Computer Shop

Ian Stewart
Robin Jones

Sinclair ZX81

Programme, Spiele, Graphik

Birkhäuser

Computer Shop
Band 1

Ian Stewart/Robin Jones

Sinclair ZX81

Programme, Spiele, Grafik

Aus dem Englischen von Tony Westermayr

Birkhäuser Verlag
Basel · Boston · Stuttgart

Die Originalausgabe erschien 1982 unter dem Titel:
"Peek, Poke, Byte & Ram! Basic Programming for the ZX81"
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:
Sinclair ZX 81 : Programme, Spiele, Grafik /
Ian Stewart ; Robin Jones. Aus d. Engl. von
Tony Westermayr. – Basel ; Boston ; Stuttgart :
Birkhäuser, 1983.
(Computer-Shop ; Bd. 1)
Einheitssacht.: Peek, poke, byte and ram BASIC
programming for the ZX81 ⟨dt.⟩
ISBN 3-7643-1398-6
NE: Jones, Robin:; GT

Umschlaggestaltung: Konrad Bruckmann
Printed in Germany
ISBN 3-7643-1398-6

Inhalt

CHIPS MIT GRIPS

Eine kleine Zahl von Erfindungen hat einen Großteil der Menschheitsgeschichte bestimmt. Als Höh, der Höhlenmensch, das Rad, und seine Frau Böh das Feuer erfand, ahnten sie nur dumpf, daß sie dem *Homo sapiens* auf den Weg geholfen hatten. Seitdem ging es immer so weiter: Pfeil und Bogen, Pflug, das Pferd als Transportmittel, Wasserrad, Dampfmaschine – die beiden letzten Erfindungen bewirkten die erste industrielle Revolution – Automobil, Flugzeug, Radio . . .

Nun ist es uns gelungen, den Mikrochip zu erfinden (und, wie manche sagen, uns um Arbeitsplätze zu bringen). Wir lebten, so heißt es, im Computerzeitalter.

Wieviele Computer gab es Anfang der fünfziger Jahre? Man konnte sie an den Fingern einer Hand abzählen. Buchstäblich: Es waren genau vier Stück.

Bald soll es vier Millionen geben – wenn sie nicht längst schon da sind: das hängt nämlich davon ab, wie man "Computer" definiert.

Tom Watson von IBM soll, als er zum erstenmal von elektronischen Computern hörte, geschätzt haben, die Bedürfnisse der Vereinigten Staaten könnten von ganzen drei Exemplaren befriedigt werden. Ganz so ist es nicht gekommen: Niemand konnte im Einzelnen die ungeheure Explosion der Informationstechnologie voraussehen, die dadurch entstand. Das Computerzeitalter ist jetzt dreißig Jahre alt, und wir werden für den Rest unseres Daseins damit leben müssen, ob uns das gefällt oder nicht.

Es gibt nicht nur mehr Computer, sie sind auch kleiner und schneller und zuverlässiger geworden. 1963 war EDSAC von der Universität Cambridge, auf den man zu Recht stolz sein durfte, ein Kasten voll elektronischer Schaltkreise (in der Hauptsache Radioröhren) von der Größenordnung eines kleinen Lieferwagens, nur viel unzuverlässiger – mehrmals am Tag fiel er aus. Bis 1966 war er völlig veraltet: Röhren waren "out" und Transistoren "in". Dann traten an die Stelle der Transistoren integrierte Schaltungen und der allgegenwärtige Mikrochip. Heute hat der ZX81 von Sinclair die Größe einer Zigarrenkiste, kostet unter DM 200,– und leistet alles, was EDSAC konnte, und sogar mehr.

Der Computer ist auch nicht mehr allein auf die Bereiche von Hochschule und Industrie beschränkt. Er ist ins Heim vorgestoßen. Es gibt heute auf dem Markt vermutlich mehr Typen von Heimcomputern als Biersorten. Sie kosten zwischen 200,– und 10 000,– Mark, je nachdem, was man an Leistung verlangt und wieviel man für hübsches Design ausgeben möchte. Eine Randbemerkung: Viele teure "Geschäfts-Mikrocomputer" sind elegant verkleidete Heimcomputer zum dreifachen Preis.

Dieses Buch handelt von einem solchen Computer, vom schon genannten ZX81, einer Schöpfung von "Onkel Clive" Sinclair.

Warum befassen wir uns gerade mit dem ZX81? Ganz ehrlich gesagt: Er ist nicht gerade das raffinierteste Gerät auf dem Markt. Manche seiner Eigenschaften – beispielsweise das Sensor-Keyboard – erfordern schon ein Quentchen Geduld. Aber was kann man für einen solchen Preis verlangen? R2D2, den kleinen rundlichen Roboter aus dem Film "Krieg der Sterne"? Der ZX81 hat für sein Geld ganz erstaunlich viel zu bieten. Der Umgang mit einem Heimcomputer wird dadurch für jeden erschwinglich, der den Preis für einen Rasenmäher aufbringen will. Der Personal-Computer ist nicht mehr nur beschränkt auf einen kleinen Kreis von Enthusiasten, sondern in die breite Öffentlichkeit gedrungen.

Der ZX81 hat zwar – vor allem ohne seinen Zusatzspeicher – seine Leistungsgrenzen, verfügt aber über die meisten Eigenschaften, die einen Computer zu dem machen, was er ist.

Der ZX81 ist das ideale Gerät für den *Anfänger*. Will ich wirklich einen Personal-Computer haben? Werde ich mich damit vielleicht langweilen? Werde ich mit ihm auch etwas anfangen können, wenn ich ihn bekomme? Mit dem ZX81 kann man gewissermaßen die Zehen in das rauschende Mikro-Wildwasser tauchen, ohne ein großes finanzielles Risiko einzugehen.

Enorm viele ZX81-Geräte dürften allein schon deswegen gekauft werden, damit die *Kinder* den Computer ausprobieren können. Und Papa wird natürlich ein strenges Vaterauge auf sie haben, während er, ähm, nur einmal rasch untersucht, wie man mit dem Ding eigentlich umgeht, nur für den Fall, daß sie einmal Hilfe brauchen sollten . . . Ich wette zehn zu eins, daß die Kinder nach sechs Monaten vom Computer mehr verstehen als er.

Das ist völlig in Ordnung – um mit dem Computer leben zu können, muß man ihn erst einmal lieben lernen, und das heißt eben, ihn benützen, ihm die Kunststücke abverlangen, die man unbedingt sehen möchte . . . aber – nun, die Sache hat einen kleinen Haken. Zeit und Mühe spielen dabei keine so große Rolle; die wendet man schon auf. Papa muß eben nicht so oft in die Kneipe oder auf den Golfplatz gehen. Daß der häusliche Fernsehapparat für den Computer als Monitor gebraucht wird, erfordert einen genauen Zeitplan, das ist alles.

Nein, das Problem liegt einfach darin, daß das Programmieren von Computern am Anfang außerordentlich problematisch sein kann. Man erkennt jetzt erst, wie *dumm* diese alberne Maschine in Wahrheit ist. Dadurch wird das wundersame Mikrogerät auf die Zigarettenkisten-Hirngröße zurechtgestutzt. Ein Beispiel: Man befiehlt dem Ding, eine Grafik zu zeichnen, und aus Versehen geraten einige der grafisch darzustellenden Punkte außerhalb des Bildschirms. Jedes halbwegs intelligente Wesen würde dergleichen nun ohne weiteres bewältigen – der gute Mikrocomputer bekommt einen Anfall, und das Programm bricht zusammen. (Das ist Fachsprache und heißt: Es kommt zum Stillstand, und man muß wieder von vorne anfangen. Wie Sie sehen, haben Sie schon etwas gelernt, und dabei ist das erst die Einleitung.) Oder man hat mühselig ein langes Programm eingegeben und auf "Run" gedrückt, und nun bewirkt es nicht, was man gewollt hat; also geht es darum, herauszufinden, *warum* nicht. Dabei liegt selten ein Computerfehler vor; es ist ein "Fehler" (die Computerleute sagen "bug" dazu – damit ist ein lästiges Insekt gemeint) im Programm, ein Irrtum, der *Ihnen* unterlaufen ist. Wenn man hier nicht systematisch vorgeht, kann einen die Suche nach dem Fehler zur Verzweiflung bringen.

Manche Leute versuchen allen derartigen Problemen dadurch aus dem Weg zu gehen, daß sie vorgekaute Programme auf Kassetten kaufen. Aber wenn Sie es so machen, lernen Sie, was Computer angeht, nie etwas. Sie *müssen* sich die Hände schmutzig machen. Jeder Dummkopf kann ein fremdes Magnetband in einen Computer überspielen und auf Knöpfe drücken; das Interessante bei der Sache – nämlich das, womit Sie sich eine Geschicklichkeit erwerben, die in der Zukunft sehr wertvoll sein könnte – ist eben, seine eigenen Programme aufzubauen. Übrigens würden Sie staunen, wenn Sie wüßten, wie viele fertig erarbeitete Programme Fehler haben, die Sie ohnehin beheben müssen.

Dieses Buch hat zum Ziel, dem Anfänger behilflich zu sein. Über Mikrocomputer gibt es viele Bücher. Über den ZX81 gibt es eine kleinere Anzahl. Nur ganz wenige sind für den echten Anfänger gedacht, – also für jemand, der nicht nur nicht weiß, was ein Prioritätscode ist oder eine Einrichtung für Datenfolge-Abwicklung, sondern der noch nicht einmal genau versteht, wie man die Tasta-

tur bedient. Sogar das Handbuch von Sinclair, Beigabe zum Gerät, hat einem dazu soviel zu sagen, daß es nur wenige Zugeständnisse machen kann.

Aus diesem Grunde haben wir eine ganz behutsame Einführung zu *einigen* der wichtigsten Eigenschaften des ZX81 zusammengestellt. Wenn Sie dieses Buch gelesen haben, kann Sie das Handbuch von Sinclair nicht mehr schrecken. Unter anderem gibt es Abschnitte über Anschließen der Maschine, Sicherstellen von Programmen auf Magnetband, BASIC-Programmierung, Fehlersuche, Zahlenknacken und Grafiken. Wir haben versucht, dabei so viele nützliche und leicht verständliche Tips wie nur möglich zu geben und das Ganze ein bißchen unterhaltsam zu gestalten. Alles ist abgestellt auf den ZX81 mit 1 K, also das Grundgerät ohne den Zusatzspeicher.

Außerdem haben wir versucht, alles in leichtverdauliche Bissen aufzuteilen, damit Sie Gelegenheit haben, jeweils ein, zwei Stunden an der Maschine zu verbringen und mit dem Gefühl aufhören zu können, daß Sie konkret etwas erreicht haben.

Viele Computerbücher erwecken den Eindruck, beweisen zu wollen, wie klug ihre Verfasser sind. Das ist nicht unsere Absicht; hier wird alles klar und einfach dargestellt. Wir sind der Meinung, daß eine *Einführung* in die Welt des Computers genau so und nicht anders aussehen sollte. Beim Fliegen ist auch das Schwerste, vom Boden hochzukommen.

Bis jetzt sind wir kollektiv als "wir" aufgetreten, aber im späteren Verlauf ist das nicht mehr so vorteilhaft. Persönliche Erfahrungen im Plural wiederzugeben, erscheint oft merkwürdig. Da jeder Einzelabschnitt jeweils nur einen Verfasser hat, entschieden wir uns für das Wörtchen "ich", womit wir beide gemeint sind. Sollte Sie das stören, so bedenken Sie bitte, wie oft einzelne Verfasser von sich als "wir" reden.

ACHTUNG – FERTIG – LOS!

Wenn Sie einen ZX81 richtig aufgebaut haben und damit arbeiten können, brauchen Sie das Folgende nicht zu lesen. Im anderen Fall erspart Ihnen das möglicherweise einigen Ärger.

Er lag mattschimmernd und unergründlich in seiner Schaumstoffwiege, geschützt sogar gegen die Paketwerfer im Postamt. Mein ZX81! Computermacht in Megabytes, verfügbar durch Berührung eines Ellenbogens oder was auch immer.

Nachdem ich das Gerät ausgepackt hatte, bestand der nächste Schritt darin, ihn an den Fernsehapparat anzuschließen. Wenn man das nicht tut, läuft der ZX81 zwar auch, aber Sie erfahren nicht, was er treibt – das ist besonders unangenehm, wenn jemand, wie ich, zum Mikrowahn neigt, also zu dem eingewurzelten Verdacht, Mikrocomputer seien bestrebt, die Herrschaft über die Welt an sich zu reißen. Nun weiß ich zwar, daß das Handbuch von Sinclair genau erklärt, was man tun muß, und das in leicht lesbarem Druck von kristallener Klarheit, aber . . . konkrete gewonnene Erfahrung ist stets wertvoll, daher dieses Angebot fachmännischen Rates.

Sitzen Sie bequem? Gut. Das ist nicht bloß leeres Gerede, wie Sie rasch erkennen werden, wenn Sie einmal zwei Stunden vor dem Fernseher am Boden gekniet haben.

Also: Greifen Sie zum Adapter – das ist das schwarze Kästchen, das aussieht wie ein allzu wohlgenährter Stromstecker. An ihm befindet sich an einem langen Kabel ein kleiner, schwarzer Bananenstecker. Entreißen Sie ihn dem Kater, der damit gespielt hat, während Sie dieses literarische Meisterwerk studiert haben, und schieben Sie den Bananenstecker in die Buchse des ZX81, an der "9V DC" steht. Damit sind 9 Volt Gleichstrom gemeint. Mit Düsenflugzeugen irgendwelcher Art hat das nichts zu tun.

Schieben Sie den Stromstecker des Adapters in die Steckdose, nah genug am Fernseher, daß das Kabel auch ausreicht.

Als nächstes nehmen Sie das Verbindungskabel zur Hand. Sollte es nicht zur Stelle sein, dann lassen Sie den Kater durchleuchten, bevor Sie sich bei Sinclair beschweren. Es handelt sich um ein kurzes Einzelkabel mit Steckern an beiden Enden, die zwar verschieden sind, beide aber eine Art Innendorn, umgeben von einem Metallrand, besitzen, ähnlich wie bei einem Teigausstecher. Ein Ende wird in den ZX81 gesteckt, und zwar dort, wo "TV" steht, das andere in den Fernsehapparat, wo sonst der Antennenanschluß hineinkommt. Die Stecker sind leicht zu unterscheiden, weil jeder nur in die für ihn bestimmte Fassung hineinpaßt. Elektrotechniker sprechen recht unvorsichtig von "männlichen" und "weiblichen" Steckern, aber in Wahrheit sind sie alle Hermaphroditen, also zweigeschlechtlich. Man muß eben experimentieren.

Ihr Fernseher muß UHV-Signale empfangen können. Das heißt "Ultra High Frequency" (ultrahohe Frequenz), was nach "Very High Frequency" kam (sehr hohe Frequenz), früher bekannt als VHF – ein interessanter Umstieg aufs Lateinische, wohl deshalb, weil das eindrucksvoller klingt. Wenn Sie das Glück haben, sogar zwei Antennenbuchsen zu besitzen, verwenden Sie die mit der Bezeichnung "UHF". Versteht sich.

Vergewissern Sie sich, daß der Fernseher angeschlossen ist und ziehen Sie den Adapter *nicht* heraus – Sie brauchen also zwei verschiedene Steckdosen oder einen Zwischenstecker.

Der nächste Schritt ist ganz wichtig. *Drehen Sie am Fernseher den Ton weg*, den Lautstärkeknopf also ganz bis zum Anschlag zurück. Wenn Sie das nicht tun, bekommt Oma einen hysterischen Anfall, der Kater verschwindet im Kamin und macht alles voll Ruß, und . . . Lassen Sie sich warnen: Die Welt ist ein grausamer Ort für den einsamen Programmierer, der einen hungrigen Mikro zu füttern hat.

Stellen Sie jetzt den Fernseher an.

Wenn Sie verschwommene, laufende Bilder sehen, haben Sie gestern abend das ARD- oder ZDF-Programm eingeschaltet lassen. Dem ZX81 mag es Spaß machen, das Programm zu sehen oder auch nicht – das ist eine zutiefst philosophische Frage, die noch kein ZX81 zur menschlichen Zufriedenheit hat beantworten wollen – aber mit Ihnen spricht er nur, wenn Sie den richtigen Kanal wählen.

Wenn Ihr Gerät einen drehbaren, numerierten Kanalwähler mit Nummern hat, dann schalten Sie auf den letzten Kanal und lassen Sie vier Kanäle dazwischen aus. Auf keinen Fall Kanal 3 wählen, der ist abscheulich. (Sie hörten soeben ein Witzchen).

Wenn das nicht der Fall ist, werden Sie eine Reihe von Drucktasten finden, von 1 bis 6 (oder 8 oder 1Ø) numeriert.

Ihre nächste Aufgabe besteht darin, die Knöpfe für die Feineinstellung zu finden. Haben Sie nicht? Glauben Sie nur das nicht; da haben Sie nur nicht gründlich genug gesucht. Die geistige Anstrengung liegt irgendwo zwischen Mastermind und dem Zauberwürfel von Rubik, weil die Hersteller sie nämlich *verstecken*, damit die Bedienungstafel nicht so kompliziert aussieht; Leute mit Hasenherz könnten da allzusehr erschrecken. Ich will das erklären.

Bei den meisten Geräten wählt man mit Knopf 1 das Erste Programm, also die ARD, mit Knopf 2 das Zweite Fernsehen ZDF, mit dem dritten das jeweilige Regionalfernsehen. Das liegt daran, daß die Geräte so gebaut sind, richtig? Falsch. Sie könnten Ihr Fernsehgerät jederzeit so einstellen, daß bei jeder Drucktaste das ZDF erscheinen würde. (Ich kann mir allerdings nicht vorstellen, daß irgend jemand das wünschenswert fände.) Nein, der Techniker, von dem Ihr Gerät angeschlossen wurde, hat diese Kanäle so eingestellt. *Die Ziffern an*

Abbildung 1
Nein, Sie haben immer noch Herrn Köpcke dran

den Tasten haben mit den jeweiligen Sendern nichts zu tun. Sie sind willkürlich gewählte Erinnerungsstützen. Also.

Sie brauchen dann nur noch die Knöpfe zu suchen, an denen er gedreht hat, und ebenfalls daran zu drehen. Bei meinem Fernseher drückt man mit einem Kraftaufwand mittlerer Art auf den Herstellernamen, dann öffnet sich eine Klappe und gibt den Blick auf das frei, was wir suchen. Bei Ihnen könnte es sein, daß das Ganze wie eine Schublade vorne oder seitlich herausschnellt. Daß Sie es gefunden haben, wissen Sie, sobald Sie es sehen, weil sich eine ganze Reihe von Rändelschrauben präsentiert, an denen man drehen kann, für jeden Kanal eine. (Wenn Sie wirklich großes Glück haben, erfahren Sie dort auch, welche wohin gehört.)

Suchen Sie sich eine Nummer aus, die Sie noch nicht für einen Sender vergeben haben. Der Übersetzer kann mit seinem Gerät beispielsweise neben ARD, ZDF und Bayern-Regional noch die beiden österreichischen Fernsehprogramme empfangen, aber welche Taste man nimmt, spielt keine Rolle, weil sie im Grunde alle gleich sind. Wenn Sie sich eine Taste ausgesucht haben, drücken Sie sie hinein. Übrigens können Sie auch Ihren freundlichen Fernsehtechniker darum bitten, daß er Ihnen einen Kanal auf den Computerbetrieb einstellt. (Anmerkung des Übersetzers)

Drehen Sie an der Feinabstimmung für den gewählten Kanal und beobachten Sie, wie der Bildschirm sich verändert. Sie sehen vermutlich einen dunklen Hintergrund mit weißen, tanzenden Punkten wie ein tollgewordenes Ameisennest oder ein Luftbild von Tokio bei Nacht. Wenn Sie so weit sind, können Sie die Lautstärke *ganz* vorsichtig aufdrehen; Sie werden hören, warum ich Ihnen geraten habe, den Ton ganz wegzunehmen. Das kracht ganz erbärmlich, was aber am Fernsehgerät selbst liegt – sie tun das alle – und nicht die Schuld von Sinclair ist.

Drehen Sie am Abstimmknopf. Wenn Sie bis zum Anschlag gekommen sind und sich nichts getan hat, drehen Sie ihn in die *andere* Richtung. Sobald Sie an die richtige Stelle kommen, sehen Sie unregelmäßige, flackernde Flecken entlang der linken Bildschirmseite. Drehen Sie noch ein bißchen weiter; jetzt sollten Sie in der unteren linken Ecke ein ganz kleines Viereck erkennen (Abb. 2).

In Wahrheit ist das ein weißes K in einem schwarzen Quadrat – man nennt das Video-Inversion, zu deutsch Negativschrift – aber wenn Sie nicht zufällig

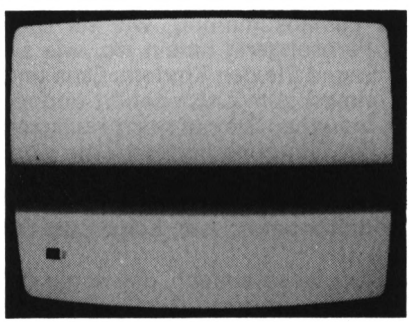


Abbildung 2
Fast geschafft . . .

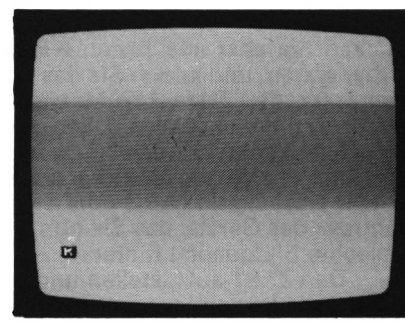


Abbildung 3
Ideal!

Glück gehabt haben, werden Sie es nicht sehr deutlich sehen. Regulieren Sie Helligkeit und Kontrast, bis es so deutlich lesbar ist, wie das nur geht – ein grauer Hintergrund, nicht zu hell, mit einem klaren, kleinen Quadrat, wo das "K" deutlich hervortritt (Abb. 3). Wenn Sie statt dem "K" irgend etwas anderes sehen, haben Sie versehentlich die Tastatur des ZX81 berührt. Drücken Sie gleichzeitig mit einem Finger "SHIFT" und mit dem anderen "RUBOUT", bis das Zeichen verschwindet und das "K" wieder auftaucht. Wenn es nicht anders geht, drehen Sie am Farbabstimmknopf, bis der Bildschirm schwarz-weiß wird.

Genügt Ihnen die Schärfe immer noch nicht, dann versuchen Sie, die einzelnen Stecker in den Fassungen zu drehen, damit ein besserer Kontakt hergestellt wird. Falls nichts von dem eintritt, wovon ich gesprochen habe, können Sie sich das Fernsehanschlußkabel vornehmen und darauf untersuchen, ob das Kabel in Ordnung ist oder ein Kurzschluß besteht, und zwar mit einer Batterie nebst Taschenlampen-Glühbirne; oder Sie können den Ausgang des Gleichstrom-Adapters prüfen (nehmen Sie aber eine 9 Volt-Birne, sonst brennt sie durch.) Wenn es immer noch nicht klappt, schicken Sie das Gerät zurück und versuchen Sie es mit einem neuen Computer.

Wenn alles stimmt, drücken Sie am ZX81 ein paar Tasten. Im Gegensatz zu Schreibmaschinentasten reagieren diese auf Fingerdruck, und man spürt kaum, daß da etwas "nachgibt", wenn man drückt; aber auf dem Bildschirm müßten Sie Wörter und Buchstaben entstehen sehen. Jetzt können Sie damit beginnen, das Handbuch zu lesen und mit dem Computer zu arbeiten. Wenn Sie das Zeug, das Sie eingegeben haben, nicht löschen können, ziehen Sie einfach den Stecker (9V DC) heraus. Oder löschen Sie alles, indem Sie mehrmals auf "RUBOUT" drücken, oder – noch besser – EDIT (1 mit SHIFT) und dann NEWLINE.

Experimentieren Sie ein bißchen und versuchen Sie sich an die Tastatur zu gewöhnen. Nichts, was Sie eingeben, kann Schaden anrichten. Wenn Sie hängenbleiben, ziehen Sie einfach das Stromkabel aus der Steckdose heraus und schließen es wieder an.

Als ich das Gerät zum erstenmal aufstellte, achtete ich besonders darauf, alles anzuschließen, bevor ich einschaltete, aber darauf kommt es in Wirklichkeit gar nicht an, solange alles an der richtigen Stelle angeschlossen wird. Mit 9 Volt kann man sich kaum durch einen elektrischen Stromstoß zu Tode bringen, das heißt, es geht gar nicht. Sinclair betont, daß der ZX81 nicht beschädigt wird, wenn man die Stecker in die falschen Buchsen schiebt, aber beim Anschluß an das Stromnetz müssen Sie selbstverständlich vorsichtig sein.

Wenn Sie abschalten, lassen Sie die Feinabstimmung, wie sie ist – drücken Sie aber alle Tasten wieder in das Fernsehgerät hinein, so, wie sie vorher waren, und lassen Sie das künftig so. Sagen Sie den Kindern, Oma und dem Kater, daß Taste 6 (oder welche auch immer) zum ZX81 gehört und sie nicht daran herumspielen sollen. Von nun an brauchen Sie nur noch Helligkeit und Kontrast neu einzustellen. Wenn Sie mit der Computertätigkeit fertig sind, empfiehlt es sich aus Gründen des häuslichen Friedens, alles wieder auf einen Sender einzuregeln (ARD oder ZF oder was Sie wollen). Sonst werden andere Benutzer des Geräts, das Sie nun als Ihren Monitor betrachten, Klage über die schlechte Bildqualität führen.

Den ZX81 aufzustellen und anzuschließen, ist so einfach, daß sogar der Kater dazu imstande wäre; es gibt aber noch ein anderes Gerät, das ganz entschieden mehr Schwierigkeiten macht, nämlich ein Kassettenrecorder, um Programme auf Tonband mit SAVE sicherzustellen und mit LOAD wieder in den Computer einzugeben. Sie werden diese Möglichkeit wohl gleich ausprobieren

wollen – vor allem dann, wenn Sie, wie ich, den ZX81 auf 14 Tage zur Ansicht gekauft haben, vom Urlaub nach Hause kommen und entdecken, daß er vor 12 Tagen geliefert worden ist. Schlagen Sie in diesem Fall Seite 47 auf, wo nützlicher Rat zu diesem Problem auf Sie wartet. Die Kapitel bis dorthin können Sie ruhig überblättern – zu diesem Zweck werden sie nicht gebraucht. Ich hätte dieses Kapitel gleich hier anschließen können – aber zwei Abschnitte hintereinander nur über Geräte bieten zu wenig Abwechslung, und außerdem ist die nächste wirklich wichtige Aufgabe diejenige, mit dem *Schreiben* von Programmen zu beginnen – unser Hauptziel.

FERTIGPROGRAMME

Selbst wenn Sie in Ihrem ganzen Leben vorher noch nie einen Computer gesehen haben, mit dem Programmieren können sie sofort anfangen. Erst programmieren, dann fragen!

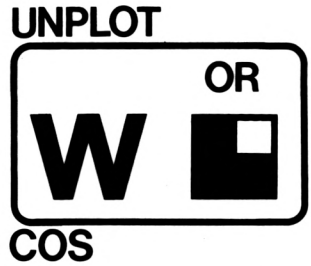
Am Ende dieses Buches werden Sie eine Reihe von Programmlistings finden, die ab Seite 113 beginnen. *Sie können jederzeit eines dieser Programme eingeben und laufen lassen*, ob Sie die im Programm enthaltenen Befehle nun verstehen oder nicht. Sie brauchen nur die einzelnen Schritte des vorgegebenen Programms über die Tastatur einzugeben, sich zu vergewissern, daß kein Fehler unterlaufen ist, und RUN sowie NEWLINE zu drücken. Wir hoffen, daß Sie bis zum Ende des Buches dahintergekommen sind, wie diese Programme ihre Arbeit leisten, aber damit man bald selbständig wird, ist es nur gut, anderer Leute Programme laufen zu lassen. Dabei gewinnen Sie auch bereits eine Vorstellung davon, was man mit dem ZX81 alles anfangen kann.

Die Programme sind in der Regel ergänzt durch Programmhinweise, die verschiedene Eigenheiten erläutern und vereinzelt kleine Veränderungen des Programms oder auch Versuche für Sie auf ähnlichen Gebieten empfehlen. Vorausgesetzt, daß Fehler vermieden werden, laufen sie aber auf jeden Fall, ob Sie die Erläuterungen verstanden haben oder nicht. Die meisten der dort aufgeführten Programme sind Beispiele für bestimmte Techniken, die in diesem Buch dargestellt werden, manche verwenden aber auch Ideen, für deren angemessene Darstellung mir hier der Platz fehlt. Auf jeden Fall gehe ich davon aus, daß Sie nicht nur dieses Buch, sondern auch das Handbuch von Sinclair lesen, das jedem Gerät beiliegt.

Das ZX81 - Keyboard hat ein paar – nützliche – Eigenheiten. Ich möchte sie kurz streifen, weil es manchmal recht ärgerlich sein kann, wenn man das Gerät dazu bringen will, genau das auf dem Bildschirm zu zeigen, was man eingegeben zu haben glaubt.

Schreibmaschinentastaturen verfügen in der Regel über eine Umschalttaste. Wird sie gedrückt, werden Großbuchstaben geschrieben, im anderen Fall Kleinbuchstaben. Ich habe eine alte Reiseschreibmaschine vom Baujahr 191Ø, die *drei* Umschaltmöglichkeiten besitzt. Dadurch braucht man weniger Tasten und spart Platz. Der ZX81 hat praktisch *sechs* Umschaltmöglichkeiten, jedenfalls bei den meisten Tasten. Mit anderen Worten: Jede einzelne Taste kann bis zu sechs verschiedene Zeichen anbieten. Die meisten Schwierigkeiten, das zu erreichen, was man möchte, ergeben sich dann, wenn man zwar die richtige Taste drückt, aber die falsche Umschalttaste.

Damit das verständlicher wird, wollen wir uns die Taste mit dem großen W ansehen. Über dem Rand dieser Taste sieht man UNPLOT, unter ihr COS, auf der Taste selbst OR und das Grafikzeichen ▟. Die Auswahl der sechs verschiedenen Möglichkeiten geschieht nun folgendermaßen (ja, *sechs*, siehe unten!)



1 Wenn Sie die Taste drücken und das Gerät sich am Anfang einer Programmzeile befindet, oder wenn vor dieser nur eine Zeilennummer steht oder das vorangehende Zeichen THEN lautet, kommt auf den Bildschirm UNPLOT. Der ZX81 weiß, daß er ein *Schlüsselwort* erwartet – einen Befehl wie GOTO, RUN, RETURN oder was auch immer – so daß es für ihn keinen Sinn ergibt, W zu "bringen". Schlau, wie er ist, weiß er deshalb, daß Sie UNPLOT haben wollen.

2 Im anderen Fall, vorausgesetzt, Sie haben nichts von dem getan, was unten steht, zeigt er W an.

3 Wenn Sie mit einem Finger auf SHIFT drücken (übrigens die einzige Taste im ganzen Feld, die nur eine einzige Bedeutung hat!), den Finger dort lassen (und ihn niederdrücken, versteht sich) und gleichzeitig W drücken, dann erhalten Sie OR.

4 Stellen Sie die Tastatur auf "Funktionsmodus" um – das heißt, drücken Sie gleichzeitig SHIFT *und* FUNCTION/NEWLINE, was einen [F]-Cursor bringt – und ziehen Sie die Finger zurück, bevor Sie W drücken (ohne SHIFT), dann erhalten Sie COS.

5 Wenn Sie die Tastatur auf "Grafikmodus" umschalten – indem Sie SHIFT und gleichzeitig GRAPHICS/9 drücken, was zu einem [G]-Cursor führt – beide Finger zurückziehen und dann W drücken, erhalten Sie [W] – ein Negativschrift-W. (Das fehlende Zeichen von den sechs, aus recht naheliegenden Gründen auf der Taste nicht angegeben . . .)

6 Schalten Sie wie bei 5 auf Grafikmodus um, drücken aber gleichzeitig SHIFT und W, so erhalten Sie das Grafikzeichen ▟.

Die anderen Tasten funktionieren genauso wie die Taste W und zeigen die möglichen Zeichen am entsprechenden Ort an. Ziffer 5 oben, Negativschrift, ist auf keiner Taste angezeigt, aber auf allen Tasten außer SHIFT und FUNCTION/NEWLINE verfügbar. Das Negativ von SPACE ist ein schwarzes Quadrat – jawohl, genau *da* ist es!

BASIS-BASIC

ROM, RAM und REM . . . Computersprache ist gar nicht so schwer, sie sieht nur so aus.

Computer haben im Grunde nichts Geheimnisvolles an sich. Sie sind nichts anderes als Maschinen, die Befehlsfolgen ausführen. Die konkrete *Art* freilich, wie sie das tun – sie ist in der Tat geheimnisvoll, jedenfalls dann, wenn man nicht gerade in Festkörperphysik seinen Doktor gemacht hat. Aber genauso, wie man ein Auto lenken kann, ohne es selbst gebaut zu haben, vermag man auch einen Computer zu programmieren, ohne zu wissen, wie man ihn baut. Ein *gewisses* Verständnis für das eigentlich Technische daran – also das, was die Computerleute *Hardware* nennen – ist aber von Nutzen, genauso, wie es nützlich ist, zu wissen, wie eine Getriebeschaltung funktioniert, wenn man die Gänge schalten will oder sich Gedanken darüber macht, warum es überhaupt notwendig ist, zu schalten. Die Grundprinzipien des Programmierens hängen aber, was den Normalbenutzer angeht, von der Hardware kaum ab.

Maschinen, die Befehle ausführen können, gibt es schon lange. Blaise Pascale, der französische Denker und Mathematiker, baute schon 1642 eine Rechenmaschine. Charles Babbage, ein Engländer, konstruierte 1835 eine ehrgeizige "Analytische Maschine", und die britische Regierung wurde dazu bewogen, Geld in die Maschine zu investieren. Wie andere staatliche Projekte erwies sich auch dieses als mit der Technologie der damaligen Zeit nicht durchführbar, aber die Grundgedanken waren originell und hatten Hand und Fuß. Der Franzose Joseph-Marie Jacquard entwickelte Anfang des 19. Jahrhunderts ein System, bei dem Löcher in Karten gestanzt wurden, um einen Webstuhl zu steuern. Dampfpfeifenorgeln auf Jahrmärkten arbeiten nach einem ähnlichen Prinzip. Man kann sich ein eingespieltes Orchester in der Tat als eine Maschine vorstellen, die Musik hervorbringt, wobei die Notenblätter das "Programm" darstellen. Kein schlechter Vergleich.

Es hat keinen Sinn, zu einem Jacquard-Webstuhl zu gehen, einfach zu sagen, "bitte sieben Meter Karo grünblau", und zu erwarten, daß sich viel tut. Diese Sprache versteht die Maschine nicht. Alles, was sie versteht, ist ein Muster von gestanzten Löchern, und für Größe, Zahl und Anordnung dieser Löcher gibt es eine ganze Menge fester Regeln. "Hoch auf dem gelben Wagen" würde bei einem Webstuhl nichts bringen – und selbst wenn das doch der Fall wäre, käme dabei etwas heraus, das mit der Originalmelodie wenig zu tun hätte. Man muß wissen, *welches* Muster von Löchern verlangt wird, um im Stoff das gewünschte Webmuster zu erzeugen. Das Lochmuster ist also eine Art *verschlüsselter* Fassung des Webmusters, wobei der Mechanismus des Webstuhls als Mittler wirkt.

