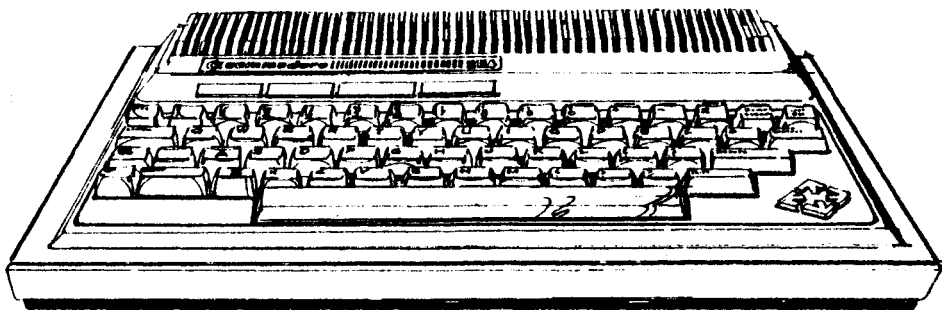


# Commodore 264 Series

---



## Preliminary Users Manual

 **commodore**  
COMPUTERS

## **USER'S GUIDE STATEMENT**

"This equipment generates and uses radio frequency energy. If it is not properly installed and used in strict accordance with the manufacturer's instructions, this equipment may interfere with radio and television reception. This machine has been tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. If you suspect interference, you can test this equipment by turning it off and on. If you determine that there is interference with radio or television reception, try one or more of the following measures to correct it:

- reorient the receiving antenna
- move the computer away from the receiver
- change the relative positions of the computer equipment and the receiver
- plug the computer into a different outlet so that the computer and the receiver are on different branch circuits.

If necessary, consult your Commodore dealer or an experienced radio/television technician for additional suggestions. You may also wish to consult the following booklet, which was prepared by the Federal Communications Commission:

"How to Identify and Resolve Radio-TV Interference Problems". This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

**PRELIMINARY**

# INTRODUCTION

## YOU'LL ADORE YOUR COMMODORE 264!

Welcome to the Commodore family! Whether you're a beginner or an expert, you'll be surprised how easy it is to use your Commodore 264 computer. The Commodore 264 is easy to program . . . but you don't have to be a programmer to use it. You don't even have to know how to type!

Commodore's full selection of COMMODORE 264 software makes it easy to use your computer for business, home and educational applications. (And don't forget the games!)

## You Bought The Best Computer

These special built-in features make your COMMODORE 264 the best home computer you can buy:

- 64K RAM (60K available for BASIC programming)
- Full Typewriter Style Keyboard
- Optional Built-in Software
- Screen Window Capability
- HELP Key
- 8 Programmed, Reprogrammable Function Keys
- Four Separate Cursor Keys
- Uses Most COMMODORE 64 and VIC-20 Peripherals
- 128 Colors (16 primary colors, 8 luminance levels)
- Over 75 BASIC Commands
- High Resolution Graphics Plotting
- Split-Screen Text With High-Res Graphics
- Graphic Character Set On Keyboard
- Keyboard Color Controls
- 320 x 200 Pixel Screen Resolution
- Reverse and Flashing Characters
- 2 Tone Generators
- Built-In Machine Language Monitor (13 commands)

## What You Can Do With Your Commodore 264

The key to any computer is SOFTWARE, and Commodore has a full "menu" of software available from your Commodore dealer. Specific applications include wordprocessing, financial calculations, learning activities, mail lists, project planning, recordkeeping, home budget, investment analysis and much more. Some software products combine several different functions, such as "Triology," which provides wordprocessing, an electronic spreadsheet, database AND graphics!

Many models of the Commodore 264 have software built into the machine. You can also buy Commodore software on plug-in cartridges, floppy diskettes and cassette tapes. And of course, you can learn to program your computer in BASIC, which comes built into your computer, or in other popular computer languages such as LOGO.

## **Built-in Software**

You may have purchased your Commodore 264 with one of several built-in software packages that are ready to use as soon as you turn on your computer. If your computer has built-in software, a separate manual describing how to use the software is included. Built-in software on the Commodore 264 is a Commodore first!

## **Commodore Software on Cartridge**

For a long time, video games were the only software programs available on cartridge. Now Commodore brings you large application programs such as wordprocessing and business programs on cartridge...at prices you can afford. Cartridges are convenient because all you have to do is plug them in and start computing; you don't need any additional equipment. There are other advantages as well...for example, an electronic spreadsheet on cartridge gives you more working space than an electronic spreadsheet on disk.

## **Software on Disk and Tape**

Commodore software also comes on floppy diskettes and on cassette tape. The Commodore Disk Drive lets you run Commodore disk-based software and the Commodore Model 1531 Datassette lets you run tape-based software.

## **Programming in BASIC — Over 50 New Commands**

It's easy to program your COMMODORE 264. Start by working through the exercises included in this manual, and if you want to get more involved in programming, Commodore offers several self-teaching courses such as INTRODUCTION TO BASIC and our BOOKWARE™ series of books and tutorials.

The BASIC on your Commodore 264 is the most powerful BASIC ever built into any Commodore home computer...it includes MORE THAN FIFTY NEW BASIC COMMANDS including full GRAPHICS PLOTTING and PROGRAM EDITING.

## **Programming in LOGO — Turtle Graphics and More**

Many new computerists prefer to learn programming with LOGO, an easy computer language created for beginners, which features the use of a "turtle" to draw pictures and create graphs and charts. LOGO is rising rapidly in popularity, especially in schools, and Commodore has a strong commitment to this "friendly" programming language.

## **The MAGIC DESK**

If you're a new computerist and you want to use your computer for wordprocessing, calculations and other functions, but you really don't want to learn any special commands or instructions...try MAGIC DESK. This unique Commodore software system uses PICTURES of familiar objects instead of written commands, and uses the computer keyboard like a typewriter. The software is automatically linked to your disk drive and printer so, for example, all you have to do is "point" at the picture of the printer to print out a paper copy of something you've typed. Ask your dealer for a demonstration.

## **Creating A Complete Computer System**

As you develop your complete computer SYSTEM, you'll find that Commodore peripherals are priced so you can afford them, and designed with superior features. Here are the major peripherals included in most typical Commodore home computer systems:

Computer: Commodore 264

Display: Commodore Color Monitor (or your television set)

Storage: Commodore Datassette (tape recorder) or Commodore Disk Drive

Printer: Commodore Printer (several models) or 1520 Printer/Plotter

Modem: Commodore VICMODEM or Commodore AUTOMODEM

Controller: Commodore Joysticks

## **Where To Go From Here?**

By now you've done enough reading and you want to get started. Here's what we recommend you do first: Send in your warranty card. Subscribe to the Commodore magazine so you get the latest information on your computer, and join the Commodore National User Club. Read this manual and try the exercises. Try some software. Keep checking in with the Commodore dealers in your area for new developments in software, bookware™ and peripherals. Enjoy!

