

TIM ONOSKO

DER COMMODORE 64

FÜR HOBBY, SCHULE UND BERUF



wie hole ich
am meisten
aus ihm raus



Tim Onosko

Der Commodore 64

Tim Onosko

Der Commodore 64 für Hobby, Schule und Beruf

Übersetzt von
Ilona und Dr. Josef Kwiatkowski



Eine Coedition der Verlage Carl Hanser und Prentice-Hall International

Titel der Originalausgabe:
Commodore 64, getting the most from it
Copyright © 1983 by Prentice-Hall International Inc.

CIP-Kurztitelaufnahme der Deutschen Bibliothek
Onosko, Tim:
Der Commodore 64 für Hobby, Schule und Beruf /
Tim Onosko. Übers. von Ilona u. Josef Kwiatkowski.
– München ; Wien : Hanser ; London : Prentice Hall
Internat., 1984.
Einheitssacht.: Commodore 64, getting the most
from it (dt.)
ISBN 3-446-14073-5 (Hanser)
ISBN 0-13-152232-9 (Prentice Hall)

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdrucks und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Eine Coedition der Verlage:
Carl Hanser Verlag München, Wien
Prentice-Hall International Inc., London
© 1984 Prentice Hall International Inc., London

Gesamtherstellung: Appl, Wemding
Umschlaggestaltung: Christa Quilici und Günther Gerstmayer
© am Lay-out Carl Hanser Verlag München Wien
Printed in Germany

Vorwort der Übersetzer

Die Hobbycomputer VC-20 und sein größerer Bruder VC-64 wurden in der Werbung des Herstellers als 'Volkscomputer' angekündigt. Die verkauften Stückzahlen dieser beiden Modelle sollten den Begriff 'Volkscomputer' mehr als gerechtfertigt erscheinen lassen. Es ist deshalb nicht verwunderlich, daß über den VC-20 und den VC-64 viel geschrieben worden ist und die Anzahl der Publikationen auch weiterhin zunimmt.

Das vorliegende Buch aus dem Ursprungsland der Computer Freaks - den USA - versucht den Leser in die Möglichkeiten des Commodore 64 einzuweisen. Der Autor hat dabei besonders auf die gesamte Darstellung der Möglichkeiten in ihrer ganzen Breite Wert gelegt und ist aus diesem Grund nicht immer unbedingt in die 'Tiefe' gegangen.

Wir haben uns bemüht, den der amerikanischen Literatur eigenen Stil (d. h. die Lebendigkeit und Frische der Darstellung des Stoffes) einerseits zu erhalten, andererseits aber auch dem europäischen Geschmack anzupassen. Das war sicherlich nicht immer ganz leicht, und es mußte oft ein Kompromiß geschlossen werden. Wir hoffen aber, daß wir den Stoff so dargestellt haben, wie es auch der Verfasser getan hätte, würde er dieses Buch für den deutschsprachigen Raum geschrieben haben.

Allen Lesern wünschen wir recht viel Erfolg mit ihren Computern.

Ilona und Josef Kwiatkowski

Frühjahr 1984

Vorwort

Wenn Sie dieses Buch lesen, haben Sie sich entweder einen Computer gekauft, Sie werden sich einen kaufen oder Sie überlegen es sich zumindest. Das Ziel dieses Buches ist es, Ihnen ausreichend Informationen über den Commodore 64 zu geben, damit Sie ihn optimal einsetzen können. Sie werden lernen, was damit gemeint ist, und wie Ihr Commodore für Sie Arbeiten erledigen kann.

Zunächst einmal muß bemerkt werden, daß dieses Buch nur für den Commodore 64 und andere ihm ähnliche Computer geschrieben wurde. Außer einigen grundsätzlichen Bemerkungen, die für die meisten Computer zutreffen und ein paar Hinweisen auf andere Computer von Commodore (besonders dem VC-20), werden Sie sehr wenig oder gar nichts über die Anwendung des Apple, TRS-80, Atari oder irgendeines anderen Personal Computers erfahren.

Stehen Sie vor der Entscheidung, welchen Computer Sie kaufen wollen? Wenn ja, dann wird Ihnen dieses Buch eine Entscheidungsfindung erleichtern. Es ist besser für Sie, wenn Sie sich die notwendigen Informationen auf diesem Wege besorgen, als wenn Sie sich auf die vielversprechenden Angaben der Prospekte oder der Aussagen der Computerverkäufer verlassen. Dieses Buch enthält neben der sachlich begründbaren Information natürlich auch ein wenig die persönliche Meinung des Autors.

Sind Sie sich noch nicht darüber im Klaren, ob Sie sich überhaupt einen Computer zulegen wollen? Dieses Buch wird Ihnen Aufschluß darüber geben, wie Sie der Computer bei Ihrer Arbeit zu Hause, in der Schule oder im Beruf unterstützt.

Sind Sie ein 'alter Hase' auf dem Gebiet der Mikrocomputer, und waren Sie von Anfang an dabei? Dann werden auch Sie noch einiges Nützliche in diesem Buch finden. Der Commodore 64 ist ein leistungsfähiger Mikrocomputer, der viele interessante Möglichkeiten bietet.

Wahrscheinlich haben Sie sich entweder vor kurzem einen Commodore 64 zugelegt, oder Sie haben die Möglichkeit, solch ein Gerät auf der Arbeit oder in der Schule zu benutzen. Vielleicht beunruhigt es Sie, noch nicht einmal genau zu wissen, was ein Computer ist; Sie glauben, die Mathematik nicht genügend zu beherrschen, oder Sie meinen, daß Ihre Kenntnisse nicht ausreichen, um einen Computer zu programmieren. Das Ganze scheint Ihnen sehr kompliziert zu sein und schüchtert Sie vielleicht sogar etwas ein.

Trotz der Fortschritte des letzten Jahrzehnts, werden Computer oft als

'seelenlose Maschinen' angesehen, und selbst die Bezeichnung 'benutzerfreundlich' seitens der Hersteller, hat nichts daran ändern können. Häufig trifft man eine sehr reservierte Haltung gegenüber dem Computer, vielleicht sogar eine gewisse Antipathie an.

Verdrängen Sie diese Gefühle einfach. Computer sind faszinierend und fesselnd zugleich, und trotzdem nicht komplizierter als ein Auto.. Außerdem benötigen Sie nicht sehr viele Vorkenntnisse, um mit Computern umzugehen. Zum Programmieren ist die Beherrschung der höheren Mathematik nicht unbedingt notwendig. Sie müssen ein Problem nur genügend durchdenken, und Sie müssen lernen, wie dieses Problem in eine Reihe von Teilproblemen aufgespalten werden kann. Auch diejenigen, die in der Schule in Mathematik Schwierigkeiten hatten, werden sehr bald feststellen, daß sie trotzdem erfolgreich im Umgang mit Computern sein können.

Obleich Programmieren Freude bereiten kann, programmieren einige Anwender von Computern - wenn überhaupt - nur sehr selten. Sie greifen lieber auf einige gute, kommerziell erhältliche Programme zurück. Für gewöhnlich erfordert die Anwendung eines Programms sehr viel weniger Wissen, als das Schreiben eigener Programme. Der Anwender braucht nämlich nur einfach den Anweisungen des Programms zu folgen. Seien Sie aber trotzdem vorsichtig: Viele leistungsfähige Programme verlangen von Ihnen, daß Sie sich über sämtliche Möglichkeiten informieren, die Ihnen geboten werden. Dazu benötigen Sie unter Umständen ebensoviel Zeit, Aufmerksamkeit und Konzentration, wie für das Erlernen des Programmierens.

Sie können den Computer aber nicht nur dazu benutzen, 'ernste' kommerzielle Probleme zu lösen, sondern Sie sollten auch daran denken, daß Sie ihn für Bereiche aus dem täglichen Leben einsetzen können. Er kann für Sie das beste Spielzeug sein, das Sie überhaupt kaufen können. Der Computer macht nach Belieben Musik und Zeichnungen für Sie und wird Sie Stunde um Stunde unterhalten. Er ist Ihnen ein Spielpartner, wenn keiner mit Ihnen spielen möchte. Computer sind ebenso zur Unterhaltung wie zu ernsthafter Arbeit geeignet. Hören sie nicht darauf, wenn Sie jemand eines Besseren belehren möchte!

Ein guter Grund dafür, einen Computer zu besitzen, ist ganz einfach der, daß man mit dieser Materie nur dann vertraut werden kann, wenn man praktisch mit ihr umgeht. Vielleicht besuchen Sie aber auch einen Kurs, um etwas über den Umgang mit dem Computer zu erfahren. Eventuell steht aber auch das Fach EDV auf dem Lehrplan Ihrer Schule, oder Sie werden von

Ihrem Arbeitgeber zu einem Kursus geschickt. Das sind natürlich ebenfalls Möglichkeiten, sich zu informieren. Der Kauf eines Personal Computers und der Umgang mit diesem ist aber in jedem Fall eine lohnende Investition, weil Sie sich praxisnah informieren können.

Einige Menschen lassen sich durch die Sprachen dieser 'denkenden' Maschinen ängstigen oder verwirren. Sehr oft scheint die Computersprache sogar ein 'Fachchinesisch' zu sein. Viele Ausdrücke sind aber zu einem großen Teil notwendig, um zu beschreiben, woraus ein Computer besteht und wie er funktioniert. Näheres darüber werden Sie ebenfalls in diesem Buch finden. Das Glossar am Ende des Buches können Sie benutzen, wenn Sie einmal die Bedeutung eines Fachausdruckes vergessen haben sollten. Was Sie dort nicht finden werden, sind Fachausdrücke, die nicht unbedingt notwendig sind.

Sind Sie ein Profi? Nehmen Sie sich Arbeit von Ihrer Dienststelle mit nach Hause? Möchten Sie Karriere machen? Gehören Sie zu denjenigen, die über wenig Zeit verfügen? Dann werden Sie wahrscheinlich die Frage stellen, was der Computer im Einzelnen für Sie tun kann. Diese Frage können Sie nur dann beantworten, wenn Sie erst einmal die Möglichkeiten und Grenzen eines Computers kennengelernt haben. Sie müssen sich dessen bewußt sein, daß Computer nicht von sich aus unbekannte Probleme lösen können (alles Andere ist glatter Unsinn).

Dieses Buch informiert Sie allgemein über den professionellen Einsatz des Computers. Vielleicht kommen Sie dadurch zu der Überzeugung, daß ein Computer Ihnen einen Teil Ihrer Arbeit abnehmen kann.

Das Buch wurde in zwei Teile untergliedert. Der erste Teil beinhaltet allgemeine Grundlagen der EDV, eine Einführung in die Programmiersprache BASIC und Informationen über die Anwendungsmöglichkeit insbesondere in der Textverarbeitung eines Computers. Am Ende des Buches steht ein Kapitel für den erfahrenen Anwender des Commodore-Computers. Dieses Kapitel behandelt die Arbeitsweise eines Computers, Computergrafik und Computermusik.

Sie sollten sich dessen bewußt sein, daß weder dieses, noch irgendein anderes Buch einen Anspruch auf Vollständigkeit erheben, und Ihnen alles Wissenswerte über Computer vermitteln kann.

Bevor Sie nun beginnen, mit Ihrem Computer zu arbeiten und dieses Buch lesen, sollten Sie einige wichtige Dinge im Auge behalten.

- Seien Sie geduldig, und lassen Sie sich nicht frustrieren. Schließ-

lich kann jeder den Umgang mit dem Computer lernen. Geben Sie nicht zu früh auf!

- Erwarten Sie nicht, daß sofort 'der Groschen fällt'. Sie müssen sich etwas Zeit lassen, um sich mit dieser Materie vertraut zu machen.

- Machen Sie niemals den zweiten Schritt vor dem ersten. Der Computer erwartet von Ihnen, daß Sie die Anweisungen der Reihe nach genau befolgen.

- Der Computer kann nicht wissen, was Sie von ihm erwarten. Er führt lediglich die Anweisungen aus, die Sie ihm mitgeteilt haben. Wenn Sie mit dem Computer arbeiten, denken sie immer daran, daß er nur eine Maschine ist, die logisch genau und konsequent Ihren Anweisungen folgt.

- Versuchen Sie, Kollegen zu finden, mit denen Sie zusammen arbeiten können. Vielleicht können Sie auch irgendjemanden aus der Familie für Ihr Hobby begeistern. Dadurch können Sie sich gegenseitig helfen, und Sie werden den Stoff schneller verstehen.

- Schließlich sollten Sie immer daran denken, daß sich die investierte Zeit und Mühe lohnen wird.

Bei all der Faszination und dem Interesse für den Computer, sollten Sie aber Ihr normales Leben nicht vergessen. Vielleicht ist das einzig 'Magische' an einem Personal Computer die Tatsache, daß er die Zeit wie im Fluge vergehen läßt. Es ist nicht richtig, sich ausschließlich mit der Maschine zu beschäftigen und Tag und Nacht vor dem Computer zu verbringen, bis ein Problem vollständig gelöst ist. Der Lernprozeß ist nicht stetig, auch wenn Sie stundenlang über ein Problem gegrübelt haben. Sollten Sie feststellen, daß Ihr Lernprozeß keine Fortschritte mehr macht, stellen Sie den Computer aus. Ein neuer Anfang und eine kurze Pause ist vielleicht alles, was Ihnen gefehlt hat. Schließlich sollte der Computer Zeit sparen und nicht Zeit kosten. Sein Zweck ist nicht, Ihnen das Leben zu erschweren, sondern er soll es Ihnen erleichtern.

Der Verfasser

Tim Onosko

Inhaltsverzeichnis

1	Historische Entwicklung des Computers	1
	Das Innenleben des Commodore 64	
2	Der erste Schritt	6
	Der erste Kontakt mit dem Computer * Ein Platz für Ihren Computer * Fernsehapparat oder Monitor? * Anschluß und Aufbau des Commodore 64 * Inbetriebnahme und Test * Funktionen der einzelnen Tasten * Das Schreiben mit der QWERTY Tastatur	
3	Einige wissenswerte Details	28
	Die RETURN-Taste * Externe Speicher * Das Laden von der Datensette * Wenn das Laden des Programms nicht richtig funktioniert * Das Speichern von Programmen auf Band * Überprüfung des Speichervorgangs * Der Start eines Programms * Der Diskettenspeicher * Die Benutzung des Diskettenspeichers * Das Disketten-Directory * Das Initialisieren einer Diskette * Das Laden eines Programms von der Diskette * SAVE und VERIFY * Löschen von Programmen und Datensätzen auf der Diskette * Diskettenfehler * Kompatibilität von Programmen * Der Commodore 64 als Rechenmaschine * Bestimmung des frei verfügbaren Speicherplatzes * Der Gebrauch von Anführungszeichen * Wie spät ist es?	

4	Eine Einführung in das Programmieren	52
	Die Grundlagen von BASIC * Der Aufbau eines Programms * Die äußere Form eines BASIC Programms * Das Programmende * Variablen * Zeichensetzung in BASIC * Der BASIC Befehl PRINT	
5	Programmierung - eine Weiterführung	77
	Der Eingabebefehl INPUT * GOTO * IF, THEN und OR * Weitere Details * Die FOR..NEXT Anweisung * GOSUB und RETURN * Die Kommentarweisung REMark	
6	Wie der Computer Informationen speichert	99
	Indizierte Variablen * Die DIM-Anweisung * READ-DATA * Datenspeicherung auf Datassette und Diskette * Ein Programm zur Adreßverwaltung * Der Programmentwurf * Vorüberlegungen * Wir beginnen mit dem Programmieren * Die Benutzung des Programms * Die Auswahl eines Datenbankprogramms	
7	Programmierung für Fortgeschrittene	137
	PEEK und POKE * CHD * SYS und USR * SPC und TAB * STOP und WAIT * ON * Mathematische Funktionen * Textverarbeitung * LEN, VAL und STR\$ * ASC und CHR\$ * LEFT\$, MID\$, RIGHT\$	
8	Textverarbeitung	154
	Was man mit einem Textautomaten anfangen kann * Hardware für Textverarbeitung: Drucker und Externspeicher * Textverarbeitung für Fortgeschrittene * Vier Commodore 64 Textverarbeitungsprogramme * Quick Brown Fox * WordPro 3 Plus/64 * Easy Script * PaperClip * Ein paar Tips	

9 Farbe, Grafik, Musik und Spiele

168

Zufallszahlen * Rahmen und Hintergrundfarbe * Zeichendarstellung in Farbe * Grafische Symbole * Ein elektronischer Würfel * Sprite Grafik * Das Muschelspiel * Musik mit dem SID * Die Joysticks * Was fehlt noch?

10 Was kommt nach BASIC?

200

Ausbau der Hardware * Spreadsheets und professionelle Software * Andere Programmiersprachen * Mensch/Maschine Interface

Anhang 1: Wir erforschen den Commodore 64
(Jim Butterfield)

208

Der Speicher des Computers * Verschiedene Arten von Speicher * Memory Maps * RAM * Die Verschiebung von Speicherplätzen * Zusammenfassung

Anhang 2: Grafik mit dem Commodore 64
(Paul F. Schatz)

226

Wie werden Zahlen gespeichert? * Der VIC Chip * Wie programmiert man den VIC? * Sprites * Zusammenstellung der Sprite Register * Zeichen * Bit Map Grafik * Ein paar abschließende Bemerkungen

Anhang 3: Musik mit dem Commodore 64
(Dr. Frank H. Covitz)

282

Der Ton * Die Adressierung des SID * Wir machen Musik * Der wohltemperierte Computer * Musik aus dem Weltall * Geräuscheffektprogramme

Anhang 4: Fehlermeldungen	320
Anhang 5: ASCII/CHR\$ Codes und Konversionstabellen	327
Glossar	333
Schlagwortverzeichnis	339
Danksagungen	342

1 Historische Entwicklung des Computers

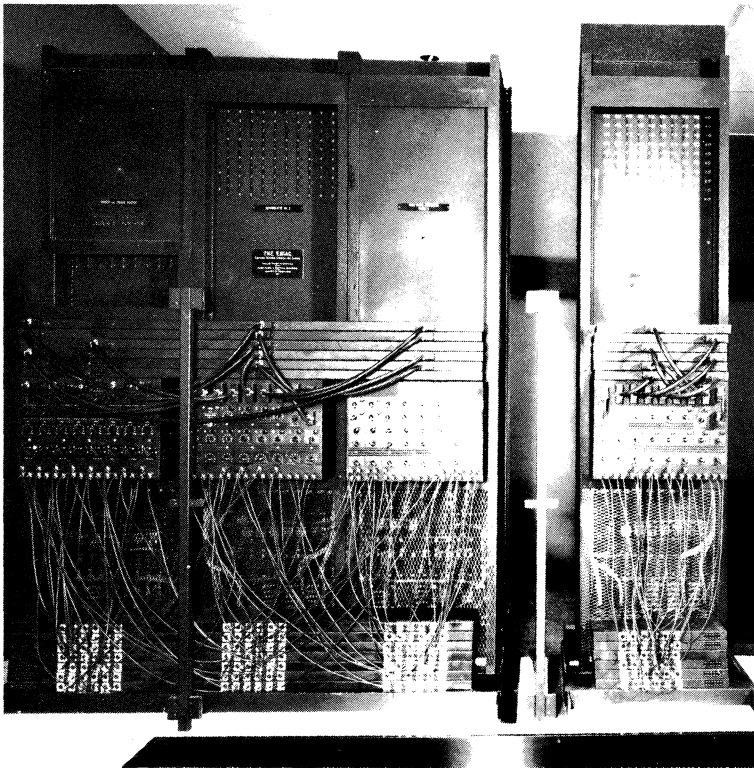
1953 kam das Schlagwort 'Elektronengehirn' auf; dreißig Jahre später war daraus die 'Maschine des Jahres' geworden, die auf der Titelseite der Times in der Ausgabe zur Jahreswende zu sehen war. Heute sitzen Sie wahrscheinlich selbst schon vor solch einem Computer. Was versteht man eigentlich unter der Bezeichnung 'Computer'?

Ein Computer hat ein wenig Ähnlichkeit mit einem menschlichen Gehirn. Ob er 'denken' kann oder nicht, ist eine Frage der Definition. Aus sich heraus kann der Computer nur wenig oder gar nichts tun und ist somit vollständig vom Menschen abhängig. Computer sind letztlich nur die Zusammenfassung einer großen Anzahl einfacher Elemente, die in einer komplizierten Struktur eingebettet sind. Ein Computer führt Anweisungen aus, speichert und löscht Daten und trifft elementare Entscheidungen. Das ist schon alles.

Die Wissenschaftler Eckert und Mauchly entwickelten in den späten 50er Jahren ENIAC, den ersten, heute als legendär empfundenen elektronischen Computer. Er war der erste 'elektronische' Nachfolger der mechanischen Rechenmaschinen, die jahrhundertlang dominierten: vom Abacus der alten Chinesen, bis zur 'analytic engine' des britischen Mathematikers Charles Babbage. Es gibt sogar Spekulationen darüber, daß das mysteriöse britische Stonehenge der 'Computer' historischer Astronomen war. All diese Maschinen waren aber keine Computer im heutigen Sinne.

Remington Rand setzte die Entwicklung des ENIAC fort und konstruierte in den 50ern UNIVAC, den ersten kommerziellen Computer. Kurze Zeit danach

stieg eine Firma in das neue Geschäft ein, deren Name später ein Symbol für Computer werden sollte: IBM. IBM konzentrierte sich damals hauptsächlich auf Büroausstattungen, einschließlich Additions- und Tabuliermaschinen für kommerzielle Zwecke. Das Kennzeichen der Datenverarbeitung mit Tabuliermaschinen war die rechteckige gestanzte Lochkarte, die damals auch mit dem Spitznamen 'IBM-Karte' bezeichnet wurde und heute nach ihrem Begründer, dem Chefindingenieur von IBM, Herman Hollerith, benannt wird.



Bei den ersten elektronischen Computern wurden Elektronenröhren (wie man sie heute noch in alten Radios finden kann) und mechanische Relay-Schaltungen als Bauteile verwendet. Mitte der 50er Jahre wurde die Halbleitertechnologie vorangetrieben und der Transistor entwickelt. Der Transistor konnte als 'elektronischer' Schalter die Röhre und das Relay ersetzen. Da-

durch wurde der Weg für die nächste Computergeneration geebnet, die größere und leistungsfähigere Maschinen hervorbrachte. Die Computer wurden kleiner, leistungsfähiger und sicherer, und dennoch waren Computer der 60er Jahre so groß wie mehrere Kühlschränke zusammen.

Der Durchbruch in der Entwicklung des Computers kam erst, als die Wissenschaftler die Möglichkeit entdeckten, zehntausende von Transistoren auf einem einzigen 'Chip' unterzubringen. Dies waren die ersten integrierten Schaltkreise. Ein Problem war jedoch noch immer nicht gelöst.

Bei älteren Computern wurden die Daten in einem 'Kernspeicher' gespeichert. Jedes Bit wurde durch einen kleinen Ring dargestellt, der in zwei Richtungen magnetisiert werden konnte. Tausende oder zehntausende solcher Bits konnten in einem Computer enthalten sein. Diese Kernspeicher stellten aufgrund des großen Raumbedarfs und der langsamen Geschwindigkeit, mit der die Daten gelesen und geschrieben werden konnten, das schwächste Glied des Computers dar. Die stetige Entwicklung auf dem Gebiet der Halbleitertechnologie führte in den 70er Jahren zu der Entwicklung der Halbleiterspeicher, wie man sie heute in modernen Computern findet. Halbleiterspeicher unterscheiden sich von den Kernspeichern dadurch, daß die Information beim Ausschalten des Computers verlorengeht.

Kurz nachdem die Halbleiterspeicher auf den Markt kamen, wurde der erste Mikroprozessor entwickelt. Ein Mikroprozessor ist ein 'kleiner Computer', der auf einem einzigen Halbleiter-Chip untergebracht ist.

'Personal' Computer sind auf solchen Mikroprozessoren aufgebaut. Die Entwicklung der ersten Mikroprozessoren verlief sehr stürmisch und verschiedene Bezeichnungen kamen auf und verschwanden wieder. Die meisten dieser 'kleinen Computer' wurden an Hobby- und Elektronik-Bastler verkauft. Sie wurden meist nur dazu eingesetzt, irgendwelche Lampen ein- bzw. auszuschalten. Trotzdem ließ sich das Spektrum ihrer späteren Anwendungsmöglichkeit schon erkennen.

1976 sahen zwei Gesellschaften ihre Chance und starteten die Entwicklung von Personal Computern, die von fast jedem benutzt werden konnten. Die beiden jungen Männer Steve Jobs und Steve Wozniak bauten für eine neu gegründete Gesellschaft - genannt Apple - einen Single-board Computer (ein Einplatinencomputer). Diesen Computer nahmen sie zu Treffen von Computer- und Elektronikfans in Nordkalifornien mit. Damit war der Apple I geboren, dem der Apple II und weitere Computer folgen sollten.

Aus einem Reparaturunternehmen für Schreibmaschinen, das von Jack

Tramiel (einem polnischen Einwanderer) gegründet wurde, entwickelte sich das Unternehmen Commodore Business Machines. Commodore wurde zu einem der größten Hersteller von Rechenmaschinen. Trotzdem wäre das Unternehmen fast an der starken Konkurrenz zweier Ingenieure gescheitert, die von dem Pionier der Mikroprozessoren, Chuck Peddle, angeleitet wurden. Sie begannen mit Arbeiten an einer Eigenentwicklung, dem PET von Commodore. Sowohl PET als auch Apple hatten denselben Mikroprozessorchip, den 6502 als Grundlage. Dieser wurde von MOS Technology, einem Unternehmen hergestellt, das inzwischen zu Commodore gehört. Eine Version dieses Chips 6502 wurde später als Grundlage für die sehr erfolgreichen Videospiele von Atari benutzt. Der PET war ursprünglich für Radio-Shack entwickelt worden, das auch den Vertrieb dieses Computers übernehmen wollte. Radio-Shack entschied sich aber dann, selbst einen eigenen Computer zu entwickeln und zu vermarkten: den TRS-80. Ende 1977 gab es deshalb drei Personal Computer auf dem Markt. Das Zeitalter des Personal Computers begann.

Das Innenleben des Commodore 64

Das Design des Commodore 64 ist dem des PET und dem des VC-20 sehr ähnlich. Ein früher Vorgänger des Commodore 64 wurde zwar entworfen, aber niemals gebaut. Er benutzte denselben Mikroprozessor wie der PET, hatte das gleiche Konzept in der Tastatur für die Farbgrafik und bot die Möglichkeit der Computermusik durch einen eingebauten Sound-Generator. Wie der PET hatte er einen seltsamen Namen: der TOI Computer. TOI stand gut gemeint für die Abkürzung 'The Other Intellect' (das andere Intellekt). Scherzhaft wurde gesagt, daß es für 'Tool of Idiots' (Werkzeug für Idioten) stand. Der TOI wurde aber zugunsten des VC-20 niemals gebaut. Der VC-20 sollte ein preiswerter Computer sein, mit dem es möglich sein sollte, Farbgrafik darzustellen. Viele Merkmale des TOI wurden jedoch in dem Commodore 64 benutzt.

Das Innenleben des 64 besteht aus einer neuen Version des 6502 Mikroprozessors. Die meisten Computer werden so konzipiert, daß das Betriebssystem (operating system) und die Programmiersprache BASIC permanent in einem ROM (Read Only Memory) gespeichert sind. Diese Konzeption bewirkt, daß dem Benutzer der komplette RAM-Bereich (Random Access Memory) frei zur Verfügung steht. Der 64 bricht jedoch mit dieser traditionellen Philo-

sophie. Die einzelnen Funktionen sind im Rechner nicht mehr klar voneinander getrennt. So ist es beispielsweise möglich, daß der im Rechner implementierte BASIC-Interpreter durch einen externen Interpreter ersetzt werden kann oder daß zwischen ROM und RAM beliebig hin- und hergeschaltet werden kann. Diese 'Chamäleon'-Struktur ist sehr vorteilhaft und wird jetzt durch andere Computerhersteller nachgeahmt.

Zusätzlich ist es mit dem Commodore 64 möglich, ausgezeichnete Farbgraphik und Computermusik zu machen, da er zwei neue Bausteine, einen Video Controller und einen Tongenerator enthält. Weitere integrierte Bausteine erlauben den Anschluß mannigfaltiger Peripheriegeräte.

Um das Innenleben dieses Computers richtig kennenzulernen, muß man einen Forschergeist besitzen. Durch dieses Buch werden Sie angeleitet, Ihren Computer richtig kennenzulernen, indem Sie spezielle BASIC-Programme für verschiedene Zwecke schreiben. Dadurch werden Sie erfahren, wie Sie Ihren Computer für sich nutzen können. Fertig? Wir wollen jetzt anfangen...

2 Der erste Schritt

In diesem Kapitel werden Sie den ersten Kontakt mit dem Computer bekommen. Wir sollten die Angelegenheit zunächst von der praktischen Seite angehen.

Der erste Kontakt mit dem Computer

Wie die meisten Leute, so werden auch Sie sicherlich schon den Computer ausgepackt und an das Stromnetz angeschlossen haben, obwohl in den Bedienungsanleitungen immer steht: 'Lesen Sie diese Anweisungen zuerst.' In einem Sturm der Begeisterung ist es aber sicherlich schwierig, diesen Anweisungen Folge zu leisten und das Bedienungshandbuch zuerst durchzulesen, bevor man den Computer anschließt.

Ihre Begeisterung sollte Sie aber nicht dazu verführen, alles ausprobieren zu wollen. Vergewissern Sie sich erst, daß die Lieferung vollständig ist. So sollte beispielsweise die Verpackung außer dem Commodore 64 das Netzteil und ein Videokabel enthalten. Ist alles vollständig?

Schauen Sie ebenfalls nach, ob die einzelnen Teile funktionsfähig und unbeschädigt sind. Es kann durchaus vorkommen, daß der Computer durch den Transport Schaden genommen hat. Lassen Sie sich nicht darauf ein, daß der Verkäufer diese Fehler eventuell mit der Bemerkung 'Das ist ja nur ein Schönheitsfehler' abtut.

Ein Platz für Ihren Computer

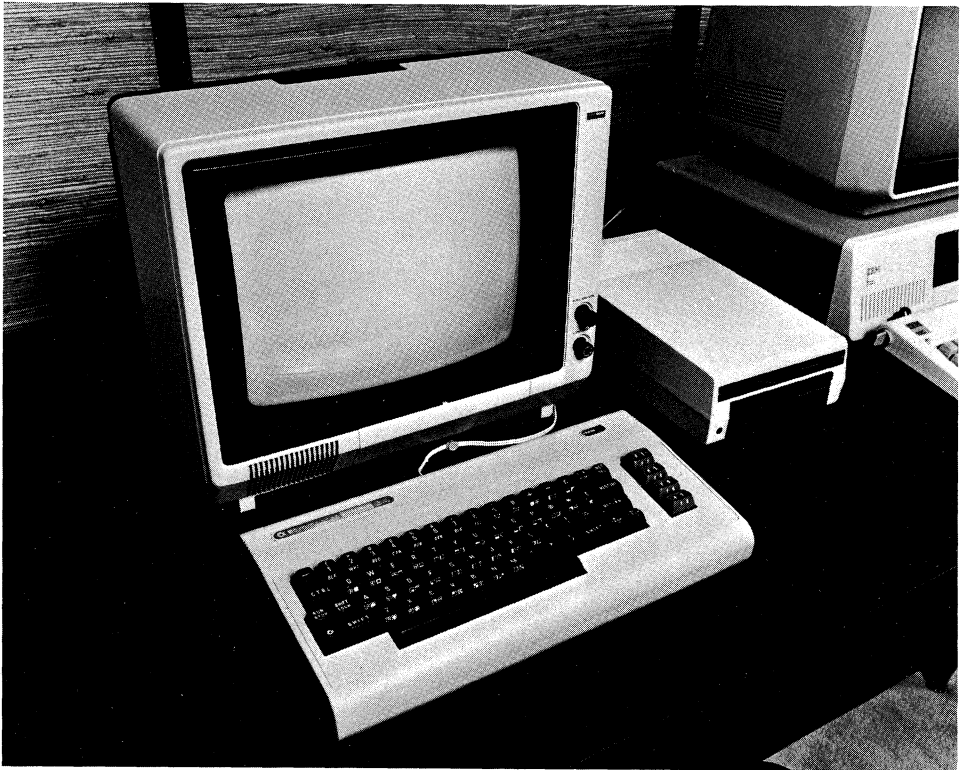
Sie sollten sich den Platz für Ihren Computer gründlich überlegen. Vielleicht beabsichtigen Sie, den Computer in Ihrem Wohnzimmer in der Nähe des Fernsehapparates aufzustellen. Denken Sie aber daran, daß der Computer kein Videospiel ist. Ein Videospiel hat nämlich keine Tastatur wie der Commodore 64. Sie würden sicherlich nicht auf die Idee kommen, eine Schreibmaschine auf den Teppich Ihres Wohnzimmers zu stellen, oder doch?



Wenn es möglich ist, wählen sie deshalb einen anderen Platz. Einen Platz, an dem Sie in Ruhe ungestört denken und arbeiten können. Wo schreiben Sie beispielsweise Ihre Briefe? Solch ein Platz wäre auch für Ihren Computer sehr geeignet.

Wie eine Schreibmaschine, sollte Ihr Computer auf einem Tisch stehen. Da der Commodore 64 zu einem größeren Computersystem ausgebaut werden

kann, sollten Sie gleich berücksichtigen, daß Sie eventuell zusätzlichen Platz benötigen. Für den Anfang benötigen Sie mindestens einen kleinen Fernseher oder einen Monitor. Vielleicht wollen Sie dieses System später einmal durch den Kassettenspeicher 'Datassette' oder durch eine Floppy Disc erweitern. Vielleicht wollen Sie aber auch ein Modem anschließen, um mit Ihrem Computer mit anderen Computern zu kommunizieren. Alle diese Zusatzgeräte benötigen aber Platz! Auch das Netzteil, welches extern ange-



schlossen werden muß, benötigt Platz. Die einzelnen Teile Ihres Computersystems sollten so platziert werden können, daß sie nicht beschädigt werden.

Weiterhin benötigen sie Platz, um Ihre Bücher oder um Papier für Aufzeichnungen und dergleichen in greifbarer Nähe zu haben.

Die Beleuchtung des Arbeitsplatzes ist ebenfalls von großer Bedeutung. Eine Deckenbeleuchtung ist sicherlich die beste Lösung. Eine Tischlampe

reicht aus, um die Tastatur (keyboard) ausreichend zu beleuchten. Denken Sie immer daran, daß das Licht weder zu hell, noch zu dunkel sein darf. Reflexionen auf dem Bildschirm können sehr störend sein, und sie lassen das Auge schnell ermüden. Stellen Sie den Bildschirm auch nicht in das Gegenlicht, d.h. in die Nähe eines Fensters. Das Auge kann sonst leicht überanstrengt werden.

Die Stromversorgung für Ihr Computersystem kann ebenfalls Probleme bereiten. So benötigen Sie beispielsweise für den Computer selbst für den Bildschirm, für die Floppy Disc oder für die Datensette je einen eigenen Stromanschluß. In den meisten Fällen hat sich hier eine Vielfachsteckdose mit Verlängerungskabel bestens bewährt.

Fernsehapparat oder Monitor?

Für Ihren Commodore 64 können Sie als Sichtgerät zwischen einem Fernseher oder einem Video-Monitor wählen. Es gibt Unterschiede zwischen beiden Versionen!

Ein Fernseher ist ein Empfänger. Wie ein Radio empfängt es die Wellen, die von einer Fernsehstation ausgesendet werden. Der Commodore 64 besitzt einen Ausgang, der ähnlich wie die Fernsehstationen Radiowellen aussendet. In dem Fernsehapparat werden durch elektronische Schaltungen die Videosignale zur Steuerung der Bildröhre herausgefiltert.

Ein Video-Signal unterscheidet sich von einem Radio-Signal. Ein Video-Monitor ist darauf ausgerichtet, Video-Signale zu empfangen. Die meisten Fernsehgeräte können dieses nicht. Da bei einem Video-Monitor die Aufarbeitung des Radio-Signals entfällt, ist das Bild meist erheblich klarer und brillanter.

Nicht alle Fernsehapparate - vor allem älterer Bauart - arbeiten befriedigend als Bildschirm für Personal Computer.

Die Darstellung der Farben kann ebenfalls ein Problem sein, wenn Ihr Fernsehapparat nicht richtig funktioniert. Es ist unbedingt notwendig, daß sämtliche Farben mit dergleichen Intensität ausgestrahlt werden.

Wenn Sie einen Fernsehapparat benutzen wollen, dann kann man generell dazu sagen, daß ein billiger - aber neuer - Fernsehapparat bessere Ergebnisse liefert, als ein alter.

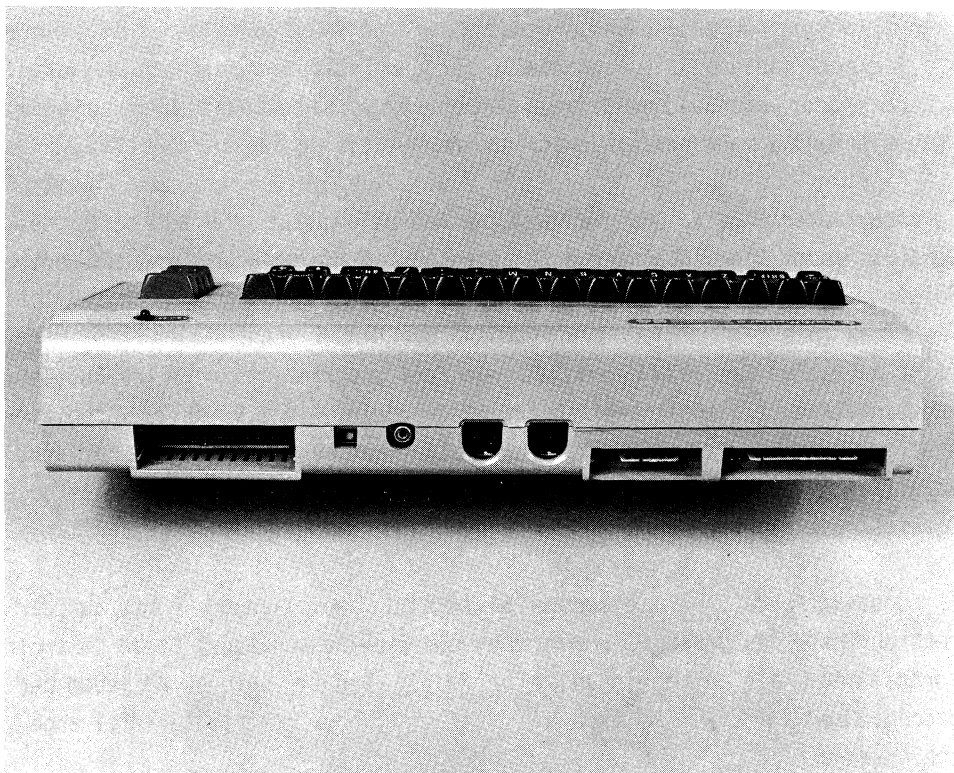


Es gibt einen Unterschied zwischen der Qualität von Schwarz/Weiß- und Farbmonitoren. Alle Bildröhren enthalten eine Schicht, die aus Phosphor und anderen chemischen Substanzen besteht, die durch einen Elektronenstrahl zum Leuchten angeregt werden.

Farbbildröhren bestehen aus drei Schichten für die Farben: Grün, blau und rot. Der Farbmonitor erzeugt also Schwarz/Weiß Bilder durch Kombination dieser drei Grundfarben. Deshalb sind die Konturen meist nicht so scharf wie bei Verwendung eines Schwarz/Weiß-Monitors.

Anschluß und Aufbau des Commodore 64

Der Anschluß des Commodore 64 ist so einfach, wie der Aufbau eines Stereo-Hifi Systems. Auch dann, wenn Sie technisch unbegabt sind, werden Sie



keine fremde Hilfe benötigen.

Nachfolgend sind die einzelnen Anschlüsse des Computers beschrieben.

Spannungsversorgung. Den Anschluß für die Spannungsversorgung finden Sie auf der rechten Seite des Gerätes. Hier wird der Gerätestecker des Kabels angeschlossen, das den Rechner mit dem Netzgerät verbindet.

Recorder-Anschluß. Der Recorder-Anschluß dient zum Anschluß für die Datasette zur Speicherung von Programmen oder Daten. Er befindet sich auf der Rückseite des Gerätes als zweiter Anschluß von rechts. Die Datasette wird über einen Flachstecker angeschlossen.

TV-Anschluß. Über diesen Anschluß können Sie das Fernsehgerät mit dem Antennenanschluß (75 Ohm) anschließen. Das Signal enthält sowohl die Bild- als auch die Toninformation. Dieser Anschluß ist der zweite Anschluß

von links auf der Rückseite des Gerätes.

Serieller Ausgang. Der serielle Ausgang dient zum Anschluß eines Druckers oder eines Disketteneinzellaufwerks. Er befindet sich auf der Rückseite und ist der dritte Anschluß von rechts.

Modul-Steckplatz. Die rechteckige Aussparung auf der Rückseite des Gerätes (ganz links) ist der Modul-Steckplatz. Hier können Programm- oder Spielmodule eingesteckt werden.

User Port. Der User Port ist ein frei programmierbarer Ein-/Ausgang und bietet Anschlußmöglichkeiten für Steckmodule, wie z.B. RS-232-Schnittstelle. Er wird auch oft als 'Paralleler Port' bezeichnet, da er acht parallele Datenleitungen enthält. Er befindet sich auf der Rückseite des Gerätes ganz rechts.

Control Ports. Der Commodore 64 besitzt zwei Control Ports auf der rechten Seite des Gerätes, links neben dem Ein-/Ausschalter. Diese Control Ports können als Steuereingänge für Spiele benutzt werden. An jedem der beiden Eingänge kann ein Joystick, ein Paddle oder ein Lichtgriffel angeschlossen werden.

A-V Port. Der A-V Port ist ein Audio- und Video-Ausgang. Hier kann das isolierte Tonsignal abgenommen und einer Hifi-Anlage zugeführt werden. Außerdem steht hier das Video-Signal zur Verfügung, mit dem ein Monitor gespeist werden kann. Dieser Port befindet sich auf der Rückseite des Gerätes (dritter Stecker von links).

Inbetriebnahme und Test

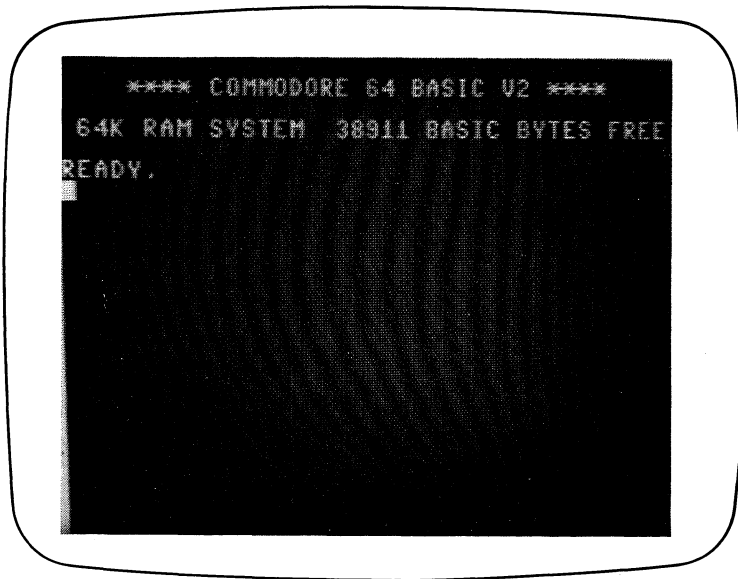
Wenn sie schon die einzelnen Komponenten Ihres Computersystems miteinander verbunden haben, und wenn Ihr System korrekt läuft, können Sie dieses Kapitel überschlagen. Wenn Sie jedoch Ihren Computer gerade gekauft haben und Sie wissen wollen, wie alles funktioniert und funktionieren sollte, bekommen Sie nun Hinweise, wie Sie Ihr System testen können.

Vergewissern Sie sich, daß Ihr Computer läuft. Verbinden Sie Ihren

Commodore 64 mit dem Sichtgerät; je nachdem ob Sie einen Monitor oder einen Fernsehapparat benutzen, nach den im vorhergehenden Abschnitt erläuterten Anweisungen. Verbinden Sie die Datassette oder die Floppy Disc mit dem Computer. Wenn Sie einen Drucker anschließen wollen, verbinden Sie ihn nicht mit dem Computer, sondern mit der freien Steckbuchse auf der Rückseite des Floppy Disc Laufwerks.

Beim Einschalten solch eines kompletten Computersystems müssen Sie die einzelnen Komponenten separat einschalten. Beginnen Sie dabei mit dem Einschalten des Commodore 64, und schalten Sie dann nacheinander die Datassette, die Floppy Disc, den Monitor und den Drucker ein. Nach dem Einschalten des Computers werden Sie auf dem Bildschirm folgenden Text sehen:

```
**** COMMODORE 64 BASIC V2 ****  
64K RAM SYSTEM 38911 BASIC BYTES FREE  
READY.
```



Unter dem Wort READY werden sie ein kleines blaues blinkendes Rechteck feststellen. Dieses Rechteck wird 'Cursor' genannt. Das Wort ist französischen Ursprungs und bezeichnet die laufende Eingabestelle.

Was bedeutet die ausgegebene Meldung?

Die erste Zeile sagt aus, daß es sich um BASIC V2 oder Version 2 der Programmiersprache BASIC handelt. 64 'K' (für Kilo oder 1000) RAM bedeutet, daß in dem System 64000 Zeichen RAM, also Random Access Memory (Direktzugriffsspeicher) verfügbar sind. Schließlich bedeutet '38911 BASIC BYTES FREE', daß 38911 Bytes frei verfügbar für BASIC-Programme sind. Warum können aber nur 38911 Bytes und nicht die gesamten 64 K des Arbeitsspeichers des Computers benutzt werden? Das liegt daran, daß der Computer den Rest selbst benötigt.

Wenn der Computer die Meldung '38911 BYTES FREE' nicht ausgibt, ist irgendetwas nicht in Ordnung und Sie sollten ihn zur Reparatur bringen.

Wenn der Cursor nicht erscheint und nicht blinkt, ist ebenfalls etwas nicht in Ordnung und der Rechner sollte überprüft werden.

Wenn Ihr Computer soweit in Ordnung ist, können Sie nun einen kleinen Test machen, indem Sie ein kurzes Programm eingeben. Haben Sie keine Angst, daß Sie noch nicht genug über Programmierung wissen. Für diesen kleinen Text brauchen Sie keine Einzelheiten zu kennen.

Die Tastatur des Commodore 64 unterscheidet sich geringfügig von der Tastatur einer normalen Schreibmaschine.

Zum einen gibt es mehr Tasten, zum anderen werden sämtliche Buchstaben nur groß geschrieben, wenn Sie den Computer einschalten. Wenn Sie die SHIFT-Taste drücken und einen Buchstaben schreiben, werden Sie sehen, daß ein ungewöhnliches Symbol auf dem Bildschirm erscheint. Stören Sie sich bitte aber im Moment nicht daran.

Geben Sie genau das nachstehende Programm ein. Nach dem Eintippen jeder Zeile müssen Sie bei der Eingabe des Programms die RETURN-Taste drücken. Erst dadurch wird dem Computer die in dieser Zeile enthaltene Information übergeben.

```
10 PRINT "DAS IST MEIN COMMODORE COMPUTER"  
20 GOTO 10
```

Die Zeile mit der Zeilennummer 10 bewirkt die Ausgabe des Textes DAS IST MEIN COMMODORE COMPUTER auf dem Bildschirm. Aufgrund der GOTO-Anweisung mit der Zeilennummer 20 wird die Anweisung mit der Zeilennummer 10 fortlaufend wiederholt.

Geben Sie jetzt das Wort RUN ein und drücken Sie dann die RETURN-Tas-

te. Damit wird dieser Befehl dem Computer übergeben und das Programm gestartet.

Der Satz DAS IST MEIN COMMODORE COMPUTER wird jetzt ganz schnell wiederholt ausgegeben. Wenn der Bildschirm vollständig gefüllt ist, sollte die letzte Zeile flimmern. Dieses bedeutet, daß eine Anzahl neuer Zeilen auf dem Bildschirm dargestellt wird. Obwohl der letzte Satz flimmert, sollte man ihn klar und einfach lesen können. Falls es auf dem Bildschirm 'schneit' oder sich andere Störungen zeigen, ziehen Sie den Händler zu Rate, bei dem Sie Ihren Computer gekauft haben. Bringen Sie ihn zur Reparatur oder lassen Sie sich einen neuen geben.

Es kann durchaus vorkommen, daß ein Computer mit ein paar Mängeln vom Fließband kommt. Ein häufig auftretendes Problem ist, daß es auf dem Bildschirm 'schneit', solange ein Programm läuft. Sie sollten sich aber mit einem derartigen Fehler nicht abfinden, sondern ein fehlerfrei funktionierendes Gerät verlangen.

Das obige Programm würde endlos weiterlaufen. Sie können das Programm beenden, indem Sie die RUN/STOP-Taste drücken. Diese Taste befindet sich auf der linken Seite der Tastatur. Auf dem Bildschirm sollten dann folgende Zeilen erscheinen:

```
BREAK IN LINE 10  
READY.
```

oder

```
BREAK IN LINE 20  
READY.
```

Der Cursor sollte nun wieder sichtbar sein und blinken.

Versuchen Sie nun, dieses Programm auf die Datassette zu speichern und wieder zu laden. Die Datassette ist eine Art Kassettenrecorder, den Sie von Commodore kaufen können. Der Anschluß eines einfachen Kassettenrecorders ist ohne Modifikationen leider nicht möglich. Um das Programm auf die Datassette zu speichern, müssen Sie folgenden Text eingeben:

```
SAVE "TEXT "
```

Drücken Sie nun die RETURN-Taste. Der Computer sollte jetzt die Meldung

PRESS RECORD & PLAY ON TAPE

auf dem Bildschirm ausgeben. Machen Sie einfach das, was Ihnen der Computer sagt. Drücken Sie die Tasten RECORD und PLAY auf der Datassette. Der Bildschirm wird dann gelöscht, und Sie erhalten einen leeren Bildschirm in einer hellblauen Farbe für ca. 20 Sekunden. Danach wird auf dem Bildschirm die Meldung READY. ausgegeben. Spulen Sie nun die Kassette zurück.

Wie schon gesagt, ist es nicht möglich, einen normalen Kassettenrecorder zu benutzen, falls Sie keinen Adapter haben. Die Datassette von Commodore hat eine spezielle Vorrichtung, mit der die Stellung der Tasten vom Programm abgefragt werden kann.

Nachdem Sie die Kassette zurückgespult haben, schalten Sie den Computer aus und kurz danach wieder an. Dadurch wird bewirkt, daß das kleine Programm, welches Sie vorhin eingetippt hatten, aus dem Speicher gelöscht wird. (Das ist nur eine Möglichkeit, den Speicher der Maschine zu löschen. Sie werden später bessere Möglichkeiten erfahren.) Geben Sie nun das Wort LOAD ein und drücken Sie die RETURN-Taste. Der Computer gibt auf dem Bildschirm folgende Meldung aus:

PRESS PLAY ON TAPE

Drücken Sie die Taste PLAY auf der Datassette. Der Bildschirm wird wieder für einige Sekunden gelöscht. Wenn sich die Datassette selbst ausgeschaltet hat, erscheint auf dem Bildschirm folgende Meldung:

FOUND TEST

Dies bedeutet, daß das Programm mit dem Namen TEST auf dem Band gefunden worden ist. Sie müssen jetzt das Programm von der Datassette in den Speicher laden. Dazu müssen Sie die Taste mit dem Commodore-Zeichen drücken. Sie finden diese Taste auf der linken unteren Seite der Tastatur. Der Inhalt des Bildschirms wird wiederum gelöscht und es erscheint die Meldung

LOADING

READY.

Geben Sie das Wort RUN ein und drücken Sie auf die RETURN-Taste. Das Programm läuft nun wie vorher ab. Da sämtliche gespeicherten Daten mit dem Ausschalten des Computers gelöscht wurden, ist dies ein Beweis dafür, daß ein Programm von der Datensette geladen und auf dieser wieder gespeichert werden kann. (LOAD = laden, SAVE = speichern)

Wenn Sie eine Floppy Disk besitzen, versuchen Sie, auf ihr das Programm zu speichern und wiederum zu laden. Stellen Sie das Laufwerk an und vergewissern Sie sich, daß keine Diskette eingelegt ist.

Zur Beachtung: Im Gegensatz zu vielen anderen Computermodellen kann es bei Ihrem VC 64 vorkommen, daß die auf einer Diskette gespeicherten Daten gelöscht werden, wenn das Laufwerk mit einer eingelegten Diskette an- oder ausgeschaltet wird.

Legen Sie eine neue Diskette in das Laufwerk ein. Schieben Sie die Diskette soweit ein, bis Sie Widerstand spüren. Sie können merken, wie die Diskette einrastet. Schließen Sie jetzt die Klappe an dem Diskettenlaufwerk.

Bevor Sie mit einer Diskette arbeiten können, müssen Sie diese initialisieren. Dadurch werden unsichtbare Spuren auf die Oberfläche der Diskette geschrieben. Es müssen nur Disketten initialisiert werden, die entweder neu sind, oder vollständig gelöscht werden sollen. Immer wenn eine Diskette gelöscht wird, werden sämtliche Daten oder Programme, die vorher auf dieser abgespeichert wurden, gelöscht.

Um eine Diskette zu initialisieren, tippen Sie folgende Befehle ein:

OPEN 1,8,15 (RETURN-Taste drücken)

PRINT #1, "N:Programmdiskette,PD" (RETURN-Taste drücken)

Jetzt leuchtet das rote Lämpchen auf der Vorderseite des Laufwerkes auf. Gleichzeitig ist ein knackendes Geräusch zu hören.

Es gibt zwei kleine Lämpchen auf der Vorderseite des Diskettenlaufwerkes. Die grüne Lampe zeigt an, daß das Gerät eingeschaltet ist. Die rote Lampe besitzt zwei Funktionen. Zunächst zeigt sie an, daß Daten zwischen Diskette und Computer übertragen werden. Wenn die rote Lampe jedoch blinkt, so ist das ein Zeichen dafür, daß etwas nicht in Ordnung ist. Der Grund dafür kann entweder ein Programmfehler Ihrerseits, oder ein Nichtfunktionieren des Computers sein.

Wenn die rote Lampe blinkt, sollten Sie es noch einmal versuchen und

sich vergewissern, daß der oben erwähnte Text korrekt eingegeben ist. Blinkt das rote Lämpchen nicht, haben Sie nun eine sachgemäß präparierte Diskette, mit der Sie arbeiten können. Tippen Sie jetzt:

```
SAVE"TEXT",8
```

ein und drücken Sie die RETURN-Taste. Innerhalb weniger Sekunden wird nun das Programm auf der Diskette gespeichert.

Wenn das Programm gespeichert ist, erlischt die rote Lampe. Nehmen Sie die Diskette aus dem Laufwerk heraus, schalten Sie den Computer aus und wieder an. Legen Sie die Diskette wieder in das Laufwerk ein. Danach geben Sie folgenden Text ein:

```
LOAD "$", 8 (RETURN-Taste drücken)
```

Sobald READY auf dem Bildschirm erscheint, geben Sie das Wort LIST ein. Nachdem Sie nun die RETURN-Taste gedrückt haben, sehen Sie Folgendes auf dem Bildschirm:

```
0 "Programmdiskette" PD 2A
1 "TEXT" PRG
663 BLOCKS FREE.
READY.
```

Dies ist das Inhaltsverzeichnis (Directory). Das Directory enthält den Namen der Diskette (Programmdiskette), die gespeicherten Programme (TEST) und gibt Ihnen den noch zur Verfügung stehenden Speicherplatz an. In dem vorliegenden Fall sind auf der Diskette noch 663 Blocks (das sind ca. 168 000 Bytes) frei. Dieser Platz kann zum Speichern weiterer Programme oder Daten benutzt werden. Soweit so gut. Geben Sie jetzt

```
LOAD "TEXT",8 (RETURN-Taste drücken)
```

ein. Unser Programm wird jetzt wieder von der Diskette in den Speicher des Computers geladen, und mit dem RUN-Befehl können Sie es wie oben bereits erwähnt starten.

Es ist wichtig, daß Sie diesen Test mit Ihrem Floppy Disk Laufwerk

durchführen. Ähnlich wie bei dem Computer selbst, ist es auch hier möglich, daß irgendein Fehler vorliegt. Um völlig sicher zu sein, sollten Sie diesen Test einige Male wiederholen.

Bis jetzt haben Sie den grundsätzlichen Weg kennengelernt, Programme auf der Datassette oder auf einer Diskette zu speichern und wieder zu laden. Wir haben damit ein kleines System zur ersten Überprüfung Ihres Computers erstellt. Natürlich kann das kein vollständiger Funktionstest sein. Wenn der Test funktioniert, wissen Sie nämlich nur, ob Ihr Computer korrekt Programme speichert und wieder lädt. Funktioniert alles so, wie es funktionieren sollte? Gut. Jetzt werden wir uns die Tastatur vornehmen.

Funktionen der einzelnen Tasten

Die Tastatur des Commodore 64 ist in ihrem Aufbau ein wenig ungewöhnlich. Mit jeder Taste können normalerweise drei verschiedene Funktionen erzielt werden. Diese Mehrfachbelegung der einzelnen Tasten kann etwas verwirrend sein, da nicht immer alle Tasten deutlich gekennzeichnet sind. Um mit der Tastatur und den Funktionen der einzelnen Tasten vertraut zu werden, sollten Sie sich ein wenig damit beschäftigen. Seien Sie nicht entmutigt, wenn Sie nicht alles auf einmal aufnehmen und behalten können. Sie brauchen



etwas Zeit, um sich an die einzelnen mannigfaltigen Funktionen der Tasten zu gewöhnen.

Die RETURN-Taste. Die RETURN-Taste bewirkt, daß eine eingegebene Zeile in den Hauptspeicher des Computers übernommen wird. Sie ist somit wahrscheinlich die wichtigste Taste der gesamten Tastatur. Bei einigen anderen Computern ist diese Taste mit ENTER bezeichnet. (Die deutsche Übersetzung von ENTER ist dem Sinne nach 'Eintreten'.) Somit bezeichnet dieses Wort vielleicht etwas genauer die Funktion dieser Taste. Der Name RETURN ist in Anlehnung an das Funktionieren einer Schreibmaschine entstanden. Nach dem Ende einer jeden Zeile mußte man nämlich bei den älteren Schreibmaschinen einen Wagenrücklauf geben. So ist auch der Name RETURN zu verstehen.

Die SHIFT-Taste. Mit Hilfe der SHIFT-Taste können bei einer Schreibmaschine Groß- und Kleinbuchstaben geschrieben werden. Für den Computer hat diese Taste dieselbe Funktion. Der Unterschied ist, daß die Tastatur des Computers nicht nur Groß- und Kleinbuchstaben aufweist. Wenn Sie die SHIFT LOCK-Taste drücken, haben Sie damit die SHIFT-Taste festgestellt. Sie haben dann dasselbe erreicht, als wenn Sie die SHIFT-Taste die ganze Zeit niederdrücken. Hierbei kann es Probleme geben. Um diese möglichst zu vermeiden, sollten Sie die SHIFT LOCK-Taste nur wenn unbedingt nötig benutzen. Um die SHIFT-Taste wieder in Normalstellung zu bringen, müssen Sie die SHIFT LOCK-Taste noch einmal drücken.

Die RUN/STOP-Taste. Durch die RUN/STOP-Taste kann ein laufendes Programm unterbrochen oder abgebrochen werden. Bei anderen Computern ist diese Taste mit BREAK (break = Bruch, Abbruch des Programms) bezeichnet. Werden die RUN/STOP und die SHIFT-Taste gleichzeitig gedrückt, können so Programme von der Datensette geladen und gestartet werden.

Groß-/Kleinschrift und Großschrift/Grafikmodus. Sie können Ihren Commodore auf zwei Arten betreiben: Im Groß-/Kleinschriftmodus arbeitet Ihr Commodore wie eine Schreibmaschine. Im Großschrift-/Grafikmodus dagegen können Sie Ihren Commodore dazu benutzen, spezielle Grafiksymbole auf dem Bildschirm auszugeben.

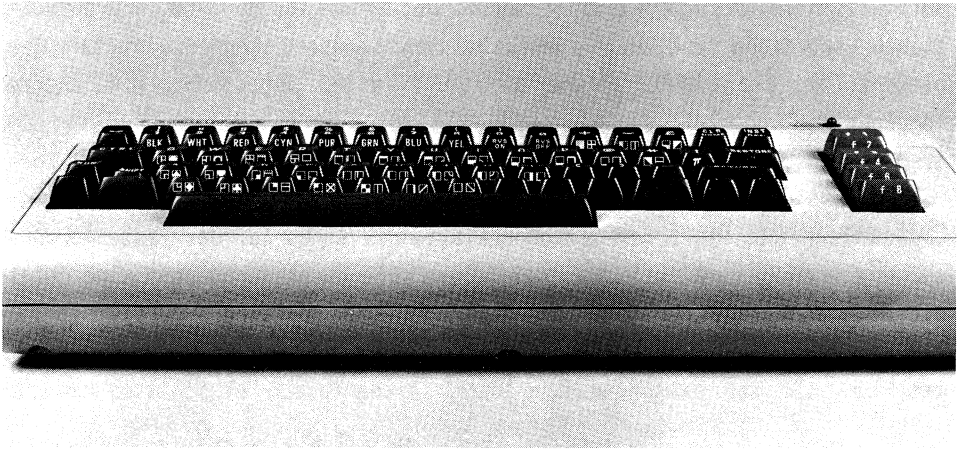
Die COMMODORE-Taste. Wenn Sie den Computer einschalten, so ist er automatisch im Großschrift/Grafikmodus. Das bedeutet, daß Sie in Großbuchstaben schreiben und alle Grafikzeichen benutzen können. Drücken Sie gleichzeitig die COMMODORE und die SHIFT-Taste. Jetzt wird eine andere Kombination von Zeichen eingeschaltet. Sie befinden sich nun in dem Groß-/Kleinschriftmodus. Die Bezeichnung ist eigentlich nicht ganz korrekt, da Sie auch hier einige Grafikzeichen zur Verfügung haben. Sie können nämlich mit Hilfe der Commodore-Taste die links auf der Tastenvorderseite stehenden Grafikzeichen erreichen. Die Buchstabentasten sind jetzt mit Kleinbuchstaben belegt, und Sie können wie bei einer Schreibmaschine die Großbuchstaben einschalten. Eine weitere Funktion der Commodore-Taste ist, daß Sie mit dieser weitere acht Zeichenfarben einschalten können.

Grafische Symbole. Sie werden sicherlich bemerkt haben, daß auf der Vorderseite von fast allen Tasten zwei kleine Rechtecke zu finden sind. Jedes dieser Rechtecke enthält ein spezielles grafisches Symbol. Drücken Sie im Grafikmodus die Taste mit dem W. Es wird ein W ausgegeben. Drücken Sie nun die SHIFT-Taste und die W-Taste gleichzeitig. Sie sehen nun ein Symbol, welches an den Buchstaben O erinnert. Es ist aber nicht der Buchstabe O, sondern ein grafisches Symbol. Drücken Sie nun die Commodore-Taste und dann die W-Taste. Sie sehen jetzt ein Symbol, welches einem um 90 Grad gedrehtem T ähnelt. Sie sehen, daß diese Taste mit einer dreifachen Funktion belegt ist.

Drücken Sie nun die SHIFT und die Commodore-Taste zusammen und wiederholen Sie den obigen Vorgang. Sie werden sehen, das einmal ein kleines w, dann ein großes W und schließlich das grafische Symbol ausgegeben wird.

Die CTRL-Taste. Wir haben vorhin erfahren, daß die Commodore-Taste eine ähnliche Funktion wie die SHIFT-Taste besitzt. Eine weitere Taste mit ähnlicher Funktion ist die CTRL-Taste. CTRL ist eine Abkürzung für 'control'.

Reverse Modus. Eine Möglichkeit, die CTRL-Taste zu benutzen ist die folgende: Drücken Sie die Taste mit der Zahl neun, während Sie die CTRL-Taste gedrückt halten. Tippen Sie nun irgendeinen Text ein. Sie werden feststellen, daß jetzt alles 'umgekehrt' geschrieben wird. 'Umgekehrt' bedeutet hierbei, daß nicht mehr 'schwarz auf weiß', sondern 'weiß auf



schwarz' geschrieben wird. Dieser Reverse-Modus kann sehr nützlich sein, wenn bestimmte Zeichen hervorgehoben werden sollen. Sie können den Reverse-Modus ausschalten, wenn Sie bei gedrückter CTRL-Taste die Taste mit der Ziffer 0 drücken. Auf der Vorderseite der Tasten mit der Ziffer 9 und 0 sehen Sie die Kennzeichnung RVS ON, bzw. RVS OFF. RVS ON bedeutet 'reverse on'; RVS OFF bedeutet 'reverse off'.

Farben. Drücken Sie nun die CTRL-Taste und irgendeine Taste mit den Ziffern 1-8. Beobachten Sie dabei den Cursor. Sie werden bemerken, daß der Cursor seine Farbe ändert. Nachstehend finden Sie eine Liste der Farben, die Ihnen bei Ihrem Commodore zu Verfügung stehen. CTRL-2 bedeutet, daß Sie die CTRL-Taste und die Taste mit der Ziffer 2 gleichzeitig drücken müssen. COMM-3 bedeutet beispielsweise, daß Sie die Taste mit dem Commodore-Zeichen und die Taste mit der Ziffer 3 gleichzeitig drücken müssen.

<u>Taste</u>	<u>Farbe</u>
CTRL-1	schwarz
CTRL-2	weiß
CTRL-3	rot
CTRL-4	türkis
CTRL-5	violett
CTRL-6	grün

CTRL-7	blau
CTRL-8	gelb

Obgleich es auf der Tastatur nicht aufgeführt ist, können acht weitere Farben direkt durch die Kombination der COMMODORE-Taste und den Ziffern von 1-8 erzeugt werden:

Taste	Farbe
COMM-1	orange
COMM-2	braun
COMM-3	hellrot
COMM-4	grau 1
COMM-5	grau 2
COMM-6	hellgrün
COMM-7	hellblau
COMM-8	grau 3

Erschrecken Sie nicht, wenn Sie das, was Sie eingetippt haben, unter Umständen nicht lesen können. Sie haben dann wahrscheinlich eine Farbe gewählt, die nicht genügend zum Hintergrund kontrastiert. Viele Commodore 64 Computer (besonders die älteren Datums) sind in ihren Farbkombinationen eingeschränkt. Sie sollten selbst einmal versuchen herauszufinden, welche Farben und welcher Hintergrund gut zusammen passen (d.h. gut kontrastieren).

Die Funktionstasten. Schauen Sie sich einmal die vier größeren Tasten auf der rechten Seite der Tastatur an. Diese Tasten sind mit f1, f3, f5, f7 und die Vorderseite dieser Tasten mit f2, f4, f6 und f8 bezeichnet. Die auf der Vorderseite der Tasten angegebenen Bezeichnungen werden durch Drücken der SHIFT-Taste angesprochen. Diese Funktionstasten können von einem Programm abgefragt werden. Sie sind sehr nützlich und können für die unterschiedlichsten Dinge verwendet werden.

Die RESTORE-Taste. Die RESTORE-Taste auf der rechten Seite der Tastatur besitzt eine sehr wichtige Funktion. Die gleichzeitige Betätigung der RUN/STOP- und der RESTORE-Taste bewirkt die Unterbrechung eines Programms. Dasselbe Ergebnis erhalten Sie, wenn Sie nur die RUN/STOP-Taste drücken.



Der Computer wird in den sogenannten 'Grundzustand' versetzt.

Wie beim vorher beschriebenen Aus- und Einschalten des Computers, wird das Programm unterbrochen, jedoch bleibt dieses Programm im Speicher erhalten. Sie können das unterbrochene Programm also direkt wieder mit RUN erneut starten. Das Ein- und Ausschalten des Computers zur Unterbrechung eines Programms wird sehr oft als 'Kaltstart' bezeichnet. Dieser Name resultiert aus der Tatsache, daß das im Speicher befindliche Programm gelöscht wird. Der Abbruch eines Programms mit RESTORE wird dagegen als 'Warmstart' bezeichnet, da das Programm im Speicher erhalten bleibt.

Die Editing-Tasten. Die Editing-Tasten des Commodore 64 gehören zu den leistungsfähigsten Tasten der gesamten Tastatur. Commodore Computer hatten schon immer vorzügliche Korrekturmöglichkeiten, die das Schreiben von Programmen zu einem Kinderspiel werden lassen. Außer zum Korrigieren eingegebener Programme können die Editing-Tasten in besonderer Weise beim Programmieren benutzt werden. Nachfolgend sind einige Bemerkungen zu den Editing-Funktionen dieser Tasten aufgeführt.

Die CRSR-Tasten. Auf der unteren rechten Seite der Tastatur befinden sich die CRSR-Tasten (CRSR = CuRSor), die es ermöglichen, den Cursor in eine gewünschte Position auf dem Bildschirm zu bringen. Entsprechend den auf den Tasten abgebildeten Pfeilen, läßt sich der Cursor nach oben, bzw. nach unten und nach rechts oder links bewegen. Drücken Sie die linke

Cursor-Taste (für vertikale Bewegungen). Der Cursor bewegt sich nun nach unten. Zusammen mit der SHIFT-Taste kombiniert, erreichen Sie durch diese Taste eine Bewegung nach oben. Die rechte CRSR-Taste (für vertikale Bewegungen) erzielt Bewegungen nach rechts bzw. (zusammen mit der SHIFT-Taste) nach links. Sollte die CRSR-Taste nicht wie oben erläutert funktionieren, schauen Sie lieber noch einmal nach, ob Sie die SHIFT LOCK-Taste ausgerastet haben.

Die CLR/HOME-Taste. Die Editing-Tasten auf der oberen rechten Seite der Tastatur sind mit CLR/HOME und INST/DEL gekennzeichnet. HOME bezeichnet die 'Home-Position' des Cursors, die sich in der linken oberen Ecke des Bildschirms befindet. Die CLR/HOME-Taste bringt also den Cursor wieder in seine Home-Position. CLR bedeutet: Löschen des Bildschirms. Die CLR/HOME- und die SHIFT-Taste zusammen löschen den Bildschirm und bringen den Cursor in seine Home-Position zurück. Sollten Sie versehentlich die SHIFT- und die CLR/HOME-Taste drücken, bevor Sie die RETURN-Taste gedrückt haben, geht Ihnen die zuletzt eingegebene Zeile wieder verloren.

Die INST/DEL-Taste. Diese Bezeichnungen werden von DElete (= löschen) und INSert (=einfügen) abgeleitet. Geben Sie probeweise eine beliebige Zeile ein und drücken Sie dann die INST/DEL-Taste. Sooft diese Taste gedrückt wird, bewegt sich der Cursor um eine Position zurück und löscht das Zeichen an der betreffenden Stelle. Zusammen mit der SHIFT-Taste können Zeichen in eine bereits geschriebene Zeile eingefügt werden. Geben Sie wiederum irgendeinen Text ein. Bewegen Sie mit der entsprechenden CRSR-Taste den Cursor in die Mitte der Zeile. Drücken Sie die SHIFT-Taste und ein paar Mal die INST/DEL-Taste. Alles, was rechts vom Cursor steht, wird weiter nach rechts verschoben. Zurück bleibt an diesen Stellen ein Leerraum, den Sie nach Belieben nutzen können.

Diese acht Funktionen bieten somit eine einzigartige und äußerst bequeme Möglichkeit zur Dateneingabe. Aller Anfang ist bekanntlich schwer, und deshalb müssen Sie sich an dieses Editing-System erst gewöhnen. Danach ist aber auch für Sie die Text- oder Dateneingabe zu einem Kinderspiel geworden. Besonders für Textverarbeitung ist es wichtig, die Funktionen dieser Tasten zu beherrschen, da sie in den meisten Programmen verwendet werden.

Repeat-Tasten. Bei dem Commodore 64 besitzen vier Tasten eine Wiederholungsfunktion. Dazu gehören die beiden CRSR-Tasten für vertikale- und horizontale Bewegungen, die INST/DEL-Taste und die Leertaste, die sich ganz unten auf der Tastatur befindet. Diese Tasten wurden mit einer Wiederholungsfunktion versehen, da sie bei der Eingabe eines Programms oder bei der Eingabe von Daten häufig verwendet werden. Wiederholungsfunktion bedeutet, daß die Funktion dieser Tasten fortlaufend ausgeführt wird, solange Sie gedrückt bleiben. Machen Sie sich mit all diesen Tasten vertraut, indem Sie einen Text eintippen und diesen nach Belieben korrigieren und ändern.

Das Schreiben mit der QWERTY Tastatur

Viele Menschen haben so ihre Schwierigkeiten mit einer Schreibmaschinentastatur. Ihr Commodore Computer hat, wie fast alle anderen Personal Computer auch, eine sogenannte QWERTY-Tastatur, die nach der Abfolge ihrer Buchstaben der obersten Buchstabenreihe benannt ist. Die gesamte obere Reihe besteht aus folgenden Buchstaben: QWERTYUIOP. Eine Legende besagt, daß diese Reihenfolge so festgelegt wurde, damit ein Verkäufer, der die ersten Schreibmaschinen von 1873 verkaufte, das Wort 'typewriter' (= Schreibmaschine) nur auf der obersten Buchstabenreihe schreiben konnte. (Es geht tatsächlich).

Wirklicher Grund für diese etwas schwerfällige Anordnung der Tastatur ist, daß die Konstrukteure damit eine Verlangsamung der Schreibgeschwindigkeit erreichen wollten. Sie waren der Ansicht, daß die Typisten durch die weite Trennung der am häufigsten verwendeten Tasten der Tastatur dazu gezwungen würden, so langsam zu schreiben, daß sich die frühen, etwas primitiven Druckmechanismen nicht verhaken. Heute, hundert Jahre später, benutzen wir noch immer die veraltete und unkomfortable QWERTY-Tastatur. Gerade nach der Erfindung der Personal Computer hätten die alten Erfinder der QWERTY-Tastatur allen Grund dazu, über uns zu lachen. Es gibt heute mehr QWERTY-Tastaturen denn je. Warum hat man eigentlich die Anordnung dieser Buchstaben nicht geändert? Nun, manche haben es versucht. Beispielsweise wurde auf den kleinen Taschencomputern, die in den letzten Jahren verkauft wurden, die herkömmliche Buchstabenanordnung durch eine alphabetische ersetzt. Dann stellte sich heraus, daß man diese Tastatur

noch schlechter handhaben konnte, als die alte QWERTY-Tastatur. Als Ergebnis der Nachforschungen zweier Jahrzehnte wurde 1932 eine weitere Variante angeboten: Die DSK (für Dvorak Simplified Keyboard)-Tastatur. Hier wurden die am häufigsten gebrauchten Buchstaben auf der 'Home row' zusammengelegt. Man behauptete, daß nun die 3 000 am häufigsten verwendeten Wörter der englischen Sprache auf dieser einzigen Zeile geschrieben werden könnten. Im letzten Jahrzehnt versuchte man sogar, eine Einhand-Tastatur einzuführen, bei welcher mehrere Finger gleichzeitig benötigt wurden, um einen einzelnen Buchstaben zu schreiben.

Bis heute herrscht die QWERTY-Tastatur unangefochten vor und andere Versionen sind schon längst wieder vergessen. Einige Schreibmaschinen sind mit genannten alternativen Tastaturen ausgestattet, aber bei Personal Computern findet man dieses nicht. Um mit Ihrem Personal Computer arbeiten zu können, müssen Sie sich mit dieser unkomfortablen Tastatur auseinandersetzen. Ein Vorteil dabei ist, daß Sie fast mühelos das Schreiben auf einer Tastatur lernen und Sie mit dieser zunehmend vertraut werden. Nachdem Sie diesen Abschnitt gelesen haben, wundern Sie sich vielleicht, warum sich überhaupt noch jemand eine Schreibmaschine zulegt.

3 Einige wissenswerte Details

Unabhängig davon, ob Sie Ihren Computer selber programmieren wollen oder ob Sie stattdessen auf kommerziell erhältliche Programme zurückgreifen, müssen Sie noch einige grundlegende Dinge für den Umgang mit dem Computer wissen. Sie müssen lernen, wie Programme von einer Datensette oder Diskette geladen, gespeichert und gestartet werden. Sie sollten ebenfalls über einige Besonderheiten beim Editieren von BASIC-Programmen Bescheid wissen.

Die RETURN-Taste

Im weiteren Verlauf dieses Buches werden Sie häufig aufgefordert, Beispielpprogramme zur Demonstration einzutippen. Sie sollten dabei beachten, daß erst durch Drücken der RETURN-Taste die entsprechenden Zeilen dem Speicher des Computers übergeben werden. Wird die RETURN-Taste nicht gedrückt, wartet der Computer in der Annahme, daß weitere Eingaben folgen werden. Nach der Eingabe jeder BASIC-Programmzeile, nach der Korrektur einer Zeile oder für den Fall, daß ein Befehl direkt von der Tastatur dem Computer übergeben werden soll, muß die RETURN-Taste gedrückt werden.

Externe Speicher

Normalerweise werden Computerprogramme über die Tastatur in den Speicher des Rechners eingegeben. Es wäre aber eine unvorstellbare Zeitverschwendung, würde vor jeder Benutzung das betreffende Programm erst wieder neu eingetippt werden. Aus diesem Grunde ist es möglich, Programme auf einem externen Speicher zu speichern. Solch ein externer Speicher kann entweder die Datensette oder eine Floppy Disk sein. Die Floppy Disk wird auch oft Diskette genannt.

Die Programme werden mit dem SAVE-Befehl auf einen externen Speicher gespeichert, und mit dem LOAD-Befehl wieder in den Speicher des Computers geladen. Wenn Sie ein Programm speichern, wird auf der Datensette oder der Diskette eine Kopie von dem Programm gemacht, welches sich gerade im Speicher des Computers befindet. Nach dem Speichern des Programms auf einen Extern-Speicher existiert das Programm deshalb weiterhin in dem Speicher des Computers. Wird das Programm von einem externen Speicher in den Speicher des Computers mit dem LOAD-Befehl geladen, so wird auch hier wieder eine Kopie des Programms gemacht. Das Programm existiert also nach dem Laden weiterhin auf der Diskette bzw. auf der Datensette.

Der Speicher des Computers wird RAM (Random Access Memory) genannt. Dieser Speicher ist nur ein temporärer Speicher, da sämtliche Informationen (Programme oder Daten) verlorengehen, sobald der Strom - sei es auch nur für den Bruchteil einer Sekunde - abgeschaltet wird.

Durch den LOAD-Befehl wird der Inhalt des RAM-Speichers mit dem neuen Programm überschrieben. Es ist also nicht möglich, durch die zweimalige Anwendung des LOAD-Befehls ein Programm einem anderen hinzuzufügen.

Das Laden von der Datensette

Für die nachstehenden Tests können Sie entweder die Kassette nehmen, die Sie in dem vorigen Abschnitt präpariert haben, oder Sie können eine andere Kassette benutzen, die irgendein aufgezeichnetes Programm enthält.

Der LOAD-Befehl kann mit oder ohne Programmnamen benutzt werden. Beispiel:

LOAD (RETURN-Taste drücken)

oder

LOAD "PROGRAMM EINS" (RETURN-Taste drücken)

Bei der Ausführung der ersten Version erhalten Sie folgende Meldung auf dem Bildschirm:

PRESS PLAY ON TAPE

Nachdem Sie die Taste PLAY der Datassette gedrückt haben, wird der Bildschirm vorübergehend gelöscht. Danach erscheint die Meldung

FOUND

Das bedeutet, daß der Computer auf der Datassette das entsprechende Programm gefunden hat. Wurde dem Programm ein Name gegeben, so erscheint dieser auf dem Bildschirm. Jetzt sind Sie wieder an der Reihe. Drücken Sie die COMMODORE-Taste auf der unteren linken Seite der Tastatur. Haben Sie nach LOAD keinen Namen angegeben, so wird das erste Programm, welches sich auf der Kassette befindet, geladen. Haben Sie jedoch einen Namen angegeben, wird der Computer das gewünschte Programm auf der Kassette heraussuchen.

Ist das Programm in den Speicher des Computers geladen worden, erscheint auf dem Bildschirm das Wort

READY.

Nachdem Sie das Wort RUN eingegeben und die RETURN-Taste gedrückt haben, wird das Programm ausgeführt.

Eine weitere Methode, das erste Programm von einer Kassette zu laden ist, die SHIFT- und gleichzeitig die RUN/STOP-Taste zu drücken. Daraufhin erscheinen sofort die Worte LOAD und PRESS PLAY ON TAPE auf dem Bildschirm. Auch nachdem das Wort FOUND auf dem Bildschirm erschienen ist, müssen Sie die COMMODORE-Taste gedrückt halten. In diesem Fall wird das Programm sofort automatisch gestartet.

Wenn Sie LOAD zusammen mit einem Programmnamen verwenden, wird nur das genannte Programm geladen. Der Programmname muß immer in Anführungs-

zeichen stehen.

Gehen wir einmal von einem kleinen Beispiel zur Veranschaulichung aus. Angenommen, drei Programme "PROGRAMM EINS", "PROGRAMM ZWEI" und "PROGRAMM DREI" sind auf derselben Seite einer Kassette gespeichert. Durch den Befehl LOAD "PROGRAMM DREI" wird nur das Programm mit diesem Namen geladen.

Die Commodore Computer arbeiten nach der Technik des 'pattern matching' um das Programm, das sie laden sollen, zu identifizieren. Wenn Sie nur den ersten Buchstaben des Programmnamens (z. B. LOAD P) eingeben, lädt der Computer jedes Programm, das mit diesem Buchstaben beginnt. In unserem Beispiel würde irgendein Programm, das mit P beginnt, geladen werden: "PROGRAMM EINS", "PROGRAMM ZWEI", "PROGRAMM DREI" oder auch "PING PONG" (falls dieses Programm auf dem Band gespeichert ist).

Wenn Sie Programme mit der Technik des 'pattern matching' laden wollen, dürfen Sie das letzte Anführungszeichen nicht benutzen.

```
LOAD "PRO
```

```
LOAD "PROGRAMM"
```

Mit dem ersten Befehl wird jedes Programm geladen, dessen Programmname mit den Buchstaben PRO beginnt. Mit dem zweiten Befehl wird nur das Programm mit dem Namen PROGRAMM geladen.

Wir werden später einmal einen anderen Befehl verwenden, um ein Programm zu laden:

```
LOAD "PROGRAMM NAME" ,1,1
```

Was bedeutet das? Die erste Ziffer sagt dem Computer, von welcher Einheit das Programm geladen werden soll. In dem vorliegenden Fall handelt es sich um die Datensette als Externspeicher, die immer mit der Einheitennummer 1 bezeichnet wird. Näheres darüber werden Sie in einem der nächsten Abschnitte erfahren.

Die zweite Ziffer (in diesem Fall ebenfalls eine 1) teilt dem Computer mit, in welchen Speicherbereich des RAMs das Programm geladen werden soll.

Wenn das Laden des Programms nicht richtig funktioniert

Es gibt eine Menge Gründe dafür, warum ein Programm nicht richtig geladen wird. Nachfolgend sind die wichtigsten aufgeführt.

Grund 1: Irgendetwas stimmt in Ihrer Datensette nicht. Entweder ist das Verbindungskabel defekt oder irgendein elektronischer Schaltkreis in der Datensette funktioniert nicht richtig. Ist der Fehler sichtbar, wenden Sie sich bitte an Ihren Händler.

Grund 2: Die Datensette arbeitet zwar, aber sie ist nicht richtig justiert. Das ist oft der Fall, wenn Sie Programme laden können, die Sie gespeichert haben, aber beim Laden von Programmen, die irgendjemand auf seiner Datensette gespeichert hat, Schwierigkeiten haben. In diesem Fall müssen den Schreib-/Lesekopf der Datensette justieren. Sie sollten diese Arbeit aber unbedingt einem Fachmann überlassen.

Grund 3: Obwohl der Computer und die Datensette einwandfrei arbeiten, kann es Schwierigkeiten beim Laden eines Programms geben, weil der Programmierer sein Werk vor unberechtigtem Kopieren schützen wollte.

Grund 4: Achten Sie unbedingt darauf, daß das Magnetband in Ordnung ist. Es kann nämlich sein, daß Sie einen Fehler der Datensette oder des Computers vermuten, obwohl lediglich die Kassette beschädigt ist.

Das Speichern von Programmen auf Band

Der Befehl `SAVE` wird normalerweise immer zusammen mit einem Programmnamen benutzt.

```
SAVE "PROGRAMM NAME"
```

Durch diesen Befehl wird ein Programm auf die Datensette gespeichert. Wird hinter `SAVE` kein Name angegeben, so wird das Programm zwar gespeichert, aber es kann später beim Laden nicht mehr einwandfrei identifiziert werden. Die Befehle `SAVE` und `LOAD` können nicht für solche Programme verwendet werden, die gegen unberechtigtes Kopieren geschützt worden sind oder die nicht in BASIC geschrieben wurden.

Nachdem Sie `SAVE` und den Programmnamen eingegeben haben (Vergessen Sie nicht die `RETURN`-Taste), erhalten Sie auf dem Bildschirm folgende Meldung:

PRESS RECORD AND PLAY ON TAPE

Wenn Sie diese beiden Tasten auf Ihrer Datassette drücken, wird der Bildschirm solange gelöscht, bis das Programm auf der Kassette gespeichert ist.

Denken Sie unbedingt daran, beide Tasten PLAY und RECORD zu drücken, wenn Sie auf Band speichern wollen. Falls Sie dieses nicht beachten, wird Sie der Computer nicht darauf aufmerksam machen können, und er reagiert so, als wenn Sie nur die PLAY-Taste gedrückt hätten. Damit wird das Programm natürlich nicht gespeichert.

Wollen Sie das Programm vor unbeabsichtigtem Löschen schützen, entfernen Sie die Plastikabdeckung an der Aussparung in der hinteren Ecke der Kassette. Dadurch wird verhindert, daß Sie RECORD und PLAY gleichzeitig benutzen können. Falls Sie doch später einmal das Band überschreiben wollen, können Sie mit etwas Klarsichtfolie diese Aussparung wieder abdecken.

Überprüfung des Speichervorgangs

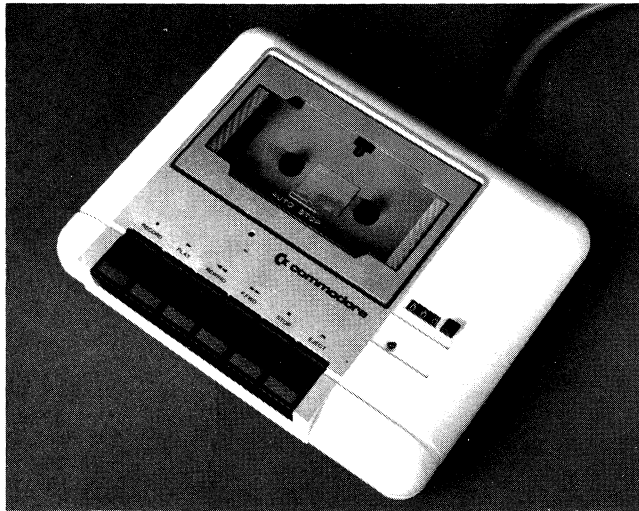
Jetzt möchten Sie sicherlich gerne wissen, ob Sie Ihr Programm auch richtig abgespeichert haben. Mit dem VERIFY-Befehl können Sie dieses nachprüfen. Dadurch wird der Inhalt des Bandes mit dem Inhalt des im RAM vorhandenen Speichers verglichen. Der VERIFY-Befehl zusammen mit einem Programmnamen bewirkt, daß das entsprechende Programm auf der Datassette gesucht wird. Wird nur der Befehl VERIFY eingegeben, wird lediglich das erste Programm auf dem Band überprüft.

