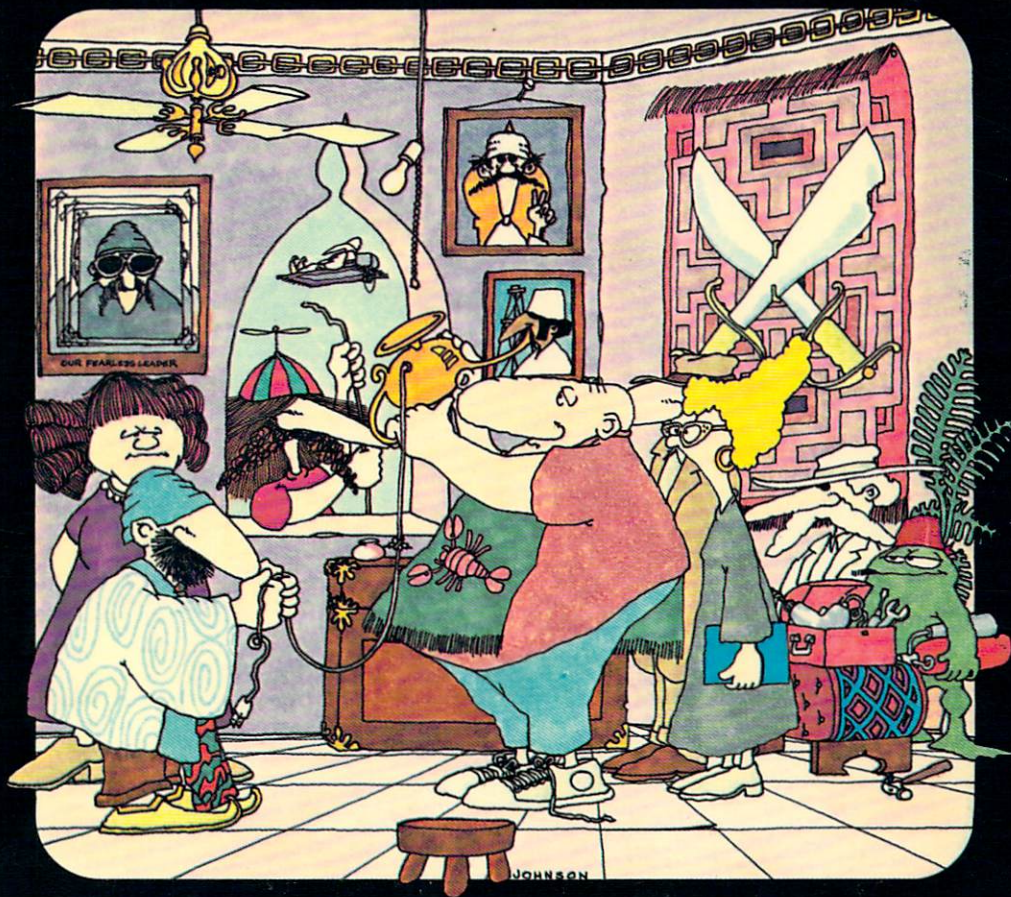


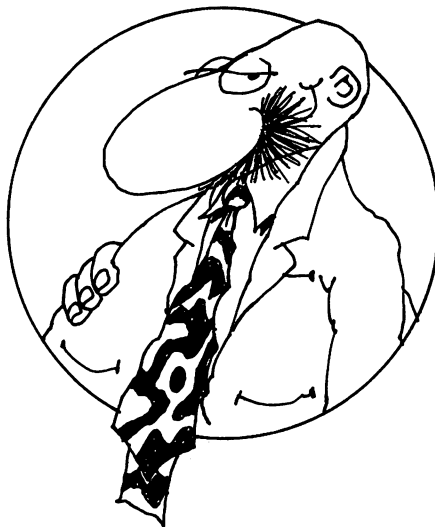
DR. C. WACKO  PRESENTS
COMMODORE 64 BASIC
&
THE WHIZ-BANG MIRACLE MACHINE



David L. Heller & John F. Johnson

Dr. C. Wacko Presents Commodore 64 BASIC and The Whiz-Bang Miracle Machine

David L. Heller and John F. Johnson



ADDISON-WESLEY PUBLISHING COMPANY, INC.
Reading, Massachusetts • Menlo Park, California • Don Mills, Ontario
Wokingham, England • Amsterdam • Sydney • Singapore • Tokyo
Mexico City • Bogotá • Santiago • San Juan

Cover and Book Design-Teapot Graphics/John Johnson

Commodore 64 is a registered trademark of Commodore Business Machines, Inc.

Library of Congress Cataloging in Publication Data

Heller, David L.

Dr. C. Wacko presents COMMODORE BASIC and the whiz bang miracle machine.

Includes index.

1. Commodore 64 (Computer)—Programming. 2. Basic (Computer program language) I. Johnson, John F. II. Title. III. Title: Dr. C. Wacko presents COMMODORE BASIC and the whiz bang miracle machine.

QA76.8C64H45 1984 001.64'2 84-21712

ISBN 0-201-11494-1

Copyright © 1985 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

Printed from camera ready boards supplied by Teapot Graphics, Santa Cruz, California.

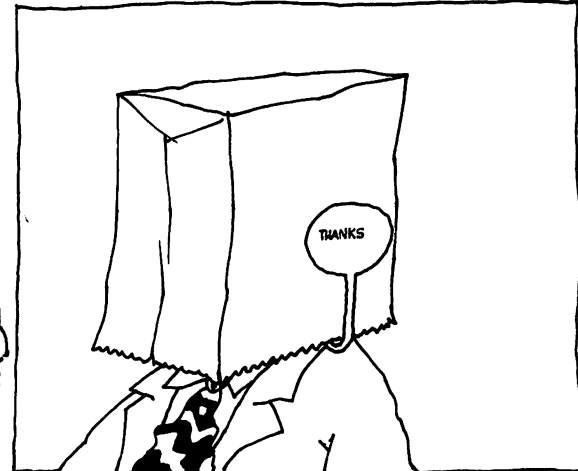
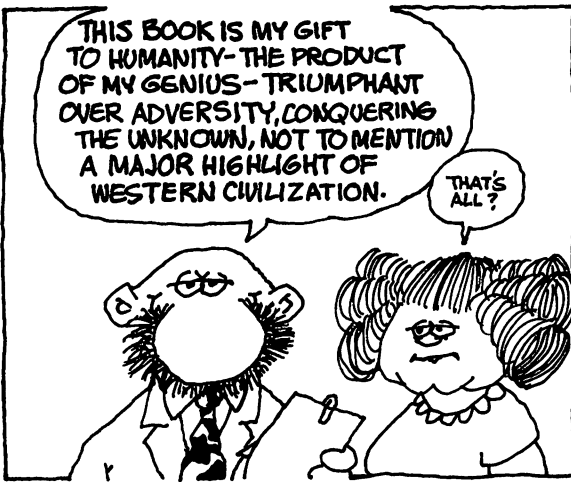
ABCDEFGHIJ-HA-898765

First printing, March 1985

Contents

| | |
|---|------------|
| Foreword: Preramble to the Convolution | 1 |
| 1. Limber Up Your Fingertips | 10 |
| 2. Basic BASIC Training | 22 |
| 3. Entering The Great White Expanse | 54 |
| 4. Graphics Power: An End to The Desert Blues | 156 |
| 5. Desert ZOUNDS! | 184 |
| 6. Weaving The Perfect Flying Carpet, or, The Art of Programming | 208 |
| Appendix A: Programming Conventions | 241 |
| Appendix B: Loading and Saving Files | 243 |
| Appendix C: ASCII Codes | 245 |
| Appendix D: Sneaky Peek's Pokes & Peeks | 247 |
| Appendix E: Screen Display Codes | 250 |
| Index | 252 |

DR. WACKO PRESENTS COMMODORE BASIC



Foreword

Preramble to The Convolution



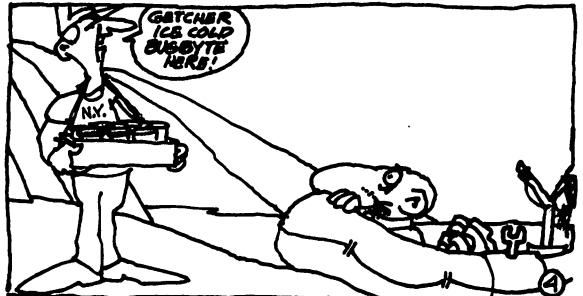
IT ALL STARTED MANY YEARS AGO WHILE I WAS STEAMING OFF THE COAST OF MADAGASCAR WITH THE NEPALESE NAVY.



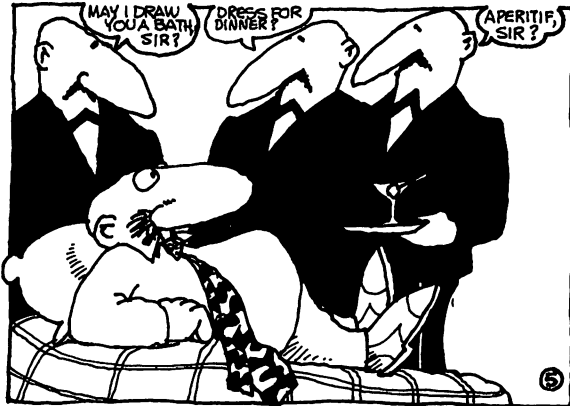
A SUDDEN SQUALL HAD COME UP, POUNDING THE FLEET LIKE MATCH BOXES. IT DAMPENED MY ENTHUSIASM SOMEWHAT...



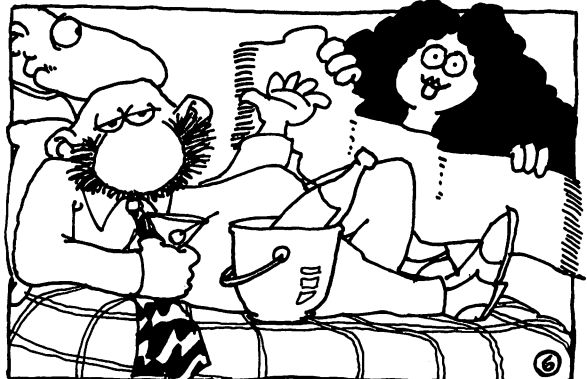
WE TURNED INTO THE WIND AND HEADED FOR A SHELTERED LAGOON NESTLED BETWEEN THE EDGE OF THE DESERT AND THE RAGING SEAS



STIFLING HOT, DRY AIR BLOWN FROM THE DESERT COMPOUNDED BY A DESPERATE THIRST MADE ME THINK I WAS HALLUCINATING.



THE FIRST THING I NOTICED WHEN I CAME TO WAS THE INSIDE OF A RIPPLING PURPLE TENT SMELLING OF INCENSE. I WAS SURROUNDED BY A BAND OF WEIRD LOOKING CHARACTERS - WEIRD BECAUSE THEY ALL APPEARED UPSIDE DOWN AS THEY BENT OVER ME. I STRUGGLED TO A SITTING POSITION, AND MUCH TO MY SURPRISE, THEY STILL LOOKED WEIRD.

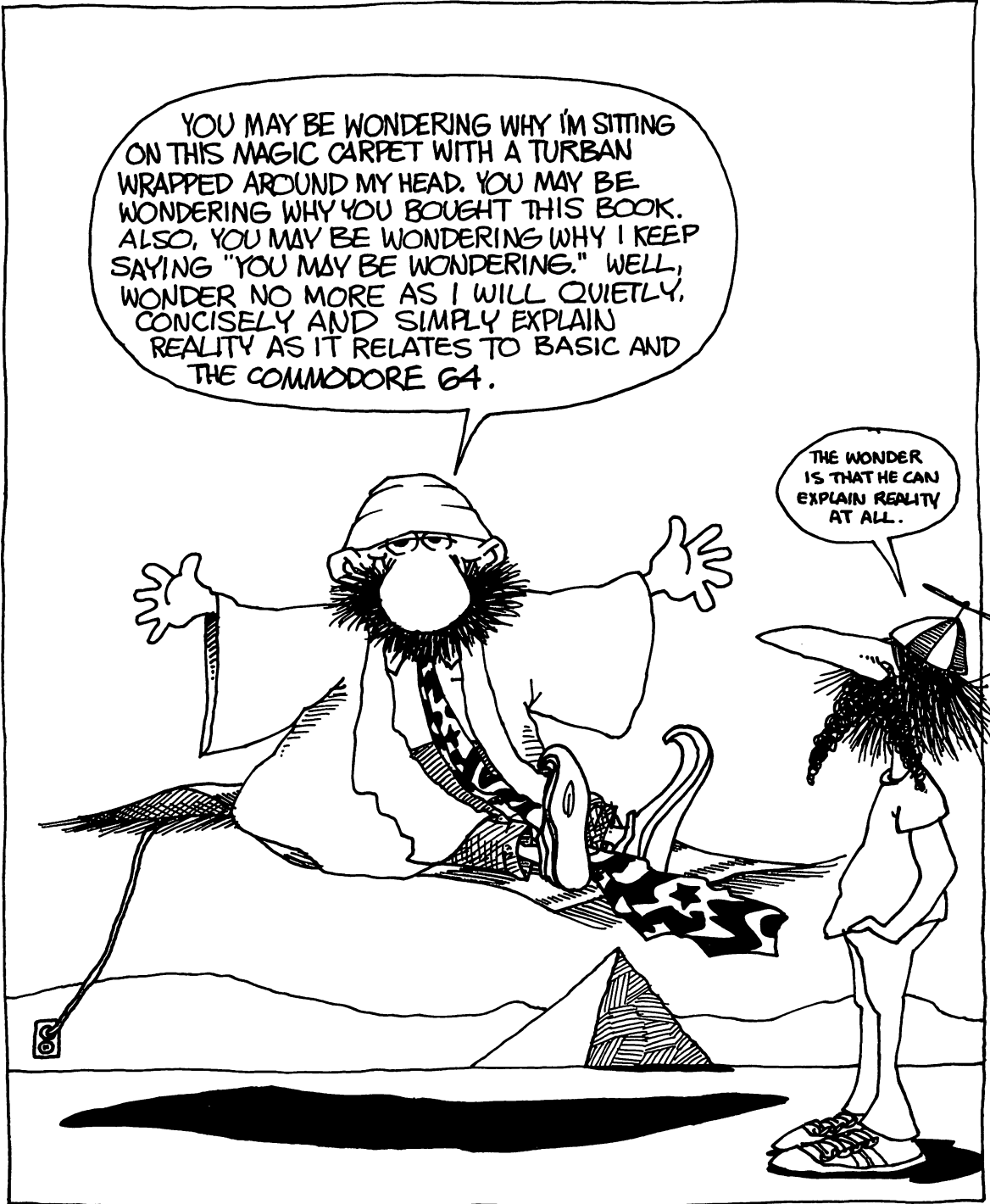


AND SO BEGAN MY ADVENTURE THROUGH THE DESERT IN SEARCH OF THE WHIZ-BANG MIRACLE MACHINE AND ITS UNIVERSAL ABILITY FOR BASIC COMMUNICATION. TO MAKE A LONG STORY LONGER, WE DID INDEED FIND THE MIRACLE MACHINE, AND NOW WE INVITE YOU TO JOIN US FOR THE AMAZING ADVENTURE OF DISCOVERY THROUGH THE DESERT OF KNOWLEDGE. SO SADDLE CLYDE AND TURN THE PAGE.

DR. WACKO PRESENTS COMMODORE BASIC

YOU MAY BE WONDERING WHY I'M SITTING ON THIS MAGIC CARPET WITH A TURBAN WRAPPED AROUND MY HEAD. YOU MAY BE WONDERING WHY YOU BOUGHT THIS BOOK. ALSO, YOU MAY BE WONDERING WHY I KEEP SAYING "YOU MAY BE WONDERING." WELL, WONDER NO MORE AS I WILL QUIETLY, CONCISELY AND SIMPLY EXPLAIN REALITY AS IT RELATES TO BASIC AND THE COMMODORE 64.

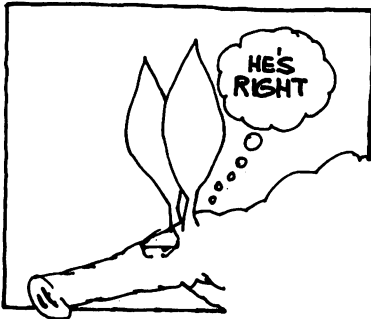
THE WONDER IS THAT HE CAN EXPLAIN REALITY AT ALL.



Introduction

So, You Want to Talk to Your Commodore 64?

Holy whiz bang! If you want to talk to your computer, this is it! The most fabulous book on Commodore 64 BASIC ever written! But, a word of caution: Take care as you flip through these pages. The characters residing herein are absolutely wacko. Don't catch their wackiness! Friends, neighbors, relatives, aardvarks, and other terrestrial beings might not understand.



You've been warned, and you're still reading? Well, brave soul, since you've taken the plunge, you might as well find out what's in store for you.

This Book Was Expressly Written (It took two days.)

First, this book was expressly written for *all* Commodore 64 Computer owners and their camels.



DR. WACKO PRESENTS COMMODORE BASIC

- A fancy typewriter (or word processor)
- A file cabinet (or database)
- An accountant (your very own number cruncher)
- A game machine (fun)
- A machine that teaches you about itself (conceited)
- A spelling helper (my favorite)
- An artist's palette (messy)
- A paper weight (functional!)

You'll learn how to change your computer into all sorts of weird things, plus lots of other miraculous stuff as you meander through this book.

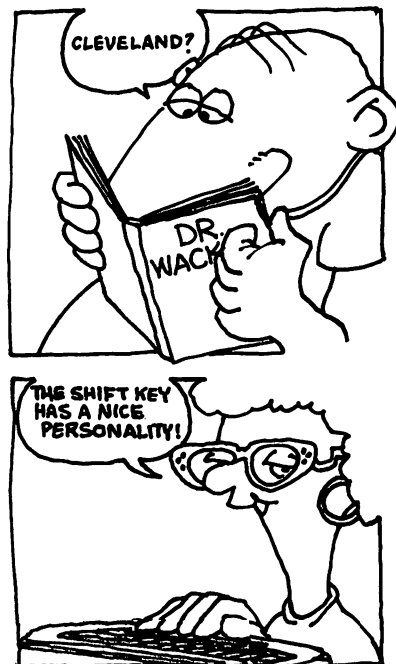
Tripping Through This Book (Oops!)

This book is so well organized that there is absolutely, positively *no way* you can get lost during your journey. (Dr. Livingston, I presume?)

Limber Up Your Fingers: Your trip begins as you exercise your fingers and get acquainted with the keyboard. Once you've savored Commodore 64's many keyboard delights, you march bravely on to the rigors of Commodore 64 Boot Camp.

Basic BASIC Training: Here, I'll introduce you to some BASIC fundamentals and reveal some of my more subtle BASIC tricks. You'll be astounded by many new and exciting things, like PRINT statements, variables, and line numbers. But have no fear, Dr. Wacko will be right behind you all the way.

On, To the Great White Expanse: When you leave Basic BASIC Training (either by graduating or going AWOL), you'll slip into the central section of this book, otherwise known as the Great White Expanse. Here, as you wander through the desert, bold examples of BASIC usage pop



Foreword

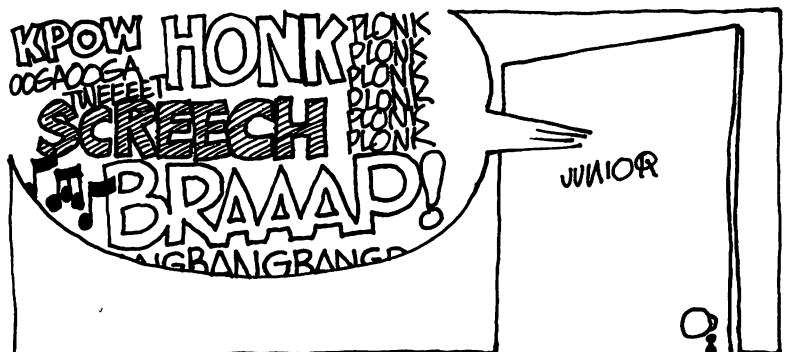
off each page like lush oases to dazzle your eyes. (Water! Water!)

Build Your BASIC Vocabulary: Having trouble expressing yourself? In this section the Wackos and I help you build your BASIC vocabulary in a logical, step-by-step way. Short, fun-filled, and utterly ridiculous programming examples show you what each BASIC command does, how to use it, and how to combine it with other BASIC words so you can really start conversing with your computer.

Obnoxious Colors, Weird Shapes, and Bright Lights: After leaving the stark white desert, you enter the vibrant and colorful world of Commodore 64 graphics. Here you'll see and do things with your knowledge of the BASIC language that you never dreamed possible. (Well, almost never.)

You'll learn my favorite, and until now, carefully guarded graphics tricks. After you've wandered through only a few pages of this section, you'll astound your friends, neighbors, and yourself as strange shapes, obnoxious colors (pugnacious purple), and bright lights dance on your screen, all under your control.

ZOUNDS!: Piercing sounds tingle your eardrums, eerie flute-like tones float melodiously from your computer room, and the neighborhood cats howl as you breeze through this section. So close the door, put cotton in your ears, and get set for some real audio excitement as you delve in to the wild world of Commodore 64 sound!



DR. WACKO PRESENTS COMMODORE BASIC

Sounds good, doesn't it? But, that's not all. There's more!

Weaving The Perfect Flying Carpet-The Art of Programming: Here's where it all comes together. Now you'll be prepared to fully utilize your BASIC knowledge, talent, and creativity to design some of the weirdest, most useful, and most exciting programs in the universe. Never before in history (well, almost never) has there been a book that teaches you *the fine art of programming*. It's all here in this one-of-a-kind section.

You'll start with an idea or need, and develop programs using my patented "Wacko Modular Approach." I'm with you each step of the way, right down to inflicting your finished product on your unsuspecting friends.

You will learn much, much more than BASIC programming. You'll learn to "see" every detail of *any* problem clearly; you'll discover and use the *creative* (Wacko) side of your brain, and you'll learn that programming is just like making anchovy burritos. (Honest!)

Building-Blocks and Lots of Schlocks: I wrap up this wondrous book with creative building-block projects that show you how to build your own Mini Word Processor, Secret Text Coder/Decoder, Infamous Model Of The Universe and, *much, much more!*

Are you ready to go? OK, just hop aboard your camel and we'll start our adventure!

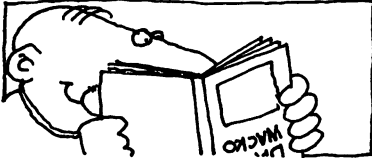
What's this camel stuff, anyhow? Good question, and it will all become crystal clear (as mud) after you meet the weird gang that will accost you during the rest of this great saga.

OK gang, let's get this caravan rolling. Do your stuff!



Foreword

EZ Book Instructions



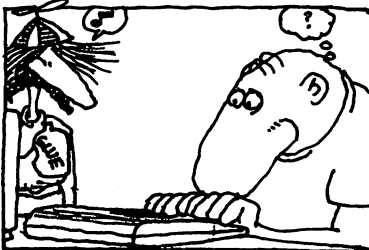
1. Open this mystical book.



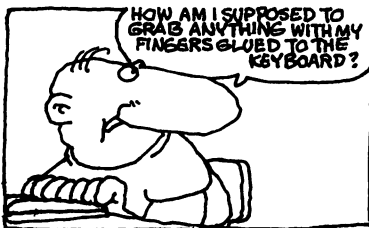
2. Place next to Commodore 64 computer.



3. Turn on your fingers.



4. Place your fingers on the keyboard.



5. Grab the closest magic carpet, put on your turban, fez, burnoose, or beanie, and enjoy your voyage through Dr. C. Wacko's exciting world of Commodore 64 BASIC programming!



DR. WACKO PRESENTS COMMODORE BASIC

DR. WACKO

EVICTED FROM IOWA IN 1936, INVENTED WIND-UP COMPUTER IN 1947. SWITCHED TO BASIC IN AN ATTEMPT TO TALK TO HIS SON JUNIOR. DISCOVERED BASIC WAS USEFUL IN PLACES LIKE NEPAL, MADAGASCAR AND SILICON VALLEY.



CAPTAIN ACTION

RESIDENT FIXIT & SWASHBUCKLING HERO OF GAME FREAKS, HE LEFT REALITY YEARS AGO TO WORK FOR WACKO. NEVER RETURNED, EITHER. SOMETIMES KNOWN TO TEASE JUNIOR.



MRS. PETUNIA WACKO

WACKO'S CHARMING AND ATTRACTIVE WIFE. AN ADVANCED MAINFRAME COMPUTER PROGRAMMER AND DR. WACKO'S MOST VOCAL CRITIC. SHE IS ALSO THE ONLY ONE IN THE GROUP WITH ANY PRACTICAL KNOWLEDGE.



SNIDLEY SEERSUCKER

LOVES TO FIND MISTAKES IN DR. WACKO'S PROGRAMS. HE'S WACKO'S 2ND. COUSIN TWICE REMOVED (TO THE COUNTY JAIL) AND OCCASIONALLY HELPS WACKO WITH COMPUTER GRAPHICS.



JUNIOR

LOVINGLY CALLED CEMENTHEAD BY HIS MANY FRIENDS, HE'S BEING GROOMED TO TAKE OVER THE BUSINESS WHEN HIS FOLKS RETIRE TO THEIR CABIN IN CLEVELAND. HE COLLECTS TEST PATTERNS.

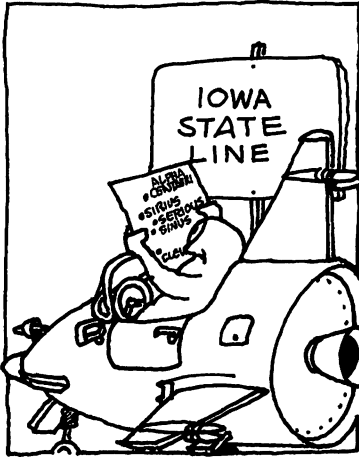


MS. PEEKY

FAMOUS FOR PEEKING AT THE LAST PAGE FIRST, SHE'S OUT HERE TO EXPERIENCE ADVENTURE, EXCITEMENT AND (HOPEFULLY) TRUE ROMANCE.

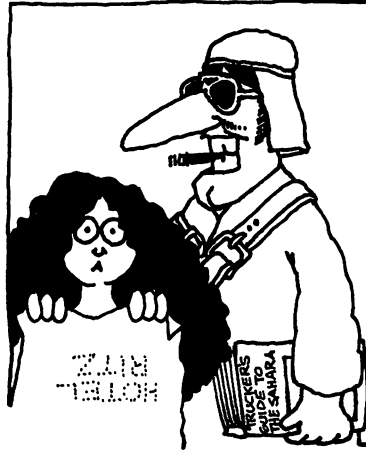


Foreword



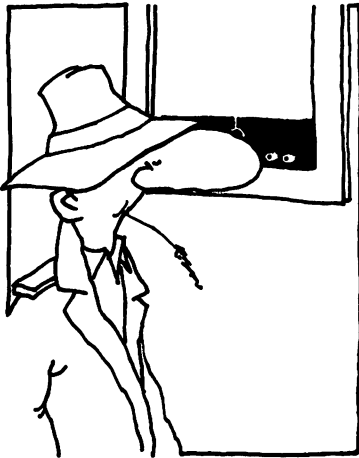
GROVER

AN ILLEGAL ALIEN WHO GOT LOST ON THE WAY TO ALPHA CENTAURI BECAUSE OF A BENT WARP DRIVE ON HIS '59 CADDY SPACE SHIP. IT CANT FLY OVER 200 FEET BECAUSE CAPTAIN ACTION FIXED IT.



BERNICE BERNOOSE & LAWRENCE OF NEWARK

KICKED OFF OF THE CLEVELAND BALLET, BERNICE HAS BEEN KICKING AROUND THE SAHARA HOT SPOTS PURSUED BY LAWRENCE, WHO'S LOOKING FOR SILICON CHIPS IN THE SAND.



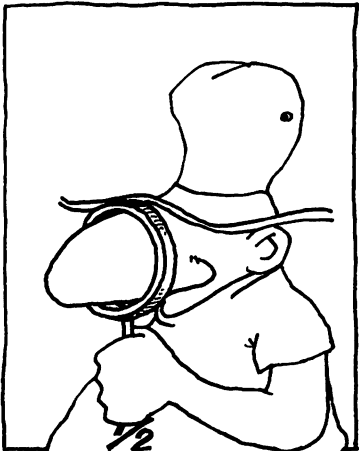
SNEAKY PEEK

THE GROUP'S EXPERT AT WINDOW SHOPPING FOR PEEK LOCATIONS. A FRIEND OF LAWRENCE OF NEWARK, HE'S SEEN THE CALVIN COOLIDGE STORY 16 TIMES.



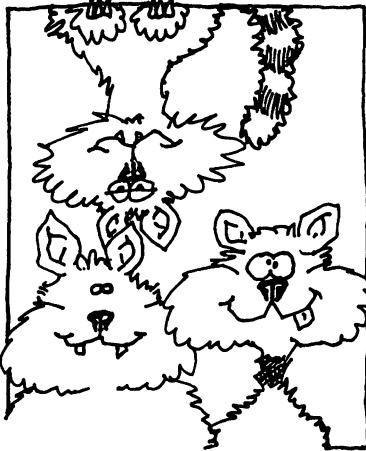
CLYDE

CHIEF TRANSPORT FOR EVERYBODY BUT JUNIOR, HE OCCASIONALLY SPITS OUT A FEW NUMBERS FOR WACKO



SLOW POKE

AFTER 13 YEARS OF CORRESPONDENCE SCHOOL, HE'S THE POKE EXPERT HERE...MAINLY BECAUSE HE'S USUALLY POKING AROUND SOMEBODY ELSE'S BUSINESS.



THE WACKO CATS

KEYS, PADDLES & JOYSTICK LOVE TO WALK ON THE COMPUTER AND MUNCH CONTROLS WHEN NO ONE IS LOOKING. TOTALLY UNHOUSEBROKEN.

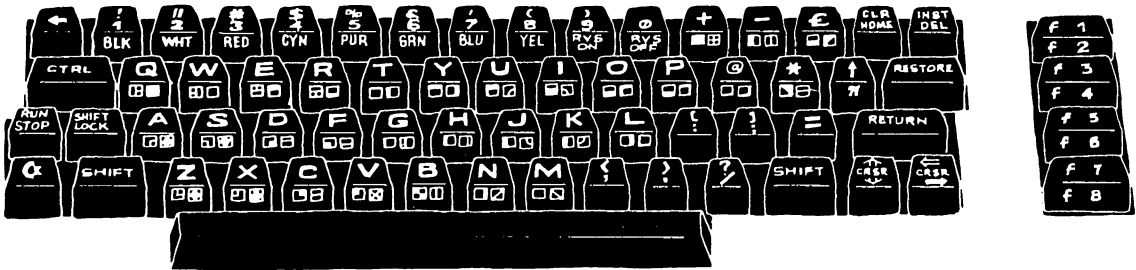
DR. WACKO PRESENTS COMMODORE BASIC

1

Limber Up Your Fingertips



Here it is, the keyboard you've all been waiting for. You should be sitting in front of a replica of this picture. If you aren't, you took the wrong flight!



Get To Know Your Keyboard

Your keyboard is your best friend. It provides a real handy way of communicating with your computer. You talk to it by banging on the keys. Simple, isn't it?

Give It a Try

All set? Ready to limber up your fingers? Ok, let's go for it.



Limber Up Your Fingers

To get the ball rolling, type in some words and symbols, press **RETURN** a few times, and really fill that screen of yours with nonsense. Don't be timid, the worst that can happen is that you'll dent your fingertips. What have you got to lose?

Ooops! A SYNTAX ERROR

Every time you pressed **RETURN** I'll bet the message, ?SYNTAX ERROR, appeared on your screen followed by the word READY. Your computer is just telling you that even though it doesn't understand a word you're saying, it's still READY to accept more nonsense. But have heart. By the time you finish this great book you'll be speaking to your Commodore 64 like a native.

Clean Up That Mess!

Boy, are you sloppy! Look at the mess you left on the screen. It looks a little like Junior's bedroom. Don't panic! It's real easy to clean up. Hold down **SHIFT**, press **CLR/HOME**, and your screen is as clean as a whistle. (How clean is a whistle, anyhow?)

Some Cursory Cursor Movement

Now that everything is straight-arrow and shipshape it's time for some cursory cursor movement. Moving that blinking white square (known as a "cursor" by real computer wackos), is simple. The two cursor control keys are located at the bottom left corner of your keyboard. Here's what they look like.



NOTE: WE
SHOW THEM
LIKE THIS...



SO THEY
FIT THE
TEXT.

To move the cursor *down*, press **↑CRSR↓**.

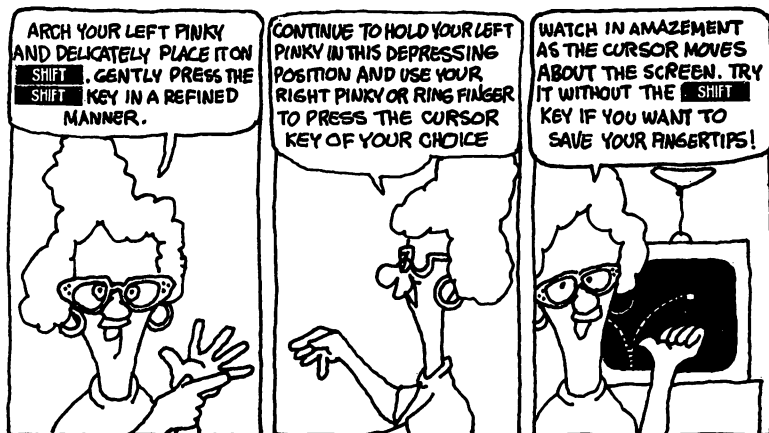
To move the cursor *up*, hold down **SHIFT** and press **↑CRSR↓**.

To move the cursor to the *right*, press **←CRSR→**.

DR. WACKO PRESENTS COMMODORE BASIC

To move the cursor to the *left*, hold down **SHIFT** , and press **←CRSR→** .

I know you understand how this cursory cursor movement works, but moving the cursor the Wacko way requires good manners and a sense of proper etiquette. Here's Ms. Peeky to demonstrate.



Not only does this method work, but if anyone sneaks into your computer room, you'll look like a real pro programmer (or a real weirdo)!

Bore yourself by moving the cursor all around the screen. Stop before you start crying in frustration and locate the **↵** key on the left side of your keyboard.

Did You find it? OK, hold down **SHIFT** and press the **↵** key *once*, then type "the first thing that comes to your mind."

A Case of Uppercase Shift

Your Commodore 64 must be getting old — it's acting like an old-fashioned typewriter. All those words you typed appear on your screen in lowercase!

Limber Up Your Fingers



Just to prove my point, hold down one of the **SHIFT** and press any other key. Voila! An uppercase letter or symbol! See, I told you. It's acting just like a typewriter.

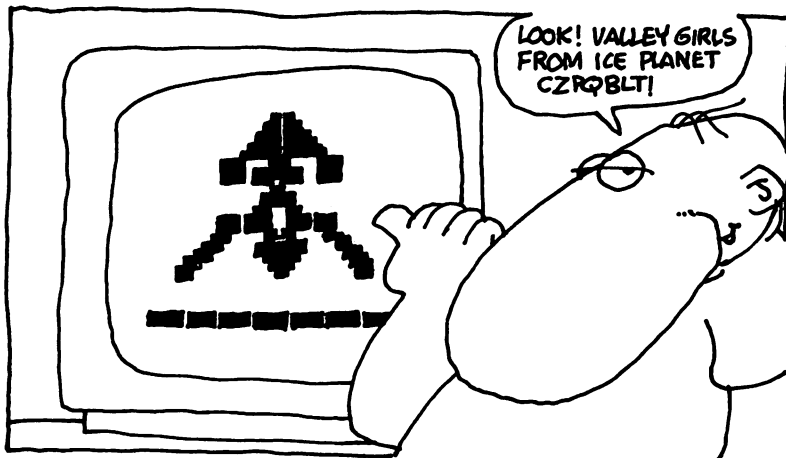
Lock Up Your Keyboard!

Before we move on, lock up your keyboard, in uppercase that is, before it starts believing that it's a typewriter. Just press the **SHIFT/LOCK** key, or better yet, hold down the **SHIFT** key and press the **⇧** key again. When you press both of these keys the *entire* screen switches between upper and lowercase characters. Geeez!

DR. WACKO PRESENTS COMMODORE BASIC

Weird Shapes and Odd Graphics

Now that you've set things straight, scrunch down in your chair so your eyes are level with the sides of the keys. Pretty uncomfortable, isn't it? But it's well worth the effort. I'm going to show you how to make all those weird shapes and odd graphics characters appear on your screen with a little help from the **SHIFT** and **⇧** keys!



There are two symbols printed on the sides of most of the keys. Just hold down **SHIFT**, press a key of your choice and the symbol to the *right* is printed on your screen.

Hold down the **⇧**, press a key of your choice, and the symbol on the left side of the key appears on your screen. Amazing!

You can use these shapes to create your own strange pictures! For example, hold down **SHIFT** and press **S**, **A**, **Z** and then the **X** key. Now you're ready for a game of Gin Rummy, or Junior's favorite, Fifty-two Pickup!

Limber Up Your Fingers

RETURN To The Keyboard...Please!

I know that Junior's having fun playing cards, even if he isn't playing with a full deck. But before you both get totally engrossed, please return to the task at hand by pressing **RETURN** !

Now's a good time to mention that the **RETURN** key is one of the most important keys on the keyboard. Press it now, and the cursor moves down one space and whizzes over to the left margin, you see that embarrassing message, ?SYNTAX ERROR, and the word READY. It's *almost* like pressing the carriage return on a typewriter. But when you are programming in Commodore 64 BASIC, pressing **RETURN** wakes up your computer and lets it know that you are telling it something.

As soon as you finish your tour of the keyboard and move on to Basic BASIC Training, you'll learn a lot more about the wondrous **RETURN** key and its many mystical powers. Now, back to the keyboard.

Sixteen Magnificent Colors

If you were paying attention when I asked you to look at the sides of the keys, you might have noticed that abbreviations for different colors are printed on the sides of the **1** , **2** , **3** , **4** , **5** , **6** , **7** , and **8** keys.

These important keys are used with **CTRL** and **⇧** to make your text come alive in as many as sixteen different colors. Or you may want to do some colorful drawing with Commodore's fancy character keys.

Getting colorful is a snap, and here's a handy chart that explains all.

DR. WACKO PRESENTS COMMODORE BASIC

Colors Using the **CTRL** Key

| | | | | |
|-------------|---|----------|---|---------------------|
| CTRL | + | 1 | = | Black |
| CTRL | + | 2 | = | White |
| CTRL | + | 3 | = | Red |
| CTRL | + | 4 | = | Cyan (Bright Green) |
| CTRL | + | 5 | = | Purple |
| CTRL | + | 6 | = | Green |
| CTRL | + | 7 | = | Blue |
| CTRL | + | 8 | = | Yellow |

Colors Using the **C=** Key

| | | | | |
|-----------|---|----------|---|----------------------|
| C= | + | 1 | = | Orange |
| C= | + | 2 | = | Brown |
| C= | + | 3 | = | Light Red |
| C= | + | 4 | = | Gray 1 (Dark Gray) |
| C= | + | 5 | = | Gray 2 (Medium Gray) |
| C= | + | 6 | = | Light Green |
| C= | + | 7 | = | Light Blue |
| C= | + | 8 | = | Gray 3 (Bright Gray) |

Colorful Playing Cards

Remember Junior's infamous deck? Let's brighten it up a bit by adding some colors. Follow along with Wacko and you're guaranteed to have the most colorful deck of cards in the desert.

Limber Up Your Fingers

Here we go!

A red spade: Press **CTRL** plus the **3** key, then press **SHIFT** and the **A** key.

A purple heart: Press **CTRL** plus the **5** key, then press **SHIFT** and the **S** key.

A light green diamond: Press the **⇧** key plus the **6** key, then press **SHIFT** and the **Z** key.

A yellow club: Press **CTRL** plus the **8** key, then press **SHIFT** and the **X** key.

Mind-boggling, isn't it?

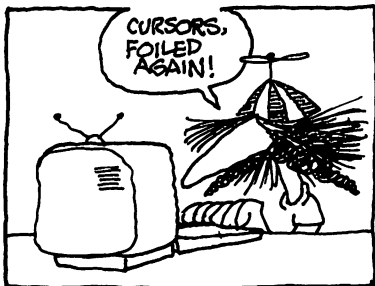


Your text can appear in any color you choose. I prefer white, but the choice is up to you. (Ms. Peeky just adores pugnacious purple.)

When you enter Basic BASIC training, and then move on to Commodore color graphics, you'll learn how to imbed colors into your programs to really impress your fellow desert wanderers...wow!

Editing...Looking Good

I never make any typing or spelling mizztakes when I'm writing letters or entering programs. But if you ever do, the Commodore 64 lets you make instant corrections on the screen, using its world-renowned and easy to use screen-editing functions.



Send Your Cursor Home

You already know how to clear your screen. Remember the old **SHIFT CLR/HOME** trick? If you want to send your cursor home (the upper left corner of your screen) *without* clearing the screen, simply press **CLR/HOME** key. Give it a try. See, no problem!

DR. WACKO PRESENTS COMMODORE BASIC

Now that your knowledge of cursory cursor movement is immense, you are well prepared to start fixing up those pesky screen-entry mistakes.

Ready to have some fun? OK, let's play a game of "Simple Wacko Says," as you learn all about screen editing.

First clear your screen by pressing **SHIFT** and **CLR/HOME** and we'll get started.

Simple Wacko says, "Type the following sentence *exactly* as it appears."

I here by volunteer for BASIC Trainng.

Notice any mistakes? You do. Well, let's set things straight.

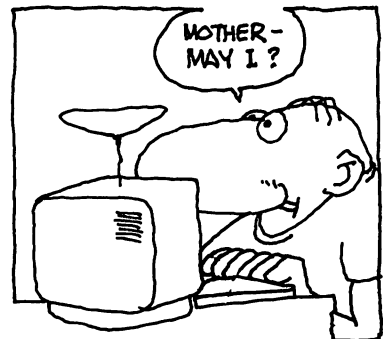
Insert A Weird Character: **SHIFT** + **INST/DEL**

Simple Wacko says, "Place your cursor over the second *n* in the word 'trainng'." There yet? OK, Simple Wacko says, "Hold down **SHIFT** and press **INST/DEL** ." You've just created a space between the two *n*'s. To correct my horatious* spelling, just type the lowercase letter *i* and you're all set.

Did you do it? Gotcha! Simple Wacko didn't say, "Type the lowercase letter."

You don't care? You don't want to play any more? OK, be that way. Well, even if you are fed up with that silly game, you've just done some fancy insertion, and your sentence is really starting to shape up. Here's how it should look now:

I here by volunteer for BASIC Training.



*Horatious: horrible and atrocious mixed together. Wacko/Webster Space Duck Dictionary.

Limber Up Your Fingers

Delete That Character: **INST/DEL**

Oops! There's a space between the words "here" and "by" that shouldn't be there. It should read "hereby."

To set things right, place the cursor over the small letter "b", and press **INST/DEL** once.

You did it! Your sentence is perfectimundo! Here's how it looks now:

I hereby volunteer for BASIC Training.

You do? Weren't you told *never* to volunteer! OK, in a minute you can flip the page and join the rest of the troops.

Isn't Editing Easy?

All editing on your Commodore 64 is done with these four keys:

CLR/HOME **INST/DEL** **←CRSR→** **↑CRSR↓**

If you make any mistakes while entering a program, or want to add more information to a program line, you simply march your cursor up to your mistake, use **SHIFT** + **INST/DEL** or **INST/DEL** to correct it, press **RETURN**, and everything is hunky-dory. Have no fear. You'll get plenty of practice as you march through this book.

Very Special Computer Keys

Stop! Before you go marching off to Basic BASIC Training, let's take a look at some very special computer keys.

DR. WACKO PRESENTS COMMODORE BASIC

STOP That Camel. I Want to Get Off!

You'll use **RUN/STOP** to interrupt your computer while it's thinking about your BASIC programming.

RESTORE It. Set Things Straight

Pressing **RUN/STOP** and **RESTORE** clears your screen and sets it back to its original two-tone blue display, complete with the word READY and a blinking cursor.

Follow along with Wacko, and I'll show you how to mess things up, then set them straight again.

First, press **CTRL** and the **3** key to make everything you type display red. Now type you're favorite limerick or laundry list.

When you get tired of staring at a red laundry list, simply press **RUN/STOP** and **RESTORE** and, voila, your screen is restored to its original display!

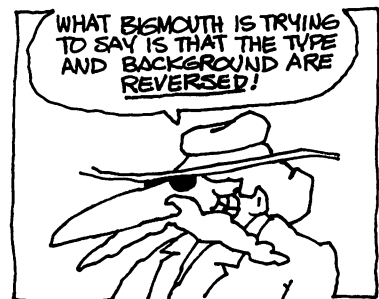
Once you begin entering programs, you'll discover that you can use this method to RESTORE your screen *without* erasing your program. But more on this later.

The Perverse Inverse Character Keys

Now, last but not least, the infamous inverse character keys: **RVS ON** and **RVS OFF**. These two keys are waiting for you right below the **9** and **0** keys at the top of your keyboard.

Did you find them? If you did, you're in for a big surprise. What are you waiting for? Press **CTRL** and **RVS ON**!

Now, type something. All the characters are printed on your screen in inverse video!



Limber Up Your Fingers

When you get tired of staring at perverse inverse characters, press **CTRL** and **RVS Off** to return to normal print.

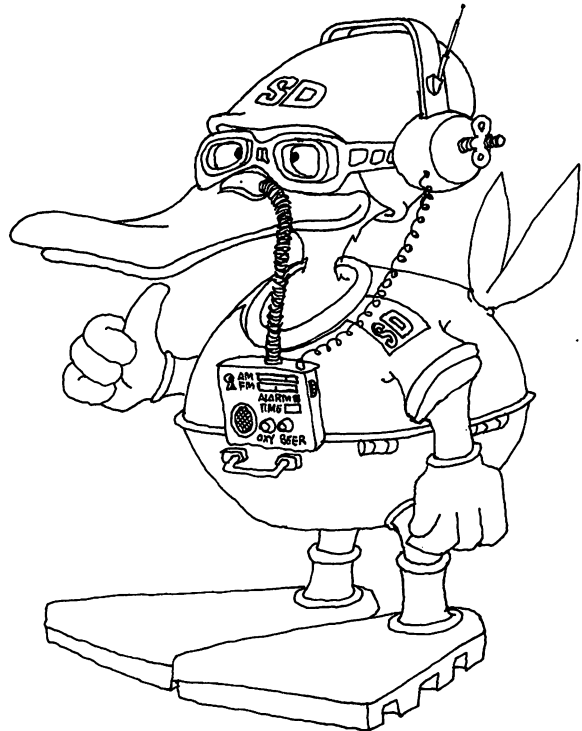
Those **f 1** Through **f 8** Keys

Do you see the row of keys on the right side of the keyboard numbered f1 through f8? Press them!

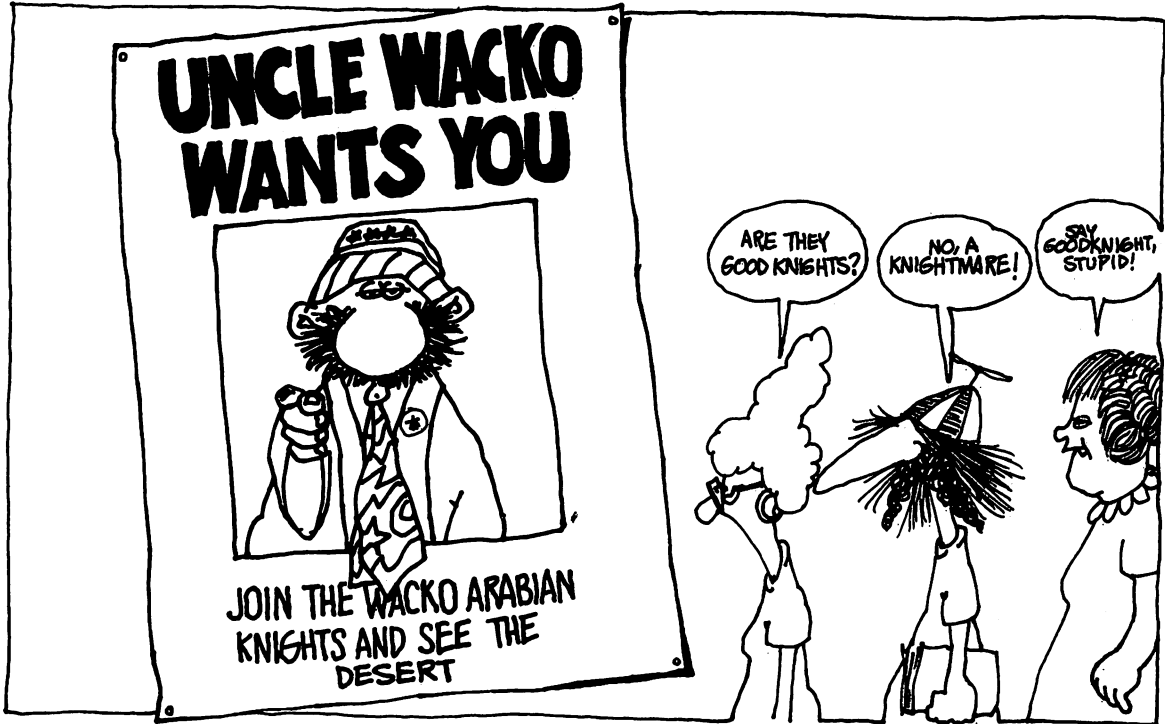
I'll bet that nothing happened when you banged on those *special function* keys. That's because these keys have been set aside for you! That's right, later you'll learn how to program and use these four miraculous keys to make your programs spiffier, and easier to use.

You're Limbered Up!

You did it! Your fingers are limbered up and your adventure through BASIC programming is about to begin! So buckle your seat belts, hold on tight, and get set to march into Basic BASIC Training.



2
Basic BASIC Training



BASIC training was a snap for me. I breezed through the short course in a few days, and just look where it got me!

If you've already graduated from boot camp, you may want to breeze through this chapter, pick up the highlights, and move bravely on to the Great White Expanse.

But just so you don't skip ahead of the game, here's what I'll be covering during Basic BASIC Training:

- The Immediate mode
- PRINT
- Lower, higher, and higher-than-lower arithmetic
- Line numbers
- Statements
- Variables

Basic BASIC Training



Sign up for BASIC training, and you'll be in good company. Look at that enthusiastic group at the top of the page. Don't they inspire and instill a sense of confidence? If they can make it, so can you! As you march along with these bright-eyed recruits, you'll learn to use many of the tools of the BASIC programming trade. And once the basics are out of the way, you'll be prepared to reach into your wacky self and produce some wacked out programs.

After graduation, you'll have a working knowledge of the most useful (and useless) programming tools, and as you progress through BASIC training I'll help you use these basic tools, plus your creative wackiness, to design *any* program (no matter how wacko) to suit almost any job, task, or whim.