# TABLE OF CONTENTS

<b>CHAPTER 1</b>	Unpacking and Setting Up the Commodore 264 .....	1
<b>CHAPTER 2</b>	Using the Keyboard and the Screen .....	8
<b>CHAPTER 3</b>	Getting to Know Your Commodore 264 .....	16
<b>CHAPTER 4</b>	Using Software .....	29
<b>CHAPTER 5</b>	Beginning BASIC .....	37
<b>CHAPTER 6</b>	Using Graphics and Color .....	47
<b>CHAPTER 7</b>	Making Sound and Music .....	67
<b>CHAPTER 8</b>	BASIC Tricks .....	75
<b>APPENDICES</b>	.....	93
<b>A</b>	Error Messages .....	94
<b>B</b>	BASIC 3.5 Commands, Statements, and Functions .....	99
<b>C</b>	BASIC 3.5 Abbreviations .....	130
<b>D</b>	TEDMON .....	133
<b>E</b>	Converting BASIC Programs to Commodore BASIC 3.5 .....	139
<b>F</b>	Memory Register Map .....	140
<b>G</b>	Musical Note Table .....	142
<b>H</b>	ASCII and CHR\$ Codes .....	143
<b>I</b>	Screen Display Codes .....	146
<b>J</b>	Screen and Color Memory Maps .....	148
<b>K</b>	Deriving Mathematical Functions .....	150
<b>L</b>	Programs To Try .....	151
<b>M</b>	Book List .....	154
<b>INDEX</b>	.....	156

# CHAPTER 1

## UNPACKING AND SETTING UP THE COMMODORE 264

- Unpacking your Commodore 264.
- Getting to know the switches and sockets.
- Setting up your Commodore 264.
- Troubleshooting chart.

## UNPACKING YOUR COMMODORE 264

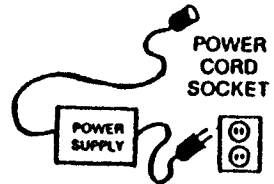
Now that you've opened the box containing your new C-264 box and found this manual, the first thing that you should do is check to make sure that you have all the items on this list. You should have:

1. Your Commodore 264

2. The power supply

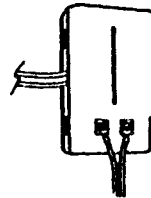
This is a black box with a line cord coming out of one end, and a round plug coming out the other. The round plug goes into the back of the C-264.

The other end has a standard three-prong line plug that goes into a regular wall socket.



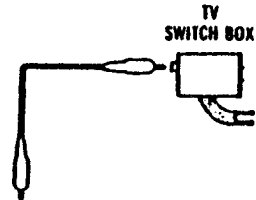
3. The TV switchbox

This is the silver and black box that connects to your antenna jack on the back of your TV. You don't need the switchbox if you plan to connect your C-264 to a monitor.



4. The RF cable

This is a thin black cable, with a single prong jack at each end, used to connect the TV switchbox to the RF output jack on the side of the C-264. You don't need this cable if you are planning to connect your C-264 to a monitor.



5. The user's guide  
(Obviously!)

6. Other assorted literature, such as the  
warranty card, etc.

If you don't find all these items in the box, check with your dealer immediately for replacements.

Before you connect anything, you should look over these drawings of your computer. These drawings identify all the outlets so you can set up your computer system quickly and easily.

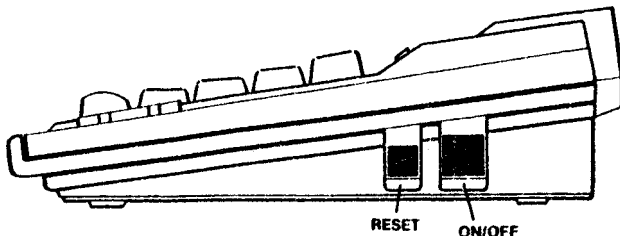


# GETTING TO KNOW THE SWITCHES AND SOCKETS

## THE RIGHT SIDE OF YOUR C-264

### THE ON/OFF SWITCH

Your C-264 should be turned off when you install or remove cartridges or any peripheral device such as a printer or disk drive. The OFF position is marked so you can be sure when your machine is off.



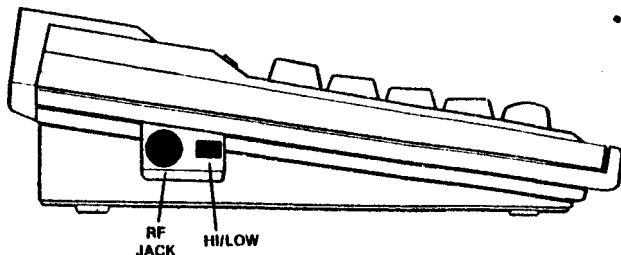
### THE RESET BUTTON

There are two ways to use the RESET button:

1. You can use the RESET button to clear your screen and reset your computer as if you'd just turned it on. Just press the reset button once. Be careful: when you press the reset button, you will lose any BASIC program currently in memory.\*
2. If you want to reset your Commodore 264 and KEEP your BASIC program, hold down the RUN/STOP key and then press the RESET button. When you do this, your C-264 goes to the built in machine language monitor. Type an X and press the RETURN key to get back to BASIC. Your program is still completely intact in the C-264 memory. Just type LIST to display the program on your screen.

\*When you press RESET, C-264 automatically issues a NEW command. This can be reversed. See the Commodore 264 Programmer's Reference Guide for information on UNNEWing your program after you've pressed the reset button by accident.

## THE LEFT SIDE OF YOUR C-264



The socket and the switch on the left side of the C-264 (if you're facing it) both have to do with TV connections. Neither is used if you're connecting your C-264 to a monitor.

### THE RF JACK

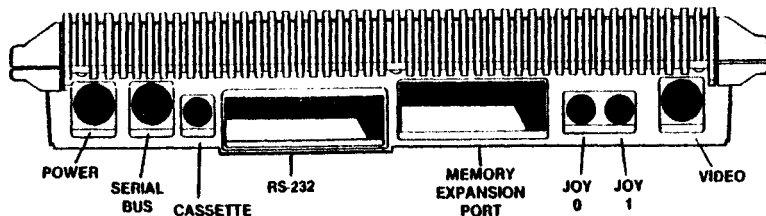
This is where you plug one end of the RF cable (the thin black cable). You can plug either end into this socket and the other end into the TV switch box.