Ist ein Programm ordnungsgemäß gespeichert, erscheint auf dem Bildschirm die Meldung "OK", anderenfalls erhalten Sie die Meldung

? VERIFYING ERROR

Das bedeutet, daß die Kopie des Programms auf Kassette nicht dem im Computer gespeicherten Programm entspricht. Verantwortlich dafür ist eventuell eine schlechte Bandqualität.

Der VERIFY-Befehl kann ebenfalls dazu benutzt werden, eine bestimmte Stelle auf dem Band herauszufinden. Angenommen, Sie haben zwei Programme "PROGRAMM EINS" und "PROGRAMM ZWEI" auf derselben Seite des Bandes ge-



speichert. Jetzt haben Sie nachträglich noch einige Änderungen in PROGRAMM ZWEI vorgenommen und möchten diese Version wieder auf Band speichern. In diesem Fall müßten Sie erreichen, daß der Schreib-/Lesekopf hinter das PROGRAMM ZWEI auf dem Band positioniert wird.

Wenn Sie versuchen, dieses zu erreichen, indem Sie PROGRAMM EINS laden, würde das im RAM-Speicher des Computers enthaltene Programm überschrieben werden. Damit das nicht geschieht, brauchen Sie PROGRAMM EINS nur zu verifizieren. In diesem Fall läuft das Magnetband bis zu Beginn des "PROGRAMM ZWEI" vor, ohne daß das im Speicher des Computers befindliche Programm gelöscht wird. Sie erhalten zwar die oben angegebene Fehlermeldung "? VERIFYING ERROR", aber Sie brauchen sich daran nicht zu stören. Geben Sie einfach danach den Befehl SAVE PROGRAMM ZWEI an, und die alte Version des Programms zwei wird auf der Kassette durch das neue "PROGRAMM ZWEI" überschrieben. Achten Sie aber unbedingt darauf, daß das neue Programm nicht wesentlich länger als das alte wird, da sonst ein nachfolgendes Programm überschrieben werden würde.

Der Start eines Programms

Mit dem RUN-Befehl kann ein Programm gestartet werden. Die meisten Programme können mit der RUN/STOP-Taste unterbrochen werden.

Wenn Sie nach dem Unterbrechen des Programms den Befehl erneut eingeben, so beginnt das Programm mit der Ausführung wieder von Anfang an. Das bedeutet, daß sämtliche Rechnungen erneut durchgeführt werden.

In vielen Fällen ist dies jedoch nicht erwünscht. Man möchte vielmehr ein Programm kurzzeitig unterbrechen, um Zeit zu haben, sich die Ergebnisse genau anzusehen. Dann möchte man an der Stelle fortfahren, an der das Programm unterbrochen worden ist. Dies kann mit dem Befehl CONT geschehen. CONT steht für CONTinue (= fortfahren, weitermachen). Tippen Sie den Befehl direkt ein und drücken Sie danach die RETURN-Taste.

Sie brauchen aber ein Programm auch nicht beginnend mit der ersten Zeilennummer auszuführen. Wenn Sie den Befehl RUN geben, wird normalerweise das Programm beginnend mit der Zeile, die die kleinste Zeilennummer trägt, ausgeführt. Wenn Sie jedoch hinter dem Befehl RUN eine Zeilennummer angeben, beginnt der Computer die Ausführung des Programms mit der Anweisung, die diese Zeilennummer trägt.

Der Diskettenspeicher

Mit dem Commodore 1541 Diskettenspeicher können wie mit der Datensette Daten und Programme gespeichert werden. Die Information wird hier allerdings nicht auf einem Magnetband, sondern auf einer magnetisierbaren Scheibe gespeichert. Diese magnetisierbare Scheibe wird Diskette genannt. Der große Vorteil des Diskettenspeichers ist, daß er erheblich schneller als der Kassettenspeicher ist. Das Laden eines Programms von einer Diskette geht ungefähr siebenmal schneller als das Laden eines Programms von einer Kassette. Es kann außerdem auf die einzelnen Informationen, die auf einer Diskette gespeichert worden sind, direkt zugegriffen werden. Ein direkter Zugriff bedeutet, daß die einzelnen Informationen nicht wie bei der Kassette in der Reihenfolge abgerufen werden müssen, in der sie auch

gespeichert worden sind.

Auf der rechten Seite der Diskette finden Sie in dem Plastikumschlag eine Ausparung. Wenn die Ausparung vorhanden ist, bedeutet dieses, daß Sie Daten von der Diskette lesen und Daten auf die Diskette schreiben können. Wenn die Ausparung jedoch abgedeckt ist (beispielsweise durch einen Plastikfilm) ist die Diskette schreibgeschützt (writeprotected). Das bedeutet, daß Daten von der Diskette geladen, aber nicht auf dieser gespeichert werden können.

Neue Disketten müssen erst initialisiert werden, bevor sie benutzt werden können. Kaufen Sie sich Disketten mit der Kennzeichnung 'single-sided single density'. Bei Disketten dieser Art kann nur eine Seite der Diskette für das Speichern von Daten verwendet werden. Einige Diskettenlaufwerke erlauben es, Daten beidseitig zu speichern. Bei dem Laufwerk 1541 ist das jedoch nicht der Fall. 'Single density' bedeutet, daß diese Diskette mit einfacher Schreibdichte beschrieben wird. Hochwertige Diskettenspeicher erlauben es, sehr große Datenmengen abzuspeichern. In diesem Fall müßten Sie auch hochwertige Disketten mit der Bezeichnung 'double density' benutzen.

Eine andere Bezeichnungsart für Disketten ist 'soft sector' oder 'hard sector'. Sie brauchen sich an dieser Unterscheidung nicht zu stören, denn sowohl die eine als auch die andere Sorte wird mit Ihrem 1541 Diskettenlaufwerk funktionieren. Bei der Auswahl Ihrer Disketten brauchen Sie nicht einen bestimmten Hersteller zu beachten. Disketten sämtlicher namhafter Hersteller können benutzt werden.

Das Initialisieren einer Diskette wird auch 'formatieren' genannt. Wenn eine Diskette formatiert werden muß, sind einige Dinge zu tun. Sie müssen der Diskette einen Namen und eine Identifikation geben, die zwei Zeichen lang sein darf. Ein Teil der Diskette wird für das sogenannte Directory (Inhaltsverzeichnis) benutzt. In diesem Directory werden die Namen der Programme oder Datensätze gespeichert. Die Oberfläche der Diskette ist in 35 Spuren (track) eingeteilt. Jede dieser Spuren wird zur Speicherung von Informationen benutzt. Eine besondere Spur, die Spur mit der Nummer 18, wird für das Directory benutzt. Jede Spur ist wiederum in Sektoren unterteilt, die jeweils eine Speicherkapazität von 256 Bytes haben. Diese Sektoren werden auch Blocks genannt. Die gesamte Speicherkapazität einer Diskette beträgt 664 Blocks oder ca. 170 000 Bytes.

Sie müssen beachten, daß Sie keine Diskette mit Ihrem Commodore 64

benutzen können, die auf dem Computer eines anderen Herstellers (z. B. Apple oder Atari) präpariert wurde und zwar auch dann nicht, wenn sie BASIC-Programme enthält, die auf Ihrem Commodore 64 ausgeführt werden können. Jeder Computerhersteller benutzt nämlich eine andere Methode, die Daten auf der Diskette zu speichern. Dieses trifft übrigens auch für das Speichern von Daten auf einem Kassettenrecorder zu.

Die Disketten sollten sehr behutsam gehandhabt werden, da sonst die gespeicherte Information verlorengehen kann. Nachfolgend ein paar Tips zur sachgemäßen Handhabung und Aufbewahrung von Disketten:

- Berühren Sie niemals die Oberfläche der Diskette. Sie dürfen selbstverständlich den Umschlag berühren; nicht aber die magnetisierbare Schicht, die durch die Aussparungen in dem Umschlag zu sehen ist.
- Stecken Sie die Diskette nach Gebrauch immer wieder in den Schutzumschlag zurück. Bewahren Sie die Diskette in einer staubdichten Box auf.
- Setzen Sie die Diskette nicht stark magnetischen Feldern aus.

Die Benutzung des Diskettenspeichers

An der Vorderseite des Diskettenspeichers sind zwei farbige Lämpchen zu sehen. Es handelt sich hierbei nicht um reguläre Lampen, sondern um sogenannte LEDs (Light Emitting Diodes = Leuchtdioden). Die grüne Leuchtdiode zeigt an, daß das Diskettenlaufwerk eingeschaltet ist. Die rote Leuchtdiode erfüllt zwei Funktionen: Sie leuchtet entweder permanent, oder sie blinkt, wenn ein Programm gespeichert bzw. geladen wird. Wenn die rote Leuchtdiode blinkt, bedeutet dieses, daß ein Fehler (ERROR) aufgetreten ist. Beobachten Sie deshalb immer die rote Leuchtdiode, wenn Sie das Diskettenlaufwerk benutzen.

Die vorher erläuterten Befehle LOAD und SAVE können ebenfalls im Zusammenhang mit dem Diskettenlaufwerk benutzt werden. Wie bereits erwähnt, muß bei dem LOAD- bzw. SAVE-Befehl die Einheitennummer angegeben werden. Alle Einheiten, wie die Datassette, ein Diskettenlaufwerk, ein Drucker oder Modems werden mit Einheitennummern bezeichnet. Diese Einheitennummern sind festgelegt. Die Datassette hat immer die Einheitennummer 1. Der Drucker hat die Einheitennummer 4 und das Diskettenlaufwerk die Einheitennummer 8.



Das Disketten-Directory

Wenn Sie in einem Buch nach einem bestimmten Kapitel suchen, schauen Sie sich das Inhaltsverzeichnis an. Ähnliches geschieht bei einer Diskette. Sie können den Inhalt einer Diskette anhand des Directorys (= Inhaltsverzeichnis) ablesen. Das Inhaltsverzeichnis enthält den Namen der Diskette, ein zwei Zeichen langes Identitätskennzeichen, die Programmnamen und die Anzahl der noch nicht benutzten Blocks oder Sektoren. Hier ein Beispiel, wie ein Disketten-Directory aussehen kann. Die erste Ziffer der ersten Zeile bezeichnet die Nummer des Diskettenlaufwerks. Die Numerierung beginnt bei 0. Man kann nämlich an einen Commodore-Computer mehrere Diskettenlaufwerke anschließen.

0 "DEMO"	ZX 2A
21 "FINANZ"	PRG
7 "LISTE"	REL
21 "PING PONG"	PRG
8 "SCHACH"	PRG
19 "GEHALT"	SEQ
5 "KING KONG"	PRG
583 BLOCKS FREE	

Auf die Nummer des Diskettenlaufwerkes folgt der Name der Diskette, in diesem Fall 'DEMO'. Die nächsten beiden Buchstaben, 'ZX', sind das Diskettenidentitätskennzeichen. Dieses Diskettenidentitätskennzeichen muß eindeutig sein. Die Bezeichnung '2A' dient zur Kennzeichnung der Version des Betriebssystems. Das Betriebssystem soll im Moment nicht weiter besprochen werden.

Auf diese erste Zeile folgt eine Tabelle. In jeder Zeile dieser Tabelle ist ein Programm oder ein Datensatz aufgeführt. Die erste Spalte der Tabelle gibt an, wieviele Sektoren zur Speicherung des Programms bzw. des Datensatzes benutzt werden. In der zweiten Spalte steht in Anführungszeichen eingeschlossen der Programmname bzw. der Name eines Datensatzes. Die letzte Spalte kennzeichnet die Art der abgespeicherten Information. PRG steht beispielsweise für 'program'. Es handelt sich also um ein Programm, welches mit dem LOAD-Befehl geladen und ausgeführt werden kann. Die Bezeichnung 'SEQ' kennzeichnet einen Datensatz. Es handelt sich hierbei um einen Datensatz, der sequentiell organisiert ist. Dieser Datensatz kann mit Hilfe eines BASIC-Programms eingelesen werden. Es gibt noch weitere Möglichkeiten, Daten zu organisieren. Eine dieser Möglichkeiten ist ein relativ organisierter Datensatz mit der Kennzeichnung 'REL'. Details hierzu werden später gegeben.

Die letzte Zeile gibt schließlich an, wieviele Sektoren noch zur Speicherung weiterer Informationen zur Verfügung stehen.

Sie können sich das Directory ansehen, indem Sie folgenden Befehl eingetippen:

```
LOAD "$", 8
```

Mit diesem Befehl wird das Directory von der Diskette geladen. Wenn Sie es

sich auf dem Bildschirm ansehen wollen, müssen Sie den Befehl LIST (RETURN Taste drücken) eingeben. Denken Sie daran, daß durch das Laden des Directories der Inhalt des Speichers des Computers zerstört wird. Ein Programm, welches sich also gerade im Speicher des Computers befindet, wird zerstört und durch das Directory überschrieben.

Wenn die Diskette noch nicht initialisiert ist, kann das Directory natürlich nicht geladen werden. In diesem Fall leuchtet zunächst das rote Lämpchen auf und beginnt nach einer Weile zu blinken.

Das Initialisieren einer Diskette

Sie können eine Diskette mit den folgenden Befehlen initialisieren:

```
OPEN 1,8,15  
PRINT#1, "N:DISK NAME,ID"
```

Das erste Kommando (OPEN 1,8,15) dient zur Vorbereitung. Die erste Zahl nach OPEN - in dem vorliegenden Fall die 1 - wird Filenummer genannt. Sie kann jede Zahl zwischen 0 und 255 sein. Sie brauchen also nicht unbedingt mit der Zahl 1 zu beginnen. Die zweite Zahl - in diesem Fall die 8 - ist die Einheitennummer. Wir haben bereits darüber gesprochen, daß die Diskette die Einheitennummer 8 hat. Die letzte Zahl wird Sekundäradresse genannt. Wir werden später noch einmal auf diese Zahl zurückkommen. Momentan sollten Sie sich nur merken, daß Sie immer die Zahl 15 für die Sekundäradresse angeben sollten, wenn Sie einen Befehl übermitteln.

Ein File, der mit dem Befehl OPEN (=öffnen) geöffnet worden ist, sollte mit dem Befehl CLOSE wieder geschlossen werden. Ein File wird geschlossen, indem Sie hinter dem Befehl CLOSE die Filenummer angeben. In unserem Fall müßten wir also schreiben: CLOSE 1.

Mehr darüber werden Sie später erfahren.

Mit dem PRINT-Befehl wird nun dem Rechner das eigentliche Kommando übermittelt. Das Kommando wird dem File 1 übergeben. Dieses Kommando muß in Anführungszeichen stehen. Der Buchstabe N bedeutet, daß eine neue (New) Diskette initialisiert werden soll. Hinter dem Doppelpunkt wird der Name der Diskette angegeben. Danach folgt die zwei Zeichen lange Diskettenidentitätskennzeichnung, die von dem Diskettenamen durch ein Komma ge-

trennt sein muß.

Es dauert ein paar Minuten, bis die Diskette initialisiert ist. Haben Sie deshalb keine Angst, wenn das rote Lämpchen zunächst aufleuchtet. Wenn der Vorgang des Initialisierens beendet ist, wird das rote Lämpchen erlöschen und auf dem Schirm bekommen Sie die Meldung 'READY'. Sie können die Initialisierung überprüfen, indem Sie sich das Directory ansehen. Es wird in dem vorliegenden Fall folgendermaßen aussehen:

```
0 "DISK NAME "ID 2A
664 BLOCKS FREE
```

Das Laden eines Programms von der Diskette

Das Laden eines Programms von der Diskette ist genauso einfach wie das Laden eines Programms von der Datassette. Erinnern Sie sich an die besprochene Verfahrensweise. Wenn Sie das Programm mit dem Namen 'FINANZ' von der Diskette laden wollen, tippen Sie einfach folgenden Befehl ein:

```
LOAD FINANZ ,8
```

Sie müssen dabei unbedingt beachten, daß der Programmname so geschrieben werden muß, wie er im Directory steht, bzw. wie er bei dem SAVE-Befehl angegeben wurde.

Wie bei der Datassette, können Sie auch bei der Diskette beim Laden die Technik des 'Pattern Matching' benutzen. Der formale Aufbau des Befehls ist hier nur etwas anders. Anstelle des fehlenden schließenden Anführungszeichens müssen Sie ein 'Sternchen' benutzen. Das rechte schließende Anführungszeichen muß geschrieben werden. Ein paar Beispiele:

- Wenn Sie `LOAD "*",8` schreiben, wird das erste Programm geladen, welches im Directory aufgeführt ist.
- Wenn irgendein Programm geladen worden ist, wird durch den Befehl immer genau dieses Programm wieder geladen, und zwar selbst dann, wenn dieses nicht das erste Programm in dem Directory war.
- Mit dem Befehl `LOAD"PRO*",8` wird das erste Programm in dem Directory geladen, dessen Name mit den Buchstaben P, R, O beginnt.

Einige Programme erfordern, daß sie an eine bestimmte Stelle in den RAM-Speicher des Computers geladen werden. Diese Programme wurden nicht in BASIC geschrieben. Wir haben dieses Thema bei dem LOAD-Befehl für die Datensette bereits besprochen. Hier ein Beispiel:

```
LOAD "PROGRAMM NAME",8,1
```

SAVE und VERIFY

Sie können mit dem Befehl SAVE ein Programm auf die Diskette speichern. Dieser Befehl ist ähnlich zu benutzen, wie wir es schon im Zusammenhang mit der Datensette besprochen haben. Der einzige Unterschied ist, daß nach dem Programmnamen die Einheitennummer 8 angegeben werden muß. Diese Einheitennummer ist durch ein Komma von dem Programmnamen zu trennen. Nachfolgend ein Beispiel:

```
SAVE "PROGRAMM NAME",8
```

Wenn Sie mehrere Diskettenstationen an Ihren Commodore angeschlossen haben, müssen Sie außerdem die Nummer des Laufwerks angeben, damit der Computer weiß, auf welches Laufwerk er das Programm abspeichern soll. Wenn Sie das Programm mit dem Namen 'SCHACH' auf die Diskettenstation mit der Nummer 1 abspeichern wollen, geben Sie bitte folgenden Befehl ein:

```
SAVE "1:SCHACH",8
```

Wird die Nummer der Diskettenstation nicht angegeben, wird das Programm automatisch auf die Diskettenstation mit der Nummer 0 gespeichert.

Während einer Programmentwicklung wird ein Programm meistens nach und nach verbessert, verfeinert und von Fehlern befreit. Im Laufe der Entwicklung eines Programms, wird also solch ein Programm mehrfach auf einer Diskette gespeichert werden müssen. Dieses 'mehrfach' bedeutet aber, daß immer die letzte Version des Programms durch die neue Version überschrieben wird. In diesem Fall muß bei dem SAVE-Befehl das Zeichen @ benutzt werden. Das Zeichen @ wird vor dem Programmnamen angegeben. Danach folgt ein Doppelpunkt.

```
SAVE"@:SCHACH",8
```

oder

```
SAVE"@0:SCHACH",8
```

Mit dem VERIFY-Befehl kann geprüft werden, ob das Programm korrekt abgespeichert worden ist. Der VERIFY-Befehl wird folgendermaßen angegeben:

```
VERIFY "FINANZ",8
```

Wenn das Programm nicht korrekt abgespeichert worden ist, werden Sie die Fehlermeldung VERIFY ERROR auf dem Bildschirm erhalten.

Löschen von Programmen und Datensätzen auf der Diskette

Auch wenn eine einzige Diskette eine Speicherkapazität von nahezu 170 000 Bytes hat, werden Sie sehr bald feststellen, daß dieser Platz schnell verbraucht ist. Sie können sich Platz schaffen, indem Sie die Programme löschen, die Sie nicht mehr benötigen. Dazu tippen Sie folgende Befehle ein:

```
OPEN 1,8,15
```

```
PRINT#1," S:FINANZ ",8
```

Mit diesem Befehl wird das Programm mit dem Namen 'FINANZ' von der Diskette gelöscht. Der Buchstabe S steht für 'Scratch' (= löschen). Es ist möglich, mit einem Befehl mehrere Programme zu löschen. Sie müssen diese Programmnamen dann durch Kommata getrennt angeben. So löscht beispielsweise der Befehl

```
PRINT#1,"S:FINANZ, SCHACH, PING PONG",8
```

die Programme 'FINANZ', 'SCHACH' und 'PING PONG'.

Diskettenfehler

Wie Sie ja bereits wissen, tritt ein Fehler auf, wenn das rechte Lämpchen blinkt. Das Blinken zeigt zwar an, daß ein Fehler aufgetreten ist; Sie wissen aber immer noch nicht welcher. Wie können Sie also erkennen, welcher Fehler aufgetreten ist?

Testen Sie es mal, indem Sie einen Fehler absichtlich herbeiführen. Laden Sie einfach ein Programm, welches nicht auf der Diskette vorhanden ist. Sie werden feststellen, daß das rote Lämpchen erst aufleuchtet und dann zu blinken beginnt. Tippen Sie nun folgendes kleines Programm ein und starten Sie es.

```
10 OPEN 1,8,15
20 INPUT#1,A$,B$,C$,D$
30 PRINT A$,B$,C$,D$
40 CLOSE 1
```

Sie werden feststellen, daß auf dem Schirm folgende Meldung erscheint:

```
62 FILE NOT FOUND 00 00
```

Die erste Zahl ist die Fehlernummer. Danach folgt der Fehler im Klartext. FILE NOT FOUND heißt, daß das Programm mit dem angegebenen Namen nicht auf der Diskette gefunden werden konnte.

Ein paar Vorsichtsmaßnahmen im Umgang mit Diskettenspeichern

Die Floppy Disk bietet im Umgang mit dem Computer einige Vorteile gegenüber der Datassette. Da diese Vorteile zum Teil in der höheren Geschwindigkeit liegen, wird Ihr Computersystem dadurch noch leistungsfähiger.

Aber: nichts und niemand ist perfekt, und das gilt auch für das Diskettenlaufwerk 1541. Deshalb sollten Sie die folgenden Regeln für den Umgang mit Ihrem Diskettenlaufwerk beachten.

- Wenn Sie vergessen haben, die Klappe Ihres Diskettenlaufwerks zu schließen, nachdem Sie die Diskette eingelegt haben und nun ver-

suchen ein Programm zu speichern oder zu laden, wird auf dem Bildschirm die Meldung FILE NOT FOUND ERROR ausgegeben. Diese Fehlermeldung bedeutet, daß das Programm auf der Diskette nicht gefunden werden konnte. Die Fehlermeldung bleibt bestehen, auch wenn Sie die Klappe schließen und dasselbe Programm noch einmal laden. Um den Fehler zu beheben, müssen Sie die Klappe öffnen, die Diskette herausnehmen, wieder einlegen und die Klappe schließen.

- Wenn Sie zwei Diskettenlaufwerke benutzen wollen, muß einem von beiden eine von acht verschiedenen Einheitennummern zugewiesen werden. Bitten Sie Ihren Händler, die Einheitennummer für Sie zu ändern.
- Sie sollten vorsichtig sein, wenn Sie Ihre Diskette mit dem Laufwerk 1541 und mit dem Original Commodore 4040 Doppeldiskettenlaufwerk verwenden wollen. Diese beiden Geräte sind nicht immer kompatibel, obwohl sie es sein sollten.
- Einige 1541 Diskettenlaufwerke können Fertigungstoleranzen unterliegen. Dabei ist in den meisten Fällen eine unterschiedliche Umdrehungsgeschwindigkeit der Disketten problematisch. Sie können Ihre Diskette mit dem Programm 'Performance Test' prüfen. Dieses Programm befindet sich auf einer Test-Diskette, die bei dem Kauf Ihres Diskettenlaufwerks mitgeliefert wird.
- Decken Sie niemals die Lüftungsöffnungen an der Rückseite des Laufwerks ab. Es passiert sehr leicht, indem Sie Bücher oder irgendwelche Blätter mit Aufzeichnungen auf das Gerät legen. In diesem Fall kann es leicht zur Zerstörung des Diskettenlaufwerks durch Überhitzung kommen.

Kompatibilität von Programmen

Unter Kompatibilität von Programmen versteht man die Möglichkeit, daß ein Programm auf mehreren Computern ausgeführt werden kann. Dieses Programm ist dann innerhalb dieser verschiedenen Computer kompatibel. Leider sind die einzelnen Computer der Commodore-Familie nicht vollständig kompatibel.

Zu den Programmen, die nicht auf dem Commodore 64 laufen, gehören folgende:

- Die meisten Programme, die nicht in BASIC geschrieben sind.
- BASIC-Programme, die Programmteile in Maschinensprache enthalten (PEEK-, POKE-Befehle).
- Programme, die für den VC 20 geschrieben worden sind.
- Programme, die mit den Befehlen der 4.0 BASIC-Version geschrieben worden sind.

Vielleicht haben Sie jetzt den Eindruck, daß es ja nicht mehr sehr viele Programme geben kann, die kompatibel sind. Das ist aber nicht der Fall. Sie sollten durch probieren selbst herausfinden, welche Programme auf andere Computer übertragbar sind.

Zu den Programmen, die auf Ihrem VC 64 geschrieben worden sind und nicht auf einem anderen Computertyp von Commodore zur Ausführung gebracht werden können, gehören folgende:

- Programme, die nicht in BASIC geschrieben wurden.
- Programme, die Farbgrafikbefehle enthalten und Programme, die den Tongenerator des VC 64 benutzen.

Ferner gilt:

- Programme, die die Funktionstasten des VC 64 benutzen, sind nicht auf andere Computertypen übertragbar.
- Programme, die davon ausgehen, daß der Bildschirm mehr als 40 Zeichen pro Zeile aufnehmen kann, sind ebenfalls nicht auf dem VC 20 ausführbar.
- Das Programm darf nicht mehr als 32 KB Speicherplatz benötigen.

Es gibt Umsetzer, die automatisch ein Programm so umstrukturieren, daß es auf verschiedenen Computertypen ausgeführt werden kann. Diese Umsetzer nennt man Emulatoren.

Der Commodore 64 als Rechenmaschine

Sie haben die Möglichkeit, Ihren Computer auch als Rechenmaschine zu benutzen, ohne dafür extra ein Programm schreiben zu müssen. Bei einem herkömmlichen Taschenrechner wird eine mathematische Gleichung mit dem Gleichheitszeichen beendet:

$$12 + 484.25 =$$

Im Unterschied hierzu wird bei dem Computer das '='-Zeichen weggelassen und stattdessen der PRINT-Befehl der Gleichung vorangestellt.

```
PRINT 12 + 484.25 (RETURN-Taste drücken)
```

Das Ergebnis wird in einer neuen Zeile auf dem Bildschirm ausgegeben. Eine weitere Möglichkeit, eine mathematische Gleichung mit dem Computer darzustellen, ist der Gebrauch von Variablen, die Sie als Buchstaben auf Ihrer Tastatur finden. Sollten Sie die Bedeutung der Variablen nicht kennen, werden Sie später mehr darüber erfahren. Geben Sie folgende Zeile an:

```
A = 484.25  
PRINT 12 + A
```

Verwenden Sie zur Addition und Subtraktion das normale Plus- (+) bzw. Minus- (-) Zeichen, zur Multiplikation das '*'-Zeichen und zur Division den Querstrich (/). Negativen Zahlen wird ein '-'-Zeichen vorangestellt, und das Zeichen für die Exponentiation ist der nach oben gerichtete Pfeil. Wurzeln können mit dem Ausdruck SQR berechnet werden. Der Computer stellt Ihnen darüberhinaus die logischen Operatoren AND, OR, NOT, die Vergleichsoperatoren <, > und die trigonometrischen Funktionen, wie SIN (Sinus), COS (Cosinus), TAN (Tangens), ATN (Arcustangens) und LOG (Logarithmus) zur Verfügung. Diesen Funktionsbezeichnungen folgen in Klammern die Parameter. Die Wurzel aus 2 können Sie also beispielsweise wie folgt berechnen:

```
PRINT SQR (2)
```

Wie in der Mathematik können auch Klammern benutzt werden, um die Priorität der mathematischen Operatoren zu ändern.

```
PRINT (12 + 484.25)*53
```

In dieser Gleichung werden zuerst die Zahlen 12 und 484.25 addiert; das Ergebnis dieser Summe wird dann mit 53 multipliziert.

Zwei wichtige Hinweise: Achten Sie darauf, daß Sie niemals die Tausender, wie im kommerziellen Bereich üblich, durch Punkte kennzeichnen (1 484 252 und nicht 1.484.252) . Eine Eingabezeile sollte nicht mehr als 80 Zeichen enthalten, also nicht mehr als zwei Bildschirmzeilen lang sein.

Für den PRINT-Befehl gibt es eine Abkürzung: Das Fragezeichen (?). Anstelle des Ausgabebefehls PRINT geben Sie einfach das Fragezeichen ein. Probieren Sie es einmal. Das Kapitel über BASIC-Programmierung wird diesen Punkt noch einmal aufgreifen.

Bestimmung des frei verfügbaren Speicherplatzes

Wie Sie bereits wissen, verfügt der Commodore 64 über einen Speicherplatz von 64 000 Bytes. Die Meldung, die Sie nach Einschalten des Computers erhalten, besagt, daß 38 911 dieser Bytes frei sind, d. h. für Ihr BASIC-Programm verfügbar sind.

Wenn Sie ein Programm eingetippt oder von der Datassette oder Diskette geladen haben, können Sie sich über den noch zur Verfügung stehenden Speicherplatz informieren:

```
PRINT FRE(0)
```

Das Wort FRE ist eine Abkürzung FREe Bytes (= freie Bytes). In den Klammern kann eine beliebige Zahl oder ein Buchstabe stehen. Da die Angabe in Klammern hier keine Bedeutung hat, wird sie auch 'Dummy' genannt. Die Zahl, die auf dem Bildschirm erscheint, gibt die Anzahl der freien Bytes minus drei an. Wie Sie erkennen können, benötigt der Befehl PRINT FRE(0) für sich selbst drei Bytes.

Wenn die Zahl auf dem Bildschirm negativ ist, müssen Sie 65 536 von

ihr subtrahieren. 65 536 entspricht 64 KB. Ein KB Speicherkapazität (1 KB = 1 000 Bytes beträgt in Wirklichkeit 1 024 Bytes).

Ist hier dem Computerhersteller ein kleiner Fehler unterlaufen? Die Antwort werden Sie später erhalten.

Der Gebrauch von Anführungszeichen

Wie schon erwähnt, sollten Sie sich mit den Editing-Funktionen des Commodore vertraut machen und den Umgang in der Praxis üben. Besser ist jedoch, alle BASIC-Programmbeispiele, die in diesem Buch enthalten sind, einzugeben. Nur auf diesem Wege der praktischen Erfahrungen können Sie sich mit dem vortrefflichen Editor Ihres Computers vertraut machen.

Sollten Sie einmal die Gelegenheit haben, mit anderen Computern arbeiten zu können, werden Ihnen die Vorzüge Ihres VC 64 sicherlich sehr bald auffallen.

Es kann aber vorkommen, daß die Editing-Funktionen bei dem Gebrauch von Anführungszeichen nicht so funktionieren, wie sie sollten. Das ist der Fall, wenn Sie eine ungerade Anzahl von Anführungszeichen eingegeben haben.

Beim Eingeben einer Zeile kann es Ihnen beispielsweise passieren, daß Sie unbeabsichtigt ein Anführungszeichen eingegeben haben. Nun wollen Sie dieses mit der CRSR-Taste korrigieren. Anstelle des nach links wandernden Cursors erscheinen auf dem Bildschirm merkwürdige Symbole. Was ist passiert?

Sie arbeiten ungewollt in dem 'Quote-Modus'. Das bedeutet, daß die Editing-Tasten eine andere Funktion haben, als Sie bisher kennengelernt haben. Mehr darüber erfahren Sie in dem Kapitel über BASIC-Programmierung.

Wie spät ist es?

Wußten Sie, daß in Ihrem Commodore 64 eine Uhr eingebaut ist? Sie läuft, sobald Sie den Computer einschalten. Diese Uhr wird durch die Variablen TI und TI\$ repräsentiert. TI und TI\$ werden deshalb auch 'reservierte Variablen' genannt; 'reserviert' deshalb, weil TI und TI\$ nicht für andere Zwecke verwendet werden können. Der Computer behandelt TI als Zahl und

TI\$ als Zeichenkette. Um die Ziffern auf dem Bildschirm sichtbar zu machen, müssen Sie den Befehl

```
PRINT TI$
```

eingeben. Sie werden eine sechsstellige Zahl wie beispielsweise 012236 sehen können. Was bedeuten die einzelnen Ziffern? Unterteilen Sie die sechsstellige Zahl in drei Zahlenpaare: 01 22 36. Die ersten beiden Ziffern repräsentieren die Anzahl der Stunden, die der Computer seit Einschalten in Betrieb ist. Die folgenden beiden Ziffern stellen die Minuten und die letzten zwei Ziffern die Sekunden dar. In unserem Beispiel war der Computer 1 Stunde, 22 Minuten und 36 Sekunden seit letztem Einschalten in Betrieb.

Damit Sie den Computer nicht ein- und anschließend wieder ausschalten müssen, um die Uhr auf den Zeitpunkt 0 wieder zurückzusetzen, können Sie folgende Zeile eingeben:

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

Tippen Sie jetzt den Befehl `PRINT TI$` ein. Sie werden wahrscheinlich nur ein paar Sekunden verbraucht haben.

Versuchen Sie nun folgendes:

```
TI$ ="020510"
```

Wenn Sie jetzt wieder `PRINT TI$` eingeben, beginnt die Uhr mit der Startzeit von 2 Stunden, 5 Minuten und 10 Sekunden zu laufen. Sie können also mit dem Befehl die Uhrzeit beliebig einstellen. Die Zeit, die eingestellt werden soll, muß in Anführungszeichen gesetzt werden, da Sie sonst eine Fehlermeldung erhalten. Diese Uhr wird oft auch als 'Real-Time' Uhr bezeichnet, da sie die echte Zeit anzeigen kann. Die Uhr erlaubt eine 24 Stunden Anzeige, d. h. wird sie um Mitternacht auf 000000 eingestellt, zeigt sie um 1 Uhr Mittags 13 00 00 Uhr und um 11 Uhr nachts 23 00 00 Uhr an.

Sie sollten aber beachten, daß die Uhrzeit nicht weiterläuft, wenn Sie von der Datensette ein Programm laden oder ein Programm auf Datensette speichern.

Es gibt eine zweite Uhr in Ihrem Computer, die allerdings für Zeitan-
sagen weniger geeignet ist. Diese interne Uhr wird jede 1/60 Sekunde um
eine Einheit hochgezählt. Sie startet bei dem Wert 0 und läuft, bis sie
den Wert 51 839 999 erreicht hat. Dann wird sie automatisch wieder auf 0
zurückgesetzt. Der gegenwärtige Stand dieser Uhr kann abgefragt werden,
indem Sie

PRINT TI

eingeben.

4 Eine Einführung in das Programmieren

Vor nicht allzu langer Zeit wurde noch jeder, der einen Computer programmieren konnte, für ein Genie gehalten. In Wirklichkeit kennt jedoch derjenige, der einen Computer programmieren kann, nur dessen Fähigkeiten und Grenzen, und er ist dazu in der Lage, größere Probleme in eine Reihe von Teilproblemen aufzulösen.

Heute hält man diese Leute nicht mehr für überdurchschnittlich begabt und intelligent, sondern man schreibt Ihnen Wissen und Erfahrung auf diesem Gebiet zu. Sehr viel mehr Menschen sind in der Lage, einen Computer zu programmieren, als die Anzahl derjenigen, die das Programmieren zu ihrem Beruf gemacht haben.

Um Programmieren zu lernen, müssen Sie sich zunächst mit einigen grundlegenden Dingen vertraut machen.

Nachdem Sie dieses Kapitel durchgearbeitet haben, werden Sie wissen, was ein Computerprogramm eigentlich ist, wie der Computer Texte und Zahlen speichert und wie der PRINT-Befehl funktioniert.

Die Anfänger unter den Lesern sollten sich ermutigen lassen: Sie werden staunen, was Sie alles mit den wenigen Befehlen, die Sie jetzt lernen werden, tun können.

Im weiteren Verlauf des Buches werden Sie immer wieder kleine BASIC Programme als Beispiel finden, die zur Erläuterung der Theorie dienen. Wenn Sie wissen wollen, wie diese Beispiele funktionieren, tippen Sie das Programm in Ihren Commodore 64 ein und lassen Sie es laufen. Sie sollten aber zuvor den Inhalt des Speichers löschen, damit Sie das mög-

licherweise im Speicher vorhandene 'alte' Programm löschen. Tippen Sie dazu den Befehl NEW ein und drücken Sie die RETURN-Taste.

Solange Sie lernen, werden Sie Fehler machen. Das gehört zu einem normalen Lernprozeß; lassen Sie sich durch die Probleme, die Ihnen dabei begegnen, nicht frustrieren. Nach jedem Fehler sollte man einen neuen Versuch wagen und das Beispielprogramm erneut eingeben.

Informationen über Fehlermeldungen, die Ihnen der Computer ausgibt, finden Sie in den nächsten Abschnitten. Durch die ERROR-Meldung (= Fehlermeldung) teilt der Computer Ihnen mit, daß Sie einen Fehler gemacht haben. Der Rechner spezifiziert den Fehler näher und gibt Ihnen Hinweise für die möglichen Ursachen des Fehlers. Diese Fehlermeldungen können jedoch manchmal etwas irreführend sein.

Die Grundlagen von BASIC

Die am häufigsten verwendete Computersprache für Commodore Computer ist BASIC. BASIC wurde von John Kemeny und Thomas Kurtz in den 60er Jahren zu dem Zweck entwickelt, einem Anfänger den Einstieg in den Umgang mit Computern zu erleichtern. Die Befehle der Programmiersprache BASIC sind dem englischen Wortschatz entlehnt. Der Name BASIC steht für Beginners' All Purpose Symbolic Instrucion Code. Aufgrund der Bezeichnung mag man meinen, daß BASIC die grundlegende Computersprache ist, auf der sich andere Sprachen aufbauen. Das ist aber falsch. BASIC ist lediglich eine Sammlung einfacher, sinnvoller Befehle. BASIC ist auch nicht die mächtigste Computersprache. Es gibt Computersprachen, die für bestimmte Zwecke erheblich besser geeignet sind als BASIC. So eignet sich FORTRAN beispielsweise hervorragend zur Bearbeitung von naturwissenschaftlich-technischen Problemstellungen. COBOL dagegen ist eine Sprache, die hauptsächlich für Problemlösungen aus dem kommerziellen Bereich verwendet wird. Die Programmiersprache LISP ist eine Programmiersprache, die zunehmend zur Bearbeitung von Problemen aus dem Bereich der 'künstlichen Intelligenz' benutzt wird.

Wie jede andere Computersprache auch, besitzt BASIC ihr eigenes Vokabular. Die einzelnen Vokabeln einer jeden Programmiersprache kann durch die Syntax und durch die Semantik beschrieben werden. Die Syntax beschreibt den formalen Aufbau, die Semantik die Bedeutung der einzelnen Vokabeln.



BASIC eignet sich hervorragend dafür, Anfänger im Umgang mit dem Computer vertraut zu machen und sie in die allgemeine Problemstellung einzuführen.

Fast jedes Programm kann prinzipiell in BASIC geschrieben werden; dennoch sind andere Sprachen oft weniger aufwendig in der Handhabung und benötigen für die Ausführung erheblich weniger Computerzeit.

Eine Bemerkung zur Ausführungszeit von Programmen: Die BASIC-Version, die für den Commodore 64 verwendet wird, wird auch BASIC-Interpreter genannt. 'Interpreter' heißt, daß der Rechner das Programm Punkt für Punkt die einzelnen Anweisungen des Programms nacheinander ausführt. Im Gegensatz hierzu wird bei einem BASIC-Compiler das Programm erst in ein Maschinenprogramm übersetzt. Dieses Maschinenprogramm kann von dem Computer

sehr viel schneller ausgeführt werden. Programme, die mit einem BASIC-Compiler übersetzt und dann ausgeführt werden, laufen deshalb sehr viel schneller, als wenn sie mit einem BASIC-Interpreter übersetzt werden.

Näheres darüber werden Sie in einem der folgenden Kapitel finden.

Die in Ihrem Commodore 64 enthaltene BASIC-Version ist ein Dialekt von Microsoft-BASIC. Die Firma Microsoft hat diese Version ursprünglich für den MITS Altair, einem früheren Personal Computer, der nicht mehr hergestellt wird, geschrieben. Mittlerweile ist das BASIC von Microsoft fast zu einem Standard für Kleincomputer-BASIC geworden.

Dennoch gibt es einige Unterschiede in den einzelnen BASIC-Versionen, die Microsoft für den TRS-80, den Apple II und für andere Computer entwickelt hat.

Viele BASIC-Befehle sind für sämtliche Personal-Computer gleich. Dennoch gibt es aber bedeutende Unterschiede zwischen den einzelnen Computern. So kann es vorkommen, daß ein Programm, welches auf dem Commodore 64 geschrieben wurde, nicht auf einem Apple-Computer läuft (und umgekehrt). Sie müssen dieses einfach als Tatsache hinnehmen. Später werden Sie lernen, worin die Unterschiede der einzelnen BASIC-Dialekte bestehen.

Der Aufbau eines Programms

Ein Computerprogramm besteht aus einer Folge von Anweisungen, die der Computer ausführen soll. Denken Sie daran, daß solch ein Programm in einzelne Abschnitte untergliedert ist. Diese einzelnen Abschnitte können ein selbständiges Programm darstellen. In diesem Fall bezeichnet man solch einen selbständigen Programmabschnitt auch als Unterprogramm.

So, wie Sie ja auch viele Arbeiten aus dem täglichen Leben, wie z. B. das Zubereiten einer Mahlzeit, in einzelne in sich abgeschlossene Teilaufgaben unterteilen, so sollten Sie auch bei der Programmierung eines Programms versuchen, einzelne Abschnitte getrennt voneinander zu programmieren.

Die äußere Form eines BASIC Programms

Jedes BASIC-Programm besteht aus einzelnen Zeilen. Jede Zeile kann eine

oder mehrere Anweisungen enthalten. Zur Identifizierung der Zeilen muß jede Zeile eine Zeilennummer enthalten. Die Zeilennummern dürfen dabei in einem Bereich von 0 und 63 999 variieren.

Wenn Sie das eingetippte Programm auf dem Bildschirm ausgeben wollen, müssen Sie den Befehl LIST eingeben und die RETURN-Taste drücken. Die Zeilen werden dann nacheinander so schnell auf dem Bildschirm ausgegeben, daß Sie sie nicht richtig lesen können. Um die Geschwindigkeit, mit der die Zeilen auf dem Bildschirm erscheinen, zu reduzieren, müssen Sie die CTRL-Taste drücken. Dadurch wird die Geschwindigkeit, mit der der Computer arbeitet, drastisch reduziert. Wenn Sie die CTRL-Taste wieder loslassen, läuft der Computer mit seiner ursprünglichen Geschwindigkeit wieder weiter.

Wenn die RUN/STOP-Taste gedrückt wird, wird das Auflisten des Programms unterbrochen und die Meldung READY erscheint auf dem Bildschirm. Um fortzufahren, müssen Sie wieder LIST eingeben. Mit LIST können Sie sich aber auch gezielt bestimmte Programmzeilen oder Abschnitte ansehen. Die nachfolgende Tabelle gibt einen Überblick über die Möglichkeiten, die Sie mit LIST haben.

Befehl	Bedeutung
LIST	Das komplette Programm wird aufgelistet.
LIST 100	Nur die Programmzeile 100 wird gelistet.
LIST 100 - 150	Die Zeilen 100 - 150 werden gelistet.
LIST - 150	Sämtliche Programmzeilen, von Beginn des Programms bis einschließlich Zeile 150, werden gelistet.
LIST 150 -	Alle Programmzeilen, beginnend mit der Zeile 150 bis zum Ende des Programms, werden gelistet.

Beim Schreiben eines Programms sollten Sie stets mit einer Anweisungsnummer beginnen, die größer als 0 ist. Sie können so später bei Bedarf weitere Zeilen vor der ersten Zeile einfügen. Erfahrene Programmierer wählen aus diesem Grund eine Numerierung in 10er Schritten. So hat man genügend Platz, um später bei Änderung des Programms einzelne Programmenteile einzufügen.

Eine BASIC-Anweisungszeile darf länger als die Zeile des Bildschirms

sein. Jede Zeile kann 80 Zeichen lang sein, wobei die Zeilennummer eingeschlossen ist. Da eine Bildschirmzeile genau 40 Zeichen enthält, kann eine BASIC-Anweisungszeile genau zwei Bildschirmzeilen lang sein.

Bevor Sie mit dem Programmieren beginnen, vergewissern Sie sich, daß der Inhalt des Speichers gelöscht ist. Dazu können Sie entweder den Befehl NEW eintippen (RETURN-Taste nicht vergessen) oder Sie können den Computer kurz ausschalten. Von der letzten Möglichkeit sollten Sie allerdings nur sehr wenig Gebrauch machen. Vermeiden Sie es bitte unbedingt, den Computer sofort nach dem Ausschalten wieder einzuschalten, da dadurch die elektronischen Bauteile des Computers beschädigt werden können.

Nun können Sie ein neues Programm in den Computer eingeben. Jeder BASIC-Anweisungszeile muß eine Zeilennummer vorangehen. Nachdem Sie die Anweisungszeile eingetippt haben, drücken Sie bitte die RETURN-Taste. Erst dadurch wird die Anweisungszeile in den Speicher des Computers übertragen. Während der Eingabe können Sie mit den beschriebenen Editing-Möglichkeiten durch die INST/DEL- und die CRSR-Taste beliebig die Programmzeilen korrigieren und ändern.

Nachdem die RETURN-Taste gedrückt ist, wird die eingetippte Anweisungszeile in den Speicher übertragen und der Cursor springt automatisch auf dem Bildschirm zur nächsten verfügbaren Eingabezeile. Denken Sie bitte immer daran, daß die Zeilenlänge einer BASIC-Zeile maximal 80 Zeichen betragen darf.

Sie können eine einzelne Anweisungszeile dadurch löschen, indem Sie ihre Anweisungsnummer eingeben und direkt danach die RETURN-Taste drücken. Wenn beispielsweise die Zeile 100 gelöscht werden soll, muß nur die Zahl 100 in einer freien Eingabezeile eingetippt und danach die RETURN-Taste gedrückt werden. Mit dem LIST-Befehl können Sie sich davon überzeugen, daß diese Zeile gelöscht worden ist.

Mit dem Editor können Sie Zeilen oder Teile von Zeilen duplizieren. Probieren Sie es einmal aus. Geben Sie folgende BASIC-Zeile ein:

```
10 PRINT "ZEILE 10"
```

Drücken Sie die RETURN-Taste. Bewegen Sie nun den Cursor zu der eingegebenen Zeile und ändern Sie die Zahl 10 jeweils in 20 um. Drücken Sie dann die RETURN-Taste. Bewegen Sie den Cursor zu einer freien Eingabezeile und geben Sie das Kommando LIST ein. Sie werden dann sehen, daß Sie folgenden

Ausdruck bekommen:

```
10 PRINT "ZEILE 10"  
20 PRINT "ZEILE 20"
```

Auf diese Art und Weise können Sie sich das Eintippen eines Programms sehr erleichtern, wenn Programmzeilen dupliziert werden sollen oder wenn ähnliche Programmzeilen aufeinander folgen.

Das Programmende

Bei den meisten Computern muß ein BASIC-Programm mit dem Befehl END am Ende eines Programms abgeschlossen werden. Läuft der Computer während der Ausführung eines Programms auf den END-Befehl, so wird das Programm beendet. Obwohl es einem guten Programmierstil entspricht, das END zu benutzen, ist es für Commodore-BASIC nicht erforderlich. Der END-Befehl kann aber auch mitten im Programm benutzt werden, wenn das Programm an dieser Stelle beendet werden soll. Im Gegensatz zu dem oben beschriebenen ENDE wird dieses 'logisches ENDe' genannt. Auf dieses Thema wird später noch einmal zurückgekommen.

Variablen

In einem BASIC-Programm wird jede Information mit einem Namen identifiziert. Dabei spielt es keine Rolle, ob es sich bei diesen Informationen um Zahlen, Worte, Sätze oder sogar Bilder handelt. Diese Namen werden als 'Variablennamen' bezeichnet, da der Wert der Information, den sie repräsentieren, jederzeit verändert werden kann. Es gibt zwei verschiedene Arten von Variablen: Variablen, die nur Zahlen repräsentieren und Variablen, die Buchstaben und andere Zeichen repräsentieren. Nachstehend folgen einige Beispiele für Variablennamen:

```
A = 10.20  
A$ = "DAS IST MEIN COMMODORE COMPUTER"  
AB = 4096
```

```
AB$ = "DAS IST EIN ANDERER COMPUTER"  
A2 = 45.50  
A2$ = "45.50"
```

In dem obigen Beispiel wird den einzelnen Variablen direkt ein Wert zugewiesen. Bei dieser Wertzuweisung steht der Name der Variablen auf der linken Seite des Gleichheitszeichens. Rechts von dem Gleichheitszeichen steht der Wert, der der Variablen zugewiesen wird. Wenn Sie den Computer fragen, welchen Wert A hat, wird er Ihnen mit 10.20 antworten; für AB\$ wird er Ihnen den Satz DAS IST EIN ANDERER COMPUTER angeben.

Variablennamen dürfen in BASIC ein oder zwei Zeichen lang sein. Das erste Zeichen muß ein Buchstabe, das zweite Zeichen kann ein Buchstabe oder eine Zahl sein. Ein Variablenname kann somit aus einem einzigen Buchstaben (A = 10.20), aus zwei Buchstaben (AB = 4096) oder aus einem Buchstaben und einer Ziffer (A2 = 45.50) bestehen. Damit der Computer eine eindeutige Unterscheidung zwischen einer Variablen und einer Zahl machen kann, darf ein Variablenname nicht mit einer Ziffer beginnen. 22 = 25 wäre somit falsch!

Variablennamen, die Zahlen repräsentieren, werden numerische Variablen genannt (A = 10.20, AB = 4096 und A2 = 45.50 - siehe obiges Beispiel). Variablennamen, die Wörter, Sätze oder andere Zeichen repräsentieren, werden als String-Variablen (string = Kette) bezeichnet. Eine String-Variable ist dadurch gekennzeichnet, daß sie wie eine numerische Variable gebildet wird, ihr aber ein \$-Zeichen angehängt wird (A\$ = "DAS IST MEIN..."- siehe obiges Beispiel). Strings werden auch Zeichenketten genannt und müssen immer in Anführungszeichen gesetzt werden. Ziffern können Teil eines String sein; der Computer behandelt sie wie jedes andere Zeichen, sei es nun ein Buchstabe oder ein Symbol.

Journalisten verwenden Anführungszeichen häufig dazu, wörtliche Kommentare einer zitierten Person im Text hervorzuheben. Eine ähnliche Funktion besitzen die Anführungszeichen auch in einer BASIC-Zeile. Sie markieren den Anfang und das Ende einer Zeichenkette.

Das oben erwähnte Beispiel A2\$ = "45.50" ist sicherlich etwas verwirrend, da 45.50 eine Zeichenkette und nicht eine Zahl ist. Den entscheidenden Hinweis zur Lösung dieses Problems geben uns die Anführungszeichen. Wenn Anführungszeichen gesetzt werden, bedeutet dieses, daß alles, was zwischen diesen Anführungszeichen steht, als String behandelt

werden soll.

Denken Sie immer daran, daß `A2$ = "45.50"` nicht dasselbe wie die Zeile `A2 = 45.50` ist. In dem ersten Fall ist `45.50` aufgrund der einschließenden Anführungszeichen eine Zeichenkette, in dem zweiten Beispiel ist `45.50` eine Zahl.

Mit numerischen Variablen können arithmetische Rechenoperationen wie Addition, Subtraktion, Multiplikation und Division durchgeführt werden.

Wenn `AB` für 10 und `CD` für 250 stehen, so ergibt `AB` plus `CD` 260. Dementsprechend ergibt `CD` minus `AB` 240.

Auf dieselbe Art und Weise können auch Zeichenketten addiert werden. Wenn `AB$` `"DAS IST MEIN"` und `CD$` `"COMMODORE COMPUTER"` repräsentieren, so ergibt die Addition beider Zeichenketten den Satz `"DAS IST MEIN COMMODORE COMPUTER"`. Hierzu das entsprechende Beispielprogramm:

```
10 AB$ = "DAS IST MEIN"
20 CD$ = "COMMODORE COMPUTER"
30 EF$ = AB$ + CD$
```

Zeichenketten können zwar 'addiert' werden, es ist aber nicht möglich, Zeichenketten zu subtrahieren. Ferner ist es natürlich nicht möglich, eine numerische Variable zu einer String-Variablen zu addieren. Wenn z. B. `A2 = 45.50` und `A2$ = "45.50"` ist, so können sie nicht zu der Zahl 91.00 addiert werden. Hier stellt sich natürlich die Frage nach dem 'Warum'. Der Grund hierfür ist allein der, daß `A2` als Zahl und `A2$` als eine Kette (bzw. Aneinanderreihung) einzelner Zeichen betrachtet wird. Ohne hier näher darauf eingehen zu können sei bemerkt, daß numerische Größen und Zeichenketten im Speicher eines Rechners anders dargestellt werden. Es gibt eine Möglichkeit, eine Zeichenkette in eine Zahl umzuwandeln. Wir werden später auf dieses Thema noch einmal zurückkommen.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
TYPE MISMATCH ERROR
```

Dieser Fehler entsteht immer dann, wenn man in einem Ausdruck numerische Größen (also Zahlen) und Texte miteinander mischt. Solch ein Fehler würde beispielsweise bei folgender Programmzeile auftreten:


```
10 A$ = 9 (oder) A$ = A
```

```
oder
```

```
10 A = "BEISPIEL" (oder) A = A$
```

Ähnlich wie das Dollar-Zeichen kann auch das Prozent-Zeichen in einem Variablennamen vorkommen und hat eine besondere Bedeutung. Der Buchstabe A kann jede Zahl - einschließlich eine Dezimalzahl - repräsentieren. Die Zahl 10.5 ist die Summe der Zahlen 10 und 5/10 (bzw. 1/2). Folgt einem Variablennamen das Prozent-Zeichen (z. B. A%) wird erreicht, daß diese Variable nur ganze Zahlen repräsentieren darf. Diese Zahlen dürfen in dem Bereich von -32 768 bis 32 767 liegen. Sollten Sie einmal mit größeren oder kleineren Zahlen arbeiten müssen, so können Sie diese einfach als numerische Variable (wie A = 365 0000) speichern.

Wozu benötigen Sie diese Integer Variablen (= Variablen für ganze Zahlen)? Zum einen laufen Programme etwas schneller ab, wenn sie nur Integer-Größen enthalten. Der Zeitunterschied ist jedoch gering. Erwarten Sie keine enorme Steigerung in der Ausführungsgeschwindigkeit. Zum anderen benötigen Sie für das Abspeichern von ganzen Zahlen weniger Speicherplatz, als zum Abspeichern von Dezimalzahlen. Integer-Variablen können dazu benutzt werden, Dezimalzahlen in ganze Zahlen umzuwandeln. Probieren Sie einmal folgendes Beispiel aus:

```
10 A% = 103.987
```

```
20 PRINT A%
```

A% wird als 103 ausgegeben. Wie Sie feststellen können, ist die Zahl nicht gerundet worden, da sonst 104 und nicht 103 ausgegeben werden würde.

```
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
ILLEGAL QUANTITY ERROR
```

Integer Variablen werden vom Computer aufgrund des Prozent-Zeichens erkannt. Wenn der Fehler ILLEGAL QUANTITY ERROR in einer Zeile vorkommt, die eine Integer Zahl enthält, müssen Sie unbedingt nachprüfen, ob sich Ihre

Integer-Variable tatsächlich im Bereich von -32 768 bis 32 767 befindet.

Ein weiteres Problem ergibt sich, wenn Sie Variablennamen benutzen wollen, die mehr als zwei Zeichen lang sind. Nun, Sie können Variablennamen mit mehr als zwei Zeichen bilden. Der Commodore Computer benutzt zwar diese Namen in dem Programm, nimmt aber nur die ersten beiden Zeichen zur Kenntnis. Diese ersten beiden Zeichen sind signifikant, d. h. nur sie dienen zur Unterscheidung der Variablen. Seien Sie deshalb vorsichtig und verwirren Sie den Computer nicht. Verwenden Sie beispielsweise die Variablen PROZENT und PREIS, wird mit beiden Namen - obwohl sie unterschiedlich sind - die gleiche Variable bezeichnet. Einige Personal Computer (besonders Atari Computer) erlauben die Verwendung längerer Variablennamen. Ihr Commodore 64 kann dies nicht.

Schließlich sollten Sie auch noch wissen, daß Sie bei Ihrem Commodore 64 bestimmte Variablennamen nicht verwenden dürfen. Zu diesen 'verbotenen Namen' gehören IF, TO, OR und GO, da sie zum BASIC-Vokabular gehören. Man nennt diese Namen auch reservierte Wörter. Darüberhinaus dürfen TI und TI\$ nicht verwendet werden, da sie als Variablennamen für die interne Uhr benutzt werden. Der Variablenname ST\$ kann für Zeichenketten; ST jedoch nicht für numerische Größen (Zahlen) verwendet werden. Nachstehende Liste gibt an, welche Variablen benutzt werden dürfen und welche nicht.

Variablenname	zulässig?	Begründung
A\$, A oder A%	ja	Jeder Buchstabe von A-Z kann benutzt werden.
AA\$, AA oder AA%	ja	Die meisten Kombinationen bestehend aus 2 Buchstaben sind zulässig.
22\$, 2B oder 22%	nein	Das erste Zeichen muß ein Buchstabe sein.
A1\$, A1 oder A1%	ja	Kombinationen von Zahlen und Buchstaben sind in dieser Reihenfolge zulässig.
TI\$, TI oder TI%	nein	Dieser Variablenname wird für die interne Uhr des Computers benötigt
TO\$, TO oder TO%	nein	TO gehört zum BASIC-Vokabular

ST	nein	ST gehört zum BASIC-Vokabular
ST\$	ja	Ist ein Variablenname, aus 2 Buchstaben bestehend.
HOTEL\$ oder HOTEL	ja	Variablennamen mit mehr als 2 Zeichen können zwar verwendet werden; der Computer wird aber nur die ersten beiden Zeichen beachten.
HOME\$ oder HOME	nein	verstößt gegen die obigen Regeln.

Bei numerischen und Zeichenketten-Variablen gibt es Einschränkungen bezüglich des Wertebereiches.

Jede Zahl, die als numerische Variable abgespeichert wird, darf nur höchstens neun Stellen lang sein. Beispielsweise kann der Variablenname AB der Zahl 999999999 oder der Zahl .999999999 entsprechen.

Zahlen, die größer als 999999999 und kleiner als .000000001 sind, können ebenfalls gespeichert werden. Der Computer stellt diese Zahlen jedoch in der halblogarithmischen Schreibweise dar. Wenn Sie dem Variablennamen AB die Zahl 200 000 000 000 zuordnen, und den Computer anschließend nach dem Wert von AB fragen, erscheint auf dem Bildschirm: 2E+11.

Falls es Sie interessiert, werden Sie nachfolgend ein paar Informationen über die halblogarithmische Schreibweise erhalten.

Die Schreibweise 10^3 bedeutet, daß die Zahl 10 dreimal mit sich selbst multipliziert werden muß. 10^3 ergibt also 1 000. Die Zahl 3 wird dabei Exponent, die Zahl 10 Basis genannt. Aus diesem Grunde steht bei der Schreibweise 2E+11 das Zeichen 'E' für Exponent. 2E+11 bedeutet, daß die Zahl 2 mit 10^{11} multipliziert werden muß.

Wenn Sie sehr große oder sehr kleine Zahlen benutzen, schaltet der Rechner automatisch auf die halblogarithmische Darstellung um. Haben Sie keine Angst, viele Programme, die Sie benutzen, stellen die Zahlen in der gewohnten Darstellungsweise dar.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
OVERFLOW ERROR
```

Sie erhalten die Fehlermeldung OVERFLOW ERROR, wenn Sie versuchen, mit der größten für den VC 64 darstellbaren Zahl zu arbeiten. Die größte darstell-

bare Zahl ist 1.70141884E+38. Sie sollten es nach Möglichkeit vermeiden, daß diese Zahl in einem Programm erreicht wird. Das ist jedoch nicht immer möglich.

Auch bei den Zeichenketten gibt es eine Einschränkung. Diese ist aber wesentlich einfacher zu verstehen, als die Einschränkung bei den numerischen Werten. Keine Zeichenkette darf mehr als 255 Zeichen enthalten.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
STRING TOO LONG ERROR

Wie bereits erwähnt, dürfen Zeichenketten nur 255 Zeichen lang sein. Wenn Sie verschiedene Zeichenketten durch den Plus-Operator miteinander verbinden, können Ihnen dabei sehr leicht Fehler unterlaufen. Prüfen Sie deshalb die Länge jeder vorkommenden Zeichenkette, wenn Sie die Fehlermeldung STRING TOO LONG ERROR erhalten haben.

In einigen wenigen BASIC-Programmen werden Sie das Wort LET finden. LET definiert eine Variable und kann nach Belieben in einem Commodore-BASIC Programm verwendet werden. Das Wort LET erleichtert die Lesbarkeit eines Programms. Die Zeilen

10 LET AB = 4096

oder

10 AB = 4096

sind gleichbedeutend.

Zeichensetzung in BASIC

Wie jede Programmiersprache, besteht BASIC aus einem bestimmten Vokabular. Dieses Vokabular wird nach festgelegten Regeln gebildet. Aus diesem Grunde ist es verständlich, daß es natürlich auch Regeln hinsichtlich der Zei-

chensetzung gibt. Die Regeln für die Zeichensetzung in BASIC sind wesentlich leichter, als die Regeln für die Zeichensetzung einer natürlichen Sprache (wie z. B. der deutschen Sprache).

Ähnlich wie einige kurze deutsche Ausdrücke eine ganze Zeile ersetzen können, können auch BASIC-Aussagen eine Programmzeile einsparen. In der deutschen Sprache trennt der Punkt einzelne kurze Aussagen, z. B.:

Komm her. Ich jetzt.

In einer BASIC-Zeile werden die einzelnen Aussagen durch einen Doppelpunkt (:) getrennt. Eine BASIC-Zeile mit mehr als einer Aussage sieht beispielsweise wie folgt aus:

```
10 PRINT "DAS IST MEIN COMPUTER":GOTO 10
```

In dieser BASIC-Zeile wird der Doppelpunkt dazu verwendet, zwei verschiedene BASIC-Aussagen voneinander zu trennen. Wie schon erwähnt, kann eine BASIC-Programmzeile (einschließlich der Anweisungsnummer und Leerzeichen) bis zu 80 Zeichen - das sind zwei Bildschirmzeilen - lang sein. BASIC-Zeilen können durch den Doppelpunkt auch lesbarer gestaltet werden. Zu diesem Zweck können einer Anweisungsnummer ein- oder mehrere Doppelpunkte folgen. Zwischen der Zeilennummer und dem Programmlisting existiert somit ein kleiner Zwischenraum:

```
10 PRINT "DAS IST MEIN COMPUTER"  
20 ::GOTO 10
```

Wie Sie anhand des Beispiels sehen können, entsteht durch die Verwendung von zwei Doppelpunkten eine Art 'visueller Einschnitt', aufgrund dessen sich diese Programmzeile vom übrigen Text ein wenig abhebt.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
SYNTAX ERROR
```

Der Syntax-Fehler kann viele Gründe haben; er ist aber immer das Ergebnis einer formalen falschen Anwendung von BASIC. Die häufigsten Ursachen sind Rechtschreibfehler oder fehlende Satzzeichen (besonders Doppelpunkt und Komma werden häufig vergessen). Es kann außerdem vorkommen, daß anstelle

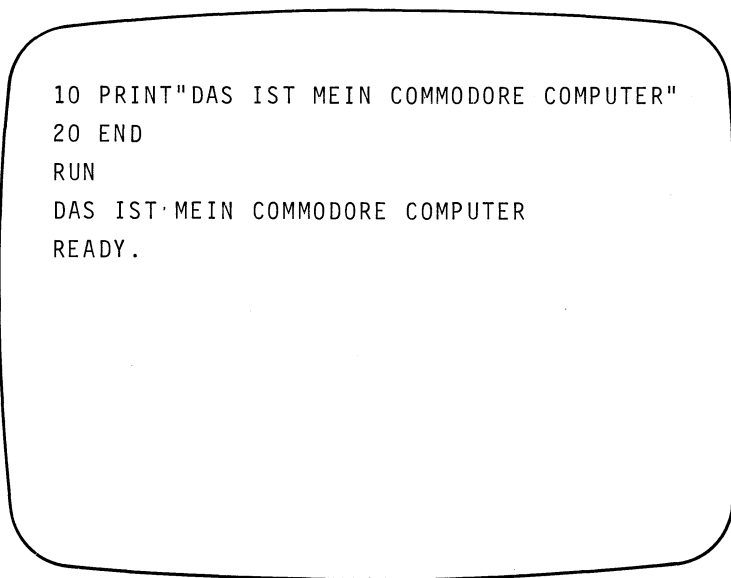
eines Doppelpunktes ein Semikolon verwendet wurde oder daß die Anzahl der geöffneten Klammern nicht der Anzahl der schließenden Klammern entspricht.

Der BASIC Befehl PRINT

PRINT ist aufgrund seiner Funktion sicherlich eines der am meisten verwendeten Befehle in BASIC.

Der Befehl PRINT dient dazu, Daten auszugeben. Durch die Verwendung verschiedener Satzzeichen kann die Ausgabe der Daten durch den PRINT-Befehl gestaltet werden. Obgleich die Bedeutung des PRINT-Befehls an sich leicht verständlich ist, gibt es gerade für diesen Befehl so viele Variationen, daß die vollständige Beherrschung dieses Befehls mehr Zeit in Anspruch nimmt, als es zunächst den Anschein haben mag.

Der PRINT-Befehl veranlaßt den Computer, die jeweils gewünschten Daten auf dem Bildschirm auszugeben. Worte, Sätze, Ergebnisse von Rechnungen und sogar Zeichnungen können auf dem Bildschirm erscheinen. Die erste Zeile eines Programms, das wir bereits in einem der vorangegangenen Abschnitte behandelt haben, lautete:



```
10 PRINT"DAS IST MEIN COMMODORE COMPUTER"  
20 END  
RUN  
DAS IST MEIN COMMODORE COMPUTER  
READY.
```

```
10 PRINT "DAS IST MEIN COMMODORE COMPUTER"
```

Nach dem Befehl PRINT folgt in Anführungszeichen der Text, der ausgegeben werden soll. In dem obigen Beispiel ist es der Text DAS IST MEIN COMMODORE COMPUTER. Sie müssen unbedingt darauf achten, daß der Text, der ausgegeben werden soll, in Anführungszeichen gesetzt wird. Tippen Sie das Wort RUN ein. Damit starten Sie das Programm und der Satz DAS IST MEIN COMMODORE COMPUTER erscheint auf dem Bildschirm.

Der PRINT-Befehl kann auch benutzt werden, um Zahlen auszugeben. Nachfolgend sehen Sie ein Beispiel:

```
10 A = 4096
20 B$ = "DAS IST MEIN COMMODORE COMPUTER"
30 PRINT A
40 PRINT B$
```

Eine kleine Vereinfachung. Geben Sie das Programm genau so, wie es nachfolgend steht - einschließlich der Zwischenräume - ein. Anstelle des Wortes PRINT geben Sie das Fragezeichen ein.

```
10 ? "DAS IST MEIN COMMODORE COMPUTER"
```

Der Computer weiß, daß das Fragezeichen eine andere Bezeichnung für den PRINT-Befehl ist. Die PRINT-Anweisung kann somit durch das Fragezeichen ersetzt werden. Diese Kurzversion des PRINT-Befehls können Sie immer verwenden; auf Ausnahmen werden Sie in diesem Buch hingewiesen.

Die Ausgabe einer Leerzeile. Geben Sie probeweise den PRINT-Befehl ohne nachfolgenden Text ein.

```
10 PRINT "DAS IST MEIN COMPUTER"
20 PRINT
30 PRINT "DAS IST MEIN COMPUTER"
```

Auf dem Bildschirm erscheinen zwei identische Sätze, die durch eine Leerzeile voneinander getrennt sind. Wird also der PRINT-Befehl ohne Text verwendet, so erfolgt die Ausgabe einer Leerzeile.

Versuchen Sie nun, das obige Programm in einer einzigen Anweisungszeile zu schreiben. Sie können dies tun, indem Sie die einzelnen Anweisungen mittels Doppelpunkt voneinander trennen.

```
10 PRINT "DAS IST MEIN COMPUTER":PRINT:PRINT " DAS IST MEIN COMPUTER "
```

In Zusammenhang mit dem PRINT-Befehl werden zwei weitere Satzzeichen - das Komma (,) und das Semikolon (;) - verwendet. Mit diesen beiden Zeichen können Sie den Ausdruck gestalten.

PRINT mit Semikolon. Geben Sie das Programm genau so, wie es unten steht (einschließlich der Zwischenräume) ein. Anstelle des Wortes PRINT kann natürlich das Fragezeichen eingetippt werden.

```
10 PRINT "DAS IST ";  
20 PRINT "MEIN COMPUTER"
```

Starten Sie das Programm mit dem RUN-Befehl. Beachten Sie bitte das Leerzeichen nach dem Wort IST, das letzte Anführungszeichen in Zeile 10 und das Semikolon am Ende dieser Zeile. All das zusammen bewirkt, daß der Satz wie in der Abbildung ausgegeben wird.

Normalerweise wird vor Ausführung eines PRINT-Befehls ein Zeilenvorschub gemacht. Jedes der nach PRINT angegebenen Elemente wird somit in eine neue Zeile geschrieben. Wird jedoch die Angabe eines Elementes mit einem Semikolon abgeschlossen, wird das darauffolgende Element direkt nach dem ersten ausgegeben. Aus diesem Grund wird der Text DAS IST MEIN COMPUTER in einer einzigen Zeile ausgegeben.

Auf diese Art und Weise können zwei einzelne Elemente eines einzelnen Satzes oder einer Gruppe von Zeichen miteinander verbunden werden.

Das Leerzeichen nach dem Wort IST in Zeile 10 und dem folgenden Anführungszeichen ist von großer Bedeutung. Ohne dieses Leerzeichen würde nämlich folgender Text ausgegeben werden:

```
"DAS ISTMEIN COMPUTER "
```

Selbstverständlich kann auch die Ausgabe von Zahlen entsprechend gesteuert werden:


```
10 A = 100
20 B = 1
30 C = A + B
40 PRINT C
```

Der Variablen A wird in Zeile 10 der Wert 100 zugewiesen und der Variablen B der Wert 1 (Zeile 20). Anschließend werden die Werte der beiden Variablen addiert und der Variablen mit dem Namen C übergeben. In Zeile 40 wird

```
10 PRINT"DAS IST ";
20 PRINT"MEIN COMPUTER"
30 END
RUN
DAS IST MEIN COMPUTER
READY.
```

schließlich der Wert der Variablen C ausgegeben. Ohne genaue Kenntnis des Programms ist die ausgegebene Zahl (also das Ergebnis von A plus B) nur schwer zuzuordnen. Zur besseren Identifizierung dieser Zahl sollte deshalb zusätzlich ein entsprechender 'Kommentar' ausgegeben werden. Ändern Sie deshalb Programmzeile 40 wie folgt (Geben Sie nicht NEW ein):

```
40 PRINT "A + B =" ;C
```

Sie glauben vielleicht, daß Sie sich damit nur wiederholt haben. Seien Sie

aber beruhigt; das ist nicht der Fall. Zeile 40 enthält die Anweisung `PRINT A + B =`. Durch das Semikolon wird auch der weitere Text in dieser Zeile ausgegeben. Der Computer gibt den Wert der Variablen C (die Summe von A plus B oder 101) aus. Wenn Sie dieses Programm laufen lassen, werden Sie feststellen, daß die Zahl 101 nicht direkt nach dem Gleichheitszeichen ausgegeben wird. Der Rechner behandelt also bei Verwendung des Semikolons Zahlen anders als Texte. Texte werden nämlich direkt hintereinander geschrieben, d. h. ohne Einfügen von Leerzeichen. Bei der Ausgabe von Zahlen wird zwischen die einzelnen Zahlen ein Leerzeichen eingefügt. Sie können das auch an einem weiteren Programm feststellen. Geben Sie nachstehendes Programm ein und starten Sie es mit dem RUN-Befehl:

```
10 A = 100
20 B = 1
30 C = A + B
40 PRINT A;B;C
```

Die Werte der in der PRINT-Anweisung aufgeführten Variablen werden durch zwei Leerzeichen getrennt ausgegeben. Testen Sie folgendes Programm:

```
10 A = 100
20 B = 1
30 C = A + B
40 PRINT A;B;C; "DAS SIND DIE WERTE VON A, B und C"
```

PRINT mit Komma. Das Komma ist ein weiteres Separationszeichen, das zusammen mit dem PRINT-Befehl verwendet werden kann.

```
10 A$ = "SPALTE 1"
20 B$ = "SPALTE 2"
30 C$ = "SPALTE 3"
40 D$ = "SPALTE 4"
50 A = 1
60 B = 2
70 C = 3
80 D = 4
90 PRINT A$,B$,C$,D$
100 PRINT A,B,C,D
```

Dieses Programm zeigt, wie das Komma zur Gestaltung einer Ausgabe benutzt werden kann. Wenn Sie das Programm starten, können Sie sehen, daß die 40-Zeichen lange Bildschirmzeile in 4 Zonen (Spalten) zu je 10 Zeichen aufgeteilt worden ist. Außerdem können Sie in diesen Spalten den Unterschied in der Ausgabe von Zeichenketten und in der Ausgabe von Zahlen erkennen.

```
10 A$="SPALTE 1"
20 B$="SPALTE 2"
30 C$="SPALTE 3"
40 D$="SPALTE 4"
50 A=1
60 B=2
70 C=3
80 D=4
90 PRINT A$,B$,C$,D$
100 PRINT A,B,C,D
RUN
SPALTE 1  SPALTE 2  SPALTE 3  SPALTE 4
  1         2         3         4
READY.
```

Jede Spalte beginnt mit der Ausgabe einer Zeichenkette in der ersten Position. Diese Zonen beginnen also auf dem Bildschirm mit den Positionen 1, 11, 21 und 31. Die erste Position zählt als Nummer 1.

Beim Ausdruck von Zahlen wird die erste Position scheinbar übersprungen. Die Positionen, bei denen in unserem Beispiel mit der Ausgabe begonnen wird, sind demnach 2, 12, 22 und 32. Das liegt daran, daß wir bei dem Beispiel nur positive Zahlen vorliegen haben. Bei positiven Zahlen wird nämlich das Vorzeichen (+) weggelassen. Wenn Sie die Zeilen 50-80 des obigen Programms in

```
50 A = -1
60 B = -2
70 C = -3
80 D = -4
```

ändern, werden Sie sehen, daß das Vorzeichen '-' anstelle des vorangestellten Leerzeichens ausgegeben wird. Lassen Sie das Programm einmal laufen.

Ändern Sie Zeile 40 des Programms folgendermaßen:

```
40 D$ = "DAS IST DIE SPALTE MIT DER NUMMER VIER"
```

Starten Sie das Programm. Sie werden sehen, was passiert, wenn der Text länger als eine Zone ist; d. h. mehr als 10 Zeichen lang ist. In diesem Fall wird der Computer über die Zone hinaus in die darauffolgende Zone schreiben. Sie müssen vorsichtig sein, da solch ein langer Text eine geordnete Tabelle durcheinanderbringen kann. Wählen Sie deshalb sorgfältig das Separationszeichen aus.

Beispiele für die Benutzung des PRINT-Befehls

10 PRINT A	Auf dem Bildschirm wird die Zahl A ausgegeben.
10 PRINT A;B;C;D	Die Zahlen A, B, C, D werden in einer Zeile, jeweils zwei Leerzeichen voneinander getrennt, ausgegeben.
10 PRINT A,B,C,D	Die Zahlen A, B, C und D werden in den entsprechenden Zonen einer Bildschirmzeile ausgegeben.
10 PRINT A;B	A und B werden in einer Zeile durch zwei Leerzeichen getrennt ausgegeben; C und D stehen in der darauffolgenden Zeile.
20 PRINT C;D	
10 PRINT A,B	A, B, C und D werden in zwei Zeilen, jeweils in die ersten beiden Zonen einer Zeile ausgegeben.
20 PRINT C,D	
10 PRINT A\$	Die Zeichenkette A\$ erscheint auf dem Bildschirm und der Cursor springt zur nächsten Zeile.
10 PRINT A\$;B\$;C\$;D\$	Die Zeichenketten A\$, B\$, C\$ und D\$ werden ohne

	Leerzeichen hintereinander in einer Zeile ausgegeben.
10 PRINT A\$;B\$,C\$,D\$	Die Zeichenketten A\$, B\$, C\$ und D\$ erscheinen auf dem Bildschirm jeweils in den entsprechenden vier Zonen.
10 PRINT A\$;B\$ 20 PRINT C\$;D\$	Die Zeichenketten A\$ und B\$ werden in einer Zeile direkt hintereinander ausgegeben. In der nächsten Zeile folgt dann die Ausgabe der Zeichenketten C\$ und D\$ direkt hintereinander.
10 PRINT A\$,B\$ 20 PRINT C\$,D\$	Die Zeichenketten A\$, B\$, C\$ und D\$ werden in zwei Zeilen ausgegeben, wobei die einzelnen Zeichenketten in die entsprechenden Zonen geschrieben werden.

Ausgabe im Direktmodus. Sie können den PRINT-Befehl auch direkt ausführen, d. h. Sie müssen ihn nicht unbedingt innerhalb eines Programms verwenden. In diesem Fall tippen Sie einfach den Befehl PRINT und danach die entsprechende numerische oder Zeichenkettenvariable ein. Sie müssen darauf achten, daß Sie dem PRINT-Befehl keine Zeilennummer voranstellen. Wenn Sie

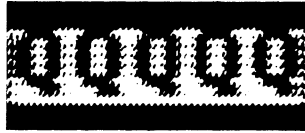
PRINT (oder ?) AB

eintippen, wird der Rechner sofort den entsprechenden Zahlenwert für AB ausgeben. Zusammen mit dem PRINT-Befehl können auch verschiedene Funktionen, einschließlich der CRSR-Funktion, der HOME-Funktion und der INST/DEL-Funktion verwendet werden.

In diesem Fall handelt es sich um eine Steuerung des Cursors mit dem PRINT-Befehl. Geben Sie folgendes Programm ein:

```
10 PRINT ZEILE 1  
20 PRINT (Drücken Sie die CRSR DOWN-Taste 5 mal)
```

Wenn Sie die Zeile 20 eingeben, werden Sie feststellen, daß der Cursor nicht wandert, sondern Sie werden fünf Symbole auf dem Schirm sehen können, die wie nachstehende Abbildung aussehen:



Tippen Sie die Zeile 30 des Programms ein:

```
30 PRINT "ZEILE 2"
```

Starten Sie jetzt das Programm. Das Programm wird den Text ZEILE 1, dann fünf Leerzeilen und schließlich ZEILE 2 ausgeben.

Auch die weiteren Kommandos zur Cursor-Steuerung (nach oben, unten, rechts und links) funktionieren entsprechend.

Fügen Sie nun in das bereits bestehende Programm eine weitere Zeile ein:

```
5 PRINT (Drücken Sie die SHIFT-Taste und anschließend die CLR/HOME-Taste)
```

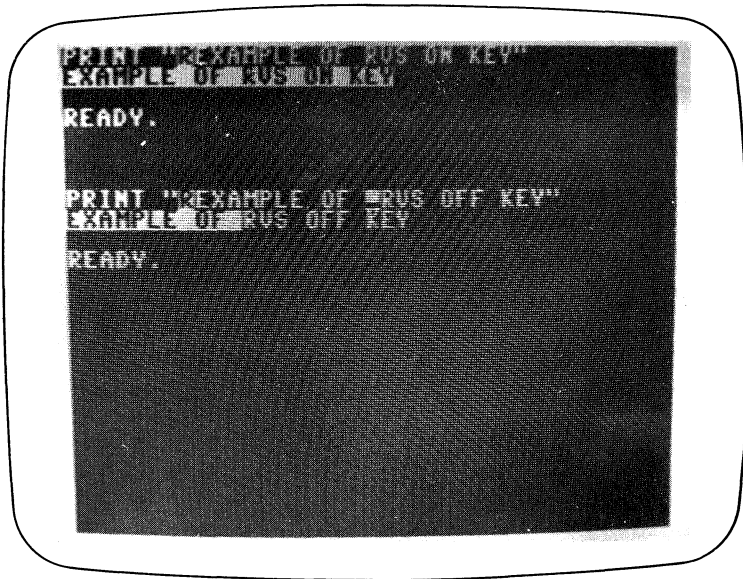
Sie werden jetzt ein anderes Symbol sehen, welches herzförmig ist. Starten Sie das Programm noch einmal. Mit dem Befehl in Zeile 5 wird der Bildschirm vor dem Start des Programms gelöscht. Der Cursor wird in die linke obere Ecke des Bildschirms bewegt.



Wenn Sie die CLR/HOME-Taste ohne die SHIFT-Taste gedrückt haben, werden Sie ein weiteres Symbol innerhalb der Anführungszeichen sehen. Dieses Sym-



bol ist ein S. Wenn Sie dieses Programm starten, wird der Cursor wiederum in die linke obere Ecke bewegt; der Inhalt des Bildschirm bleibt jedoch erhalten, er wird nicht gelöscht. Vielleicht haben Sie festgestellt, daß der Cursor mit diesen Befehlen nicht in die erste Zeile des Bildschirms, sondern in die zweite Zeile gesetzt wird. Wenn Sie nämlich kein Semikolon nach dem PRINT-Befehl in Zeile 5 schreiben, springt der Cursor immer in die zweite Zeile. Warum das so ist, haben Sie bereits im vorangegangenen Abschnitt erfahren.



Sie können die Ausgabe auf dem Bildschirm durch die Tasten RVS ON und RVS OFF beeinflussen. Tippen Sie ein Fragezeichen, dann ein Anführungszeichen ein, drücken Sie die CTRL- und RVS ON-Tasten gleichzeitig und geben Sie wiederum ein Anführungszeichen ein. Ein nachfolgender Text wird jetzt im 'Reverse-Modus' ausgegeben und zwar so lange, bis entsprechend der Eingabe von RVS ON der normale Ausgabemodus mit RVS OFF eingeschaltet wird. Der Reverse-Modus bedeutet, daß die Hintergrundfarbe und die Farbe zur Darstellung der Zeichen vertauscht werden. Schreiben Sie folgendes Beispielprogramm:

```
10 PRINT "NORMALMODUS (CTRL- und RVS ON-Taste drücken) REVERSE MODUS  
   (CTRL-Taste und RVS OFF drücken) NORMALMODUS"
```

Auf ähnliche Art und Weise können Sie auch die Farben auf dem Bildschirm einstellen. In diesem Fall schreiben Sie nach dem PRINT-Befehl Anführungszeichen, danach drücken Sie gleichzeitig die CTRL- und die Taste für die gewünschte Farbe. Beenden Sie die Zeile mit den Anführungszeichen. So schalten Sie beispielsweise mit dem Befehl

```
PRINT (CTRL- und WHT-Tasten drücken)
```

die Farbe Weiß ein. Sie werden sicherlich bereits festgestellt haben, daß der PRINT-Befehl in der Anwendung sehr kompliziert ist. Das liegt einfach daran, daß Sie mit dem PRINT-Befehl sehr viele Möglichkeiten haben. Die Möglichkeiten des PRINT-Befehls sind bei den einzelnen BASIC-Dialekten sehr unterschiedlich. Es gibt durchaus BASIC-Versionen, bei denen der PRINT-Befehl sehr einfach gehandhabt werden kann, da er nur wenige Anwendungsmöglichkeiten bietet.

5 Programmierung – eine Weiterführung

BASIC verfügt, wie jede natürliche Sprache, über einen Grundwortschatz, der aus den am häufigsten verwendeten Wörtern besteht. Mit den 10 BASIC-Begriffen INPUT, GOTO, IF und THEN, GET, FOR und NEXT, GOSUB und RETURN und REM, die in diesem Kapitel behandelt werden, können Sie schon eine ganze Menge anfangen, und bei Verwendung des bereits besprochenen PRINT-Befehls sind Sie sogar in der Lage, ein kleines Programm zu schreiben. Auf diese Art und Weise werden Sie mit der Logik des Programmierens vertraut.

Der Eingabebefehl INPUT

Der Befehl INPUT kann als Umkehrung des Befehls PRINT verstanden werden. Die meisten Regeln, die für PRINT gelten, gelten somit auch für INPUT. Der INPUT-Befehl dient zur Eingabe von Daten über die Tastatur. Wird eine INPUT-Anweisung ausgeführt, so erscheint auf dem Bildschirm ein Fragezeichen zum Zeichen dafür, daß nun Daten eingegeben werden müssen. Vergessen Sie bitte nicht, nach Eingabe der Daten die RETURN-Taste zu drücken. Der Computer fährt nun mit der Abarbeitung des Programms fort.

Die INPUT-Anweisung wird zusammen mit einer Variablen verwendet:

```
10 INPUT A
```

Wird dieses Programm gestartet, wird ein Fragezeichen ausgegeben und der Anwender muß jetzt ein Datum eingeben. Die eingegebene Zahl wird dann der Variablen A zugeordnet.

```
10 INPUT A$
```

Bei dieser Zeile wird wie bei obigem Programm verfahren, es muß nur anstelle einer Zahl eine Zeichenkette, also ein Text, eingegeben werden. Achten Sie darauf, daß dieser Text keine Kommata enthält. Das Komma dient nämlich bei der Eingabe mehrerer Variablen mit der INPUT-Anweisung als Separationszeichen. Der Rechner würde in diesem Fall meinen, daß Sie 2 Texte eingeben wollen. Da aber nur die Eingabe eines Textes verlangt wird, ignoriert der Rechner den Text, der nach dem Komma folgt. Es wird in diesem Fall eine Fehlermeldung ausgegeben.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
EXTRA IGNORED
```

Diesen Fehler erhalten Sie, wenn Sie mehr Daten eingeben, als von der entsprechenden INPUT-Anweisung gefordert wird. Wahrscheinlich haben Sie dann mehr Zahlen eingegeben, als numerische Variablen in der INPUT-Anweisung enthalten sind, oder Sie haben einen Text eingegeben, der Kommata enthält.

```
*****
```

Sie werden sehr bald feststellen, daß es für den Anwender eines Programms nicht sehr vorteilhaft ist, wenn lediglich durch die Ausgabe eines Fragezeichens ohne zusätzlichen erläuternden Text die Eingabe von Daten verlangt wird. Der Anwender weiß dann unter Umständen nicht, welche Daten er eingeben soll. Zu einer guten Benutzerführung gehört es nämlich, daß bei jeder Eingabezeile die einzugebenden Daten durch einen entsprechenden Text erläutert werden. Dieser Text kann sehr einfach mit der INPUT-Anweisung ausgegeben werden, wenn Sie ihn folgendermaßen schreiben:

```
10 INPUT "WELCHEN WERT WOLLEN SIE FUER A EINGEBEN";A
```

Auf dem Bildschirm erscheint die Zeile WELCHEN WERT WOLLEN SIE FUER A EIN-

GEBEN und der Computer wartet dann auf die Eingabe der geforderten Antwort. Wird die Eingabe mehrerer Daten verlangt, müssen alle Daten angegeben werden, bevor die RETURN-Taste gedrückt wird. Achten Sie bei dieser Form der INPUT-Anweisung darauf, daß der auszugebende Text in Anführungszeichen gesetzt wird. Danach folgt ein Semikolon.