Ebenso sind die Notenblätter eines Orchesters eine verschlüsselte Version der angestrebten Töne, das Orchester selbst ist der Mittler. (Einen kleinen Unterschied gibt es freilich: Orchester arbeiten nicht so präzise wie eine Maschine, und der Dirigent behält eine gewisse Freiheit, die Musik nach seinen Vorstellungen zu interpretieren. Aber es kommt der Sache durchaus nah.)

Bei Computern ist es genauso, nur noch in stärkerem Maß. Mit jeder Maschine muß man in der Sprache reden, die sie versteht.

In den Anfängen der Computerzeit hieß das: Man mußte lange Listen schreiben, die etwa so angingen: Ø1ØØØ11 1Ø1ØØ11Ø1Ø11111111ØØØ1Ø11Ø ØØ1Ø1ØØ11 . . ., was für das Auge einigermaßen beschwerlich ist. "Tief im Innersten" *denken* Computer immer noch so (wobei "Ø" für "kein elektrischer Stromdurchgang" steht und "1" für "ein gewisses Maß an elektrischem Stromdurchgang"): diese eher schlichte Sprache ist der *Maschinencode* für den gewählten Computertyp . . . Wunderbarerweise kann man damit alles machen, was man will, und im allgemeinen hat das den Vorteil, daß es sehr *schnell* geht . . . aber Programme damit zu schreiben, ist doch etwas kompliziert.

Zum Glück hat man das heute nicht mehr nötig.

Der Grund: Einem Computer kann man *beibringen*, Befehle in einer Sprache anzunehmen, die nicht seine "Muttersprache" ist. Das geht so vor sich: Irgendein fleißiger Programmierer schreibt eine Art "Wörterbuch", um die neue Sprache in die Maschinensprache zu übersetzen, und füttert damit den Computer (entweder über eine Außenquelle oder über einen eingebauten Teil des Computers). Das Wörterbuch wird *Compiler* oder *Interpreter* genannt (je nachdem, was es im einzelnen bewirken soll).

Der ZX81 besitzt einen eingebauten Interpreter für eine Sprache, die BASIC heißt, eine Abkürzung für "Beginners All-purpose Symbolic Instruction Code" – was soviel bedeutet wie: einfache höhere Programmiersprache für Teilnehmer-Rechensysteme. Diese Sprache wurde Mitte der sechziger Jahre in den USA entwickelt und ist weithin in Gebrauch. Sie hat den Vorteil, vergleichweise verständlich zu sein, eine Art Kreuzung zwischen normalem Englisch und Schulalgebra. Was den Anfangsprogrammierer angeht, so spricht der ZX81 nur BASIC. (Über die Taste USR ist die Maschinensprache ZX80 aber durchaus zugänglich – und wenn Sie nicht wissen, *warum* Sie sie vielleicht benützen sollten, dann tun Sie es *nicht*. Jedenfalls jetzt noch nicht.)

Ein BASIC-Programm

Statt mit der "Grammatik" von BASIC anzufangen, wollen wir uns einmal ein einfaches BASIC-Programm ansehen, um festzustellen, was es leistet und wie es das tut. Der Vorteil dabei: Sie können sofort erkennen, wie einfach es ist. Die Feinheiten grammatischer Art kommen später:

```
1Ø  PRINT "QUADRATWURZELN"
2Ø  INPUT X
3Ø  LET Y = SQR X
4Ø  PRINT X,Y
5Ø  STOP
```

Sie erraten vermutlich schon auf den ersten Blick, was das bewirkt, (Hinweis: SQR bedeutet "Quadratwurzel von"). Aber zuerst muß einiges zu der Form gesagt werden, in der das auftritt.

a Es besteht aus einer Folge von *Zeilen*.
b Jede Zeile ist "zulässig" (das heißt, logisch vernünftig) als BASIC-*Befehl* oder -*anweisung* oder -*kommando*. (Diese drei Ausdrücke bedeuten praktisch alle dasselbe.)
c Jede Zeile beginnt mit der Nummer, die (wen kann es wundern?), *Zeilennummer* heißt (Computer verwenden oft Ø für Null, damit Verwechslungen mit dem Buchstaben 0 vermieden werden.)

Diesen ja doch recht naheliegenden Bemerkungen muß ich einige Erläuterungen anfügen. Manches davon gilt für *alle* BASIC-Interpreter, wie sie auch bei anderen Geräten verwendet werden, einige nur für den ZX81. Um mir Arbeit zu ersparen, will ich mich nicht lange damit aufhalten, wo nun was wann gilt – wenn Sie zu einem größeren Computer aufsteigen, werden Sie bald dahinterkommen.

Zu a) braucht man weiter nichts zu sagen, außer, daß beim ZX81 eine Programmzeile länger sein darf als eine Einzelzeile, wenn sie auf dem Fernseh-Bildschirm geschrieben wird – der Maschine macht das nichts aus.

Genaueres zu b) hängt mit der "Grammatik" von BASIC zusammen, auf die wir später kommen.

Zu c): Der Grund dafür, daß man Zeilen numeriert, ist der, daß man zu Beginn seiner Laufbahn als Programmierer den Computer wird anweisen wollen, bestimmten Zeilen eines Programms zu gehorchen. Eine Anweisung "GOTO 73Ø" teilt ihm mit, daß er tun soll, was in Zeile 73Ø steht . . . aber er muß wissen, was für eine Zeile das ist. Es hat seine Vorteile, *nicht* einfach "1, 2, 3 . . ." zu zählen, wie wir noch sehen werden. Die Zeilennummer muß eine ganze Zahl zwischen 1 und 9999 sein (beide eingeschlossen.).

Wenn das Gerät sich durch ein Programm arbeitet, geht es von einer Zeile zur nächsten (außer, es erhält als Teil des Programms die Anweisung, anderswo hinzugehen.) Es braucht also die Zeilennummern selbst dann, wenn der Programmierer sich gar nicht auf sie beziehen will. Der ZX81 nimmt Programmzeilen ohne Nummer nicht an (führt aber stattdessen möglicherweise den Befehl für diese Zeile aus, als sei er ein Taschenrechner.) *Vergessen Sie die Zeilennummer nicht!*

WAS MACHT ER (der ZX81 nämlich)?

Tippen Sie das oben erwähnte Programm in Ihren ZX81 ein. Ich will davon ausgehen, daß Sie wissen, wie das geht. Falls nicht, lesen Sie den vorherigen Abschnitt noch einmal durch und nehmen Sie sich dann Seiten 13 und 17 des ZX81 BASIC PROGRAMMING (künftig nur noch "Handbuch" genannt) vor, das Sie zusammen mit dem Gerät bekommen haben.

Gut. Sie haben das Programm also eingetippt, es steht da oben auf dem Bildschirm, aber bis jetzt tut sich noch nichts. Das liegt daran, daß Computer nicht nur Befehlen gehorchen, sondern im Grunde völlig beschränkt sind. Selbst wenn ganz offenkundig ist, was von man ihnen will, muß man es ihnen trotzdem sagen. Drücken Sie also Taste R, was RUN ergibt, und dann NEWLINE, damit der Computer weiß, daß Sie mit dem Befehl fertig sind.

Was Sie auf dem Bildschirm sehen, sieht aus wie RUN L, wobei, wie Sie in diesem Buch immer wieder feststellen werden, Quadrate "Negativ-Videozeichen" bedeuten – ein weißes L in einem schwarzen Quadrat. Sobald Sie NEWLINE drücken, verschwindet das.

Auf dem Bildschirm erscheint die Mitteilung QUADRATWURZELN. Das ist die Antwort des Computers auf Zeile 1Ø: PRINT "QUADRATWURZELN". Das hat er sehr schnell gemacht und wartet nun darauf, daß *Sie Ihren* Teil von Zeile 2Ø leisten: INPUT X.

Zur Erinnerung steht in der linken unteren Bildschirmecke ein L. Es will von Ihnen erfahren, was X ist.

Damit Sie das mitteilen können, geben Sie eine Zahl ein, gefolgt von NEWLINE. Versuchen Sie

```
25  NEWLINE
```

Blitzschnell zeigt der Computer

```
25
```

am oberen Bildschirmrand an. Etwa eine Sekunde danach fügt er hinzu

```
25          5
```

(und, in der unteren Ecke, die Mitteilung 9/5Ø, mit der wir uns noch nicht befassen wollen – sie bedeutet nur, daß das Gerät die Aufgabe richtig erfüllt hat.)

Versuchen Sie es noch einmal. Dazu brauchen Sie nur RUN, gefolgt von NEWLINE zu drücken, und es geht von neuem los. Wenn der Computer X verlangt, probieren Sie etwas anderes aus: Wenn Sie es mit 756.2912 versuchen und NEWLINE drücken, sollten Sie erhalten

```
756.2912          27.5ØØ749
```

Lassen Sie das Programm noch ein paarmal mit RUN laufen und zwar mit verschiedenen Werten für INPUT X. Geben Sie sie ganz beliebig ein. Versuchen Sie es mit Ø, 1, 4, 9, 16.

Das sollte Sie davon überzeugen, daß die Maschine, egal, was Sie als X eingeben, auf dem Bildschirm immer zwei Zahlen anzeigt, zuerst X und dann die Quadratwurzel von X mit einer ganzen Reihe von Dezimalstellen.

Ich gehe hier davon aus, daß Sie mit INPUT einen vernünftigen Wert von X eingeben – eine positive Zahl und für das Gerät nicht zu groß. Versuchen Sie, nur einmal zum Spaß, X mit – 2 einzugeben. In der linken unteren Ecke erhalten Sie eine Mitteilung die A/3Ø lautet. Der Computer protestiert dagegen, die Quadratwurzel aus einer negativen Zahl ziehen zu sollen . . . und weigert sich, es zu tun.

Experimentieren Sie. Sie sollten Ergebnisse erhalten wie

```
49              7
6E+17           774596670
5E+3Ø           2.236Ø68E+15
```

(Die Zahlen mit einem E sind Zahlen in Exponential-Schreibweise: vergleiche dazu Seite 26 des Handbuchs.)

WIE MACHT DAS GERÄT DAS?

Ich meine nicht im einzelnen – da kämen wir auf Hardware, Maschinensprache und ähnlich komplizierte Dinge zu sprechen. Aber: Wie sollten Sie als Benutzer sich die Vorgänge im Computer vorstellen, wenn er Ihr Programm ausführt? Was findet in seinem Siliciumchip-Minigehirn statt? Wenn wir uns das einmal ein bißchen vermenschlicht vorstellen, geht dergleichen ungefähr so vor sich:

```
1Ø PRINT
   "QUADRATWURZELN"
2Ø INPUT X
3Ø LET Y = SQR X
4Ø PRINT X, Y
5Ø STOP
```

O je, da kommen ein paar Befehle für mich, die ich mir merken muß. Mach dich lieber ran.
Bin bloß neugierig, ob er bald fertig ist.

RUN NEWLINE

Schön. Geht schon los. Wie war die erste Zeile? Aus dem Speicher holen: 1Ø PRINT "QUADRATWURZELN". Ich muß irgend etwas anzeigen. Da steht ein Anführungszeichen "; dann übernehme ich, was folgt . . .

QUADRATWURZELN

bis ich zum nächsten" komme. Das weist mich an, mit dem Anzeigen aufzuhören.
Sonst kommt nichts mehr. Also weiter zur nächsten Zeile: 2Ø INPUT X. Er wird mir eine Zahl angeben, und ich habe sie X zu nennen. Ich gebe ihm einen . . .

L

-Cursor, um ihn zu erinnern.
Na schön, ich warte. Erbärmlich langsam, diese Menschen.

25 NEWLINE

Ah, X ist 25. Also, was nun?
3Ø LET Y = SQR X. Das heißt, ich muß die Quadratwurzel von X berechnen . . . nämlich $\sqrt{25}$, also 5, und das Ergebnis Y nennen. Y ist demnach 5.
Nächster Befehl? 4Ø PRINT X. Ich muß X und Y anzeigen, das sind

25 5

Nächste Zeile? 5Ø STOP. Das wär's also. Schließ ich demnach ab mit der Meldung . . .

9/5Ø

Nehmen Sie die Späßchen in der rechten Spalte nicht so ernst. Sehen Sie, wie das Gerät die Liste der Befehle durchgeht und sie der Reihe nach ausführt? Die Maschine "weiß" nicht einmal, worum es bei dem Programm geht. Aber Sie, der Programmierer, wissen es – es berechnet Quadratwurzeln.

Die Aufgabe des Programmierers ist jetzt klar. Wenn der Computer für Sie etwas leisten soll, müssen Sie eine Folge von Befehlen zusammenstellen, die, sobald ausgeführt, dieses Ziel erreichen.

Dazu müssen Sie BASIC genauer kennen. Ihr ZX81 ist zu allerhand Raffinessen imstande, wenn Sie wissen, wie man mit ihm reden muß. Es gibt natürlich alle möglichen Verfeinerungen – ein *gutes* Programm sollte nicht nur sein Ziel erreichen, sondern schnell sein, wirkungsvoll und klar. Aber mit den Verfeinerungen eilt es nicht: Es kommt darauf an, Programme zu schreiben, die auch funktionieren.

Also, fangen wir an.

ARITHMETIK IN BASIC

Addition, Subtraktion, Multiplikation und Division sind die Grundlagen aller Mathematik. Beim Computer ist es genauso. Am vernünftigsten fängt man also damit an, deutlich zu machen, wie das geht.

Eigentlich ist das gelogen – es ist Algebra. Aber ich wollte Sie nicht gleich abschrecken. Algebraische Berechnungen kommen fast in jedem Programm vor, das man schreiben kann, selbst wenn es nur um Buchhaltung geht. Am besten setzt man also hier ein.

Wie die gewöhnliche Algebra verwendet auch BASIC Buchstaben für "natürliche" Zahlen. In der Fachsprache werden diese *Variable* – genau *Zahlen*-variable – genannt (also eine Größe, deren Wert sich verändern kann). Im Grunde heißt das nur, daß das Bezeichnungen sind wie X, Y, Z, A, B, C und so weiter. Man kann sie dazu benützen, algebraische Ausdrücke wie X + Y − Z aufzustellen, die der Computer dann zu berechnen vermag, wenn Sie ihm durch das Programm sagen, welche Werte für X, Y und Z gelten. Ein Beispiel: Wenn X = 14, Y = 3, Z = 9, dann ist X + Y − Z = 14 + 3 − 9 = 8. Beim ZX81 können Sie für Variable kompliziertere Symbole verwenden als nur einzelne Buchstaben; da diese aber mehr nach normaler Algebra aussehen, dürfte es doch besser sein, am Anfang bei Einzelbuchstaben zu bleiben. Außerdem nehmen lange Namen Speicherplatz weg.

Es gibt einige kleine, aber wichtige Unterschiede zwischen BASIC-Algebra und gewöhnlicher Algebra. Erstens werden alle Buchstaben groß geschrieben – Kleinbuchstaben können Sie gar nicht schreiben. Die Zeichen + (plus), − (minus) und / (geteilt durch) entsprechen den üblichen. (÷ können Sie nicht verwenden). Multiplikation dagegen wird mit einem Sternchen * angezeigt. 5*7 bedeutet also 5×7, was 35 ergibt. Der doppelte Stern ** (H mit SHIFT; Achtung: zweimal das Symbol *, nämlich B mit SHIFT, wird vom Computer *nicht* richtig verstanden!) bedeutet "erhoben zur Potenz". Beispiel: 2**3 stellt dar, was ein Mathematiker mit 2^3 ausdrücken würde, also 2 hoch 3 (oder 2 *kubiert*), mit dem Wert 2×2×2 = 8.

Sie können Zahlen, Variable und diese arithmetischen Zeichen kombinieren, um kompliziertere Ausdrücke zu formulieren. Zum Beispiel ist

```
A*X**2+B*X+C
```

das beim Mathematiker beliebte ax^2+bx+c.

Kapitel 4 des Handbuchs erläutert diese Dinge und weist auf die Wichtigkeit der Reihenfolge hin, in der das Gerät die Berechnungen ausführt. Nehmen wir an, bei dem obigen Ausdruck weiß der Computer, daß A = 4, X = 5, B = 3, C = 7. Man möchte nun meinen, er würde das Ergebnis auf folgende Weise errechnen:

A*X = 4*5 = 2Ø.

A*X**2 = 2Ø**2 = 4ØØ.

A*X**2+B = 4ØØ+B = 4Ø3.

A*X**2+B*X = 4Ø3*X = 2Ø15.

A*X**2+B*X+C = 2Ø15+C = 2Ø22.

Das erhält man, wenn man in der Gleichung von links nach rechts vorgeht. Das entspräche aber nicht der Gleichung ax^2+bx+c des Mathematikers, denn da käme heraus

$$4\times5^2 + 3\times5+7 = 1ØØ+15+7 = 122.$$

Was ist also schiefgegangen?

Der springende Punkt ist der: Die gewöhnliche Algebra steckt voller Regeln darüber, in welcher Reihenfolge die Operationen vorgenommen werden. (Denken Sie mal an Ihre Schulzeit. Na, vergessen wir das . . .) Und die Sprache BASIC hat ähnliche Regeln. In der Praxis befaßt sie sich *zuerst* mit allen **, dann mit allen * und /, und zuletzt mit den + und −. Sobald irgendwelche Zweifel auftauchen, geht sie bei jeder Gleichung der Reihe nach von links nach rechts vor. Sie bewältigt A*X**2+B*X+C folgendermaßen:

Zuerst die **:	X**2 = 5*2 = 25
Dann die *:	A*X**2 = 4*25 = 1ØØ
	B*X = 3*5 = 15
Jetzt die +:	A*X**2+B*X = 1ØØ+15 = 115
	A*X**2+B*X+C = 115+C = 115+7 = 122.

Sie sehen, wieviel Ähnlichkeit das mit gewöhnlicher Algebra hat.

Genau wie bei der Algebra wollen Sie manchmal einen Ausdruck berechnen, der diesen Regeln *nicht* naturgemäß folgt. Die Lösung ist die gleiche: Sie verwenden *Klammern* ( ). Jeder Ausdruck in der Klammer wird als Ganzes vorweg aufgelöst.

So würde

(A*X)**2

aufgelöst werden mit (4*5)**2, also (20)**2, wenn Sie den Ausdruck in der Klammer zuerst berechnen, was 400 ergibt.

Es kommt nur eine einzige Art von Klammer vor (nicht wie [ ] oder { } in der Algebra). Sie müssen also besonders vorsichtig sein, wenn Sie Klammern in Klammern setzen und Ausdrücke schreiben wie (3+5*(A−B)**4, wo der Mathematiker $[3+5(a-b)]^4$ schreiben würde.

Die Hauptsache ist, man begreift, daß das wirklich sehr viel Ähnlichkeit mit gewöhnlicher Algebra hat, nur mit nicht ganz so bekannten Symbolen.

WIE MAN EINER VARIABLEN EINEN WERT ZUTEILT

Das geschieht durch den LET-Befehl (Taste L). Wenn Sie die Taste dort benützen, wo das angebracht ist, erkennt der kluge ZX81 das automatisch als LET und nicht als L (durch einen Vorgang, der "automatische Syntaxprüfung" genannt wird, was aber nur heißt, daß er ein Auge darauf hat, ob das, was Sie eintippen, Sinn ergibt. Eine Programmzeile wie

```
40  LET X = 5
```

teilt dem Computer mit, daß die Variable X *von jetzt an und solange, bis ihm ein anderer Teil des Programms etwas anderes mitteilt*, den Wert 5 besitzt. Versuchen Sie es mit folgendem Programm:

```
10  LET A = 4
20  LET B = 3
30  LET C = 7
40  LET X = 5
50  PRINT A*X**2+B*X+C
```

Wenn in dem winzigen Hirn alles in Ordnung ist, werden Sie oben am Bildschirm die Lösung 122 auftauchen sehen.

Es gibt einfachere Wege, zu dieser Lösung zu gelangen. Versuchen Sie es mit:

```
10  PRINT 4*5**2+3*5+7
```

Sie können sogar die Zeilennummer weglassen; siehe dazu Kapitel 4 des Handbuchs. "Der ZX81 als Rechner".

Aber wir wollen Ihnen zeigen, was LET bewirkt.

Die rechte Seite eines LET-Befehls kann ein Ausdruck sein, von dem der Computer schon weiß, wie er ihn berechnen soll. So könnten Sie beispielsweise Zeile 50 des oben aufgelisteten Programms verändern, um eine neue Variable Y zu definieren, die den Wert haben soll A*X**2+B*X+C:

```
50  LET Y = A*X**2+B*X+C
```

Setzen Sie fort mit

```
60  PRINT Y
```

und Sie können überprüfen, daß er richtig arbeitet.

Beispiel: Fallende Körper

Laut Galilei fällt ein Körper aus dem Ruhezustand unter der Einwirkung der Schwerkraft in einer Zeit von t Sekunden annähernd $5t^2$ Meter. Um festzustellen, wie weit er in 17 Sekunden fällt, könnten Sie folgendes Programm verwenden:

```
10 LET T = 17
20 PRINT 5*T**2
```

Sie sollten als Ergebnis 1445 (also Meter) erhalten.

Wenn Sie die Lösung für einen anderen Zeitraum haben wollen, können Sie die erste Programmzeile verändern. Versuchen Sie es. Wie weit wird der Körper fallen in

a 97 Sekunden?

b 3 Sekunden?

c 24.5779 Sekunden?

d 10000 Sekunden?

EINE VERBESSERUNG: DIE INPUT-ANWEISUNG

Wahrscheinlich werden Sie es schon satt haben, die Zeile 10 andauernd verändern zu müssen. Ein Programm, das Ihnen schon mehr entgegenkommt, würde den INPUT-Befehl nutzen, der den Computer veranlaßt, Sie zu fragen, welcher Wert für eine Variable gilt. Tippen Sie

```
10 INPUT T
20 PRINT 5*T**2
```

und geben Sie RUN. Der [L]-Cursor erscheint, der Computer wartet also auf die Mitteilung, welchen Wert T haben soll. Geben Sie 17 ein und drücken Sie NEWLINE; Sie erhalten die Antwort, die wir oben schon hatten. Um nun die Lösungen für a) bis d) zu finden, brauchen Sie nur noch RUN zu drücken, müssen aber den neuen Wert von T eingeben. Probieren Sie es aus.

PROGRAMMIERAUFGABEN

Hier sind drei Programme für Sie zum Schreiben, und zwar nur mit den Befehlen, die wir uns bisher angesehen haben. Sie sind leichte Abwandlungen des Programms für fallende Körper. Wenn Sie ins Stocken kommen sollten, am Ende des Kapitels finden Sie eine Eselsbrücke.

e Das Volumen eines Würfels, dessen Seite x ist, wird mit x^3 bestimmt (das heißt in BASIC X**3, wenn X die Seitenlänge bedeutet). Schreiben Sie ein Programm, das X als INPUT gibt und mit PRINT das Volumen des entsprechenden Würfels anzeigt.

f Am Ende eines Seiles hängt ein Eimer. Das Seil ist um eine Winde in der Form eines Zylinders vom Radius r gewickelt. Für diesen Eimer wird den Lehrbüchern der Mechanik gemäß die Fallbeschleunigung bestimmt durch

$$a = \frac{32}{1 + 3r^2}$$

Zerbrechen Sie sich nicht den Kopf darüber, sondern schreiben Sie ein Programm, bei dem Sie mit INPUT r eingeben und durch PRINT die Beschleunigung a erhalten können.

g Im Supermarkt kostet Honig pro Glas 61 Pence, Brühwürfel kosten 25 Pence pro Packung, Woskus-Superfleisch kostet 32,5 Pence die Dose. Schreiben Sie ein Programm, das mit PRINT die Gesamtkosten von H (Gläser Honig), B (Packungen Brühwürfel) und W (Dosen Woskus-Superfleisch) nennt, wenn Sie die Werte für H, B und W eingeben. (Sie brauchen drei INPUT-Befehle).

Ein Vorgeschmack von Schleifen

Die Befehle FOR und NEXT erlauben uns, ein paar von den genannten Gedanken auszuprobieren, ohne allzuviel Arbeit leisten zu müssen. (In vielen Kreisen wird die Vermeidung von Arbeit als "Faulheit" bezeichnet, aber Programmierer haben gegenüber Normalsterblichen einen Vorteil – sie können von "effektivem Programmieren" sprechen und zum Beweis auf Ersparnisse an Speicherplatz oder Zeit verweisen). Über FOR-NEXT-Schleifen läßt sich viel sagen, und ein paar Seiten später äußern wir uns auch in gebührender Ausführlichkeit dazu. Fahren Sie zuerst dieses Programm, damit Sie sehen können, was sich damit machen läßt.

```
10  FOR X = 1 TO 20
20  PRINT X, X*X
30  NEXT X
```

Sie werden eine Liste der Zahlen von 1 bis 20 finden und eine zugeordnete Liste ihrer Quadratzahlen 1, 4, 9, 16 . . . 361,400. Die FOR/NEXT-Befehle bewirken nämlich, daß die Maschine immer wieder durch die Folge von Befehlen zwischen ihnen geschickt wird (hier nur Zeile 20), wobei die Variable X der Reihe nach den Wert 1, 2, 3 . . . erhält, bis 20 erreicht ist. Zeile 10 setzt diese Begrenzungen fest und bringt die Schleife zum Laufen, Zeile 30 schickt den Computer wieder durch die Schleife.

Wenn Sie Zeile 20 in diesem kleinen Programm verändern, können Sie alles Mögliche tabellarisch darstellen. Sie wollen Kubikzahlen? Versuchen Sie es mit

```
20  PRINT X,X**3
```

Verändern Sie Zeile 20 jedesmal in der unten angegebenen Weise und achten Sie auf die Unterschiede. Die ganz geringen Abwandlungen von einer Zeile zur nächsten bewirken, daß der Computer die Berechnungen in unterschiedlicher Reihenfolge mit unterschiedlichen Ergebnissen ausführt. Was berechnen die Programme in gewöhnlicher algebraischer Zeichenform?

h 2Ø PRINT X, 1/X+1

i 2Ø PRINT X, 1/(X+1)

j 2Ø PRINT X, 1/1+X

k 2Ø PRINT X, 1/(1+X)

l 2Ø PRINT X, 1+1/X

m 2Ø PRINT X, (1+1)/X

Beachten Sie schließlich, daß in diesen Programmen die Werte der Variablen nicht durch den Befehl LET zugeteilt werden. Der Computer erhält den Auftrag vielmehr durch den FOR-Befehl.

LÖSUNGEN

Fallende Körper

a 47Ø45

b 45

c 3Ø2Ø.3658

d 5Ø1ØØØ5ØØØØØ. Lassen Sie sich von der scheinbaren Genauigkeit nicht täuschen. Der Computer bietet keine Genauigkeit von 12 gültigen Stellen. Er hat die richtige Lösung abgerundet.

Programmieraufgaben

e 1Ø INPUT X
2Ø PRINT X**3

f 1Ø INPUT R
2Ø PRINT 32/(1+3*R**2)

g 1Ø INPUT H
2Ø INPUT Z
3Ø INPUT W
4Ø PRINT 61*H+25*Z+32.5*W

Ihr Programm braucht nicht *genauso* auszusehen wie dieses, um richtig zu sein, Vor allem ist es ganz allein Ihre Sache, welche Symbole für Variable Sie nehmen. Sie können das Mathematische auch auf andere Weise bewältigen; bei e) ist nichts einzuwenden gegen 2Ø PRINT X*X*X, und bei f) nichts gegen 2Ø PRINT 32/(1+3*R*R).

h $\frac{1}{x} + 1$

i $\frac{1}{x+1}$

j 1+x [berechnet zuerst 1/1, also 1, und addiert x]

k wie i)

l wie h)

m $\frac{2}{x}$

HIIIIIIIIIIIEEELFEEEEEEE!

Es läuft nicht alles so gut, wie Sie gehofft hatten? Vielleicht brauchen Sie Hilfe.

Was tut man, wenn etwas nicht so richtig klappt?

Im allerschlimmsten Fall ziehen Sie kurz den Stromstecker heraus. Damit bricht das Programm zusammen, alles, was Sie geschrieben haben, wird gelöscht – aber damit sind auch alle Schwierigkeiten behoben (es sei denn, sie lägen in der Hardware, dann haben Sie Pech gehabt). Aber in der Regel gibt es bessere Wege.

a Wenn Sie in einer Programmzeile etwas Falsches eingetippt haben: Drücken Sie die Tasten 5 und 8 mit den Pfeilen (vergessen Sie nicht, daß Sie gleichzeitig SHIFT drücken müssen, um den [L]-Cursor gleich hinter das falsche Zeichen zu setzen und es mit RUBOUT auszulöschen.

b Wenn Sie eine Zeile verändern wollen, die schon im Programm steht: Falls sie ganz gelöscht werden soll, tippen Sie die betreffende Zeilennummer ein und drücken auf NEWLINE. Sonst führen Sie den [>]-Cursor mit den Tasten 6 und 7 an den Zeilen hinauf und hinunter und drücken danach die Taste EDIT, um die Zeile dort hinunterzustellen, wo der Cursor sie wunschgemäß verändern kann. Dann weiter wie bei a).
Verschiedene Tricks: Es macht ja allerhand Mühe, den Cursor Zeile für Zeile von 1Ø bis 53Ø zu bewegen. Nehmen Sie eine Zeilennummer kurz vor 53Ø, die Sie nicht verwendet haben, sagen wir 529. Schreiben Sie 529 NEWLINE und drücken Sie danach auf EDIT. Sie werden sehen, daß Zeile 53Ø herunterkommt. (Warum tut sie das?) Oder geben Sie stattdessen ein LIST 53Ø und anschließend EDIT. Wenn der Schirm voll ist, müssen Sie CLS drücken, bevor Sie auf EDIT gehen.

c Wenn das Programm steckenbleibt und sich nichts mehr zu rühren scheint: Das kommt oft vor beim Fehlersuchen (auf Englisch heißt das "Debugging", also sind wieder die lästigen Insekten gemeint). Das Letzte, was Sie wollen, ist, alles zu löschen und wieder neu einzugeben; außerdem tut es doch aller Wahrscheinlichkeit nach wieder genau dasselbe.

c1 Wenn das Programm noch läuft, Sie es aber zum Stillstand bringen wollen: Drücken Sie BREAK.

c2 Wenn das Programm bei einem numerischen INPUT-Befehl steckenbleibt: BREAK nützt da nichts, dagegen aber STOP. Wenn Sie aus Versehen dort etwas anderes stehen haben, das nicht richtig ist, erhalten Sie einen [S]-Cursor als Hinweis auf einen Syntaxfehler, was sehr ärgerlich ist. Löschen Sie die falsche Eingabe mit RUBOUT und drücken Sie anschließend STOP.

c3 Wenn das Programm bei einer Buchstabeneingabe stecken bleibt – [L]-Cursor: Das ist der, den das Handbuch nicht erwähnt – mich hat das einmal eine ganze Stunde Arbeit gekostet, weil das Programm verlorenging. Dumm von mir: Drücken Sie RUBOUT und *dann* STOP.

d Wenn das Programm läuft, aber nicht das Richtige tut: Lesen Sie die Abschnitte über Debugging weiter unten.

e Falls Sie eine neue Methode kennenlernen, hängenzubleiben, *geben Sie mir Bescheid*: Wenn ich Glück habe, erlebt das Buch eine neue Auflage!

SCHLEIFEN

Zehn lagen im Bett, sprach einer sehr nett:
"Dreht euch rum, dreht euch rum!"
Drehten alle sich rum und einer fiel raus.
Neun lagen im Bett, sprach einer . . .

Mit Hilfe der FOR- und NEXT-Befehle können Sie den Computer veranlassen, einen Befehl so oft und so lange auszuführen, wie Sie das haben wollen. Das mag sich zwar nicht so anhören, als sei es besonders interessant, aber in Wahrheit ist das eine der nützlichsten Waffen im Arsenal des Anwenders, weil man durch Variable verändern kann, was mit jedem Schritt bewirkt wird.

Auf welche Weise FOR/NEXT-Schleifen es uns ermöglichen, Tabellen von Werten einer Funktion wie X**3 anzuzeigen, haben wir schon gesehen. Hier eine nicht ganz so simple Anwendung:

Die Fakultätsfunktion n! wird, wie jeder Schüler weiß, definiert durch:

$$n! = n\,(n-1)\,(n-2)\,(n-3)\ldots 3\times 2\times 1$$

Das zeigt die Anzahl möglicher Wege an, n Dinge der Reihe nach zu ordnen. Zum Beispiel: $3! = 3\times 2\times 1 = 6$; die Buchstaben abc können in sechsfach verschiedener Weise angeordnet werden: abc, acb, bac, bca, cab, cba.

Um n! zu berechnen, können wir Schleifen benützen. Dabei geht es darum, in Stufen zu rechnen: $1; 1\times 2; 1\times 2\times 3; 1\times 2\times 3\times 4;$ . . . und so weiter, bis die größte Zahl n ist. In jedem Fall nimmt man das Ergebnis des *vorherigen* Schritts und multipliziert mit der nächsthöheren Zahl. Das ist genau der Aufbau, den eine Schleife braucht. In der Tat hat die Befehlsfolge

```
1Ø  LET I = 1
2Ø  FOR K = 2 TO N
3Ø  LET I = I*K
4Ø  NEXT K
```

genau diese Wirkung.

Was geschieht da? Und auf welche Weise? Zeile 1Ø setzt einfach einen Anfangswert für die Variable I. Zeile 2Ø weist den Computer an, sich anschließende Zeilen immer wieder auszuführen, wobei K der Reihe nach die Werte 2, 3, 4, 5 . . . N annimmt. Zeile 4Ø sagt ihm, wann er das Ende dieser Befehle erreicht hat und wieder an den Anfang der Schleife zurückkehren soll. Beim ersten Durchgang nimmt der Computer also K = 2, und Zeile 3Ø ergibt

$$I = I*K = 1\times 2$$

Beim nächsten Durchgang ist K gleich 3, aus I wird 1×2, also rechnet der Computer I*K, jetzt also $1\times 2\times 3$. Beim nächstenmal hat K den Wert 4, und das Gerät rechnet $1\times 2\times 3\times 4$. So geht es weiter bis zur letzten Stufe, wo K nun N entspricht und so gerechnet wird: $I = 1\times 2\times 3\times 4\times \ldots \times N$. Dank der in Zeile 2Ø gesetzten Begrenzungen weiß der Computer dann, daß die Schleife beendet ist.

Das errechnet N! aber noch nicht für Sie, weil das Programm keine Anweisungen darüber enthält, was N ist, oder dem Computer befiehlt, die Antwort anzuzeigen. Sie müssen die Schleife also in ein größeres Programm einfügen:

```
 5 INPUT N
1Ø LET I = 1
2Ø FOR K = 2 TO N
3Ø LET I = I*K
4Ø NEXT K
5Ø PRINT "FAKULTAET□"; N; "□IST□"; I
```

Zeile 5Ø ist nur für ausführliche Anzeige – Sie erhalten Ergebnisse wie

FAKULTAET 6 ist 720

Beachten Sie die Zwischenräume, dargestellt durch Kästchen, damit das Ganze auch hübsch aussieht.

VERMEHRUNG WIE BEI DEN KANINCHEN

Etwa um das Jahr 1220 stieß ein Leonardo von Pisa, genannt Fibonacci (Sohn der guten Laune), auf ein interessantes Problem bezüglich Kaninchen.