So, now that you and your Commodore 64 are present and accounted-for, let's dive right in. It's time to start having some fun with that computer squatting in front of you.

Attenhut! Programming is simple. Just follow my incomprehensible instructions, use your imagination

DR. WACKO PRESENTS COMMODORE BASIC

and brain power, and you're on your way! Here, I'll show you. But remember to put the plug in!

First press **RUN/STOP** and **RESTORE** to set up your screen. Now take a look at this clever line of programming.

```
PRINT "[CTRL + 2]WHY [CTRL + 3]AM [CTRL + 4]  
[CTRL + 5]DOING [CTRL + 6]THIS[CTRL + 8]?"
```

Pay special attention to the words within quotation marks inside brackets, like **[CTRL + 2]**. This is an instruction, (known as a "convention" to real wackos) and is not to be typed in literally. It means "press **CTRL** plus the **2** key." When you do this, the word "WHY" in this program line displays in bright white, after you press **RETURN**.

I've listed all the *conventions* in Appendix A. So, take a side-trip back there now to familiarize yourself with all the programming conventions before moving on.

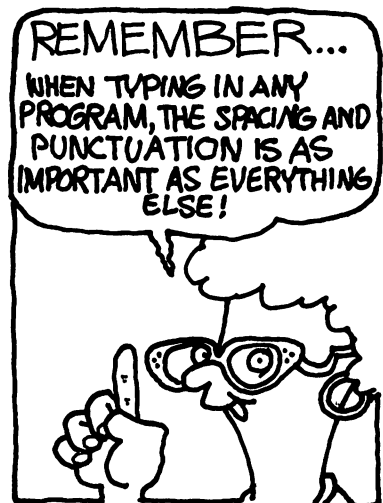
Did you enjoy the convention?

Great! Now type the short program, and the color-control characters *exactly as written*, then press **RETURN**

You've just seen one of your Commodore 64's amazing graphics tricks, entered your first program, learned about conventions, *and* gotten some instant and weird feedback. You made your computer do something wacko instead of just sitting there like a sleek piece of furniture! But that's not all. You entered a program in what is called the *immediate* mode.

The Immediate Mode: Life in The Fast Lane

The *immediate* mode of program entry is just like living life in the fast lane. After you type the words and press **RETURN** to "tell" your computer what to do, you in-



Basic BASIC Training

stantly see the results of your programming. Learning to give orders to your Commodore 64 is what Basic BASIC Training is all about. And when you issue orders in the *immediate* mode, your Commodore 64 carries them out immediately. Immediately after you press **RETURN** , that is.

When you get sick of staring at a questioning computer screen, you can clean the slate by pressing **RUN/STOP** and **RESTORE** .

RESTORE

After you press **RUN/STOP** and **RESTORE** (don't worry, you won't scramble your Commodore 64's brain) the screen turns blue and the word "READY" appears in the upper left-hand corner. Your computer is talking back to you. It's saying, "I'm READY, Sarge, to accept more of your silly commands."



Don't ignore this insubordination! Now's a good time to get really angry. Yell at the computer. Or, if you're like me, yell at Junior. After you've got it out of your system, you're READY to start pushing *it* around!

PRINT (?)

In Commodore 64 BASIC, your computer responds to a number of simple, one-word commands. One way it does this is by printing words on its screen. To make it say what you want it to, you use the PRINT command. If you want to get sneaky, just use a question mark (?) instead of the word "PRINT." Your computer will understand. Trust me!

Type in the word "PRINT" and press **RETURN** .

Uh, oh! There's that word "READY" again! But don't fly off the handle. If you look at your screen closely, you'll

DR. WACKO PRESENTS COMMODORE BASIC

see that the computer *did* print something. It printed a blank line!

Not impressed? Neither am I. So, let's get some real utility out of the PRINT command. Let's make it print something really important. Ready? OK, type these words (including the quotation marks) and press

RETURN :

PRINT "Stop banging on my keyboard!!"

Geez, it seems you can never make it happy. Now it's talking back, and being insubordinate! But, I'm sure you get the idea.

*To make your computer print something on the screen just type the word PRINT, enclose your words of wisdom within quotation marks, and press **RETURN** .*

Anything you put between quotation marks after a PRINT command is printed on, or does something to the screen display. "Anything" includes numbers, symbols, all sorts of fancy colors, and strange happenings! Type this short message, and press **RETURN** to see the result.

**PRINT "[CLRHM]": PRINT "E = MC↑2,
SOMETIMES."**

Wow! First the screen cleared, then the short Einsteinian example was printed!

Pause before you move on, and take a few minutes to make up some weird PRINT statements. Wrap quotation marks around some graphic symbols, or, if you want to push ahead of the game, try adding color to your prose like I did in my first horrendous example. Fun, isn't it? Stick with Dr. Wacko because the best is yet to come! Later in your journey you'll discover more of your Commodore's powerful color and graphics features.

Basic BASIC Training

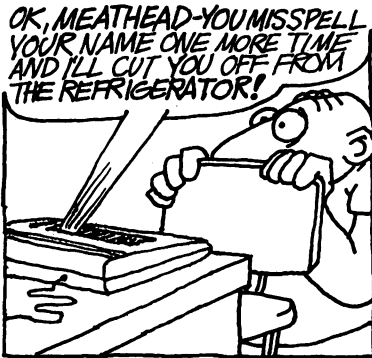
If You Have a Printer

If you have a printer and would like to see your words printed on paper, type your message like this:

When you see: “[RETURN]” just press **RETURN** .

OPEN 1,4:PRINT#1 “IT’S GREAT TO SEE MY WORDS IN PRINT!”:CLOSE 1 [RETURN]

OPEN 1,4 OPENS up communications with your printer, **PRINT#1** works just like a **PRINT** statement but prints your words to the printer, and **CLOSE 1** CLOSEs down communications when the printing is done. But don’t use this snazzy method of communications if you don’t have a printer or if your printer is turned off. If you do, your Commodore will get angry and respond with the message, “DEVICE NOT PRESENT ERROR.”



Your Computer Isn't Very Bright

There's one thing you should know before you start bossing your computer around. Your computer isn't very bright. It only understands commands that are entered perfectly, one-hundred-percent correctomundo. For example, if you entered PRIR when you meant to enter the command PRINT, your Commodore 64 would get confused, burp, and tell you it doesn't understand PRIR by displaying the words “?SYNTAX ERROR.” Don't get upset by this. Just remember that it is, after all, only a computer.



Lower Arithmetic

Don't worry, Ms. Peeky. Math has very little to do with BASIC programming. Actually, all you need to know is simple arithmetic, the kind you learned in grade school. In fact, your computer can help you. It's just a super-calculator that's really easy to use.

DR. WACKO PRESENTS COMMODORE BASIC

Addition

Try typing these simple equations:

```
PRINT 5 + 5 [RETURN]
```

```
PRINT 4 + 6 + 7 [RETURN]
```

See how easy it is? And you don't even have to use an equal sign! You also don't put quotation marks around the numbers. If you do, your computer will print them instead of performing the math. Wrap quotes around the numbers to see what I mean.

Subtraction

Check out these easy examples:

```
PRINT 10-5 [RETURN]
```

```
PRINT 20-10-5 [RETURN]
```

If you want to get real fancy, try combining addition and subtraction functions like this:

```
PRINT 20 + 5-10 [RETURN]
```

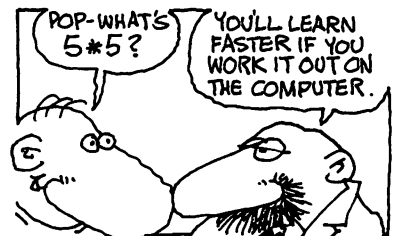
Higher-Than-Lower-Arithmetic

Now I'm going to move on to the two biggies, multiplication and division.

Multiplication

An asterisk (*) is used as the symbol for multiplication by your computer. (The "*" key is located next to the "@" key on the right side of your keyboard.)

```
PRINT 5*5 [RETURN]
```



Basic BASIC Training

Division



A slash (/) is used as the symbol for division by your computer. (The "/" key is on the same key as the "?" at the lower right-hand side of your keyboard.)

PRINT 25/5 [RETURN]

Lofty Concepts of Lower and Higher Arithmetic

Lofty Concept 1: A Set of Logical Rules. When you combine multiplication, division, addition, and subtraction in a single statement, things get a little hairy (even if you're as bald as I am). But they do follow a specific set of logical rules.

Here's an example. Type it in, press **RETURN**, and check out the answer. It should be 9.25.

PRINT 5-2 + 5*5/4

Don't run away, Ms. Peeky. It's no big deal! Here's Petunia, she'll explain it to you.



DR. WACKO PRESENTS COMMODORE BASIC

If you enter: **PRINT 5 + 7 + 8**

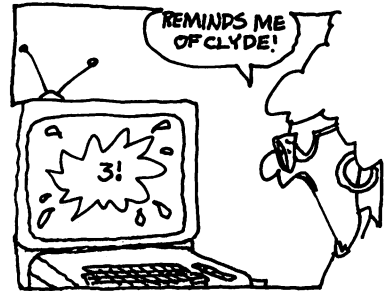
Your computer first adds $5 + 7 = 12$, then it adds 12 to 8 to arrive at the answer: 20

Here's another example. But, this time I've mixed in subtraction as well as addition.

If you enter: **PRINT -5 + 9 - 3 + 2**

Your computer first adds $-5 + 9 = 4$, then it adds $4 - 3 = 1$, and finally adds $1 + 2$ to arrive at the answer: 3

When it's all finished calculating, it spits the answer out and displays it on your screen.

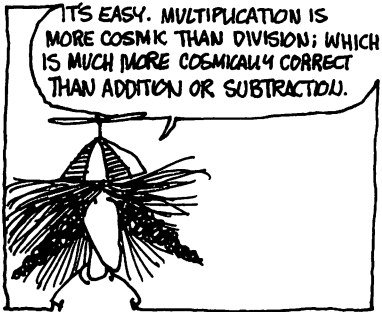


I'll summarize this simple operating order, then show you some more examples.

The Cosmic Order of Things

1. For addition and subtraction only, the computer calculates left to right across the screen.
2. For multiplication and division only, the computer performs the multiplication first, then the division, regardless of the order.
3. When combining addition, subtraction, multiplication, and division, the computer performs the

Basic BASIC Training



multiplication first, then the division, and, last but not least, the addition and subtraction from left to right across the screen.

Let's work through the arithmetic problem that scared Ms. Peeky, and you'll see how the computer follows these three operating rules to arrive at 9.25 as the answer.

Here's the problem again:

PRINT 5-2 + 5*5/4

Following the proper order, the computer first performs the multiplication:

$$5*5 = 25$$

Next, moving right along, the computer does the division:

$$25/4 = 6.25$$

Last, but again, not least, the computer does the addition and subtraction from left to right and arrives at its answer:

$$5-2 + 6.25 = 9.25$$

See, *no problem* at all! Your computer was simply following orders. And even if you switch the order of things around like this:

PRINT 5*5/4 + 5-2

The answer will still be — you guessed it — 9.25!

DR. WACKO PRESENTS COMMODORE BASIC

Lofty Concept 2: Operations in Parentheses are Done First. You can use the parentheses to change the order of things, and get your own way, so to speak.

Suppose, in the above example, you really want to mess things up. You want the division performed last, *after* the multiplication, addition, and subtraction. Well, it's simple. Just use parentheses, like this:

PRINT (5-2 + 5*5)/4

Here's how your computer will operate on this problem.

First it performs the multiplication inside the parentheses:

$$5 * 5 = 25$$

Next it does the addition and subtraction within the parentheses:

$$5 - 2 = 3$$

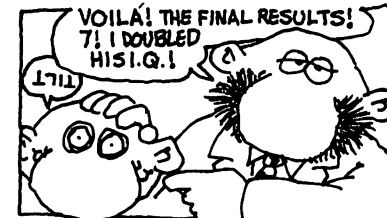
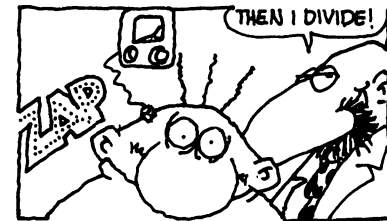
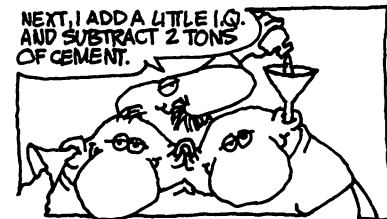
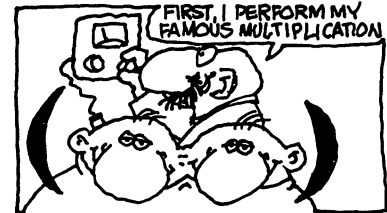
Then it adds the two results within the parentheses together:

$$3 + 25 = 28$$

And now that it's finished operating within the parentheses, it hops out and performs the division:

$$28 / 4 = 7$$

The answer is 7! The same numbers were used in both examples. But in the first case the answer was 9.25 and in the last example the answer was 7, and all because of the parentheses!



Basic BASIC Training

Here's another example that uses the same numbers and parentheses. But this time, amazingly enough, the answer is different than the first two examples.

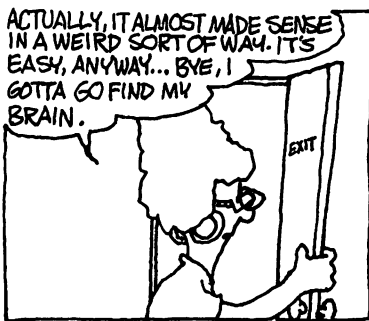
PRINT (5-2 + 5)*5/4

When you type this problem into your computer and press **RETURN**, the answer is 10. Step through this example yourself to see why.

You can breathe easier. We are finished with the arithmetic. Now you know enough to be an expert BASIC programmer. Ms. Peeky, you should be awarded a medal for your perseverance.

See, I told you it was easy! Here's a short summary list to help you remember these simple arithmetic concepts:

- Multiplication first
- Division second
- Addition and subtraction third; left to right
- Operations in parentheses are done first



Leaving the Fast Lane: Bye, Bye, Immediate Mode



DR. WACKO PRESENTS COMMODORE BASIC

Up to this point, you've been living in the fast lane, entering stuff into your computer and experiencing instant results. But living recklessly has its drawbacks. Here's the Einsteinian program you typed in before:

```
PRINT "E = MC↑2" [RETURN]
```

A *program* tells your computer to do something, and after you typed this example and pressed **RETURN** you saw the message written on your screen. But if you cleared your screen (by pressing the the **RUN/STOP** and **RESTORE** keys) and wanted to repeat the message, you had to type the entire line again!

The Infamous Programming Mode: Line Numbers and the Perfect 10 & 20

You'll be happy to hear that there is another way of entering a program: Use *line numbers* in the infamous *programming mode*.

To show you what I mean, here's another nifty example. But this time there is a subtle difference. There are two lines, and each line has a *line number* in front of it. The perfect 10 and the perfect 20! See them?

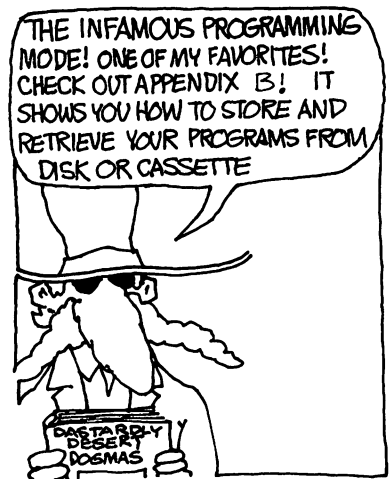
Brilliant Wacko

```
10 PRINT "[CLRHM] [CTRL + 2]"  
20 PRINT "WACKO ";;GOTO 20
```

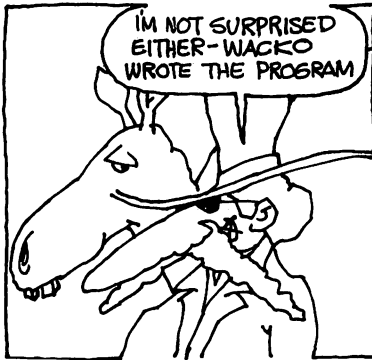
Turn down your computer's intensity, or put on your sunglasses, and get set for a real wacked-out experience.

Type this program, just as it's shown. Press **RETURN** at the end of each line, and don't forget the perfect 10 and 20.

Did you do it? What happened?



Basic BASIC Training



Nothing appeared to happen, and I'm not surprised.

But appearances can be deceiving, Snidely. Fooled you! Something very important *did* happen after you pressed **RETURN**. The Brilliant Wacko program was stored in your computer's memory!

RUN, RUN, RUN Around

Now that you know that I'm hiding somewhere inside your computer, it's time to find me, drag me out, and get me to do my thing, so to speak. To get a program to run around and act weird, all you have to do (you guessed it!) is type the RUN command and press **RETURN**. Simple, isn't it?



What are you waiting for! Go for it! Type RUN and press **RETURN**.

Now your Commodore 64 is doing its own thing and telling you that it's having fun by going completely wacko! There's nothing like having a weird computer to bang on.



STOP This Nonsense!

If your eyeballs are popping out of your head, or if you're about to fall asleep, you can stop me in mid-wack by pressing **RUN/STOP**.

After you press the **RUN/STOP** key (you won't stop your Commodore 64's heart) the message, "BREAK IN 10" appears. You've stopped your computer right in the middle of its "I'm going wacko" routine!

CONTINUE Where You Left Off!

If you feel that you've hurt your Commodore 64's feelings by stopping its brilliant display, there are two things you can do to make it happy again.

DR. WACKO PRESENTS COMMODORE BASIC

You can either type `CONT` and press `RETURN` to make your Wacko message `CONTInue` where it left off. Or (if you want to get snooty) you can type the word `"RUN"` and press `RETURN` to `reRUN` your program.

By adding line numbers to a program, you've told your computer to store it in its memory. Then, when you type the word `"RUN"` and press `RETURN`, it's off and `RUNning` — acting strange! The best part of using line numbers is that *you don't have to retype the program when you want to replay it!*

Take a Closer Look at That Program and Make an Important STATEMENT

Get out your magnifying glass and really examine the Brilliant Wacko program.

You'll see that line 20 is made up of two *Statements*. Each statement is separated by a colon (:).

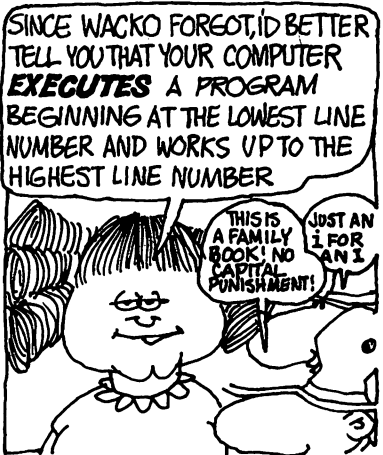
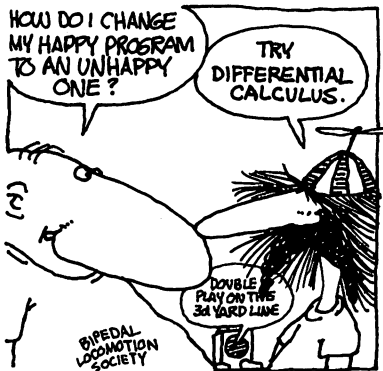
The first statement in line 20 is: `PRINT "WACKO";`. This tells your Commodore 64 to print my name on the screen.

The second statement is: `GOTO 20`. This tells your computer to go back to the beginning of line 20 and print my name again!

Here's how both lines of this program read in English: Line 10 says, "Hey, Ahmed, clear the screen then make all the printed characters appear in brilliant white. The first statement in line 20 tells the computer to print `"WACKO"` on the screen, and the second statement tells it to go back and print it again, and again, and...



Basic BASIC Training



Old Lines for New

Let's get obnoxious and break the Wacko program up by using three line numbers, one line number for each statement. But before you can create a new breezy program you'll have to get rid of the old one — either by replacing it line-by-line or totally annihilating it!

Junior's method of changing a line, *replacing an old line with a new one*, works just fine. So, to change a program line, all you've got to do is type its replacement and press **RETURN**. Your new line will replace the old one.

By the way, you can use any whole number as a line number. For example, you can assign line numbers 1,2,3, and 4 to your program. But expert programmers like me number programs by tens; 10, 20, 30, 40... This method leaves me plenty of room between each line to squeeze all my creative second thoughts into the program sometime later.

Now that you know all about line numbering, let's revise line 20 of the Brilliant Wacko program. Here's the original program again, in all its obnoxious glory:

```
10 PRINT "[CLRHM] [CTRL + 2]"
20 PRINT "WACKO ";GOTO 20
```

Now, just type this new line 20 and press **RETURN** :

```
20 PRINT "WACKO ";
```

Did you do it? If you did, that long line 20 with its two wild and crazy statements has just been replaced with a real shorty.

Now, let's assign a new line number to the second statement in the original line 20.

```
30 GOTO 20
```

DR. WACKO PRESENTS COMMODORE BASIC

Here's how your new, improved, and spiffier program looks when you're finished (Don't forget to press **RETURN** after you type each line of programming!):

```
10 PRINT "[CLRHM] [CTRL + 2]"
20 PRINT "WACKO ";
30 GOTO 20
```

This new Brilliant Wacko works *exactly* like the two-line program you started with; even though it now has three program lines! If you don't believe me, check it out by typing RUN and pressing **RETURN** .

Do you remember how to turn it off? You do? Well, before Captain Action gets violent, (and Ms. Peeky gets violets) *turn it off!!!*

CLEAR Up This Mess

By now your screen is probably full of program lines, STOP signs, and other nonsense. Pretty confusing, isn't it? Well, before your screen becomes a jumbled mess, let's clean it up! Just hold down **RUN/STOP** , and press **RESTORE** , and your screen clears instantly. Go ahead and try it.

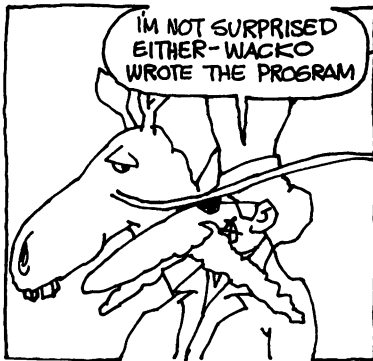
Type LIST

Don't panic! Your three-line program is just waiting patiently inside your Commodore 64's memory. To get it to appear on the screen, type LIST and press **RETURN** . Go ahead, type LIST and press **RETURN** already!



Wheew, there's your program again. As a matter of fact, any time you want to see your program *listed* on the screen, simply type LIST, and press **RETURN** .

Basic BASIC Training



Nothing appeared to happen, and I'm not surprised.

But appearances can be deceiving, Snidely. Fooled you! Something very important *did* happen after you pressed **RETURN**. The Brilliant Wacko program was stored in your computer's memory!

RUN, RUN, RUN Around

Now that you know that I'm hiding somewhere inside your computer, it's time to find me, drag me out, and get me to do my thing, so to speak. To get a program to run around and act weird, all you have to do (you guessed it!) is type the RUN command and press **RETURN**. Simple, isn't it?



What are you waiting for! Go for it! Type RUN and press **RETURN**.

Now your Commodore 64 is doing its own thing and telling you that it's having fun by going completely wacko! There's nothing like having a weird computer to bang on.



STOP This Nonsense!

If your eyeballs are popping out of your head, or if you're about to fall asleep, you can stop me in mid-wack by pressing **RUN/STOP**.

After you press the **RUN/STOP** key (you won't stop your Commodore 64's heart) the message, "BREAK IN 10" appears. You've stopped your computer right in the middle of its "I'm going wacko" routine!

CONTINUE Where You Left Off!

If you feel that you've hurt your Commodore 64's feelings by stopping its brilliant display, there are two things you can do to make it happy again.

DR. WACKO PRESENTS COMMODORE BASIC

Here's how your new, improved, and spiffier program looks when you're finished (Don't forget to press **RETURN** after you type each line of programming!):

```
10 PRINT "[CLRHM] [CTRL + 2]"
20 PRINT "WACKO ";
30 GOTO 20
```

This new Brilliant Wacko works *exactly* like the two-line program you started with; even though it now has three program lines! If you don't believe me, check it out by typing RUN and pressing **RETURN** .

Do you remember how to turn it off? You do? Well, before Captain Action gets violent, (and Ms. Pecky gets violets) *turn it off!!!*

CLEAR Up This Mess

By now your screen is probably full of program lines, STOP signs, and other nonsense. Pretty confusing, isn't it? Well, before your screen becomes a jumbled mess, let's clean it up! Just hold down **RUN/STOP** , and press **RESTORE** , and your screen clears instantly. Go ahead and try it.

Type LIST

Don't panic! Your three-line program is just waiting patiently inside your Commodore 64's memory. To get it to appear on the screen, type LIST and press **RETURN** . Go ahead, type LIST and press **RETURN** already!

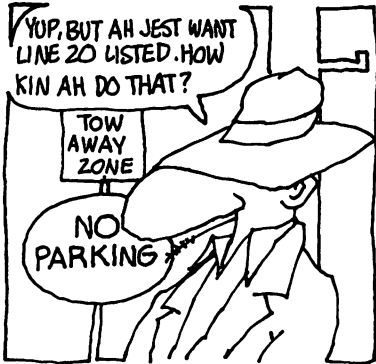


Wheew, there's your program again. As a matter of fact, any time you want to see your program *listed* on the screen, simply type LIST, and press **RETURN** .

Basic BASIC Training

If you want to list a program on your printer, type:

```
OPEN 1,4:CMD1:LIST:PRINT# 1:CLOSE 1  
[RETURN]
```



If you want to list one specific line of your program, (just line 20, for example), simply type the command LIST and the line number you want listed, then press **RETURN**. Like this:

```
LIST 20 [RETURN]
```

To list line 20 to your printer, type:

```
OPEN 1,4:CMD1:LIST20:PRINT# 1:CLOSE 1  
[RETURN]
```

That's all there is to it! "But," you may ask (I might not answer), "how do I list a specific number of program lines?" It's simple! First type the word LIST followed by the lowest line number you want listed, then type a dash (-), and finally the highest line number you want listed. A picture is worth a desert-full of words. So, here's how you list *only* lines 20 and 30 of the Brilliant Wacko program:

```
LIST 20-30 [RETURN]
```

Listing lines 20 and 30 to your printer is a snap!

```
OPEN 1,4:CMD1:LIST 20-30:PRINT# 1:CLOSE 1  
[RETURN]
```

CTRL = **Slow Motion!**

Later, you'll be working with longer listings. And, as the listing scrolls down your screen, you might want to slow it down to look for a line of code in closer detail, then stop the list to do a little editing or bail out of the

DR. WACKO PRESENTS COMMODORE BASIC

computer room. Here's how to make your program scroll down your screen in slow motion.

After you type LIST [RETURN], put your left ring finger on the **CTRL** key. Slow motion! It's as easy as that!

To stop the program as it scrolls down your screen, simply press **RUN/STOP**, then lean forward and stare glassy-eyed at your masterpiece. To send it on its way again, simply type LIST and press **RETURN**, and off you go!

Total Line Annihilation

I know that you're quick on the trigger. So, now that you've got your program back, let's get vicious and annihilate it, line by line! (Snidely would approve.) To totally annihilate a line of code, simply type its line number and press **RETURN**. Poof! it's gone!

If you've followed Snidely's dastardly instructions, line 10 of your three-line program has disappeared, erased completely from your Commodore 64's memory! You don't believe me? LIST the program again — I'll wait while you go through the motions.

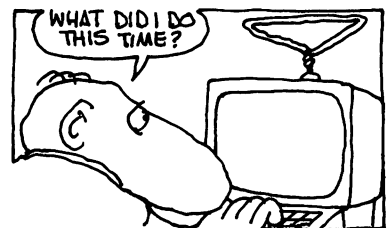
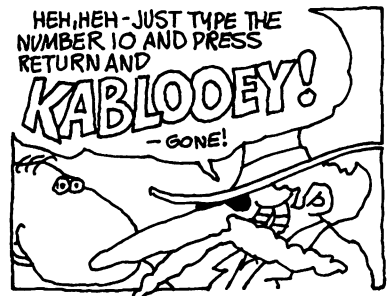
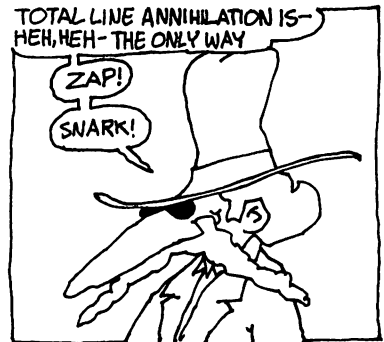
See, line 10 has been ZAPPED. It's vanished. Now, you've really done it!

When you really want to be devious, just type 20 and press **RETURN**, then 30. When you're through you will have completely annihilated me.

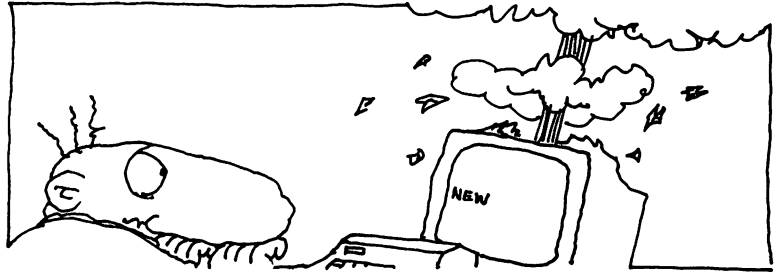
What's NEW?

If you think that's nasty, just watch while I reveal Snidely's secret weapon: The NEW command.

Typing the word "NEW" and pressing **RETURN** completely and irrevocably eradicates your entire program. Gadzooks!



Basic BASIC Training



CAUTION: The NEW command is extremely powerful. Use with extreme care. It not only destroys me, but completely erases your computer's brain! It's almost as shocking as turning off your computer!

You're Becoming a Real Smarty!



From this point on, I won't always remind you to press **RETURN** after entering each program line or immediate command. You know all about that now!

Now that you know about line numbers, the PRINT command, Higher and Lower Math, Brilliant Wackos, and lots of other stuff, it's time to make your computer go bzzzrk.

Warning: The Two-Line Limit



But first, I have to let you know one very important limit to your programs. *You can only enter a maximum of two screen-lines of information after a line number.* If

DR. WACKO PRESENTS COMMODORE BASIC

you exceed this limit your computer gets angry, and you'll see ?SYNTAX ERROR displayed on your screen.

Enter this program to see what happens after you press

RETURN :

```
10 PRINT "I LIKE TO RAMBLE ON AND ON,  
WHILE I PROGRAM. SOMETIMES THIS GETS  
ME INTO VERY SERIOUS TROUBLE...OOPS!"
```

The best way to overcome this limitation is to squeeze as much information as possible into the Two-Line limit. One way to do this is by leaving out spaces between commands, line numbers, and colons. Here's what I mean:

```
10PRINT"I LIKE TO RAMBLE ON AND ON"
```

Scrunching up a program like this makes it a little hard to read, but it is sometimes necessary when you've just got to get your information on one programming line. When you become an advanced wacko, (or a demented one like me) and move on to my Commodore Arcade Game book, you'll use scrunching to shorten those long listings

Now that you understand all about the Two-Line Limit, obliterate this program with the NEW command, and we'll go on to more sensible stuff!

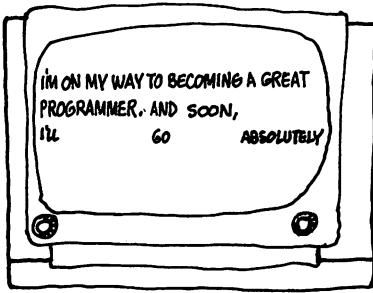
Colons: Commas, Semicolons; and The Remarkable REM

Here's a short program that I'd like you to type in and run:

Basic BASIC Training

Absolutely Wacko

```
10 PRINT "I'M ON MY WAY TO BECOMING A GREAT  
GREAT":PRINT "PROGRAMMER. AND  
SOON,"  
20 PRINT "I'LL", "GO", "ABSOLUTELY"  
30 REM:PRINT "WACKO";  
40 REM:GOTO 30
```



After you run this program, your screen will look like this:

Pretty weird, isn't it?

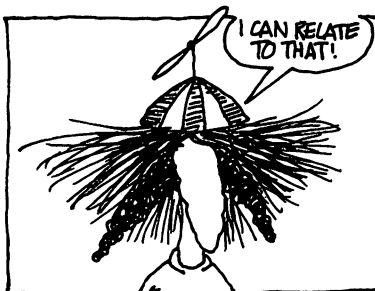
Two lines of text are printed on your screen, then (and here's the weird part) the words "I'LL," "GO," and "ABSOLUTELY" are each separated by 10 blank spaces.

What's going on here? Allow me to elucidate.

The COLON (:)

Two PRINT statements are squeezed onto line 10. Each statement is separated by a colon. This accounts for the *two* lines of text you saw on your screen. *Using a colon lets you put more than one instruction in a program line.* I used this nifty space-saving method earlier when I showed you my Brilliant Wacko program. But, beware of the Two-Line Limit!

The COMMA (,)



Careful placement of the *comma* lets you print your output in columns ten spaces apart. That's why the words "I'LL," "GO," and "ABSOLUTELY" are so spaced out.

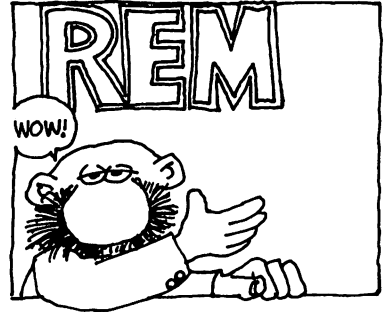
Press **SHIFT** and **CLR/HOME** to clean up the mess on your screen, LIST the program, and I'll show you what all this REM stuff is about.

DR. WACKO PRESENTS COMMODORE BASIC

REM

I use a REM statement at the beginning of a program instruction or a bunch of words when:

1. I don't want that line of code to be executed.
2. I want to leave a message to myself or other wackos about my program.
3. I want to make my program listing easy to read and understand.



I'm a pacifist and have used REM statements in lines 30 and 40 to prevent these two lines from being executed. If you're curious, and just have to see what happens when these REM statements are removed, I'll tell you how to go about it. But please give me a few seconds to leave the room before you run the program!

Here goes! Follow these instructions precisely, and you will cause mayhem and disorder.

1. Use the up, down, left and right arrow keys to march your cursor up the screen.
2. Halt when your cursor is positioned directly above the P of PRINT in line 30.
3. Press **INST/DEL** five times...until the word "REM" and the colon are completely obliterated.
4. Press **RETURN** .
5. Repeat this diabolical process again to remove the REM in line 40.
6. Run the program after I run out of the room! EEEYOW!

WACKO, WACKO, WACKO! That's what you'll see when you run the program after you've removed the REM statements. I can't stand it! Please, I beg you, press **RUN/STOP** and CLEAR the screen before I return to the room.

Whew! Thanks. I'm wacked-out enough as it is.

Basic BASIC Training

Semiwacko, or, The Semicolon (;) Did it!

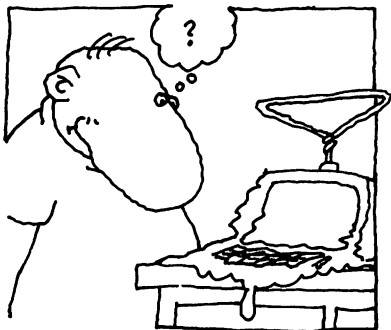
Do you see that semicolon at the very end of line 30? You do? Well, that's what caused the screen to fill up with WACKO's!

Remove that pesky semicolon. Position the cursor one space past the semicolon, then press **INST/DEL** once and press **RETURN**. Thanks. Now **RUN** the program again. See, the WACKO's aren't going across the screen anymore, just down the screen. Oh, well.

Here's one semipractical use of the semicolon. Press **RUN/STOP** to stop the program, and clean out your computer's brain with the **NEW** command. Now enter and **RUN** this program.

```
10 PRINT "2 CAMELS PLUS 1 MAGIC  
LAMP = ";  
20 PRINT "3 SHEKELS...SOMETIMES"
```

Since you are, by now, semiwacko, you'll want to use lots of semicolons to print stuff from different lines next to each other.



If you've been overtly alert and paying attention (and paying some bribes on the side) most of the mystery of that computer in front of you has probably evaporated.

Let's see, you've entered your first computer programs in the *immediate* mode, learned about the **PRINT** command, crawled through Lower and Higher, and Higher-than-lower Arithmetic, made a few statements, entered some programs, watched them acting silly, and banged on the keyboard a lot.

It's Time for a Party!

If you are still standing while sitting and reading this sentence (what?), you've got it made — or you've got

DR. WACKO PRESENTS COMMODORE BASIC

heartburn! You've reached the pinnacle of Basic BASIC Training. All that drill, cleaning sand out of your boots, listening to bird calls, and eating C-rations is over.



But before you get carried away (literally), I'm going to introduce you to two important programming concepts. Then you'll be ready to hop on your camel, charge off to the Great White Expanse, and change that magical computer of yours into all sorts of weird and wondrous things.

Important Concept #1: NUMERIC VARIABLES/

Variables are, you guessed it, things that vary or change.

Some examples of things that *do* vary are:

- My waist size after I eat too much falafel
- Petunia's hair styles
- Captain Action's beer consumption
- The price of a good camel
- Ms. Peeky's boyfriends
- Time



Basic BASIC Training

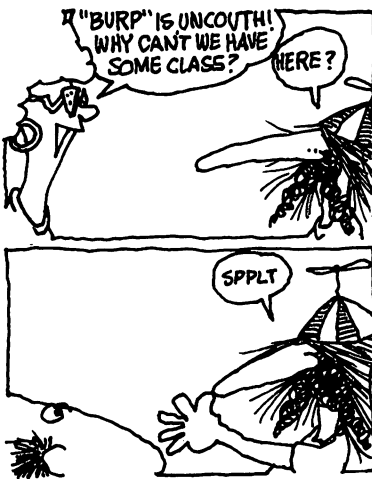
Captain Action uses his computer to keep track of his Bug Byte Beer consumption. To make things simple, he chose the shortened *variable name* BURP to represent the number of cans of beer he has consumed.

BURP is not as outrageous as it sounds. You can assign any name or abbreviation you want to a variable, as long as it begins with a letter.

Uh Oh! It Doesn't Recognize More Than Two Characters.

One Wacko word of caution: Captain Action selected the four-letter variable name **BURP**, but your Commodore 64 only recognizes the first two letters — the **B** and the **U**! Your computer gets confused if your program contains two or more numeric variables with the same first two letters. If you use BURP and BUMMER, for example, your computer will think they are the same variable and go bzzzrk!

You're right, Ms. Peeky. If Captain Action was a real gentleman, he might have assigned the variable name CANS, BEER, or...BLAZZT!

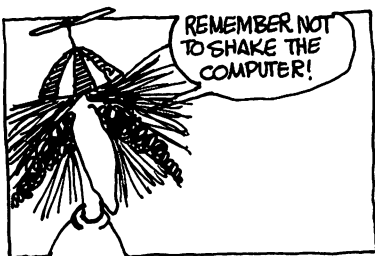


LET's BURP Along with Captain Action

Turn on your fingertips again, and we'll BURP along with Captain Action. Captain Action consumed 99 cans of Bug Byte Beer today, and LET his computer know about his gluttony by typing:

```
LET BURP = 99
```

Now because the variable BURP has been assigned a value of 99, his (and your) Commodore 64 is filled with 99 cans of beer.



DR. WACKO PRESENTS COMMODORE BASIC

You Don't Have to Use LET

Your Commodore 64 is really smart. You don't have to use LET to let it know that a value is assigned to a variable. You can write it like this: **BURP = 99**, and your Commodore 64 will know what you're talking about. But in general, it's a good idea to LET your Commodore 64 know officially that you're assigning a value to a variable.

Now before your computer gets tipsy, tell it to:

```
PRINT BURP
```

It will respond by (politely!) belching the number 99.

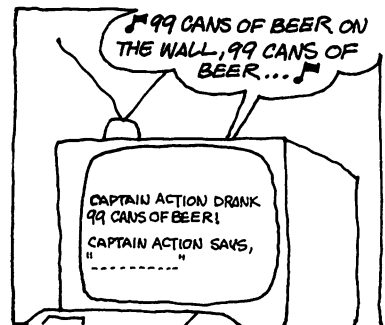
Hmm, before things get too messy, let's get out of the *immediate* mode and write a program using the variable BURP.

Burp

```
10 BURP = 99
20 PRINT "CAPTAIN ACTION DRANK";
   BURP;"CANS OF BEER!"
```

Your screen will look like this after you run the BURP program:

Since BURP is equal to 99, 99 is printed on your screen when line 20 is executed. Replace the 99 in line 10 with your favorite number, and RUN the program again...hic!



Semicolons Are Surrounding BURP!

Do you see the semicolons surrounding the variable BURP? The first semicolon prints 99 after the word "DRANK", and the second semicolon insures that the word "CANS" is printed on your screen directly after

Basic BASIC Training

the 99. *Always surround a variable with semicolons when it's embedded in a PRINT statement.*

BURP Is a NUMERIC Variable

BURP is a *numeric variable* because you assigned a *number* to it. Here are some examples of numeric variables:

1. My waist size after eating too much falafel: INCHES = 52
2. The number of Petunia's hair-style changes in an average week: STYLES = 12
3. The price of a good camel: SHEKELS = 120
4. Junior's grades: GRADES = 0

Short and Sweet

When you leave this book, and start going wacko on your own, you'll want to keep your variable names as short as possible. I'm using extra long variable names now for the fun of it, and to show you how variables are used. But, when you're off and running, remember the Two-Line Limit! Keep your programs short and sweet. For example, you can use the variable name "IN" instead of "INCHES", or "ST" instead of "STYLES". And if you really want to save space, just use one letter as a variable's name. For example, you can use "G" instead of "GRADES" to really shorten up your program listing.

Add % to Your Variable's Name

Huh? That's what Junior said when I told him to add a percent symbol (%) to the end of his numeric variable names.

If you write a long program that uses *only whole numbers* (or integers, as they are called), it's a good idea to add a percent symbol (%) to the end of each variable's name.

DR. WACKO PRESENTS COMMODORE BASIC

This nifty trick conserves lots of computer memory because your Commodore reserves less space for variables ending with % than it does for simple variables — honest.

Enter and RUN these examples, and Dr. Wacko will explain all.

```
10 GRADES = 5.432
20 PRINT GRADES
```

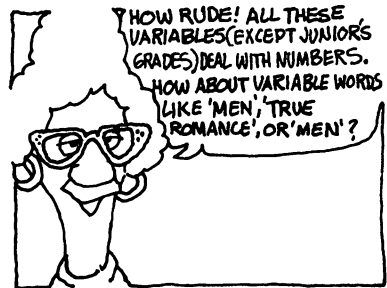
Did you RUN the program? You did? OK, what you see is what you got. Your computer printed 5.432. No problem. Now try this:

```
10 GRADES% = 5.432
20 PRINT GRADES%
```

The answer, this time, is 5! Surprised? Not to worry. It's 5 because you've added % to the end of a variable's name, and the answer is *rounded down* to the nearest whole number. And it did use less computer memory. Amazing!

Tuck this little gem of programming wizardry in the back of your head or under the Persian rug. You'll want to pull it out and use it when you write your next gigantic award-winning program.

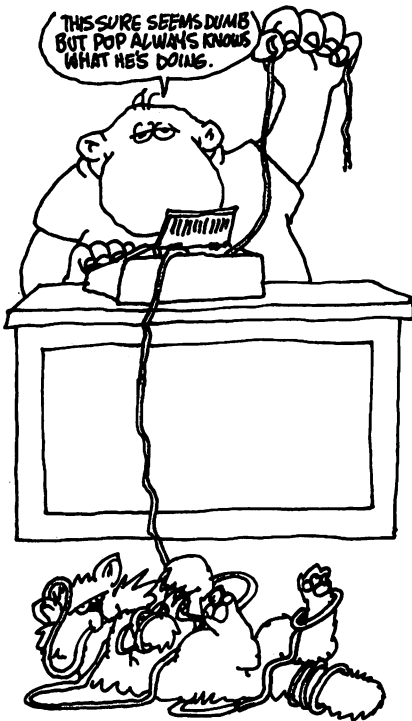
Ms. Peeky's got a point. All those variable examples do deal with numbers. In the BURP program, BURP equals the *number* 99.



Important Concept #2: STRING VARIABLES, or, The Verbal Variable

In Commodore 64 BASIC there's one other type of variable, called a *string variable*, that's just perfect for invariably verbal people like Ms. Peeky.

Basic BASIC Training



What's a String?

A *string* is no more than a stringing together of letters, characters, or spaces. Each character, word, space, sentence, and paragraph on this page is a string. Or you could think of all the words in this book as one long string!

Using string variables is lots of fun! They let you get really verbose and obnoxious, like me! Also, string variables are easy to use. Even the Wacko Cats (Keys, Paddles, and Joystick) like to play with string.



But First, Tell Your Commodore 64

To tell your Commodore 64 that you are using a string variable, rather than a numeric variable, just put a dollar sign (\$) after the variable's name and surround the string with quotation marks like this:

A\$ = "BELCH!"

In this example the *string variable name* is A\$, and the *string* is BELCH!

Your variable name doesn't have to be "A\$", any single letter, or short word followed by a dollar sign will work just fine. Here's what string variables might look like if you're weird like me:

DR. WACKO PRESENTS COMMODORE BASIC

```
A$ = "BELCH!"  
NAME$ = "STRING"  
INCHES$ = "FIFTY-TWO"  
TIME$ = "2 P.M."  
NAME$ = "Dr. C. Wacko"  
GRADES$ = "0"
```

Numbers, when they're surrounded by quotation marks, become strings. Miraculous!

Let's write a short program using a the string "BELCH!" Stand back...here it comes.

```
10 A$ = "BELCH!"  
20 PRINT "CAPTAIN ACTION DRANK ";A$;  
   "CANS OF BEER"
```

Here's another example:

```
10 A$ = "BELCH!"  
20 B$ = "Captain Action"  
30 PRINT B$;" drinks too much beer...";A$
```

Now, lets get real tipsy and use numeric and string variables in one stupendous and horrific program:

Horrific Program

```
10 A$ = "BELCH!"  
20 B$ = "Captain Action"  
30 BURP = 99  
40 PRINT B$;" drank ";BURP;" cans of  
   beer...";A$
```


Now Type CLR To Clear Up This Mess!

Numeric and string variables are two of my favorite programming concepts. I'll be showing off while showing you how to use these two phenomenal concepts throughout the rest of this phenomenal book. But for now, I'd like to show you what happens if you don't clear out your computer's memory after you've assigned strings and numbers to variables.

First, enter and RUN the Horrific Program. READY? That's what your computer should say after you've followed my instructions.

Now, enter the following in the *immediate* mode:

PRINT A\$

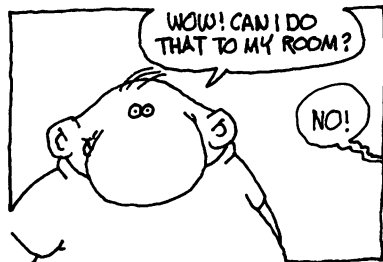
Oops! Your Commodore 64 just belched again! Now, try this in the *immediate* mode:

PRINT B\$

My best friend's name! And, this:

PRINT BURP

Unfortunately, once you assign a repugnant name or number to a variable, your computer remembers it forever — almost.

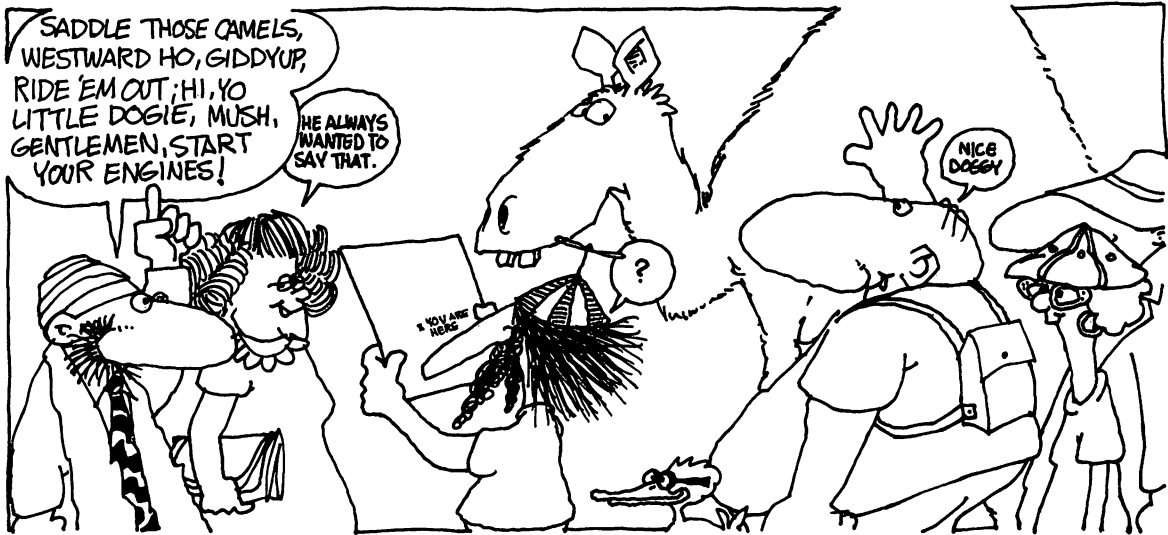


To clear up this mess, and clean up your computer's act, just type:

CLR [RETURN]

Now, your computer's variable tables are as empty as the desert. But finally, it's time to leave boot camp and push on to the Great White Expanse. So put on your sunglasses, hop on your camel, and we're off!!