Sollen in einer Zeile mehrere Daten eingegeben werden, müssen Sie die INPUT-Anweisung in der folgenden Form verwenden:

```
10 INPUT "GEBEN SIE BITTE DIE WERTE FUER A, B UND C EIN";A,B,C
```

In diesem Fall folgt nach dem Text eine Variablenliste. In dieser Variablenliste werden die einzelnen Variablen durch Kommata getrennt angegeben. Bei der Ausführung dieser Anweisung gibt der Rechner GEBEN SIE BITTE DIE WERTE FUER A, B UND C EIN aus. Nun wartet der Rechner auf Ihre Antwort. Geben Sie beispielsweise folgende Zeile ein:

```
100, 10, 25 (RETURN-Taste drücken)
```

Durch die Eingabe dieser Zeile wird der Variablen A der Wert 100, der Variablen B der Wert 10 und der Variablen C der Wert 25 zugewiesen. Die einzelnen Zahlen müssen also durch Kommata voneinander getrennt werden.

Wollen Sie diese drei Zahlen nicht in einer einzigen Zeile durch Kommata getrennt eingeben, sondern jeweils eine neue Zeile für jede Zahl benutzen, müssen Sie dieses Programm folgendermaßen schreiben:

```
10 INPUT "GEBEN SIE DEN WERT FUER A AN";A
20 INPUT "GEBEN SIE DEN WERT FUER B AN";B
30 INPUT "GEBEN SIE DEN WERT FUER C AN";C
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
REDO FROM START
```

Die Meldung REDO FROM START wird ausgegeben, wenn Sie anstelle der durch eine INPUT-Anweisung geforderten Zahl eine Zeichenkette eingeben. Korrigieren Sie diesen Fehler einfach, indem Sie eine Zahl eingeben.

```
*****
```

Wenn das Fragezeichen erscheint, und Sie tippen weder eine Zahl, noch eine Zeichenkette ein, sondern drücken lediglich die RETURN-Taste, wird angenommen, daß Sie entweder eine 0 (bei Eingabe einer Zahl) oder nichts (bei Eingabe einer Zeichenkette) eingeben wollen. Diese leere Zeichenkette wird oft auch als 'Nullstring' bezeichnet. Solch eine Zeichenkette kann z. B. auch folgendermaßen geschrieben werden:

```
A$ = ""
```

Zwischen den Anführungszeichen ist kein Leerzeichen eingegeben worden! Wie wir später noch feststellen werden, können Nullstrings sehr nützlich sein.

Um den INPUT-Befehl besser zu verstehen, sollten Sie das folgende Programm eingeben.

```
10 INPUT "WIE HEISSEN SIE"; NA$
20 INPUT "WIE ALT SIND SIE"; AL
30 PRINT "IHR NAME IST"; NA$
40 PRINT "SIE SIND"; AL; " JAHRE ALT"
```

Dieses Programm wird zuerst die Frage WIE HEISSEN SIE? ausgeben. Dann wartet der Computer auf Ihre Antwort. Nachdem Sie Ihren Namen eingegeben haben, wird die zweite Frage gestellt: WIE ALT SIND SIE? Nun müssen Sie Ihr Alter eingeben. Danach werden die Sätze SIE HEISSEN (eingegebener Name), und SIE SIND (Ihr Alter) JAHRE ALT ausgegeben.

Wie Sie sicherlich schon bemerkt haben, ist der INPUT-Befehl einfach und unkompliziert. Bei der Benutzung des INPUT-Befehls gibt es nur sehr wenig Fehlerquellen. Sie müssen aufpassen, wenn Sie den INPUT-Befehl in dem Direktmodus außerhalb eines Programms benutzen.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
ILLEGAL DIRECT
```

Sehr viele BASIC-Befehle können im Direktmodus benutzt werden. Sie brauchen damit also nicht Bestandteil eines Programms zu sein. Bei dem INPUT-Befehl ist jedoch die Anwendung des Direktmodus nicht möglich. Er kann nur innerhalb eines Programms verwendet werden. Überlegen Sie sich einmal,

warum das so ist.

GOTO

Ein sehr wichtiger Punkt beim Programmieren ist, daß die einzelnen Befehle eines Programms nicht unbedingt gemäß ihrer arithmetischen Reihenfolge bezüglich der Anweisungsnummern ausgeführt werden müssen. Der GOTO oder GO TO-Befehl gestattet es, zu einer bestimmten Anweisung zu springen. Er teilt damit dem Computer mit, welche Anweisung als nächste ausgeführt werden soll. GOTO kann ebenfalls für die Programmierung sogenannter 'Schleifen' verwendet werden. Hier ein Beispiel:

```
10 PRINT "DAS IST MEIN COMMODORE COMPUTER"
20 GOTO 10
```

In diesem Programm ist eine 'Endlosschleife' programmiert worden. Der Befehl GOTO 10 bedeutet nämlich, daß der Computer nach Abarbeitung der Anweisung mit der Nummer 10 die Anweisung Nummer 20 durchführt und aufgrund des GOTO-Befehls werden die Programmzeilen fortlaufend wiederholt.

Der Befehl GOTO wird aber nicht nur für die Programmierung von Schleifen verwendet.

```
10 PRINT "DAS IST MEIN COMPUTER"
20 GOTO 40
30 PRINT "DIESE ZEILE WIRD NICHT GESCHRIEBEN"
40 PRINT "DAS IST MEIN COMPUTER"
```

Zeile 20 gibt dem Computer den Befehl, direkt zu Zeile 40 zu springen. Da die Zeile 30 nicht ausgeführt wird, erscheint sie auch nicht auf dem Bildschirm.

Es ist natürlich nur Spielerei, Programme (wie das obige) mit überflüssigen Zeilen (Zeile 30) zu schreiben. Der GOTO-Befehl wird deshalb oft in Zusammenhang mit einer Bedingung benutzt. Diese Bedingung gibt dann an, ob der Sprung durchgeführt werden soll, oder nicht. Wir werden im nächsten Abschnitt mehr darüber erfahren.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
UNDEFINED STATEMENT ERROR
```

Nach den Befehlen GOTO und GOSUB müssen Anweisungsnummern angegeben werden, die wirklich im Programm vorkommen. Es kann hierbei leicht ein Fehler unterlaufen, wenn beispielsweise Zeilennummern geändert werden und die Änderung in den GOTO- und GOSUB-Befehlen, die sich auf diese Zeilen beziehen, vergessen wurde.

```
*****
```

IF, THEN und OR

Die Stärke des Computers besteht gerade darin, einfache Entscheidungen zu treffen, die auf Antworten zu einfachen Fragen oder Tests basieren. Nach dem Wort IF wird eine Bedingung angegeben. Nach THEN, das immer zusammen mit IF verwendet wird, folgt eine neue Anweisung, deren Ausführung von der IF-Bedingung abhängig ist. Mit den Ihnen bekannten Worten können Sie nachstehendes Programm bereits schreiben und verstehen:

```
10 INPUT "ANTWORTEN SIE MIT JA ODER NEIN. MOEGEN SIE COMPUTER ";A$
20 IF A$ = "JA" THEN PRINT "GUT . LERNEN SIE WEITER."
30 IF A$ = "NEIN" THEN PRINT "AUF WIEDERSEHEN."
```

Dieses Programm trifft eine Entscheidung. Durch die INPUT-Anweisung in Zeile 10 wird die Frage MOEGEN SIE COMPUTER ausgegeben. Die Antwort, die Sie eingeben, ist entweder JA oder NEIN und wird unter dem Variablennamen A\$ gespeichert.

Das Programm wählt unter drei Möglichkeiten eine aus und trifft somit seine Entscheidung. Zeile 20 fragt ab, ob die Antwort (A\$) JA ist. Wenn das der Fall ist, wird die Antwort GUT. LERNEN SIE WEITER ausgegeben. Wenn nicht, fährt der Computer mit der nächsten Zeile fort. Diese Zeile funktioniert in ähnlicher Weise; anstelle der Antwort JA wird hier die Antwort NEIN verarbeitet. Die dritte und letzte Möglichkeit ist, daß weder mit JA, noch mit NEIN, sondern mit irgendeinem anderen Wort geantwortet wurde. In diesem Fall wird keine Antwort ausgegeben.

Die IF-Anweisung kann sowohl mit Zahlen, als auch mit Zeichenketten verwendet werden. Sie können beispielsweise Zahlen vergleichen; Sie können aber auch prüfen, ob sich eine Zahl in einem bestimmten Bereich befindet. In diesem Fall verwenden Sie nicht das Gleichheitszeichen (=), sondern die Zeichen 'Kleiner als' (<) und 'Größer als' (>). Sehen wir uns einmal folgendes Beispiel an:

```
10 INPUT "WIE ALT SIND SIE";A
20 IF A < 18 THEN PRINT "SIE SIND UNTER 18 JAHRE ALT"
30 IF A > 18 THEN PRINT "SIE SIND UEBER 18 JAHRE ALT"
```

Wie bei dem vorangegangenen Programm ist auch bei diesem Programm die Wahl des auszugebenden Textes abhängig von der Antwort auf eine Frage. Dieses Beispiel unterscheidet aber nicht zwischen JA/NEIN, sondern ordnet die Antwort auf die Frage zwei Gruppen zu (unter 18 Jahre, über 18 Jahre). Die auszuführenden Anweisungen für A kleiner als 18 (A ist eine Zahl) sind in Zeile 20 angegeben; ansonsten wird Zeile 30 ausgeführt.

Eine problematische Stelle dieses Programms ist, daß Personen, die genau 18 Jahre alt sind, nicht berücksichtigt werden. Wird nämlich die Zahl 18 eingegeben, antwortet der Computer überhaupt nicht. Eine Möglichkeit, dieses Problem zu lösen, ist die zusätzliche Einfügung folgender Zeile:

```
40 IF A = 18 THEN PRINT "SIE SIND GENAU 18 JAHRE ALT"
```

Nun werden Sie in jedem Fall eine Antwort erhalten.

Da die 'Kleiner als' und 'Größer als'-Zeichen zusammen mit dem Gleichheitszeichen verwendet werden können, (was 'größer oder gleich' oder auch 'kleiner oder gleich' bedeutet), bietet sich eine weitere Möglichkeit an:

```
20 IF A <= 18 THEN PRINT "SIE SIND ENTWEDER GENAU 18 ODER UNTER 18"
30 IF A >= 18 THEN PRINT "SIE SIND ENTWEDER GENAU 18 ODER UEBER 18"
```

In Worten formuliert bedeutet Zeile 20: 'Wenn A kleiner oder gleich 18 ist; Zeile 30 bedeutet: 'Wenn A größer oder gleich 18 ist'.

Es ist möglich, IF-Anweisungen in einer Aussage ineinander zu schachteln.

```
10 INPUT "WIE ALT SIND SIE"; A
20 IF A >= 13 AND IF A <= 19 THEN PRINT "SIE SIND EIN TEENAGER."
30 IF A < 13 THEN PRINT "ZU JUNG."
40 IF A > 19 THEN PRINT "ZU ALT."
```

Die Zeilen 30 und 40 zeigen die bisherige Anwendung der IF-Anweisung. In Zeile 20 folgen zwei Bedingungen aufeinander. Diese Zeile wird folgendermaßen gelesen: Wenn A größer oder gleich 13 ist, und wenn A kleiner oder gleich 19 ist, dann drucke: SIE SIND... Die Antwort SIE SIND EIN TEENAGER wird nicht erscheinen, wenn auch nur eine dieser beiden Bedingungen nicht zutrifft. Das UND prüft jeweils, ob alle in der IF-Anweisung stehenden Bedingungen zutreffen. Das ODER (OR) prüft, ob eine der in den IF-Anweisungen formulierten Bedingungen wahr ist. Ein Beispiel für das ODER gibt folgendes Programm:

```
10 INPUT "WIE ALT SIND SIE?"; A
20 IF A <= 12 OR A >= 20 THEN PRINT "SIE SIND KEIN TEENAGER."
```

Die Zeichen 'Kleiner als' und 'Größer als' können auch zusammen verwendet werden. Dieses neue Zeichen bedeutet: 'Verschieden von' und ist somit die Umkehrung des '='-Zeichens. Nun ein kleines Ratespiel:

```
10 PRINT "ICH HABE MIR EINE ZAHL ZWISCHEN 1 UND 100 AUSGEDACHT."
20 INPUT "WIE HEISST DIE ZAHL? "
30 IF A <> 56 THEN PRINT "FALSCH":GOTO 10
40 IF A = 56 THEN PRINT "RICHTIG"
```

Dieses Spiel ist natürlich 'etwas' eintönig, da die richtige Antwort immer 56 ist. (Sie erhalten zunächst die Mitteilung: ICH HABE MIR EINE ZAHL ZWISCHEN 1 und 100 AUSGEDACHT. Mit der INPUT-Anweisung wird die Frage WIE HEISST DIE ZAHL ausgegeben. Sie müssen jetzt eine Zahl eingeben, die in dem vorgegebenen Bereich liegt. In Zeile 30 wird mit der IF-Anweisung (IF A <> 56) abgefragt, ob die eingegebene Zahl verschieden von 56 ist. Ist das der Fall, erhalten Sie die Meldung FALSCH, und das Spiel beginnt erneut. Ist die eingegebene Zahl 56, wird die Meldung RICHTIG ausgegeben. Damit ist das Spiel dann beendet. Die IF-Anweisung wird von dem Computer

streng logisch abgearbeitet und kann mit Variablen in fast jedem BASIC-Befehl verwendet werden. Viele Anfänger lassen sich dadurch verwirren, daß der Computer die ganze Programmzeile ignoriert, wenn die in der IF-Anweisung formulierte Bedingung nicht zutrifft. Aus diesem Grunde darf in einer IF-Zeile nichts stehen, was unabhängig von der Bedingung ausgeführt werden soll.

```
10 INPUT "JA ODER NEIN";A$  
20 IF A$ = "JA" THEN PRINT "SIE HABEN MIT JA GEANTWORTET" : A = 100
```

Ist die Antwort NEIN, erhält A den Wert 0; ist die Antwort JA, erhält A den Wert 100. In der Praxis kommt es öfters vor, daß wichtige Aussagen, die nicht von einer Bedingung abhängig sind, trotzdem in eine IF-Anweisung eingebaut werden. Diese Aussagen werden ignoriert, falls die Bedingung in der IF-Anweisung nicht zutrifft. Seien Sie deshalb im Umgang mit der IF-Anweisung etwas vorsichtig; Sie erhalten nämlich in einem solchen Fall keine Fehlermeldung, und das Auffinden dieses Fehlers ist mühsam und zeitaufwendig.

Weitere Details

Außer der INPUT-Anweisung kann auch die GET-Anweisung dazu verwendet werden, Informationen einzugeben. Wie bei der INPUT-Anweisung wartet der Computer auch bei der GET-Anweisung auf die Eingabe von Daten. Bei INPUT können jedoch mehrere Zeichen oder Zahlen eingegeben werden, bei GET hingegen kann jeweils nur ein Zeichen eingegeben werden. Ein weiterer Unterschied zur INPUT-Anweisung ist, daß bei der GET-Anweisung nach Eingabe des Zeichens die RETURN-Taste nicht gedrückt zu werden braucht. GET wird meist zusammen mit der IF..THEN Anweisung verwendet (siehe Beispiel). Nach GET steht wie bei der INPUT-Anweisung ein Variablenname.

```
10 PRINT "DRUECKEN SIE IRGEND EINE TASTE, UM DAS PROGRAMM ZU BEENDEN "  
20 GET C$:IF C$="" THEN GOTO 20  
30 PRINT "PROGRAMM BEENDET"
```

Diese Anweisungen kommen sehr oft in einem BASIC-Programm vor. Sie bieten die Möglichkeit, so lange mit der Abarbeitung des Programms zu warten, bis der Anwender eine Fortführung des Programms durch Eingabe eines Zeichens wünscht. Das ist besonders dann von Vorteil, wenn der Leser Zeit haben soll, die Information auf dem Bildschirm zu lesen, bevor sie gelöscht wird, oder wenn bei Spielprogrammen dem Benutzer vor dem nächsten Spiel eine Pause gönnt werden soll.

In diesem Beispiel wird der Satz DRUECKEN SIE IRGEND EINE TASTE, UM DAS PROGRAMM ZU BEENDEN ausgegeben. Mit GET wird die Ausführung des Programms so lange gestoppt, bis irgendeine Taste gedrückt wird. Durch die IF-Anweisung wird überprüft, ob der Wert von C\$ "" (Nullstring) ist. Die zwei aufeinanderfolgenden Anführungszeichen werden als Nullstring bezeichnet, d. h. - wie Sie sich vielleicht erinnern können - diese Zeichenkette enthält keine Zeichen (noch nicht einmal Leerzeichen). Wenn C\$ kein Zeichen (auch kein Leerzeichen!) enthält, wird die Zeile 20 wiederholt. Enthält C\$ irgendein Zeichen, d. h. es wurde irgendeine Taste gedrückt, wird die Ausführung des Programms mit der nächsten Zeile fortgesetzt.

Bei dieser Anwendung der GET-Anweisung ist es logischerweise egal, welche der Tasten - einschließlich der SHIFT-Tasten, der SHIFT LOCK-, RESTORE-, CTRL- und der COMMODORE-Taste - gedrückt wird. Die RUN/STOP-Taste gehört allerdings nicht dazu, da das Programm mit dieser Taste gestoppt werden würde.

Die GET-Anweisung dient natürlich nicht nur dazu, kleine Pausen in einem Programmablauf zu schaffen. Sie dient vielmehr dazu, einem Variablennamen eine Zahl oder eine Zeichenkette zuzuordnen.

```
10 PRINT "GEBEN SIE J FUER JA UND N FUER NEIN EIN"
20 GET A$: IF A$= "" THEN GOTO 20
30 IF A$= "J" THEN PRINT "IHRE ANTWORT WAR JA"
40 IF A$= "N" THEN PRINT "IHRE ANTWORT WAR NEIN"
50 IF A$ <> "J" OR IF A$ <> "N" THEN GOTO 10
```

In diesem Programmbeispiel wird unter der Zeichenkettenvariable A\$ ein Buchstabe gespeichert. Die Buchstaben J und N fungieren als Auswahlkriterium; von Ihnen hängt ab, was als nächstes getan werden soll. Falls der eingegebene Buchstabe weder J noch N ist, wird mit Zeile 50 wieder zum Anfang des Programms (d. h. zur Zeile 10 gesprungen). Dieser Programmab-

lauf wiederholt sich solange, bis entweder ein N oder ein J eingegeben wird.

Mit GET können Sie auch aus einer auf dem Bildschirm erscheinenden Liste einen bestimmten Punkt auswählen. Eine solche Liste wird im Computer-Jargon als 'Menü' bezeichnet. Wir wollen es einmal mit einem Menü versuchen, das uns vier verschiedene Auswahlmöglichkeiten bietet. Anstelle der vertrauten Ziffern-Tasten werden wir uns bei dieser Gelegenheit die bisher noch nicht besprochenen Funktionstasten auf der rechten Seite der Tastatur vornehmen. Die oberen Bezeichnungen dieser Tasten sind: 'f1', 'f3', 'f5' und 'f7'. Der Computer behandelt diese Tasten wie jede andere Taste, auch wenn beim Drücken dieser Taste auf dem Bildschirm nichts ausgegeben wird. In nachstehendem Beispiel ist die jeweils zu drückende Funktionstaste in Klammern angegeben. Schreiben Sie nicht den Inhalt der Klammern ab, sondern drücken Sie die entsprechende Funktionstaste!

```
10 PRINT "(Drücken Sie die SHIFT- und CLR/HOME-Taste gleichzeitig)"
20 PRINT "WAEHLN SIE IRGEND EINEN PUNKT AUS DER FOLGENDEN LISTE AUS"
30 PRINT "DRUECKEN SIE DIE FUNKTIONSTASTEN AUF DER RECHTEN SEITE DER
    TASTATUR"
40 PRINT "F1--DAS JAHR DER ERSTEN MONDLANDUNG"
50 PRINT "F3--DIE HOEHE DES EIFFELTURMS"
60 PRINT "F5--DIE LICHTGESCHWINDIGKEIT"
70 PRINT "F7--DIE TEMPERATUR VON KOCHENDEM WASSER"
80 GET C$: IF C$= "" THEN 80
90 IF C$="(f1-Taste drücken)" THEN PRINT "1969"
100 IF C$="(f3)" THEN PRINT "300 METER"
110 IF C$="(f5)" THEN PRINT "300 000 KILOMETER PRO SEKUNDE"
120 IF C$="(f7)" THEN PRINT "100 GRAD CELSIUS"
130 PRINT "DRUECKEN SIE IRGEND EINE TASTE, UM DAS PROGRAMM FORTZUSETZEN"
140 GET C$: IF C$= "" THEN 140
150 GOTO 10
```

Mit Zeile 10 dieses Programmbeispiels wird der Inhalt des Bildschirms gelöscht. Die Zeilen 20 und 30 geben Ihnen die Anweisung, einen Punkt aus der Liste auszuwählen und die entsprechende Funktionstaste zu drücken. Die Zeilen 40-70 enthalten das eigentliche Menü. GET in Zeile 80 bewirkt eine kleine Pause. Hier wird das Drücken der Funktionstaste abgewartet.

Wie Sie vielleicht bemerkt haben, fehlt in den Zeilen 80-140 nach THEN das Wort GOTO. Sie können das Wort GOTO in diesem Fall weglassen, da der Computer weiß, daß er bei der nach THEN folgenden Anweisung mit der Ausführung des Programms fortfahren soll. Durch die Anweisung IF C\$= ""THEN 80, wird das Programm mit Zeile 80 fortgesetzt. Die Angabe der Zeilennummer nach THEN reicht somit aus.

In Zeile 90 können Sie erkennen, wie die Funktionstasten arbeiten. Normalerweise wird auf dem Bildschirm kein Zeichen ausgegeben, wenn eine Funktionstaste gedrückt wird. Wenn Sie aber zuerst ein Fragezeichen (?) drücken, werden Sie sehen, daß für jede Funktionstaste ein bestimmtes Zeichen ausgegeben wird. Wenn Sie beispielsweise die f1- und die f3-Tasten drücken, werden Sie dann ein Rechteck sehen können, welches durch eine horizontale Linie geteilt ist. Auf jeden Fall wird das Drücken einer Funktionstaste so behandelt, als wenn Sie irgendein anderes Zeichen eingeben.

Wird eine Funktionstaste in dem obigen Beispiel gedrückt, wird die entsprechende Information (z. B. das Jahr der ersten Mondlandung, die Höhe des Eiffelturms usw.) ausgegeben. Anschließend erhalten Sie die Anweisung DRUECKEN SIE IRGEND EINE TASTE, UM DAS PROGRAMM FORTZUSETZEN. Wenn Sie nun irgendeine Taste drücken, wird das Programm erneut ausgeführt, da durch die Anweisung mit der Anweisungsnummer 150 (GOTO 10) wieder an den Anfang des Programms gesprungen wird.

Mit dem Befehl GET können Sie sowohl Zahlen, als auch Zeichen einlesen. Das bedeutet, daß GET im Zusammenhang mit numerischen und alphanumerischen Variablen benutzt werden kann. Wenn keine Zahl eingegeben wird, wird ähnlich wie bei der alphanumerischen Eingabe (Nullstring) eine Null angenommen. Nachfolgend ein kleines Beispiel:

```
10 PRINT "WAEHLEN SIE EINE ZAHL ZWISCHEN 5 UND 8"  
20 GET A:IF A<5 OR A > 8 THEN 20
```

In Zeile 20 wird überprüft, ob die eingegebene Zahl kleiner als 5 oder größer als 8 ist. Wird keine Zahl eingegeben, geht der Computer davon aus, daß A gleich Null ist. Da Null aber kleiner als 5 ist, wird die Zeile so oft wiederholt, bis eine Zahl zwischen 5 und 8 eingetippt wird.

Die For .. NEXT-Anweisung

Sie haben bereits die GOTO-Anweisung kennengelernt, mit der Sie Schleifen programmieren können. Sie haben gesehen, daß Schleifen unendlich fortgeführt werden, wenn keine besonderen Vorkehrungen getroffen werden.

Es gibt aber eine Möglichkeit, die Anzahl der Durchgänge in einer Schleife zu kontrollieren. Das geschieht mit der FOR..NEXT-Anweisung, die übrigens die einfachste Art ist, eine Schleife zu programmieren.

```
10 FOR I=1 TO 20
20 PRINT "DAS IST MEIN COMMODORE COMPUTER"
30 NEXT I
```

Schauen Sie sich einmal das erste kleine Programm an, das in diesem Buch vorgestellt wurde. Sie werden sicherlich eine Ähnlichkeit des obigen Programms mit diesem ersten kleinen Programm feststellen können. Bei der ersten Version dieses Programms wurde der Satz DAS IST MEIN COMMODORE COMPUTER so lange auf dem Bildschirm ausgegeben, bis die RUN/STOP-Taste gedrückt wurde. Das vorliegende Programm gibt den Satz jedoch nur 20mal aus.

Wie funktioniert dieses Programm?

In gewisser Weise ähnelt die FOR-Anweisung der Let-Anweisung, sie ist aber wesentlich bedeutungsvoller. Die FOR..NEXT-Anweisung definiert eine numerische Variable, die in diesem Fall mit I bezeichnet ist. Der Computer setzt nun für I den Wert 1 ein, durchläuft das Programm bis zu der Next-Anweisung, springt dann wieder zur FOR-Anweisung zurück und erhöht den Wert von I um 1. I nimmt somit nacheinander die Werte 1, 2, 3,...,20 an. Die Schleife wird so oft wiederholt, bis die Variable der FOR-Anweisung den nach TO angegebenen Wert erreicht.

Wenn Sie nachprüfen wollen, ob die Variablen I tatsächlich fortlaufend um 1 erhöht wird, müssen Sie nur Zeile 20 durch folgende Anweisung ersetzen:

```
20 PRINT I; "DAS IST MEIN COMMODORE COMPUTER "
```

Jede Zeile wird nun mit dem aktuellen Wert von I auf dem Bildschirm ausgegeben.

Nachstehendes Programm, das eine Multiplikationstabelle ausdruckt, ba-

siert auf folgender Überlegung:

```
10 PRINT "(SHIFT- und CLR/HOME-Tasten drücken)
   MULTIPLIKATIONSTABELLE FUER 18"
20 FOR I = 1 TO 20
30 PRINT I; "MAL 18 IST GLEICH ";I*18
40 NEXT I
```

Diese Art von FOR..NEXT-Schleifen können ineinander geschachtelt werden. Ineinander geschachtelte FOR..NEXT-Schleifen werden als 'FOR..NEXT-Nest' bezeichnet. Stellen Sie sich vor, Sie zählen 10mal bis 10. Das wäre schon ein solches FOR..NEXT-Nest. Geschachtelte FOR..NEXT-Anweisungen werden hauptsächlich dann verwendet, wenn eine Variable unabhängig von einer anderen um irgendeinen Wert erhöht werden soll. Wird in das vorangegangene (s. o.) Programm eine weitere Schleife eingefügt, gibt es sämtliche Multiplikationstabellen zwischen 1 und 100 aus.

```
10 FOR I=2 TO 100
20 PRINT "(SHIFT- und CLR/HOME-Tasten gleichzeitig drücken)
   MULTIPLIKATIONSTABELLE FUER";I
30 FOR J=1 TO 20
40 PRINT J; "MAL ";I;"IST GLEICH";J*I
50 NEXT J
60 PRINT "DRUECKEN SIE IRGENDEINE TASTE; UM DAS PROGRAMM FORTZUSETZEN"
70 GET C$: IF C$= "" THEN 70
80 NEXT I
```

Für Anfänger sind diese geschachtelten Schleifen sicherlich ein wenig verwirrend. Mit der Zeit werden Sie sich sicherlich daran gewöhnen. In Zeile 10 wählen Sie die Anzahl der Multiplikationstabellen aus. In unserem Fall sollen 99 Tabellen ausgegeben werden. Der Wert wird in der ersten (äußeren) Schleife gesetzt (FOR I = 2 TO 100). Zeile 20 gibt eine Titelzeile zu jeder Multiplikationstabelle aus. Wenn I den Wert 2 hat, wird der Computer die Zeile MULTIPLIKATIONSTABELLE FUER 2 ausgegeben.

Mit Zeile 30 beginnt die zweite (innere) Schleife: FOR J = 1 TO 20. Hiermit wird der Bereich für jede Tabelle angegeben. Zeile 40 druckt jede Zeile der Multiplikationstabelle aus. Zeile 50 ist das Ende der inneren

Schleife.

In den Zeilen 60 und 70 benutzen wir einen kleinen Trick, indem wir mit dem GET-Befehl das Programm anhalten. Damit bleibt die Tabelle auf dem Bildschirm bestehen, so daß wir sie in Ruhe lesen können. Wäre diese GET-Anweisung nicht in dem Programm vorhanden, würden die Tabellen so schnell auf dem Bildschirm ausgegeben werden, daß sie nicht mehr gelesen werden könnten.

Zeile 80 beendet die äußere Schleife. NEXT I gibt dem Computer die Anweisung, eine neue Multiplikationstabelle für eine neue Zahl auszugeben.

Die gewünschte Pause kann aber nicht nur mit der GET-Anweisung, sondern auch mit einer sogenannten 'Warteschleife' programmiert werden. Löschen Sie die Zeilen 60 und 70. Dazu müssen Sie die entsprechende Zeilennummer eingeben und danach die RETURN-Taste drücken. Geben Sie nun folgende Zeile ein:

```
60 FOR K=1 TO 1000:NEXT K
```

Jetzt haben Sie drei ineinandergeschachtelte und unabhängig voneinander arbeitende Schleifen. Die Schleife in Zeile 60 ist eine 'leere Schleife', da sie nur den Wert von K fortlaufend um 1 erhöht. Dadurch soll der Programmablauf verzögert werden (Warteschleife). Sie haben jetzt nämlich mehr als zwei Sekunden Zeit, jede Multiplikationstabelle zu lesen. Die Zeit wird kaum dazu ausreichen, jede Tabelle genau zu studieren; der Vorteil ist aber, daß der Programmablauf nicht unterbrochen wird und keine Taste zur Fortführung des Programms gedrückt werden muß.

Je geringer die Anzahl der zu durchlaufenden Schleifen für K ist, desto weniger Zeit wird zur Ausführung dieser Schleife (Zeile 60) benötigt. Geben Sie eine andere Zahl ein, z. B. 5000, und schauen Sie sich das Ergebnis an.

Nebenbeibemerkt wird Ihnen früher oder später vielleicht auffallen, daß zusammen mit der FOR..NEXT-Anweisung sehr oft die Variablen I, J und K verwendet werden. Diese Buchstaben besitzen eigentlich keine besondere Bedeutung. Es kann natürlich auch jede andere zulässige Variable verwendet werden.

Sie sollten sich ein paar wichtige Punkte im Zusammenhang mit FOR..NEXT-Schleifen merken:

1. Sie sollten niemals den 'roten Faden' beim Programmieren verlie-

ren. Sie könnten sehr schnell durcheinanderkommen, wenn Sie mehr als zwei ineinandergeschachtelte Schleifen verwenden.

2. Versuchen Sie GOTO und andere Sprunganweisungen in den Schleifen zu vermeiden.

Der Begriff STEP (= Schritt) wird nur im Zusammenhang mit der FOR..NEXT-Anweisung benutzt. Mit STEP wird festgelegt, wie groß die Schrittweite ist, um die eine Variable fortlaufend erhöht wird.

```
10 FOR I=1 TO 20 STEP 2
20 PRINT I; "DAS IST MEIN COMMODORE COMPUTER"
30 NEXT I
```

Bei diesem Beispiel beträgt die Schrittweite 2. I wird also fortlaufend um den Wert 2 erhöht. Wenn Sie dieses Programm laufen lassen, werden Sie sehen, daß die Zahlen 1, 3, 5, 7, 9 usw. erscheinen. Geben Sie nach STEP verschiedene Zahlen an, und schauen Sie sich dann das Ergebnis an.

Es ist zulässig, in einer Schleife eine negative Schrittweite anzugeben. Ändern Sie Zeile 10 wie folgt um:

```
10 FOR I = 20 TO 1 STEP -1
```

Aufgrund dieser negativen Zahl (-1) wird jede Zeile mit absteigender Reihenfolge (von 20 bis 1) numeriert.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
NEXT WITHOUT FOR ERROR
```

Sie erhalten diese Meldung, wenn Sie eine NEXT-Anweisung eingeben, ohne vorher die Schleife mit FOR geöffnet zu haben. Das kann Ihnen unbeabsichtigt passieren, wenn Sie in der NEXT-Anweisung eine unzulässige Variable verwenden.

```
10 FOR I = 1 TO 10
20 NEXT K
```

Der umgekehrte Fall ruft keine Fehlermeldung hervor: FOR wird ohne NEXT

verwendet. In diesem Fall läuft die Schleife nicht so ab, wie sie es sollte.

GOSUB und RETURN

GOTO ist ein 'vorwärtsgerichteter' BASIC-Befehl. GOTO bedeutet nämlich, daß der Computer das Programm nicht sequentiell in der Reihenfolge der Anweisungsnummern abarbeiten soll, sondern das Programm soll zu der nach dem GOTO-Befehl angegebenen Zeile verzweigen. Der GOSUB-Befehl ist ähnlich zu verstehen. Auch nach dem GOSUB-Befehl wird eine Zeilennummer angegeben, bei der die Abarbeitung des Programms fortgesetzt werden soll. Der Unterschied zwischen GOSUB und GOTO ist, daß bei der GOSUB-Anweisung nach dem Sprung gezielt wieder zu der Anweisung zurückgesprungen wird, die der GOSUB-Anweisung unmittelbar folgt. Der Rücksprung wird durch die RETURN-Anweisung veranlaßt. Das Wort GOSUB setzt sich aus den Worten Goto und SUBroutine zusammen (= Gehe zu dem Unterprogramm...).

Unterprogramme sind beim Programmieren von großer Bedeutung, da sie es ermöglichen, ein komplexes Programm in kleine überschaubare Teilprogramme zu zerlegen. Jedes Unterprogramm hat somit seine spezielle Funktion. Mithilfe der Unterprogrammtechnik kann vermieden werden, daß BASIC-Anweisungen in einem Programm an verschiedener Stelle wiederholt geschrieben werden müssen. Unterprogramme ermöglichen eine bessere Strukturierung der Programme, da ein komplettes Programm in ein Haupt- und mehrere Unterprogramme unterteilt werden kann.

Sobald der Computer einen GOSUB-Befehl antrifft, springt er zu der ersten Zeile dieses Unterprogramms und führt diese Anweisung und die darauf folgenden Anweisungen aus. Gelangt der Computer zu dem RETURN-Befehl, springt er wieder in das Hauptprogramm zurück. Die BASIC-Anweisung RETURN hat nichts mit der RETURN-Taste zu tun!

Wahrscheinlich werden Sie schon jetzt feststellen können, daß der GOSUB-Befehl wesentlich leistungsfähiger als der GOTO-Befehl ist.

Auf den vorangegangenen Seiten haben wir bereits sehr viel aus dem Programm gelernt, welches mit den Funktionstasten ein Menü aufbaute. Wir werden jetzt versuchen, dieses Wort unter Verwendung von Unterprogrammen zu schreiben.

Zunächst einmal schreiben wir ein Unterprogramm, das den Inhalt des Bildschirms löscht und das Menü ausgibt:

```
100 PRINT "(SHIFT- und CLR/HOME-Tasten drücken)"
110 PRINT "WAEHLEN SIE IRGEND EINEN PUNKT AUS DER FOLGENDEN LISTE  AUS"

120 PRINT "DRUECKEN SIE EINE DER FUNKTIONSTASTEN"
130 PRINT "F1--DAS JAHR DER ERSTEN MONDLANDUNG
140 PRINT "F3--DIE HOEHE DES EIFFELTURMS
150 PRINT "F5--DIE LICHTGESCHWINDIGKEIT"
160 PRINT "F7--DIE TEMPERATUR VON KOCHENDEM WASSER"
170 RETURN
```

Ohne RETURN in Zeile 170 würden diese Zeilen ein komplettes Programm darstellen. Wenn Sie versuchen, dieses Programm zu starten, werden Sie eine Fehlermeldung erhalten.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
RETURN WITHOUT GOSUB
```

Die GOSUB- und die RETURN-Anweisungen treten immer zusammen in einem Unterprogramm auf. Die Meldung RETURN WITHOUT GOSUB erscheint auf dem Bildschirm, wenn das Programm auf eine RETURN-Anweisung trifft, ohne daß dieser eine GOSUB-Anweisung vorangegangen ist. Eine Verwechslung des GOSUB- mit dem GOTO-Befehl ist sehr oft der Grund für diese Fehlermeldung.

```
*****
```

Wie bereits erwähnt, liegt der Vorteil eines Unterprogramms darin, daß dieses die häufige Wiederholung von Programmzeilen einspart und man somit das ganze Programm vereinfacht. In dem ersten Menü-Programm wurde der GET-Befehl zweimal verwendet. Auch dieses kann man wie folgt vereinfachen:

```
200 GET C$: IF C$ = "" THEN 200
210 RETURN
```

Damit haben wir nun zwei Unterprogramme: das Programm 100 gibt das Menü aus, und mit dem Unterprogramm 200 wird ein Zeichen eingelesen. Das vollständige Programm können wir nun folgendermaßen schreiben:

```
200 GET C$: IF C$ = "" THEN 200
210 RETURN
```

Damit haben wir nun zwei Unterprogramme: das Programm 100 gibt das Menü aus, und mit dem Unterprogramm 200 wird ein Zeichen eingelesen. Das vollständige Programm können wir nun folgendermaßen schreiben:

```
10 GOSUB 100
20 GOSUB 200
30 IF C$= "(f1-Taste drücken)" THEN PRINT "1969"
40 IF C$= "(f3)" THEN PRINT "300 METER"
50 IF C$= "(f5)" THEN PRINT "300 000 KILOMETER PRO SEKUNDE"
60 IF C$= "(f7)" THEN PRINT "100 GRAD CELSIUS "
70 PRINT "DRUECKEN SIE IRGEND EINE TASTE, UM DAS PROGRAMM FORTZUSETZEN"
80 GOSUB 200
90 GOTO 10

100 PRINT "(Drücken Sie die SHIFT- und die CLR/Home-Taste)"
110 PRINT "WAEHLEN SIE IRGEND EINE N PUNKT AUS DER FOLGENDEN LISTE AUS "
120 PRINT " DRUECKEN SIE DIE FUNKTIONSTASTEN AUF DER RECHTEN SEITE DER
    TASTATUR "
130 PRINT "F1--DAS JAHR DER ERSTEN MONDLANDUNG "
140 PRINT "F3--DIE HOEHE DES EIFFELTURMS"
150 PRINT "F5--DIE LICHTGESCHWINDIGKEIT"
160 PRINT "F7--DIE TEMPERATUR VON KOCHENDEM WASSER "
170 RETURN
200 GET C$:IF C$ = "" THEN 200
210 RETURN
```

Obwohl dieses Programm etwas anders aufgebaut ist, erzielt dieses Programm dasselbe Ergebnis wie die erste Version des Menü-Programms. Eine sehr nützliche Änderung ist, daß das Programm immer dann zu dem Unterprogramm in Zeile 200 springt, wenn der GET-Befehl verwendet wird.

Die Anwendung von Unterprogrammen kann sich bei falscher Benutzung als Fehlerquelle erweisen. Das Unterprogramm muß nämlich vom Hauptprogramm ganz klar getrennt werden. Nehmen Sie versuchsweise Zeile 90 aus dem Programm heraus (geben Sie nur die Zeilennummer ein und drücken Sie danach

die RETURN-Taste). Auf dem Bildschirm wird dann eine Fehlermeldung erscheinen. Das Programm wird nicht korrekt ablaufen, da der Computer nicht weiß, wo er anhalten soll.

Ein Unterprogramm sollte mit dem Wort END vom Hauptprogramm getrennt werden. END bildet zwischen dem Haupt- und dem Unterprogramm sozusagen eine 'Barriere'.

90 END

Jetzt läuft das Menü-Programm zwar logisch richtig ab, es wird aber nicht ohne Unterbrechung arbeiten. Wenn Sie das Unterprogramm wiederholt aufrufen, ist in Commodore-BASIC der END-Befehl nicht unbedingt notwendig. Das ist die einzige Ausnahme, die in der Praxis öfters vorkommt.

Stellen Unterprogramme nur eine Möglichkeit dar, Programme anders aufzubauen? Diese Frage muß verneint werden, da Unterprogramme nicht nur die Anzahl der Programmzeilen auf ein Minimum reduzieren, sondern auch das Programm lesbarer machen. Jedes Unterprogramm kann leicht identifiziert werden: Es beginnt mit der auf die GOSUB-Anweisung folgenden Zeile und endet mit der RETURN-Anweisung.

Es wird an BASIC-Programmen oft kritisiert, daß diese schwer lesbar seien. Es ist tatsächlich möglich, daß Sie Ihr eigenes Programm sechs Monate, nachdem Sie es geschrieben haben, nicht mehr lesen können. Das Einfügen von Unterprogrammen in Hauptprogramme wird als 'strukturiertes Programmieren' bezeichnet, da es das Hauptprogramm 'strukturiert'. Viele erfahrene Programmierer sind der Ansicht, daß strukturiertes Programmieren einem guten BASIC-Stil entspricht.

Die Kommentaranweisung REMark

Besonders Anfänger sollten darauf achten, daß ihre Programme leicht lesbar sind. Für jede Sprache (sei es nun Deutsch oder BASIC) sollte man sich einen guten Schreibstil aneignen. Ein BASIC-Ausdruck ermöglicht es Ihnen, in Ihr Programm Kommentare oder Bemerkungen einzufügen. Es handelt sich hier um den REM-Ausdruck (REM ist eine Abkürzung für REMark = Bemerkung). Einige Leute erklären den Ursprung dieses Ausdrucks mit dem englischen Wort REMember (= sich erinnern an), da sie sich selbst daran erinnern wol-

len, daß REM eine in ein Programm eingefügte Bemerkung ist. Grundsätzlich kann jeder Kommentar in ein Programm eingefügt werden; er muß lediglich mit REM gekennzeichnet sein.

10 REM DAS IST EINE KOMMENTARZEILE

REM kann am Ende einer Programmzeile stehen; wenn die REM-Anweisung von der restlichen Programmzeile durch einen Doppelpunkt getrennt ist. Die Anweisungen vor dem Doppelpunkt werden wie gewohnt behandelt; alle der REM-Anweisung folgenden Texte werden als Kommentare interpretiert und nicht vom Computer ausgeführt.

10 A=5:REM A WIRD AUF DEN WERT 5 GESETZT

Eine Besonderheit des Commodore-BASIC ist die Regelung, nur ungeshiftete Buchstaben und Zahlen in der REM-Anweisung zu verwenden. Die Probleme, die sich sonst ergeben können, sind leicht zu erkennen. Geben Sie folgende Zeile ein:

```
10 REM (Drücken Sie die SHIFT-Taste und schreiben Sie ABCDEFGHI)
Geben Sie nun den Befehl LIST ein und schauen Sie sich die Zeile an:
10 REM ATNPEEKLENSTR$VALASCCHR$LEFT$RIGHT$
```

Es scheint etwas ganz anderes zu sein, als das, was Sie unter REM eingegeben haben. Denken Sie bitte deshalb immer daran, daß niemals geshiftete Buchstaben oder Zahlen eingegeben werden dürfen.

Es ist noch gar nicht so lange her, daß der Arbeitsspeicher eines Computers sehr teuer und nur von begrenzter Kapazität war. Daher besaßen die Personal Computer relativ wenig Speicherkapazität und die Programmierer sahen sich gezwungen, mit jedem Byte sehr sorgsam umzugehen. Da jeder einzelne Kommentar viel Speicherplatz benötigt (pro Zeichen, Buchstabe oder Zahl ein Byte) wurde er nicht so häufig wie heute verwendet.

Heute ist die Frage des zur Verfügung stehenden Speicherplatzes von sekundärer Bedeutung. Der Commodore 64 besitzt 5mal mehr an Speicherplatz, als der Commodore PET. Mit zunehmender Speicherkapazität steigt ebenfalls die Anwendungshäufigkeit des Kommentarbefehls. In ein Programm eingebaute Kommentare ermöglichen für den Anwender und Programmierer ein einfacheres

Lesen dieser Programme. Wie bereits erwähnt, sollte uns der Einsatz des Computers das Leben erleichtern und nicht komplizieren. Was wäre uns damit geholfen, wenn ein wertvolles Lexikon, ein Nachschlagewerk oder eine Enzyklopädie unmöglich zu lesen und zu verstehen wäre? Denken Sie immer an diesen kleinen Vergleich, wenn Sie ein Computerprogramm schreiben, da dieses letztlich nichts anderes ist.

6 Wie der Computer Informationen speichert

Das Speichern von Informationen im Computer ist nicht schwer (Sie kennen den Vorgang bereits für Variablen). Die nächsten wichtigen Punkte sind: die Organisation der Information, Änderungen und die Ausgabe der gespeicherten Informationen. Diese Themen bilden den Schwerpunkt dieses Kapitels.

Das Abspeichern von Daten gehört zum praktischen Umgang mit dem Computer. So kann beispielsweise Ihre private Buchhaltung durch den Computer erledigt werden. Im Unterricht können Lehrer und Schüler den Computer verwenden, um Tests zu schreiben und deren Ergebnisse zu notieren. Am Arbeitsplatz ist eine sorgfältige Buchführung unbedingt notwendig. Fakten und Zahlen können nach Belieben eingegeben und geändert werden. Der Computer wird diese Arbeiten für Sie zuverlässig erledigen. Seine Speicherkapazität reicht normalerweise für die persönlichen Bedürfnisse völlig aus. Bei einem professionellen Einsatz des Computers wird meistens die Floppy Disk als Erweiterung des Arbeitsspeichers benutzt (die Speicherkapazität der Diskette beträgt mehr als die dreifache Speicherkapazität des Computers).

In diesem Abschnitt wird besprochen, wie auf der Datasette und der Floppy Disk Informationen gespeichert werden können. Sie werden außerdem etwas mehr über Variablen erfahren. Viele BASIC-Befehle werden zur Handhabung von Informationen verwendet. Zu diesen Befehlen gehören: DATA, READ, DIM, PRINT#, INPUT#, GET#, OPEN, CLOSE, RESTORE und CLR. Wie kann man Programme schreiben, die es erlauben, Informationen (Zahlen und Texte)

zu handhaben und zu speichern? Wir werden einige Programme kennenlernen und ausführlich besprechen.

Schließlich werden Sie auch noch einige Vorteile der kommerziell erhältlichen Programme kennenlernen, und Sie werden erfahren, was Sie beim Kauf dieser Programme beachten sollten.

Indizierte Variablen

Wie Sie sich erinnern können, sind Variablen Namen, die die im Computer abgespeicherte Information identifizieren. Sehen Sie sich folgende Beispiele an:

```
A$ = "AUTOTYPEN"  
B$ = "VOEGEL"  
C$ = "STAEDTE"  
D$ = "HUNDE"
```

Diese Liste ist, wie wir bereits wissen, eine Liste von Zeichenkettenvariablen, da jedem Buchstaben ein Dollar-Zeichen folgt. Jeder Variablen werden bestimmte Elemente zugeordnet: A\$ steht für Automarken, B\$ für Vögel usw. Unter jedem Variablennamen können wir zusätzliche Informationen speichern: verschiedene Autotypen, Vögel und Hunde- bzw. Stadtnamen.

Die Lösung dieses Problems ist, eine andere Variablenart zu benutzen, die als indizierte Variablen bezeichnet werden. Sie sollten jetzt nicht gleich in Erwartung eines schwierigen Kapitels das Buch fortlegen; so schwer ist es nämlich nicht. Ein Index ist eine Zahl, die in einer Gruppe von Elementen ein Element identifiziert. In der Mathematik sehen indizierte Variablen wie folgt aus:

$$A_6 \quad B_{25} \quad C_{100}$$

Die oben angegebenen Beispiele bedeuten: das sechste Element von A, das fünfundzwanzigste Element von B und das tausendste Element von C. Da der Computer zwar in der Lage ist, durch diese indizierten Variablen Informationen strukturiert zu speichern, Indizes aber nicht lesen kann, muß die indizierte Variable wie folgt geschrieben werden:

A(6) B(25) C(1000)

Diese Klammern geben an, welche Variablen indiziert bzw. durch eine Nummer identifiziert werden. A(6) wird 'A von sechs' gelesen. Das bedeutet, daß das sechste Element von A gemeint ist.

- 1 VW Polo
- 2 Opel Kadett
- 3 Ford Fiesta
- 4 Mercedes 280
- 5 Honda Civic
- 6 Morris Mini
- 7 Audi 100 LS
- 8 BMW 520
- 9 Fiat Ritmo
- 10 Renault R10

Diese Liste enthält Automobilnamen, die von 1 bis 10 durchnummeriert sind. Jeder einzelne Name kann mit A\$ aufgerufen werden, wenn diese Variable indiziert wird. Die jeweilige Position in der Liste entspricht der laufenden Nummer. Dem Namen 'Morris Mini' würde somit die indizierte Variable A\$(6) und dem Namen Fiat Ritmo die indizierte Variable A\$(9) zugeordnet werden.

Indizierte Variablen können für Zeichenketten (s. o.) oder für Zahlen definiert werden.

Um sich selbst einmal die Funktionsweise von indizierten Variablen zu verdeutlichen, sollten Sie nachstehendes Programmbeispiel eingeben:

```
10 A$ = "AUTOTYPEN"
20 A$(1) = "VW POLO"
30 A$(2) = "OPEL KADETT"
40 A$(3) = "FORD FIESTA"
50 A$(4) = "MERCEDES 280"
60 A$(5) = "HONDA CIVIC"
70 A$(6) = "MORRIS MINI"
80 A$(7) = "AUDI 100 LS"
90 A$(8) = "BMW 520"
100 A$(9) = "FIAT RITMO"
```

```
110 A$(10) = "RENAULT R10"  
120 PRINT "(CLR)"  
130 PRINT A$  
140 FOR I=1 TO 10  
150 PRINT A$(I)  
160 NEXT I
```

Dieses Programm ordnet der Variablen A\$ das Wort AUTOTYPEN zu. Dann wird von A\$(1) bis A\$(10) eine Liste von verschiedenen Autotypen angegeben. Beachten Sie bitte, daß A\$ nicht mit A\$(1) identisch ist.

Der Index innerhalb der Klammern kann jede positive Zahl zwischen 0 und 32 767 sein. Es ist aber auch möglich, als Index eine numerische Variable oder einen numerischen Ausdruck zu verwenden.

Die Zeilen 140-160 stellen eine einfache Schleife dar. In dieser Schleife FOR I=1 TO 10 wird jeder Autotyp entsprechend dem Index ausgedruckt.

Nachstehendes Programm gibt ein kleines Beispiel, wie Zahlen in einer indizierten Variablen gespeichert werden können.

```
10 FOR I = 0 TO 10  
20 A(I) = I + 25  
30 NEXT I
```

Die Schleife FOR I = 0 TO 10 ordnet der indizierten Variablen A() Werte zu. Um zu prüfen, um welchen Wert es sich handelt, fragen Sie direkt den Computer über die Tastatur.

```
PRINT (oder ?) A(5)
```

Die Antwort sollte 30 sein.

Die DIM-Anweisung

Um indizierte Variablen überhaupt verwenden zu können, muß in BASIC festgelegt werden, wieviele Elemente die indizierte Variable maximal umfassen soll. Wenn Sie die vorangegangenen Seiten aufmerksam gelesen haben, werden

Sie jetzt sicherlich stutzig werden. In den bereits besprochenen Programmen wurden indizierte Variablen verwendet, ohne daß zunächst eine bestimmte Anzahl festgelegt wurde. Fügen Sie diese Programmzeile in das zuletzt eingegebene Programm ein:

```
115 A$(11) = "CITROEN 2CV"
```

Ändern Sie Zeile 140 in

```
140 FOR I=1 TO 11
```

Starten Sie nun das Programm. Was passiert? Sie werden folgende Fehlermeldung erhalten:

```
?BAD SUBSCRIPT ERROR IN LINE 115
```

Wir werden mit dieser Zeile daran erinnert, daß wir vor dem Gebrauch von indizierten Variablen erst die Anzahl der uns zur Verfügung stehenden Elemente festlegen müssen. Warum erhalten wir diese Meldung aber erst jetzt? Die Antwort ist, daß in BASIC standardmäßig 10 Elemente verwendet werden können, ohne daß die gewünschte Anzahl der Elemente festgelegt werden muß.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
BAD SUBSCRIPT ERROR
```

Dieser Fehler bedeutet, daß wir einen unerlaubten Index benutzt haben. Steht in der DIMension-Anweisung beispielsweise DIM A\$(100), dürfen wir keine Variablen mit einem Index, der größer als 100 ist, verwenden. So wäre die Anweisung A\$(101) unzulässig. Dieser Fehler tritt aber auch auf, wenn wir keine DIM-Anweisung benutzt haben und bei indizierten Variablen einen Index, der größer als 11 ist, verwenden.

```
*****
```

Mit der DIM-Anweisung erhält der Computer die notwendige Information bezüglich der maximalen Elemente, die eine indizierte Variable umfassen soll. DIM ist die Abkürzung für DIMension und bezeichnet somit den Umfang

(oder Ausmaß) einer indizierten Variablen. Indizierte Variablen werden auch als 'Feld' bezeichnet. Die DIMension-Anweisung definiert also stets ein Feld.

```
10 DIM A$(100)
```

Mit dieser Anweisung werden 101 numerische Speicherplätze für die indizierte Variable A\$ freigehalten (auch Null kann verwendet werden ; deshalb sind 101 Elemente möglich). Von diesen Speicherplätzen können beliebig viele in Anspruch genommen werden; es dürfen jedoch auf keinen Fall mehr Elemente verwendet werden, als in der DIM-Anweisung vereinbart wurden. Ein Feld kann ein oder mehrere Dimensionen enthalten. In dem obigen Beispiel handelt es sich um ein eindimensionales Feld. Das bedeutet, daß die Variablen wie folgt angeordnet sind:

```
A$(1), A$(2), A$(3) ..., A$(100)
```

Ein Feld kann aber auch mehr als eine Dimension enthalten:

```
10 DIM A$(100,5)
```

Jede Variable A\$() wird nun durch zwei Indizes bestimmt. Man kann sich die einzelnen Elemente dieses Feldes schachbrettförmig angeordnet denken. Zur Veranschaulichung könnte man sie folgendermaßen schreiben:

```
A$(1,5), A$(2,5), A$(3,5).....A$(100,5)
A$(1,4), A$(2,4), A$(3,4).....A$(100,4)
A$(1,3), A$(2,3), A$(3,3).....A$(100,3)
A$(1,2), A$(2,2), A$(3,2).....A$(100,2)
A$(1,1), A$(2,1), A$(3,1).....A$(100,1)
A$(1,0), A$(2,0), A$(3,0).....A$(100,0)
```

Auch wenn Sie das Prinzip der mehrfach indizierten Felder nicht sofort ganz verstanden haben, sollten Sie sich nicht weiter daran stören. Mit zunehmender Programmiererfahrung wird Ihnen auch dieses Thema geläufig werden.

Jetzt wollen wir einmal ein zweidimensionales Feld anhand einer Tabelle veranschaulichen. Die in der Tabelle enthaltenen Informationen sind: die Monate von Januar bis April (fortlaufend beziffert) und die Wochen jedes Monats (ebenfalls von 1-4 beziffert). Für jede Woche eines Monats soll in der Tabelle eingetragen werden, wie oft der Personal Computer in Betrieb war.

Woche/Januar (1)	Februar (2)	März (3)	April(4)
1/25	15	30	17
2/19	6	11	22
3/13	17	5	3
4/15	9	8	21

Die Wochenzahlen stehen in den Spalten, die Monate in den ersten Zeilen der Tabelle. Die Information dieser Tabelle kann in einem zweidimensionalen Feld gespeichert werden.

```

10 DIM A(4,4)
20 A(1,1) = 25
30 A(1,2) = 15
40 A(1,3) = 30
50 A(1,4) = 17
60 A(2,1) = 19
70 A(2,2) = 6
80 A(2,3) = 11
90 A(2,4) = 22
100 A(3,1) = 13
110 A(3,2) = 17
120 A(3,3) = 5
130 A(3,4) = 3
140 A(4,1) = 15
150 A(4,2) = 9
160 A(4,3) = 8
170 A(4,4) = 21

```

Starten Sie dieses Programmbeispiel. Lassen Sie anschließend die Zahlen der Tabelle, die im Computer gespeichert sind, durch folgenden Befehl aus-

geben.

```
PRINT (oder?) A(Wochennummer, Monatsnummer)
```

Soll ausgegeben werden, wie oft der Computer in der ersten Woche des Monats Februar in Betrieb war, müssen Sie die Anweisung `PRINT 'A(1,2)` eingeben. Die ausgegebene Zahl sollte der Zahl 15 (siehe Tabelle) entsprechen.

Auch wenn Sie in Ihrem Programm die DIM-Anweisung nur mit eindimensionalen Feldern verwenden, sollten Sie sich immer überlegen, ob es nicht zweckmäßig wäre, mehrere dieser eindimensionalen Felder zu einem mehrdimensionalen Feld zusammenzufassen.

Mit einer DIMension-Anweisung können mehrere Variablen als Felder definiert werden:

```
10 DIM A$(100), B$(100), C$(100), D$(100)
```

Jedes dieser Felder enthält 100 Elemente. Bevor Sie ein Feld vereinbaren, müssen Sie aber genau wissen, wieviel Speicherplatz Sie für indizierte Variablen reservieren wollen. Nachträglich können Sie nämlich die Anzahl der reservierten Elemente nicht ändern, falls Sie es sich anders überlegen sollten.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR  
REDIM'D ARRAY ERROR
```

Die DIM-Anweisung darf nur einmal verwendet werden. Es ist zweckmäßig, sie an den Anfang des Programms zu setzen. Ist ein Feld definiert worden, kann es nicht undefiniert werden.

```
*****
```

Um Felder zu definieren, können auch Variablen verwendet werden.

```
10 INPUT "WIEVIELE AUTOTYPEN WOLLEN SIE SPEICHERN?";A  
20 DIM A$(A)
```

Bei diesem Programm kann der Benutzer eingeben, wie groß die Anzahl der

zu speichernden Automobilzahlen sein soll. Diese Anzahl wird unter dem Namen A gespeichert. Anschließend wird mit der DIMension-Anweisung genau die Anzahl der benötigten Elemente des Feldes A\$() festgelegt.

READ-DATA

Ein Programmbeispiel zeigte uns, wie eine Liste von Automobilnamen als indizierte Zeichenkettenvariable gespeichert wurde. Das war zwar die einfachste, aber nicht die beste Möglichkeit. Müssen die Namen der Liste geändert werden, so müßten ebenfalls sehr viele Programmzeilen geändert werden. Für die Bearbeitung einer größeren Informationsmenge wäre das natürlich sehr unpraktisch.

BASIC bietet für dieses Problem eine andere Lösung. Warum sollte diese Liste nicht direkt in dem Programm gespeichert werden? Alle Werte für die vorkommenden Variablen könnten dann direkt aus dieser Liste entnommen werden.

Die BASIC-Befehle READ und DATA ermöglichen dieses. Die mit DATA bezeichneten Zeilen enthalten die Informationen. Mit dem Befehl READ können nun Teile dieser Information aus der Liste in eine Variable gelesen werden.

Jedes einzelne Element der DATA-Zeile wird durch Kommata voneinander getrennt. Wir kennen das bereits, da ja auch bei einer Eingabezeile die einzelnen Datenelemente durch Kommata voneinander getrennt sind. Eine DATA-Zeile sieht folgendermaßen aus:

```
10 DATA 1,2,3,4,5,6,7,8,9,10,11,12,13
```

oder

```
10 DATA AUF, AB, RECHTS, LINKS, ZUR SEITE
```

Das Wort READ wird immer in Verbindung mit einer Variablen, die entweder eine numerische oder eine Zeichenkettenvariable sein kann, verwendet.

```
10 FOR I=1 TO 5
```

```
20 READ A$(I)
```

```
30 NEXT I
```

oder

```
10 FOR I=1 TO 5
20 READ N$
30 A$(I)=N$
40 NEXT I
```

Das Programmbeispiel, das eine Liste der Autotypen ausgibt, kann nun umgeschrieben werden:

```
10 DIM A$(11)
20 FOR I=1 TO 11
30 READ N$
40 A$(I)=N$
50 NEXT I
60 FOR I=1 TO 11
70 PRINT A$(I)
80 NEXT I
100 DATA VW POLO, OPEL KADETT, FORD FIESTA
110 DATA MERCEDES 280, HONDA CIVIC, MORRIS MINI, AUDI 100 LS
120 DATA BMW 520, FIAT RITMO, RENAULT R10
130 CITROEN 2 CV
```

Die DATA-Liste kann mehr Informationen enthalten, als durch die READ-Anweisung gefordert wird. In dem obigen Programm würden nur die ersten 11 Autotypen verwendet werden. Würde die DATA-Liste nur 10 Namen enthalten, würde das Programm mit einer Fehlermeldung unterbrochen werden.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
OUT OF DATA
```

Zählen Sie die Anzahl der Informationseinheiten in den DATA-Anweisungen. Prüfen Sie, wie oft die READ-Anweisung im Programm verwendet wird. Sie werden feststellen, daß mehr Daten eingelesen werden sollen, als in den DATA-Zeilen enthalten sind.

```
*****
```

In den DATA-Listen können numerische und Zeichenkettendaten nebeneinander

verwendet werden. Beachten Sie aber bitte immer, daß durch die READ-Anweisung immer der richtige Datentyp eingelesen wird. Als Beispiel soll das vorangegangene Programm erweitert werden. Neben den Autotypen sollen auch die Preise für die Autos in der Datenliste enthalten sein.

```
10 DIM A$(11),A(11)
20 FOR I=1 TO 11
30 READ A$(I),A(11)
40 NEXT I
50 PRINT "(CLR) AUTOPREIS"
60 FOR I=1 TO 11
70 PRINT A$(I), "$";A(I)
80 NEXT I
100 DATA VW POLO, 13000, OPEL KADETT, 13500
120 DATA HONDA CIVIC, 14100, MORRIS MINI, 10500
130 DATA AUDI 100 LS, 27000, BMW 520, 35000
140 DATA FIAT RITMO, 11500, RENAULT R 10, 14000, CITROEN 2 CV, 8700
```

In Zeile 10 werden zwei Felder definiert: A\$(11) für 11 Autonamen als Zeichenkettenvariable und A(11) für 11 Preise als numerische Variable. In den Zeilen 20-40 werden die Namen der Autotypen und die dazugehörigen Preise eingelesen. Mit Zeile 50 wird der Bildschirm gelöscht und die Titelseite AUTOPREIS ausgegeben. Durch die Zeilen 60-80 wird dann eine Liste der Autotypen mit den dazugehörigen Preisen auf dem Bildschirm erscheinen.

Wie Sie sehen, wird durch die Schleife 20-40 die Datenliste nacheinander eingelesen. Der Computer setzt somit einen Zeiger immer entsprechend um 1 weiter. Möchten Sie diese Datenliste von Anfang an lesen, können Sie durch die RESTORE-Anweisung den Zeiger auf das erste Element der ersten DATA-Zeile zurücksetzen. Es ist somit möglich, einen DATA-Block vom Beginn der DATA-Liste erneut einzulesen.

Mit dem Befehl CLR können Sie sämtliche Informationen, die in den Variablen gespeichert sind, löschen. Seien Sie aber im Umgang mit CLR etwas vorsichtig, da durch diesen Befehl die Information sämtlicher Variablen gelöscht wird. Wenn Sie nur die Information bestimmter Variablen löschen wollen, müssen Sie beispielsweise eingeben:

```
10 A$ = ""
```

oder

```
10 A=0
```

oder

```
10 FOR I=1 TO 10
```

```
20 A$(I)= ""
```

```
30 NEXT I
```

Beachten Sie bitte, daß der Befehl CLR nichts mit der CLR/HOME-Taste zu tun hat.

Datenspeicherung auf Datassette und Diskette

Bevor Sie jetzt weiterlesen, sollten Sie sich vergewissern, daß Sie den Abschnitt, in dem das Commodore Diskettenlaufwerk und die Datassette behandelt wurden, wirklich gelesen und verstanden haben.

Sobald die Information in den Computer eingelesen worden ist, muß sie irgendwie auf der Diskette oder der Datassette gespeichert werden. Andernfalls würde die Information verlorengehen, sobald der Computer ausgeschaltet würde. Eine Möglichkeit wäre, die Daten in einer DATA-Liste zu speichern. Die programminternen Daten würden dann direkt mit dem Abspeichern des Programms auf Diskette oder Datassette ebenfalls abgespeichert werden. Diese Methode ist jedoch für größere Datenmengen nicht geeignet. Computer wie der Commodore 64 werden als File-orientierte Computer bezeichnet. Das heißt, daß der Computer jede Information, die er erhält oder ausgeben soll, von einem File liest, bzw. auf einem File ausgibt. Bevor solch ein File benutzt wird, muß er geöffnet werden. Nach Benutzung des Files muß er wieder geschlossen werden. Das Öffnen geschieht mit dem Befehl OPEN, das Schließen mit dem Befehl CLOSE. Um auf einen File zu schreiben, benutzen Sie den Befehl PRINT#; um von einem File zu lesen, werden die Befehle INPUT# und GET# verwendet. Diese Befehle sind den Befehlen PRINT, INPUT und GET - wie wir sie bereits kennengelernt haben - sehr ähnlich.

Es gibt verschiedene Arten von Files, die der Commodore 64 auf der Datassette oder der Diskette einrichten kann. Die Information wird meistens in sequentiellen Files gespeichert. Die in diesen Files enthaltene Information wird nacheinander (sequentiell) gespeichert. Diese Datenfiles

sind von den Programmen verschieden und können deshalb nicht mit dem Befehl LOAD in den Computer geladen werden.

Mit dem BASIC-Befehl OPEN wird ein File geöffnet. OPEN kann für mehrere Zwecke verwendet werden; an dieser Stelle soll aber nur die Möglichkeit, sequentielle Files einzurichten, besprochen werden.

Nach OPEN folgt eine File-Nummer, eine Device-Nummer und eine Sekundäradresse.

```
10 OPEN 1,8,3
```

In diesem Beispiel ist 1 die File-Nummer, 8 die Device-Nummer und 3 die Sekundäradresse.

In einem Programm können bis zu 5 sequentielle Files gleichzeitig benutzt werden. Die Datassette, das Diskettenlaufwerk und der Drucker erhalten je eine Device-Nummer. Da der Computer die Worte wie Datassette, Diskettenlaufwerk und Drucker nicht 'verstehen' kann, muß für jedes dieser Geräte eine entsprechende Zahl vergeben werden. Die Device-Nummern sind bereits festgelegt. Die Datassette hat immer die Nummer 1, der Drucker stets die Nummer 4. Sollten Sie zwei Drucker an Ihrem Commodore 64 angeschlossen haben, erhält der zweite Drucker die Device-Nummer 5. Die Nummer 8 bezeichnet das Diskettenlaufwerk und die Nummer 9 - falls vorhanden - ein zweites Diskettenlaufwerk.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
NOT INPUT FILE ERROR  
NOT OUTPUT FILE ERROR
```

Man unterscheidet INPUT und OUTPUT-Files. Um welchen File es sich handelt, wird durch die Sekundäradresse festgelegt. Die Sekundäradresse 1 bedeutet, daß es sich um einen OUTPUT-File handelt. Sie können also in Verbindung mit diesem File die Befehle INPUT# oder GET# nicht benutzen.

```
*****
```

Wenn Sie mit Disketten-Files arbeiten, können Sie die Sekundäradresse frei wählen. Jeder Disketten-File sollte seine eigene Sekundäradresse besitzen. Sie können Informationen speichern oder wieder abrufen, ohne eine speziel-

le Sekundäradresse - wie sie bei der Datassette notwendig ist - zu benutzen.

Wenn Sie das Commodore Diskettenlaufwerk 1541 verwenden, sollten Sie nur die Zahlen 2 oder 3 als Sekundäradresse benutzen und nicht mehr als zwei Files gleichzeitig geöffnet haben. Das ist aber keine Einschränkung. Sie werden sehr bald feststellen, daß Sie fast alle Probleme mit nur einem einzigen Disketten-File bearbeiten können.

Die Sekundäradresse 15 wird zu einem speziellen Zweck benutzt. Sie können damit Kommandos an das Diskettenlaufwerk senden und die Art des Fehlers abfragen, wenn die rote Leuchtdiode blinkt.

Nach der File-Nummer, der Device-Nummer und der Sekundäradresse können wir einem File auch einen Namen geben. Mit diesem Namen kann der File identifiziert werden, wenn wir die Daten von der Datassette oder der Diskette abrufen wollen.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
FILE NOT OPEN ERROR  
FILE OPEN ERROR
```

Wenn Sie die Befehle PRINT#, INPUT# und GET# verwenden, müssen Sie vorher den File mit dem OPEN-Kommando geöffnet haben. Anderenfalls gibt der Computer die Fehlermeldung FILE NOT OPEN ERROR aus. Das kann ebenfalls vorkommen, wenn Sie das CMD-Kommando benutzen, um eine Information auf dem Bildschirm oder dem Drucker auszugeben und der File vorher nicht eröffnet wurde. Sollten Sie versuchen, einen File zu eröffnen, der bereits eröffnet wurde, erhalten Sie die Fehlermeldung FILE OPEN ERROR. Sie müssen auf jeden Fall einen File mit dem Befehl CLOSE schließen, bevor Sie ihn erneut eröffnen.

```
*****
```

Nachfolgend stehen einige typische Beispiele, wie ein File eröffnet werden kann. Schauen Sie sich die einzelnen Beispiele an, und versuchen Sie, diese zu verstehen.

OPEN 2,1,1,"TEST"

Auf der Datassette (Device 1) wird ein File mit der Nummer 2 eröffnet.

Dieser File dient zum Abspeichern von Daten (OUTPUT-File). Es handelt sich hierbei um einen sequentiellen File mit dem Namen TEST. Die File-Nummer 2 ist durch den Programmierer frei gewählt worden.

OPEN 3,1,0,"TEST"

Dieser Befehl eröffnet auf der Datensette einen sequentiellen INPUT-File mit der Nummer 3 und dem Namen TEST.

OPEN 1,1

Es wird auf der Datensette (Device 1) ein sequentieller INPUT-File mit der Nummer 1 eröffnet. Die File-Nummer ist vom Programmierer bestimmt worden.

OPEN 2,1,2,"ENDE DES BANDES"

Auf der Datensette (Device 1) wird ein OUTPUT-File mit der Nummer 1 und dem Namen ENDE DES BANDES eröffnet. Am Bandende wird eine bestimmte Marke, die das Ende des Files kennzeichnet, geschrieben.

OPEN 1,8,2,"TEST, SEQ, WRITE"

Hiermit wird auf der Diskette (Device 8) ein sequentieller OUTPUT-File mit der Nummer 1 und dem Namen Test eröffnet. Die File-Nummer und die Sekundäradresse sind vom Programmierer frei gewählt.

OPEN 3,8,3 "TEST"

Auf der Diskette (Device 8) wird ein sequentieller INPUT-File mit der Nummer 3 und dem Namen TEST eröffnet. Die File-Nummer und die Sekundäradresse sind vom Programmierer frei gewählt worden.

OPEN 2,8,2,A\$+"SEQ, WRITE"

Mit diesem Befehl wird auf Diskette (Device 8) ein sequentieller OUTPUT-File mit der Nummer 2 und dem Namen, der in der Zeichenkettenvariablen A\$ gespeichert ist, eröffnet. Filenummer und Sekundäradresse sind frei gewählt worden.

OPEN 1,9,2,"TEST, SEQ, WRITE"

Auf dem zweiten Diskettenlaufwerk (Device 9) wird ein sequentieller OUTPUT-File mit der Nummer 1 und dem Namen Test eröffnet. File-Nummer und Sekundäradresse sind frei gewählt worden.

OPEN 4,8,1,"0:"+A\$+"SEQ,WRITE"

Auf der Diskette (Device 8) wird ein sequentieller OUTPUT-File mit der Nummer 4 eröffnet. Der Name des Files ist in der Variablen A\$ gespeichert. Der File wird auf der Diskette #0 eines Doppeldiskettenlaufwerks eröffnet. Dazu ist ein IEEE-488 Interface-Adapter notwendig. File-Nummer und Sekundäradresse sind vom Programmierer frei gewählt worden.

OPEN 1,8,1,"TEST"

Es wird auf der Diskette (Device 8) ein sequentieller INPUT-File mit der Nummer 1 und dem Namen TEST eröffnet. Die Sekundäradresse ist frei gewählt worden.

OPEN 1,8,15

Auf Diskette (Device 8) wird ein File mit der Nummer 1 eröffnet. Damit soll ein Kommando dem Diskettenlaufwerk übermittelt werden, oder es soll die Art des Fehlers festgestellt werden. Die File-Nummer ist von dem Benutzer frei gewählt worden, die Sekundäradresse 15 ist jedoch nur für den oben genannten Zweck reserviert.

OPEN 1,4,3

Auf dem Drucker (Device 4) wird ein File mit der Nummer 1 und mit dem Zweck, Text auszugeben, eröffnet. File-Nummer und Sekundäradresse sind frei gewählt worden.

OPEN 4,4,7

Dieser Befehl eröffnet auf dem Drucker (Device 4) einen File mit der Nummer 4, um dem Drucker ein spezielles Kommando (Sekundäradresse) zu übermitteln. File-Nummer und Sekundäradresse sind frei gewählt.

Mit dem BASIC-Befehl PRINT# können Informationen der Datassette, der Diskette oder dem Drucker übermittelt werden. Die Befehle PRINT und PRINT# sind nicht identisch; auch wenn sie einander sehr ähneln. Mit PRINT werden Informationen auf dem Bildschirm ausgegeben. PRINT# gibt die Information (Zeichenketten oder Zahlen) auf einem File aus, der nach dem PRINT-Befehl folgt.

Wir haben bereits kennengelernt, daß Sie das Wort PRINT mit einem Fragezeichen ersetzen können. Es ist aber nicht möglich, PRINT# durch das

Fragezeichen in Verbindung mit dem '#'-Symbol (?#) zu ersetzen. Versuchen Sie, diesen Fehler zu vermeiden.

Ähnlich wie INPUT das Gegenteil von PRINT ist, ist INPUT# die Umkehrung von PRINT#. INPUT# liest Informationen von einem File der Datensette oder der Diskette ein. Mit diesem Befehl können Sie sowohl Zeichenketten als auch Zahlen einlesen.

Sie sollten darauf achten, daß innerhalb einer Zeichenkette kein Komma vorkommt. Der Computer und die Diskette benutzen das Komma zu speziellen Zwecken. Jede Information nach einem Komma wird ignoriert. GET# ist natürlich ähnlich wie GET zu verwenden. GET liest ein einfaches Zeichen über die Tastatur ein. GET# liest ein einzelnes Zeichen von einem sequentiellen File ein, der auf der Datensette oder der Diskette gespeichert ist.

Wir beginnen bei den sequentiellen Files mit dem BASIC-Befehl OPEN. Es ist eigentlich selbstverständlich, daß wir den File mit dem Befehl CLOSE schließen, wenn wir sämtliche Informationen eingelesen haben. Der Befehl CLOSE wird in Verbindung mit der File-Nummer verwendet, mit der der File eröffnet wurde.

Files auf der Datensette. Sequentielle Files werden auf der Datensette wie Programme gespeichert. Wie bereits erwähnt, können sie nicht mit dem Befehl LOAD geladen werden. Die auf der Datensette gespeicherte Information wird in der Reihenfolge, in der sie gespeichert wurde, wieder eingelesen. Deshalb ist das Programm zum Einlesen der Daten dem Programm, welches zum Speichern der Daten verwendet wurde, sehr ähnlich.

Wir wollen nun versuchen, die Daten des oben beschriebenen Programms auf einen sequentiellen File zu speichern.

Dazu wollen wir folgendes Programm benutzen:

```
10 DIM A$(11): A$= "AUTOTYPEN"
20 FOR I=1 TO 11
30 READ N$
40 A$(I)=N$
50 NEXT I
60 OPEN 2,1,1, "LIST"
70 PRINT#2,A$
80 FOR I=1 TO 11
```

```
90 PRINT#2,A$(I)
100 NEXT I
110 CLOSE 2
200 DATA VW POLO, OPEL KADETT, FORD FIESTA
210 DATA MERCEDES 280, HONDA CIVIC, MORRIS MINI, AUDI 100 LS
220 DATA BMW 520, FIAT RITMO, RENAULT R10
230 DATA CITROEN 2 CV
```

In Zeile 10 wird zunächst ein Feld definiert, um 10 Autotypen abspeichern zu können. In den Zeilen 20-50 werden die in der DATA-Liste enthaltenen Autotypen auf das Feld A\$ eingelesen. Nun können diese Daten gespeichert werden. In Zeile 60 wird auf der Datassette (Device 1) ein sequentieller File mit der Sekundäradresse 1 geöffnet. Die Ziffer 1 besagt, daß es sich um einen OUTPUT-File handelt (auf diesem File sollen Daten gespeichert werden). Der Name des Files ist LIST. Trifft der Computer auf diesen Befehl, wird auf dem Bildschirm die Meldung

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

ausgegeben. Die Schleife der Zeilen 80-100 schreibt die einzelnen Automotortypen auf den File. Zum Schluß wird der File mit der Nummer 2 in Zeile 110 durch den Befehl CLOSE geschlossen.

Wir haben nun die Daten auf der Datassette in einem sequentiellen File gespeichert. Wir können die Daten dieses Files jedoch nicht mit dem Befehl LOAD von der Datassette laden. Um diese Daten wieder abrufen zu können, benötigen wir ein kleines Programm. Geben Sie den Befehl NEW ein. Spulen Sie nun die Kassette zurück, und geben Sie folgendes Programm ein:

```
10 DIM J$(11)
20 OPEN 1,1
30 INPUT#1,J$
40 FOR I=1 TO 11
50 INPUT#1,J$(I)
60 NEXT I
70 CLOSE 1
80 PRINT "(CLR)"
90 PRINT J$
```



```
100 FOR I=1 TO 11
110 PRINT J$(I)
120 NEXT I
```

Das vorliegende Programm arbeitet wie folgt: Es dimensioniert ein Feld mit 11 Elementen für J\$() und öffnet dann auf der Datensette einen sequentiellen INPUT-File. Es ist nicht notwendig, eine Sekundäradresse anzugeben, da der File geöffnet wird, um die gespeicherte Information abzurufen. Die Sekundäradresse ist in diesem Fall 0. In der Schleife der Zeilen 40-60 wird nacheinander die in dem File enthaltene Information in die einzelnen Elemente des Feldes J\$() eingelesen. Vorher wird allerdings die Titelzeile der Tabelle von der Datensette gelesen. Dann wird der File mit Zeile 70 geschlossen. Um nachzuprüfen, ob die Information von der Datensette richtig gelesen wurde, werden anschließend in den Zeilen 90-120 die eingelesenen Daten wieder ausgegeben. Wie Sie erkennen können, wird in dem Programm, welches den File auf die Datensette schreibt, eine andere File-Nummer verwendet, als in dem Programm, das die Daten des Files von der Datensette liest. Dieses ist möglich. Es ist aber darauf zu achten, daß innerhalb eines Programms immer die gleiche File-Nummer verwendet wird. Sie sehen außerdem, daß wir bei beiden Programmen unterschiedliche Variablenamen benutzt haben. In dem ersten Programm haben wir die Variablen A\$ und A\$() genannt, in dem zweiten Programm wurden die Variablen mit J\$ und J\$() bezeichnet. Es werden also nur die Daten- und nicht auch die Variablennamen in dem sequentiellen File gespeichert.

Diskettenfiles. Wenn Sie die Handhabung der sequentiellen Files auf Datensette richtig verstanden haben, werden Sie auch keine Probleme mit Diskettenfiles haben.

Wenn Sie sich das Disketten-Directory richtig angesehen haben, haben Sie sicherlich auch drei verschiedene Abkürzungen in der rechten Spalte des Bildschirms feststellen können. Programme, die mit dem LOAD-Befehl geladen und mit RUN gestartet werden können, werden mit PRG bezeichnet. Ein sequentieller File wird dagegen mit SEQ bezeichnet.

Wie bei der Datensette muß auch in diesem Fall dem Diskettenlaufwerk mitgeteilt werden, daß es sich um einen sequentiellen File handeln soll. Das geschieht nicht durch eine bestimmte Sekundäradresse, sondern durch eine Textfolge. Auf den Befehl OPEN folgen immer eine File-Nummer (die

File-Nummer ist frei wählbar), eine Device-Nummer (in den meisten Fällen ist es die 8 für die Diskette) und eine Sekundäradresse (wählen Sie entweder eine 2 oder eine 3).

Nun ein Beispiel:

```
10 OPEN 1,8,3,"TEST,SEQ, WRITE"
```

Mit diesem Kommando wird dem Computer mitgeteilt, daß ein File mit der Nummer 1 auf Device 8 eröffnet werden soll. Der Name des Files ist TEST, und der File soll ein sequentieller File sein. Das Wort WRITE sagt aus, daß der File zum Speichern von Daten (OUTPUT-File) verwendet werden soll.

Wenn Sie an Ihren Commodore 64 zwei 1541 Diskettenlaufwerke angeschlossen haben, trägt das erste Diskettenlaufwerk die Device-Nummer 8; dem zweiten muß eine andere Device-Nummer zugeordnet werden (meistens erhält dieses Laufwerk die Device-Nummer 9). Sie können diese beiden Laufwerke nicht durch die Nummern 0 und 1 ansprechen.

Es ist allerdings möglich, der Form halber ein Präfix vor den File-Namen zu schreiben. Das Präfix muß von dem File-Namen durch einen Doppelpunkt getrennt sein. Wenn Sie anstelle eines sequentiellen Files auf der Datassette einen Diskettenfile für das obige Programm verwenden wollen, schreiben Sie für Zeile 60 folgende Anweisung:

```
60 OPEN 2,8,2, "0: LIST,SEQ,WRITE"
```

Das Programm, welches die Daten von dem File liest, muß dann folgendermaßen abgeändert werden:

```
20 OPEN 1,8,3, "0:LIST,SEQ"
```

Wie müssen Sie verfahren, wenn Sie nicht immer den gleichen File-Namen benutzen wollen? In diesem Fall kann die INPUT-Anweisung verwendet werden, um so den File-Namen in eine String-Variable einzulesen. Sie können dann diese String-Variable in dem OPEN-Kommando verwenden.

```
10 INPUT "WIE LAUTET DER FILENAME";N$  
20 OPEN 2,8,2,N$+",SEQ,WRITE"
```

Sie werden sicherlich bemerkt haben, daß es bei Diskettenfiles notwendig ist, den File-Namen anzugeben. Handelt es sich um einen File auf der Datassette, ist es nicht unbedingt notwendig, den File-Namen anzugeben. Wird der File-Name nicht angegeben, wird der erste File von der Datassette gelesen. Da eine Diskette sehr viele Files enthalten kann, ist es notwendig, den File-Namen immer als Information anzugeben.

Sie sollten unbedingt darauf achten, daß ein Disketten-File ordnungsgemäß mit dem Befehl CLOSE geschlossen wird. Indem Sie sich das Disketten-Directory ansehen, können Sie feststellen, ob ein File tatsächlich geschlossen ist. Erscheint mit dem File-Namen ein Sternchen (*), ist der File noch geöffnet. Ein solcher File ist nutzlos, da Sie auf diesen File keine Informationen schreiben können.

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
FILE NOT FOUND
```

Diese Meldung erscheint, wenn Sie ein Programm oder einen File von der Diskette laden wollen und dieses Programm oder dieser File nicht auf der Diskette abgespeichert ist. Sehen Sie sich das Disketten-Directory an, und überprüfen Sie, ob dieses Programm oder dieser File vorhanden ist.

```
*****
```

Zum Schluß noch etwas, was Sie unbedingt beachten müssen, wenn Sie mit Datassetten- oder Disketten-Files arbeiten. Sie haben schon etwas über eine String-Variable gehört, die keine Information enthält. Wir haben diese Zeichenketten als Null-Strings bezeichnet. Obwohl diese leeren Zeichenketten in BASIC erlaubt sind, kann es Probleme geben, wenn Sie versuchen, eine solche Zeichenkette auf einen sequentiellen File zu speichern.

```
10 A$= "JA":B$= "":C$="NEIN"
20 OPEN 1,8,2,"0:TEST,SEQ,WRITE"
30 PRINT#1,A$,B$,C$
40 CLOSE 1
```

Wenn Sie dieses Programm laufen lassen, wird der Computer das Abspeichern

von B\$ auslassen, da B\$ eine leere Zeichenkette enthält. Wenn Sie nun versuchen, die Daten des Files einzulesen, wird B\$ den Wert NEIN erhalten. Eine Möglichkeit, dieses Problem zu umgehen, wäre die Überprüfung, ob ein Null-String vorliegt oder nicht. Handelt es sich tatsächlich um eine leere Zeichenkette, soll diese durch irgendeine andere Zeichenkette ersetzt werden.

```
10 A$="JA":B$= "" : C$="NEIN"
20 OPEN 1,8,2,"0:TEST,SEQ,WRITE"
30 PRINT#1,A$
40 IF B$= "" THEN B$="***"
50 PRINT#1,B$
60 PRINT#1,C$
70 CLOSE 1
```

In diesem Beispiel wird in Zeile 40 überprüft, ob B\$ einen Null-String enthält. Ist das der Fall, wird der Null-String durch drei Sternchen ersetzt. Diese drei Sternchen sind nur ein Beispiel für eine Zeichenkette, die den Null-String ersetzen soll. Es kann natürlich auch jede andere Zeichenkette (z. B. "FFF" oder "???") gewählt werden.

Sollen die Daten des Files wieder geladen werden, müssen wir nachprüfen, ob ein Null-String vorhanden ist. Folgendes Programm hilft uns dabei.

```
10 OPEN 1,8,2, "TEST,SEQ"
20 INPUT#1,A$
30 INPUT#1,B$
40 IF B$="***" THEN B$= ""
50 INPUT#1,C$
60 CLOSE 1
```

In Zeile 40 wird überprüft, ob die Zeichenkette '***' ist. Wenn ja, wird B\$ ein Null-String zugewiesen.

Drucker-Files. Mit Ihrem Commodore 64 können Sie jede Information auf dem Drucker ausgeben, indem Sie einen File eröffnen und die Information auf den Drucker übertragen. Die Sekundäradresse kann dabei eine wichtige Rolle spielen, sie kann aber auch bei den einzelnen Druckern eine unter-

schiedliche Bedeutung haben. Ein Drucker kann beispielsweise durch die Sekundäradresse von dem Textmodus in den Grafikmodus umgeschaltet werden. Es existiert eine große Typenvielfalt der Drucker; aus diesem Grund kann hier nicht auf die verschiedenen Typen eingegangen werden.

Ein Drucker-File wird eröffnet, indem die File-Nummer und die Device-Nummer angegeben wird. Für den Ausdruck wird anstelle des PRINT-Befehls der PRINT#-Befehl verwendet.

```
10 OPEN 1,4
20 PRINT#1, "HALLO, MEIN DRUCKER ARBEITET!"
30 CLOSE 1
```

Ein Programm zur Adreßverwaltung

Ein Programm zur Verwaltung einer Datenbank erlaubt es, Informationen zu speichern und zu lesen. In Abhängigkeit von der jeweiligen Problemstellung können diese Programme einfach oder auch sehr kompliziert sein.