Wenn ein Paar Kaninchen einmal im Monat ein Paar Nachkommen bekommt, es einen Monat dauert, bis auch die neuen Paare sich fortpflanzen, und Sie mit einem Paar beginnen – wie rasch vermehren sie sich dann? Der Einfachheit halber gehen wir von regelmäßigem Nachwuchs in jedem Monat ebenso aus wie davon, daß jedes Paar aus einem Männchen und einem Weibchen besteht.

Es ist von Nutzen, eine Tabelle anzulegen:

Monat	Zahl der Zuchtpaare	Zahl der neuen Paare
Im Monat Ø haben wir 1 Zuchtpaare, Ø neue Paare, demnach		
Ø	1	Ø
Im Monat 1 erhalten wir nur 1 neues Paar		
1	1	1
Im Monat 2 erhalten wir 1 neues Paar, und das vorherige neue Paar wird produktiv:		
2	2	1
Im Monat 3 sind sie alle produktiv, und wir erhalten 2 neue Paare:		
3	3	2
und so weiter:		
4	5	3
5	8	5
6	13	8
7	21	13

Mit der Zeit werden es enorm viele Kaninchen, was ja eigentlich auch nicht verwundert.

Wenn die Gesamtzahl der Kaninchenpaare im Monat M also F(M) betragen soll, dann haben wir F(Ø) = 1, F(1) = 2, F(2) = 3, F(3) = 5, F(4) = 8, und so weiter.

Nun wollen wir allgemeiner werden. Nehmen wir an, im Monat M hätten wir P produktive Paare und N neue Paare. Im Monat darauf werden sie alle produktiv, was P+N *produzierende* Paare ergibt; und die P-Produzenten liefern uns P neue Paare. Damit hat die Tabelle zwei aufeinanderfolgende Zeilen, die so aussehen:

Monat	Zahl der Zuchtpaare	Zahl der neuen Paare
M	P	N
M+1	P+N	P

Diese Tabelle zeigt an, wie wir F(M) mit Hilfe von Schleifen berechnen können. Um von Monat M zu Monat M+1 zu gelangen, müssen wir das alte P in P+N verwandeln und das alte N in P. Das geht so:

```
1Ø  LET B = 1
2Ø  LET N = Ø
3Ø  INPUT M
4Ø  FOR T = 1 TO M
5Ø  LET C = B
6Ø  LET B = B+N
7Ø  LET N = C
8Ø  NEXT T
9Ø  PRINT "F(";M;")□IST□"; P+N
```

Wenn Sie sich das gründlich ansehen, werden Sie erkennen, daß sich das Programm genau an die Schritte hält, nach denen die Tabelle angelegt wurde. Zeilen 1Ø und 2Ø setzen die Anfangswerte von P und N. Zeile 3Ø fragt, für welchen Wert von N wir F(M) haben wollen. Die Zeilen 4Ø bis 8Ø bilden eine Schleife, die aufeinanderfolgende Zeilen der Tabelle hervorbringt. Beachten Sie Zeile 5Ø, die den *alten* Wert von P speichert und ihn C nennt, für die Verwendung in Zeile 7Ø. Wenn Sie das nicht tun, verändert Zeile 6Ø den Wert P zu früh, und Zeile 7Ø liefert den falschen Wert für N.

Die Zahlen F(M) werden *Fibonacci-Reihen* genannt. Wenn Ihnen aufgefallen ist, daß die Kolumnen P und N in der Tabelle dieselben Zahlen enthalten, aber um eine Zeile getrennt (warum übrigens?), werden Sie sehen, daß F(M+2) = F(M+1)+F(M) ist; das heißt, jede Fibonacci-Zahl ist die Summe der beiden vorangegangenen.

Was ist der Wert von F(14)? Von F(77)?

VERSCHACHTELUNGSSCHLEIFEN

Es kommt noch besser. Sie können Schleifen in Schleifen oder sogar Schleifen innerhalb von Schleifen in Schleifen einbauen (soweit der Speicherplatz reicht). Sie werden sich wundern, wie oft das nötig wird.

Ein Beispiel: Nehmen wir an, Sie wollen eine Tabelle von Werten der Fakultätsfunktion erstellen. Sie können eine Schleife von der Art verwenden, wie sie bei X**3 auf Seite 26 benützt wurde, brauchen dann aber eine *zweite Schleife* für die Berechnung von n! Sie erhalten also etwa Folgendes:

```
 5  FOR N = 1 TO 2Ø
1Ø  LET I = 1
2Ø  FOR K = 2 TO N
3Ø  LET I = I*K
4Ø  NEXT K
5Ø  PRINT N, I
6Ø  NEXT N
```

innere Schleife

äußere Schleife

Beachten Sie, daß die Schleife "FOR N/NEXT N" völlig außerhalb der Schleife "FOR K/NEXT K" steht. Das muß so sein, damit das Ganze Sinn ergibt – es hat dieselbe Wirkung, als würde man es in Klammern setzen. Ein Ausdruck [a+(b−2c)]+d ergibt sehr wohl Sinn, aber [a+(b−2c])+d ergibt keinen. Tauschen Sie einmal die Zeilen 4Ø und 6Ø aus und verfolgen Sie, was dann geschieht. Das hat nicht viel Sinn, nicht wahr? Der Haken: Der ZX81 nimmt Programme mit falsch verschachtelten Schleifen an und führt sie aus, aber natürlich nicht mit den beabsichtigten Ergebnissen. Der Computer zeigt *keinen* Fehlerhinweis, also Vorsicht!

SCHRITTGRÖSSE

Wenn Sie lediglich schreiben FOR I = 1 TO 2Ø, dann *unterstellt* der Computer, die Variable I solle der Reihe nach die Werte 1, 2, 3, 4, 5 . . ., 19, 2Ø haben. Das heißt, der Computer geht davon aus, daß Sie in *Schritten* von Größe 1 vorgehen wollen.

Das brauchen Sie aber nicht zu tun. Sie können mit der Taste STEP eine andere Schrittgröße festlegen. Beispielsweise läßt der Befehl

```
1Ø  FOR J = −3 TO 3 STEP .5
```

J die Werte −3, −2.5, −2, −1.5, −1, −Ø.5, Ø, Ø.5, 1, 1.5, 2, 2.5, 3. durchlaufen. Ebenso durchläuft

```
1Ø  FOR J = 3 TO 4 STEP .Ø1
```

die Werte 3, 3.Ø1, 3.Ø2, 3.Ø3 . . . in Hundertstelschritten bis zu 3.98, 3.99, 4. Sie können mit STEP auch Abwärtsschritte machen:

```
10  FOR J = 10 TO 0 STEP -1
```

läßt J die Folge 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 ablaufen. Erinnert Sie das an etwas? Geben Sie folgendes Programm ein:

```
10  FOR J = 10 TO 0 STEP -1
20  PRINT J
30  NEXT J
40  PRINT "ABHEBEN"
```

Nicht gerade überwältigend neu, aber immerhin . . .

ZUFALLSZAHLEN

In den ZX81 ist ein klein wenig Unberechenbarkeit eingebaut.

Der Befehl RND liefert eine "Zufalls"-Zahl zwischen Ø und 1, die gleich Ø, aber nicht gleich 1 sein kann. Eigentlich ist sie nicht wirklich zufällig, sondern eine *Pseudo*-Zufallszahl, die sich alle 65537mal wiederholt, aber in der Praxis wird Ihnen das nicht auffallen. Da niemand weiß, wie eine "Zufalls"-Zahl wirklich aussieht, ist die Bezeichnung "Pseudo", was ja soviel wie "unecht" heißt, ebenfalls ein bißchen unecht.

Nutzen können Sie das bei Spielprogrammen. Um ein Beispiel zu geben: Wenn Sie das Würfelspiel simulieren wollen, gehen Sie davon aus, daß 6*RND eine Zufallszahl zwischen Ø und 6 ist (6 nicht eingeschlossen), also ist INT (6*RND) zufällig entweder Ø, 1, 2, 3, 4 oder 5; 1 + INT (6*RND) ist demnach eine Zufallszahl unter 1, 2, 3, 4, 5, 6. Genau das geschieht beim Würfeln. Um eine zufällige Spielkarte aus einem Paket von 52 Stück herauszusuchen, würden Sie ein ähnliches Spiel mit 52*RND machen, dazu aber schon geschickt programmieren müssen, um Zahlen zwischen Ø und 51 in Namen von Spielkarten wie "KARO-BUBE" zu verwandeln. Es geht aber, wenn Sie es klug anstellen.

Sie können Zufallszahlen auch für statistische Simulation verwenden. Ein hübsches Beispiel dafür finden Sie auf Seite 120 unter dem Titel MONOPOLY-WÜRFELN.

DEBUGGING I

Es heißt, der einprägsame Ausdruck "die Macken austreiben" (englisch: "bugs", also schon wieder die Insekten), sei in der Anfangszeit der Computertechnik entstanden, als Insekten in die Maschinen zu krabbeln pflegten und Kurzschlüsse hervorriefen. Wenn heute ein Computer Fehler macht, liegt es in der Regel am Anwender. Um sie zu beheben, muß man aber trotzdem Bescheid wissen über Debugging.

Jeder, der programmiert, lernt sehr rasch eine harte Wahrheit kennen: Beim erstenmal funktioniert kaum je ein Programm. Das Verfahren, aus einem Programm die Fehler auszumerzen, heißt Fehlersuche oder Debugging, was wir übernehmen wollen. Man muß hier unbedingt systematisch vorgehen, weil selbst bei einem kurzen Programm die Fehlerquelle nicht unbedingt leicht zu finden ist. Und es gibt wenig, was ärgerlicher wäre als Programme mit schwer faßbaren Fehlern.

Sehen wir uns als erstes an, was für Fehler auftreten können. Grob gesprochen, zerfallen sie in zwei Gruppen: *Syntaxfehler* und *Ablauffehler.*

Syntaxfehler

Das sind Fehler, die der Computer sofort erkennen kann, wenn Sie sie eingegeben haben. Angenommen, ich tippe die Zeile

```
5Ø  FOR P = 1-7
```

weil ich von der falschen Annahme ausgehe, das Symbol "−" könne als Entsprechung für den Ausdruck TO gelten. (Das ist, wenn Sie so wollen, ein Fehler bei der Anwendung der Grammatik dieser Sprache; daher der Ausdruck "Syntax").

In solchen Fällen ist der ZX81 für Anfänger besonders nützlich. Er läßt zwar zu, daß Sie das Symbol "−" eingeben, erkennt aber, daß es keine Umstände gibt, unter denen eine vollständige Programmzeile als 5Ø FOR P = 1−7 auftreten kann, zeigt ein Hinweissymbol [S] (um mitzuteilen, daß ein Syntaxfehler vorliegt) und weigert sich, die Zeile anzunehmen, bis das unzulässige "−" ausgetauscht wird. (Viele bekannte Mikrocomputer haben überhaupt nichts dagegen einzuwenden, daß Sie ein Programm mit solchen Fehlern spicken, und erheben erst Einwände, wenn Sie es laufen lassen wollen. Einem erfahrenen Anwender macht das nichts aus, aber der Anfänger neigt zu solchen Fehlern, und man spart eine Menge Zeit, wenn sie einem sofort angezeigt werden).

An dieser Stelle mag manche Leser eine Frage quälen, die wir so formulieren können: "Wenn der ZX81 weiß, daß nach der "1" in Anweisung 5Ø nur das Schlüsselwort "TO" erscheinen kann, warum fügt die Maschine dann nicht von selbst das "TO" ein?"

Die Antwort: Soviel, daß er das tun könnte, weiß er nun auch wieder nicht. Zum Beispiel könnte nach der "1" eine andere Zahl kommen, etwa in

```
5Ø  FOR P = 12 TO 4Ø
```

Es gibt noch andere, kompliziertere Möglichkeiten. Der Befehl

```
5Ø  FOR P = 1-7 TO 5
```

ist zulässig und bedeutet dasselbe wie 5Ø FOR P = − 6 TO 5.)

Das [S]-Hinweiszeichen kann auftauchen, bevor Sie ans Ende einer Zeile kommen. Beispiel:

```
2Ø  LET E* S = Q
```

Der Computer erwartet nach E ein "="-Symbol – oder einen anderen Buchstaben oder eine Zahl, damit er das Hinweiszeichen [S] vor das "*" setzen kann; das tut er allerdings erst, nachdem NEWLINE gedrückt worden ist.

Die Regel bei Syntaxfehlern lautet also: *Achten Sie auf das Hinweiszeichen* [S].Wenn es auftaucht, liegt der Grund meistens auf der Hand. Falls nicht, vergleichen Sie den Befehl, den Sie schreiben, mit den entsprechenden Abschnitten des Handbuchs. Eine Fehlerquelle, die gelegentlich sehr verwirrend sein kann, tritt dann auf, wenn Sie versehentlich ein Schlüsselwort ausschreiben, statt die Sondertaste dafür zu verwenden. Beispiel: Wenn Sie statt des Schlüsselworts TO die beiden Buchstaben "T" und "O" eingeben. Das nimmt der Computer nicht an, obwohl auf dem Bildschirm kein Unterschied zu bemerken ist).

Ablauffehler

Wenn ein Programm gefahren wird, können viele verschiedene Fehlerarten auftreten. Ich stieß einmal auf einen Computerablauf, der infolge eines kaum erkennbaren Fehlers nur bei Programmen mit einer geraden Zahl von Zeichen funktionierte; bevor er behoben wurde, konnte man das dadurch umgehen, daß man irgendwo dort, wo es nicht schädlich war, dem, was nicht laufen wollte, einfach ein zusätzliches Zeichen anfügte.

Es ist nützlicher, konkrete Beispiele von möglichen Fehlern zu geben, als sie allgemein darzustellen. Befassen Sie sich zum Anfang einmal mit den folgenden Programmzeilen:

```
1Ø  FOR P = −5 TO 5
2Ø  LET A = 1Ø/P
3Ø  PRINT A
4Ø  NEXT P
```

Jede einzelne Zeile ist absolut gültiges BASIC, so daß Syntaxfehler nicht vorzuliegen scheinen. Tatsächlich läuft das Programm nach Drücken der Taste "RUN" auch ganz richtig an und liefert

−2	[d.h. 1Ø/(−5)]
−2,5	[d.h. 1Ø/(−4)]
−3,3333333	[d.h. 1Ø/(−3)]
−5	[d.h. 1Ø/(−2)]
−1Ø	[d.h. 1Ø/(−1)]

Auf einmal bleibt es aber mit der Fehlermeldung

```
6/2Ø
```

stehen. Hier ist das Problem ziemlich klar – auch ohne einen Blick auf die Fehlernummer im Handbuch. Erstens zeigt die Mitteilung, daß das Gerät Einspruch gegen die Zeile 20 erhebt, die so lautet

```
2Ø  LET A = 1Ø/P
```

Zweitens ist diese Anweisung schon mehrmals ohne Schwierigkeiten ausgeführt worden – das Problem muß also in einer der Größen liegen, die sich *verändern*, das heißt, beim Wert von A oder P. Der Wert von P, der eben schon erfolgreich behandelt wurde, beträgt −1, der laufende Wert von P demnach Ø. Mit anderen Worten: Der Computer versucht, 1Ø/Ø zu rechnen, was unendlich ist (oder nicht nach Geschmack definiert) und damit nicht bewältigt werden kann, gleichgültig, wieviel Platz die Maschine für die Lösung bereitstellt.

Sie sind zu dieser Schlußfolgerung wahrscheinlich längst schon gekommen, bevor Sie die obige (eher langatmige) Analyse durchgelesen haben, aber ich benütze dieses schlichte Beispiel dazu, auf die *Art* von Fingerzeigen hinzuweisen, die Sie beachten sollten, wenn Sie mit einem Fehler nicht zu Rande kommen.

1 Nehmen Sie sich die Zeile mit dem Problem vor – die Zahl nach "/" in der Schlußmeldung gibt sie an.

2 Stellen Sie fest, ob diese Anweisung mindestens einmal ausgeführt worden ist, bevor sie die Fehlermeldung hervorgerufen hat. Wenn ja, liegt das Problem im gerade geltenden Wert einer der Variablen zu dem Augenblick, als der Fehler auftrat.

3 Nutzen Sie den Meldecode (Handbuch Seite 189–19Ø oder Beilageblatt) – den Wert vor "/" in der Schlußmeldung – als weiteren Hinweis. Bei unserem Beispiel ist das "6", und das ZX81-Handbuch sagt Ihnen, was das heißt: "Arithmetischer Überlauffehler. Die Rechenvorgänge haben zu einer Zahl geführt, die größer als $1Ø^{38}$ ist." Beachten Sie, daß die Fehlermeldung nicht *exakt* mitteilt, was geschehen ist (so sagt sie zum Beispiel nicht: "Versuch, durch Null zu teilen"), so daß es nicht immer ausreicht, einfach einen Blick auf die Fehlermeldung zu werfen, in der Hoffnung, das werde genau erklären, was falsch gelaufen ist.

Das Problem, daß durch Null geteilt wird, tritt ziemlich oft auf und nicht unbedingt immer auf eine so deutliche Weise, wie hier geschildert. Zum Beispiel ruft

```
10  INPUT P
20  INPUT Q
30  INPUT R
40  LET A = (P+Q-R*2)/(5+(P-R)*(P-R)-2*Q)
```

dasselbe Problem hervor, wenn für P, Q und R die Werte 7, 15 und 2 eingegeben werden. (Ausprobieren!) Im allgemeinen empfiehlt es sich, die Divisoren darauf zu prüfen, ob sie Null sind, bevor man zu rechnen versucht. Wir könnten das obige Beispiel auf folgende Weise umschreiben:

```
10  INPUT P
20  INPUT Q
30  INPUT R
32  LET D = 5+(P-R)*(P-R)-2*Q
34  IF D = 0 THEN GOTO 100
40  LET A = (P+Q-R*2)/D
...
100  PRINT "KANN DAS NICHT"
...
```

Die Bedeutung der GOTO-Befehle ist, wie ich hoffe, deutlich genug!

Debugging-Problem

Zum Abschluß hier noch eine Gelegenheit zur Prüfung, ob Sie das bisher Gesagte auch völlig verstanden haben. Das folgende Programm soll eine Folge positiver Zahlen akzeptieren und ihr Mittel angeben. Wenn wir das Mittel von 2.4, 8.1, 7 und 14 errechnen wollten, müßten wir eingeben:

```
2.4
8.1
7
14
-1
```

Das "-1" am Ende soll lediglich anzeigen, daß keine weiteren Daten eingegeben werden sollen, und gehört nicht zu den Daten. Ein solcher Wert wird oft als Begrenzer bezeichnet; wie er sich auswirkt, erkennen Sie, wenn Sie sich Zeile 40 des folgenden Programms ansehen.

```
10  LET S = 0
20  LET C = 0
30  INPUT N
40  IF N < 0 THEN GOTO 100
50  LET C = C+I
60  LET Z = S+N
70  GOTO 30
100  PRINT "DAS MITTEL IST□"; S/C
```

Das Listing enthält einige Schreibfehler. Versuchen Sie, ob Sie sie ausbessern können, indem Sie *das Programm so eingeben, wie es hier steht und daran herumbasteln. Wenn Sie es versucht haben, vergleichen Sie Ihre Lösung mit meiner auf Seite 55.*

Viel Glück bei der Fehlersuche!

GRAFISCH DARSTELLEN

Der ZX81 kann noch mehr!

Der ZX81 kann nicht nur rechnen, sondern auch Grafiken zeichnen. Vielleicht nicht so meisterhaft wie Leonardo da Vinci, aber immerhin gut genug, um attraktive Darstellungen zu liefern, die Grundprinzipien der Computergrafik vorführen. Die Grafiken beim ZX81 sind von "niedrigem Auflösungsvermögen" – das heißt *grob* – aber das besagt nur, daß er statt mit kleinen Punkten auf einer großen mit großen Punkten auf einer kleinen Fläche zeichnet. Was Sie also durch Grobgrafik lernen können, läßt sich bei erheblich leistungsfähigeren Geräten in viel größerem Umfang anwenden.

Der ZX81 kann sogar auf zweierlei verschiedene Weise zeichnen: mit PLOT und PRINT. Wir fangen mit PLOT an, weil das ein bißchen einfacher ist. PRINT wird später besprochen, auf Seite 62.

Beim Fernseh-Bildschirm wird für die Zwecke von PLOT davon ausgegangen, daß er in 2816 winzige Quadrate (eigentlich sind es Rechtecke) aufgeteilt sei, die in der Fachsprache *Pixel* heißen (von *pic*ture *el*ement*s* – was auf deutsch schlicht "Bildelemente" heißt). Sie werden bestimmt durch zwei

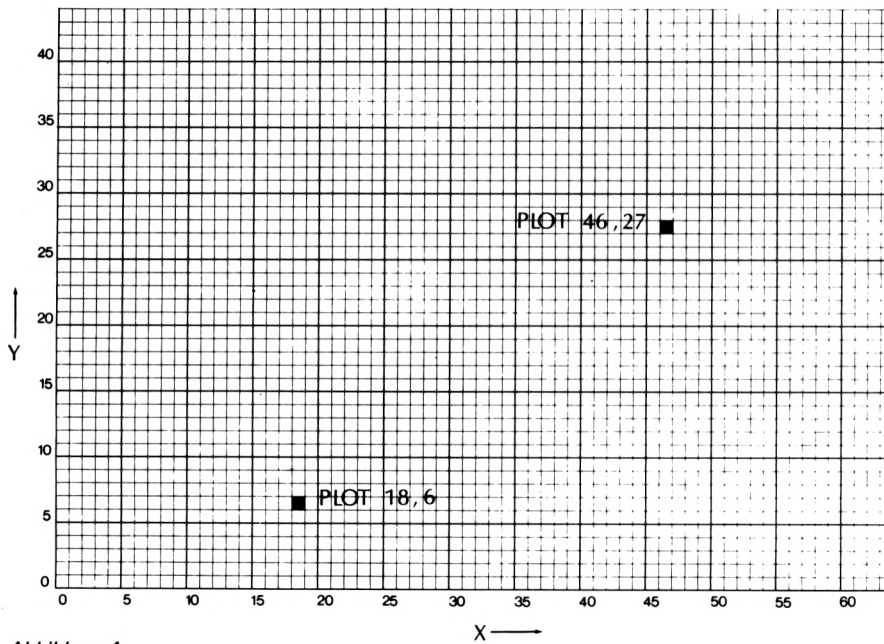


Abbildung 4

Koordinaten, die angeben, wie weit sie sich auf dem Bildschirm von links nach rechts gesehen und wie weit oben sie sich befinden. Hier das vollständige Diagramm aller 2816 Quadrate. Die horizontale Koordinate reicht von Ø bis 63, die vertikale von Ø bis 43.

Der Befehl PLOT X, Y füllt das Quadrat mit der horizontalen Koordinate X und der vertikalen Koordinate Y schwarz aus. Mehr geschieht nicht. Mehr braucht man aber auch nicht, um zeichnen zu können: Einen Bleistift (geschwärzte Quadrate), Papier (die Anordnung von 64 × 44 Pixel auf dem Bildschirm), und die Möglichkeit, zu entscheiden, wohin der Bleistift geführt werden soll (X und Y).

Auf die gleiche Weise entfernt UNPLOT X, Y den bei X, Y gezeichneten Punkt (wenn dort einer ist) – wie ein Radiergummi.

Aus diesen beiden Befehlen müssen wir alle anderen Formen *aufbauen*, die wir darstellen wollen. Angenommen, wir wollen eine horizontal verlaufende Linie zeichnen. Wir können sie uns als eine Folge von PLOT-bestimmten Quadraten nebeneinander vorstellen. Wir könnten einen Code schreiben, der folgendermaßen aussieht:

```
1Ø  PLOT 1Ø, 1Ø
2Ø  PLOT 11, 1Ø
3Ø  PLOT 12, 1Ø
4Ø  PLOT 13, 1Ø
5Ø  PLOT 14, 1Ø
```

um bei Y = 1Ø eine Linie zu ziehen, die 5 Pixel lang ist.

Da der Wert X in gleiche Einserschritte (1Ø, 11, 12, 13, etc.) unterteilt ist, empfiehlt es sich, das mit einer FOR-Schleife umzuschreiben:

```
1Ø  FOR X = 1Ø TO 14
2Ø  PLOT X, 1Ø
3Ø  NEXT X
```

Jetzt fällt es leicht, beispielsweise ein Rechteck zu zeichnen. Gehen wir davon aus, daß die linke untere Ecke bestimmt wird durch die X- und Y-Werte LX (das X ganz links) und UY (das Y ganz unten), und daß Höhe und Breite H und B sein sollen.
Das Programm sieht dann so aus:

```
 8Ø  FOR X = LX TO LX+B
 9Ø  PLOT X, UY
1ØØ  NEXT X
11Ø  FOR Y = UY TO UY + H
12Ø  PLOT LX+B,Y
13Ø  NEXT Y
```

zeichnet den unteren Rand

zeichnet die rechte Spalte

```
14Ø  FOR X = LX+B TO LX STEP -1
15Ø  PLOT X, UY + H
16Ø  NEXT X
17Ø  FOR Y = UY+H TO UY STEP -1
18Ø  PLOT LX,Y
19Ø  NEXT Y
```

zeichnet den oberen Rand

zeichnet die linke Seite

Das wollen wir noch vertiefen. Wenn wir uns eine Linie als eine Reihe von Pixel vorstellen können, warum dann nicht auch eine Fläche als eine Reihe von Linien? Also etwa so:

```
5ØØ  FOR Y = 2Ø TO 3Ø
51Ø
52Ø
53Ø
54Ø  NEXT Y
```

zeichnet eine Linie zwischen X = 1Ø und 18

Das sollte eine Fläche zwischen Y = 2Ø und 3Ø und X = 1Ø und 18 ergeben.

Wie der Auftrag, eine Linie für jeden beliebigen Wert von B zu zeichnen, aussieht, wissen wir schon:

```
FOR X = 1Ø TO 18
PLOT X, Y
NEXT X
```

Das Ganze sieht demnach so aus:

```
5ØØ  FOR Y = 2Ø TO 3Ø
51Ø  FOR X = 1Ø TO 18
52Ø  PLOT X, Y
53Ø  NEXT X
54Ø  NEXT Y
```

Ich habe mich damit deshalb so lange aufgehalten, weil die Vorstellung einer Schleife innerhalb einer Schleife einfach ein bißchen verwirrend ist; und obwohl wir damit schon kurz zu tun hatten, ergibt sich nicht auf Anhieb, daß genau das nötig ist, um Balken zu zeichnen. Das ist eine zulässige Konstruktion, die sehr häufig auftritt. Ich wiederhole: Jede innere Schleife muß von der äußeren *völlig* eingeschlossen sein. Es ist also keine gute Idee, wenn man schreibt:

```
5Ø  FOR X = 2Ø TO 4Ø
6Ø  FOR Y = 3Ø TO 35
...
1ØØ  NEXT X
11Ø  NEXT Y
```

Wenn Sie es sich recht überlegen, ergibt das auch keinen Sinn. Seltsamerweise "gehorcht" der ZX81 hier trotzdem, aber die Resultate werden nicht dem entsprechen, was Sie von einer verschachtelten Schleife erwarten.

Es wäre ganz leicht, die beiden Programme, die wir haben, so miteinander zu verbinden, daß wir auf dem ganzen Bildschirm Rechtecke und Flächen in jedem Muster zeichnen können, das uns vorschwebt – ein ganz grundlegendes Beispiel für rechnerunterstütztes Konstruieren. Wir brauchen nur die Einzelheiten von Größe und Form einzugeben, zu prüfen, ob das Ganze auf den Schirm paßt (damit das Programm nicht zusammenbricht), zu fragen, ob Fläche oder Rechteck gezeichnet werden soll, und die entsprechende Programmverzweigung vorzunehmen. An deren Ende kehren wir zu den Input-Anweisungen zurück, damit eine neue Form eingegeben werden kann. Das Ganze sieht dann so aus:

```
1Ø  INPUT LX
2Ø  INPUT UY
3Ø  INPUT H
4Ø  INPUT B
45  IF LX+B<=63 AND UY+H<=43 THEN GOTO 6Ø
5Ø  PRINT AT Ø,Ø;"UEBER RAND"
55  GOTO 1Ø
6Ø  INPUT T              (T = Ø für Rechteck, 1 für Fläche)
7Ø  IF T = 1 THEN GOTO 5ØØ
8Ø  FOR X = L X TO LX + B
9Ø  PLOT X, UY
1ØØ  NEXT X
11Ø  FOR Y = UY TO UY + H
12Ø  PLOT LX + B, Y
13Ø  NEXT Y
14Ø  FOR X = LX + B TO LX STEP -1
15Ø  PLOT X, UY + H
16Ø  NEXT X
```

```
17Ø  FOR Y = UY + H TO UY STEP -1
18Ø  PLOT LX, Y
19Ø  NEXT Y
2ØØ  GOTO 1Ø
5ØØ  FOR Y = UY TO UY + H
51Ø  FOR X = LX TO LX + B
52Ø  PLOT X, Y
53Ø  NEXT X
54Ø  NEXT Y
55Ø  GOTO 1Ø
```

Probieren Sie das Programm aus, und Sie werden sehen, daß wir eine einfache Grundlage für ein grafisches Konstruktionsprogramm haben (vorausgesetzt, Sie mögen Mustertapeten in Schwarzweiß). Ihnen fallen sicher noch andere Standardformen ein (Dreiecke? Parallelogramme?), die man mit zusätzlichen Programmen aufbauen kann.

AUF BAND SPEICHERN

Gute Programme aufzubewahren, lohnt sich. Beim ZX81 geht das mit Hilfe eines ganz gewöhnlichen Kassettenrecorders. Aber es erfordert Geschick, will man auf Band speichern.

Sie haben drei Stunden an einem gemein komplizierten Programm gearbeitet, um Luke Skywalker aus dem Todesstern zu befreien, oder die Strömung einer komprimierbaren Flüssigkeit um eine Banane oder sonstwas zu berechnen, und endlich läuft das Ding. Es dauert freilich nicht so lange, das Programm einzutippen, wie es überhaupt zum Laufen zu bringen, aber es wäre schon sehr deprimierend, wenn Sie das bei jedem Programm tun müßten, das Sie aufbewahren wollen.

Dank jenem Genie jedoch, das den Kassettenrecorder erfunden hat, müssen Sie das auch gar nicht tun. Sie können Programme mit SAVE auf Tonband speichern und je nach Bedarf mit LOAD in den ZX81 laden.

Wenn alles gutgeht – und das wird wahrscheinlich der Fall sein – steht im Handbuch genug für Sie, um das zu bewältigen. Aber ganz so einfach ist es doch nicht, und es gibt ein, zwei Kniffe, die das Handbuch nicht erwähnt. Ich bin der Meinung, sie könnten nützlich für Sie sein.

Sicherstellen

Sie brauchen: den ZX81, ein Fernsehgerät, die üblichen Anschlußkabel und natürlich einen Kassettenrecorder. Die meisten Modelle sind geeignet; die billigen Monogeräte sind sogar *besser*. Sie werden vermutlich schon ein Gerät haben und es, wenn das irgendwie geht, auch verwenden wollen.

Dem ZX81 ist ein Anschlußkabel für Tonbandgeräte beigefügt: ein Kabel aus zwei Strängen mit kleinen schwarzen Steckern an beiden Enden. An beiden Enden eines Kabelstrangs ist ein gelber Streifen zu sehen.

Schreiben Sie, wie gewohnt, mit dem Computer ein Programm. Es braucht für eine Probe nichts zu *leisten:* Sie benötigen nur eine längere Liste von Zeichen, damit Sie Zeit haben, sich die Vorgänge anzusehen. Am raschesten geht das mit vielen REM-Befehlen, gefolgt von willkürlich eingetippten Buchstaben.

```
1Ø REM "TEST3"
2Ø REM HHHHHHQQQ
etc.
```

wie in Abbildung 5.

Schließen Sie den Kassettenrecorder an und vergewissern Sie sich, daß er funktioniert. (Sie können ihn auch mit Batterien betreiben – das ist vielleicht sogar von Vorteil; siehe weiter unten.) Legen Sie eine leere oder auch eine bespielte Kassette ein, bei der es nichts ausmacht, wenn sie gelöscht wird.

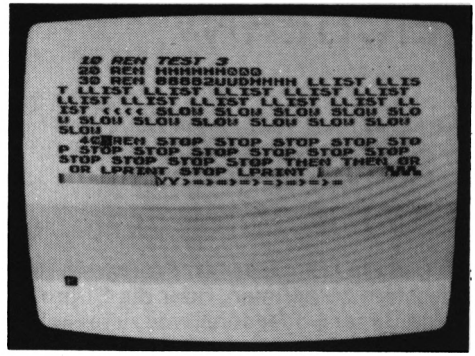


Abbildung 5
Mit vielen REMs und beliebigen Eingaben rasch proben

Schließen Sie die MIC-Buchse des Kassettenrecorders an die MIC-Buchse des ZX81 an. *Schließen Sie jetzt noch nichts an die Buchsen EAR oder MONITOR an.* Verwenden Sie *eines* der beiden Kabel, die Sinclair mitliefert – meinetwegen das mit den gelben Streifen. Das andere lassen Sie. (Sie brauchen sie aber nicht auseinanderzureißen!).

Drehen Sie den Lautstärkeknopf des Kassettenrecorders bis zum Anschlag auf und die Tonhöhe auf ganz hell (damit der Klang möglichst blechern wird). Bei den meisten Geräten spielt das für die Aufnahme kaum eine Rolle, aber später müssen Sie das doch tun und machen es deshalb vorsichtshalber gleich.

Rütteln Sie an den Steckern in den Buchsen, um sich zu vergewissern, daß sie fest sitzen. In der Regel spürt man, ob es "richtig" ist.

Stellen Sie das Kassettengerät genauso ein wie zu einer Aufnahme – drücken Sie also auf die Tasten RECORD und PLAY. Die Bandspulen beginnen sich zu drehen; keine Sorge, genau das ist beabsichtigt. Anlaß zur Eile besteht ja nicht. Achten Sie darauf, daß das Vorspannband durchgelaufen ist, bevor Sie den nächsten Schritt tun.

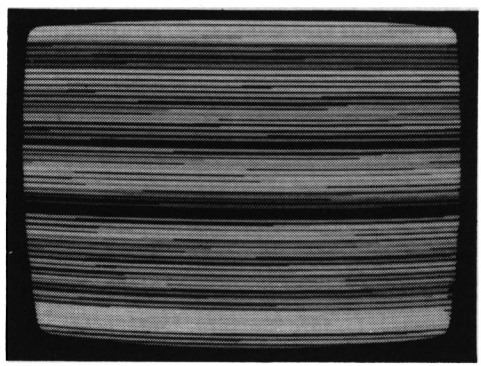


Abbildung 6
Das Programm wird gesichert.

Drücken Sie am ZX81 auf SAVE (also auf die Taste S; das wird automatisch als "SAVE" gelesen), dann "X" (das ist unbedingt erforderlich; darunter tut es der Computer nicht) und schließlich NEWLINE. Auf dem Bildschirm müßte nun etwas vorgehen.

Als erstes wird er für ungefähr 5 Sekunden ganz leer.

Dann sollten Sie eine Vielzahl von Querstreifen sehen, wie auf Abbildung 6. Sie kommen von dem Programm, das eingelesen wird und bleiben etwa 15 bis 2Ø Sekunden auf dem Schirm.