3 Entering the Great White Expanse



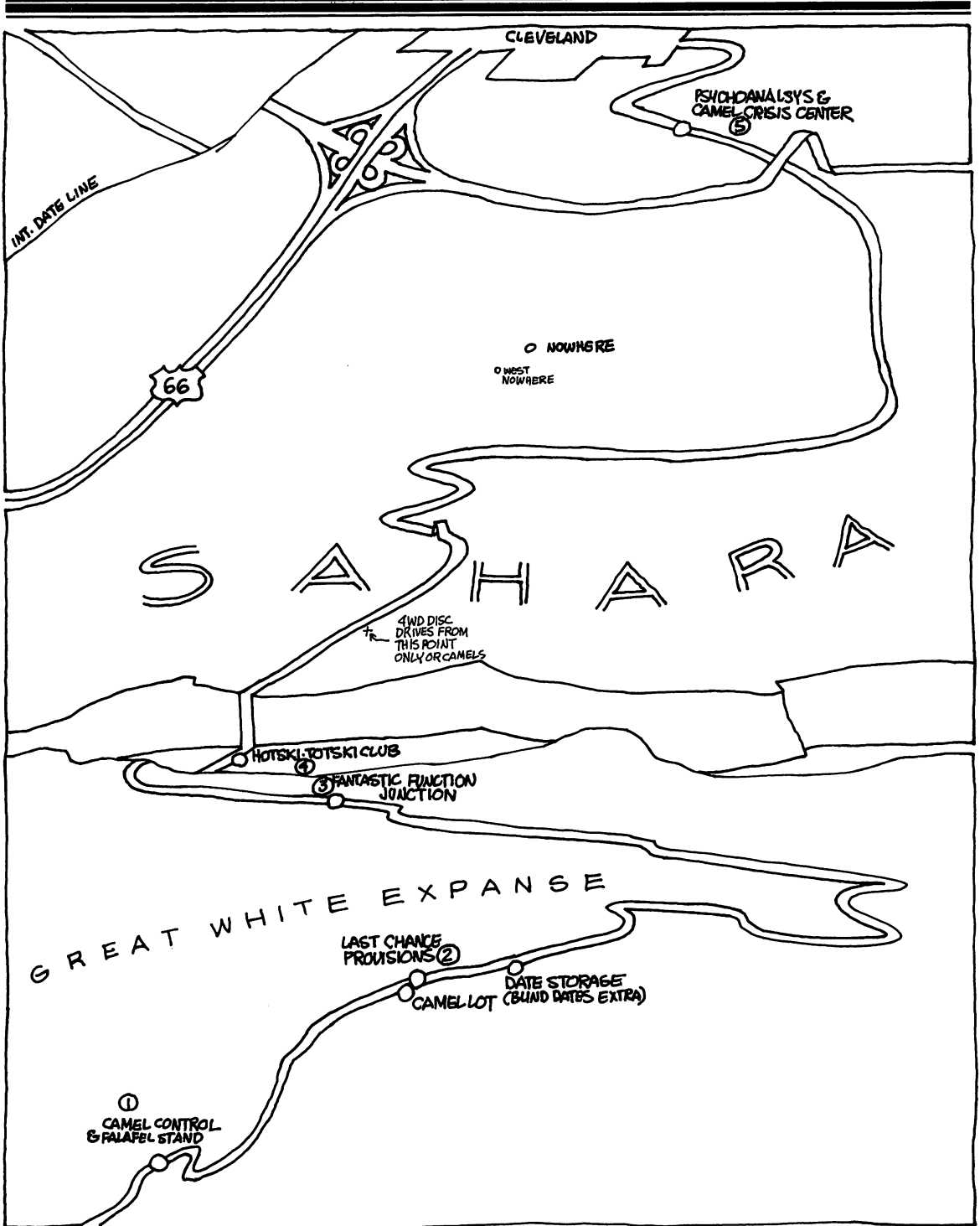
Nice to see that you made it through Basic BASIC Training! You made it without going AWOL, your hair's starting to grow back, and you have a working knowledge of some useful BASIC programming tools. Now, it's time to join the caravan and enter the Great White Expanse.

You'll build your BASIC vocabulary with each step through these pages. Every page is like a small oasis, presenting short, and if you're hungry, easily digested programming treats.

As you journey through the Great White Expanse you'll discover exciting BASIC words, with simple examples that show you how to use each word to talk to your computer and get your programs under control.

Before you go charging off, let's take a quick look at this map.

The Great White Expanse



DR. WACKO PRESENTS COMMODORE BASIC

Oasis 1: Control That Camel. When the caravan stops at this oasis, you'll have a snack and learn all the BASIC words and commands that will get your program under control. You wouldn't want your program to run off without you, would you?

Oasis 2: Provision Your Caravan and Store Your Dates. This oasis is loaded with more tasty delights, just ripe for the picking. Here, I'll show you how to store your treats inside your computer so you can continue across the Great White Expanse with a full stomach.

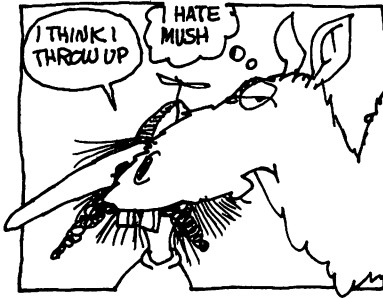
Oasis 3: Fantastic Function at the Oasis. Its time for a little revelry. At this oasis, you'll learn three BASIC functions, SGN, INT, and RND, that are guaranteed to add a touch of professionalism and wackiness to your programs.

Oasis 4: Don't String Me Along, Including Strings Revisited, ASCII Codes, and the Hotski-Totski Chart. The sun will really get to you by the time you arrive at this oasis. When we arrive, we'll take a closer look at all the fun things you can do with strings. Then, I'll introduce you to the frivolous Hotski-Totski Charts!

Oasis 5: Chat with Your Fellow Travelers and Psychoanalyze Your Computer. Your journey is almost over. And as the sun sets over the Sahara, it's time to swap camel jokes, talk to your programs, disk drive, and cassette recorder, then peek inside your Commodore 64's brain and poke around.

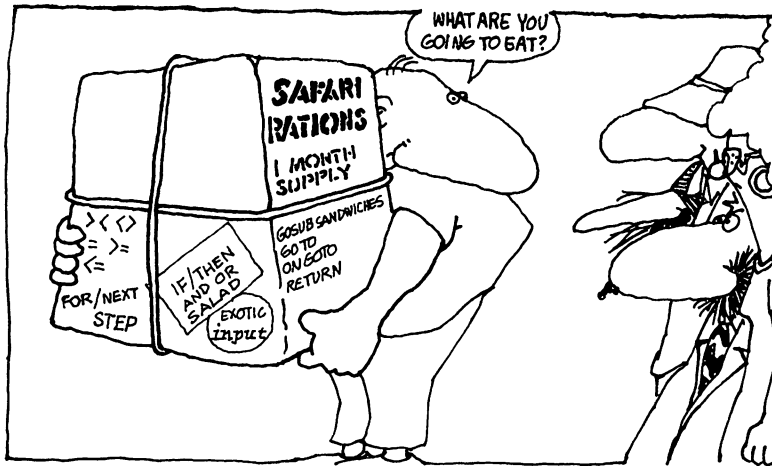
After you've gathered a gigantic BASIC vocabulary and charted each oasis, you'll fly off (on your magic carpet, of course!) to the mystical worlds of Graphics, Sound, and Creative Programming where you will reinforce and expand your new knowledge. And here's the best part: If you want a quick review, you can always revisit any oasis to sip a refreshing drink from its well of knowledge.

The Great White Expanse



Have you packed your sunglasses? Ready to join the caravan through the Great White Expanse? Then put on your mukluks, and we'll get going! Mush, Clyde!

Oasis 1: Eat Some Falafels and Control Your Camel



Welcome to your first oasis. You must be a little bushed and hungry. So, pull up a cushion, make yourself comfortable, eat a few falafels, and learn how to control that camel of yours.

Here beneath the swaying palms, I'll show you how to combine your Basic BASIC Training skills with some new BASIC words to increase your programming prowess and really take control of your programs.

The Querulous INPUT

You've probably heard the expression "garbage in, garbage out." Well, the fabulous INPUT statement lets you make this expression become a reality. When you use INPUT, your computer asks a weird question and waits politely for you to enter an even stranger answer, and press **RETURN**. Simply put, an INPUT statement lets you (you guessed it) put information into your computer.

Here, I'll show you one way to use the INPUT statement in a short, but fattening, program:

The Great White Expanse

Falafel Calorie Counter

```
10 PRINT "HOW MANY FALAFELS DID YOU  
EAT";  
15 REM ***  
20 INPUT FALAFELS  
25 REM ***  
30 CA = FALAFELS*200  
40 PRINT :PRINT CA;" CALORIES! MS. PEEKY  
WOULDN'T"  
50 PRINT "APPROVE!"
```

The Falafel Counter program was one of Ms. Peeky's inventions. She originally designed it to embarrass everyone at the oasis so she could sneak extra falafels on the sly. Still, she did a great programming job! She demonstrated one way to use the INPUT statement. Here's what else she did:

1. *She provided a nomad-friendly users guide.* In line 10, Ms. Peeky was thoughtful enough to tell us what information to type into the program. If she hadn't made her program nomad-friendly, all you'd see on your screen would be a lonely question mark.

Ms. Peeky also added a semicolon (;) at the end of the PRINT statement so the question mark (generated by the INPUT statement) appears on your screen right after she asks you how many falafels you ate.

Take out the semicolon, run the program again, and watch what happens to that querulous question mark. Wheew!

2. *She made her program easy to read.* In lines 15 and 25, Ms. Peeky used REM statements to make the INPUT statement on line 20 stand out.

DR. WACKO PRESENTS COMMODORE BASIC

She also went to a lot of extra trouble when she chose “FALAFELS” and “CA” (to represent Calories) as her variable names. Her choices make the program easier to understand. She could have used really abbreviated forms, like *F* for “FALAFEL,” and *C* for “CALORIES.”

3. *She was overwhelmed in line 30.* She made the numeric variable “CA” equal to “FALAFELS*200”, and then imbedded “CA” in the PRINT statement on line 40.

INPUT and the Automatic PRINT Statement

Ms. Peeky’s Falafel Calorie Counter works great. It does just what it’s supposed to. But it can be improved! Here’s a wacko way to combine lines 10 and 20 into one nifty INPUT statement that uses what I call the “Automatic PRINT Statement.” Just follow along with Wacko and you’ll see what I mean.

First totally annihilate lines 10 and 20 of the Falafel Calorie Counter program, then replace them both with this new line 10:

```
10 INPUT “HOW MANY FALAFELS DID YOU  
EAT”;FALAFELS
```

Now, RUN the program and watch the result.

Zonkeroo! It’s the same!

Using the Automatic PRINT Statement is a snap. Just use the format I’ve used, and *make sure that the prompt message is 38 characters or less.*

You can use INPUT with a PRINT statement like I did in the original program, or get spiffy and use the Automatic PRINT Statement. The choice is yours. They both achieve the same results!

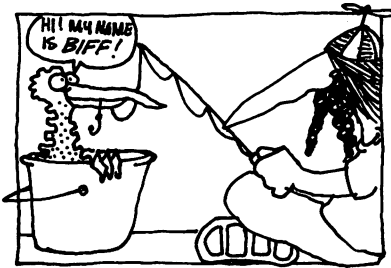
The Great White Expanse

Using INPUT with a String Variable

Here's one of Paddle's favorite programs:

No Fishing at the Oasis!

```
10 PRINT "HEY CAT! WHAT KIND OF FISH IS  
    THAT"  
15 REM ***  
20 INPUT FISH$  
25 REM ***  
30 PRINT :PRINT "THIS ISN'T A FISH. IT'S  
    A ";FISH$  
35 REM ***  
40 PRINT :PRINT "CAN'T YOU READ?"  
50 PRINT "IT SAYS, NO ";FISH$;"ING!!"
```



When the Oasis Warden asks you what kind of fish you've caught, have some fun and enter a strange name, like "LOBSTER", "WACKO", or "MS. PEEKY"!

The Easy To Use GOTO

Even if Clyde's not going anywhere, your program can, with the easy-to-use, and utterly simple, GOTO statement. Try this:

```
10 PRINT " BUY AN EDESEL! ";  
20 GOTO 10
```

Besides filling up your screen with advertisements for defunct products, the versatile GOTO statement can spiffy up programs like the No Fishing gem you just played with.

Every time you want to use the No Fishing program you'll have to type the word RUN and press **RETURN** . What a waste of time! It's also hard on the fingernails. Well, fear not, GOTO to the rescue! Add the following

DR. WACKO PRESENTS COMMODORE BASIC

new line 60 to the No Fishing program and all your nail-splitting problems will be solved:

60 GOTO 10

When you've finished razzing the Oasis Warden, you'll notice that the program goes to line 20 and you can go fishing again!

Here's another short example:

Another Short Example

```
10 INPUT "HOW MANY SHEKELS DID YOUR  
CAMEL COST";SHEKELS  
20 ? :? SHEKELS;" SMACKERS! GREAT DEAL!"  
30 ? :GOTO 10  
40 REM — A QUICK REMINDER: '?' IS THE  
SAME AS 'PRINT'
```

See how much time the GOTO statement saves you? But that's not all! You can also use GOTO to count your shekels.

Counting Shekels

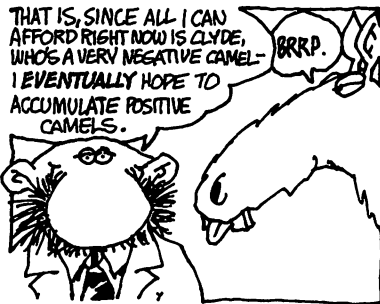
```
5 REM WE START WITH 1 SHEKEL.  
(INITIALIZE THE VARIABLE)  
10 S = 1  
15 REM NOW WE PRINT OUR BALANCE.  
20 PRINT S  
25 REM SHEKELS EQUALS SHEKELS PLUS 1.  
30 S = S + 1  
35 REM BACK TO LINE 20 TO BEGIN AGAIN.  
40 GOTO 20
```

(To keep things manageable I've used **S** to represent "Shekels".)



The Great White Expanse

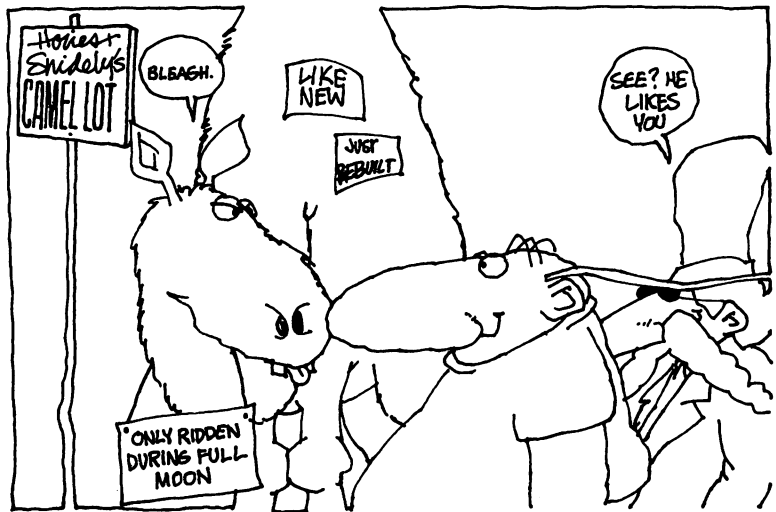
Accumulating the Maximum



I play the Game of Life to *accumulate* the maximum number of positive camels! Unfortunately, Snidely Seersucker doesn't play by the same rules. His goal is to sell the maximum number of used camels, and accumulate the maximum number of shekels! Snidely's dastardly accumulations are made possible with a little help from the GOTO statement.

Regardless of whether you take your camel with one hump or two, the best way to experience Snidely's accumulating experience is to walk on to his Used Camelot and start buying a few camels.

So, first enter Snidely's Used Camelot, lose all your money, get in debt, and I'll counsel you when you return.



DR. WACKO PRESENTS COMMODORE BASIC

Honest Snidely's Used Camelot

```
10 SH = 5000:CA = 1:T = 0:SP = 0
20 PRINT "***SNIDELY'S FRIENDLY USED
   CAMELS***"
30 PRINT:PRINT "WHAT DO YOU OFFER FOR
   CAMEL #";CA;
40 INPUT SP
50 PRINT:PRINT "YOU NOW, (SNICKER)
   OWN";CA;"USED CAMEL(S)"
55 REM ***
60 CA = CA + 1:SH = SH - SP:T = T + SP
65 REM ***
70 PRINT "YOU'VE GOT";SH;"SHEKELS LEFT"
80 PRINT "THAT'S $";T;"WISELY SPENT.
   (SNARK!)"
90 PRINT
100 GOTO 30
```

A Tour through Camelot

SH: The number of **SH**ekels you have (or don't have!).

CA: The number of each **CA**mel, and the number of **CA**mel's purchased.

T: The *accumulated Total*

SP: The amount you foolishly **SP**end on each camel

SNARK: A dastardly exclamation.

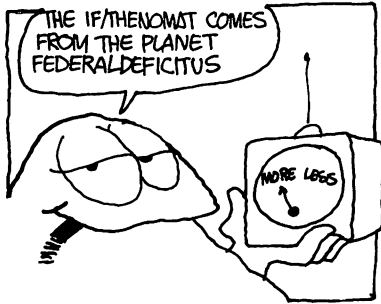
SNICKER: Snidely's way of saying hello.

In line 10 all the variables are "initialized" (set to their beginning values). So, you started with 5000 shekels, and, if you weren't careful and paid too much for each used camel, you quickly discovered that you were in debt to Snidely. One of his nastier tricks.

Line 60 is where Snidely keeps track of how many camels he's sold, $CA = CA + 1$. He then keeps tabs of how much money you spend and have left,

The Great White Expanse

$SH = SH - SP$, and finally accumulates all his ill-gained wealth, $T = T + SP$.



The Dynamic, Decision-making IF/THEN

It's a good thing that we brought the brilliant IF/THEN statement along with us on our trek across the Great White Expanse. IF/THEN is really the brains of this motley outfit. It makes most of our decisions, and we'd be lost without it.

The IF/THEN statement compares two things and divines the truth. It has a simplistic, but effective, philosophy. To the IF/THEN, something is either true or false. But before you use it, you need to know how to indicate comparisons.

- > Means *greater than*
- < Means *less than*
- = Means *equal to*
- > = Means *greater than or equal to*
- < = Means *less than or equal to*
- < > Means *not equal to*

How the IF/THEN Makes Comparisons

- IF $A > B$ THEN Means *If A is greater than B*
- IF $A < B$ THEN Means *If A is less than B*
- IF $A = B$ THEN Means *If A is equal to B*
- IF $A > = B$ THEN Means *If A is greater than or equal to B*
- IF $A < = B$ THEN Means *If A is less than or equal to B*
- IF $A < > B$ THEN Means *If A is not equal to B*
- IF $A = X$ AND $B = X$ THEN Means *If both A and B are equal to X*
- IF $A = X$ AND $B = Y$ THEN Means *If A equals X and B equals Y*
- IF $A = X$ OR $B = Y$ THEN Means *If A equals X or B equals Y*

DR. WACKO PRESENTS COMMODORE BASIC

What Happens THEN?

After the IF/THEN statement makes a comparison and divines that it is true, your computer does whatever you put after THEN. If the comparison is false, you computer moves on to the next program line.

Take out the following magic lamp, rub it a few times, and the mystery of the IF/THEN statement will be revealed.

ASK THE GENI

```
10 INPUT "HOW MANY WISHES DO YOU
    WANT";WISHES
15 REM ***
20 IF WISHES > 3 THEN GOTO 40
25 REM ***
30 PRINT "YOUR WISH IS MY
    COMMAND.":GOTO 10
35 REM ***
40 PRINT "AREN'T YOU BEING A LITTLE
    GREEDY?":GOTO 10
```



IF the comparison in line 20 is true (you got greedy and asked for more than three wishes), **THEN** the program goes to line 40, and the Geni tells you off!

IF the comparison in line 20 is not true (you asked for three or less wishes) **THEN** the program skips on to line 30, and your wish is the Geni's command.



The Use 'M All Program

Ask the Geni shows how the IF/THEN statement makes a greater than (>) comparison. The Use 'M All program uses *virtually* all the IF/THEN comparisons possible. Note that the INPUT statement on line 20 of this program lets you enter and compare *two* numbers. You can enter your choice of numbers in either of two ways:

The Great White Expanse

1. Enter the first number, press **RETURN** and a double question mark appears (??), then enter the second number and press **RETURN** again.
2. Enter the first number, then enter a comma, enter the second number, and press **RETURN** .

This short program will surprise you with its brilliance, regardless of which method you use to enter your two numbers. So type it in, enter some numbers, and watch the amazing results.

Use 'M All

```
10 PRINT "ENTER ANY TWO (2) NUMBERS"  
20 INPUT N1,N2  
30 IF N1 = N2 THEN GOTO 100  
40 IF N1 < > N2 THEN GOTO 200  
50 IF N1 > N2 THEN GOTO 300  
60 IF N1 < N2 THEN GOTO 400  
70 IF N1 = 50 OR N2 = 100 THEN GOTO 500  
80 IF N1 < 100 AND N2 > 100 THEN GOTO 600  
90 PRINT:GOTO 10  
92 REM ***  
95 REM ***  
97 REM ***  
100 PRINT "N1 EQUALS N2":GOTO 40  
200 PRINT "N1 IS NOT EQUAL TO N2":GOTO 50  
  
300 PRINT "N1 IS GREATER THAN N2":GOTO  
60  
400 PRINT "N1 IS LESS THAN N2":GOTO 70  
500 PRINT "EITHER N1 EQUALS 50 OR N2  
EQUALS 100":GOTO 80  
600 PRINT "N1 IS LESS THAN 100 AND N2 IS  
GREATER THAN 100":PRINT :GOTO 10
```

DR. WACKO PRESENTS COMMODORE BASIC

Here's what your brilliant computer says when you enter the numbers 50 and 500:

```
N1 IS NOT EQUAL TO N2
N1 IS LESS THAN N2
EITHER N1 EQUALS 50 OR N2 EQUALS 100
N1 IS LESS THAN 100 AND N2 IS GREATER
THAN 100
```

Pretty smart, isn't it?

IF/THEN String Comparisons

There's some exciting news at the casbah! The IF/THEN statement can make more than just numerical comparisons. It can be used to compare two or more strings! Later, when we revisit strings, you'll learn lots of weird ways to get tangled up. But for now, here's a neat example that illustrates one common usage of IF/THEN string comparisons.

Ryde Clyde

```
10 INPUT "WHAT'S YOUR FIRST
   NAME";NAME$
20 PRINT NAME$;" , WOULD YOU LIKE TO
   RYDE CLYDE";
30 INPUT A$
35 REM ***
40 IF A$ = "Y" OR A$ = "YES" THEN GOTO 60
45 REM ***
50 PRINT "I DON'T BLAME YOU!":PRINT:GOTO
   10
60 PRINT "OUTRAGEOUSLY COURAGEOUS,"
   ;NAME$;"!":PRINT:GOTO 10
```

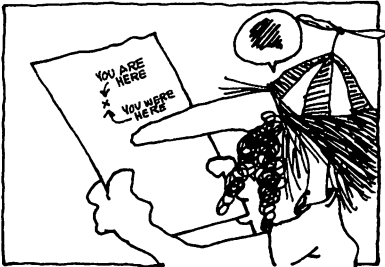


Here's how this outrageous program works.

The Great White Expanse

Line 10 waits for you to enter your name and press **RETURN**, then line 20 waits for and accepts your decision. Line 40 is where all the decision-making occurs. If you enter a string that begins with either a capital Y or the word YES the program assumes you say, “YES” and zips outrageously to line 60. But if you enter any other letter or letters, the program *assumes* that you mean “NO,” and zips to line 50 where you are praised for your common sense!

The IF/THEN statement’s decision-making and logical branching abilities are key programming elements. As you continue your adventure and begin developing your own brilliant programs, the IF/THEN statement will become one of your most valuable tools — almost as valuable as a date picker.



Going around in Circles with the Dizzy FOR/NEXT Loop

It’s easy to get lost out here in the desert. Caravans have been known to go around in circles, repeating the same course for days, before getting back on track again.

Getting dizzy is very unhealthy for caravans. But going around in circles can actually improve and enhance your programs! When you need to repeat part of your program a specific number of times, the dizzy FOR/NEXT loop is just what the shiek ordered.

The FOR/NEXT loop has many camel-boggling applications. But since a picture is worth a thousand shekels, here’s a very short, and eye-popping, example. Some of the elements contained in this short eye popper might be new to you. Don’t panic! Take a chance and type it in anyhow. The results are well worth the risk.

DR. WACKO PRESENTS COMMODORE BASIC

Eye Popper

```
10 FOR X = 1 TO 255
20 POKE 53281,X
30 NEXT X
```

Don't be squeamish! Open your eyes wide and run this short program.

Wasn't that fabulous? Your screen looked like the aora-boringalofus, and your eyes are probably bloodshot!

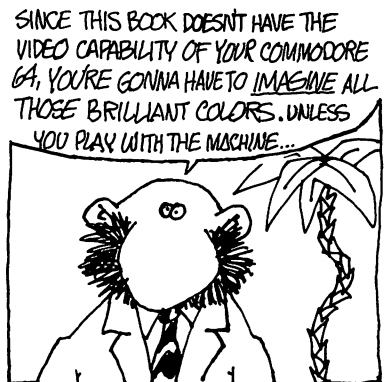
To set your screen back to blue just enter the following in the *immediate* mode: **POKE 53281,6**

Thanks to the FOR/NEXT loop, you've just sampled all the brilliant colors your Commodore-64 can deliver. Later, I'll show you how to astound your camels with more psychedelic color wizardry. But now it's time to show you how the FOR/NEXT loop made it all possible.

The Eye Popper program is real easy to understand. The dizzy FOR/NEXT routine begins on line 10 with the statement **FOR X = 1 TO 255**. This simply means that the value of X equals 1 when the program starts, and will equal 255 when FOR/NEXT is finished looping around.

All that snazzy color-changing takes place in line 20. The value of X in **53281,X** is changed from 1 to 255, to whatever the current value of X is by the FOR/NEXT loop that surrounds it. Line 30, **NEXT X**, tells the program to go back to line 10 and add 1 to the value of X.

A FOR/NEXT loop can be thought of as a garbanzo bean sandwich — a handful of programming statement(s) squashed between a FOR and NEXT statement.



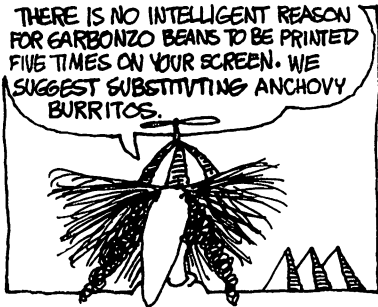
The Great White Expanse

Here's another short example that will help you get the picture — or heartburn.

Garbanzo Beans

```
10 FOR A = 1 TO 5
20 PRINT "GARBANZO BEANS"
30 NEXT A
```

See! The garbanzo beans *are* squashed between a FOR and NEXT statement! When you run this tasty program, the words "Garbanzo beans" are printed on your screen five times.



If you really like garbanzo beans, just change the number 5 in line 10 to a larger number and run the program again.

Now, to get real sneaky, add this line and run your program.

```
25 PRINT A
```

Wow! The value of A increases by 1 each time the program takes a bite of the sandwich.

I've used the variable A instead of X in this example. You can use any variable you like. You've just got to be consistent. If you use A after the FOR statement, use it again after the NEXT.

Stay in STEP with More FOR/NEXT Loop Madness

The STEP statement is used to make a FOR/NEXT loop count in increments other than 1.

Here's a high-step'n example that illustrates this wondrous delight:

DR. WACKO PRESENTS COMMODORE BASIC

High Step'n

```
10 FOR X = 0 TO 20 STEP 2
20 PRINT X
30 NEXT X
```

Run this spiffy number and you'll see that the program is now counting by twos!

To really grasp this concept, make it count by fours by changing the number that follows STEP to a 4.

You can also use STEP to count *down* by subtracting instead of adding to X. Here's an example that counts down from 20 to 1 in one-step increments:

Countdown

```
10 FOR X = 20 TO 1 STEP -1
20 PRINT X
30 NEXT X
```

Experiment with these concepts a little until you have a good feeling for the way STEP works in a FOR/NEXT loop. Remember, the number following STEP can be either positive or negative. It can even be a Decimal, like 0.5!

Pause for a Moment!

Now that you're familiar with the FOR/NEXT loop and STEP, here's a common application I call PAUSE. It's one that you'll use often in your programs.

If things are happening too fast in your program — the display doesn't stay on the screen long enough, a sound is too short, or you want a color to remain on the screen for a fixed period — it's time to use my infamous Pause program.

The Great White Expanse

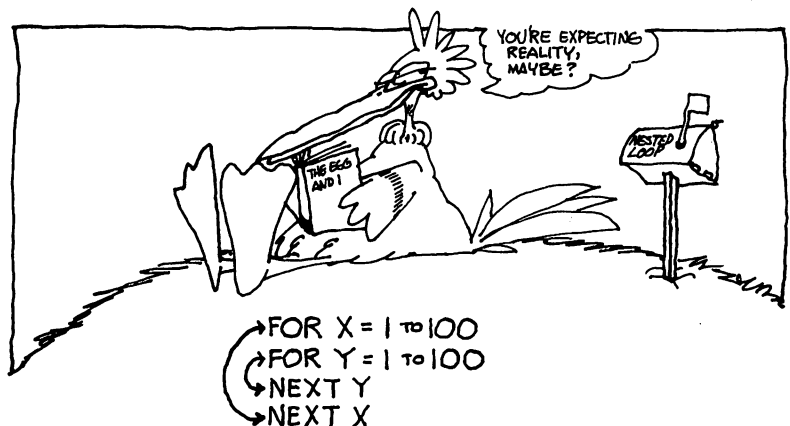
Pause

```
10 FOR X = 1 TO 10
20 PRINT,X
25 .
30 FOR PAUSE = 1 TO 100
40 NEXT PAUSE
45 .
50 NEXT X
```

When you run the Pause program you'll notice a pause before the computer prints each number on your screen. That's because the Pause routine is cleverly inserted at lines 30 and 40 of the program. If you want to shorten the delay, just decrease the value of the number following TO. To lengthen the interval, increase the value.

Nested FOR/NEXT Loops

I'll bet you didn't realize when you typed in the Pause program that you were entering a "nested loop." A nested loop is simply one or more FOR/NEXT loops nested inside other FOR/NEXT loops.



DR. WACKO PRESENTS COMMODORE BASIC

In the Pause program, the PAUSE routine on lines 30 and 40 is a nested loop because it's inside another FOR/NEXT loop.

This next program, Captain Action's Hallucination, is one of my favorites. It gets lonely out here in the desert, and Captain Action wanted to show off his hallucinations. A mirage or two is always a welcome relief from the tedium of desert life. So, turn off your mind, turn on your Commodore, and RUN this short program. It shows how nested loops are structured, and I guarantee it will make you weird.

Captain Action's Hallucination

```
10 FOR A = 0 TO 15
20 FOR B = 15 TO 0 STEP -1
30 FOR PAUSE = 1 TO 20
35 REM ***
40 POKE 53281,A
50 POKE 53280,B
55 REM ***
60 NEXT PAUSE
70 NEXT B
80 NEXT A
```

Line 10 of Captain Action's Hallucination works with the POKE statement on line 40 to loop through all of Commodore's sixteen background colors. Line 20 works with the POKE statement on line 50 to loop through the border colors, beginning with the last color and working down to the first color. (You'll learn all about color and POKEs in a little while.)

Line 30 is the first part of the PAUSE routine. Changing the 20 to any other number will vary the length of the Captain's hallucination. (The shorter the better.)



The Subterranean Subroutine, Starring GOSUB and RETURN

Lurking in the depths of the oasis is the dark and mysterious subterranean subroutine. A subroutine is a chunk of programming that your program uses one or more times to do something special.

When I get hungry I send out for a date-anchovy pizza from the oasis pizzeria. My special treat is delivered in a flash by the local merchant. When your program wants a special treat, it sends its order to the subroutine and, voila, instant delivery!



I'll show you a devastating example in just a second. But before you send an order to a subroutine, you should know that GOSUB is used to send the order to the subroutine and RETURN zips your computer back to the main part of your program after the subroutine has filled the order.

It Remembers from Whence It Came!

GOSUB works just like GOTO. It sends your program racing to the line number placed after it. And, with a little help from its partner RETURN, the program remembers where it came from and returns after it has finished executing the subroutine.

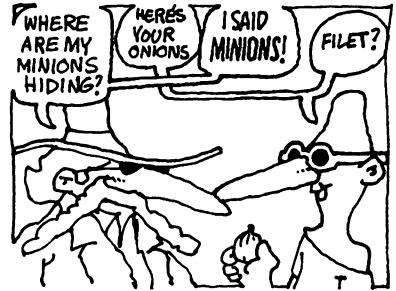
DR. WACKO PRESENTS COMMODORE BASIC

Snidely, always the perfect villain, has devised this subterranean example:

Where Are My Minions?

```
10 REM *** THE MAIN PART OF THE
PROGRAM:
15 REM ***
20 PRINT "[CLRHM][CTRL + 6]SNIDELY:
WHERE ARE MY MINIONS HIDING?":GOSUB
60
30 PRINT "SNIDELY: SNARK!"
40 END
45 REM ***
50 REM *** THE SUBTERRANEAN SUBROUTINE:

55 REM ***
60 PRINT "[CTRL + 3]LURKING IN THE
DEPTHS OF YOUR PROGRAM.
[CTRL + 6]":RETURN
```



The GOSUB in line 20 tells the program to race down to the subroutine in line 60. After Snidely receives his answer in line 60, the RETURN sends the program back to line 30, where Snidely makes a snide remark.

The END in line 40 is used to separate the main part of the program from the subroutine below it.

Snidely's dastardly program shows how the GOSUB works with RETURN to get to and return from a subroutine. But remember, subroutines come in real handy when you've written a program containing short chunks that it uses repeatedly. Later, when I help you weave the perfect flying carpet, you'll experience more subroutine madness.

The Smart and Speedy ON GOTO

The ON GOTO statement is smart, speedy, and patient. Working with an INPUT statement, it waits for numeric input, then races to the correct line number. Here's what a typical ON GOTO statement looks like:

```
ON N GOTO 100,200,300,400
```

In this example, if the value of N is 1, your program races to line 100, the first line number after the command GOTO. If N equals 3, line 300, the third number after GOTO is executed. (Off with its head!)

Use the NEW command to clean out your Commodore's brain, then enter and run this short program to see how smart the ON GOTO really is.



Smarty Harem Pants

```
10 PRINT:"ENTER A NUMBER FROM 1 TO 5";
20 INPUT N
25 REM ***
30 ON N GOTO 100,200,300,400,500
40 REM ***
100 PRINT "IT'S A 1! (OR A 0, OR A NUMBER
    GREATER THAN 5).":GOTO 10
150 REM ***
200 PRINT "YOU ENTERED A 2.":GOTO 10
250 REM ***
300 PRINT "YOU ENTERED A 3.":GOTO 10
350 REM ***
400 PRINT "YOU ENTERED A 4.":GOTO 10
450 REM ***
500 PRINT "YOU ENTERED A 5.":GOTO 10
```

See how smart ON GOTO is? The program goes, in order, to the correct line number. Just for fun, scramble line 30 like I've done below, and run SMARTY HAREM PANTS again.

DR. WACKO PRESENTS COMMODORE BASIC

30 ON N GOTO 200,300,500,100,400

Now, when you enter the number 1, the program goes to line 200! (The first line number after GOTO.)

If you enter 0 or a number greater than 5, the program goes to the first line number on ON GOTO's list. But don't enter a negative number or a number greater than 255. If you do, your computer will get heartburn and spit out an error message. Try it, if you have a strong constitution.

I use the smart and speedy ON GOTO in programs to give the you a choice of options. What's Up includes some nifty programming techniques (in line 10) to clear the screen and turn off the cursor.

This stupendous program also contains a PAUSE subroutine in line 1000. Since the program uses PAUSE three times, the subroutine saves a lot of typing and makes this program superefficient!

What's Up

```
5 REM *** LINE 10 CLEARS THE SCREEN.
10 PRINT "[CLRHM]"
15 REM ***
20 PRINT "1. WHAT'S CLYDE'S FAVORITE
   WORD?"
30 PRINT "2. WHAT DOES SNIDELY SAY?"
40 PRINT "3. WHAT IS JUNIOR'S I.Q.?"
50 PRINT :PRINT "ENTER A NUMBER FROM 1
   to 3";
55 REM ***
60 INPUT N
65 REM ***
70 ON N GOTO 100,200,300
75 REM ***
100 PRINT "GRUNT!":GOSUB 1000:GOTO 10
120 REM ***
200 PRINT "SNARK!":GOSUB 1000:GOTO 10
```



The Great White Expanse

```
220 REM ***
300 PRINT "—35":GOSUB 1000:GOTO 10
320 REM ***
340 REM *** HERE'S THE SUBROUTINE!
350 REM ***
1000 FOR PAUSE = 1 TO 1000:NEXT
    PAUSE:RETURN
```

ON GOSUB

ON GOSUB is just about the same as ON GOTO. A typical ON GOSUB statement looks like this:

```
10 ON N GOSUB 1000,2000,3500
```

In this example, a response of 1 sends the program racing to a subroutine on line 1000. If the response is 3, the program goes to the subroutine in line 3500. After the subroutine has finished doing its thing, the program returns to the next statement after the GOSUB.

In this example if N has a value less than 1 or greater than 3, the program will omit any subroutine and go on to the next line. Negative numbers, or numbers greater than 255, make the computer display an error message. (Oops!)

DR. WACKO PRESENTS COMMODORE BASIC

Oasis 2: Provision Your Caravan and Store Your Dates

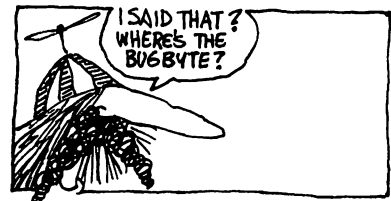


Congratulations! You just left Oasis 1, one of the longest stopovers in this entire book! You've taken a giant step, written some programs, and learned how to take control of them. This knowledge put you on top of the sand pile. And, as Captain Action once said, it's all down dune from now on. Wow!

I've been emptying sand out of my shoes waiting for you. Junior's been emptying sand out of his head. So, welcome to your second oasis. First, with a little help from a couple of really wacked-out date nuts, READ and DATA, I'll show you how and where to store all the supplies you've been lugging around. After breezing through a few pages you'll be twirling information in and out of your programs like a Sahara sand storm!

Next, after you've scattered some sand and really made a mess of things, my beautiful and talented wife, Petunia, will make an appearance to help us *all* get organized with her special storage system, *Arrays!*

This bit of desert paradise is chock full of easy-to-understand programming examples that, with a little im-



The Great White Expanse

agination, can be expanded into real blockbusters; not to mention something useful. So settle in, and check out the scenery.

Getting Organized with DATA

In many programs there's a real danger of tripping over numbers, strings, and figs at every stop of the way. What's a nomad to do? Just enter Plaudit to Wacko, RUN it, and I'll show you how it solves your storage problems.



Plaudit to Wacko

```
10 READ X
20 IF X = -1 THEN GOTO 50
30 PRINT X
40 GOTO 10
50 PRINT "WHO DO YOU APPRECIATE?
WACKO, WACKO, WACKO!":END
100 DATA 2,4,6,8,-1
```

After RUNning this strange program and viewing the humble results, you probably asked yourself, "What's being stored, where is it stored, and what's in store for me next?" I'll answer all these questions in order. After all, this chapter is about getting organized!



What's Being Stored and Where Is It?

The provisions you take with you on your programming adventure include *numbers and strings*. That's what you need to store! And to make your journey a snap, Commodore 64 BASIC lets you store them after a DATA statement!

DR. WACKO PRESENTS COMMODORE BASIC

“Wow” Is in Store for You Next

Storing numbers or strings after a DATA statement is just like placing words in a book. For example the word “Wow” has been waiting patiently for you to come along and *read* it. And what’s amazing is that you can go back and read it again, and again, and again, and... just like stored information you’ll find after a DATA statement, as you’ll soon see!

In Plaudit to Wacko I’ve stored the numbers 2,4,6,8 and —1 after the DATA statement in line 100. Here’s line 100 again so you can take a closer look at it:



```
100 DATA 2,4,6,8,—1
```

Numbers or strings are always placed after a DATA statement according to these four simple rules:

1. A comma *always* separates each chunk of data.
2. There is no comma before the first chunk or after the last chunk of data. So, in this example, there’s no comma before the number 2 or after the —1.
3. You can store as many numbers or strings on a line of data as you want, as long as you don’t exceed the Two-Line Limit we mentioned in Basic BASIC Training.
4. You can position a line, or lines, of data *anywhere* in your program, and the program will still work. Sound impossible? Just for fun, in Plaudit to Wacko change line 100 to read line 5, and rerun the program. It still works!

Go Bonkers and READ It!

Eeeeyargh! You know that numbers and strings can be stored after a DATA statement. But just like the words in this book, if you don’t *read* them, they won’t drive you bonkers. If you’re fearless, and want your program to read the stuff you place after DATA statements,

The Great White Expanse

you'll have to use the logically literate READ statement!

READ is always used with a variable, like this:

```
10 READ X
```



Here I've used the variable X, but you can use any variable you'd like. When your program reads string data, use a string variable, like A\$.

READING with an Invisible Finger



Junior always uses his finger to read. Likewise, READ has a mysterious, invisible finger (called a "pointer") that points to each number or string after the DATA statement.

The pointer moves from left to right (just like your eyes scanning a book) and sequentially reads *each* number or string. As the pointer "looks" at each number or string, the variable following the READ statement is assigned that value.

In Plaudit to Wacko, the invisible pointer first looks at the number 2. The program prints 2 on your screen because X equals 2, then the PRINT statement in line 30 is executed. Next, it points to the number 4, and 4 is printed on your screen. Next, well, I'm sure you get the point!

Just to make sure that you've got it, enter and run this stripped-down version of the Plaudit to Wacko program.

Stripped-down Plaudit

```
10 READ X
20 PRINT X
30 GOTO 10
40 DATA 2,4,6,8
```

DR. WACKO PRESENTS COMMODORE BASIC

Oops, a boo-boo! I'll bet you thought your camel was running on high octane and you were home free at the oasis, until the "?OUT OF DATA ERROR IN 10" appeared on your screen. Uh Oh!

Not to worry. Everybody goofs (except Dr. Wacko). Your computer is just telling you that the pointer ran out of data. After reading the number 8, it fell off the right edge of the program. Plop!

FLAG It down and END this Nonsense!

The original program didn't plop because I cleverly placed the number `-1` at the end of my data, right after the number 8.

I used `-1` as a *flag* to tell the pointer it had reached the last chunk of data. After the pointer read `-1`, the IF/THEN statement in line 20 told the program to go to line 50, print some nonsense, and come to a screeching halt — END.

Any number (or string) can be used as a *flag*. To see what I mean, modify Stripped-down Plaudit like so:

1. Change line 40 to read: **40 DATA 2,4,6,8,0**
2. Add this line: **15 IF X = 0 THEN END**

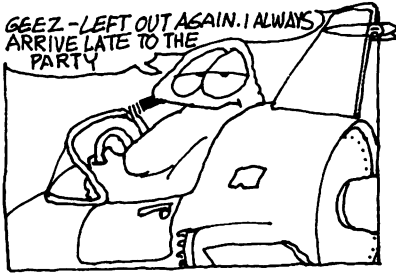
This time I used zero (0) as the *flag*. When you run your modified program, the IF/THEN statement in line 15 stops the program when it is flagged down by the 0.

Strings of Dates and Data

The DATA statement comes in real handy when you want to store a list of names, dates, pomegranates, or brilliant ideas. Here's a motley program full of motley wackos to demonstrate the DATA statement's string-handling versatility.

The Great White Expanse

Motley Crew



```
10 PRINT "PRESENTING OUR MOTLEY CREW:"
```

```
20 READ NAME$
```

```
30 PRINT NAME$
```

```
40 IF NAME$ = "SNIDELY" THEN END
```

```
50 GOTO 30
```

```
100 DATA DR. WACKO, MRS. PETUNIA  
WACKO, JUNIOR, CAPTAIN ACTION, MS.  
PEEKY
```

```
110 DATA CLYDE, SMOKEY PEEK, SLOW  
POKE, SNIDELY
```



Instead of numbers, this motley program is filled with wackos! We're all crammed into lines 100 and 110, right after the DATA statements, and each of us is separated by a comma. I've listed my name first, of course, and naturally Snidely Seersucker (that dastardly villain) is last. "SNIDELY" is the *flag* that indicates the end of the list. And because he wanted to see his name printed on the silver screen, I placed the IF/THEN statement in line 40, *after* the PRINT statement. Clever, no?



Juggling More Than One!

You know how to make your program point to and READ one number or string at time. No problem! Now comes the exciting part. Stand back as I show you how to turn your computer into a data-juggling virtuoso! Take a look at this READ statement — if you dare:

```
10 READ X,Y
```

This strange-looking statement contains two, count 'em, two variables separated by a comma! Right again, your computer can juggle more than one variable at a time!

Here, I'll show you. Just enter my infamous HI! program. When you're all set, RUN it! I'll explain all, after

DR. WACKO PRESENTS COMMODORE BASIC

you've witnessed your Commodore 64's juggling prowess.

HI!

```
10 PRINT "[CLRHM]"
20 READ X,Y
30 IF Y = -1 THEN END
40 POKE 1024 + X + 40*Y,102
50 GOTO 20
60 DATA 10,2,10,3,10,4,10,5,10,6
70 DATA 11,4,12,4,13,4
80 DATA 14,2,14,3,14,4,14,5,14,6
90 DATA 16,2,16,3,16,4,16,5,16,6
100 DATA 19,2,19,3,19,4,19,6
110 DATA -1,-1
```

Did you see that? A gigantic "HI!" This nifty program actually juggles two numbers to perform this mystical feat!

First, the pointer reads the value for X, then it switches hands and reads the value for Y. In the HI! program, starting at line 60, it first reads 10, assigns this value to X, then 2 and assigns this value to Y. Then it reads the second 10, assigns it to X, and finally 3, assigning it to Y. It continues this juggling act until it arrives at the -1 in line 110.

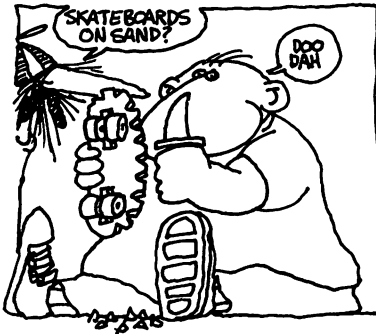
The program reads each set of values, uses these values in the fancy statement in line 40 to print a series of hatched squares on your screen. It takes a set of numbers to position stuff on your screen, but more about the amazing statement in line 40 later.



RESTORE's also in Store for You!

Out here in the desert, one day seems to blend into the next. Junior has come up with a crude method of keeping track of each day of the week so he knows when to

The Great White Expanse



watch his favorite TV shows. At the end of each day's march he notches his skateboard; one notch for Monday, two for Tuesday, and so on. But even my brilliant son gets confused after a while. To help him translate his skateboard notches into days of the week, I have devised this simple but effective program, Notcheroonio. The great thing about Notcheroonio is that it introduces even more exciting DATA tricks!

Here's Notcheroonio. Before I explain all the new stuff that's in it, enter it, RUN it, and start having some fun!

Notcheroonio

```
10 RESTORE
20 INPUT "ENTER A NOTCH BETWEEN 1 AND
  7";N
30 READ A,DAY$
40 IF N = A THEN PRINT DAY$:GOTO 10
50 IF N > 7 OR N < 1 THEN PRINT "OOPS!";
  :GOTO 10
60 GOTO 30
100 DATA 1,MONDAY,2,TUESDAY,3,
  WEDNESDAY,4,THURSDAY,5,FRIDAY
110 DATA 6,SATURDAY,7,SUNDAY
```

I'll bet the first weird thing you noticed about this program was the RESTORE in line 10. RESTORE resets the pointer back to the beginning of a DATA statement.

In line 40, when the number you've entered (N) is the same as the number just read by the pointer (A), the pointer comes to a screeching halt. For example, if you enter the number 3, the pointer reads up to and including "3,WEDNESDAY." Then the computer prints "WEDNESDAY" (DAY\$) on your screen, and the program returns to line 10. In line 10 the RESTORE statement returns the pointer to the beginning of the DATA statement so you can try again.



DR. WACKO PRESENTS COMMODORE BASIC

Some programs you may develop might include sets of DATA on different lines. To reset a specific line of data just place the appropriate line number after RESTORE, like this:

```
10 RESTORE 100
```

In this case, the program will RESTORE only the data beginning on line 100.

Now for the other weird thing about Notcheroonio. Yes, it is possible, as you have just seen, to read string and numeric data at the same time. Just make sure that you alternate numeric and string data, as I've done in line 100. Remember, your program can't read a string into a numeric variable.

Ms. Peeky's Black Book

The other day Ms. Peeky lost her little black book, and I found it. She'd listed all her favorite men in it and included comments on each. What an embarrassing revelation!

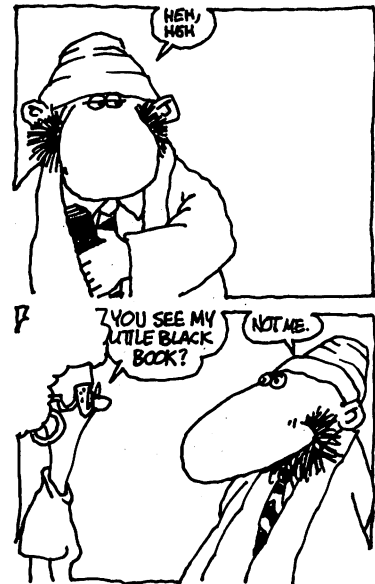
Stand By For a New Concept!

There is only one new concept in Ms. Peeky's Little Black Book, and it occurs in line 40. Before you peek at Ms. Peeky's book, here's line 40, complete with new concept:

```
40 IF MID$(DISC$,1,3) = MID$(NAME$,1,3)  
   THEN PRINT DISC$:GOTO 20
```

Let's check out those two weird—looking string variables: **MID\$(DESC\$,1,3)** and **MID\$(NAME\$,1,3)**.

DESC\$ and **NAME\$** by themselves aren't unique at all. But by placing **MID\$** before each variable, adding a comma and **1,3** after each one, and wrapping the



The Great White Expanse

whole thing in parentheses, they've become Selective Strings!

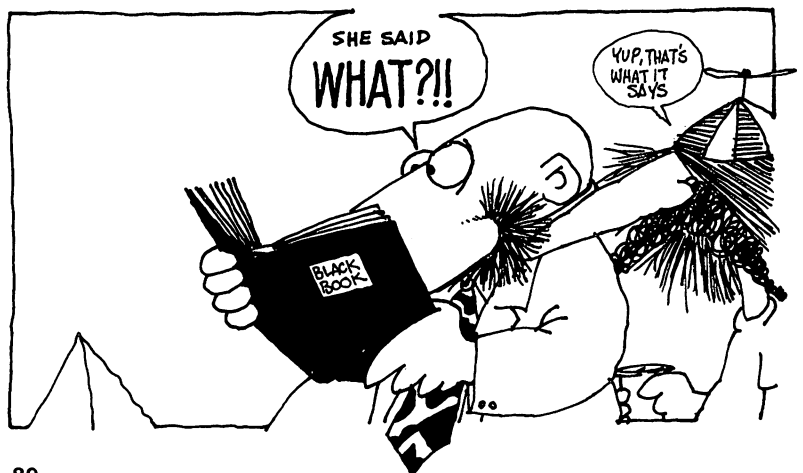
The two numbers inside the parentheses select a portion of the string variable, in this case, the first three letters. In Ms. Peeky's Little Black Book, this neat trick is used to compare *only* the first three letters of each of her boyfriends' names.