Am Ende dieses Kapitels werden Sie ein einfaches Programm zur Verwaltung einer Datenbank finden. Es bewerkstelligt die Verwaltung einer Adreßkartei. Sie können dieses Programm sogar praktisch einsetzen. Außerdem können Sie sehen, wie eine einfache Datenbank gehandhabt werden kann. Viele kommerziell erhältliche Datenbankprogramme sind wesentlich flexibler und bieten damit mehr Möglichkeiten. Das vorliegende Programm beschränkt sich darauf, die Informationen in dem Arbeitsspeicher des Computers zu speichern und auf einen sequentiellen File auszugeben.

Wenn Sie Anfänger sind, sollten Sie dieses Kapitel gut studieren, bevor Sie das Programm eingeben.

Bei der Eingabe des Programms müssen Sie darauf achten, daß Sie die Befehle genau so eingeben, wie sie hier abgedruckt sind. Wie bereits in früheren Beispielprogrammen erwähnt, müssen Sie die Tasten, die in Klammern angegeben sind, drücken.

Wenn Sie ein Programm eingeben, sollten Sie es öfters mit dem Befehl SAVE auf der Datensette oder der Diskette abspeichern. Sie stellen so sicher, daß die ganze Arbeit des Eintippens nicht umsonst war, falls der Computer versehentlich ausgeschaltet werden sollte. Sie können die Kassette zurückspulen und das vorher gespeicherte Programm überschreiben. Wenn

Sie das Programm auf der Diskette mit der neuen Version überschreiben wollen, müssen Sie den Befehl SAVE mit "@" verwenden. Schauen Sie in Kapitel 3 nach, falls Sie diesen Befehl vergessen haben sollten.

Wahrscheinlich werden Sie während der Eingabe des Programms einige Fehler machen. Wenn Sie eine ERROR-Meldung (= Fehlermeldung) bekommen, wird es meistens die Fehlermeldung SYNTAX ERROR sein. Das bedeutet, daß Sie einen formalen Fehler gemacht haben. Sollten Sie diese Fehlermeldung nicht verstehen können, schauen Sie einfach das Buch durch, und Sie werden diesen Fehler sicherlich finden.

Der Programmentwurf

Was erwarten wir von einem Programm, welches eine Adreßverwaltung vornimmt? Bevor wir mit dem Programmieren beginnen, sollten wir uns überlegen, was unser Programm eigentlich machen soll. Dazu stellen wir einen Plan auf. Dieser Plan enthält sämtliche Anforderungen, die unser Programm erfüllen soll.

Die Adreßverwaltung sollte das Abspeichern von Namen, Adressen und Telefonnummern ermöglichen. Wir sollten diese Informationen natürlich auch lesen, falls nötig ändern und sie auf Diskette oder Datensette speichern können. Weiterhin sollte es möglich sein, die Information auszudrucken. Schließlich müssen wir auch bestimmte Teile der abgespeicherten Information (oder sogar die ganze Information löschen können). Nachstehend finden Sie eine detaillierte Beschreibung der einzelnen Funktionen des Programms:

Enter. Dieser Teil des Programms dient zur Eingabe und Speicherung der Informationen.

Read. Mit diesem Programmteil können wir uns die ganze Information oder eine bestimmte Information ansehen.

Change. Die gespeicherte Information muß unter Umständen geändert werden, wenn sich z. B. die Telefonnummer oder die Anschrift ändert. Deshalb kann mit diesem Teil des Programms die Information geändert werden.

Erase. Wir sind natürlich bei der Kapazität des Arbeitsspeichers beschränkt. Deshalb können wir nur die wichtigste Information speichern. Von Zeit zu Zeit wird es deshalb notwendig sein, eine Adresse zu löschen, um Platz für neue, wichtigere Adressen zu schaffen.

Print. Dieser Teil des Programms wird benötigt, wenn wir einen Teil der Adressenliste über den Drucker ausgeben wollen. Es ist relativ einfach, die Informationen von dem File auf den Drucker zu übertragen.

Load und Save Auch wenn wir für diese Programmteile die Namen LOAD und SAVE verwenden, hat das nichts mit den BASIC-Befehl LOAD und SAVE zu tun. In diesen Programmteilen werden die Daten auf einen sequentiellen File gespeichert oder können von dem sequentiellen File wieder in den Arbeitsspeicher des Computers zurückgerufen werden.

Menü. Das Programm soll durch Menüs gesteuert ablaufen. Ein solches Programm wird die Funktionstasten benutzen. Ein durch Menüs gesteuertes Programm ist benutzerfreundlich und kann sehr einfach angewendet werden. Jedes der oben beschriebenen Programmteile kann über ein 'Hauptmenü' ausgewählt werden.

Vorüberlegungen

Die Erstellung einer Adressenliste ist nicht besonders schwer, da das Programm lediglich eine besondere Art von Informationen (Namen, Adressen, Telefonnummern usw.) abspeichert. Für jede Teilinformation (also für die Namen, Straßennamen, Hausnummern, Telefonnummern, Postleitzahlen usw.) benötigen wir ein Feld. Um die Anzahl der Eingaben feststellen zu können, verwenden wir eine numerische Variable als laufenden Zähler. Folgende Liste zeigt, wie die Information angeordnet werden kann.

N\$()-Namen

A\$()-Adressen (Straßennamen)

C\$()-Städte

S\$()-Länder

Z\$()-Postfächer

P\$()-Telefonnummern

Dadurch ist die Flexibilität des Programms limitiert. Wenn dieses Programm ganz allgemein geschrieben werden sollte, würde daraus ein ziemlich großer Aufwand an Zeit und Mühe resultieren.

Wir werden nun den Platz für 100 Eintragungen in die Adressenliste schaffen. Die numerische Variable N wird die Anzahl der Eintragungen zählen. Folgende Zeile ist notwendig, um den Zähler N auf den Anfangswert 0 zu setzen und die Felder zu dimensionieren:

```
15 X=101:DIM N$(X),A$(X), C$(X), S$(X), Z$(X), P$(X):N=0
```

Die Variable X steht stellvertretend für die Gesamtzahl der Eintragungen, die gespeichert werden sollen. So ist DIM N\$(X) identisch mit DIM N\$(101). Warum verwenden wir die Variable X anstelle der Zahl 101? Wir können die maximale Anzahl der Eintragungen ganz einfach dadurch ändern, indem wir X auf einen anderen Wert setzen.

Wir beginnen mit dem Programmieren

Die bei der Ausgabe auf dem Bildschirm verwendeten Farben sind bereits von Commodore eingestellt. Der Inhalt des Bildschirms ist aber mit diesen Farben etwas schwierig zu lesen. Wir können andere Farben auswählen, indem wir folgende Befehle eingeben:

```
20 POKE 53280,12:REM RAHMENFARBE GRAU 2
30 POKE 53281,15:REM HINTERGRUNDFARBE GRAU 3
```

Die Zahlen 12 und 15 werden mit dem Befehl POKE auf die Plätze des Arbeitsspeichers mit den Nummern 53280 und 53281 geschrieben. Diese beiden Speicherplätze sind für die auf dem Bildschirm ausgegebenen Farben verantwortlich. Dabei können Sie durch das Schreiben eines entsprechenden Wertes auf den Speicherplatz 53280 die Rahmenfarbe und durch Schreiben eines bestimmten Wertes auf den Speicherplatz 53281 die Hintergrundfarbe bestimmen. Sie können die Farben auswählen, indem Sie eine Zahl zwischen 0 und ~~51~~₁₅ angeben. Im Verlauf des Kapitels werden Sie eine Tabelle finden,

in der die Farben mit den dazugehörigen Zahlen enthalten sind.

Schauen wir uns einmal folgende Zeilen an:

```
40 PRINT CHR$(8)
50 PRINT CHR$(14)
```

In BASIC können durch den Befehl CHR\$() auf eine besondere Art und Weise Zeichen auf dem Bildschirm ausgegeben werden. Jedes Zeichen entspricht einer Nummer, die in Klammern hinter CHR\$ angegeben werden muß. CHR\$(8) und CHR\$(14) sind aber zwei spezielle Zeichen, die nicht auf dem Bildschirm ausgegeben werden.

Wird CHR\$(8) angegeben, schaltet der Computer vom 'Großschrift/Grafikmodus' in den 'Groß-/Kleinschriftmodus' um. Wie Sie wissen, können Sie dieses Ergebnis auch durch gleichzeitiges Drücken der SHIFT- und der Commodore-Taste erreichen. Mit der oben beschriebenen Methode können Sie aber mithilfe eines Programms die Betriebsarten umschalten.

In einem Menü-Programm werden alle Aktivitäten, die das Programm ausführt, durch einen Hauptteil gesteuert. In diesem Hauptteil wird lediglich der Text für das Menü ausgegeben, und es wird das Drücken einer Funktionstaste abgewartet. Dieser Hauptteil beginnt in Zeile 60 und endet mit einem GOTO 60 in Zeile 220. Immer wenn eine Taste gedrückt wird, wird das gewünschte Unterprogramm angesprungen.

Das Menü wird folgendermaßen ausgegeben:

```
f1--Enter
f3--Read
f5--Print
f7--Erase

f2--Load
f4--Save
f6--Change
```

Es folgt nun eine Liste der wichtigen Unterprogramme:

Zeile #	Zweck des Programms
10000	ENTER-Programm zur Eingabe von Daten
11000	READ-Liest Namen und Adressen
12000	PRINT-Druckt eine Liste auf dem Drucker
13000	ERASE-Löscht Namen/Adressen von der Liste
14000	LOAD-lädt einen File von der Diskette der oder Datensette
15000	SAVE-speichert einen File auf Diskette oder Datensette
16000	CHANGE-ändert Informationen

Die Benutzung des Programms

Das Programm ist sozusagen 'selbsterklärend', d. h. das Hauptmenü ist so aufgebaut, daß der Anwender ohne Kenntnis des Programms leicht verstehen kann, welche Funktionen er auf welche Art und Weise anwählen muß. Durch Drücken der entsprechenden Funktionstaste wird der Computer veranlaßt, das zu tun, was gerade gewünscht wird. Es existieren noch zwei weitere Befehle: SCAN und MATCH. Diese Befehle wurden nicht in der Liste aufgeführt, da sie Teile eines Hauptmenüs (also ein 'Untermenü') darstellen. Mit SCAN können Sie sich nacheinander alle Informationen ausgeben lassen, die in der Datei enthalten sind. Mit MATCH können Sie überprüfen, ob ein eingegebener Name in der Adressenliste vorkommt.

Bei dem Eingabemenü (ENTER) wird Ihnen das Programm sagen, welche Information Sie eingeben müssen. Wenn Sie Daten ändern wollen, werden Sie von dem Computer gefragt, welcher Art diese Daten sind.

Denken Sie immer daran, daß das Programm in BASIC geschrieben ist und deshalb typische Einschränkungen aufweist. Die wichtigste Einschränkung ist, daß eine Datenzeile kein Komma enthalten darf (schauen Sie sich vielleicht noch einmal das Kapitel, in dem die INPUT-Anweisung besprochen wurde, an).

Die Adresse

Becker, Maria
Poststraße 12
4600 Dortmund

muß ohne Komma wie folgt eingegeben werden:

Maria Becker
Poststraße 12
4600 Dortmund

Informationen können auch ausgelassen werden, indem ein Leerzeichen eingegeben wird oder einfach die RETURN-Taste gedrückt wird.

Seien Sie vorsichtig, wenn Sie innerhalb des Programms von einem Menü zu einem anderen springen. Es kann passieren, daß Sie dabei einen Fehler erzeugen. Wollen Sie beispielsweise einen File (Datensatz) von der Datassette laden, der nicht auf der Datassette abgespeichert ist, werden Sie damit einen Fehler erzeugen. Die Kassette läuft bis zum Bandende, ohne daß Sie Ihren gewünschten Datensatz eingelesen haben.

Sie werden sicherlich die Möglichkeit vermissen, mit vorliegendem Programm die Adressen sortieren zu können. Eine Sortierung kann in BASIC natürlich durchgeführt werden, aber sie würde sehr viel Zeit in Anspruch nehmen.

Eine weitere Einschränkung ist, daß nur sequentielle Files benutzt werden können. Die im Arbeitsspeicher enthaltenen Daten werden auf die Diskette oder auf die Datassette übertragen. In dem Programm können bis zu 600 String-Variablen benutzt werden. Diese Anzahl ist in der DIMension-Anweisung festgelegt. Jede Zeichenkette kann maximal 255 Zeichen enthalten. Das bedeutet, daß der Arbeitsspeicher des Computers zu klein ist, wenn sämtliche 600 String-Variablen die maximal mögliche Länge haben. Mit dem VC 64 kann aber noch eine andere Art von Files aufgebaut werden. Es handelt sich in diesem Fall um die 'relativen' Files. Unter Benutzung von relativen Files kann ein Teil dieser Einschränkungen umgangen werden. (Relative Files sind schwierig zu benutzen und zu programmieren. Sie würden den Rahmen dieses Buches sprengen. Sie können sich über dieses Thema anhand der Literatur für Fortgeschrittene informieren.)

Die Auswahl eines Datenbankprogramms

Wenn Sie sich ein ähnliches Datenbankprogramm (wie das vorliegende Adreßverwaltungsprogramm) kaufen wollen, sollten Sie sich dieses Programm sorg-

fältig ansehen.

Die erste Überlegung, die Sie anstellen müssen, ist, in welcher Sprache das Datenbankprogramm geschrieben ist. BASIC ist nur in den seltensten Fällen für ein Datenbankprogramm geeignet. Sie sollten sich kein BASIC-Programm kaufen, da diese Programme sehr langsam und nicht gerade flexibel sind. Die besten Datenbankprogramme sind in Maschinensprache geschrieben.

Die Flexibilität des Programms ist ein sehr wichtiger Punkt. Einige Datenbankprogramme besitzen zu viele Einschränkungen hinsichtlich des Typs, der Größe und der Länge der einzelnen Dateneintragungen.

Jedes gute Datenbankprogramm sollte die Möglichkeit bieten, die abgespeicherten Daten schnell zu sortieren. Weiterhin sollte es möglich sein, die Information nach mehreren Gesichtspunkten zu sortieren.

Die meisten guten Programme erlauben es, einen 'Report' zu drucken. Ein Report ist eine Liste von Informationen, die Sie sich aus der ganzen in der Datenbank gespeicherten Information herausgesucht haben. Ein komfortabler Reportgenerator wird Ihnen außerdem die Möglichkeit bieten, den Ausdruck zu gestalten.

Ein Datenbankprogramm für den Commodore Computer sollte immer relative Files benutzen.

Schließlich werden Sie sofort erkennen, ob es sich um ein gutes Datenbankprogramm handelt. Obwohl diese Programme sehr kompliziert sind und sehr viele Möglichkeiten bieten, sind sie in der Anwendung relativ einfach, wenn sie gut und logisch aufgebaut sind. Sie sollten nämlich mit diesem Programm arbeiten können, ohne daß ein Benutzerhandbuch griffbereit neben Ihrem Computer liegen muß.

```
10 rem ** adressverwaltungsprogramm **
15 x=101:dim n$(x),a$(x),c$(x),s$(x),z$(x),P$(x):n=0
20 poke 53280,12: rem rahmenfarbe grau 2
30 poke 53281,15: rem hintergrundfarbe grau 3
40 print chr$(14):rem gross/kleinschrift-modus
50 print chr$(8)
60 print"(clr) (blk) (rvs on) Adressverwaltung"
70 print:print"(rvs on)f1(rvs off) Enter"
80 print"(rvs on)f3(rvs off) Read"
90 print"(rvs on)f5(rvs off) Print"
100 print"(rvs on)f7(rvs off) Erase"
```

```
110 print:print"(rvs on) f2(rvs off) Load"
120 print"(rvs on)f4(rvs off) Save"
130 print "(rvs on)f6(rvs off) Change"
140 get c$:if c$="" then 140
150 ifc$="(f1)" then gosub 10000
160 ifc$="(f3)" then gosub 11000
170 ifc$="(f5)" then gosub 12000
180 ifc$="(f7)" then gosub 13000
190 ifc$="(f2)" then gosub 14000
200 ifc$="(f4)" then gosub 15000
210 ifc$="(f6)" then gosub 16000
220 goto 60
10000 rem ** enter **
10002 if n>=x-1 then return
10004 print "(clr) (rvs on) f1(rvs off) weiter"
10005 print"(rvs on)f8(rvs off) Hauptmenue"
10006 get c$: if c$="" then 10006
10007 if c$="(f1)"then 10010
10008 if c$="(f8)" then return
10009 goto 10000
10010 n=n+1
10020 print"(clr) (rvs on) Namen/Adressen eingeben"
10030 print"Adress-Nr. #";n;"von";x-1
10040 input"Name";n$(n)
10050 input"Adresse";a$(n)
10060 input"Stadt";c$(n)
10070 input"Land";s$(n)
10080 input"Postfach";z$(n)
10090 input"Telefonnummer";p$(n)
10100 print:print"(rvs on)Weitere Adresse (J/N)"
10110 get c$:if c$="" then 10110
10115 if n>=x-1 then return
10120 if c$="J" then goto 10010
10130 if c$="N" then return
10140 goto 10110
11000 rem ** read **
11005 if n=0 then return
```

```
11010 print"(clr) (rvs on)Lesen von Namen/Adressen"
11020 print
11030 print "(rvs on)f1(rvs off)saemtliche Adressen"
11040 print "(rvs on)f3(rvs off)Suchen von Adressen"
11050 get c$:if c$="" then 11050
11060 if c$="(f1)" then 11090
11070 if c$="(f3)"then 11220
11080 goto 11000
11090 for i=1 to n
11100 print"(clr)Adress-Nr. #";i;"von";n
11110 print"Name:           ";n$(i)
11120 print"Adresse:        ";a$(i)
11130 print"Stadt:         ";c$(i)
11140 print"Land:          ";s$(i)
11150 print"Postfach:      ";z$(i)
11160 print"Telefonnummer: ";p$(i)
11170 print:print"(rvs on)Druecken Sie irgendeine Taste um fortzufahren"
11180 get c$:if c$=""then 11180
11200 next i
11210 return
11220 input"(clr)Welcher Name soll gesucht werden";t$
11230 for i=1 to n
11240 if n$(i)=t$ then goto 11290
11250 next i
11260 print:print"(rvs on)Name nicht in der Datei vorhanden"
11270 print:print"irgendeine Taste druecken, um zum Hauptmenue zu kommen"
11280 get c$:if c$="** then 11280
11285 return
11290 print"(clr)Adress-Nr. #";i;"von";i
11300 print "Name:           ";n$(i)
11310 print "Adresse:        ";a$(i)
11320 print "Stadt:         ";c$(i)
11330 print "Land:          ";s$(i)
11340 print "Postfach:      ";z$(i)
11350 print "Telefonnummer  ";p$(i)
11360 print:print"(rvs on)irgendeine Taste druecken, um zum Hauptmenue zu
      kommen"
```

```
11370 get c$: if c$="" then 11370
11380 return
12000 rem ** print **
12004 print"(clr) (rvs on)f1(rvs off) weiter"
12005 print"(rvs on)f8(rvs off)Hauptmenue"
12006 get c$:if c$="** then 12006
12007 if c$="(f1)"then 12010
12008 if c$="(f8)"then return
12009 goto 12000
12010 print"(clr) (rvs on)Ausdruck"
12020 open 1,4,4
12030 for i=1 to n
12040 print#1,n$(i)
12050 print#1,a$(i)
12060 print#1,c$(i)
12070 print#1,s$(i)
12080 print#1,z$(i)
12090 print#1,p$(i)
12100 print#1,chr$(13)
12110 next i
12120 close 1
12130 return
13000 rem ** erase **
13002 if n=0 then return
13004 print"(clr) (rvs on)f1(rvs off) weiter"
13005 print"(rvs on)f8(rvs off) Hauptmenue"
13006 get c$: if c$="" then 13006
13007 if c$="(f1)"then 13010
13008 if c$="(f8)"then return
13009 goto 13000
13010 print"(clr) (rvs on)Loeschen von Namen/Adressen"
13030 print"(rvs on)f1(rvs off)alle Adressen"
13040 print"(rvs on)f3(rvs off)einen bestimmten Namen"
13050 get c$:if c$="" then 13050
13060 if c$="(f1)"then 13090
13070 if c$="(f3)"then 13230
13080 goto 13000
```

```
13090 for i=1 to n
13100 print"(clr)Adress-Nr. #";i;"von";n
13110 print "Name:          ";n$(i)
13120 print "Adresse:         ";a$(i)
13130 print "Stadt:           ";c$(i)
13140 print "Land:            ";s$(i)
13150 print "Postfach:        ";z$(i)
13160 print "Telefonnummer:   ";p$(i)
13170 print:print"(rvs on)f1(rvs off)weiter"
13180 print:print"(rvs on)f8(rvs off)loeschen"
13190 get c$:if c$=""then 13190
13200 if c$="(f1)" then goto 13225
13210 if c$="(f8)" then gosub 13800:goto 13224
13220 goto 13190
13224 n=n-1:i=i-1
13225 next i
13226 return
13230 input"(clr)Welcher Name soll gesucht werden";t$
13240 for i=1 to n
13250 if n$(i)=t$ then gosub 13800:n=n-1:i=i-1
13260 next i
13265 return
13270 print:print"(rvs on)Name nicht in der Datei vorhanden"
13280 print:print"irgendeine Taste druecken, um zum Hauptmenue zu kommen"
13290 get c$:if c$="" then 11280
13800 for k=i to n
13810 n$(k)=n$(k+1)
13820 a$(k)=a$(k+1)
13830 c$(k)=c$(k+1)
13840 s$(k)=s$(k+1)
13850 z$(k)=z$(k+1)
13860 p$(k)=p$(k+1)
13870 next k
13880 return
14000 rem ** load **
14004 print"(clr)(rvs on)f1(rvs off) weiter"
14005 print"(rvs on)f8(rvs off) Hauptmenue"
```



```
14006 get c$: if c$="" then 14006
14007 if c$="(f1)" then 14010
14008 if c$="(f8)" then return
14009 goto 14000
14010 print"(clr) (rvs on) Laden des Files"
14020 print:print"(rvs on)f1(rvs off)  Laden von Diskette"
14030 print"(rvs on)f4(rvs off) Laden von Datassette"
14040 get c$:if c$="" then 14040
14050 if c$="(f1)" then 14100
14060 if c$="(f4)" then 14250
14070 goto 14040
14100 open 1,8,3,"0:mail list,seq"
14140 input#1,n
14150 for i=1 to n
14160 input#1,n$(i)
14170 input#1,a$(i)
14180 input#1,c$(i)
14190 input#1,s$(i)
14200 input#1,z$(i)
14210 input#1,p$(i)
14220 next i
14230 close 1
14240 return
14250 open 1,1,1,"mail list"
14260 goto 14140
15000 rem ** save **
15002 if n=0 then return
15004 print"(clr) (rvs on)f1(rvs off) weiter"
15005 print"(rvs on)f8(rvs off) Hauptmenue"
15006 get c$: if c$="" then 15006
15007 if c$="(f1)"then 15010
15008 if c$="(f8)"then return
15009 goto 15000
15010 print"(clr) (rvs on)Speichern des Files"
15020 print:print"(rvs on)f1(rvs off)Speichern auf Diskette"
15030 print"(rvs on)f4(rvs off)Speichern auf Datassette"
15040 get c$:if c$="" then 15040
```

```
15050 if c$="(f1)"then 15100
15060 if c$="(f4)"then 15250
15070 goto 15040
15100 open 1,8,15
15110 print#1,"s0:mail list"
15120 close 1
15130 open 1,8,3,"0:mail list,seq,write"
15140 print#1,n
15150 for i=1 to n
15160 print#1,n$(i)
15170 print#1,a$(i)
15180 print#1,c$(i)
15190 print#1,s$(i)
15200 print#1,z$(i)
15210 print#1,p$(i)
15220 next i
15230 close 1
15240 return
15250 open 1,1,1,"mail list"
15260 goto 15140
16000 rem ** change **
16002 if n=0 then return
16004 print"(clr) (rvs on)f1(rvs off) weiter"
16005 print"(rvs on)f8(rvs off) Hauptmenue"
16006 get c$: if c$="" then 16006
16007 if c$="(f1)"then 16010
16008 if c$="(f8)"then return
16009 goto 16000
16010 print"(clr) (rvs on)Aendern einer Adresse"
16020 print:print"(rvs on)f1(rvs off) alle Adressen"
16030 print"(rvs on)f3(rvs off)Aendern eines bestimmten Namens"
16040 get c$:if c$="" then 16040
16050 if c$="(f1)"then 16100
16060 if c$="(f3)"then 16300
16070 goto 16040
16100 for i=1 to n
16110 print"(clr)";n$(i)
```

```
16120 print a$(i)
16130 print c$(i)
16140 print s$(i)
16150 print z$(i)
16160 print p$(i)
16170 print:print"(rvs on)f1(rvs off)weiter"
16180 print"(rvs on)f3(rvs off)Aendern"
16190 get c$:if c$=""then 16190
16200 if c$="(f1)" then goto 16230
16210 if c$="(f3)" then gosub 16800:goto 16230
16220 goto 16190
16230 next i
16240 return
16300 print"(clr)"
16310 input"Namen eingeben";m$
16320 for i=1 to n
16330 if n$(i)=m$ then gosub 16500:return
16340 next i
16350 print"(rvs on)Name nicht in der Datei vorhanden"
16360 print"(rvs on)irgendeine Taste druecken, um zum Hauptmenue zu
        kommen"
16370 get c$:if c$=""then 16370
16380 return
16500 print"(clr)";n$(i)
16510 print a$(i)
16520 print c$(i)
16530 print s$(i)
16540 print z$(i)
16550 print p$(i)
16800 print"1 -- Name"
16810 print"2 -- Adresse"
16820 print"3 -- Stadt"
16830 print"4 -- Land"
16840 print"5 -- Postfach"
16850 print"6 -- Telefonnummer"
16860 print:input "Was soll geaendert werden?";c
16870 on c gosub 16910, 16920, 16930, 16940, 16950, 16960
```

```
1688Ø return
1691Ø print:input"Aenderung";d$:n$(i)=d$:i=i-1:return
1692Ø print:input"Aenderung";d$:a$(i)=d$:i=i-1:return
1693Ø print:input"Aenderung";d$:c$(i)=d$:i=i-1:return
1694Ø print:input"Aenderung";d$:s$(i)=d$:i=i-1:return
1695Ø print:input"Aenderung";d$:z$(i)=d$:i=i-1:return
1696Ø print:input"Aenderung";d$:p$(i)=d$:i=i-1:return
```

7 Programmierung für Fortgeschrittene

Sie werden sicherlich überrascht sein, wenn Sie feststellen, daß Sie schon sehr viele Programme mit den wenigen BASIC-Befehlen schreiben können, die Sie bisher kennengelernt haben. Es gibt aber noch weitere Befehle.

Dieses Kapitel wird Sie weiter in die Programmiersprache BASIC einführen. Sie werden lernen, was PEAK und POKE, SYS und USR, CMD, SPC und TAB, STOP und WAIT und ON bedeuten.

Wenn Sie Ihren Computer für mathematische Zwecke verwenden wollen, werden Sie etwas über die mathematischen Funktionen INT, ABS, SGN, SQR, SIN, COS, TAN, ATN, DEF und FN erfahren.

Sie werden außerdem noch einiges zum Thema Textverarbeitung erfahren und den Umgang mit den Befehlen LEN, VAL, STR\$, ASC, CHR\$, LEFT\$, MID\$ und RIGHT\$ lernen.

Wenn Sie all das aufmerksam gelesen haben, werden Sie allmählich einen guten Überblick über die Programmiersprache BASIC erhalten.

PEEK und POKE

Die Befehle PEEK und POKE sind hinsichtlich ihrer englischen Bedeutung leicht zu verstehen. Mit PEEK kann man sich den Inhalt eines Speicherplatzes im Arbeitsspeicher des Computers ansehen. POKE schreibt ein Datum in eine Speicherzelle des Arbeitsspeichers. Diese beiden Befehle werden sehr oft dem Befehlssatz der Maschinensprache zugeordnet, da sie einen direkten

Zugriff auf eine Speicherzelle des Arbeitsspeichers erlauben.

Der Befehl POKE ist wichtig, falls Sie grafische Darstellungen in Farbe bzw. Musik mit dem Computer machen wollen.

Nach dem Befehl POKE steht immer die Nummer einer Speicherzelle. Danach folgen ein Komma und eine Zahl (oder eine numerische Variable, die den Wert spezifiziert, der auf den betreffenden Speicherplatz geschrieben werden soll). Mit dem POKE Befehl können nur Zahlen zwischen 0 und 2⁵5 auf einen Speicherplatz geschrieben werden. (Warum das so ist, erfahren Sie später). Der Commodore 64 hat 64 KB Arbeitsspeicherkapazität. Das bedeutet, daß insgesamt 65 536 verschiedene Speicherzellen vorhanden sind. Die Zahl, die direkt auf den POKE Befehl folgt, muß deshalb eine ganze Zahl zwischen 0 und 65 535 sein. Die Nummer des Speicherplatzes kann auch als Variable angegeben werden.

POKE kann in einem Programm oder im Direkt Modus benutzt werden.

POKE 53272,23

oder

10 POKE 53272,23

In diesem Beispiel wird die Zahl 23 auf den Speicherplatz 53272 geschrieben. Damit wird der Computer vom Grafik-/Großschrift-Modus in den Großschrift-/Kleinschrift-Modus umgeschaltet. Wird auf die Speicherzelle 53272 eine 21 geschrieben, schaltet der Computer wieder in den Großschrift-/Grafik-Modus zurück. Mit dieser Speicherzelle wird also der Video-Chip angesprochen.

Auf ähnliche Art und Weise können Sie noch mehr im Computer beeinflussen. Wenn Sie den Befehl POKE verwenden, schauen Sie sich die Speicherbelegung des Commodore 64 (am Ende des Buches) an. In der Zwischenzeit sollten Sie einmal Zahlen von 0 bis 15 auf die Speicherplätze 53280 und 53281 schreiben und sich dann das Ergebnis auf dem Bildschirm ansehen.

Benutzen Sie den Befehl PEEK immer im Zusammenhang mit Nummern von Speicherplätzen, die zwischen 0 und 65535 liegen. Die Nummer des Speicherplatzes sollte in Klammern angegeben werden. Sie können den Befehl PEEK entweder im Direkt-Modus oder innerhalb eines Programms verwenden. Mit dem Befehl

PRINT PEEK(53272)

können Sie sich den Inhalt der Speicherzelle 53272 per Bildschirm ausgeben lassen.

Wenn Sie den Befehl PEEK in einem Programm benutzen, können Sie natürlich den Inhalt dieser Speicherzelle einer numerischen Variablen zuordnen.

```
10 A=PEEK(53272)
```

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR  
ILLEGAL QUANTITY ERROR
```

Wenn dieser Fehler in einem Programm mit der POKE-Anweisung auftritt, sollten Sie überprüfen, ob die Zahl, die auf den Speicherplatz geschrieben werden soll, zwischen 0 und 255 liegt.

```
*****
```

Die Befehle PEEK und POKE können ein Programm sehr unübersichtlich machen, da Sie nicht direkt sehen, was passiert. Bei häufiger Verwendung von PEEK und POKE wird ein Programm zu anderen Computern inkompatibel. In einigen Fällen kommen Sie nicht umhin, PEEK und POKE zu verwenden. Sie sollten aber die Anwendung dieser Befehle nach Möglichkeit vermeiden.

SYS und USR

SYS steht stellvertretend für SYStem und wird benutzt, um von einem BASIC-Programm ein Unterprogramm aufzurufen, das in Maschinensprache geschrieben ist. Die Maschinensprache ist eine Programmiersprache, die relativ schwer zu erlernen ist. Sie besteht aus elementaren Befehlen, die direkt vom Computer verstanden werden. Der Vorteil von Programmen, die in Maschinensprache geschrieben wurden, ist, daß sie erheblich schneller als in BASIC geschriebene Programme ausgeführt werden. Es kann somit vorkommen, daß in Maschinensprache geschriebene Programme 100mal schneller ausgeführt werden, als die entsprechenden BASIC-Programme.

Auf den Befehl SYS folgt in Klammern die Nummer des Speicherplatzes, auf dem das in Maschinensprache geschriebene Unterprogramm steht. SYS kann in einem Programm oder im Direkt-Modus verwendet werden. Zum Beispiel:

SYS(2061)

oder

10 SYS(2061)

Nachdem das in Maschinensprache geschriebene Unterprogramm ausgeführt worden ist, wird die Ausführung des BASIC-Programms fortgesetzt.

USR steht für USer (= Benutzer) und ist dem Befehl SYS ähnlich. USR ist allerdings komplizierter als SYS anzuwenden.

10 USR(10)

Wenn der Befehl USR benutzt wird, springt das BASIC-Programm in ein Maschinenprogramm, dessen Startadresse in den Speicherplätzen 785 und 786 abgelegt ist. Der in Klammern nach USR angegebene Wert wird an das Maschinenprogramm übergeben.

Da sowohl SYS als auch USR sehr starken Bezug zur Maschinensprache haben, sind sie für den Anfänger schwer verständlich. Eventuell werden Sie später manchmal den Befehl SYS anstelle des RUN-Befehls verwenden, um ein Maschinenprogramm zu starten.

CMD

CMD ist eine Abkürzung für CoMmand (= Befehl). Mit dem Befehl CMD werden Ausgaben, die normalerweise auf dem Bildschirm erscheinen würden, auf einem anderen Gerät (Drucker oder Datenfile auf Band oder Diskette) ausgegeben. Das Gerät und der File müssen vorher mit OPEN geöffnet werden. Dem CMD-Befehl muß eine ganze Zahl oder eine numerische Variable folgen, die sich auf die Filenummer bezieht.

Nun ein Beispiel, aus dem Sie entnehmen können, wie der Befehl CMD arbeitet. Die Befehle geben die Liste eines BASIC-Programms auf dem Drucker aus. Sie können die einzelnen Befehle entweder im Direkt-Modus eingeben oder in einem Programm anwenden.

OPEN 1,4,4

CMD 1

LIST

CLOSE 1

Der OPEN-Befehl eröffnet eine Datei mit der Nummer 1 auf dem Gerät mit der Nummer 4 (Drucker) und der Sekundäradresse 4. CMD 1 bezieht sich auf die File-Nummer. LIST würde normalerweise das Programm auf dem Bildschirm ausgeben. Mit CMD wird aber die Ausgabe auf den Drucker umgeschaltet. Der File mit der Nummer 1 wird durch den Befehl CLOSE 1 geschlossen.

SPC und TAB

Beide Befehle werden in Verbindung mit dem PRINT-Befehl verwendet. SPC steht für SPaCe (= Leerzeichen) und TAB bedeutet TABulator.

```
10 PRINT SPC(17) "TEST"
```

oder

```
10 PRINT TAB(17) "TEST"
```

Jeder dieser Befehle bewirkt, daß der Cursor um 17 Leerstellen von seiner momentanen Position nach rechts wandert und dann das Wort TEST ausgegeben wird. TAB und SPC unterscheiden sich dadurch, daß TAB nicht in Zusammenhang mit PRINT# (d. h. in Verbindung mit einem File) verwendet werden kann. Die Benutzung von SPC in Verbindung mit PRINT# ist hingegen zulässig.

STOP und WAIT

STOP kann zur Überprüfung von Programmen verwendet werden. Sie können den Befehl STOP an irgendeiner Stelle in Ihrem Programm einbauen. Wenn der Rechner auf den Befehl STOP trifft, erhalten Sie dasselbe Ergebnis, als wenn Sie die RUN/STOP-Taste gedrückt hätten. Sie werden auf dem Bildschirm die Meldung

```
BREAK IN LINE 100 (oder irgend eine andere Zeilennummer)
```

erhalten. Sie wissen damit, in welcher Zeile das Programm angehalten wor-

den ist. Das Programm kann nach Eingabe von CONT fortgesetzt werden. (Wenn Sie den Befehl RUN eingeben, beginnt der Computer mit der Ausführung des Programms wieder bei der ersten Zeile.)

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
CAN'T CONTINUE ERROR
```

Der Befehl CONT kann verwendet werden, um mit der Ausführung eines Programms fortzufahren, nachdem die RUN/STOP-Taste gedrückt wurde oder das Programm auf den Befehl RUN/STOP trifft. Wurde eine Programmzeile geändert, kann das Programm mit dem Befehl CONT nicht fortgeführt werden, und der obige Fehler tritt auf.

```
*****
```

WAIT wird benutzt, um ein Programm so lange anzuhalten, bis eine bestimmte Speicherzelle einen bestimmten Wert angenommen hat. Viele Programmierer verwenden den WAIT-Befehl nicht sehr gerne, und sogar Commodore rät, diesen Befehl nicht anzuwenden. Wenn Sie trotzdem unbedingt wissen wollen, wie der Befehl arbeitet, schauen Sie in dem Programmiererhandbuch von Commodore nach.

ON

Der Befehl ON bietet Ihnen beim Programmieren sehr viele Möglichkeiten. Obwohl er sehr einfach anzuwenden ist, meiden sehr viele Programmierer diesen Befehl.

ON wird mit numerischen Variablen und mit dem GOTO- oder GOSUB-Befehl benutzt. Eine typische Anwendung ist diese:

```
20 ON A GOSUB 100,150,180,220
```

Wenn A den Wert 1 hat, springt der Rechner zu dem Unterprogramm, das mit der Zeile 100 beginnt. Hat A den Wert 2, wird der Rechner zu dem Unterprogramm mit der Zeile 150 springen. Für einen Wert von 3 für A wird der Rechner das Unterprogramm, das mit Zeile 180 beginnt, ausführen. Geben Sie

folgendes Programm ein:

```
10 INPUT"GEBEN SIE EINE ZAHL ZWISCHEN 1 UND 45 AN";A
20 ON A GOSUB 100,150,180,220
30 GOTO 10
100 PRINT "EINS"
110 RETURN
150 PRINT "ZWEI"
160 RETURN
180 PRINT "DREI"
190 RETURN
220 PRINT "VIER"
230 RETURN
```

Das Programm ^{wird} gibt zunächst die Meldung GEBEN SIE EINE ZAHL ZWISCHEN 1 UND 5 EIN ausgeben und dann die Eingabe einer Zahl erwarten. Geben Sie eine Zahl ein, und drücken Sie die RETURN-Taste. Die eingegebene Zahl wird der Variablen A zugeordnet. Wenn (in Zeile 20) A den Wert 1 erhält, wird das Unterprogramm, das mit Zeile 100 beginnt, ausgeführt. Dieses Unterprogramm gibt auf dem Bildschirm die Meldung EINS aus. Aufgrund des RETURN-Befehls springt das Unterprogramm wieder in das Hauptprogramm zurück; d. h. Zeile 30 wird ausgeführt. Durch den GOTO-Befehl in Zeile 30 wird das Programm wieder von Anfang an ausgeführt. Tippen Sie irgendeine andere Zahl ein. Sie werden bemerken, daß der Rechner immer dann die Zahl in Worten wiederholt, wenn Sie eine Zahl zwischen 1 und 4 eingegeben haben. Geben Sie dagegen eine 0 oder eine Zahl größer als 4 (z. B. 5) ein, wird der ON-Befehl ignoriert.

Die ON-Anweisung kann mit IF..THEN-Anweisungen simuliert werden. Nachfolgend ist obiges Programmbeispiel mit einer Folge von IF..THEN-Anweisungen programmiert worden:

```
10 INPUT "GEBEN SIE EINE ZAHL ZWISCHEN 1 UND 5 EIN";A
20 IF A=1 THEN GOSUB 100
30 IF A=2 THEN GOSUB 150
40 IF A=3 THEN GOSUB 180
50 IF A=4 THEN GOSUB 220
60 GOTO 10
100 PRINT "EINS"
```

```
110 RETURN
150 PRINT "ZWEI"
160 RETURN
180 PRINT"DREI"
190 RETURN
220 PRINT"VIER"
230 RETURN
```

Wie Sie erkennen können, ist das Programm, das die ON-Anweisung verwendet, sehr viel eleganter.

Mathematische Funktionen

Eine Anwendung des Computers kann die Durchführung mathematischer Berechnungen sein. Dieses Buch soll aber kein Mathematikbuch ersetzen. Wir wollen hier nur etwas über den Commodore 64 erfahren und u. a. lernen, wie man in BASIC programmiert. Deshalb soll die Mathematik nur ganz kurz besprochen werden.

INT steht für INTeGer (ganze Zahl). Mit dem Befehl INT kann eine Dezimalzahl in eine ganze Zahl umgewandelt werden. Beachten Sie dabei aber, daß die Zahl nicht gerundet wird. Die Nachkommastellen werden lediglich abgeschnitten.

```
10 A=10.258
20 PRINT INT(A)
```

Bei diesem Programmbeispiel wird also die Zahl 10 ausgegeben. Hätten wir der Variablen A den Wert 10,568 zugewiesen, so würde ebenfalls 10 - und nicht 11 - ausgedruckt werden.

ABS bedeutet ABSolutwert. Der Absolutwert einer Zahl ist die Zahl ohne Vorzeichen.

```
10 A=-1
20 PRINT ABS(A)
```

SGN ist eine Abkürzung für SiGN (= Vorzeichen). Mit der SGN-Anweisung kann

überprüft werden, welches Vorzeichen eine Zahl besitzt; d. h. ob die Zahl positiv oder negativ ist.

```
10 A=-20.258
20 PRINT SGN(A)
```

oder

```
10 A=-20.258
20 IF SGN(A)=1 THEN PRINT "POSITIVE ZAHL"
30 IF SGN(A)=-1 THEN PRINT "NEGATIVE ZAHL"
```

Wenn Sie den Wert für A ändern, werden Sie feststellen können, wie der SGN-Befehl arbeitet. (Nebenbei bemerkt: $\text{SGN}(0)=0$)

SQR berechnet die Quadratwurzel einer Zahl (Square Root).

```
10 PRINT SQR(2)
```

SIN, COS und TAN berechnen den SINus, COSinus und TAngens eines Winkels. Beachten Sie dabei bitte, daß der Winkel nicht im Gradmaß, sondern im Bogenmaß angegeben werden muß.

ATN berechnet den Arkus TaNgens einer Zahl. Der Winkel wird im Bogenmaß ausgegeben.

```
10 PRINT SIN(2)
20 PRINT COS(2)
30 PRINT TAN(2)
40 PRINT ATN(2)
```

Mit DEF kann eine Funktion definiert werden. Unter einer Funktion versteht man die Bildung eines arithmetischen Ausdrucks. Schauen wir uns folgendes Beispiel an:

```
10 DEF FN B(D) = X+Y+D
20 X=5:Y=6
30 FOR D=1 TO 10
40 PRINT FN B(D)
50 NEXT D
```

In Zeile 10 wird eine Funktion mit dem Namen B definiert. Diese Funktion besitzt den Parameter D und berechnet die Summe von X, Y und D. In Zeile 20 wird X der Wert 5 und Y der Wert 6 zugewiesen. In der Schleife (30-50) wird die Summe berechnet, wobei D nacheinander die Werte von 1 bis 10 durchläuft.

Textverarbeitung

Der Computer kann nicht nur Zahlen, sondern auch Texte verarbeiten. Wir haben bereits kennengelernt, wie zwei Texte miteinander verkettet werden können.

Die Textverarbeitung bietet in BASIC sehr viele Möglichkeiten. LEN ist eine Abkürzung für LENgth (= Länge) und bestimmt die Anzahl der Zeichen in einem Text. VAL bedeutet VALue (= Wert) und gibt den Zahlenwert eines Zeichens an. STR\$ wandelt jede Zahl in ein Zeichen um. ASC steht für ASCII. ASCII (American Standard Code for Information Interchange) ist die Abkürzung für einen vielfach benutzten Code. Mit ASC kann also der ASC-Code eines Zeichens bestimmt werden. CHR\$ ist die Umkehrung des Befehls ASC.

LEFT\$, MID\$ und RIGHT\$ sind Befehle, mit denen aus einer Zeichenkette eine bestimmte Teilkette herausgefiltert werden kann.

LEN, VAL und STR\$

Eine Zeichenkette kann maximal 255 Zeichen lang sein. Es kommt in einem Programm häufig vor, daß Sie die genaue Anzahl der Zeichen einer Zeichenkette wissen müssen. Mit dem Befehl LEN können Sie die aktuelle Länge einer Zeichenkette erfahren. Sie können diesen Befehl im Direkt-Modus oder auch in einem Programm verwenden.

```
A$= "ABCD1234"
```

```
PRINT LEN(A$)
```

oder

```
10 A$="ABCD1234"
```

```
20 X=LEN(A$)
```

In beiden Fällen wird LEN von A\$ 8 betragen. Der Befehl LEN wird Ihnen in diesem Buch noch öfters begegnen.

Wie Sie wissen, kann eine Zeichenkette aus einer beliebigen Kombination von Zeichen bestehen. Es gibt alphabetische Zeichen (die Buchstaben A, B, C usw.), numerische Zeichen (die Ziffern 1,2,3 usw.) und Sonderzeichen (*, \$, % usw.). Beachten Sie bitte, daß "1234" eine Zeichenkette ist und daß 1234 eine Zahl ist. Sie können Zeichenketten und Zahlen nicht miteinander verbinden; Sie können aber eine Zeichenkette in eine Zahl umwandeln. Fragen Sie in diesem Fall den Rechner ganz einfach nach dem Wert (VALue) der Zeichenkette.

```
10 A$= "1234"
20 B=1234
30 A=VAL(A$)
40 PRINT A+B
```

Mit dem Befehl VAL können wir nun eine Schwierigkeit meistern, die wir damals im Zusammenhang mit dem Befehl GET hatten, als wir Zahlen einlesen wollten. Tippen Sie dieses Programm ein:

```
10 GET C$:IF C$= "" THEN GOTO 10
20 C=VAL(C$)
30 PRINT C+100
```

Der GET-Befehl wird dazu verwendet, eine Zeichenkette mit dem Namen C\$ einzulesen. Da wir ursprünglich aber eine Zahl einlesen wollten, wandeln wir einfach mit dem Befehl VAL die Zeichenkette in eine Zahl um. Das obige Programm arbeitet einwandfrei, wenn Sie eine Zahl zwischen 0 und 9 eingeben. Was passiert aber, wenn Sie eine Zahl zwischen 0 und 99 eingeben wollen? Hier die Lösung dieses Problems:

```
10 PRINT "GEBEN SIE EINE ZAHL ZWISCHEN 0 UND 99 EIN"
20 GET C$:IF C$= "" THEN GOTO 20
30 PRINT C$;
40 GET D$:IF D$= "" THEN GOTO 40
50 PRINT D$
60 E$=C$+D$
```

```
70 E=VAL(E$)
80 PRINT E+100
```

Sie können aber nicht nur eine Zeichenkette in eine Zahl, sondern Sie können auch eine Zahl in eine Zeichenkette umwandeln. Hierzu müssen Sie den Befehl STR\$ benutzen.

```
10 A=1234
30 A$=STR$(A)
```

ASC und CHR\$

Wie bereits erwähnt, steht ASC für ASCII. ASCII ist der Name eines Codes, der zur Verschlüsselung von Zeichen dient. Nach ASC wird in Klammern das Zeichen angegeben, dessen ASCII-Code ermittelt werden soll.

```
PRINT ASC("5")
```

Auf dem Schirm werden Sie in diesem Fall nicht die 5, sondern die 53 sehen können. Warum?

Die Zahl 53 ist der Code für das Zeichen '5'.

Commodore Computer verwenden aber nicht den wirklichen ASCII-Code. Das liegt ganz einfach daran, daß bei dem ASCII-Code nur 128 Zeichen verschlüsselt werden können. Der Commodore 64 benötigt aber einen Code, der erheblich mehr als 128 Zeichen darstellen kann. Denken Sie beispielsweise an die vielen Grafiksymbole, den Cursor und die Editing-Steuerung. Deshalb müssen Sie vorsichtig sein und immer daran denken, daß die Codierung bei Ihrem Commodore 64 eine Variante des wirklichen ASCII-Codes ist.

Wenn Sie den ASCII-Code einer Zeichenkette bestimmen wollen, die länger als ein Zeichen ist, erhalten Sie nur den ASCII-Code des ersten Zeichens.

```
PRINT ASC("5000 Koeln")
```

In diesem Fall wird ebenfalls die Zahl 53 ausgegeben, da 5 das erste Zeichen in der Zeichenkette ist.

Vergessen Sie nicht, daß dem Null-String kein ASCII-Code zugeordnet wird. Sie bekommen deshalb die Fehlermeldung ILLEGAL QUANTITY ERROR, wenn Sie ASC("") schreiben würden.

CHR\$ ist die Umkehrfunktion von ASC.

```
PRINT CHR$(53)
```

Aufgrund dieser Anweisung wird wieder das Zeichen 5 ausgegeben. Warum so umständlich? Man könnte doch einfach PRINT 5 schreiben! Nun, es gibt einige Zeichen, die nicht ausgegeben werden können. Versuchen Sie beispielsweise, ein Anführungszeichen auszugeben. Sie würden es sicherlich mit folgendem Programm versuchen:

```
10 A$= ""  
20 PRINT A$
```

Sie werden sehen, daß dieses Programm nicht arbeitet. Versuchen Sie jetzt folgendes:

```
10 A$=CHR$(34)  
20 PRINT A$
```

Nun wird das gewünschte Anführungszeichen ausgegeben, da 34 der ASCII-Code für das Anführungszeichen ist. Sie können ebenfalls einen Carriage Return (= Wagenrücklauf) nicht ausdrucken. Es geht aber, wenn Sie

```
PRINT CHR$(13)
```

eingeben.

Eine weitere Möglichkeit ist, Steuersignale an den Drucker zu schicken oder verschiedene Betriebsweisen des Rechners einzuschalten.

Es folgt eine Liste, aus der Sie entnehmen können, was Sie mit dem Befehl PRINT CHR\$() alles machen können.

PRINT CHR\$()	BEDEUTUNG
8	Blockiert die SHIFT- und die COMMODORE-Tasten
9	Entriegelt die SHIFT- und die COMMODORE-Tasten, nachdem sie durch CHR\$(8) blockiert worden sind.
13	Gibt einen 'Wagenrücklauf' aus. Der Cursor wird an den Anfang der nächsten Zeile gesetzt.
14	Schaltet vom Kleinschrift- / Großschrift-Modus in den Großschrift-/Grafikmodus um.
34	Gibt ein Anführungszeichen aus.
142	Schaltet vom Großschrift-/Grafikmodus in den Großschrift-/Kleinschrift-Modus um.

LEFT\$, MID\$, RIGHT\$

Der beste Weg, diese drei Kommandos zu verstehen ist, sie einfach auszuprobieren. Alle drei Befehle gestatten es, aus einer Zeichenkette eine Teilkette zu extrahieren. Mit LEFT\$ bekommen Sie die linken Zeichen einer Zeichenkette, mit RIGHT\$ die rechten. Mit MID\$ können Sie eine Teilkette aus der Mitte einer Zeichenkette herausfiltern. Beachten Sie bei nachstehendem Programm die Verwendung des Befehl LEN.

```
10 A$ ="1234567890"  
20 L=LEN(A$)  
30 FOR I=1 TO L  
40 PRINT LEFT$(A$,I)  
50 NEXT I
```

Wenn Sie dieses Programm laufen lassen, werden Sie folgenden Ausdruck erhalten:

```
1
12
123
1234
12345
123456
1234567
12345678
123456789
1234567890
```

Welchen Grund gibt es dafür? Mit `LEFT$` werden die linken Zeichen einer Zeichenkette ausgegeben. Nach `LEFT$` folgt in Klammern die Zeichenkette, von der die Teilkette (beginnend mit dem ersten linken Zeichen) gebildet werden soll. Danach wird die Länge der Teilkette angegeben. Die Länge der Teilkette wird in dem obigen Beispiel durch die Variable `I` bezeichnet. Diese Variable ist Bestandteil der Schleife und erhält somit nacheinander die Werte 1 bis 10. Wir wollen jetzt das gleiche Programm laufen lassen; in Zeile 40 ersetzen wir nur `LEFT$` durch `RIGHT$`. Folgendes wird auf dem Bildschirm ausgegeben:

```
0
90
890
7890
67890
567890
4567890
34567890
234567890
1234567890
```

Bei `RIGHT$` wird also nicht eine Zeichenkette von der 'linken' Seite, sondern von der 'rechten' Seite der Zeichenkette herausgefiltert.

Der Befehl `MID$` ist ein wenig komplizierter. Nach `MID$` folgt in Klammern die Zeichenkette und zwei Zahlen.

```
10 A$="1234567890"  
20 PRINT MID$(A$,5,2)
```

Die erste Zahl bezeichnet die Position in der Zeichenkette A\$, mit der die Bildung der Teilkette beginnt. Die zweite Zahl gibt an, wie lang die Teilkette sein soll. Bei obigem Programmbeispiel wird also 56 auf dem Bildschirm ausgegeben.

Die Wirkungsweise von MID\$ kann durch nachstehendes Programm noch besser verstanden werden.

```
10 A$= "123456789"  
20 L=LEN(A$)  
30 FOR I=1 TO L  
40 FOR K=1 TO L  
50 PRINT MID$(A$,I,K)  
60 NEXT K  
70 NEXT I
```

Geben Sie dieses Programm in Ihren Computer ein und starten Sie es. Zweckmäßigerweise sollten Sie die CTRL-Taste drücken, damit die Ausgabe auf dem Bildschirm verlangsamt wird.

Sie sollten nun einmal die Befehle LEFT\$, MID\$, und RIGHT\$ in einem kleinen Programmbeispiel in Verbindung mit der eingebauten Uhr Ihres Commodore ausprobieren. Der Commodore 64 wird so als Uhr arbeiten. In die linke obere Ecke des Bildschirm soll (wie bei einer Digitaluhr) die Uhrzeit fortlaufend angegeben werden. Tippen Sie folgendes Programm ein und starten Sie es mit RUN.

```
10 INPUT "STUNDE";H$  
20 INPUT "MINUTE";M$  
30 TI$=H$+M$+"00"  
40 PRINT "(CLR-Taste drücken)"  
50 PRINT "(HOME-Taste drücken)";  
60 PRINT "UHRZEIT";  
70 PRINT LEFT$(TI$,2);":";MID$(TI$,2,2);":";  
   RIGHT$(TI$,4)  
80 GOTO 50
```

Sie müssen zunächst die Uhr auf einen Startwert setzen, d. h. die Uhr 'einstellen'. Wenn es z. B. 1:03 Uhr ist, geben Sie 01 für die Stunden, und 03 für die Minuten ein. Das Programm geht davon aus, daß mit 0 Sekunden gestartet wird. Wie wir bereits in einem der vorangegangenen Abschnitte erwähnt haben, beginnt die Uhr nun zu arbeiten. Wir hatten beschrieben, wie in der Variablen TI\$ die Uhrzeit gespeichert ist. Wir müssen jetzt nur noch aus dieser Variablen die Angaben für die Stunden und Minuten so extrahieren, daß wir unsere gewohnte Darstellungsart der Uhrzeit bekommen. Das geschieht mit Zeile 70 des Programms. Mit dem nachfolgenden Befehl GOTO 50 wird der Bildschirm wieder gelöscht, und die Uhrzeit wird erneut ausgegeben. Dieses passiert so schnell, daß wir nicht in der Lage sind zu erkennen, wie der Bildschirm gelöscht und die Uhrzeit erneut geschrieben wird. So ist es leicht verständlich, daß immer die aktuelle Uhrzeit angegeben wird.

Sie können dieses Programm auch als Unterprogramm einsetzen, wenn Sie den GOTO-Befehl in Zeile 80 durch RETURN ersetzen.

8 Textverarbeitung

'Das ist ja ein ganz schönes Gerät, aber wozu kann man es wirklich gebrauchen?'

Das ist eine Frage, die sehr oft dem Besitzer eines Personal Computers gestellt wird. Diese Frage ist meistens auch sehr schwer zu beantworten. In diesem Kapitel wird ein sehr wichtiges Anwendungsgebiet für den Computer besprochen: die Textverarbeitung.

In den letzten Jahren wurden die Schreibmaschinen in den Büros zunehmend durch Textautomaten ersetzt. Diese Textautomaten bestehen teilweise aus Computern, die in vielen Fällen nicht größer als der Commodore 64 sind. Der Unterschied dieser Computer zu Ihrem Commodore 64 ist der, daß diese nur zur Textverarbeitung verwendet werden.

Textverarbeitung kann mit kleinen oder großen Computern durchgeführt werden. Eines der ersten Textverarbeitungsprogramme, die auf großen Computern durchgeführt wurden, war ATMS (Automated Text Management System). Dieses Programm lief in den 70er Jahren auf großen IBM Computern.

Was man mit einem Textautomaten anfangen kann

Der Name 'Textautomat' gibt schon einen wichtigen Hinweis auf seine eigentliche Funktion. Ein Textautomat enthält ein Programm, mit dem Texte ver- und bearbeitet werden können. Diese Verarbeitung von Texten geschieht durch Funktionen.

Texteingabe. Die Texteingabe dient dazu, einen Text über die Tastatur eingeben und in den Arbeitsspeicher des Rechners abspeichern zu können.

Text-Editor. Ein Text-Editor ist eine Funktion, die die Handhabung eines Textes ermöglicht. Dazu gehört beispielsweise, daß einzelne Wörter oder Sätze gelöscht oder eingefügt werden können. Eine weitere Editor-Funktion ist das Ändern von Texten.

Speichern von Texten. Der Text, den Sie über die Tastatur in den Arbeitsspeicher des Rechners eingegeben haben, kann auf einem peripheren Speicher (z. B. Datassette oder Diskette) gespeichert werden. Sie benötigen dann natürlich eine weitere Funktion, mit der Sie den gespeicherten Text von dem Externspeicher wieder in den Arbeitsspeicher des Rechners holen können.

Druck von Texten. Letztlich gehört zu einem Textverarbeitungsprogramm auch ein sogenannter 'Report-Generator'. Mit diesem Report-Generator können Sie den Text ausdrucken. Dabei ist es oft möglich, den Ausdruck nach den Wünschen des Anwenders zu gestalten.

Das ist im großen und ganzen das, was Ihnen ein Textverarbeitungsprogramm bieten kann. Wie Sie sehen, wird Ihr Computer dadurch zu einer sehr leistungsfähigen elektronischen Schreibmaschine.

Wer kann von einem Textautomaten profitieren? Eigentlich jeder, der mit Texten umgehen muß: Studenten, Geschäftsleute, Autoren usw.

Hardware für Textverarbeitung: Drucker und Externspeicher

Wenn Sie mit Ihrem Computer Textverarbeitung machen wollen, benötigen Sie natürlich einen Drucker. Außerdem ist ein Externspeicher, entweder ein Diskettenlaufwerk oder die Datassette, notwendig.

Drucker. Ein Drucker ist eine nützliche Investition und zwar auch dann, wenn Sie keine Textverarbeitung machen wollen. Sie können den Drucker nämlich zum Ausdruck eines BASIC-Programms verwenden.

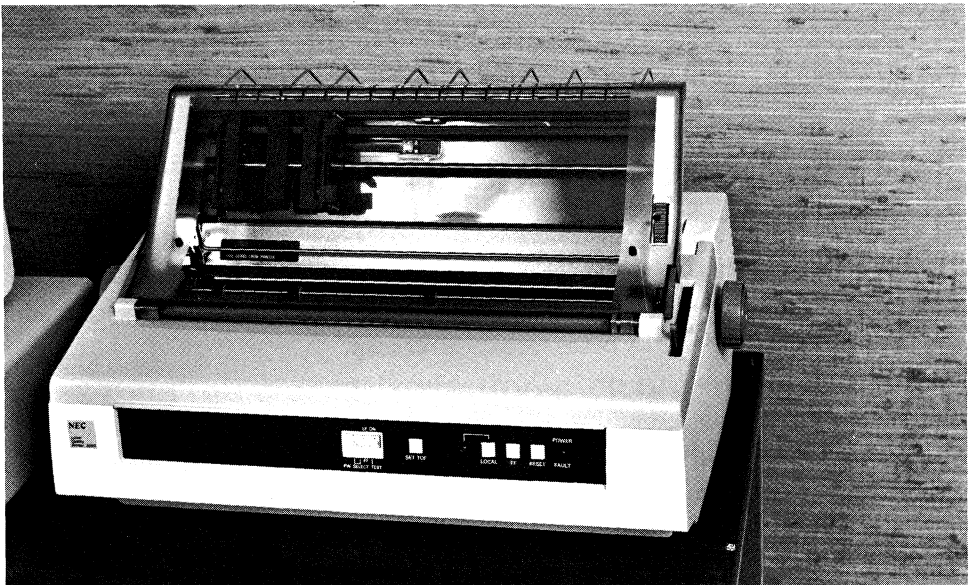
Zwei verschiedene Arten von Druckern sind üblich: Matrix-Drucker und Typenrad-Drucker. Bei Matrix-Druckern wird jeder Buchstabe durch eine An-

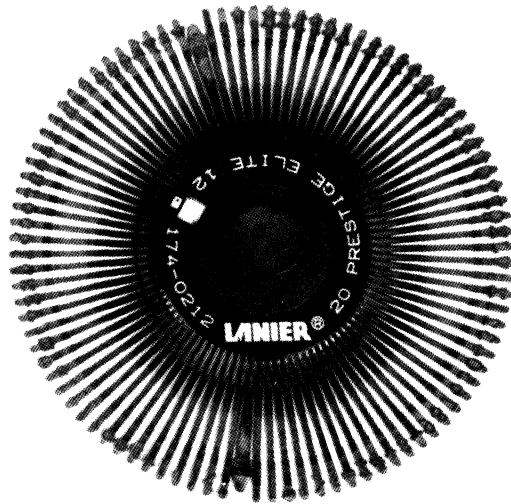
einanderreihung kleiner Punkte gebildet. Diese Punkte sind in einer Matrix angeordnet. Es werden nur die Punkte der Matrix gedruckt, die notwendig sind, um das betreffende Zeichen darzustellen. Deshalb sind Texte, die mit Matrix-Druckern geschrieben werden, manchmal schwer lesbar.

1350 IF I<N THEN 13	10 goto9000:rem (c)
1360 IF H=1 THEN 13	20 getc\$:ifc\$goto20
1370 IF LEFT\$(NO\$,1	25 getc\$:ifc\$thenpr
1380 Q\$=INKEY\$:IF C	27 print" <c-l>";:g
1390 REM TO DRAW RE	30 get#1,c\$:x=0:ifc

Die Geschwindigkeit der Matrix-Drucker beträgt 80-160 Zeichen pro Sekunde. Die Drucker sind relativ preiswert; man kann einen solchen Drucker schon für weniger als 1000,- DM bekommen. Ein Vorteil der Matrix-Drucker ist, daß mit dem Drucker verschiedene Schriftarten dargestellt werden können. Diese Schriftarten können Sie mit speziellen BASIC-Befehlen in Ihrem Programm auswählen. Matrix-Drucker bieten außerdem die Möglichkeit, den Drucker als Plotter zu verwenden. Plotter sind Zeichengeräte, mit denen grafische Darstellungen ausgegeben werden können.

Wie schon erwähnt ist es ein großer Nachteil der Matrix-Drucker, daß





der Ausdruck nicht in Korrespondenzqualität erfolgt. Falls Sie eine hochwertige Schriftqualität benötigen, müssen Sie einen Typenrad-Drucker wählen. Das 'Herz' eines Typenraddruckers ist das Typenrad, auf dem die einzelnen Zeichen angeordnet sind.

Die Druckgeschwindigkeit eines Typenrad-Druckers liegt bei 15-55 Zeichen pro Sekunde. Leider sind diese Typenrad-Drucker komplizierter aufgebaut und deshalb teurer.

Eine weitere Möglichkeit wird in zunehmendem Maße von Besitzern eines Personal Computers wahrgenommen. Sie können nämlich Ihre elektronische Schreibmaschine auch als Drucker verwenden. Dazu benötigen Sie ein Interface, das den Anschluß Ihrer Schreibmaschine an den Computer ermöglicht. Der Nachteil ist jedoch, daß die Druckgeschwindigkeit in den meisten Fällen relativ gering ist. Außerdem ist der Zeichensatz einer Schreibmaschine sehr begrenzt. Sonderzeichen, die in der Computersprache verwendet werden (z. B. \$, < , > , # usw.), sind meistens nicht auf einer Schreibmaschinentastatur vorhanden.

Interfaces. Die meisten Drucker können nicht direkt an Ihren Commodore 64 angeschlossen werden; Sie benötigen deshalb ein Interface. Die Daten

können auf zwei Arten vom Computer zum Drucker übertragen werden: parallel oder seriell. Man bezeichnet die Verbindung des Druckers mit dem Computer auch als Schnittstelle. Eine serielle Standardschnittstelle ist die RS-232 Schnittstelle. Eine weitere Schnittstelle ist die IEEE-488 Schnittstelle, die normalerweise bei den Commodore-Computern Standard ist.

Die Daten, die der Computer sendet, werden in einem Interface so aufbereitet, daß sie von dem Drucker 'verstanden' werden können. Das wird durch eine elektronische Schaltung ermöglicht, die sich auf einer Karte befindet. Diese Karte ist meistens direkt in Ihrem Drucker enthalten.

Textverarbeitung für Fortgeschrittene

Wie Sie bereits wissen, können Sie mit einem Word Processor (ein anderer Begriff für 'Textverarbeitung') Texte über die Tastatur eingeben, das Geschriebene ändern, den Text auf der Datensette oder Diskette speichern und schließlich auch den Text ausdrucken. Die verschiedenen Textverarbeitungsprogramme unterscheiden sich sehr in ihrer Leistungsfähigkeit.

Ein gutes Textverarbeitungsprogramm erlaubt eine sogenannte 'Bausteinverarbeitung'. 'Bausteinverarbeitung' bedeutet, daß selbständige Textblöcke innerhalb eines Textes an beliebiger Stelle eingefügt werden können. Dadurch ist es möglich, Texte aus verschiedenen Textdateien miteinander zu mischen (= merge). Durch die Bausteinverarbeitung können Sie sich also einen Satz bestimmter Standardtextblöcke anlegen und diese Standardtextblöcke zu verschiedenen Texten zusammensetzen.

Bei sehr langen Texten ist es sinnvoll, ein Inhaltsverzeichnis über den Text führen zu können, damit bestimmte Textstellen leichter aufgefunden werden können. Ein solches Inhaltsverzeichnis wird auch als Index-File bezeichnet.

Haben Sie sich bei einem Wort verschrieben? Ein guter Word Processor erlaubt es Ihnen, diese Stellen innerhalb des Textes aufzusuchen und das falsche Wort durch das richtige zu ersetzen.

Bei Ihrem Commodore 64 können leider nur 40 Zeichen pro Zeile dargestellt werden. Durch eine Erweiterung der Hardware ist es aber möglich, 80 Zeichen pro Zeile darzustellen. Die Zeichen sind dann sehr klein und somit schwer lesbar. Aus diesem Grund ist eine variable Zeilenlänge vorzuziehen. Sie können die Anzahl der Zeichen pro Zeile in Ihrem Textverarbei-

tungsprogramm frei wählen. Das Textverarbeitungsprogramm erlaubt es Ihnen, eine Zeile nach rechts und nach links zu bewegen, so daß nacheinander die komplette Zeile überblickt werden kann.

Vier Commodore 64 Textverarbeitungsprogramme

Zur Zeit gibt es für Ihren Commodore 64 sehr viele Textverarbeitungsprogramme, die kommerziell erhältlich sind. In diesem Buch sollen vier Programme als Beispiel besprochen werden. Jedes Programm ist ein sehr leistungsfähiger Word Processor, der nicht in BASIC, sondern in Maschinsprache geschrieben wurde.

Quick Brown Fox

QBF (= Quick Brown Fox = schneller brauner Fuchs) kann als Miniwordprocessor beschrieben werden. QBF wird auf einer Kassette geliefert, die in Verbindung mit der Datassette das Laden des Programms ermöglicht. Wenn der Computer eingeschaltet wird, erscheint folgende Meldung auf dem Bildschirm:

```
Quick Brown Fox  
(C) 1982
```

Nach dem Drücken der RETURN-Taste können Sie sich das Hauptmenü ansehen.

```
. . . Quick Brown Fox  
Type  
View  
Print  
B.View  
G.Edit  
L.Edit  
Move  
Delete  
Zap Memory
```

Send
Receive
Clerk.?

Nun können Sie zwischen den angebotenen Menüs auswählen, indem Sie einfach den ersten Buchstaben des Menünamens eintippen. Es ist also nicht notwendig, den ganzen Namen einzugeben. Das Menü 'Type' bezeichnet die Möglichkeit, über die Tastatur einen Text in den Arbeitsspeicher eingeben zu können. Insgesamt stehen 37 886 Bytes zur Verfügung. Das sind in etwa 25 (in doppeltem Zeilenabstand geschriebene DIN A4) Schreibmaschinenseiten. QBF besitzt einen file-orientierten Editor. Wenn Sie mit Type einen Text eingeben, können Sie nur die Zeile editieren, die Sie gerade eintippen. Soll eine vorher eingegebene Zeile editiert werden, muß das auf einem anderen Weg geschehen.

Mit L.Edit kann eine Zeile des Textes editiert werden. Die Zeile, die Sie gerade geschrieben haben, erscheint in der unteren Hälfte des Bildschirms. Wenn Sie die Taste CRSR DOWN drücken, erhalten Sie die nächste Zeile. Wird die CRSR UP-Taste gedrückt, wird eine neue Zeile (die vorangegangene) ausgegeben. Es werden beispielsweise folgende Zeilen ausgegeben:

ueber den traegen Hund.
Der schnelle braune Fuchs springt

Auf diese Art und Weise können Sie mit QBF Texte editieren.

Neben dem Befehl L.Edit (= line edit), der das Editieren einer Zeile ermöglicht, gibt es den Befehl G.Edit. Mit G.Edit können Sie einen Abschnitt eines Textes editieren.

'Delete' und 'Move' sind zwei weitere Editierfunktionen. Mit 'Delete' können Texte gelöscht werden, mit 'Move' können Zeilen von einer Stelle zu einer anderen innerhalb des Textes bewegt werden.

Der Befehl Zap Memory löscht den im Arbeitsspeicher enthaltenen Text. Zap Memory muß benutzt werden, bevor z. B. ein Text von der Diskette oder der Datensette geladen werden soll.

Mit den Befehlen Send und Receive können zwei Computer über ein Telefonmodem gekoppelt werden. Diese Befehle ermöglichen den Austausch eines Textes über die Telefonleitung.

Der Befehl Clerk speichert den geschriebenen Text auf einem Externspeicher ab oder lädt den Text von diesem.

Mit View und Print können Sie sich den Text ansehen, den Sie eingegeben haben. Mit View wird der Text auf dem Bildschirm, mit Print auf dem Drucker ausgegeben. Damit können Sie z. B. einen Schreibrand oder einen Tabulator setzen, Wörter als Überschrift zentrieren und verschiedene Schriftarten wählen.

Das Handbuch von QBF ist sehr benutzerfreundlich geschrieben. Gerade deshalb wird der Stoff nur in begrenztem Umfang dargeboten, und einige Kommandos sind nicht präzise erklärt. Da QBF kein allzu kompliziertes Programm ist, wird es Ihnen nicht schwer fallen, mit diesem Programm umzugehen.

Wie brauchbar ist das Programm Quick Brown Fox? Nun, das hängt davon ab, wie intensiv Sie dieses Programm benutzen wollen. Denken Sie daran, daß Sie kein Diskettenlaufwerk benötigen. Das Programm wird auf einer Kassette geliefert, die mit der Datasette geladen werden kann. Das Programm ist aus diesem Grunde sehr gut dazu geeignet, einen Einblick in die Textverarbeitung zu bekommen. Obwohl dieses Programm von einer vollkommenen Perfektion weit entfernt ist, hält es das, was es verspricht (Lieferant: Quick Brown Fox, Inc., 548 Broadway, Suite 4F, New York, NY 10012).

WordPro 3 Plus/64

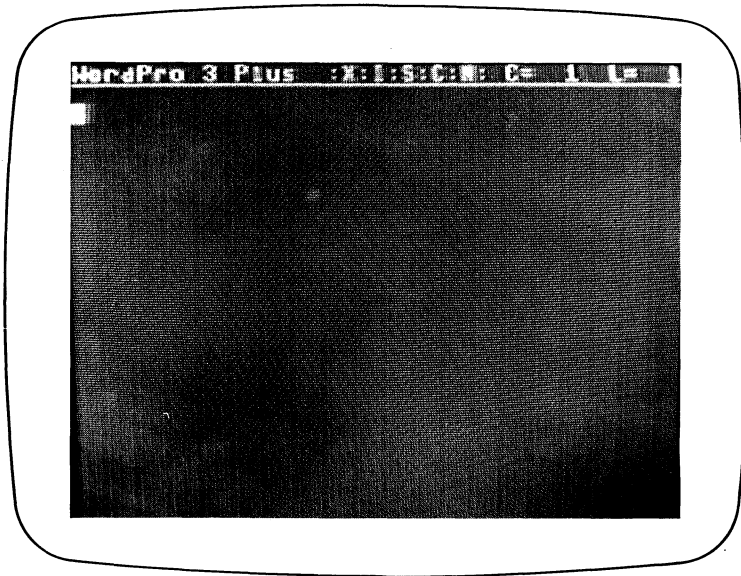
WordPro ist das Textverarbeitungsprogramm, welches standardmäßig von Commodore geliefert wird. Es ist von dem Kanadier Steve Punter programmiert worden. Das Programm ist über mehrere Jahre in verschiedenen Versionen entwickelt worden. Was bedeutet der Name WordPro 3 Plus/64. WordPro '3' bedeutet, daß diese die dritte Version des Programms ist. Es existiert auch eine vierte Version für die CBM 8032 Professional Computer. 'Plus' bedeutet, daß das Programm erheblich verbessert worden ist, nachdem es für den Commodore PET eingeführt wurde.

Das Programm erfordert ein Diskettenlaufwerk und einen Drucker (Sie können natürlich zwischen verschiedenen Druckern wählen). WordPro ist leistungsfähig, schnell und ziemlich einfach zu benutzen. Das WordPro Handbuch ist sehr ausführlich und gibt eine gute Einführung in die Benutzung des Programms.

Nachdem WordPro von der Diskette geladen worden ist, können Sie verschiedene Parameter setzen. WordPro stellt beispielsweise die Frage, wieviele Textzeilen in dem Arbeitsspeicher abgespeichert werden sollen. Es sind maximal 329 Zeilen erlaubt. Das entspricht 13160 Bytes oder etwa 9 mit doppeltem Zeilenabstand geschriebene DIN A4 Seiten.

Insgesamt erscheinen 23 Zeilen auf dem Bildschirm. Vor der ersten Zeile ist eine Statuszeile sichtbar, die wie folgt aussieht:

WordPro 3 Plus :X:I:S:C:N C=1 L=1



Diese Zeile gibt Ihnen einige Informationen. C steht für 'Control'. Wenn Sie die Control-Tasten drücken, können Sie damit die WordPro Funktionen anwählen. Es erscheint dann die Farbe des C (in :X:I:S:C:N) im Reverse-Modus.

Auch die anderen Buchstaben haben eine Bedeutung. I steht für 'insert' (= einfügen). Wenn Sie in diesem Modus arbeiten, wird alles, was Sie gerade eingeben, in den bereits auf dem Bildschirm vorhandenen Text eingefügt.

X bedeutet 'extra text', d. h. Sie befinden sich in dem Extra Text Speicher - und nicht in dem Hauptspeicher. Der gesamte Speicherbereich ist nämlich von WordPro in zwei Speicher aufgeteilt worden. In dem Hauptbereich wird der Text abgespeichert. Sie können in einem weiteren (Extra Text Bereich) Speicher Text abspeichern, der ggf. mit dem Text des Hauptspeichers vermischt werden kann. S bedeutet, daß alles, was Sie eingeben, in Großbuchstaben geschrieben wird. N ist eine Abkürzung für 'numerischer Modus'.

C= und L= geben an, wo der Cursor gerade in dem verfügbaren Speicher steht. C bedeutet 'Column' (= Spalte), L bedeutet 'Zeilennummer' (= line number).

Der Bildschirm, den WordPro zeigt, kann als eine Art Fenster verstanden werden. Durch dieses Fenster sehen Sie einen Ausschnitt aus dem Textspeicher. Was immer Sie mit dem Text machen, es bezieht sich auf den Text in dem Speicher. Wenn Sie eine zusätzliche Zeile zu den 23 möglichen auf dem Bildschirm eintippen, wird eine Zeile nach oben aus dem Bildschirm 'herausgeworfen.'. Werden die Tasten CRSR UP und CRSR DOWN gedrückt, wird das Fenster damit nach oben bzw. nach unten in den Textspeicher verschoben. In WordPro gibt es insgesamt 46 verschiedene Kontrollfunktionen; zu viele, um sie hier detailliert darstellen zu können. Es können Sätze gelöscht werden (Control-D), ein Text kann nach Belieben verschoben werden (Control-R und Control-T) und es kann außerdem Text ausgedruckt werden (Control-U).

Zusätzlich zu den obigen Kontrollfunktionen gibt es 23 verschiedene Kommandos, mit denen Sie den Text formatieren bzw. gestalten können.

WordPro war der erste Word Processor für den Commodore Computer, der neben der reinen Textverarbeitung und Textbearbeitung auch einfache arithmetische Funktionen erlaubte. So können im Text vorkommende Zahlen addiert und subtrahiert werden. Kompliziertere mathematische Funktionen müssen jedoch außerhalb von WordPro in einem speziellen Programm durchgeführt werden.

Obwohl WordPro 3 Plus/64 ein guter Word Processor ist, ist er natürlich im Verhältnis zu professionellen System hinsichtlich der Möglichkeiten beschränkt. Andererseits besitzt WordPro einen großen Teil der Funktionen, die Sie auch bei anderen anspruchsvolleren Textverarbeitungsprogrammen finden werden (WordPro 3 Plus/64-Professional Software, Inc., 51 Fremont Street, Needham, MA 02194).

Easy Script

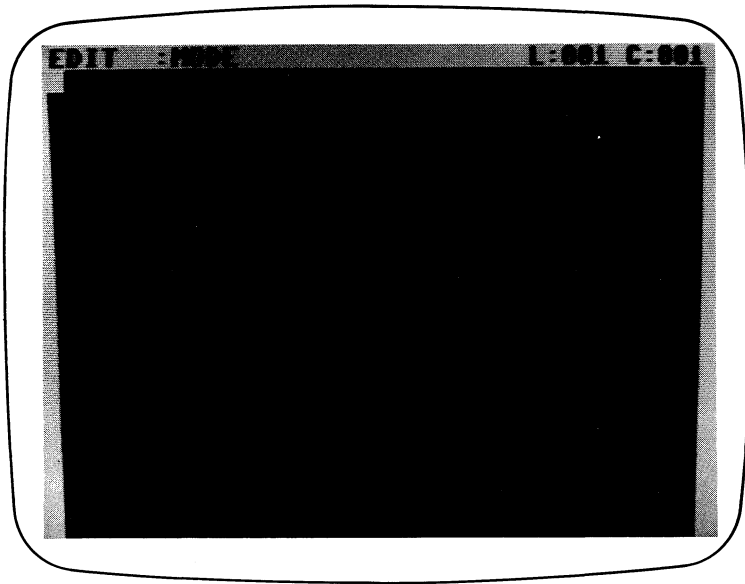
Easy Script ist der 'offizielle' Word Processor der Commodore 64 Computerfamilie. Easy Script wurde von Precision Software - einer britischen Firma - für Commodore entwickelt und ist WordPro sehr ähnlich. Die Arbeitsweise beider Programme ist dagegen sehr unterschiedlich. Wie WordPro ist Easy Script ein zeichenorientierter Word Processor, bei dem mithilfe des Bildschirms ein 'Fenster' von dem im Arbeitsspeicher abgespeicherten Text gezeigt wird. Ein Diskettenlaufwerk oder eine Datassette und ein Drucker sind notwendig. Das bedeutet also, daß Sie ein Programm auch ohne Diskettenlaufwerk benutzen können.

Erfreulicherweise ist Commodore nicht so weit gegangen, nur den Anschluß von Commodore Druckern zu gestatten. Sie können also ebenfalls Drucker der Firmen Epson, NEC, Spinwriter, Qume, Diablo u. a. verwenden. Es können Drucker angeschlossen werden, die sowohl mit einer parallelen oder mit einer seriellen Schnittstelle arbeiten.

Mit Easy Script können 764 Bildschirmzeilen (das sind 30 560 Bytes oder über 20 mit doppeltem Zeilenabstand geschriebene Seiten) bearbeitet werden.

Im Gegensatz zu anderen Word Processoren benutzt Easy Script die Funktionstasten des Commodore 64. Damit wird der Umgang mit diesem Programm für den Anwender erheblich vereinfacht. Die F1-Taste ist die am häufigsten verwendete Funktionstaste. Diese Funktionstaste arbeitet wie die Kontroll-Taste in WordPro und erlaubt das Umschalten in die verschiedenen Betriebsarten des Word Processors. Mit F4 können Sie den Text auf Diskette oder Datassette speichern bzw. den Text von diesen wieder laden.

Sie können (wie in WordPro) einen Text eingeben und gleichzeitig editieren. Eine Statuszeile, die oben im Bildschirm erscheint, gibt Ihnen die Information, in welchem Modus Sie sich befinden und wo der Cursor in dem Textspeicher steht. Dabei haben Sie die Möglichkeit eine variable Zeilenlänge anzugeben. Sie können angeben, wieviele Zeichen in einer Zeile gespeichert werden sollen. Dabei ist irgendein Wert zwischen 40 und 240 erlaubt. Auf dem Bildschirm sind aber nur 40 Zeichen sichtbar. Damit Sie sich die ganze Zeile anschauen können, ist es möglich, zusätzlich zu der Bewegung nach oben bzw. nach unten das Bildschirmfenster nach rechts und links über den Textspeicher zu verschieben. Mit speziellen Kommandos von Easy Script kann der Text für den Ausdruck besonders gestaltet werden. Die



Kommandos zur Gestaltung des Textes sind denen von WordPro sehr ähnlich. So bedeutet beispielsweise

```
lm10:rm70:cn1
```

daß der linke Rand auf 10, der rechte Rand auf 70 eingestellt wird und die Zeile in der Mitte der Seite zentriert werden soll.

Da Commodore das Textverarbeitungssystem Easy Script unterstützt, hat dieser Word Processor eine gute Chance, Standard für die Computerfamilie des Commodore 64 zu werden. So gut Easy Script auch sein mag, es hat dennoch einige Nachteile. Es ist beispielsweise nicht möglich, numerische Operationen durchzuführen (im Gegensatz zu WordPro).

(Easy Script, Commodore Business Machines, 1200 Wilson Drive, West Chester, PA 19380)

PaperClip

PaperClip (= Büroklammer) ist so leistungsfähig, daß es sehr viel Zeit in Anspruch nehmen kann, bis Sie dieses Programm vollständig beherrschen. Es gibt nahezu 150 verschiedene Kommandos. Trotzdem sind nur ein paar dieser Kommandos notwendig, wenn Sie mit der Arbeit beginnen wollen.

PaperClip ist gewissermaßen eine Zusammenfassung von WordPro und Easy Script. Obwohl es in dem Handbuch nicht extra erwähnt ist, kann man mit PaperClip Dateien einlesen, die mit WordPro oder Easy Script erstellt worden sind. Umgekehrt ist das jedoch nicht möglich!

PaperClip ermöglicht den Anschluß fast aller Drucker.

Wie in Easy Script, können Sie auch in PaperClip Zeilen eingeben, die länger als 40 Zeichen sind. Eine Zeile darf allerdings nicht länger als 126 Zeichen sein. PaperClip erlaubt neben der Textverarbeitung und -bearbeitung Operationen mit Informationen, die in Tabellenform gespeichert sind. So können Sie beispielsweise einfache arithmetische Operationen durchführen (z. B. Addition und Subtraktion) und Zahlen oder Texte sortieren.

Ein kritischer Punkt dieses Programms sollte allerdings erwähnt werden. Das Bedienungshandbuch gibt keine ausführlichen Erklärungen. Es ist eigentlich sehr schade, daß ein derart leistungsfähiges Programm nicht entsprechend dokumentiert ist. Die Herstellerfirma hat aber bereits versprochen, daß dieses Handbuch bald überarbeitet wird.

(PaperClip 64-Batteries Included, 71 McCaul Street, Toronto, Ontario, Canada M5T-2X1)

Ein paar Tips

Die Benutzung eines Word Processors wird Ihre Art und Weise zu schreiben verändern. So können Sie beispielsweise einfach und schnell Fehler korrigieren. Das ist noch längst nicht alles. Die Qualität Ihres Textes kann durch die Möglichkeiten verbessert werden, die Ihnen ein Textsystem bietet. Beispielsweise können Sie in dem Ihnen vorliegenden Text ein Wort durch ein neues, passenderes Wort ersetzen. Ohne Textsystem wäre das eine sehr aufwendige Angelegenheit. Mit einem Word Processor ist das Problem schnell gelöst.

Jeder kann somit seinen eigenen Stil im Umgang mit dem Word Processor finden. Hierzu sollen ein paar Ratschläge allgemeiner Art gegeben werden.

- Wenn möglich, machen Sie sich auf einer zweiten Diskette eine Kopie des Textverarbeitungsprogramms, bevor Sie dieses anwenden.
- Überlegen Sie sich, wie Sie den Text auf der Diskette organisieren wollen. Sie sollten beispielsweise den vorhandenen Platz in Dateien für Geschäftsbriefe, Briefe an Freunde und Bekannte usw. aufteilen.
- Gewöhnen Sie sich daran, die erste Zeile eines Textes als Kommentarzeile zu verwenden. Sie sollten in dieser Zeile den nachfolgenden Text ganz kurz beschreiben, damit Sie später immer wissen, um welchen Text es sich handelt.
- Machen Sie sich immer eine Kopie von den Textdateien. Sie vermeiden damit, daß Ihre ganze Arbeit umsonst war, falls ein Diskettenfehler oder ein Fehler der Datassette auftritt.
- Schreiben Sie sich die wichtigsten Befehle Ihres Textprogramms aus dem Handbuch heraus, und legen Sie sich diese Befehlsliste neben die Tastatur, wenn Sie mit dem Programm arbeiten. Sie vermeiden dadurch ein zeitraubendes Nachschlagen in einem großen Bedienungshandbuch.
- Wenn Sie den Text auf Fehler durchsehen, ist es einfacher, sich dazu einen Ausdruck auf Papier zu erstellen. Es ist nämlich leichter, diesen Ausdruck durchzusehen, als den Text auf dem Bildschirm überprüfen zu müssen.
- Arbeiten Sie nicht ununterbrochen, sondern gönnen Sie sich ein paar kleine Pausen, da die Bildschirmarbeit sehr ermüdend ist.
- Kommen Sie nicht in die Versuchung, Ihren Text ständig verbessern zu wollen. Dieser Gefahr ist man leicht erlegen, da ein Textsystem sehr viele Möglichkeiten dazu bietet. Versuchen Sie auf keinen Fall mehr Zeit für die Arbeit aufzuwenden, als Sie bei Benutzung einer normalen Schreibmaschine benötigen würden.

9 Farbe, Grafik, Musik und Spiele

Einige Besitzer eines Personal Computers nehmen leider oft nicht die Vorteile wahr, die ihnen der Computer hinsichtlich Farbe, grafischer Darstellungen und der Möglichkeit der Musikerzeugung bietet. Für andere Benutzer ist das der wichtigste Punkt überhaupt. Anderen Meinungen entsprechend sind diese Möglichkeiten des Computers unwichtig für 'seriöse' Anwendungen. Insbesondere wird dabei auch an die Video-Spiele gedacht.

