G$
```

Nun wird festgestellt ob dieser Datensatz für uns von Interesse ist:

```
6Ø  IF MID$ (BD$, 2, 2) = "Ø6" THEN GOSUB 5ØØ: REM HAB EINEN
```

Mit anderen Worten: Ist der Geburtsmonat Juni? Ich verwende MID$ (siehe Kapitel 17), um die mittleren beiden Zeichen von BD$ auszuwählen und sie mit Ø6 zu vergleichen. Deshalb habe ich das Geburtsdatum im *String*format genommen. Es ist viel einfacher, die Mitte eines Strings als mehrere Ziffern einer Zahl auszuwählen. Und ich kann ihn oder Teile davon mit VAL (Kapitel 17) jederzeit in Zahlen zurückverwandeln, wenn es sein muß. Haben wir jemanden gefunden, der im Juni geboren ist, wird die Steuerung der Subroutine ab 5ØØ übertragen, und darüber zerbrechen wir uns später den Kopf.

Nun den nächsten Datensatz holen:

```
7Ø  GOTO 2Ø
```

Die 'Hab-einen'-Routine

Nun zur Subroutine. Wir müssen G$ durchgehen auf der Suche nach 1 (Karte) und 2 (Geschenk) oder beidem.

```
5ØØ  FLAGGE = Ø
51Ø  FOR P = 1 TO 4
52Ø  IF MID$ (G$, P, 1) = "1" THEN FLAGGE = 1: PRINT "KARTE"
53Ø  IF MID$ (G$, P, 1) = "2" THEN FLAGGE = 1: PRINT "GE-
     SCHENK"
54Ø  NEXT P
55Ø  IF FLAGGE = Ø THEN RETURN
56Ø  PRINT
57Ø  PRINT N$
58Ø  PRINT AD$
59Ø  PRINT: PRINT
6ØØ  RETURN
```

Wir gehen G$ auf der Suche nach '1' oder '2' durch. Wenn wir '1' finden, wird KARTE angezeigt, bei 2 GESCHENK. In beiden Fällen wird eine *Flagge* auf 1 gesetzt, die ursprünglich Null war. Wenn wir also aus der Schleife kommen und die Flagge immer noch Null ist, wissen wir, daß wir den Geburtstag dieser Person nicht berücksichtigen, so daß es keinen Sinn hat, Namen und Adresse anzuzeigen. Daher Zeile 55Ø, die dafür sorgt, daß wir es *nicht* tun. Ist die Flagge aber auf 1 gesetzt, schicken wir ihm oder ihr etwas; das Programm macht also weiter und zeigt Namen und Adresse in geeignetem Format an, bevor es zurückspringt.

Aufgabe 2

Wandeln Sie diese Routine so ab, daß sie für jeden Monat verwendet werden kann.

Aufgabe 3

Schreiben Sie eine Routine, die Weihnachtskarten und -geschenke bewältigt und veranlassen Sie den Computer, Ihnen mitzuteilen, wieviele Weihnachtskarten Sie kaufen müssen.

Die Ausgabedatei wechseln

Wenn Sie eine PRINT-Anweisung geben, geht das System normalerweise davon aus, Sie wünschen die Ausgabe auf dem Bildschirm des Sichtgeräts. Wie wir gesehen haben, können Daten dadurch, daß die Form einer PRINT-Anweisung geändert wird, zu jedem gewünschten Gerät geschickt werden. Was ich aber noch nicht erklärt habe, ist, daß Daten sogar ohne Änderung der ganzen PRINT-Anweisungen umgelenkt werden können.

Das geschieht mit dem CMD-Befehl, dessen Grundform lautet:

```
CMD Kanal
```

Wenn ich schreibe:

```
CMD 5
```

schaltet das die Ausgabe vom Bildschirm zum Kanal 5. Kanal 5 muß natürlich vorher in einer OPEN-Anweisung errichtet worden sein, etwa durch