Here's a short example that shows how this concept works:

```
10 A$ = "CARAVAN"  
20 PRINT MID$(A$,1,3)
```

After you run this program the word "CAR" will appear on your screen! That's because the program selects *only* the first through the third letters of "CARAVAN." To print the word "VAN," just change line 20 to read: **20 PRINT MID\$(A\$,5,7)** and the fifth through the seventh letters of "CARAVAN" will print on your screen!

Now that we understand those weird string variables, it's time to peek at Ms. Peeky's Little Black Book. Just enter her friend's first names to see what happens. It's absolutely shocking!



DR. WACKO PRESENTS COMMODORE BASIC

Little Black Book

```
10 REM *** MS. PEEKY'S LITTLE BLACK
   BOOK
20 RESTORE :INPUT "ENTER HIS FIRST
   NAME";NAME$
30 READ DISC$
40 IF MID$(DISC$,1,3) = MID$(NAME$,1,3)
   THEN PRINT DISC$:GOTO 20
50 IF DISC$ = "END" THEN GOTO 70:REM
   'END' IS THE FLAG
60 GOTO 30
70 PRINT NAME$; "IS NOT IN MY LITTLE
   BLACK BOOK!":GOTO 20
72 REM ***
75 REM *** HERE COMES THE DATA!
77 REM ***
100 DATA MARVIN MAINSTREAM (A REAL
    MACHO GUY!)
110 DATA WEIRD HAROLD(WEIRD ME OUT.
    REALLY!)
120 DATA SLOW POKE (A REAL SWEETHEART
    XXXX)
130 DATA GRUESOME GEORGE (GOOD IN A
    CLINCH.)
140 DATA SNEAKY PEEK (EEEUK!),CAPTAIN
    ACTION (FAAR OUT!!)
150 DATA WACKO (GAG ME WITH A LADLE!)
160 DATA END
```

Think of Ms. Pecky's Black Book as a rudimentary file cabinet called a *database*. The program stores information in the DATA statements, points to it, READs it, then displays it on your screen. The best part of this program is its ability to selectively search and retrieve information. By modifying it, you can tailor this short program to perform all sorts of data-storage tasks. Here are some ideas:

The Great White Expanse



1. Store and retrieve names, addresses, and phone numbers
2. File those mouth-watering clam dip recipes
3. Store information about your favorite lacrosse team
4. Organize your record collection
5. List all your creditors and the amounts you owe them.
6. Keep track of the tamale sauce and anchovy burritos
7. Store and retrieve the birthdays of all your worst enemies, so you remember not to send them cards.

Use a FOR/NEXT Loop to READ Data

Before I run off and make way for my charming wife, Petunia, I'd like to share one final DATA trick with you.

You can incorporate a FOR/NEXT loop within your data-reading program to let you select the *specific* chunks of data you'd like to read. Here's a final example that illustrates this FOR/NEXT reading trick.

FOR/NEXT DATA Trick

```
10 FOR X = 1 TO 5
20 READ A
30 PRINT A;
40 NEXT X
50 END
60 DATA 1,2,3,4,5,6,7,8,9,10
```

When you RUN this program, you'll see that it READS only the first five numbers and prints them on your screen. Change the FOR/NEXT loop in line 10 to **FOR X = 1 TO 10**, and the program will present *all* the data on this line!

Now that you know all about DATA statements, it's time for me to go and feed Clyde. Anyway, here comes Petunia to tell you all about *arrays*! Chow, bambino, see you later!

Presenting Mrs. Petunia Array Wacko!



That Wacko! Finally the truth can be told. He left in such a hurry because he really doesn't understand arrays. Shocking, isn't it? But he is cute, don't you agree?

Actually, arrays are easier to understand than my husband. You just have to be organized, like me. I did most of the packing for this motley caravan and became an expert at storing stuff in wicker baskets. I then labelled each basket so I'd know what was in it.

Array names are just like variable names, only different. Variable names can identify a basketfull of information. For example, in the expression $X = 100$, "X" is the label for a basket that contains the number 100. Variable baskets, however, have one important limitation. They can only hold one piece of information at a time.

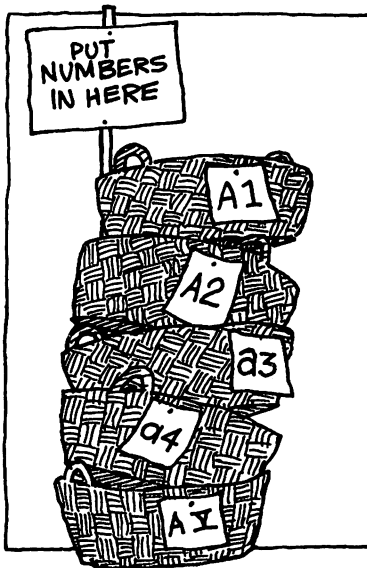
But here's the exciting part! Arrays let you store a whole bunch of numbers in individually marked wicker baskets, *without* having to use a whole bunch of different variable names! How is this possible? Simple, I'll show you how it's done.

The Great White Expanse

DIMension Those Arrays

First you have to know how many numbers you want to store. Then you've got to DIMension the array to *tell your computer to set aside individual baskets for each of those numbers*. For example, if you want to store five numbers, you use a **DIM** statement to dimension your array like this:

DIM A(5)



This means that you are reserving five wicker baskets, labelled A(1), A(2), A(3), A(4), and A(5) to hold each of your five numbers.

You don't have to use the letter A; you can use any variable name as the first part of the label, then put a number within parentheses right after the variable name. After you've labelled all your baskets, you can store numbers in each one and know where each number is stored.

I think it's about time to show you how all this works in a short, but organized example named after my brilliant son, Junior.

Basket Case

```
10 DIM A(5)
15 REM *** 1. SET UP THE ARRAY.
20 FOR X = 1 TO 5
30 A(X) = X:REM *** EACH VALUE OF "X" IS
  ASSIGNED TO THE ARRAY HERE.
40 NEXT X
45 REM *** 2. PUT NUMBERS INTO THE
  ARRAY.
50 FOR X = 1 TO 5
60 PRINT "PUT A NUMBER IN BASKET";X;
70 INPUT N
80 A(X) = N
90 NEXT X
```

DR. WACKO PRESENTS COMMODORE BASIC

Let's just belly-dance our way through this program.

The FOR/NEXT loop in lines 20 through 40 sets up the array. Line 30 labels each of the five empty baskets A(1) through A(5) respectively. The loop in lines 50 through 90 lets you place one number in each wicker basket.

When you run this program, you'll be asked to put a number in each basket. Just so we're on the same light beam, put 10 in basket 1, 20 in basket 2, and any numbers you want in the remaining three baskets. When your screen says, "READY," you can look at the contents of each basket by using a PRINT statement in the *immediate* mode, like this:

```
PRINT A(1) [RETURN]
```

See! A 10 is stored in basket A(1)! Check out basket A(2). I'll bet you find a number 20 in it!

Put Stuff into Arrays from DATA

Basket Case shows how to use an INPUT statement to put stuff into an array. Here's a short program that takes numbers stored after a DATA statement and READs them into an array.

DATA Array

```
10 DIM A(5)  
15 REM *** 1. SET UP THE ARRAY.  
20 FOR X = 1 TO 5  
30 A(X) = X  
40 NEXT X  
45 REM *** 2. READ NUMBERS INTO THE  
ARRAY.
```

The Great White Expanse

```
50 N = 0
60 READ A
70 IF A = -1 THEN END
80 N = N + 1
90 A(N) = A
100 GOTO 60
110 DATA 10,20,30,1,2,-1
```

This program is quite similar to Basket Case. But instead of using an INPUT statement, the program takes numbers stored after the DATA statement on line 110 and READs them into each basket. Check it in the *immediate* mode just like you did with Basket Case.

Stay in Shape!



Junior is a junk-food junky! I've tried and tried to get him to eat the right foods, but he just doesn't listen. Finally, out of sheer exasperation, I put him on a 500-calorie-per-meal diet. To keep track of the number of calories he munches each meal, I developed a short calorie-counting program that shows off a great array application.

Here's a list of some of Junior's favorite foods, along with calories per serving for each.

- Anchovy Burritos: 280 Calories each
- Twinkle Cakes: 340 Calories a look
- Guacamole Juice: 90 Calories per slurp
- Clam Dip: 70 Calories a dip
- Greaso Burgers: 470 Calories per bun
- Quicko TV Dinner: 400 Calories a tray
- Pizza a la Hollandaise Sauce: 900 Calories a sniff



Now you can see how serious the situation is! If he eats just one Anchovy Burrito, looks at a single Twinkle Cake, slurps some Guacamole Juice, and sniffs a Pizza a la Hollandaise Sauce, he's in big trouble!

DR. WACKO PRESENTS COMMODORE BASIC

Just to show you how effective arrays are in curbing excessive appetites, enter Calorie Counter and RUN it. It is a long listing. Take your time and type it in carefully. It'll be well worth the effort, and you'll probably burn off a few calories in the process!

Calorie Counter

```
10 DIM C(21):T = 0
15 REM *** 1) EMPTY THE BASKETS.
20 FOR X = 1 TO 21:C(X) = 0:NEXT X
25 REM *** 2) CLEAR THE SCREEN.
30 PRINT "[CLRHM]"
40 PRINT "HOW MANY JUNKY MEALS DID
    YOU DEVOUR"
45 REM *** 3) INPUT NUMBER OF MEALS.
50 INPUT N
60 PRINT:PRINT "ENTER
    CALORIES/MEAL—":PRINT
70 FOR X = 1 TO N:PRINT "MEAL #";X;": ";
75 REM *** 4) PUT CALORIES IN EACH
    BASKET.
80 INPUT M
90 C(X) = M
95 REM *** 5) KEEP TRACK OF TOTAL
    CALORIES.
100 T = T + C(X)
110 NEXT X
120 PRINT
130 PRINT T;" CALORIES IN ";N;" MEALS."
135 REM *** 6) 'A%' EQUALS AVERAGE
    CALORIES PER MEAL.
140 A% = T/N
150 PRINT A%;" CALORIES PER MEAL."
160 IF A% > 500 THEN PRINT "GRODY!
    GREASE ME OUT!":GOTO 180
170 PRINT "JUNIOR, YOU'RE DOING GREAT!"
180 PRINT:PRINT "PRESS F7 TO EAT AGAIN"
185 REM *** 7) FANCY PRESS 'F7' ROUTINE.
```

The Great White Expanse

```
190 GET A$:IF A$ = "" THEN GOTO 190
200 IF ASC(A$) < > 136 THEN GOTO 190
205 REM *** CLEAR OUT THE ARRAY AND
    START AGAIN
210 CLR:GOTO 10
```

Here's what my screen looked like after I entered Junior's first day (three meals) of dieting.

HOW MANY JUNKY MEALS DID YOU DEVOUR
?3

ENTER CALORIES/MEAL

MEAL#1:?620

MEAL#2:?470

MEAL#3:?490

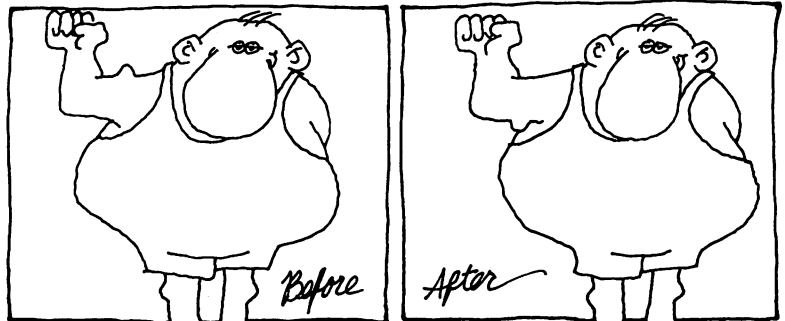
1580 CALORIES IN 3 MEALS

526 CALORIES PER MEAL.

GRODY! GREASE ME OUT!

PRESS F7 TO EAT AGAIN

Junior went wild and overdid it a bit. When I wasn't looking, he must have scarfed down a bunch of anchovy burritos topped off with a generous helping of Twinkle Cakes! But things have improved. Just look at these before-and-after pictures.



DR. WACKO PRESENTS COMMODORE BASIC

The results of this calorie-counting program were made possible, in large part, by the amazing array. Although this program is long, it's really easy to understand. To help you, I put REM statements, numbered 1 through 7, in the program. Here's a broader (no pun intended) explanation of Calorie Counter.

First, empty the baskets. It's always good practice to clear out each basket before you start stuffing anything into it. Line 20 places zeros in each array location. I DIMensioned twenty-one locations to hold a maximum of three-week's gorging. That's about the limit of Junior's attention span. On any other subject it's three minutes.

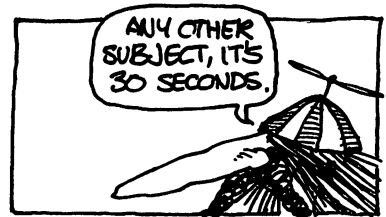
Second, clear the screen. Line 30 does just that. Remember, to make your program do something that's usually done at the keyboard, use a PRINT statement and surround the correct keystrokes with quotation marks.

Third, input the number of meals Junior devoured. In line 50 the number of meals, N, is INPUTed. The program also uses this number to label each basket with the FOR/NEXT loop that begins in line 70.

Fourth, put calories in each basket. All the action takes place in line 90.

In the FOR/NEXT loop and INPUT routine beginning on line 70 and ending on line 110, you enter the calories per meal into the array.

Here's a close look at these important program lines. The FOR/NEXT loop begins on line 70, and is set to the number of meals (N) you've provided in line 50. Each meal number (X) is printed on the screen by the second statement in line 70.



The Great White Expanse

The INPUT statement in line 80 lets you enter total calories for each meal. Then, in line 90, the calories are put into array C(X).

Line 100 keeps a running total of calories. Finally, in line 110, the loop goes back to line 70 to continue the process. When the program has finished looping around, it moves on to line 120. Wheew!

Fifth, keep track of the total calories. Line 100 is a simple counting routine, just like the one Snidely used to count his shekels on page 62.

Sixth, A% equals the average calories per meal. In line 140, A% equals the total calories divided by the total number of meals. The answer is *rounded down* because I've added a percent symbol (%) to the variable 'A'. Clever, no?

Seventh, try a fancy press 'f7' key routine. The GET statement on line 190 GETs your keyboard input and assigns it to A\$. Next, IF A\$ is equal to nothing (two quotations marks right next to each other) THEN the program goes back to the beginning of line 190 and waits for you to press a key.

In line 200, IF the ASCII value of the key you pressed (A\$) is *not equal* to 136, the program loops back to line 190 again, and waits for you to press the **f 7** key. (136 is the ASCII value of the **f 7** key. Check out the Hotski-Toski Charts on page 245.)

When you press **f 7** the program drops down to line 210.

Hang in there! Don't get hung up by new terms like "ASCII value" and "ASC". It will all become brilliantly clear when we visit Oasis 4 and I *string* you along.

Finally, clear the keyboard and start again. The CLR in line 210 is used to clear out your computer's memory so you can start again.

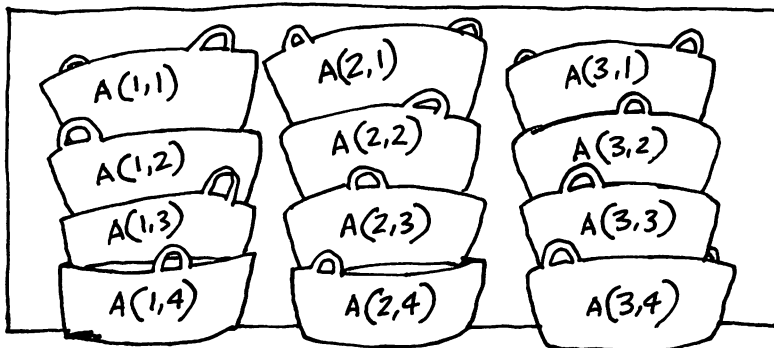
DR. WACKO PRESENTS COMMODORE BASIC

Now that Junior's eating habits are under control, play with the Calorie Counter program. Modify it to your heart's content. When you really understand how it works, you'll be ready to move on to my next specialty.

The Marvelous Matrix

That calorie-counting program was great, but it can only be used to keep track of one hedonist at a time. To keep tabs on all the Wackos, I had to resort to a fancier storage system, called a *matrix*.

A matrix is just a fancy array that lets you organize stuff in columns and rows. Each basket in each column and row is individually labelled and looks just like this weird drawing.



The great thing about a matrix is that it lets you store information in a really organized fashion. For example, you could store the calories gorged by Junior in the baskets in column 1, Dr. Wacko's prodigious intake in column 2, and Ms. Peeky's petite pickings in column 3.

Setting Up A Matrix

Setting up a matrix is almost like setting up a simple array. First you DIMension the matrix like this:

```
DIM A(3,4)
```


The Great White Expense

This means that you are reserving three columns, and that each column consists of a stack of four baskets. If you want to set up a matrix with ten columns having five baskets in each column, just DIMension the matrix like this:

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

After you DIMension the matrix it's always a good idea to empty all the baskets. Here's how its done:

Empty the Baskets

```
10 DIM A(3,4)
20 FOR X = 1 TO 3
30 FOR Y = 1 TO 4
40 A(X,Y) = 0
50 NEXT Y
60 NEXT X
```



This short routine places zeros in each basket. To check this out, just **PRINT A(1,1)** through **A(3,4)** in the *immediate* mode. You'll see "0" printed on your screen each time you press **RETURN**.

After you've emptied each basket, it's time to start filling each one with information. The best ways to do this are to use an **INPUT** statement, or to use **DATA** and **READ** statements to read numbers into each basket. You can also put stuff into a matrix the hard way by filling up each basket one-by-one.

This short program shows how easy it is to put data into a matrix. It sets up a matrix with two columns and two rows.

DR. WACKO PRESENTS COMMODORE BASIC

Matrix Stuffer

```
10 DIM A(2,2)
20 FOR X = 1 TO 2:FOR Y = 1 TO
    2:A(X,Y) = 0:NEXT Y:NEXT X
30 PRINT "PUT A NUMBER INTO EACH BASKET"
40 FOR X = 1 TO 2:FOR Y = 1 TO 2
50 PRINT X;" ";Y;
60 INPUT NUMBER
70 A(X,Y) = NUMBER
80 NEXT Y:NEXT X
```

All the stuffin' takes place in the FOR/NEXT loop in lines 40 through 80. Here's what my screen looked like when I put numbers into each basket:

```
PUT A NUMBER INTO EACH BASKET
1,1 ?20
1,2 ?40
2,1 ?60
2,2 ?80
```

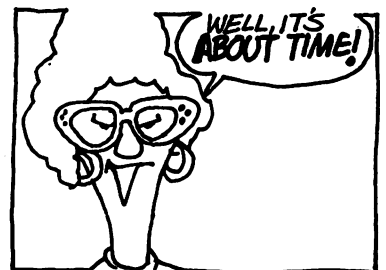
Run the program and put numbers into each location. Then, when your screen says "READY," look into each basket in the *immediate* mode like this:

```
PRINT A(1,1) [RETURN]
```

If you put 20 in basket (1,1), 20 will be printed on your screen!

Now that you know how to DIMension a matrix and put numbers into it, I'll show you how to take information out of the matrix and do something useful with it.

The next program, Wacko Quotient, really puts matrices to the test. In fact, I developed it to test up to five wackos and average each of their scores.



The Great White Expanse

I asked each member of this weird group to rate themselves on a scale of 1 to 10 by filling out this simple questionnaire.



SIMPLE QUESTIONNAIRE FOR SIMPLE WACKOS

RATE YOUR WACKO QUOTIENT ON A SCALE OF 1 TO 10. ENTER YOUR RATING IN THE SPACE PROVIDED

1. OUT TO LUNCHNESS _____
2. BRAIN POWER _____
3. SLUGGISHNESS _____
4. SEX APPEAL _____
5. TAP DANCING ABILITY _____

After I collected all this meaningful information, I entered it into my Wacko Quotient program. Here are Captain Action's and Junior's results. Captain Action is "WACKO 1," Junior is "WACKO 2."

HOW MANY WACKOS?2

REMEMBER, THERE ARE FIVE QUESTIONS!

WACKO 1: ANSWER TO QUESTION 1?10

WACKO 1: ANSWER TO QUESTION 2?5

WACKO 1: ANSWER TO QUESTION 3?8

WACKO 1: ANSWER TO QUESTION 4?10

WACKO 1: ANSWER TO QUESTION 5?10

WACKO 2: ANSWER TO QUESTION 1?10

WACKO 2: ANSWER TO QUESTION 2?1

WACKO 2: ANSWER TO QUESTION 3?10

WACKO 2: ANSWER TO QUESTION 4?1

WACKO 2: ANSWER TO QUESTION 5?1

WACKO 1's TOTAL IS:43

AVERAGE 8.6

WACKO 2's TOTAL IS:23

AVERAGE 4.6

DR. WACKO PRESENTS COMMODORE BASIC

Here's the great program that made it all possible!

Wacko Quotient

```
10 DIM A(5,5)
20 REM *** CLEAN UP YOUR ACT...
30 FOR X = 1 TO 5:FOR Y = 1 TO
   5:A(X,Y) = 0:NEXT Y:NEXT X
40 PRINT:PRINT "HOW MANY WACKOS";
50 INPUT W
60 REM *** ONLY ACCEPT UP TO 5 WACKOS
70 IF W > 5 OR W < 1 THEN GOTO 40
80 PRINT "REMEMBER, THERE ARE 5
   QUESTIONS!"
90 Q = 5
92 REM ***
94 REM ***
96 REM ***
100 REM *** PUTIN' ANSWERS INTO THE
   ARRAY
110 PRINT
120 FOR X = 1 TO W:FOR Y = 1 TO Q
130 PRINT "WACKO";X;":ANSWER TO
   QUESTION";Y;
140 INPUT ANSWER
150 REM *** ONLY ACCEPT ANSWERS FROM
   1 TO 10
160 IF ANSWER < 1 OR ANSWER > 10 THEN
   PRINT "OOPS!":GOTO 130
170 A(X,Y) = ANSWER
180 IF Y = Q THEN PRINT
190 NEXT Y:NEXT X:GOTO 260
192 REM ***
194 REM ***
196 REM ***
200 REM *** TAKIN' STUFF OUT OF THE
   ARRAY
210 PRINT
220 FOR X = 1 TO W:FOR Y = 1 TO Q
```

The Great White Expanse

```
230 PRINT "A(";X;",";Y;") = ";A(X,Y)
240 IF Y = Q THEN PRINT
250 NEXT Y:NEXT X
260 T = 0
270 FOR X = 1 TO W:FOR Y = 1 TO Q
280 T = T + A(X,Y)
285 REM *** LINE 290 IS SQUASHED TO FIT
    WITHIN THE 2 LINE LIMIT
290 IF Y = Q THEN PRINT "WACKO";X;"S TOTAL
    IS:";T:A = T/Q:PRINT
    "AVERAGE";A:PRINT:T = 0
300 NEXT Y:NEXT X
```



The first section, lines 10 through 90, clears out the array by putting zeros into all the baskets, then lets you enter up to five wackos.

The second section, lines 100 through 190, lets you fill the array with information from the questionnaire. It also limits the range of numbers you can enter to those between 1 and 10.

The final section, lines 200 through 290, takes information out of the array, totals it, averages it, and prints it out on your screen.

As you can see, a matrix is ideal when it comes to rating a bunch of wackos, but it's also perfect for anything that can be charted in columns and rows.

The best way to set up a matrix is to first get out the ol' pencil and paper, draw a chart containing columns and rows, then write labels across the top and down the side. Once you've finished, simply transfer your masterpiece to your matrix program.

DR. WACKO PRESENTS COMMODORE BASIC

Here's the chart I drew before I programmed Wacko Quotient:

| | WACKO 1 | WACKO 2 | WACKO 3 | WACKO 4 | WACKO 5 |
|----------|---------|---------|---------|---------|---------|
| ANSWER 1 | | | | | |
| ANSWER 2 | | | | | |
| ANSWER 3 | | | | | |
| ANSWER 4 | | | | | |
| ANSWER 5 | | | | | |
| TOTALS | | | | | |
| AVERAGE | | | | | |

An Egocentric String Array

Before I run off and drag my darling hubby back into the book, I'd like to show you a final bit of array madness. I call it "madness" because it shows just how egocentric Dr. Wacko can be, and it really gets me mad! (Sometimes)

This next invaluable gem shows how versatile your Commodore 64 is, and how easy it is to stick words and other nonsense into an array.

Enter Egocentric Wacko, RUN it, follow my inane instructions, and you'll see why the Doctor is sometimes a little bit difficult to live with. (Much less get stuck with out here in the desert!)

Egocentric Wacko

```
10 DIM A$(10)
20 FOR X = 1 TO 10:PRINT X;
30 INPUT A$(X)
40 NEXT X
50 FOR X = 1 TO 10:PRINT A$(X);" ";:NEXT X
```

The Great White Expanse

A series of numbers, from 1 through 10, each followed by a question mark, appear on your screen. Type a word or phrase after each question mark and press **RETURN** . Here's how my husband answered the questions:

- 1 ?DR. WACKO
- 2 ?IS
- 3 ?A GREAT
- 4 ?FELLER
- 5 ?AND
- 6 ?IF YOU DON'T
- 7 ?BELIEVE
- 8 ?ME
- 9 ?JUST ASK.
- 10 ?BURMA SHAVE

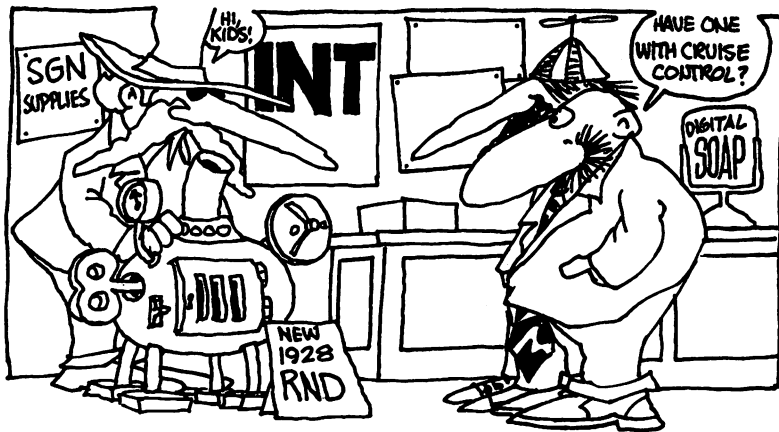
If you enter the same words Wacko did, you'll soon understand why I get so upset with him, sometimes. You'll also see how easy it is to put strings into an array! The methods that worked so well when we explored numeric arrays work just fine when it comes to string arrays!

Well, that's it for now. Anyway, here comes my lovable, cute, and egocentric husband to invite you to a fantastic function at the oasis. Bye!



DR. WACKO PRESENTS COMMODORE BASIC

Oasis 3: Three Fantastic Functions



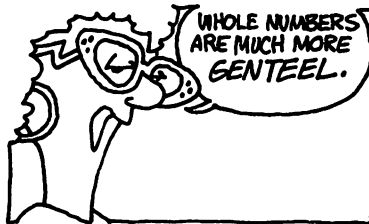
During your short stay at this Oasis you'll meet three fantastic *functions*: INT, RND, and SGN, that are guaranteed to add pizzazz (or pizza) to your programs.

So, join in on the fun, and I'll introduce you to the first important *function*.

The Round-Down INT

INT, which is an abbreviation for the word "integer," rounds off answers to math problems. By using INT you can round fractional answers *down* to integers or whole numbers. You use the INT function when you *haven't* followed a variable with a percent symbol (%).

The best way to see what INT does is to put it to work and watch it in action. Try these simple math examples in the *immediate* mode, and you'll discover how easy INT is to use.



First, here's a division example.

```
PRINT 55/20 [RETURN]
```


The Great White Expanse

The answer is



2.75

Now suppose you don't want to fool around with parts of camels, but you do want a whole number as your answer. Not to worry! To round *down* the answer to the nearest whole number, just use INT.

PRINT INT(55/20) [RETURN]

The answer, amazingly enough, is 2 because of the fantastic INT function. In plain English, the statement **PRINT INT(55/20)** means divide 55 by 20, round down the answer to the nearest whole number, and print it.

Here's a multiplication example.

PRINT 2.75*18 [RETURN]

The answer is

49.5

To round this *down* to the nearest whole number use INT, like this.

PRINT INT(2.75*18) [RETURN]

Now, the answer is

49

Amazing!

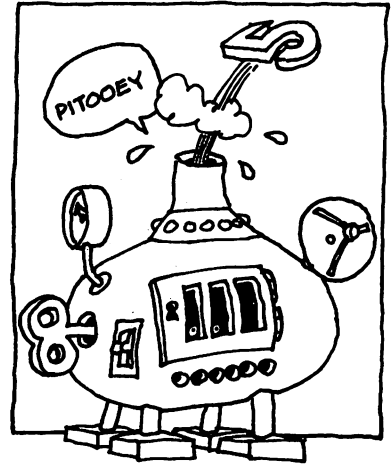
To use INT, just type in INT, a beginning parenthesis, the math, and a closing parenthesis. Then press RETURN.

DR. WACKO PRESENTS COMMODORE BASIC

Always keep in mind that INT rounds *down* the answer. So, if the math inside the parentheses equals 2.9999, INT will round the answer *down* to 2.

The Random RND

Your Commodore 64 computer has a built-in random-number generator. Allah be praised! By using RND and rubbing gently, you can make your computer spit out *random* numbers that you can use in all sorts of inventive programs. For example, you can use random numbers in games of chance whenever you want to roll dice, select cards, or simulate the spin of a roulette wheel.



Random numbers also liven up adventure games by varying the computer's response to a player's choice. Every time the player "opens" a door or decides to "walk" down a corridor, he or she is guaranteed a surprise from a new and randomly selected experience. Just like in real life. Ms. Peeky once said, "It's the romantic random!"

The rollicking RND makes quiz games more rambunctious by offering your players a variety of randomly selected questions to answer.

Although programs that use the RND function are unpredictable and exciting, generating random numbers is predictable and easy.

Here's a short program that generates whole numbers between 1 and 10. Enter it, run it, and, when you've seen how it works, I'll explain all. Trust me!

Random-Number Generator

```
10 N = INT(RND(0)* 10) + 1
20 PRINT N
30 GOTO 10
```

The Great White Expense

It works pretty well, doesn't it? After you ran this program, a string of random numbers flowed down your screen. And it was all made possible by the random RND in line 10.

Here's a close-up view of the entire line so you can see how it all fits together.

```
10 N = INT(RND(0)*10) + 1
```

I've placed an INT function before RND to generate whole numbers. Remove the INT, run the program again, and you'll get the picture. Or indigestion.



First, Start Spitting

The + 1 at the end of the line tells RND where to start. This program spits out numbers (just like Clyde) starting with 1.

How Many Numbers?

The *10 tells the random generator how many numbers to spit out. Here we get ten numbers. Replace the 10 with a 5, and the program spits out five numbers between 1 and 5.

Will The Real Random Generator Please Stand Up!



DR. WACKO PRESENTS COMMODORE BASIC

RND(0) is the generator. A zero in parentheses always follows **RND**. Here are a two more short examples that show the random generator in action.

RND Example 1

```
0 REM THIS GENERATOR STARTS AT 5 AND  
  SPITS OUT SIX NUMBERS: 5,6,7,8,9,  
  and 10.  
10 N = INT(RND(0)*6) + 5  
20 PRINT N  
30 GOTO 10
```

RND Example 2

```
0 REM THIS GENERATOR STARTS AT 1 AND  
  SPITS OUT FIVE NUMBERS: 1,2,3,4, and 5.  
10 N = INT(RND(0)*5) + 1  
20 PRINT N  
30 GOTO 10
```

I now present The Standard Guess My Number Program. This unoriginal program is almost identical to the dull programs found in every other BASIC programming book! Almost, but not quite. This program not only illustrates a cute use of the random **RND**, but it has a few wacky touches of my own to spice it up. So here it is!

The Standard

```
10 REM *** THE STANDARD 'GUESS MY  
  NUMBER' PROGRAM  
20 PRINT:T = 0:N = INT(RND(0)* 100) + 1  
30 PRINT "I'M THINKING OF A NUMBER  
  BETWEEN 1 AND"  
40 PRINT "100. SO, WHAT'S MY  
  NUMBER...HUH";  
50 INPUT GUESS  
55 T = T + 1
```

The Great White Expense

```
60 IF GUESS = N THEN PRINT "WOW, GEE,  
COWABONGA! AND , IN";T;"TRIES!":  
GOTO 20  
70 IF GUESS < N THEN PRINT "YOUR GUESS IS  
TOO LOW. TRY AGAIN.":GOTO 50  
80 IF GUESS > N THEN PRINT "YOUR GUESS IS  
TOO HIGH. TRY AGAIN.":GOTO 50
```

When you look at it, this standard program is pretty standard. It generates a number between 1 and 100 in line 10, uses a counter ($T = T + 1$) to keep track of the total number of guesses, and has a bunch of IF/THEN statements that check for a correct guess. No problem!

A Random-Problem Generator



If you want to generate a lot of problems, a random-problem generator is just the ticket!

To help Junior improve his math skills, I developed this short Multiplication Practice program. It uses two generators to create baffling random problems.

Two numbers for each problem are generated in lines 30 and 40. Just change the number 10 to 1000 and really make them tough!

Multiplication Practice

```
10 REM *** JUNIOR'S MULTIPLICATION  
PRACTICE  
20 C = 0:W = 0  
25 REM *** BEWARE! PROBLEMS  
GENERATED BELOW:  
30 A = INT(RND(0)* 10) + 1  
40 B = INT(RND(0)* 10) + 1  
45 .  
50 PRINT "WHAT'S";A;"TIMES";B;  
60 INPUT ANSWER
```

DR. WACKO PRESENTS COMMODORE BASIC

```
70 IF C*10 = 100 THEN PRINT :PRINT
   "***WOW! 100%. MY SON THE
   GENIUS!***":GOTO 20
80 IF C + W = 10 THEN PRINT "YOU GOT"
   ;C;"RIGHT AND MISSED";W;;GOSUB
   110:GOTO 20
90 IF ANSWER = A*B THEN PRINT "RIGHT ON
   JUNIOR!":PRINT:C = C + 1:GOTO 30
100 IF ANSWER < > A*B THEN PRINT "OOOPS!
   HERE IT IS AGAIN!":W = W + 1:GOTO 50
110 PRINT ":";C*10;"%":PRINT:RETURN
```

The principles used in this multiplication program can easily be expanded into one gigantic program that generates a random addition, subtraction, multiplication, and division quiz. No kidding!

Enter and SGN in, Mystery Guest

Presenting last, but not least, a little-known and less-understood friend of Lawrence of Newark, the mysterious SGN.

Don't panic! SGN has nothing whatsoever to do with trigonometry. It is not sine/cosine stuff.

When you use SGN, as I have below, it sets the variable *A* equal to -1 , 0 , or 1 to correspond with the value inside the parentheses (either negative, zero, or positive). Here's what it looks like.

A = SGN(N)

And now, presenting a very short program that demystifies SGN, but tells you nothing at all about Lawrence of Newark.

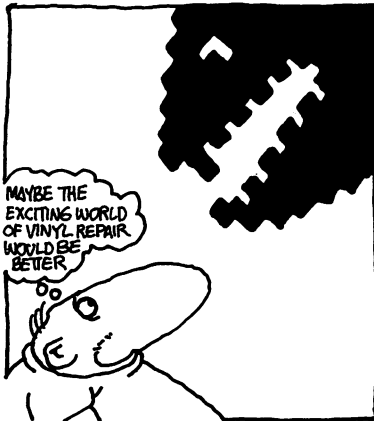


The Great White Expanse

SGN In

```
10 INPUT "ENTER A NUMBER";N
15 REM ***
20 A = SGN(N)
25 REM ***
30 IF A = -1 THEN PRINT "IT'S A NEGATIVE
    NUMBER!":GOTO 10
40 IF A = 1 THEN PRINT "IT'S A POSITIVE
    NUMBER!":GOTO 10
50 IF A = 0 THEN PRINT "IT'S JUNIOR'S
    GRADE POINT AVERAGE.":GOTO 10
```

Don't run away! Just run this program and enter some numbers. You might try a positive number like 100 first, then enter a negative number like -5 , and finally a 0.



SGN has some pretty esoteric uses in BASIC programming. It can control the movement of screen images in BASIC arcade games, and lots of other stuff. (Check out my other book, *Dr. C. Wacko's Miracle Guide To Designing And Programming Your own Commodore 64 Computer Arcade Games!*) I just thought I'd show you SGN now so you can add it to your bag of programming tricks. By the way, who is Lawrence of Newark, and where is he?

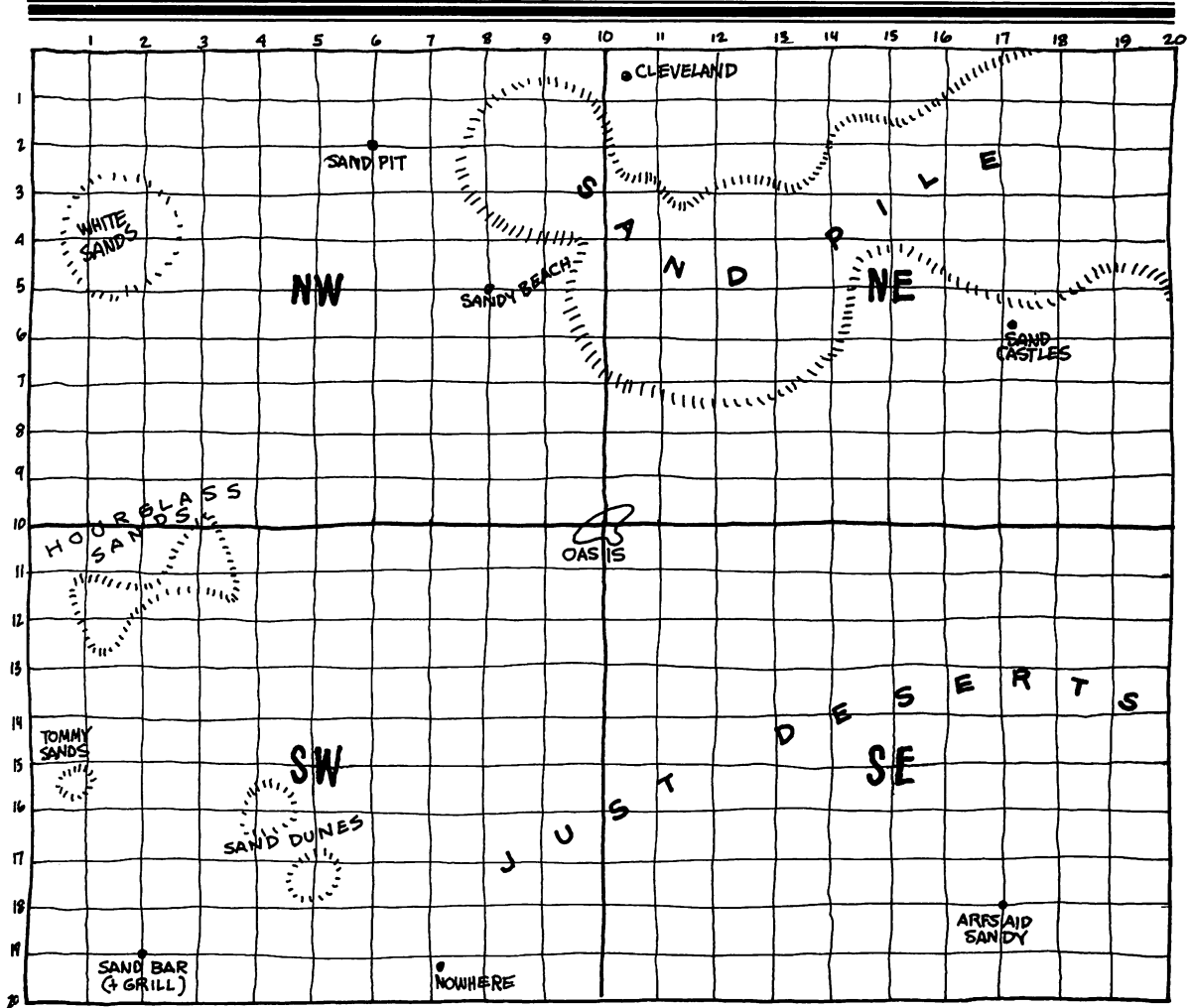
Lawrence of Newark Is Lost in the Desert!



Lawrence is lost in the desert, and you can help him find his way back to the oasis! So, get set to hone up your navigation skills by playing a few games of Rescue Lawrence.

I've placed this bonus game here, at the end of this oasis, because it uses all the programming elements you've learned so far. It really puts the RND function to good use. And besides, it's time to have some fun!

DR. WACKO PRESENTS COMMODORE BASIC



Here's a map of the desert. The oasis is in the center, and you and Lawrence are lost somewhere out in the sand dunes. You've got ten days (turns) to find the oasis. You can travel up to five miles each day in any direction(s) you choose. To help you, your compass displays the direction you are in relation to the oasis. What makes the game challenging is that you don't know how far you are from the oasis. Uh Oh!

Here's the program listing. Type it in, RUN it, then try not to get lost! After you've enjoyed the game I'll review some of this program's highlights and lowlights.

The Great White Expanse

Rescue Lawrence

```
10 T = 1
15 REM *** GENERATE YOUR INITIAL
    LOCATION
20 X = INT(RND(0)*20) + 1
30 Y = INT(RND(0)*20) + 1
35 REM ***
37 REM *** MAKE SURE THAT YOU DON'T
    START AT THE OASIS
40 IF X = 10 AND Y = 10 THEN GOTO 20
45 REM ***
50 PRINT:PRINT "DAY";T;"YOU ARE ";
60 IF T = 11 THEN PRINT"OOOPS! OUT OF
    WATER!":GOTO 10
65 REM *** YOU'VE ARRIVED AT THE OASIS!
70 IF X = 10 AND Y = 10 THEN PRINT
    "WELCOME TO THE OASIS!":GOTO 10
75 REM *** PRINT YOUR POSITION RELATIVE
    TO THE OASIS
80 IF Y > 10 THEN PRINT "SOUTH ";
90 IF Y < 10 THEN PRINT "NORTH ";
100 IF X < 10 THEN PRINT "WEST ";
110 IF X > 10 THEN PRINT "EAST ";
115 REM ***
120 PRINT "OF THE OASIS!"
130 IF T = 10 THEN PRINT:PRINT"*** YOU'RE
    OUT OF WATER! TRY AGAIN***":GOTO 10
140 PRINT:PRINT "HOW MANY MILES NORTH
    TODAY";
150 INPUT N:GOSUB 300
160 IF N > 0 THEN GOTO 190
170 PRINT "HOW MANY MILES SOUTH
    TODAY";
180 INPUT S:GOSUB 300
190 PRINT "HOW MANY MILES EAST TODAY";
200 INPUT E:GOSUB 300
210 IF E > 0 THEN GOTO 240
220 PRINT "HOW MANY MILES WEST TODAY";
```

DR. WACKO PRESENTS COMMODORE BASIC

```
230 INPUT W:GOSUB 300
240 T=T+1
245 REM *** UPDATE YOUR POSITION RELATIVE
    TO THE OASIS
250 IF X > 10 THEN X=X-W
260 IF X < 10 THEN X=X+E
270 IF Y > 10 THEN Y=Y-N
280 IF Y < 10 THEN Y=Y+S
290 GOTO 50
295 REM ***
297 REM *** THIS SUBROUTINE CHECKS FOR
    AN INPUT GREATER THAN 5
300 IF N > 5 OR S > 5 OR E > 5 OR W > 5
    THEN PRINT "YOU CAN'T TRAVEL MORE
    THAN 5 MILES":GOTO 140
310 RETURN
```

Here's what my screen looked like after I played a couple of rounds of Rescue Lawrence:

DAY 1 :YOU ARE SOUTH EAST OF THE OASIS!

HOW MANY MILES NORTH TODAY?5
HOW MANY MILES EAST TODAY?0
HOW MANY MILES WEST TODAY?3

DAY 2 :YOU ARE EAST OF THE OASIS!

HOW MANY MILES NORTH TODAY?0
HOW MANY MILES SOUTH TODAY?0
HOW MANY MILES EAST TODAY?0
HOW MANY MILES WEST TODAY?3

DAY 3 :YOU ARE WELCOME TO THE OASIS!

I made it in two moves. Wheew! Here's how this fabulous game works.



The Great White Expanse

The program randomly generates your initial location in lines 20 and 30. The map is a 20-column by 20-row grid, and the random generators define your position in terms of a row and a column. A position of 3,4, for example, starts you off in the upper left, or Northwest quadrant of the map. (You're 3 across and 4 down, on the left side of the map.)

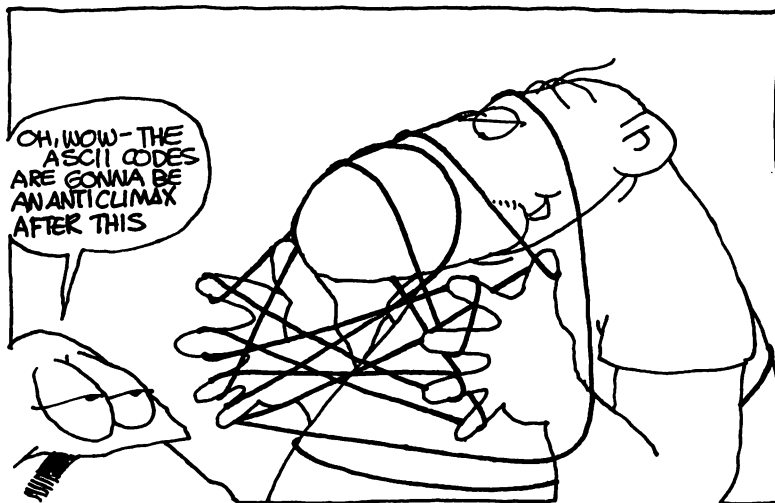
The IF/THEN statement in line 40 makes sure that you don't start off *at* the oasis (10 across and 10 down).

Lines 80 through 90 print your position relative to the oasis. Let's say you start at 3 across and 4 down, ($X = 3$ and $Y = 4$). Because Y is less than 10, the program displays "NORTH," and because X is also less than 10 the word "WEST" is displayed. The combined display reads like this,

Day 1:YOU ARE NORTH WEST OF THE OASIS!

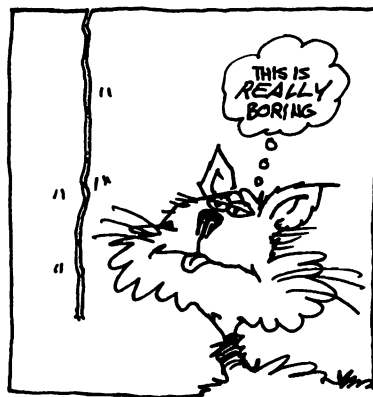
In lines 250 through 280 the program adds or subtracts your mileage inputs to or from your current position to update your position relative to the oasis. New, updated values of X and Y are then sent back up to line 50 to determine whether you begin another turn, run out of water, or arrive at the oasis.

Oasis 4: Don't String Me Along Including Strings Revisited plus ASCII Codes and the Hotski- Totski Chart



“What’s so advanced about string?” That’s a question I hear daily from our desert wandering cats; Keys, Joystick, and Paddles. To them, string is just string. But to wackos like me, advanced string-handling is one of the niftiest treats Commodore 64 BASIC has to offer.

During your visit at this oasis, I’ll show you some very special string-handling tricks that you can use immediately to make your programs sizzle with excitement!



Presenting Three Talented String Manipulators — MID\$, LEFT\$ and RIGHT\$!



When we stopped at Oasis 2, I introduced you to **MID\$**, a very selective string function. **MID\$** helped Ms. Peeky choose an escort from her Little Black Book. Remember? Well if you don't, or if you skipped over my explanation, here it is again.

The Selective MID\$

By entering the **MID\$** function, and following it with a string variable and two numbers in parentheses like this, **MID\$(A\$,5,4)**, you can select a portion of the string.

The following short program shows this concept in action.

Carefree

```
10 A$ = "CAREFREE"  
20 PRINT MID$(A$,1,3)  
30 PRINT MID$(A$,1,4)  
40 PRINT MID$(A$,5,4)  
50 PRINT MID$(A$,2,1)
```

After you run Carefree your screen will look like this.

```
CAR  
CARE  
FREE  
A
```

In line 20, the statement **MID\$(A\$,1,3)** selects three characters, beginning with the first character; line 30 prints the first through the fourth characters; line 40

DR. WACKO PRESENTS COMMODORE BASIC

selects and prints four characters, beginning with the fifth character of the string; and finally, line 50 singles out just the second character for printing.

Knowing Your RIGHT\$ From Your LEFT\$

In actual practice you will normally use **MID\$** to pull out chunks of text from the *middle* of a string. Its two companions, **RIGHT\$** and **LEFT\$**, can help you select portions of a string, starting at the far *right*, or far *left* of the string.

Here's a modified version of Carefree that puts **RIGHT\$**, **LEFT\$**, and **MID\$** through their paces.

Modified Carefree

```
10 A$ = "CAREFREE"  
20 PRINT LEFT$(A$,3)  
30 PRINT LEFT$(A$,4)  
40 PRINT RIGHT$(A$,4)  
50 PRINT MID$(A$,2,1)
```

When you run Modified Carefree, the words CAR, CARE, FREE, and A appear on your screen. Although slightly different (and a little weird), this amazing program delivers the same results as the original Carefree program!

In line 20, **LEFT\$(A\$,3)** selects the *three leftmost* characters, **CAR**.

In line 30, **LEFT\$(A\$,4)** selects the *four left most* characters, **CARE**.

RIGHT\$(A\$,4), in line 40, selects the *four rightmost* characters, **FREE**.

Finally, in line 50, my old friend **MID\$** isolates the second character, **A**.

The Great White Expanse

Summarizing MID\$, RIGHT\$, and LEFT\$ Statements

MID\$(A\$,B,X) starts at the “Bth” character of string **A\$** and selects **X** number of characters, (including spaces and symbols).

RIGHT\$(A\$,X) selects **X** number of characters from the rightmost portion of **A\$**.

LEFT\$(A\$,X) selects **X** number of characters from the leftmost portion of **A\$**.

Now that that’s all taken care of, here’s Captain Action’s Be Cool program. It uses **RIGHT\$**, to “look” at the last three characters of a string. If the last three characters are **MAN**, then the phrase **ALL RIGHT!** prints on your screen. Wow!

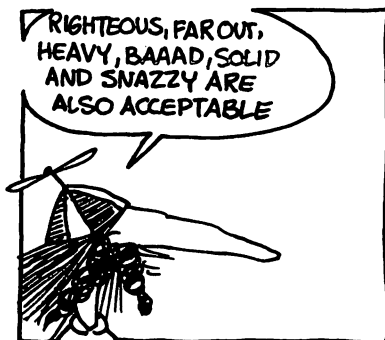
Be Cool



```
10 INPUT A$  
20 IF RIGHT$(A$,3) = "MAN" THEN PRINT "ALL  
RIGHT!":GOTO 10  
30 PRINT "NOT SO COOL.":GOTO 10
```

Here are two runs of Be Cool:

```
?YOU'RE A REAL COOL DUDE MAN  
ALL RIGHT!
```



```
?YOU'RE A REAL GARBANZO BEAN  
NOT SO COOL.
```

Captain Action really likes to end his sentences with “man,” so he went to a lot of effort to produce his Be Cool program. The statement in line 20 did all the work for him, and it’s real easy to understand. Here’s the important part.