Das ist aber nicht ganz richtig. So fällt es dem Menschen in der Regel leichter, eine grafische Darstellung zu verstehen, als eine Tabelle von Zahlen. Deshalb sind diese Möglichkeiten, die uns der Computer bietet, sehr wichtig, und nicht nur 'Spielerei'.

In diesem Kapitel werden Sie erfahren, wie Sie mit Ihrem Commodore 64 farbige, grafische Darstellungen und Musik erzeugen können. Dieses Kapitel erhebt keinen Anspruch auf Vollständigkeit; es soll nur Ihr Interesse für diese Thematik wecken. Etwas mehr erfahren Sie nämlich am Ende des Buches. In diesem Kapitel werden Sie deshalb nur einige grundlegende Dinge lernen. So werden Sie beispielsweise erfahren, wie Sie Ihren Programmen einen reizvollen optischen Rahmen geben können und wie Sie mit dem Computer einfache Lieder spielen können. Wenn Sie sich für Spielprogramme interessieren, werden Sie in diesem Kapitel auf Ihre Kosten kommen. Sie werden beispielsweise erfahren, wie man einen elektronischen Würfel oder ein Glücksrad programmieren kann. Letztlich werden Sie etwas mehr über die 'Control Ports' erfahren. Die Control Ports sind die Steckbuchsen, in die Sie Ihre Joysticks und Game Paddles stecken können.

VIC und SID

VIC steht für Video Interface Controller und SID für Sound Interface Device. Der VIC und der SID sind elektronische Bausteine (Chip) mit bestimmten Eigenschaften.

Mit dem VIC-Chip können Sie den Computer in verschiedenen Grafikbetriebsarten benutzen. Die erste Betriebsart ist der Zeichenmodus. Der Zeichenmodus wird automatisch angewählt, wenn Sie den Computer einschalten. Mit dem PRINT-Befehl können Sie irgendetwas auf dem Bildschirm ausgeben. Es ist möglich, in dem Zeichenmodus den Zeichensatz des Computers neu zu definieren. So kann beispielsweise mithilfe des VIC-Chips von einem Standardalphabet zu anderen Zeichen umgeschaltet werden. Diese anderen speziellen Zeichen können z. B. mathematische Symbole, griechische Buchstaben usw. sein. Sie haben sogar die Möglichkeit, selbst eigene Zeichen zu erzeugen.

In dem Zeichen- (character) Modus ist der Bildschirm in 1 000 einzelne Punkte (25 Zeilen a 40 Zeichen) aufgeteilt. Jeder Punkt wird durch ein Byte des Arbeitsspeichers repräsentiert. Insgesamt werden also 1 000 Bytes benötigt, um die Information eines Bildschirms darzustellen. Da Sie 16 verschiedene Farben mit Ihrem Commodore 64 anwählen können, wird zusätzlicher Speicherplatz benötigt. Jede Farbe braucht aber nur ein halbes Byte, d. h. 4 Bits. Weitere 500 Bytes werden somit benötigt, um die Farben auf dem Bildschirm zu steuern.

Ein anderer Grafik-Modus wird Hochauflösungsgrafik genannt. Diese Betriebsart bietet erheblich mehr Möglichkeiten, als die Betriebsart im Character-Modus. Wenn Sie Ihren Computer mit der Hochauflösungsgrafik betreiben, wird der Bildschirm vollständig in einzelne kleine Punkte, auch Pixels genannt, aufgeteilt. (Den PRINT-Befehl können Sie in diesem Modus allerdings nicht verwenden.) Jeder Punkt wird durch ein einzelnes Bit im Arbeitsspeicher repräsentiert. Bei der Hochauflösungsgrafik wird der Bildschirm in 320 Punkte waagerecht und in 200 Punkte senkrecht aufgeteilt. Das bedeutet, daß insgesamt 8000 Byte Arbeitsspeicher benötigt werden.

Bei der Hochauflösungsgrafik können Sie zwischen zwei Farben wählen; mit einer Farbe wird der Vordergrund, mit der anderen der Hintergrund bestimmt. Eine Variation des Hochauflösungsmodus ist, daß Sie bestimmte Bereiche (regions) auf dem Bildschirm definieren können.

Wenn Sie den Computer in den Multicolor-Modus umschalten, können Sie

die doppelte Anzahl der Farben benutzen. Es können dann gleichzeitig vier verschiedene Farben angewählt werden; die Auflösung des Bildschirms ist in diesem Fall nur halb so groß. In der Multicolor-Betriebsart ist der Schirm in 160 Punkte waagrecht und 200 Punkte senkrecht aufgeteilt. (Der PRINT-Befehl kann auch in diesem Modus nicht verwendet werden.)

Sprites können in jeder Grafikbetriebsweise benutzt werden. Sprites sind kleine Bilder, die in Hochauflösung dargestellt werden und auf verschiedene Plätze des Arbeitsspeichers abgespeichert werden können. Der VIC-Chip erkennt diese Bilder und gibt sie auf dem Schirm aus. Die Sprites können dann auf dem Bildschirm bewegt werden. Ein Sprite kann maximal 540 Punkte (24 waagrecht und 21 senkrecht) umfassen. Das bedeutet, daß für ein Sprite 63 Bytes benötigt werden. Ein Sprite kann in einer der 16 möglichen Farben dargestellt werden und kann in der Höhe, in der Breite oder sogar in der Höhe und in der Breite verdoppelt werden. Beim Commodore 64 werden vornehmlich diese Sprites zu grafischen Darstellungen verwendet. Eine andere Möglichkeit ist der Multicolor-Sprite. Diese Art Sprites sind in der Anwendung wesentlich komplizierter, als die oben genannten Einfachfarben-Sprites.

Sprites, selbstdefinierte Zeichen und hochauflösende Grafik werden in einem der folgenden Kapitel näher erklärt.

Der VIC-Chip ist nicht der einzige Chip, mit dem Sie wunderbar spielen können. Was der VIC-Chip für Ihr Auge ist, ist der SID-Chip für Ihr Ohr.

Mit dem SID-Chip können Sie nämlich Geräusche erzeugen. Es wird in naher Zukunft sogar möglich sein, mit dem SID Sprache zu erzeugen. Einige Programmierer arbeiten bereits an einem Sprach-Synthesizer für den SID.

Der Klang eines Tones wird durch vier Eigenschaften bestimmt: Tonhöhe, Lautstärke, Klangfarbe und Anschlag. Die Klangfarbe und der Anschlag sind dafür verantwortlich, daß überhaupt verschiedene Instrumente mit dem Gehör unterschieden werden können. Mit einem entsprechenden Programm können Sie alle vier Eigenschaften eines Tones bestimmen.

Die Tonhöhe wird physikalisch durch die Frequenz bestimmt. Die Frequenz wird in Hertz (Abkürzung: Hz) angegeben. Eine hohe Zahl entspricht einem hohen Ton, eine tiefe Zahl einem tiefen Ton. Da das menschliche Gehör individuell verschieden ist, können Töne von 16 Hz bis 16 000 Hz wahrgenommen werden. Der Kammerton a hat beispielsweise die Frequenz 440 Hz.

Die Lautstärke ist in 16 verschiedenen Stufen einstellbar; von 0 (aus-

geschaltet) bis 15 (volle Lautstärke).

Die Klangfarbe wird hauptsächlich durch die Art der akustischen Wellen bestimmt. Sie ist unter anderem für ein bestimmtes Instrument charakteristisch. Der Commodore bietet vier verschiedene Grundformen von akustischen Wellen an: Dreieck, Sägezahn, Rechteck und Rauschen.

Die letzte Eigenschaft, durch die ein Ton bestimmt ist, ist der Anschlag. Jeder Klavierspieler weiß, wie wichtig die Art des Anschlages für den Ausdruck eines Tones ist. Der Anschlag und der Verlauf eines Tones wird durch das Anklingen und das Abschwellen dieses Tones bestimmt.

Wie bereits erwähnt, gibt es keine direkten BASIC-Befehle, um den VIC- und den SID-Chip zu programmieren. Sie müssen deshalb den POKE-Befehl benutzen und direkt auf angegebene Speicherplätze bestimmte Werte schreiben. Es existieren insgesamt 24 Speicherplätze, die für die Programmierung des VIC zuständig sind (53 248 bis 53 271). Ähnliches gilt für den SID. Hier stehen 25 Speicherplätze zur Verfügung: 53 272 bis 53 296.

Zufallszahlen

Sie werden in zwei Programmen dieses Kapitels den BASIC-Befehl RND entdecken können. RND ist eine Abkürzung für 'random' (= Zufall). Mit diesem Befehl können Zufallszahlen erzeugt werden.

Zufallszahlen sind besonders für Computerspiele von großer Bedeutung. Sollten Sie beispielsweise einen elektronischen Würfel oder ein Glücksrad programmieren wollen, benötigen Sie dazu Zufallszahlen. Diese werden in dem Computer durch eine mathematische Routine - dem Zufallszahlengenerator - erzeugt. Die Zahlen sind natürlich nicht wirklich 'zufällig', sie werden nur so berechnet, daß Sie keine Wiederholungen oder andere Gesetzmäßigkeiten feststellen können.

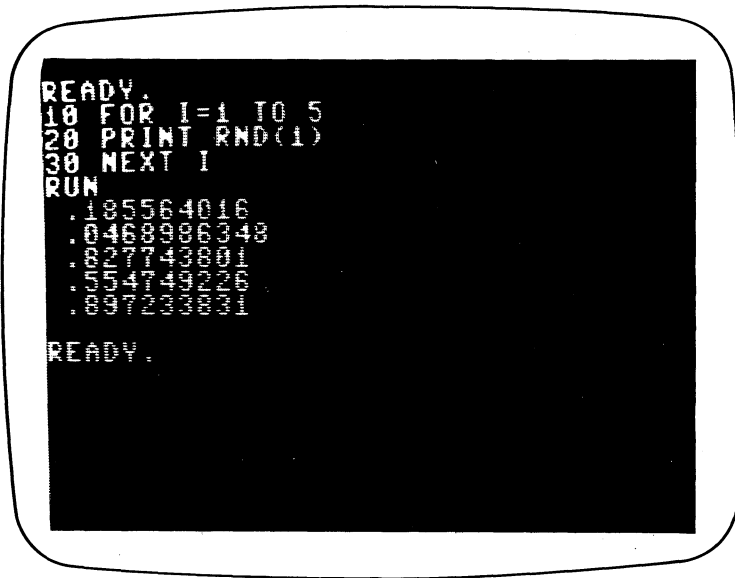
Nach dem RND-Befehl wird in Klammern eine Zahl angegeben, mit der der Zufallszahlengenerator initialisiert wird. Es können verschiedene Zahlen angegeben werden.

Schalten Sie Ihren Computer aus und wieder an. Tippen Sie nun folgendes Beispiel ein:

```
10 FOR I=1 TO 5
20 PRINT RND(1)
30 NEXT I
```

Sie werden nun fünf Zahlenreihen auf dem Bildschirm sehen können:

```
.185564016
.0468986348
.827743801
.554749226
.897233831
```



Bei diesen Zahlen werden Sie sicherlich keine Gesetzmäßigkeit feststellen können. Wenn Sie den Computer ausschalten und das Programm erneut laufen lassen, werden Sie immer wieder diese Zufallszahlen bekommen. Sie werden sich vielleicht fragen, warum Sie immer wieder die gleichen Zufallszahlen erhalten, da diese doch eigentlich 'zufällig' sein sollten. Das ist eine gute Frage. Um dieses verstehen zu können, müssen Sie wissen, wie der Zu-

fallszahlengenerator arbeitet. Wenn der Computer eingeschaltet wird, wird eine Zahl, die den Zufallszahlengenerator initialisiert, automatisch ausgewählt. Diese Zahl erzeugt eine Liste von Zahlen. Immer dann, wenn die RND Funktion mit irgendeiner positiven Zahl, die in Klammern nach RND angegeben wird, aufgerufen wird, wird die nächste Zahl aus der Liste der Zufallszahlen ausgewählt. Wird der Befehl RND mit einer darauffolgenden negativen Zahl verwendet, wird eine neue Liste von Zufallszahlen erzeugt. Wenn RND mit der Ziffer 0 initialisiert wird, wird der Zufallszahlengenerator mit dem aktuellen Wert der Uhr initialisiert. Hier ein Beispiel:

```
10 X=RND(-1)
20 FOR I=1 TO 10
30 PRINT RND(1)
40 NEXT I
```

Sooft Sie dieses Programm laufen lassen, werden Sie die gleiche Liste von Zufallszahlen erhalten, die mit RND(1) generiert wurde. Wenn Sie nun die erste negative Zahl ändern, werden Sie damit eine andere Zufallszahlenliste erzeugen. Versuchen Sie es einmal! Ändern Sie RND(-1) in RND(-2) um.

Wie können Zufallszahlen am besten erzeugt werden? Eine Möglichkeit wäre, den Zufallszahlengenerator unter Verwendung der eingebauten Uhr mit einer negativen Zahl zu initialisieren.

```
10 N=TI
20 X=RND (-N)
30 FOR I=1 TO 10
40 PRINT RND(1)
```

Immer dann, wenn das Programm gestartet wird, erzeugt es eine unterschiedliche Liste von Zufallszahlen.

Eine andere Möglichkeit ist die Verwendung von RND(0). Hier wird ebenfalls eine Liste von Zufallszahlen generiert, wobei der Zufallszahlengenerator mit einem Wert initialisiert wird, der sich nach einer Formel aus dem aktuellen Wert der eingebauten Uhr berechnet.

Sie werden sicherlich schon bemerkt haben, daß die erzeugten Zufallszahlen nicht immer für das jeweilige Spiel direkt verwendet werden können. Sie werden meistens keine Dezimalzahlen, sondern ganze Zahlen (z. B. für

ein Würfelspiel) benötigen. Die mit RND erzeugten Zufallszahlen liegen nämlich immer zwischen 0 und 1. Um ganze Zahlen zu erhalten, müssen Sie die mathematische Funktion INT (für INTe ger = ganze Zahlen) verwenden. Wenn Sie den Bereich der Zufallszahlen erweitern wollen, müssen Sie die Zufallszahl mit der entsprechenden Zahl multiplizieren.

Lassen Sie uns nun Zufallszahlen zwischen 0 und 10 erzeugen. Es ist ganz einfach. Schauen Sie sich folgendes Programm an:

```
10 N=INT(10*RND(0))
20 PRINT N
30 GOTO 10
```

Wenn Sie dieses Programm starten, werden Sie immer eine Zahl zwischen 0 und 10 erhalten. Die Zahlen werden nacheinander auf dem Bildschirm ausgegeben. Die Zahl, mit der Sie RND multiplizieren, legt den Bereich fest. INT gibt dem Computer die Information, daß nur ganze Zahlen verwendet werden sollen. In einigen Fällen werden Sie evtl. die Zahl 0 ausschließen wollen. Um das zu erreichen, müssen Sie nur eine 1 addieren. Schauen Sie sich das Programm an:

```
10 N=INT(10*RND(0)+1)
20 PRINT N
30 GOTO 10
```

Wenn Sie RND benutzen, müssen Sie unbedingt darauf achten, daß Sie keinen Fehler bei der Benutzung von Klammern machen. Denken Sie immer daran, daß die Anzahl der öffnenden gleich der Anzahl der schließenden Klammern sein muß.

Rahmen und Hintergrundfarbe

In dem VIC-Chip kontrollieren zwei Speicherplätze die Farbe des Bildschirms. Dabei wird zwischen Rahmen- und Hintergrundfarbe unterschieden. Die Farben können mit dem POKE-Befehl gesetzt werden. Zahlen zwischen 0 und 15 sind möglich. Jede Zahl repräsentiert eine bestimmte Farbe. Mit POKE 53280 können Sie die Rahmenfarbe, mit POKE 53281 die Hintergrundfar-

be wählen.

10 POKE 53280,11:POKE 53281,15

Diese Anweisung setzt die Rahmenfarbe auf den Wert Grau 1 und die Hintergrundfarbe auf den Wert Grau 3. Grau 3 ist ein helles Grau (fast weiß); Grau 1 ist ein dunkles Grau. Jede Farbe des Commodore 64 hat ihre eigene Nummer.

POKE 53280 53281	Farbe
0	schwarz
1	weiß
2	rot
3	türkis
4	violett
5	grün
6	blau
7	gelb
8	orange
9	braun
10	hellrot
11	grau 1
12	grau 2
13	hellgrün
14	hellblau
15	grau 3

Zahlen sind leider relativ schwer zu behalten. Deshalb können Sie anstelle der Zahlen auch mnemonische Variablen verwenden. Schauen Sie sich nachstehendes Programm einmal an:

```
10 RF=53280: HF=53281
20 REM FARBEN
30 SW=0: WE=1: RO=2: TM=3: VI=4: GR=5
40 BL=6: GE=7: OG=8: BR=9: HR=10: G1=11
50 G2=12: HG=13: HB=14: G3=15
60 POKE RF, SW: REM POKE RAHMENFARBE SCHWARZ
```

70 POKE HF,GR: REM POKE HINTERGRUNDFARBE GRUEN

Das ist eine gute Methode, sich das Programmieren der Farben ein wenig zu vereinfachen. Sie sollten aber darauf achten, daß Sie mit den Variablennamen nicht in Bedrängnis kommen. So ist beispielsweise der Variablenname OR für 'orange' verboten, da er bereits für OR (das logische Oder) vergeben ist. Deshalb ist für die Farbe Orange die Abkürzung OG gewählt worden.

Zeichendarstellung in Farbe

Wie Sie wissen, können Sie durch gleichzeitiges Drücken der CTRL- und einer Zahlentaste (0 bis 9) die Farbe ändern, mit der die Zeichen auf dem Bildschirm ausgegeben werden. Sie können weitere acht Farben auswählen, indem Sie die COMMODORE-Taste und eine Zifferntaste gleichzeitig drücken. Die Namen der ersten acht Farben sind auf der Vorderseite der entsprechenden Tasten vermerkt. Die Namen des zweiten Farbensatzes können Sie dort allerdings nicht finden.

Sie müssen die Farbe, in der Sie die Zeichen auf dem Bildschirm ausgeben, unter Umständen dann ändern, wenn Sie vorher die Hintergrundfarbe verändert haben. Es kann nämlich vorkommen, daß die Farben nicht ausreichend kontrastieren und Sie die ausgegebenen Zeichen auf dem Bildschirm nicht erkennen können.

Das Anwählen der Farben geschieht mit der PRINT-Anweisung.

Versuchen Sie, folgendes Programm zu starten (Denken Sie daran, die in Klammern angegebenen Tasten zu drücken.)

05 POKE 53281,11:POKE 53280,15	
10 PRINT "(CTRL und BLK) SCHWARZ	(CTRL und WHT)WEISS"
20 PRINT "(CTRL und RED) ROT	(CTRL und CYN) TUERKIS"
30 PRINT "(CTRL und PUR) VIOLETT	(CTRL und GRN) GRUEN"
40 PRINT "(CTRL und BLU) BLAU	(CTRL UND YEL) GELB"
50 PRINT "(COMM und BLK) ORANGE	(COMM und WHT) BRAUN"
60 PRINT "(COMM und RED) HELLROT	(COMM und CYN) GRAU 1"
70 PRINT "(COMM und PUR) GRAU 2	(COMM und GRN) HELLGRUEN"
80 PRINT "(COMM und BLU) HELLBLAU	(COMM und YEL) GRAU 3"

Wenn Sie obiges Programm starten, werden Sie sehen, daß jeder Name einer Farbe in der entsprechenden Farbe ausgegeben wird. Nur eine Farbe, Grau 3, wird nicht ausgegeben, da als Hintergrundfarbe ebenfalls Grau 3 gewählt wurde. So ist es verständlich, daß Sie hier keinen Kontrast bekommen. Wählen Sie mit POKE 53281 eine andere Hintergrundfarbe, und Sie werden sehen, daß immer die entsprechende Farbe des Hintergrundes nicht als Text zu sehen ist.

Grafische Symbole

Der Commodore 64 verfügt über verschiedene grafische Symbole. Diese Symbole sind auf der Vorderansicht einiger Tasten zu sehen. Die Anwendung dieser grafischen Symbole ist relativ einfach. Sie können Sie nämlich mit dem PRINT-Befehl ausgeben. Durch die Kombination verschiedener Symbole können Sie eine 'Mosaikgrafik' erzeugen. Wir nennen sie deshalb 'Character-Grafik'.

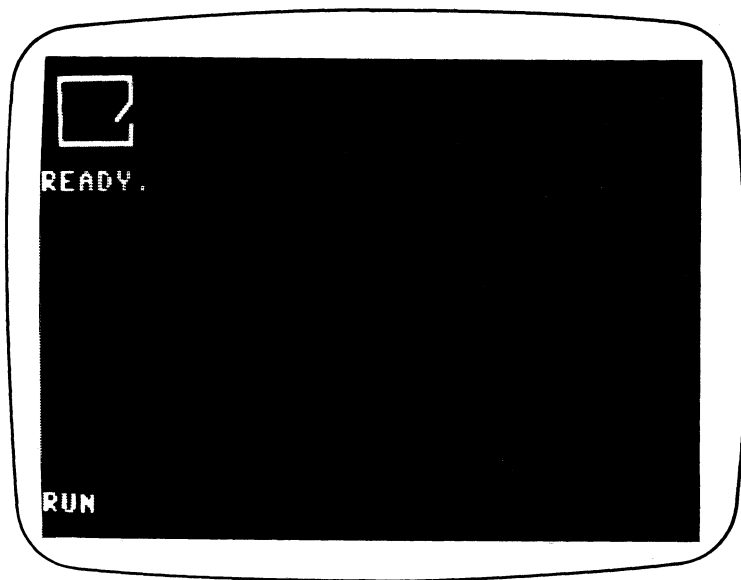
Nun wollen wir einmal ein Kästchen auf dem Bildschirm ausgeben. Für das Programm benutzen wir die grafischen Symbole. Geben Sie folgendes Programm ein (COMM bedeutet, daß die COMMODORE-Taste gedrückt werden muß).

```
10 PRINT "(CLR)(CRSR DN)";  
20 PRINT"(SPACE)(SHIFT O)(COMM Y)(COMM Y)(SHIFT P)"  
30 PRINT"(SPACE)(COMM H)(SPACE)(SPACE)(SHIFT N)"  
40 PRINT"(SPACE)(SHIFT L)(COMM P)(COMM P)(SHIFT@)"
```

Wenn Sie dieses Programm starten, wird ein Quadrat in die linke obere Ecke des Bildschirms gezeichnet. Sie können dieses Quadrat aber auch an eine andere Stelle des Bildschirms setzen. Benutzen Sie die CRSR DOWN-Taste, und ändern Sie Zeile 10 wie folgt:

```
10 PRINT "(CLR)(5 CRSR DN)";
```

Drücken Sie die CRSR DOWN-Taste fünfmal. Jetzt wird das Quadrat auf dem Bildschirm etwas tiefer ausgegeben. Fügen Sie vor Zeile 20 bis Zeile 40 nicht nur ein, sondern mehrere Leerzeichen ein. Das Quadrat wird sich nun von links nach rechts bewegen.



Mit SHIFT U, I, J und K stehen Grafiksymbole zur Verfügung, mit denen Sie 'runde Ecken' darstellen können. Wenn Sie alle vier zusammen benutzen, erzeugen Sie damit einen Ball (er wird ein wenig eckig erscheinen). Eine Möglichkeit, den Ball auf dem Bildschirm darzustellen ist, zwei verschiedene PRINT-Anweisungen zu verwenden.

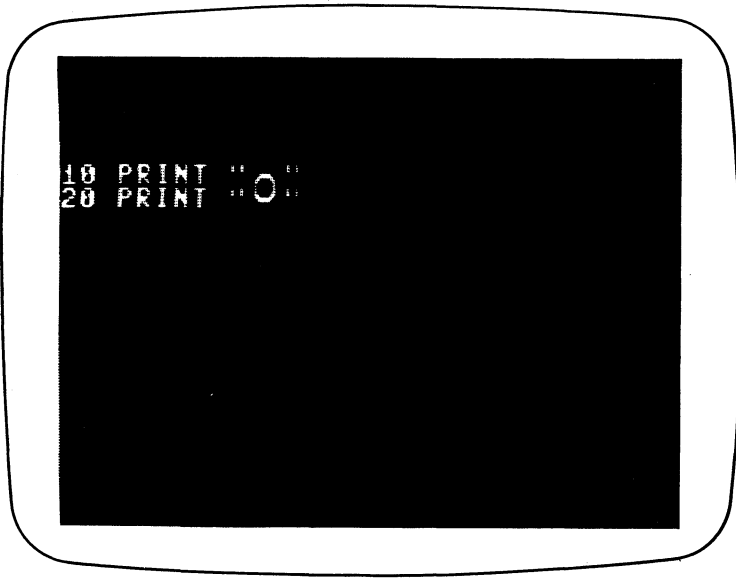
```
10 PRINT"(SHIFT U)(SHIFT I)"
```

```
20 PRINT"(SHIFT J)(SHIFT K)"
```

Um dieses Programm etwas zu vereinfachen, können wir auch den ganzen Ball einer einzigen Textvariablen zuweisen. Diese Textvariable wird dann mit einer PRINT-Anweisung ausgegeben. Schauen Sie sich das folgende Programm an:

```
10 A$="(SHIFT U)(SHIFT I)(CRSR LT)(CRSR LT)(CRSR DN)(SHIFT J)(SHIFT K)"
```

```
20 PRINT"(CLR)";A$
```



Die Bildschirmgrafik kann gelöscht werden, indem Leerzeichen ausgegeben werden. Wenn wir in dem obigen Beispiel den 'Ball' löschen wollen, müssen wir einen anderen String als einen Block von vier Leerzeichen und die Bewegungsabläufe definieren. Erweitern Sie das Programm um folgende Zeilen:

```
30 FOR I=1 TO 1000: NEXT I
40 B$="(2 SPACES)(2 CRSR LT)(CRSR DN)(2 SPACES)"
50 PRINT"(HOME)";B$
```

Wird die Zeile 30 ausgeführt, macht der Computer eine kleine Pause. Wir haben diese Warteschleife bereits kennengelernt. Mit Zeile 40 wird ein String definiert, der die gleiche Größe wie der Ball hat. Mit diesem String kann dann in Zeile 50 der Ball gelöscht werden.

Wird eine Grafik als String gespeichert, hat das den Vorteil, daß diese Grafik an irgendeiner Stelle auf dem Bildschirm ausgegeben werden kann. Wie leicht das möglich ist, können Sie daran erkennen, daß Sie beispielsweise die Zeile 20 des obigen Programms nur folgendermaßen ändern müssen:

```
20 PRINT"(CLR)(CRSR DN)(CRSR RT)";A$
```

Spiele Sie ein wenig mit der CRSR-Right und CRSR-Down Taste. Vergessen Sie nicht, daß Ihr Bildschirm nur 25 Zeilen und 40 Spalten hat.

Auch geübte Programmierer machen Fehler, da das Ausgeben und der Entwurf einer Grafik sehr viel Konzentration erfordern. Die meisten Programme laufen leider nicht sofort. Jedes Programm muß aus diesem Grund gut ausgetestet werden.

Ein elektronischer Würfel

Wir können den BASIC-Befehl RND und die Character-Grafik in einem Programm verwenden, um das Spiel mit einem Würfelpaar zu simulieren. Die Liste dieses einfachen Programms sehen Sie nachstehend. Geben Sie dieses Programm ein, und denken Sie daran, daß Sie die entsprechenden Tasten drücken und nicht den in Klammern stehenden Text eintippen. '(8 CRSR DN)' bedeutet, daß Sie achtmal die CRSR DOWN-Taste drücken müssen.

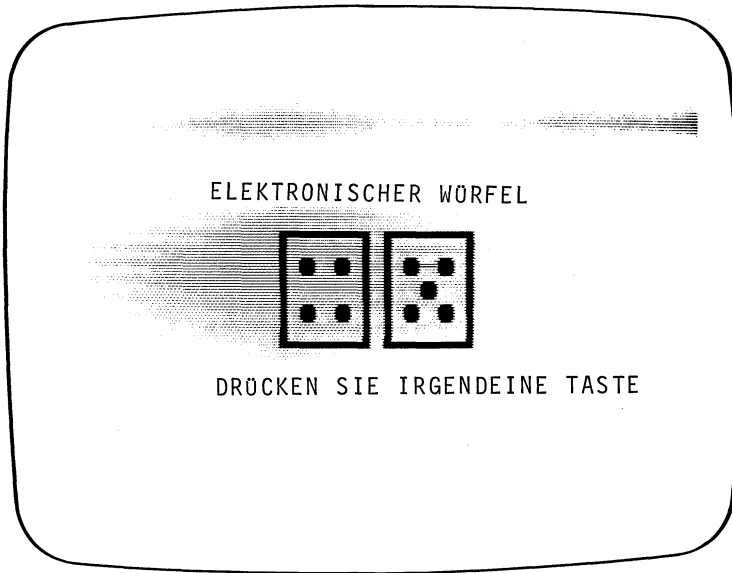
In Zeile 10 wird die Rahmen- und Hintergrundfarbe festgelegt. Mit Zeile 30 beginnt eine FOR..NEXT-Schleife.

```
30 FOR I=1 TO 25
```

In dieser FOR..NEXT-Schleife werden zwei Zufallszahlen erzeugt, die den Variablen D1 und D2 zugeordnet werden. Jede Zufallszahl liegt in dem Bereich von 1 und 6. Sie symbolisiert die Augenzahl des jeweiligen Würfels. Ein Bild eines jeden Würfels ist in einem Unterprogramm, das mit Zeile 1100 beginnt, gespeichert.

Der Cursor wird in Zeile 60 in die HOME Position gesetzt. Dann wird er 10 Zeilen tiefer und 20 Zeichen nach rechts versetzt. Der Befehl ON in Zeile 70 wird dazu verwendet, den entsprechenden Würfel auszugeben. Wenn D1 1 ist, wird ein Bild eines Würfels mit der entsprechenden Augenzahl ausgegeben (hier 1). Der Cursor wird in Zeile 80 neu gesetzt und der zweite Würfel entsprechend ausgegeben.

Beachten Sie, daß jeder Würfel bis auf die Augenzahl gleich aussieht. Wenn die Würfel schnell hintereinander ausgegeben werden, erhält man den Eindruck, daß die Würfel sich bewegen. Dazu dient die Schleife FOR I=1 TO



25. Die Würfel ändern also insgesamt 25mal die Augenzahl. Der Computer hält in Zeile 120 an und wartet mit dem GET-Befehl auf das Drücken irgendeiner Taste. Sobald eine Taste gedrückt wird, beginnt das Spiel erneut.

Das Eintippen dieses Programms ist erheblich einfacher, als es auf den ersten Blick aussehen mag. Geben Sie das Unterprogramm von Zeile 1100 bis 1170 mit den Werten für den ersten Würfel ein. Versuchen Sie sich einmal daran zu erinnern, wie man Texte editieren kann. Es ist ganz einfach, das Unterprogramm für den zweiten Würfel zu duplizieren. Schauen Sie einfach nach, falls Sie es nicht mehr wissen sollten.

"Elektronischer Wuerfel"

10 POKE 53280,11:POKE 53281,15

20 PRINT"(CLR) (BLK) (8 CRSRS DN) (15 CRSR RT) Elektronischer Wuerfel";

30 FOR I=1 TO 25

40 D1=INT(6*RND(0))+1

50 D2=INT(6*RND(0))+1

60 PRINT"(HOME) (10 CRSR DN) (15 CRSR RT)";

```
70 ON D1 GOSUB 1100,1200,1300,1400,1500,1600
80 PRINT"(HOME) (10 CRSR DN) (21 CRSR RT)";
90 ON D2 GOSUB 1100,1200, 1300,1400,1500,1600
100 NEXT I
110 PRINT"(HOME) (16 CRSR DN) (14 CRSR RT) DRUECKEN SIE IRGENDEINE TASTE"
120 GET C$:IF C$=""THEN 120
130 GOTO 30
1100 PRINT"(SHIFT O) (COMM Y) (COMM Y)(COMM Y)(SHIFT P) (CRSR LEFT) (CRSR
LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1110 PRINT"(COMM H) (3 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1120 PRINT"(COMM H) (SHIFT Q) (COMM N) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1150 PRINT"(COMM H) (3 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1160 PRINT"(SHIFT L) (COMM P) (COMM P) (COMM P) (COMM O)"
1170 RETURN
1200 PRINT"(SHIFT O) (COMM Y) (COMM Y)(COMM Y) (SHIFT P) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1210 PRINT"(COMM H) (SHIFT Q) (2 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1220 PRINT"(COMM H) (3 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1250 PRINT"(COMM H) (2 SPACES)(SHIFT Q) (COMM N) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1260 PRINT"(SHIFT L) (COMM P) (COMM P) (COMM P) (COMM O)"
1270 RETURN
1300 PRINT"(SHIFT O) (COMM Y) (COMM Y) (COMM Y) (SHIFT P)(CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1310 PRINT"(COMM H) (SHIFT Q) (2 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1320 PRINT"(COMM H) (SPACE) (SHIFT Q) (SPACE) (COMM N) (CRSR LEFT)
(CRSR LEFT)(CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1350 PRINT"(COMM H) (2 SPACES) (SHIFT Q) (COMM N) (CRSR LEFT) (CRSR LEFT)
(CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1360 PRINT"(SHIFT L) (COMM P) (COMM P) (COMM P) (COMM O)"
1370 RETURN
```

```
1400 PRINT"(SHIFT O) (COMM Y) (COMM Y) (COMM Y) (SHIFT P) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1410 PRINT"(COMM H) (SHIFT Q) (SPACE) (SHIFT Q) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1420 PRINT"(COMM H) (3 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1450 PRINT"(COMM H) (SHIFT Q)(SPACE) (SHIFT Q) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1460 PRINT"(SHIFT L) (COMM P) (COMM P) (COMM P) (COMM O)"
1470 RETURN
1500 PRINT"(SHIFT O) (COMM Y) (COMM Y) (COMM Y) (SHIFT P) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1510 PRINT"(COMM H) (SHIFT Q) (SPACE) (SHIFT Q) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1520 PRINT "(COMM H) (SPACE) (SHIFT Q) (SPACE) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1550 PRINT"(COMM H) (SHIFT Q) (SPACE) (SHIFT Q) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1560 PRINT"(SHIFT L) (COMM P) (COMM P) (COMM P) (COMM O)"
1570 RETURN
1600 PRINT"(SHIFT O) (COMM Y) (COMM Y) (COMM Y) (SHIFT P) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1610 PRINT"(COMM H) (SHIFT Q) (SHIFT Q) (SHIFT Q) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1620 PRINT"(COMM H) (3 SPACES) (COMM N) (CRSR LEFT) (CRSR LEFT)(CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR DOWN)";
1650 PRINT"(COMM H) (SHIFT Q) (SHIFT Q) (SHIFT Q) (COMM N) (CRSR LEFT)
      (CRSR LEFT) (CRSR LEFT) (CRSR LEFT) (CRSR LEFT)(CRSR DOWN)";
1660 PRINT"(SHIFT L) (COMM P) (COMM P) (COMM P) (COMM O)"
1670 RETURN
```

Sprite Grafik

Sprite Grafik ist erheblich schwieriger als Character Grafik. Sie können den PRINT-Befehl nämlich nicht zur Ausgabe eines Sprites auf dem Bildschirm benutzen. Die Programmierung von Sprites erfolgt ausschließlich mit dem POKE- (evtl. auch mit dem PEAK-) Befehl. Dieses Thema soll später noch einmal genauer behandelt werden. Hier soll nur ein kleiner Überblick über das Funktionieren der Sprite Grafik gegeben werden.

Ein Sprite stellt ein frei programmiertes Objekt in hochauflösender Grafik dar. Durch Angabe der Position kann das Sprite auf dem Bildschirm verschoben werden. Es kann außerdem die Farbe der Sprites verändert werden; der Zusammenstoß zweier Sprites kann auf einfache Weise registriert werden; ein Sprite kann sich vor oder hinter einem anderen Sprite vorbeibewegen und kann mit einem Befehl in seiner Größe verändert werden.

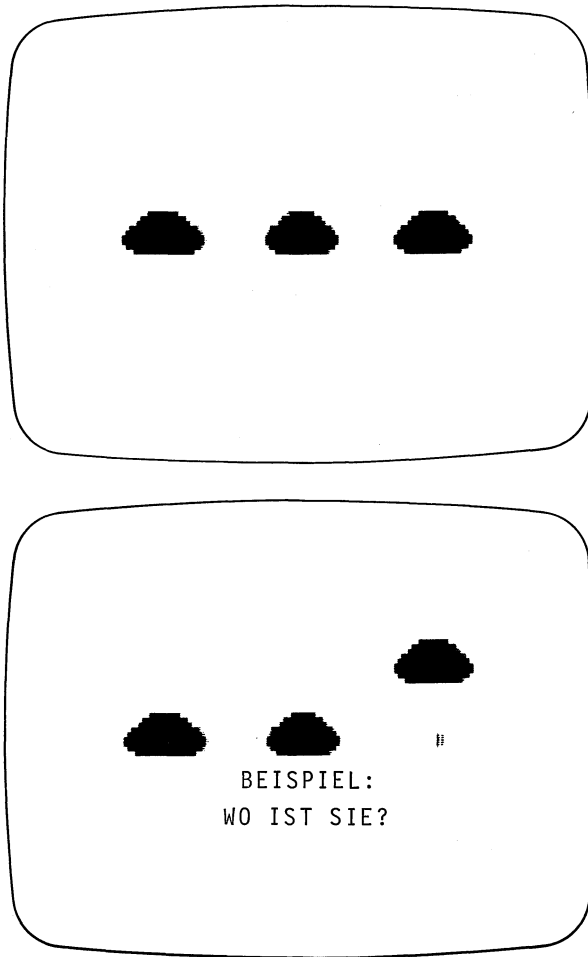
Bei der Anwendung dieser Sprites treten vier Nachteile auf:

1. Aufgrund des eingeschränkten Speicherplatzes kann mit höchstens drei verschiedenen Sprites gleichzeitig gearbeitet werden.
2. Nur wenn Sie einen Sprite über einen Teil des Schirms bewegen, ist die Angelegenheit nicht ganz so kompliziert.
3. Sprites sind relativ schwierig zu entwerfen. Sie benötigen nämlich viel Zeit, um sich ein Bild aus verschiedenen einzelnen Punkten zusammenzusetzen.
4. Es existiert keine Möglichkeit, Sprites mit BASIC-Befehlen zu programmieren. Sie müssen immer den PEAK- bzw. POKE-Befehl verwenden. Dadurch erhalten Sie aber schwer lesbare Programme.

Das Muschelspiel

Um zu sehen, wie man mit Sprites arbeitet, sollten Sie sich das folgende Beispielprogramm ansehen. Es ist eine Version des alten 'Muschelspiels'. Eine Erbse wird unter einer von drei Muschelschalen versteckt. Sie müssen nun raten, unter welcher Muschelschale sich die Erbse befindet. Die Mu-

schelschalen werden hin und her geschoben. Nun müssen Sie erneut raten, unter welcher Schale sich die Erbse befindet. Bei diesem Spiel werden die Sprites dazu verwendet, um diese Muschelschalen auf dem Bildschirm darzu-



stellen. Mit diesen 'Muschelschalen' wird dann die 'Erbse' verdeckt, die mit Character- Grafik dargestellt wird. So wird die Erbse, ein in grüner Farbe dargestelltes geshiftetes Q, durch den Muschelschalensprite verdeckt.

Die Erbse wird natürlich nicht tatsächlich bewegt. Ein Zufallszahlengenerator erzeugt Zufallszahlen zwischen 1 und 3. Die 'Erbse' wird dann unter die entsprechende 'Muschelschale' gesetzt, bevor diese hochgehoben wird. Dieses Spiel (so wie es hier beschrieben wurde) ist eigentlich gar kein richtiges Spiel, da es von selbst kontinuierlich läuft. Sie können es aber abändern, indem Sie einen INPUT- oder einen GET-Befehl einfügen. Damit sind dann Sie derjenige, der die 'Muschelschale hochhebt'. Aus dem nachstehenden Programm können Sie ein richtiges Spiel machen, in dem Sie raten müssen, unter welcher 'Muschelschale' sich die 'Erbse' befindet.

In diesem Programm wird der REMark-Befehl sehr häufig verwendet, damit Sie verstehen können, was die einzelnen Anweisungen bedeuten.

In Zeile 30 werden die Sprites aufgebaut. Hier wird dem VIC-Chip mitgeteilt, welche Sprites und wieviele Sie benutzen wollen. Das Programm benutzt nur drei einander identische Sprites. Die Beschreibung der Sprites wird mit POKE in einem Sprite-Block mit der Nummer 13 abgelegt. Die Speicherplätze 2040, 2041 und 2042 sind nicht Bestandteil des VIC-Chips. Der Vic-Chip schaut sich jedoch diese Speicherplätze an und weiß somit, wo der Aufbau der Sprites abgespeichert ist.

Mit den Zeilen 60, 70 und 80 werden die Sprites positioniert. Denken Sie sich den Bildschirm als ein Fenster, in welchem die Sprites bewegt werden können. In diesem Fenster gibtes ca. 344 horizontale und 255 vertikale Positionen, in denen ein Sprite dargestellt und positioniert werden kann. Die Variable X wird für die horizontale und die Variable Y für die vertikale Positionierung benutzt.

MUSCHEL

```
10 PRINT"(CLR)";:POKE 53280,12:POKE 53281,15:REM HINTERGRUNDFARBE
    HELLGRAU
20 V=53248:REM VIDEO CHIP
30 POKEV+21,7:REM INITIALISIEREN DER SPRITES
35 FOR I=0 TO 62: READD:POKE 832+I,D:NEXT:REM LIES DATEN FUER SPRITE IN
    BLOCK 13
40 POKE 2040, 13:POKE 2041,13: POKE 2042,13: REM ALLE SPRITES WERDEN IN
    BLOCK 13 GESPEICHERT
50 POKEV+39,9:POKEV+40,9:POKEV+41,9:REM FARBE DER SPRITES BRAUN
55 POKEV+23,7:POKE V+29,7: REM AUSDEHNUNG DER SPRITES IN X UND Y
```

RICHTUNG

```

60 POKEV+0,85:POKEV+1,120:REM SETZE SPRITE 0 AUF 85,120
70 POKEV+2,160: POKE V+3,120:REM SETZE SPRITE 1 AUF 160,120
80 POKEV+4,235:POKE V+5,120:REM SETZE SPRITE 2 AUF 235,120
90 PRINT"(BLU)(HOME)(15 CRSR DN)(5 SPACES) WILLKOMMEN BEIM MUSCHELSPIEL"
100 GOSUB 30000:REM WARTESCHLEIFE
110 GOSUB 15000:REM LOESCHEN DES TEXTES
120 PRINT"(BLU)(HOME)(15 CRSR DN)(6 SPACES) DER SINN DES SPIELES IST ES,"
130 PRINT"(BLU)(HOME)(15 CRSR DN)(14 SPACES) DIE ERBSE ZU FINDEN"
135 GOSUB 30000: REM DELAY
140 PRINT"(BLU)(HOME)(19 CRSR DN)(13 SPACES) PEA > (GRN)(SHIFT Q)
    (BLU) < PEA"
145 GOSUB 30000
150 GOSUB 15000:REM LOESCHEN DES BILDSCHIRMS
165 GOSUB 30000:REM WARTESCHLEIFE
170 PRINT"(BLU)(HOME)(14 CRSR DN)(16 SPACES) BEISPIEL:"
175 GOSUB 30000: REM WARTESCHLEIFE
190 GOSUB 12000: REM ERBSE UNTER DER MITTLEREN MUSCHEL
200 GOSUB 13000: REM AUFDECKEN DER MUSCHEL
255 GOSUB 30000:REM WARTESCHLEIFE
270 GOSUB 14000:REM SCHIEBEN DER MUSCHELN
365 GOSUB 30000:REM WARTESCHLEIFE
370 PRINT"(BLU)(HOME)(15 CRSR DN)(14 SPACES) WO IST SIE?"
375 GOSUB 30000: REM WARTESCHLEIFE
380 A=INT(3*(RND(0))+1):
382 IF A=1 THEN GOSUB 12000:GOSUB 13200:GOSUB 13100:GOSUB 13000
384 IF A=2 THEN GOSUB 12050:GOSUB 13000:GOSUB 13100: GOSUB 13200
386 IF A=3 THEN GOSUB 12100:GOSUB 13200: GOSUB 13000: GOSUB 13100
410 GOSUB 15000: REM LOESCHEN DES TEXTES
420 GOSUB 30000: GOSUB 30000:GOSUB 30000:REM ZEITVERZOEGERUNG
999 RUN
9999 REM SPRITE DATEN
10000 DATA 0,0,0,0,0,0,0,0,0
10010 DATA 0,0,0,0,0,0,0,0,0
10020 DATA 0,0,0,0,0,0,0,0,0
10030 DATA 1,254,0,7,255,128,15,255,192
10040 DATA 63,255,240,127,255,248,255,255,252

```

```
10050 DATA 255,255,252,127,255,248,31,255,224
10060 DATA 0,0,0,0,0,0,0,0,0
12000 REM; ERBSE UNTER DER MITTLEREN MUSCHEL
12010 PRINT"(HOME)(12 CRSR DN)(19 SPACES)(GRN)(SHIFT Q)(BLU)(11 SPACES) "
12020 RETURN
12050 REM: ERBSE UNTER DER LINKEN MUSCHEL
12060 PRINT"(HOME)(12 CRSR DN)(10 CRSR RT)(GRN)(SHIFT Q)(BLU)(30 SPACES)"
12070 RETURN
12100 REM: ERBSE UNTER DER RECHTEN MUSCHEL
12110 PRINT"(HOME)(12 CRSR DN)(29 SPACES)(GRN)(SHIFT Q)(BLU)"
12120 RETURN
13000 REM: AUFDECKEN MITTLERE MUSCHEL
13010 FOR I=120 TO 90 STEP -1
13020 POKE V+2,160:POKE V+3,I
13030 NEXT I
13040 FOR I=1 TO 500: NEXT I
13050 FOR I=90 TO 120
13060 POKE V+2,160:POKE V+3,I
13070 NEXT I
13080 RETURN
13100 REM: AUFDECKEN RECHTE MUSCHEL
13110 FOR I=120 TO 90 STEP -1
13120 POKE V+0,235:POKE V+1,I
13130 NEXT I
13140 FOR I=1 TO 500:NEXT I
13150 FOR I=90 TO 120
13160 POKE V+0,235:POKE V+1,I
13170 NEXT I
13180 RETURN
13200 REM: AUFDECKEN LINKE MUSCHEL
13210 FOR I=120 TO 90 STEP -1
13220 POKE V+4,85: POKE V+5,I
13230 NEXT I
13240 FOR I=1 TO 500:NEXT I
13250 FOR I=90 TO 120
13260 POKE V+4,85: POKE V+5,I
13270 NEXT I
```



```
13280 RETURN
14000 REM:VERSCHIEBEN DER MUSCHELN
14010 FOR I=160 TO 85
14020 POKE V+2,I:POKE V+3,120
14030 NEXT I
14040 FOR I=85 TO 235
14050 POKE V+0,I:POKE V+1,120
14060 NEXT I
14070 FOR I=235 TO 85 STEP -1
14080 POKE V+4,I:POKE V+5,120
14090 NEXT I
14100 RETURN
15000 REM: LOESCHEN DES TEXTES
15010 PRINT"(BLU)(HOME)(14 CRSR DN)(39 SPACES)"
15020 PRINT"(BLU)(HOME)( 15 CRSR DN) (39 SPACES)"
15030 PRINT"(BLU)(HOME)(16 CRSR DN)(39 SPACES)"
15040 PRINT"(BLU)(HOME)(17 CRSR DN)(39 SPACES)"
15050 PRINT"(BLU)(HOME)(18 CRSR DN)(39 SPACES)"
15060 PRINT"(BLU)(HOME)(19 CRSR DN)(39 SPACES)"
15070 RETURN
30000 REM:WARTESCHLEIFE
30010 FOR I=1 TO 1000:NEXT I
30020 RETURN
```

Musik mit dem SID

Die Programmierung von Musik ist wie die Programmierung von Grafik eine wirkliche Herausforderung für jeden Programmierer. Wie beim VIC-Chip können Sie auch mit dem SID-Chip leider nicht in der Programmiersprache BASIC arbeiten. Das ist nur über den POKE-Befehl möglich. Die ganze Angelegenheit wird noch etwas komplizierter, da Sie sich mit dem PEAK-Befehl nicht die Zahlen ansehen können, die auf den Speicherplätzen des SID-Chips gespeichert sind. PEAK wird immer die Zahl 0 zurückgeben.

Nun ein paar Fakten darüber, wie Sie mit dem SID Musik und Geräusche erzeugen können. Wenn Sie nicht alles sofort verstehen sollten, braucht Sie das nicht zu ängstigen. Sie sollten zunächst einmal einen allgemeinen Überblick bekommen.

Damit Sie überhaupt ein Geräusch erzeugen können, müssen Sie zuerst die Lautstärkeregelung des SID-Chips setzen. Es gibt 16 verschiedene Stufen (von 0 bis 15). 0 bedeutet dabei, daß der SID praktisch ausgeschaltet ist; mit der Stufe 15 erhalten Sie die volle Lautstärke. Sie können die Stufe der gewünschten Lautstärke setzen, indem Sie eine Zahl zwischen 0 und 15 mit dem POKE-Befehl auf die Speicherzelle 50296 schreiben.

Wie bereits erwähnt, können Sie den Klang eines Tones mit verschiedenen Eigenschaften bestimmen.

Der Anschlag und das Halten eines Tones bilden eine Einheit. Sie werden gesetzt, indem zwei Zahlen zusammen als Summe auf einen Speicherplatz geschrieben werden. Die Zahl für das Halten des Tones kann 128, 64, 32, 16 oder die Summe irgendeiner dieser Zahlen sein. Zählen Sie die Zahl für den Anschlag des Tones mit irgendeiner Zahl für das Halten des Tones (8, 4, 2 oder 1 oder die Summe irgendeiner dieser Zahlen) zusammen. Je größer die Zahl für den Anschlag ist, desto länger braucht der Ton, bis er die maximale Lautstärke erreicht. Wenn die Zahl für den Anschlag 0 ist, wird der Ton sofort in der vorher gewählten vollen Lautstärke ausgegeben. Mit dem Anschlag können Sie also das Anklingen des Tones regeln. Das Abschwellen des Tones wird durch die Zahl für das Halten des Tones bestimmt. Je größer diese Zahl ist, desto länger klingt der Ton aus.

Setzen Sie den Anschlag und das Halten für Stimme 1, schreiben Sie die Summe dieser beiden Zahlen mit dem POKE-Befehl in die Speicherzelle 54277. Für die Stimmen 2 und 3 können Sie eine ähnliche Zahl in die Speicherzellen 54284 und 54291 schreiben.

Die bisherigen Ausführungen in Bezug auf Anklingen und Ausklingen eines Tones bezogen sich auf das Nachahmen von Musikinstrumenten, die keinen gehaltenen Ton erzeugen können. Ein solches Musikinstrument ist z. B. das Klavier. Bei Musikinstrumenten, die einen Ton halten können (z. B. die Orgel), wird das Anklingen als 'Sustain' und das Abschwellen als 'Release' bezeichnet. Sustain und Release können wie das Anklingen und Abschwellen eines Tones bestimmt werden, indem die Summe für die entsprechenden Werte auf den Speicherplatz von 54278 für die erste Stimme, auf den Speicherplatz 54285 und auf den Speicherplatz 54292 für 2. und 3. Stimme setzen.

Die Frequenz des Tones wird mit dem Befehl POKE in zwei Speicherplätzen abgelegt. Diese Speicherplätze werden in einen oberen (hi = high) und in einen niederen (lo = low) Speicherplatz aufgespalten. Schauen Sie sich die Werte für die einzelnen Noten in der Tabelle an, die Sie ein paar Seiten weiter finden. Schreiben Sie dann mit dem POKE-Befehl auf die Speicherzelle 54272 die Zahl, die Sie in der Spalte LOW finden und auf die Speicherzelle 54273 die Zahl, die Sie in der Spalte HIGH finden. Für die zweite Stimme werden die Speicherplätze mit 54279 und 54280; für die dritte Stimme mit 54286 und 54287 beziffert.

Die Klangfarbe wird hauptsächlich durch die Art der Wellen bestimmt. Der Commodore 64 bietet Ihnen vier Grundformen an: Dreieck, Sägezahn, Rechteck und Rauschen. Die Wellenform für Stimme 1 wird durch den Wert bestimmt, der auf dem Speicherplatz 54276 steht.

POKE 54276,33 für Sägezahn

POKE 54276,17 für Dreieck

POKE 54276,65 für Rechteck

POKE 54276,129 für Rauschen

Für die Stimmen 2 und 3 müssen Sie mit dem POKE-Befehl dieselben Zahlen auf die Speicherplätze 54283 (Stimme 2) und 54290 (Stimme 3) setzen. Wenn Sie dieses in einem Programm durchführen, bleibt der Ton kontinuierlich stehen. Um den Ton auszuschalten, schreiben Sie für jede Stimme eine 0 auf den entsprechenden Speicherplatz. Jetzt können Sie eine neue Note zum Erklingen bringen.

Sie können sich nun ein kleines Beispielprogramm ansehen, mit dem Sie einen Ton erzeugen. Vergessen Sie bitte nicht, den Lautstärkeregler an Ihrem Fernsehapparat aufzudrehen.

```
10 VL=54296: REM LAUTSTAERKE
20 LO=54272:HI=54273:REM FREQUENZ
30 WF=54276:REM WELLENFORM
40 AD=54277: REM ANSCHLAG/HALTEN
50 SR=54278: REM SUSTAIN/RELEASE
60 POKE VL,15
70 POKE AD,190:POKE SR,0
80 POKE WF,33
90 POKE LO,28: POKE HI,214
100 FOR I=1 TO 500:NEXT I:REM TONDAUER
110 POKE AD,0:POKE SSR,0: POKE WF,0:POKE VL,0
```

Wie Sie sehen, ist es ein ganzes Stück Arbeit, einen einzigen Ton zu produzieren. Spielen Sie ein wenig, indem Sie die einzelnen Werte für Anschlag, Halten, Sustain, Release, Wellenform und Tondauer ändern.

Auf diese Art und Weise macht selbst das Spielen eines einfachen Liedes sehr viel Arbeit. Mit ein bißchen Einfallsreichtum schreiben Sie ein Programm, das eine Note nach der anderen spielt. Die Programmliste für ein solches Programm können Sie nachstehend sehen. Dieses Programm erzeugt Felder, in denen die Frequenzen der Noten eines Liedes und die Tondauer abgespeichert sind. Die Information ist in DATA-Statements gespeichert.

Die DATA-Anweisung beginnt mit der Zeilennummer 10 000 und enthält die in Gruppen gespeicherte Informationen. Der erste Wert C4 ist die Musiknote C der vierten Oktave. Die nächsten beiden Zahlen enthalten jeweils die entsprechende Frequenz. (Denken Sie an den Speicherplatz HI und LO.) Immer wenn Sie das Zeichen '#' sehen, ist ein Halbtonschritt gemeint. Gegenwärtig sind nur zwei Oktaven als DATA-Block abgespeichert. Sie können die Tonleiter entsprechend erweitern und auch die restlichen Töne in den DATA-Block eintippen.

Am Ende der DATA-Anweisung in Zeile 10 040 sehen Sie ein paar ungewöhnliche Zeichen und Zahlen. R,0,0 sthet für den musikalischen 'Rest'.

Die READ-Anweisung des Programms verwendet das X als letztes Zeichen, damit erkannt werden kann, daß keine weitere Informationen folgen.

Die DATA-Anweisungen, die mit Zeile 20 000 beginnen, enthalten die Informationen der Noten für das entsprechende Lied. Diese Informationen sind in zwei Gruppen aufgeteilt. Die erste Gruppe enthält die Noten, die zweite Gruppe den Notenwert (= Dauer des Tones). Der Notenwert 16 bedeutet eine

ganze Note, der Notenwert 8 eine halbe Note, der Notenwert 4 eine viertel Note usw.

In der Programmliste sind (wie bereits erwähnt) nur zwei Oktaven gespeichert. Durch Änderung der Noten und der Notenwerte können Sie jedes beliebige Lied programmieren. Das Tempo wird durch den Wert der Variablen T in Zeile 300 bestimmt.

LIED

```

10 SID=54272:REM SPEICHERPLATZ SID-CHIP
20 DIM N$(94), NL(94),NH(94),S$(200),D(200),SH(200),SL(200)
30 X=1:REM X-1 NOTENDATEN
40 READ N$(X): IF N$(X)="X" THEN 60: REM PRUEFUNG AUF ENDE DER NOTENDATEN
50 READ NH(X):READ NL(X):X=X+1:GOTO 40
60 N=1:REM N-1 NOTEN DES LIEDES
70 READ S$(N): IF S$(N)="X" THEN GOTO 90
80 READ D(N):N=N+1: GOTO 70
90 FOR I=1 TO N-1
100 FOR K=1 TO X-1: IF S$(I)=N$(K) THEN SL(I)=NL(K):SH(I)=NH(K):GOTO 120
110 NEXT K
120 NEXT I
300 VL=54272:VH=54273:AD=54277:SR=54278:CR=54276:VO=54296:T=10:
  PRINT"(CLR)"
310 POKE VO,15:REM VOLLE LAUTSTAERKE
320 FOR I=1 TO N-1: REM ANFANG DER SCHLEIFE FUER DAS LIED
330 POKE VL,SL(I):POKE VH,SH(I)
340 PRINT"POKE SH,";SH(I);"POKE SL,";SL(I);"NOTENWERT";D(I)
350 POKE AD;90:POKE SR,128: POKE CR,17
360 FOR K=1 TO T*D(I): REM SCHLEIFE NOTENWERT
370 NEXT K
380 POKE AD,0: POKE SR,0:POKE CR,0:REM AUSSCHALTEN
390 NEXT I
400 POKE VO,0: REM LAUTSTAERKE ZURUECKNEHMEN
9999 REM NOTENDATEN
10000 DATA C4,16,195,C#4,17,195,D4,18,209,D#4,19,239
10010 DATA E4,21,31,F4,22,96,F#4,23,181
10020 DATA G4,25,30,G#4,26,156,A4,28,49,A#4,29,233,B4,31,165,C5,33,135

```

10030 DATA C#5,35,134,D5,37,162,D#5,39,233,E5,42,62,F5,45,198,F#5,47,107

10040 DATA G5,50,60,G#5,51,197,A5,56,99,A#5,59,190,B5,64,188,C6,68,149,R,0,0,X

19999 REM NOTEN UND NOTENWERTE DES LIEDES

20000 DATA C4,16,D4,16,E4,16,F4,16,G4,16,A4,16,B4,16,R,16,R,16

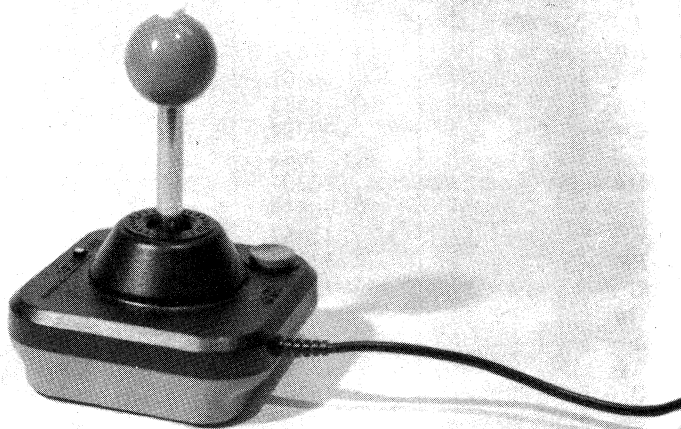
20010 DATA C5,16,D5,16,E5,16,F5,16,G5,16,A5,16,B5,16,C6,16,X

Noten und zugehörige Frequenzen

Note		Frequenzen	
Oktave	Dezimal	HI	Low
C - 0	268	1	12
C# - 0	284	1	28
D - 0	301	1	45
D# - 0	318	1	62
E - 0	337	1	81
F - 0	358	1	102
F# - 0	379	1	123
G - 0	401	1	145
G# - 0	425	1	169
A - 0	451	1	195
A# - 0	477	1	221
B - 0	506	1	250
C - 1	536	2	24
C# - 1	568	2	56
D - 1	602	2	90
D# - 1	637	2	125
E - 1	675	2	163
F - 1	716	2	204
F# - 1	758	2	246
G - 1	803	3	35
G# - 1	851	3	83
A - 1	902	3	134
A# - 1	955	3	187
B - 1	1012	3	244
C - 2	1072	4	48
C# - 2	1136	4	112
D - 2	1204	4	180
D# - 2	1275	4	251
E - 2	1351	5	71
F - 2	1432	5	152
F# - 2	1517	5	237
G - 2	1607	6	71
G# - 2	1703	6	167
A - 2	1804	7	12
A# - 2	1911	7	119

Oktave	Dezimal	HI	Low
B - 2	2025	7	233
C - 3	2145	8	97
C# - 3	2273	8	225
D - 3	2408	9	104
D# - 3	2551	9	247
E - 3	2703	10	143
F - 3	2864	11	48
F# - 3	3034	11	218
G - 3	3215	12	143
G# - 3	3406	13	78
A - 3	3608	14	24
A# - 3	3823	14	239
B - 3	4050	15	210
C - 4	4291	16	195
C# - 4	4547	17	195
D - 4	4817	18	209
D# - 4	5103	19	239
E - 4	5407	21	31
F - 4	5728	22	96
F# - 4	6069	23	181
G - 4	6430	25	30
G# - 4	6812	26	156
A - 4	7217	28	49
A# - 4	7647	29	223
B - 4	8101	31	165
C - 5	8583	33	135
C# - 5	9094	35	134
D - 5	9634	37	162
D# - 5	10207	39	223
E - 5	10814	42	62
F - 5	11457	44	193
F# - 5	12139	47	107
G - 5	12860	50	60
G# - 5	13625	53	57
A - 5	14435	56	99
A# - 5	15294	59	190
B - 5	16203	63	75
C - 6	17167	67	15
C# - 6	18188	71	12
D - 6	19269	75	69
D# - 6	20415	79	191
E - 6	21629	84	125
F - 6	22915	89	131
F# - 6	24278	94	214
G - 6	25721	100	121
G# - 6	27251	106	115
A - 6	28871	112	199
A# - 6	30588	119	124
B - 6	32407	126	151
C - 7	34334	134	30
C# - 7	36376	142	24

Note		Frequenzen	
Oktave	Dezimal	HI	Low
D - 7	38539	150	139
D# - 7	40830	159	126
E - 7	43258	168	250
F - 7	45830	179	6
F# - 7	48556	189	172
G - 7	51443	200	243
G# - 7	54502	212	230
A - 7	57743	225	143
A# - 7	61176	238	248
B - 7	64814	253	46



Die Joysticks

Nachdem Sie sich mit den Möglichkeiten Ihres Commodore 64, Grafik und Musik darzustellen, vertraut gemacht haben, haben Sie sicherlich auch den Wunsch, Ihre eigenen Programme zu schreiben. Im Zusammenhang mit Spielprogrammen sind die Joysticks von großer Bedeutung. Die Benutzung der Joysticks ist bei dem Commodore 64 sehr einfach.

Joysticks sind keine neue Erfindung. Sie wurden bereits Mitte der 70er Jahre für Video-Spiele eingesetzt. Der Atari Joystick wurde sehr schnell bei diesen Video-Spielen zur Standardausrüstung. Die heutigen Joysticks arbeiten auf ähnliche Art und Weise.

Der Atari Joystick ist ein Joystick, der nach dem Schalterprinzip arbeitet. Ein anderes Prinzip ist der analoge Joystick, der mit dem Schalterjoystick nicht kompatibel ist. In dem Joystick befinden sich fünf Schalter, vier für die Richtungen Nord (nach oben), Süd (nach unten), Ost (nach rechts) und West (nach links), sowie einen Schalter für 'Feuer'. Die vier anderen Richtungen (Nordwest, Nordost, Südwest und Südost) können mit gleichzeitigem Schließen von zwei Schaltern geschlossen werden.

Wenn Sie einen Atari Joystick besitzen (oder irgendeinen anderen, der wie der Atari arbeitet), stecken Sie den Stecker einfach in den Control Port 2 auf der rechten Seite des Commodore 64. Warum Port 2? Control Port 1 kann ebenfalls benutzt werden; die Programmierung ist aber ein wenig schwieriger. Aus diesem Grund wollen wir für diesen Abschnitt erst einmal bei Port 2 bleiben.

Wenn wir uns einen bestimmten Speicherplatz anschauen, können wir feststellen, in welche Richtung der Joystick bewegt worden ist.

```
10 PRINT PEEK(56320): GOTO 10
```

Wenn Sie dieses kleine Programm starten, werden nacheinander eine Reihe von Zahlen auf dem Bildschirm erscheinen. Die Zahl sollte 127 sein. Drücken Sie nun den Knüppel des Joysticks nach vorne. Was geschieht? Da Sie intern einen Schalter schließen, ändert sich die Zahl. Eine Zahl von 0 bis 255 wird durch 8 Bits dargestellt. Fünf dieser Bits repräsentieren die Schalter innerhalb des Joysticks. Wir können deshalb die ausgegebene Zahl dazu verwenden, um festzustellen, in welcher Richtung der Joystick bewegt wird.

```
10 A=PEEK(56320)
20 IF A=127 THEN PRINT"ZENTRUM";
30 IF A=126 THEN PRINT"NORDEN";
40 IF A=125 THEN PRINT"SUEDEN";
50 IF A=123 THEN PRINT"WESTEN";
60 IF A=119 THEN PRINT"OSTEN";
70 IF A=122 THEN PRINT"NORDWESTEN";
80 IF A=118 THEN PRINT"NORDOSTEN";
90 IF A=117 THEN PRINT"SUEDOSTEN";
100 IF A=121 THEN PRINT"SUEDWESTEN";
110 B=PEEK(56320) AND 16
120 IF B=0 THEN PRINT"--FEUER":GOTO 10
130 PRINT"":GOTO 10
```

Diese Zahlen, die Ihnen von dem Joystick über den Control Port vermittelt werden, können Sie in einem Programm benutzen, und Sie können entscheiden, welche Aktion durchgeführt werden soll.

Wie funktionieren die Game Paddles, die mit den Joysticks verwandt sind? Mit Ausnahme des 'Feuer' Knopfes haben Paddles keine Schalter. Anstelle der Schalter besitzen sie Potentiometer, wie sie auch für die Lautstärkeregelung in Ihrem Fernsehapparat oder in Ihrer Stereoanlage verwendet werden. Der Computer kann auch bei diesen Game Paddles die Richtung erkennen. Bei einem Paddle bekommen Sie Zahlen zwischen 0 und 255. Stellen Sie ein Paddle ganz nach rechts, und Sie werden die Zahl 0 bekommen. Wenn Sie jetzt das Paddle ganz nach links drehen, erhalten Sie die Zahl 255.

Wenn Sie Paddles benutzen wollen, stecken Sie diese in einen anderen Stecker: den Control Port 1. Führen Sie nun das kleine Programm aus.

```
10 PRINT PEEK (54298): GOTO 10
```

Starten Sie diese Zeile und schauen Sie sich den Bildschirm an. Drehen Sie dabei an einem der Paddles. Sollte sich die Zahl auf dem Bildschirm nicht ändern, probieren Sie das andere Paddle aus. Sie können den 'Feuer'-Knopf der Paddles lesen, wenn Sie sich mit PEAK die Speicherzelle 56321 ansehen.

Was fehlt noch?

Wie Sie gesehen haben, müssen Sie sehr oft die Befehle PEAK und POKE verwenden, um Grafik oder Musik erzeugen zu können. Dadurch wird das Programmieren sehr umständlich, und es ist schwierig, Programme wirklich fehlerfrei zu schreiben.

Commodore erkannte dieses Problem und entwickelte ein Programm mit dem bezeichnenden Namen 'Easy Graphics'. Dieses Programm erweitert das vorhandene BASIC des Commodore 64 so, daß Sie einen großen Teil der Grafikbefehle über neue BASIC-Befehle programmieren können. Damit wird das Programmieren sehr erleichtert.

Schauen Sie sich einmal die folgenden Seiten dieses Buches genauer an. Hier werden 'Werkzeuge' beschrieben, die Ihnen die Programmierung von Grafik und Musik erheblich erleichtern.

10 Was kommt nach BASIC?

Ihr Computer ist ein Teil eines größeren Systems. Später einmal werden Sie sicherlich Ihr System ausbauen, wie Sie es vielleicht mit Ihrem Stereo- oder Videosystem bereits getan haben. Jede Erweiterung wird Ihnen weitere Möglichkeiten bieten. Der Computer ist nicht nur ein Spielzeug, an dem Sie sich erfreuen können. Seine Leistungsfähigkeit nimmt mit einer Erweiterung des Systems stark zu.

Welche Richtung sollten Sie dabei einschlagen? Welche neuen Möglichkeiten erhalten Sie, wenn Sie die Erweiterung durchführen? Wieviel sollten Sie dafür anlegen, und welche zusätzlichen Komponenten sollten Sie kaufen? Diese Fragen können letztlich nur Sie selbst beantworten. Das vorliegende letzte Kapitel gibt einen kleinen Einblick in die Möglichkeiten, die sich für die Erweiterung Ihres Computersystems bieten.

Ausbau der Hardware

Wenn Sie bis jetzt noch keines haben, sollte der erste Punkt Ihres 'Wunschzettels' ein Commodore Diskettenlaufwerk 1541 sein. Das Laden und Speichern geht von der Diskette erheblich schneller als von der Datassette.

Außerdem ist eine Diskette für anspruchsvollere Programme wie beispielsweise Textverarbeitungsprogramme, Datenbankprogramme und Programme zur Tabellenverarbeitung unbedingt notwendig.

Wenn Sie Ihren Computer in einer Schule, einem Labor oder in einem größeren Büro einsetzen wollen, gibt es die Möglichkeit, die einzelnen Computer zu einem Kommunikationsnetzwerk zu verbinden. Dieses Kommunikationsnetzwerk wird 'Local Area Network' (LAN) genannt. Das LAN verbindet kleine Computer miteinander, so daß sie untereinander Informationen und Programme austauschen können. Ein weiterer Vorteil dieses LAN Netzwerkes ist, daß Peripheriegeräte wie Drucker, Diskettenlaufwerke oder Plattenlaufwerke gemeinsam benutzt werden können.

Spreadsheets und professionelle Software

Immer dann, wenn ein Computer auf den Markt kommt, folgt eine wahre Flut von Software, die Programmangebote aus verschiedenen Bereichen beinhaltet. Der Commodore 64 ist keine Ausnahme. Einige Programme erfüllen jedoch nicht die gesteckten Ziele. Gute Programme sind leider oft zu teuer und können sich deshalb nicht durchsetzen. ein gutes Spiel kann leicht von weniger guten Spielen unterschieden werden: Entweder es gefällt Ihnen, oder es gefällt Ihnen nicht. Die Auswahl eines Programmes für die praktische oder professionelle Anwendung ist - wie Sie sich sicherlich denken können - erheblich schwieriger.

In einem der vorangegangenen Kapitel haben Sie ja bereits etwas darüber gelesen. Als Beispiel hatten wir ein Datenbankprogramm besprochen.

Unter dem Namen 'Spreadsheet' wird eine besondere Art von Programmen zusammengefaßt. Diese Spreadsheet-Programme arbeiten ähnlich wie ein Textverarbeitungsprogramm. Sie sind allgemein anwendbar, da man mit ihnen die verschiedensten Problemstellungen lösen kann. So können beispielsweise diese Programme dazu dienen, Ihre Haushaltskalkulation oder Ihre Terminplanungen durchzuführen. Mit diesen Programmen können Werte in eine Tabelle eingetragen und verwaltet werden. Eine Tabelle besteht aus einer Anzahl von Zeilen und Spalten, in die die entsprechenden Informationen eingetragen sind. Soll eine Zahl in der Tabelle geändert werden, müssen auch die anderen Zahlen, die von dieser Zahl abhängig sind, geändert werden. Mit den Spreadsheet-Programmen sind Sie in der Lage, eine große Tabelle aufzubauen. Sie können sich über den Bildschirm ein 'Fenster' schaffen, mit dem Sie sich Teile der Tabelle ansehen können. Dabei ist es möglich, die einzelnen Daten in der Tabelle logisch zu verknüpfen oder bestimmte Daten

abzufragen.

Das erste dieser Programme war 'VisiCalc'. Andere 'Calc' Programme folgten; darunter waren einige bessere und natürlich auch schlechtere. Da fast alle 'Calc' Programme in BASIC geschrieben wurden, sind sie leider für die praktische Benutzung zu langsam.

Ein ausgezeichnetes Spreadsheet-Programm für den Commodore 64 wurde in Schweden geschrieben. Dieses Programm, das den Namen 'Calc Result' trägt, nutzt die Möglichkeiten des Commodore 64 voll aus. Sie können damit eine Tabelle aufbauen, die 63 Spalten und 252 Zeilen enthält. Diese Tabelle wird als eine Seite dargestellt. Insgesamt können 32 verschiedene Seiten auf einmal benutzt werden. Aufgrund dieser Aufteilung in Tabellen und Seiten, wird Calc Result auch als dreidimensionales Spreadsheet bezeichnet.

Andere Programmiersprachen

Den menschlichen Sprachen werden bestimmte Eigenschaften zugeschrieben. Französisch gilt beispielsweise als sehr ausdrucksvolle Sprache, und die Sprachen der Antike - Griechisch und Latein - sind für ihre Präzision und ihren strengen logischen Aufbau bekannt.

Zieht man eine Parallele zu den Computersprachen, so kann man BASIC mit Englisch vergleichen: BASIC ist weltweit verbreitet und sehr komplex. BASIC hat fast genauso viele Ausnahmen wie Regeln.

An dieser Stelle sollen nun einige andere Programmiersprachen kurz besprochen werden.

Pilot. Eine der neuen Programmiersprachen ist Pilot. Von dieser Programmiersprache wird behauptet, daß sie sehr einfach und trotzdem sehr leistungsfähig sei. Die Anwendungen sind allerdings stark eingeschränkt. Pilot wurde für die schulische Ausbildung entwickelt. Die Sprache ist für den Unterricht sehr beliebt geworden.

Die Kommandos von Pilot sind relativ einfach. Selbst kleine Kinder können in Pilot programmieren.

Wie BASIC hat Pilot verschiedene Entwicklungen durchlaufen. Eine Erweiterung dieser Programmiersprache war die Möglichkeit der grafischen Darstellung. Leider besitzt das Commodore 64 Pilot diese Möglichkeit nicht. Pilot kann von einem Diskettenlaufwerk geladen werden. Wenn Sie mit Pilot

arbeiten, wird Ihnen der erste große Unterschied zu BASIC sofort auffallen. Sie können in Pilot mit vier verschiedenen Betriebsarten arbeiten.

Pilot beginnt immer mit dem Command-Modus. Hier können Sie angeben, was Sie als nächstes tun wollen. Eine Möglichkeit ist, zu dem 'Immediate Modus' überzugehen. In diesem Modus können Sie Pilot-Kommandos testen, die aus nur einem einzigen Buchstaben bestehen. Wenn Sie sich einmal mit diesen Kommandos vertraut gemacht haben, ist es möglich, diese auch in dem 'Edit-Modus' einzugeben, um damit Programme zu schreiben. Mit dem 'Run-Modus' wird das Pilot Programm schließlich gestartet.

Ohne sich in Einzelheiten zu verlieren, ist für Lehrer sicherlich die Möglichkeit, ein Frage-Antwort-Spiel für Schüler zu entwickeln, besonders interessant.

Schauen wir uns einmal folgendes Programm an:

TS: WELCHE FARBEN HAT DIE KANADISCHE FLAGGE?

A:

M:ROT & WEISS

Aufgrund der ersten Zeile wird auf dem Bildschirm der Text WELCHE FARBEN HAT DIE KANADISCHE FLAGGE ausgegeben. T bedeutet Type ('Type' entspricht dem BASIC-Befehl PRINT'); mit S wird der Bildschirm gelöscht (S für Screen = Bildschirm). Der nächste Befehl, A, steht für 'Accept' (= annehmen). Hier wird auf eine Antwort gewartet. Der Befehl A ist in etwa mit dem INPUT-Befehl vergleichbar. M ist die Abkürzung für 'Match' (= vergleichen). Die Antwort wird mit der richtigen Antwort ROT & WEISS verglichen. Durch Abänderung des Match-Kommandos ist es möglich, daß der Rechner sogar die Antwort WEISS & ROT (umgekehrte Reihenfolge) oder falsch geschriebene Wörter (wie ROD und WEIS) verstehen kann. Die Möglichkeiten, die Pilot bezüglich grafischer Darstellungen bietet, sind ebenfalls sehr interessant.

Anstelle des Sprite Generators gestattet Pilot die Verwendung einfacher Grafikbefehle.

In Pilot können auch logische Operationen und Rechnungen durchgeführt werden.

LOGO. LOGO findet bei einigen Programmierern sehr viel Interesse.

LOGO wurde am Massachusetts Institute of Technology entwickelt und wird manchmal als Dialekt von LISP bezeichnet. LISP ist eine Programmiersprache, die hauptsächlich zur Programmierung von Problemen in dem Bereich der 'künstlichen Intelligenz' (artificial intelligence) eingesetzt wird. Die Grundkonzepte von LISP und LOGO sind sehr ähnlich.

Die große Stärke von LOGO ist das Prozedurenkonzept. Die Leistungsfähigkeit liegt darin, daß aus alten neue Prozeduren gebildet werden können und diese neuen Prozeduren wiederum benutzt werden können, um andere Prozeduren zu schaffen. Deshalb wird LOGO oft als 'rekursiv' bezeichnet. (Diesen Begriff kann man sich an folgendem Beispiel verdeutlichen: Stellen Sie sich ein Bild vor, das einen Maler zeigt, der ein Bild malt. Dieses Bild zeigt den gleichen Maler, der wiederum ein Bild mit gleichem Inhalt malt etc.) Genau diese Fähigkeit des 'Lernens' macht LOGO so faszinierend. Das bekannteste Merkmal von LOGO ist vielleicht das Turtle. Mit 'Turtle' wird ein dreieckiger Cursor bezeichnet, mit dem Grafiken in Hochauflösung auf den Bildschirm gebracht werden können. Man kann sich den Cursor als einen Bleistift vorstellen, der abgesenkt und angehoben werden kann, um die Zeichnung zu machen. Wenn der Bleistift abgesenkt ist, wird eine Linie gezeichnet. Folgendes Programm zeichnet ein Quadrat:

```
TO SQUARE
PENDOWN
REPEAT 4 (FORWARD 50 RIGHT 90)
END
```

Der Vorgang ist einfach. Die Feder wird mit dem Befehl PENDOWN abgesenkt. Nun bewegt sie sich 50 Einheiten vorwärts. Eine Einheit entspricht einem Punkt auf dem hochauflösenden Grafikbildschirm. Anschließend wird der 'Bleistift' in einem Winkel von 90 Grad um weitere 50 Einheiten nach rechts bewegt. Dieser Vorgang wird viermal wiederholt, um das Quadrat zu zeichnen. Haben Sie erst einmal dem Computer gezeigt, wie ein Quadrat gezeichnet wird, brauchen Sie die obige Befehlsfolge nicht ständig zu wiederholen. Sie schreiben einfach das Wort SQUARE und der Computer zeichnet Ihnen ein Quadrat.

In LOGO können Sie kein Textverarbeitungs- oder Spreadsheet-Programm schreiben. Komplizierte mathematische Zusammenhänge können ebenfalls nicht programmiert werden. Ähnliches gilt für Pilot. Diese beiden Sprachen sind

nicht als Ersatz für BASIC gedacht und weisen somit eigene Besonderheiten und Eigenschaften auf. Beide Sprachen versuchen unsere Vorstellungskraft anzusprechen und den Gedankengängen nachzukommen, die sich in unserem Gehirn abspielen.

Schnelles BASIC. Der größte Nachteil von BASIC ist, daß es sehr langsam ist. Warum eigentlich?

Der Grund dafür ist, daß BASIC meistens interpretativ ausgeführt wird. Man spricht in diesem Fall auch von einem BASIC-Interpreter. Das bedeutet, daß der Befehl während der Laufzeit entschlüsselt wird und erst danach ausgeführt werden kann. Nach der Entschlüsselung des Befehls schaut der Rechner in dem BASIC-ROM nach und führt den entsprechenden Befehl aus. Das alles kostet sehr viel Zeit.

Mit einem BASIC-Compiler werden BASIC-Programme erheblich schneller. In diesem Fall wird das BASIC-Programm vor der Ausführung in ein Programm in Maschinsprache übersetzt. Diese Übersetzung geschieht mithilfe des Compilers. Der Computer kann die Maschinsprache direkt verstehen und direkt ausführen. Deshalb läuft ein kompiliertes Programm erheblich schneller.

BASIC-Compiler werden von verschiedenen Firmen angeboten. Sie müssen darauf achten, daß dieser BASIC-Compiler den kompletten Wortschatz Ihres Commodore 64 BASIC enthält. Es kann durchaus vorkommen, daß Sie bei der compilativen Ausführung des BASIC-Programms einige BASIC-Befehle nicht benutzen dürfen. So ist es beispielsweise gar nicht selten, daß die Befehle für die Programmierung der Diskette (z. B. Öffnen von Diskettenfiles und Schreiben auf diesen Diskettenfiles) von einem BASIC-Compiler nicht unterstützt werden. Wenn Sie mit einem BASIC-Compiler arbeiten, benötigen Sie auch in den meisten Fällen ein Diskettenlaufwerk. Sie müssen in diesem Fall möglicherweise Ihren Rechner erweitern.

Assembler Sprache. Wenn Sie zeitkritische Probleme mit Ihrem Commodore 64 lösen wollen, müssen Sie schließlich zur Assembler-Sprache greifen. Die Assembler-Sprache wird auch als maschinenorientierte Sprache bezeichnet. Dieser Terminus entstand durch die Tatsache, daß sich die Befehle, die in einer Assembler-Sprache möglich sind, an der Maschine, d. h. an Ihrem speziellen Computer, orientieren. Wie bereits erwähnt, können Sie in BASIC geschriebene Programme bis zu einem gewissen Grad ohne Änderungen

auch auf irgendeinem anderen Computer ausführen. Bei Programmen, die in Assembler-Sprache geschrieben wurden, ist das jedoch nicht möglich. Eine Assembler-Sprache ist schwieriger als BASIC zu erlernen. Wenn Sie sich gut in BASIC eingearbeitet haben, sollten Sie es nicht versäumen, doch irgendwann einmal die Assembler-Sprache zu erlernen. Sie bekommen dadurch sehr viel Verständnis für die Arbeitsweise Ihres Computers.

Mensch/Maschine Interface

Der Begriff Mensch/Maschine Interface wird immer häufiger für die Art und Weise, wie man mit einem Computer kommunizieren kann, angetroffen. In den meisten Fällen arbeiten Sie mit der Tastatur, wenn Sie irgendwelche Daten in den Rechner eingeben wollen. Das ist aber nicht die einzige Möglichkeit. Sie haben bereits die Joysticks und die Game Paddles kennengelernt. Besitzen Sie auch einen Lichtgriffel?

Sie können mit einem Lichtgriffel Daten direkt auf dem Schirm eingeben. Der Computer kann die Position des Lichtgriffels auf dem Schirm erkennen und in einem Programm verwerten. In diesem Lichtgriffel befindet sich ein lichtempfindlicher Transistor. Der Griffel wird an Control Port 1 angeschlossen.

Die Anwendung eines menügesteuerten Programms ist bei einem Lichtgriffel sehr einfach. Sie zeigen einfach auf das gewünschte Menü, und der Rechner erkennt, welches Menü Sie ausgewählt haben. Mit dem Befehl PEEK kann erkannt werden, auf welche Stelle Sie auf dem Bildschirm gezeigt haben. Dieses ist bei dem Commodore 64 nicht schwierig, da der VIC-Chip bestimmte Speicherplätze für die Programmierung eines Lichtgriffels enthält.

Zu Beginn dieses Buches wurde Ihnen gesagt, daß in Ihrem Computer eine kleine Welt existiere, und daß die Möglichkeiten praktisch unbegrenzt seien. Vielleicht haben Sie sich mit diesem Buch davon überzeugen lassen. Es kann sehr anregend sein, über die Möglichkeiten dieser kleinen Welt, bei der man immer wieder auf Grenzen stößt, nachzudenken und ein kleiner Teil in der Entwicklung des menschlichen Lebens durch den Personal Computer zu sein.

Wir sollten nicht auf unsere Maschinen, sondern auf das, was wir aus ihnen machen können, stolz sein. Erfreuen Sie sich nicht an dem, was Sie haben, sondern an dem, was Sie können. Die wahren verborgenen Schätze un-

seres Lebens liegen in unseren eigenen Fähigkeiten.

```
100 R: GESCHICHTSQUIZ
110 T: WER WAR KEIN PRÄSIDENT?
120 T:
130 T: FORD GRANT EDISON
140 T:
150 T:
160 T:
150 A: $
160 M: EDISON
170 T:
180 TY: DAS IST RICHTIG!
190 TN: NEIN, EDISON WAR EIN ERFINDER.
BEISPIEL EINES PILOT PROGRAMMS
```



Anhang 1: Wir erforschen den Commodore 64

(Jim Butterfield)

In dem Commodore 64 befindet sich eine Welt, die es zu erforschen gilt. Dazu brauchen Sie den Computer nicht zu öffnen. Mit dem Befehl PEEK können Sie sich jeden einzelnen Speicherplatz ansehen. Der Computer muß ebenfalls nicht geöffnet werden, um irgendwelche Verbindungsdrähte zu ziehen. Das ist nämlich mit dem Befehl POKE möglich. Sie können so den Computer durch ein Programm quasi anders verdrahten. Haben Sie keine Angst davor, die Befehle PEEK und POKE zu verwenden. Sie können Ihren Computer dadurch nicht zerstören. Es kann Ihnen nur passieren, daß ein Programm nicht mehr arbeitet und der Computer anhält. Wenn dieses geschieht, müssen Sie den Computer ausschalten und dann erneut einschalten. Machen Sie aber zwischen dem Aus- und Einschalten eine kleine Pause von ca. 5 Sekunden.

Der Speicher des Computers

Der Arbeitsspeicher des Computers ist ein sehr wichtiger Bestandteil des gesamten Systems. Er enthält unter anderem das Programm und die unmittelbar dazu benötigten Daten. Der Speicher besteht aus verschiedenen Speicherzellen, in denen die Information gespeichert werden kann. Jede Speicherzelle hat eine Adresse, mit der die betreffende Zelle identifiziert werden kann. Es gibt insgesamt 65 536 Speicherzellen in Ihrem Commodore 64. Die Adressen beginnen mit 0 und gehen bis 65 535. Jede Speicherzelle kann irgendeinen Wert zwischen 0 und 255 enthalten. Bestimmte Bereiche des

Speichers sind für ganz spezielle Aufgaben reserviert. Nach einiger Zeit werden Sie automatisch wissen, welcher Bereich für welchen Zweck reserviert ist.

Es gibt sozusagen drei verschiedene Arten von Speichern in Ihrem Commodore 64. Wir werden sie nacheinander behandeln.

Verschiedene Arten von Speicher

RAM ist ein Speicher, der geschrieben (POKE) und gelesen (PEEK) werden kann. Wir können in diesem Speicher Informationen ablegen. Später können wir uns dann die Information ansehen, die wir vorher gespeichert haben. Wenn Sie ein Programm in Ihren Commodore 64 eingeben, wird dieses Programm in dem RAM-Speicher gespeichert. Wenn Sie mit Ihrem Programm Daten verarbeiten, werden diese Daten ebenfalls in dem RAM-Speicher gespeichert. RAM steht für Random Access Memory (wahlfreier Speicher). 'Wahlfreier Speicher' heißt, daß Sie direkt auf eine gewünschte Speicherzelle in diesem Speicher zugreifen können.