### THE HIGH/LOW SWITCH

This switch controls which channel will display your C-264 video output. Set the H/L switch to L for output on channel 3. Set the H/L switch to H for video on channel 4.

You can use either channel 3 or 4 on your TV to display the video picture from your computer. If you have a channel 3 TV station in your area, select channel 4, and vice versa. You'll probably want to experiment to see which setting gives you the best picture.

## THE BACK OF YOUR COMPUTER



The sockets on the back of your computer connect a variety of accessories to your Commodore 264. Each connector is different. Be sure you plug each accessory into the right socket.

### THE POWER SOCKET

The round end of the power cable fits in here. The power cable is the one connected to the power supply box. Plug the other end into a standard wall socket for three-prong plugs.

### THE SERIAL BUS

You can plug a disk drive or a printer into this socket. If you want to plug in both, plug the disk drive in here, and then plug the printer into the back of the disk drive.

### THE CASSETTE PORT

The Commodore cassette tape recorder plugs in here.

### THE RS-232 PORT

Accessories such as a modem and an RS-232 adapter plug in here.

### THE MEMORY EXPANSION PORT

C-264 software cartridges and the C-264 fast disk drive plug in here. When you install or remove cartridges, be sure to turn off your C-264.

### JOY 0 AND JOY 1: THE GAME PORTS

You can plug joy sticks into these sockets. The Commodore 264 uses specially designed joy sticks available from your Commodore dealer.

### THE VIDEO SOCKET

This is where you plug in the cable that connects a monitor to your C-264. Although this socket is an 8-pin connector, you can use a 5-pin cable in this socket as well. The Commodore 1700 Series Color Monitor comes with an 8-pin cable for use with the C-264.

## **SETTING UP YOUR C-264**

- You need to set up your C-264 somewhere with access to at least two wall plugs: one for your C-264 and one for your TV or monitor.
- If you're installing a disk drive and a printer, you'll need additional wall plugs.
- Your C-264 needs a place to sit that is a comfortable distance from your TV.
- Make sure that your computer is OFF before you start the setup. Check the ON/OFF switch on the side of the C-264 to be sure.

## **CONNECTING YOUR C-264 TO YOUR TV**

If you are connecting the C-264 to a television set, you will need a small screwdriver to attach the TV switchbox. The way you connect the switchbox depends on what type of antenna connection your TV set has.

**NOTE:** If your antenna is connected to your TV by a single round-ended cable (the 75-ohm co-ax type), you will need either the 300 ohm to 75 ohm adapter, which came with your TV, or you must get a replacement 75 ohm to 75 ohm switchbox. The adapter is a small plastic part with a co-ax connector on one side and two screws on the other. If you do not have one, it is available at most electronics stores. Once you attach the adapter to the co-ax connector on your set, you can follow the rest of these instructions.

- STEP 1.** Disconnect the antenna from your TV: use a screwdriver to loosen the screws on the TV. Remove the two antenna leads.
- STEP 2.** Connect the TV switch box to the TV where the antenna leads were: use a screwdriver to attach the leads on the box to the antenna input screws on your TV.
- STEP 3.** Connect the antenna to the switch box: use a screwdriver to attach the leads from the antenna to the screws on the switch box.

If you have the round coax type antenna connection on your TV, and you have a replacement 75-ohm TV switch box:

- STEP 1.** Disconnect the antenna from your TV: unscrew the antenna wire. Just unthread it by hand.
- STEP 2.** Connect the switch box to the TV: hand turn it onto the antenna input post on your TV.
- STEP 3.** Connect the antenna to the switch box: hand turning the antenna cable into the switch box.

Once the switch box is in place, get the RF cable (the thin, jack-ended cable) that came with your C-264. Plug one end into the socket in the top of the switch box. Connect the other into the socket marked RF on the left side of your computer.

Once you've attached the switch box to the TV, you never have to do it again. When you want to use the computer, just move the switch to **COMPUTER**. When you want to watch TV, just move the switch to the **TV** position. The switch box will not interfere with your TV reception.

## **SELECTING A CHANNEL ON YOUR TV**

As we explained earlier, your TV should be set on either channel 3 or 4 when you are using your computer. Don't choose a channel that broadcasts in your area. If you use channel 3, set the H/L switch on the side of the computer to **L**. If you use channel 4, set this switch to **H**.

## **CONNECTING YOUR COMMODORE 264 TO A MONITOR**

If you're connecting your computer to a monitor instead of a TV, follow the instructions in the manual that comes with the monitor. Hooking up a monitor like the Commodore 1703 Color Monitor is simple. It requires only one cable that goes directly from your monitor to the **VIDEO** socket on the back of your computer.

## **FINAL STEPS**

1. Attach the power supply cable with the power box to your C-264. Plug the round end of the cable into the **POWER** socket on the back of the computer; plug the power supply into the wall socket.
2. If you are using a TV, make sure that the setting on the modulator (**H/L**), and the channel on your TV (3 or 4) are in agreement. Make sure that the switchbox is set to the **COMPUTER** setting.

If you are using a 1702 or 1703 monitor, use the rear jacks, and check that the back/front switch is set to **BACK**.

3. Turn on your computer. (The switch is on the right side as you face the C-264.)
4. If all went well, this message will appear on your screen:

**COMMODORE BASIC 3.5 60671 BYTES FREE**  
**READY.**

The flashing cursor under the **READY** message tells you that the C-264 is waiting for you to start typing. The background color will be white, while the letters will be printed in black.

5. Check the troubleshooting chart if you have problems. You may need to adjust your TV set to get a sharper picture.

## TROUBLESHOOTING CHART

Symptom	Cause	Remedy
Indicator Light not "On"	Computer not "On"	Make sure power switch is in "On" position
	Power cable not plugged in	Check power socket for loose or disconnected power cable.
	Power supply not plugged in	Check connection with wall outlet
	Bad fuse in computer	Take system to authorized dealer for replacement of fuse .
No picture	TV on wrong channel	Check other channel for picture (3 or 4)
	Incorrect hookup	Computer hooks up to VHF antenna terminals
	Video cable not plugged in	Check TV output cable connection
	Computer set for wrong channel	Set computer for same channel as TV (3 or 4)
Random pattern on TV with cartridge in place	Cartridge not properly inserted	Reinsert cartridge after turning off power
Picture without color	Poorly tuned TV	Retune TV
Picture OK, but no sound	TV volume too low	Adjust volume of TV
	Aux. output not properly connected	Connect sound jack to aux. input on amplifier and select aux. input

**NOTE:** Some TV sets cannot display the entire C-264 screen. Instead, their picture crops off the leftmost and rightmost column of the C-264 screen display. We recommend you use a different TV set or a monitor such as the Commodore 1702 or 1703 color monitor.