DR. WACKO PRESENTS COMMODORE BASIC

IF RIGHT\$(A\$,3) = "MAN" THEN

If this line of BASIC programming were written in English it might read, "If the three *rightmost* letters are *MAN*, then...do something!"

The program begins counting back on the last character, the **N**. From there it goes back two places to arrive at the first letter of the word **MAN**. If you change the **3** to a **4**, the program will isolate the last four characters, **MAN** plus the space that precedes it. If you change the **3** to **2**, only **AN** is isolated.

The best way to understand Captain Action's masterpiece is to mess around with his program. Just change the **3** to another number, and change **MAN** to a word of your own choosing, and you're on your way to becoming a **RIGHT\$** expert! **RIGHT\$** on!

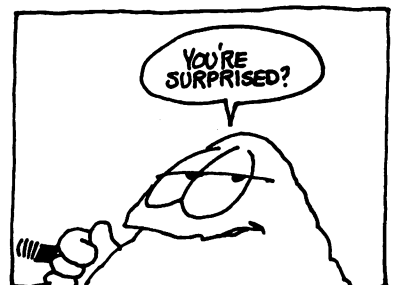
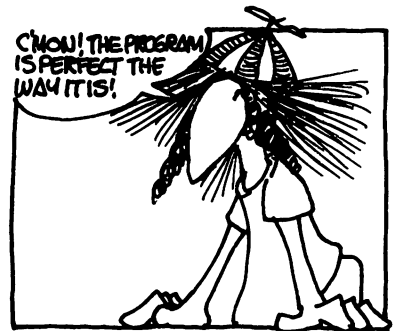
All of the concepts I've shown you are really important when you're designing programs that do lots of word manipulation. And, when you put numeric variables inside the parentheses, these *selective strings* become super-powerful!

And to show you just *how* super-powerful, I present my super-powerful Random Nonsense Generator! It shows you how to use the selective **MID\$** function with numeric variables.

Random Nonsense Generator

```
10 B = INT(RND(0)*44) + 1
20 X = INT(RND(0)*7) + 1
30 A$ = "CRUNCH SNORF WOW! ZONKERS
      GLOP ZING DING BAT BLRRP"
40 PRINT MID$(A$,B,X);
50 GOTO 10
```

Finally the truth can be revealed. I used this nonsensical program to write this book! It generates random



The Great White Expanse

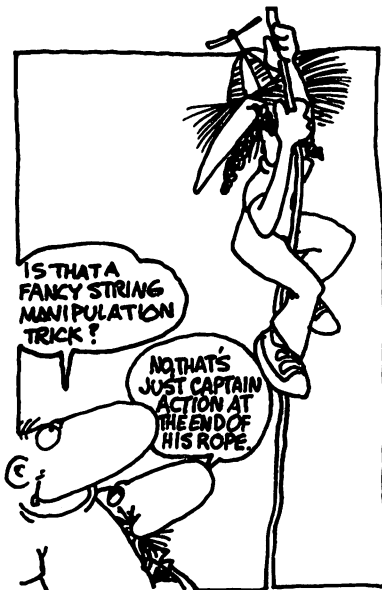
words by continually creating random values of **B**, the beginning character, and random values for the **X** number of characters selected by the **MID\$(A\$,B,X)** statement.

After printing a character, or set of characters in line 40, the program goes back up to line 10 to get another set of random numbers. With each new set of random numbers, your screen displays a different letter or group of letters.

If you look closely at the **RND** statements in lines 10 and 20, you'll see that they spit out numbers between 1 and 44, and between 1 and 7. I didn't want the length of any printed string to exceed 7 characters. That's why the **X** random-number generator spits out seven numbers. I also didn't want to go past the fifty-character length of **A\$**, and that's why the **B** random number generator begins at forty-four.

Now that you know how it's done, you can write the Great American Novel or a letter to Captain Action.

Before moving on, check out line 30. In line 30, **A\$** contains fifty characters; a space counts as a character. But, amazingly enough, I didn't have to count each character and space to find the number of characters in that line. I used the **LEN** statement!



The LENgth of Your String

When you want to write nonsensical programs, knowing how long your string is comes in real handy. If you know how long a string is, you can also slice it up to perform some pretty fancy string-manipulation tricks.

To find out how I measured the length of my string in the infamous Random Nonsense Generator, try this little program:

DR. WACKO PRESENTS COMMODORE BASIC

```
10 A$ = "CRUNCH SNORF WOW! ZONKERS GLOP  
    ZING DING BAT BLRRP"  
20 PRINT LEN(A$)
```

After you run this program, the answer 50 appears on your screen. That's the length (including the spaces between the words), of string A\$.

To show you how LEN works, I've modified this program to accept your wacked-out inputs.

```
10 INPUT "ENTER SOME NONSENSE";A$  
20 PRINT LEN(A$):GOTO 10
```

Now, for one wild and crazy application.

Limit the Length of an Input

When you're designing a computerized questionnaire, it's sometimes (but not always) nice to limit how much data you can enter. Here's an example that limits the length of a string input to 10 characters.

Limiter

```
10 PRINT :PRINT "WHAT'S YOUR FAVORITE  
    SPORT, SPORT."  
20 PRINT "(10 CHARACTERS OR LESS,  
    PLEASE.)"  
30 INPUT A$  
35 REM ***  
40 IF LEN(A$) > 10 THEN PRINT A$;  
    "? THAT'S";LEN(A$);"CHARACTERS!":  
    GOTO 10  
45 REM ***  
50 PRINT "WHAT DO YOU MEAN BY  
    THAT?":GOTO 10
```

The Great White Expanse

The LEN in line 40, in combination with the IF/THEN statement, does all the work by checking the length of A\$. The following sample run of Limiter previews the typical results you can expect from this zany program.

**WHAT'S YOUR FAVORITE SPORT, SPORT.
(10 CHARACTERS OR LESS, PLEASE.)**

?CAMEL RACING

CAMEL RACING? THAT'S 12 CHARACTERS!

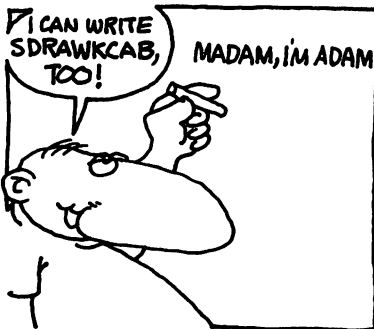
**WHAT'S YOUR FAVORITE SPORT, SPORT.
(10 CHARACTERS OR LESS, PLEASE.)**

?CMLRCNG

WHAT DO YOU MEAN BY THAT?

Now that you know how long your string is, and how to select parts of it, let's have a little fun.

**ESREVER NI YLERITNE
NETTIRW EREW KOOB TAERG
SIHT FI TAHW**



What? What if this great book were written entirely in reverse! Well, it was, until Junior got tired of standing on his head, and I got tired of talking to his feet.

And here's the short program that started all the confusion. It's Reverseroonio (It used to be Oinooreserver), and it demonstrates just how much fun you can have with a simple LEN statement and a MID\$ selective string function.

DR. WACKO PRESENTS COMMODORE BASIC

Reverseroonio

```
10 ? "TYPE IN A WORD OR SENTENCE. PLEASE  
    DON'T EXCEED TWO LINES OF TEXT."  
20 INPUT A$  
30 FOR X = LEN(A$) TO 1 STEP -1  
40 PRINT MID$(A$,X,1);  
50 NEXT X  
60 ??:GOTO 10
```

Reverseroonio counts backwards, from the last string character to the first. All this topsy-turvy counting takes place in line 30, then line 40 prints the result on your screen. Short, sweet, and zany, isn't it?

Camel Latin

The people out here in the desert speak a strange and mysterious dialect, called camel Latin. It was a mystery to me, until Junior told me how it works.

They take the first letter of each word, put it at the end of the word, then add the letters AY. Instead of saying, "Hello," they say "Ellohay" — almost like Hawaiian. In camel Latin the word *desert* translates into *esertday*. Wheew!

I wondered how Junior was able to do all this fancy translating. Then one day I spotted a short program that he had hidden underneath a skateboard at the back of his closet.

This program, called Easy Latin, can only convert one word at a time from English to Camel Latin. So, after you enjoy it, I'll dazzle you with my next creation, Ard-hay Atinlay. It can translate entire sentences. Wow!

Both of these programs show even more ways to use the LEN statement in combination with a selective string function. So, avehay unfay!



The Great White Expanse

Easy Latin

```
10 PRINT "ENTER ONE WORD, THEN PRESS  
RETURN."  
20 INPUT A$  
30 FOR X = 2 TO LEN(A$):PRINT MID$(A$,X,1);  
40 NEXT X  
50 PRINT LEFT$(A$,1);"AY"  
60 PRINT :GOTO 10
```

Ere'shay owhay histay emgay orksway. In line 30, the program counts from the second letter of the word you've entered, **X = 2**, through the last letter of the word, **LEN(A\$)**, and prints each letter, one at a time; **PRINT MID\$(A\$,X,1)**.

When the FOR/NEXT loop gets tired of looping, the program drops down to line 50, where it uses **LEFT\$(A\$,1)** to add the first letter. Then, for the final touch of nonsense, it adds the letters **AY** to the end of each word.

All this was great. But I wanted a program that could translate entire sentences to camel Latin. So, here's Ardhay Atinlay!

One word of caution! Ardhay Atinlay has lots of problems when it comes to translating the one-letter words *I* and *A*. That's because it works by taking the first letter of a word, placing this letter at the end of the word, and then adding the letters *AY*. When your computer tries to do this with one-letter words, its brain gets scrambled and the program comes to a screeching halt! So, be nice to this program by not entering any one-letter words. After all, your Commodore 64 is only a computer.

DR. WACKO PRESENTS COMMODORE BASIC

Ardhay Atinlay

```
10 S = 2
20 ? "TYPE IN A WORD OR SENTENCE. PLEASE
    DON'T EXCEED TWO LINES OF TEXT."
30 INPUT A$
40 FOR X = 1 TO LEN(A$)
50 IF MID$(A$,X,1) = " " THEN PRINT
    MID$(A$,S,X-S);MID$(A$,S-1,1);"AY ";
    :S = X + 2
60 IF X = LEN(A$) THEN PRINT
    MID$(A$,S,X);MID$(A$,S-1,1);"AY"
70 NEXT X
80 PRINT "THAT'S ALL FOLKS!"
```

Isn't that great! Now you can talk to camels and other assorted desert wanderers.

Ardhay Atinlay is really a lot like the Easy Latin program. The FOR/NEXT loop (lines 30 through 70) counts from the first letter of the sentence through, and including, the last letter. All the translating takes place in line 50. Here's a close-up shot of this amazing programming line:

```
50 IF MID$(A$,X,1) = " " THEN PRINT
MID$(A$,S,X-S);MID$(A$,S-1,1);"AY ";:S = X + 2
```

Pretty long, isn't it? But don't panic; here's what it all means. "If `MID$(A$,X,1)` equals a blank space (" "), then print the word represented by `A$` from its second character to its last character (which is a blank space) minus one. Then print the first character of this word and add the letters `AY` and a space. Finally, move `S` (Start) to the second character of the next word by making the value of `S` equal to `X + 2`. Comprene? (If not, read it again!)

The Last Word

Everything is hunky-dory until the program reaches



The Great White Expense

the last word in the sentence. The program always looks ahead for a blank space then retreats to rearrange the previous word. However, when it gets to the last word in the sentence, it looks ahead, finds nothing, and goes brzzzrk!

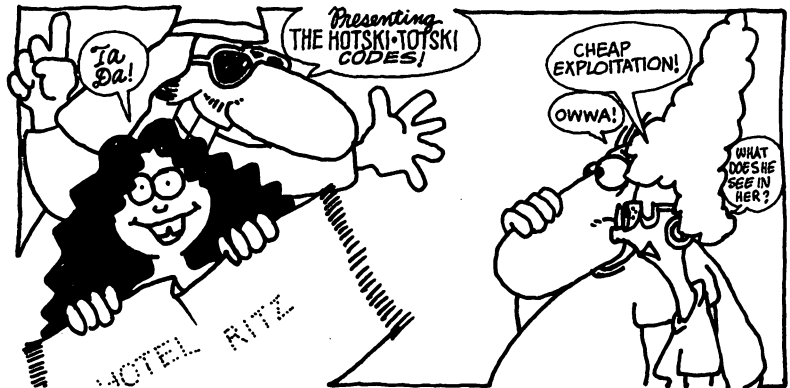
I added line 60 to remedy this little glitch. Here it is.

```
60 IF X = LEN(A$) THEN PRINT  
    MID$(A$,S,X);MID$(A$,S-1,1);"AY"
```

In line 60, when X equals the last letter of the sentence, the program prints the last word (minus its first letter), followed by the first letter of the last word, and finally the letters AY.

Understanding this program may require a little thought. The best way to really understand it is to step through each small substatement until you've got it down cold. Then, before you freeze, warm up a bit by moving on to the Hotski-Totski codes!

The Hotski-Totski Codes! Also (Sometimes) Known As The ASCII Codes



DR. WACKO PRESENTS COMMODORE BASIC

Don't flip out! Just flip to Appendix C at the back of this book and take a look at the ASCII Code chart. See it! Three columns filled with numbers, characters, functions, and typing instructions.

The first column lists numbers from 0 to 191. Next to each number is a character or description of a function. The third column contains a typing instruction.

If you want a character to appear on your screen, just follow the simple instructions next to each character's picture. For example, if you want a heart shape (number 115) to appear on your screen, just follow Space Duck's instructions.

To see what will be printed on your screen when you use a PRINT statement to perform control functions, type in and run this simple example.



Ms. Peeky's Favorites

```
10 PRINT "[CLRHM]:FOR X = 1 TO 10:  
   PRINT:NEXT X  
20 PRINT"[CTRL + 6] GREEN AND [CTRL + 5]  
   PURPLE ARE ";  
30 PRINT"[CTRL + 2] MS. PEEKY'S ":PRINT  
   "[CTRL + 8] FAVORITE COLORS!"
```

That was easy, wasn't it? Now, on to the decimal code numbers. Many of the numbers on the chart represent letters or characters that are printed on your screen. For example, 65 represents the uppercase letter *A*, and 66 represents *B*. So far so good.

Now, here comes the sneaky part. Some of these numbers represent control functions. A control function isn't printed on your screen. It does something, like clear the screen, change the color of the characters, or move the cursor.

The Great White Expanse

Most control functions can be performed by putting them in a PRINT statement within quotation marks. But there are some control functions that can't be put inside quotation marks. For example, look at the Hotski-Totski chart and check out decimal numbers 8,9,13, and 14. You'll see that there is no symbol in quotes in the fourth column! This means that you cannot make these functions perform by putting them in a PRINT statement.

Presenting CHR\$

Here's a new BASIC statement called CHR\$ that lets you put *all* the control functions to use. Enter this line in the *immediate* mode:

```
PRINT CHR$(65) [RETURN]
```

The letter *A* appears on your screen! Replace the number inside the parentheses with 66 and the letter *B* appears. Just look at the decimal code column in the Hotski-Totski chart and you'll see how it all works!

CHR\$ converts all those numbers to their associated characters, and it can also put the control characters through their paces.

Here's the Ms. Pecky's Favorites program again. But this time all the work is done with help from CHR\$.

Ms. Pecky's Favorites II

```
10 PRINT CHR$(147):FOR X = 1 TO  
10:PRINT:NEXT X  
20 PRINT CHR$(30);"GREEN AND ";CHR$(156);  
"PURPLE ARE ";CHR$(5);  
30 PRINT "MS. PEEKY'S"  
40 PRINT CHR$(158);"FAVORITE COLORS!"
```

DR. WACKO PRESENTS COMMODORE BASIC

This version of Ms. Peeky's Favorites uses the CHR\$ statement to achieve the same results you saw in the original program.

In line 10, CHR\$(147) clears the screen. Then the remaining lines, CHR\$(30), CHR\$(156), CHR\$(5), and CHR\$(158) color the characters *green*, *purple*, *white*, and *yellow*.

Now, here comes a short program that takes two control functions that *can't* be put in a PRINT statement, and puts them through their paces.

This program is called "Exercise," and was one of my favorites when Junior went through boot camp. Enter Exercise, run it, and we'll puff through it together.

Exercise

```
10 PRINT CHR$(147)
20 PRINT CHR$(30)
30 PRINT "HOW ABOUT A FEW PUSH—UPS?"
40 PRINT:PRINT "SURE, NO PROBLEM!"
50 PRINT
60 PRINT CHR$(177); CHR$(142);
   CHR$(157)::FOR X = 1 TO 150:NEXT X
70 PRINT CHR$(178);CHR$(14);
   CHR$(157)::FOR X = 1 TO 150:NEXT X
80 GOTO 60
```

CHR\$(147) in line 10 clears the screen and sends the cursor home.

In line 20, CHR\$(30) takes all the text in lines 30 and 40 and colors it green.

In line 60, CHR\$(177) prints an upside-down T below the text, CHR\$(142) switches to uppercase printing, and CHR\$(157) forces the cursor to the left so the upside-down T stays in position. The FOR/NEXT loop



The Great White Expanse

at the end of the line pauses the action, so you can see what's going on.

In line 70, **CHR\$(178)** prints a rightside-up **T** to provide an animation effect, **CHR\$(14)** switches to lowercase printing, and **CHR\$(157)** forces the cursor to the left so the rightside-up **T** stays in position.

Line 80 branches the program back up to line 60 to repeat the action.

CHR\$ really works out well in this program, because it's the *only* way to switch between upper- and lowercase printing...you can't put this command in a **PRINT** statement.

I use **CHR\$** a lot, and you'll really get an eyeful (of **CHR\$**'s that is) when we design a full-fledged word processor later in our journey. By the way, if you return to the Ardhay Atinlay program, and look at line 50, you'll see that you can replace the two quotation marks with **CHR\$(32)**, the space bar!

Finally, here's a real short routine that displays all the print characters on your screen.

```
0 REM *** DISPLAY THEM ALL!  
10 FOR X = 33 TO 127  
20 PRINT CHR$(X);  
30 NEXT X  
40 FOR X = 161 TO 191  
50 PRINT CHR$(X);  
60 NEXT X
```

Wow!

Ask the ASC

The ASC has all the answers. Just place a character, or control function in quotation marks, surround the quotes with parentheses, and the amazing ASC tells you the character's decimal code number. Now you won't have to look at the Hotski-Totski chart to get a character's number, the ASC does all the looking for you!

Here's how it's used.

```
PRINT ASC("A") [RETURN]
```

When you type this in, and press the RETURN key, the number 65 appears on your screen. Put any letter or character inside quotes, wrap parentheses around the quotes, and watch the results. Amazing! It's just like the Hotski-Totski chart!

The abbreviated instructions for entering control characters are listed in column three of the Hotski-Totski chart after the description of the function each character performs. For example, here are the abbreviated instructions for entering the clear-screen control function: "CLRHM"

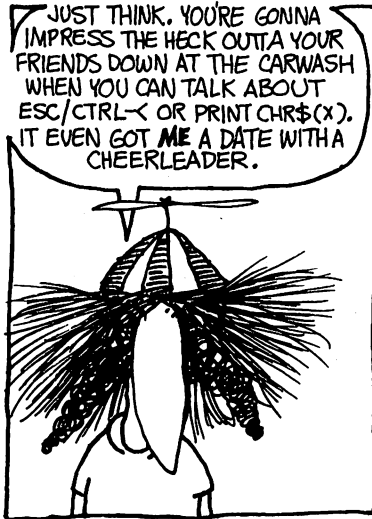
Tying String Together

Things are starting to wind down here at this oasis, but before we load up our camels and hit the trail I'd like to show you one last string trick.

Commodore 64's BASIC language lets you *tie a bunch of strings together!* And you don't even have to know lots of fancy knots. It's easy. Here, I'll show you a quick example.

The Great White Expense

Remember when we cut apart the string "CARAVAN?" This short example shows you how to put it back together again!



Tying the Knot

10 A\$ = "CAR"

20 B\$ = "A"

30 C\$ = "VAN"

40 PRINT A\$ + B\$ + C\$

After you run this spiffy program, your CARAVAN will be back in one piece, and we can move on to the final oasis.

Now that you know most of my string-manipulation tricks, use them to dazzle (and confuse) your admirers when you conjure up a wordy program!

Oasis 5: Chat with Your Fellow Travelers and Psychoanalyze Your Computer



All that programming you've learned during your journey through the desert lets you perform some pretty snazzy computer magic. But if your programs can't talk to and control the printer, disk drive, or cassette recorder, you're missing out on lots of pizazz.

This is the last oasis you'll visit before entering the dazzling worlds of graphics and sound. It might very well be the most important. In the last part of this book we'll be designing a word processor and secret text coder/decoder. To run these programs, your computer has to chit-chat with other parts of your system. So if you want to join in the coffee klatch, spend a little time at this oasis and I'll show you how your computer raps with its fellow travelers.

This oasis has another treat in store for you. I'll show you how to look inside your computer's brain, scramble it up, and produce some pretty weird effects. So grab a cushion, sit down in front of your Commodore 64, munch a fig, and we'll get started.



Garbage In, Garbage Out

Your Commodore 64 talks to its *peripheral devices* (like the printer, disk drive, cassette recorder, or anything else that talks back, like Clyde,) by using a special number, called a *Device #* for each device. Here's a list of these numbers.

| Device | # |
|-------------------|---|
| Cassette recorder | 1 |
| Printer | 4 |
| Disk drive | 8 |
| Junior | 0 |

A Few Important Steps

1. *OPEN up a channel.* Before you can tell your Commodore 64 to chat with one of the devices, you've got to OPEN up a channel for communication.

There are two hundred and fifty-five (255) input and output channels, numbered 1 through 255, that your Commodore 64 sets aside for all this chit-chat. But because numbers over 128 were originally designed for other uses, it's a good idea to only use numbers below 127 for input and output channels.

You open up a channel with the OPEN statement, like this.

OPEN 1,4,7

Captain Action's got a good point. What is all that stuff behind the OPEN?

2. *Choose a channel.* First, the 1 means that channel 1 has been opened. Remember, you can use any channel number between 1 and 255, the choice is yours.



DR. WACKO PRESENTS COMMODORE BASIC

3. *Select a device.* The second number is the *Device #*. In this example, because the second number is a **4**, we've opened up a channel to the *printer*.
4. *Boss the device around.* The next number is the order number. It lets you tell your computer and the device what they're supposed to be doing.

In this example, **7** means that you want to *print upper and lower case characters on the printer*.

Petunia's drawn this simple chart (called "Petunia's Chart," naturally) that shows all the Device and Order Numbers we'll use in this book. Bless her heart.

Petunia's Chart

| Device Order | Device Number | Number |
|--------------|--------------------------------|------------------------------|
| Cassette 1 | 0 = Input | 1 = Output |
| Printer 4 | 0 = Uppercase/ Graphics | 7 = Uppercase/ Lower Case |
| Disk drive 8 | 2-14 = Send or receive data | 15 = Send commands |

There's Even More!

When you're communicating with the cassette recorder or disk drive, descriptive text in quotes always follows the OPEN statement's three numbers. This format helps you really get your point across to the device.

Cassette: To communicate with your cassette recorder, follow the OPENing statements with a final comma and a *file name* like this:

OPEN 1,1,1,"FILENAME"

In this example, the first **1** after OPEN is the channel number. The next **1** is the cassette device number, and the third number **1** tells the cassette recorder that you

The Great White Expanse

intend to *send* information to the cassette. Once this is settled, you end with a final comma and a *file name* in quotes.

Helpful hint: Before you move on, now's a good time to take a minute and flip back to Appendix B to learn how to load and save programs to disk or cassette, and how to name files.

Disk Drive: When you want to send data *to* your disk drive, or get data *from* your disk drive, you open a *data channel* by using any Order Number between 2 and 14. In this case, the descriptive information that follows the OPEN statement's final comma is very precise and meaningful. Here's an example:

OPEN 1,8,2,"0:FILENAME,S,W"

This statement OPENS a data channel between your computer and disk drive 0. The information within quotes is broken down into four segments. Let's look at each one in detail.

1. **0:** is the *drive number*. "0:" is your first drive, a "1:" would call out your second drive.
2. **FILENAME** is the *name* you've assigned to this file.
3. **S** identifies the *type* of file you are working with. In this example "S" is the abbreviation for *sequential file*. As you breeze through the rest of this book you'll have plenty of opportunity to see sequential files in action. It's the only type of file I've used!
4. **W** means that you want to **Write** information *to* the disk drive.

To **Read** information *from* the disk drive replace the **W** with an **R**.

DR. WACKO PRESENTS COMMODORE BASIC

In a minute I'll show you how to put these fabulous OPEN statements to use. But before I get ahead of myself and get carried away, here's a translation of the OPEN statement we just reviewed:

OPEN 1,8,2,"0:FILENAME,S,W"

Translation: Open a data channel (channel 1) to disk drive 0. Name this file, "FILENAME", make it a sequential file, and write to the disk. Wheew!

CLOSE That Channel!

It's easy (but nasty) to pull the plug and disconnect your computer from one of its devices. Still, you should close each channel after it's been used. If you don't, your information won't be stored on disk or tape! So, when you're ready to close a channel, just use the CLOSE statement like this.

CLOSE 1

This example closes channel 1. But of course, you can close any channel you've opened.

Now that you know how to OPEN and CLOSE channels, it's time to show you how to put all those devices to work.

GET# and PRINT# (That Famous Vaudeville Team?)



The Great White Expanse

The GET# and PRINT# statements work together like pizza and beer (Captain Action's two favorites!). After you OPEN a channel, you can use GET# to *get* information from a device. Or, if you want to be sneaky, you can use PRINT# to *print* information into, or on, a device! Sounds weird, doesn't it? Don't worry (famous last words), I'll show you how to put this mish-mash to practical use.

Using GET# and PRINT# to Load and Save to Your Disk or Cassette Recorder

The heart of the word processor we'll be designing later uses the infamous GET# and PRINT# commands to save your prose to either disk or cassette recorder and load it in again. Here are some examples that show how this pair of dingalings work with the OPEN statement to get the job done.

Use PRINT# to Save Your Prose

The modest Super-Saver program that follows shows how to save information to your disk or cassette recorder. After you type in Super-Saver, and before you run it and start going wacko, spend a few moments on the Guided Tour that follows the program listing.

Super-Saver

```
1 REM ** SUPER—SAVER **
10 DIM B$(10)
20 FOR X = 1 TO 6
30 READ A$
40 B$(X) = A$
50 NEXT X
60 INPUT "SAVE TO DISK OR CASSETTE
(D OR C):";S$
```

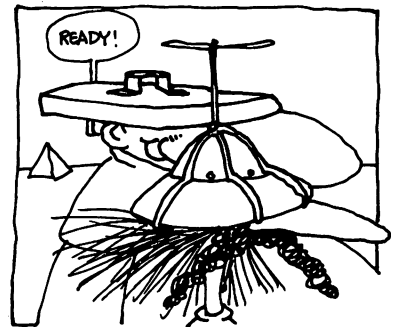
DR. WACKO PRESENTS COMMODORE BASIC

```
70 IF S$ = "C" THEN GOTO 90
75 REM **
80 OPEN 1,8,2,"0:WACKED,S,W":GOTO 100
85 REM **
90 OPEN 1,1,1,"WACKED"
95 REM **
100 FOR X = 1 TO 6
110 PRINT#1,B$(X);
120 NEXT X
130 CLOSE 1
140 DATA DOCTOR ,WACKO ,IS ,A ,GENIUS
150 DATA *
```

A Guided Tour Through Super-Saver

Put on your safari hat and let's venture into Super-Saver.

Lines 10 through 50, read the modest data I've stored in lines 140 and 150, and put it all into the string array, `B$(X) = A$`, in line 40.



Once this is done it's time to ask the big question in line 60 — "SAVE TO DISK OR CASSETTE?" You answer this by entering a "D" for disk or a "C" for cassette.

A Wacko Message: I could have designed this program to write the data directly onto disk or tape, rather than store it in an array first. But, since an array storage technique is used in the word processor at the end of this book, becoming familiar with this method now will help prepare you for that creative adventure.

Now comes the exciting part, lines 80 and 90. These two lines are what this oasis is all about.

If you elect to save to your disk, line 80 opens a data channel to the disk drive. It also creates a **S**quential file named "WACKED", and tells the program that you're going to **W**rite to the disk drive.

The Great White Expanse

If you selected the cassette option, the program bops down to line 90. This OPEN statement opens a channel to your cassette recorder, tells the program that you intend to output, or save, information, and assigns the file name "WACKED" to the whole mess.

Now that all channels are open, and the computer knows what it's supposed to do, the program continues on to lines 100 through 120, where the fabulous PRINT# gets into the act.

In lines 100 and 120, the FOR/NEXT loop which surrounds the PRINT#1 statement takes each of the six data statements (including the "*" in line 150) *from* array B\$(X), and in line 110, PRINTs each data statement to the disk drive via channel 1. The number after the PRINT# statement, 1 in this example, always refers to the opened channel. Always place the channel number *directly* after the "#" symbol like I did in the program. If you leave a space, your computer will go bzzrk!

When the FOR/NEXT loop runs out of steam, the program drops down to line 130, where CLOSE 1 turns off the channel.

Enter Your Own File Name

Before you move on I'd like to show you how to modify Super-Saver so you can enter a file name of your choice, rather than put up with the "WACKED" name I've chosen.

First, add this line to Super-Saver.

```
55 INPUT "ENTER A FILE NAME";F$
```

DR. WACKO PRESENTS COMMODORE BASIC

Now replace lines 80 and 90 with these two new lines.

```
80 OPEN 1,8,2,"0:" + F$ + ",S,W":GOTO 100
90 OPEN 1,1,1,"F$"
```

That's all there is to it! The name you enter becomes F\$, and drops into the OPEN statement. I use this nifty trick in the word processor at the end of this book to open files for loading and saving information.

Now that you understand what Super-Saver does, and how it does it, run the program and save all that data to your disk or cassette.

All set?

OK, now it's time to put the GET# statement to work, and retrieve all my egotistical ramblings.

Use GET# to Load Your Prose

Save the Super-Saver program to your disk or cassette for future reference, then type NEW, and enter Lazy-Loader.

Lazy-Loader

```
10 INPUT "LOAD FROM DISK OR CASSETTE
(D OR C)";L$
20 IF L$ = "C" THEN GOTO 40
30 OPEN 1,8,2,"0:WACKED,S,R":GOTO 50
40 OPEN 1,1,0,"WACKED"
50 GET# 1,A$
60 IF A$ = "*" THEN CLOSE 1;END
70 PRINT A$;
80 GOTO 50
```

Simple, isn't it? When you run Lazy-Loader you'll see my prose printed on your screen!



The Great White Expanse

Lazy-Loader is a short program because it doesn't include the fancy array techniques used in Super-Saver.

After you decide whether to load from disk or cassette, the program moves down to either line 30 or 40. Here, channels open between your Commodore 64 and the disk drive or cassette recorder.

GET#1,A\$ in line 50 gets each character and assigns it to **A\$**. (The number *1* after the “#” symbol is the channel number.)

PRINT A\$; in line 70 prints each character on the screen, and then, in line 80, the program loops back to line 50 for the next character.

In line 60, when **A\$** equals an asterisk (*), channel 1 is closed and the program comes to a screeching halt. I inserted the asterisk in the Super-Saver program as a *flag* to tell the Lazy-Loader program when it has reached the end of the file.

Using INPUT# and PRINT#

Use the **INPUT#** statement almost like you would use the **GET#** statement. But, **INPUT#** enters large chunks of data, up to 80 characters in one fell swoop, whereas the **GET#** statement gets only one character at a time.

Here are a set of Save and Load routines that show you how **INPUT#** and **PRINT#** work together. Type in and run Save-Routine first, then move on to Load-Routine. Both of these programs are designed for a disk drive. If you've got a cassette recorder, just modify the **OPEN** statements. No sweat, right?

DR. WACKO PRESENTS COMMODORE BASIC

Save-Routine

```
10 INPUT "ENTER SOME TEXT";T$
20 INPUT "ENTER A FILE NAME";F$
30 OPEN 1,8,2,"0:" + F$ + ",S,W"
40 PRINT#1,T$
50 CLOSE 1
```

You'll be asked to enter some text after you run this program. Type in anything you want, within the limits of decency, *but don't exceed the two-line screen limit!*

Load-Routine

```
10 INPUT "ENTER A FILE NAME";F$
20 OPEN 1,8,2,"0:" + F$ + ",S,R"
25 REM **
30 INPUT#1,A$
35 REM **
40 PRINT A$
50 CLOSE 1
```

The big news is `INPUT#1,A$` in line 30 of Load-Routine. Here, `INPUT#1` enters *all* the text you've saved and assigns it to `A$`.

Your Prose in Print

Now that you've made it through the tricky part, I'll show you how to print out your prose on your printer.

It's really easy. All you've got to do is open a channel to the printer and use a `PRINT#` statement to do all the printing. Turn on your printer, type in and run this short example, and startle your aardvarks.

```
10 OPEN 1,4,7:PRINT#1,"STARTLE YOUR
  AARVARKS? IS HE SERIOUS?"
20 CLOSE 1
```


The Great White Expense

No problem. Now, let's get really fancy. Here's a short program called "Printeroonio" that GETs your typed input, opens a channel to your printer, and lets you send all the great prose you've written on the screen to your printer!

Here's the program. Type it in, run it, and start banging on your keyboard.

To send the contents of your screen to your printer: Type an asterisk (), and after a short pause your printer will come to life (if it's turned on).*

Printeroonio

```
10 DIM B$(500):T = 1
20 GET A$:IF A$ = " " THEN GOTO 20
30 IF A$ = "*" THEN GOTO 1000
40 PRINT A$;
50 B$(T) = A$
60 T = T + 1:GOTO 20
1000 REM ** PRINT OUT ROUTINE **
1010 OPEN 1,4,7
1020 FOR X = 1 TO T
1030 PRINT#1,B$(X);
1040 NEXT X
1050 PRINT#1:CLOSE 1:T = 1:GOTO 20
```



In line 20, this short program first GETs each character from the keyboard. Don't insert any space between the two quotation marks that follow `A$ = "`. The two quotation marks, with no space in between, tell the IF/THEN statement to return back to the beginning of line 20 if *no key* is pressed.

When you press a key, the program moves down to line 40 where it PRINTs each character on the screen. Finally, it goes down to line 50 and places each character into array `B$(T)`. Far out!

DR. WACKO PRESENTS COMMODORE BASIC

The Printing Takes Place in Lines 1010 through 1040

When you type the asterisk (*), the program branches down to line 1000 and the Print Out Routine goes into action.

First, in line 1010, the program opens an upper-lowercase channel to the printer. Then the FOR/NEXT loop (lines 1020-1040) PRINTs each character on the printer.

When the loop runs out of steam, the program goes to line 1050, PRINTs a blank to clear the printer's built-in memory (printer buffer), CLOSEs the channel to the printer, resets the value of T to 1, and returns back to line 20 to get the next keyboard input. Nifty!

In line 10, I DIMensioned B\$ to store 500 characters. Increase this number if you'd like to print out longer bits of prose. Wheew!

Send Commands To Your Disk Drive

If you flip back to Petunia's Chart you'll see that she's listed **15** as the order number used to send commands to your disk drive. This special order number is used when you want to modify or eradicate files that you've saved on disk.

Using this special order number lets you:

- Format a new disk
- Rename a disk file
- Erase a disk file
- Perform other esoteric disk-file tasks

The Great White Expanse

Formatting a New Diskette

Before you can store programs on a disk, you've got to format it so it's compatible with your Commodore 1541 disk drive. Formatting a disk also creates a disk directory, so you'll know what you've stored on it.

To look at the disk directory just enter the following statements in the *immediate* mode.

LOAD "\$",8 [RETURN]

When your screen says "READY", type LIST, and press RETURN.

One word of *caution*. When you format a disk you automatically erase *everything* that was previously stored on it.

Here's how to use order 15 to format a diskette.

1. Open up a channel (in this example I've used channel 1) and activate order 15:

OPEN 1,8,15

2. Follow the OPEN statement with this fancy PRINT# statement:

PRINT# 1,"NEW0:NAME,ID"

The words in quotes are critical, and here's what they mean. "NEW" is a command, and is always used to format your disk. The "0:" is the disk drive. "NAME" should be replaced with any name of your choosing. It will be printed on the disk's directory. "ID" should be replaced with any two characters you want to use to identify the disk.

3. Close the channel:

CLOSE 1



DR. WACKO PRESENTS COMMODORE BASIC

Here's how I formatted a new disk:

```
OPEN 1,8,15:PRINT#1,"NEW0:WACKO,DH":  
CLOSE 1
```

Renaming a Disk File

Renaming a disk file is real easy. Here's how it's done.

Open up a channel and activate order 15, follow this with a fancy print statement, then close the channel.

```
OPEN 1,8,15:PRINT#1,"RENAME0:NEWNAME =  
OLDNAME":CLOSE 1
```

RENAME is the rename command and is always used to, you guessed it, rename a file. This command is followed by the disk drive number **0**, and a colon. **NEWNAME** is the name you've chosen as a replacement name for the existing file. **OLDNAME** is the name of the existing file.

Here's how I changed the name of the **JUNIOR** file to **PETUNIA**.

```
OPEN 1,8,15:PRINT#1,"RENAME0:PETUNIA =  
JUNIOR":CLOSE 1
```

Erasing a File

Erasing, or **SCRATCHing**, a file is easy to do. But, once you have eliminated a file, it's gone forever. So, take care!

```
OPEN 1,8,15:PRINT#1,"SCRATCH0:FILENAME":  
CLOSE 1
```

The Great White Expanse

SCRATCH is the command you use to erase a file. **0** is the drive number. **FILENAME** is the name of the file you want to Obliterate. Here's how I got rid of a file named CLYDE.

```
OPEN 1,8,15:PRINT#1,"SCRATCH0:CLYDE":  
CLOSE 1
```

The camel who never returned!

Now that we've opened, closed, totally rearranged, and destroyed a bunch of files, we can proceed to psychoanalyze your computer.

Psychoanalyze Your Computer

Throughout this book you may have noticed a couple of weird-looking statements hidden in the recesses of some of the programming examples. If you haven't spotted these statements yet, you'll be getting an eyeful of them in the Graphics and Sound chapters. Here's what they may have looked like.

```
POKE 53280,0
```

or

```
PEEK (53281)
```

"What," you may have asked yourself, "is Dr. Wacko doing?"

Elementary, my dear desert wanderer. I was peeking inside the computer with the PEEK statement, and poking around with the POKE statement.

The PEEK statement lets you look inside your computer's brain. When you find out what's in it, you can use the POKE statement to scramble it up!



DR. WACKO PRESENTS COMMODORE BASIC

There are lots of hidden recesses inside your computer's memory, called *locations* with *addresses*, that do special things, or hold certain types of information. Sneaky Peek's listed a bunch of his favorite locations back in Appendix D, but they won't do you much good unless you know how to get in there and really mess them up.

So get out your scalpel and we'll cut through all the rhetoric and peek inside your computer's brain.

Here's how to use a PEEK statement to see what's going on inside a location.

PRINT PEEK(646) [RETURN]

When you execute this short routine in the immediate mode, the number 14 appears on your screen. That's what's inside location 646. Let's try another one.

PRINT PEEK(650) [RETURN]

This time 0 appears on your screen. You guessed it. The number 0 is stored in location 650.

It's easy! The number inside the parentheses, after the PEEK statement, is the location you're peeking into.

Now, let's POKE around inside location 650, and put our own number inside.

POKE 650,128 [RETURN]

Uh oh, I think something's happened! Press down on *any* key, and it repeats itself across the screen. Amazing! To set things straight again, just:

POKE 650,0 [RETURN]

Wheew, back to normal! That was a close call. But it did demonstrate the power of POKE.



The Great White Expanse

You can POKE a location with any whole number between 0 and 255 by placing it after the comma, behind the location number.

Now that you know how to PEEK inside your computer's brain and POKE around, I'll show you how to use some of my favorites.

To change the color of the characters that appear on your screen, just POKE 646 with a number from 0 to 16. Try this.

POKE 646,4 [RETURN]

My favorite color, putrid purple! Try some other numbers, then to return to the normal display, POKE 646,14. Simple, isn't it?

Changing The Screen's Color

Now for another of my colorful favorites. Changing the color of the screen and it's border.

To change the screens color, just POKE 53281 with any number from 0 to 16. To change the border color, POKE 53280 with any number from 0 to 16. Here's how to get everyone's attention. A black screen, bright green border, and yellow printing!

POKE 53281,0:POKE 53280,3:POKE 646,7

Good grief!

There are lots more POKES and PEEKS waiting for you in Sneaky Peek's Appendix. So, if you'd like, wander back there now, and have some fun!

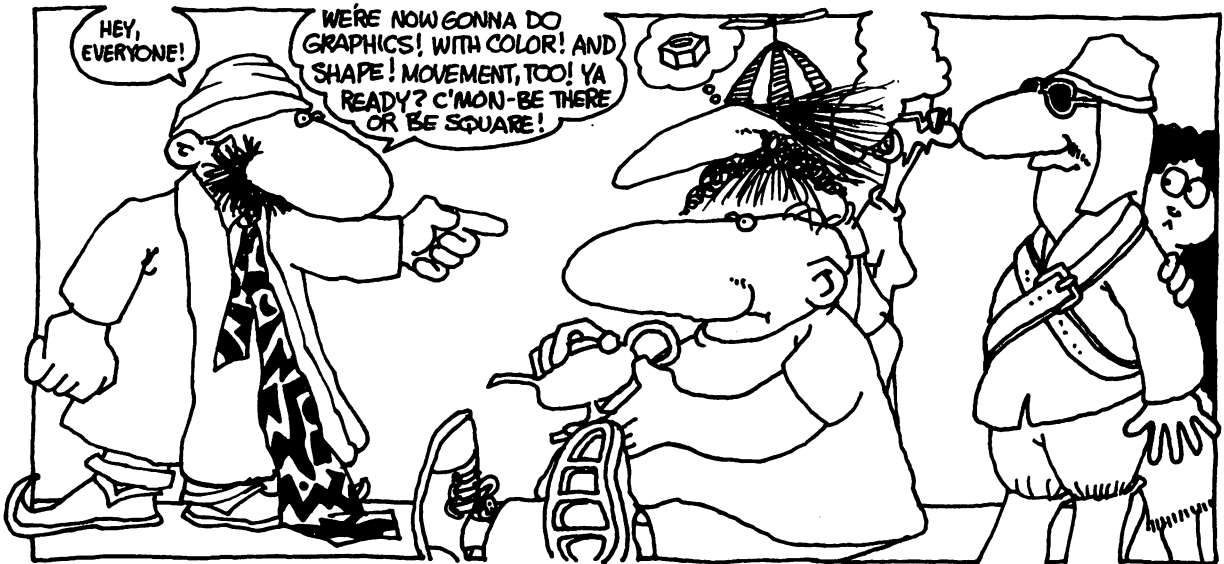


You Made It!

Congratulations! You've made it through the desert, and now it's time for total pixel control! So flip the page, leave your sunglasses on, and we'll explore the dazzling world of Commodore 64 graphics!

4

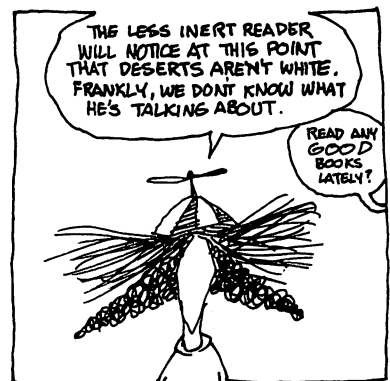
Graphics Power: An End to The Desert Blues



Up to this point (with a few minor exceptions) you've been sitting in front of a blue screen filled with white printing, and I'll bet that you're about to hang it up and return to reality. Things can get awfully dull without color.

I know. Three months of traveling around in the desert was enough to dull my senses and make me almost color blind! White, white, WHITE! I couldn't stand it any more, and neither could Petunia; Snidely Seersucker; Ms. Peeky; or Captain Action. (Clyde and my brilliant son, Junior, loved it!) I almost had a mutiny on my hands!

Your Commodore 64 computer has the graphics power to dazzle and mesmerize you with vibrant colors, weird shapes, and strange movable objects. So, don't mutiny! I'm about to show you how easy it is to take charge of Commodore 64's computing power and end those desert blues.

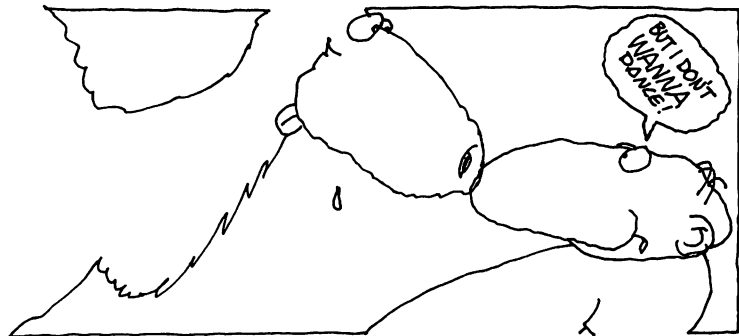


Graphics Power

Get set for a real treat. In this humongous chapter you'll watch in amazement as my name streams before you in brilliant color, place your bets and create your own colorful Camel Racetrack, and for the grand finale, astound the nomads by recreating my infamous Model of the Universe. Zonkers!

But seeing is believing. Leave your sunglasses on, turn on your creative self, and we'll get started.

I'd like you to enter, run, and have some fun with the short program below. It will give you a taste of what's in store for you as you wander through this chapter. The program is called "Bijou Billboard." It puts my name (of course) in lights, and can be modified to put yours there, too! Some of the commands and statements in Bijou Billboard may be new to you. Many of them may be old hat. Don't worry! When you've finished this chapter, Bijou Billboard will seem as easy as riding a camel or dancing at the oasis.



Bijou Billboard

```
10 REM ** BIJOU BILLBOARD **
20 POKE 53280,0:POKE 53281,0:PRINT"[CLR]";
30 T$ = "[CTRL + 3] W [CTRL + 8] A [CTRL + 6]
    C [CTRL + 5] K [CTRL + 4] O [CTRL + 7] "
40 PRINT "[HOME]":FOR Y = 0 TO 12 PRINT
    "[CRSR-D]";:PRINT "[RVS ON]"TAB(12)T$
50 A$ = LEFT$(T$,1):T$ = MID$(T$,2) + A$
60 FOR T = 1 TO 35:NEXT T
70 GOTO 40
```

DR. WACKO PRESENTS COMMODORE BASIC

I'll bet you noticed a few oddball words as you typed in Bijou Billboard — words like TAB, and those two wierd POKE's in line 20. As I said before, don't sweat it! (Not easy out here in the desert.) You'll learn *all* in a little while. Now it's time to run the program and have some fun.

Since you've sampled a bit of your Commodore 64's colorful capability, it's time to push on to the nitty, gritty, and sometimes glamorous world of Commodore 64 graphics.

The Sheik of Araby — Lights, Camera, Action. Rudolph Wackentino?

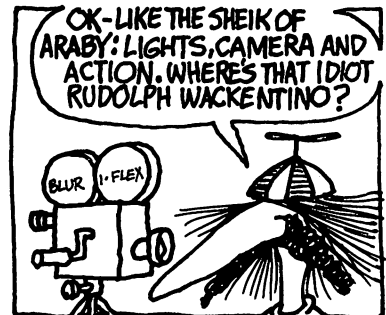
I know I look handsome, sheik (sic), and suave in this get-up (hic). But I'll bet you're wondering why I'm dressed up this way? Well, to be perfectly honest, I did it to attract your attention! Ever since boot camp, Petunia and the other recruits refused to talk to me, and I had to find some way to get their attention.

Now that I've gotten your attention — well almost everyone's attention — it's time to tell you the secrets of my success on the silver screen.

Your Commodore 64's Screen

I perform some of my most dazzling feats right here on your Commodore 64's screen, not at the local casbah. And so can you!

I always like to ham it up on the Commodore 64. This great computer offers so many ways to put colorful characters on the screen that I can always find the right method for *any* performance — the weirder, the better!



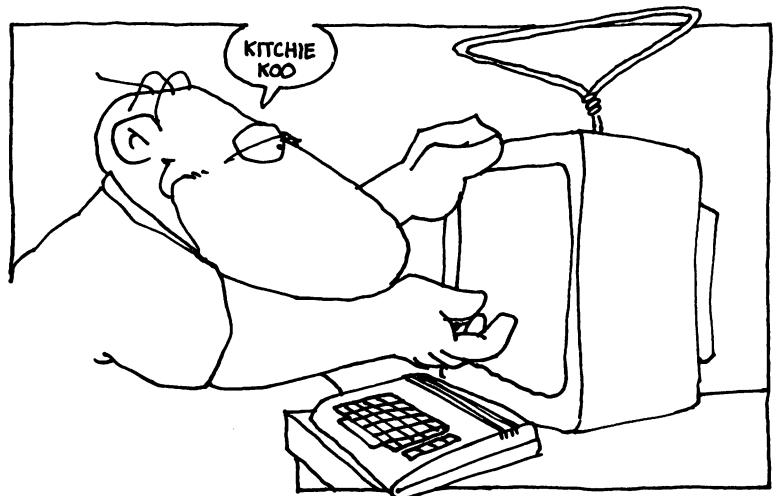
Graphics Power

Each graphics display method lets me express my brilliant talent in a different way. I use some graphics methods to display the text titles at the beginning of the show, as Bijou Billboard so brilliantly demonstrates, and others to wow the audience with my artistry during the show.

Setting The Stage: Commodore 64's Amazing Screen

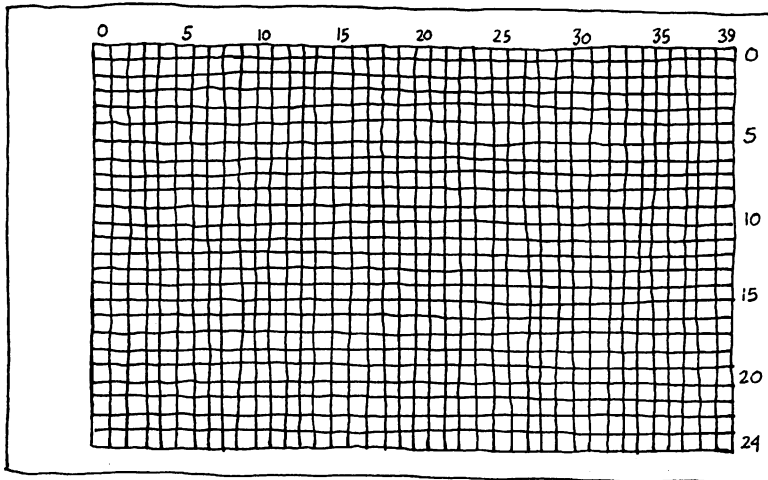


Constructing a stage set and backdrop for your desert fantasy takes a lot of backbreaking work, so to make things easier, you first need to get acquainted with the Commodore 64 computer's world-famous screen.