Dann sollte der Bildschirm wieder leer werden und die Meldung Ø/Ø zeigen. Das bedeutet, die Sache ist erledigt und der Computer mit dem Ergebnis zufrieden. Drücken Sie beim Recorder auf STOP.

Hat das Gerät wirklich gesichert?

Der Haken bei der Sache ist der: Das Ø/Ø bedeutet lediglich, daß der ZX81 mit dem zufrieden ist, was er glaubt, ausgegeben zu haben; es sagt nichts darüber aus, was der Kassettenrecorder aufgenommen hat. (Versuchen Sie, das alles ohne einen Recorder zu machen; der ZX81 merkt nichts davon, daß gar keiner da ist).

Um festzustellen, ob das Programm richtig gesichert ist, müssen Sie versuchen, es zu laden, aber bevor Sie das tun, können Sie bis zu einem gewissen Grade prüfen, ob überhaupt Hoffnung besteht, und zwar, indem Sie das Band abhören.

Lassen Sie es einfach bis zum Anfang zurücklaufen und spielen Sie dann mit PLAY ab.

Als erstes ist ein ziemlich mechanisch klingendes Brummen zu hören. Das gibt der ZX81 aus, wenn er nicht wirklich sichert. Sinclair beschreibt das als "leise", aber ganz stimmt das nicht.

Anschließend bleibt es eine Weile relativ still, was der Leerung des Bildschirms entspricht. Wenn es sich *nicht* still anhört, haben Sie vielleicht Schwierigkeiten mit Netzbrummen oder Ähnlichem; dazu später noch.

Schließlich meldet sich ein schrilles, unregelmäßiges Geräusch, das wie Morsen bei Hochgeschwindigkeit klingt. Das ist das Programm. Der Bildschirm sieht dann aus wie auf Abbildung 6. Bei voll aufgedrehtem Ton ist das ohrenbetäubend laut. Sie müssen in diesem Fall den Ton zurückdrehen. Hören Sie das nicht, dann haben Sie vielleicht erneut Probleme.

Laden

Bis jetzt haben wir nichts anderes getan, als unter Beigabe einiger Fotos das Handbuch noch einmal durchzukauen, aber jetzt kommt etwas Zusätzliches. Nehmen wir an, alles scheint bis dahin gut gegangen zu sein. Jetzt wollen Sie das Programm wieder in den Computer zurückladen.

Damit Sie Gewißheit haben, daß das, was da schließlich auftaucht, auch wirklich vom Tonband kommt, drücken Sie auf NEW und löschen damit das Programm im ZX81. (Jetzt sehen Sie, warum ich gesagt habe, Sie sollten viele REM-Zeilen und sinnloses Zeug nehmen: wenn das Laden nicht klappt, haben Sie alles verloren).

Lassen Sie das Band wieder zurücklaufen.

Entfernen Sie den Anschluß zwischen Kassettenspieler und ZX81 und verbinden Sie die Buchse EAR am ZX81 mit der Buchse EAR oder MONITOR (Lautsprecher) am Kassettengerät.

Drehen Sie den Regler auf etwa Dreiviertel der vollen Lautstärke. Das ist eine erste Annäherung; Sie müssen vielleicht experimentieren; Genaueres später.

Geben Sie mit dem ZX81 LOAD "X" ein und danach NEWLINE. Der Bildschirm wird ein Muster in der Art von Abbildung 7 zeigen, vermutlich ohne die vertikalen Zickzacklinien.

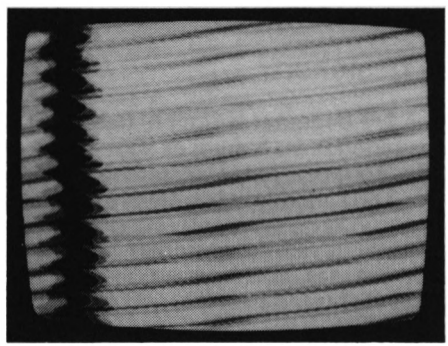


Abbildung 7
"Stummes" Bild beim Laden

Lassen Sie das Kassettengerät laufen. (Drücken Sie nicht aus Versehen auf RECORD, sonst löschen Sie alles und müssen ganz von vorne anfangen). Das Bild auf dem Schirm kann für Sekunden ein wenig durcheinandergeraten, wenn auf dem Tonband Nebengeräusche sind. Sobald Sie zum "stummen" Teil kommen, müßten Sie wieder ein Bild wie auf Abb. 7 sehen (oder etwas Ähnliches).

Wenn das Band zum "Programmteil" gelangt – Abbildung 6, bei der Eingabe – tritt eine auffallende Veränderung ein. Sie sollten etwas in der Art von Abb. 8 sehen. Es kann sogar sein, daß Sie im Fernseher ganz schwach einen Morseton hören (Bildton).

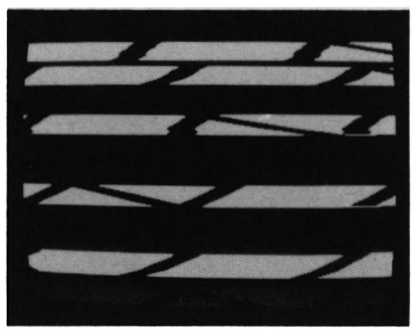


Abbildung 8.
Das Programm wird richtig geladen

Wenn das vorbei ist, sollte der Bildschirm leer werden und die Meldung Ø/Ø erscheinen. Geschieht das, ist alles in Ordnung. Drücken Sie LIST. Ihr Programm müßte dann erscheinen. Drücken Sie RUN, und es müßte laufen. (Nicht, daß REM-Serien viel leisten . . .)

Aber manchmal kommt es nicht so – vor allem beim erstenmal noch nicht . . .

Und wenn nicht . . .

Nicht verzweifeln. Im allerschlimmsten Fall schickt Ihnen Sinclair ein neues Gerät, und DM 200,– von Ihrem Geld liegen eine Weile fest; oder Sie müssen vielleicht doch DM 120,– oder so für einen geeigneten Kassettenrecorder lockermachen. Aber vielleicht ist es gar nicht so schlimm.

Die Bilder auf dem Schirm verraten schon vieles.

a Es sieht immer aus wie auf Abbildung 7. Versuchen Sie zuerst, das Ganze zu wiederholen, aber mit stärker aufgedrehtem Tonregler. Während des "Programm"teils – dem Morsen – erhalten Sie vielleicht Abbildung 7, wenn der Ton zu leise eingestellt ist. Bauen Sie alles genauso auf wie vorher, drehen Sie aber bei dieser Bandstelle am Tonregler; vermutlich erleben Sie, daß der Schirm plötzlich auf Abbildung 8 umschaltet. (Das hält die Gefahr, daß Brummen den "stummen" Teil überlagert, möglichst gering).

Manchmal hilft das nichts. Mir ist es so gegangen, und es hat eine Weile gedauert, bis ich dahinterkam, was nicht in Ordnung war. Die Sache ist die: Vorausgesetzt, Ihr Kassettenrecorder überträgt im Kopfhörer ein gutes Signal, sollte der Bildschirm mehr bieten, als Sie auf Abbildung 7 sehen können, und wenn er es nicht tut, ist mit dem ZX81, oder, was näherliegt, mit den Anschlußkabeln etwas nicht in Ordnung. Es könnte sein, daß die Stecker nicht richtig sitzen. Rütteln Sie daran, während Sie laden. Drehen Sie gleichzeitig den Tonregler hin und her.

Prüfen Sie die Anschlußkabel darauf, ob sie unbeschädigt sind und keine Kurzschlüsse vorliegen. (Bei mir ist es so gewesen – in einem der Stecker war ein Kurzschluß.) Ganz einfach läßt sich das mit einer Batterie und der Birne einer Taschenlampe überprüfen. Siehe dazu Seite 54. Wenn das alles nicht hilft, schicken Sie das Gerät an Sinclair zurück. Teilen Sie mit, was für einen Recorder Sie haben. Warten Sie. (Bedauerlich, aber so ist das, wenn Hardware ausfällt).

b Sie erhalten statt Abbildung 8 Abbildung 9.

Versuchen Sie, beim Laden den Ton *leiser* zu stellen. Vermutlich ist er zu laut.

c Alles scheint in Ordnung zu sein, aber das Programm wird trotzdem nicht geladen, und Sie erhalten kein Ø/Ø. Am ehesten könnte Brummen auf dem Band die Schuld tragen. Bei Batteriebetrieb sollte das aufhören. (Das ist der Grund, warum ich empfehle, nicht *beide* Kabel anzuschließen: Wie schon im Handbuch steht, kann das zu einem Rückkoppelungs-Schwingkreis führen und auf dem Band Brummen hervorrufen. Nach meiner Erfahrung kommt das ziemlich häufig vor). Wenn das immer noch nichts hilft, hören Sie sich den "stummen" Bandteil noch einmal an. Wie "stumm" geht es da zu? Wenn man Geräusche hört oder der "Programm"teil undeutlich ist, haben Sie vielleicht doch einen Recorder, der nichts taugt.

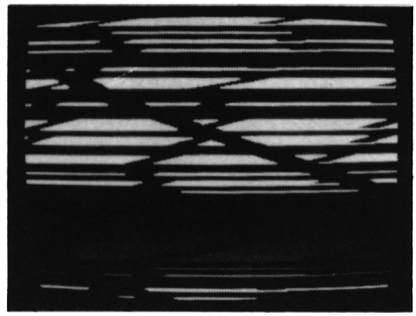


Abbildung 9.
Probleme. Stellen Sie den Ton leiser.

d Wenn Sie immer noch Schwierigkeiten haben, borgen Sie sich bei einem Freund dessen Kassettengerät, und zwar möglichst von einem anderen Fabrikat. Besser noch, suchen Sie sich jemand, dessen ZX81 richtig lädt, und ersetzen Sie in seiner Anlage den Computer durch den Ihrigen, um festzustellen, woran es liegt. Wenden Sie sich an den User-Club in Ihrem Heimatort, falls Sie niemand kennen. Siehe dazu auch Seite 111.

Feinheiten

Sie können einem Programm beim Sichern einen *Namen* geben, indem Sie nicht bloß eintippen.

```
SAVE "X"
```

sondern (meinetwegen)

```
SAVE "MARION"
```

Der ZX81 weiß dann, daß "MARION" das Programm ist, das Sie eben sichergestellt haben. Zum Laden geben Sie ein

```
LOAD "MARION"
```

und er wird alle anderen Programme nicht beachten, bis MARION auftaucht. Der Bildschirm wird bei anderen Programmen alle richtigen Bilder zeigen, aber das Ø/Ø nicht bringen.

Sie können das ausprobieren, indem Sie auf dem Band mehrere Programme laufen lassen. Die erste Programmzeile kann als Hinweis auf den verwendeten Namen dienen:

```
1Ø REM "MARION"
```

Nachbemerkung: Beim Befehl LOAD wird *nicht* nach diesem Namen, sondern nach jenem gesucht, den Sie für SAVE gewählt hatten. Sie können SAVE befehlen mit MARION, REM mit TONY – was geladen werden muß, bleibt trotzdem MARION.

Tips für Tonbänder

Jedes Tonband von handelsüblicher Qualität müßte geeignet sein, aber die guten rauschen meist weniger und haben im oberen Tonbereich einen besseren Frequenzgang. Nehmen Sie ein *kurzes* Band (C 3Ø oder C 45, falls erhältlich). Dafür gibt es zwei Gründe. 1) Sie werden gewiß nicht zehn Minuten darauf warten wollen, daß das Band sich bis zu dem Programm abspult, das Sie suchen. 2) Kurze Bänder sind in der Regel dicker, und bei langer Lagerung kommt es nicht zu starkem "Kopiereffekt" von magnetischen Signalen.

Nehmen Sie den Namen des Programms kurz vor dessen Beginn auf Band, indem Sie ihn ins Mikrofon sprechen. Das hat den Zweck, es leichter wiederzufinden.

Wenn Ihr Kassettenrecorder über einen Bandzähler verfügt, haben Sie gut lachen. Wenn nicht, lohnt es sich dann, ein neues Gerät anzuschaffen? Überlegen Sie sich das.

Sorgen Sie dafür, daß die Tonköpfe immer ganz sauber sind. Am besten geht das mit Alkohol und Wattestäbchen. Der ZX81 hat Einwände gegen Geräusche, die bei Udo Lindenberg oder Punkrock in Ordnung sein mögen. Achten Sie auch darauf, daß die Tonköpfe magnetisiert bleiben.

Nehmen Sie kurze Namen – mit langen vergeuden Sie Zeit beim Laden.

Selbst wenn Sie die Lautstärke für Ihre eigenen Programmbänder richtig gewählt haben, kann es vorkommen, daß sie bei handelsüblichen Software-Kassetten verändert werden muß, weil diese bei anderen Lautstärken aufgenommen sein können.

Nicht verwendbare Kassettenrecorder

Falls Ihr Gerät nicht schon uralt oder ein ganz ausgefallenes ist, sollte es kaum Schwierigkeiten geben. Dagegen sind schon Probleme bei den folgenden Fabrikaten aufgetaucht:

BOOTS CTR 6ØØ, HITACHI TRQ-299, ITT KB SL52, KASUGA

KC-8Ø78, PRINTZSOUND TR255C, RT-VC KIT.

Sie machen nicht *immer* Schwierigkeiten: Ich habe ein HITACHI-Gerät benützt, das richtig funktionierte; allerdings war es beim Laden für die Lautstärke ein bißchen empfindlich. Wenn Sie aber Schwierigkeiten feststellen und eines der genannten Geräte haben, ist zu empfehlen, daß Sie sich einen anderen Recorder ausborgen, damit Sie verfolgen können, was geschieht, *bevor* Sie alles an den Hersteller zurückschicken (das kann nämlich *ewig* dauern) . . .

Wie man prüft, ob die Kabel in Ordnung sind

Schließen Sie mit zwei langen Drähten, wie oben dargestellt, eine Batterie an eine Taschenlampenbirne an (1,5 Volt). Wenn die Drähte an ein Kabel gehalten werden, das keine Brüche aufweist, müßte die Birne aufleuchten. Hält man sie an Kabel, die nicht mehr verwendet werden sollten, dürfte die Birne nicht aufleuchten. (Falls Sie einen Stromprüfer haben, fein; aber dann brauchen Sie sich mit dem Abschnitt hier gar nicht zu befassen).

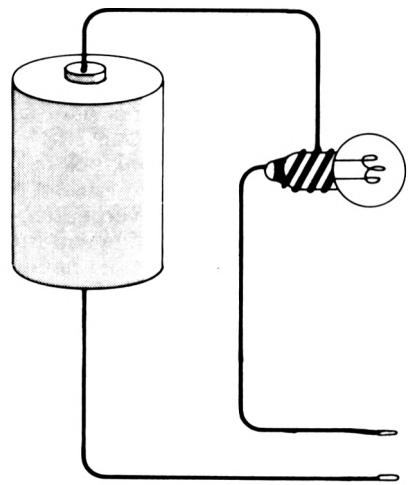


Abbildung 10

DEBUGGING II

Bleistift und Papier haben immer noch ihren Nutzen . . .

Vorher, auf Seite 40/41, habe ich Sie bei dem Programm zur Berechnung des Mittels alleingelassen, das unten noch einmal gezeigt wird (damit Sie nicht dauernd hin- und herblättern müssen), und vorgeschlagen, es einmal mit der Fehlersuche zu probieren.

```
10  LET S = 0
20  LET C = 0
30  INPUT N
40  IF N < 0 THEN GOTO 100
50  LET C = C + 1
60  LET Z = S + N
70  GOTO 30
100 PRINT "MITTEL IST□"; S/C
```

Wir wollen an dieses Problem schrittweise herangehen. Offenkundig muß man als erstes ausprobieren, ob das Programm so funktioniert, wie es dasteht. Wir geben es also in den ZX81 ein und fahren es mit verschiedenen einfachen Dateneingaben. Versuchen wir es einmal mit

3

7

5

−1 (Achtung: dieser Wert dient nur als Abschlußeingabe oder Begrenzer)

Herauskommen *müßte* 5. Wenn Sie das Programm ablaufen lassen, erscheint aber die Fehlermeldung 2/50. Hmm. Das bedeutet, daß in Zeile 50 eine vorher nicht definierte Variable verwendet worden ist. Aus dem Listing können wir erkennen, daß das I ist. Geben wir I also einen Wert, meinetwegen

```
5  LET I = 0
```

Das behebt die Dinge so weit, daß das Programm über Zeile 50 hinaus weiterläuft. Wenn wir Zeile 5 anfügen und erneut RUN drücken, stellen wir fest, daß die Meldung

MITTEL IST

zwar richtig angezeigt wird, dann aber eine Fehlermeldung folgt, nämlich

6/10Ø

Dieser Meldung sind wir schon einmal begegnet. Sie erinnern sich vielleicht, daß die "6" anzeigt, der Computer habe versucht, einen Rechenvorgang auszuführen, der zu einem so großen Resultat führt, daß er es nicht bewältigen kann, und daß die "1ØØ" die Zeilennummer angibt, wo das Problem entstanden ist. Man darf also vermuten, daß das Programm, wie schon beim letztenmal, als die Fehlermeldung auftauchte, versucht hat, durch Null zu teilen. Bevor wir uns ernsthaft als Detektive betätigen, wollen wir noch ein paar Spuren sammeln. Versuchen Sie es mit anderen Daten:

2

1

4

6

−1

Dabei kommen genau dieselben Meldungen heraus. Als Arbeitshypothese scheint sich demnach anzubieten: "Gleichgültig, welche Daten eingegeben werden, das Programm schließt mit einer Meldung über arithmetischen Überlauf ab." Probieren Sie das Programm mit drei oder vier anderen Dateneingaben aus, und Sie werden sehen, daß unsere Arbeitshypothese immer glaubhafter wird.

Wo fangen wir mit dem Suchen also an? Ich habe vorhin von "Detektivarbeit" und "Spuren" gesprochen und das nicht leichtfertig getan. Programme zu "entfehlern", hat durchaus Ähnlichkeit mit detektivischer Tätigkeit, und jeder Detektiv wird Ihnen sagen, daß man, um Erfolg zu haben, wie ein Übeltäter denken muß, nach dem Spruch: "Nur ein Dieb fängt einen Dieb." Der Übeltäter ist in diesem Fall der ZX81, also wird von uns der Versuch verlangt, so zu denken wie der Computer. Als erstes müssen Sie Ihre Gedankenabläufe verlangsamen. Das hat Sie überrascht, nicht wahr? Sie standen unter dem Eindruck, Computer wären besonders schnell. Das sind sie auch, aber über ein Problem "denken" sie in der Regel recht umständlich nach. Als nächstes nehmen Sie sich ein Modell vom Speicher des Computers vor, oder wenigstens den Teil von ihm, der für das vorliegende Problem von Belang ist. In unserem Beispiel sind fünf Speicherelemente festgelegt worden und haben die Namen S, C, N, I und Z erhalten. Ein geeignetes Modell wird also eine Tabelle sein, an der wir zeigen können, wie der Inhalt dieser Elemente sich verändert, während das Programm ausgeführt wird. Außerdem kann es nützlich sein, einen Hinweis auf die Zeilennummer zu haben, wo eine Verzweigung stattfindet; das ist allerdings nicht unbedingt erforderlich. Die Tabelle könnte also folgendes Aussehen haben:

Zeile Nr.	S	C	N	I	Z	Verzweigung

Ich habe eine zusätzliche Spalte mit der Bezeichnung "Verzweigung" eingeführt. Auf ihre Bedeutung komme ich gleich zu sprechen. Wir bauen die Tabelle nun so auf, daß wir uns der Reihe nach ansehen, was jeder Befehl bewirkt. Wir müssen dazu eine Datenfolge definieren. Nehmen wir:

2
1
4
6
−1

Die ersten Anweisungen, die zu befolgen sind, stehen also in den Zeilen 5 und 1Ø und setzen die Speicherelemente I und S mit Null fest. Die Tabelle hat dann:

Zeile Nr.	S	C	N	I	Z	Verzweigung
5				Ø		
1Ø	Ø					

Nachdem Zeile 2Ø ausgeführt ist, haben wir:

Zeile Nr.	S	C	N	I	Z	Verzweigung
5				Ø		
1Ø	Ø					
2Ø		Ø				

Zeile 3Ø übernimmt den ersten Wert aus der Datenfolge und setzt ihn bei N ein. Das ergibt:

Zeile Nr.	S	C	N	I	Z	Verzweigung
5				Ø		
1Ø	Ø					
2Ø		Ø				
3Ø			2			

Zeile 4Ø ist eine "Wenn"-Anweisung und führt deshalb nur die Probe darauf aus, ob N negativ ist. Das ist nicht der Fall (zur Zeit ist es 2), also ist die Probe falsch, was wir in der Spalte "Verzweigung" mit einem x anzeigen; der Sprung zu 1ØØ findet also nicht statt. Wenn eine Probe richtig ist und der Sprung stattfindet, werden wir das in der Spalte "Verzweigung" mit einem "√" anzeigen. Nun haben wir:

Zeile Nr.	S	C	N	I	Z	Verzweigung
5				Ø		
1Ø	Ø					
2Ø		Ø				
3Ø			2			
4Ø						×

Zeile 5Ø verlangt vom Computer, daß er den Inhalt von I holt, den Inhalt von C dazuaddiert und das Resultat nach C zurückstellt. Das sollte uns ein bißchen argwöhnisch machen, weil wir wissen, daß I anfangs nicht definiert war. Zeile 5 anzufügen, war vielleicht doch nicht so gut. Zunächst wollen wir aber unseren simulierten Programmablauf fortführen.

Zeile Nr.	S	C	N	I	Z	Verzweigung
5				Ø		
1Ø	Ø					
2Ø		Ø				
3Ø			2			
4Ø						×
5Ø		Ø				

Selbstverständlich erfüllt es keinen nützlichen Zweck, Null und Null zu addieren. Das sollte uns noch argwöhnischer machen. Wenn wir uns erinnern, daß wir Schreibfehler suchen, könnten wir auf den Verdacht kommen, I sollte gar nicht I sein, sondern 1, ein Fehler, der ziemlich oft vorkommt.
Wir fangen also jetzt mit einer neuen Tabelle ganz von vorne an und gehen von der neuen Vermutung aus, Zeile 5 sei überflüssig und Zeile 5Ø laute:

```
5Ø  LET C = C + 1
```

Zeile Nr.	S	C	N	Z	Verzweigung
1Ø	Ø				
2Ø		Ø			
3Ø			2		
4Ø					×
5Ø		1			
6Ø				2	
7Ø					√
3Ø			1		
4Ø					×
5Ø		2			
6Ø				1	
7Ø					√
3Ø			4		
4Ø					×
5Ø		3			
6Ø				4	
7Ø					√
3Ø			6		
4Ø					×
5Ø		4			
6Ø				6	
7Ø					√
3Ø			–1		
4Ø					√
1ØØ					

Wenn das Programm Zeile 1ØØ erreicht, zeigt es die Meldung "MITTEL IST", gefolgt vom Wert für S/C, der Null ist!

Das Programm funktioniert also immer noch nicht. Interessant ist jetzt aber, daß der ZX81 keine Fehlermeldung mehr liefert. Wir haben eine neue Art von Fehler eingeführt – einen Logikfehler. Der ZX81 kann die Arbeitsgänge zwar ausführen, die wir verlangt haben, aber wenn er das getan hat, liefert er eben die falsche Lösung.

Sehen wir uns die Tabelle an, die wir aufgestellt haben. Es ist jetzt ziemlich klar, welche Funktion C hat. Jedesmal, wenn in N ein neuer Wert eingegeben wird, vergrößert sich der Wert in C um einen Punkt, so daß C, sobald der Sprung zu Zeile 1ØØ erfolgt, die Zahl aller eingegebenen Werte enthält, in diesem Fall also 4. Aber mit S ist, seit es auf Null festgesetzt wurde, überhaupt nichts geschehen, und Z enthält lediglich dieselben Werte wie N, nur etwas später. Vielleicht sollten S und Z in Wahrheit derselbe Platz sein, und da S in Zeile 1Ø zuerst erwähnt wird, wollen wir annehmen, Z sei ein Schreibfehler für S. Das heißt, Zeile 6Ø muß lauten:

```
6Ø  LET S = S + N
```

und die Tabelle erhält dadurch folgendes Aussehen:

Zeile Nr.	S	C	N	Verzweigung
10	0			
20		0		
30			2	
40				x
50		1		
60	2			
70				√
30			1	
40				x
50		2		
60	3			
70				√
30			4	
40				x
50		3		
60				
70	7			√
30			6	
40				x
50		4		
60	13			
70				√
30			–1	
40				√
100				

Angezeigt wird uns jetzt:

MITTEL IST 3.25

was zutrifft!

Das Verfahren, das ich geschildert habe, nennt man einen Schreibtischtest oder Probelauf. Dabei handelt es sich um eine ganz übliche Art und Weise der Fehlersuche. Selbstverständlich ist es nicht immer notwendig, alle Einzelheiten, die ich dargestellt habe, in einer solchen Tabelle anzuführen (beispielsweise waren die Zeilennummern für uns in diesem Fall nicht sehr nützlich), und oft braucht man eine Tabelle auch gar nicht fertigzustellen, bevor einem ein Licht aufgeht, aber ich glaube, man kann schon erkennen, daß das eine gute Methode ist, Sie zum Zwangsjackendenken der Maschine zu zwingen und gleichzeitig deutlich zu machen, wie das Programm abläuft.

Sie mögen nun sagen: "Alles recht und schön, aber wer macht schon solche Schreibfehler, wenn er Programme eintippt?"

Die Antwort: Das kommt dauernd vor, und zwar aus verschiedenen Gründen. Wenn Sie ein Programm aus einer Zeitschrift abschreiben, besteht erstens schon einmal die Möglichkeit, daß der Fehler vorgedruckt ist. Zweitens kann

Ihnen bei einem langen Programm sehr leicht ein Tippfehler unterlaufen, also l statt 1, der Buchstabe O statt Null, 2 statt Z und so weiter. Drittens: Auch wenn Sie ein eigenes Programm schreiben, kann Ihnen ein Fehler unterlaufen, der einem Schreibfehler entspricht.

Ein Beispiel: Angenommen, Sie nennen am Beginn eines Programms eine Variable B3. Sie schreiben mehrere Tage an dem Programm, verbessern hier, modeln dort eine Zeile um. Am Ende müssen Sie noch ein paar Zeilen schreiben, in der diese Variable vorkommt, von der Sie ganz genau wissen, daß sie B2 hieß, und deshalb nicht mehr nachgucken.

Sie glauben nicht, daß das passieren kann? Warten Sie nur, bis Sie ein paar Monate lang programmiert haben.

GRAFIK

Der ZX81 kann zeichnen. Sobald er weiß, was er zeichnen soll, kann er die Zeichnung auch bewegen. Sobald er sie bewegen kann, ist der User in der Lage, die Bewegung durch das Keyboard zu steuern. Vor allem dann, wenn Sie Computer-Spielprogramme schreiben wollen, müssen Sie davon etwas verstehen.

Eine ganz besonders erfreuliche Eigenschaft der Computer ist die, daß sie Bilder zeichnen können – wenn die Hardware ausreicht, sogar wunderschöne und komplizierte Bilder in vielen Farben. Der ZX81 hat da zwar viel engere Grenzen, verfügt aber über genug Fähigkeiten, um eine anregende Einführung in die Computergrafik bieten zu können.

Der Abschnitt "GRAFISCH DARSTELLEN" in diesem Band schildert *eine* Möglichkeit, mit dem Computer zu zeichnen; da haben wir uns jedoch mit recht trockenen Dingen wie Rechtecken befaßt. Mit der PRINT-Einrichtung dagegen kann man Gebilde zeichnen, die dem Auge wohlgefälliger sind.

Der "PRINT"-Befehl

PRINT X$ ist hier der Grundbefehl, X$ ein Zeichen oder eine Folge von Zeichen. Das erklärt sich beinahe von selbst. Wenn Sie damit experimentieren, werden Sie aber bald dahinterkommen, daß der Befehl PRINT allein Ihnen wenig Gewalt darüber verleiht, *wo* ein gegebenes Zeichen angezeigt wird. Der Computer beginnt nämlich bei der Ausführung jedes Befehls dieser Art an der linken Bildschirmseite in der nächstverfügbaren Zeile, was nicht immer Ihren Absichten entsprechen wird.

PRINT AT X, Y regelt das. Ganz ähnlich wie bei PLOT muß man sich den Bildschirm in Quadrate aufgeteilt vorstellen, bezeichnet durch zwei Koordinaten X und Y. Es gibt aber mehrere Unterschiede. Erstens sind die Quadrate viermal so groß. Das bedeutet, daß es in jeder Richtung nur halb so viele davon gibt, also nur ein Viertel der Gesamtzahl. Außerdem ist die Zählweise ganz anders – und viel klarer. Die erste Zahl, also X, ist auf dem Bildschirm eine *Zeilen*nummer; sie verläuft von Ø ganz oben bis 21 ganz unten auf der Fläche, die für die Anzeige zur Verfügung steht. Die zweite, also Y, ist eine *Spalten*nummer; sie gibt die Entfernung entlang einer horizontalen Linie an und geht von Ø bis 31. Abbildung 11 zeigt dieses System genauer.

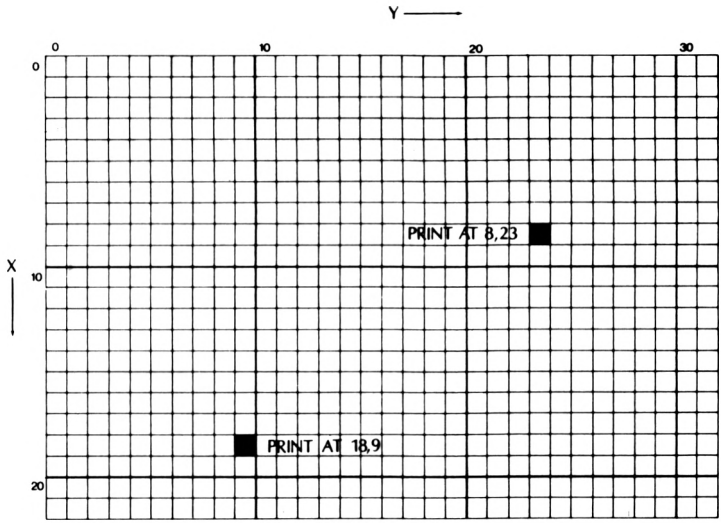


Abbildung 11

Angenommen, Sie wollen das Grafikzeichen ▩ in der Bildschirmmitte anzeigen Der exakte Mittelpunkt ist nicht zu erreichen, aber das Quadrat in Zeile 11 und Spalte 15 kommt ihm sehr nah. Sie würden also schreiben

```
1Ø  PRINT AT 11,15; "▩"
```

Beachten Sie den Strichpunkt (;). Er ist notwendig, um dem Computer mitzuteilen, er solle gleichzeitig ausführen, was er für *zwei* Befehle hält: Geh nach 11, 15; zeig etwas durch PRINT an.

Wenn Sie mehrere Grafikzeichen aneinanderfügen, können Sie schon interessantere Wirkungen erzielen. Der geradeste Weg dazu, nämlich, hintereinander immer wieder PRINT AT einzugeben, schluckt kostbaren Speicherplatz wie ein Straßenkreuzer Sprit; aber zunächst wollen wir uns über wirtschaftliche Nutzung noch keine Gedanken machen – es kommt nur auf das Prinzip an. Versuchen Sie es mit diesem Programm:

```
1Ø  PRINT AT 1Ø, 1Ø; "■■■▬▬▬▬"
2Ø  PRINT AT 11,8; "▩▩▩▩▩▩▩"
3Ø  PRINT AT 12,8; "▌0 0 0 0 0▐"
```

Das müßte Sie an etwas Militärisches erinnern. (Wenn nicht, dürfen Sie es auf meine mäßigen Zeichenkünste schieben . . .)

Wenn man solche Grafikbilder anzeigen lassen will, ist es oft nützlich, sie vorher auf kariertes Papier zu zeichnen und die Zeilen und Spalten zu numerieren; die erforderlichen Befehle sind dann leichter abzulesen. Das obige Programm wird auf eben diese Weise durch Abbildung 12 gewonnen.

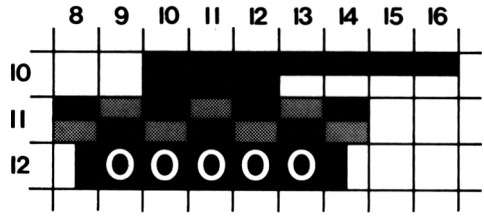


Abbildung 12

Der ZX81 verfügt im Zeichenvorrat über 21 besondere Grafikzeichen (Zahlen 1–1Ø und 128–138; (22, wenn Sie die Leertaste mitrechnen, Zeichen Nummer Ø) – siehe dazu Handbuch Seite 181). Sie können aber auch andere Zeichen sinnvoll nutzen (wie im obigen Beispiel). Zeichen in Negativschrift (weiß auf schwarz) sind in diesem Zusammenhang besonders geeignet.

Positionswechsel

Wenn man dahintergekommen ist, wie man auf einer ganz bestimmten Stelle auf dem Bildschirm anzeigen kann, fällt es nicht mehr schwer, das Programm so abzuwandeln, daß Sie diese Bilder genau dort anzeigen lassen können, wo Sie wollen. Der Panzer (ja, richtig, das sollte es sein) dort oben steht auf Zeile 12, mit dem linken Rand in Spalte 8. Wenn wir ihn so zeichnen wollen, daß er auf Zeile A steht, linker Rand in Spalte B, numerieren wir die Zeilen und Spalten auf dem Bild einfach, um das zu erreichen (vgl. Abbildung 13).

Daraus können wir das neue Programm ablesen:

```
1Ø  PRINT AT A-2,B+2; "■■■▄▄▄▄▄"
2Ø  PRINT AT A-1,B; "▓▓▓▓▓▓▓▓"
3Ø  PRINT AT A,B; "▌ Ø Ø Ø Ø Ø ▐"
```

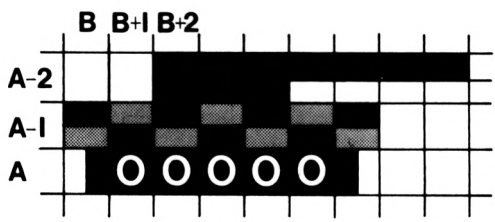


Abbildung 13

Selbstverständlich läuft das Programm dann nicht, wenn Sie den Wert von A und B nicht vorher eingeben. Es geht auch nicht, wenn der Computer auf Werte von A und B stößt, durch die das Bild am seitlichen Bildrand hinausrutscht. Wenn wir uns den linken Rand ansehen, heißt das, daß B den Wert von Ø oder größer haben muß; rechts muß B+9 31 oder kleiner sein, also B = 22 oder kleiner. Ebenso muß A 2 sein oder größer und 21 oder kleiner.

Dergleichen einmal ganz allgemein zu betrachten, hat den großen Vorteil, daß wir unsere Panzer dann, wenn wir bloß A und B genau bezeichnen, auf dem Bildschirm überall hinzeichnen können. Hier ein paar Beispiele: Setzen Sie sie in den Zeilen 1Ø–3Ø oben ein (die Zeilennummern werden vom ZX81 automatisch aufgerufen) und lassen Sie sie ausführen, um zu sehen, was dabei herauskommt.

```
a    1  LET B = 7
     2  FOR A = 2 TO 21
     4Ø NEXT A
b    1  LET A = 15
     2  FOR B = Ø TO 22
     4Ø NEXT B
c    1  LET A = 2+15*RND
     2  LET B = 2Ø*RND
     4Ø GOTO 1
```

(Geben Sie BREAK, um den Computer zu erlösen!)