```
OPEN 5, KAS, AUS, "BILDSCHIRMABZUG"  [unterstellt, daß KAS und
                                      AUS wie vorher auf 1 ge-
                                      setzt sind]
```

Nun werden alle ursprünglich für den Bildschirm bestimmten Daten auf die Kassette übertragen. Die einzigen Ausnahmen sind Fehlermeldungen, die nach wie vor auf dem Bildschirm erscheinen.

Um das System in seinen ursprünglichen Zustand (das heißt, Ausgabe auf den Bildschirm) zurückzuversetzen, wird zuerst eine Nullzeile (mit anderen Worten, einfach eine Newline) zur Datei ausgegeben:

```
PRINT# 5
```

und die Datei dann geschlossen:

```
CLOSE 5
```

Hier tritt eine kleine Seltsamkeit auf. Man möchte annehmen, man werde statt 'PRINT# 5' einfach 'PRINT' schreiben können, weil zu Kanal 5 ohnehin Daten ausgegeben werden, aber das kann man nicht. Fragen Sie mich nicht nach dem Grund.

Aufgabe 4

Erinnern Sie sich an die Ablaufüberwachung und Profile in Debugging IV? Es gibt einen Weg, eine Ablaufüberwachung so zu verwenden, daß sie ein Profil liefert.

Das geht so: Schreiben Sie das Programm unter Einschluß von Überwachungsanweisungen wie in Debugging IV, geben Sie die Überwachungsausgaben aber zum Kassettenrecorder aus. Die <- und >-Zeichen um die Zeilennummern lassen Sie dabei weg. Sie sind nicht nötig, weil die Überwachungseinzelheiten jetzt zu einem anderen Platz als der normalen Ausgabe gehen und das Dasein beim nächsten Schritt außerdem ein bißchen schwieriger wird.

Der nächste Schritt: Schreiben Sie ein Programm, das eine Zeilennummer anfordert, dann die Datei liest, die Sie eben geschaffen haben, und zählt, wie oft diese Zeilennummer vorkommt. Zeigen Sie diesen Wert an, dann wissen Sie, wie oft die entsprechende Anweisung ausgeführt worden ist!

Lösungen

Aufgabe 1

```
1ØØØ  G$ = "": REM G$ IST ZU ANFANG AUF NULL GESETZT
1Ø1Ø  INPUT "SCHICKST DU EINE GEBURTSTAGSKARTE?
      (J/N)"; Q$
1Ø2Ø  IF Q$ = "J" THEN G$ = G$ + "1"
1Ø3Ø  INPUT "SCHICKST DU EIN GEBURTSTAGSGESCHENK?
      (J/N)"; G$
1Ø4Ø  IF Q$ = "J" THEN G$ = G$ + "2"
1Ø5Ø  INPUT "SCHICKST DU EINE WEIHNACHTSKARTE? (J/N)"; Q$
```

```
1Ø6Ø  IF Q$ = "J" THEN G$ = G$ + "3"
1Ø7Ø  INPUT "SCHICKST DU EIN WEIHNACHTSGESCHENK?
      (J/N)"; Q$
1Ø8Ø  IF Q$ = "J" THEN G$ = G$ + "4"
1Ø9Ø  RETURN
```

Sehen Sie, wie G$ schrittweise aufgebaut wird?

Die Ähnlichkeit der Fragen und die Wirkung auf G$ sollten eine Alternative nahelegen. Wie wäre es, wenn wir zu Beginn des Programms das folgende Stringarray aufbauen:

	BC$
1	GEBURTSTAGSKARTE
2	GEBURTSTAGSGESCHENK
3	WEIHNACHTSKARTE
4	WEIHNACHTSGESCHENK

Dann könnten wir schreiben:

```
1ØØØ  G$ = " "
1Ø1Ø  FOR P = 1 TO 4
1Ø2Ø  PRINT "SCHICKST DU EIN/EINE□"; BC$ (P):
1Ø3Ø  INPUT Q$
1Ø4Ø  IF Q$ = "J" THEN G$ = G$ + STR$ (P)
1Ø5Ø  NEXT P
1Ø6Ø  RETURN
```

Beachten Sie, daß wir in Zeile 1Ø4Ø mit STR$ P in seine Stringentsprechung verwandeln müssen.

In diesem Fall lohnt die Mühe eigentlich nicht, weil die drei ersparten Zeilen mehr als ausgeglichen werden, wenn wir BC$ setzen. Gibt es aber mehrere zusätzliche Bedingungen, dann wäre das eine geeignete Methode.

Aufgabe 2

Das einfachste ist, eine zusätzliche Zeile einzubauen:

```
15  INPUT "GIB MONAT ALS ZAHL VON ZWEI ZIFFERN EIN"; MTH$
```

und Zeile 6Ø umzuschreiben:

```
6Ø  IF MID$ (BD$, 2, 2) = MTH$ THEN GOSUB 5ØØ
```

Noch besser: Sie könnten gegen zufällige Eingabe einer Einzelziffer (etwa 5 statt Ø5) schützen mit:

```
16  IF LEN (MTH$) = 1 THEN MTH$ = "Ø" + MTH$
```

Aufgabe 3

Hier brauchen wir keine Dateninformation, also wird aus Zeile 6Ø:

```
6Ø  GOSUB 5ØØ
```

und 52Ø und 53Ø verändern sich zu:

```
52Ø  IF MID$ (G$, P, 1) = "3" THEN NK = NK + 1: FLAGGE = 1:
     PRINT "KARTE"
53Ø  IF MID$ (G$, P, 1) = "4" THEN FLAGGE = 1: PRINT "GE-
     SCHENK"
```

Jedesmal, wenn eine Weihnachtskarte verlangt ist, wird eine Variable namens NK um 1 erhöht, so daß wir sicherheitshalber für den Anfang auf Null setzen sollten.:

```
6  NK = Ø
```

Und am Ende müssen wir anzeigen:

```
75  PRINT "ZAHL DER ERFORDERLICHEN KARTEN:"; NK
```

Aufgabe 4

Ihr Originalprogramm muß eine Datei zuteilen:

```
1Ø  OPEN 3, KAS, AUS, "UEBERWACHUNG"  [unterstellt, daß KAS
                                       und AUS wie vorher
                                       definiert sind
```

und dann in jeder Anweisung, die verfolgt werden soll, PRINT# 3 enthalten:

```
15Ø  PRINT#, 3, "15Ø": REM DIESE ANWEISUNG VERFOLGEN
```

Ans Ende des Programms, aber bevor die Datei geschlossen wird, schreiben Sie einen geeigneten Begrenzer für die Datei:

```
8ØØ  PRINT# 3, "–1"
81Ø  CLOSE 3
82Ø  END
```

Um das Profil einer bestimmten Zeile zu erhalten, könnten Sie schreiben:

```
1Ø  IN = Ø: KAS = 1: ZAEHLUNG = Ø
2Ø  OPEN 5, KAS, IN, "UEBERWACHUNG"
3Ø  INPUT "GEWUENSCHTE ZEILENNUMMER"; N
4Ø  INPUT 5, N
5Ø  IF N < Ø THEN PRINT ZAEHLUNG: END
6Ø  IF L = N THEN ZAEHLUNG = ZAEHLUNG + 1
7Ø  GOTO 4Ø
```

Einfach und praktisch!

Sie könnten das erweitern, um das Profil einer Anzahl von Zeilen zu erhalten, ohne das Band zurückspulen zu müssen. Sie brauchen dafür ein Array, das die für das Profil vorgesehenen Zeilen festhält, Zeile 3Ø wird zu einer Schleife, in die alle gewünschten Zeilennummern eingegeben werden, und Zeile 6Ø zum Aufruf einer Subroutine, die das Array nach einer Entsprechung absucht.

Den Code überlasse ich Ihnen.

Anhang 1
Byteumwandlung Binär/Dezimal

Binär	Dezimal	Binär	Dezimal	Binär	Dezimal	Binär	Dezimal
ØØØØØØØØ	Ø	ØØ1ØØØØØ	32	Ø1ØØØØØØ	64	Ø11ØØØØØ	96
ØØØØØØØ1	1	ØØ1ØØØØ1	33	Ø1ØØØØØ1	65	Ø11ØØØØ1	97
ØØØØØØ1Ø	2	ØØ1ØØØ1Ø	34	Ø1ØØØØ1Ø	66	Ø11ØØØ1Ø	98
ØØØØØ11	3	ØØ1ØØØ11	35	Ø1ØØØ11	67	Ø11ØØ011	99
ØØØØ1ØØ	4	ØØ1ØØ1ØØ	36	Ø1ØØ1ØØ	68	Ø11ØØ1ØØ	1ØØ
ØØØØ1Ø1	5	ØØ1ØØ1Ø1	37	Ø1ØØ1Ø1	69	Ø11ØØ1Ø1	1Ø1
ØØØØ11Ø	6	ØØ1ØØ11Ø	38	Ø1ØØ11Ø	7Ø	Ø11ØØ11Ø	1Ø2
ØØØØ111	7	ØØ1ØØ111	39	Ø1ØØ111	71	Ø11ØØ111	1Ø3
ØØØ1ØØØ	8	ØØ1Ø1ØØØ	4Ø	Ø1ØØ1ØØØ	72	Ø11Ø1ØØØ	1Ø4
ØØØ1ØØ1	9	ØØ1Ø1ØØ1	41	Ø1ØØ1ØØ1	73	Ø11Ø1ØØ1	1Ø5
ØØØ1Ø1Ø	1Ø	ØØ1Ø1Ø1Ø	42	Ø1ØØ1Ø1Ø	74	Ø11Ø1Ø1Ø	1Ø6
ØØØ1Ø11	11	ØØ1Ø1Ø11	43	Ø1ØØ1Ø11	75	Ø11Ø1Ø11	1Ø7
ØØØ11ØØ	12	ØØ1Ø11ØØ	44	Ø1ØØ11ØØ	76	Ø11Ø11ØØ	1Ø8
ØØØ11Ø1	13	ØØ1Ø11Ø1	45	Ø1ØØ11Ø1	77	Ø11Ø11Ø1	1Ø9
ØØØ111Ø	14	ØØ1Ø111Ø	46	Ø1ØØ111Ø	78	Ø11Ø111Ø	11Ø
ØØØ1111	15	ØØ1Ø1111	47	Ø1ØØ1111	79	Ø11Ø1111	111
ØØ1ØØØØ	16	ØØ11ØØØØ	48	Ø1Ø1ØØØØ	8Ø	Ø1111ØØØ	112
ØØ1ØØØ1	17	ØØ11ØØØ1	49	Ø1Ø1ØØØ1	81	Ø1111ØØ1	113
ØØ1ØØ1Ø	18	ØØ11ØØ1Ø	5Ø	Ø1Ø1ØØ1Ø	82	Ø1111Ø01Ø	114
ØØ1ØØ11	19	ØØ11ØØ11	51	Ø1Ø1ØØ11	83	Ø1111Ø011	115
ØØ1Ø1ØØ	2Ø	ØØ11Ø1ØØ	52	Ø1Ø1Ø1ØØ	84	Ø1111Ø1ØØ	116
ØØ1Ø1Ø1	21	ØØ11Ø1Ø1	53	Ø1Ø1Ø1Ø1	85	Ø1111Ø1Ø1	117
ØØ1Ø11Ø	22	ØØ11Ø11Ø	54	Ø1Ø1Ø11Ø	86	Ø1111Ø11Ø	118
ØØ1Ø111	23	ØØ11Ø111	55	Ø1Ø1Ø111	87	Ø1111Ø111	119
ØØ11ØØØ	24	ØØ111ØØØ	56	Ø1Ø11ØØØ	88	Ø11111ØØØ	12Ø
ØØ11ØØ1	25	ØØ111ØØ1	57	Ø1Ø11ØØ1	89	Ø11111ØØ1	121
ØØ11Ø1Ø	26	ØØ111Ø1Ø	58	Ø1Ø11Ø1Ø	9Ø	Ø11111Ø1Ø	122
ØØ11Ø11	27	ØØ111Ø11	59	Ø1Ø11Ø11	91	Ø11111Ø11	123
ØØ111ØØ	28	ØØ1111ØØ	6Ø	Ø1Ø111ØØ	92	Ø11111ØØ	124
ØØ111Ø1	29	ØØ1111Ø1	61	Ø1Ø111Ø1	93	Ø11111Ø1	125
ØØ1111Ø	3Ø	ØØ11111Ø	62	Ø1Ø1111Ø	94	Ø1111110	126
ØØ11111	31	ØØ111111	63	Ø1Ø11111	95	Ø1111111	127

Binär	Dezimal	Binär	Dezimal	Binär	Dezimal	Binär	Dezimal
10000000	128	10100000	160	11000000	192	11100000	224
10000001	129	10100001	161	11000001	193	11100001	225
10000010	130	10100010	162	11000010	194	11100010	226
10000011	131	10100011	163	11000011	195	11100011	227
10000100	132	10100100	164	11000100	196	11100100	228
10000101	133	10100101	165	11000101	197	11100101	229
10000110	134	10100110	166	11000110	198	11100110	230
10000111	135	10100111	167	11000111	199	11100111	231
10001000	136	10101000	168	11001000	200	11101000	232
10001001	137	10101001	169	11001001	201	11101001	233
10001010	138	10101010	170	11001010	202	11101010	234
10001011	139	10101011	171	11001011	203	11101011	235