If this is not possible you can remedy this problem by pressing the ESCape and N keys. This reduces the computer screen display size so the entire picture can appear. You must repeat this each time you turn on your computer.

## CHAPTER 2

# USING THE KEYBOARD AND THE SCREEN

- A tour of the keyboard.
- The special keys.
- The graphics keys.
- The programmable function keys.
- The HELP key.

## TRY TYPING THIS ON YOUR KEYBOARD:

1 PRINT "your name"                      press the **RETURN** key  
2 PRINT "HAS A NEW 264"                press the **RETURN** key  
RUN    press the **RETURN** key

## A TOUR OF THE KEYBOARD

Many of the keys on the C-264 keyboard are like the keys on a typewriter, but every key can do more than a typewriter can. In this section, we'll show you how to use the special keys, like the **C** key and the cursor arrow keys. And we'll show you the extra powers of every key, including how to print the graphic symbols on the fronts of many of the keys.

While we guide you on the tour of the C-264 keyboard, you should find the keys and practice using them.

## THE SPECIAL KEYS

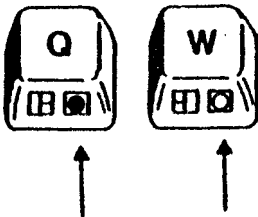
### **RETURN**

You have to press the RETURN key at the end of each line of instructions you enter on your Commodore 264 keyboard. You might think of this key as an ENTER key because RETURN actually enters information and instructions into the computer.


### **SHIFT**

This key works like the shift key on a regular typewriter. Your C-264 has two SHIFT keys and a SHIFT LOCK, which works like the shift lock on a typewriter.

With the SHIFT key, you can get the graphic symbol on the right side on each graphics key when you are in uppercase/graphics mode.



Your C-264 is automatically in uppercase/graphics mode when you turn it on. In uppercase/graphics mode, all the letters appear uppercase when typed without the SHIFT key.

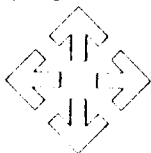
The SHIFT key gets upper case letters when you are in upper/lowercase text mode. You can tell you're in this mode when the letters you type are in lower case except when you use the SHIFT key. You can go back and forth between uppercase/graphics and upper/lowercase text modes by pressing the SHIFT and  key at the same time.

#### **RUN STOP**

Press this key to tell your C-264 to STOP what it is doing and give control back to you. When the C-264 is running a BASIC program, you can stop the program with this key.

When you hold down the SHIFT key and type this key, RUN tells the C-264 to load and run the first program on a disk in the disk drive.

#### **THE CURSOR KEYS**



It's easy to move the cursor quickly around the screen in any direction. Just press the cursor arrow key that points in the direction you want to go. Like all keys on the C-264 keyboard, each cursor key has a REPEAT feature. This automatic repeat keeps the cursor moving until you release the key.

**NOTE:** You can move the cursor over letters and numbers on the screen without affecting those characters.

#### **INST/DEL**

You can INSERT and DELETE characters from the line you are typing by pressing this key. When you press DEL, that character immediately to the left of the cursor disappears, and the cursor moves over to where the missing character was. You can use the cursor keys to go back to the middle of a line and then use DEL to DELETE a letter. When you do this, the letter to the left is deleted, and the rest of the letters on the line move over one space to the left to close the gap left by the deleted letter.

You can open up space to insert letters and numbers by using the SHIFT and INST keys. Space opens to the right of the cursor; the cursor itself does not move. When you insert space in the middle of a line of letters, the rest of the line moves to the right.

The INST/DEL key saves a lot of time when you want to edit or change what you've typed.



## CLR/HOME

When you press this key, the cursor immediately moves to the top left corner of the screen. This is called the HOME position. Your screen stays the same. If you hold down the SHIFT key and press CLR/HOME, not only does the cursor move to HOME, but the screen clears. All that remains on the screen is the blinking cursor at the top left corner of the screen. Press HOME twice to cancel a screen window and return the cursor to the HOME position.

## CTRL




This key doesn't do anything by itself. It always works with another key. The CTRL key works like the SHIFT key: you must hold it down while you press the other key.

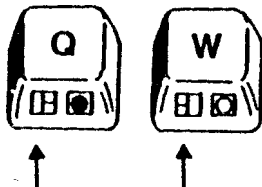
1. As the COLOR KEYS section explains, CTRL and a color key let you choose the color of the text printed on the screen.
2. You can pause a program that is PRINTING on the screen by pressing CTRL and the S key (press any key to resume program output).
3. CTRL is also used with the reverse on/off and flash on/off keys.



In addition, some software programs that you buy use the CTRL key for special functions.



Like the CTRL key, the Commodore key doesn't do much by itself. It has four functions:


1. When used with the SHIFT key, the  key lets you switch between uppercase/graphics mode and upper/lower case text mode.
2. When you're in either mode, the  key acts as a shift to let you type the graphics symbol on the LEFT side of each key. Just hold down  and press the graphic key you want.



3. When you want to change the color of the screen characters to one of the 8 colors listed on the BOTTOM row on the face of the color keys, press  and the color key you want.
4. When you want to slow down a scrolling program display, hold down the  key. The display scrolling speed slows down considerably. When you release the key, the display resumes normal speed.

## THE COLOR KEYS

You can change the colors of the letters and numbers on the screen to any one of the 16 colors available on your C-264. It's simple to do:

- If you want one of the 8 colors listed on the TOP row on the front of the color keys (like BLACK), just hold down the CTRL key and then press the color key with the color you want.
- If you want one of the 8 colors listed on the BOTTOM row on the front of the color keys (like ORaNGe), just hold down the  key and then press the color key with the color you want.

After you change the color, every character typed AFTERWARDS is in the color you last chose.



Your C-264 lets you print the reverse image of any character. In other words, if you are using black letters on a yellow background, you can use the reverse image keys to print yellow letters on a black background.

Here's all you do to get reversed images: press the CTRL key and the RVS ON key. Now everything you type will be displayed in reverse until you press the CTRL and RVS OFF, the RETURN key, or the ESCape key. This will return you to typing normal (non-reversed) characters.



You can make the characters on your screen flash continuously. Just press CTRL and the FLASH ON key to make whatever you type flash. Typing CTRL and FLASH OFF, RETURN, or ESCape lets you type normal (non-flashing) characters again.

## **ESCAPE**

Use the ESCape key to perform many special screen editing functions, such as setting the top and bottom of a display window.