In der Standardkonfiguration ist der RAM-Speicher von Adresse 2 bis Adresse 40 959 und von Adresse 49 152 bis 59 151 frei verfügbar. Der 'untere' Block wird für BASIC-Programme und als Arbeitsbereich verwendet. Der 'obere' Block wird nicht von BASIC benutzt. Wie wir später noch sehen werden, ist der RAM-Speicher des Commodore 64 noch größer.

ROM ist ein sogenannter 'Festwertspeicher'. Sie können diesen Speicher lesen (PEEK); Sie können aber keine Werte abspeichern. Die in dem ROM-Speicher gespeicherten Daten sind bereits vom Hersteller fest eingegeben worden und Sie können diese Daten nicht ändern. ROM steht für 'Read Only Memory'. Das bedeutet, daß Sie diesen Speicher nur lesen können. Es mag vielleicht als Einschränkung erscheinen, daß Sie die Daten des ROM-Speichers nicht ändern können. Sie können sich aber dadurch behelfen, daß Sie die Daten aus dem ROM lesen, in eine RAM schreiben und dann dort ändern. Der ROM enthält unter anderem die Arbeitsprogramme des Computers, wie z. B. ein Programm zur Berechnung der Quadratwurzel.

Der Commodore 64 besitzt einen ROM, der die Adressen 40 960 bis 49 151 und den Adressenblock von 57 644 bis 65 535 enthält. Der erste Block dient zur Entschlüsselung der BASIC-Befehle. In dem zweiten Block stehen die Ein- und Ausgaberroutinen, die notwendig sind, um beispielsweise die Ein-

gabe von Daten über die Tastatur, die Ausgabe von Daten über den Bildschirm und das Schreiben oder Lesen von Diskette oder Datassette zu bewerkstelligen. Es gibt in dem Commodore 64 einen dritten ROM, den wir normalerweise nicht lesen können. Dieser ROM enthält den Zeichen-Generator, der dazu dient, ein Zeichen auf dem Bildschirm zu erzeugen. Ein ROM ist ebenfalls oft bei einem Video-Spiel, das Sie als Steckmodul an Ihren Commodore 64 anschließen, zu finden.

Die dritte Art von Speichern wird Interface genannt. Ein Interface ist kein wirklicher Speicher, da Sie nicht sicher sein können, daß die Information, die dort gespeichert ist, zu einem späteren Zeitpunkt auch wieder abgerufen werden kann. Die Hauptaufgabe eines Interface ist, die Kommunikation von peripheren Einheiten mit dem Computer zu ermöglichen. So haben wir bereits den Video Interface Chip kennengelernt, der auf dem Bildschirm Bilder erzeugt. Ein weiteres Interface ist der SID (Sound Interface Device). Es gibt weitere Interfaces, die beispielsweise den Anschluß der Joysticks und der Game Paddles ermöglichen.

Der Video Interface Chip hat die Adressen von 53 262 bis 53 286. Das SID hat den Adressenblock 54 272 bis 54 300. Zwei weitere komplexe Interface Adapter, die die Verbindung der Tastatur, der Diskette, der Joysticks usw. vornehmen, haben die Adressenblöcke 56 230 bis 56 335 und 56 576 bis 56 591. Die Speicherplätze 0 und 1 haben eine spezielle Interfacefunktion. Ihre Hauptaufgabe ist, die Arbeitsweise anderer Chips zu kontrollieren.

Memory Maps

'Memory Map' ist ein englischer Begriff, der sehr oft auch in unserer Sprache verwendet wird. Unter einem Memory Map versteht man eine Speicherbelegungstabelle. Sie werden am Ende dieses Kapitels eine Speicherbelegungstabelle finden. Diese Speicherbelegungstabelle enthält einige wichtigen Speicherplätze und deren Funktionen.

Das Arbeiten mit dem RAM-Speicher ist sehr einfach. Man kann Daten in den RAM-Speicher speichern und diese sich wieder ansehen.

Der Inhalt eines ROM-Speichers kann nicht geändert werden. Wenn wir wissen wollen, wie der Computer arbeitet, müssen wir uns den Inhalt des ROM-Speichers ansehen. Um die in dem ROM-Speicher gespeicherten Daten und Befehle zu verstehen, müssen wir uns mit einer neuen Computersprache, der

6502-Maschinensprache beschäftigen:

Der 6502 Prozessor ist dem 6510 Mikropozessor, der mit dem Commodore 64 benutzt wird, sehr ähnlich. Deshalb sind auch die Sprachen sehr ähnlich, in denen diese beiden Mikroprozessoren programmiert werden müssen.

Die Interface-Chips können in der Anwendung sehr praktisch sein. Einige der Speicherplätze, die von diesen Chips benutzt werden, sind in dem 64 User's Guide erwähnt. Sie müssen diese Speicherplätze kennen, wenn Sie beispielsweise mit Ihrem Rechner Musik machen wollen. Eine Speicherzelle ist durch ihre Adresse eindeutig identifizierbar. Diese Adresse setzt sich aus verschiedenen Bits zusammen, die bei Benutzung folgender Zahlen codiert wird:

128 64 32 16 8 4 2 1

Die Inhalte dieser Speicherzellen sind ähnlich codiert. Wenn Sie beispielsweise sehen, daß der Inhalt einer Speicherzelle den Wert 20 enthält, so ist das Bit, das die 16 bezeichnet und das Bit, das die 4 bezeichnet 'eingeschaltet'. Alle anderen Bits sind ausgeschaltet. Wie Sie sich selbst überzeugen können, ist es nicht möglich, durch eine andere Kombination von eingeschalteten Bits die Zahl 20 zu erzeugen.

RAM

Der RAM-Speicher ist so interessant, daß wir uns näher mit ihm beschäftigen wollen. Wir werden uns dazu einige Adressen ansehen und ein paar besonders interessante näher erläutern.

Die Adressen 2 bis 1023 sind der Arbeitsbereich des Computers. Hier finden wir sehr viele interessante Speicherzellen. Der Bereich von 2 bis 255 wird 'Zero Page' genannt und enthält Werte, mit denen der Computer arbeitet. Wenn Sie beispielsweise das Kommando PRINT PEEK(203) eingeben, werden Sie feststellen können, ob eine Taste gerade niedergedrückt ist oder nicht. Erhalten Sie den Wert 64, bedeutet das, daß keine Taste gedrückt ist. Dieser Befehl ist im Direkt-Modus natürlich nicht besonders sinnvoll anzuwenden; denn Sie müssen wirklich sehr schnell sein, um die RETURN-Taste vor Ausgabe der Meldung loszulassen. Innerhalb eines Programms können Sie mit PEEK(203) feststellen, ob irgendeine Taste gedrückt

ist. Mit dem GET-Statement erfahren Sie, um welche Taste es sich handelt. Die Speicheradressen 256 bis 511 sind für den internen Gebrauch bestimmt. Diese Adressen sind der 'Stack' des Rechners. Hier werden Sie kaum nützliche Informationen finden.

Die Plätze 512 bis 677 werden sehr häufig benutzt. Mit dem Speicherplatz 650 wird die Wiederholungsfunktion einiger Tasten gesteuert. Sie werden sicherlich schon festgestellt haben, daß die Leertaste mit einer Wiederholungsfunktion ausgestattet ist. Halten Sie diese Taste längere Zeit niedergedrückt. Die Funktion der Taste wird nun automatisch wiederholt. Speichern Sie jetzt auf den Speicherplatz 650 nacheinander die Werte 255 und 55. Schauen Sie sich das Ergebnis an. Speichern Sie nun den Wert 126, und schließlich den Wert 0 ab. Sehen Sie sich auch jetzt das Resultat an.

Die Speicherplätze 768 bis 819 haben eine spezielle Funktion. Um die Funktionen dieser Speicherplätze zu verstehen, muß man allerdings mit der Programmierung in Maschinensprache sehr vertraut sein.

Die Speicherplätze 828 bis 1019 sind der Datensettenpuffer. Er wird nur dann benutzt, wenn Daten auf die Datensette gespeichert oder Daten von der Datensette gelesen werden sollen. Ansonsten ist dieser Speicherbereich frei und kann z. B. für Sprites verwendet werden.

Die Speicherplätze 1024 bis 2047 stellen den Bildschirmspeicher dar. Wenn Sie in diesen Speicherbereich einen Wert schreiben, wird auf dem Bildschirm ein Zeichen erscheinen. Probieren Sie es einfach mal aus, indem Sie mit dem POKE-Befehl irgendeinen Wert zwischen 0 und 255 auf einen dieser Speicherplätze schreiben. Dieser Bereich wird sehr gern für Spielprogramme ausgenutzt.

In den Speicherzellen 2048 bis 40959 wird das BASIC-Programm gespeichert, das Sie eingeben. Wenn das Programm ausgeführt wird, werden die Werte der Variablen und Felder ebenfalls in diesem Bereich gespeichert.

Der Bereich 2048 bis 40959 ist ein großer Bereich. Er wird nicht vollständig gebraucht, um das BASIC-Programm aufzunehmen. Sie können diesen Bereich mit ein wenig Geschick für andere Dinge (wie z. B. Sprites) verwenden. Da wir hier nicht zu sehr in Details gehen können, sei nur erwähnt, daß in den Speicherzellen 43 und 44 der Start des BASIC-Programms und in den Speicherplätzen 55 und 56 die Grenze des BASIC-Programms gespeichert ist. Wie können Sie eine Adresse lesen, die in zwei Bytes gespeichert ist? Nun, Sie schauen sich mit PEEK das erste Byte an, schauen

sich dann mit PEEK das zweite Byte an, multiplizieren das zweite Byte mit 256 und addieren diesen Wert zu dem ersten Byte.

Es gibt noch einen großen freien Speicherbereich von 49 152 bis 53 247, der normalerweise nicht benutzt wird. In diesem Bereich können Sie Ihr Maschinenprogramm ausprobieren und experimentieren.

Die Verschiebung von Speicherplätzen

Es sieht so aus, als hätten alle Daten in dem Speicher ihren Platz. Aber - der Commodore 64 hat ein Geheimnis. Hinweis: Sie können es schon am Namen des Computers erkennen.

Der Commodore 64 hat 'überall' einen RAM-Speicher, 64K. Das bedeutet, daß 65 536 Speicherplätze vorhanden sind. Auch dort, wo wir einen ROM haben, versteckt sich ein RAM.

Wenn wir den Befehl PRINT PEEK(45000) eingeben, erhalten wir den Inhalt des ROMs. Wenn wir aber POKE 45000,100 schreiben, weiß der Computer, daß er diesen Wert nicht in den ROM speichern darf. Deshalb wird der Wert 100 in den RAM-Speicher hinter dem ROM-Speicher geschrieben. Können Sie diesen RAM-Speicher lesen? Natürlich nicht, denn der ROM-Speicher ist Ihnen im Weg. PEEK(45000) wird Ihnen den Inhalt des ROM-Speichers zeigen. Sie haben aber die Möglichkeit, den ROM-Speicher 'wegzuwischen', indem Sie die Controls in Adresse 1 benutzen.

Warten Sie einen Augenblick damit: wenn Sie den ROM-Speicher eliminieren, verlieren Sie auch BASIC. Das ist nicht vorteilhaft, da Sie ja die Ihnen zu Verfügung stehenden BASIC-Befehle ausnutzen wollen; auch POKE ist nämlich ein BASIC-Befehl. Gibt es irgendeinen Weg, den ROM-Speicher auszuschalten und trotzdem BASIC zu behalten?

Die Lösung ist ganz einfach. Kopieren Sie nur den ROM-Speicher in den RAM-Bereich. Sie können dann den ROM-Speicher verschwinden lassen, da BASIC noch im RAM-Speicher vorhanden ist. Damit können wir die BASIC-Befehle ändern.

Wenn Sie BASIC von dem ROM- in den RAM-Speicher kopieren wollen, müssen Sie folgende Befehle eingeben:

```
FOR J=40960 TO 49151: POKE J, PEEK (J): NEXT J
```

Auf den ersten Blick scheint das Unsinn zu sein, denn wir schreiben mit POKE den gleichen Wert in jede Speicherzelle, die wir vorher mit dem PEEK Kommando gelesen haben. Jetzt verstehen wir, daß wir den ROM lesen und den Wert dann in den RAM schreiben. Es dauert etwas, bis das BASIC vom ROM in den RAM kopiert ist, da der Rechner hier doch einiges tun muß. Wenn der Kopiervorgang beendet ist, geben Sie

POKE 1,54

ein, und BASIC wird nun aus dem RAM-Bereich des Speichers ausgeführt. Versuchen Sie nun folgendes:

POKE 41848,66

Der Computer wird nun nicht mehr mit READY, sondern mit BEADY antworten. Damit ist das BASIC verändert. Sie können aber wieder zu dem BASIC umschalten, welches im ROM-Speicher steht. Geben Sie dazu einfach POKE 1,55 ein. Aus der Meldung BEADY wird wieder READY.

Zusammenfassung

In der nachstehenden Tabelle sehen Sie die Speicherbelegung des Commodore 64. Die Adresse des Speicherplatzes ist hexadezimal und dezimal angegeben. In der nächsten Spalte folgt eine kurze Beschreibung der Bedeutung der einzelnen Speicherplätze.

Speicherbelegung des Commodore 64, zusammengestellt von Jim Butterfield

Hexadezimal	Dezimal	Beschreibung
0000	0	6510 Daten Richtungsregister
0001	1	6510 Ausgaberegister
0002	2	nicht benutzt
0003-0004	3-4	Vektor zur Umrechnung Float - Fixed
0005-0006	5-6	Vektor zur Umrechnung Fixed - Float
0007	7	Suchzeichen

Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
0008	8	Flag für Gänsefüßchen Modus
0009	9	TAB Spaltenzähler
000A	10	0 = LOAD, 1 = VERIFY
000B	11	Zeiger für Eingabepuffer / Anzahl Elemente
000C	12	Flag für Standard DIM
000D	13	Typ:FF = String, 00 = numerisch
000E	14	Typ:80 = Integer, 00 = Fließkomma
000F	15	Flag für DATA/LIST
0010	16	Element/FNx Flag
0011	17	00 = INPUT, 40 = GET, 98 = READ
0012	18	Vorzeichen des ATN Flag für Gleichheit bei Vergleich
0013	19	aktuelles E/A Gerät
0014-0015	20-21	Integer Wert
0016	22	Zeiger auf temporären Stringstapel
0017-0018	23-24	Letzter temporärer String Vektor
0019-0021	25-33	Stapel für temporäre Strings
0022-0025	34-37	Bereich für Hilfszeiger
0026-002A	38-42	Bereich für Produkt bei Multiplikation
002B-002C	43-44	Zeiger auf Basic Anfang
002D-002E	45-46	Zeiger auf Variablen Anfang
002F-0030	47-48	Zeiger auf Beginn der Arrays
0031-0032	49-50	Zeiger auf Ende der Arrays
0033-0034	51-52	Zeiger auf Stringspeicher (bewegt sich abwärts)
0035-0036	53-54	Hilfszeiger für Strings
0037-0038	55-56	Zeiger auf Speichergrenze
0039-003A	57-58	Nummer der aktuellen Basic Zeile
003B-003C	59-60	Nummer der vorhergehenden Basic Zeile
003D-003E	61-62	Zeiger auf Basic Statement für CONT
003F-0040	63-64	Nummer der aktuellen DATA Zeile
0041-0042	65-66	Adresse des aktuellen DATA Elementes

Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
0043-0044	67-68	Sprungvektor für Input
0045-0046	69-70	aktueller Variablenname
0047-0048	71-72	Adresse der aktuellen Variablen
0049-004A	73-74	Variablenzeiger für FOR...NEXT
004B-004C	75-76	Zwischenspeicher für BASIC Zeiger
004D	77	Akkumulator für Vergleichssymbole
004E-0053	78-83	verschieden genutzter Arbeitsbereich (Zeiger, usw.)
0054-0056	84-86	Sprungvektor für Funktionen
0057-0060	87-96	verschieden genutzter Bereich für numerische Operationen
0061	97	Fließkomma Akkumulator #1 (FAC): Exponent
0062-0065	98-101	Fließkomma Akkumulator #1 (FAC): Mantisse
0066	102	Fließkomma Akkumulator #1 (FAC): Vorzeichen
0067	103	Zeiger für Polynom Auswertung
0069-006E	105-110	Fließkomma Akkumulator #2 Exponent usw.
006F	111	Vorzeichen Vergleich Akku #1 / Akku #2
0070	112	niederwertige Stelle Akku #1 (Rundung)
0071-0072	113-114	Länge des Kassettenpuffers
0073-008A	115-138	CHRGET Subroutine (hole ein Zeichen)
007A-007B	112-123	Basic Zeiger innerhalb der Subroutine
008B-008F	139-143	Startwert des Zufallgenerators
0090	144	Statusbyte ST
0091	145	Flag für STOP und RVS Taste
0092	146	Zeitkonstante für Kassette
0093	147	0 = LOAD, 1 = VERIFY
0094	148	serieller Ausgang: Flag für zurückge- stelltes Zeichen

 Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
0095	149	zurückgestelltes Zeichen
0096	150	EOT von Kassette empfangen (Cassette Sync #)
0097	151	Speicher für Register
0098	152	Anzahl offener Files
0099	153	Eingabegerät (normal = 0)
009A	154	Ausgabe (CMD) Gerät (normal =3)
009B	155	Paritätsbyte von Band
009C	156	Flag für Byte empfangen
009D	157	Ausgabe Kontrolle (80 = direkt, 00 = RUN)
009E	158	Fehler vom Band / Zeichenpuffer
009F	159	Fehler vom Band korrigiert
00A0-00A2	160-162	interne Uhr (HML)
00A3	163	serieller Bit Zähler Flag für EOI
00A4	164	Zyklen Zähler
00A5	165	Abwärtszähler beim Schreiben auf Kassette
00A6	166	Zeiger auf Kassettenpuffer
00A7-00AB	167-171	Flags für Schreiben und Lesen von Kassette
00AC-00AD	172-173	Zeiger für Prg.Start
00AE-00AF	174-175	Zeiger für Programmende
00B0-00B1	176-177	Zeitkonstanten für Band
00B2-00B3	178-179	Zeiger auf Anfang des Kassettenpuffers
00B4	180	Band Timer (1 = gesetzt), Bit Zähler
00B5	181	Band EOT/ RS 232 nächstes Bit zum Senden
00B6	182	Lesefehler
00B7	183	Anzahl Zeichen im Filenamen
00B8	184	aktuelle logische Filenummer
00B9	185	aktuelle Sekundäradresse
00BA	186	aktuelles Gerät
00BB-00BC	187-188	Zeiger auf Filenamen
00BD	189	Schreiben/Schieberegister
00BE	190	Anzahl zum Lesen/Schreiben

Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
		verbleibender Blocks
00BF	191	serieller Wortpuffer
00C0	192	Kassettenmotor Flag
00C1-00C2	193-194	E/A Startadresse
00C3-00C4	195-196	Zeiger auf Vektoradressen des KERNAL
00C6	198	Anzahl Zeichen im Tastaturpuffer
00C7	199	RVS Flag für Bildschirm
00C8	200	Zeiger auf Zeilenende für Eingabe
00C9-00CA	201-202	Position des Eingabecursors (Zeile, Spalte)
00CB	203	gedrückte Taste (64=keine Taste)
00CC	204	Cursor an/aus (0 = Cursor blinkt)
00CD	205	Zähler für blinkenden Cursor
00CE	206	Zeichen in Cursorposition
00CF	207	Cursor in Blinkphase
00D0	208	Eingabe von Bildschirm/Tastatur
00D1-00D2	209-210	Zeiger auf Bildschirmzeile
00D3	211	Zeiger auf Bildschirmspalte
00D4	212	0=direkter Cursor, sonst programmiert (QUOTE MODE)
00D5	213	Länge der aktuellen Bildschirmzeile (40/80)
00D6	214	Zeile, in der sich der Cursor befindet
00D7	215	Letzte Taste/Prüfsumme/Puffer
00D8	216	Anzahl ausstehender inserts
00D9-00F0	217-240	Bildschirmzeilen Verknüpfungstabelle
00F1	241	unechte Bildschirmzeile
00F2	242	Bildschirmzeilen Marke
00F3-00F4	242-244	Zeiger auf Bildschirm Farbe
00F5-00F6	245-246	Zeiger auf Tastatur Decodiertabelle
00F7-00F8	247-248	RS232 Empfangszeiger
00F9-00FA	249-250	RS232 Übertragungszeiger
00FB-00FE	251-254	Freier Platz in Page 0 für Betriebssystem
00FF	255	Basic Speicher
0100-010A	256-266	Arbeitsbereich zur Umwandlung von Fließ-

Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
		komma in ASCII
0100-013E	256-318	Bandfehler
0100-01FF	256-511	Bereich des Prozessor Stapels
0200-0258	512-600	Basic Eingabepuffer
0259-0262	601-610	Tabelle der logischen Files
0263-026C	611-620	Tabelle der Gerätenummern
026D-0276	621-630	Tabelle der Sekundäradressen
0277-0280	631-640	Tastaturpuffer
0281-0282	641-642	Startadresse des RAM für Betriebssystem
0283-0284	642-644	Ende des RAM für Betriebssystem
0285	645	Flag für Zeitüberschreitung auf seriellen Bus
0286	646	aktueller Farbcode
0287	647	Farbe unter Cursor
0288	648	Bildschirmspeicher Adresse (Page)
0289	649	maximale Größe des Tastaturpuffers
028A	650	Tastenwiederholung (128=alle Tasten)
028B	651	Zähler für Wiederholungsgeschwindigkeit
028C	652	Zähler für Wiederholungsverzögerung
028D	653	Flag für SHIFT/CNTRL
028E	654	letztes SHIFT Muster der Tastatur
028F-0290	655-656	Zeiger auf Tastatur Decodiertabelle
0291	657	SHIFT Modus (0=gesetzt, 128=blockiert)
0292	658	automat. Scrolling abwärts (0=ein, #0=aus)
0293	659	RS232 Kontrollregister
0294	660	RS232 Befehlsregister
0285-0296	661-662	nicht Standard (Bit Zeit)
0297	663	RS232 Statusregister
0298	664	Anzahl zu sendender Bits
0299-029A	665-666	Baud Rate
029B	667	RS232 Empfangszeiger
029C	668	RS232 Eingabezeiger

} receive

Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
029D	669	RS232 Übertragungszeiger
029E	670	RS232 Ausgabezeiger
029F-02A0	271-672	enthält IRQ-Vektor während Kassettenbetrieb
02A1-02FF	673-767	Timer/Interrupt
0300-0301	768-769	Vektor für Fehlermeldungen
0302-0303	770-771	Vektor für Basic Warmstart
0304-0305	772-773	Umwandlung von Schlüsselwörtern in Tokens
0306-0307	774-775	Umwandlung der Tokens in Schlüsselwörter
0308-0309	776-777	neuen Basic Befehl ausführen
030A-030B	778-779	arithmetisches Element holen
030C	780	Speicher für 6502 „A Register
030D	781	Speicher für 6502 „X Register
030E	782	Speicher für 6502 „Y Register
030F	783	Speicher für 6502 „P Register
0310-0313	784-787	USR Sprung
0314-0315	788-789	Hardware Interrupt (IRQ) (EA31)
0316-0317	790-791	Break Interrupt (FE66)
0318-0319	792-793	nicht maskierbarer Interrupt (NMI)(FE47)
031A-031B	794-795	OPEN (F40A) (F34A)
031C-031D	796-797	CLOSE(F291)
031E-031F	798-799	Kanal für Eingabe (F2C7) (F209)
0320-0321	800-801	Kanal für Ausgabe (F250)
0322-0323	802-803	Wiederherstellung der E/A (Löschen aller offenen Kanäle)(F333)
0324-0325	804-805	INPUT(F157)
0326-0327	806-807	OUTPUT(F1CA)
0328-0329	808-809	STOP-Taste prüfen (F770)(F6ED)
032A-032B	810-811	GET(F13E)
032C-032D	812-813	Close aller Kanäle (F32F)
032E-032F	814-815	Benutzer-IRQ (FE66)
0330-0331	816-817	RAM laden (F4A5)
0332-0333	818-819	RAM speichern (F5ED)

Speicherbelegung des Commodore 64

Hexadezimal	Dezimal	Beschreibung
033C-03FB	828-1019	Kassettenpuffer
0400-07FF	1024-2047	IK Bildschirmspeicher
(0400-07E7	1024-2023	Video Matrix)
(07F8-07FF	2040-2047	Zeiger für Sprites)
0800-9FFF	2048-40959	Basic Benutzerspeicher
A000-BFFF	40960-49151	8K Basic ROM
C000-CFFF	49152-53247	4K RAM
D000-D02E	53248-53294	Video Chip (6566)
D400-D41C	54272-54300	Tongenerator Chip (6581 SID)
D800-DBFF	55296-56319	Farbspeicher
DC00-DC0F	56320-56335	Interface Chip 1, IRQ (6526 CIA)
DD00-DD0F	56576-56591	Interface Chip 2, NMI (6526 CIA)
D000-DFFF	53248-53294	Alternativer Zeichensatz
E000-FFFF	57344-65535	ROM: Betriebssystem
E000-FFFF	57344-65535	Alternativer RAM
FF81-FFFF	65409-65525	Sprungtabelle

Processor I/O Port (6510)

\$0000	IN	IN	OUT	IN	OUT	OUT	OUT	OUT	DDR 0
\$0001			Tape Motor	Tape Sense	Tape Write	D-ROM Switch	EF RAM Switch	AB RAM Switch	PR 1

SID (6581)

SID (0381)							Stimme 1	Stimme 2	Stimme 3	
Stimme 1	Stimme 2	Stimme 3								
\$D400	\$D407	\$D40E	Frequenz				L	54272	54279	54286
\$D401	\$D408	\$D40F					H	54273	54280	54287
\$D402	\$D409	\$D410	Tastverhältnis				L	54274	54281	54288
\$D403	\$D40A	\$D411					H	54275	54282	54289
\$D404	\$D40B	\$D412	Wellenform: Rauschen/Rechteck/Sägezahn/Dreieck				Key	54276	54283	54290
\$D405	\$D40C	\$D413	Anschlag 2ms - 8ms		Abschwellen 6ms - 24 sec			54277	54284	54291
\$D406	\$D40D	\$D414	Halten		Ausklängen 6ms 24 sec			54278	54285	54292
Stimmen										

\$D415	0	0	0	0	0	L	54293
\$D416	Filterfrequenz					H	54294
\$D417	Resonanz		Filterstimmen				54295
			Ext	V3	V2	V1	
\$D418	Filtermodus		Lautstärke				54296
	aus Hochpaß Bandpaß Tiefpaß		Volume				
	Filter + Volumen						

Paddle X (A/D #1)
Paddle Y (A/D #2)
Oszillator 3
Hüllkurve 3

CIA (IRQ) (6526)

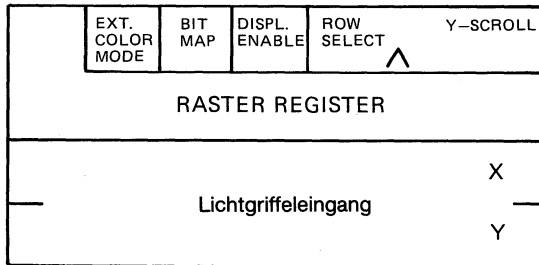
\$DC00	Paddle Sell		Joystick 0					PRA	56320
	A	B	Feuer	rechts	links	unten	oben		
\$DC01	Keyboard Row Select (inverted)							PRB	56321
			Joystick 1						
			Fire	Right	Left	Down	Up		
\$DC02	Keyboard Column Read							DDRA	56322
\$DC03	\$FF-All Output								
\$DC04	\$00-All Input							DDRB	65323
\$DC05	Timer A								
\$DC06	Timer B							TAL	56324
\$DC07									
	~							TAH	56325
	~							TBL	56326
								TBH	56327
\$DC0D			Tape Input			Timer Interrupt		ICR	56333
\$DC0E						B	A		
\$DC0F				One Shot	Out Mode	Time PB6 Out	Timer A Start	CRA	56334
				One Shot	Out Mode	Time PB7 Out	Timer B Start		
								CRB	56335

CIA 2 (NMI) (6526)

\$DD00	Serial IN	Clock IN	Serial OUT	Clock OUT	ATN OUT	RS-232 OUT	VIC II addr 15	VIC II addr 14	PRA	56576
\$DD01	DSR IN	CTS IN		DCD ^o IN	RI ^o IN	DTR OUT	RTS OUT	RS-232 IN	PRB	56577
\$DD02	\$3F-Serial								DDRA	56578
\$DD03	\$00-P.U.P. All Input				or	\$06-RS-232			DDRB	56579
\$DD04	Timer A								TAL	56580
\$DD05									TAH	56581
\$DD06	Timer B								TBL	56582
\$DD07									TBH	56583
~										
\$DD0D				RS-232 IN			Timer Interrupt B A		ICR	56589
\$DD0E							Timer A Start		CRA	56590
\$DD0F							Timer B Start		CRB	56591

Summary

VIDEO (6566)



53265

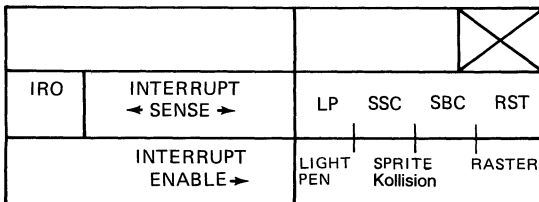
53266

53267

53268



53270

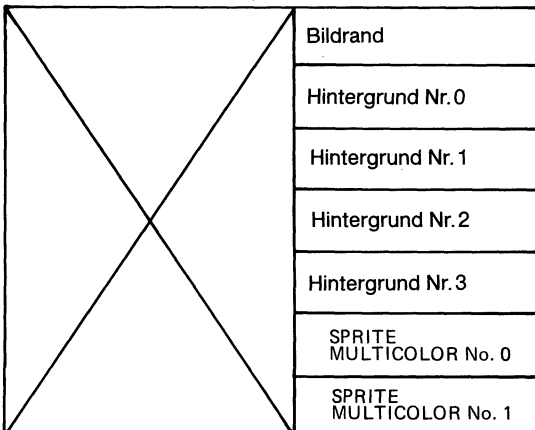


53272

53273

53274

COLOR REGISTER



53280

53281

53282

53283

53284

53285

53286

SPRITE VIDEO		SPRITE	SPRITE
POSITION X Y		0 7	
		↓	↓
		53248 . . . 53262	
		53249 . . . 53263	
COLOR		53287 . . . 53294	
SPRITE No. 7 6 5 4 3 2 1 0			
X-POSITION HIGH		53264	
SPRITE ENABLE		53269	
Y-EXPAND		53271	
BACKGROUND PRIORITY		53275	
MULTICOLOR		53276	
X-EXPAND		53277	
INTERRUPT: SPRITE KOLLISION		53278	
INTERRUPT: HINTERGRUND KOLLISION		53279	

Anhang 2: Grafik mit dem Commodore 64

(Paul F. Schatz)

Der Commodore 64 bietet sehr gute Möglichkeiten zur Programmierung von grafischen Darstellungen: hochauflösende ('bit map') Grafik, mehrere Farbgrafikmöglichkeiten, definierbare Zeichen und bewegbare Farbobjekte (d. h. Sprites). Die Programmierung einer grafischen Darstellung kann ein sehr komplexer Vorgang sein. Das wird besonders dadurch erschwert, daß Sie die grafischen Darstellungen nicht in BASIC programmieren können. Die Programmierung von grafischen Darstellungen auf dem Commodore 64 benötigt nämlich ein recht tiefes Verständnis dafür, wie der Computer arbeitet. Dieses Kapitel wird in leicht verständlicher Form erklären, wie Zahlen in dem Speicher des Computers dargestellt werden und wie sie mit einem Bild auf dem Schirm in Beziehung zu setzen sind. Sie werden etwas über drei verschiedene Grafikmöglichkeiten (Sprites), programmierbare Zeichen und Bit Map Grafik erfahren. Jede dieser drei Möglichkeiten wird an einfachen Beispielprogrammen demonstriert. Auf diese Programme aufbauend können Sie kompliziertere Grafiken darstellen.

Wie werden Zahlen gespeichert?

Wenn Sie wissen, wie Zahlen benutzt werden, um eine grafische Darstellung zu definieren, haben Sie damit den Schlüssel zu einem richtigen Verständnis für Grafik. Zunächst müssen Sie verstehen, was mit dem Begriff 'Byte' gemeint ist. Wenn Sie den Computer einschalten, werden Sie in der ersten

Zeile folgende Meldung sehen können:

64K RAM SYSTEM 38911 BYTES FREE

BYTES FREE bezieht sich dabei auf die Anzahl von Speicherplätzen, die auch Register genannt werden. Auf diesen Speicherplätzen wird das BASIC-Programm abgespeichert. Sie haben ja bereits gelernt, daß in einem Byte ein Zeichen gespeichert werden kann. Diese Zeichen werden durch die Zahlen 0 bis 255 dargestellt. Warum nur 256 Zeichen? Warum nicht mehr, bzw. weniger? Die Antwort ist in der Art und Weise der Darstellung von Zahlen im Computer zu finden. Jeder Speicherplatz kann als ein Satz von 8 Schaltern verstanden werden, die von 0 bis 7 numeriert sind. Jeder Schalter kann ein- oder ausgeschaltet sein. Da jeder Schalter einer Zahl zugeordnet ist (siehe Abb. 1), kann jede Zahl von 0 bis 255 durch Addition der eingeschalteten Schalter dargestellt werden.

Schalter	7	6	5	4	3	2	1	0
Zahl	128	64	32	16	8	4	2	1

Ob ein Schalter ein- oder ausgeschaltet ist, kann durch 1 und 0 dargestellt werden. Eine 1 bedeutet 'eingeschaltet', eine 0 'ausgeschaltet'. Eine Folge dieser beiden Ziffern wird als 'Binärzahl' bezeichnet. Sie kann anstelle der üblichen Dezimalzahl benutzt werden. Die Zahlen, die uns im täglichen Leben begegnen, werden Dezimalzahlen genannt, da sie 10 verschiedene Ziffern kennen: 0 bis 9. Diese Schalter werden im Computerjargon 'Bits' genannt. Ein Byte ist eine Serie von 8 Bits. Die Zahl 16 wird beispielsweise dadurch dargestellt, daß der Schalter mit der Nummer vier eingeschaltet (1) ist und alle anderen Schalter ausgeschaltet (0) sind.

Schalter	7	6	5	4	3	2	1	0
an/aus	0	0	0	1	0	0	0	0
Zahl	128	64	32	16	8	4	2	1

Um die Zahl 129 darzustellen, müssen wir den Schalter mit der Nummer 1 und der Nummer 7 einschalten. Alle anderen Schalter müssen ausgeschaltet sein.

Schalter	7	6	5	4	3	2	1	0
an/aus	1	0	0	0	0	0	0	1
Zahl	128	64	32	16	8	4	2	1

Die größte durch ein Byte darstellbare Dezimalzahl ist 255. In diesem Fall sind alle 8 Bits eingeschaltet. Rechnen Sie selbst einmal nach! Damit Sie schnell bei einer gegebenen Dezimalzahl die entsprechenden Schalterstellungen ermitteln können, müssen Sie Tabelle 1 benutzen.

In der Computergrafik wird ein Byte zur Definition einer Zeile von Punkten oder eines grafischen Symbols auf dem Bildschirm verwendet. Diese 'Bilder' werden Pixels genannt. Das linke Pixel in einem Satz von 8 Pixeln korrespondiert mit Bit Nummer 7 eines Byte. Wenn ein Bit in einem Byte eingeschaltet ist, wird das Pixel ebenfalls eingeschaltet. Entsprechendes gilt, wenn das Bit ausgeschaltet ist.

Sprites, programmierbare Grafiksymbole und Bitmapgrafik ähneln sich in einem Punkt: Die grafische Darstellung wird durch Zeilen von 8 Pixeln erzeugt. Diese Pixel sind durch den Inhalt eines Speicherplatzes definiert. Unterschiede bestehen in der Anzahl der Speicherplätze, die für die Darstellung der Grafik benutzt werden.

Tabelle 1 Binär-Dezimal Konversion

Bit 0–3 entspricht der Spaltenüberschrift, Bit 4–7 der Zeilenüberschrift
 Beispiel: Die Dezimalzahl wird im Dualsystem dargestellt als: 00010010 (zweite Zeile, 3. Spalte)

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0001	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0010	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
0011	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0100	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
0101	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
0110	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
0111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
1000	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
1001	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
1010	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
1011	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
1100	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
1101	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
1110	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
1111	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Der VIC Chip

Die Grafik wird durch den Inhalt der Speicherplätze dargestellt und durch den Video Interface Controller Chip (VIC Chip) erzeugt. Dieser VIC Chip ist ein sehr leistungsfähiger Mikrochip, der das Video-Signal erzeugt. Prinzipiell läßt sich das, was der VIC-Chip tut, auch durch ein Programm bewerkstelligen. Viele Computer arbeiten auf diese Art und Weise. Das würde aber bedeuten, daß der Computer langsamer arbeitet, als es bei einer Durchführung dieser Aufgabe durch einen Chip der Fall ist. Wie der VIC-Chip die grafische Darstellung auf dem Bildschirm ausgibt, ist mit dem Inhalt der 47 Speicherplätze 53248-53294 bestimmt.

Einige dieser Speicherplätze sind Gruppen von Ein- und Ausschalter. So kann z. B. der Speicherplatz 53269 als eine Summe von 8 Schaltern verstanden werden, die individuelle Sprites ein- und ausschalten. Es gibt weitere Schalter, mit denen die einzelnen Grafikmöglichkeiten angewählt werden können (wie Grafiksymbole, Bit Map und Farbgrafik). Andere Speicherplätze bestimmen, wo die Information für die grafische Darstellung in dem Speicher des Computers abgelegt ist.

Bevor Sie weiterlesen, sollten Sie wissen, daß der VIC-Chip einige Beschränkungen aufweist. Die wichtigste davon ist, daß der Speicher insgesamt nur 16 384 (16K) Speicherzellen umfaßt. Erwähnt werden sollen auch die Speicherplätze 0 bis 827, die von BASIC und dem Betriebssystem zur Speicherung von Zeigern (Pointer) und anderen Zahlen benutzt werden. Ein weiterer potentieller Konfliktpunkt ist der Bereich des Speichers, in dem ein BASIC Programm gespeichert wird. Der Computer speichert ein BASIC Programm von Speicherstelle 2048 an aufwärts. Viele Programme sind kurz genug, so daß am Ende des ersten 16K Blocks noch Platz zur Abspeicherung der Video-Information bleibt. Ein langes Programm kann dennoch den kompletten Speicherbereich von 2048 bis 16383 einnehmen. In diesem Fall ist es möglich, dem VIC mitzuteilen, daß er seine Video Information von einem anderen 16K Block des Speichers holen soll. Da die vorliegenden Ausführungen nur eine erste Einführung sein sollen, werden wir auf diese Problematik nicht näher eingehen können.

Wie programmiert man den VIC?

Der VIC kann mit den Befehlen PEEK, POKE, AND, OR und NOT programmiert werden. Da Sie wahrscheinlich noch nicht sehr lange programmieren, sind Ihnen diese Befehle sicherlich noch nicht ganz vertraut.

Mit PEEK können Sie sich den Inhalt eines Speicherplatzes ansehen. Geben Sie folgenden Befehl ein:

```
PRINT PEEK(53272)
```

Der Computer antwortet, indem er die Zahl 21 auf dem Bildschirm ausgibt. Der Inhalt der Speicherzelle 53272 enthält die Stellen, wo sich der Zeichensatz für die Darstellung auf dem Bildschirm befindet. Schalten Sie einen anderen Zeichensatz ein, indem Sie die COMMODORE- und die SHIFT-Taste gleichzeitig drücken. Schauen Sie sich jetzt wieder die Speicherstelle 53272 an. Sie werden nun feststellen, daß sich auf diesem Speicherplatz eine andere Zahl befindet. Der Computer holt sich die Informationen für diesen Zeichensatz von einem anderen Teil des Speichers.

Mit POKE können Sie Zahlen auf bestimmte Speicherplätze schreiben. Sie können also den Zeichensatz auch durch Verwendung des POKE-Kommandos ändern.

```
POKE 53272,21
```

Welcher Zeichensatz wird nun von dem Computer verwendet? Ändern Sie wiederum den Zeichensatz, indem Sie

```
POKE 53272,23
```

eingeben. Nur ganze Zahlen zwischen 0 und 255 können mit POKE verwendet werden. Andere Zahlen erzeugen die Fehlermeldung ILLEGAL QUANTITY ERROR.

Die Wörter AND, OR und NOT werden auch logische Operatoren genannt. Sie werden in der Grafik dazu benutzt, um spezielle Schalter oder Bits ein- und auszuschalten, ohne die anderen Bits zu verändern. Wir sehen uns einmal an, wie sie arbeiten.

OR wird verwendet, um Bits zu setzen. Wenn wir beispielsweise das Bit

1 des Speicherplatzes 53272 einschalten wollen, ohne dabei die anderen Bits zu verändern, benutzen wir folgendes Kommando:

POKE 53272, PEEK(53272) OR 2

Mit PEEK bekommen wir das Zeichen, welches in der Speicherzelle 53272 abgespeichert ist. Die Zahl, die auf OR folgt, bestimmt, welche Bits gesetzt werden sollen. OR 2 setzt das Bit 1 in der Binärzahl auf 1. Mit POKE wird die auf diese Weise geänderte Zahl wieder zurück auf die Speicherzelle 53272 gespeichert.

Schalter	7	6	5	4	3	2	1	0
an/aus	0	0	0	0	0	0	1	0
Zahl	128	64	32	16	8	4	2	1

Diesem Beispiel können Sie entnehmen, wie die Zahl 2 dargestellt wird. Der einzige Schalter, der eingeschaltet ist, ist der Schalter mit der Nummer 1. Alle anderen Schalter sind ausgeschaltet. OR 129 würde die beiden Bits 7 und 0 auf 1 setzen. Die Tabelle kann Ihnen helfen, die richtigen Bits zu finden.

Schalter	7	6	5	4	3	2	1	0
an/aus	1	0	0	0	0	0	0	1
Zahl	128	64	32	16	8	4	2	1

Bits können gelöscht werden, indem das Kommando AND und NOT benutzt wird. Wenn Sie das Bit 1 des Inhalts der Speicherzelle 53272 löschen wollen, müssen Sie folgenden Befehl eingeben:

POKE 53272, PEEK(53272) AND NOT 2

Mit dem PEEK Kommando wird der Inhalt des Speicherplatzes gelesen. Bit 1 wird mit dem AND NOT 2 Kommando gelöscht, und das Ergebnis wird wieder in Speicherzelle 53272 zurückgeschrieben.

5. Definieren Sie die Farbe des Sprites.
6. Sagen Sie dem Computer, wo Ihr Sprite Bild auf dem Bildschirm erscheinen soll.
7. Schalten Sie das Sprite ein.

Wie können Sie die Zahlen ermitteln, mit denen Sie ein Sprite darstellen können? Nun, das ist einfacher, als Sie denken. Sie benötigen zunächst ein paar kleine Hilfsmittel: ein Blatt kariertes Papier, einen Bleistift, ein Radiergummi - ein GROßES Radiergummi. Sie sollten dann auf dem Papier 24 Karos waagerecht und 21 Karos senkrecht umrahmen. Diese 504 Karos repräsentieren die 504 Pixels (63 Bytes), mit denen das Sprite Bild dargestellt wird.

Entwerfen Sie nun Ihr Sprite Bild, indem Sie die einzelnen Karos des karierten Blattes ausfüllen. Sie werden Ihr Bild sicherlich sehr oft verändern und verbessern wollen. Wenn Sie mit dem Sprite Bild zufrieden sind, müssen Sie die einzelnen Zahlen ermitteln. Dazu müssen Sie das Ihnen vorliegende Bild in drei senkrechte Spalten von je 8 Karos unterteilen. Diese Spalten korrespondieren mit den Spalten von Bytes in Abbildung 2. Die entsprechenden Binärzahlen jedes dieser 8 Pixels werden ermittelt, indem Sie eine 1 für ausgefülltes Karo und eine 0 für ein leeres Karo setzen. Sie können die entsprechenden Dezimalzahlen in Tabelle 1 finden.

Ein kleines Beispiel sehen Sie nachstehend.

0, 0, 0			
0, 0, 0			
0, 0, 12			**.
0, 0, 62			*****.
0, 0, 126			*****.
0, 0, 255			*****.
0, 1, 251		*	*****.
0, 127, 241		*****	*****.
3, 255, 240	**	*****	*****.
3, 255, 240	**	*****	*****.
4, 255, 240	*	*****	*****.
0, 255, 240		*****	*****.
0, 224, 60		***	*****.
1, 96, 20	*	**	*.*.
2, 192, 10	*	**	*.*.
2, 128, 10	*	*	*.*.
2, 128, 4	*	*	*.*.
12, 128, 2	**.	*	*.*.
0, 128, 0		*	*.*.
0, 192, 0		**	*.*.
0, 64, 0		*	*.*.

Abb. 3: Pferd Sprite

Dieses pferdähnliche Sprite zeigt den Umriß eines Pferdes. Ein Stern bedeutet dabei 'Pixel eingeschaltet', ein Punkt bedeutet 'Pixel ausgeschaltet'. Sie können die Umrisse des Pferdes leicht erkennen. Wenn Sie sich dieses Bild genau ansehen und studieren, können Sie sehen, daß die Zahlen, die links von dem Sprite stehen, das Bild beschreiben.

Nun müssen wir 63 Speicherplätze in dem Arbeitsspeicher des Computers finden, wo wir die Sprite Daten ablegen können. Der VIC Chip teilt den 16K Grafik Speicherblock in 256 Sprite Speicherblocks ein. Diese Blocks enthalten 64 Speicherplätze. Der letzte Speicherplatz in einem Sprite Datenblock wird nicht benutzt. Die Blocks werden von 0 bis 255 numeriert. Sie müssen unbedingt beachten, daß diese Blocks nicht andere grafische Daten (wie Grafiksymbole und Bitmaps) überlappen.

Die Nummer des ersten Speicherplatzes für ein Sprite Block ist immer ein ganzzahliges Vielfaches der Zahl 64 (Adresse 64, 128, 192 usw.) Wenn Sie nicht mehr als drei Sprites verwenden, ist ein geeigneter Platz für die Abspeicherung der Daten der Datensettenpufferbereich. Er umfaßt die Speicherplätze 828 bis 1023. Die Sprite Blocks in diesem Puffer sind Nummer 13 (Speicherplatz 832-895), Nummer 14 (Speicherplatz 896-959) und Nummer 15 (Speicherplatz 960-1023).

Die Beschränkung auf drei Sprite Data Blocks bedeutet aber nicht, daß die Anzahl der Sprites auf drei beschränkt ist. Sie können insgesamt bis zu acht Sprites und nur drei verschiedene Formen benutzen. Wenn Sie mehr als drei verschiedene Sprite Formen in einem Programm benutzen, ist es erforderlich, spezielle Pointer zu setzen und RAM Speicherplatz am Ende eines Grafikblocks zu reservieren. Wir werden dieses Thema später noch einmal aufgreifen. Nachdem wir den Sprite entworfen haben und uns entschieden haben, in welchem Bereich wir die Daten abspeichern wollen, müssen wir ein kleines Programm schreiben, das die Sprite Daten liest und in den entsprechenden Bereich speichert. Das folgende Programm wird die Zahlen für das 'Pferd' Sprite von dem DATA Statement lesen und in Sprite Block 13 speichern.

```
100 READ SB: IF SB < 0 THEN 1000
110 SS = SB*64
120 FOR I=0 TO 62: READ SD
130 POKE SS + I, SD:NEXT I
140 GOTO 100
```

```
150 DATA 13
160 DATA 0, 0, 0, 0, 0, 0, 0, 0, 12
170 DATA 0, 0, 62, 0, 0, 126, 0, 0, 255
180 DATA 0, 1, 251, 1, 127, 241, 3, 255, 240
190 DATA 3, 255, 240, 4, 255, 255, 0, 255, 240
200 DATA 0, 224, 60, 1, 96, 20, 2, 192, 10
210 DATA 2, 128, 10, 2, 128, 4, 12, 128, 2
220 DATA 0, 128, 0, 0, 192, 0, 0, 64, 0
.
.
.
999 DATA -1
1000 REM: START PROGRAMM
```

In Zeile 100 wird das erste DATA Statement gelesen. Hiermit wird die Sprite Block Nummer festgelegt, die benutzt wird, um die Adresse des ersten Speicherplatzes in Zeile 110 zu berechnen. Die Zeilen 120 und 130 lesen die nächsten 63 Zahlen des DATA Statements. Anschließend werden sie in die 63 aufeinanderfolgenden Speicherplätze, beginnend mit dem Speicherplatz, der in Statement 110 berechnet wird, abgespeichert (Speicherplatz 832).

Nachdem 63 Zahlen gelesen worden sind, wird der Computer in Zeile 140 veranlaßt, wieder zu Zeile 100 zurückzukehren. Das READ Statement liest die nächste Nummer des Sprite Blocks. Wenn diese Zahl negativ ist, weiß der Rechner, daß keine weiteren Sprite Daten folgen. Das Programm wird nun mit der Zeile 1000 fortgesetzt. Von hier ab beginnt das eigentliche Programm.

Der VIC-Chip muß wissen, welcher Datenblock für die Zeichnung eines Sprites benutzt werden soll. Der Commodore 64 benutzt die letzten acht Speicherplätze des Bildschirmspeichers als Zeiger (Pointer) für die Sprite Blocks. Die Sprite Block Pointer der Sprites 0-7 befinden sich in den Speicherplätzen 2040-2047. Wir wollen nun mit dem Pferd-Sprite Programm fortfahren. Die Sprites 0-3 können mit folgenden Programmzeilen als Pferde definiert werden:

```
1010 FOR I=0 TO 3
1020 POKE 2040+I,13:NEXT I
```


In diesem Fall würden die Pferde erscheinen. Wie Sie sehen, können Sie mehr als ein Sprite erzeugen, obwohl nur ein einziges Sprite Bild in dem Speicher gespeichert ist.

Die Farbe der Pferde wird dadurch ausgewählt, indem in den Speicherplätzen 53287 bis 53294 Zahlen gespeichert werden. Diese Zahlen kontrollieren die Farbe der acht Sprites, die von 0 bis 7 durchnummeriert sind.

```
1030 POKE 53287, 0: REM SPRITE 0 - SCHWARZ
1040 POKE 53288, 1: REM SPRITE 1 - WEISS
1050 POKE 53289, 9: REM SPRITE 2 - BRAUN
1060 POKE 53290, 12: REM SPRITE 3 - MITTELGRAU
```

Die acht Sprites von 0 bis 7 können ein- oder ausgeschaltet werden, indem die entsprechenden Bits der Speicherstelle 53269 auf 1 oder 0 gesetzt werden. Bit 0 schaltet das Sprite 0 ein oder aus, Bit 1 schaltet das Sprite 1 ein oder aus usw. So muß z. B. die Zahl 2 (00000010) auf den Speicherplatz 53269 gebracht werden, um das Sprite 1 einzuschalten. Auch hier ist wieder die Tabelle 1 ganz brauchbar, um die entsprechenden Zahlen zu finden. Es können bis zu 8 Bits gleichzeitig ein- oder ausgeschaltet werden. Sie müssen folgenden Befehl eingeben, um vier Sprites einzuschalten:

```
1070 POKE 53269, 15: REM EINSCHALTEN DER SPRITES 0-3
```

Am kompliziertesten ist es, die Sprites auf eine bestimmte Stelle des Bildschirms zu setzen. Jedes Sprite benötigt zwei Speicherplätze, sowie einen Teil eines dritten Speicherplatzes, um seine Position auf dem Bildschirm festzulegen. Ein weiterer erschwerender Faktor ist, daß die Video Map, auf der die Sprites dargestellt werden, größer als der Bildschirm ist! Die Sprites können dadurch von rechts oder von links auf dem Bildschirm erscheinen und entsprechend wieder verschwinden.

Damit ein Sprite Bild ganz auf dem Bildschirm erscheint, muß die vertikale Position zwischen 50 und 229 liegen. Die horizontale Position muß zwischen 24 und 320 angegeben werden. Die VIC Register, die die Positionen beschreiben, sind in Tabelle 2 angegeben.

Tabelle 2 Sprite Position Register

Sprite	Vertikale Position	Horizontale Position
0	53249	53248 + bit 0 of 53264
1	53251	53250 + bit 1 of 53264
2	53253	53252 + bit 2 of 53264
3	53255	53254 + bit 3 of 53264
4	53257	53256 + bit 4 of 53264
5	53259	53258 + bit 5 of 53264
6	53261	53260 + bit 6 of 53264
7	53263	53262 + bit 7 of 53264

Da die vertikale Position niemals größer als 255 ist, reicht ein Register aus, um diese Position darzustellen. Die horizontale Position überschreitet dagegen den Wert von 255. Deshalb ist ein zweiter Speicherplatz notwendig, um die entsprechende Zahl darzustellen. Diese zusätzliche Zahl wird niemals größer als 1, da die Bits in dem Speicher des Speicherplatzes 53264 benutzt werden, um anzuzeigen, daß die Zahl 256 zu dem entsprechenden Wert zu addieren ist. Das ist ein guter Trick. Aufgrund dieses zusätzlichen Bits können die Zahlen bis zu 511 gehen. Sie werden das Sprite allerdings nicht sehen, wenn die horizontale Positionsnummer größer als 344 ist. Um Sie nicht unnötig zu verwirren, werden die Sprites in unserem ersten Beispiel keine horizontalen Positionsnummer bekommen, die größer als 255 ist. Fügen Sie die folgenden BASIC-Zeilen dem oben beschriebenen Programm hinzu. Sie können so die Pferde auf dem Bildschirm positionieren.

```

1080 POKE 53248, 15: POKE 53249, 60: REM SPRITE 0 POSITION
1090 POKE 53250, 160: POKE 53251, 150: REM SPRITE 1 POSITION
1100 POKE 53252, 255: POKE 53253, 229: REM SPRITE 2 POSITION
1110 POKE 53254, 100: POKE 53255, 235: REM SPRITE 3 POSITION
1120 POKE 53281, 13: REM HELLGRUENER SCHIRM
1130 PRINT "(CLR)"

```

Das Programm kann jetzt gestartet werden. Ein Teil des schwarzen Pferdes erscheint in der linken Ecke des Bildschirms, ein weißes Pferd in der Mitte des Bildschirms, ein braunes Pferd unten rechts und ein Teil eines grauen Pferdes in der unteren Ecke.

Da das Programm nur ein Register für die horizontale Position benutzt, erscheint kein Sprite in dem rechten Drittel des Bildschirms. Wie Sprites in diesem Bereich positioniert werden können, wird später behandelt.

Ein interessantes Experiment ist, die Sprites so zu positionieren, daß sie 'überlappen'. Wenn Sprite 0, das schwarze Pferd, ein anderes Sprite überlappt, wird Sprite 0 dargestellt. Sprite 1, das weiße Pferd, hat ebenfalls Priorität gegenüber den Sprites 2 bis 7. Diese Rangfolge wird bis Sprite 7 fortgesetzt. Sprite 7 verschwindet somit hinter den Sprites 1 bis 7.

Versuchen Sie, das Programm auszulisten, während die Sprites noch auf dem Schirm sind. Beachten Sie, daß die Zeichen hinter den Sprites erscheinen. Die Priorität der Sprites gegenüber den Zeichen wird durch den Inhalt der Speicherzelle 53275 bestimmt. Das ist ein anderes der Register, bei denen jedes Bit die Art und Weise der korrespondierenden Sprites auswählt. Wenn ein Bit auf 1 gesetzt ist, wird das korrespondierende Sprite hinter dem Zeichen erscheinen. Ist das Bit jedoch auf 0 gesetzt, hat das Sprite gegenüber dem Zeichen Priorität. Geben Sie

POKE 53275,15

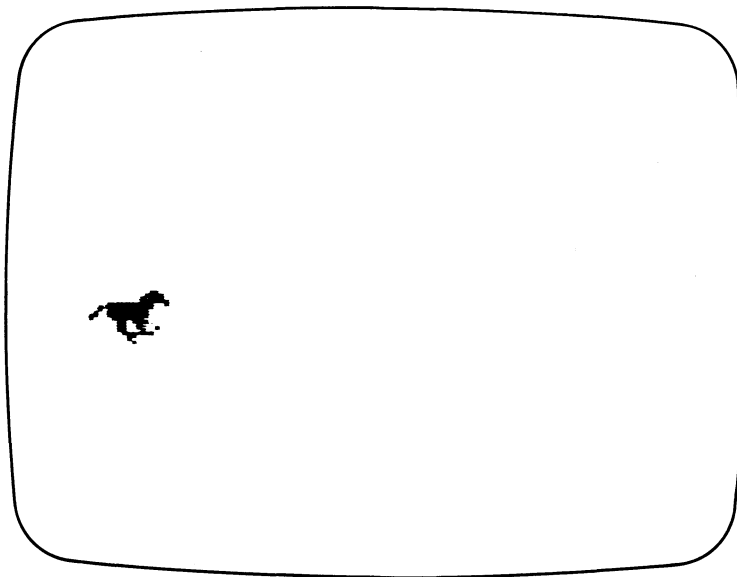
ein. Nun werden alle Sprites hinter den Zeichen der Programmliste erscheinen. Wird POKE 53275,3 eingegeben, so wird Sprite 0 und 1 hinter den Zeichen und Sprite 2 und 3 vor den Zeichen erscheinen.

Wenn es einen Konflikt hinsichtlich der Prioritäten gibt, hat das Sprite über Sprite-Priorität Vorrang. Beispiel: Wenn Sprite 1 mit Vorrang gegenüber Zeichen definiert ist und Sprite 0 so definiert ist, daß es hinter den Zeichen erscheint, bedeutet eine Überlappung aller drei Sprites, daß die Zeichen vor Sprite 0 erscheinen. Wird Sprite 0 gelöscht, werden die Zeichen gelöscht, und Sprite 1 bleibt zurück. Wenn Sprite 1 jetzt gelöscht wird, werden die Zeichen wieder erscheinen.

Die Ausdehnung von Sprites kann sowohl horizontal als auch vertikal erfolgen. Für die vertikale Ausdehnung wird dabei das entsprechende Bit in den Speicherplatz 53271 gesetzt. Soll das Sprite horizontal ausgedehnt werden, muß das entsprechende Bit in der Speicherstelle 53277 auf 1 gesetzt werden. Führen Sie folgendes Experiment durch. Ändern Sie in dem Pferd Sprite Programm die Statements 1080, 1090, 1100, und 1110.

```
1080 POKE 53248, 30: POKE 53249, 70: REM SPRITE 0 POSITION
1090 POKE 53250, 200: POKE 53251, 70: REM SPRITE 1 POSITION
1100 POKE 53252, 30: POKE 53253, 170: REM SPRITE 2 POSITION
1110 POKE 53254, 200: POKE 53255, 170: REM SPRITE 3 POSITION
```

Wenn Sie dieses Programm starten, erscheinen die vier Pferde auf dem Bildschirm. Geben Sie nun POKE 53271,10: POKE 53277,12 im Direktmodus ein. Die Sprites werden sich verändern. Sprite 0 bleibt unverändert, Sprite 1 wird



in der Höhe und Sprite 2 in der Breite verdoppelt. Sprite 3 wird in der Breite und Höhe verdoppelt.

Bis jetzt war die ganze Angelegenheit noch ziemlich einfach, da wir keine horizontale Position größer als 155 benutzt haben. Wenn Sie mit Sprites arbeiten, möchten Sie diese Einschränkung sicherlich nicht einfach hinnehmen. In dem folgenden Programm wird gezeigt, wie ein Sprite außerhalb der Grenze von 255 der horizontalen Position bewegt werden kann. Es zeigt außerdem, wie eine Bewegung durch Änderung der Form eines Sprites

simuliert werden kann. Das Programm definiert acht verschiedene Sprite Formen, die die einzelnen Bewegungsphasen eines galoppierenden Pferdes zeigen. Bei jeder Bewegung wird der Sprite 0 Datenblock Pointer, die Speicherzelle 2040, geändert. Das Programm überprüft, ob der letzte Datenblock benutzt wird. Ist das der Fall, wird der Datenblock Pointer für den Sprite 0 wieder auf den ersten Block gesetzt.

Um das Sprite außerhalb der horizontalen Position von 255 zu bewegen, müssen wir die Speicherzelle 53264 benutzen. Wenn das Bit des entsprechenden angezeigten Sprites 1 ist, wird die Zahl 256 zu der Zahl in der entsprechenden Speicherzelle für die horizontale Position addiert. Mit den Statements 1090 bis 1100 wird Sprite 0 positioniert.

Wir wollen uns nun einmal genauer ansehen, wie das Sprite 0 auf dem Schirm positioniert wird. Die vertikale Position (Speicherplatz 53249) wird konstant gehalten. Die Variable X entspricht der horizontalen Position. In Statement 1090 wird die horizontale Position durch AND mit 255 verknüpft und auf Speicherplatz 53248 gespeichert. Die horizontale Position wird so eingestellt, daß die Zahl, die auf diesem Speicherplatz abgespeichert wird, zwischen 0 und 255 liegt. Falls man nämlich versucht, eine Zahl auf diesem Speicherplatz zu speichern, die außerhalb des oben genannten Bereiches liegt, wird vom Computer eine Fehlermeldung ausgegeben. Wenn die horizontale Position kleiner als 256 ist, so ist das Ergebnis von X AND 255 mit X identisch. Ist die horizontale Position größer als 256, so wird das Ergebnis von X AND 255 den Wert 256-X haben. Das Programm prüft, ob die horizontale Position größer als 255 ist. Wenn das der Fall ist, wird Bit 0 des Speicherplatzes 53264 gesetzt, und das Programm springt zum nächsten Programmschritt. Das folgende Statement (Statement 1100 wird ausgeführt, wenn die horizontale Position des Sprites kleiner als 256 ist. Dieses Statement löscht Bit 0 des Speicherplatzes 53264.

"GALOPP"

10 POKE 52,62: POKE 56,62:CLR

20 READ SB: IF SB < 0 THEN 1000: REM LESEN SPRITE DATEN ADRESSE

30 SS=SB*64

40 FOR I=0 TO 62: READ SD

50 POKE SS+I,SD

60 NEXT I

70 GOTO 20

100 DATA 248, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0

110 DATA 30, 0, 0, 62, 0, 124, 255, 1, 255, 243, 3, 255, 240, 6, 255, 240, 12

120 DATA 255, 224, 0, 239, 224, 1, 231, 192, 1, 193, 192, 2, 129, 64, 2, 130, 64

130 DATA 4, 140, 64, 0, 0, 64, 0, 0, 64, 0, 0, 32, 249, 0, 0, 0, 0

140 DATA 0, 0, 0, 0, 96, 0, 0, 240, 0, 1, 240, 0, 1, 248, 0, 255, 200

150 DATA 3, 255, 192, 2, 255, 128, 4, 255, 128, 12, 255, 128, 8, 127, 0, 0, 55

160 DATA 128, 0, 51, 64, 0, 51, 32, 0, 49, 64, 0, 31, 192, 0, 12, 0, 0

171 DATA 8, 0, 0, 4, 0, 0, 0, 0, 250, 0, 0, 0, 0, 0, 0, 0, 0

180 DATA 96, 0, 0, 240, 0, 1, 240, 0, 1, 248, 0, 199, 200, 1, 255, 128, 7

190 DATA 255, 128, 15, 255, 128, 25, 255, 128, 49, 255, 128, 1, 223, 192, 0, 194, 64

200 DATA 0, 193, 64, 0, 229, 64, 0, 147, 64, 0, 136, 64, 0, 68, 0, 0, 64

210 DATA 0, 0, 32, 0, 251, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30

220 DATA 0, 0, 62, 0, 0, 127, 0, 0, 251, 0, 63, 249, 1, 255, 240, 3, 255

230 DATA 240, 7, 127, 240, 4, 127, 240, 0, 115, 248, 0, 240, 8, 0, 176, 8, 1

240 DATA 160, 12, 1, 32, 10, 1, 32, 1, 1, 16, 0, 1, 16, 0, 0, 136, 0

250 DATA 252, 0, 0, 0, 0, 0, 0, 0, 12, 0, 0, 62, 0, 0, 126, 0

260 DATA 0, 254, 0, 1, 251, 0, 127, 241, 3, 255, 240, 3, 255, 240, 4, 255, 240

270 DATA 0, 255, 240, 0, 224, 60, 1, 96, 20, 2, 192, 18, 2, 128, 10, 2, 128

280 DATA 10, 12, 128, 4, 0, 128, 2, 0, 192, 0, 0, 64, 0, 253, 0, 0, 0

290 DATA 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 28, 0, 0, 126, 0, 0

300 DATA 254, 0, 115, 251, 3, 255, 243, 6, 255, 240, 12, 255, 240, 0, 255, 224, 1

310 DATA 255, 224, 3, 193, 240, 7, 128, 72, 31, 0, 68, 2, 0, 66, 2, 0, 33

320 DATA 2, 0, 32, 2, 0, 32, 2, 0, 16, 254, 0, 0, 0, 0, 0, 0, 0

330 DATA 0, 0, 0, 0, 0, 0, 0, 28, 0, 0, 62, 0, 0, 126, 0, 255, 251

340 DATA 3, 255, 241, 6, 255, 240, 12, 255, 240, 0, 255, 224, 1, 199, 224, 3, 193

350 DATA 240, 3, 0, 144, 14, 0, 136, 2, 0, 132, 4, 0, 130, 12, 1, 1, 0

360 DATA 1, 0, 0, 0, 128, 255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

370 DATA 12, 0, 0, 28, 0, 0, 62, 0, 120, 254, 3, 255, 247, 6, 255, 241, 12

380 DATA 255, 224, 0, 255, 224, 1, 255, 224, 1, 195, 224, 3, 1, 96, 2, 1, 32

390 DATA 4, 2, 32, 8, 2, 16, 0, 4, 16, 0, 4, 16, 0, 4, 8, 0, 4

```
400 DATA 8,-1
1000 PRINT "(CLR)"
1010 POKE 53287, 9: REM SPRITE 0 - BRAUN
1020 POKE 53269, 1: REM SPRITE 0 EINSCHALTEN
1030 POKE 53249, 150: REM SPRITE 0 - Y POSITION
1040 POKE 53277, 1: REM DOPPELTE SPRITE AUSDEHNUNG
1050 POKE 53281, 13: REM HELLGRUENER SCHIRM
1060 SD=248: POKE 2040, SD: REM WAHL DES BLOCKES
1070 FOR X=0 TO 345 STEP 6: REM BEWEGUNG UEBER DEN BILDSCHIRM
1080 FOR T=0 TO 50 : NEXT T: REM WARTESCHLEIFE
1090 POKE 53248, X AND 255: IF X > 255 THEN POKE 53264, 1: GOTO 1110
1100 POKE 53264, 0
1110 SD = SD+1: IF SD=256 THEN SD=248: REM AENDERUNG DER FORM
1120 POKE 2040, SD
1130 NEXT X
1140 GOTO 1070
```

Zusammenstellung der Sprite Register

Sprite Register können in zwei verschiedene Typen eingeteilt werden. Der erste Typ umfaßt die Register, die die Informationen für nur ein Sprite enthalten. Diese Register sind in Tabelle 3 zusammengestellt.

Tabelle 3: 8 Bit Sprite Register

Sprite	Horizontale Position (Bits 0–7)	Vertikale Position	Farbe
0	53248	53249	53287
1	53250	53251	53288
2	53252	53253	53289
3	53254	53255	53290
4	53256	53257	53291
5	53258	53259	53292
6	53260	53261	53293
7	53262	53263	53294

Der zweite Typ benutzt ein Register, um die Informationen für alle acht Sprites zu speichern. Jedes Bit kontrolliert die Darstellung des entsprechenden Sprites. Diese Register sind in Tabelle 4 zusammengestellt.

Tabelle 4: 1 Bit Sprite Register

Register	Funktion
53264	Horizontale Position größer als 255
53269	An- und Ausschalten der Sprites
53271	Vertikale Verdopplung
53275	Priorität gegenüber Zeichen
53277	Horizontale Verdopplung

Ein bestimmtes Bit kann auf 1 gesetzt werden, indem der bestehende Wert in einem Register durch OR mit der entsprechenden Zahl verknüpft wird. Ein Bit kann gelöscht und damit auf 0 gesetzt werden, indem der bestehende Wert durch 0 mit dem Komplement der Zahl verknüpft wird. Die allgemeine Form, ein Bit zu setzen, ist:

POKE(Speicherzelle) OR B2

Die allgemeine Form, ein Bit zu löschen, ist:

POKE(Speicherzelle), PEEK(Speicherzelle) AND NOT B2

Bei diesen beiden Anweisungen bedeutet 'Speicherzelle' eines der angegebenen Sprite Register; B2 ist von der Sprite Nummer abhängig.

Der Entwurf und die Eingabe von Sprites können sehr zeitaufwendig sein. Ein Sprite Editor ist hier sehr nützlich. Nachstehendes Programm stellt einen Sprite Editor dar, mit dem Sie ein Sprite erzeugen und es auf dem Bildschirm ausgeben können. Sie können außerdem das Sprite mit mehreren Kommandos handhaben. Der Bildschirm wird durch das Programm in drei Bereiche aufgeteilt. Der 24x21 Zeichen große Bereich ist für das Editieren des Sprites bestimmt. Wenn das Sprite erzeugt wird, wird es in dem oberen Bildschirmbereich erscheinen. Vergrößerte Sprites werden in der unteren rechten Ecke des Bildschirms ausgegeben. Das Programm läßt mehrere Betriebsarten zu. Der Cursor Modus wird mit einem Buchstaben angezeigt. Im Zeichenmodus, D, werden Pixels eingeschaltet, wohin der Cursor auch bewegt werden mag. Im Erase Modus (= Löschmodus) E, werden die Pixels ausgeschaltet. Die Cursor Bewegung hat im normalen Modus, N, keinen Einfluß auf die Pixels. Nachdem ein Sprite erzeugt worden ist, werden die Kommandos und DATA Statements, die dieses Sprite in einem Programm erzeugen würden, auf dem Bildschirm oder über den Drucker ausgegeben. Die Sprite Editor Kommandos sind in Tabelle 5 zusammengestellt.

Tabelle 5: Sprite Editor Kommandos

Kommando	Funktion
*	Einschalten Pixel bei Cursor Position
Leerzeichen	Ausschalten Pixel bei Cursor Position
Cursor links	Cursor Bewegung nach links
Cursor rechts	Cursor Bewegung nach rechts
Cursor nach oben	Cursor Bewegung nach oben
Cursor nach unten	Cursor Bewegung nach unten
Return	Cursor Bewegung in die linke Ecke
Home	Cursor Bewegung in die obere linke Ecke
Clear	Löschen Bildschirm und Cursor
X	Bewegung in die linke obere Ecke -
Y	Horizontale Ausdehnung des Sprite
B	Vertikale Ausdehnung des Sprite
	Ausdehnung des Sprites in
	X/Y Richtung
L	Liste der Kommandos
F1	Hintergrundfarbe
F3	Spritefarbe
F7	Cursor Modus
Q	Eingabe des neuen Kommandos

"SPRITE"

```

5 CS=2:BG=1:PL=0: REM WEISSER HINTERGRUND, ROTES SPRITE
7 PM(0)=14: PM(1)=4:PM(2)=5
10 POKE 53280,6: POKE 53281,1: REM BLAUE RAHMENFARBE, WEISSE HINTER-
    GRUNDFARBE
20 SC=1024 :SP=832:CR=55296: REM BILDSCHIRM UND FARBSPEICHER
30 REM ZEICHENFARBE TUEBKIS
40 PRINT"(CLR)": FOR I=0 TO 999: POKE CR+I, 3: NEXT I
50 PRINT"(HOME) (BLU) (RVS ON)   SPRITE EDITOR VON PAUL SCHATZ  (RVS OFF)"
60 REM ARBEITSBEREICH
70 POKE SC+81,112: POKE SC+106,110:POKE SC+961,109: POKE SC+986,125
80 FOR I=0 TO 23:POKE SC+82+I,114:POKE SC+962+I,113:NEXT I
90 FOR I=0 TO 20: POKE SC+121+(40*I),107:POKE SC+146+(40*I),115:NEXT I
100 REM SPEICHERBEREICH FARBE ROT
110 FOR I=0 TO 20:FOR J=0 TO 23:POKE CR+122+(40*I)+J,2:NEXT J,I
115 GOSUB 1700: REM FARB-SPRITES
120 REM ENTWURF SPRITES
130 POKE SC+110,112:POKE SC+115,110:POKE SC+270,109:POKE SC+275,125
140 FOR I=0 TO 3: POKE SC+111+I,64:POKE SC+271+I,64:NEXT I
150 FOR I=0 TO 2: POKE SC+150+(40*I),93:POKE SC+155+(40*I),93:NEXT I
160 FOR I=0 TO 2:FOR J=0 TO 3:POKE SC+151+(40*I)+J,160:NEXT J,I
170 PRINT"(HOME)(CRSR DOWN)(CRSR DOWN)(28 CRSR RT)S(CRSR DOWN)
    (CRSR LEFT)P(CRSR DOWN)(CRSR LEFT)R(CRSR DOWN)(CRSR LEFT)I
    (CRSR DOWN)(CRSR LEFT)T(CRSR DOWN)(CRSR LEFT)E"
200 REM AUSDEHNUNG
210 POKE SC+710,112:POKE SC+718,110:POKE SC+990,109:POKE SC+998,125
220 FOR I=0 TO 6: POKE SC+711+I,64:POKE SC+991+I,64:NEXT I
230 FOR I=0 TO 5:POKE SC+750+(40*I),93:POKE SC+758+(40*I),93:NEXT I
240 FOR I=0 TO 5: FOR J=0 TO 6:POKE SC+751+(40*I)+J,160:NEXT J,I
250 PRINT"(HOME)(16 CRSR DN)(28 CRSR RT)";
260 PRINT"(CRSR DOWN)E(CRSR DOWN)(CRSR LEFT)X(CRSR DOWN)(CRSR LEFT)P
    (CRSR DOWN)(CRSR LEFT)A(CRSR DOWN)(CRSR LEFT)N(CRSR DOWN)(CRSR LEFT)D
    (CRSR DOWN)(CRSR LEFT)E(CRSR DOWN)(CRSR LEFT)D(HOME)"
300 REM BEFEHLSTABELLE
305 PRINT"(HOME)(7 CRSR DN)(30 CRSR RT) CURSOR:":PRINT
310 PRINT"(28 CRSR RT)BEFEHLE:"

```

```
320 PRINT"(29 CRSR RT) (LT RED)L(BLU) IST"
330 PRINT"(29 CRSR RT)(LT RED)Q(BLU)UIT"
340 PRINT"(29 CRSR RT) AUSDEHNUNG (LT RED)X(BLU)"
350 PRINT"(29 CRSR RT) AUSDEHNUNG (LT RED)Y(BLU)"
360 PRINT"(29 CRSR RT) AUSDEHNUNG (LT RED)B(BLU)EIDE"
370 PRINT"(CRSR UP)(29 CRSR RT)(LT RED) F1 F3 F7(BLU)"
380 POKE SC+318,PM(PL):POKE CR+318,10
400 REM      AUSGABE DES SPRITES AUF DEM BILDSCHIRM
410 GOSUB 1000
420 REM
430 GOSUB 1200
440 REM LOESCHEN BILDSCHIRM
450 PRINT"(RED)(HOME)(CRSR DOWN)(CRSR DOWN)":POKE 211,2:POKE 204,0
500 REM EINGABE KOMMANDO
510 GET A$: IF A$="" THEN 510
520 IF A$="Q" THEN POKE 53269,0:PRINT"(CLR)":END
530 IF A$="L" THEN GOSUB 2000: GOTO 40
540 IF A$="X" THEN GOSUB 1400
550 IF A$="Y" THEN GOSUB 1500
560 IF A$="B" THEN GOSUB 1600
570 IF A$="(HOME)"THEN GOSUB 1300:GOTO 450
580 IF A$="(CLR)" THEN GOSUB 1100: GOSUB 900:GOTO 450
590 IF A$="(CRSR UP)"THEN GOSUB 3000
600 IF A$="(CRSR DOWN)"THEN GOSUB 3100
610 IF A$="(CRSR LEFT)" THEN GOSUB 3200
620 IF A$="(C-R)"THEN GOSUB 3300
630 IF A$=CHR$(13) THEN GOSUB 3400
640 IF A$=" " THEN GOSUB 3500:GOSUB 3300
650 IF A$="*" THEN GOSUB 3600:GOSUB 3300
660 IF A$="(F1)" THEN BG=BG+1:GOSUB 1700
670 IF A$="(F3)" THEN CS=CS+1:GOSUB 1800
680 IF A$="(F7)" THEN PL=PL+1:IF PL > 2 THEN PL=0
690 POKE SC+318,PM(PL)
700 GOTO 510
900 REM LOESCHEN BILDSCHIRM
905 POKE 204,1:REM LOESCHEN CURSOR
910 FOR I=0 TO 20:FOR J=0 TO 23:POKE SC+122+(40*I)+J,32:NEXT J,I
```

920 RETURN

1000 REM UNTERPROGRAMM - AUSGABE DER SPRITES AUF DEM SCHIRM

1005 POKE 204,1:REM LOESCHEN CURSOR

1010 FOR I=0 TO 20:FOR J=0 TO 2: FOR K=7 TO 0 STEP-1

1020 CH=32:IF (PEEK(SP+(3*I)+J) AND(2(UP ARROW)K)) THEN CH=204

1030 POKE(SC+122+(40*I)+(8*J)+(7-K)),CH

1040 NEXT K,J,I:RETURN

1100 REM UNTERPROGRAMM - LOESCHEN SPRITE

1110 FOR I=0 TO 62:POKE 832+I,0:NEXT I: RETURN

1200 REM UNTERPROGRAMM - SPRITE AUF DEN SCHIRM

1210 POKE 2040,13:REMDATENIN SPEICHERBEREICH 832-894 (64*13=832)

1220 POKE 53248,20:POKE 53264,1:REM X KOORDINATE VON SPRITE 0

1230 POKE 53249,75:REM Y KOORDINATE VON SPRITE 0

1240 POKE 53287,CS:REM FARBE VON SPRITE 0

1250 POKE 53269,1:REM EINSCHALTEN SPRITE 0

1260 RETURN

1300 REM CURSOR BEWEGEN

1310 LO=SC+(40*PEEK(214))+PEEK(211):REM POSITION

1320 CH=32:POKE 204,1:REM AUSSCHALTEN CURSOR

1330 IF(PEEK(LO) AND 127) < > 32 THEN CH=204

1340 POKE LO,CH:RETURN

1400 REM X AUSDEHNUNG

1410 POKE 2041,13:REM SPRITE DATEN 832-894

1420 POKE 53277,2:REM X AUSDEHNUNG SPRITE 1

1430 POKE 53271,0:REM KEINE Y AUSDEHNUNG

1440 POKE 53250,20:POKE 53264,PEEK(53264)OR 2:REM X KOORDINATE VON
SPRITE 1

1450 POKE 53251,197:REM Y KOORDINATE VON SPRITE 1

1460 POKE 53288,CS:REM FARBE VON SPRITE 1

1470 POKE 53269,3:REM EINSCHALTEN SPRITE 0 UND SPRITE 1 (2+1=3)

1480 RETURN

1500 REM Y AUSDEHNUNG

1510 POKE 2041,13:REM SPRITE DATEN 832-894

1520 POKE 53277,0:REM KEINE X AUSDEHNUNG

1530 POKE 53271,2:REM Y AUSDEHNUNG FUER SPRITE 1

1540 POKE 53250,20:POKE 53264,PEEK(53264) OR2:REM X KOORDINATE VON
SPRITE 1

```

1550 POKE 53251,197:REM Y KOORDINATE VON SPRITE 1
1560 POKE 53288,CS:REM FARBE VON SPRITE 1
1570 POKE 53269,3:REM EINSCHALTEN VON SPRITE 0 UND SPRITE 1 (2+1=3)
1580 RETURN
1600 REM X UND Y AUSDEHNUNG
1610 POKE 2041,13:REM SPRITE DATEN 832-894
1620 POKE 53277,2:REM X AUSDEHNUNG FUER SPRITE 1
1630 POKE 53271,2:REM Y AUSDEHNUNG FUER SPRITE 1
1640 POKE 53250,20:POKE 53264,PEEK(53264) OR 2:REM X KOORDINATE VON
    SPRITE 1
1650 POKE 53251,197:REM Y KOORDINATE VON SPRITE 1
1660 POKE 53288,CS:REM FARBE VON SPRITE 1
1670 POKE 53269,3: REM EINSCHALTEN SPRITE 0 UND SPRITE 1 (2+1=3)
1680 RETURN
1700 REM HINTERGRUNDFARBE FUER SPRITE BEREICH
1710 IF BG > 15 THEN BG=0
1720 FOR I=0 TO 2:FOR J=0 TO 3:POKE CR+151+(40*I)+J,BG:NEXT J,I
1730 FOR I=0 TO 5: FOR J=0 TO 6:POKE CR+751+(40*I)+J,BG:NEXT J,I:
    RETURN
1800 REM FARBE SPRITES
1810 IF CS > 15 THEN CS= 0
1820 POKE 53287,CS:POKE 53288,CS:RETURN
2000 REM LIST ROUTINE
2010 POKE 204,1:POKE 53269,0:AUSSCHALTEN
2020 INPUT"(CLR)(CRSR DOWN)(BLU) WELCHER SPRITE";S:IF S > 7 OR S < 0
    THEN 2020
2030 INPUT"WELCHER DATENBLOCK";B
2040 INPUT"AUF DEM (RVS ON) D(RVS OFF)RUCKER ODER(RVS ON)S(RVS OFF)
    CHIRM";B$:PRINT"(CLR)"
2050 OD=3:IF B$="D" THEN OD=4
2060 OPEN 1,OD:IF OD=4 THEN GOSUB 2500
2070 VC=53248:BP=2040
2080 PRINT#1," 1000 POKE 53269, PEEK(53269) OR"; 2(UP ARROW)S;
2085 PRINT#1," : REM EINSCHALTEN SPRITE";S;CHR$(13);
2090 PRINT#1," 1010 POKE";BP+S;",";B;
2095 PRINT#1," :REM SPRITE";S;"DATEN VON";B*64;"BIS";B*64+62;CHR$(13);
2100 PRINT#1," 1020 FOR I=0 TO 62: READ DT: POKE";B*64;" +I,Q:NEXT I";

```

```

    CHR$(13);
2105 PRINT#1," 1030 IF ABS(X/256) THEN POKE 53264, PEEK(53264) OR";
2107 PRINT#1,2(UP ARROW)S;":X=X-256";CHR$(13);
2110 PRINT#1," 1040 POKE";VC+2*S;":X:REM X KOORDINATE VON SPRITE";S;
    CHR$(13);
2120 PRINT#1," 1050 POKE";VC+1+2*S;
2125 PRINT#1,"Y: REM Y KOORDINATE VON SPRITE";S;CHR$(13);
2130 PRINT#1," 1060 POKE"; 53287+S;":":CS;": REM FARBE DES SPRITES";S;
    CHR$(13);
2140 PRINT#1,"1070 POKE 53277, PEEK(53277) OR";2(UP ARROW)S;
2145 PRINT#1,":REM X AUSDEHNUNG DES SPRITES";S;CHR$(13);
2150 PRINT#1," 1080 POKE 53271, PEEK(53271) OR"; 2(UP ARROW)S;
2155 PRINT#1,":REM Y AUSDEHNUNG DES SPRITES";S;CHR$(13);
2160 IF OD=3 THEN PRINT#1, CHR$(13);"(RVS ON) DRUECKEN SIE IRGENDEINE
    TASTE UM FORTZUFAHREN(RVS OFF)": GOSUB 2300
2170 FOR I=0 TO 6
2180 PRINT#1,1090 +I*10;"DATA";
2190 FOR J=0 TO 7: PRINT#1, PEEK(SP+I*9+J);":":NEXT J
2200 PRINT#1,PEEK(SP+I*9+8);CHR$(13);: NEXT I
2210 PRINT#1:CLOSE 1:PRINT:PRINT"(RVS ON) DRUECKEN SIE IRGENDEINE TASTE
    UM ZUM EDITOR ZURÜCKZUKEHREN (RVS OFF)"
2300 GET C$: IF C$=""THEN 2300
2310 RETURN
2500 REM UNTERPROGRAMM - AUSGABE DES SPRITE BILDES
2505 OPEN 4,4,4
2510 PRINT#4,"SPRITE BILD";CHR$(13);CHR$(13);
2520 PRINT#4,".....";CHR$(13);
2530 FOR I=0 TO 20:LN$=""
2540 FOR J=0 TO 2:FOR K=7 TO 0 STEP -1
2550 CA$=" ":IF(PEEK(SP+(3*I)+J)AND(2(UP ARROW)K)) THEN CA$="*"
2560 LN$=LN$+CA$:NEXT K,J
2570 PRINT#4,".":LN$;".":CHR$(13);:NEXT I
2580 PRINT#4,".....";CHR$(13);
2585 PRINT#4,CHR$(13);CHR$(13);
2590 RETURN
3000 REM AUFWAERTSBEWEGUNG

```

```
3010 IF PEEK(214) < 4 THEN RETURN
3012 IF PL=2 THEN GOSUB 3500
3014 IF PL=1 THEN GOSUB 3600
3020 GOSUB 1300:CM=PEEK(211):RO=PEEK(209)-40
3025 IF RO < 0 THEN RO=256+RO:POKE 210, PEEK(210)-1
3030 POKE 209, RO:POKE 211,CM:POKE 204,0:REM CURSOR BEWEGEN UND BLINKEN
    LASSEN
3040 POKE 214, PEEK(214)-1:POKE 201,PEEK(201)-1:RETURN
3100 REM BEWEGUNG NACH UNTEN
3110 IF PEEK(214) > 22 THEN RETURN
3112 IF PL=2 THEN GOSUB 3500
3114 IF PL=1 THEN GOSUB 3600
3120 GOSUB 1300:CM=PEEK(211):RO=PEEK(209)+40
3125 IF RO > 255 THEN RO=RO-256:POKE 210,PEEK(210)+1
3130 POKE 209,RO:POKE 211,CM:POKE 204,0:REM CURSOR BEWEGEN UND BLINKEN
    LASSEN
3140 POKE 214,PEEK(214)+1:POKE 201,PEEK(201)+1:RETURN
3200 REM BEWEGUNG NACH LINKS
3210 IF PEEK(211) < 3 THEN RETURN
3212 IF PL=2 THEN GOSUB 3500
3214 IF PL=1 THEN GOSUB 3600
3220 GOSUB 1300
3230 POKE 211, PEEK(211)-1:POKE 204,0:REM CURSOR BEWEGEN UND BLINKEN
    LASSEN
3240 RETURN
3300 REM BEWEGUNG NACH RECHTS
3310 IF PEEK(211) > 24 THEN RETURN
3312 IF PL=2 THEN GOSUB 3500
3314 IF PL=1 THEN GOSUB 3600
3320 GOSUB 1300
3330 POKE 211, PEEK(211)+1:POKE 204,0:REM CURSOR BEWEGEN UND BLINKEN
    LASSEN
3340 RETURN
3400 REM RETURN
3410 GOSUB 1300
3420 POKE 211,2:POKE 204,0
3430 RETURN
```

3500 REM BIT LOESCHEN

3510 LO=SC+(40*PEEK(214))+PEEK(211)

3520 POKE LO,32

3530 J=INT((PEEK(211)-2)/8):K=9-PEEK(211)+(J*8)

3540 I=PEEK(214)-3

3550 POKE(SP+(3*I)+J),PEEK(SP+(3*I)+J)AND(255-(2(UP ARROW)K))

3570 RETURN

3600 REM BIT SETZEN

3610 LO=SC+(40 *PEEK(214))+PEEK(211)

3620 POKE LO,204

3630 J=INT((PEEK(211)-2)/8):K=9-PEEK(211)+(J*8)

3640 I=PEEK(214)-3

3650 POKE(SP+(3*I)+J),PEEK(SP+(3*I)+J)OR(2(UP ARROW)K)

3670 RETURN

Zeichen

Der Commodore 64 hat zwei eingebaute Zeichensätze, die jeweils 256 Zeichen umfassen. Beide Zeichensätze können alternativ benutzt werden. Die Umschaltung erfolgt durch gleichzeitiges Drücken der SHIFT- und der COMMODORE-Taste. Wenn der Commodore 64 eingeschaltet wird, befindet er sich automatisch im Großbuchstaben/Grafikmodus. Der alternativ wählbare Zeichensatz hat Groß- und Kleinbuchstaben. Der jeweils angezeigte Zeichensatz wird durch den Inhalt der Bits 1-3 von Register 53272 bestimmt. Versuchen Sie einmal folgendes. Schalten Sie den Computer ein und geben Sie

```
PRINT PEEK(53272)
```

ein. Es wird der Wert 21 ausgegeben. Drücken Sie nun die SHIFT- und die COMMODORE-Taste, um den anderen Zeichensatz einzuschalten. Schauen Sie sich mit dem PEEK-Befehl den Inhalt der Speicherzelle 53272 an. Der Wert wird jetzt 23 sein.