DR. WACKO PRESENTS COMMODORE BASIC

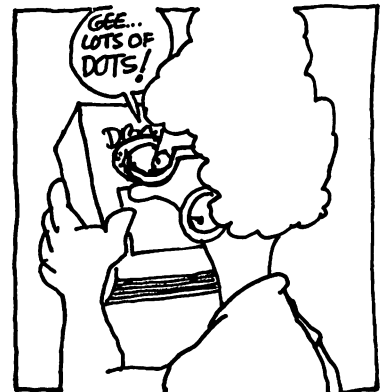
The first pearl of knowledge I'm going to impart is this: Your screen's size is 40 columns across by 25 rows down. No biggie, but this means that you can fill your screen with up to one-thousand (1000) weird shapes, characters, spaces, and nonsense! Take a look at Figure 4.1. This drawing shows how your screen is layed out, complete with numbers across its top and down its side. What, you might ask, are all those numbers? No problem....



The Wacko Unified Hole Theory: Columns, Rows, and Coordinates

Take a magnifying glass (or squint a lot) and look at the color cover of this book.

Right, Ms. Peeky. And characters and colorful graphics are displayed on your computer's screen in the exactly the same way: your screen contains one-thousand "holes" (pixels) waiting to be filled in. Just like the well at the oasis waiting to be filled with water.

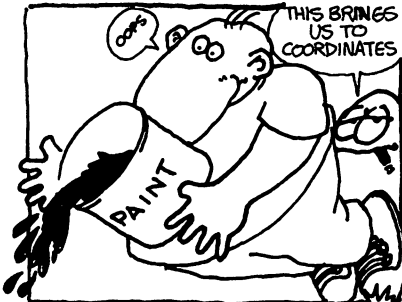


Graphics Power

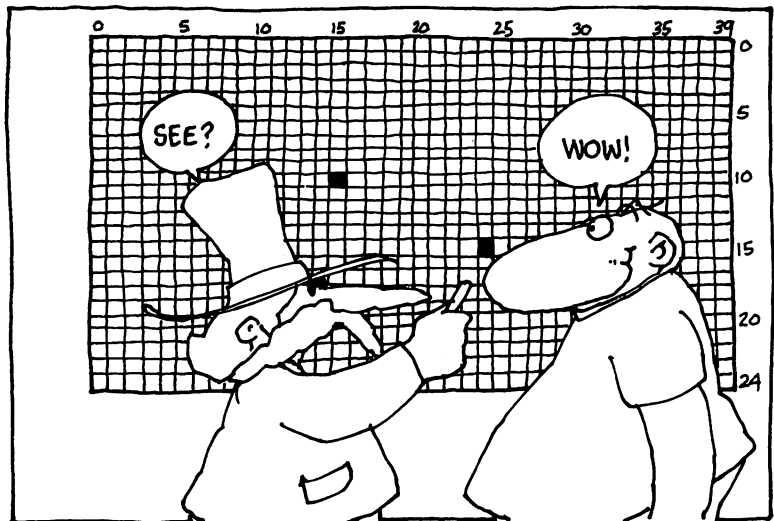


Coordinates

Now on to those numbers across the top and down the side of the drawing. It's simple! Each hole (or pixel) is assigned its own two-number location called a *coordinate*. This nifty system works just the same as the map you used to rescue Lawrence of Newark on page 116. Here's how the coordinate system applies to Commodore 64 graphics.



The numbers across the top of the illustration assign a value to each column; these are called "X" locations. The numbers down the illustration's side assign a value to each row; these are called "Y" locations. *Each pixel is identified by its two-number location coordinate.* I always state the pixel's column location (X) first, followed by the pixel's row location (Y).



Snidely's filled in the holes at location 15,10(X,Y) and at location 24,15 (X,Y) to illustrate this simple concept.

Fill in Those Holes

Now that we've gotten the basics out of the way it's time to get fancy and start filling in those holes with truly meaningful information.

DR. WACKO PRESENTS COMMODORE BASIC

Curses — Cursors within Quotes!

One of the easiest ways to print any character on your screen, *just where you want it*, is by putting a cursor movement symbol inside quotes, then following it with the character you want printed. For example, here's how to put the letter *A* at location 18(X),10(Y) — just about smack in the middle of your screen.

Middle A

```
10 REM ** MIDDLE A **
20 PRINT "[CLRHM]"
30 FOR X = 0 TO 18:PRINT "[CRSR-R]";:NEXT X
40 FOR Y = 0 TO 10:PRINT "[CRSR-D]";:NEXT Y
50 PRINT "A"
```

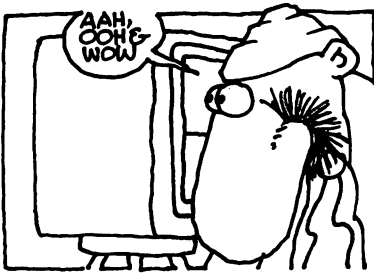
Line 20 gets this program off to a good start by clearing the screen and sending the cursor home to the upper left-hand corner of your screen. Then, in line 30 the cursor is moved 19 spaces to the right, from X location 0 to location 18. Line 40 then moves the cursor down 11 spaces, from Y location 0 to location 10. Finally, in line 50, the letter *A* is printed, just where you want it!

The Amazing TAB!

Your Commodore 64 is equipped with the amazing TAB command that makes positioning your characters a snap. The TAB command is always followed by a number enclosed in parentheses like this: **TAB(19)**. Just place the TAB command before a character, and the character is printed to the right of the left margin — how far to the left or right is determined by the number of spaces specified inside the parentheses. **PRINT TAB(19)"A"**, for example, prints the letter *A* in column 19 of your Commodore's screen. Try it in the *immediate* mode.

Graphics Power

To see how effective TAB is in actual practice, delete line 30 from the Middle A program, and change line 50 to: **50 TAB(19)“A”**. TAB really simplifies your positioning chores, and if you look back at line 40 of the Bijou Billboard, you’ll see how I put it through its paces.



Some Cursory Magic

Using cursory movement to put your characters where you want them is great. I used cursory movement in my Bijou Billboard program. But there’s even more! Cursory movement can be used to perform computer magic!

This next program, “Vanish”, prints the letter A on your screen, then, after a short pause and a wave of my magic wand, makes it disappear!

Vanish

```
10 REM ** VANISH **
20 PRINT “[CLRHM]”
30 FOR Y = 1 TO 10:PRINT “[CRSR-D]”;:NEXT Y
40 PRINT “A”;
50 FOR P = 1 TO 500:NEXT P
60 PRINT “[CRSR-L] ”
```

When you enter Vanish make sure that you leave a space after [CRSR-L] in line 60. The space is the secret of this amazing trick. By the way, you can replace the space with **CHR\$(32)** (Remember the Hotski-Totski Charts?) if you want to make sure it’s included in the program. Here’s how line 60 would look:

```
60 PRINT “[CRSR-L]”;CHR$(32)
```

Lines 20 through 40 position a letter A ten spaces down from the top of the screen. Then, line 50 adds suspense to the trick by pausing for a few seconds.

DR. WACKO PRESENTS COMMODORE BASIC

Line 60 performs the vanish by moving the cursor back (to the left) one space and printing a blank space over the letter A. Easy, no?

Stand back! Here's my next bit of cursory magic. Yes folks, the great Dr. C. Wacko will now perform one of his more mystical feats. The astonishing Movable A! With nothing up my sleeves (except my arms), I will magically whisk the letter A across the screen. Type in and run Movable A to witness this heretofore impossible bit of prestidigitation.

Movable A

```
10 REM ** MOVABLE A **
20 PRINT "[CLRHM]
30 FOR Y = 1 TO 10:PRINT"[CRSR-D]";:NEXT Y
40 FOR X = 1 TO 39:PRINT "A";
50 FOR P = 1 TO 100:NEXT P
60 PRINT "[CRSR-L] ";:NEXT X
```

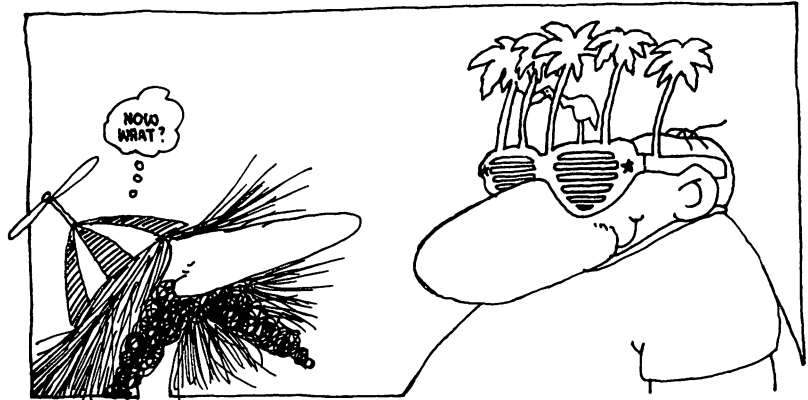
Again, make sure that you include a blank space after [CRSR-L] in line 60.

Lines 20 and 30 start the cursor off ten lines down from the top of the screen.

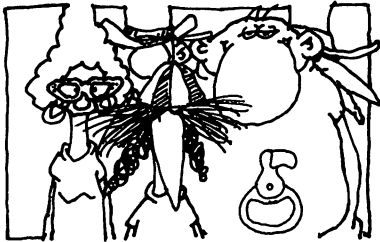
All the magical movement takes place in lines 40 through 60. First, in line 40, the letter A is printed, movement is slowed down in line 50, and then in line 60 the letter A vanishes. This loop continues from 1 to 39, mystically moving the letter A across the screen.

I've revealed these two tricks because they play a BIG part in the Camel Racetrack program that you'll be going to soon. But, for now, let's add color to our characters.

Colorful Characters



I'm surrounded by a band of weird and colorful characters. And, if you press the right keys, you can be too.



Do you still remember the Bijou Billboard program at the beginning of this chapter? You don't? Well, not to worry, here's a three-line program that should jog your memory.

```
10 PRINT "[CLRHM]":POKE 53280,0:POKE  
53281,0  
20 FOR X = 1 TO 12:PRINT "[CRSR-D]";NEXT X  
30 PRINT TAB(12)"[RVS ON][CTRL + 3] W  
[CTRL + 8] A [CTRL + 6] C [CTRL + 5] K  
[CTRL + 4] O [CTRL + 7] "
```

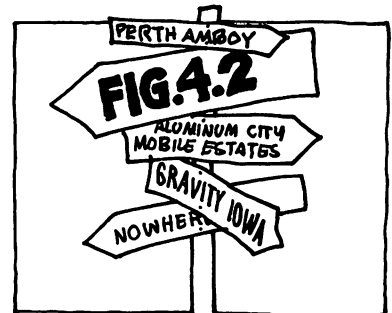
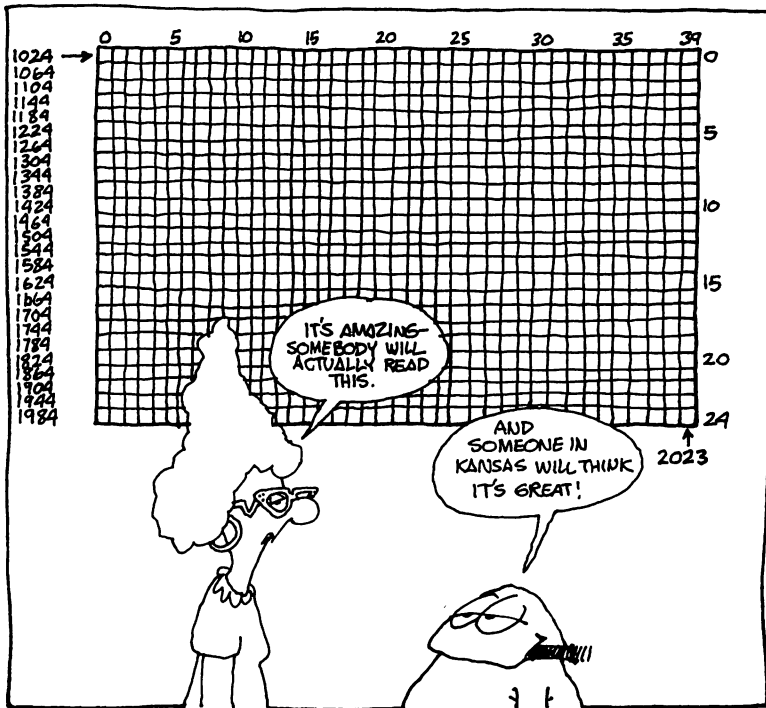
A multicolored Wacko on a black screen! Two things are going on here. First, in line 10 I used the two POKEs I introduced in Oasis 5 to color the border and background black. If you'd like to know how to get the most from these two powerful POKEs just flip back to Sneaky Peek's Appendix on page 247.

Next, in line 30, I printed my name in brilliant colors by typing a quotation mark, then turned reverse printing on by pressing the CTRL key plus the RVS ON key, and used the CTRL key and number keys to imbed color in the print statement.

DR. WACKO PRESENTS COMMODORE BASIC

Now that you know how to put colorful characters where you want them and, magically make them disappear and move about, it's time to introduce you to a snazzier way of doing very much the same thing, or what is called "Screen Graphics."

Snazzy Screen Graphics



Take a look at the Screen Map, Figure 4.2. It looks almost like the drawing on page 160 that shows how your screen display is laid out. But, I've added some very important numbers down the left side, and at the top-right and bottom-right of this illustration.

Each of the one thousand holes now has the familiar two-number coordinate, plus its own special number. For example, coordinate 0,0 has been assigned the number 1024, and coordinate 39,24, at the lower right-hand corner of the chart, has been assigned the number 2023.

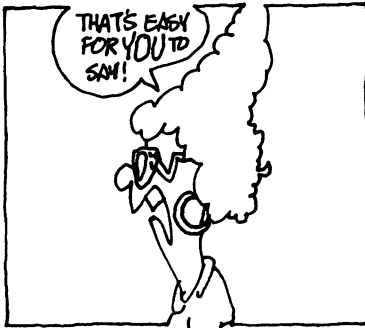
Graphics Power

You use these special numbers to POKE shapes and characters anywhere you'd like on the screen! These numbers work with the *screen codes* listed in Appendix E.

Here's how to use one of these special numbers and a screen code to place the letter A near the center of your screen, at coordinate 20,10.

```
10 POKE 52381,0:POKE 1424 + 21,1
```

Run this one-liner and a letter A will appear near the center of your screen. Amazing! And a lot faster and easier than the cursory method. Here's how it works.



Take a look at the chart. The number 1424 is on row 10. From here, to place the A in column 20, I've added 21 to 1424 (remember, the column numbers start with 0). Next, I POKEd 1424 + 21 with 1, the number listed next to A in the Screen Code chart. That's all there is to it.

I showed the addition of 1424 + 21 in the line 10 to make it easier for you to follow, but you can shorten things up by doing the addition beforehand and writing the program like this:

```
10 POKE 52381,0:POKE 1445,1
```

You can place any character listed on the Screen Code chart anywhere you'd like by using the special number assigned to each location and POKEing it with a Screen Code number. Here's a short program that replaces the 1 in the above example with each of characters available.

```
5 POKE 52381,0
10 FOR X = 0 TO 127
20 POKE 1445,X
30 FOR P = 1 TO 100:NEXT P
40 NEXT X
```

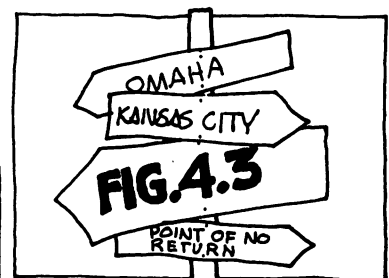
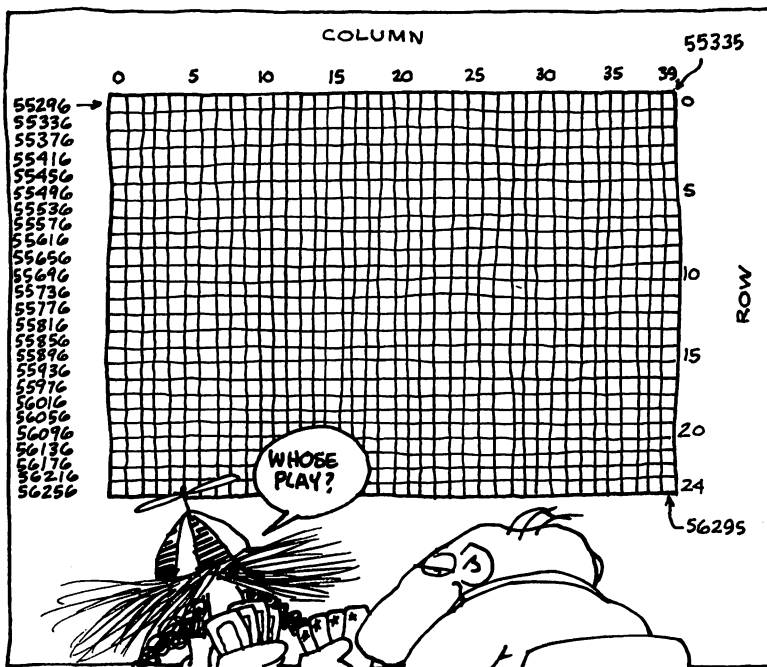
DR. WACKO PRESENTS COMMODORE BASIC

Here's a short routine that draws a thatched line across the screen to show you the versatility of this snazzy display method.

```
10 POKE 52381,0:FOR X = 1424 TO  
1424 + 39:POKE X,102:NEXT X
```

Colorful Screen Displays

Using screen graphics POKE techniques to print characters and draw lines on your screen is pretty snazzy, but probably dull, unless those characters and lines are in vibrant color! Adding color to your display is easy. It just requires another chart, and sixteen color numbers.



Take a look at the Color Map, Figure 4.3. Yup, it's just like the Screen Map, but it's got different numbers down the left-hand side, and in both right-hand corners. No sweat. After you've used the special numbers from the Screen Map to put a character or characters on your screen, you use the numbers from the Color Map to add color to each character.

Graphics Power

To choose a color, you just POKE each of these special numbers with a color number from 0 to 15. Before I move on, here's a list of the sixteen available colors and their color numbers.

Colors and Color Numbers

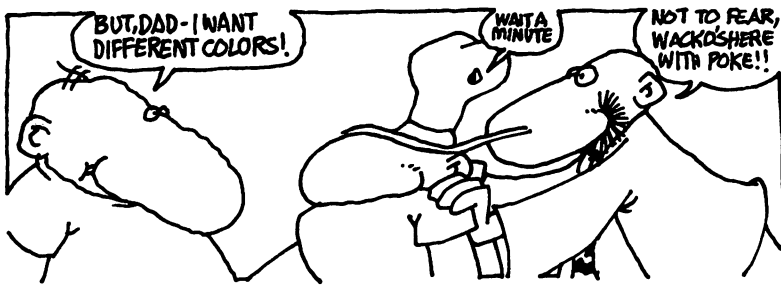
| | |
|----------|----------------|
| 0 BLACK | 8 ORANGE |
| 1 WHITE | 9 BROWN |
| 2 RED | 10 LIGHT RED |
| 3 CYAN | 11 GRAY 1 |
| 4 PURPLE | 12 GRAY 2 |
| 5 GREEN | 13 LIGHT GREEN |
| 6 BLUE | 14 LIGHT BLUE |
| 7 YELLOW | 15 GRAY 3 |

Here are a couple of examples that show how easy it is to use the Color Map and color numbers to liven up your display.



5 POKE 52381,0
10 POKE 1424 + 21,1
20 POKE 55696 + 21,4

That was easy. But the result was an obnoxious, pugnacious-purple A. Eeyuch!



POKE 55696 + 21 in line 20 positions the color directly above the letter A at column 20, row 10. Then, the A is colored purple by POKEing 55696 + 21 with 4. Just change the 4 to any number between 0 and 15 to change the A's color.

DR. WACKO PRESENTS COMMODORE BASIC

To perform an instant vanish, just change the 4 to a 0 (BLACK) and the A will disappear. It becomes the same color as the background!

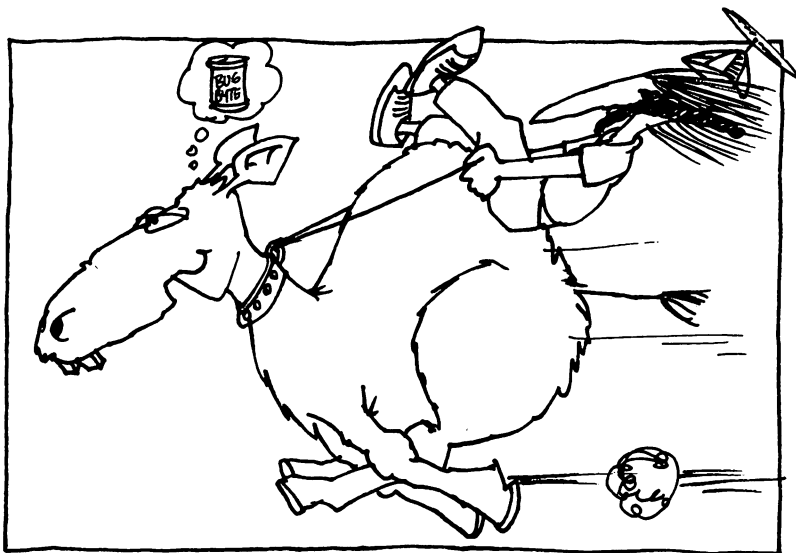
Coloring a whole row of characters is just as easy as coloring one character.

```
5 POKE 52381,0
10 FOR X = 1424 TO 1424 + 39:POKE
  X,102:NEXT X
20 FOR X = 55696 TO 55696 + 39:POKE
  X,4:NEXT X
```

A thatched *purple* line!

Hello, Camel Racing Fans!

Now that you've seen all of your Commodore's graphics tricks, it's time to put some shekles in your pocket and stroll over to Clyde's Camel Race Track.



Graphics Power

Before the race begins we're going to get the camels on the move, build the track, then set up a finish line. All the things you'd expect of a real race track.

Get Those Camels Moving

We're going to use the pi symbol (right below the up-arrow at the right of your keyboard) to represent each camel, and we'll use cursory movement to get each of these ornery creatures on the move (Clyde doesn't like to be poked).

I'm going to present each part of this fabulous program bit-by-bit. Movin' Camels comes first, then we'll add Lanes, an option that lets you determine the number of laps to be raced, and finally a finish line and a winner's announcement.

The line numbers may seem out of wack at first, but the program grows as we add to it. Paleeze use the line numbers I've used. It all fits together before the race starts. Trust me!

Just enter Movin' Camels, run it, and then I'll explain how it gets those camels on the move.



DR: WACKO PRESENTS COMMODORE BASIC

Movin' Camels

```
5 REM *** CLYDE's CAMEL RACE TRACK
  ***
10 PRINT "[CLRHM]":POKE 53281,0
20 B = 2:D = 0:REM 'D = 0' IS USED LATER
100 FOR S = 1 TO 5:X(S) = 0:NEXT S:FOR S = 1
    TO 5:Y(S) = B:B = B + 3:NEXT S
110 M = INT(RND(0)*5) + 1
120 PRINT "[HOME]"
130 Y = Y(M):X = X(M)
140 FOR A = 1 TO Y:PRINT"[CRSR-D]";:NEXT A
150 FOR A = 1 TO X:PRINT "[CRSR-R]";:NEXT A
160 PRINT "[CRSR-L]";CHR$(32);"π"
170 X(M) = X(M) + 1
180 IF X(M) = 39 THEN PRINT
    "[CRSR-L][CRSR-L]";CHR$(32);:X(M) = 0
190 REM *** RESERVED FOR FUTURE
    ADDITION
200 GOTO 110
```

After you run *Movin' Camels* you'll see five white camels race across your screen. When they get to the right edge of the screen they loop around to the left side and continue on their way. Not a bad start. Those camels are really movin' and groovin', and here's how it's done.

Line 10 clears the screen, sends the cursor home, and colors the racetrack black. Then, in line 20, variable **B** is set to 2. You'll see how **B** is used in a second.

In line 100 the elements in the array $X(1)$ through $X(5)$ are set to zero (0), and elements in array $Y(1)$ through $Y(5)$ are set to 2, 5, 8, 11, and 14 using the variable **B** as a counter. These are the X, Y starting positions for each of the five camels. Here's what each element of each array equals when line 100 is finished looping around:

Graphics Power

| | |
|----------|-----------|
| X(1) = 0 | Y(1) = 2 |
| X(2) = 0 | Y(2) = 5 |
| X(3) = 0 | Y(3) = 8 |
| X(4) = 0 | Y(4) = 11 |
| X(5) = 0 | Y(5) = 14 |

Now that the starting position of each camel is set, the random generator in line 110 randomly selects one of the five camels for movement.

Because we're going to use cursory movement, line 120 sends the cursor home.

In line 130 Y is made equal to Y(M) and X is made equal to X(M), then lines 140 and 150 move the selected camel's cursor down (Y) and to the right (X).

The actual movement takes place in line 160. First the cursor is moved back one space to erase a previous camel (just like the Vanish program), then a space (CHR\$(32)) followed by the camel is printed, moving it one column to the right.

Once the camel has moved, the array **X(M)** in line 170 is updated so we can keep track of each camel's position on the screen.

Line 180 was added to erase the camel from the right side of the screen and print it on the left after each lap. When a camel reaches the left side of the screen **X(M) = 39** a couple of left cursors are printed, a space is inserted, and **X(M)** is set to 0, the right side of the screen. All this fancy programming is needed to prevent the camel from dropping down one line each time it wraps around the screen.

Finally, line 200 sends the program back to line 110 where the next camel is randomly selected.

DR. WACKO PRESENTS COMMODORE BASIC

All this was pretty nifty. But I wanted to add color to the display. I felt that each camel should be identified with its own unique color. The solution to this was fairly straightforward. Just modify line 160 like I've done below, run the program again, and you'll see five differently colored camels scamper across your screen.

```
160 PRINT "[CRSR-L]";CHR$(32);  
      CHR$((M) + 148);"π"
```

The statement **CHR\$((M) + 148)** does all the colorful work. If you check the Hotski-Totski charts you'll see that codes 149 through 153 represent the colors brown, light red, grey 1, grey 2, and light green. So, if the random generator selects camel 1 for movement ($M = 1$), its color will be brown, or if camel 5 is selected ($M = 5$) its color will be light green. This way each of the five camels has its own unique color. Pretty neat, no?



With five colorful camels scampering across the screen, it's time to put them into lanes so they don't bump into each other. Adding six lines to your program does the trick.

Lanes

```
40 REM *** DRAW LANES ***  
50 FOR X = 1064 TO 1703:POKE X,82:NEXT X  
60 FOR X = 55376 + D TO 55455 + D:POKE  
  X,32:NEXT X  
70 IF D > 440 THEN GOTO 100  
80 D = D + 120:GOTO 60  
90 REM ***
```

If you add line 20 (**B = 2:D = 0**), you can run this short program just as it is to see how the lanes are formed, then insert it into the Camel Race Track program and watch how they work together.

Graphics Power

I've used screen graphics techniques to POKE the lanes onto the screen. Refer to the Screen and Color Maps on pages 166 and 168 as I explain how the lanes are drawn.

First, in line 50 lanes are drawn down the screen from row 1 to row 16. Then in line 60 the lanes in rows 3 and 4 are erased by POKEing each location with 32, a blank space. Next the program goes down to line 80 where D is increased by 120 (three rows) each cycle. The value of D is added in line 60, and these three lanes are erased. This continues until, in line 70, D is greater than 440. When this happens the lanes are all set up, the program moves to line 100, and the race begins.



Those camels are really running now! But, there's got to be a way to determine the winner, or there's no race.



These final additions and modifications to the program add a lap counter, determine and announce the winner, and let you race again.

Just follow my instructions to modify your program.

1. Add line 30.

```
30 PRINT:INPUT"[CTRL-8]HOW MANY LAPS";  
L:L=(30*L)+(5*L)+5*(L-1):  
PRINT"[CLRHM]"
```

2. Change line 100 to read:

```
100 FOR S = 1 TO 5:X(S) = 0:C(S) = 0:NEXT S:FOR  
S = 1 TO 5:Y(S) = B:B = B + 3:NEXT S
```

3. Change lines 170 and 180 to read:

```
170 X(M) = X(M) + 1:C(M) = C(M) + 1  
180 IF X(M) = 39 THEN PRINT  
"[CRSR-L][CRSR-L]";CHR$(32);:X(M) = 0:  
C(M) = C(M) + 1
```

DR. WACKO PRESENTS COMMODORE BASIC

4. Add line 190.

```
190 IF C(M) = L THEN PRINT  
    "[CRSR-L][CRSR-L][CTRL + 2]*":GOTO 210
```

5. Add lines 210 through 280.

```
210 PRINT "[HOME]"  
220 FOR X = 1 TO 18:PRINT"[CRSR-D]";:NEXT X  
230 PRINT TAB(11)"[CMDR + 7]YES, RACING  
    FANS."  
240 PRINT:PRINT TAB(9)"THE WINNER IS  
    CAMEL";M;"!"  
250 PRINT:PRINT TAB(8)"[CMDR + 1]PRESS  
    [CMDR + 6]F7[CMDR + 1] TO RACE AGAIN"  
260 GET A$: IF A$ = "" THEN GOTO 260  
270 IF ASC(A$) < > 136 THEN GOTO 260  
280 GOTO 10
```

Quite a few changes. But that does it. Now you're ready for some real racing excitement. Here's what all these additions and changes do.

First, I added line 30 to allow you to select the number of laps each race would run. I decided that the race would finish when the lead camel crossed column 35 on the screen. All that fancy math at the end of line 30 ensures that the race will end at the same spot on the screen each time it's run. Remember, the screen is 40 columns across, and without the fancy math the race would end at a different place each time. Take out the formulas and you'll see what I mean, then work through the formulas and you'll see how they solve the problem.

In lines 100 and 170 I've added $C(S) = 0$ and $C(S) = C(S) + 1$. The array $C(S)$ is used to keep track of how many spaces each camel has moved so the running total can be compared to the lap counter to determine the winner.

Graphics Power

In line 180, $C(S) = C(S) + 1$ has been added to account for the one space increase that occurs when the camel goes from the right side of the screen to the left.

The winner is identified in line 190. Here, if the counter ($C(M)$) is equal to laps (L) a white asterisk (*) is printed on the screen and the program bops down to line 210.

In lines 210 through 250 the winner is announced, and you are asked to press the f7 key to race again.

Line 260 waits for you to press a key. If no key is pressed, A\$ equals nothing and the program loops back to 260 to continue waiting. Don't put any space between the two quotes after A\$. Two quotation marks, without a space, signify nothing.

When you press any key, the program goes down to line 270 which checks to see if you've pressed the f7 ($CHR\$(136)$) key. If you haven't, it goes back up to line 260 to wait for you to press another key. If you have pressed f7 the program goes to line 10 to begin again.

Clyde's Masterpiece

You've made quite a few additions and modifications to the original bit of programming, and things may be a little disarrayed. Here's Clyde's entire masterpiece to help you see how all those pieces fit together.



DR. WACKO PRESENTS COMMODORE BASIC

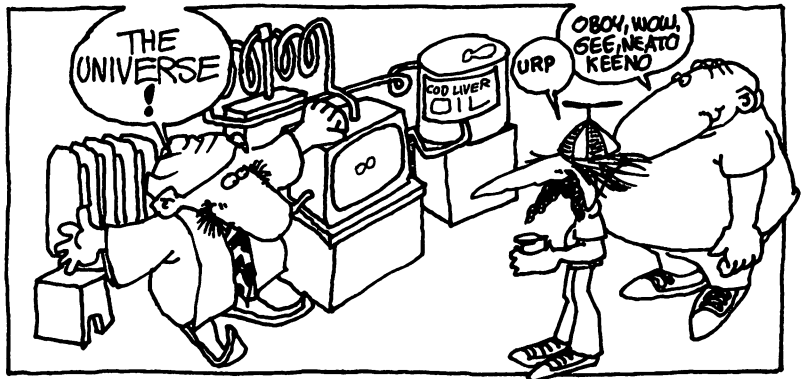
Clyde's Camel Race Track

```
5 REM *** CLYDE'S CAMEL RACE
  TRACK ***
10 PRINT "[CLRHM]":POKE 53281,0
20 B = 2:D = 0
30 PRINT:INPUT"[CTRL-8]HOW MANY LAPS";
  L:L = (30*L) + (5*L) + 5*(L-1):
  PRINT"[CLRHM]"
40 REM *** DRAW LANES ***
50 FOR X = 1064 TO 1703:POKE X,82:NEXT X
60 FOR X = 55376 + D TO 55455 + D:POKE
  X,32:NEXT X
70 IF D > 440 THEN GOTO 100
80 D = D + 120:GOTO 60
90 REM ***
100 FOR S = 1 TO 5:X(S) = 0:C(S) = 0:NEXT S:FOR
  S = 1 TO 5:Y(S) = B:B = B + 3:NEXT S
110 M = INT(RND(0)*5) + 1
120 PRINT "[HOME]"
130 Y = Y(M):X = X(M)
140 FOR A = 1 TO Y:PRINT"[CRSR-D]";NEXT A
150 FOR A = 1 TO X:PRINT "[CRSR-R]";NEXT A
160 PRINT "[CRSR-L]";CHR$(32);
  CHR$((M) + 148);"π"
170 X(M) = X(M) + 1:C(M) = C(M) + 1
180 IF X(M) = 39 THEN PRINT
  "[CRSR-L][CRSR-L]";CHR$(32);X(M) = 0:
  C(M) = C(M) + 1
190 IF C(M) = L THEN PRINT
  "[CRSR-L][CRSR-L][CTRL + 2]*":GOTO 210
200 GOTO 110
210 PRINT "[HOME]"
220 FOR X = 1 TO 18:PRINT"[CRSR-D]";NEXT X
230 PRINT TAB(11)"[CMDR + 7]YES, RACING
  FANS."
240 PRINT:PRINT TAB(9)"THE WINNER IS
  CAMEL";M;"!"
```

Graphics Power

```
250 PRINT:PRINT TAB(8)"[CMDR + 1]PRESS  
[CMDR + 6]F7[CMDR + 1] TO RACE AGAIN"  
260 GET A$: IF A$ = "" THEN GOTO 260  
270 IF ASC(A$) < > 136 THEN GOTO 260  
280 GOTO 10
```

Wacko's Infamous Model of the Universe!



And now, here's the wondrous program you've been waiting for. I've titled it "Universe." It was developed at the Wacko Institute of Alchemy many years ago to check out the Big Bong Theory and demonstrate the improbability of life occurring in a random environment.

Here's the experiment you'll watch on your screen after you enter and run UNIVERSE.

Two randomly controlled orange and light green balls bounce around on Commodore 64's two-dimensional screen. Each ball is controlled by its own random generator.

A condition, or "Law of Life" is set. In this creative experiment, "life" occurs when both balls arrive at adjacent positions at the same time. When they arrive, the orange ball must land on the orange marker, and the light green ball must land on the light green marker.



DR. WACKO PRESENTS COMMODORE BASIC

Some Real Heavyweight Stuff

Since this is a simple model (for simple wackos), Petunia told me that the probability of “life” occurring can be calculated by using a simple binomial distribution method. (Easy for her to say. Wheew!)

Her calculations made two assumptions.

They assumed that the random movement of each ball will be *truly random*. Because *true random behavior* is real hard to simulate on a computer, her calculations might be off a little bit.

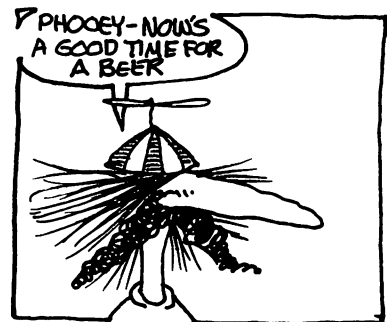
Her calculations also assumed that each ball will behave independently — Do its own thing, so to speak.

Petunia’s binomial distribution calculations for this model show that there is a probability of 2,470,864.5:1 that “life” will occur in each cycle. This means that “life” should statistically occur after 2,470,864.5 cycles. But, “life” may occur during the first cycle, or “life” may never occur at all. Uh oh!

As a matter of scientific interest, if each cycle’s duration was 1 second, life “should” occur in 28 days!

One reason for baffling you with such alchemy is to demonstrate how versatile your computer really is. Modeling, from stress analysis (I’m usually under a lot of stress) to a model of the universe, is all within the realm of possibility. That chunk of machinery in front of you is very powerful, if you use your imagination!

There’s another reason I’m showing you this great program; it uses many of the elements that you have learned in this chapter. It’s also a great example of how to POKE colored characters on the screen. So without any further “pad dew” (water on a camel’s foot), here’s Universe!



Graphics Power

Universe

```
10 REM ** UNIVERSE **
20 T = 1
30 PRINT "[CLRHM]":POKE 53281,0:POKE
  53280,6
40 FOR X = 1904 TO 2023:POKE X,102:NEXT X
50 FOR X = 56176 TO 56295:POKE X,7:NEXT X
60 A = INT(RND(0)*880) + 1024
70 B = INT(RND(0)*880) + 1024
80 A1 = A + 54272:B1 = B + 54272
90 POKE 1444,91:POKE 1445,91:POKE
  55716,8:POKE 55717,13
100 POKE A,81:POKE A1,8
110 POKE B,81:POKE B1,13
120 POKE A,32:POKE B,32
130 T = T + 1
140 GOSUB 160
150 GOTO 60
160 PRINT "[HOME]";:FOR X = 1 TO
  23:PRINT "[CRSR-D]";:NEXT X
170 PRINT TAB(2)"[CTRL + 2]CYCLES";
  T;TAB(24)"[CMDR + 1]A";A;SPC(2)
  "[CMDR + 6]B";B;
180 IF A = 1444 AND B = 1445 THEN GOTO 200
190 RETURN
200 PRINT "[HOME]";:FOR X = 1 TO
  10:PRINT "[CRSR-D]";:NEXT X
210 PRINT TAB(18)"[CMDR + 3]LIFE!!":END
```



The Universe Explained (Almost)

Here's a quick journey through Universe.

T, which is used to keep track of cycles, is set to 1 in line 20.

Line 30 clears the screen, sends the cursor home, and colors the background black and the border blue.

DR. WACKO PRESENTS COMMODORE BASIC

Lines 40 and 50 POKE a yellow display area at the bottom of the screen.

The random generators in lines 60 are used to move each ball about the screen. They're set to spit out numbers from 1024 (the upper left corner) to $1024 + 880$, at the left just above the yellow display screen.

In line 80 **A1** is set to the color of ball A, and **B1** tracks ball B's color.

Line 90 places two crosses, one orange and the other light green, near the center of the screen. These two crosses are at the position where "life" will occur if both balls land on them simultaneously.

Lines 100 and 110 put each ball, with its appropriate color, on the screen.

Line 120 erases each ball by putting a blank space (32) at each ball's position.

The cycle counter is updated by 1 in line 130, then in line 140, the program zips to the display routine beginning on line 160.

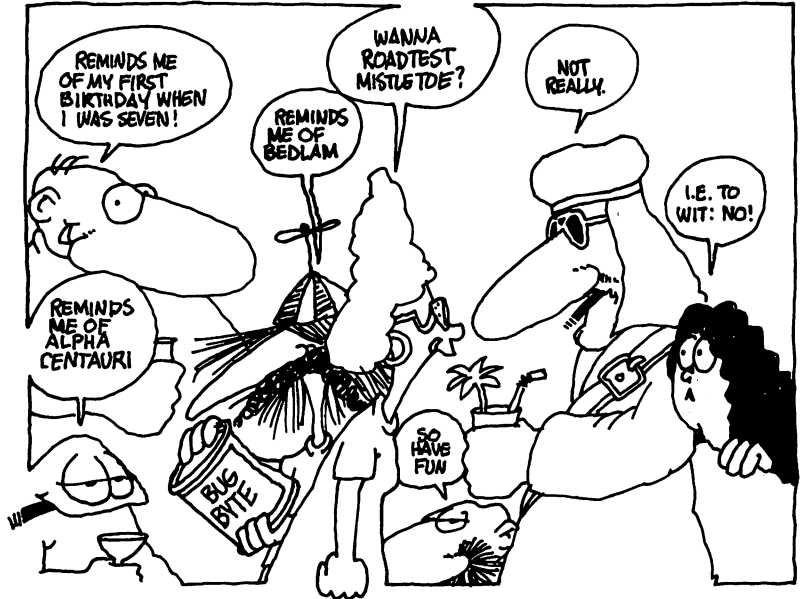
Lines 160 and 170 display the number of cycles and the current location of each ball.

In line 180, if the balls arrive at the center of the screen at the same time, the program goes down to line 200, where "LIFE!!" is displayed on the screen.

CONGRATULATIONS! Pat yourself on the back, have a wild and crazy party! You understand and can now design programs like this one. You've come a long way and know how to make exciting things happen on your Commodore's screen.

Graphics Power

Are you ready to add sound effects to the Camel Race Track and Universe programs, and in general make a lot of noise with your computer? Then, put in your ear-plugs, close the door, and turn the page!



5
Desert ZOUNDS!



Close your eyes and do a little desert dreaming. If you listen closely you might hear the hot wind of the Sahara blowing grains of sand across the dunes, the gentle rustling of palm leaves in the evening breeze, or the eerie sound of flutes, lambskin drums, and a dancer's ringing cymbals.

Open your eyes! (Ooops! I forgot that you can't read this with your eyes closed. Oh, well.) If you heard all those sounds when you closed your eyes you've really got a great imagination. All I ever hear are Clyde's grunts and Junior's questions.

Now that your eyes are open (I hope), and you've returned to the "real" world, close the door, put cotton in your ears, and get set for some real audio excitement as we delve into the wild world of Commodore 64 sound!

Your Commodore 64 can sing, or meow, in as many as three voices. You can use each voice solo, or you can be creative and combine a few, or all three to achieve remarkable sound effects, and some beautiful music.



Desert ZOUNDS!

But the best thing about Commodore 64 sound is that it lets you dig in and POKE around in your computer. This remarkable chapter is filled with charts, graphs, new terminology and weird cartoons — all designed to show you how to make that quiet computer of yours stand up and be heard.

Let's Go For It!

I've designed a nifty program called Wacko's Sound Machine that gets your Commodore off and singing. It's loaded with musical programming that may be new to you now. But have no fear (Wacko's here), by the time you've read this chapter you'll not only understand how this fabulous program works, but you'll be designing better ones! (What am I saying?)

So, take a chance and go for it! Type in Wacko's Sound Machine. I'll go skateboarding in the sand with Junior for a while. When you're all set, call me and I'll zip back to show you how to make noise with your Commodore.

Wacko's Sound Machine

```
10 REM *** WACKO'S SOUND MACHINE ***
20 CLR:PRINT"[CLRHM]":PRINT:PRINT
30 INPUT "(T)RIANGLE, (S)AWTOOTH OR
   (N)OISE";S$
40 IF S$ = "T" THEN W = 17
50 IF S$ = "S" THEN W = 33
60 IF S$ = "N" THEN W = 129
70 IF W = 0 THEN GOTO 20
80 PRINT:PRINT"SELECT A NOTE: EITHER
   C,D,E,F,G,A,B OR X"
90 GET A$:IF A$ = "" THEN GOTO 90
100 IF A$ = "C" THEN NH = 34:NL = 75:GOTO
    200
110 IF A$ = "D" THEN NH = 38:NL = 126:GOTO
    200
120 IF A$ = "E" THEN NH = 43:NL = 52:GOTO
    200
```

DR. WACKO PRESENTS COMMODORE BASIC

```
130 IF A$ = "F" THEN NH = 45:NL = 198:GOTO
    200
140 IF A$ = "G" THEN NH = 51:NL = 97:GOTO
    200
150 IF A$ = "A" THEN NH = 57:NL = 172:GOTO
    200
160 IF A$ = "B" THEN NH = 64:NL = 188:GOTO
    200
170 IF A$ = "X" THEN NH = 68:NL = 149:GOTO
    200
180 GOTO 20
190 REM ***
200 REM *** ATTACK/DECAY SUSTAIN/
    RELEASE UTILITY
210 REM ***
220 PRINT "[CLRHM]":PRINT:PRINT
230 FOR X = 54272 TO 54296:POKE X,0:NEXT
    X:REM ** CLEAR OUT SOUND REGISTERS
240 INPUT "ENTER ATTACK";A:A = A* 16
250 INPUT "ENTER DECAY";D
260 AD = A + D
270 INPUT "ENTER SUSTAIN";S:S = S* 16
280 INPUT "ENTER RELEASE";R
290 SR = S + R
300 IF SR > 255 OR AD > 255 THEN GOTO 20
310 POKE 54296,15:REM ** VOLUME SET TO
    MAXIMUM
320 H = 54273:L = 54272:A = 54277:S = 54278:
    REM ** ASSIGN VARIABLE NAMES
330 POKE A,AD:POKE S,SR:REM **
    ATTACK/DECAY & SUSTAIN/RELEASE
340 POKE H,NH:POKE L,NL:REM HIGH & LOW
    PART OF NOTE
350 POKE 54276,W:REM ** WAVE FORM
360 FOR T = 1 TO 1000:NEXT T:REM MAX
    DURATION OF EACH NOTE
370 POKE 54276,W-1:REM ** TURN OFF
    WAVE FORM
380 GOTO 20
```

Desert ZOUNDS!

Wheew! I'll bet your fingertips are tired after typing in that one! Just to be on the safe side, save this humongus program to disk or cassette, and we'll get started.

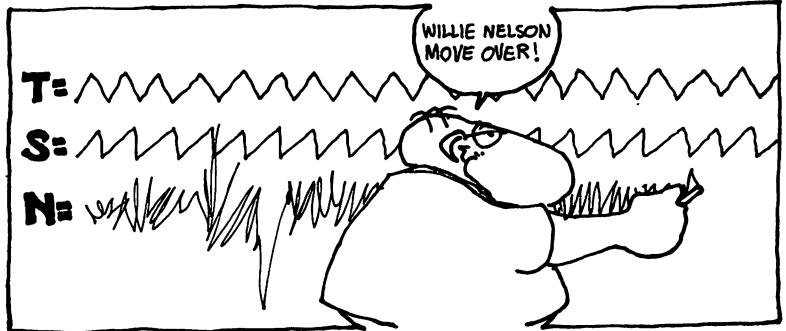
All set? OK, turn up your monitor's volumn and run my Sound Machine.

The first thing you'll see on your screen is:

(T)RIANGLE, (S)AWTOOTH OR (N)OISE ?

The Type of Sound

Your computer is asking you to enter a *T*, *S*, or *N*. These three options select the *type* of sound you'll soon hear. If you could see the shape of each type of sound your Commodore makes you'd go bonkers. The *triangle*



sound looks like a one-sided pyramid (also known as a triangle), the *sawtooth* sound looks like the jagged edge of a saw (also know as a sawtooth), and the *noise* sound looks like bad TV reception. But more important than the way each type of sound looks, is how each type sounds!

If you press *T*, for *triangle*, the tone sounds pure and crisp. Just perfect for simulating a piano, flute or ex-eye-la-
phone.

If you press *S*, for *sawtooth*, the sound has a raspy edge to it, just like a saw's blade.

DR. WACKO PRESENTS COMMODORE BASIC

If you press *N*, for *noise*, that's just what you'll hear! It'll sound as if you've tuned your TV to a nonexistent channel — lots of static!

Since you know what to expect, type *T* and press RETURN to select a pure tone.

Oops, your screen is displaying more questions!

SELECT A NOTE: EITHER C,D,E,F,G,A,B,OR X

This display is pretty self-explanatory for musical types like Slow Poke, but it took me a little while to figure out.

If you press the *C* key, a middle *C* note is selected. (If you've never heard a "middle *C*" note before, don't worry, you will in a few moments.) Press *D* and your Commodore will play a *D* note. Everything is copacetic until you press *X*. There is no *X* note! What's going on here? Simple, since there's only one *C* key on the computer I've used the *X* key to represent high *C*.

With that all squared away, press *C* and we'll move on.

Uh oh, your screen's getting really aggressive! It's asking you to:

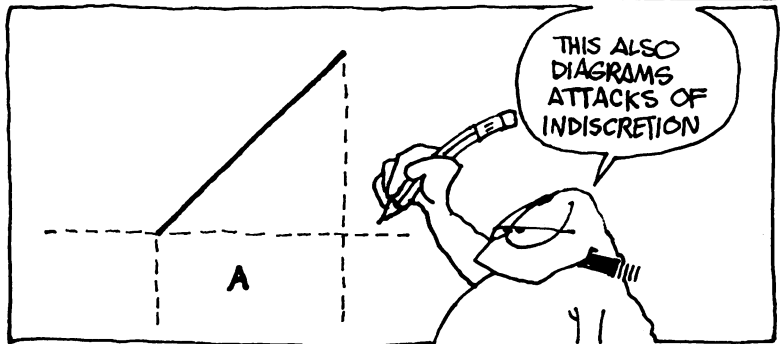
ENTER ATTACK ?

It's an ATTACK!

Attack is the time it takes a tone to go from off to maximum volume. You can enter any number from 0 to 15 in response to this aggressive question. If you enter 0 it takes almost no time at all for the tone to reach its maximum volume. If you enter 15, it takes a looong time for the tone to reach maximum volume. The lower the number, the *faster* the tone reaches maximum volume! Grover (who's not at all aggressive) drew this chart so you can see what this *attack* stuff is all about.



Desert ZOUNDS!



Time increases from left to right across the bottom of the Grover's attack chart, and volume gets higher as it moves up the chart. Grover's drawn a *medium attack*. Just as if you had entered an 8 in response to "ENTER ATTACK."

Now that you know all about *attack*, enter a 0 to get that tone to maximum fast, press RETURN, and your screen should now be displaying this weird message:

ENTER DECAY ?

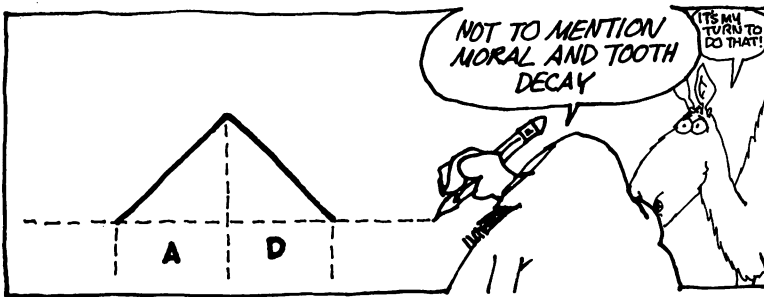
Desert DECAY

Most things quickly decay out here in the desert. Especially sound. It gets muffled by all that sand. When I yell at Junior my shout seems to instantly decay. Maybe that's why he still hasn't cleaned up his room!