Sobald Sie sich mit Unterprogrammen auskennen, werden Sie alle möglichen Verwendungsarten für diese Fähigkeit finden, ein beliebiges Bild an beliebig gewählter Stelle zu zeichnen. Der folgende Abschnitt zeigt, wie so etwas in einem häufig vorkommenden Fall genutzt werden kann.

Bewegliche Grafiken

Angenommen, Sie möchten, daß der Panzer von links nach rechts über den Bildschirm fährt. Programm b) schafft das beinahe schon, zieht aber dabei einen Rattenschwanz von Panzerhecks hinter sich her. Die können Sie allerdings loswerden, wenn Sie Zwischenräume übertippen, aber die eleganteste Lösung ist ein "unsichtbarer Rand", der das automatisch bewirkt. Verändern Sie die Zeilen 1Ø–3Ø wie folgt:

```
1  LET A = 15
2  FOR B = Ø TO 21
1Ø PRINT AT A-2,B+2; "□■■■▄▄▄▄"
2Ø PRINT AT A-1,B; "□■▒▒▒▒▒▒▒"
3Ø PRINT AT A,B; "□▌0 0 0 0 0 ▐"
4Ø NEXT B
```

Beachten Sie die Zwischenräume vor jeder Grafikzeile. (Um zu vermeiden, daß man über den Bildschirm hinausgelangt, lassen wir B in Zeile 2 nur bis 21 gehen).

Der Rand aus Leerstellen ist auf dem Bildschirm nicht sichtbar, aber die Art und Weise, wie der Computer Zeichen anzeigt, bedeutet, daß diese Leerstellen auf den unerfreulichen "Rattenschwanz" aufgesetzt werden und ihn dadurch löschen. (Nach einem alten Spruch verwischen Füchse ihre Spuren mit der Rute. Das tut unser Panzer auch, nur ist seine Rute unsichtbar).

Wenn wir den Panzer auch zurückfahren lassen wollen, benötigen wir am rechten Ende ebenfalls einen unsichtbaren Rand. Um den Tank nach oben zu bewegen, müssen wir unter ihm einen Rand haben, wenn er sich nach unten bewegen soll, einen Rand an der Oberseite. Damit er in alle Richtungen fahren kann (wobei A und B entsprechend verändert werden müssen), setzen wir den Rand rund um die gesamte Panzersilhouette, also brauchen wir in den Reihen A–3 und A+1 noch zwei Zeilen Leerstellen.

Aufgabe

Schreiben Sie ein Programm, damit der Panzer auf dem Bildschirm am Rand eines Quadrats von der Größe 1Ø mal 1Ø ständig entlangfährt. Setzen Sie ringsherum unsichtbare Ränder; finden Sie heraus, wie A und B sich verändern müssen; programmieren Sie das.

Bewegungssteuerung mit dem Keyboard

Da wir jetzt Gebilde zeichnen können, die so herumfahren, wie wir es wollen, läßt sich die Tastatur dazu benützen, die Bewegungen außerhalb des Programms zu steuern.

Die ungeschickte Art und Weise, das zu bewältigen, ist die, daß man das Programm als Schleife schreibt und per INPUT über das Keyboard eingibt. Das hält alles auf, die bewegte Bilddarstellung und alles übrige, bis die Tasten gedrückt werden.

Besser ist der INKEY$-Befehl. Eine Programmzeile wie

```
1Ø  LET C$ = INKEY$
```

weist den Computer an, nachzusehen, welche Taste gedrückt worden ist, und der Stringvariablen C$ das entsprechende Zeichen zuzuteilen. Wenn Sie mit diesem Zeichen jonglieren, können Sie alles Mögliche machen.

Ein Beispiel: Wir wollen den Panzer nach links und rechts bewegen und benützen Taste 5 für links und Taste 8 für rechts. (Das soll, wegen der Pfeile auf diesen Tasten, nur Ihrem Gedächtnis aufhelfen, aber es ist *nicht* erforderlich, SHIFT zu drücken und die Pfeile wirklich einzutippen). Was machen wir? Wir befehlen dem Computer, die INKEY$ zu lesen und die Anzeigeposition je nachdem zu ändern, ob 5 oder 8 gedrückt wird. Etwa so:

```
1   LET B = 15
2   LET A = 12
3   LET C$ = INKEY$
6   IF C$ = "5" THEN LET B = B-1
7   IF C$ = "8" THEN LET B = B+1
1Ø  PRINT AT A -2,B+2; "□■■■▬▬▬▬□"
2Ø  PRINT AT A -1,B; "□▓▓▓▓▓▓▓▓□"
3Ø  PRINT AT A,B; "□◧ØØØØØ◧□"
4Ø  GOTO 3
```

Beachten Sie den unsichtbaren Rand an beiden Seiten. Das Problem dabei ist, daß Sie seitlich zum Bildschirm hinauslaufen können, wenn Sie die Tasten zu lange drücken. Versuchen Sie das Programm so zu verändern, daß Sie das verhindern, in dem Sie Zeilen eingeben wie IF A<Ø THEN . . . mit dieser oder jener Folge. Solange Sie eine Taste gedrückt halten, läuft dieses Programm, und der Panzer fährt weiter. Manchmal kann das sehr störend sein; in Wirklichkeit wollen Sie oft, daß das Programm nur auf Veränderungen in den INKEY$ oder wenigstens auf wiederholtes Drücken reagiert.

In solchen Fällen ist Zeile 3 wie folgt abzuändern:

```
3   IF INKEY$ <> "" THEN GOTO 3
4   IF INKEY$ = "" THEN GOTO 4
5   LET C$ = INKEY$
. . .
```

Das hat die Wirkung, bei Zeile 3 alles zum Stillstand zu bringen, falls Sie immer noch auf der Taste herumdrücken. Sobald Sie loslassen, geht das Programm weiter zu Zeile 4 und bleibt dort. Sobald Sie eine neue Taste oder auch wieder die alte drücken – geht das Ding schon los!

WARNUNG: Achten Sie darauf, was Sie vom Computer beim Umgang mit diesem INKEY$ verlangen. Sie *wollen* vielleicht nur Tasten drücken, um Zahlen einzugeben und dann VAL INKEY$ berechnen, was diese Zahl aus einem Zeichen in etwas verwandelt, das Sie für arithmetische Berechnungen in der Tat verwenden können. Um das Programm aber zu starten, müssen Sie NEWLINE drücken – und manchmal liest der ZX81 *das* als INKEY$ und versucht es in eine Zahl zu verwandeln – worauf das Programm zusammenbricht. Oder noch schlimmer: Wenn überhaupt keine Tasten gedrückt werden, führt er VAL aus (den leeren String). Das kann ein bißchen rätselhaft sein, bis man begreift. Sie können sich dagegen schützen, indem Sie geeignete IF . . . THEN-Befehle verwenden, aber das kostet oft viel Speicherplatz und ist nicht immer empfehlenswert, jedenfalls nicht ohne den 16K RAM-Zusatzspeicher.

PRINT und PLOT kombiniert

Bei manchen Programmen müssen Sie vielleicht PRINT und PLOT gemeinsam verwenden, um grafische Bilder zu erhalten. Vor allem darf dabei nicht vergessen werden, daß PRINT AT X, Y und PLOT X, Y ganz verschiedene Folgen haben, weil die beiden X, Y sich auf zwei ganz verschiedene Koordinatensysteme auf dem Bildschirm beziehen. Das Handbuch von Sinclair bringt auf Seite 123 ein Diagramm, das beide zeigt, und auf Seite 12Ø findet sich eine mathematische Formel für den Wechsel von einem System zum anderen. Im allgemeinen ist der einfachste Weg dazu, die beiden Befehle richtig zu verbinden, der, auf kariertem Papier eine Rohskizze der Fläche anzufertigen, auf der man Grafik anzeigen möchte; zeichnen Sie beide Koordinationssysteme ein und sehen Sie auf der Skizze nach, wenn Sie Ihr Programm schreiben. Es ist oft lohnend, in Ruhe über den Aufbau des Programms nachzudenken, *bevor* man sich an das Keyboard setzt. Mißlingt der erste Versuch, fällt es dagegen in der Regel viel leichter, die Fehler auszumerzen, wenn man mit dem Computer experimentiert, statt sich das Gehirn zu zermartern. Es kommt eben darauf an, Zeit und Mühe möglichst sinnvoll einzusetzen.

Pause

Der ZX81 verfügt über einen PAUSE-Befehl, der den Computer veranlaßt, eine bestimmte Zeit zu warten. Das scheint für bewegliche Grafik sehr nützlich zu sein, etwa dann, wenn etwas verlangsamt werden soll, was sonst zu schnell ginge.

Im vorliegenden Fall ist das allerdings aussichtslos, weil der *Bildschirm bei PAUSE flackert*, was gräßlich aussieht. Probieren Sie einmal das "Uhren"-Programm auf Seite 127 des Handbuchs aus. Sie geraten bestimmt nie mehr in Versuchung, noch einmal PAUSE zu verwenden.

Statt PAUSE können Sie eine Schleife benützen, die nichts anderes tut, als Zeit zu verbrauchen:

```
3ØØ  FOR J = 1 TO 25
31Ø  NEXT J
```

und die 25 verändern, wenn Sie kürzere oder längere Schleifen haben wollen.

VERZWEIGUNGEN

Manchmal hängt das, was der Computer für Sie tun soll, davon ab, was bis dahin geschehen ist. In diesem Fall wenden Sie eine Methode an, die mit Verzweigungen zu tun hat.

Dieser Abschnitt befaßt sich mit Logik und *bedingten* Befehlen, wobei das, was der Computer zu tun hat, von verschiedenen anderen Dingen abhängt. Beispiel Kinobesuch: IF (wenn) Sie unter 16 sind, THEN (dann) dürfen Sie nicht in einen Film, der für Jugendliche nicht freigegeben ist. Der Arbeit des Computers können Sie ähnliche Bedingungen auferlegen.

Der Befehl, mit dem Sie das bewirken, heißt IF . . . THEN . . . – aber entscheidend ist, was an die Stelle der drei Pünktchen tritt.

Schreiben wir ein Programm, das feststellen soll, ob eine gegebene Zahl ungerade oder gerade ist. ("Ist ja toll!" höre ich Sie schon rufen, und Sie haben ja recht, aber wie an anderen Stellen hier geht es um das Prinzip und nicht um das konkrete Ergebnis).

Also los.

```
1Ø  INPUT N
2Ø  IF N = 2*INT(N/2) THEN PRINT "GERADE"
3Ø  IF N <> 2*INT(N/2) THEN PRINT "UNGERADE"
```

Wie geht das? Gerade Zahlen sind solche, die durch 2 ohne Rest teilbar sind. N ist also gerade genau dann, wenn N/2 eine ganze Zahl ist. Das heißt, bei INT bleibt sie gleich, demnach N/2 = INT (N/2). Und *das* ist dasselbe wie N = 2*INT(N/2). Zeile 2Ø sagt deshalb verklausuliert nichts anderes als

```
2Ø  IF N ist gerade THEN PRINT "GERADE"
```

was für *uns* ja einigermaßen Sinn ergibt – aber nicht für den armen ZX81, der keine Ahnung hat, was "ist gerade" bedeutet, bis wir es ihm in der Sprache sagen, die er versteht.

Nur, um das ganz klarzumachen, wollen wir ein paar Fälle darstellen.

N	22	23	24	25	26
N/2	11	11.5	12	12.5	13
INT(N/2)	11	11	12	12	13
2*INT(N/2)	22	22	24	24	26

Ebenso wird N durch K genau halbiert – bei ganzen Zahlen K, N – nur und nur dann, wenn N = K*INT(N/K) . . . Das ist eine sehr nützliche Erkenntnis.

Schließlich muß vielleicht nur noch erwähnt werden, daß in Zeile 3Ø das Symbol <> bedeutet "ist nicht gleich". Viele Mathematiker würden ≠ verwenden, aber der ZX81 bevorzugt <>. Fragen Sie mich nicht nach dem Grund.

Im Prinzip machen Sie es genauso. Der entscheidende Befehl hat die Form

```
1Ø  IF dieses THEN jenes
```

wobei "dieses" und "jenes" Anweisungen sind. In Zeile 2Ø oben ist "dieses" N = 2*INT(N/2) und "jenes" PRINT "GERADE".

Die Anweisung "dieses" muß so abgefaßt sein, daß der Computer sie als entweder richtig oder falsch erkennen kann. (IF STOP THEN . . . sagt nicht gerade viel, aber IF N = 1981 THEN . . . tut das). Wenn "dieses" richtig ist, fährt der Computer fort, "jenes" zu tun; wenn "dieses" falsch ist, *geht er weiter zur nächsten Programmzeile, ohne "jenes" zu tun.*

Nun folgen zwei besonders gebräuchliche Arten von bedingten Befehlen.

Bedingte Verzweigungen

Sie haben die Form

```
1Ø  IF dieses THEN GOTO n
```

wobei n eine Zeilennummer ist. Solche Befehle können dazu verwendet werden, gewissermaßen mitten im Strom die Pferde zu wechseln – das heißt, den ganzen Verlauf der Rechenoperation dadurch zu verändern, daß man zu einem anderen Teil des Programms geht.

Berechnet man etwa die Quadratwurzel einer Zahl, dann ist es wichtig, sich daran zu erinnern, daß negative Zahlen keine Wurzeln haben. Sie vermeiden Fehlermeldungen also dadurch, daß Sie es etwa auf diese Art machen:

```
1Ø  INPUT N
2Ø  IF N<Ø THEN GOTO 5Ø
3Ø  PRINT N, SQR N
4Ø  STOP
5Ø  PRINT "QUADRATWURZEL NICHT DEFINIERT"
```

Beachten Sie das STOP in Zeile 4Ø. Warum steht es da? Nehmen Sie es heraus und sehen Sie sich an, was dann geschieht!

Wenn Sie PRINT AT A, B verwenden, können Sie sich auf gleiche Weise gegen A und B in nicht PRINTbaren Bereichen schützen, indem Sie so vorgehen:

```
1ØØ  IF A < Ø THEN GOTO 1ØØØ
11Ø  IF A > 21 THEN GOTO 1ØØØ
12Ø  IF B < Ø THEN GOTO 1ØØØ
13Ø  IF B > 31 THEN GOTO 1ØØØ
15Ø  PRINT AT A,B; "*"
16Ø  STOP
1ØØØ  was Sie da eben für nützlich halten . . . etc.
```

Wir gehen hier davon aus, daß A, B in einem vorherigen Programmteil bestimmt werden.

Fügen Sie davor ein

```
10 INPUT A
20 INPUT B
```

und befehlen Sie dann dem Computer PRINT AT 999. −37, wobei Sie eingeben A = 999, B = −37.

Zeile 1000 soll lauten

```
1000 PRINT "ICH BIN NICHT MEHR SO DUMM"
```

Ist er's?

Wenn Sie das Gefühl haben, die Zeilen 100 – 130 sähen umständlich aus . . . dann liegen Sie richtig. Vergleichen Sie unten den Abschnitt über LOGIK . . .

Bedingte Zuweisungen

Diese haben die Form IF dieses THEN LET irgend etwas anderes. Eine Alternative zum ersten Programm oben wäre dann

```
10 INPUT N
20 LET A$ = "UNGERADE"
30 IF N = 2*INT (N/2) THEN LET A$ = "GERADE"
40 PRINT A$
```

A$ hat hier ein Dollarzeichen, weil es keine Zahl ist, sondern eine Zeichen*kette* (siehe STRINGS, Seite 91).

Was bedeutet dieses Programm?

```
10 LET S = INT(2*RND)
20 IF S = 0 THEN LET A$ = "KOPF"
30 IF S = 1 THEN LET A$ = "WAPPEN"
40 PRINT A$
```

Sobald Sie ein Programm schreiben und das, was Sie vorhaben, davon abhängt, daß bestimmte Dinge geschehen oder nicht geschehen, denken Sie an IF . . . THEN . . .

Logik

Ein großes Thema, in gelehrten Abhandlungen endlos durchgekaut – aber das Meiste davon brauchen wir nicht zu wissen. Der ZX81 beherrscht Logik – vermutlich besser als Sie und ich. Genauer gesagt, er kann Anweisungen, die in der Position "dieses" von IF dieses THEN jenes vorkommen, durch die Verwendung von AND, OR und NOT kombinieren.

Grundregeln: p AND q ist nur wahr, wenn p wahr ist und q wahr ist.
p OR q ist wahr, wenn p wahr ist oder q wahr ist oder *beide* wahr sind.
NOT p ist nur wahr, wenn p falsch ist.

Der ZX81 führt NOT-Befehle vor AND-Befehlen und AND-Befehle vor OR-Befehlen aus. Die Folge: Sie brauchen oft keine Klammern, um den Befehl klarzumachen.

Beispielsweise können wir die Zeilen 1ØØ–13Ø oben dadurch verbessern, daß wir sie in die Einzelzeile

```
1ØØ  IF A < Ø OR A> 21 OR B < Ø OR B > 31 THEN GOTO 1ØØØ
```

verwandeln.

Die Logik des ZX81 kann man in vielerlei Beziehung großartig nutzen, aber die Erklärung ist kompliziert, und ich habe nicht viel Platz, so daß ich hier, wenn auch ungern, aufhören muß. Kapitel 1Ø des Sinclair-Handbuchs führt Sie auf den Weg, aber die ganze Geschichte ist das bei weitem noch nicht.

DEBUGGING III

. . . aber warum den ZX81 nicht dazu benützen, daß er sich selbst "entfehlert"?

Im letzten Beispiel, das wir uns angesehen haben, wurde das Debugging durch ein beträchtliches Ausmaß an eigener Anstrengung bewirkt. Es wäre schön, den Computer, der bis jetzt nur ruhig dagesessen und uns ungeniert die falschen Antworten gegeben hat, dazu veranlassen zu können, daß er uns einen Teil der Arbeit abnimmt.

Die Frage lautet also: Was für Auskünfte kann uns der Computer sinnvollerweise darüber geben, wie er ein Programm ausführt? Man muß drei Hauptbereiche berücksichtigen:

1 Welche Werte erhält das Programm auf verschiedenen Stufen seiner Ausführung für die einzelnen Variablen?
2 Wohin geht das Programm (das heißt, welche Zeilen werden ausgeführt und in welcher Reihenfolge)?
3 Wie oft geht es dorthin (das heißt, wie oft werden bestimmte Zeilen oder Zeilengruppen ausgeführt)?

Manche Geräte bieten automatische Funktionen für die Anzeige zumindest eines Teils dieser Informationen, aber für DM 200,- kann man nicht alles haben (jedenfalls bis jetzt noch nicht), also müssen wir ein paar zusätzliche Anweisungen in unsere Programme einbauen, um einige dieser Einzelheiten beizusteuern. Sehen wir uns die oben erwähnten Themen der Reihe nach an.

Werte von Variablen anzeigen

Es ist sehr leicht, Werte von Variablen überall dort anzuzeigen, wo wir sie haben wollen. Wir brauchen an der richtigen Stelle im Programm nur eine PRINT-Anweisung einzufügen. So könnten wir etwa in unser Programm zur Berechnung des Mittelwerts eine Zeile 55 einsetzen:

```
55 PRINT C
```

Dadurch könnten wir die Veränderungen im Wert von C während des Programmablaufs verfolgen. Es wäre nicht einmal schwierig, vom Computer eine Kopie der Probelauf-Tabelle zu erhalten, die wir von Hand angefertigt haben – vielleicht versuchen Sie es einmal.

Das eigentliche Problem hier ist, die Wahl zu treffen, wo Zwischenwerte sparsam und überlegt ausgegeben werden sollen, weil Sie sonst Zahlenmengen geliefert bekommen, die zu analysieren genauso lange dauert wie ein Probelauf von Hand.

Beim zweiten Lauf des Mittelwert-Programms entstand, wie Sie sich erinnern, in Zeile 1Ø0 eine Fehlermeldung, so daß uns überhaupt keine Werte angezeigt wurden. Ein sinnvoller erster Schritt bestünde deshalb darin, eine Zeile 95 einzufügen:

```
95 PRINT S, C
```

Vergessen Sie nicht, daß Zeile 4Ø abgeändert werden muß:

```
4Ø  IF N < Ø THEN GOTO 95
```

weil der neue Befehl sonst nie ausgeführt wird. Das wird den Erfolg haben, unseren Verdacht zu bestätigen, C enthalte Null, aber wenig mehr.

Jetzt könnten wir Zeile 65 einschieben

```
65  PRINT C; I; N; S; Z
```

so daß wir am Ende jeder Schleife eine vollständige Liste aller Variablen erhalten (praktischerweise in alphabetischer Reihenfolge). Das ergibt eine vereinfachte Ausgabe der Probelauf-Tabelle, die trotzdem die wesentlichen Punkte zeigen wird.

Übrigens gibt es hier einen hübschen Trick: Beim Programmtesten werden Sie vielleicht eine der neuen Zeilen vorübergehend entfernen wollen, um zu verhindern, daß zu viele Werte auf einmal angezeigt werden. Das bedeutet, daß man später die ganze Zeile wieder eingeben muß, oder, noch schlimmer, daß, wie bei unserer Zeile 95, auch noch eine andere Zeile redigiert werden muß, weil Zeile 4Ø sich zurückverwandelt haben wird in

```
4Ø  IF N < Ø THEN GOTO 1ØØ
```

wenn man Zeile 95 herausnimmt. Das ist nicht nötig. Setzen Sie am Anfang jeder Zeile, die Sie unwirksam machen wollen, einfach ein REM voran. Zeile 95 wird zu:

```
95  REM PRINT S, C
```

Da das jetzt eine Bemerkung ist, beachtet der Computer sie nicht, aber ein Sprung zu Zeile 95 ist durchaus zulässig! Wenn wir die Anweisung zurückhaben wollen, löschen wir lediglich das REM. Wenn Sie nur einen 1K-Speicher haben, mögen zusätzliche REMs freilich ein Luxus sein, den Sie sich nicht leisten können. Die Moral dieser Geschichte: Kaufen Sie den 16K-Zusatzspeicher!

Den Ablauf verfolgen

Die einfachste Methode, den Weg zu verfolgen, den ein Programm nimmt, ist die, jeder Zeile eine PRINT-Anweisung folgen zu lassen, worauf die eben ausgeführte Zeilennummer vollständig angezeigt wird. Das Mittelwert-Programm zum Beispiel würde dann so aussehen:

```
1Ø  LET S=Ø
11  PRINT"1Ø"
2Ø  LET C=Ø
21  PRINT"2Ø"
3Ø  INPUT N
31  PRINT"3Ø"
etc.
```

Wieder laufen wir Gefahr, zuviel Information zu erzeugen und den Wald vor lauter Bäumen nicht mehr zu sehen; wir wollen deshalb bei dieser "Verfolgungs"-Prozedur etwas wählerischer vorgehen. Eine Frage der Art, die durch Ablaufverfolgung klar beantwortet wird, ist: "Macht das Programm einen Sprung dann, wann es das tun soll?" Wenn das der Fall ist, beschränken wir uns mit unserer Verfolgung sinnvollerweise auf Bereiche, wo Sprünge stattfinden.

Ein Beispiel: Nehmen wir an, bei einem Programm gehe es darum, einen Monatstag einzugeben. Ein solcher Wert muß im Bereich zwischen 1–31 liegen, also entspräche es guter Programmierungsübung, dafür zu sorgen, daß der User einen solchen Wert eingegeben hat, bevor es weitergeht. Wir könnten einen Programmteil schreiben, der so aussieht:

```
5Ø  INPUT D
6Ø  IF D > Ø OR D < 32 THEN GOTO 2ØØ
7Ø  HIER REM WENN UNZULAESSIGER TAG
 .
 .
 .
2ØØ  HIER REM FUER ZULAESSIGEN TAG
 .
 .
 .
```

Das Programm verhält sich nicht so, wie es das tun sollte, also geben wir nach den Zeilen 5Ø, 7Ø und 2ØØ eine Verfolgungsanweisung ein:

```
5Ø  INPUT D
51  PRINT "*5Ø*"
6Ø  IF D < Ø OR D > 32 THEN GOTO 2ØØ
7Ø  HIER REM WENN UNZULAESSIGER TAG
71  PRINT "*7Ø*"
 .
 .
 .
2ØØ  HIER REM FUER ZULAESSIGEN TAG
2Ø1  PRINT "*2ØØ*";
 .
 .
 .
```

Wir stellen fest, daß, gleichgültig, welchen Wert wir eingeben, beim Protokoll, wie man das auch nennt, jedesmal herauskommt:

```
*5Ø* *2ØØ*
```

(Ich verwende Sternchen, damit Protokoll-Zeilennummern nicht mit den vom Programm angezeigten Zahlen verwechselt werden können. Verwendbar ist jedes Sonderzeichen, das Ihnen gefällt.)

Das Programm kann also nicht dazu gebracht werden, bis zur Zeile 7Ø zu gehen. Es gibt nur eine vernünftige Schlußfolgerung: Die Bedingung D > Ø OR D < 32 wird jedesmal erfüllt. Das entspricht nur der Logik – jede dieser Zahlen ist entweder größer als Null oder kleiner als 32. Wir hätten statt dessen sagen sollen:

```
6Ø  IF D > Ø AND D < 32 THEN GOTO 2ØØ
```

Der Fehler, AND mit OR zu verwechseln, kommt wegen der schlampigen Aussprache dieser Worte im Alltagsgebrauch häufig vor (im Deutschen in diesem speziellen Fall natürlich nicht, aber bei uns sind wieder andere Verwechslungen möglich. Anm. des Übersetzers.) In diesem Fall müssen beide Bedingungen erfüllt sein, bevor wir einen gültigen Tag haben, so daß die AND-Verknüpfung verlangt wird.

Programmquerschnitte

Ein Programmquerschnitt zeigt, wie oft jede Programmzeile ausgeführt worden ist. Wie gewohnt, ist das des Guten viel zu viel. Wir sollten uns deshalb aussuchen, welche Programmteile wir im Querschnitt darstellen wollen. Das geht ganz leicht. Angenommen, wir wollen herausfinden, wie oft Zeile 42Ø eines bestimmten Programms ausgeführt wird. Wir setzen zu Beginn des Programms einen Zähler auf Null und erhöhen jedesmal, wenn Zeile 42Ø ausgeführt wird, um 1:

```
  5  PC = Ø
  .
  .
42Ø  A = A*(P-1)
421  LET PC = PC + 1
  .
  .
8Ø9  PRINT PC
81Ø  STOP
```

Sehen wir uns ein konkretes Beispiel an. Das folgende Programm soll eine Höchstzahl von 2Ø Werten, abgeschlossen durch Null, aufnehmen, und sie in aufsteigender Reihenfolge ordnen. Wenn die Eingabefolge also lautet:

3
8
1
4
2
Ø

sollte am Ende angezeigt werden

1
2
3
4
8

Die Null sollte nicht erscheinen, weil sie nur ein Begrenzer ist.

```
1Ø  DIM A (2Ø)
2Ø  FOR P = 1 TO 2Ø
3Ø  INPUT A(P)
4Ø  IF A(P) = Ø THEN GOTO 6Ø
5Ø  NEXT P
6Ø  LET N = P
65  LET F = Ø
7Ø  FOR P = 1 TO N
8Ø  IF A(P) < A (P+1) THEN GOTO 13Ø
9Ø  LET T = A(P)
1ØØ  LET A(P) = A(P+1)
11Ø  LET A(P+1) = T
12Ø  LET F = 1
13Ø  NEXT P
14Ø  IF F = 1 THEN GOTO 65
```

```
150  FOR P = 1 TO N
160  PRINT A(P)
170  NEXT P
```

Das Programm leistet nicht ganz, was es soll. (Geben Sie es ein und probieren Sie es aus.) Vielmehr gerät es in eine Endlos-Schleife.

Wo also mit dem Suchen anfangen? Die erste Schleife (20–50) sieht ganz harmlos aus, und die letzte (150–170) zeigt nur den Inhalt des Arrays an. Es liegt also nahe, sich mit der Schleife von 70 bis 130 zu befassen. Aus Zeile 80 ergibt sich, daß manchmal alle Anweisungen in der Schleife ausgeführt werden und manchmal die von 90 bis 120 unbeachtet bleiben. Wir nehmen also zwei Querschnittzählungen vor, C1 und C2; dabei wird gezählt, wie oft die Schleife begonnen und wie oft der letzte Teil der Schleife ausgeführt wird.

Das können wir erreichen mit:

```
 67  LET C1 = 0
 68  LET C2 = 0
 75  LET C1 = C1 + 1
125  LET C2 = C2+1
132  PRINT C1,C2
```

Wenn wir schon dabei sind, können wir auch gleich den Inhalt des Arrays am Ende der Schleife anzeigen, weil deutlich ist, daß darin Zahlen umgeschaufelt und nur sehr wenige andere Variable überhaupt verwendet werden. Also:

```
134  FOR Q = 1 TO N  (weil 1 TO N der relevante Teil des Arrays zu sein
                      scheint)
135  PRINT A(Q);
136  NEXT Q
137  PRINT
```

Versuchen wir es mit ein paar Dateneingaben, um zu sehen, was geschieht: wenn wir eingeben

```
3
6
1
8
5
0
```

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 der Speicher voll ist.

Tja, die Werte scheinen zwar in die richtige Reihenfolge gebracht zu werden, aber die "8" haben wir verloren, und woher stammt auf einmal das Nullenpaar? Außerdem läuft das Programm beharrlich sechsmal durch die Hauptschleife, aber die Zahl der Durchgänge in der Unterschleife verringert sich ständig, bis sie 1 erreicht und dabei bleibt.

Eine der Nullen ist offenkundig der Begrenzer, die andere ein Element des Arrays, das nicht während des Ablaufs gesetzt, sondern durch das System als Null etikettiert wird. Mit anderen Worten: Das Programm behandelt zwei Werte zuviel. Schreiben wir Zeile 6Ø also um:

```
6Ø  LET 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 Fortschritt – die Nullen sind wir losgeworden. Aber unsere "8" haben wir immer noch nicht zurückbekommen.

Schwer zu erkennen, wo sie verlorengegangen sein soll. Vielleicht ist sie noch da, wird aber nicht angezeigt. Wo zeigen wir sie an? Mit den Zeilen 15Ø–17Ø. Der Bereich 1 TO N muß zu klein sein. Vergrößern wir ihn um 1:

```
15Ø  FOR P = 1 TO N + 1
```

Und weil wir schon dabei sind: Der Überwacher in Zeile 134 leidet vermutlich unter denselben Problemen. Beheben wir das also auch gleich:

```
134  FOR Q = 1 TO N + 1
```

Ahem – von neuem in die Bresche, Freunde, noch einmal . . . Diesmal erhalten wir (für dieselben Daten):

```
4        2
31658
4        2
13568
4        Ø
13568
1
3
5
6
8
```

Prima! Wir haben es geschafft. Jetzt läuft es richtig. Wirklich? Versuchen wir es mit:

```
3
5
2
1
5
Ø
```

Diesmal kommt:

```
4        3
32155
4        3
21355
4        2
12355
4        1
12355
4        1
12355
4        1
12355
.
.
.
etc.
```

Die richtige Antwort bekommt das Programm heraus, verläßt aber die Schleife nicht. Wir bemerken, daß C2 in diesem Fall nie bis Null kommt, also kann man darauf wetten, daß das Programm dadurch beendet wird.

Was entscheidet darüber, ob das Programm in die Unterschleife eintritt oder nicht? Zeile 8Ø:

```
8Ø  IF A(P) < A(P+1) THEN GOTO 13Ø
```

Der Unterschied zwischen den beiden Datenmengen ist der, daß die zweite zwei identische Werte enthält. Da 5 nicht weniger als 5 ist, wird die Unterschleife jedesmal dann ausgeführt, sobald der Computer auf die doppelte 5 stößt. Deshalb läuft das Programm stets einmal durch die Unterschleife. Vielleicht sollte die Frage lauten?

```
8Ø  IF A(P) < = A(P+1) THEN GOTO 13Ø
```

Diesmal funktioniert alles.