Press the ESCape key and one of the letter keys listed below:

- A Automatic insert mode
- B Set the bottom of the screen window
- C Cancel automatic insert mode
- D Delete current line
- I Insert a line
- J Move to the start of the current line
- K Move to the end of the current line
- L Turn on scrolling
- M Turn off scrolling
- N Return to normal screen display size
- O Cancel insert, quote, and reverse modes
- P Erase everything up to the start of the current line
- Q Erase everything up to the end of the current line
- R Reduce screen display
- T Set the top of the screen window
- V Scroll up
- W Scroll down
- X Cancel the escape function

## **SETTING SCREEN WINDOWS**

Windows let you define an area of the screen as your work area. Everything you type after setting a window will take place within the window (the lines you type, LISTs of your programs, etc.) without affecting the rest of the screen. You can put a window anywhere on the screen.

To set a window, follow these steps:

1. Move the cursor to the screen position you want as the top left corner of the window.
2. Type the ESCape key, then type T.
3. Move the cursor to the screen position you want as the bottom right corner of the window.
4. Type the ESCape key, then type B.


All screen output is confined to the window area you defined. Cancel the window by pressing the HOME key twice. The cursor will go to the HOME position.

## **THE GRAPHICS KEYS**


As we mentioned before, when you turn on the C-264, it is in uppercase/ graphic mode. When you're in this mode, you can type the full set of more than 60 graphics you see on the fronts of many of the keys, as well as all


upper case letters without using the SHIFT key. The SHIFT key lets you type graphics, not uppercase letters.

There are two graphic symbols on each graphics key:

- To print the graphic symbol on the right, hold down the SHIFT key while you press the key.
- To print the graphic symbol on the left, hold down the  key while you press the key.

You can create pictures, charts, and designs by printing graphics side by side or on top of each other, like building blocks. Chapter 6 explains more about graphics.

You can switch between uppercase/graphics mode and upper/lowercase mode by pressing the SHIFT and  keys at the same time. In either mode, type BASIC commands without holding down the SHIFT key.

In upper/lowercase text mode, you can type upper and lowercase letters, just like a regular typewriter. (You will have to shift for uppercase letters.) You also can use all the graphics on the left side of the graphics keys, which you type the same way as in uppercase/graphics mode: hold down  and press the graphics key. The left side graphics are ideal for creating charts, graphs, and business forms.

## SPECIAL SYMBOLS

The C-264 keyboard also contains special symbols not found on many typewriters, or even on most computers. These special symbols include the English Pound sign (£), pi (π), greater and less than signs (< >), brackets ([ ]), and arrows (↑ ↔). These special symbols keys are used in programming your C-264.

## PROGRAMMABLE FUNCTION KEYS

The four keys at the top of your keyboard are special function keys that let you save time by performing repetitive tasks with the stroke of just one key.

You can display what each key does by typing KEY and pressing RETURN.

KEY

KEY 1, "GRAPHIC" or "SYS ####: PROGRAM NAME"

KEY 2, "DLOAD" + CHR\$(34)

KEY 3, "DIRECTORY" + CHR\$(13)

KEY 4, "SCNCLR" + CHR(13)

KEY 5, "DSAVE" + CHR\$(34)

KEY 6, "RUN" + CHR\$(13)

KEY 7, "LIST" + CHR\$(13)

KEY 8, "HELP" + CHR\$(13)

Here's what each key does:

**KEY 1** enters one of the GRAPHIC modes when you supply the number of the graphics area (e.g., GRAPHIC 2, which is split screen, high resolution mode) and a RETURN.

**KEY 2** prints DLOAD " on the screen. All you do to load a program from disk is enter the program name and the closing quotes, and hit RETURN.

**KEY 3** lists a directory of files on the disk in the disk drive.

**KEY 4** clears the screen (even in one of the graphic modes.)

**KEY 5** prints DSAVE " on the screen. All you do to save the current program on disk is enter the program name and the closing quotes, and hit RETURN.

**KEY 6** runs the current program.

**KEY 7** displays a listing of the current program.

**KEY 8** (the HELP key) highlights errors in program statements.

To use one of these functions, just press the function key. You need to use the SHIFT key for keys 4, 5, 6, and 7.

You can redefine any of these keys to perform a function that suits your needs. Redefining is simple: just use the KEY command, which is explained in Chapter 8 and in Appendix B. You can redefine the keys from BASIC programs, or change them at any time in direct mode. (The new definitions are erased when you turn off your C-264.) You can redefine as many keys as you want and as many times as you want.

## THE HELP KEY

When you make an error in a program, the C-264 displays an error message to tell you what you did wrong. These error messages are explained in Appendix A.

You can get more assistance with errors in BASIC programs by using the HELP key. After an error message, press HELP to locate your error exactly. When you press HELP, the line with the error is displayed on the screen with the error printed in reverse. For example:

?SYNTAX ERROR IN LINE 10

C-264 displays this

HELP

You press HELP

10 **PRONT "COMMODORE COMPUTERS"**

C-264 displays this with your error highlighted

## **CHAPTER 3**

# **GETTING TO KNOW YOUR COMMODORE 264**

- Some simple C-264 programs.
- How to correct typing mistakes.
- Introduction to the C-264 text screen.
- More about PRINTing on the screen.
- Introduction to color and reverse printing.

## TRY TYPING THIS PROGRAM:

Type this program exactly as it appears here. Don't leave out the numbers at the beginning of the line. Be sure to press the RETURN key at the end of each line.

This line tells your computer to print C-264 on the TV screen.

This line tells your computer to go back to line 1 and print C-264 again.

This commands your computer to do what the 2 lines tell it to do.

```
1 PRINT "C-264"  
2 GOTO 1  
RUN
```

RETURN  
RETURN  
RETURN

Press the RUN/STOP key to stop the program. Why did your C-264 print its name so many times? GOTO tells your computer to go back to line 1 and PRINT C-264 again and again. This repetition is called a loop.

Now type this:

Here you tell the computer to forget the last program and get ready for a new one.

Same line 1 as last time. PRINT tells your C-264 to display everything between the quotes.

Here you tell the computer to change the color of the screen.

```
NEW  
READY  
1 PRINT "C-264"  
2 COLOR 0,8  
RUN
```

RETURN  
RETURN  
RETURN  
RETURN  
RETURN

You don't type this; your C-264 does, to tell you it's READY for a new program.

This time there's no GOTO loop in the program, so the orders are carried out just once.