Decay is the length of time it takes a tone to go from maximum volume to off, or to an intermediate (sustain) level. You can enter any number from 0 to 15 in response to this decaying question. If you enter 0, it takes almost no time at all for the tone to turn off or reach it's sustaining volume. If you enter 15, it takes a looong time for the tone to turn off or reach a sustaining volume. The lower the number, the *faster* the tone goes down and gets lost in the sand! Clyde, who's real familiar with sand, drew this chart so you can witness for yourself this decaying phenomenon.

DR. WACKO PRESENTS COMMODORE BASIC



This drawing is an extension of the "Attack Chart." Time increases from left to right across the bottom of Clyde's decay chart, and volume gets higher as it moves up the chart. Clyde's hoofed in a *slow decay*. Just as if you had entered 15 in response to "ENTER DECAY."

Now that you know why Junior doesn't hear me when I shout at him, **enter a 10 to slow down the rate of the tone's decay**, press RETURN, and your screen should now look like this.

ENTER SUSTAIN ?

SUSTAINing Sahara Sustenance

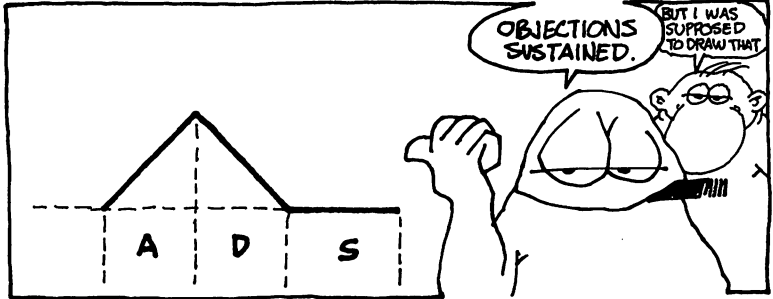
You've got to have sustenance to sustain yourself at a decent level out here in the Sahara.

Sustain in musical terms, is the volume level that a note can *decay* to after it's reached its maximum. The *sustain* level, if used, can be any level higher than 0 or lower than 15. You can enter any number from 0 to 15 in response to this question. If you enter 0 there is no sustain level, and the note just *decays* to 0. If you enter 15 the note doesn't *decay* at all, but stays at maximum volume. If you enter another number (1 - 14), the note *decays* to that level and stays there until it's *released*.

Tones can be made by using just *attack* and *decay*, but using Commodore's *sustain* feature adds a touch of class to your notes as you'll soon discover after you futz around with my Sound Machine.

Desert ZOUNDS!

Junior, who's the most level-headed wacko in our group, scribbled this chart so you can see what a *sustain* level is, and how it works with *attack* and *decay*.



Junior's hieroglyphics show how a note starts at 0 volume, *attacks* (rises) to its maximum level, 15, *decays* at a moderate rate, and is *sustained* at a medium level, just as if you had entered an 8 in response to "ENTER SUSTAIN."

Now that you've deciphered Junior's scribbling, **enter a 0 and press RETURN**. There's no *sustain* level. The note just *decays* from a maximum volume of 15, down to 0. For your first audible experiment, you'll be listening to a simple tone, made up of just *attack* and *decay*.

Now, your screen should be displaying one final question:

ENTER RELEASE ?

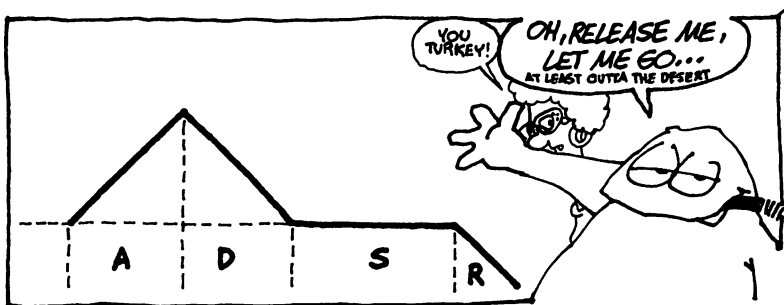
"RELEASE Me, You Cad!"



That's what Ms. Peeky yelled during her last encounter with Lawrence of Newark. But I don't think she really meant it.

DR. WACKO PRESENTS COMMODORE BASIC

Release is the length of time a note is held at its *sustaining* volume level before being dropped to zero. You can enter any number from 0 to 15 in response to this pleading question. If you enter 0 the note isn't *sustained*, it's *released* immediately. If you enter 15, the tone stays at its *sustaining* volume for a looong time before being *released*. After much persuasion, and flattery, Ms. Peeky agreed to draw this chart so you can get a close up of *release* in action.



Ms. Peeky's pencil sketch shows a note starting at 0 volume, *attacking* (rising) to its maximum level, 15, *decaying* at a moderate rate, *sustained* at a medium level, for a moderate amount of time before being *released* — just as if you had entered an 8 in response to "ENTER RELEASE."

Now that you've been shocked by Ms. Peeky's revealing sketch, **enter a 0 and press RETURN**. There's no *release* level. The note just *decays* from a maximum volume of 15, down to 0. As I mentioned before, you'll hear a simple tone, made up of just *attack* and *decay*.

Well, how did it sound?

Before you start experimenting with the Sound Machine, here's a short refresher list of all the sound terms you've just learned:

Triangle: A pure and crisp sound. Its waveshape looks like a triangle.

Desert ZOUNDS!

Sawtooth: A raspy sound. Its waveshape looks like a saw's teeth.

Noise: Sounds like your TV does when it's tuned to a nonexistent channel — noisy! It doesn't have a defined waveshape.

Attack: The length of time it takes a tone to go from off to maximum volume.

Decay: The length of time it takes a tone to go from maximum volume to off, or to an intermediate (sustain) level.

Sustain: The volume level that a note can *decay* to after it's reached its maximum.

Release: The length of time a note is held at its *sustaining* volume level before being dropped to zero.

Playing with Your Sound Machine



Now that you know what you're doing (I wish I could say the same), start playing with your Sound Machine. Here are a few entries for you to try. After you've heard these, go for it on your own until you've got all these new terms and concepts down cold.

Sample Sound Machine Entries

First select a *type* of sound, then choose a note.

| When you see | Enter |
|--------------|-------|
| ATTACK | 5 |
| DECAY | 8 |
| SUSTAIN | 5 |
| RELEASE | 9 |

Sounds just like a mandolin. Doesn't it? Now try this one.

| | |
|---------|----|
| ATTACK | 12 |
| DECAY | 0 |
| SUSTAIN | 0 |
| RELEASE | 0 |

DR. WACKO PRESENTS COMMODORE BASIC

A decreasing tone. Sounds like, "whoop!" Here's a familiar one, a xylophone or ping!

| | |
|---------|----|
| ATTACK | 0 |
| DECAY | 10 |
| SUSTAIN | 0 |
| RELEASE | 0 |

The ping sound is just the opposite of the whoop sound. The whoop sound slopes slowly up to its maximum volume, then falls rapidly to zero. The ping sound rises quickly to its maximum volume, then falls slowly to zero.

Experiment with noise until you wake up your pet turtle, then I'll show you how to program all this great sound.

A Sound Program

I thought I had a sound program before I started on this adventure. But the desert sun has fried my brain, and all I can remember is how to program sound on the Commodore 64! It's a good thing too, because that's just what I'm going to do.

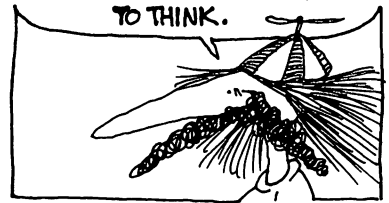
All of your Commodore's built-in sound features are stored in memory in sound registers 54272 to 54296. We'll be POKEing around in these registers, (in a logical fashion of course) to produce all sorts of wild sound.

Six Easy Steps to Orderly Sound

When you write a sound program you have to keep everthing in proper order. You usually:

1. Clear out the sound registers first.
2. Adjust the *volume*.
3. Set *attack/decay*.

IT HELPS TO TRY TO MAKE SOUNDS THAT THAT MATCH WHATS HAPPENING ON THE SCREEN. FOR EXAMPLE, "PINGPINGPING" ISNT QUITE THE NOISE THAT CLYDE MAKES WHILE SLOGGING THROUGH THE DESERT, WHILE "CLANG BRACKBRACK GRUNCH" SOUNDS JUST LIKE JUNIOR TRYING TO THINK.



Desert ZOUNDS!

4. Set *sustain/release*.
5. Set the *high* and *low* frequency of the tone.
6. Select the type (waveform) of tone.

1. Clear Out the Sound Registers

Line 230 of Wacko's Sound Machine shows how to empty out those sound registers.

```
230 FOR X = 54272 TO 54296:POKE X,0:NEXT X
```

This simple FOR/NEXT loop places zeros in each of the sound registers.

2. Adjust the Volume

Line 310 sets the volume level to its maximum by POKEing the volume register, 54292, with 15.

```
310 POKE 54296,15
```

You can POKE the volume register with any number from 0 to 15. POKEing 54296 with 0 turns off the volume, and POKEing it with 15 turns the volume up to its maximum. There is only one volume register. This means, that if you're using more than one voice, (your Commodore 64 has three) this one setting adjusts the volume for *all* voices.

3. Set the Attack and Decay

Each of Commodore's three voices has its own attack/decay register. Here's a chart that shows the values you POKE into each voice's attack/decay register to vary these settings.

DR. WACKO PRESENTS COMMODORE BASIC

ATTACK RATE SETTINGS

| | ATTACK SETTING | HIGH ATTACK | MEDIUM ATTACK | LOW ATTACK | LOWEST ATTACK |
|---------|-------------------|----------------|------------------|---------------|------------------|
| VOICE 1 | 54277 | 128 | 64 | 32 | 16 |
| VOICE 2 | 54284 | 128 | 64 | 32 | 16 |
| VOICE 3 | 54291 | 128 | 64 | 32 | 16 |

DECAY RATE SETTINGS

| | DECAY SETTING | HIGH DECAY | MEDIUM DECAY | LOW DECAY | LOWEST DECAY |
|---------|------------------|---------------|-----------------|--------------|-----------------|
| VOICE 1 | 54277 | 8 | 4 | 2 | 1 |
| VOICE 2 | 54284 | 8 | 4 | 2 | 1 |
| VOICE 3 | 54291 | 8 | 4 | 2 | 1 |

You'll notice that I've cleverly divided the attack/decay chart into two sections, one section for *attack* and the other for *decay*. I'll explain the *attack* portion of the chart first.

Attack

The numbers in the *attack* portion range from super-slow (128) to super fast (16). You can add the numbers in this portion of the chart together to increase the time it will take for a tone to reach its maximum volume. For example, if you want the note to really take its time reaching maximum volume you total all the numbers on this side of the chart, $128 + 64 + 32 + 16 = 240$.

Take a look at line 240 of the Sound Machine and I'll explain how you were able to enter a number from 0 to 15 to adjust the *attack* speed.

240 INPUT "ENTER ATTACK";A:A = A* 16

Desert ZOUNDS!

What I've done here is take your input (**A**) and multiply it by sixteen ($A = A * 16$). So, when you entered 15, register 54277 is POKEd with 240 ($15 * 16 = 240$), and the tone really takes its time reaching maximum volume. In normal practice you'll probably just POKE the *attack* portion of this register with a number, or combinations of numbers, from my chart.

Decay

The numbers on the right side of the chart are used to set the *decay* for each voice. The numbers in the *decay* portion range from super slow (8) to super fast (1). You can also add the numbers in this portion of the chart together to increase the time it will take for a tone to descend to the *sustain* level or zero volume. For example, if you want the note to really take its time reaching zero volume, you total all the numbers on this side of the chart, $8 + 4 + 2 + 1 = 15$.

Combining Attack and Decay

Once you've selected an *attack* and *decay* value, you add the two together to arrive at the number to POKE into the attack/decay register. For example, if you want Voice 1 to generate a sound that has a *moderate attack* (64), and a *super-slow decay* (15), you POKE register 54277 with 79 ($64 + 15 = 79$).

4. Set Sustain and Release

Each of Commodore's three voices has its own sustain/release register. Here's a chart that shows the values you POKE into each voice's sustain/release register to vary these settings.

DR. WACKO PRESENTS COMMODORE BASIC

| | SUSTAIN | | | | |
|---------|--------------------|-----------------|-------------------|----------------|-------------------|
| | SUSTAIN SETTING | HIGH SUSTAIN | MEDIUM SUSTAIN | LOW SUSTAIN | LOWEST SUSTAIN |
| VOICE 1 | 54278 | 128 | 64 | 32 | 16 |
| VOICE 2 | 54285 | 128 | 64 | 32 | 16 |
| VOICE 3 | 54292 | 128 | 64 | 32 | 16 |

| | RELEASE RATE SETTINGS | | | | |
|---------|-----------------------|-----------------|-------------------|----------------|-------------------|
| | RELEASE SETTING | HIGH RELEASE | MEDIUM RELEASE | LOW RELEASE | LOWEST RELEASE |
| VOICE 1 | 54278 | 8 | 4 | 2 | 1 |
| VOICE 2 | 54285 | 8 | 4 | 2 | 1 |
| VOICE 3 | 54292 | 8 | 4 | 2 | 1 |

I've done it again! I've shrewdly divided the sustain/release chart into two sections, one section for *sustain* and the other for *release*. I'll explain the *sustain* portion of the chart first.

Sustain

The numbers in the *sustain* portion range from a really high *sustain* volume of 128, to a really low of 16. You can add the numbers in this portion of the chart together to increase the volume at which your note is *sustained*. For example, if you want the note sustained at peak volume, you total all the numbers on this side of the chart, $128 + 64 + 32 + 16 = 240$.

Line 270 of Wacko's Sound Machine is set, just like line 240, to allow you to enter a number from 0 to 15. It then converts this number to fall within the 16 to 128 range of the *sustain* portion of the register by multiplying your input by sixteen.

Desert ZOUNDS!

270 INPUT "ENTER SUSTAIN";S:S = S* 16

In actual practice you'll probably just POKE the *sustain* portion of this register with a number, or combinations of numbers, from my chart.

Release

The numbers on the right side of the chart are used to set the *release* for each voice. They range from super slow (8) to super fast (1). You can also add the numbers in this portion of the chart to increase the time the note or tone will be held at the *sustaining* level. For example, if you want the note to stay at its *sustaining* level for a long time before dropping of to 0, you total all the numbers on this side of the chart, $8 + 4 + 2 + 1 = 15$.

Combining Sustain and Release

Once you've selected a *sustain* and *release* value, you add the two together to arrive at the number to POKE into the sustain/release register. For example, if you want Voice 1's tone sustained half way down from maximum volume (120) for a long time (15) before dropping off to zero volume, you POKE register 54278 with 135 ($120 + 15 = 135$).

5. Set the High and Low Parts of a Note

You've got to POKE two separate voice registers to produce a tone or note. This Musical POKE Chart will give you the idea.

DR. WACKO PRESENTS COMMODORE BASIC

Musical POKE Chart

| VOICE NUMBER & FREQUENCY | POKE NUMBER | C | C# | D | D# | E | F | F# | G | G# | A | A# | B | C |
|-----------------------------|----------------|----|----|-----|-----|----|-----|-----|----|-----|-----|-----|-----|-----|
| VOICE1/HIGH | 54273 | 34 | 36 | 38 | 40 | 43 | 45 | 48 | 51 | 54 | 57 | 61 | 64 | 68 |
| VOICE1/LOW | 54272 | 75 | 85 | 126 | 200 | 52 | 198 | 127 | 97 | 111 | 172 | 126 | 188 | 149 |
| VOICE2/HIGH | 54280 | 34 | 36 | 38 | 40 | 43 | 45 | 48 | 51 | 54 | 57 | 61 | 64 | 68 |
| VOICE2/LOW | 54279 | 75 | 85 | 126 | 200 | 52 | 198 | 127 | 97 | 111 | 172 | 126 | 188 | 149 |
| VOICE3/HIGH | 54287 | 34 | 36 | 38 | 40 | 43 | 45 | 48 | 51 | 54 | 57 | 61 | 64 | 68 |
| VOICE3/LOW | 54286 | 75 | 85 | 126 | 200 | 52 | 198 | 127 | 97 | 111 | 172 | 126 | 188 | 149 |

For example, if you want Voice 1 to sound off in C-sharp (C#), you:

POKE 54273,36:POKE 54272,85

Each tone or note is made up of two parts — a *high* part (note high) and *low* part (note low).

If you look at line 100 of Wacko's Sound Machine, you'll see that to play C, I've set the Note High variable to 34 (NH = 34), and the Note Low variable to 75 (NL = 75).

100 IF A\$ = "C" THEN NH = 34:NL = 75:GOTO 200

6. Select the Waveform

Here's a simple chart that shows you how to select a *triangle*, *sawtooth*, or *noise* waveform.

ADSR AND WAVEFORM CONTROL SETTINGS

Note Start/Stop Numbers

| | CONTROL REGISTER | TRIANGLE | SAW- TOOTH | PULSE | NOISE |
|---------|---------------------|----------|---------------|-------|---------|
| VOICE 1 | 54276 | 17/16 | 33/32 | 65/64 | 129/128 |
| VOICE 2 | 54283 | 17/16 | 33/32 | 65/64 | 129/128 |
| VOICE 3 | 54290 | 17/16 | 33/32 | 65/64 | 129/128 |

Desert ZOUNDS!

If you want Voice 1 to play in pure crisp tones (triangle waveform), just *POKE 54276,17*. To turn off this tone *POKE 54276,16*.

Examples of waveform useage are on lines 350 and 370 of Wacko's Sound Machine.

**350 POKE 54276,W:REM ** TURN ON
WAVEFORM**

**370 POKE 54276,W-1:REM ** TURN OFF
WAVEFORM**

Putting Those Six Steps to Work

Now that you're totally blown away, I'll show how easy it really is to put these six steps to work in a simple program. (A giant step for munchkinkind.)

Easy Sound

```
10 FOR X = 54272 TO 54296:POKE X,0:NEXT  
X:REM 1. CLEAR OUT SOUND REGISTERS  
20 POKE 54296,15:REM 2. SET VOLUME TO  
MAXIM  
30 POKE 54277,10:REM 3. SET  
ATTACK/DECAY (A = 0 D = 10)  
40 POKE 54278,0:REM 4. SET  
SUSTAIN/RELEASE (S = 0 R = 0)  
50 POKE 54273,45:POKE 54272,198:REM 5.  
SET VOICE 1 TO F NOTE  
60 POKE 54276,17:REM 6. TURN ON PURE  
WAVEFORM  
70 FOR D = 1 TO 1000:NEXT D:REM *  
MAXIMUM DURATION OF NOTE  
80 POKE 54276,16:REM * TURN OFF PURE  
WAVEFORM
```

DR. WACKO PRESENTS COMMODORE BASIC

Just peek at all those little charts, and check out each line of this simple program and you'll see how easy it is to get your Commodore 64 to stand up and be heard! Before you move on, spend some time with Easy Sound. Change the note on line 50, mess with the waveform on lines 60 and 80, futz with the attack/decay and sustain/release settings, really go zonkers! (Like me.)

Now Zat You Understand ZOUND

Now that you understand how your Commodore 64's sound system works, it's time to show you how to use sound in your programs. But first, here is an extra-special sound treat for you to enjoy. Spend a little time working through this program and then we'll get started.



Marrakesh Express

```
10 REM ** MARRAKESH EXPRESS **
20 POKE 53281,6:POKE 53280,6:REM SET
  SCREEN BLUE
30 PRINT "[CLRHM][8X CRSR-D]";TAB(15)
  "[CMDR + 6]JUNIOR'S"
40 PRINT:PRINT TAB(11) "[CTRL + 8]
  MARRAKESH EXPRESS"
```

Desert ZOUNDS!

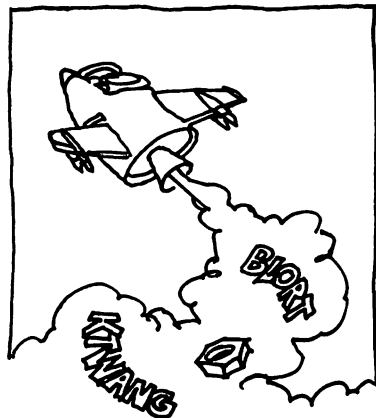
```
50 PRINT:PRINT TAB(17) "[CTRL + 5]AND"
60 PRINT:PRINT TAB(14) "[CMDR + 3]JAZZ
   BAND"
70 R = INT(RND(0)*50) + 1
80 IF R = 30 THEN GOSUB 200
90 FOR X = 15 TO 0 STEP -1—P:GOTO 130
100 REM ***
110 REM *** CHOO-CHOO SOUND ***
120 REM ***
130 POKE 54296,15:REM VOLUME
140 POKE 54277,9:REM MEDIUM DECAY
150 POKE 54273,38:POKE 54272,126:REM
   VOICE 1 SET TO 'D'
160 POKE 54276,129:REM NOISE WAVEFORM
170 NEXT X:P = P + .03:REM TIMING LOOP
180 POKE 54276,128:POKE 53281,6:GOTO
   70:REM TURN OFF WAVEFORM — SCREEN
   BLUE
190 REM ***
200 REM *** TOOT SUBROUTINE ***
210 REM ***
220 POKE 53281,5:REM SCREEN YELLOW
230 POKE 54296,15:REM VOLUME
240 POKE 54277,15:POKE 54284,15:REM
   DECAY SET TO SUPER-LONG
250 POKE 54273,68:POKE 54272,149:REM
   VOICE 1 SET TO HIGH 'C'
260 POKE 54280,51:POKE 54279,97:REM
   VOICE 2 SET TO 'G'
270 POKE 54276,17:POKE 54283,17:REM
   VOICES 1 & 2 SET TO TRIANGULAR
   WAVEFORM
280 FOR T = 1 TO 500:NEXT T:REM MAX
   DURATION OF NOTES
290 POKE 54276,16:POKE 54283,16:REM TURN
   OFF BOTH WAVEFORMS
300 RETURN
```

Sound and the IF/THEN Statement

The IF/THEN statement is often used to introduce sound into a program. It's easy to use. IF, something happens in your program or game — a certain key is pressed, a ball hits a wall or a rocket ship takes off — THEN your program zips to a sound subroutine, and glorious sound accompanies and enhances the action.

Marrakesh Express uses the powerful IF/THEN in line 80 to zip to the Toot subroutine in line 200 every time the random generator spits out a 30.

If you feel real daring, now may be a good time to pull out Clyde's Camel Race Track or my Universe program, and use an IF/THEN statement to spiffy up the race, or let your relatives hear what's going on in my Universe!

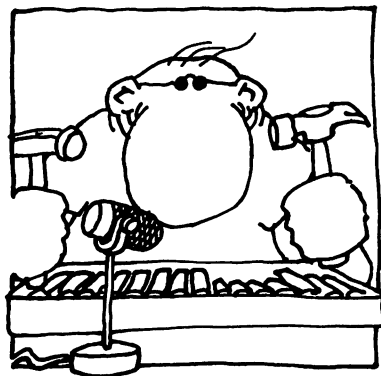


DATA and A Cacophony of Multivoiciferous Zounds

Junior's been taking xylophone lessons for the past thirteen years. I recorded his last session and transferred it to my Commodore 64 so all my former pals could hear it.

This program, called Junior's Two-Part Harmony, uses two voices to play a simple melody. The great thing about it is that all the notes are stored in a DATA line and read into the note registers!

If you'd like to hear Junior's virtuosity, wait 'till I leave the room, then enter and run the program. When you've heard enough, (usually after about two seconds) I'll come back.



Desert ZOUNDS!

Junior's Two-part Harmony

```
10 REM *** JUNIOR'S TWO-PART HARMONY
   ***
20 FOR X = 54272 TO 54296:POKE X,0:NEXT
   X:REM ** CLEAR OUT SOUND REGISTERS
25 REM ** ASSIGN VARIABLE NAMES TO
   SOUND REGISTERS
30 H1 = 54273:L1 = 54272:H2 = 54280:
   L2 = 54279:A1 = 54277:A2 = 54284:
   W1 = 54276:W2 = 54283
40 POKE 54296,15:REM ** SET VOLUME TO
   MAXIMUM
50 POKE A1,10:POKE A2,10:REM SET DECAY
   ONLY
60 FOR N = 1 TO 5
70 READ A,B,C,D:REM READ THOSE NOTES
80 IF A = -1 THEN RESTORE:GOTO 60:REM
   PLAY IT AGAIN SAM
90 POKE H1,A:POKE L1,B:POKE H2,C:POKE
   L2,D:REM ** SET HIGH & LOW PART OF
   NOTES
100 POKE W1,17:POKE W2,16:REM ** TURN
   OFF WAVEFORM
130 NEXT N
140 END
145 REM ** HI LOW VALUE OF NOTES IN SETS
   OF FOUR NUMBERS
150 DATA 34,75,51,97,38,126,51,97,45,198,
   57,172,64,188,38,126,-1,-1,-1,-1
```

Complex and changing sounds can be made by READING values and POKEing them into the *high* and *low* note registers. In Junior's Two-Part Harmony I've done this for two notes at once in line 70.

70 READ A,B,C,D

A and **B** read the *high* and *low* values for Voice 1, and **C** and **D** read in the values for Voice 2.

DR. WACKO PRESENTS COMMODORE BASIC

Random Music

You can make truly strange sounds by using a random generator to randomly select *high* and *low* note register values.

Here's a gruesome example. I've named this short program Clyde's Lament. Listen and you'll understand why.



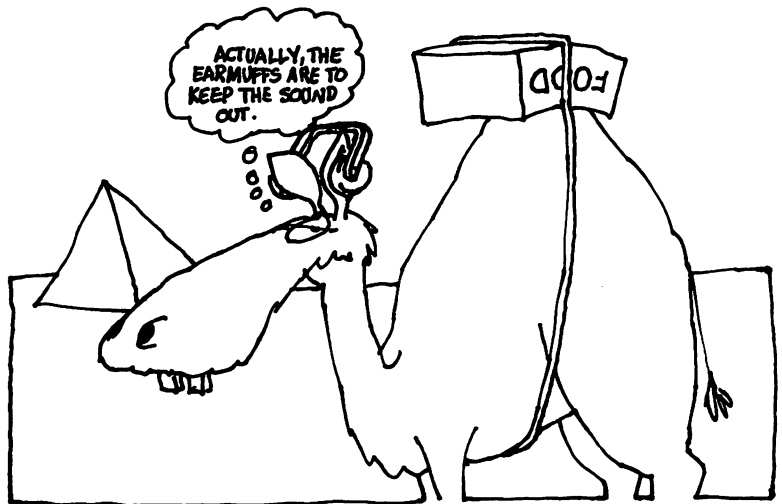
Clyde's Lament

```
10 REM *** CLYDE'S LAMENT ***
20 T = 1
25 REM ** CHANGE WAVEFORM
30 IF T = 1 THEN W = 17
40 IF T = 2 THEN W = 33
50 IF T = 3 THEN W = 129
60 NH = INT(RND(0)*255) + 1:REM HIGH PART
  OF NOTE
70 NL = INT(RND(0)*255) + 1:REM LOW PART
  OF NOTE
80 REM ***
90 POKE 54296,15:REM ** SET VOLUME TO
  MAXIMUM
100 H = 54273:L = 54272:A = 54277:
  S = 54278:REM ** ASSIGN VARIABLE
  NAMES
110 POKE A,10:POKE S,0:REM **
  ATTACK/DECAY & SUSTAIN/RELEASE
120 POKE H,NH:POKE L,NL:REM HIGH & LOW
  PART OF EACH NOTE
130 POKE 54276,W:REM ** SET WAVEFORM
140 FOR D = 1 TO 100:NEXT D:REM MAXIMUM
  DURATION OF EACH NOTE
150 POKE 54276,W-1:REM ** TURN OFF
  WAVEFORM
160 T = T + 1
170 IF T = 4 THEN T = 1
180 GOTO 30
```

Desert ZOUNDS!

Even though Snidely's wild about Clyde's Lament, most camels (except Clyde) have a more refined musical sense. (No offense to your sense intended, Clyde.) They deserve to hear the real McCoy. Music with a beat. Music they can relate to. Music that sounds like music! It's all possible with Commodore 64 sound, but that's another book.

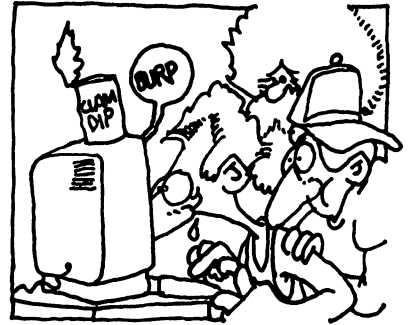
Now that you've developed a sound foundation, and are all set to move on the the next glorious chapter, add sound and weird noise to all your programs. It'll give them a professional touch, and wake up your caravan.



6 Weaving the Perfect Flying Carpet, or, The Art of Programming

If you've got a strong constitution, walk into a hot stuffy room filled with wacked-out programmers (all munching stale tuna fish sandwiches), give them the same problem, and they'll all solve it in their own unique and creative way.

I'll give you an example. I walked into a room full of wackos (and wackettes) and asked them to create a simple version of the game Black Jack (21) to play against the computer.



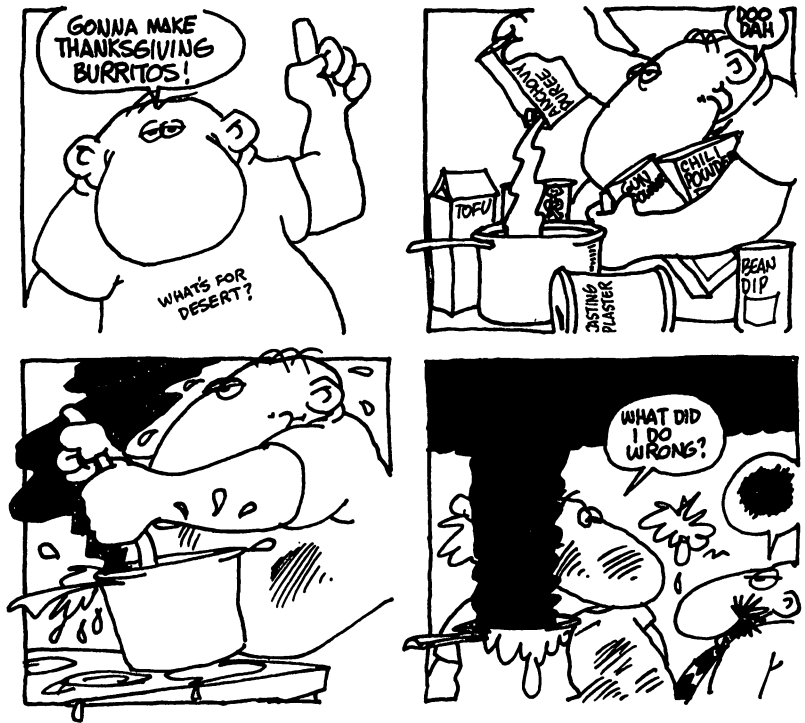
Captain Action's first thought was to use a joystick for the player/computer interface. Ms. Peeky felt that touching the keys was more reassuring, and Petunia went high-tech and decided to get information into the computer by using a light pen. Junior had no comment.

Right off the bat, I knew that these four weirdos would design their programs differently. But it goes even deeper than individuality. The logical thought process

The Art of Programming

that each programmer goes through while creating a program is unique. He or she can effectively solve the same problem in many different ways.

Pay Attention to Detail, Especially When Making Anchovy Burritos



Programming is just like making anchovy burritos for Thanksgiving dinner. You've got to work out a recipe, go shopping for the ingredients, then put it all together and cook it up. If you've paid close attention to detail, your burritos will be scrumptious. If not, you may end up with burned pots, disgruntled relatives, and lots of former friends.

DR. WACKO PRESENTS COMMODORE BASIC

To show you what goes on in a programmer's brain, I'll let you take a glimpse into mine as I tell you how I designed the Mini Word Processor listed on page 230. (A *word processor* is just a computerized typewriter. I designed my computerized typewriter after the keys of my old electric got jammed with sand.)

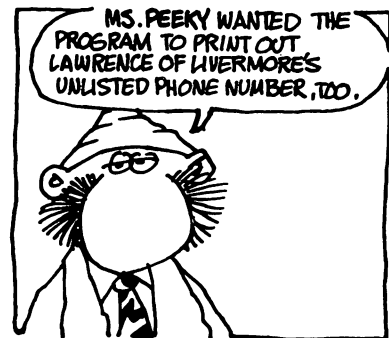
Know the Rules of the Game and Define the Problem

I always write down what I want my program to do *before* I start doing any programming. I do this so I'll have a written framework to build my program on. Defining the problem — by creating a written description of it — is always an essential first step, regardless of what type of program you are designing. This first step may require more of your brainpower and attention to detail than all of the actual programming! Here's what happened to me....

I wanted my word processor to have these six principal features.

- 1. To show text on the screen, and in a variety of colors if desired**
- 2. To include at least some rudimentary editing functions**
- 3. To store text to disk or cassette and load it back into the computer when needed**
- 4. To print out the contents of the screen onto a printer**
- 5. To erase the contents of memory and the screen at the press of a key**
- 6. To be friendly to writers. All options should be easy to remember and easy to use.**

I originally thought that designing a simple word processor would be, well, simple. But after writing down all the ingredients I needed, and mulling over the programming consequences, I discovered that I had never really



The Art of Programming

looked closely at the mechanics of a word processor. I had never broken it down and examined all the little details.

To write a workable program, I had to examine every feature to gain a thorough and complete understanding of it. In the process, I discovered that there was more to a word processor than meets the eye, and I now have a much better understanding of how a word processor operates.

Programming on the Right Side of the Brain

The process of really looking at a programming problem is the same process that artists go through when they paint still-lives or portraits. Artists must “see” every minute detail of the subject before they can create a picture on canvas.

A Short Exercise in Seeing

Here’s a short exercise in *seeing*. It involves what is called “experiential” learning (learning by doing). Once you’re through, you will have experienced total vision, and maybe understand the relationship between art and the art of programming.

Take a piece of paper and a pencil and sit down at the kitchen table. Lay the paper in front of you and place your hand at the side of the paper. Now, take a really close look at that hand. See every line, every curve, every shadow. In a few minutes you’ll know more about your hand than you thought possible.

Starting at any point, move your eyes very, very slowly over the edge of your hand. As your eyes do this let your pencil draw this outline on your paper. Really get into “seeing” every curve and shape that make up the outline of your hand. Take your time. Go slowly. Really concentrate.

DR. WACKO PRESENTS COMMODORE BASIC

When you are finished drawing the outline of your hand, go back and fill in all the other details: the curves of your fingernails, the small creases in your hand, even the pores and minute strands of hair on the top of each finger. You'll be pleasantly surprised at the results. This drawing (if you're not already an artist) is probably the most detailed and realistic you've ever done. And all because you *really saw* your hand for the first time!



The Wacko Side of Your Brain

While you were drawing, you may have noticed that you lost sense of time and didn't hear that radio or TV playing in the background or any external sounds. You were completely engrossed in your work. That's because you were using what I call the "Wacko" (creative or right) side of your brain. You've also learned to appreciate the effort and attention to detail that goes into a work of art, such as your drawing of your hand.



But what does this have to do with programming? EVERYTHING! When you are programming or designing a program, you need to "see" the final result with the detail artists "see" before they begin painting pictures. *When you program, you must use the Wacko side of your brain!*

"Like, Totally, Really!"

You must totally see your programming problem, whether you are designing a word processor, a game, or something totally wacko. By seeing the problem, you gain a total understanding of the problem and can successfully design a program that effectively solves the problem. Total involvement helps you appreciate good programming in yourself and in others.

Now That You Understand the Ground Rules

Once I've defined a program and have an in-depth understanding of its functions, I review each function, apply my knowledge of BASIC, and add a touch of wackiness.

You can start working with any of the steps you've written down. But I usually start at the top, with the first one, and work my way down the list. This is the same way that your computer executes a program. So, working this way makes it easier to put together the total program. As you get deeper into the programming you may want to change the order of things to make your program more efficient. Go for it! This is part of the creative programming process.

The Modular Approach

I examine and work through each step, treating it as a small program unto itself, which I call a "module." I write a "shopping list" of programming ingredients needed to make each module of the program work on its own, then formulate a recipe that blends its ingredients.

After I've completed a module, I first test it by itself, and then within the entire program. Sometimes I'll design a module directly on the computer. I enter all the programming, then experiment with my module (adding a pinch of this and pinch of that) until it "tastes" just right. This method also works when making spaghetti sauce!

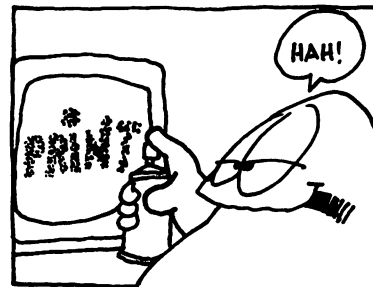
A Word Processor, Step-by-Step

Enough of this conceptualizing stuff. Let's design a word processor! (If you'd like to look at the finished product before we begin, just flip to page 230, type in the program, and start writing that great novel.) All set?

OK, but I'd like to point out one thing before we start designing and programming. The line numbers I use below to explain each module are the same as I've used in the finished word processor. This will help you see how it all fits together.

MODULE 1: Print to the Screen

The first thing a word processor must do is print text on the screen, and display a cursor and RETURN marker so you know where you are. So, I designed Module 1 to do just that.



Module 1

```
200 DIM B$(5000):T = 1:L = 1
210 PRINT CHR$(147):PRINT:PRINT
220 PRINT CHR$(153);CHR$(162);CHR$(5);
230 GET A$:IF A$ = "" THEN GOTO 230
240 REM **
250 REM ** OPTIONS FIT IN HERE **
260 REM **
700 REM ** END OF LINE ROUTINE **
710 IF ASC(A$) = 13 THEN PRINT CHR$(157);
    CHR$(150);CHR$(179);CHR$(5);
    CHR$(13);
720 PRINT CHR$(157);A$;CHR$(153);
    CHR$(162);CHR$(5);
730 REM **
800 B$(T) = A$:T = T + 1:GOTO 230
```

In line 200, the array **B\$** is DIMensioned to store text that's put into it in line 800 of the module. **T** is set equal to 1, and is used to put the characters, in order, into the array, and **L** is set equal to 1 for use later in the printer routine.

The Art of Programming

In line 210 the screen is cleared by printing **CHR\$(147)**;, and then a few extra lines are printed so that your writing won't start at the very top of the screen.

Line 220 creates and prints a green cursor on your screen. **CHR\$(153)** turns on bright green character printing, **CHR\$(162)** prints the cursor, and **CHR\$(5)** returns the printed characters to bright white. I chose **CHR\$(162)** as the cursor because it remains the same in both upper — and lowercase modes.

The computer waits for a key stroke input in line 230. When you press any key, this character is assigned to **A\$**, and the program moves down to line 710.

Line 710 prints a red marker on the screen every time you press RETURN. Here's how this nifty line works.

IF ASC(A\$) = 13, waits for you to press a key. If you press RETURN the ASCII value of **A\$** is equal to 13 (check out the ASCII chart in Appendix C). **THEN** a lot of fancy printing takes place. First a left cursor is printed (**CHR\$(157)**), and **CHR\$(150)** turns on bright red printing. The cursor is printed with **CHR\$(179)**, the screen is returned back to bright white printing with **CHR\$(5)**, and finally, **CHR\$(13)** prints a RETURN and sends the cursor whizzing down one line and over to the left of your screen. Wheew!

Line 720 first prints a left cursor, then the key you've pressed (**A\$**), and the cursor.

Line 800 ends the routine by storing each of the characters in array **B\$(T)**, updates the counter **T = T + 1** and returns back to line 230 to wait for your next key stroke.

Type in and run Module 1. You've got a very rudimentary word processor! Each character you type is printed on your screen and stored in array **B\$(T)**!

DR. WACKO PRESENTS COMMODORE BASIC

The OPTIONS, like printing, saving, loading, erasing, and editing the text will fit between lines 230 and 700 of the program.

MODULE 2: Edit

Everything was hunky-dory until I discovered that each time I corrected text by pressing the INST DEL key, the ASCII value of the DEL key, 20, was stored in array B\$.

This just wouldn't fly. I realized that if I tried to print or save the text, these extra characters would get in the way and bollix up the works. I had to come up with a way to delete each character on the screen, *without* placing the ASCII value of the DEL key into the array.

Here's the solution I finally came up with, after much experimentation, and futzing around.

Module 2

```
240 IF ASC(A$) = 20 THEN PRINT A$;CHR$(157);  
    CHR$(153);CHR$(162);CHR$(5);  
    :T = T - 1:GOTO 230
```

I inserted this short module and solved the problem. Here's how it works. If you press the INST DEL key, the ASCII value of A\$ is 20. When this happens the program prints a left cursor, and the green cursor moves one space to the left, where it erases the character it lands on.

T = T - 1 is the heart of this module. Each time the cursor does its thing on the screen, the array counter is set *back* one notch.

To finish things off, the program returns to line 230 to GET the next character.



The Art of Programming

The solution looks pretty straightforward from here. But it took me quite a bit of experimentation to arrive at the result I was looking for. One lesson I learned while working through this problem was to have a very clear idea of the result I wanted *before* I tried to find a solution. It all gets back to defining the problem!

SINCE YOUR COMPUTER CANT JUMP TO CONCLUSIONS ABOUT WHAT YOU'RE TRYING TO DO, ITS IMPORTANT TO OUTLINE YOUR PROGRAM BEFORE YOU START. OTHERWISE YOU'LL END UP LIKE JUNIOR WHO PLANNED FOR THE TRIP BY PACKING 346 ANCHOVY BURRITOS.

I DID THAT ON PURPOSE!



MODULE 3: Read and Review

Since most novels are longer than twenty-four screen lines, I decided to incorporate a feature that lets you scroll a document to *read* and *review* it.

Look at the routine first, then I'll explain how it works and how it's used.

Module 3

```
360 REM ** READ AND REVIEW — 'R' OR  
      'F' + COMMODORE KEY **
```

```
370 IF ASC(A$) = 178 THEN S = 50:  
      PRINT CHR$(147):GOSUB 10000:  
      GOTO 220
```

```
380 IF ASC(A$) = 187 THEN S = 1:  
      PRINT CHR$(147):GOSUB 10000:  
      GOTO 220
```

```
10000 REM ** SUBROUTINES **
```

```
10100 REM ** READ & REVIEW **
```

```
10200 FOR X = 1 TO T-1:PRINTB$(X);  
      :FOR P = 1 TO S:NEXT P:NEXT X:RETURN
```

DR. WACKO PRESENTS COMMODORE BASIC

If you press the Commodore key plus the R key, the ASC value of A is 178. Then, S equals 50, the screen is cleared (**PRINT CHR\$(147)**), and the program GOSUBs to line 10000.

In line 10200 the first FOR/NEXT loop prints the contents of the array on the screen. The inner FOR/NEXT loop (**FOR P = 1 to S**) is set to delay the printing by the value (50) that was assigned to S. This ensures that the words don't zoom by too fast to read.

If you press the Commodore key plus the F key, the value of S in line 10200 is set to 1, (Fast Review) and the text whizzes across your screen.

When the printing is finished, the program returns back to either lines 370 or 380, then goes up to 220 where the cursor is printed on the screen and the program waits patiently for your next keystroke.

Here's how to use the Read feature.

1. Press the Commodore key plus the R key to slowly scroll your text down the screen.
2. Press the Commodore key plus the F key for fast scroll.

MODULE 4: Saving Text

I wanted to design this module to make saving text as easy as possible. Here's how I wanted it to work.

1. Press the Commodore key and the S key and the screen clears.
2. A message appears on the screen asking whether you want to save your text to disk or cassette recorder.
3. You enter D for disk, or C for cassette, and press RETURN.



The Art of Programming

4. The screen clears again, and you are asked to enter a file name. You enter a name of your choosing and press RETURN.
5. Text that is stored in array B\$(T) is saved to the proper device.
6. After the text is stored, it is printed on the screen and the program goes back to line 230 to wait for your next key stroke.

Once I had written down what I wanted to accomplish, I started programming. Here's the result:

Module 4

```
500 REM ** SAVE —'S' + COMMODORE
    KEY **
510 IF ASC(A$) = 174 THEN B$(T) = "*":
    GOTO 3000
3000 REM *** SAVE ROUTINE ***
3010 GOSUB 11000:GOSUB 12000
3020 IF D$ = "C" THEN GOTO 3080
3030 OPEN 1,8,2,"@0:" + F$ + ",S,W"
3040 FOR X = 1 TO T
3050 PRINT#1,B$(X);
3060 NEXT X
3070 CLOSE 1:A$ = CHR$(187):GOTO 380
3080 REM ** CASSETTE SAVE **
3090 OPEN 1,1,1,F$
3100 FOR X = 1 TO T
3110 PRINT#1,B$(X);
3120 NEXT X
3130 CLOSE 1:A$ = CHR$(187):GOTO 380
10000 REM ** SUBROUTINES **
10900 REM ** SAVE/LOAD — SCREEN FORMAT
    **
11000 PRINT CHR$(147)
11100 FOR X = 1 TO 10:PRINT:NEXT X
11200 PRINT TAB(8); "[CTRL + 2]D[CMDR + 6]ISK
    OR [CTRL + 2]C[CMDR + 6]ASSETTE:
    [CTRL + 2]";
```

DR. WACKO PRESENTS COMMODORE BASIC

```
11300 INPUT D$
11400 RETURN
11900 REM ** SAVE — SCREEN FORMAT
      SUBROUTINE **
12000 PRINT CHR$(147)
12100 FOR X = 1 TO 10:PRINT:NEXT X
12200 PRINT TAB(5);
      “[CMDR + 6]SAVE[CTRL + 2] FILENAME:”;
12300 INPUT F$
12400 RETURN
```

GOTO the Save Text Routine. In line 510, if you press the S key, the ASC value of A\$ is 174 and an asterisk (*) is added to the end of the file so the Load Routine will know when the file ends. Then the program goes to the *Save Text Routine* beginning on line 3000.

The Save Text Routine. In line 3010, the program first GOSUBs to line 11000 to clear the screen, prints the question DISK OR CASSETTE:?, and accepts your input (D\$) on line 11300. It then RETURNS to line 3010 and GOSUBS to line 12000.

In lines 12000 through 12300 the screen is cleared, the question SAVE FILENAME:? is printed, and your choice of filename is assigned to F\$. The program then RETURNS to line 3020.

In line 3020, if you’ve elected to save your prose to your cassette recorder, the program goes to line 3090. If not, the program moves on to line 3030. Here’s a close-up of this important line.

```
3030 OPEN 1,8,2,“@0:” + F$ + “,S,W”
```

This line opens a **Sequential** file named F\$ via data channel 1 on the disk. It also tells the disk drive that you are going to **Write** to the disk. The @0: symbols at the beginning of the string allow you to *replace*, or write over, a previous file of the same name. This feature

The Art of Programming

comes in real handy when you edit prose and want to save the updated version and give it the same file name you gave the original version.

The FOR/NEXT loop surrounding line 3110 prints the characters saved in array **B\$(X)** onto the disk. When the FOR/NEXT loop runs out of characters to print, the program goes down to line 3130.

The channel to the disk drive is closed in line 3130, then **A\$** is made equal to an uppercase letter *F* (**A\$ = CHR\$(187)**), and the program goes to line 380 (the fast-review routine), where your text is scrolled down the screen.

Cassette Save. The Cassette Save routine on lines 3090 to 3130 works just like the Disk Save routine, except it uses the special Cassette OPEN statement on line 3090.



MODULE 5: Loading Text

To keep things simple, I designed this module to load text like the module that saves text. Here's how I wanted it to work.

1. Press the Commodore key + L and the screen clears.
2. A message appears on the screen asking whether you want to load your text from disk or cassette recorder.
3. You enter D for disk, or C for cassette, and press RETURN.
4. The screen clears again, and you are asked to enter a file name. You enter a name of your choosing, and press RETURN.
5. Text that is stored on either disk or cassette is loaded into array **B\$(T)**.
6. After the text is loaded into the computer the program goes back to line 230 to wait for your next key stroke.

DR. WACKO PRESENTS COMMODORE BASIC

By making the save and load features of this word processor similar, I eased my programming chores and made the program easy to use. Here's the Module 5 part of this program.

Module 5

```
600 REM ** LOAD — 'L' + COMMODORE
    KEY **
610 IF ASC(A$) = 182 THEN GOTO 4000
4000 REM *** LOAD ROUTINE ***
4010 GOSUB 11000:GOSUB 13000
4020 IF D$ = "C" THEN GOTO 4090
4030 OPEN 1,8,2,"0:" + F$ + ",S,R"
4040 GET#1,A$
4050 IF A$ = "*" THEN CLOSE 1:
    A$ = CHR$(187):GOTO 380
4060 B$(T) = A$
4070 T = T + 1:GOTO 4040
4080 REM ** CASSETTE LOAD **
4090 OPEN 1,1,0,F$
4100 GET#1,A$
4110 IF A$ = "*" THEN CLOSE 1:
    A$ = CHR$(187):GOTO 380
4120 B$(T) = A$
4130 T = T + 1:GOTO 4100
10000 REM ** SUBROUTINES **
11000 PRINT CHR$(147)
11100 FOR X = 1 TO 10:PRINT:NEXT X
11200 PRINT TAB(8);"[CTRL + 2]D[CMDR + 6]ISK
    OR [CTRL + 2]C[CMDR + 6]ASSETTE:
    [CTRL + 2]";
11300 INPUT D$
11400 RETURN
12900 REM ** LOAD — SCREEN FORMAT **
13000 PRINT CHR$(147)
13100 FOR X = 1 TO 10:PRINT:NEXT X
13200 PRINT TAB(5);
    "[CMDR + 6]LOAD[CTRL + 2] FILENAME:";
```

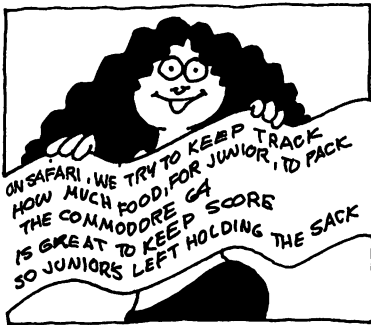
The Art of Programming

```
13300 INPUT F$
13400 T = 1:RETURN
```

GOTO the Load Text Routine. This routine is almost the same as the *GOTO Save Text Routine* you've just seen. But it waits for you to press the Commodore key plus L before going to the load routine.

The Load Text Routine. This routine is almost identical to the *Save Text Routine*. But it **GETs** characters from either disk or cassette and puts them into array **B\$(T)**.

In lines 4050 and 4110 **A\$ = "*" .** If the device is closed.



MODULE 6: Printer Routine

I decided that it would be nice to add a feature to allow you to print the contents of the screen to a printer. And here's the module that does this.