```
4     2
32155
4     2
```

```
21355
4    1
12355
4    Ø
12355
1
2
3
5
5
```

Jetzt geht es wunderbar, und wir können die Testzeilen herausnehmen.

Ich hoffe, ich habe hier ein paar wichtige Punkte deutlich gemacht. Erstens mußten wir nicht genau wissen, wie das Verfahren abläuft. Wenn Sie diesen Abschnitt sorgfältig durchgearbeitet haben, ist das inzwischen aber wohl ziemlich klar, und ein paar Probeläufe würden Sie vermutlich davon überzeugen, daß Sie es begriffen haben. (Probeläufe sind überhaupt eine gute Methode, Computerabläufe zu verstehen. Ich habe oft ein Dutzend davon oder noch mehr bei zweifelhaften Programmteilen – von anderen Leuten, versteht sich – absolviert, bevor mir wirklich klar vor Augen stand, was vorging.) Zweitens bildet man sich, wenn ein Programm beim erstenmal erfolgreich läuft, stets gleich ein, die Arbeit sei getan und man könne sich nun in der Eckkneipe ein Glas vergönnen. Wie wir gesehen haben, ist die Arbeit *nicht* getan, weil es vielleicht andere Datenmengen gibt, bei denen das Programm scheitert. Im übrigen hat die Kneipe schon geschlossen, falls Ihnen die Zeit beim Schreiben von Programmen genauso schnell vergeht wie mir.

SUBROUTINEN (UNTERPROGRAMME)

Wenn Sie eine Aufgabe in eine Reihe von unterscheidbaren Unteraufgaben zerlegen können, wird eine mächtige Waffe im Arsenal des Benutzers zugänglich:

Es gibt eine Geschichte über Mathematiker, die ebensogut für Computerprogrammierer gelten könnte (ja, da *gibt* es Unterschiede!). Wie bringt man Teewasser zum Kochen? Der Mathematiker wird zunächst aufgefordert, die Schritte zu beschreiben, die zu einer Tasse Tee führen, wenn der Teekessel in der Küche an einem Haken hängt. Darauf sagt er etwa: "Nimm den Kessel vom Haken, füll ihn mit Wasser, stell ihn auf den Herd, zünd ein Streichholz an . . ." und so weiter. Dann wird er gebeten, die Schritte zu beschreiben, die nötig sind, um eine Tasse Tee zuzubereiten, wenn der Kessel auf dem Küchentisch steht – und er antwortet: "Häng den Kessel an den Haken und fahr fort wie zuvor."

Er betrachtet die erste Folge von Tätigkeiten demnach als ein *Unterprogramm*. Im zweiten Fall verändert er die bestehende Situation, damit das Unterprogramm anwendbar wird, und setzt es dann so ein, als wäre es ein einziger unteilbarer Vorgang.

In der Computer-Fachsprache ist eine Subroutine ein Programmteil, der eigens geschrieben und wiederholt als Teil eines größeren Programms verwendet werden kann. Subroutinen stellen eine sehr zivilisierte Weise dar, Programme zu schreiben, weil sie in der Regel die Beobachtung der Vorgänge erleichtern. Außerdem kann man ein mit Subroutinen verfaßtes Programm leichter von Fehlern befreien, weil man jedes für sich von Fehlern reinigen kann und dann nur noch zu überprüfen braucht, ob sie richtig miteinander verbunden sind.

Der hier relevante Befehl heißt GOSUB. Das ist ähnlich wie GOTO, aber weitaus vielseitiger. Im typischen Fall erscheint er auf folgende Weise:

```
1ØØ  GOSUB 5ØØ
11Ø  allerlei anderes Zeug
...
5ØØ  etwas tun
...
57Ø  RETURN
```

wobei, wie gewohnt, die Pünktchen für andere Programmteile stehen. Damit wird bewirkt:

a Erreicht das Programm Zeile 1ØØ, springt es zu Zeile 5ØØ und *merkt sich, wo es hergekommen ist.*

b Es führt Zeile 5ØØ und alles, was ihr folgen mag, aus, bis es zu Zeile 57Ø kommt und den RETURN-Befehl erhält.

c Dann geht es zurück zur Ursprungszeile – hier 1ØØ – und fährt mit dem *nächsten* Befehl nach dieser Zeile fort.

Im Prinzip ist das wie GOTO, nur erfolgt die Rückkehr zum Ausgangspunkt automatisch. Das Wichtige dabei ist aber: Man kann aus *verschiedenen* Zeilen desselben Programms in dieselbe Subroutine eintreten; der Computer behält im Gedächtnis, bei welcher er angefangen hat, und kehrt nach RETURN zu dieser Zeile zurück.

Als Beispiel will ich Schritt für Schritt erklären, wie man ein Programm schreibt, das

d die Tasten 5, 6, 7, 8 dazu benützt, einen [P]-Cursor auf dem Bildschirm zu bewegen – und

e) anstelle des Cursors jede Zeicheneingabe über das Keyboard anzeigt.

Für diese Zeicheneingabe müssen wir INKEY$ benützen. Wir lesen die Taste, die das gerade verwendet, und teilen es einer Stringvariablen A$ zu. Wir wollen, daß das Programm nur auf neu gedrückte Tasten reagiert, wie auf Seite 67 beschrieben, also brauchen wir einen Programmteil in folgender Form:

```
1ØØØ  IF INKEY$ <> "" THEN GOTO 1ØØØ
1Ø1Ø  IF INKEY = "" THEN GOTO 1Ø1Ø      wird zum Unterprogramm
1Ø2Ø  LET A$ = INKEY$
```

Die 1ØØØer werden verwendet, weil das eine Subroutine wird und wir sie weit wegstellen wollen (allerdings kann man oft ein paar Programmzeilen sparen, wenn man alle Subroutinen *voran*setzt und den Lauf des Programms nicht mit RUN, sondern mit GOTO beginnt – aber das ist eine Verfeinerung, auf die sich einzulassen hier nicht lohnt). Um wieder herauszugehen, brauchen wir die zusätzliche Zeile

```
1Ø3Ø  RETURN
```

Weiter: Was müssen wir tun, um den [P]-Cursor zu bewegen? Auf jeden Fall müssen wir wissen, *wo* wir ihn anzeigen wollen, also erfinden wir zwei Variable A und B; sie liefern die Werte für die Zeilen und Spalten, wo er angezeigt werden soll. Damit uns nicht alles zusammenbricht, bevor wir richtig angefangen haben, müssen wir ihnen Werte zuweisen. Die Mitte des Bildschirms ist ein guter Ausgangspunkt:

```
1Ø  LET A = 1Ø
2Ø  LET B = 15
```

Nun wollen wir die Tasten 5, 6, 7, 8 dazu benützen, A und B zu verändern und damit den Cursor zu bewegen. Die brave Subroutine oben wird die Tastatur lesen, also ist das Nächstliegende wohl

```
30  GOSUB 1ØØØ
```

Danach wird A$ uns sagen, ob Taste 5, 6, 7 oder 8 gedrückt worden ist. Wir wollen das P in Richtung der vier Pfeile auf diesen Tasten bewegen. (Deshalb verwenden wir sie; die Pfeile sind gute Erinnerungsstützen!)

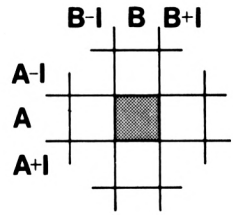


Abbildung 14

Sehen Sie sich Abbildung 14 an. Sie zeigt Position A, B und die vier Nachbarn. Wenn wir uns an die Hinweise auf dem Bild halten, sehen wir, was wir haben wollen:

Taste 5 soll B zu B − 1 verändern und A unverändert lassen
Taste 6 soll A zu A + 1 verändern und B unverändert lassen
Taste 7 soll A zu A − 1 verändern und B unverändert lassen
Taste 8 soll B zu B + 1 verändern und A unverändert lassen.

Hier *ein* Weg dazu:

```
4Ø  IF A$ = "5" THEN LET B = B − 1
5Ø  IF A$ = "6" THEN LET A = A + 1
6Ø  IF A$ = "7" THEN LET A = A − 1
7Ø  IF A$ = "8" THEN LET B = B + 1
```

Nachdem wir den Cursor bewegt haben, möchten wir sehen, wo er hingeraten ist:

```
8Ø  PRINT AT A, B; "P"
```

Was nun? Wir wollen ein Zeichen eingeben, das anstatt dieses P angezeigt werden soll. Wieder eine Subroutine!

```
9Ø  GOSUB 1ØØØ
```

Diesmal liest der Computer die Tastatur, teilt A$ den Wert zu, den er dort vorfindet (immer das, was wir drücken, A, B, C, D . . ., 7, 3, >, . . .) und kehrt zur *nächsten Zeile nach Zeile* 9Ø *zurück*. Und um dem Computer zu sagen, daß er das gefundene Zeichen anzeigen soll, brauchen wir:

```
1ØØ  PRINT AT A, B; A$
```

Fast schon fertig. Bis jetzt läuft das Ganze nur einmal. Wir wollen zum Anfang zurückkehren und von vorne beginnen – aber das neue A, B behalten und 1Ø und 15 *nicht* umstellen. Ganz einfach:

```
11Ø  GOTO 3Ø
```

(3Ø schickt das Programm dann sofort wieder zu 1ØØØ. Warum kann die Zeile 11Ø GOSUB 1ØØØ nichts nützen?)

Geben Sie alle obengenannten Zeilen ein (wir haben dafür gesorgt, daß die Zahlen in der richtigen Reihenfolge kommen, was nicht immer der Fall ist, wenn man ein Programm erstmals schreibt; lassen Sie sich also nicht irreführen!) und drücken Sie RUN. Tun wird sich nichts, aber wenn Sie 5, 6, 7 oder 8 drücken, werden Sie den [P]-Cursor an der neuen Position sehen. Sobald Sie irgend etwas anderes drücken, erscheint er am alten A, B – ein zusätzlicher Gewinn, den wir eigentlich gar nicht geplant hatten. Drücken Sie dann eine Taste, sagen wir T. Das [P] verschwindet und wird durch ein T ersetzt. Bewegen Sie den Cursor – der jetzt unsichtbar ist – mit 5, 6, 7, 8, zeigen Sie ihr nächstes Zeichen an und machen Sie weiter. Sie können Wörter auf den ganzen Bildschirm schreiben. (Wie wäre es mit einem Programm für computerunterstützte Kreuzworträtsel-Konstruktion?)

Um zu verhindern, daß am Bildrand das Programm zusammenbricht, ist ein kleiner Userschutz von Nutzen:

```
75  IF A < Ø OR A > 21 OR B < Ø OR B > 31  THEN GOTO 3Ø
```

und Ihnen fallen ohne Zweifel auch noch Verfeinerungen ein.

Aufgabe

Nehmen Sie statt der INKEY$-Subroutine beliebige Eingaben für 5, 6, 7, 8 und der mit PRINT anzeigbaren Zeichen, lassen Sie das laufen und warten Sie ab, was dabei herauskommt. Verwenden Sie, um Zufallszeichen anzeigen zu lassen, Befehle wie PRINT CHR$ INT (38+26*RND), was eine Zufallsauswahl aus dem Alphabet bewirkt. (Warum?)

Hier noch ein anderes Beispiel – eine Einführung in die Computerkunst. Gezeichnet werden zufällige Quadrate, schwarz oder schraffiert, bis der Speicher voll ist oder Sie mit BREAK aufhören.

```
1Ø   LET A = 1Ø*RND
2Ø   LET B = 1Ø*RND
3Ø   LET Q = 5*RND
4Ø   LET R = 5*RND
5Ø   LET K = INT(2*RND)
6Ø   IF K = Ø THEN LET M$ = "■"
7Ø   IF K = 1 THEN LET M$ = "▒"
8Ø   GOSUB 1ØØØ
9Ø   GOTO 1Ø
1ØØØ FOR I = A TO A + Q
1Ø1Ø FOR J = B TO B + R
```

```
1Ø2Ø  PRINT AT I, J; M$
1Ø3Ø  NEXT J
1Ø4Ø  NEXT I
1Ø5Ø  RETURN
```

Aufgabe

Erinnern Sie sich an das Programm für das Darstellen von offenen oder geschlossenen Rechtecken (Seite 45)? Das Programm für geschlossene rechteckige Flächen war viel kürzer als das für offene Rechtecke. Mit Hilfe von Unterprogrammen können Sie das Programm (auf Kosten vermehrter PLOT-Zeit) wie folgt verkürzen.

1 Schreiben Sie ein Programm, das mit PLOT oder UNPLOT eine rechteckige Fläche von bestimmter Größe oder Position, je nachdem, ob eine Variable T 1 oder Ø ist, zeichnet und wieder beseitigt.

2 Verwenden Sie das als Subroutine in einem Programm, das mit PLOT eine rechteckige Fläche von gegebener Größe und Position zeichnet und dann (wenn es entsprechend angewiesen wird) *dieselbe* Subroutine dazu benützt, mit UNPLOT eine kleinere Fläche im Inneren herauszunehmen, so daß ein ausgehöhltes Rechteck bleibt.

Irgendwo mittendrin brauchen Sie Zeilen wie

```
27Ø  IF T = 1 THEN PLOT X, Y
28Ø  IF T = Ø THEN UNPLOT X, Y
```

(Die Zeichen "=1" und "=Ø" können Sie hier sogar weglassen. Warum? Siehe Handbuch Seite 71.)

Aufgabe: Würfeln mit Grafik

Lassen Sie sich durch INT (1+6*RND) eine Zufallszahl zwischen 1 und 6 geben. Zeigen Sie je nach der Größe dieser Zahl, die wir N nennen wollen, auf dem Bildschirm in Nähe Bildschirmmitte N Punkte genauso an, wie sie auf einem Würfel zu finden sind. Wiederholen Sie das, sobald NEWLINE gedrückt ist.

Schreiben Sie, um das zu leisten, für jede der sechs Möglichkeiten eine Subroutine. Wenn die Subroutine für N mit (meinetwegen) der Zeile 5ØØ +1ØØ*N begonnen wird, (also bei 6ØØ, 7ØØ etc.) und längst zu Ende ist, bevor die nächste beginnt, können Sie schwindeln und GOSUB 5ØØ+1ØØ*N verwenden, um komplizierte bedingte Sprünge zu vermeiden. Auf jeden Fall brauchen Sie aber sechs RETURN-Befehle . . .

Für die NEWLINE-Wiederholung nehmen Sie eine Zeile INPUT K$, die ein Eingabezeichen verlangt; lassen Sie einen GOTO-Befehl folgen, der alles zum Anfang zurückschickt. K$ hat zwar keine Aufgabe, aber der Computer wartet darauf, erhält NEWLINE und geht weiter zum GOTO. Begriffen?

DEBUGGING IV

Läuft das Programm wirklich?

Wie können wir schlüssig beweisen, daß ein Programm genau das tut, was zu leisten es geschrieben worden ist? Ich möchte mich nicht auf eine allzu komplizierte philosophische Diskussion einlassen (wir wären auf dem besten Weg dazu), aber grob gesprochen ist das so ähnlich, als wollte man einen Astronom fragen, ob morgen die Sonne aufgehen wird. Wenn er sehr pedantisch ist, antwortet er vielleicht, die Erde bewege sich schon lange Zeit um die Sonne, und wir besäßen ein System physikalischer Gesetze, die nahelegten, daß sie das regelmäßig weiterhin tun wird, so daß man einiges darauf verwetten könne, es werde auch morgen noch der Fall sein; er würde aber hinzufügen, daß er nicht wissen könne, ob unsere physikalischen Gesetze richtig sind und das, was wir Jahrtausende hindurch beobachtet haben, nicht in Wahrheit die Manifestation eines viel komplexeren Gesetzes sei, dessen Wirkung morgen darin bestehen könne, daß die Richtung der Erdrotation umgekehrt oder die Erde ganz aus ihrer Bahn geholt wird.

Analog: Weil ein Programm sich bei der Eingabe der ersten tausend Datenmengen richtig verhalten hat, heißt das noch nicht mit absoluter Gewißheit, daß es das auch beim eintausendunderstenmal tun wird. In der Tat werden Fehler oft erst Monate oder sogar Jahre, nachdem ein Programm scheinbar erfolgreich abgeschlossen wurde und mehrere dutzendmal oder sogar mehrere hundertmal ohne Probleme gelaufen ist, erkennbar. Das ist im Grunde auch kein Wunder; schließlich übersieht auch der Programmierer am ehesten das, was am seltensten vorkommt.

Hier ein Beispiel:
Wir schreiben eine Reihe von Programmen für die E-Werke Nieder und Hoch, um ihre Kundenkonten zu verwalten. Man teilt uns mit, es gäbe zwei Tarife, A und B. Beim Tarif A zahle der Kunde im Vierteljahr eine Grundgebühr von 60 Mark und pro Einheit dann 10 Pfennig. Bei Tarif B sei keine Grundgebühr fällig, dafür koste die Einheit 70 Pfennig. Wir verfassen also ein Programm, das folgendermaßen aussieht:

```
100  INPUT T$
105  INPUT EINHEITEN
110  IF T$ = "A" THEN GOTO 300
120  IF T$ = "B" THEN GOTO 140
130  GOTO 5000
140  LET RECHNUNG = 70*EINHEITEN / 100
150  PRINT RECHNUNG
160  GOTO 100
```

```
300  LET RECHNUNG = 60 + 10*EINHEITEN / 100
310  PRINT RECHNUNG
320  GOTO 100
...
5000  PRINT "TARIF NICHT GUELTIG"
5010  STOP
```

Na gut. Ich weiß, das Programm könnte straffer sein, und in Wirklichkeit würden wir mehr Informationen brauchen, etwa Namen und Kontonummer des Kunden, aber Sie wissen schon, was ich meine.

Wir testen also das Programm, es läuft richtig, und wir entfernen uns mit den gemurmelten Worten, unsere bemerkenswerten Gaben seien eigentlich vergeudet, wenn man uns mit derlei Krimskrams behellige.

Und das Programm läuft jahrelang wunderbar, bis es eines Tages eine Rechnung über DM 0.00 druckt. Das fällt natürlich keinem Menschen auf, weil es eine Rechnung unter Tausenden ist, die vermutlich ohnehin automatisch kuvertiert wird. Der Empfänger ist verwirrt und amüsiert sich vermutlich über die Rechnung, weil sie wieder einmal beweist, wie dumm Computer in Wirklichkeit sind, aber Anlaß für eine Reaktion scheint es nicht zu geben, also wirft er die Rechnung in den Papierkorb. Leider haben wir gleichzeitig ein Programm geschrieben, mit dem das Absendedatum jeder Rechnung festgehalten wird, und wenn es nicht die Bestätigung erhält, daß die Rechnung innerhalb von 30 Tagen bezahlt worden ist, schickt es eine Mahnung. Diesmal ist der Empfänger weniger belustigt als verärgert, wirft aber auch die Mahnung in seinen Papierkorb. Von da an geht's bergab. Das Programm, das die Zeitspanne zwischen Rechnungszustellung und Geldeingang prüft, weist die technische Abteilung an, dem Kunden den Strom abzuschalten, wenn er nach 60 Tagen nicht bezahlt haben sollte.

Was ist passiert? Ganz einfach! Der Kunde ist Rentner und hat ein Langzeit-Urlaubsangebot für Senioren genutzt. Er war ein gutes Vierteljahr im Ausland und hat ein ganzes Rechnungsquartal lang keinen Strom verbraucht. Außerdem verbraucht er allgemein nur sehr wenig Strom und zahlt aus diesem Grund nach Tarif B. Deshalb hat das System eine Zahlungsaufforderung für Null Mark geschickt. Freilich wird dergleichen nicht oft vorkommen, weil nur sehr wenige Menschen so lange von zu Hause fort sind und es auch kaum Leute geben wird, die Tarif B wählen. Damit das Problem überhaupt auftreten kann, muß der Kunde beiden Bedingungen entsprechen.

Wenn man den Fehler erst einmal entdeckt hat, ist er ganz leicht auszumerzen:

```
145  IF RECHNUNG = 0 THEN GOTO 100
```

womit die PRINT-Anweisung hier vermieden wird. Das erwähnte Problem soll bei einem frühen Computersystem aufgetreten sein; ich kann allerdings nicht beurteilen, ob es nicht nur erfunden ist. Jedenfalls zeigt es, wie ich meine, auf hübsche Weise, daß ein Fehler beinahe eine Ewigkeit lang unbemerkt bleiben kann.

Die Moral davon: Wenn Sie Daten erfinden, um ein Programm zu testen, tun Sie das nicht aufs Geratewohl. Wählen Sie Werte, die den Sprungwerten im Pro-

gramm entsprechen oder ihnen sehr nahekommen. Wenn eine Anweisung lautet:

```
3Ø5  IF U < 3Ø THEN GOTO 5ØØ
```

machen Sie einen Test mit U bei 29.999, einen zweiten mit U = 3Ø und einen weiteren mit U = 3Ø.ØØØ1. Sie könnten gemeint haben:

```
3Ø5  IF U < = 3Ø THEN GOTO 4ØØ
```

Wenn Sie nur U = 1 und U = 16Ø testen, bemerken Sie den Fehler nicht.

Sorgen Sie für Testdaten, mit denen jeder Programmabschnitt einmal getestet wird. Und achten Sie vor allem darauf, daß Sie genau wissen, was bei jeder Testdatenmenge herauskommen muß.

STRINGS

Computer sind nicht einfach "Zahlenknacker". Sie können auch Zeichen knacken, also mit Symbolen umgehen.

Ein String ist eine Folge von *Zeichen*. Die Zeichen sind aufgeführt auf den Seiten 181–187 des Handbuchs, und für jedes gibt es einen Code, über den wir uns dann gleich unterhalten.

Hier ist ein String:

STRINGSTRINGSTRINGSTRINGSTRING.

Hier ein zweiter:

AB334*./>>>><<-B+++++QQJ.

(Wenn Sie die Satzpunkte eingeben, sind auch sie Bestandteil des Strings!)

Ein String kann auch ein einziges Zeichen umfassen, etwa <, oder *überhaupt keines*!

Wenn Sie eine Stringvariable zuordnen, müssen Sie den String in Anführungszeichen setzen:

```
10 LET A$ = "STRINGSTRINGSTRINGSTRINGSTRING"
```

Jede Stringvariable muß ein einzelner Buchstabe sein, gefolgt vom Zeichen $. Für einen String ohne Zeichen genügt LET A$="".

Strings sind zu allem zu gebrauchen, wenn man weiß, wie man mit ihnen umgehen muß. Nehmen Sie einen String, der nur aus einem einzigen Zeichen besteht, etwa:

3

Man kann ihn betrachten als:

a die Zahl 3
b als einen String "3"

und Sie können auf unterschiedliche Weise von der einen zur anderen wechseln. Nehmen wir ihn einmal als String und ordnen wir ihn zu:

```
10  LET A$ = "3"
```

Angenommen, Sie möchten 3 + 5 rechnen. Es hat keinen Sinn, dem ZX81 Folgendes zu befehlen:

```
20  LET B = A$ + 5
30  PRINT B
```

Das geht nicht – probieren Sie es aus, wenn Sie mir nicht glauben. Und warum nicht? Dem dummen Ding ist mitgeteilt worden, daß 3 als *String* zu betrachten

sei, und es weiß nicht, daß das auch eine *Zahl* ist. Aber – aha! – wir können sie mit VAL in eine Zahl *verwandeln*. Versuchen Sie

```
2Ø  LET B = VAL A$ + 5
3Ø  PRINT B
```

Im allgemeinen weiß der Computer, wenn er einen String erhält, der zufällig auch ein arithmetischer Ausdruck ist wie

```
1Ø  LET A$ = "2 + 2 +5*3"
```

nicht, daß er auch als Zahl berechnet werden kann; er hält ihn einfach für einen String von *Zeichen*

```
2 + 2 + 5*3
```

So können Sie das sechste Zeichen herausnehmen:

```
2Ø  PRINT A$ (6)
```

und erhalten die Lösung *. (Das kann nützlich sein: als arithmetischer Ausdruck entspricht es 19, das kein sechstes Zeichen *hat*.) Aber wenn Sie das in eine Zahl verwandeln *wollen*, wird

```
2Ø  PRINT VAL A$
```

als Lösung 19 präsentieren.

Sie können eine Zahl auf zwei Arten in einen String verwandeln. Erstens dadurch, daß Sie ihn in Anführungszeichen setzen, also "3336" oder was auch immer. ABER: Wenn Sie sich innerhalb eines Programms befinden und A + B oder was weiß ich berechnen wollen, was 3336 *ergibt*, hat es keinen Zweck, wenn Sie "A + B" schreiben und hoffen, es käme "3336" heraus; statt dessen erhalten Sie den aus drei Zeichen bestehenden String

```
A + B
```

was etwas völlig anderes ist.

CHR$ verwandelt eine Zahl zwischen Ø und 255 in ein einzelnes Zeichen entsprechend der Liste von Codes im Handbuch auf den Seiten 181–187. Zum Beispiel ist CHR$ 12 das Pfund-Zeichen £.

Manche Zahlen werden nicht verwendet.

CODE bewirkt das Umgekehrte. CODE "£" ist 12. Wenn Sie es mit CODE "£35/H" versuchen, erhalten Sie trotzdem 12; der Computer befaßt sich nur mit dem ersten Zeichen.

LEN sagt Ihnen, wie lang ein String ist.

Bei weitem das Interessanteste an Strings ist, daß Sie Stücke herausnehmen können, die Substrings heißen. Der Befehl

```
A$ (3 TO 7)
```

nimmt den String heraus, die der Reihe nach aus dem 3., 4., 5., 6. Zeichen in A$ besteht. Wenn A$ also = "STRINGSTRINGSTRINGSTRINGSTRING", dann ist A$ (3 TO 7) = "RING". Sie können statt 3 und 7 auch jede andere Zahl nehmen. Und A$ (5) holt nur das fünfte Zeichen heraus – hier also "N".

Ein Beispiel: Sie wollen eine Zahl zwischen 1 und 7 eingeben, um zu erfahren, welcher Wochentag das ist (1 als Sonntag, 2 als Montag usw.). Umständlich wäre da

```
10 INPUT N
20 IF N = "1" THEN PRINT "SO"
30 IF N = "2" THEN PRINT "MO"
...
```

bis alle sieben Tage durch sind.

Aber wenn wir Substrings nehmen:

```
10 LET A$ = "SOMODIMIDOFRSA"
20 INPUT N
30 PRINT A$ (3*N - 2 TO 3*N)
```

geht dasselbe mit 5 Zeilen weniger; siehe auch TAGEFINDER auf Seite 132.

Sie können Strings mit + verbinden und aneinanderreihen wie hier:

```
"HOT" + "DOG" = "HOTDOG"
```

oder sie "alphabetisch" mit < ordnen. Ziehen Sie das Handbuch zu Rate und experimentieren Sie. Man kann oft viel Platz sparen, wenn man Strings behutsam einsetzt.

Aufgabe

Schreiben Sie ein Programm, das für Namen die Initialen finden soll. Das heißt, aus FRIEDERICH KRAFFT KNIRSCH soll FKK werden, aus GESELLSCHAFT MAGNETISCHER BAHNEN HINDUSTAN demnach GMBH.

Ein Wink: Geben Sie die Namen mit INPUT samt Zwischenräumen und allem als Zeichenstring ein. Gehen Sie sie auf der Suche nach Zwischenräumen durch; nehmen Sie den ersten Buchstaben und jeden Buchstaben, der nach einem Zwischenraum folgt, fügen Sie alles zusammen und zeigen Sie mit PRINT das Ergebnis an.

DIAGRAMME

Nur für Leser, die an Mathematik besonders interessiert sind!

Von *Koordinaten*, die zur Bestimmung der Lage von Punkten, Kurven und Flächen, also zur Beschreibung geometrischer Sachverhalte dienen, haben Sie gewiß schon gehört. Für alle Fälle wollen wir aber das Prinzip noch einmal kurz erläutern. Zwei aufeinander senkrecht stehende Gerade heißen Koordinatenachsen und werden mit x und y bezeichnet. An ihnen können wir Entfernungen x und y anzeichnen (negative Werte an der x-Achse nach links, an der y-Achse abwärts) und mit Hilfe der beiden Werte einen Punkt mit den *Koordinaten* x und y festlegen. Bei den Pixels war das ganz genauso (Abbildung 15).

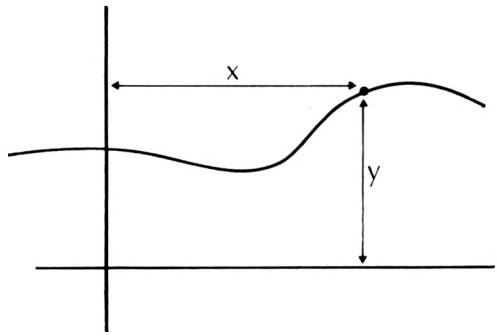


Abbildung 15
Koordinaten für Kurvenberechnung

Wenn wir uns jetzt vorstellen, daß x sich entlang der x-Achse verändert und der Wert von y das ebenfalls tut, und zwar auf eine Weise, die von x abhängig ist, bewegt sich auch der von Koordinaten (x, y) bestimmte Punkt; in der Regel beschreibt er eine *Kurve*. Wenn man eine Formel dafür angibt, auf welche Weise y von x abhängt, etwa $y = x^2 - 3$, dann hat man eine Formel für diese Kurve genannt. (Dieser Gedanke von Descartes aus dem Jahr 1637 ermöglicht es, geometrische Kurven mit Algebra zu erfassen: der Ausgangspunkt für höhere Mathematik.)

Mit PLOT-Befehlen können wir ähnliche Kurven zeichnen – allerdings nur ungenau, weil uns für x nur 64 Werte zur Verfügung stehen.

Genug von dieser Theorie. Versuchen wir es einmal:

```
1Ø  FOR J = Ø TO 63
2Ø  PLOT J,J/2
3Ø  NEXT J
```

Das müßte eine Art Pixel-Treppe ergeben, die auf dem Bildschirm von links unten nach rechts oben hinaufsteigt. Der Computer versucht hier, eine gerade Linie zu zeichnen, aber wegen der Ungenauigkeit der verfügbaren vertikalen Positionen geht das nicht besser. Ein Mathematiker würde von der graphischen Darstellung der Funktion y = x/2 sprechen.

Um endlos viele verschiedene Kurven zu zeichnen, können Sie das ganz einfach abwandeln, indem Sie J/2 gegen irgendeinen anderen Ausdruck für J austauschen. Um etwa die Quadratwurzel von J als eine Funktion von J zu berechnen, müssen Sie das J/2 nur durch SQR J ersetzen und haben dann:

```
10 FOR J = 0 TO 63
20 PLOT J, SQR J
30 NEXT J
```

Das ist schon ein bißchen interessanter, nicht wahr?

Ein Experiment: Ändern Sie das J/2 erneut ab. Versuchen Sie es mit folgendem:

```
(Parabel)          20 PLOT J, .01*J*J
(Sinuskurve)       20 PLOT J, 20*SIN(J/3)+20
(Kosinuskurve)     20 PLOT J, 20*COS(J/3)+20
(Kettenlinie)      20 PLOT J, (EXP(.1*(J-30))
                   +EXP(.1*(30-J)))*1.5
```

. . . Augenblick mal, warum sind die so kompliziert? Was *will* er denn eigentlich?

Wenn Sie für J jeden beliebigen Ausdruck nehmen, der Ihnen gerade einfällt, gibt es Probleme, und zwar damit, was auf den Bildschirm paßt. Der Wert der senkrechten Koordinate muß zwischen 0 und 43 liegen, sonst dreht der ZX81 durch und macht den Laden zu. (Das arme Ding kann mit PLOT nichts anderes anfangen und ist sauer, wenn man das verlangt.) Also muß man den *Maßstab* ändern, damit wieder alles hineinpaßt. Sie werden bald sehen, warum, wenn Sie Folgendes ausprobieren:

```
20 PLOT J, J*J
20 PLOT J, SIN J
20 PLOT J, COS J
20 PLOT J, EXP(J) + EXP(- J)
```

Natürlich gibt es Methoden, dieses Problem zu umgehen – siehe unten bei *Normierung*. Aber bevor Sie sich damit befassen, sind hier noch ein paar interessante Funktionen, die geeignet *sind*, weil wir sie eigens danach ausgesucht haben.

```
2Ø  PLOT J, EXP(-J/2Ø)*SIN(J/2)*2Ø+2Ø
2Ø  PLOT J, 2Ø+2Ø*COS SQR (1Ø*J)
2Ø  PLOT J, ABS(J-3Ø)
2Ø  PLOT J, 2Ø + 15*LN (1+ABS SIN (J*.5))
2Ø  PLOT J, 3*ABS (J-3Ø)**.666
2Ø  PLOT J, 1Ø*ABS(J-3Ø)**.25
2Ø  PLOT J, 4Ø*EXP(-.1*(J-3Ø)*(J-3Ø))
```

Damit man keine komplizierten Ausdrücke schreiben muß, kann man die Formel stufenweise aufbauen, etwa so:

```
2Ø  LET T = J/6
25  PLOT J, 3Ø+.Ø25*T*(T-2)*(T-4)*(T-6)*(T-8)*(T-1Ø)
```

Normieren

Zunächst wollen wir uns nur mit Funktionen *positiver* Zahlen und mit positiven Werten befassen; gut geeignet ist hier SQR, die Quadratwurzel. Nehmen Sie das obige Programm zur graphischen Darstellung und verändern Sie Zeile 2Ø der Reihe nach so:

```
a   2Ø  PLOT J, .5*SQR J
b   2Ø  PLOT J, SQR J
c   2Ø  PLOT J, 1.5*SQR J
d   2Ø  PLOT J, 2*SQR J
e   2Ø  PLOT J, 3*SQR J
f   2Ø  PLOT J, 4*SQR J
g   2Ø  PLOT J, 5*SQR J
h   2Ø  PLOT J, 6*SQR J
```

Sie werden ziemlich rasch bemerken, daß jedes Diagramm der Reihe nach auf dem Bildschirm höher geht; bei g) bleibt das Gerät sogar stehen, weil Kurvenpunkte den Schirm ganz verlassen. Je größer der Wert in *SQR J, desto stärker wird die Kurve in vertikaler Richtung gedehnt. Dieser Wert ist ein *Normierungsfaktor*, und wenn Sie ihn verändern, können Sie erreichen, daß Diagramme genau auf den Bildschirm passen.

Wenn der Normierungsfaktor zu klein ist, werden die Darstellungen so zusammengequetscht, daß Sie gar nichts erkennen können. Probieren Sie

```
2Ø  PLOT J, .1*SQR J
```

Wenn die Funktion, die Sie darstellen wollen, zu groß wird, können Sie sie durch Anpassen des Normierungsfaktors auf den Bildschirm zurückholen. Zum Beispiel geht

```
2Ø  PLOT, J*J
```

über den Bildschirm hinaus, weil 7*7 = 49, was schon zu groß ist. Die größte Zahl, die bei der graphischen Darstellung in Frage kommt, ist übrigens 63*63 = 3969. Wenn Sie das durch 1ØØ teilen, erhalten Sie 39.69, was gut in die zulässigen 43 hineinpaßt. Sie erhalten also ein schönes Diagramm, vorausgesetzt, Sie nehmen 1/1ØØ als Normierungsfaktor:

```
2Ø  PLOT J, .Ø1*J*J
```

Es gibt eine recht naheliegende Grundregel, die dafür sorgt, daß der Normierungsfaktor passend ausgewählt wird. Nehmen wir an, der größte Wert, den die Funktion erlangt, wenn J von Ø bis 63 reicht, sei M (für "Maximum"). Mit dem Normierungsfaktor S ist dann alles in Ordnung, vorausgesetzt, S*M ist nicht größer als 43 – und am besten ziemlich nah an diesem Wert, damit das Diagramm nicht zu stark zusammengequetscht wird. Sie können S*M = 43 sogar festlegen, wenn Sie S = 43/M eingeben. (Für runde Zahlen kann 40/M besser sein, und für M brauchen Sie keine genaue Zahl, sondern eine angemessene Schätzung.)

Sie könnten sogar ein Programm schreiben, um M zu errechnen. Bleiben wir aber bei der J*J-Funktion; dann geht das so:

```
1Ø  LET M = Ø
2Ø  FOR J = Ø TO 63
3Ø  LET Q = J*J
4Ø  IF Q > M THEN LET M = Q
5Ø  NEXT J
```

Das Diagramm wird aber nicht dargestellt, so wie das Programm hier steht. Sie fügen jetzt das PLOT-Programm an:

```
6Ø  FOR J = Ø TO 63
7Ø  PLOT J, (43/M)*J*J
8Ø  NEXT J
```

Ein Nachteil: Sie müssen alle Berechnungen *zweimal* ausführen. Auf ökonomische Weise läßt sich das schwer vermeiden, es sei denn, Sie wissen, *welches* J den größten Wert für J*J ergibt. In diesem Fall ist das offenkundig J = 63, aber so leicht läßt sich das im voraus oft nicht erkennen. (Sie *können* zwar einen Vektor V(I) der Größe 64 dimensionieren, die Werte von J*J als V(J+1) speichern und diese für das PLOT verwenden, aber Vektoren und Arrays beanspruchen viel Speicherplatz! Weitere Auskünfte zu Vektoren und Arrays finden Sie im Handbuch.)

Sie können nicht nur die senkrechte, sondern auch die waagrechte Achse normieren. Die Funktion SQR J oder J*J zeigt das nicht sehr deutlich, dehalb nehme ich 2Ø+2Ø*SIN J, wo das der Fall ist. Versuchen Sie es einmal mit Folgendem:

```
2Ø  PLOT J, 2Ø+2Ø*SIN(.Ø5*J)
2Ø  PLOT J, 2Ø+2Ø*SIN(.1*J)
2Ø  PLOT J, 2Ø+2Ø*SIN(.15*J)
2Ø  PLOT J, 2Ø+2Ø*SIN(.2*J)
2Ø  PLOT J, 2Ø+2Ø*SIN(.25*J)
```

Diesmal verändert sich der *horizontale* Maßstab – aber der Normierungsfaktor wirkt sich ganz anders aus (ist Ihnen das aufgefallen?) Je größer jetzt der Normierungsfaktor, desto stärker wird die Darstellung in horizontaler Richtung zusammengequetscht; die Kurve hat mehr Krümmungen. Warum?

Wenn J von Ø bis 63 reicht, dann die Zahl .Ø5*J von Ø.5*Ø bis .Ø5*63, also von Ø bis 3.15.

Reicht J von Ø bis 63, dann .1*J von .1*Ø bis .1*63, also von Ø bis 6.3.

Im zweiten Fall wird also der *doppelte* Bereich an Werten in denselben horizontalen Raum gequetscht.

Bei einem Normierungsfaktor S, das heißt also, bei Verwendung von