Here's another short program that gives you a hint of how easy it is to draw graphics on your C-264:

don't forget this line:

```
NEW  
10 COLOR 0,12  
20 GRAPHIC 2,1  
30 CIRCLE,160,100,65,10  
RUN
```

RETURN  
RETURN  
RETURN  
RETURN  
RETURN

Now your screen should be pink and an ellipse should have been drawn on the screen. Try this:

1. Type LIST. The program is listed at the bottom of the screen.
2. Use the cursor key to move to the number 10 at the end of line 30.
3. Change 10 to 50,
4. Press RETURN,
5. Move the cursor to a blank line.

Now type RUN to draw a circle on the screen.

Finally, type this line:

```
40 GRAPHIC 0,1  
RUN
```

RETURN

RETURN

The circle is drawn again, but this time it disappears quickly and the READY message is back at the top of the screen.

Here's another little program that draws on your screen:

```
NEW  
10 GRAPHIC 2,1  
20 BOX 1,100,100,50,50  
30 BOX 1,100,100,150,150  
RUN
```

RETURN

RETURN

RETURN

RETURN

RETURN

Now add this line:

```
40 BOX 1,150,150,50,50  
RUN
```

RETURN

RETURN

Now add this line:

```
40 GRAPHIC 0,1  
RUN
```

RETURN

RETURN



## HOW TO CORRECT TYPING MISTAKES

If you make a mistake when you're typing something, there are several ways to make changes.

1. **YOU CAN RETYPE A LINE** anytime, even after you've RUN the program. The C-264 automatically replaces the old line with the new one. The C-264 doesn't put the new line in the same place as the old one, and the old one still appears on the screen, but the C-264 ignores it. When you have two statements with the same line number, the C-264 only acknowledges the last one entered. For example, if your program looks like this:

```
10 COKOR 0,3  
20 PRINT "C-264"
```

mistake

Press the RETURN key to get to a fresh line, and just retype line 10 correctly:

```
10 COLOR 0,3
```

RETURN

Now the first line 10 is replaced by the second line 10. You can check this by typing LIST, which displays a fresh copy of your program. When you LIST a program, all lines appear in correct order and the replaced lines don't appear:

```
LIST
```

```
10 COLOR 0,3  
20 PRINT "C-264"
```

RETURN

RETURN

RETURN

Replacing lines in a program is also a good way to experiment with your computer. When you replace a line, the new one doesn't have to be anything like the old line. For example, instead of correcting the spelling of COLOR, you can type this:

```
10 PRINT "I LOVE MY"
```

RETURN

Now RUN the program and see what happens.

2. **YOU CAN ERASE AN UNWANTED LINE** just by typing the number of the line and pressing RETURN. The computer ignores the line even though it might still appear on the screen. Type LIST to make sure the line is gone from the program.

```
10 PRINT "I LOVE MY"  
20 PRINT "C-264"  
10  
LIST  
20 PRINT "C-264"
```

RETURN

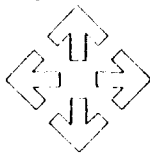
RETURN

RETURN

RETURN

RETURN

3. **YOU CAN EDIT A LINE.** Use the cursor keys to move to the place in the line that you want to change. Now just type over what you want to change. Press RETURN when you finish.



**NOTE:** You don't have to be at the end of the line to press RETURN. Your C-264 will remember the whole line even if you press RETURN in the middle of the line.

10 PRINT "SARAH IS MY NAME" **RETURN**

If you want to change the name to GLENN, move the cursor to the S in SARAH.

10 PRINT "**S**ARAH IS MY NAME"

cursor

And now just type GLENN over SARAH and press **RETURN**.

10 PRINT "GLENN IS MY NAME" **RETURN**

**NOTE:** When you type a quotation mark after a PRINT statement, you enter QUOTE MODE. In quote mode, some keys work differently. For example, if you press a cursor-down arrow while you're in quote mode, the cursor won't move and you'll see a reverse Q printed on the screen. When the PRINT statement executes, the reverse Q isn't PRINTed; instead the cursor moves down. In quote mode, the computer assumes that everything you type is something you want to display or do later when the PRINT statement executes.

4. YOU CAN OPEN UP SPACES IN A WORD OR LINE with the INST key (get this insert key by holding down SHIFT while you press INST/DEL). Hold the keys down until you open up as many spaces as you need. (Notice that the cursor stays in the same place while spaces open up to the right.) Then just type what you want to insert.

10 PRINT "264"

RETURN

To change this to I LOVE MY 264, move the cursor to the first 4 and press the SHIFT and INST keys until enough space opens up. Don't bother to count out the spaces. You can just guess and then open up more if there aren't enough.

10 PRINT "■ ←

264"

cursor

Now add the other words:

10 PRINT "I LOVE MY 264"

RETURN

5. YOU CAN ERASE CHARACTERS AND CLOSE UP SPACE with the DEL key (get this delete key by pressing INST/DEL). This key erases characters or spaces immediately to the LEFT of the cursor.

10 PRINT "ANDREW IS MY NAME"

RETURN

You can change this to MIKE IS MY NAME by moving the cursor to the D in ANDREW, pressing the INST/DEL key twice, and typing MIKE.

10 PRINT "ANDREW IS MY NAME"

cursor

and press INST/DEL twice.

10 PRINT "DREW IS MY NAME"

Type in MIKE and press RETURN.

## A SIMPLE C-264 PROGRAM

Now that you've experimented a little with your C-264, here's a simple program to try:

- STEP 1:** Clear the screen by holding down the SHIFT key while you press the CLR/HOME key. This erases your screen.
- STEP 2:** Clear out old programs by typing NEW. Press the RETURN key.
- STEP 3:** Type this program exactly as it appears. Remember to type the line numbers and all punctuation marks. Use the tips for correcting mistakes (turn to the preceding pages to find them) if you type something incorrectly. Don't forget to press RETURN after each line.

**NOTE:** Remember, you can stop a program by pressing the RUN/STOP key.


```
NEW
1 COLOR 0,8
2 PRINT "I LOVE MY C-264 ";
3 COLOR 0,3
4 PRINT "MY NAME IS your name ";
5 COLOR 0,7
6 PRINT "♥♥♥♥♥♥♥♥";
7 GOTO 5
RUN
```

Be sure to leave a space here.

Be sure to leave a space here.

Get the heart by holding down SHIFT while you press the S key 6 times.

After you STOP the program (use the RUN/STOP key), type LIST. When the program is displayed on your screen, look at the tips for correcting errors and change this program so that a friend's name appears in the program.

**TIP:** Want to slow down this program without stopping it? Just hold down the  key.

## INTRODUCTION TO THE C-264 TEXT SCREEN

Try typing this program:

press SHIFT and S. → NEW  
1 PRINT "♥";  
2 GOTO 1  
RUN

RETURN  
RETURN  
RETURN  
RETURN

Now your screen fills with hearts. When the entire screen is covered with hearts, press the RUN/STOP key to end the program. This program shows you how big your C-264's screen is.