10001100	140	10101100	172	11001100	204	11101100	236
10001101	141	10101101	173	11001101	205	11101101	237
10001110	142	10101110	174	11001110	206	11101110	238
10001111	143	10101111	175	11001111	207	11101111	239
10010000	144	10110000	176	11010000	208	11110000	240
10010001	145	10110001	177	11010001	209	11110001	241
10010010	146	10110010	178	11010010	210	11110010	242
10010011	147	10110011	179	11010011	211	11110011	243
10010100	148	10110100	180	11010100	212	11110100	244
10010101	149	10110101	181	11010101	213	11110101	245
10010110	150	10110110	182	11010110	214	11110110	246
10010111	151	10110111	183	11010111	215	11110111	247
10011000	152	10111000	184	11011000	216	11111000	248
10011001	153	10111001	185	11011001	217	11111001	249
10011010	154	10111010	186	11011010	218	11111010	250
10011011	155	10111011	187	11011011	219	11111011	251
10011100	156	10111100	188	11011100	220	11111100	252
10011101	157	10111101	189	11011101	221	11111101	253
10011110	158	10111110	190	11011110	222	11111110	254
10011111	159	10111111	191	11011111	223	11111111	255

Anhang 2
Sprite-Register, leicht gemacht

Adresse	Inhalt								Funktion
V + Ø	Spaltennummer Sprite Ø								Spritepositionen
V + 1	Reihennummer Sprite Ø								
V + 2	Spaltennummer Sprite 1								
V + 3	Reihennummer Sprite 1								
V + 4	Spaltennummer Sprite 2								
V + 5	Reihennummer Sprite 2								
V + 6	Spaltennummer Sprite 3								
V + 7	Reihennummer Sprite 3								
V + 8	Spaltennummer Sprite 4								
V + 9	Reihennummer Sprite 4								
V + 1Ø	Spaltennummer Sprite 5								
V + 11	Reihennummer Sprite 5								
V + 12	Spaltennummer Sprite 6								
V + 13	Reihennummer Sprite 6								
V + 14	Spaltennummer Sprite 7								
V + 15	Reihennummer Sprite 7								
V + 16	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	Versetzungsflagge
V + 21	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	Aktivieren/Sperren
V + 23	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	erweitern vertikal
V + 29	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	erweitern horizontal
V + 3Ø	Sp7	Sp6	Sp5	Sp4	Sp3	Sp2	Sp1	SpØ	Kollisionsflagge
V + 39	Farbcode Sprite Ø								Farben
V + 4Ø	Farbcode Sprite 1								
V + 41	Farbcode Sprite 2								
V + 42	Farbcode Sprite 3								
V + 43	Farbcode Sprite 4								
V + 44	Farbcode Sprite 5								
V + 45	Farbcode Sprite 6								
V + 46	Farbcode Sprite 7								
2Ø4Ø	Zeiger Sprite Ø								Zeiger
2Ø41	Zeiger Sprite 1								
2Ø42	Zeiger Sprite 2								
2Ø43	Zeiger Sprite 3								
2Ø44	Zeiger Sprite 4								
2Ø45	Zeiger Sprite 5								
2Ø46	Zeiger Sprite 6								
2Ø47	Zeiger Sprite 7								

Anhang 3
Bibliothek von Sprite-Subroutinen

Diese Sammlung von Routinen soll den Umgang mit Sprites erleichtern. Sie sollten aus dem Hauptprogramm mit GOSUB aufgerufen werden, nachdem die notwendigen Parameter gesetzt sind. Wie immer ist V = 53248 die Startadresse der Sprite-Register.

Zeiger für Sprite K auf PTR setzen