```
2Ø  PLOT J, 2Ø+2Ø*SIN(S*J)
```

stellt man den Bereich von Ø bis S*63 auf der ganzen Bildschirmbreite dar. Je größer S, desto größer der Bereich, und umso stärker quetscht sich alles zusammen.

Wenn Sie also in einem bestimmten Bereich, sagen wir 1ØØØ, darstellen wollen, brauchen Sie S*63 = 1ØØØ, das heißt, S = 1ØØØ/63. Ganz allgemein: Wenn Sie den Bereich Ø bis N wünschen, brauchen Sie einen Normierungsfaktor N/63.

Um zusammenzufassen:

$$\text{bester Normierungsfaktor vertikal} = \frac{43}{\text{größter dargestellter Wert}}$$

$$\text{bester Normierungsfaktor horizontal} = \frac{\text{höchster Wert des Variablenbereichs}}{63}$$

Aufgabe

Wenn Sie nicht wissen, wie etwas am besten aussehen wird, können Sie ein "interaktives" Programm schreiben. Das gibt Ihnen (über INPUT) die Möglichkeit, die beiden Normierungsfaktoren zu wählen und das Diagramm mit PLOT darzustellen; wenn Ihnen das Ergebnis nicht gefällt, fahren Sie das Programm einfach noch einmal mit veränderter Normierung.

Schreiben Sie ein solches Programm für die Funktion 2Ø+2Ø*SIN(J).

Ein Wink: Wenn H der horizontale Normierungsfaktor ist und V der vertikale, dann lautet die funktionsfähige Programmzeile

```
2Ø  PLOT J, V*(2Ø+2Ø*SIN(H*J))
```

Achsenverschiebung

Sie werden sich fragen: "Wozu dauernd dieses 2Ø+2Ø*SIN(J)?" Oder auch: "Alles schön und gut, aber was ist, wenn die Zahlen negativ sind?" Die Antwort auf beide Fragen ist die gleiche. Bei negativen Zahlen verschiebste deine Achsen, wie Fritz, der Fuhrmann, einmal sagte.

Probieren Sie dieses Programm aus:

```
1Ø  INPUT S
2Ø  FOR J = Ø TO 63
3Ø  PLOT J, S+SQR J
4Ø  NEXT J
```

Geben Sie mit Input ein S = Ø,5,1Ø,2Ø, etc.

Was Sie zu sehen bekommen, ist dasselbe Diagramm, das aber entsprechend dem jeweiligen Wert von S auf dem Bildschirm immer höher rückt. Man kann sich das in zweifacher Form vorstellen.

Einmal, daß Sie verschiedene Funktionen darstellen wie 5+SQR J oder 1Ø+SQR J.

Die andere Möglichkeit ist die, daß Sie stets SQR J darstellen, die Position der x-Achse auf dem Bildschirm sich aber verändert. Siehe dazu Abbildung 16, die eigentlich schon genug erklärt.

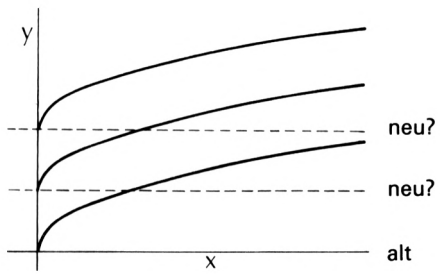


Abbildung 16
Eine Konstante hinzufügen hat dieselbe Wirkung, wie die x-Achse zu verschieben

Wenn Sie beispielsweise eine klare Sinuskurve erhalten wollen, müssen Sie das SQR J von oben durch 2Ø*SIN(.1*J) ersetzen und S so wählen, daß Sie das Ganze in die Mitte des Bildschirms bekommen. Mit S = 2Ø geht das wunderbar: daher also die vielen 2Ø+2Ø*SIN-Zeilen. Dadurch bewegt sich die x-Achse. Um die y-Achse zu bewegen, können Sie den Bereich statt von Ø bis 63 auf meinetwegen −3Ø bis 33 festlegen.

Sie können diese Verschiebungen mit Normierungsveränderungen in beiden Richtungen kombinieren und brauchen die Achsen nicht in der Mitte zu halten – das ist Ihre Sache. Ich hatte sogar schon eine detaillierte Beschreibung darüber verfaßt, wie man eine gegebene Kurve in einem gegebenen Bereich auf dem Bildschirm am besten darstellt, dann warf ich einen Blick auf den Wust von Algebra, den ich geschrieben hatte, und das Ganze landete im Papierkorb. Manchesmal verstellt die Mathematik den Blick auf das Wesentliche, und das ist eine dieser Gelegenheiten.

Statt dessen empfehle ich, zu experimentieren, und zwar mit folgendem Programm:

```
10  INPUT A
20  INPUT B
30  INPUT C
40  INPUT D
50  LET U = -B/(A*2)
60  LET V = 21-D/2
70  PRINT AT V-1,0
90  FOR J = 0 TO 31
100 PRINT "-";
110 NEXT J
120 PRINT AT 0,0
130 FOR J = 0 TO 20
140 PRINT TAB U-1; "*"
150 NEXT J
160 FOR J = 0 TO 63
170 PLOT J, C*SIN(A*J+B)+D
180 NEXT J
```

Hier verschieben A, B, C, D Achsen und Maßstab. Die x-Achse wird mit ----------, die y-Achse mit ********** dargestellt, aber ich habe nicht gegen ein Zusammenbrechen des Programms geschützt, wenn die Funktion den Spielraum überschreitet – abgesehen von meiner Faulheit unterstreicht das die Notwendigkeit, bei der Auswahl von Maßstab und Position für Achsen vorsichtig zu sein.

(Versuchen Sie A = .2,B = -5,C = 10,D = 20. Ein negativer Wert für B ist in der Regel notwendig.)

Balkendiagramme

Um eine Darstellung zu erzielen, die besser aussieht als ein Haufen quadratischer Punkte, können Sie statt dessen vertikale (oder, wenn Sie wollen, auch horizontale) Balken zeichnen.

Dazu verändern wir das Programm. Hier ein typischer Fall.

```
1Ø  FOR J = Ø TO 63
2Ø  LET T = 2Ø+2Ø*SIN(.1*J)
3Ø  FOR I = Ø TO T
4Ø  PLOT J,I
5Ø  NEXT I
6Ø  NEXT J
```

Um auf diese Weise verschiedene Funktionen darzustellen, verändern Sie Zeile 2Ø zu

```
2Ø  LET T = die Funktion von J, die Sie wollen
```

Das macht schon einen viel besseren Eindruck; ein guter Programmierer zeichnet sich eben auch dadurch aus, daß er Daten überlegt präsentieren kann.

Aufgaben

1 Horizontale Balken.
2 Balken, die von oben herunterkommen.
3 Balken, die in der Mitte des Bildschirms anfangen und nach oben oder unten gehen, je nachdem, ob das Ende sich über oder unter der x-Achse befindet.

DEBUGGING V

Zahlen, die gleich aussehen, sind es manchmal gar nicht.

Die Art von Fehlern, nach denen wir bisher gesucht haben, hatten wir uns selbst zu verdanken und waren, einmal erkannt, vergleichsweise leicht zu beheben. Es gibt eine andere Art von Fehler, die durch die Bauart des Geräts selbst hervorgerufen werden. Das ist kein Konstruktionsfehler, sondern eine Folge des inneren Aufbaus aller Computer. Es hat zu tun mit der Präzision, mit der Computer Zahlen speichern. Wenn wir uns durch den Kopf gehen lassen, wie Zahlen im allgemeinen festgehalten werden, zeigt sich rasch, daß für die Zahl der Stellen, die festgehalten werden können, eine Grenze besteht. So kann etwa der Kilometerzähler eines Autos nur sechs Stellen aufnehmen, weil er nur 6 "Fenster" besitzt. Bei einem Computer ist das genauso. Keine Zahl kann mehr besetzen als eine feste Zahl von "Fenstern". Allerdings entspricht nicht jedes Fenster einer Dezimalstelle. Der interne Maschinencode für Zahlen unterscheidet sich von der Art, wie wir sie uns vorstellen, grundlegend, und mit den gräßlichen Einzelheiten möchte ich Sie gar nicht langweilen. Die Tatsache, daß aus Prinzip Ungenauigkeit *und* eine Codeumwandlung verwendet werden, bedeutet, daß die äußere Darstellung einer Zahl (wie auf dem Bildschirm angezeigt) nicht ganz daselbe sein muß wie die innere Darstellung. Ich will Ihnen ein Beispiel für das Gemeinte anhand der Logarithmen geben, die wir aus der Schulzeit kennen. Wenn Sie logarithmisch 2 mit 2 multiplizieren, erhalten Sie:

Zahl	Log
2	Ø.3Ø1Ø
2	Ø.3Ø1Ø
3.999	Ø.6Ø2Ø+

also 2 × 2 = 3.999!!

Die Ungenauigkeit rührt daher, daß Logarithmen nur bis zu 4 Stellen genau sind (sie dürfen also nur 4 Fenster einnehmen) und eine Codeumwandlung (Zahl in Logarithmus, Logarithmus wieder in Zahl) stattfindet.

Hier ist ein Programm, das dieselbe Art von Problem hervorruft:

```
1Ø  FOR P = Ø TO .3 STEP .Ø1
2Ø  LET Q = ATN(TAN(P))
3Ø  IF P <> Q THEN PRINT P, Q
4Ø  NEXT P
```

Mit Zeile 1Ø suchen wir die Tangente von P und kehren den Prozeß sofort um, indem wir den Arkustangens des Resultats suchen. Mit anderen Worten: Q sollte denselben Wert haben wie P. Zeile 3Ø dürfte also nie die Wirkung haben, P und Q anzuzeigen, weil sie immer gleich sind. Wenn das Programm gefahren wird, erhalten wir folgende Ausgabe:

Ø.Ø2	Ø.Ø2
Ø.Ø3	Ø.Ø3
Ø.Ø4	Ø.Ø4
Ø.Ø5	Ø.Ø5
Ø.Ø7	Ø.Ø7
Ø.Ø9	Ø.Ø9
Ø.11	Ø.11
Ø.12	Ø.12
Ø.13	Ø.13
Ø.14	Ø.14
Ø.16	Ø.16
Ø.18	Ø.18
Ø.2Ø	Ø.2Ø
Ø.21	Ø.21
Ø.22	Ø.22
Ø.26	Ø.26
Ø.28	Ø.28

Das ist ein sehr merkwürdiges Ergebnis, weil der Computer nicht nur überhaupt Werte anzeigt und damit behauptet, sie wären verschieden, sondern sie auch noch so anzeigt, als wären sie gleich! Geschehen ist Folgendes: Die komplexen mathematischen Prozesse, die hier abliefen, haben zu kleinen Ungenauigkeiten bei der inneren Darstellung der Zahlen geführt, die ihrerseits die Unterschiede zwischen P und Q erklären. Ungenauigkeiten waren aber auch beim Entschlüsseln des internen Formats zurück zu den auf dem Bildschirm angezeigten Dezimalzahlen beteiligt, so daß sie identisch zu sein scheinen, obwohl der Computer steif und fest behauptet, sie wären es nicht. Beachten Sie, daß bei manchen Werten die internen Codes wirklich identisch *sind*, zum Beispiel bei Ø.Ø6 und Ø.Ø8.

Diese Art von Fehler kann schwerstes Kopfzerbrechen bereiten, und manchmal kommt man nur zurecht, wenn man in die IF-Anweisung einen kleinen Fehler einbaut, so daß wir dann haben:

```
IF ABS(P-Q) < Ø.ØØØØØ1 THEN . . .
```

Die ABS-Funktion ist erforderlich, weil Q größer sein könnte als P. Falls zum Beispiel P=3 und Q=3.1, dann P−Q=−Ø.1, was weniger ist als Ø.ØØØØØ1, so daß die Bedingung erfüllt wäre, wenn es die ABS-Funktion (die das Minuszeichen einfach wegschneidet) nicht gäbe. ABS(−Ø.1)=Ø.1 und damit größer als Ø.ØØØØØ1, so daß die Bedingung nicht erfüllt ist. Genau das wollten wir.

PEEK & POKE

Über den inneren Aufbau des ZX81 brauchen Sie sich in der Regel den Kopf nicht zu zerbrechen. Aber wenn Sie das wollen, können Sie das Gerät dazu veranlassen, Ihnen zu verraten, was es treibt – und zu Ihrem eigenen Nutzen Einfluß darauf nehmen.

Über PEEK und POKE allein könnte ich noch einmal einen Band von diesem Umfang schreiben. Das Meiste davon wäre allerdings viel zu technisch, um von Nutzen zu sein. Aber diese extrem wichtigen Dinge ganz fortzulassen, wäre jammerschade. Wenn man sich mit PEEK und POKE befaßt, kann man nämlich in der Tat sehr viel über die *Arbeitsweise* der Computer erfahren. Was Sie also brauchen, sind ein paar erste Hinweise darauf, wie man es anstellt, den ZX81 dazu zu überreden, daß er von seinen innersten Geheimnissen etwas preisgibt.

Auf Seite 17 habe ich erwähnt, daß Computer Informationen als Folgen von Ø und 1 speichern. Jede solche Ø oder 1 wird ein *Bit* genannt (abgekürzt für *b*inary dig*it*, zu deutsch: Binärziffer). Sie können sich einen String von Ø und 1 als eine Zahl im *Binärsystem* vorstellen, das dem Dezimalsystem ganz ähnlich ist, nur werden anstelle von Einer, Zehner, Hunderter, Tausender etc. die Zahlen eins, zwei, vier, acht und so fort verwendet. In den Programmlistings am Ende des Bandes finden Sie ein Programm für die Umwandlung Binär-Dezimal, das diese Vorgänge ein bißchen näher erläutert.

Vom Binärsystem brauchen wir eigentlich nichts zu wissen, aber wir *müssen* begreifen, daß Computer mit *Bits* arbeiten. Sie haben ihre Bits in der Regel zu Gruppen zusammengefaßt, die *Bytes* heißen. Ein Byte ist eine Folge von acht Bits. 1Ø11ØØØ1 und ØØ111Ø11 sind also typische Bytes.

Es gibt 256 mögliche verschiedene Bytes; wenn Sie in Dezimalzahlen umwandeln, erhalten Sie die Zahlen Ø bis 255. Diese Zahlen werden Ihnen vielleicht bekannt vorkommen: die CODE-Nummern der ZX81-Zeichen sind Zahlen zwischen Ø und 255. Und in der Tat wird jedes Zeichen exakt durch ein Byte vertreten.

Ein Programm – und einige der geleisteten Schritte, um es auszuführen – ist nichts als eine Folge von Zeichen, damit der Computer es als eine Folge von Bytes speichern kann. Um dafür zu sorgen, daß alles in der richtigen Reihenfolge bleibt, erteilt er jedem Byte eine Bezugsnummer, die seine *Adresse* genannt wird. Sie sollten sich demnach das Programm so vorstellen, als wäre es außerhalb des Computers in einer Form ganz ähnlich wie dieser existent:

Adresse	Byte
1	11ØØ11ØØ
2	Ø1111ØØØ
3	ØØØØØØØØ
4	1111Ø1111
.....	

Leider läßt die Computer"architektur" ein Schema wie dieses etwas zu vereinfacht erscheinen.

Der ZX81 besteht aus vier Mikrochips mit den Namen

ROM (Read Only Memory)	=	Festspeicher oder Nur-Lese-Speicher
RAM (Random Access Memory)	=	Speicher mit wahlfreiem Zugriff oder Direktzugriffsspeicher
CPU (Central Processing Unit)	=	Zentraleinheit, (hier: Mikroprozessor)
SLC (Sinclair Logic Chip)	=	Sinclair-Logikchip

Der ROM speichert den BASIC-Interpreter, der RAM Ihr Programm und alles, was erarbeitet werden muß, während es läuft, CPU leistet die Arithmetik und die Logik, und SLC organisiert, wie die anderen zusammenwirken.

CPU oder SLC interessieren uns hier nicht, dafür aber ROM und RAM. Und wir können genau herausfinden, unter welchen Adressen in ROM und RAM welche Bytes gespeichert sind, indem wir PEEK verwenden.

So zeigt ein Befehl

```
PRINT PEEK 837
```

das Byte an, das unter Adresse 837 gespeichert ist – nämlich 2Ø3, oder, in Binärfassung, 11ØØ1Ø11. Angezeigt wird jedoch die entsprechende Dezimalzahl; es schadet also nicht, sich ein Byte einfach als eine Zahl zwischen Ø und 255 vorzustellen.

Wo sollen wir nachsehen, wenn wir PEEK sinnvoll nutzen wollen?

Die ROM-Adressen gehen von Ø bis 8191. Es gibt ganz gewiß gute Gründe dafür, mit PEEK in den ROM hineinzuspähen – Sie können in Erfahrung bringen, wie der ZX81 das Fernsehgerät anweist, ein bestimmtes Zeichen anzuzeigen und damit Spielchen zu machen (etwa, es viermal so groß als üblich anzuzeigen).

Interessanter sind aber die RAM-Adressen. Sie beginnen bei 16384; bis zu 165Ø8 sind sie eingegeben und werden vom Gerät verwendet. Ihr Programm wird angelegt in den Adressen ab 165Ø9. Wie weit, hängt von der Programmlänge ab und davon, ob Sie den Standard 1K-RAM haben, der in den Computer eingebaut ist, oder den Zusatz-RAM mit 16K.

Um herauszufinden, was im RAM gespeichert ist, läßt man am besten den Computer die Arbeit machen. Hier ein Programm, das dafür sorgt:

```
1ØØØ  LET Q = 165Ø8
1Ø1Ø  LET R = PEEK Q
1Ø2Ø  PRINT Q; "□□"; R; "□□"; CHR$ R
1Ø3Ø  LET Q = Q+1
1Ø4Ø  GOTO 1Ø1Ø
```

Ich habe hohe Zeilennummern gewählt, weil es darauf ankommt, das an ein Programm anzuhängen und es mit

GOTO 1ØØØ

laufen zu lassen (also *nicht* mit RUN, weil sonst zuerst das andere Programm läuft!), damit man feststellen kann, wie dieses Programm im RAM gespeichert wird.

Geben Sie das ein und tippen Sie dazu noch

1Ø REM START

2Ø PRINT AT Ø, 1Ø;"*"

Drücken Sie anschließend GOTO 1ØØØ und passen Sie auf. Sie erhalten ein Protokoll folgender Art:

165Ø8	Ø	
165Ø9	Ø	
1651Ø	1Ø	
16511	7	
16512	Ø	
16513	234	REM
16514	56	S
16515	57	T
16516	38	A
16517	55	R
16518	57	T
16519	118	?
1652Ø	Ø	
16521	2Ø	=
16522	23	*
16523	Ø	
16524	245	PRINT
16525	193	AT
16526	28	Ø
16527	126	?
16528	Ø	
16529	Ø	

Was erkennen wir daraus?

Nun, wir können verfolgen, wie unser Programm – jedenfalls der größte Teil davon – – in der dritten Kolonne aufgebaut wird. REM, S, T, A, R, T, . . . dann PRINT, AT, . . . Aber dazu kommt noch anderes. Die dritte Kolonne trägt zum Verständnis nichts bei, aber die zweite tut es. So finden wir unter Adresse 1651Ø in der zweiten Kolonne 1Ø, unter 16521 ist es 2Ø. Das sind offenkundig die *Zeilennummern* aus dem Programm. Sie scheinen in der vorherigen Adresse eine Ø zu haben, dann kommt allerlei Zeug und schließlich wieder eine Ø. Und was bedeuten die Fragezeichen? Wir können im Handbuch unter Anhang A nachschlagen oder das Beilageblatt zur Hand nehmen. Adresse 16519 hat das Byte 118, nämlich NEWLINE. Adresse 16527 hat 126, "Zahl" – eine rätselhafte Feststellung, die sich ohne Zweifel darauf bezieht, wie der ZX81 seine Zahlen intern speichert.

Also: Wir müssen das mit den Zahlen zwar noch ausführlicher ergründen, scheinen aber erkennen zu können, wo das Programm steckt. Zur Überprüfung können wir die *Auflistung des RAM-Inhalts fortsetzen*, wenn wir CONT drücken und dann NEWLINE. Dadurch wird unser kleines Programm mit den Zeilen 1ØØØ–1Ø4Ø zu einem wahren Himmelsgeschenk. Wir erhalten wieder einen Bildschirm voller Angaben:

1653Ø	Ø	
16531	Ø	
16532	Ø	
16533	26	,
16534	29	1
16535	28	Ø
16536	126	?
16537	132	▛
16538	32	4
16539	Ø	
1654Ø	Ø	
16541	Ø	
16542	25	;
16543	11	"
16544	23	*
16545	11	"
16546	118	?

Wenn Sie noch einmal CONT drücken, werden Sie feststellen, daß wir in die Zeilen ab 1ØØØ gelangen, die munter anzeigen, wo sie selbst gespeichert sind! Aber das ist nicht die allerbeste Methode, das anzupacken.

Versuchen Sie es in den Zeilen 1Ø, 2Ø, 3Ø, etc. mit ein paar anderen Programmen. Lassen Sie die Zeilen 1ØØØ–1Ø4Ø unverändert und gehen Sie mit GOTO 1ØØØ in Ihre Programme, um PEEK zu nutzen. Sie werden in der Auflistung bald ein Schema erkennen.

Eine offenkundig sonderbare Sache ist die Art, wie *Zahlen* gespeichert werden. So scheint die 1Ø in PRINT AT Ø, 1Ø die Adressen 16534–16541 zu besetzen. Für eine so kleine Zahl dem Anschein nach viel Platz, aber das hängt mit der Tatsache zusammen, daß der ZX81 *Fließpunkt*-Arithmetik beherrscht (also mit Dezimalzahlen in der Art von 23.567 zurechtkommt) und einen geeigneten Code verwenden kann. Sie können sich stundenlang allein damit amüsieren, herausfinden zu wollen, wie eine gegebene Zahl tatsächlich gespeichert wird.

Aber das eigentlich Schöne dabei ist, daß Sie, sobald Sie einmal wissen, *wo* ein bestimmtes Programmbyte zu Hause ist, es *verändern* können. Der Befehl, der Ihnen diese ungeheure Macht über den armen ZX81 verleiht, lautet

POKE

Sehen wir uns das an.

Was leistet unser kleines Programm oben? Es zeigt mit PRINT bei Position Ø,1Ø * an. Drücken Sie RUN und sehen Sie selbst.

Das wollen wir mit POKE genauer verfolgen. Fügen Sie zwei Programmzeilen hinzu:

```
3Ø  POKE 16544, 12
4Ø  GOTO 1Ø
```

Lassen Sie das Programm laufen, indem Sie eingeben

GOTO 3Ø

Statt * wird £ angezeigt.

Warum? Weil Zeile 3Ø mit POKE in Adresse 16544 (wo ursprünglich das * gespeichert war) das neue Zeichen mit dem Code 12 eingibt, nämlich £.

Wenn Sie den Computer mit LIST beauftragen, werden Sie feststellen, daß Zeile 2Ø des ursprünglichen Programms jetzt so lautet:

```
2Ø  PRINT AT Ø,1Ø; "£"
```

Experimentieren wir noch ein bißchen weiter. Drücken Sie BREAK und löschen Sie Zeilen 3Ø und 4Ø. Verwandeln Sie Zeile 2Ø mit EDIT in ihre ursprüngliche Form zurück. Verändern Sie gleichzeitig die Ø zu 1, dann haben Sie also

```
2Ø  PRINT AT 1, 1Ø; "*"
```

Geben Sie nun GOTO 1ØØØ ein und sehen Sie sich mit PEEK alles noch einmal an. Drücken Sie jedesmal, wenn auf dem Bildschirm kein Platz mehr ist, CONT, damit es weitergeht.

Sie werden die Veränderung bemerken. Die Adressen 16526 und 16528 haben nun

16526 29 1

16528 129 ■

Ansonsten bleibt alles beim Alten.

Fügen Sie, damit wir unser neues Wissen erproben können, die folgenden Programmzeilen an:

```
3Ø POKE 16526, 29
4Ø POKE 16528, 129
5Ø GOTO 2Ø
```

Verwandeln Sie Zeile 2Ø wieder zurück zu PRINT AT Ø,1Ø;"*" und drücken Sie anschließend RUN.

Der Bildschirm sollte *zwei* Sternchen zeigen, und zwar auf den Bildzeilen Ø und 1. Der Computer geht das erste Programm durch und zeigt auf dem Bildschirm Ø an, dann holt er mit POKE die erforderlichen Werte für Bildzeile 1, und GOTO 2Ø veranlaßt ihn, die neue Zeile erneut anzuzeigen.

Versuchen Sie LIST; erneut werden Sie feststellen, daß Zeile 2Ø zu PRINT AT 1, 1Ø;"*" geworden ist.

Beim ROM können Sie POKE natürlich nicht nutzen – "Nur-Lesen" heißt genau das, was es sagt! Was auch gut ist, weil sonst ein versehentliches POKE den BASIC-Interpreter zerstören könnte. Aus diesem Grund können Sie nach Laune mit POKE hineinfahren, um zu erfahren, was sich dann tut.

Die von POKE eröffneten Möglichkeiten sind riesengroß. Um aber auch richtig damit umgehen zu können, müssen Sie mehr über die Einzelheiten der internen Codes wissen, die vom ZX81 verwendet werden. Im Handbuch liefern Kapitel 27 und 28 dafür das Grundwissen, das Sie brauchen. Wenn Ihr Eifer groß genug ist, können Sie den Rest dadurch erarbeiten, daß Sie das obige PEEK-Programm verwenden, Zeilen 1ØØØ–1Ø4Ø. Mehr will ich dazu nicht sagen, weil es besser ist, das auf eigene Faust zu machen, als eine komplizierte Beschreibung durchzuackern, die ein anderer geschrieben hat. Aber wie mit anderen Abschnitten dieses Buches wollte ich anregen, daß Sie sich in sinnvoller Weise eigene Gedanken machen. Wenn Sie bis hierher gekommen sind, werden Sie erkannt haben, daß PEEK und POKE nicht die schrecklichen Geheimnisse sind, als die sie oft hingestellt werden. Es ist nur so, daß sie tiefer in das *System* eindringen, nach dem der ZX81 funktioniert. Experimentieren Sie damit; Umgang mit POKE ist eine faszinierende Herausforderung, vielleicht sogar eine sehr befriedigende.

WOVON ICH NICHT GESPROCHEN HABE

"Adam?"
"Ja, mein Schatz?"
"Wozu soll das Feigenblatt gut sein?"

Das ist eine ganze Menge. Manches erklärt sich von selbst, wie GOTO, FAST, SLOW (siehe Tastenerklärung auf dem Beilageblatt). Manches auch nicht, aber der Platz reicht einfach nicht aus: CLEAR, RAND, LIST, LLIST, Vektoren, Arrays usw. Einiges ist ausgesprochen kompliziert: Hier wäre vor allem die vollständige Darstellung von PEEK und POKE zu nennen. Ziehen Sie, wenn Sie mehr wissen wollen, das Handbuch zu Rate, und was USR betrifft, blättern Sie um zum Abschnitt "Wie geht es weiter?"; dieser Abschnitt befaßt sich auch mit Maschinencode. Tut mir leid. Ich würde Ihnen gern noch mehr erzählen, aber für alles reicht einfach der Platz nicht, und außerdem soll das ja eine *Einführung* sein.

Ich habe Ihnen auch keine Einzelheiten über die verschiedenen Funktionen wie SIN, COS, TAN, ARCSIN, ARCCOS, ARCTAN, LN, EXP, ABS mitgeteilt. Ich gehe eigentlich davon aus, daß Sie, wenn Sie so etwas brauchen, auch wissen, was es bedeutet; man findet es auf jedem wissenschaftlichen Taschenrechner. Es gibt verschiedene Beschränkungen für Wertebereiche; das Handbuch teilt Ihnen die meisten, aber bedauerlicherweise nicht alle mit. Beachten Sie, daß Trigonometrie in *Radianten* und nicht in Grad gerechnet wird; X Grad ist X*PI/18Ø *rad*, und das müssen Sie dem ZX81 sagen. Ich glaube aber nicht, daß dieses Buch der richtige Ort ist, Ihnen Trigonometrie beizubringen.

Bei einigen Programmlistings im Anhang kommen jedoch solche Funktionen vor. Wenn Sie sie nicht verstehen, grämen Sie sich nicht: *dem ZX81 hat das keiner verraten*. Sie können das Programm trotzdem eingeben und sich ansehen, was dann vorgeht. (Ich behaupte nicht, daß es immer einfach wäre – ob etwa POKE sich einer solchen Analyse rasch erschließen würde, bezweifle ich.) Nichts, was Sie auf den Bildschirm tippen, kann dem ZX81 schaden; schlimmstenfalls geraten Sie in ein Durcheinander und müssen von vorne anfangen.

Sie haben bis hierher also alles überstanden. Herzlichen Glückwunsch!
Die Frage ist nur: Wie geht es weiter?

WIE GEHT ES WEITER?

1 Natürlich ist immer noch das Handbuch von Sinclair da.

2 Es gibt in der Bundesrepublik nun eine Zeitschrift ZX USER-CLUB (gegen Schutzgebühr) bei:

Cooperation GmbH
Prinzenstraße 43
8000 München 19

Das Blatt möchte ein Sprachrohr werden, wo es gilt, Schwierigkeiten der Benutzung mit dem Computer auszuräumen, Verbesserungen, Erweiterungen, Software-Programme und vieles mehr an die Benutzer weiterzuleiten.

3 Userclubs sind schon vorhanden oder im Entstehen.

4 In der Reihe "Computer-Shop" bei Birkhäuser, Basel, erscheinen weitere Bände über den ZX81.

5 Bestimmt brauchen können Sie den 16K-Erweiterungs-RAM. Bei den ersten Versionen gab es Anschlußprobleme, und man kann auch jetzt nur davor warnen, den ZX81 mit angeschlossenem 16K-RAM zu bewegen, während er arbeitet, weil das dazu führen kann, daß aus dem Speicher Teile verlorengehen. Lassen Sie sich davon nicht beirren. Ein nützlicher Trick besteht darin, Lötzinn auf die "Finger" der Flachbaugruppe zu tun, wo der 16K-RAM eingesteckt wird. Ein anderer Weg, den manche Techniker gehen – die beiden Geräte fest zusammenlöten! Der einzige Haken dabei ist, daß Sie dann keine anderen Geräte anschließen können. Angelötete Drähte oder Spezialanschlüsse können von Nutzen sein. Außerdem gibt es auch RAMs, die nicht von Sinclair sind; lesen Sie die Fachzeitschriften.

6 Wir empfehlen nicht, sich bei der Beschaffung von Zusatzgeräten für den ZX81 zu sehr gehen zu lassen. Ausnahme: Der 16K-RAM. Das ist so: Wenn Sie viel mehr Geld auszugeben bereit sind, tun Sie besser daran, auf einen leistungsfähigeren Computer umzusteigen, bei dem es Merkmale wie Farbanzeige, Ton, Diskettenspeicherung schon gibt (oder bald geben wird) – und möglicherweise auch Lichtgriffel, Stapler, hochauflösende Grafik und ein richtiges Keyboard (das hat aber jetzt der neue, in Kürze auf dem Markt erscheinende ZX-Spectrum!). Der besondere Reiz des ZX81 liegt eben darin, daß er *billig* ist und man mit ihm viel lernen kann. Wer viele Zusatzgeräte kauft, geht dieses Vorteils verlustig.

7 Was das Gehirnschmalz angeht: BASIC wird Ihnen nach einer Weile ein bißchen begrenzt und schwerfällig vorkommen, und dann muß man Maschinencode lernen. (Ein eigener Band darüber von den Verfassern dieses Bandes, auf den ZX81 zugeschnitten, erscheint in Kürze bei Birkhäuser in der Reihe "Computer-Shop".)

8 Bis Sie Lust verspüren, auf ein raffinierteres Gerät umzusteigen, wird sich angesichts des vorgelegten Tempos die ganze Szene wieder völlig verändert haben. Kaufen Sie sich verschiedene Zeitschriften, um auf dem Laufenden zu bleiben. Wissen Sie noch, wie die ersten Taschenrechner einige Hundert Mark gekostet haben, während man sie heute fast geschenkt bekommt? Bei Personal-Computern ist es nicht ganz so wild,

aber für das gleiche Geld bekommt man immer mehr geboten. In ein, zwei Jahren mag es da ruhiger werden, aber jetzt sollten Sie, um herauszufinden, was für Sie wirklich nutzvoll ist, sich überall umsehen und Vergleiche anstellen.

9 Viel Spaß beim Programmieren!

PROGRAMMBEISPIELE

Die folgenden Programme sind dazu gedacht, verschiedene Merkmale des ZX81 zu erläutern und Ihnen zu zeigen, was man mit 1 K Speicherkapazität alles machen kann. Jedes Programm ist in sich vollständig, muß also nur eingegeben und mit RUN zum Laufen gebracht werden. Die Befehle in den Listings braucht man nicht zu verstehen.

Bis Sie dieses Buch durchgearbeitet haben, sollten Sie aber schon fähig sein, zu analysieren, wie diese Programme funktionieren. Leicht wird das aber nicht immer sein, weil es mitunter schwerfällt, sich an den Programmierstil einer anderen Person zu gewöhnen.

Und weil wir gerade beim Stil sind: Die begrenzte Speicherkapazität des ZX81 und einige Besonderheiten der BASIC-Sprache für diesen Computer haben mich dazu gezwungen, die Programme so zu verfassen, daß sie zwar tauglich, aber nicht so klar sind wie ein möglichst perfektes Programm. Guter Stil verlangt Klarheit, einesteils, weil es eine Qual ist, Programme umarbeiten zu wollen, die aussehen wie die "Kritik der reinen Vernunft" auf Chinesisch, und anderenteils, weil man dann ohnehin weniger Fehler macht.

Ich habe bei vielen Programmen bewußt ungerade Zeilennummern stehen lassen. Es ist erstaunlich, wie oft sich Fehler einschleichen, wenn man ein Programm umnumeriert. Und ich wollte betonen, daß Sie *keine* Zeilennummern verwenden müssen, die Vielfache von 1Ø sind. Die einzigen Gründe dafür, das zu tun, sind, abgesehen von Mode und Pedanterie, die, Platz für Veränderungen im Programm zu lassen oder beim Debugging Ablaufüberwacher einzubauen.

Bei diesen Listings habe ich die Grafikzeichen so wiedergegeben, wie sie auf dem Keyboard stehen. Buchstaben, die in einem Kästchen stehen wie [H] bedeuten Negativschrift; ein leeres Kästchen zeigt einen Zwischenraum an. Zwar sind naheliegende Zwischenräume nicht eigens bezeichnet, dafür aber solche, die wichtig sind und leicht übersehen werden könnten.

Die Programme haben keine bestimmte Reihenfolge. Also ran!

Als mein erstes fertiges Programm habe ich eines genommen, dessen innere Abläufe ziemlich nah an der Oberfläche stattfinden. Es berechnet:

DIE FLÄCHE EINES DREIECKS

Wenn die Seiten eines Dreiecks a, b, c sind, wird die Fläche bestimmt durch die Formel $\sqrt{s(s-a)\ (s-b)\ (s-c)}$, wobei $s = \frac{1}{2}(a+b+c)$. Dieses Programm benützt die Formel dazu, die Fläche eines Dreiecks zu berechnen, dessen Seiten der Reihe nach durch INPUT eingegeben werden.

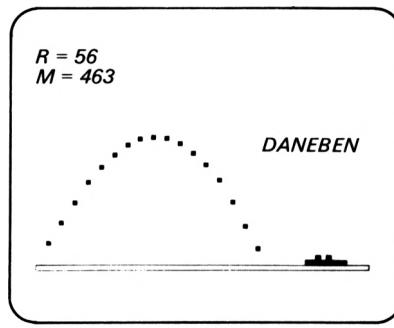


Abbildung 17

6 Um das erstemal oder neu anzufangen, drücken Sie RUN, gefolgt von NEWLINE, wie schon gewöhnt.
Übrigens: Um zu verhindern, daß die lieben Kleinen NEW drücken und Ihr mühsam eingegebenes Programm löschen, können Sie eine Pappschablone auf das Keyboard setzen, mit Löchern nur über den benötigten Tasten. Das ist überhaupt oft sehr nützlich.

7 Der Bildschirm ist hier der Simulation entsprechend 64ØØ Fuß breit.

8 Das Programm ist einigermaßen "userfreundlich" und läuft selbst dann weiter, wenn die Granate oben über den Schirm hinausfliegt. Das liegt an Zeile 18Ø.

9 Die Grafikzeichen in Zeile 4Ø werden erreicht durch S mit SHIFT, in Zeile 6Ø durch 6,W,W,6, alles mit SHIFT.

1Ø In den Zeilen 13Ø und 14Ø ist "PI" Taste M im Funktionsmodus, also *nicht* die Zeichen P und I!

Mathematische Anmerkungen

Die Bahn der Granate wird berechnet nach einer mathematischen Formel, die (bei Vernachlässigung des Luftwiderstands) für einen unter dem Einfluß der Schwerkraft bewegten Körper gilt – wobei man von 981 cm/sec² ausgeht. In Zeile 15Ø steht J für die Zeit, A ist die horizontale Komponente der Geschwindigkeit und B die vertikale; das PI*E/18Ø wandelt aus Grad in Radianten um. Zeile 16Ø berechnet die Höhe zur Zeit J und verwandelt in Bildschirmkoordinaten, indem sie durch 1ØØ teilt. (Jedes PLOT-Pixel hat 1ØØ Fuß im Quadrat.) In Zeile 15Ø ist B/16 die Zeit, die gebraucht wird, bis die Granate im Wasser auftrifft. In Zeile 21Ø ist A*B/16ØØ die Entfernung, die sie bis dorthin zurückgelegt hat; 32ØØ wird gebraucht, weil wir in Zeile 6Ø einen PRINT-Befehl geben, so daß die Entfernung bis zum Schiff 2ØØ*T beträgt.

Wenn Sie mit irgendwelchen anderen Teilen des Programms Schwierigkeiten haben, versuchen Sie es zu verändern und verfolgen Sie, was dabei herauskommt.

Hinweis: Ein Computerspiel wie dieses hier zu *spielen*, hat zwar nur einen bescheidenen Lernwert (die richtige Taste drücken, Entfernung und Winkel schätzen), aber es *schreiben* oder *verstehen* verlangt bestimmte Kenntnisse in Programmierung und Mathematik und kann als Motivierung im Klassenzimmer oder zu Hause von Vorteil sein.

Hier eine Abwandlung des PLOT-Programms. Das ergibt einige sehr schöne Kurven:

LISSAJOUS-FIGUREN

Diese Figuren sind 1815 von Nathaniel Bowditch erfunden worden, haben ihren Namen aber von Jules-Antoine Lissajous, der sie 1857 neu erfand. Sie zeigen einen weiteren Weg, wie man mit PLOT interessante Kurven zeichnen kann: Man nutzt, was die Mathematiker *Parameterdarstellung* nennen. Das heißt: Man macht X und Y von einer Variablen T abhängig und befiehlt PLOT X, Y für Folgen von T.