Now type this program:

press SHIFT and CLR/HOME. → NEW  
1 PRINT "□"  
2 FOR X = 1 TO 40  
3 PRINT "♥";  
4 NEXT  
RUN

RETURN  
RETURN  
RETURN  
RETURN  
RETURN

When you RUN this program, the first row on your screen fills completely with hearts. There are 40 hearts altogether. Since the row is full, you can see that there are 40 positions across the screen. Each position across the screen is called a COLUMN.

Now type this program:

press SHIFT and CLR/HOME. → NEW  
1 PRINT "♥"  
2 FOR X = 1 TO 25  
3 PRINT "♦"  
4 NEXT  
RUN

RETURN  
RETURN  
RETURN  
RETURN  
RETURN  
RETURN

When you RUN this program, the first column on your screen fills with diamonds. There are 25 diamonds printed, but the first three disappear at the top of the screen because the word READY surrounded by two blank lines always appears at the end of the program. There are, then, 25 rows. So your C-264 has 40 columns and 25 rows.

The C-264 has 1000 different positions on the screen for letters, numbers, graphic symbols, etc.

**NOTE:** Sometimes you'll type a particularly long line on your C-264, such as this:

**1 PRINT " ANTIDISESTABLISHMENTARIANISM OPPOSES ANARCHY"**

You'll notice that as you type this, you run out of room on one row. But keep typing; the C-264 automatically moves on to the next row and continues printing there until your line is finished.

Now try **RUN**ning this one line program. The message is printed on two rows. If your line is longer than one row, the C-264 lets it spill over to the next row. The C-264 considers the line ended when you press the **RETURN** key, not when you type to the end of the row. You'll get used to this as you use your C-264.

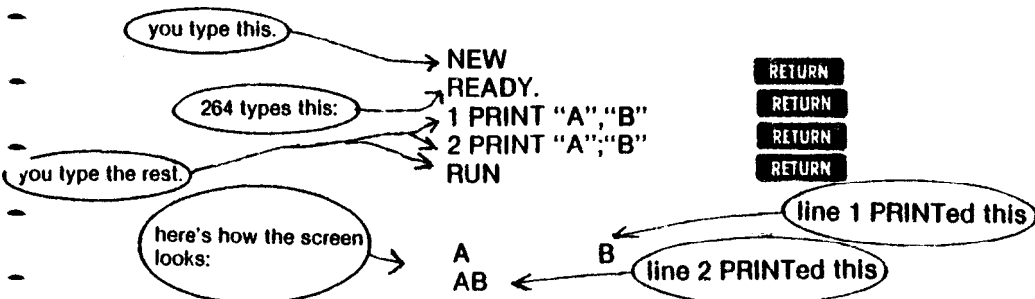
Now type this program:



When you **RUN** this program, you can see that it's possible to tell the C-264 exactly where to **PRINT** something on the screen.

## MORE ABOUT PRINTING ON THE SCREEN

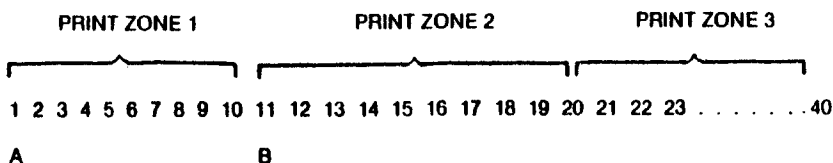
Try typing this program:



If line 1 and line 2 seems so much alike, why is there such a difference in what they PRINT on the screen? The difference is in the punctuation between the items this program PRINTs.

When you use a comma to separate items in a PRINT statement, the items are PRINTed several spaces apart. When you use a semicolon, the items are PRINTed right next to each other.

As you recall, the C-264's screen has 40 columns across. These columns are divided into four 10 space areas, called PRINT ZONES. When you use a comma to separate PRINTed items, the C-264 PRINTs the first item in the first print zone, the second item in the second print zone, etc. The commas work like tabs on a typewriter.



If you ask the C-264 to PRINT more than 4 items separated by commas, the C-264 automatically goes to the next line to PRINT. For example:

PRINT "A","B","C","D","E","F"

PRINTs this:

A      B      C      D  
 E      F

When you use semicolons to separate items in a PRINT statement, the C-264 ignores the print zones and PRINTs all the items one after another.

PRINT "A";"B";"C";"D";"E";"F"

PRINTs this:

ABCDEF

Here's what happens if the first PRINT item is 12 letters long and the second item is separated by a comma:

```
PRINT "ABCDEFGHijkl","M"
```

PRINTs this:

```
ABCDEFGHijkl      M
```

print  
zone 1

print  
zone 2

print  
zone 3

Now clear your screen and type this program:

```
NEW
```

```
1 PRINT 1,2
```

```
2 PRINT 1;2
```

```
RUN
```

```
1           2  
1  2
```

This program shows you two new things:

1. Numbers don't have to be put in quotes in a PRINT statement. (You'll learn more about this in Chapter 4).
2. Numbers are displayed with a space on both sides of them, so when you use a semicolon, the numbers aren't PRINTed right next to each other the way letters are.

Numbers are PRINTed with a space in front because the C-264 leaves room for a number's sign. If the number is negative, a minus sign appears in the space before the number. If the number is positive, you see a blank space because the C-264 doesn't ordinarily display the plus sign.

Now type this program:

```
NEW
```

```
1 PRINT -1,-2
```

```
2 PRINT 1,2
```

```
RUN
```

```
- 1           - 2  
 1           2
```

Notice that the numbers are in the same place whether they are positive or negative.

These examples give you an idea of how you can put messages on the C-264's screen. As you read this manual and practice using your C-264, you'll learn more about displaying text and graphics.



## INTRODUCTION TO COLOR AND REVERSE PRINTING

The C-264 can display numbers, letters and graphic symbols in 16 different colors. You can also display all these characters in reverse.

STEP 1. Clear your screen by pressing SHIFT and CLR/HOME.

STEP 2. Hold down the CTRL key and press the RVS ON key:




STEP 3. Release the keys and hold down the space bar (the long bar at the bottom of the keyboard).

STEP 4. Hold down the space bar as long as you want. While you hold down the space bar, a line the same color as the letters on your screen should get longer. If the line gets to the end of the row, it continues on the next row.

STEP 5. Release the space bar (but don't press the RETURN key).

STEP 6. Hold down the CTRL key and press one of the color keys (not a color that's already on your screen). As soon as you do this, the cursor will be the color of the key you pressed.