```
20000  POKE 2040 = K, PTR
20010  RETURN
```

Daten für Sprite K mit POKE in PTR-Block abspeichern

```
20100  FOR G = 0 TO 62
20110  READ H
20120  POKE 64 * PTR + G, H
20130  NEXT G
20140  RETURN
```

(Diese Routine unterstellt, daß die erforderlichen DATA-Anweisungen im Hauptprogramm gegeben sind.)

Sprite K einschalten

```
20200  POKE V + 21, (PEEK (V + 21)) OR 2 ↑ K
20210  RETURN
```

Sprite K abschalten

```
20300  POKE V + 21, (PEEK (V + 21)) AND (255 – 2 ↑ K)
20310  RETURN
```

Sprite K auf Farbe CR setzen

```
20400  POKE V + 39 + K, CR
20410  RETURN
```

Sprite K in Reihe Y, Spalte X setzen

```
20500  POKE V + 2 * K + 1, Y
20510  IF X > 255 THEN OF = 1
20520  POKE V + 2 * K, X – 256 * OF
```

```
2Ø53Ø  IF OF = 1 THEN POKE V + 16, PEEK(V + 16) OR 2 ↑ K
2Ø54Ø  IF OF = Ø THEN POKE V + 16, PEEK(V + 16) AND (255 – 2 ↑ K)
2Ø55Ø  OF = Ø
2Ø56Ø  RETURN
```

Sprite K in horizontaler Richtung erweitern

```
2Ø6ØØ  POKE V + 29, PEEK (V + 29) OR 2 ↑ K
2Ø61Ø  RETURN
```

Sprite K in horizontaler Richtung zusammenziehen

```
2Ø7ØØ  POKE V + 29, PEEK (V + 29) AND (255 – 2 ↑ K)
2Ø71Ø  RETURN
```

Sprite K in vertikaler Richtung erweitern

```
2Ø8ØØ  POKE V + 23, PEEK (V + 23) OR 2 ↑ K
2Ø81Ø  RETURN
```

Sprite K in vertikaler Richtung zusammenziehen