Module 6

```
400 REM ** PRINTER — 'P' + COMMODORE
    KEY **
410 IF ASC(A$) = 175 THEN GOTO 2000
2000 REM *** PRINTER ROUTINE ***
2110 PRINT CHR$(147):OPEN 1,4,7:
    FOR P = 1 TO T
2120 IF B$(P) = CHR$(32) AND L > 60
    THEN PRINT# 1, CHR$(13)::PRINT
        CHR$(32)::L = 1:NEXT P
2130 IF B$(P) = CHR$(13) THEN L = 1
2140 PRINT B$(P)::PRINT# 1, B$(P)::L = L + 1:
    NEXT P:PRINT# 1:CLOSE 1
2150 A$ = CHR$(187):GOTO 380
```

GOTO the Printer Routine. In line 410, if you press P the ASC value of **A\$** equals 175 and the program goes to the *Printer Routine* beginning on line 2000.

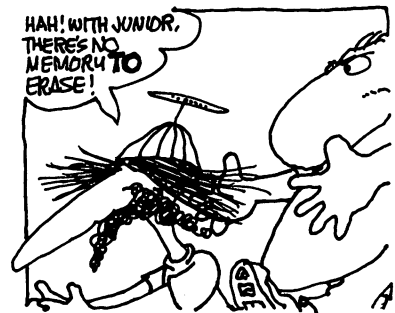
DR. WACKO PRESENTS COMMODORE BASIC

Printer Routine. The Printer Routine on lines 2110 through 2150 performs the following functions.

1. *Line 2110:* Here, the screen is cleared and an upper — /lowercase channel is opened to the printer.
2. *Line 2120:* This snazzy line ensures that a carriage return occurs after the first occurrence of a blank space *after* a minimum of 60 characters have been printed out (**IF B\$(P) = CHR\$(32) AND L > 60**). This feature automatically formats your printout!
3. *Line 2130:* If you press RETURN (**CHR\$(13)**), L is reset to equal 1.
4. *Line 2140:* This line prints your text on the screen, then on your printer. It next advances the counter (**L = L + 1**), and closes channel 1 when the printing is finished.
5. *Line 2150:* When printing is completed, **A\$** is made equal to F and the program goes to 380, where your text is fast-scrolled down the screen.

MODULE 7: Clear Screen and Erase Memory

I designed this short module so you can clear the screen and memory by pressing the Commodore key plus E. I selected the letter *E* to represent “Erase,” so the operation would be easy to remember.



Module 7

```
440 IF ASC(A$) = 177 THEN CLR:GOTO 200
```

Simple, isn't it? If you press the Commodore key and E, the ASC value of **A\$** is 177, memory is cleared (**CLR**), and the program goes to line 200 and begins again.



A Multi-color Word Processor

Yes, you heard me right! This amazing word processor will print your prose in a multitude of exciting and vibrant colors! In lines 270 through 320 I've included routines that let you take advantage of Commodore's colorful printing capabilities. Just think of it. A blue poem for a rainy day, or a red letter to really express your emotions!

Here's one line of this colorful routine to show you how it works.

```
280 IFASC(A$) > 148ANDASC(A$)< 157  
    THENPRINT"[CRSR-L] ";CHR$(ASC(A$));  
    "[CMDR + B][CTRL + 2][CRSR-R]";GOTO 800
```

If you check out the Hotski-Totski chart, you'll see that decimal codes 149 through 156 represent the text colors available when you press the Commodore key in combination with a number key between 1 through 8, plus purple (CTRL key + 5).

When you press one of the colorful combinations, the cursor is moved one space to the left and the color you've pressed is activated. Then the squiggly graphics character that resides on the side of the B key is printed in the color you've chosen. The printout color is then changed back to white, the cursor is moved one space to the right, and the program goes to line 800 to wait for your next keystroke. Wheew!

Lines 290 through 320 work just the same, but they wait for you to enter different colors.

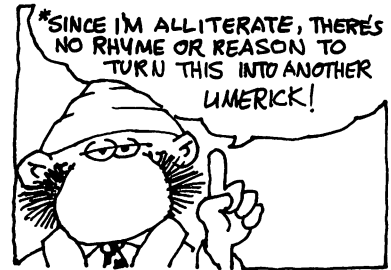
Use the *Review* option to view your colorful prose.

DR. WACKO PRESENTS COMMODORE BASIC

Two Last-minute Additions

After I finished the program, and started to use it, I discovered a couple of ways to improve it.*

1. I added the End-of-line Routine on line 710 to print a red marker on the screen every time you press RETURN.
2. The Quotation-mark Replacement Routine on line 260 replaces a standard quotation mark (") with a ('). I did this because it's bad manners to enter a quotation mark within a string. If you do, your Commodore will get sick or go bzzrk!



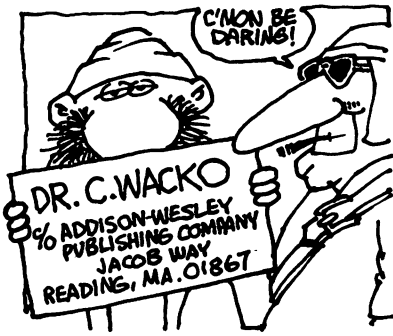
The Word Processor That Ate Cleveland

As I designed this program I kept wanting to add more features. Even though your printout is formatted, I thought it would be nice to add a "wraparound" feature to automatically shift (wrap) the last word of a line to the left margin of the next line on the screen. This common feature on most professional word processors eliminates pressing RETURN at the end of each line.

I also wanted to add printer functions that would automatically center text, line up text evenly on the right-hand side of the printout, and underline or highlight words.

I also thought that incorporating a "search-and-replace" feature would be real helpful. And, most of all, I wanted to make use of all the editing functions; not just the DEL backspace.

But I knew that if I added all these features, the program would be extremely long and difficult to explain, and you wouldn't have a chance to experiment on your own!



Keep the Cards and Letters Coming!

I'd like to see your modified versions of this word processor. So, if you make some additions, please send me a short note and your program listing. Dr. Wacko will answer you. Trust me!

Adding Some Documentation

Once you've finished your program, worked out the bugs, and really think it's got possibilities, it's time to write operating instructions so your friends can use it. To give you the idea, here's the documentation for Wacko's Amazing Word Processor.

Wacko's Amazing Word Processor

This word processor lets you create text (up to 5000 characters in length) on your screen in your choice of colors, save it on either a disk or cassette, and print out the contents of the screen on a printer.

Easy Operating Instructions

1. *Getting Started.* A title page appears on your screen after you load and run your word processor. Just press the f7 key and you're ready to begin typing.
2. *To Save Your Text.* Press the Commodore key and the "S" key simultaneously, and your screen will look like this.

DISK OR CASSETTE:?

DR. WACKO PRESENTS COMMODORE BASIC

Type a letter *D* to save to disk or a letter *C* to save your prose to cassette, then press RETURN and your screen will look like this.

[SAVE] FILENAME:?

Enter a filename of your choice, then press RETURN to save your text. Your entry should look like this.

FILENAME [RETURN]

3. *To Load Your Text.* Press the Commodore key and the L key simultaneously, and your screen will look like this.

DISK OR CASSETTE:?

Type a letter *D* to load from disk or a letter *C* to load your prose into the computer from cassette, then press RETURN and your screen will look like this.

[LOAD] FILENAME:?

Enter the name of the file you want to load in, then press RETURN to load it. Your entry should look like this.

FILENAME [RETURN]

4. *To Print Text on Your Printer.* Press the Commodore key and the P key and your text will be printed out. Don't forget to turn on your printer!

5. *Editing.* The only editing function available on this word processor is the DEL key. Each time you press the DEL key the cursor moves back one space and deletes one character.

The cursor doesn't travel up the screen when it reaches a carriage return. It continues to march across the screen! If you continue to press the DEL key, characters are still being deleted from the word processor's

The Art of Programming

memory. Use the Read and Review option to see your edited text.

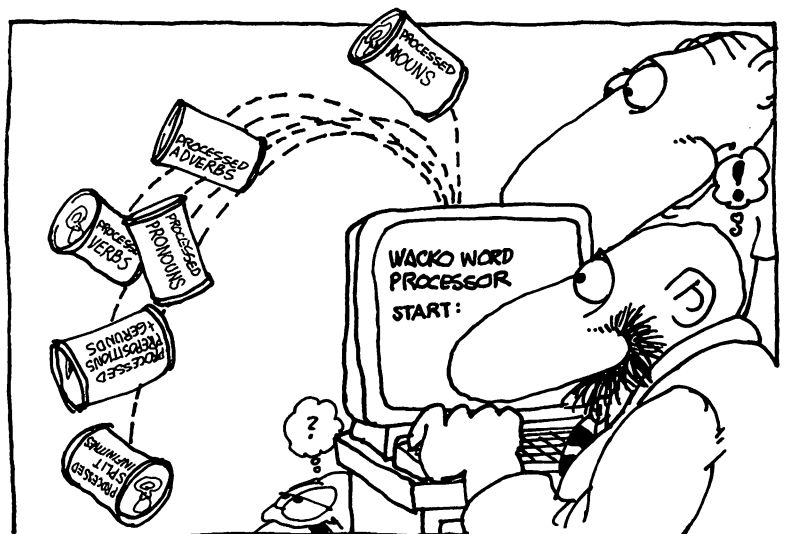
6. *Read and Review.* The Read and Review option is used during editing to see edited text and to scroll and read text that is longer than the length of the screen.

Slow Scroll. Press the Commodore key and the R key to slow scroll the text down the screen.

Fast Scroll. To fast scroll, press the Commodore key and the F key.

7. *Colorful Characters.* To add color to your display, press the Commodore key or the CTRL key in combination with keys 1 through 8. A colored marker will appear on your screen. Use the Read and Review option to view your colorful text.

Now that you know how to design and use a word processor, here's the program you've been waiting for. Type it in carefully, run it, and write that great novel or a nasty letter to me.



DR. WACKO PRESENTS COMMODORE BASIC

The first section of this amazing word processor, the COVER (lines 1 through 110), is listed in upper- and lowercase letters. This is because I've used both upper- and lowercase characters in the title. Just press the Commodore key plus the SHIFT key to enter this section, then press these two keys again before entering the balance of the program.

I've combined two screen-printing techniques within the program to show off your Commodore 64's versatility:

- 1) By imbedding keystrokes inside quotation marks.
For an example look at line 280.
- 2) By using CHR\$ to print the character or function.
Line 220 is a good example of this.

In most cases the two techniques are interchangeable.

Wacko's Amazing Word Processor

```
1 rem *** Dr. C. Wacko's Miraculous Word
  Processor ***
3 rem **
5 rem *** COVER ***
10 poke 53280,0
20 print chr$(14);chr$(147);
30 for x = 1 to 8:print:next x
35 print tab(11);"[CTRL + 1]-----"
40 print tab(12);"[CTRL + 2]Dr. [CMDR + 6]C.
  [CTRL + 5]Wacko's"
50 print tab(15);"[CTRL + 4]Miracle"
60 print tab(12);"[CTRL + 8]Word Processor"
65 print tab(11);"[CTRL + 1]-----"
70 for x = 1 to 10:print:next x
80 print "[CMDR + 6]Press the
  [CTRL + 2]f7[CMDR + 6] key to get things
  started!"
90 get a$:if a$ = "" then goto 90
100 if asc(a$)< > 136 then goto 90
```

The Art of Programming

```
110 print chr$(147);chr$(142)
120 REM **
130 REM *** WORD PROCESSOR
    BEGINS ***
140 REM**
200 DIM B$(5000):T = 1:L = 1
210 PRINT CHR$(147):PRINT:PRINT
220 PRINT CHR$(153);CHR$(162);CHR$(5);
230 GET A$:IF A$ = "" THEN GOTO 230
235 REM ** DELETE/BACKSPACE EDIT
    ROUTINE **
240 IF ASC(A$) = 20THENPRINTA$;
    CHR$(157);CHR$(153);CHR$(162);
    CHR$(5);:T = T - 1:GOTO230
250 REM ** QUOTATION MARK REPLACE
    ROUTINE **
260 IF ASC(A$) = 34 THEN A$ = ""
270 REM ** COLORS — SCREEN PRINT
    ROUTINES **
280 IFASC(A$) > 148ANDASC(A$) < 157
    THENPRINT"[CRSR-L]";CHR$(ASC(A$));
    "[CMDR + B][CTRL + 2][CRSR-R]";
    :GOTO800
290 IFASC(A$) > 157ANDASC(A$) < 160
    THENPRINT"[CRSR-L]";CHR$(ASC(A$));
    "[CMDR + B][CTRL + 2][CRSR-R]";
    :GOTO800
300 IFASC(A$) > 29ANDASC(A$) < 32
    THENPRINT"[CRSR-L]";CHR$(ASC(A$));
    "[CMDR + B][CTRL + 2][CRSR-R]";
    :GOTO800
310 IFASC(A$) = 129ORASC(A$) = 144
    THENPRINT"[CRSR-L]";CHR$(ASC(A$));
    "[CMDR + B][CTRL + 2][CRSR-R]";
    :GOTO800
320 IFASC(A$) = 5ORASC(A$) = 28
    THENPRINT"[CRSR-L]";CHR$(ASC(A$));
    "[CMDR + B][CTRL + 2][CRSR-R]";
    :GOTO800
330 REM ***
```

DR. WACKO PRESENTS COMMODORE BASIC

```
340 REM *** SELECT OPTIONS ***
350 REM ***
360 REM ** READ AND REVIEW — 'R' OR 'F'
    + COMMODORE KEY **
370 IF ASC(A$) = 178 THEN S = 50:PRINT
    CHR$(147):GOSUB 10000:GOTO 220
380 IF ASC(A$) = 187 THEN S = 1:PRINT
    CHR$(147):GOSUB 10000:GOTO 220
390 REM **
400 REM ** PRINTER — 'P' + COMMODORE
    KEY **
410 IF ASC(A$) = 175 THEN GOTO 2000
420 REM **
430 REM ** CLEAR SCREEN & MEMORY —
    'C' + COMMODORE KEY **
440 IF ASC(A$) = 177 THEN CLR:GOTO 200
500 REM ** SAVE — 'S' + COMMODORE
    KEY **
510 IF ASC(A$) = 174 THEN B$(T) = "*":
    GOTO 3000
600 REM ** LOAD — 'L' + COMMODORE
    KEY **
610 IF ASC(A$) = 182 THEN GOTO 4000
700 REM ** END OF LINE ROUTINE **
710 IF ASC(A$) = 13 THEN PRINT
    CHR$(157);CHR$(150);CHR$(179);
    CHR$(5);CHR$(13);
720 PRINT CHR$(157);A$;CHR$(153);
    CHR$(162);CHR$(5);
730 REM **
800 B$(T) = A$:T = T + 1:GOTO 230
1000 REM **
1010 REM ** MAIN ROUTINES **
1020 REM **
2000 REM *** PRINTER ROUTINE ***
2110 PRINT CHR$(147):OPEN 1,4,7:FOR P = 1
    TO T
2120 IF B$(P) = CHR$(32) AND L > 60
    THEN PRINT#1,CHR$(13);:PRINT
    CHR$(32);:L = 1:NEXT P
```

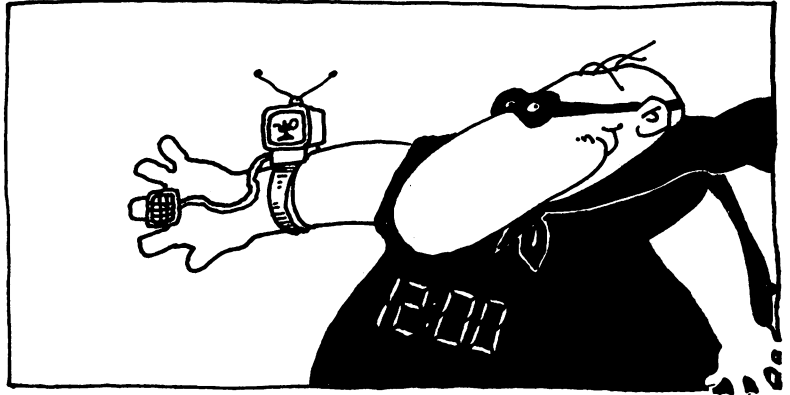
The Art of Programming

```
2130 IF B$(P) = CHR$(13) THEN L = 1
2140 PRINT B$(P);:PRINT#1,B$(P);:L = L + 1:
      NEXT P:PRINT#1:CLOSE1
2150 A$ = CHR$(187):GOTO 380
3000 REM ** SAVE ROUTINE **
3010 GOSUB 11000:GOSUB 12000
3020 IF D$ = "C" THEN GOTO 3080
3030 OPEN 1,8,2,"@0:" + F$ + ",S,W"
3040 FOR X = 1 TO T 3050 PRINT#1,B$(X);
3060 NEXT X
3070 CLOSE 1:A$ = CHR$(187):GOTO 380
3080 REM ** CASSETTE SAVE **
3090 OPEN 1,1,1,F$
3100 FOR X = 1 TO T
3110 PRINT#1,B$(X);
3120 NEXT X
3130 CLOSE 1:A$ = CHR$(187):GOTO 380
4000 REM ** LOAD ROUTINE **
4010 GOSUB 11000:GOSUB 13000
4020 IF D$ = "C" THEN GOTO 4090
4030 OPEN 1,8,2,"0:" + F$ + ",S,R"
4040 GET#1,A$
4050 IF A$ = "*" THEN CLOSE 1:
      A$ = CHR$(187):GOTO 380
4060 B$(T) = A$ 4070 T = T + 1:GOTO 4040
4080 REM ** CASSETTE LOAD **
4090 OPEN 1,1,0,F$
4100 GET#1,A$
4110 IF A$ = "*" THEN CLOSE 1:
      A$ = CHR$(187):GOTO 380
4120 B$(T) = A$
4130 T = T + 1:GOTO 4100
9900 REM **
10000 REM ** SUBROUTINES **
10050 REM **
10100 REM ** READ & REVIEW **
10200 FOR X = 1 TO T-1:PRINTB$(X);:FOR P = 1
      TO S:NEXT P:NEXT X:RETURN
```

DR. WACKO PRESENTS COMMODORE BASIC

```
10900 REM ** SAVE/LOAD — SCREEN
      FORMAT **
11000 PRINT CHR$(147)
11100 FOR X = 1 TO 10:PRINT:NEXT X
11200 PRINT TAB(8);"[CTRL + 2]D[CMDR + 6]ISK
      OR [CTRL + 2]C[CMDR + 6]ASSETTE:
      [CTRL + 2]";
11300 INPUT D$
11400 RETURN
11900 REM ** SAVE — SCREEN FORMAT **
12000 PRINT CHR$(147)
12100 FOR X = 1 TO 10:PRINT:NEXT X
12200 PRINT TAB(5);"[CMDR + 5]SAVE[CTRL + 2]
      FILENAME:";
12300 INPUT F$
12400 RETURN
12900 REM ** LOAD — SCREEN FORMAT **
13000 PRINT CHR$(147)
13100 FOR X = 1 TO 10:PRINT:NEXT X
13200 PRINT TAB(5);"[CMDR + 5]LOAD[CTRL + 2]
      FILENAME:";
13300 INPUT F$
13400 T = 1:RETURN
```

A Special Bonus Coder/Decoder



Once you've created a message using Wacko's Amazing Word Processor, you can protect it from prying eyes with the next two bonus programs: Coder and Decoder.

Here are the two program listings. Just enter and save each one, then I'll tell you how to protect your text.

These programs were designed to work with a disk drive. If you have a cassette recorder, just modify the OPEN statements.

Coder

```
10 REM ** CODER **
20 CLR:T = 1:DIM B$(5000):PRINT CHR$(147);
30 POKE 53281,6:POKE 53280,0
40 FOR X = 1 TO 5:PRINT:NEXT X:PRINT
   TAB(15);"[CTRL + 8]CODER"
50 FOR X = 1 TO 5:PRINT:NEXT X
60 PRINT TAB(8);"[CMDR + 6]ENTER TODAY'S
   CODE:[CTRL + 2]";
70 INPUT D
80 IF D > 100 OR D < 1 THEN GOSUB
   320:GOTO 20
```

DR. WACKO PRESENTS COMMODORE BASIC

```
90 REM **
100 REM ** LOAD CLEAR TEXT FILE **
110 REM **
120 PRINT:PRINT TAB(8);"[CMDR + 6]CLEAR
    TEXT[CTRL + 2] FILENAME:";
130 INPUT F$
140 OPEN 1,8,2,"0:" + F$ + ",S,R"
150 GET#1,A$
160 IF A$ = "*" THEN CLOSE 1:GOTO 220
170 B$(T) = CHR$(ASC(A$) + D)
180 T = T + 1:GOTO 150
190 REM **
200 REM ** SAVE CODED FILE **
210 REM **
220 PRINT CHR$(147)
230 FOR X = 1 TO 10:PRINT:NEXT X
240 PRINT TAB(3);"[CMDR + 6]SAVE[CTRL + 2]
    CODED FILENAME:";
250 INPUT S$
260 OPEN 1,8,2,"@0:" + S$ + ",S,W"
270 FOR X = 1 TO T
280 PRINT#1,B$(X);
290 NEXT X
300 PRINT#1,"*";
310 CLOSE 1:PRINT:PRINT:PRINT TAB(16);
    "[CMDR + 3]FINITO![CTRL + 2]":END
320 PRINT:PRINT"[CMDR + 3]ENTER A NUMBER
    BETWEEN 1 AND 100 PLEASE[CTRL + 2]"
330 FOR P = 1 TO 1000:NEXT P
340 RETURN
```

Decoder

```
10 REM ** DECODER **
20 CLR:T = 1:DIM B$(5000):PRINT CHR$(147);
```


The Art of Programming

```
30 POKE 53281,6:POKE 53280,0
40 FOR X = 1 TO 5:PRINT:NEXT X:PRINT
   TAB(15);"[CTRL + 8]DECODER"
50 FOR X = 1 TO 5:PRINT:NEXT X
60 PRINT TAB(8);"[CMDR + 6]ENTER TODAY'S
   CODE:[CTRL + 2]";
70 INPUT D
80 IF D > 100 OR D < 1 THEN GOSUB
   310:GOTO 20
90 REM **
100 REM ** LOAD CODED TEXT FILE **
110 REM **
120 PRINT:PRINT TAB(8);"[CMDR + 6]CODED
   TEXT[CTRL + 2] FILENAME:";
130 INPUT F$
140 OPEN 1,8,2,"0:" + F$ + ",S,R"
150 GET#1,A$
160 IF A$ = "*" THEN CLOSE 1:GOTO 220
170 B$(T) = CHR$(ASC(A$)-D)
180 T = T + 1:GOTO 150
190 REM **
200 REM ** SAVE CLEAR TEXT FILE **
210 REM **
220 PRINT CHR$(147)
230 FOR X = 1 TO 10:PRINT:NEXT X
240 PRINT TAB(3);"[CMDR + 6]SAVE[CTRL + 2]
   CLEAR FILENAME:";
250 INPUT S$
260 OPEN 1,8,2,"@0:" + S$ + ",S,W"
270 FOR X = 1 TO T
280 PRINT#1,B$(X);
290 NEXT X
300 CLOSE 1:PRINT:PRINT:PRINT TAB(16);
   "[CMDR + 3]FINITO![CTRL + 2]":END
310 PRINT:PRINT"[CMDR + 3]ENTER A NUMBER
   BETWEEN 1 AND 100 PLEASE[CTRL + 2]"
320 FOR P = 1 TO 1000:NEXT P
330 RETURN
```

DR. WACKO PRESENTS COMMODORE BASIC

To Code Your Text

Important: Use only UPPERCASE LETTERS when you compose your text for coding on Wacko's Amazing Word Processor. Also, don't use an asterisk (*) in your text. Both programs recognize an asterisk as the end of text.



Run the code program and you'll be asked to "ENTER TODAY'S CODE." Enter any *whole* number between 1 and 100 and press RETURN. (Remember this number. You'll need it to decode your text!)

Next, you'll be asked for the filename of the text (stored on disk) that you want coded. Your screen will look like this:

CLEAR TEXT FILENAME: ?

Respond by entering the name of the text file stored on your disk and pressing RETURN.

FILENAME [RETURN]

When the red busy light on your disk drive goes off, the words, **SAVE CODED FILENAME: ?**, will appear on your screen.

Enter the filename you've chosen for your coded text. When FINITO! appears on your screen, a new file of coded text has been created on your disk. If you'd like to see what it looks like, just load it into your Miracle Word Processor!

To Decode

Run the Decode program, and you'll be asked to "ENTER TODAY'S CODE." Enter the same number you used when coding your text. Press RETURN.



The Art of Programming

Next, you'll be asked for the filename of the text (stored on disk) that you want to decode. Your screen will look like this.

CODED TEXT FILENAME: ?

Respond with the name of the text file stored on your disk and press RETURN.

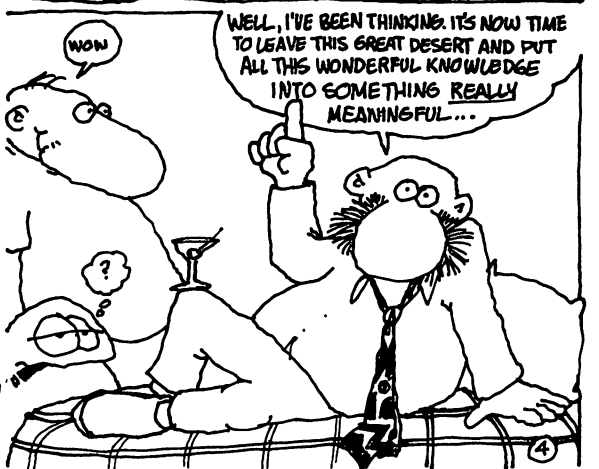
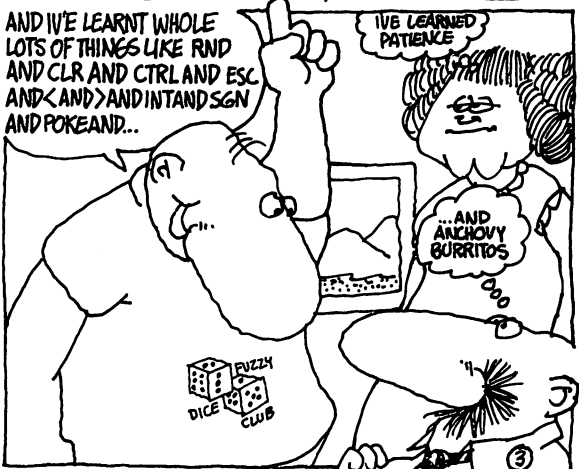
FILENAME [RETURN]

When the red busy light on your disk drive goes off, the words, **SAVE CLEAR FILENAME: ?**, will appear on your screen.

Enter the filename you've chosen for your decoded text. When FINITO! appears on your screen, a new file of decoded text has been created on your disk.

I've really enjoyed traveling with you through this wasteland of knowlege. But it's time to return to reality. So, until we meet again, enjoy your new word processor and secret Coder/Decoder, and bring along some suntan lotion next time! (what?)

DR. WACKO PRESENTS COMMODORE BASIC



Appendix A: Programming Conventions

Some of the programs in this book contain instructions within brackets. Here's what to do when you see them.

[CMDR + Key]: Press the Commodore key plus another key. Take a look at this short example.

```
10 PRINT "[CMDR + 6]DR. WACKO"
```

After typing the first quotation mark, press the Commodore key plus the 6 key, then type **DR. WACKO** and a final quotation mark. The result, when you run this one line program, will be a *green* **DR. WACKO**. How outrageous!

[CTRL + Key]: Press the CTRL key plus another key. Here's an example.

```
10 PRINT "[CTRL + 3]DR.[CTRL + 8] WACKO"
```

After the first quotation mark, press the CTRL key plus the 3 key, then type **DR.**, press the CTRL key plus the 8 key, a space, my last name and a final quotation mark.

When you run this program, **DR.** is printed in *red* and **WACKO** is printed in *yellow*. Good grief!

[CLRHM]: Press SHIFT plus the CLR/HOME key. Here's an example.

```
10 PRINT "[CLRHM]"
```

After the first quotation mark, press SHIFT plus the CLR/HOME key, then type a final quotation mark. Run this program and your screen will clear, and the cursor will wait for you in the upper-left corner of the screen.

DR. WACKO PRESENTS COMMODORE BASIC

[HOME]: Press the CLR/HOME key when you see this. Don't press SHIFT. This imbedded command sends the cursor "home" to the upper-left of your screen *without* clearing the screen.

[RVS ON]: Press the CTRL key plus the 9 key and all characters will print in *inverse video*.

[RVS OFF]: Press the CTRL key plus the 0 key to end *inverse* video printing.

[8*CRSR-D]: This strange convention means to imbed 8 CRSR Down's within quotations. Of course, the number and standard convention may vary, but the important thing here is the multiplication symbol, "*."

Cursory Movement

[CRSR-U]: Press SHIFT and the CRSR Up key

[CRSR-D]: Press SHIFT and the CRSR Down key

[CRSR-L]: Press SHIFT and the CRSR Left key

[CRSR-R]: Press SHIFT and the CRSR Right key

Summary of Conventions

When You See: Do This:

[CMDR + Key] Press Commodore key plus the key indicated.

[CTRL + Key] Press CTRL key plus the key indicated.

[CLRHM] Press SHIFT plus the CLR/HOME key.

[HOME] Press the CLR/HOME key.

[RVS ON] Press the CTRL key plus the 9 key.

[RVS OFF] Press the CTRL key plus the 0 key.

Appendix B: Saving, Loading and Naming Programs

Loading Your Programs

Loading Your Program from Tape

When you're ready to load your program into your computer from your cassette recorder type:

LOAD "PROGRAM NAME" [RETURN]

The "PROGRAM NAME" can be up to 16 characters in length. If you've forgotten the "PROGRAM NAME," just type **LOAD**, and the first program on the tape loads into your computer.

When you press **RETURN**, your screen says:

PRESS PLAY ON TAPE

Press the *PLAY* key and the screen clears, the tape starts spinning, and the border of your screen changes colors.

When the program you've chosen is found, the screen displays:

FOUND PROGRAM NAME

The program has been found. Press the Commodore key to load the program into your computer. If you'd like to quit the **LOADing** procedure in mid **LOAD**, just press the **RUN/STOP** key. When your program is safely loaded into your computer the word "READY" appears on your screen.

Loading Your Program from Disk

To load a program into your computer from your disk drive type:

DR. WACKO PRESENTS COMMODORE BASIC

LOAD "PROGRAM NAME",8 [RETURN]

The "PROGRAM NAME" can be up to 16 characters in length. By ending this short statement with a comma and the number 8 you're telling your computer to access the disk drive.

When you press RETURN your disk drive whirls, the busy light comes on, and the screen displays:

SEARCHING FOR PROGRAM NAME LOADING

When the program is safely loaded into your computer's memory the word "READY" appears on your screen.

Important Wacko Note: When you load a program into your computer's memory it obliterates any program or instructions that were in previously on its mind. Make sure that you save the program you were working on before you load a new one into your computer.

Saving Your Programs

Saving Your Program on Tape

When you're ready to save your Program on tape just type:

SAVE "PROGRAM NAME" [RETURN]

The "PROGRAM NAME" can be up to 16 characters long. After you press RETURN your screen says:

PRESS PLAY AND RECORD ON TAPE

Press both the *RECORD* and *PLAY* keys on your cassette recorder. Your screen clears and the border changes colors as your program is being stored.

Appendix C: ASCII CODES

When your program is safely stored on tape the word "READY" appears on your screen.

Saving Your Program on Disk

When you're all set to save your program on disk just type:

SAVE "PROGRAM NAME",8 [RETURN]

Again, the "PROGRAM NAME" can be up to 16 characters in length.

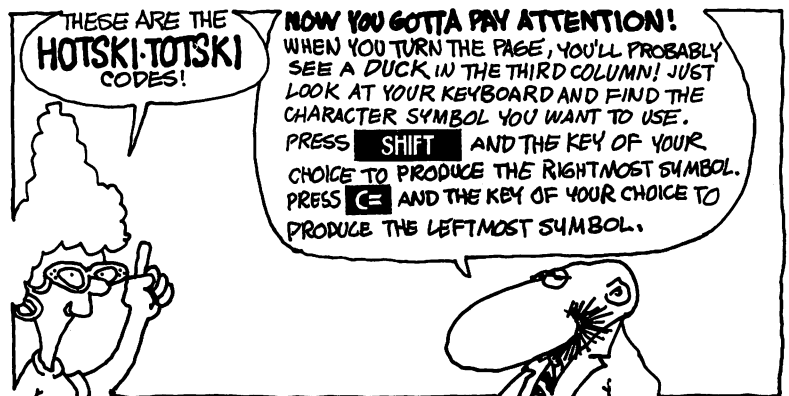
By ending this short statement with a comma and the number 8, you're letting your computer know that you want to save your program on disk.

When you press RETURN your disk whirls, the busy light comes on, and your screen says:

SAVING "PROGRAM NAME"

The busy light turns off when the program is safely tucked on your disk, and the message "READY" appears on your screen.

Appendix C: ASCII CODES



DR. WACKO PRESENTS COMMODORE BASIC

| Decimal Code | ASCII Character | Keys or Convention to Create Character | Decimal Code | ASCII Character | Keys or Convention to Create Character | Decimal Code | ASCII Character | Keys or Convention to Create Character |
|--------------|----------------------|--|--------------|-----------------|--|--------------|-----------------------------|--|
| 5 | WHITE CHARACTERS | CTRL+2 | 56 | 8 | 8 | 99 | | |
| 8 | DISABLE SHIFT | | 57 | 9 | 9 | 100 | | |
| 9 | ENABLE SHIFT | | 58 | : | : | 101 | | |
| 13 | END OF LINE | RETURN | 59 | ; | ; | 102 | | |
| 14 | SWITCH TO LOWER CASE | CMDR + SHIFT | 60 | < | < | 103 | | |
| 17 | CURSOR DOWN | CRSR-D | 61 | = | = | 104 | | |
| 18 | REVERSE PRINTING ON | CTRL+9 | 62 | > | > | 105 | | |
| 19 | HOME CURSOR | HOME | 63 | ? | ? | 106 | | |
| 20 | DELETE BACKSPACE | INST/DEL | 64 | @ | @ | 107 | | |
| 28 | RED CHARACTERS | CTRL+3 | 65 | A | A | 108 | | |
| 29 | CURSOR RIGHT | CRSR + R | 66 | B | B | 109 | | |
| 30 | GREEN CHARACTERS | CTRL+6 | 67 | C | C | 110 | | |
| 31 | BLUE CHARACTERS | CTRL+7 | 68 | D | D | 111 | | |
| 32 | SPACE | SPACEBAR | 69 | E | E | 112 | | |
| 33 | ! | ! | 70 | F | F | 113 | | |
| 34 | " | " | 71 | G | G | 114 | | |
| 35 | # | # | 72 | H | H | 115 | | |
| 36 | \$ | \$ | 73 | I | I | 116 | | |
| 37 | % | % | 74 | J | J | 117 | | |
| 38 | & | & | 75 | K | K | 118 | | |
| 39 | ' | ' | 76 | L | L | 119 | | |
| 40 | (| (| 77 | M | M | 120 | | |
| 41 |) |) | 78 | N | N | 121 | | |
| 42 | * | * | 79 | O | O | 122 | | |
| 43 | + | + | 80 | P | P | 123 | | |
| 44 | , | , | 81 | Q | Q | 124 | | |
| 45 | - | - | 82 | R | R | 125 | | |
| 46 | . | . | 83 | S | S | 126 | | |
| 47 | / | / | 84 | T | T | 127 | | |
| 48 | 0 | 0 | 85 | U | U | 129 | ORANGE CHARACTERS | CMDR + 1 |
| 49 | 1 | 1 | 86 | V | V | 133 | f1 | f1 |
| 50 | 2 | 2 | 87 | W | W | 134 | f3 | f3 |
| 51 | 3 | 3 | 88 | X | X | 135 | f5 | f5 |
| 52 | 4 | 4 | 89 | Y | Y | 136 | f7 | f7 |
| 53 | 5 | 5 | 90 | Z | Z | 137 | f2 | f2 |
| 54 | 6 | 6 | 91 | [| [| 138 | f4 | f4 |
| 55 | 7 | 7 | 92 | £ | £ | 139 | f6 | f6 |
| | | | 93 |] |] | 140 | f8 | f8 |
| | | | 94 | ↑ | ↑ | 141 | SHIFTED RETURN | SHIFT + RETURN |
| | | | 95 | ← | ← | 142 | SWITCH TO UPPER CASE | CMDR + SHIFT |
| | | | 96 | | | | | |
| | | | 97 | | | | | |
| | | | 98 | | | | | |

Appendix D: POKES & PEEKS

| Decimal Code | ASCII Character | Keys or Convention to Create Character | Decimal Code | ASCII Character | Keys or Convention to Create Character | Decimal Code | ASCII Character | Keys or Convention to Create Character |
|--------------|----------------------------|--|--------------|-------------------|--|--------------|-----------------|--|
| 144 | BLACK CHARACTERS | CTRL + 1 | 156 | PURPLE CHARACTERS | CTRL + 5 | 173 | | |
| 145 | CURSOR-UP | CRSR-U | 157 | CURSOR LEFT | CRSR-L | 174 | | |
| 146 | REVERSE PRINTING OFF | CTRL + 0 | 158 | YELLOW CHARACTERS | CTRL + 8 | 175 | | |
| 147 | CLEAR SCREEN & HOME CURSOR | CLRHM | 159 | CYAN CHARACTERS | CTRL + 4 | 176 | | |
| 148 | INSERT CHARACTER | SHIFT + INST/DEL | 160 | SPACE | SPACE BAR | 177 | | |
| 149 | BROWN CHARACTERS | CMDR + 2 | 161 | | | 178 | | |
| 150 | LIGHT RED CHARACTERS | CMDR + 3 | 162 | | | 179 | | |
| 151 | GRAY 1 CHARACTERS | CMDR + 4 | 163 | | | 180 | | |
| 152 | GRAY 2 CHARACTERS | CMDR + 5 | 164 | | | 181 | | |
| 153 | LIGHT GREEN CHARACTERS | CMDR + 6 | 165 | | | 182 | | |
| 154 | LIGHT BLUE CHARACTERS | CMDR + 7 | 166 | | | 183 | | |
| 155 | GRAY 3 CHARACTERS | CMDR + 8 | 167 | | | 184 | | |
| | | | 168 | | | 185 | | |
| | | | 169 | | | 186 | | |
| | | | 170 | | | 187 | | |
| | | | 171 | | | 188 | | |
| | | | 172 | | | 189 | | |
| | | | | | | 190 | | |
| | | | | | | 191 | | |

Codes 192-223 are the same as 96-127
 Codes 224-254 are the same as 160-190
 Code 255 is the same as 126
 Codes 0-4, 6, 7, 10-12, 15, 16, 21-27, 128, 130-132 and 143 are not used.

Appendix D: Sneaky Peek's POKES & PEEKS

This appendix includes a few of my favorite memory locations. You'll find them useful as you continue to explore the power of your Commodore 64 computer.

The PEEK function lets you "look" into a memory location and read its contents. The POKE statement lets you stuff information directly into a memory location.

DR. WACKO PRESENTS COMMODORE BASIC

788,52 — Disable the RUN/STOP Key

If you've got a program that requires lots of key board entry, you might want to disable the RUN/STOP key so someone's fingers don't accidentally stop the program. To disable the RUN/STOP key just:

POKE 788,52

Here's a short program that shows you how this works:

```
10 POKE 788,52  
20 GOTO 20
```

Run this program and try to break into it by pressing the RUN/STOP key. You can't!. The only way to stop it is by pressing the RUN/STOP key *and* the RESTORE key.

If you want to get real sneaky, and disable the RESTORE key as well, just:

POKE 792,808 If you disable the RUN/STOP *and* RESTORE keys, there's no way to break into the program. Uh oh!

203 — What Key Did You Press?

PEEK(203) returns a different value for each key you press. You can use **PEEK(203)** in an IF/THEN statement to tell your program to do something when you press a specific key. Here's what I mean.

First run this program, press some keys, and watch the results:

```
10 PRINT PEEK(203):GOTO 10
```

Now we'll use this PEEK in a short example:

```
10 IF PEEK(203) = 49 THEN GOTO 30  
20 GOTO 10
```

Appendix D: POKES & PEEKS

```
30 FOR X = 1 TO 16:POKE 53281,X:NEXT X
40 POKE 53281,6:GOTO 10
```

Just press the asterisk key (*) a few times and watch the glorious results!

53281 and 53280 — The Background and Border Colors

POKE 53281 with a number from 0 to 15 to change the background color.

POKE 53280 with a number from 0 to 15 to change the border color.

Check out the Graphics chapter for more colorful events.

204 — Turn On That Cursor!

If you really want to turn on your cursor, like really, use one of my favorites: **POKE 204,0**.

Here's how it's used. First enter and run this short program. Then get **POKE 204** into the act and you'll see the difference.

```
10 PRINT "I'M WATING FOR YOU";
20 REM
30 GET A$:IF A$ = "" THEN GOTO 30
40 PRINT:PRINT "HIGH GANG!"
```

Run this program and and press *any* key to see the results. Did you notice that the cursor wasn't waiting behind the word "YOU?" To make it appear, just enter this line 20:

```
20 POKE 204,0
```

Now rerun the program. There's a cursor blinking at you!

DR. WACKO PRESENTS COMMODORE BASIC



































































There Are Hundreds!

There hundreds and hundreds of PEEKs and POKEs that you can use to totally confuse your computer and astound your neighbors. This Appendix just gives you a feel for their power. Now that you know how to use them, and have seen some of my favorites, rush down to your local book store or library and find a book that lists all these important locations! There are many available.

Appendix E: SCREEN DISPLAY CODES

| SET 1 | SET 2 | POKE | SET 1 | SET 2 | POKE | SET 1 | SET 2 | POKE |
|-------|-------|------|------------------|-------|------|-------|-------|------|
| @ | | 0 | R | r | 18 | \$ | | 36 |
| A | a | 1 | S | s | 19 | % | | 37 |
| B | b | 2 | T | t | 20 | & | | 38 |
| C | c | 3 | U | u | 21 | ' | | 39 |
| D | d | 4 | V | v | 22 | (| | 40 |
| E | e | 5 | W | w | 23 |) | | 41 |
| F | f | 6 | X | x | 24 | * | | 42 |
| G | g | 7 | Y | y | 25 | + | | 43 |
| H | h | 8 | Z | z | 26 | , | | 44 |
| I | i | 9 | [| | 27 | — | | 45 |
| J | j | 10 | £ | | 28 | . | | 46 |
| K | k | 11 |] | | 29 | / | | 47 |
| L | l | 12 | ↑ | | 30 | 0 | | 48 |
| M | m | 13 | ← | | 31 | 1 | | 49 |
| N | n | 14 | SPACE BAR | | 32 | 2 | | 50 |
| O | o | 15 | ! | | 33 | 3 | | 51 |
| P | p | 16 | " | | 34 | 4 | | 52 |
| Q | q | 17 | # | | 35 | 5 | | 53 |

Appendix E: SCREEN DISPLAY CODES

| SET 1 | SET 2 | POKE | SET 1 | SET 2 | POKE | SET 1 | SET 2 | POKE |
|---|-------|------|---|---|------|---|---|------|
| 6 | | 54 |  | O | 79 |  | | 104 |
| 7 | | 55 |  | P | 80 |  |  | 105 |
| 8 | | 56 |  | Q | 81 |  | | 106 |
| 9 | | 57 |  | R | 82 |  | | 107 |
| : | | 58 |  | S | 83 |  | | 108 |
| ; | | 59 |  | T | 84 |  | | 109 |
| < | | 60 |  | U | 85 |  | | 110 |
| = | | 61 |  | V | 86 |  | | 111 |
| > | | 62 |  | W | 87 |  | | 112 |
| ? | | 63 |  | X | 88 |  | | 113 |
|  | | 64 |  | Y | 89 |  | | 114 |
|  | A | 65 |  | Z | 90 |  | | 115 |
|  | B | 66 |  | | 91 |  | | 116 |
|  | C | 67 |  | | 92 |  | | 117 |
|  | D | 68 |  | | 93 |  | | 118 |
|  | E | 69 |  |  | 94 |  | | 119 |
|  | F | 70 |  |  | 95 |  | | 120 |
|  | G | 71 | SPACE BAR | | | 96 | | 121 |
|  | H | 72 |  | | 97 |  |  | 122 |
|  | I | 73 |  | | 98 |  | | 123 |
|  | J | 74 |  | | 99 |  | | 124 |
|  | K | 75 |  | | 100 |  | | 125 |
|  | L | 76 |  | | 101 |  | | 126 |
|  | M | 77 |  | | 102 |  | | 127 |
|  | N | 78 |  | | 103 | | | |

DR. WACKO PRESENTS COMMODORE BASIC

INDEX

| | | | | | |
|--------------------------------|-------------------------------|---------------------------|------------------------------------|---------------------------|---------------------------|
| Absolutely Wacko Program | 43 | FILENAME | 220, 238-239 | Print to the Screen | 214-216 |
| Arithmetic | 27-33 | Flag | 84 | PRINT# | 142-149, 151-153 |
| Arrays | 92-100 | FOR/NEXT Loop | 69-73, 145 | Printer Routine | 223-224 |
| ASC | 136 | nested | 73-74 | Printeroonio Program | 149 |
| ASCII Codes | 131-133 | data reading | 91 | Programming, The Art of | 208-239 |
| codes, characters and | | arrays | 94, 98 | Programming Mode | 34 |
| keystrokes chart | 245-247 | matrix | 101 | Random Music | 206 |
| Attack | 188-189, 193-197 | GET# | 142-147 | Random Nonsense Generator | 124 |
| Be Cool Program | 123 | GOSUB | 75-76, 79 | Random Number Generator | 110-113 |
| Bijou Billboard Program | 157 | GOTO | 61-63 | Random Problem Generator | 113-114 |
| Brilliant Wacko Program | 34-38 | Graphics Characters | 14 | READ | 82-83, 85, 91, 94-95, 101 |
| Calorie Counter Program | 96-97 | IF/THEN | 65-69, 99 | Read and Review | 217-218 |
| Camel Latin | 128-131 | and sound | 204 | Release | 191-194, 195, 197-199 |
| Captain Action's Hallucination | 74 | Immediate Mode | 24 | REM | 44 |
| Carefree Program | 121-122 | INPUT | 58-61, 68, 98, 101 | Rescue Lawrence Program | 115-118 |
| Cassette | 140-141 | INPUT# | 147-148 | RESTORE | 20, 25, 38, 86-88 |
| cassette save | 221 | INSERT | 18 | Reverseroonio Program | 127-128 |
| Channels, input and output | 139 | INT | 108-110 | RIGHT\$ | 121-124 |
| CHRS\$ | 133-135 | Inverse Character Key | 20-21 | RND | 110-114, 125 |
| Clear Screen | 224 | Keyboards | 10 | RUN/STOP | 20, 35-36 |
| CLOSE | 142 | Lazy Loader Program | 146 | RVS OFF | 20-21 |
| CLR | 53 | LEFT\$ | 121-123 | RVS ON | 20-21 |
| Clyde's Camel Racetrack | | LEN | 125-127 | Saving Text | 218-221 |
| Program | 178-179 | LET | 47-48 | Sawtooth | 187, 193, 200 |
| Clyde's Lament Program | 206 | Limiter Program | 126 | Screen Graphics | 166-169 |
| Coder/Decoder Program | 235-237 | Line Annihilation | 40 | Semicolon | 45 |
| Colon | 43 | Line Numbers | 34 | SGN | 114-115 |
| Color | 16-155 | LIST | 38-39 | SHIFT | 12-13 |
| numbers | 169 | Loading Text | 221-223 | Sound, Type of | 187-188 |
| Comma | 43 | Marrakesh Express Program | 202-203 | Special Function Keys | 21 |
| Commodore Key | 12-14 | Matrix | 100-107 | STEP | 71-73 |
| and colors | 16 | Matrix Stuffer Program | 102 | Storing Data | 81 |
| CONTinue | 35 | MID\$ | 121-125 | Strings | |
| CTRL Key | 15, 39-40 | Modular Approach to | | comparisons | 68-69 |
| and colors | 16 | Programming | 213-224 | limiters | 125-127 |
| Coordinates | 161 | Musical POKE Chart | 200 | manipulators | 121-125 |
| Counting Shekels Program | 62 | NEW | 40-41 | tying strings together | 136-137 |
| Cursor | 11-12, 17, 18, 162-164 | Noise | 187-188, 193, 200 | variables | 50-52, 53, 60-61 |
| DATA | 81-82, 87, 88, 91, 94-95, 101 | Numeric Variables | 46-50, 52 | Super Saver Program | 143-144 |
| strings of DATA | 84-85 | ON GOSUB | 79 | Sustain | 190-191, 193-195, 197-199 |
| changing sound | 204-205 | ON GOTO | 77-79 | TAB | 162-163 |
| Database | 90 | OPEN | 139-142 | Triangle | 187, 192, 200 |
| Decay | 189-190, 193-197 | Parentheses Levels | 32 | Two-line Limit | 41-42 |
| DELETE | 19 | Pause Program | 72-73 | Universe Program | 179-183 |
| DIMension Statement | 93, 100-101 | PEEK | 153-155, 247-250 | Use 'm All Program | 66-67 |
| Disk Drive | 141-142, 150-153 | Peripheral Devices | 139 | Wacko Unified Hole Theory | 160 |
| Easy Sound Program | 201 | Pixels | 160-161 | Wacko's Amazing Word | |
| Editing | 17-19, 216-217 | POKE | 153-155, 167-170, 200-201, 247-250 | Processor Program | 230-234 |
| Erase Memory | 224 | PRINT | 25, 26 | Waveforms | 200-201 |
| Falafel Counter Program | 59 | | | Waveform Control Settings | 200 |

Dr. C. Wacko Presents
COMMODORE 64 BASIC
and the
Whiz-Bang Miracle Machine

Does Commodore BASIC make you a little dry in the mouth? Wondering whether the only thing writing your own programs will ever do for you is make you hot under the collar? Did reading that last BASIC programming book feel like a jog across the Sahara at high noon? Dr. C. Wacko (and friends) to the rescue!

Yes, folks, back from safari by popular demand, it's *Dr. C. Wacko Presents Commodore 64 BASIC and the Whiz-Bang Miracle Machine*. Now, even people who don't know figs about computers can learn to program and have fun with their Commodore. "But real programming can't be fun," you say? Enter the sun-drenched world of the good doctor and you'll not only get a great tan, but you'll:

- grunt your way through Basic BASIC Training and get in shape for your programming adventure
- wander through the Great White Expanse, drinking from the Well of Programming Knowledge at every oasis
- don sunglasses and enter the world of Commodore 64 graphics
- create enchanting sounds and haunting melodies with Commodore sound
- weave the perfect flying carpet while learning the Art of BASIC programming

Create word processors, computer files, games, encoders and decoders, and even models of the universe? Without knowing a byte of BASIC? Yup, Dr. C. Wacko and his band of desert renegades make it all possible.

**Take the
Wacko Challenge!**

Pick up any other BASIC programming book on the shelf. No, not that one, the one over there. Open it. Now open *Dr. C. Wacko Presents Commodore 64 BASIC and the Whiz-Bang Miracle Machine*. Which one will you trust to guide your programming career? Well, you're not alone, since surveys show that BASIC programmers prefer the Wacko method 3-1.