```
1Ø  INPUT M
2Ø  INPUT N
3Ø  INPUT P
4Ø  FOR J = Ø TO 2ØØ
5Ø  LET X = 3Ø+3Ø*SIN(P+M*PI*J/1ØØ)
6Ø  LET Y = 2Ø+2Ø*SIN(N*PI*J/1ØØ)
7Ø  PLOT X, Y
8Ø  NEXT J
```

Anmerkungen

1 M und N sollten ganze Zahlen sein, aber nicht zu große, weil sonst der Speicherplatz ausgeht. Sagen wir M, N 5 oder kleiner.
2 P sollte irgendwo zwischen Ø und 2 liegen, etwa bei .3 oder .7; wenn P größer oder negativ ist, richtet das allerdings auch keinen Schaden an.
3 Ein guter Anfang ist M = 3, N = 5, P = .5.
4 "PI" ist Taste M im Funktionsmodus.

Der ZX81 kann dank seines Zufallszahlen-Generators RND auch statistische Arbeit leisten. Dieses Programm simuliert

MONOPOLY-WÜRFELN

Wenn Sie schon einmal Monopoly gespielt haben, werden Sie sich erinnern, daß Sie zwei Würfel rollen und die Augen zusammenzählen müssen, wenn Sie erfahren wollen, wie weit Sie mit Ihrer Figur vorrücken dürfen. Vielleicht ist Ihnen auch aufgefallen, daß die Gesamt-Augenzahl 7 häufiger vorkommt als jede andere. In der Tat können Sie damit rechnen, daß bei 36 Würfen die Gesamt-Augenzahl

2 im Mittel 1mal

3 im Mittel 2mal

4 im Mittel 3mal

5 im Mittel 4mal

6 im Mittel 5mal

7 im Mittel 6mal

8 im Mittel 5mal

9 im Mittel 4mal

10 im Mittel 3mal

11 im Mittel 2mal

12 im Mittel 1mal

vorkommt. Sie werden diese Zahlen natürlich nicht genau erreichen, aber auf lange Sicht müßten Sie nah an sie herankommen . . . im Mittel, versteht sich!

Sie können den ZX81 dazu benützen, die Welt der Statistik dadurch zu erforschen, daß Sie so etwas unter Verwendung von Zufallszahlen *simulieren*. Das nachfolgende Programm "wirft" zwei Würfel 144mal (4 × 36, um Ihnen das Dasein zu erleichtern), zählt, wie oft die Gesamtzahl 2, 3, 4 usw. kommt, zeichnet das Resultat als "Balkendiagramm" und vergleicht außerdem das tatsächliche Ergebnis mit den erwarteten theoretischen Zahlen.

```
10 FAST
20 DIM A(11)
30 FOR J = 1 TO 144
40 LET D = 1+INT(6*RND) + INT(6*RND)
50 LET A(D) = A(D)+1
60 NEXT J
70 SLOW
80 FOR J = 2 TO 12
90 FOR T = 0 TO 2*(6-ABS(7-J))
100 PLOT 6+4*J,2*T
110 NEXT T
120 FOR T = 0 TO A(J-1)
130 PLOT 8+4*J,T
140 NEXT T
150 NEXT J
```

Anmerkungen

1 FAST/SLOW beschleunigt die Berechnung, gibt die Darstellung aber Schritt für Schritt.
2 Das Balkendiagramm zeigt die theoretische Zahl als punktierten, die "Versuchszahlen" als durchgehenden Balken; die Balken stellen von links nach rechts Gesamtergebnisse von 2, 3, 4, . . . 12 dar.

Aufgabe

Schreiben Sie als Teil des Programms unter die richtigen Balken 2, 3, 4, . . . 12. Gestalten Sie die Dauer des Ablaufs (hier 144) entsprechend einer INPUT-Variablen unterschiedlich. (Vorsicht bei den "theoretischen" Zahlen!)

Wiederholen Sie das Ganze für eine feste Zahl von Abläufen, meinetwegen 1Ø, und nehmen Sie von jeder Gesamtsumme das Mittel, bevor Sie darstellen.

Selbst ein kurzes Programm kann interessante Ergebnisse liefern. Sie werden sehen, warum dieses hier den Namen hat:

BILLARDHÜPFEN

```
1Ø  FOR I = Ø TO 32Ø
2Ø  LET A = I-128*INT(I/128)
3Ø  LET B = I-4Ø*INT(I/4Ø)
4Ø  LET A = 63.5-ABS(A-63.5)
5Ø  LET B = 19.5-ABS(B-19.5)+22
6Ø  PLOT A,B
7ØØ NEXT I
```

Anmerkung

Damit die Kugel hüpft, sind zwei mathematische Kniffe angewendet worden. Sie brauchen sich darüber den Kopf nicht zu zerbrechen, aber anführen möchte ich sie doch. Wir stellen uns vor, daß die Billardkugel sich auf einer Diagonale im Winkel von 45 Grad zur Horizontalen entfernt. Die Position wird dann umgewandelt zu einer anderen im Inneren eines Rechtecks von der Größe 128 × 4Ø, wobei die Zeilen 5Ø und 6Ø die horizontale Koordinate modulo 128 und die vertikale modulo 4Ø verkleinern. Dieses Rechteck wird dann durch die

Im Modus FAST ist der ZX81 ein recht tüchtiger Zahlenknacker. Wenn Sie sich davon überzeugen wollen: Nehmen Sie eine achtstellige Zahl und sehen Sie sich an, wie rasch dieses Programm aufwartet mit ihren

PRIMZAHLEN-FAKTOREN

```
 1Ø  INPUT K
 2Ø  FAST
 3Ø  PRINT K;"□=□";
 4Ø  LET KØ = K
 5Ø  IF INT(K/2)*2=K THEN GOTO 14Ø
 6Ø  LET N = 3
 7Ø  IF INT(K/N)*N = K THEN GOTO 11Ø
 8Ø  IF N*N > K THEN GOTO 17Ø
 9Ø  LET N = N+2
1ØØ  GOTO 7Ø
11Ø  PRINT N; "X";
12Ø  LET K = K/N
13Ø  GOTO 7Ø
14Ø  PRINT "2X";
15Ø  LET K = K/2
16Ø  GOTO 5Ø
17Ø  IF K = KØ THEN PRINT "PRIM"
18Ø  IF K < KØ AND K > 1 THEN PRINT K
19Ø  SLOW
```

Anmerkungen

1 INT(K/N)*N = K nur dann, wenn K durch N teilbar ist. Das prüft also auf Divisoren.

2 Das Programm versucht K durch 2 und alle ungeraden Zahlen zu teilen, die kleiner sind als die Quadratwurzel von K. Wenn K durch keine solche geteilt wird, ist es eine Primzahl.

3 Wird ein Divisor gefunden, dann teilt das Programm K durch ihn und sucht nach einem neuen Divisor derselben Größe. Findet es keinen, sucht es nach einem größeren Divisor.

4 FAST bedeutet, daß Sie nur die endgültige Ausgabe sehen, die bei einer 8stelligen Primzahl rund 35 Sekunden dauert, bei anderen nicht so lange. Wenn Sie Zeile 2Ø weglassen, dauert es ungefähr 2 Minuten und 2Ø Sekunden. Das abschließende SLOW soll nur ersparen, daß man den Computer später wieder umstellen muß – das gehört sich so! (Die erwähnten Zeitspannen gelten für die Zahl 11111117; bei anderen Zahlen werden sie sich natürlicherweise verändern.)

5 Abwandlungen: Betten Sie das in eine Schleife ein und verwenden Sie einen SCROLL-Befehl, um eine Anzeige der Primfaktoren von K, K+1, K+2, . . . und so weiter bis unendlich zu erhalten. (Beseitigen Sie das FAST, sonst spielt der Bildschirm verrückt.)

6 K darf höchstens 8 Stellen haben, weil der ZX81 höhere Zahlen nicht präzise genug bewältigt.

7 Dieses Programm könnte mathematisch leistungsfähiger gestaltet werden, zum Beispiel dadurch, daß man Vielfache von 3 oder 5 von Versuchs-Divisoren ausschließt. Können Sie das dazu verwenden, ein Programm *schneller* laufen zu lassen? Der Preis für mathematische Effizienz ist vermehrte Programmgröße – gleicht der Gewinn den Verlust aus, wenn Sie vorsichtig sind?

Jeder hat Programme zur Lösung quadratischer Gleichungen. Wie wäre es damit?

GLEICHUNGEN DRITTEN GRADES

Dieses Programm errechnet alle echten Wurzeln kubischer Gleichungen oder Gleichungen dritten Grades in der Form $ax^3+bx^2+cx+d=Ø$ mit ziemlicher Genauigkeit.

```
1Ø  INPUT A
2Ø  INPUT B
3Ø  INPUT C
4Ø  INPUT D
5Ø  LET B = B/A
6Ø  LET C = C/A
7Ø  LET D = D/A
8Ø  LET X = Ø
9Ø  LET G = 2*X*X*X+B*X*X-D
1ØØ  LET H = 3*X*X+2*B*X+C
11Ø  IF H = Ø THEN GOTO 2ØØ
12Ø  IF ABS(X-(G/H))<1.E-8 THEN GOTO 3ØØ
```

```
13Ø  LET X = G/H
14Ø  GOTO 9Ø
2ØØ  LET X = X+1
21Ø  GOTO 9Ø
3ØØ  PRINT "X1 = □"; X
4ØØ  LET A = B+X
41Ø  LET B = X*X+B*X+C
42Ø  LET D = A*A-4*B
43Ø  IF D<Ø THEN PRINT "ANDERE IMAGINAER"
44Ø  IF ABS D < 1.E-7 THEN PRINT "MOEGLICH NUMERISCH
     INSTABIL"
445  IF D<Ø THEN STOP
45Ø  PRINT "X2 = □"; (-A+SQR D)/2
46Ø  PRINT "X3 = □"; (-A-SQR D)/2
```

Anmerkungen

1 Das Programm berechnet eine Wurzel der Gleichung durch einen Iterationsprozeß, der bekannt ist unter dem Namen *Newton-Raphson-Methode*; dabei nimmt man für X einen Versuchswert an und verbessert ihn immer wieder, bis er einer Wurzel möglichst nahe kommt. Diesen Prozeß bewirken die Zeilen 9Ø–15Ø.

2 Wenn Sie sehen wollen, wie diese Iteration sprich Wiederholung funktioniert, fügen Sie

```
131  PRINT X
```

an und verfolgen Sie, wie die Zahlen sich der Lösung annähern.

3 Wenn das Programm eine Wurzel gefunden hat, teilt es durch sie, um zu einer quadratischen Gleichung zu gelangen, die sie nach der Formel in den Zeilen 4ØØ–46Ø löst.

4 Wenn diese quadratische Gleichung keine echten Wurzeln enthält, gilt Zeile 43Ø; das Programm bricht dann in Zeile 45Ø zusammen, was aber nichts schadet.

5 Eine Feinheit: Bei manchen Gleichungen dritten Grades schaukeln sich Ungenauigkeiten in der Algebra zu sehr auf, und das Programm behauptet vielleicht "ANDERE IMAGINAER", wo das gar nicht der Fall ist. Das Problem: Das Zeichen D ist entscheidend, und wenn D nahe bei Null liegt, können sich Fehler auf katastrophale Weise auswirken.

6 Versuchen Sie beispielsweise mit INPUT einzugeben A=4, B=−8, C=5, D=−51. Sie sollten dann erhalten X1=3, andere imaginär. Versuchen Sie es dann mit A=4, B=−16, C=−5, D=51: diesmal erhalten Sie X1=3, X2=2.6213203, X3=−1.6213203.

SPIELAUTOMAT

```
 1Ø  LET C = Ø
 2Ø  DIM A(3)
 3Ø  FOR T = 1 TO 3
 4Ø  LET R = INT(3*RND)
 5Ø  LET A(T) = R
 6Ø  LET I = 5
 7Ø  LET J = 1Ø*T-5
 8Ø  GOSUB 1ØØ*R+2ØØ
 9Ø  NEXT T
1ØØ  LET C = C-1
11Ø  IF A(1)=A(2) AND A(2)=A(3) THEN GOTO 7ØØ
12Ø  GOSUB 1ØØØ
13Ø  INPUT B$
14Ø  IF B$ = "S" THEN STOP
15Ø  GOTO 3Ø
2ØØ  PRINT AT I,J; "■■"; AT I+1,J; "■■"
21Ø  RETURN
3ØØ  PRINT AT I,J; "▪ ▪"; AT I+1,J; "▪ ▪"
31Ø  RETURN
4ØØ  PRINT AT I,J; "▒ ▒"; AT I+1,J; "▒ ▒"
41Ø  RETURN
7ØØ  LET C = C+9
72Ø  GOTO 12Ø
1ØØØ IF C >= Ø THEN PRINT AT
     18,5; "GEWONNEN□"; C; "□ MARK□"
1Ø1Ø IF C < Ø THEN PRINT AT 18,5; "VERLOREN □"; -C;
     "□ MARK"
1Ø2Ø RETURN
```

Anmerkungen

1 Das zeigt drei Zierquadrate an. Sind alle drei gleich, gewinnen Sie 9 Mark. Wenn nicht, verlieren Sie eine Mark. Um ein Spiel zu beginnen, können Sie jede Taste drücken außer STOP oder S – wir empfehlen NEWLINE, weil man sich damit die Berührung einer bestimmten Taste ersparen kann. Drücken Sie S, wenn Sie haltmachen wollen.

2 Im Mittel sollten Sie ohne Verlust abschneiden. Das ist mehr, als Sie in Ihrer Kneipe zu erwarten haben.

3 Beachten Sie den Gebrauch von Subroutinen, um die drei Quadrate darzustellen.

4 Achten Sie auf den richtigen Gebrauch des Bildschirms; Sie erhalten eine vernünftige Darstellung in der Bildmitte, statt einer winzigen, zusammengequetschten Anzeige in einer Ecke.

Der ZX81 erlaubt, daß Sie eine Subroutine *aus seinem Inneren herausholen. Versuchen Sie herauszubekommen, wie das nachfolgende Programm das bewerkstelligt:*

VERSTECKTE FAKULTÄTEN

```
1Ø  LET F = 1
2Ø  INPUT N
3Ø  FAST
35  LET M = N
4Ø  GOSUB 1ØØ
5Ø  PRINT M; "□FAKULTAET IST □"; F
6Ø  SLOW
7Ø  STOP
1ØØ IF N < = 1 THEN RETURN
11Ø LET F = F*N
12Ø LET N = N-1
13Ø GOSUB 1ØØ
14Ø RETURN
```

Subroutine

An welchem Wochentag sind Sie geboren? An welchem Wochentag ist Anne Boleyn gestorben? Das können Sie jetzt feststellen mit:

TAGEFINDER

Das Programm nimmt als Eingabe ein Datum T.M.J. an, wobei T die Nummer des Tages, M der Monat und J das Jahr ist (Beispiel 23.7.1Ø66), und berechnet, welcher Tag das ist (war, sein wird).

```
1Ø  LET A$ = "Ø33614625Ø35"
2Ø  LET B$ = "SOMODIMIDOFRSA"
3Ø  INPUT T
4Ø  PRINT T; ".";
5Ø  INPUT M
6Ø  PRINT M; ".";
7Ø  INPUT J
8Ø  PRINT J; "□IST□";
9Ø  LET Z = J-1
1ØØ  LET C = INT(Z/4)-INT(Z/1ØØ)+INT(Z/4ØØ)
11Ø  LET X = J+T+C+VAL A$(M)-1
12Ø  IF M > 2 AND (J = 4* INT (J/4) AND J <>
     1ØØ* INT (J/1ØØ) OR J = 4ØØ* INT (J/4ØØ)) THEN LET X = X + 1
13Ø  LET X = X - 7* INT (X/7)
14Ø  PRINT B$(3*X + 1 TO 3*X + 3)
```

Anmerkungen

1 Zeile 1Ø speichert eine Liste von zwölf "monatlichen Korrekturzahlen" in gedrängter Form als String. Zeile 1ØØ errechnet den Monat in der Liste als Teil einer Berechnung, die in Anmerkung 4 erläutert wird.

2 Zeile 2Ø nutzt einen ähnlichen Kniff, um das Anzeigeprogramm zu vereinfachen. Beachten Sie, wie Zeile 12Ø die richtige Buchstabengruppe auswählt.

3 Zeile 1ØØ berechnet, wieviele Schaltjahre seit dem Jahr Null schon vergangen sind. Nicht vergessen: Vielfache von 4 sind Schaltjahre, aber Vielfache von 1ØØ nicht, wenn es sich nicht gleichzeitig um Vielfache von 4ØØ handelt.

4 Zeile 11Ø ist der springende Punkt der Berechnung. Es geht darum, die Zahl der vergangenen Tage seit irgendeinem (im Grunde willkürlich gewählten) Bezugsdatum zu zählen, gleichzeitig aber Platz zu sparen, indem

Vielfache von 7 ausgeschieden werden, weil sie für den Tag der Woche keine Bedeutung haben. Die Zahl der Jahre J wird also mit 365 multipliziert, aber 365 = 7*52+1, so daß wir statt 365*J eben J verwenden. Die Eigenheiten der Monatslängen erledigt A$(M). Achten Sie auf die Verwendung von VAL, um ein einstelliges Zeichen in eine richtige *Zahl* zu verwandeln. Um das Programm zu kalibrieren, habe ich ein Datum genommen, für das wir den Wochentag kennen – der 3Ø.9.81 war ein Mittwoch – was mir die Verbesserung −1 am Ende lieferte. Zeile 12Ø paßt die Berechnung im Februar eines Schaltjahrs an, wo sie sonst fehlgehen würde.

5 Eine kalendarische Besonderheit fehlt. 1752 wechselte England bei der Kalenderzählung vom "alten Stil" zum "neuen Stil". Der Tag nach dem 2.9.1752 war der 14.9.72. Mein Programm geht vom neuen Kalenderjahr aus.

Aufgaben

Wandeln Sie für die Zeit vor 1752 das Programm für Daten alten Stils ab.

6 Das Programm akzeptiert unmögliche Daten wie −37.12.−992. Wandeln Sie es so ab, daß es nur vernünftige Daten annimmt.

7 Bei Daten v. Chr. funktioniert das Programm nicht. Wandeln Sie es so ab, daß es auch das beherrscht. Achtung: Es hat zwischen 1 v. Chr. und 1. n. Chr. kein Jahr Ø gegeben, obwohl das aus Gründen der Logik nötig gewesen wäre!

Eine andere Methode, bewegliche Grafik zu erzeugen, bedient sich des SCROLL-Befehls. Hier wird er in Verbindung mit ein bißchen Stringspielerei verwendet:

DASISTEINESINUSKURVE

```
1Ø  LET A$ = "DASISTEINESINUSKURVE"
2Ø  LET T = Ø
3Ø  LET C = 15+15*SIN(T/1Ø)
4Ø  LET Q = T−16*INT(T/16)+1
5Ø  PRINT AT 21,C; A$(Q)
6Ø  SCROLL
7Ø  SCROLL
8Ø  LET T = T+1
9Ø  GOTO 3Ø
```

Was ist das?

Genau.

Wenn Sie das T/1Ø in Zeile 3Ø zu T/3 verändern, wird das Bild klarer.

Strings und Zahlen stehen einander manchmal näher, als Sie vielleicht glauben – zum Beispiel in:

UMWANDLUNG BINÄR-DEZIMAL

Binärzahlen setzen sich nur aus Ø und 1 zusammen; statt Ø,1,2,3,4,5,6, . . . geht das Ø,1,1Ø,11,1ØØ,1Ø1,11Ø, . . . wobei in der Regel eine Zahl wie 11Ø1ØØ1 von rechts nach links aufgeschlüsselt wird als

$$1\times1 + Ø\times2 + Ø\times4 + 1\times8 + Ø\times16 + 1\times32 + 1\times64 = 1Ø5$$

und jede Ziffer das doppelte der vorherigen zählt. Computer verwenden für ihre interne Arbeit natürlich Binärzahlen (moderne Computer in der Praxis allerdings verbesserte Abwandlungen.)

Die folgenden beiden Programme verwandeln eine mit INPUT eingegebene Zahl vom binären ins Dezimalsystem oder umgekehrt.

Binär zu dezimal

```
1Ø  INPUT A$
2Ø  PRINT A$;
3Ø  LET L = LEN A$
4Ø  LET S = VAL A$(1)
5Ø  FOR J = 2 TO L
6Ø  LET S = 2*S+VAL A$(J)
7Ø  NEXT J
8Ø  PRINT "□IST□"; S; "□DEZIMAL"
```

Dezimal zu binär

```
1Ø  LET C$ = ""
2Ø  INPUT A
3Ø  PRINT A;
4Ø  LET D = INT(A/2)
5Ø  LET R = A-2*D
6Ø  IF R = Ø THEN LET B$ = "Ø"
7Ø  IF R = 1 THEN LET B$ = "1"
8Ø  LET C$ = B$+C$
9Ø  IF D = Ø THEN GOTO 12Ø
```

```
1ØØ  LET A = D
11Ø  GOTO 4Ø
12Ø  PRINT "□IST□"; C$; "□BINAER"
```

Anmerkungen:

1 Für die Umwandlung binär-dezimal muß mit INPUT eine Folge von Ø und 1 eingegeben werden, also etwa 11ØØØ11Ø1Ø11. Für die Umwandlung dezimal-binär muß es eine ganze Zahl sein, etwa 3427ØØ6.

2 Die beiden Programme sind zu dem Zweck geschrieben worden, einmal zu zeigen, wie unterschiedlich das Umrechnungsproblem angepackt wird. Es wäre möglich, beide einander im Aufbau ganz ähnlich zu machen.

3 Beachten Sie, daß bei der Umrechnung dezimal-binär die Zeilen 6Ø–7Ø *nicht* ersetzt werden können durch

```
6Ø  LET B$ = "R"
```

Sie können vermeiden, daß es zwei Zeilen werden, wenn Sie raffiniert sind und CHR$ verwenden, oder Sie können schreiben

```
6Ø  LET B$ = ("1" AND R) + ("Ø" AND 1−R)
```

was aus Gründen funktioniert, die damit zu tun haben, wie der ZX81 mit Logik verfährt – siehe Handbuch Seite 72.

Oder auch:

```
6Ø  LET B$ = "Ø"
7Ø  IF R = 1 THEN LET B$ = "1"
```

was ebenfalls wirkt und ein bißchen Platz spart.

Das folgende Programm soll Papa eingeben und sich damit zwanzig Minuten Frieden verschaffen, während die Kinder damit spielen:

HENKERSPIEL

Beim "Henkerspiel" erraten Spieler die Buchstaben eines insgeheim eingegebenen Wortes. Richtige Buchstaben werden an die richtigen Stellen geschrieben, falsche verlieren ein Leben. Wer zu viele Leben verliert, ist tot. Bei dieser Version gibt ein Spieler (meist ist es Papa) fünf Wörter ein, während die anderen Spieler nicht hinsehen, dann dürfen sie gegen den Computer spielen.

```
10  DIM W$(5,12)
20  FOR I = 1 TO 5
30  INPUT W$(I)
40  NEXT I
50  FOR I = 1 TO 5
51  LET C$ = ""
52  FOR L = 1 TO 12
53  IF W$(I,L)="□" THEN GOTO 55
54  NEXT L
55  LET L = L-1
61  FOR P = 1 TO L
62  LET C$ = C$+"-"
63  NEXT P
70  LET N = 6
80  PRINT AT 2,0; N,"LEBEN"
85  IF N = 0 THEN GOTO 225
90  INPUT G$
95  LET F = 1
100 FOR P = 1 TO L
110 IF W$(I,P) < > G$ THEN GOTO 150
120 LET C$(P) = G$
130 LET F = 0
150 NEXT P
160 PRINT AT 11,11; C$
170 LET N = N-F
190 FOR P = 1 TO L
200 IF C$(P) = "-" THEN GOTO 80
210 NEXT P
220 PRINT AT 15,0; "GEWINNT"
225 PRINT AT 11,11; W$(I)
230 INPUT C$
```

```
235 CLS
24Ø NEXT I
```

Anmerkungen:

1 In den Zeilen 1Ø–4Ø gibt Papa fünf Wörter mit 12 oder weniger Buchstaben ein. Dann kann er sich aufs Ohr legen, bis das Programm beendet ist.

2 Buchstaben werden durch Tasteneingaben erraten. Die richtigen Buchstaben werden in der Mitte angezeigt, die falschen verlieren ein Leben. Das Spiel geht weiter, bis alle Buchstaben erraten oder alle Leben verloren sind; der Computer zeigt dann das vorgegebene Wort an.

3 Leben gibt es insgesamt 6; in Zeile 7Ø läßt sich das verändern – beispielsweise zu 1Ø Leben (und einem längeren Schlummerstündchen). Dann muß es heißen 7Ø LET N = 1Ø.

4 Nach jedem Spiel beginnt das nächste mit NEWLINE.

5 Wenn Sie den 16K-RAM haben, können Sie das Programm in mehrfacher Beziehung verbessern.
a) Geben Sie mehr Wörter ein, indem Sie statt der 5 in Zeilen 1Ø, 2Ø und 5Ø eine höhere Zahl einsetzen.
b) Verhindern Sie, daß die Kinder mehr als einen Buchstaben oder gar nichts eingeben, indem Sie
91 IF G$ = "" THEN GOTO 9Ø
hinzufügen und das G$ in den Zeilen 11Ø, 12Ø zu G$(1) machen.

Das ist ein gutes Beispiel für bewegliche Grafik:

ZIEGELSTEIN

Aus der Luft fallen Ziegelsteine herunter. Wenn Sie sie mit Ihrem Knüppel treffen, verschwinden sie. Wenn Sie verfehlen, stapeln sie sich. Werden die Stapel zu hoch, haben Sie verloren. Wie lange halten Sie das durch?

```
1Ø LET H = 11
3Ø DIM A(4)
4Ø LET P = INT(4*RND)+1
6Ø LET J = 5*P
7Ø FOR I = 8 TO 21-A(P)
8Ø PRINT AT I,J; "▩"
81 PRINT AT I-1,J; "□"
```

```
 9Ø  LET M$ = INKEY$
1ØØ  IF M$ = "5" THEN LET H = H-1
11Ø  IF M$ = "8" THEN LET H = H+1
115  PRINT AT 15,H; "□■■■□"
12Ø  IF I = 15 AND ABS(H+2-J) <= 1 THEN GOTO 4Ø
125  NEXT I
13Ø  LET A(P) = A(P)+1
135  IF A(P) = 7 THEN STOP
14Ø  GOTO 4Ø
```

Anmerkungen:

1 Obwohl das nur eine kurze Auflistung ist, verbraucht die Grafikdarstellung viel Speicherplatz, und das Programm paßt in 1K gerade hinein. Mit dem 16K-RAM können Sie mehr Ziegelsteinsäulen (dazu müssen Sie die 4 in den Zeilen 3Ø, 4Ø ändern) aus größerer Höhe (ändern Sie die 8 in Zeile 7Ø zu Ø) herabfallen lassen.

2 Den Knüppel können Sie mit den Tasten 5 und 8 nach links oder rechts bewegen (die Pfeile zeigen die Richtung an). Aus Speicherplatzgründen ist das Programm nicht dagegen geschützt, daß der Knüppel vom Bildschirm verschwindet, also Vorsicht!

3 Mit 16K RAM können Sie zählen, wie viele Ziegelsteine heruntergefallen und wie viele getroffen worden sind und diese Zählergebnisse anzeigen. Bei 1K müssen *Sie* die Zahl der Ziegelsteine in den Säulen am Ende zählen – oder Sie können die Maschine anweisen: PRINT A(1)+A(2)+(3)+A(4).

In Stefans schlimmem Spielklub wird nicht ordinäres Roulette gespielt; da spielt man

LINETTE

Das ist Roulette, gespielt in einer geraden Linie.

```
 1Ø  LET W = 1ØØ
 2Ø  INPUT B$
 22  CLS
 26  INPUT A
```

```
 27  LET W = W-A
11Ø  FOR N = Ø TO 9
12Ø  PRINT AT 1Ø,2*N+6; CHR$(28+N)
13Ø  NEXT N
14Ø  LET R = INT(RND*5)+5
15Ø  LET D = INT(RND*1Ø)
155  LET R = R*1Ø+D
16Ø  FOR N = Ø TO R
165  LET X = N-1Ø*INT(N/1Ø)
17Ø  PRINT AT 1Ø,2*X+6; CHR$ 128
18Ø  PRINT AT 1Ø,2*X+6; CHR$(28+X)
19Ø  NEXT N
2ØØ  PRINT AT 1Ø,2*X+6; CHR$(156+X)
21Ø  GOTO 1Ø* CODE B$(1)
37Ø  IF VAL B$(1) = D THEN LET W = W+1Ø*A
38Ø  GOTO 6ØØ
42Ø  IF INT(D/2) = D/2 THEN LET W = W+2*A
43Ø  GOTO 6ØØ
52Ø  IF INT(D/2) < > D/2 THEN LET W = W+2*A
6ØØ  PRINT AT 16,9;W; "□CHIPS"
61Ø  IF W > = Ø THEN GOTO 2Ø
62Ø  PRINT"HARRY""S MOB WILL BE ROUND IN THEMORNING"
```

Anmerkungen:

1 Nachdem Sie RUN gedrückt haben, können Sie entweder auf EVEN (also alle geraden Zahlen) oder ODD (also alle ungeraden Zahlen) oder auf irgendeine Zahl zwischen Ø und 9 setzen, indem Sie das eingeben. (Jedes Wort, das mit E beginnt, wird als "even" gelesen, jedes, das mit O anfängt, als "odd".)

2 Anschließend teilen Sie mit, wieviel Sie setzen wollen und geben eine Zahl ein. Sie beginnen mit 1ØØ Pfund (oder DM), die in Zeile 1Ø stehen.

3 Wenn Sie gewonnen haben, erhalten Sie auf E oder O den doppelten Einsatz, auf Zahl das Zehnfache. Theoretisch ist das Spiel dadurch "ehrlich".

4 Sie können mehr setzen, als Sie zur Verfügung haben, aber wehe, wenn Sie verlieren. (s. Anmerkung unten)

5 In Zeile 62Ø ist bei THEMORNING kein Zwischenraum. Warum nicht? Fügen Sie ihn ein und sehen Sie sich an, was passiert. "" ist ein *Einzel*zeichen – Q mit SHIFT.

Anm.: Der letzte Satz im Programm: "MORGENFRÜH KOMMEN HARRYS SCHLÄGER"

6 Wenn Sie bei einer Buchstabeneingabe aufhören wollen, drücken Sie zuerst RUBOUT und dann STOP. Bei einer Zahleneingabe genügt STOP allein.

SIE WOLLEN LERNEN, WIE MAN EINEN MIKROCOMPUTER PROGRAMMIERT?

SIE SUCHEN EINE GUTE EINFÜHRUNG FÜR DEN ANFÄNGER?

SIE HABEN EINEN ZX81 GEKAUFT ODER WOLLEN EINEN KAUFEN?

Hier ist ein leicht verständliches Einführungsbuch – wie gemacht für Sie. Locker im Stil, aber seriös im Inhalt bietet Ihnen dieses Buch die Grundlagen des Programmierens in BASIC. Es ist speziell auf den ZX81 zugeschnitten, darüber hinaus aber auch für die Besitzer anderer Geräte mit BASIC nützlich und hilfreich.

WAS WIRD GEBOTEN?

- Wie Sie Ihren ZX81 in Gang setzen
- BASIC-Programmierung
- Über 50 betriebsbereite Programme und Spiele
- Zahlenknacken und Graphik
- Sicherstellen von Programmen auf Magnetband
- Fehlersuche

DAS – UND EINIGES MEHR
FANGEN SIE AN! VIEL SPASS DABEI!

ISBN 3-7643-1398-6

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/