```
2Ø9ØØ  POKE V + 23, PEEK (V + 23) AND (255 – 2 ↑ K)
2Ø91Ø  RETURN
```

Zusammenstoß von Sprite K mit anderem Sprite feststellen

```
21ØØØ  IF (PEEK (V + 3Ø) AND 2 ↑ K) = 2 ↑ K THEN GOSUB 3ØØØØ
21Ø1Ø  RETURN
```

(3ØØØØ ist eine Subroutine, die eingreift, wenn der Zusammenstoß stattfindet.)

Zusammenstoß zwischen Sprite K und L feststellen

```
211ØØ  IF (PEEK (V + 30) AND (2 ↑ K + 2 ↑ L)) = 2 ↑ K + 2 ↑ L THEN
3ØØØØ
2111Ø  RETURN
```

Zusammenstoß zwischen Sprite K und Text feststellen

```
212ØØ  IF (PEEK (V + 31) AND 2 ↑ K) = 2 ↑ K THEN 3ØØØØ
2121Ø  RETURN
```

Anhang 4
Klangchip-Register, einfach gemacht

S = 54272

Adresse			Inhalt		Funktion
Stimme 1	Stimme 2	Stimme 3			
S+Ø	S+ 7	S+14	Niedriges Frequenzbyte		Tonhöhe setzen
S+1	S+ 8	S+15	Hohes Frequenzbyte		
S+2	S+ 9	S+16	Niedriges Pulsbyte (nur Puls-Wellenform)		Art der Puls-Wellenform setzen
S+3	S+10	S+17	Hohes Pulsbyte (nur Puls-Wellenform)		
S+4	S+11	S+18	Wellenform-Code*		'Instrument' wählen
S+5	S+12	S+19	4 Bit-Anschlagwert	4 Bit-Abschwellwert	Anschlag/ Abschwellen
S+6	S+13	S+26	4 Bit Haltewert	4 Bit Ausklangwert	Halten/ Ausklingen
S+24			Lautstärke (0–15)		Lautstärke

*Wellenform-Codes:
17: Dreieck
33: Sägezahn
65: Puls
129: Rauschen

Anhang 5
Speicherkarte Commodore 64

Das ist ein einfacher Führer zu den Hauptabschnitten des Commodore 64-Speichers. Weitere Einzelheiten siehe Anhang 6 und 'Reference Guide'. Beachten Sie, daß manche Abschnitte mehr als eine Verwendung haben, was vom Zusammenhang und davon abhängt, wie manche Systemvariable gesetzt sind.

Adressen	Inhalt
Ø–827	Verwendung durch System
828–1Ø23	Kassettenpuffer + ungenutzter Platz
1Ø24–2Ø23	Bildschirmspeicher (Video-RAM)
2Ø4Ø–2Ø47	Datenzeiger für Sprites
2Ø48–4Ø959	Normaler Programmplatz für BASIC
4Ø96Ø–49151	BASIC ROM (bei PEEK) *oder* 8K RAM
49152–53247	4K RAM
53248–57343	Input/Output-Geräte und Farbspeicher *oder* Zeichen-ROM (siehe Kapitel 13) *oder* 4K RAM
53248–54271	VIC-Chip (Sprites + Videodisplay)
54272–55295	SID-Chip (Klang)
55296–56319	Farbspeicher
5362Ø–56575	CIA-Chip # 1 (Schnittstellen zu Zusatzgerät)-
56576–56831	CIA-Chip # 2
56832–57343	Reserviert für künftige I/O-Erweiterung
57344–65535	KERNAL*ROM (bei PEEK) *oder* 8K RAM

* KERNAL ist das zentrale Betriebssystem, so angelegt, daß geschriebene Programme, die es verwenden, mit künftigen Modellen des Computers kompatibel sein werden.

Anhang 6
Einige nützliche Systemvariable

Das ist eine Auswahl von Adressen, die das Betriebssystem verwendet, und die in nützlicher Weise mit PEEK oder POKE angegangen werden können. Eine vollständige Beschreibung findet sich im 'Reference Guide'.

Name	Adesse	Funktion
TXTTAB	43–44	Zeiger zu Start von BASIC-Textbereich
VARTAB	45–46	Zeiger zu Beginn von BASIC-Variablen
ARYTAB	47–48	Zeiger zu Beginn von BASIC-Arrays
STREND	49–5Ø	Zeiger zu Ende von BASIC-Arrays + 1
FRETOP	51–52	Zeiger zu Unterseite Stringspeicherung
CURLIN	57–58	Laufende BASIC-Zeilennummer
OLDLIN	59–6Ø	Zuletzt ausgeführte BASIC-Zeilennummer
DATLIN	63–64	Laufende DATA-Zeilennummer
FBUFPT	113–114	Zeiger zu Kassettenpuffer
RNDX	139–143	Keimwert für RND-Funktion
LSTX	197	Code der zuletzt gedrückten Taste*
USER	243–244	Zeiger zu Farb-RAM
COLOR	646	Laufende INK-Farbe
XMAX	649	Größe des Keyboard-Puffers
	53265 (Bit 5)	Bit-Modus einschalten (Kapitel 32)
	5328Ø	Border-Farbe
	53281	PAPER-Farbe

* Codes siehe Anhang 7

Hinweis: Die meisten Adressen umfassen zwei Bytes. Wenn sie sich in den Adressen X und X+1 befinden, ist der gespeichterte Wert:

```
PEEK(X)+256*PEEK(X+1)
```

(niedriges Byte als erstes, hohes Byte als zweites). Beispiel: Die Adresse für den Beginn des BASIC-Textbereichs ist:

```
PEEK(43)+256*PEEK(44)
```

Anhang 7
Keyboard-Abfragecodes

Aufgeführt ist der Inhalt von Adresse 197, wenn eine beliebige Taste gedrückt wird. Mit PEEK(197) kann die gerade niedergedrückte Taste unter Umgehung des Keyboardpuffers festgestellt werden.

Taste	Code	Taste	Code	Taste	Code
(keine)	64	@	46	T	22
*	49	A	1Ø	U	3Ø
+	4Ø	B	28	V	31
,	47	C	2Ø	W	9
–	43	D	18	X	23
.	44	E	14	Y	25
/	55	F	21	Z	12
Ø	35	G	26	RETURN	1
1	56	H	29	CLR/HOME	51
2	59	I	33	INST/DEL	Ø
3	8	J	34	CRSR ↑ ↓	7
4	11	K	37	CRSR → ←	2
5	16	L	42	←	57
6	19	M	36	f1	4
7	24	N	39	f3	5
8	27	O	38	f5	6
9	32	P	41	f7	3
:	45	Q	62	£	48
;	5Ø	R	17	Leertaste	6Ø
=	53	S	13		

Programmregister

Register allgemein

Ob Sie nun Neuling sind in der Computerei und sich den C64 als ersten Computer erstanden haben, oder von einer kleineren Maschine zu ihm aufsteigen: Dieses Buch hier ist die grundlegende Einführung in BASIC auf dem Commodore 64.

Sie finden

- Strings
- Arrays
- Klang und Musik
- Sprites
- Hochauflösende Grafik

Viele Programme sind gebrauchsfertig aufgelistet, etwa

- Planetensuche
- Schüttelreime
- Grand Prix
- Morsecode

Auf der ernsthafteren Seite führt Sie eine einmalige Kapitelserie in Fehlersuchtechniken ein, aber auch der Umgang mit Dateien und die Gestaltung von Programmen werden gründlich behandelt.
Der umfangreiche Anhangteil enthält Tabellen für Binär/Dezimal-Umwandlungen, Spriteregister, Klangchipregister und einen Plan der Speicheraufteilung. Aufgaben mit Antworten am Ende jeden Kapitels werden Ihnen helfen, am Ball zu bleiben.

ISBN 3-7643-1588-1