Um sehen zu können, welche Zeichen bei jedem Zeichensatz möglich sind, geben Sie folgendes Programm ein:

```
10 POKE 53281,0:REM WEISSER SCHIRM
20 SC=1024:CR=55286:REM BILDSCHIRM UND FARBSPEICHER
30 PRINT"(CLR)(BLK)":REM BILDSCHIRM LOESCHEN
40 FOR I=0 TO 255
50 POKE SC+80+2*I,I:POKE CR+80+2*I,0: NEXT I
60 POKE 53272,21: REM EINSCHALTEN DER GRAFIKZEICHEN
70 PRINT"(HOME)GRAFIKZEICHENSATZ"
80 GET A$:IF A$= "" THEN 80
90 POKE 53272,23:REM EINSCHALTEN DER GROSS- UND KLEINBUCHSTABEN
100 PRINT" (HOME)U/LOZEICHENSATZ GROSS-,KLEINBUCHSTABEN "
110 GET A$: IF A$="" THEN 110
120 GOTO 60--
```

Alle 256 Zeichen werden auf dem Schirm ausgegeben.

Neben diesen beiden Zeichensätzen können Sie auch einen vollständig neuen Zeichensatz definieren. So kann beispielsweise ein Zeichensatz für andere Schriftarten (Arabisch, Griechisch etc.) erzeugt werden. Die Dar-

stellung der normalen Zeichen ist im RAM-Speicher gespeichert. Für die 256 Zeichen werden 2048 Speicherplätze benötigt. Der Inhalt der Bits 1-3 von Register 53272 zeigt auf den Anfang eines 2K RAM Blocks, der die Zeichen enthält. Bit 4-7 der Speicheradresse 53272 zeigen auf den Start des Bildschirmspeichers. Es ist deshalb wichtig, Bit 1-3 zu ändern, ohne daß davon die höheren Bits betroffen sind. Das wird mit folgendem Befehl erreicht:

POKE 53272, (PEEK(53272) AND 240) OR CS

Die Werte für CS sind in Tabelle 6 angegeben.

Tabelle 6: Zeichensatz Pointer

Startadresse des Zeichensatzes		CS
4096	Grafikzeichensatz	4
6144	Großschrift/Kleinschriftzeichen-Satz	6
8192		8
10240		10
12288		12
14336		14

Beachten Sie, daß die Beschreibungen für die 'eingebauten' Zeichensätze in den Speicherzellen 4096 und 6144 abgelegt sind. Diese Speicherzellen können nicht benutzt werden, um einen benutzerdefinierten Zeichensatz zu beschreiben. Die beiden RAM Speicherzellen beinhalten nicht direkt die Beschreibung der Zeichen. Bevor der Computer die Zeichen auf dem Schirm anzeigt, schaltet er auf einen ROM um. Der Video Chip geht nun davon aus, daß dieser die Speicherzelle 4096 enthält. Der ROM enthält die Zeichenbeschreibungen von der Speicherzelle 53248 bis 57343.

Um zu verstehen, wie man einen Zeichensatz definieren kann, sollten wir uns zunächst ansehen, wie ein Zeichen erzeugt wird. Jedes Zeichen wird bei dem Commodore 64 durch ein Punktmuster erzeugt, welches mit einer 8x8 Matrix gebildet wird. So wird beispielsweise das Zeichen 0, @ durch ein Muster von Pixels (wie Abb. 4 zeigt) dargestellt.

```

60  . . * * * * . .
102 . * * . . * * .
110 . * * . * * * .
110 . * * . * * * .
96  . * * . . . . .
98  . * * . . . * .
60  . . * * * * . .
0   . . . . . . . .

```

Abb. 4: Pixel Muster von CHR\$(0)("@")

Das * bedeutet, daß das Pixel eingeschaltet ist, der • bedeutet, daß das Pixel ausgeschaltet ist. Der Computer speichert dieses Bild in acht aufeinanderfolgenden Speicherplätzen. Jeder Speicherplatz enthält eine Beschreibung des Punktmusters für eine Zeile. Die Speicherplätze sind den Zeilen entsprechend von oben nach unten geordnet. Die Inhalte der Speicherplätze sind gemäß den an- und ausgeschalteten Pixels aufgebaut. Die acht Speicherplätze, die das Zeichen @ erzeugen, haben die Werte 60, 102, 110, 110, 96, 98, 60 und 0. Diese Zahlen sind in acht aufeinanderfolgenden Speicherplätzen gespeichert. Die Beschreibung des Zeichens 1, des Buchstabens A, ist in den nächsten acht Speicherplätzen gespeichert. Wenn der Computer den Befehl erhält, das Zeichen 0 auszugeben, nimmt der 'Video Display Chip' die acht Speicherplätze, deren Inhalt das Zeichen 0 darstellt, und rekonstruiert auf dem Schirm das Zeichen @. Soll der Computer den Buchstaben A anzeigen, wird die nächste Gruppe von acht Speicherplätzen benutzt.

Wenn wir ein neues Zeichen definieren, besteht der erste Schritt darin, ein Punktbild mittels einer 8x8 Matrix zu erzeugen. Nehmen Sie sich ein Blatt kariertes Papier, und umrahmen Sie 8x8 Kästchen. Entwerfen Sie ein Zeichen, indem Sie die entsprechenden Quadrate ausfüllen. Jede ausgefüllte Zeile wird in eine Binärzahl umgewandelt, indem die gefüllten Quadrate auf 1 und die leeren Quadrate auf 0 gesetzt werden. Die entsprechende Dezimalzahl kann aus Tabelle 1 ersehen werden.

Nachdem Sie den Zeichensatz entworfen haben, können Sie mit dem Programmieren beginnen. Sie müssen zunächst den Speicherplatz für den Zeichensatz reservieren und darauf achten, daß er nicht durch das BASIC-System zerstört wird. Das geschieht, indem das Ende des Zeichenspeichers und der Anfang des String Pointers entsprechend gesetzt wird. Geben Sie folgenden Befehl ein:

```
10 POKE 52,56: POKE 56,56:CLR
```

Aufgrund dieser Anweisung kann BASIC nicht die Speicherplätze über 14336 benutzen. Der Zeichensatz, der auf den Speicherplätzen 14336 bis 16383 abgespeichert ist, wird nicht zerstört.

Es ist sinnvoll, die Zeichen des Zeichengenerator ROMs in den reservierten Bereich zu kopieren. Damit ist sichergestellt, daß für jedes Zeichen eine Zeichenbeschreibung existiert. Um den Computer den Zugriff zu den Speicherplätzen in dem ROM zu erlauben, ist es notwendig, die Eingabetastatur auszuschalten und den Zeichensatz im ROM einzuschalten. Das folgende Programm überträgt den vollständigen Zeichensatz in einen neuen Speicherbereich.

```
10 POKE 52,56:POKE 56,56:CLR
20 POKE 56334, PEEK(56334) AND 254
30 POKE 1, PEEK(1) AND 251
40 FOR I=0 TO 2047
50 POKE I+14336, PEEK(53248+I)
60 NEXT I
70 POKE 1, PEEK(1) OR 4
80 POKE 56334, PEEK(56334) OR 1
```

Die Anweisungen der Zeilen 20 und 30 erlauben den Zugriff auf die entsprechenden Speicherplätze im ROM. Die Anweisungen 70 und 80 führen den Computer in seinen Ursprungszustand zurück. Der mit PEEK gelesene Wert bestimmt, welche Zeichen übertragen werden. Tabelle 7 gibt an, welche Gruppe von Zeichenbeschreibungen auf welche Speicherplätze abgespeichert sind. Es ist möglich, verschiedene Zeichensätze zu kombinieren. Mit nachstehendem Programm wird ein Zeichensatz in der Form aufgebaut, daß die Zeichen 0 bis 27 Groß-/Kleinbuchstaben darstellen, die Zeichen 128 bis 191 Grafiksymbole und die Zeichen 192 bis 255 Grafiksymbole im Reverse Modus darstellen.

```
10 POKE 52,56:POKE 56,56:CLR
20 POKE 56334,PEEK(56334) AND 254
30 POKE 1,PEEK(1) AND 251
40 FOR I=0 TO 1023:POKE I+ 14336,PEEK(55296 + I): NEXT I
```

```

50 FOR I=0 TO 511: POKE I+15360,PEEK(53760+I): NEXT I
60 FOR I=0 TO 511: POKE I+15872, PEEK(54784+I):NEXT I
70 POKE 1,PEEK(1) OR 4
80 POKE 56334,PEEK(56334) OR 1

```

Tabelle 7: Speicheradressen des ROM für Zeichen

PEEK	Übertragene Zeichen	Code
53248+I	Großbuchstaben	0- 63
53760+I	Grafiksymbole	64-127
54272+I	Großbuchstaben im Reverse Modus	128-191
54784+I	Grafiksymbole im Reverse Modus	192-255
55296+I	Kleinbuchstaben	0- 63
55808+I	Großbuchstaben	64-127
56320+I	Kleinbuchstaben im Reverse-Modus	128-191
56832+I	Großbuchstaben im Reverse-Modus	192-255

Eine sehr nützliche Methode zur Speicherung von Beschreibungen für gebräuchliche Zeichen in den reservierten RAM Bereich ist die Verwendung einer FOR..NEXT Schleife mit READ DATA und POKE. Das folgende BASIC Programm soll diese Methode einmal anschaulich darstellen.

```

90 READ CN:IF CN < 0 THEN 1000
100 FOR I=0 TO 7: READ CD
110 POKE 14336+8*CN+I,CD:NEXT I
120 GOTO 90
130 DATA 0, 40, 16, 146, 214, 254, 238, 198, 0
140 DATA etc.
.
.
999 DATA -1
1000 POKE 53272, (PEEK(53272) AND 240) OR 14: REM BEGINN DES PROGRAMMS

```

Zeile 90 liest die erste Zahl des DATA Statements. Das ist die Zeichennummer des Zeichens, welches definiert werden soll. Mit diesem kann angegeben werden, daß kein weiteres Datum gelesen werden soll. In diesem Fall wird irgendein negativer Wert angegeben. Der Wert des letzten DATA-Statement ist nämlich negativ. Das bedeutet, daß das Programm zu einem Teil verzweigen soll, der die neuen Zeichen benutzt. Statement 100 liest die

nächsten acht Werte eines DATA Statements. Bei der Anweisung 110 werden die Zeichenummer CN und der Zeilenindex I benutzt, um zu berechnen, auf welchem Speicherplatz die Zeilenbeschreibung abgespeichert werden soll. In Zeile 120 wird das Programm wieder zu Statement 90 verzweigt, um dann eine neue Zeichenummer zu lesen. Zeile 1000 ist der Beginn des Programnteils, der den neuen Zeichensatz verwendet. Der Zeichensatz Pointer in Speicherzelle 53272 wird auf den Anfang des Programms zurückgesetzt.

Es gibt eine weitere Methode, Zeichen zu definieren. Diese Methode ist leichter zu benutzen, da die Eingabe eine Vorstellung des visuellen Eindrucks eines Zeichens erlaubt. Diese Methode ist allerdings langsamer und verbraucht überdies mehr Speicherplatz als die erste Methode. Sie ist dann sinnvoll anzuwenden, wenn nur ein paar Zeichen definiert werden sollen.

```

90 READ CN: IF CN<0 THEN 1000
100 FOR I=0 TO 7: READ CD$
110 BY = 14336+8*CN+I:POKE BY,0
120 FOR J=0 TO 7
130 IF MID$(CD$,J+1,1)= "" GOTO 150
140 POKE BY, PEEK(BY) OR 2 (7-J)
150 NEXT J: NEXT I
160 GOTO 90
170 DATA 0
171 DATA " * * "
172 DATA " * "
173 DATA "* * *"
174 DATA "*** * ***"
175 DATA "***** "
176 DATA "*** *** "
177 DATA "*** ** "
178 DATA " "
.
.
.
999 DATA -1
1000 POKE 53272, (PEEK(53272) AND 240) OR 14

```

Für die Benutzung dieser Methode zur Definition von Zeichen gibt es einige Einschränkungen. Die wichtigste Einschränkung bezieht sich auf den Bedarf an Speicherplatz, den das Programm mit den entsprechenden DATA-Anweisungen benötigt. Wenn nur ein paar Zeichen definiert werden sollen und der Rest des Programms relativ kurz ist, wird der Speicherplatz ausreichen.

Eine gute Möglichkeit, den Speicherplatz zu verringern, ist die Unterteilung eines solchen Programms in zwei Teile. Der erste Teil des Programms definiert den neuen Zeichensatz. Der zweite Teil wird dann in den Computer geladen und benutzt diesen Zeichensatz.

Folgendes Programm lädt einen Zeichensatz, der die Zeichen A-Z des grafischen Zeichensatzes durch entsprechende Zeichen in gotischer Schrift ersetzt. Beachten Sie bitte, daß nicht nur die Zeichen 1-26, sondern auch die Zeichen 129-154 ersetzt werden. Diese Zeichen definieren die Zeichen A-Z im Reverse Modus.

DER GOTISCHE ZEICHENSATZ IST GELADEN WORDEN

.

READY.

≡

"GOTISCHER ZEICHENSATZ"

10 POKE 52,56: POKE 56,56:CLR

20 POKE 56334, PEEK(56334) AND 254

30 POKE 1, PEEK(1) AND 251

40 FOR I=0 TO 511

50 POKE I+14336, PEEK(53248+I)

60 NEXT I: REM UEBERTRAGEN DES ZEICHENSATZES

70 POKE 1, PEEK(1) OR 4

80 POKE 56334, PEEK(56334) OR 1

90 READ CN: IF CN < 0 THEN 1000

100 FOR I=0 TO 7: READ CD

110 POKE 14336+8*CN+I,CD

120 POKE 14336+8*(128+CN)+I,255-CD:NEXT I

130 GOTO 90: REM DEFINIEREN NEUER ZEICHEN

140 POKE 56334, PEEK(56334) OR 1

150 READ CN: IF CN < 0 THEN 1000

160 FOR I=0 TO 7: READ CD

170 POKE 14336+8*CN+I,CD:POKE 14336+8*NEXT I

180 GOTO 150: REM DEFINIEREN NEUER ZEICHEN

200 DATA 1, 48, 72, 20, 34, 62, 34, 65, 0

202 DATA 2, 92, 34, 66, 124, 66, 34, 92, 0

204 DATA 3, 28, 34, 84, 80, 80, 34, 28, 0

206 DATA 4, 88, 100, 66, 66, 66, 100, 88, 0

208 DATA 5, 92, 34, 64, 112, 64, 34, 92, 0

210 DATA 6, 92, 34, 32, 120, 32, 32, 64, 0

212 DATA 7, 28, 34, 64, 94, 98, 62, 2, 6

214 DATA 8, 28, 34, 32, 60, 34, 34, 36, 0

216 DATA 9, 2, 60, 72, 8, 10, 60, 64, 0

218 DATA 10, 1, 2, 2, 2, 34, 68, 56, 0

220 DATA 11, 66, 36, 40, 112, 40, 36, 66, 0

222 DATA 12, 24, 36, 32, 32, 32, 33, 94, 0

224 DATA 13, 84, 42, 42, 106, 42, 42, 64, 0

226 DATA 14, 66, 50, 42, 106, 42, 42, 68, 0

228 DATA 15, 28, 34, 81, 81, 81, 34, 28, 0

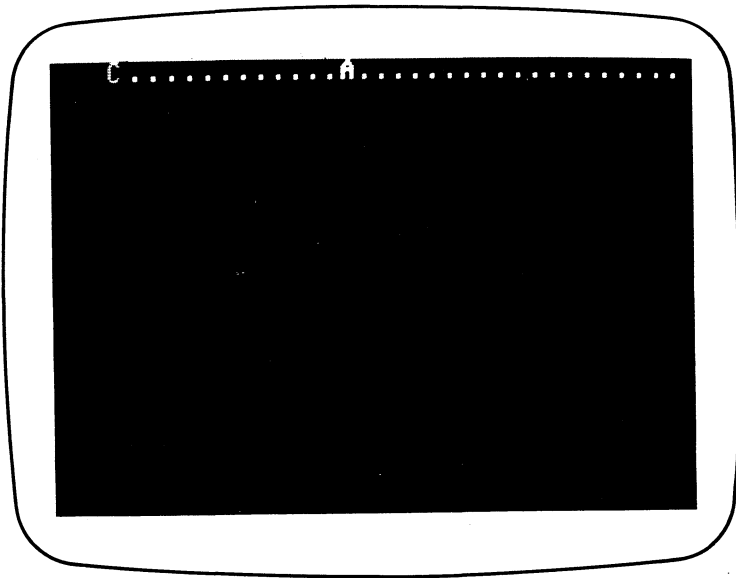
230 DATA 16, 92, 34, 34, 124, 32, 32, 64, 0

232 DATA 17, 56, 84, 34, 2, 12, 26, 124, 0


```
234 DATA 18, 92, 34, 34, 120, 36, 34, 66, 0
236 DATA 19, 2, 60, 64, 60, 2, 60, 64, 0
238 DATA 20, 1, 126, 48, 80, 80, 33, 30, 0
240 DATA 21, 33, 82, 18, 18, 18, 18, 12, 0
242 DATA 22, 76, 178, 34, 34, 34, 20, 8, 0
244 DATA 23, 128, 92, 82, 82, 82, 84, 40, 0
246 DATA 24, 34, 84, 12, 8, 24, 37, 66, 0
248 DATA 25, 66, 164, 36, 36, 26, 66, 60, 0
250 DATA 26, 126, 2, 4, 8, 16, 32, 64, 126
990 DATA -1
1000 POKE 53272, (PEEK(53272) AND 240) OR 14
1010 POKE 53281,1: PRINT"(CLR)"
1020 PRINT"(BLK)DER GOTISCHE ZEICHENSATZ IST GELADEN WORDEN"
1030 NEW
```

Aufgrund der periodischen Änderung des auf dem Bildschirm ausgegebenen Zeichens kann der Eindruck einer Bewegung entstehen. Das nachfolgende Programm wird Ihnen das kurz demonstrieren:

```
1010 S$= "(weiss).....A....."
1020 PRINT"(clear)";S$;"(home)";
1030 FOR I=0 TO 38
1040 PRINT"O(cursor left)";
1050 FOR T=0 TO 100: NEXT T
1060 PRINT"C(cursor left)";
1070 FOR T=0 TO 100: NEXT T
1080 NEXT I
```



In diesem ersten Beispiel wird der normale Zeichensatz benutzt. Das Programm kann allerdings noch wesentlich verbessert werden. Geben Sie zusätzlich zu obigem Programmteil folgende Zeilen ein:

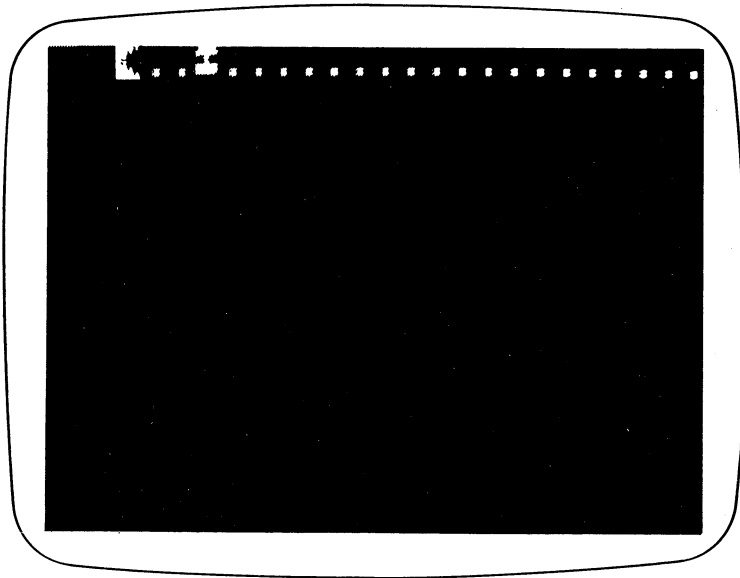
```
10 POKE 52, 56: POKE 56,56:CLR
20 POKE 56334, PEEK(56334) AND 254
30 POKE 1, PEEK(1) AND 251
40 FOR I=0 TO 2047
50 POKE I+14336, PEEK(53248+I)
60 NEXT I
70 POKE 1, PEEK(1) OR 4
80 POKE 56334, PEEK(56334) OR 1
90 READ CN: IF CN < 0 THEN 1000
100 FOR I=0 TO 7: READ CD$
110 BY=14336+8*CN+I: POKE BY,0
120 FOR J=0 TO 7
130 IF MID$(CD$,J+1,1)=" " GOTO 150
140 POKE BY, PEEK(BY) OR 2 ↑ (7-J)
150 NEXT J:NEXT I
160 GOTO 90
170 DATA 1: REM NEUDEFINITION DES ZEICHENS A
171 DATA " ***** "
172 DATA "***** "
173 DATA "* * * "
174 DATA "* * * "
175 DATA "***** "
176 DATA "***** "
177 DATA "* * * * "
178 DATA " "
180 DATA 3: REM NEUDEFINITION DES ZEICHENS C
181 DATA " *** "
182 DATA " ***** "
183 DATA "***** "
184 DATA "** "
185 DATA "***** "
186 DATA " ***** "
187 DATA " *** "
188 DATA " "
190 DATA 15: REM NEUDEFINITION DES ZEICHENS O
191 DATA " *** "
192 DATA " ***** "
```

```
193 DATA "***** "
194 DATA "***** "
195 DATA "***** "
196 DATA " ***** "
197 DATA "  ***  "
198 DATA "      "
200 DATA 46: REM DEFINITION PERIODE
201 DATA "      "
202 DATA "      "
203 DATA "      "
204 DATA "  **  "
205 DATA "  **  "
206 DATA "      "
207 DATA "      "
208 DATA "      "
1000 POKE 53272,30
1010 S$="(weiss).....A....."
1020 PRINT"(CLEAR)";S$;"(HOME)";
1030 FOR I=0 TO 38
1040 PRINT "O(CURSOR LEFT)";
1050 FOR T=0 TO 100: NEXT T
1060 PRINT "C(CURSOR LEFT)";
1070 FOR T=0 TO 100: NEXT T
1080 NEXT I
```

Starten Sie das Programm mit RUN. Beachten Sie den unterschiedlichen visuellen Eindruck dieses Programms im Vergleich zu dem vorangegangenen Programm. Um zu dem normalen Zeichensatz zurückzukehren, müssen Sie den Befehl

POKE 53272,21

eingeben.



Bit Map Grafik

Der Commodore 64 kann insgesamt 64 000 Punkte auf dem Schirm ausgeben. Sie können das leicht nachrechnen. Wie bereits erwähnt, wird jedes Zeichen auf dem Schirm durch eine 8x8 Punktmatrix dargestellt. Da eine Zeile aus 40 Zeichen besteht und der Bildschirm insgesamt 25 Zeilen umfaßt, kann der Computer 320 in der horizontalen Richtung und 200 Punkte in der vertikalen Richtung darstellen. Der Zustand eines jeden Punktes (an oder aus) wird durch ein Bit, das den Wert 0 oder 1 annehmen kann, in dem entsprechenden

Speicherplatz dargestellt. In dem Bit Map Modus kann jeder Punkt auf dem Schirm individuell an- oder ausgeschaltet werden. Die Anzeige des Bildschirms ist somit praktisch ein Abbild eines bestimmten Speicherbereichs im RAM. Da insgesamt 64 000 Punkte auf dem Schirm dargestellt werden und ein Byte 8 Bits enthält, benötigt diese Bit Map einen Block von 8000 Speicherplätzen.

<i>Zeichen Spalte</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>.....</i>	<i>38</i>	<i>39</i>	<i>Vertikale Position</i>
Zeile	0	8	16		304	312	0
	1	9	17		305	313	1
	2	10	18		306	314	2
0	3	11	19		307	315	3
	4	12	20		308	316	4
	5	13	21		309	317	5
	6	14	22		310	318	6
	7	15	23		311	319	7
	320	328	336		624	632	8
	321	329	337		625	633	9
1	322	330	338		626	634	10
	323	331	339		627	635	11
	324	332	340		628	636	12
	325	333	341		629	637	13
	326	334	342		630	638	14
	327	335	343		631	639	15
Zeichenzeile	2-23						
	7680	7688	7696		7984	7992	192
	7681	7689	7697		7985	7993	193
	7682	7690	7698		7986	7994	194
24	7683	7691	7699		7987	7995	195
	7684	7692	7700		7988	7996	196
	7685	7693	7701		7989	7997	197
	7686	7694	7702		7990	7998	198
	7687	7695	7703		7991	7999	199
Horizont.	0-7	7-15	16-23		304-311	312-319	

Abb. 5: Anordnung der Bit Map

Die Speicherplätze sind auf ähnliche Art und Weise organisiert, wie wir es bereits bei den Zeichen kennengelernt haben. Diese Anordnung wird in Abb. 5 dargestellt. Die Anordnung der Speicherplätze kann mit einer Ab-

bildung sicherlich besser als mit vielen Worten erklärt werden.

Wenn die horizontale und vertikale Position eines Pixels in dieser Bit Map als PX und PY bezeichnet wird, kann eine Gleichung abgeleitet werden, die den entsprechenden Speicherplatz ML, der das betreffende Bit enthält, berechnet. Die Zeichenzeile RW und die Zeichenspalte CL des Speicherplatzes erhält man, indem man die horizontalen und die vertikalen Koordinaten dividiert. Die Quotienten entsprechen der Zeichenzeile und der Zeichenspalte.

$$CL = \text{INT}(PX/8)$$

$$RW = \text{INT}(PY/8)$$

Der erste Speicherplatz in einem Zeichenblock einer spezifischen Zeile und Spalte (FML) ist mit folgenden Gleichungen gegeben:

$$FML = 8*CL + 320*RW$$

$$FML = 8*(CL + 40*RW)$$

Zu dem ersten Speicherplatz des Zeichenblocks muß ein Offset addiert werden, um die Speicherzelle zu erhalten, die das gewünschte Pixel beinhaltet. Der Offset OY ist der Rest bei der ganzzahligen Division von PY durch 8. Folgende Gleichungen bestimmen OY.

$$OY = PY - 8*\text{INT}(PY/8)$$

$$OY = PY - 8*RW$$

Der Speicherplatz, der das Pixel bei PX und PY enthält, ist mit dem ersten Speicherplatz des Zeichenblocks zuzüglich des Offsets identisch. Durch arithmetische Umformung und durch Substitution des Wertes OY erhält man folgende Gleichungen:

$$ML = 8*(CL + 40*RW) + OY$$

$$ML = 8*(CL + 40*RW) + PY - 8*RW$$

$$ML = 8*(CL + 39*RW) + PY$$

Werden in diesem Gleichungssystem CL und RW ersetzt, erhält man die Gleichung

$$ML = 8 * \text{INT}(PX/8) + 39 * (\text{INT}(PY/8)) + PY$$

Man nimmt dabei an, daß die Bit Map bei Adresse 0 beginnt. Die Startadresse kann jedoch auch variiert werden, indem ein sogenanntes Offset zu der Startadresse addiert wird.

$$ML = SA + 8 * \text{INT}(PX/8) + 39 * (\text{INT}(PY/8)) + PY$$

Der Platz des Bits in dem Byte, das das Pixel kontrolliert, ist der Rest der ganzzahligen Division von PX durch 8. Dieser Rest ist der Offset von rechts.

$$\text{offset von rechts} = PX - 8 * \text{INT}(PX/8)$$

So ist beispielsweise der ganzzahlige Rest bei der Division durch 8 gleich 3, wenn PX den Wert 19 hat. Das Bit, das gesetzt werden muß, ist vom linken Bit aus das dritte Bit nach rechts. Das linke Bit trägt die Nummer 7. In dem Beispiel ist das dritte Bit von rechts das Bit Nummer 4. Die Anzahl der Bits, die gesetzt oder gelöscht werden müssen, ist für eine spezielle Position PX durch folgende Gleichung gegeben.

$$BT = 7 - (PX - 8 * \text{INT}(PX/8))$$

Ein Pixel wird eingeschaltet, indem der bestehende Inhalt der Speicherzelle durch OR mit $2^{\text{hoch Bit Nummer}}$ verknüpft wird. Ein Pixel wird ausgeschaltet, indem der bestehende Wert der Speicherzelle durch AND mit dem 2er Komplement von $2^{\text{hoch Bit Nummer}}$ verknüpft wird. Die Methode, ein Punkt der Bit Map auszugeben oder zu löschen, kann wie folgt zusammengefaßt werden:

1. Suchen Sie den Speicherplatz und das Bit, welches den Pixelkoordinaten PX und PY entspricht.

$$ML = SA + 8 * (\text{INT}(PX/8) + 39 * \text{INT}(PY/8)) + PY$$

$$BT = 7 - (PX - 8 * \text{INT}(PX/8))$$

2. Geben Sie den Punkt aus.


```
POKE ML, PEEK(ML) OR (2↑BT)
```

oder löschen Sie einen Punkt:

```
POKE ML, PEEK(ML) AND NOT (2↑BT)
```

Die Farben des Vordergrundes (Pixel an) und des Hintergrundes (Pixel aus) werden durch eine Farb-Map bestimmt. Diese Farb-Map darf nicht mit dem Farb-RAM, der im Zeichensatzmodus verwendet wird, verwechselt werden. Diese Map ist ein 1K großer Speicherbereich des RAM. Der Inhalt eines jeden Bytes der Farb-Map definiert auf dem Schirm die Vordergrund- und Hintergrundfarbe eines 8x8 großen Bereiches. Die Position jedes Bereiches ist mit der Position, die ein Zeichen auf dem Schirm einnimmt, identisch. Ein Farb-Byte ist in zwei Teile unterteilt. Die Bits 0-3 definieren die Bildschirmfarbe und die Bits 4-7 definieren die Vordergrundfarbe. Alle 16 Farben sind in jeder Kombination verfügbar. Die einzige Einschränkung besteht darin, daß nicht mehr als zwei Farben innerhalb eines 8x8 Bereiches benutzt werden dürfen. Tabelle 8 gibt die Werte für alle Farbkombinationen an.

Der Bit Map Modus wird durch das Bit 5 der Speicherzelle 53265 bestimmt. Das Kommando, mit dem die Bit Map eingeschaltet wird, ist

```
POKE 53265, PEEK(53265) OR 32
```

Die Bit Map wird mit dem Befehl

```
POKE 53265, PEEK(53265) AND NOT 32
```

ausgeschaltet.

Das folgende Programm demonstriert den Bit Map Grafikmodus. Der Bereich, der für die Bit Map benutzt wird, muß zunächst geschützt werden, damit er nicht durch BASIC zerstört wird. Dazu müssen das Ende von BASIC und der Start der String Pointer geändert werden.

```
10 POKE 52,32:POKE 56,32: CLR
```

Anschließend werden die Speicherzellen für die Bit Map und für die Farb-

Tabelle 8: Farb Bytes für BITMAP Modus
Die Spalten geben die Vordergrundfarbe; die Zeilen die Hintergrundfarbe

	schwarz	weiß	rot	türkis	violett	grün	blau	gelb	orange	braun	grau 1	grau 2	hellrot	hellgrün	hellblau	grau 3
schwarz	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
weiß	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
rot	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
türkis	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
violett	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
grün	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
blau	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
gelb	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
orange	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
braun	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
hellrot	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
grau 1	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
grau 2	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
hellgrün	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
hellblau	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
grau 3	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

map ausgewählt. Die Bit Map wird in dem Bereich von 8192 bis 16383 und die Farbmap in dem Bereich von 1024 bis 2023 gespeichert. Danach wird die Bit Map angeschaltet.

```
20 SA = 8192:SC=1024
30 POKE 53272,24: REM SPEICHERBEREICH FUER BIT MAP UND FARBMAP
40 POKE 53265, PEEK(53265) OR 32: AUSSCHALTEN BITMAP
```

Die Farbmap wird nun mit Einsen gefüllt. Man erhält dadurch einen weißen Schirm und schwarze Punkte. Die Bit Map wird gelöscht, indem die entsprechenden Speicherplätzen mit Nullen gefüllt werden. Diese beiden Tätigkeiten werden als Unterprogramme geschrieben.

```
50 GOSUB 230: REM FARBE DES SCHIRMS
60 GOSUB 250: LOESCHEN DES SCHIRMS
230 FOR I=SC TO SC+999: POKE I,1: NEXT I
240 RETURN
250 FOR I=SA TO SA+7999:POKE I,0:NEXT I
260 RETURN
```

Der erste Punkt wird in die Mitte der Bit Map gezeichnet. Dieses Programm wird ebenfalls als Unterprogramm geschrieben.

```
70 PX=159:PY=99:GOSUB 270:REM AUSGABE DES PUNKTES
270 ML=SA+8*(INT(PX/8)+39*INT(PY/8)) + PY
280 BT = 7-(PX-8*INT(PX/8))
290 POKE ML, PEEK(ML) OR (2 ↑ BT)
300 RETURN
```

Nachdem der erste Punkt in der Mitte der Bit Map ausgegeben ist, liest das Programm den Zustand des Joysticks auf Port 2. Wenn irgendeine Eingabe mit dem Joystick erfolgt, gibt das Programm einen neuen Punkt aus und liest den Zustand des Joystick Ports erneut.

```
80 GOSUB 110: IF J=15 THEN 80: REM EINLESEN JOYSTICK
90 GOSUB 270
100 GOTO 80
```

```

110 J=PEEK(56320) AND 15
120 IF (J AND 8)=0 THEN PX=PX+1: REM RECHTS
130 IF (J AND 4)=0 THEN PX=PX-1: REM LINKS
140 IF (J AND 2)=0 THEN PY=PY+1: REM UNTEN
150 IF (J AND 1)=0 THEN PY=PY-1: REM OBEN
160 IF PY > 199 THEN PY=199
170 IF PY < 0 THEN PY=0
180 IF PX > 319 THEN PX=319
190 IF PX < 0 THEN PX=0
200 GET A$: IF A$= "(CLR)" THEN GOSUB 250: GOSUB 270:REM BILDSCHIRM
    LOESCHEN
210 IF A$="(F7)" THEN 310: REM ABRUCH DURCH FUNKTIONSTASTE F7
220 RETURN

```

Wenn die CLR/HOME-Taste und die SHIFT-Taste gleichzeitig gedrückt werden, wird die Bit Map gelöscht. Durch Betätigen der Funktionstaste F7 wird die Bit Map ausgeschaltet und das Programm beendet.

```

310 POKE 53272,21: REM NORMALZUSTAND BILDSCHIRM UND ZEICHEN
320 POKE 53265, PEEK(53265) AND NOT 32: REM AUSSCHALTEN BIT MAP
330 END

```

Die Beschreibung der hochauflösenden Grafik ist in diesem Punkt ziemlich eingeschränkt. Jedes BASIC-Programm ist nämlich auf 6K hinsichtlich der Größe begrenzt. Außerdem sind die Befehle, die den Umgang mit der Bit Map erlauben, sehr schwer zu handhaben. Diese Einschränkungen können durch Benutzung des Programms 'MINIGRAPH' umgangen werden. Das Programm lädt einen Satz von in Maschinensprache geschriebenen Routinen in den Arbeitsspeicherbereich 36883-37681. Damit werden Tätigkeiten wie das Einschalten der Bit Map und das Zeichnen von Punkten zu einfachen Kommandos reduziert. Die Bit Map wird außerhalb des BASIC-Programms gespeichert. Insgesamt stehen noch 33K RAM Speicher für BASIC Programme zur Verfügung. Die Routinen werden sehr schnell ausgeführt, da sie in Maschinensprache geschrieben sind. Wenn Sie dieses Programm eintippen, sollten Sie darauf achten, daß sämtliche Zahlen der DATA-Anweisungen richtig eingegeben werden. Es ist ganz wichtig, daß Sie hier keinen Fehler machen. Eine falsche Zahl bringt den Computer ins 'Niemandland'. Wenn das passiert, bleibt nur

die Möglichkeit, den Rechner kurz auszuschalten. Das bedeutet natürlich auch, daß das gerade im Speicher befindliche Programm gelöscht wird.

Geben Sie das Programm ein, und speichern Sie es auf der Datensette oder der Diskette ab. MINIGRAPH ist ein Hilfsprogramm (= utility program), das geladen und ausgeführt wird, bevor irgendein anderes Programm geladen wird, das diese in MINIGRAPH enthaltenen Routinen benutzt. Die Routinen werden mit dem SYS-Kommando aufgerufen. Ein SYS-Kommando bewirkt, daß der Computer zu einer bestimmten Speicheradresse springt und das Maschinenprogramm ausführt, welches dort gespeichert ist. Nachdem diese Routine ausgeführt worden ist, wird wieder in das BASIC-Programm zurückgesprungen. Die Bit-Map kann mit folgenden Kommando eingeschaltet werden:

SYS 50195

Die Bit Map wird mit dem Befehl

SYS 50198

ausgeschaltet. Mit dem Befehl

SYS 50207

wird die Bit Map gelöscht, d. h. alle Bits werden auf Null gesetzt. Die Farben werden mit

SYS 50210,F,B

gesetzt. F und B sind Zahlen oder Variablen, deren Wert zwischen 0 und 15 liegt. Die erste Zahl (F) setzt die Vordergrundfarbe. B setzt dementsprechend die Hintergrundfarbe. Ein Punkt wird mit

SYS 50201,X,Y,M

geplottet. X und Y sind die horizontale bzw. vertikale Position des Punktes. Die horizontale Position kann von 0 bis 319 und die vertikale Position von 0 bis 199 angegeben werden. Die obere linke Ecke hat die Koordinaten 0,0. Die letzte Variable (M) setzt den Plot-Modus. Wenn der Wert

von M 1 ist, wird ein Punkt in der Vordergrundfarbe ausgegeben. Ist M 2, wird der Punkt in der Hintergrundfarbe ausgegeben. Für M gleich 0 wird die Farbe des Punktes geändert. Wird der Punkt an der entsprechenden Stelle in der Vordergrundfarbe angezeigt, nimmt er dann die Hintergrundfarbe an und umgekehrt. Ein Strich wird mit dem Kommando

SYS 50204,X,Y,M

gezeichnet. X und Y sind die horizontale und vertikale Position des Endpunktes der Linie. Der Startpunkt der Linie wird entweder mit dem Befehl für das Plotten eines Punktes oder durch den Endpunkt der letzten Linie, die gezeichnet wurde, angegeben. Der Zeichenmodus wird wie bei der Routine für die Ausgabe eines Punktes gesetzt. Werte, die außerhalb des erlaubten Bereiches für X, Y, M, F und B liegen, erzeugen den Fehler ILLEGAL QUANTITY ERROR.

Es muß beachtet werden, daß das Kommando zum Zeichnen einer Linie immer entweder die letzte Instruktion in einer BASIC Programmzeile sein muß, oder daß es in einer separaten Zeile steht. Ist das nicht der Fall, erhalten Sie eine Fehlermeldung. Wenn Sie den Fehler SYNTAX und ILLEGAL ERROR in einem Programm erhalten, das MINIGRAPH enthält, sollten Sie nachsehen, ob es sich um den oben beschriebenen Fehler handelt. Das Programm, welches nach MINIGRAPH folgt, zeigt, wie die Routinen von MINIGRAPH benutzt werden, um verschiedene geometrische Darstellungen zu zeichnen. Sie können das Programm beenden, indem Sie irgendeine Taste drücken.

MINIGRAPH.DAT

COMMODORE 64

```

10 PRINT"(CLR)MINIGRAPH LOADER VON PAUL SCHATZ"
20 PRINT "BITTE WARTEN..."
30 A=0: FOR I=50195 TO 50975: READ BY
40 POKE I, BY: A=A+BY: NEXT I
50 IF A <> 95619 THEN PRINT "FEHLER IN DATEN STATEMENTS":STOP
60 PRINT"(CLR) MINIGRAPH BEFEHLSSATZ"
70 PRINT"(CRSR DOWN) EINSCHALTEN BIT MAP           -SYS 50195"
80 PRINT"AUSSCHALTEN BIT MAP                       -SYS 50198"
90 PRINT"BIT MAP LOESCHEN"                         -SYS 50207"
100 PRINT"FARBE BIT MAP                           -SYS 50210,F,B"

```

```

11Ø PRINT"AUSGABE EINES PUNKTES @ X,Y      -SYS 5Ø2Ø1,X,Y,M"
12Ø PRINT"ZEICHNE EINE LINIE NACH X,Y      -SYS 5Ø2Ø4,X,Y,M"
13Ø PRINT"(CRSR DOWN)F = VORDERGRUND      B = HINTERGRUND
14Ø PRINT"X = Ø TO 319          Y = Ø TO 199"
15Ø PRINT"M = Ø, 1, OR 2 (FLIP, DRAW, OR ERASE)
16Ø NEW
20Ø DATA 76,244,198,76, 7, 199, 76, 37, 196, 76,114,197, 76, 3Ø, 197,76,66
21Ø DATA 197, 169, Ø,141,4,196,32,253,174,32,235,183,224,2ØØ,144,3,76
22Ø DATA 26,199,142,3,196, 166,2Ø, 165,21, 24Ø,8 ,2Ø1,1,2Ø8,24Ø,224, 64
23Ø DATA 176,236,141,2, 196,142,1,196,32,253,174,32,158,183,224,3,176
24Ø DATA 22Ø,142,Ø, 196,173, Ø, 196,24Ø, 27, 74,144,12,32,174,196,32,135
25Ø DATA 196,29, 158,196,145,251, 96, 32,174,196,32,135,196,61,166,196,145
26Ø DATA 251,96, 32,174,196,32,135,196,93,158,196,145,251,96,173,1,196
27Ø DATA 41,7,17Ø 12Ø,16Ø,52,132,1,16Ø, Ø, 177,251,16Ø,55,132,1,88
28Ø DATA 16Ø, Ø,96,128,64,32,16,8,4, 2, 1, 127,191,223,239,247,251
29Ø DATA 253,254,269, Ø, 133,251,169,224,133,252,173,1,196,41,248,24,1Ø1
30Ø DATA 251,133,251,173,2,196, 1Ø1, 252,133,252, 173, 3, 196,72,41,7,24
31Ø DATA 1Ø1, 251,133,251,144, 2, 23Ø, 252, 1Ø4, 74, 74, 74, 1Ø, 17Ø, 189,
    236, 196
32Ø DATA 24, 1Ø1, 251,133,251,189,237,196,1Ø1,252,133,252,96, Ø,Ø,64,1
33Ø DATA 128, 2, 192, 3, Ø, 5, 64, 6, 128, 7, 192, 8, Ø, 1Ø, 64,11,128
34Ø DATA 12, 192,13,Ø, 15,64,16,128,17, 192,18, Ø, 2Ø,64,21,128,22
35Ø DATA 192, 23, Ø,25,64,26,128,27,192,28,Ø, 3Ø,162,32,169,224,133
36Ø DATA 252,169,Ø,133,251,168,145,251,2ØØ, 2Ø8,251,23Ø,252,2Ø2,2Ø8,
    246,96
37Ø DATA 32,253,174,32,158,183,224,16,144,3,76,26,199,96,169,192,133
38Ø DATA 252,169,Ø, 133,251,32,52,197,138,1Ø,1Ø,1Ø,1Ø,141,9,196,32
39Ø DATA 52,197,138,13,9,196,162,2,16Ø,Ø, 145,251,2ØØ, 2Ø8,251, 23Ø,252
40Ø DATA 2Ø2, 16,246,145, 251,2ØØ, 192,232,144,249,96,169,Ø, 141,8, 196,
    141
41Ø DATA 1Ø, 196, 32,253,174,32,235,183,224,2ØØ, 144,3, 76, 26,199,142,7
42Ø DATA 196,166, 2Ø, 165, 21, 24Ø, 8, 2Ø1,1,2Ø8,24Ø,224,64,176,236,141,6
43Ø DATA 196,142, 5, 196, 32, 253,174,32,158,183,2Ø1,3, 176, 22Ø, 142, Ø,
    196
44Ø DATA 173, 5, 196, 56, 237, 1, 196, 141, 12, 196, 173, 6, 196, 237, 2,
    196,141
45Ø DATA 13, 196, 16, 2Ø, 2Ø6, 1Ø, 196,56, 169,Ø,237,12,196,141,12,196,169

```

460 DATA 0, 237,13,196,141,13, 196, 169, 0, 141, 11, 196,173,7, 196,56,
 237
 470 DATA 3,196,141, 14,196,173,8,196,237,4,196,141,15,196,16, 20,206
 480 DATA 11,196, 56,169, 0,237,14,196,141,14,196,169,0,237,15,196,141
 490 DATA 15,196,169, 0,141,18,196,173,14,196,56,237,12,196,173,15,196
 500 DATA 237, 13,196,144,27,174,14,196,173,12,196,141,14,196,142,12,196
 510 DATA 174,15,196,173,13,196, 141, 15,196,142,13,196,206,18,196,173,12
 520 DATA 196,141,16,196,173,13,196,141,17,196,32,91, 196, 173,18,196,208
 530 DATA 18,173,1,196,205, 5,196,208,27,173,2, 196,205, 6, 196,208,19
 540 DATA 240, 16,173,3,196,205, 7,196,208,9,173,4,196, 205, 8,196,208
 550 DATA 1, 96,173,18,196,208, 6,32,192,198,76,118,198,32,218,198,32
 560 DATA 152,198,32,152,198,16,20,173,18,196,208,6,32,218, 198, 76,140
 570 DATA 198, 32,192,198,32,172,198,32,172,198,32,91,196,76,64,198,173
 580 DATA 16,196, 56,237, 14,196,141,16,196,173,17, 196, 237,15,196,141,17
 590 DATA 196,96, 173,16,196,24,109,12, 196,141, 16, 196, 173,17,196,109,13
 600 DATA 196,141,17,196,96,173,10,196,208, 9,238,1, 196,208, 3, 238,2
 610 DATA 196,96, 173, 1,196,208,3,206,2,196,206, 1,196,96,173,11,196
 620 DATA 208, 9,238,3,196,208,3,238, 4,196,96,173,3,196,208,3,206
 630 DATA 4,196,206,3,196,96,173,0, 221,41,252,141,0, 221,169, 59,141
 640 DATA 17,208,169,8,141,24,208,96,173,0,221,9,3,141, 0,221,169
 650 DATA 27, 141,17,208,169,21,141,24,208,96,32,7,199,76, 72, 178

MINIGRAPH BEFEHLSSTZ

EINSCHALTEN BIT MAP	-SYS 50195
AUSSCHALTEN BIT MAP	-SYS 50198
BITMAP LOESCHEN	-SYS 50207
FARBE BITMAP	-SYS 50210,F,B
AUSGABE EINES PUNKTES	-SYS 50201,X,Y,M
ZEICHNE EINE LINIE NACH X,Y	-SYS 50204,X,Y,M
F = Vordergrund	B = Hintergrund
X = 0 TO 319	Y = 0 TO 199
M = 0, 1, OR 2 (FLIP, DRAW, OR ERASE)	

READY.

MINIGRAPH.DEMO

COMMODORE 64

```
10 PRINT"(CLR)MINIGRAPH DEMO"
20 PRINT"VON PAUL SCHATZ"
30 GOSUB 1000: REM PAUSE
40 SYS 50207: REM LOESCHEN BIT MAP
50 SYS 50210,1,2: REM FARB BIT MAP
60 SYS 50195:REM BIT MAP EINSCHALTEN
70 N1=30: N2=33: REM ZEICHNEN VON QUADRATEN
80 FOR I=0 TO 12:X1=N1-2*I:X2=N2+I (UP ARROW) 2
90 SYS 50201,X1,X1,1: REM ZEICHNEN PUNKT
100 SYS 50204,X2,X1,1
110 SYS 50204,X2,X2,1
120 SYS 50204,X1,X2,1
130 SYS 50204,X1,X1,1
140 NEXT I
150 GOSUB 1000
160 SYS 50210, 5, 1
170 SYS 50207
180 FP=1:A1=77: B1=23: DL=9: C=0
190 FOR I=0 TO 40: C =C+DL: A=A1*C*/180: B=B1*C*/180
200 X = INT(100*SIN(A)+160.5)
210 Y=INT(80 * COS(B)+100.5)
220 IF FP=1 THEN FP=0: SYS 50201,X,Y,0:GOTO 240
230 SYS 50204, X, Y, 0
240 NEXT I
250 GOSUB 1000
260 SYS 50210, 7, 2
270 SYS 50207
280 X1 = 0: X2 = 319
290 FOR I=0 TO 199
300 Y1 =I: Y2 = 199-I
310 SYS 50201, X1,Y1,0
320 SYS 50204, X2,Y2, 0
330 NEXT I
340 Y1 =199: Y2=0
350 FOR I=0 TO 319
```

```
360 X1=I: X2=319-I
370 SYS 50201, X1, Y1, 0
380 SYS 50204, X2, Y2, 0
390 NEXT I
400 GOSUB 1000
410 SYS 50210, 0, 10
420 SYS 50207
430 FP=1:A1=13: B1=26: DL=6: C=0
440 FOR I=0 TO 80: C =C+DL: A=A1*C*/180: B =B1*C*/180
450 X = INT(100*COS(A)+160.5)
460 Y = INT(80*SIN(B) +100.5)
470 IF FP=1 THEN FP=0: SYS 50201,X,Y,1:GOTO 240
480 SYS 50204, X, Y, 1
490 NEXT I
500 GOSUB 1000
510 SYS 50210, 15, 12
520 SYS 50207
530 FOR X=0 TO 319
540 Y = INT(100+70*SIN(X/10))
550 SYS 50201, 160, 10,1
560 SYS 50204, X, Y, 1
570 NEXT X
900 GET A$: IF A$="" THEN 30
910 SYS 50198
920 END
1000 FOR I=0 TO 2000:NEXT I: RETURN
```

Ein paar abschließende Bemerkungen

Dieses Kapitel ist eine kurze Einführung in die Grafikmöglichkeiten des Commodore 64, die Sie nun sicherlich etwas besser verstehen. Sie werden außerdem eigene komplexere Programme schreiben können. In Tabelle 9 sind alle 47 Speicherplätze aufgeführt, die den VIC Chip kontrollieren. Die letzte Spalte gibt den Grafikmodus an, für den der entsprechende Speicherplatz verantwortlich ist: S steht für Sprites, B für Bit Map Modus und C für Character-Modus (= Zeichenmodus). Diese Tabelle wird sehr nützlich sein, wenn Sie sich weiter mit der Grafik für den Commodore 64 beschäftigen wollen.

Tabelle 9: Video Interface Controller Speicherplätze

Speicherplatz		Video Operation	
53248		Sprite 0 horizontale Position	S
53249		Sprite 0 vertikale Position	S
53250		Sprite 1 horizontale Position	S
53251		Sprite 1 vertikale Position	S
53252		Sprite 2 horizontale Position	S
53253		Sprite 2 vertikale Position	S
53254		Sprite 3 horizontale Position	S
53255		Sprite 3 vertikale Position	S
53256		Sprite 4 horizontale Position	S
53257		Sprite 4 vertikale Position	S
53258		Sprite 5 horizontale Position	S
53259		Sprite 5 vertikale Position	S
53260		Sprite 6 horizontale Position	S
53261		Sprite 6 vertikale Position	S
53262		Sprite 7 horizontale Position	S
53263		Sprite 7 vertikale Position	S
53264	Bits 0-7	Byte (high) der horizontalen Position	S
53265	Bits 0-2	Vertikale Position Scrolling	BC
	Bit 3	Auswahl der Anzahl der Zeilen (24 oder 25)	BC

Speicherplatz

Video Operation

	Bit 4	Leerer Bildschirm	
	Bit 5	Auswahl Bit Map Modus	B
	Bit 6	Auswahl des Farbmodus für Hintergrund	C
	Bit 7	Bit für Rasterposition	
53266		Raster Position	
53267		Horizontale Position des Lichtgriffels	
53268		Vertikale Position des Lichtgriffels	
53269	Bit 0-7	Ein- und Ausschalten der Sprites 0-7	S
53270	Bit 0-2	Horizontale Position für Scrolling	BC
	Bit 3	Auswahl der Spaltennummer (38 oder 40)	BC
	Bit 4	Auswahl des Multicolor Modus	BC
53271	Bit 0-7	Höhenverdopplung - Sprites 0-7	S
53272	Bit 1-3	Zeiger für den Zeichensatz	C
	Bit 3	Bit Map Zeiger	B
	Bit 4-7	Zeichen Bildschirm oder Farbmap	
		Zeiger	BC
53273		Interrupt Register	
53274		Interrupt Register	
53275	Bit 0-7	Prioritätsregister für Sprites 0-7	S
53276	Bit 0-7	Auswahl Multicolor Sprites 0-7	S
53277	Bit 0-7	Spriteverdopplung Breite 0-7	S
53278	Bit 0-7	Sprite mit Spritekollision	S
53279	Bit 0-7	Sprite mit Zeichenkollision	S
53280	Bit 0-3	Rahmenfarbe	BC
53281	Bit 0-3	Hintergrundfarbe	C
53282	Bit 0-3	Zeichen Multicolor 1	C
53283	Bit 0-3	Zeichen Multicolor 2	C
53284	Bit 0-3	Sprite Multicolor 1	S
53285	Bit 0-3	Sprite Multicolor 2	S
53286	Bit 0-3	Sprite 0 Farbe	S
53287	Bit 0-3	Sprite 1 Farbe	S
53288	Bit 0-3	Sprite 2 Farbe	S
53289	Bit 0-3	Sprite 3 Farbe	S
53290	Bit 0-3	Sprite 4 Farbe	S
53291	Bit 0-3	Sprite 5 Farbe	S

Speicherplatz

Video Operation

53292 Bit 0-3

Sprite 6 Farbe

S

53293 Bit 0-3

Sprite 7 Farbe

S

Anhang 3: Musik mit dem Commodore 64

(Dr. Frank H. Covitz)

Von den fünf menschlichen Sinnesorganen ist das Gehör nach den Augen das zweitwichtigste. Das Sprichwort 'Ein Bild sagt mehr als 1000 Worte' ist sicherlich nicht falsch. Die Informationsmenge, die vom Ohr empfangen und im Gehirn weiterverarbeitet wird, ist enorm. Denken Sie nur daran, wie lang eine einzige Minute sein kann: Die gesprochene Information, die Erkennung der Identität des Sprechers etc. sind sehr wichtige Informationen, die Sie aufnehmen. Das, was wir mit dem Gehör wahrnehmen können, ist sehr vielfältig. Es reicht von den künstlich erzeugten Tönen der Musikinstrumente bis zu den akustischen Signalen, die von der Natur ausgesendet werden (z. B. der Gesang eines Kanarienvogels, das Rauschen der Brandung...)

Ein Bild kann mit Worten beschrieben werden. Sie werden sicherlich einige Schwierigkeiten bekommen, wenn Sie ein akustisches Empfinden mit Worten beschreiben wollen. Der SID (Sound Interface Device) Chip des Commodore 64 ermöglicht es Ihnen, eine große Anzahl akustischer Varianten mit HIFI Qualität zu erzeugen. Bevor wir auf die Programmierung näher eingehen, wollen wir uns zunächst mit einigen Grundkenntnissen beschäftigen.

Das bedeutet, daß wir uns zunächst einmal über Schwingungen unterhalten müssen. Die Schwingungen eines Gegenstandes werden durch die Luft weitergeleitet und erreichen schließlich unser Ohr. Dort werden sie in Reize umgewandelt, die von dem Gehirn entschlüsselt werden. Wenn die Schwingungen eine sehr niedrige Frequenz haben, können wir sie nicht nur hören, sondern auch fühlen: Halten Sie Ihre Hand vor einen großen Lautsprecher einer Rock-Gruppe! Bevor wir ein Geräusch mit dem SID erzeugen, sollten

wir die Einzelheiten der Schwingungen verstehen.

Akustische Schwingungen können als Druckunterschiede der Luft verstanden werden. Diese Schwingungen werden direkt von unserem Ohr wahrgenommen. Andere Schwingungen können wir allerdings nicht direkt hören. Denken Sie beispielsweise an die Aufnahmen Ihres Kassettenrecorders oder Ihres Tonbandgerätes. Die in einer Schallplatte eingezeichneten Schwingungen sind nicht direkt hörbar. Solche Schwingungen müssen erst durch eine elektronische Schaltung und einen Verstärker auf einen Lautsprecher übertragen werden. Für unseren speziellen Zweck müssen wir dem Computer die Information über die akustische Schwingung in Form von Zahlen übermitteln. Was diese Zahlen bedeuten und wie aus ihnen ein Ton erzeugt wird, werden wir später sehen.

Wir wollen zunächst zwei Fragen beantworten: 1) Welches sind die grundlegenden Eigenschaften, die die Schwingungen als Musik erscheinen lassen? 2) Wie können wir diese Eigenschaften benutzen, um einen bestimmten Ton zu erzeugen?

Der Ton

Wir wollen uns jetzt mit dem Grundelement der Musik - dem Ton - beschäftigen. Wir gehen zunächst davon aus, daß ein Ton ein gleichbleibendes akustisches Signal mit einer bestimmten Höhe ist. Unter 'gleichbleibend' verstehen wir, daß der Ton sich im Verlauf der Zeit nicht ändert. Ausgenommen sind natürlich der Anfang und das Ende eines Tones.

Die Frequenz ist eine Zahl, die angibt, wieviele Schwingungen in einer Sekunde stattfinden. Die Einheit der Frequenz ist deshalb Schwingungen pro Sekunde. Als Abkürzung hierfür wird zu Ehren von Heinrich Rudolph Hertz, der sich auf dem Gebiet der elektromagnetischen Schwingungen hervorgetan hat, der Name Hertz verwendet. Die Abkürzung für Hertz ist Hz.

Da der Begriff 'Hertz' für das weitere Verständnis sehr wichtig ist, wollen wir uns einige Beispiele ansehen. Denken Sie beispielsweise an die Bewegung eines Pendels in der großen Standuhr Ihres Großvaters. Die Frequenz dieser Schwingungen sollte exakt ein halbes Hertz sein, da jede halbe Schwingung genau eine Sekunde dauern soll. In diesem Fall können Sie die Schwingungen direkt sehen und zählen. Bei den akustischen Signalen ist die Frequenz viel zu hoch, um die einzelnen Schwingungen sehen zu können.

Sie können das allerdings mit Hilfe eines Oszillographen möglich machen. Führen Sie ein kleines Experiment durch. Legen Sie ein längeres Messer auf die Tischkante, biegen Sie die Klinge, und lassen Sie sie dann los. Sie werden sehen können, wie die Klinge mit einer Frequenz von ca. 10 Hz schwingt.

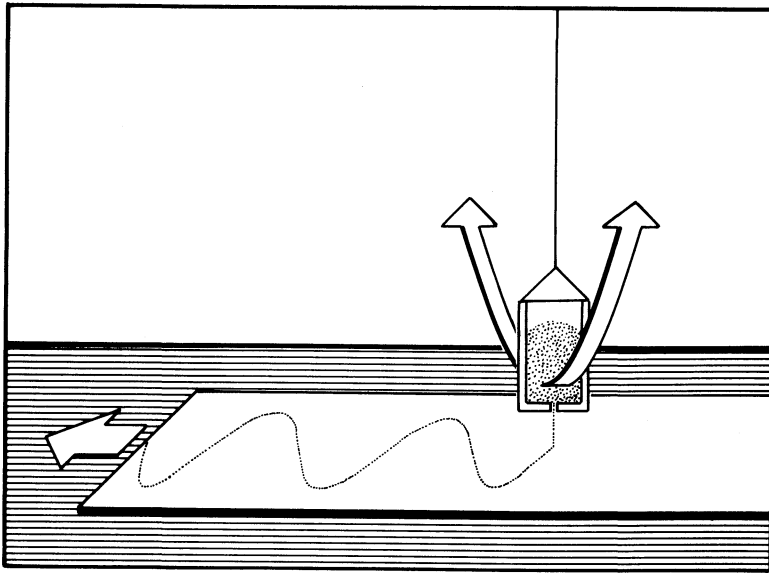
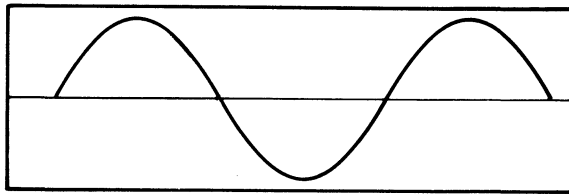
Die Hörfähigkeit des menschlichen Ohres ist begrenzt. Es ist möglich, Schwingungen zwischen 20 und 20 000 Hz zu hören. Der Durchschnittsmensch ist allerdings in der Lage, Frequenzen von 30 bis 15 000 Hz wahrzunehmen. Sie können die niedrigen Frequenzen direkt fühlen. Moment! Vorhin haben wir gesagt, daß wir nur Frequenzen, die über 20 Hz liegen, hören können. Wie ist es aber möglich, daß wir die Schwingungen des Messers hören können, obwohl sie nur 10 Hz betragen? Was Sie wirklich hören, ist das Schlagen der Klinge gegen den Tisch mit ca. 10 Schlägen pro Sekunden. Sie können jeden dieser Schläge hören.

Die Einzelheiten der Schwingungslehre sind in Wirklichkeit ein wenig komplizierter. Wir wollen uns zunächst aber nur mit den einfachen Dingen beschäftigen, die den sogenannten 'reinen' Ton erklären.

Die Schwingung des Pendels verläuft ohne 'Störung', da sich weder die Geschwindigkeit, noch die Richtung des Pendels abrupt ändern. Der Ton einer Stimmgabel kommt dem reinen Ton sehr nahe. Obwohl diese reinen Töne in der Natur nur sehr selten vorkommen, müssen wir sie verstehen, da jeder Ton aus einer Mischung reiner Töne mit der entsprechenden Frequenz und Amplitude konstruiert werden kann. Da reine Töne sehr einfach durch elektronische Schaltungen erzeugt werden können, bilden sie die Basis für künstliche Synthesizer und zur Analyse von Geräuschen.

Um diesen Punkt richtig zu verstehen, wollen wir ein langsames Pendel betrachten. In diesem Fall nehmen wir aber kein Uhrpendel, sondern wir betrachten einen kleinen Sandsack, der unten ein Loch hat. Dieser Sandsack ist an einem Faden aufgehängt. Der Sand, der durch das Loch rieselt, hinterläßt auf einem Blatt Papier, das wir mit gleichbleibender Geschwindigkeit unter dem Sandsack herziehen, ein Muster. Dieses Muster sieht so aus, wie Sie es in Abbildung 1 sehen können.

Grafisch dargestellt, sähe die Sandspur wie in Abb. 2 dargestellt aus. Die Bewegung des kleinen Sandsackes ist durch zwei Größen bestimmt. Eine Größe ist dabei die Frequenz. Wie wir bereits gesehen haben, gibt die Frequenz an, wie oft der Sandsack pro Sekunde hin und her schwingt. Die Schwingungen bewegen sich dabei um einen 'Mittelpunkt'. Die größte Ent-

*Abb. 1: Sandpendel**Abb. 2: Wellenform Sinus*

fernung von diesem Mittelpunkt wird Amplitude genannt. Diese Amplitude ist die zweite Kenngröße einer Schwingung. Die Kurvenform, die wir im Idealfall bei unserem Experiment erhalten, wird Sinus Kurve genannt. Wenn wir uns die Schwingung einer Stimmgabel ansehen, werden wir feststellen, daß diese Schwingung ein ähnliches Bild liefert.

Wir wollen nun das Experiment mit dem Pendel, das aus dem kleinen Sandsäckchen besteht, wiederholen. Lassen Sie das Pendel schwingen, indem Sie es nach außen bewegen und dann loslassen. Dieser stetigen Schwingung können Sie eine andere Schwingung entgegensetzen, indem Sie das Band, an dem das Pendel aufgehängt ist, mit einem Gegenstand leicht anschlagen. Die

langsame Schwingung des Pendels wird nun durch eine Schwingung höherer Frequenz überlagert. Schauen Sie sich die Spur an, die der Sand jetzt auf dem Papier hinterläßt. Diese Spur ist in Abb. 3 dargestellt.

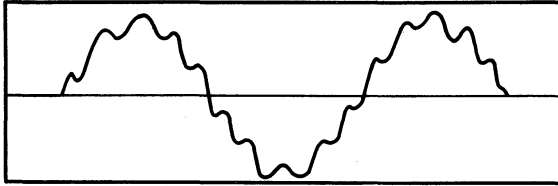


Abb. 3: Sinus-Wellenform mit Oberwellen einer höheren Frequenz

Wir erhalten wieder eine sinusähnliche Form, der allerdings Wellen überlagert sind. Diese Wellen werden auch 'Oberschwingungen' genannt. Die Klangfarbe eines Tons wird mit dieser Oberschwingung festgelegt. Dadurch ist es beispielsweise möglich, verschiedene Instrumente voneinander zu unterscheiden.

Jede Schwingung, und ist sie noch so kompliziert, kann in verschiedene Einzelschwingungen zerlegt werden. Das ist mit einem sehr komplizierten mathematischen Verfahren (der Fourier Analyse) möglich. Dabei wird eine Schwingung in ihre einzelnen Komponenten zerlegt. Jede dieser Komponenten hat einen reinen Sinus Charakter. Die einzelnen Komponenten unterscheiden sich nur durch ihre Amplitude und durch ihre Frequenz.

Die niedrigste Frequenzkomponente, die in dem Ton enthalten ist, gibt dem Ton die Höhe. Die Anteile der Komponenten mit höherer Frequenz geben dem Ton die Klangfarbe.

Wir wollen hier nicht näher auf die technischen Einzelheiten eingehen und uns der Programmierung des SID zuwenden.

Die Adressierung des SID

Stellen Sie sich den SID Chip als ein Haus vor, das Zimmer und Flure enthält und dessen Erforschung wie ein Abenteuerspiel ist. Wenn Sie in einen bestimmten Raum wollen, müssen Sie die Adresse des 'Hauses' und den 'Namen' des Raumes wissen. Es müssen nur die 'Räume' erforscht, werden, die mit

der Frequenz und der Amplitude zu tun haben. Soweit es sich auf den Mikrocomputer bezieht, wird auf den SID wie auf den Hauptspeicher zugegriffen. Wir können den Befehl POKE benutzen, um Daten in dem SID zu speichern. Der SID Chip hat 29 Register (ein Register des SID entspricht einer Speicherzelle, die bei der Speicherzelle mit der Nummer 45 272 beginnt.) Es ist besser, dieser Speicherzelle eine Variable zuzuweisen, als die Nummer der Zelle immer direkt eingeben zu müssen. Das geschieht folgendermaßen:

```
10 SID = 54272
```

(Denken Sie unbedingt daran, daß Commodore BASIC nur die ersten beiden Zeichen eines Variablennamens zur Unterscheidung beachtet; nur die ersten beiden Zeichen sind 'signifikant'. Sie sollten also in Ihrem Programm keinen weiteren Variablennamen, der mit SI beginnt, verwenden.) Der SID kann drei verschiedene Arten von Tönen erzeugen, die wir als Stimmen bezeichnen. Jede Stimme wird unter anderem durch eine Frequenz charakterisiert. Diese Information müssen wir in einem SID-Register ablegen. Dazu ist es notwendig, die Adressen der betreffenden Register zu kennen:

```
20 F1=SID: F2=SID+7: F3=SID+14
```

Für den Anfang wollen wir uns nur mit einer einzigen Stimme beschäftigen. Deshalb werden wir die anderen Stimmen dadurch ausschalten, indem wir deren Frequenz auf Null setzen:

```
30 FOR I=0 TO 1: POKE F2+I,0: POKE F3+I,0: NEXT I
```

Wir müssen den POKE-Befehl zweimal pro Stimme benutzen, um die Frequenz in das entsprechende Register einzugeben. Der Grund dafür ist, daß die Frequenz durch eine Zahl zwischen 0 und 65 535 dargestellt wird. Diese Zahl ist größer als die Zahl, die nur auf einem einzigen Speicherplatz stehen kann. (Sie erinnern sich sicherlich daran, daß Sie in einem Speicherplatz nur eine Zahl zwischen 0 und 255 ablegen dürfen). Der SID löst dieses Problem, indem zwei aufeinanderfolgende Speicherplätze die Zahl für die Frequenz speichern. Der erste Speicherplatz repräsentiert den niedrigeren (= Low) Teil der Zahl (Denken Sie an eine Bruchzahl). Der zweite Speicherplatz enthält dementsprechend den höheren (high) Teil (Stellen Sie sich

das als ganze Zahl vor).

Zur besseren Veranschaulichung sei ein Beispiel gegeben. Wir denken uns eine Zahl zwischen 0 und 65 535 aus, die wir in das entsprechende SID Register abspeichern wollen. Wir verwenden folgende Formel:

$$FH = \text{INT}(F/256) \text{ und } FL = F - 256 * FH$$

FH und FL bedeuten Frequenz Hoch (high) und Frequenz Niedrig (low). Wir werden diese Zahl mit den Befehlen POKE F1, FL: POKE F1+1, FH in das entsprechende Register speichern. F1 und F1+1 sind die Adressen der betreffenden Register des SID; FL und FH sind die Zahlen, die wir in diese Register speichern.

Ist die Zahl, die wir in das Frequenzregister speichern, nun wirklich die Frequenz, die wir hören, wenn das Programm gestartet wird? Nein. Die Frequenz in Hertz ist mit der Zahl F, die wir in das Register speichern, durch folgende Gleichung verknüpft.

$$F = \text{HZ} * 16.40439$$

Wir wollen nun eine Stimme mit einer Frequenz von 1000 Hz erzeugen. Dazu muß folgendes BASIC Programm eingegeben werden:

```
40 HZ = 1000 : C=16.40439
50 F = HZ*C
60 FH = INT(F/256)
70 FL = F-256*FH
80 POKE F1, FL: POKE F1+1, FH
```

Wenn Sie dieses Programm eingeben, werden Sie allerdings keinen Ton hören. Das liegt daran, daß Sie dem SID noch nicht die vollständige Information gegeben haben, die er benötigt. Der SID benötigt wenigstens drei weitere Angaben, bevor er einen Ton erzeugen kann: die Lautstärke (für alle drei Stimmen), die Hüllkurve (zwei Register pro Stimme) und die Wellenform (ein Register pro Stimme). Diese drei Angaben werden später im Detail besprochen; jetzt wollen wir nur die Adressen für die Register kennenlernen.

```
90 W1 = F1+4: W2 = F2+4: W3=F3+4: REM WELLENFORM
```

```

100 A1= F1+5: A2=F2+5:A3=F3+5: REM HUELLKURVENEINSTELLUNG
110 S1 = F1+6:S2 = F2+6: S3 = F3+6: REM HALTEN/AUSKLINGEN
120 VOL = SID+24: REM LAUTSTAERKE

```

Geben Sie nun die folgenden Programmzeilen ein:

```

130 POKE VOL,15
140 POKE A1,85: POKE S1,85
150 WV=64: POKE W1, WV+1
160 FOR I=0 TO 1000: NEXT I
170 POKE W1, WV

```

Wenn Sie diese Zeilen in den Computer eingegeben haben und das BASIC Programm dann starten, werden Sie einen Ton hören können. Wir sollten die Zeilen 10-120 abspeichern, da wir sie später noch einmal benutzen werden. Experimentieren Sie ein wenig, indem Sie andere Werte für A1 und S1 (Zeile 140) für WV in Zeile 150 (wählen Sie die Werte 128, 64, 32 oder 16) und für das Ende der Schleife in Zeile 160 wählen. Sie werden dann ein wenig mehr den Ablauf verstehen. Später werden Sie erfahren, was die Änderung der Werte für A1, S1, W1 und I bedeutet.

Die Zusammenfassung der Adressen, die wir bisher benutzt haben, ist in der folgenden Tabelle angegeben.

Name	Adresse	Bedeutung
Stimme #1:		
SID	54272	Startadresse für SID
F1	SID	Frequenz (low)
	F1+1	Frequenz (high)
W1	F1+4	Wellenform
A1	F1+5	Anschlag/Abschwellen
S1	F1+6	Halten/Ausklingen
Stimme #2:		
F2	SID+7	Frequenz (low)
	F2+1	Frequenz (high)
W2	F2+4	Wellenform

Name	Adresse	Bedeutung
Stimme #2		
A2	F2+5	Anschlag/Abschwellen
S2	F2+6	Halten/Ausklingen
Stimme 3		
F3	SID+14	Frequenz(low)
	F3+1	Frequenz (high)
W3	F3+4	Wellenform
A3	F3+5	Anschlag/Abschwellen
S3	F3+6	Halten/Ausklingen
Alle Stimmen		
VOL	SID+24	Lautstärke 0-15

BASIC PROGRAMM FUER DAS SETZEN VON SID ADRESSEN

```
10 SID = 54272: REM STARTADRESSE
20 C = 16.40439: REM UMWANDLUNG IN HERTZ
30 F1 = SID: F2 = SID+7:F3=SID + 14:REM FREQUENZREGISTER
40 W1=F1+4: W2=F2+4: W3=F3+4: REM REGISTER FUER WELLENFORM
50 A1=F1+5: A2=F2+5: A3=F3+5: REM HUELLKURVENREGISTER
60 S1=F1+6: S2=F2+6: S3=F3+6: REM HALTEN/AUSKLINGEN
70 VOL=SID + 24: REM SUMMENLAUTSTAERKE VON 0 BIS 15
```

Tippen Sie dieses Programm unbedingt ein, wenn Sie es nicht bereits getan haben. Speichern Sie das Programm, da es für die folgenden Programme als Vorspann benutzt wird.

Wir machen Musik

Sie werden sicherlich schon das Verständnis für Töne, die Bedeutung der Frequenz, für die Klangfarbe und die entsprechenden SID Register bekommen haben. Nun werden wir die SID Register besprechen, die den Typ der Wellenform bestimmen, verschiedene Hüllkurvencharakteristika erlauben und die Lautstärke festlegen.

Wellenform/Kontrollregister. Diese Register wurden mit W1, W2 und W3 bezeichnet. Wie einige andere Register des SID, haben auch diese Register eine zweifache Bedeutung. Jede Speicheradresse Ihres Computers kann nur eine Zahl zwischen 0 und 255 speichern. Diese Zahl wird durch 8 einzelne Bits dargestellt. Jedes Bit kann auf 0 oder auf 1 gesetzt werden, bzw. ein- oder ausgeschaltet werden. Die einzelnen Bits werden von 0 (das Bit ganz rechts) bis 7 (das Bit ganz links) durchnummeriert. Die einzelnen Bits des Wellenform Kontrollregisters haben folgende Bedeutung:

Bit	Wichtung	Bedeutung
7	128	Wellenform: Rauschen
6	64	Wellenform: Rechteck
5	31	Wellenform: Sägezahn
4	16	Wellenform: Dreieck
3	8	Rücksetzen Stimme
2	4	Ring Modulation
1	2	Synchronisation
0	1	Beginn ASDR

Die in der obigen Tabelle angegebenen Richtungen müssen für jedes Bit, welches Sie setzen wollen, zusammengezählt werden. So erhalten Sie auf jeden Fall eine Zahl zwischen 0 und 255. Wenn Sie beispielsweise bei der Variablen für die Wellenform (WC) das Bit 7 setzen wollen, geben Sie einfach

```
WC = WC OR 128: POKE W1, WC
```

ein. WC ist eine Variable, die stellvertretend für den Inhalt des Speicherplatzes W1 steht. Wir können das SID Register auch 'Write Only' nennen, da eine Zahl, die einmal mit POKE in ein Register geschrieben wurde, nicht mehr gelesen werden kann. Wenn wir nur Bit 7 ausschalten wollen, müssen wir folgende Programmzeile eingeben:

```
WC = WC AND 254: POKE W1,WC
```

Wenn Sie das nicht ganz verstehen, sollten Sie noch einmal das Kapitel in Anhang 2 lesen, das die Verwendung der logischen Operatoren AND und OR zum Ein- und Ausschalten von Bits behandelt. Einige SID-Register sind Bit-orientiert. Sie sollten sich deshalb unbedingt informieren, wie die einzelnen Bits gesetzt werden. Jede SID-Stimme muß einem Wellenformtyp zugeordnet werden. Der SID stellt die Wellenformen Dreieck, Sägezahn, Rechteck und Rauschen.

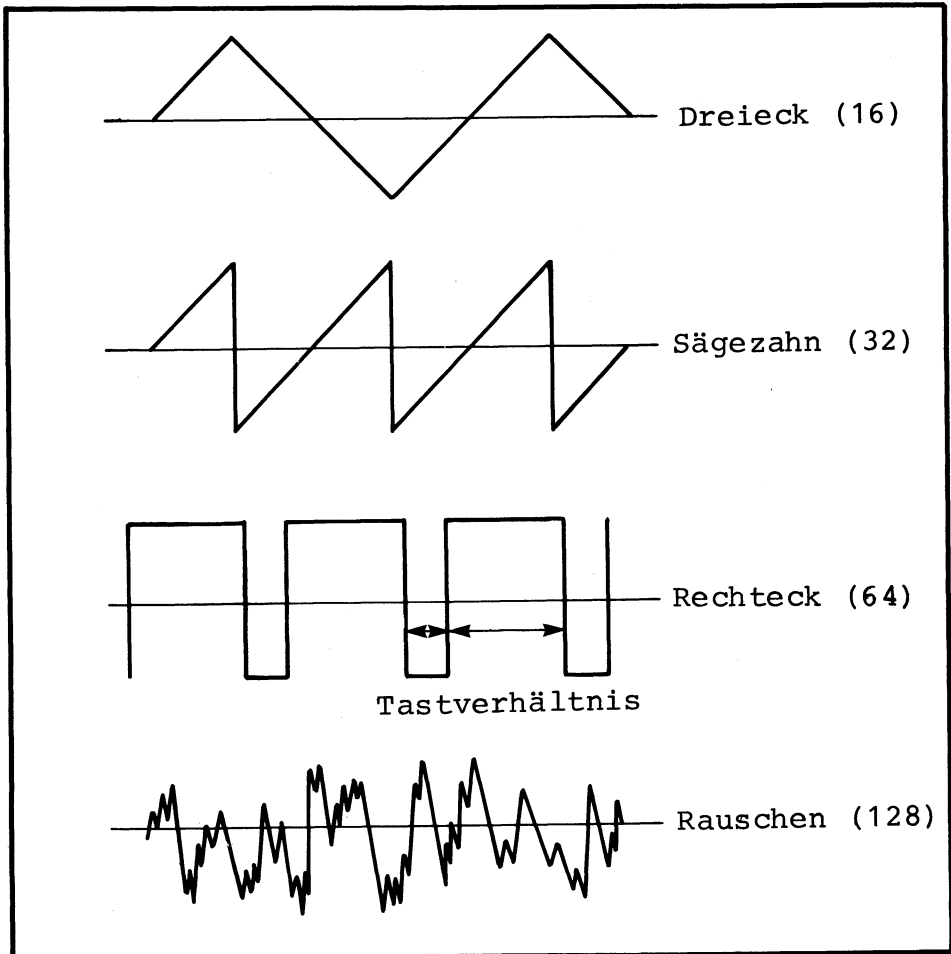


Abb. 4: Die SID-Wellenformen

eck oder Rauschen (siehe Abb. 4) zur Verfügung. Die einzelnen Wellenformen werden durch das Setzen bestimmter Bits eingeschaltet.

Wir wollen nun einige Merkmale dieser Wellenformen diskutieren. Die einfachste Wellenform ist die Dreieck Wellenform (Bit #4). Eine Periode der Schwingung besteht aus einem Dreieck. Der Wert der Kurve steigt langsam an, fällt dann bis zu einem Minimum, um dann wieder auf das Maximum anzusteigen.

Die nächste Wellenform, die der SID produzieren kann, ist die Sägezahn Wellenform (Bit #5). Das Aussehen dieser Wellenform erinnert an das Blatt einer Säge. Die Kurve steigt langsam bis zu einem Maximalwert an, um dann rapide abzufallen. Der Sägezahn gibt einen sehr brillanten Klang.

Schließlich kann der SID auch noch eine Rechteckform erzeugen (bit #6). Der Wert steigt abrupt auf einen Maximalwert an, verharrt dort einige Zeit und fällt schließlich ebenso abrupt ab. Wie man erkennen kann, gibt es zwei verschiedene Breiten des Rechtecks. Das Verhältnis dieser beiden Breiten wird Tastverhältnis genannt. Der Sägezahn hat ebenfalls eine sehr ausgezeichnete Klangfarbe, die ein wenig metallisch klingt. So besteht zum Beispiel die kräftige Klangfarbe einer Klarinette vornehmlich aus einer Wellenform, die dem Rechteck ähnelt.

Das Rauschen (Bit #7) ist die letzte Form der Stimmen, die zur Verfügung stehen. Unter 'Rauschen' versteht man die Zusammenfassung verschiedener Frequenzen.

Wir wollen nun diese verschiedenen Punkte zusammenfassen: Bevor mit dem SID ein Ton ausgegeben werden kann, muß zunächst der Wellenformtyp (Rauschen, Rechteck, Sägezahn oder Dreieck) in den Registern durch Setzen der Bits #7, 6, 5 und 4 angegeben werden. Jede Wellenform hat eine charakteristische Klangfarbe. Denken Sie daran, daß Sie nur eine Wellenform pro Stimme auswählen dürfen. Es ist somit nicht möglich, einer Stimme mehrere Wellenformen zuzuordnen.

Die anderen Bits der Wellenform Kontrollregister (W1, W2 und W3), d. h. die Bits 3, 2 und 1, sollen uns momentan nicht weiter beschäftigen. Das Bit 0 könnte man als den Schlüssel des SID Chip bezeichnen. Wenn dieses Bit auf 1 gesetzt wird, wird ein Ton erzeugt. Der Ton wird ausgeschaltet, wenn das Bit gelöscht wird. Wenn Sie beispielsweise WC auf einen zulässigen Wellenformtyp gesetzt haben, sollten Sie die Stimme #1 mit POKE W1, WC+1 aktivieren und mit POKE W1, WC wieder löschen. Sie können eine Stimme zu jeder Zeit ausschalten, indem Sie das Frequenzregister mit dem Be-

fehl POKE F1,0:POKE F1+1,0 auf Null setzen.

ADSR. Wir wollen uns nun mit dem Begriff ADSR beschäftigen. ADSR ist eine Abkürzung für Attack, Decay, Sustain und Release. Diese Begriffe bedeuten: Anschlag, Abschwellen, Halten und Ausklingen. Mit diesen vier Kenngrößen wird die Amplitude oder Lautstärke eines Tons in Abhängigkeit von der Zeit gesteuert. Viele Töne der Musik können mit einer 'Hüllkurve' beschrieben werden. Diese Hüllkurve besteht aus vier einzelnen Phasen, die wir oben erwähnt haben. Sie können mit dem SID Chip die Zeit für Attack, Decay und Release bestimmen. Der Wert für Sustain wird nicht in einer Zeiteinheit, sondern in einer Amplitude oder Lautstärke gemessen. Jede dieser ADSR Phasen kann einen Wert zwischen 1 und 16 annehmen. Dieser Wert wird mit den Zahlen 0-15 dargestellt. Dazu benötigt man ein Nybble, das vier Bit enthält. Jede Speicherzelle des SID Chip enthält 8.Bit. Die einzelnen ADSR Phasen werden durch die Paare Attack/Decay und Sustain/Release beschrieben.

Die Attack und Decay Nybbles gehen in die Register A1, A2 und A3 (jede Stimme hat ihre eigene ADSR-Kontrolle); der Sustain, die Amplitude und Release gehen in die Register S1, S2 und S3. Wenn ein Ton mit einem Musikinstrument erzeugt wird, ist der Ton nicht sofort da, sondern er muß erst gebildet werden. Dieser Effekt tritt ebenfalls auf, wenn der Ton verklingt. Wir wollen uns jetzt ansehen, wie die Phasen bei der Bildung, beim Halten und beim Ausklingen eines Tones ablaufen.

Die Lautstärke (Amplitude) eines Tones steigt von dem Wert 0 zu einem maximalen Wert an. Schauen Sie sich einmal Bild 5 an. Sie können hier die einzelnen Phasen gut verfolgen. Die erste Phase wird Attack oder Anschlag genannt.

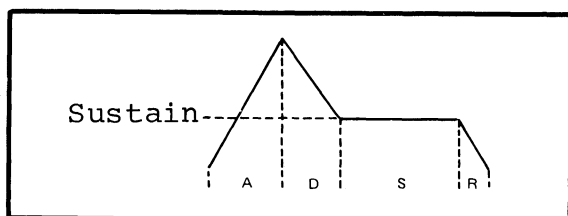


Abb. 5: ADSR-Hüllkurve

Wenn der Ton die maximale Lautstärke erreicht hat, klingt er wieder ab. Dieses Abklingen oder Abschwellen wird als Decay bezeichnet. Das Abschwellen des Tones geht aber nicht so weit, daß die Lautstärke des Tones auf 0 zurückfällt. Sie besitzt, vielmehr einen Wert, der über einen bestimmten Zeitraum gehalten wird. Dieses wird als 'Sustain' bezeichnet. Die Phase der 'Lebensdauer' eines Tones ist die Phase, die man beispielsweise bei dem Musikinstrument Orgel am längsten hört. Nach dem Halten des Tones klingt dieser schließlich wieder aus. Das Ausklingen nennt man 'Release'. Nachstehend finden Sie eine Tabelle, aus der Sie die Zeitraten für Attack, Decay und Release entnehmen können (ms = Millisekunde, s = Sekunde).

<i>Nybble</i>	<i>Attack</i>	<i>Decay/Release</i>
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

Die Attack und Decay Raten werden in die Register A1, A2 und A3 gespeichert. Die Sustain Amplitude und die Release Rate sind in den Registern S1, S2 und S3 enthalten. Diese Werte müssen nur ein einziges Mal gesetzt werden. Zum besseren Verständnis werden wir nun ein paar Musikinstrumente betrachten. Die oberen grafischen Darstellungen der einzelnen Abbildungen stellen den Verlauf der vier Phasen Attack, Decay, Sustain und Release dar. Die unteren grafischen Darstellungen geben den Punkt an, bei dem der Ton ein- und ausgeschaltet wird. Bildlich gesehen würde das bedeuten, daß ein Tor für einen Ton geöffnet oder geschlossen wird.

Als erstes Beispiel wollen wir den Ton betrachten, der von einem Blasinstrument (z. B. eine Trompete) erzeugt wird. Dieser Ton hat eine sehr schnelle Attack und Decay Rate. Der Wert für das Sustain ist relativ hoch. Die Release Rate ist hingegen ziemlich schnell. Wir wollen eine Attack Ra-

te von 16 ms, eine Decay Rate von 72 ms, eine Sustain Amplitude von 12 und eine Release Rate von 114 ms wählen. Mit Hilfe der obigen Tabelle erhalten wir:

$$AD = (\text{Attack/Decay}) = 16 \cdot 2 + 3$$

$$SR = (\text{Sustain/Release}) = 16 \cdot 12 + 4$$

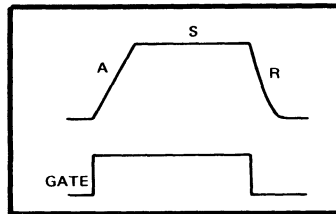


Abb. 6: *Blasinstrument*

Für die Stimme 1 müssen wir den Wert POKE A1,35: POKE S1,196 angeben. Sie hören aber noch keinen Ton. Warum nicht? Denken Sie daran, daß wir auch den Wellenformtyp und die Frequenz angeben und den SID einschalten müssen. Außerdem wird das Release nicht eingeschaltet, bevor das 'Tor' für den Ton nicht geschlossen ist. Nun folgen noch ein paar Beispiele für ADSR Werte, um Ihnen ein besseres Gefühl dafür zu geben, worauf es ankommt. Experimentieren Sie ein wenig mit den Wellenformtypen. Die ersten Zahlen beziehen sich auf Attack/Decay; dann folgen die Zahlen für Sustain und Release.

Schlagzeug - 9,8 = sehr schnelles Attack (16*0), mäßiges Decay (9), Sustain Amplitude 9 (16*0) und mittleres Release (8). (Ändern Sie das Programm so um, daß der Wellenformtyp Rauschen hörbar wird.)

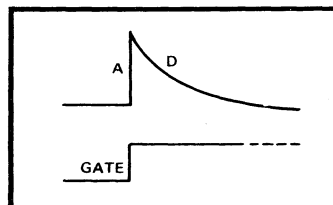


Abb. 7: *Schlagzeug*

Klavier - 10,48 = sehr schnelles Attack ($16*0$), mittleres Release (10), niedrige Sustain Amplitude ($16*4$) und sehr schnelles Release (0). (Versuchen Sie das Programm für die Wellenform Sägezahn zu ändern).

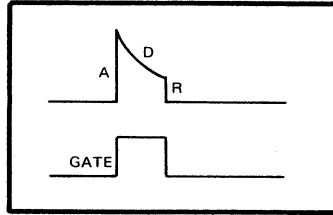


Abb. 8: Klavier

Orgel - 0,240 = sehr schnelles Attack ($16*0$), kein Decay (0), hohe Sustain Amplitude ($15*16$), sehr schnelles Release (0). (Versuchen Sie das mit dem Wellenformtyp Rechteck.)

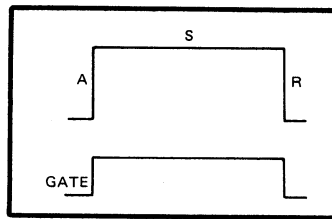


Abb. 9: Orgel

Violine - 133,165 = mäßig langsames Attack ($8*16$), mittleres Decay (5), mittlere Sustain Amplitude ($10*16$) und mittleres Release (5). (Probieren Sie die Wellenform Rechteck aus).

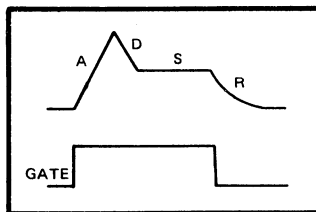


Abb. 10: Geige

Denken Sie daran, daß der SID Chip nicht dafür bestimmt ist, Instrumente

naturgetreu nachzuahmen.

Wie Sie gesehen haben, hat der SID viele Register und Funktionen. Es ist daher sinnvoll, ein Programm gut zu durchdenken, damit der Überblick nicht verloren geht. Die SID Funktionen können in zwei Kategorien unterteilt werden: Ein Typ der Funktionen bezieht sich nur auf bestimmte Stimmen, der andere beeinflusst sämtliche Stimmen gleichzeitig. Wenn wir mit der Startadresse des SID beginnen, beziehen sich die ersten drei der insgesamt sieben Gruppen auf die einzelnen Stimmen. Die nächsten vier Gruppen kontrollieren die Filter und die Lautstärke. Die verbleibenden vier Funktionen sind Spezialfunktionen (wir haben sie nicht vergessen). Wir wollen nun einmal diese Register und deren Inhalt tabellarisch festhalten.

Für die Erzeugung einfacher Geräusche bis hin zur Musik haben wir die wichtigsten Register bereits besprochen. Mit den Registern SID+0 bis 20 können wir die Frequenz, den Wellenformtyp und die Hüllkurve angeben und die Ausgabe eines Tones starten oder beenden. Das untere Nybble von Register 24 kontrolliert die gesamte Lautstärke. Für die Musikerzeugung gibt es einen einfachen Weg, die gesamte Information mit einem Programm zu handhaben. Wir werden das im nächsten Abschnitt besprechen.

<i>Register</i>	<i>Bedeutung</i>
0,7,14	Frequenz (low Byte)
1,8,15	Frequenz (high Byte)
2,9,16	Tastverhältnis (low Byte)
3,10,17	Tastverhältnis (Bit 0-13 high Nybble)
4,11,18	Kontrollregister
	Bit 0 Gatter (1 = start, 0 = löschen)
	Bit 1 Synchronisations Bit
	Bit 2 Ringmodulationsbit
	Bit 3 Testbit
	Bit 4 Dreieck Wellenform
	Bit 5 Sägezahn Wellenform
	Bit 6 Rechteck Wellenform
	Bit 7 Rauschen Wellenform
5,12,19	Attack/Decay Register
6,13,20	Sustain/Release Register
21	Filterfrequenz (Bit 0,1,2 low)
22	Filterfrequenz (high Byte)
23	Resonanz/Filterregister
	Bit 0 – Anschluß Stimme 1 an Filter
	Bit 1 – Anschluß Stimme 2 an Filter
	Bit 2 – Anschluß Stimme 3 an Filter
	Bit 3 – Anschluß Stimme eines externen Signals an das Filter
	Bit 4-7 – Resonanzfrequenz

24	Modus/Lautstärke Bit 0-3 Gesamtlautstärke Bit 4 Auswahl Tiefpaßfilter Bit 5 Bandpaßfilter Bit 6 Hochpaßfilter Bit 7 Trenner der Stimme 3 vom Audio Ausgang
----	--

Die nachfolgenden Register können nur gelesen werden:

25	Paddle X
26	Paddle Y
27	Ausgang Oszillator 3
28	Hüllkurve für Stimme 3

Der wohltemperierte Computer

Bis jetzt haben wir noch nicht besonders viel über den SID erfahren. Wir haben vornehmlich die Théorie und Organisation des SID besprochen.

Wir sollten nun einmal überlegen, welche Frequenzen wir benutzen müssen, um Musik zu machen. Wir wollen uns dabei auf die Musik beschränken, die wir in unserem westlichen Kulturkreis gewöhnt sind. Diese Musik besteht insgesamt aus 12 Tönen (do, re, mi, fa, sol, la, ti und die 5 Halbtöne) pro Oktave. In jeder Oktave ist jedem Ton eine bestimmte Frequenz zugeordnet. In der sogenannten wohltemperierten Skalierung kann die Frequenz einer jeden Note durch ein Vielfaches der vorangegangenen bestimmt werden. Das Verhältnis der Stimmung ist in Wirklichkeit ein Kompromiß, damit verschiedene Instrumente harmonisch zusammenpassen.

Das Vielfache, mit dem die einzelnen Noten zu multiplizieren sind, ist die zwölfte Wurzel aus 2 ($=1.0594630943593$). Diese Zahl bekommt den Wert 2, wenn sie zwölfmal mit sich selbst multipliziert wird. Das bedeutet, daß wir so zu der nächsten Oktave - unabhängig davon, von welchem Ton aus wir starten - gelangen, wenn wir die Frequenz verdoppeln. Wir müssen uns deshalb nur auf den Wert einer einzigen Note einigen und können dann sämtliche anderen Noten von dieser ableiten.

Die Musiker haben sich international darauf geeinigt, die Note A4 (das 'la' der vierten Oktave) auf eine Frequenz von genau 440 Hertz festzulegen. Von dieser Note ausgehend, können alle anderen Noten erzeugt werden. Wollen wir immer die Frequenz für jede einzelne Note erneut angeben, wenn wir eine Note spielen wollen? Natürlich nicht, wir nutzen die obige

Beziehung aus, um unser Programm abzukürzen.

Das BASIC Programm berechnet ein für allemal die Frequenzen der obersten Oktave. Fügen Sie diesen Programmteil dem vorher angegebenen Programm an.

```
100 DIM F(12): REM F(1) IST DIE SID FREQUENZ FUER AG, F(12) IST DIE
    SID FREQUENZ FUER 6#7
110 A7 = 8*440: REM FREQUENZ IN HZ FUER DIE NOTE A IN DER SIEBTEN
    OKTAVE
120 K = 2*(1/12): REM MULTIPLIKATIONSFAKTOR
130 FRQ A7*C: REM UMWANDLUNG IN SID-WERT, C=16.40439
140 FOR I=12 TO 1 STEP -1
150 FRQ=FRQ/K : REM NAECHSTE HALBTONSTUFE
160 F(I)=FRQ
170 NEXT I
```

Löschen Sie die SID Register und setzen einige Sie Anfangsbedingungen.

```
200 FOR I=0 TO 28: POKE SID + I,0: NEXT I
210 POKE VOL,15: REM LAUTSTAERKE-MAXIMUM
220 TI$="000000": REM TIMER AUF 0 SETZEN
230 DIM N(12) : REM FELD FUER MUSIKPARAMETER
240 FOR I=1 TO 9: READ N(I): REM PARAMETER FUER JEDE STIMME
250 NEXT I
260 REM BEISPIELDATEN KOENNEN VARIERT WERDEN
270 DATA 16,16,16: REM WELLENFORM TYP
280 DATA 55,55,55: REM ATTACK/DECAY FUER JEDE STIMME
290 DATA 55,55,55: REM SUSTAIN/RELEASE FUER JEDE STIMME
300 POKE W1,N(1): POKE W2,N(2):POKE W3, N(3)
310 POKE A1,N(4): POKE A2,N(5): POKE A3,N(6)
320 POKE S1,N(7):POKE S2,N(8): POKE S3,N(9)
330 FOR I=1 TO 3
340 READ W: N(I+9)=W*4096
360 NEXT I
370 DATA .5,.5,.5:REM MUSS ALS DEZIMALZAHL ZWISCHEN 0 UND 0.5 ANGEGBEN
    WERDEN
380 POKE F1+3,INT(N(10)/256):POKE F1+2,N(10)-256*INT(N(10)/256)
```



```
390 POKE F2+3,INT(N(11)/256):POKE F2+2,N(11)-256*INT(N(11)/256)
400 POKE F3+3,INT(N(12)/256):POKE F3+2,N(12)-256*INT(N(12)/256)
```

(Denken Sie daran, daß das Tastverhältnis nur dann die Klangfarbe beeinflußt, wenn Sie die Rechteckwellenform gewählt haben.)

Wenn Sie später versuchen wollen, ein Musikstück zu programmieren, speichern Sie dieses Programm: Die Zeilen 10-70 setzen die SID Adressen, und die Zeilen 100-160 berechnen die temperierte Stimmung. Der SID Chip wird Ihnen ein Musikstück spielen können, wenn die entsprechenden Daten eingegeben werden. Wir wollen zunächst damit beginnen, nur eine einzige Stimme zu spielen. Ändern Sie den Wellenformtyp und die ADSR Daten, um so hören zu können, was passiert.

```
490 S=1: REM KONSTANTES TEMPO
500 READ TM: REM EINLESEN DER TONDAUER
510 IF TM=0 THEN 9999
520 IF TM < 0 THEN 9999
530 T=TI+S*TM:REM ANGABE, FALLS DAS SPIELEN EINER NOTE BEENDET WERDEN
    SOLL
540 READ N,0 : REM EINLESEN DER NOTE UND DER OKTAVE
550 FR=F(N)/(2*0): REM BERECHNUNG DER FREQUENZ FUER DEN SID
560 FH = INT(FR/256): FL=FR-256*FH
570 POKE F1,FL: POKE F1+1,FH
580 POKE W1,N(1)+1: REM ATTACK
590 IF TI < T THEN 590: REM WARTESCHLEIFE FUER DIE NOTENDAUER
600 POKE W1,N(1): REM BEGINN DER RELEASE PHASE
610 GOTO 500: REM EINLESEN WEITERER DATEN

700 DATA 40,4,2
710 DATA 40,6,2
720 DATA 40,8,2
730 DATA 40,9,2
740 DATA 40,11,2
750 DATA 40,1,1
760 DATA 40,3,1
770 DATA 40,4,1
780 DATA 0
```

```
9999 FOR I=0 TO 28: POKE SID + I,0: NEXT I: END: REM AUSSCHALTEN DES SID  
UND BEENDEN DES PROGRAMMS
```

Starten Sie das Programm. Hört es sich nicht ganz gut an? Versuchen Sie das Programm mit der Wellenform Rechteck. (Geben Sie in Zeile 270 als erste Zahl eine 64 an.) Versuchen Sie außerdem verschiedene Tastverhältnisse (die erste Zahl in Zeile 370). Nun ein paar weitere Erklärungen. Die Variable TI in Zeile 590 wird für einen bestimmten Zweck benutzt. Sie entspricht wie TI\$ der internen Uhr des Computers. Der einzige Unterschied ist, daß keine Zeichenkette, sondern ein Zeitmaß in Einheiten von 1/60 Sekunde angegeben wird. Deshalb bedeutet die 40 in Zeile 700 des DATA Statements, daß der Ton 40/60 Sekunden (also ca. 0,67 Sekunden) dauert.

Zeile 590 wird also dazu verwendet, um eine genaue Zeitangabe anzugeben. Die FOR..NEXT Warteschleife erlaubt es nämlich nicht, einen akkuraten Zeitwert anzugeben, da die Ausführungszeit von BASIC abhängt. Wenn Sie sich das Programm ansehen, werden Sie feststellen, daß die Variable M negative Zahlen einschließlich der 0 annehmen kann, um die Ausführung des Programms in Abhängigkeit von diesen beiden Bedingungen zu verzweigen. Wenn TM 0 ist, verzweigt das Programm zu einem Teil, in dem Sie den Typ der Wellenform und die ADSR Werte verändern können. Eine negative Angabe für TM bedeutet, daß Sie das Programm beenden wollen. In unserem Beispiel wird in beiden Fällen zu der Anweisung 9999 (dem Ende des Programms) verzweigt.

Es gibt mehrere Möglichkeiten, die Noten zu verschlüsseln. Wir können beispielsweise Stringwerte (wie A2, C#3, Bb1...) benutzen. Wir können aber auch direkt die Frequenzen angeben (z. B. 880, 231,987, .2,), die Angaben do, re mi... benutzen oder die Noten in einer grafischen Form eingeben. Ähnliches können wir natürlich auch mit der Notendauer vornehmen. Der Weg, den ich zeigen werde, ist ein guter Kompromiß zwischen der Lesbarkeit der Notenangabe und der Geschwindigkeit, mit der das Programm ausgeführt wird. Später können Sie ja auch selbst eine komfortable Methode für die Angabe von Noten entwickeln und programmieren. Jetzt sollten Sie sich darauf beschränken, die nachfolgenden Programme zu verstehen.

Note #	Musik Symbol
1	la oder A
2	(le) oder A# oder Bb
3	ti oder B

Note	Musiksymbol
4	do oder C
5	(di) oder C# oder Db
6	re oder D
7	(ra) oder D# oder Eb
8	mi oder E
9	fa oder F
10	(fi) oder F# oder Gb
11	sol oder G
12	(si) oder G# oder Ab

Computernotierung		Musiknotierung		
3,1	1,1	B	A	
11,2	9,2	G	F	
8,2	6,2	E	D	
4,2	3,2	C	B	
1,2	11,3	A	G	
9,3	8,3	F	E	
6,3	4,3	D	C	
3,3	1,3	B	A	
11,4	9,4	G	F	
8,4	6,4	E	D	
4,4	3,4	C	B	
1,4	11,5	A	G	
9,5	8,5	F	E	
6,5		D		

Violin- oder G Schlüssel

Bass- oder F Schlüssel

Notenerhöhung (#) +1
Noterniedrigung (b) -1

Jetzt wissen wir, warum wir die Sequenz 4,6,8,9,11,1,3,4 für unsere Noten benutzen, um eine C-Dur Tonleiter zu spielen. In der musikalischen Notation wird die Tondauer als Bruch einer ganzen Note angegeben: eine viertel Note, eine halbe Note etc. Die Angabe erfolgt durch grafische Symbole. Wir wollen nun willkürlich als Wert für eine ganze Note den 160/60 Teil einer

Sekunde wählen. Eine ganze Note ist deshalb mit dem Notenwert 160 belegt, eine halbe Note bekommt den Wert 80, eine viertel Note den Wert 40 usw. Mit dieser Technik können auch punktierte Noten codiert werden. Folgt nach einer Note ein Punkt, so bedeutet das, daß die Note um den halben Wert ihres ursprünglichen Wertes verlängert wird. Mit einer halben punktierten Note erhalten wir eine $3/4$ Note. ($1/2 + 1/4 = 3/4$) In der Tabelle sind sämtliche Notenwerte mit dem entsprechenden Wert aufgeführt.

Name	Symbol	Wert	Name	Symbol	Wert
punktierte ganze		240	punktierte achte		30
ganze		160	viertel Triole		27
punktierte halbe		120	achte		20
halbe		80	achte Triole		13
punktierte viertel		60	sechzehntel		10
viertel		40	zweiund-dreißigstel		5

Nun gut. Bis jetzt haben wir nur mit einer einzigen Stimme gearbeitet. Wir werden nun auch mit mehrstimmiger Musik experimentieren. Unser DATA Statement wird - anstelle eines einzigen Satzes - aus drei Sätzen von Informationen bestehen. Wenn alle drei Noten für jede Stimme zur gleichen Zeit beginnen und enden, ist das ganz einfach. Normalerweise ist das aber nicht der Fall. In einigen Musikstücken werden die Noten quasi nacheinander gespielt. Das bedeutet, daß Sie die Zeitangaben und die Frequenzen für jeden Ton in einer separaten Tabelle speichern können, die Sie nacheinander abarbeiten. Diese Technik ist leicht zu verwirklichen, wenn wir für jede Stimme einen separaten Timer benutzen. In der anderen Technik werden alle

drei Stimmen gleichzeitig gespielt. Beispielsweise könnten die Trompete, das Klavier und die Klarinette den gleichen Ton zur gleichen Zeit spielen. Diese 'vertikale' Anordnung der Töne ist in einem Programm viel leichter zu realisieren, als die im vorangegangenen Absatz beschriebene 'horizontale' Anordnung der Töne.

In dem folgenden Beispiel werden drei Noten zur gleichen Zeit gestartet. Jede dieser Noten hat allerdings einen anderen Zeitwert. Eine Note soll eine viertel, die anderen eine halbe Note sein. Wir können diese drei Noten in einzelne Noten aufteilen. Für jede Note verwenden wir eine neue DATA Anweisung. In der ersten Phase starten wir alle drei Noten mit der ihnen charakteristischen Attack. Die Zeitdauer für unseren Timer erhält den Wert einer viertel Note. Es folgt ein kleines Beispielprogramm, das drei Stimmen benutzt. Starten Sie das Programm mit den Datenzeilen 800-880. Sie müssen die nachfolgenden BASIC-Zeilen den bereits vorhandenen Zeilen anfügen.

```

410 BYTE=256: Z=0: W=1: M=-1: REM KONSTANTEN
420 DIM FH(12,6), FL(12,6),O(6): REM FREQUENZ- UND OKTAVENMULTIPLIKA-
    TOR
430 FOR J=0 TO 6: O(J)=2*J: NEXT J
440 FOR I=1 TO 12: FOR J=0 TO 6
450 FR=F(I)/O(J): FH(I,J)=INT(FR/BY): FL(I,J)= FR-BY*FH(I,J)
460 NEXT J,I
490 S=1: REM TEMPOANGABE
500 READ TM: T=TI+S*TM: REM EINLESEN DER NOTENDAUER UND SETZEN DES
    TIMERS
510 IF TM=Z THEN 9999
520 IF TM < Z THEN 9999
530:
540 READ N1, O1,N2,O2,N3,O3,: REM NOTE UND OKTAVE #'S
550 REM BEGINN RELEASE
560 IF N1 THEN POKE W1,N(1)
570 IF N2 THEN POKE W2,N(2)
580 IF N3 THEN POKE W3,N(3)
590 REM ATTACK
600 IF N1 > Z THEN POKE F1,FL(N1,O1):POKE F1+W,FH(N1,O1): POKE W1,
    N(1)+W

```

```
610 IF N2 > Z THEN POKE F2,FL(N2,02):POKE F2+W,FH (N2,02) : POKE W2,  
    N(2)+W  
620 IF N3 > Z THEN POKE F3,FL(N3, 03): POKE F3 + W, FH(N3,03): POKE  
    W3,N(3)+W  
630:  
640 IF TI < T THEN 500: REM EINLESEN WEITERER DATEN  
650 GOTO 640: REM TIMER
```

Dieses Programm unterscheidet sich ein wenig von unseren bisherigen Musikprogrammen. Wir haben den bedeutenden Konstanten Variablen zugeordnet - 256, 1, und -1 - um Zeit zu gewinnen, da BASIC in diesem Fall nicht ständig die entsprechenden Konversionen durchführen muß. Wir haben außerdem die Multiplikationsfaktoren zur Erzeugung der Oktaven vorberechnet und in das Feld 0(I) gespeichert. Es wird eine Frequenztabelle FH(I,J) und FL(I,J) aufgebaut. Sie sollten die REMark Anweisungen in den Programmen löschen, da sie unnötigerweise Speicherplatz belegen. Die Geschwindigkeit ist jetzt ein wenig kritisch, da wir drei Parametersätze in das Programm eingebaut haben. Unser Ohr ist in Bezug auf Zeitverschiebungen sehr empfindlich. Wir müssen also große Ausführungszeiten vermeiden, damit wirklich alle drei Stimmen gleichzeitig zu hören sind. Auch dieses Programm sollten Sie mit SAVE speichern. Sie können es ja SIDMUSIK3 nennen. Tippen Sie nun die folgenden DATA Statements ein.

```
800 DATA 40,04,2,04,1,00,0  
810 DATA 40,06,2,03,1,00,0  
820 DATA 40,08,2,01,2,00,0  
830 DATA 40,09,2,11,2,00,0  
840 DATA 40,11,2,09,2,00,0  
850 DATA 40,01,1,08,2,00,0  
860 DATA 40,03,1,06,2,00,0  
870 DATA 40,04,1,04,2,00,0  
880 DATA 0
```

```
9999 FOR I=0 TO 28: POKE SID+I,0: NEXT I: END: REM AUSSCHALTEN SID  
    BEENDEN DES PROGRAMMS
```

Wenn Sie dieses Programm starten, werden Sie hören können, was geschieht. Jede der beiden Stimmen wird synchronisiert; eine Stimme spielt die C-Dur Tonleiter aufwärts, die andere abwärts. Wir werden nun das Programm etwas erweitern. Sie werden dann feststellen, warum wir die Bedingung (If N1>Z) in dem Programm benötigen.

```
800 DATA 40,04,2,04,1,00,0
810 DATA 40,06,2,00,0,00,0
820 DATA 40,08,2,01,2,00,0
830 DATA 40,09,2,00,0,00,0
840 DATA 40,11,2,09,2,00,0
850 DATA 40,01,1,00,0,00,0
860 DATA 40,03,1,06,2,00,0
870 DATA 40,04,1,00,0,00,0
```

Die 0 für die Notenidentität (Note ID) in der zweiten Stimme bedeutet, daß diese Stimme nichts tun soll; der Ton soll beispielsweise gehalten werden. Wir haben jetzt eine Serie von 1/4 Noten in Stimme 1, die gegen eine Serie von 1/2 Noten in Stimme 2 spielt. Merken Sie, daß das Spielen einer halben Note nicht mit dem Spielen von zwei 1/4 Noten identisch ist? Erinnern Sie sich daran, daß immer dann, wenn eine Note gespielt wird, diese zunächst in die Anschlagsphase geht. Tippen Sie folgende Programme ein, damit Sie den Unterschied verstehen (Löschen Sie aber zuvor Zeile 880).

```
900 DATA 40,04,2,04,1,00,0
910 DATA 40,06,2,04,1,00,0
920 DATA 40,08,2,01,2,00,0
930 DATA 40,09,2,01,2,00,0
940 DATA 40,11,2,09,2,00,0
950 DATA 40,01,1,09,2,00,0
960 DATA 40,03,1,06,2,00,0
970 DATA 40,04,1,06,2,00,0
980 DATA 0
```

Das ist doch ein Unterschied! Lassen Sie uns jetzt Folgendes probieren. Anstatt einer viertel Note eine halbe Note gegenüberzustellen, wollen wir jetzt in der zweiten Stimme eine viertel Note mit einer anschließenden

viertel Pause spielen. In unserer Stimme war keine Pause vorgesehen. In der Musik sind Pausen von großer Bedeutung. Denken Sie daran; wir verwenden die 1 in unserer Notenverschlüsselung zur Angabe, daß eine Note ohne sofortigen Anschlag verstummen soll. Deshalb benötigen wir eine besondere Bedingung. Um den Effekt hören zu können, müssen nachstehende DATA-Anweisungen eingegeben werden.

```

900 DATA 40,04,2,04,1,00,0
910 DATA 40,06,2,-1,0,00,0
920 DATA 40,08,2,01,2,00,0
930 DATA 40,09,2,-1,0,00,0
940 DATA 40,11,2,09,2,00,0
950 DATA 40,01,1,-1,0,00,0
960 DATA 40,03,1,06,2,00,0
970 DATA 40,04,1,-1,0,00,0
980 DATA 0

```

Hören Sie den Unterschied? Wir können jetzt alle drei Stimmen zusammenfassen und einen Teil einer Sinfonie von Bach spielen. Nachstehend sehen Sie die Noten.

Sinfonie 10. (J. S. BACH)



Abb. 13: Beispiel einer Musikpartitur

Um das Programm besser verfolgen zu können, haben wir das komplette Musikstück in die DATA Zeilen für die einzelnen Takte aufgeteilt. Der jeweiligen REMark Anweisung können Sie entnehmen, um welche Takte es sich handelt. Um einen besseren Überblick zu bekommen, sollten Sie nach Eingabe der Daten für jeden Takt das Programm einmal starten und sich die Musik anhören. Hier sind die Datenzeilen für den ersten Takt.

795 REM TAKT 1

800 DATA 20,00,0,00,0,11,5

802 DATA 20,11,3,00,0,00,0

804 DATA 10,00,0,00,0,03,4

806 DATA 10,10,3,00,0,00,0

808 DATA 10,11,3,00,0,00,0

810 DATA 10,01,2,00,0,00,0

812 DATA 10,03,2,00,0,11,5

814 DATA 10,04,2,00,0,00,0

816 DATA 10,06,2,00,0,00,0

818 DATA 10,03,2,00,0,00,0

9997 DATA 0

Starten Sie das Programm mit diesen Datenzeilen und hören Sie sich die Musik an. Geben Sie anschließend den nächsten Takt ein.

819 REM TAKT 2

820 DATA 20,08,2,00,0,04,4

822 DATA 20,06,2,00,0,00,0

824 DATA 20,04,2,00,0,00,0

826 DATA 20,03,2,00,0,00,0

828 DATA 20,01,2,00,0,06,4

830 DATA 20,11,3,00,0,00,0

832 DATA 20,10,3,00,0,00,0

834 DATA 20,08,3,00,0,00,0

836 DATA 20,06,3,00,0,06,5

838 DATA 20,04,2,00,0,00,0

840 DATA 20,03,2,00,0,00,0

842 DATA 20,01,2,00,0,00,0

Die zweite Stimme war bis jetzt noch nicht zu hören. In Takt 3 kommt nun die zweite Stimme hinzu. Geben Sie die folgenden DATA Zeilen ein, und Sie werden diese Stimme hören können.

```
844 REM TAKT 3
846 DATA 40,03,2,00,0,11,5
848 DATA 40,00,0,06,3,00,0
850 DATA 20,00,0,00,0,11,4
852 DATA 20,01,2,05,3,00,0
854 DATA 20,03,2,06,3,00,0
856 DATA 20,05,2,08,3,00,0
858 DATA 20,06,2,10,3,10,4
860 DATA 20,00,0,11,3,00,0
862 DATA 20,00,0,01,2,00,0
864 DATA 20,00,0,10,3,00,0
866 REM TAKT 4
868 DATA 10,00,0,03,2,11,4
870 DATA 10,00,0,01,2,00,0
872 DATA 10,00,0,11,3,00,0
874 DATA 10,00,0,10,3,00,0
876 DATA 10,00,0,08,3,01,3
878 DATA 10,00,0,06,3,00,0
880 DATA 10,00,0,05,3,00,0
882 DATA 10,00,0,03,3,00,0
884 DATA 10,05,2,01,3,01,4
886 DATA 10,00,0,11,3,00,0
888 DATA 10,00,0,10,3,00,0
890 DATA 10,00,0,08,3,00,0
```

Beachten Sie bitte die Pause für Stimme 3 in Takt 5 (-1,0). Stimme 3 ist in Takt 6 nicht zu hören.

```
892 REM TAKT 5
894 DATA 10,06,2,10,3,06,4
896 DATA 10,00,0,01,3,00,0
898 DATA 10,00,0,03,3,00,0
900 DATA 10,00,0,05,3,00,0
902 DATA 10,00,0,06,3,-1,0
```

904 DATA 10,05,2,08,3,00,0
906 DATA 10,06,2,10,3,00,0
908 DATA 10,08,2,11,3,00,0
910 DATA 10,10,2,01,2,00,0
912 DATA 10,11,2,03,2,00,0
914 DATA 10,01,1,04,2,00,0
916 DATA 10,10,2,01,2,00,0
918 REM TAKT 6
920 DATA 10,03,1,06,2,00,0
922 DATA 10,01,1,00,0,00,0
924 DATA 10,11,2,03,2,00,0
926 DATA 10,10,2,00,0,00,0
928 DATA 10,08,2,11,3,00,0
930 DATA 10,06,2,00,0,00,0
932 DATA 10,04,2,08,3,00,0
934 DATA 10,03,2,00,0,00,0
936 DATA 10,04,2,01,2,00,0
938 DATA 10,11,2,00,0,00,0
940 DATA 10,10,2,00,0,00,0
942 DATA 10,08,2,00,0,00,0

Stimme 3 ist in Takt 7 wieder zu hören und greift das Thema erneut auf.

944 REM TAKT 7
946 DATA 10,06,2,00,0,00,0
948 DATA 10,00,0,04,2,00,0
950 DATA 10,00,0,03,2,11,4
952 DATA 10,00,0,01,2,00,0
954 DATA 10,00,0,03,2,00,0
956 DATA 10,06,2,00,0,10,4
958 DATA 10,08,2,00,0,11,4
960 DATA 10,09,2,00,0,01,3
962 DATA 10,11,2,00,0,03,3
964 DATA 10,00,0,01,2,04,3
966 DATA 10,00,0,03,2,06,3
968 DATA 10,00,0,11,3,03,3

Wie gefällt Ihnen das Stück? Experimentieren Sie ein bißchen, indem Sie die Stimmen ändern oder andere Werte für das Tempo setzen. In der Life-Musik ist die Dauer einer Note natürlich niemals exakt gleich. Das ist von den Komponisten teilweise auch bewußt gewollt. So gibt es in der Notation verschiedene Befehle für Tempoänderungen, z. B. rit. für Verzögerungen, acc. für steigendes Tempo usw. Geben Sie andere Notenwerte ein und machen Sie sich mit diesem Programm etwas vertraut.

Musik aus dem Weltall

Sie können bei Geräuscheffekten nichts falsch machen, wenn Sie sich an ein paar einfache Regeln halten. Die nützlichste Funktion des SID für Geräuscheffekte ist der Rauschgenerator, den Sie bisher noch nicht benutzt haben. Dieser Rauschgenerator hat keine definierte Tonhöhe; er hat vielmehr eine 'Farbe', die kontrolliert werden kann. Der SID erzeugt das Rauschen, indem Zufallszahlen in Wellenform ausgegeben werden. Die Rate der Frequenz wird durch den Wert bestimmt, der im Frequenzregister gespeichert ist. Bei niedrigen Raten wird das Rauschen 'rumpelnd' klingen und bei hohen Raten zischend. Eine weitere Kontrolle kann mithilfe von Filtern durchgeführt werden, die wir ebenfalls noch nicht benutzt haben. Die Rauschstimme ist außerdem mit ADSR kontrollierbar. Wir wollen jetzt einige dieser Effekte ausprobieren. Laden Sie zur Definition einiger SID Adressen folgendes Programm.

```

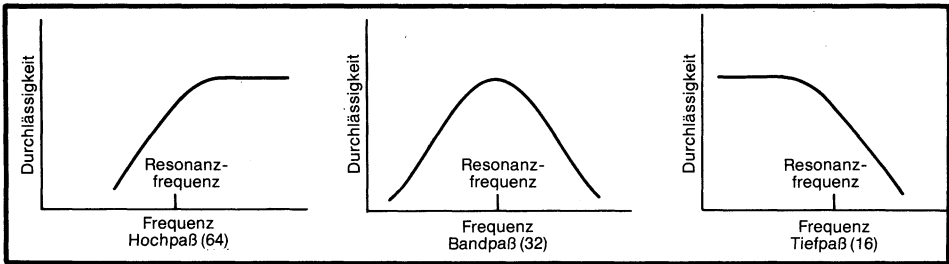
100 POKE VOL,15: REM LAUTSTÄERKE AUF MAXIMUM STELLEN
110 WV=128: REM WELLENFORM RAUSCHEN
120 AT=1: REM ATTACK
130 DE=8: REM DECAY
140 SU=8: REM SUSTAIN AMPLITUDE
150 RE=10: REM RELEASE
160 FRQ=1000: REM FREQUENZ IN HZ
170 TM=6: REM ZEITANGABE IN SEKUNDEN
180:
190 FR=FR*C: FH=INT(FR/256): FL=FR-256*FH: REM HIGH UND LOW BYTE
210 TI$="000000": TM=TM*60:
220 POKE A1,16*AT+DE: REM ANGABE DER ATTACK/DECAY RATE

```

```
230 POKE S1,16*SU+RE: REM ANGABE DER SUSTAIN AMPLITUDE UND RELEASE
    RATE
240 POKE F1,FL: POKE F1+1, FH: REM ANGABE DER FREQUENZ
250:
260 T=TI+TM: POKE W1, WV+1: REM STARTEN DES GERAUSCHES
270 IF TI < T THEN 270: REM HALTEN DER NOTE
280 POKE W1,WV: REM BEENDEN DES TONES
```

Führen Sie dieses Programm mit RUN aus, und versuchen Sie, verschiedene Werte für FRQ (FRQ darf nicht größer als 3900 Hz sein). Probieren Sie dann verschiedene Werte für AT, DE, SU und RE. Beachten Sie bitte, daß TM nicht zu klein sein darf, da sonst die Release Phase eingeschaltet wird, bevor die Sustain Phase erreicht werden kann. Versuchen Sie beides: sehr niedrige Frequenzen (um 20 Hz) und sehr hohe Frequenzen (um 3000 Hz). Mit einer schnellen Attack Rate (AT=0) und ausgeschaltetem Sustain (SU=0) können Sie einen Pistolenknall, das Zuschlagen einer Tür, Schritte und verschiedene andere Impulsvorgänge simulieren. Mit mittleren Werten für Attack und Decay, Sustainamplituden und langem Release können Sie Wind, Wellen, entfernte Explosionen, Sturm und andere Arten von "rauschenden Geräuschen" erzeugen.

Um die Klangfarbe der Geräusche zu ändern, ist die Benutzung von Filtern möglich. Die Benutzung von Filtern kann sehr kompliziert sein. Das gilt besonders für den Fall, wenn der SID mit BASIC programmiert wird. Sie können aber die folgenden Geräuschbeispiele studieren und die Zahlen, die mit POKE in die entsprechenden Register geschrieben werden, ändern. Wie der Name schon sagt, bedeutet ein Hochpaßfilter, daß nur Frequenzen, die höher als die entsprechende Resonanzfrequenz sind, das Filter passieren können. Das Tiefpaßfilter erlaubt nur Frequenzen, die tiefer als die Resonanzfrequenz sind, das Filter zu passieren. Mit dem Bandpaßfilter können nur Frequenzen, die geringfügig über oder unter der Resonanzfrequenz liegen, das Filter passieren. Bild 14 gibt eine Erklärung. Auf der X-Koordinate ist die Frequenz aufgezeichnet; auf der Y-Koordinate ist der Anteil des Tones, der das Filter passieren kann, angegeben (Durchlässigkeit). Mit CUTOFF ist der Punkt bezeichnet, bei dem der Ton 'abgeschnitten' wird.



Um die Wirkungsweise der Filter auszuprobieren, sollten Sie folgendes Programm eingeben und obigem Programm anfügen.

```

172 MODE=32 : REM FILTERMODUS
174 RS = 15: REM RESONANZ
176 RFRQ = 10000 : REM FILTER FREQUENZ
178 RF = (RFRQ-30)/5.8: RH = INT(RF/8): RL= RF-8*RH : REM RESONANZ-
    FREQUENZ
242 POKE SID+23,16*RS+1: REM EINGABE RESONANZ, VERKNUEPFUNG DER STIMME
    1 MIT DEM FILTER
244 POKE SID+21, RH: POKE SID+22,RL:REM EINGABE FILTERFREQUENZ
246 POKE VOL,M0+15: REIM EINSCHALTEN DES FILTERMODUS, SETZEN DER LAUT-
    STAERKE AUF MAXIMALEN WERT

```

Starten Sie das Programm und experimentieren Sie ein bißchen mit MODE (Zeile 172), indem Sie die Werte 64 (Hochpaß), 32 (Bandpaß) und 16 (Tiefpaß) ausprobieren. Prüfen Sie ebenfalls die Effekte, die Sie bekommen, wenn Sie FRQ (Zeile 160) und RFRQ (Zeile 176) ändern. Sie werden feststellen, daß die Lautstärke schwächer wird, da die Filter einen großen Teil der Energie beanspruchen. Verändern Sie auch die Werte für Attack/Decay und Sustain/Release.

Den vollen Effekt eines Filters können Sie besser verfolgen, wenn Sie die Resonanzfrequenz verändern, während der Ton gehalten wird. Sie können das erreichen, indem Sie die folgenden Zeilen dem obigen Programm hinzufügen:

```

270 ML=1.1: REM FAKTOR FUER DIE AENDERUNG DER RESONANZFREQUENZ

```

```
280 E=8: W=1: HI=2000: LO=500
290 IF TI > TM THEN 360
300 RF=RF*ML
310 IF RF > HI AND ML > W THEN ML=W/ML: RF=RF*ML: REM AENDERN DER
    RESONANZFREQUENZ
320 IF RF < LO AND ML < W THEN ML=W/ML: RF=RF*ML
330 RH = INT(RF/E): RL = RF-E*RH: REM BERECHNUNG LOW UND HIGH BYTE
340 POKE SID + 21, RL :POKE SID+22 RH: REM EINGABE RESONANZFREQUENZ
350 GOTO 300
360 POKE W1, WV: REM RELEASE PHASE
```

Nun können Sie ein bißchen mit den Filtern spielen. Versuchen Sie ML (Zeile 270) zu ändern. Speichern Sie dieses Programm und geben Sie ihm vielleicht den Namen SIDNOISE.

Sie haben sich nach und nach mit den einzelnen Möglichkeiten des SID vertraut gemacht. Sie werden daher schon bemerkt haben, daß Sie eine Menge verschiedenster Effekte mit dem SID erzielen können.

Zum Schluß wollen wir ein paar Geräuscheffekte ausprobieren, um das, was wir vorher gelernt haben, anschaulich zu machen. Wie üblich, sollten Sie das Programm mit den Zeilen 10-90 (wie zu Beginn des Kapitels erläutert) eröffnen. Es folgen nun einige Beispiele. Aus dem Programmnamen und aus den REM Anweisungen können Sie ersehen, um welchen Geräuscheffekt es sich gerade handelt. Sie sollten ein bißchen mit den Daten spielen, um die verschiedenen Effekte auszuprobieren.

Geräuscheffektprogramme

KANONENSCHUß

```
100 POKE F1,200 : POKE F1+1,40 : REM STIMME 1
110 POKE A1,0+15: REM SCHNELLER ATTACK, LANGSAMER DECAY
120 POKE S1,0: REM KEIN SUSTAIN, SCHNELLES RELEASE
130 FOR I = 15 TO 0 STEP -1: REM 15 SCHRITTE
140 POKE W1, 128+1: REM WELLENFORM RAUSCHEN FUER STIMME 1, START ATTACK
150 POKE VOL,I: REM DYNAMISCHE VERAENDERUNG DER LAUTSTAERKE
160 NEXT I
170 POKE W1,0 : REM BEENDEN STIMME 1
```

BABYPLAERREN

```
100 POKE VOL,15 : REM LAUTSTAERKE AUF MAXIMUM SETZEN
110 POKE A1,15: REM SCHNELLER ATTACK, LANGSAMER DECAY
120 POKE S1,0: REM KEIN SUSTAIN, SCHNELLES RELEASE
130 POKE F1+1,40:REM HOHE FREQUENZ FUER STIMME 1 KEINE AENDERUNG
140 POKE W1,32+1: REM WELLENFORM SAEGEZAHN FUER STIMME 1, START ATTACK
150 FOR I=200 TO 5 STEP -2
160 POKE F1,I:
170 NEXT I
180 FOR I=150 TO 5 STEP -2
190 POKE F1,I
200 NEXT I
210 POKE W1,0: REM BEENDEN STIMME 1
```

VIBRATO

```
100 POKE SID+3,4 : REM TASTVERHAELTNIS 1:4
110 POKE A1,2*16+9: POKE S1,5*16+9: REM SETZEN DER ADRS WERTE FUER
    STIMME 1
120 POKE F3,120: REM FREQ: FUER STIMME 3 auf 7,5 HZ SETZEN
130 POKE W3,16: REM WELLENFORM DREIECK FUER STIMME 3
140 POKE VOL,128+15: REM STIMME 3 ABSCHALTEN, MAXIMALE LAUTSTAERKE
150 READ FR,DR : REM EINLESEN DER FREQUENZ UND DER NOTENDAUER
```



```
160 IF FR=0 THEN END
170 POKE W1,64+1: REM STIMME 1 ATTACK
180 FOR T = 1 TO DR*3 : REM SCHWINGUNGSDAUER
190 FQ = FR+PEEK(SID+27)/2: REM EINLESEN STIMME 3, EINSTELLEN DER
    FREQUENZ/
200 FH = INT(FQ/256) : FL = FQ AND 255: REM FREQUENZ HIGH; LOW BYTE
210 POKE F1, FL: POKE F1+1,FH
220 NEXT T
230 POKE W1,64 : REM AUSSCHALTEN STIMME 1
240 GOTO 150: REM EINLESEN WEITERER DATEN
250 :
260 DATA 4817,2,5103,2,5407,2
270 DATA 8583,4,5407,2,8583,4
280 DATA 5407,4,8583,12,9634,2
290 DATA 10207,2,10814,2,8583,2
300 DATA 9634,4,10814,2,8583,2
310 DATA 9634,4,8583,12
320 DATA 0,0
```

SIRENE

```
100 R3 = SID+27
110 BYTE=256 : K=3.5 : REM KONSTANTEN
111 POKE F3,5 : REM STIMME 3 FREQUENZ 1/3 HZ
112 POKE W3,16: REM STIMME 3 WELLENFORM DREIECK
120 POKE F1+3,2
130 POKE VOL,128+15: REM AUSSCHALTEN STIMME 3, SETZEN DER LAUTSTAERKE
    AUF MAXIMUM
140 POKE S1,15*16+0: REM STIMME 1 SUSTAIN MAXIMUM, SCHNELLES RELEASE
150 POKE A1,0: REM SCHNELLES ATTACK; SCHNELLES DECAY
160 POKE W1,64+1: REM STIMME 1 ATTACK MIT RECHTECK WELLENFORM
170 FR = 7000: REM STIMME 1 FREQUENZ 440 Hz
180 FOR T=1 TO 400
190 FQ=FR+ K*PEEK(R3): REM EINSTELLEN DER FREQUENZ
200 FH = INT(FQ/BY): FL = FQ-BY*FH
210 POKE F1,FL: POKE F1+1, FH: REM NEUE FREQUENZ ANWAehlen
```

220 NEXT T

230 POKE W1,0 : REM BEENDEN STIMME1

HAEMMERN

100 RF = SID+23: REM RESONANZFILTER

110 FC = SID+21: REM EINSTELLEN DES FILTERPUNKTES

120 POKE F1+1,30: REM STIMME 1 FREQUENZ 480 HZ

130 POKE A1,0+6: REM SCHNELLES ATTACK, MITTLERES DECAY

140 POKE S1,0: REM KEIN SUSTAIN, SCHNELLES RELEASE

150 POKE FC+1,150: REM FILTERFREQUENZ

160 POKE RF,0+1: REM KEINE RESONANZ, VERBINDUNG STIMME 1 ZUM FILTER

170 POKE VOL,64+15: REM HOCHPASSFILTER; MAXIMALE LAUTSTAERKE

180 FOR I=1 TO 15

190 POKE W1,128+1: REM ATTACK STIMME 1 RAUSCHEN

200 FOR J=1 TO 200: NEXT J: REM VERZOEGERUNG

210 POKE W1,128: REM AUSSCHALTEN STIMME 1

220 FOR J=1 TO 100 : NEXT J: REM VERZOEGERUNG

230 NEXT I

240 END

MOSKITO

100 POKE F1+1,100 : REM STIMME 1, FREQUENZ 1600 Hz

110 POKE A1,13*16+13: REM STIMME 1, LANGSAMES ATTACK; LANGSAMES DECAY

120 POKE S1,0 : REM STIMME 1 KEIN SUSTAIN, SCHNELLES RELEASE

130 POKE F3+1,28: REM STIMME 3 FREQUENZ 450 Hz

140 POKE VOL,15 : REM MAXIMALE LAUTSTAERKE

150 POKE W1,16+2+1: REM STIMME 1, WELLENFORM DREIECK, SYNCHRONISIERT
MIT STIMME 3, ATTACK

160 FOR I = 1 TO 5000 : NEXT I : REM VERZOEGERUNG

170 POKE W1, 16+2: REM AUSSCHALTEN STIMME 1

180 FOR I=1 TO 1000: NEXT I : REM VERZOEGERUNG

190 POKE W1,0 : REM AUSSCHALTEN STIMME 1

200 END

GLOCKENSPIEL

```
100 POKE F1+1,130: REM STIMME 1 FREQUENZ 2000 HZ
110 POKE A1,0+9: REM STIMME 1, SCHNELLES ATTACK, MITTLERES DECAY
120 POKE F3+1,30: REM STIMME 3, FREQUENZ 480 HZ
130 POKE VOL,15: REM MAXIMALE LAUTSTAERKE
140 FOR I=1 TO 12: REM 12 SCHLAEGE
150 POKE W1,16+4+1: REM STIMME 1 WELLENFORM DREIECK; RINGMODULATION MIT
    STIMME 3, ATTACK
160 FOR J=1 TO 500: NEXT J: REM VERZOEGERUNG
170 POKE W1,16+4: REM AUSSCHALTEN STIMME 1
180 FOR J = 1 TO 100: NEXT J: REM VERZOEGERUNG
190 NEXT I : REM NAECHSTER GLOCKENSCHLAG
200 POKE W1,0: REM AUSSCHALTEN STIMME 1
210 END
```

Anhang 4: Fehlermeldungen

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
BAD SUBSCRIPT ERROR
```

Ein Index (= Subscript) wird in Zusammenhang mit Feldern benutzt, die Sie mit der DIMension-Anweisung vereinbart haben. Wie Sie wissen, müssen Felder mit maximal 10 Elementen nicht dimensioniert werden. Sie erhalten den Fehler immer dann, wenn das Programm versucht, ein Element eines Feldes anzusprechen, dessen Nummer außerhalb des in der DIM-Anweisung vorgegebenen Bereichs liegt.

```
*****
```

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
CAN'T CONTINUE ERROR
```

CONT kann benutzt werden, um ein Programm fortzusetzen, nachdem es mit der RUN/STOP-Taste angehalten wurde oder der Befehl STOP gegeben wurde. Wenn Sie irgendeine Zeile in dem Programm ändern, werden Sie obige Fehlermeldung erhalten. Sie erhalten diesen Fehler auch dann, wenn das Programm vorher nicht mit RUN gestartet wurde. In diesem Fall müssen Sie das Programm mit RUN erneut starten.

```
*****
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
EXTRA IGNORED
```

Mit der INPUT-Anweisung ist festgelegt, wieviele Daten eingegeben werden müssen. Jedes Datum wird mittels Komma von dem folgenden Datum getrennt. Werden zu viele Daten eingegeben, erhalten Sie diese Fehlermeldung. Nur die durch die INPUT-Anweisung gewünschte Anzahl von Daten wird vom Computer berücksichtigt. Sie sollten daran denken, daß bei Eingabe einer Zeichenkette Kommata als Separationszeichen verschiedener Strings interpretiert werden.

```
*****
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
FILE NOT FOUND
```

Diese Meldung erscheint auf dem Bildschirm, wenn Sie versuchen, ein Programm oder einen File von der Diskette zu laden und dieser File nicht auf der Diskette abgespeichert ist. Wenn Sie sicher sind, das Programm doch abgespeichert zu haben, schauen Sie sich das Disk Directory an. Sie haben den Namen des Programms oder des Files möglicherweise falsch geschrieben.

```
*****
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
FILE NOT OPEN ERROR
FILE OPEN ERROR
```

Wenn Sie für die Handhabung von Dateien die Befehle PRINT#, INPUT# und GET# verwenden, müssen Sie zuvor einen entsprechenden File eröffnet haben. Anderenfalls werden Sie die Meldung FILE NOT OPEN ERROR erhalten. Diese Fehlermeldung tritt auch dann auf, wenn Sie mit dem Befehl CMD arbeiten, um Informationen auf dem Bildschirm oder auf dem Drucker auszugeben. Sie müssen nämlich auch hier vorher mit OPEN den entsprechenden File eröffnen.

Die Fehlermeldung FILE OPEN ERROR erscheint, wenn Sie einen File eröffnen wollen, der bereits eröffnet und mit dem Befehl CLOSE noch nicht geschlossen wurde. Achten Sie deshalb immer darauf, daß Sie nach Benutzung eines Files diesen mit dem Befehl CLOSE wieder schließen.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
ILLEGAL DIRECT

Viele BASIC-Befehle können direkt von der Tastatur aus durchgeführt werden und müssen nicht innerhalb eines Programms verwendet werden. Man spricht dabei von dem 'Direkt-Modus'. Wie man leicht einsehen kann, gibt es Befehle, die logischerweise nicht im Direkt-Modus verarbeitet werden können. Ein Beispiel hierfür ist der INPUT-Befehl, der aus verständlichen Gründen im Direkt-Modus nicht sinnvoll benutzt werden kann.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
ILLEGAL QUANTITY ERROR

Eine Zahl, die als Argument einer Funktion oder einer Anweisung benutzt wurde, liegt außerhalb des erlaubten Bereiches. Passen Sie in diesem Zusammenhang besonders bei der Verwendung des POKE-Befehls auf. Denken Sie daran, daß die hier angegebenen Zahlen zwischen 0 und 255 liegen müssen.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
ILLEGAL QUANTITY ERROR

Integer Variablen (= ganzzahlige Variablen) werden mit einem Prozentzeichen gekennzeichnet, das dem Variablennamen direkt folgt. Wenn dieser

Fehler im Zusammenhang mit einer Integer-Variablen auftritt, sollten Sie sich vergewissern, daß der Wert für diese Variable nicht größer als 32767 oder kleiner als -32768 ist.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
NEXT WITHOUT FOR ERROR

Mit dem Befehl NEXT wird eine Schleife sozusagen geschlossen. Wenn Sie den Befehl NEXT ohne eine vorangegangene Eröffnung der Schleife mit dem Befehl FOR verwenden, werden Sie diese Fehlermeldung erhalten. Das kann Ihnen leicht passieren, wenn Sie das Wort NEXT mit einer falschen Variable benutzen.

```
10 FOR I=1 TO 10
20 NEXT K
```

Sie erhalten keine Fehlermeldung, wenn FOR ohne den entsprechenden NEXT-Befehl verwendet wird. In diesem Fall wird die Schleife nicht so ausgeführt, wie sie ausgeführt werden sollte.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
NOT INPUT FILE ERROR
NOT OUTPUT FILE ERROR

Files auf der Datensette können entweder als INPUT- oder als OUTPUT-Files eröffnet werden. Die Auswahl können Sie durch die Sekundäradresse treffen. Die Adresse 1 bedeutet, daß der File auf der Datensette geöffnet wird, um Daten aufzunehmen. Sie können also nicht INPUT# oder GET# benutzen, da Sie sonst erstere Fehlermeldung erhalten. Wird keine Sekundäradresse angegeben, geht der Computer davon aus, daß die auf der Datensette gespeicherte Information wieder abgerufen wird. Wenn Sie in diesem Fall den

Befehl PRINT# eingeben, werden Sie die zweite Fehlermeldung erhalten.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
OUT OF DATA

Zählen Sie die Daten in dem DATA-Statement. Schauen Sie nach, wieviele Da-
ten Sie mit dem READ-Statement einlesen wollen. Wenn Sie versuchen, mit
READ mehr Daten zu lesen, als in den DATA-Statements angegeben sind, er-
halten Sie obige Fehlermeldung.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
OVERFLOW ERROR

Wenn das Ergebnis einer Rechnung größer als die größte erlaubte Zahl (d.h.
1,70141884E+38) ist, wird obige Fehlermeldung ausgegeben. Versuchen Sie
deshalb, den Fehler zu vermeiden, indem Sie nachprüfen, ob diese Zahl
möglicherweise überschritten wird. Sie sollten in einem solchen Fall ver-
suchen, ein Überschreiten der Zahl durch Abbruch des Programms zu umgehen.
Das ist leider nicht immer möglich.

ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
REDIM'D ARRAY ERROR

Ein Feld kann nur einmal DIMensioniert werden. Wird eine Feldvariable (in-
dizierte Variable) aufgerufen, bevor sie dimensioniert wurde, führt der
Rechner eine automatische DIM Operation aus, wobei die DIMension auf 10
gesetzt wird. Jede folgende DIM-Anweisung wird diesen Fehler verursachen.

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
REDO FROM START
```

Diese Fehlermeldung erhalten Sie, wenn Sie mit einem INPUT-Statement die Eingabe einer Zahl anfordern, dann aber eine Zeichenkette eingeben. Korrigieren Sie diesen Fehler, indem Sie einfach die gewünschte Zahl eingeben.

```
*****
```

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
RETURN WITHOUT GOSUB
```

GOSUB und RETURN gehören zusammen. Diese Fehlermeldung wird ausgegeben, wenn das Programm auf den Befehl RETURN trifft, ohne daß es vorher durch den Befehl GOSUB zu der entsprechenden Stelle gekommen ist. Der Fehler tritt meistens dann auf, wenn eine Zeile eines Unterprogramms mit GOTO (und nicht mit GOSUB) angewählt wird.

```
*****
```

```
*****  
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER  
STRING TOO LONG ERROR
```

Ein String kann maximal 255 Zeichen enthalten. Der Fehler tritt oft dann auf, wenn Sie versuchen, zwei Strings mit dem Operator + zu verknüpfen. Prüfen Sie deshalb immer die Länge jeder Zeichenkette und vergewissern Sie sich, daß die zusammengesetzte Zeichenkette nicht größer als 255 Zeichen ist.

```
*****
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
SYNTAX ERROR
```

Ein Syntax Fehler kann die verschiedensten Ursachen haben. Meistens wurde ein BASIC Befehl nicht korrekt eingegeben (beispielsweise werden Kommata und Doppelpunkte sehr oft falsch verwendet).

```
*****
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
TYPE MISMATCH ERROR
```

Dieser Fehler wird erzeugt, wenn Sie anstelle einer Zeichenkette eine Zahl (oder umgekehrt) eingeben. Ein Beispiel hierfür wäre:

10 A\$=9 (oder) A\$=A

oder

10 A= BEISPIEL (oder) A=A\$

```
*****
```

```
*****
ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ERROR ER
UNDEFINED STATEMENT ERROR
```

Sie dürfen die Befehle GOSUB und GOTO nur im Zusammenhang mit Zeilennummern benutzen, die wirklich in Ihrem Programm vorhanden sind. Sehr oft werden Programme oder Zeilennummern geändert und die in dem Programm enthaltenen GOTO- und GOSUB-Anweisungen nicht diesen Änderungen angepaßt. Da der Befehl RUN auch im Zusammenhang mit irgendeiner Zeilennummer benutzt werden kann, können Sie auch hiermit den Fehler UNDEFINED STATEMENT ERROR erzeugen.

```
*****
```

Anhang 5: ASCII/CHR\$ Codes und Konversionstabellen

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	0		17	"	34	3	51
	1		18	#	35	4	52
	2		19	\$	36	5	53
	3		20	%	37	6	54
	4		21	&	38	7	55
	5		22	•	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
Blockieren  	8		25	*	42	;	59
Lösen  	9		26	+	43		60
	10		27	,	44	=	61
	11		28	-	45		62
	12		29	.	46	?	63
	13		30	/	47	@	64
	14		31	0	48	A	65
	15		32	1	49	B	66
	16	!	33	2	50	C	67

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
D	68		99		130		161
E	69		100		131		162
F	70		101		132		163
G	71		102	f1	133		164
H	72		103	f3	134		165
I	73		104	f5	135		166
J	74		105	f7	136		167
K	75		106	f2	137		168
L	76		107	f4	138		169
M	77		108	f6	139		170
N	78		109	f8	140		171
O	79		110	SHIFT RETURN	141		172
P	80		111	Umschaltung auf Großbuchstaben	142		173
Q	81		112		143		174
R	82		113	RIK	144		175
S	83		114	CRSR	145		176
T	84		115	RVS OFF	146		177
U	85		116	CLR HOME	147		178
V	86		117	INST DEL	148		179
W	87		118	Braun	149		180
X	88		119	Hellrot	150		181
Y	89		120	Grau 1	151		182
Z	90		121	Grau 2	152		183
[	91		122	Hellgrün	153		184
£	92		123	Hellblau	154		185
]	93		124	Grau 3	155		186
↑	94		125	PUR	156		187
←	95		126	CRSR	157		188
	96		127	VEL	158		189
	97		128	CYN	159		190
	98	Orange	129	SPACE	160		191

192 bis 223 = 96 bis 127

224 bis 254 = 160 bis 190

255 = 126

Konversionstabelle

<i>BIN</i>	<i>Dez</i>	<i>HEX</i>	<i>BIN</i>	<i>Dez</i>	<i>HEX</i>
0	0	00	110101	53	35
1	1	01	110110	54	36
10	2	02	110111	55	37
11	3	03	111000	56	38
100	4	04	111001	57	39
101	5	05	111010	58	3A
110	6	06	111011	59	3B
111	7	07	111100	60	3C
1000	8	08	111101	61	3D
1001	9	09	111110	62	3E
1010	10	0A	111111	63	3F
1011	11	0B	1000000	64	40
1100	12	0C	1000001	65	41
1101	13	0D	1000010	66	42
1110	14	0E	1000011	67	43
1111	15	0F	1000100	68	44
10000	16	10	1000101	69	45
10001	17	11	1000110	70	46
10010	18	12	1000111	71	47
10011	19	13	1001000	72	48
10100	20	14	1001001	73	49
10101	21	15	1001010	74	4A
10110	22	16	1001011	75	4B
10111	23	17	1001100	76	4C
11000	24	18	1001101	77	4D
11001	25	19	1001110	78	4E
11010	26	1A	1001111	79	4F
11011	27	1B	1010000	80	50
11100	28	1C	1010001	81	51
11101	29	1D	1010010	82	52
11110	30	1E	1010011	83	53
11111	31	1F	1010100	84	54
100000	32	20	1010101	85	55
100001	33	21	1010110	86	56
100010	34	22	1010111	87	57
100011	35	23	1011000	88	58
100100	36	24	1011001	89	59
100101	37	25	1011010	90	5A
100110	38	26	1011011	91	5B
100111	39	27	1011100	92	5C
101000	40	28	1011101	93	5D
101001	41	29	1011110	94	5E
101010	42	2A	1011111	95	5F
101011	43	2B	1100000	96	60
101100	44	2C	1100001	97	61
101101	45	2D	1100010	98	62
101110	46	2E	1100011	99	63
101111	47	2F	1100100	100	64
110000	48	30	1100101	101	65
110001	49	31	1100110	102	66
110010	50	32	1100111	103	67
110011	51	33	1101000	104	68
110100	52	34	1101001	105	69

Konversionstabelle

<i>BIN</i>	<i>Dez</i>	<i>HEX</i>	<i>BIN</i>	<i>Dez</i>	<i>HEX</i>
1101010	106	6A	10011111	159	9F
1101011	107	6B	10100000	160	A0
1101100	108	6C	10100001	161	A1
1101101	109	6D	10100010	162	A2
1101110	110	6E	10100011	163	A3
1101111	111	6F	10100100	164	A4
1110000	112	70	10100101	165	A5
1110001	113	71	10100110	166	A6
1110010	114	72	10100111	167	A7
1110011	115	73	10101000	168	A8
1110100	116	74	10101001	169	A9
1110101	117	75	10101010	170	AA
1110110	118	76	10101011	171	AB
1110111	119	77	10101100	172	AC
1111000	120	78	10101101	173	AD
1111001	121	79	10101110	174	AE
1111010	122	7A	10101111	175	AF
1111011	123	7B	10110000	176	B0
1111100	124	7C	10110001	177	B1
1111101	125	7D	10110010	178	B2
1111110	126	7E	10110011	179	B3
1111111	127	7F	10110100	180	B4
10000000	128	80	10110101	181	B5
10000001	129	81	10110110	182	B6
10000010	130	82	10110111	183	B7
10000011	131	83	10111000	184	B8
10000100	132	84	10111001	185	B9
10000101	133	85	10111010	186	BA
10000110	134	86	10111011	187	BB
10000111	135	87	10111100	188	BC
10001000	136	88	10111101	189	BD
10001001	137	89	10111110	190	BE
10001010	138	8A	10111111	191	BF
10001011	139	8B	11000000	192	C0
10001100	140	8C	11000001	193	C1
10001101	141	8D	11000010	194	C2
10001110	142	8E	11000011	195	C3
10001111	143	8F	11000100	196	C4
10010000	144	90	11000101	197	C5
10010001	145	91	11000110	198	C6
10010010	146	92	11000111	199	C7
10010011	147	93	11001000	200	C8
10010100	148	94	11001001	201	C9
10010101	149	95	11001010	202	CA
10010110	150	96	11001011	203	CB
10010111	151	97	11001100	204	CC
10011000	152	98	11001101	205	CD
10011001	153	99	11001110	206	CE
10011010	154	9A	11001111	207	CF
10011011	155	9B	11010000	208	D0
10011100	156	9C	11010001	209	D1
10011101	157	9D	11010010	210	D2
10011110	158	9E	11010011	211	D3

Konversionstabelle

<i>BIN</i>	<i>Dez</i>	<i>HEX</i>	<i>BIN</i>	<i>Dez</i>	<i>HEX</i>
11010100	212	D4	11101010	234	EA
11010101	213	D5	11101011	235	EB
11010110	214	D6	11101100	236	EC
11010111	215	D7	11101101	237	ED
11011000	216	D8	11101110	238	EE
11011001	217	D9	11101111	239	EF
11011010	218	DA	11110000	240	F0
11011011	219	DB	11110001	241	F1
11011100	220	DC	11110010	242	F2
11011101	221	DD	11110011	243	F3
11011110	222	DE	11110100	244	F4
11011111	223	DF	11110101	245	F5
11100000	224	E0	11110110	246	F6
11100001	225	E1	11110111	247	F7
11100010	226	E2	11111000	248	F8
11100011	227	E3	11111001	249	F9
11100100	228	E4	11111010	250	FA
11100101	229	E5	11111011	251	FB
11100110	230	E6	11111100	252	FC
11100111	231	E7	11111101	253	FD
11101000	232	E8	11111110	254	FE
11101001	233	E9	11111111	255	FF

Glossar

Array

Ein Array ist ein Feld, in dem verschiedene Informationen, die dem gleichen Datentyp angehören müssen, zusammengefaßt werden.

ASCII

ASCII ist die Abkürzung für American Standard Code For Information Interchange und enthält die Codierung von 128 Zeichen, die in Buchstaben, Ziffern, Symbole oder Signale unterteilt werden können. Commodore Computer benutzen eine Variante des ASCII Codes, der insgesamt 255 Code Zeichen zuläßt.

BASIC

BASIC ist eine Abkürzung für Beginners All Purpose Symbolic Instruction Code. BASIC ist eine Programmiersprache, die hauptsächlich zur Programmierung der Personal Computer eingesetzt wird.

Betriebssystem

Das Betriebssystem (operating system) ist die Summe aller Programme, die für den Betrieb eines Rechners notwendig sind.

Bildschirmspeicher

Der Teil des Speichers, der den Text oder die Bilder enthält, die auf dem Bildschirm ausgegeben werden, wird als Bildschirmspeicher bezeichnet.

Bit

Ein Bit ist die kleinste Informationseinheit. Ein Bit kann nur zwei Zustände bezeichnen, die meistens mit den Ziffern 0 und 1 symbolisiert werden. Ein Bit kann mit einem Schalter verglichen werden, der ein- bzw. ausgeschaltet sein kann. 8 Bit werden zu einem Byte zusammengefaßt.

Byte

Ein Byte ist die Standardgröße eines Speicherplatzes. Ein Byte kann Zahlen von 0-255 enthalten und besteht aus 8 Bit.

Copy Protect

Copy Protect bezeichnet eine Methode, mit der Sie sich vor unberechtigten Kopien Ihres Programms schützen können.

Crash

Ein Crash tritt auf, wenn ein Computer nicht ordnungsgemäß arbeitet.

Cursor

Mit 'Cursor' wird die jeweils laufende Eingabestelle auf dem Bildschirm bezeichnet. Meistens wird der Cursor durch ein blinkendes Rechteck angezeigt.

Datenbank

Eine Datenbank ist eine organisierte und strukturierte Sammlung von Informationen, die durch ein Programm, das als 'Datenbank Manager' bezeichnet wird, gehandhabt werden können.

Datum

Unter einem Datum versteht man irgendeine Information.

Dekrement

Wenn eine Zahl um 1 verkleinert wird, wird sie 'dekrementiert'.

DOS

DOS ist ein Betriebssystem (Disk Operating System), das den Einsatz eines Disketten- oder eines Plattenlaufwerks ermöglicht.

Floppy Disk

Die Floppy Disk ist eine flexible und magnetisierbare Scheibe auf Kunststoffbasis. Auf dieser Floppy Disk können Informationen gespeichert werden.

Grafik Modus

In dem Grafik Modus können Sie nur Großbuchstaben und (anstelle der Kleinbuchstaben) grafische Symbole benutzen.

Hacker

Ein Hacker ist ein außerordentlicher Computerfanatiker.

Inkrement

Wenn eine Zahl um 1 erhöht wird, wird sie um 1 'inkrementiert'.

Integer

Mit Integer werden ganze Zahlen bezeichnet.

Interface

Ein Interface ist eine elektronische Schaltung, die es erlaubt, daß externe (Peripherie-) Geräte an einen Computer angeschlossen werden können. Normalerweise werden sie dazu verwendet, um Drucker oder Floppy Disk Laufwerke an den Rechner anzuschließen.

BASIC Interpreter

Ein BASIC Interpreter ist ein Programm, das im Gegensatz zum Compiler das entsprechende BASIC Programm nicht in Maschinsprache übersetzt, sondern jeden BASIC Befehl quasi während der Ausführung entschlüsselt.

K

K ist die Abkürzung für 'Kilo' und bedeutet 1000. Ein K Speicher enthält allerdings 1024 - und nicht 1000 - Bytes.

Kaltstart

Der Computer wird mit einem Kaltstart in seinen 'Urzustand' zurückversetzt. Das bedeutet, daß der Inhalt des Speichers und damit sämtliche Programme, die in dem Speicher enthalten waren, gelöscht werden. Sie erzeugen einen Kaltstart, indem Sie den Computer aus- und wieder einschalten.

Kassettenpuffer

Der Kassettenpuffer ist ein Speicherbereich im Arbeitsspeicher des Rechners, der zur temporären Speicherung von Daten verwendet wird, die von der Datensette in den Computer geladen, bzw. von dem Computer auf die Datensette gespeichert werden.

Kompiliertes BASIC

BASIC kann kompilativ ausgeführt werden. BASIC Programme werden somit erst mit einem Übersetzer (= Compiler) in Maschinensprache überführt. Der Vorteil ist, daß ein kompiliertes BASIC Programm erheblich schneller als ein Programm, das mit einem BASIC Interpreter ausgeführt wird, abläuft.

Maschinencode

Der Maschinencode ist der Elementarcode, der direkt vom Computer verstanden werden kann. Der Maschinencode wird auch 'Assembler Code' genannt.

Menü

Ein Menü ist eine Liste verschiedener Aktivitäten, unter denen der Benutzer eines Programms wählen kann.

Mikroprozessor

Unter einem Mikroprozessor versteht man einen integrierten elektronischen Schaltkreis, der programmiert werden kann und seinen eigenen Befehlssatz hat.

Nesting

Geschachtelte Schleifen werden auch als 'Nest' bezeichnet.

Nullstring

Eine leere Zeichenkette, die durch zwei aufeinanderfolgende Anführungszeichen gekennzeichnet ist, wird als Nullstring bezeichnet.

Nybble

Ein halbes Byte, also 4 Bit, sind ein Nybble.

Parallel

Bei der Datenübertragung können die einzelnen 8 Bit einer Information entweder seriell (hintereinander) oder parallel (gleichzeitig) übertragen werden.

Pixel

Ein Pixel ist ein Bildelement, das aus einem oder mehreren Punkten bestehen kann und mit hochauflösender Grafik erzeugt wird.

Programm

Unter einem Programm versteht man eine Liste von Befehlen, die in einer Computersprache formuliert wurden.

Programmiersprache

In einer Programmiersprache werden Befehle, Regeln und Schlüsselwörter zusammengefaßt, mit denen Sie die Lösung eines Programms formulieren können.

RAM

RAM ist die Abkürzung für Random Access Memory (= Direktzugriffsspeicher). In diesem Speicher können Informationen gespeichert und wieder gelöscht werden. Normalerweise ist die Information verloren, wenn der Computer ausgeschaltet wird.

ROM

ROM ist die Abkürzung für Read Only Memory. In einem ROM ist die Information permanent gespeichert. ROM Speicher können nur gelesen werden. Die Information bleibt erhalten, wenn der Computer ausgeschaltet wird.

Schleife

Unter einer Schleife versteht man einen Programmabschnitt, der wiederholt ausgeführt werden soll. Schleifen werden in BASIC mit dem GOTO oder dem FOR..NEXT Befehl programmiert.

Sequentieller File

Ein File auf Diskette oder Datassette, der Informationen aber kein Programm enthält, wird als sequentieller File bezeichnet.

Seriell

Bei einer seriellen Datenübertragung werden die einzelnen Bit eines Byte nacheinander übertragen.

Speicherbelegungstabelle

Aus der Speicherbelegungstabelle geht hervor, für welchen Zweck bestimmte Speicherplätze des Computers verwendet werden.

Sprite

Ein Sprite ist ein Bild, das mit hochauflösender Grafik erzeugt wird. Ein Sprite wird im Speicher eines Rechners definiert und kann irgendwo auf dem Bildschirm erscheinen.

Strukturierte Programmierung

Unter strukturierter Programmierung versteht man eine Technik, bei der die Benutzung des GOTO Befehls nach Möglichkeit vermieden wird. Bei konsequenter Anwendung der strukturierten Programmierung erhält man leicht zu lesende Programme.

Syntax

Unter Syntax versteht man den formalen Aufbau eines Befehls.

Unterprogramm

Ein Unterprogramm ist ein Programm innerhalb eines Programms, das mit dem Befehl GOSUB aufgerufen werden kann.

Warmstart

Mit einem Warmstart wird ein 'Urzustand' des Computers hergestellt, bei dem der Inhalt des Speichers nicht zerstört wird. Ein eingegebenes Programm bleibt also erhalten.

Word Processing

Word Processing ist die Textverarbeitung mit einem Computer.

Zeichenkette

Eine Zeichenkette enthält eine lineare Aneinanderreihung einer endlichen Anzahl von Zeichen. Diese Zeichen können alphabetische, numerische Zeichen oder Sonderzeichen sein.

Zero Page

Mit Zero Page wird die 'erste Seite' des Arbeitsspeichers bezeichnet. Sie enthält die Speicherzellen von 0 bis 255.

Schlagwortverzeichnis

- ABS 137,144
- Adresse 208,288
- Amplitude 285
- AND 231,47
- Anführungszeichen 49
- Anschlag 190
- Anweisungsnummer 56,81
- Anweisungszeile 56
- Arbeitsprogramm 209
- ASC 137,146,148
- Assembler Sprache 205
- ATMS 154
- ATN 47,137
- Attack 294
- A-V Port 12

- BASIC 53
- BASIC Compiler 54,205
- BASIC Dialekte 55
- Bausteinverarbeitung 158
- Betriebssystem 39,230
- Bildschirm 9,87
- Binärzahl 232,255
- Bit Map 265
- Block 18,36
- BREAK 141
- Byte 228

- Character Grafik 177,184
- CHR\$ 125,137,146
- CLOSE 40,99,141
- CLR/HOME Taste 25,99
- CMD 137,140
- COBOL 53
- COMMODORE Taste 21
- CONT 142
- Control Port 12,168,198,206
- COS 47,137,145
- CRSR Tasten 24,49
- CTRL Taste 22
- Cursor 13,15,75,245

- DATA 257,99,107,192
- DATA Block 192

- Datassette 15,110,115,200
- Datenbankprogramm 128
- Decay 294
- DEF 137,145
- Device Nummer 111
- Dezimalzahl 61,227
- Dialekt 204
- DIM 99,103
- Directory 18,36,38
- Direktmodus 73,80,138,146
- Diskette 17,110
- Diskettenlaufwerk 17,44,200
- Diskettenspeicher 35
- Doppelpunkt 42,65
- Drucker 120,155

- Easy Script 164
- Editing Tasten 24
- Editor 57
- Einheitennummer 40
- Elektronengehirn 1
- Emulator 46
- END Befehl 58,96
- ENIAC 1

- Farbmonitor 10
- Fernsehapparat 198,92
- File 110
- Filenummer 40,111
- Flexibilität 124
- Floppy Disc 13,14
- FN 137
- FOR..NEXT 77,89
- FRE 48
- Frequenz 283
- Funktionstasten 23,87,164

- Game Paddle 168,198
- Geräusch 190
- GET 7785,147,186,212
- GET\$ 99,110
- GOSUB..RETURN 77,93,94
- GOTO 77,82,186
- Grafikmodus 169

Grafiksymbole 21,178,228,256
grafische Darstellung 168
Großschrift/Grafikmodus 20
Groß-/Kleinschriftmodus 20

Halten 190
Hertz 170
Hintergrundfarbe 174
Hochauflösungsgrafik 169,272
HOME Position 180
Hüllkurve 288

IF 77,82
Index 100,102
indizierte Variable 101,102
Initialisieren 36
INPUT 77
INPUT# 99,110
INST/DEL Taste 25
INT 137,144
Integer Variable 61
Interface 157
Interpreter 54

Joystick 168,197

Kassettenspeicher 35
Klang 190
Klangfarbe 171,191
Komma 70
Kommando 232
Kompatibilität 45
Koordinaten 267

LAN 201
Lautstärke 170,298
Leerzeichen 68
LEFT\$ 137,146,150
LEN 137,146
LET 64
Lichtgriffel 206
LISP 204
LIST 56,141
LOAD Befehl 29,37
LOG 47
logische Operatoren 231
LOGO 204

Maschinenprogramm 54
Matrix Drucker 155

Memory Map 210
Menü 87,160
Mikroprozessor 3
Modul Steckplatz
Musik 168,211
MID\$ 137,146,151

Netzteil 6
NEXT 77
NOT 47,231
Notenverschlüsselung 308
Notenwert 192

Oktave 299
ON 137,142
OPEN 40,99,111,141
OR 47,231

Pattern Matching 41
PEEK, POKE 46,137-139,190,208,253
PET 4
Pilot 202
Pixel 228,235,254
Plotter 230
Pointer 230,255
POKE siehe PEEK
Position 71
PRG 117
PRINT 46,52,66,176
PRINT# 99,110,121
Prozessor 211
Punktmatrix 265

QWERTY Tastatur 26

Rahmenfarbe 174
RAM 4,29,209,254,269
Rauschen 293
READY 13,99,107
Recorderanschluß 11
Relay 2
Release 191,294
REM 96
Report Generator 155
RESTORE 23,99
RETURN Taste 18,20,28,57,77,127,153
Reverse Modus 22
RIGHT\$ 137,146,151
ROM 4,209,254
RUN 34

- RUN/STOP Taste 20,34,141
RVS OFF 75
RVS ON 75
- SAVE 29,37,42
Sägezahn 293
Schleifen 90
Schnittstelle 158
Schriftart 253
schwarz-weiß Monitor 10
Schwingung 284
Scratch 43
Serieller Ausgang 12
Sektor 36
Sekundäradresse 40,111
Semantik 53,68
Semikolon 68
SEQ 117
SGN 137,144
SHIFT 20, 253
SID 189,282,287
SIN 47,137,145
Spannungsversorgung 11
SPC 137,141
Speicher 208
Speicherbelegungstabelle 210
Speicherbereich 213
Speicherkapazität 43
Speicherplatz 259
Speicherzelle 208
Spreadsheet 201
Sprite 226,228,233
Sprite Block 236
Sprite Generator 202
Sprite Grafik 184
Spur 36
SQR 47,137
Startadresse 298
STEP 92
STOP 137,141
String Variable 119
STR\$ 137
Sustain 191,294
SYNTAX 53,122,204
Syntax Fehler 65
SYS 273
SYS 137,139
- TAB 137,141
TAN 47,137,145
Tastatur 7,14,19
Tastverhältnis 293
Text Editor 155
Textautomat 155
Texteingabe 155
Textsystem 166
Textvariablen 178
Tonhöhe 170
Turtle 204
TV Anschluß 11
Typenrad Drucker 155
- Uhr 49
Umsetzer 46
Unterprogramm 93,153
User Port 12
USR 137,140
- VAL 137,146,147
Variable 58,175
Variablenname 62
VERIFY Befehl 33,43
VIC 169,230
Video Chip 254
Video Spiele 197
Videokabel 6
VisiCalc 202
- WAIT 137,142
Warteschleife 91,302
Wellenform 291
Word Processor 159
WordPro 161
Würfelspiel 174
- Zeichengenerator 256
Zeichenkette 50,59,127
Zeichenmodus 169,279
Zeichensatz 157,253
Zeiger 230
Zufallszahlen 171

Danksagungen

Wie jedes andere Buch, ist auch dieses Buch nicht ganz ohne fremde Hilfe geschrieben worden. Ich möchte an dieser Stelle denjenigen danken, die für meine Arbeit Interesse gezeigt haben und mir mit Informationen, Tips und Ratschlägen geholfen haben.

Ich danke Herrn Steve Murri, Herrn Bob Fairbairn und Frau Diane LeBold von der Firma Commodore Business Machines. Die Zusammenarbeit mit ihnen hat mir sehr geholfen.

Ich möchte weiterhin denjenigen danken, die das Manuskript auf Fehler geprüft haben. Ich danke besonders Herrn Gene Streitmatter, dessen Kommentare für mich sehr wertvoll waren und mir bei der Arbeit an meinem Buch sehr geholfen haben.

Ich danke Herrn Michael Richter, dessen Programme zur Vorbereitung der Programmlistings verwendet wurden. Herrn William Seiler danke ich für seine freundschaftliche Hilfe und sein umfangreiches technisches Wissen, das er mir zur Verfügung gestellt hat.

Meinen Mitautoren Dr. Frank Covitz, Jim Butterfield und besonders Paul Schatz möchte ich für die Abhandlung spezieller Themen im Anhang dieses Buches danken.

Mein Dank geht außerdem an den Verlag Prentice-Hall und an seine Mitarbeiter Robert J. Brady, Harry Gaines, David Culverwell, Terry Anderson und Laura Dysart Marcy.

Ich möchte es nicht versäumen, auch meiner Familie und besonders meinem Vater zu danken, der sich schon seit langer Zeit mit dem Themenbereich 'Computer' befaßt hat und seine Entwicklungen frühzeitig abschätzen konnte.

Schließlich möchte ich Frau Beth Abroahams für ihre Unterstützung danken, die mir am meisten bei diesem Buch geholfen hat.

PCprofessionell

Arbeitsplatzrechner in Ausbildung und Praxis

Herausgeber: Prof. Dr.-Ing. Werner Heinzl

Banahan/Rutter, UNIX – Lernen, Verstehen, Anwenden

Feuer, Das C Puzzle Buch

Fuchs, Btx-Anwendungen mit Arbeitsplatzrechnern

Germain, Das Programmierhandbuch für den IBM PC und XT

Gillner, Datenbanken auf Arbeitsplatzrechnern

Heinzl, Arbeitsplatzrechner

Kernighan/Pike, Das Betriebssystem UNIX

Kernighan/Ritchie, Programmieren in C

Kwiatkowski, Textverarbeitung mit Arbeitsplatzrechnern

Meißner, Arbeitsplatzrechner im Verbund

Partosch, PASCAL mit Arbeitsplatzrechnern

Shoup, Numerische Verfahren mit Arbeitsplatzrechnern

Carl Hanser Verlag, 8000 München 86





Die authentische Übersetzung des außerordentlich erfolgreichen Standardwerkes

Von Brian W. Kernighan und Dennis M. Ritchie. Die deutsche Ausgabe besorgten Prof. A. T. Schreiner und Dr. E. Janich. Reihe: PC, professionell (Arbeitsplatzrechner in Ausbildung und Praxis). 276 Seiten. 1983. Kartoniert.

C ist eine universelle Programmiersprache, die alle modernen Methoden des Software-Engineering unterstützt. Im Vergleich zu PASCAL und ADA erlaubt sie auch hardware-nahe Konzeptionen und unterstützt somit nicht nur Anwendungsprogrammierer aller Sparten, sondern auch die Systemprogrammierung. Im Gefolge der Verbreitung des UNIX-Betriebssystems findet C heute vielfältige Verwendung, besonders auf den neuen 16-Bit-Computern vom Mikro bis zum mittleren System.

Dieses Buch ist die leicht aktualisierte Übersetzung des von den Sprachautoren selbst herausgegebenen „Standards“ – eine leicht verständ-



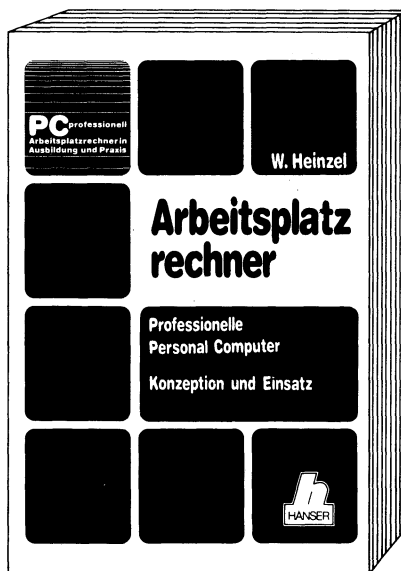
liche, informelle Einweisung in C mit allen Feinheiten. Die Autoren bedienen sich dabei vieler Beispiele, die meist vollständige, „echte“ Programme darstellen, und verzichten auf das formale Regelwerk einer abstrakten Sprachbeschreibung. Die Kenntnis einiger Grundbegriffe wie Variable, Schleife usw. wird vorausgesetzt. Neben praktischen Übungsaufgaben bietet das Buch viele Algorithmen und Design-Tips. Ein Stichwortverzeichnis hilft beim Nachschlagen.

Carl Hanser Verlag
Postfach 86 04 20, 8000 München 86



PC^{professionell}

Arbeitsplatzrechner in Ausbildung und Praxis



Professionelle Personal Computer – Konzeption und Einsatz. Von Prof. Dr.-Ing. Werner Heinzel, Fachhochschule Fulda. Reihe: PC professionell (Arbeitsplatzrechner in Ausbildung und Praxis). 205 Seiten. 1984. Kartiert.

Das Buch führt in die verschiedenen Hard- und Softwarekonzepte von Arbeitsplatzrechnern bzw. Personal Computern ein. Aufgrund des Aufbaus dieses Werkes eignet sich das Buch auch zum Selbststudium und wendet sich u.a. an Praktiker, die anstehende Aufgaben mit den Mitteln der DV-Technik lösen wollen und an Studenten der Fachrichtungen Betriebswirtschaft, Informatik und Ingenieurwissenschaften.

Inhaltsübersicht: Der Weg zum Arbeitsplatzrechner – Konzeption von Arbeitsplatzrechnern – Peri-

pherie-Schnittstellen – Gerätestruktur – Peripherie-Systeme – Kopplung von Arbeitsplatzrechnern – Betriebssysteme – Sprachsysteme und Softwarewerkzeuge – Typische Arbeitsplatzrechner – Aspekte der Datensicherung und des Datenschutzes – Gesichtspunkte zur Systemauswahl – Einsatzalternativen – Ausblick – Literaturverzeichnis – Anlagen.

Carl Hanser Verlag
Postfach 8604 20, 8000 München 86



PC Fachbücher für Einsteiger und Anwender



Vom Einstieg bis zum Standard. Von Leo Brodie. 460 Seiten. 1984.
Kartoniert. Coedition: Carl Hanser Verlag/Prentice Hall International.

Im englischen Sprachraum hat dieses Buch eine außerordentliche Verbreitung gefunden und sich zum Standardwerk der Programmiersprache FORTH entwickelt. Als umfassender und gut verständlicher Leitfaden führt es von einer Einführung und den fundamentalen Grundregeln bis zu anspruchsvollen, FORTH-Anwendungen und Programmiertechniken.

Charles H. Moore, der die Programmiersprache FORTH entwickelt hat, sagt im Geleitwort über dieses Buch: „Die FORTH-Gemeinde kann mit der Publikation von »Program-

mieren in FORTH« ein bedeutendes Ereignis feiern. In diesem Buch stecken mehr Mühe, Talent und engagierte Verpflichtung als in irgendeiner früheren Einführung.“ Es gibt wirklich keinen besseren Weg, FORTH zu lernen, als dieses vernünftig geschriebene Fachbuch durchzuarbeiten.

Carl Hanser Verlag
Postfach 8604 20, 8000 München 86

Dieser handliche Leitfaden bringt eine vollständige Beschreibung aller drei Versionen des Commodore 64.

Dieses Buch wurde speziell für diejenigen Commodore-Benutzer entwickelt, die geringe oder gar keine Computererfahrung haben.

Es beschreibt darüber hinaus eine Vielzahl von Anwendungen, deren Spektrum von der Textverarbeitung und Computergrafik bis zu Computerspielen und zur Ton- und Musikerzeugung reicht.