STEP 7. Hold the space bar down again. Now your C-264 will draw a line in the new color. Continue changing colors with the CTRL or  keys and the color keys. Then hold down the space bar to make different colored lines.

STEP 8. Turn off reverse print by holding down CTRL and pressing the RVS OFF key. If you press the RETURN key, reverse will also be turned off.



Try typing some letter in reverse. Just hold down CTRL and RVS ON to turn on reverse, and then type whatever you want. Reverse letters make excellent headlines. You can also use them to highlight special words and numbers. Try this:

```
NEW
10 PRINT "R COMMODORE";
20 GOTO 10
RUN
```

press CTRL and ReVerSe ON

Now press RUN/STOP and replace line 20 with this line to make letters flash on your screen:

```
20 PRINT" 264"
RUN
```

press CTRL and FLASH ON

## **CHAPTER 4**

# **USING SOFTWARE**

- Introduction.
- How to use built-in software.
- How to load cartridges.
- How to load cassette tapes.
- How to save programs on cassette tapes.
- How to load programs from diskette.
- How to header a diskette.
- How to save programs on diskette.
- How to find out what programs are on a diskette.
- Getting software through telecommunications.

## **INTRODUCTION**

The family of software available for your C-264 is growing quickly. Your dealer can keep you up-to-date on new products and inform you about the features of software that's currently available.

Your Commodore 264 can use software that is built-in or that is recorded on CARTRIDGES, CASSETTE TAPES, and DISKETTES that are available from your Commodore dealer. All you do is load them into your C-264. And you can create and store your own programs on cassette tapes or floppy disks.

## **HOW TO USE BUILT-IN SOFTWARE**

Your C-264 can be equipped with a wide variety of function key software packages. These are programs built into the C-264, which you call by pressing the appropriate FUNCTION key. The programs range from Logo to The Magic Desk. The built-in package is always available for use. When you turned on your C-264, both BASIC and the function key software packages announce themselves through their power-on messages. Also, you can use the KEY command to see the function key definitions. If there is function key software built into your C-264, the definition for KEY 1 will be: SYSXXXX:package name. Just press function key f1 and hit RETURN to enter the built-in program.

Function key software packages are also available on cartridge. If you have plugged in a function key cartridge into your C-264, start the program by pressing function key f2.

## **HOW TO LOAD CARTRIDGES**

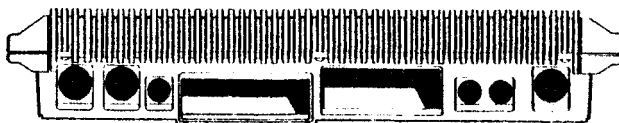
You can use a variety of business and personal programs, as well as exciting, arcade-style games available on cartridge for your C-264.

Follow these steps to load cartridges:

**STEP 1**    **TURN OFF** your C-264.

**IMPORTANT: YOU MUST TURN OFF YOUR COMPUTER BEFORE YOU INSERT OR REMOVE CARTRIDGES. IF YOU DON'T, YOU MAY DAMAGE THE CARTRIDGE AND THE COMPUTER.**

**STEP 2.**    Holding the cartridge with the nameplate facing UP, insert the cartridge firmly in the cartridge slot on the back of your computer.



Plug cartridge  
in here.

**STEP 3.**    Turn on your C-264.

**STEP 4.**    Begin the game or program according to the instructions that come with the software. Usually all you do is press one key to start.

## HOW TO LOAD CASSETTE TAPES

A variety of software products for the C-264 is available on cassette tapes. These cassette tapes look just like the ones with recorded music that you can play on a stereo. Cassette tapes run in the Datassette tape recorder, available from your Commodore dealer.

You can also use cassette tapes and the Datassette to store programs you write yourself. The next section explains how to save programs on tape.

The steps for loading tape are the same whether you are using purchased software or programs you saved yourself.

**STEP 1.** Insert the cassette into your cassette recorder and close the door.

**STEP 2.** Type **LOAD** and press the **RETURN** key. The computer displays this message:

**PRESS PLAY ON TAPE.**

**STEP 3.** Rewind the tape to the beginning. The screen will go blank. Just press the **REWIND** button on the cassette recorder.

**STEP 4.** When the tape is rewound to the beginning, press the **PLAY** button on the datassette.

When the program is found, the screen displays this message:

**FOUND program name**

**STEP 5.** Press the **Commodore** key to load this program. If there is more than one program on the tape, and the program the C-264 found isn't the one you want, press the space bar to keep searching.

When the program is loaded, the word **READY** appears. If you want to stop the loading before it's complete, press the **RUN/STOP** key. After the software is loaded, type **RUN** to start the program. You can also **LIST** it or change it, assuming it is a **BASIC** program.

To **LOAD** a specific program on the tape, use the **LOAD "program name"** form of the **LOAD** command.

## **HOW TO SAVE PROGRAMS ON CASSETTE TAPES**

When you write a program and want to save it on cassette tape, follow these steps:

**STEP 1. Type:**  
**SAVE "program name"**

The program name you use can be anything you want, but can be no more than 16 letters and/or numbers long.

**STEP 2. Press the RETURN key. The computer displays this message:**  
**PRESS RECORD AND PLAY ON TAPE**

**STEP 3. Press the RECORD and PLAY buttons on your cassette recorder.**  
The screen goes blank. When your program is saved, the word  
**READY** appears on the screen.

**Examples of SAVE Commands for Cassette Tape:**

**SAVE "MYJOB"**  
**SAVE "3TEST"**

## HOW TO LOAD PROGRAMS FROM DISKETTE

Disks are fast and easy to use. Be sure to handle your disks and your disk drive carefully.

The steps are the same for loading all disks:

- STEP 1.** Make sure that your disk drive is ON.
- STEP 2.** Insert the disk into the disk drive. The label side of the disk must face up. Put the disk in so that the labelled end goes in last while the end with the exposed oval goes in first. Look for a little notch on the disk (it might be covered with a little sticker). This notch is on the left side as you put in the disk, assuming that you're facing your disk drive. Be sure the disk is in all the way.
- STEP 3.** Close the protective door on the disk drive after you insert the disk.
- STEP 4.** Type:  
DLOAD "program name"
- STEP 5.** Press the RETURN key. The disk spins and your screen says:  
SEARCHING FOR PROGRAM NAME  
LOADING  
READY.  
■
- STEP 6.** Your software is ready to use.  
Type RUN and RETURN to start the program.

If the red light on the disk drive blinks after the DLOAD is finished, something went wrong. You can find out what happened by typing:

?DS\$

**RETURN**

Examples of DLOAD commands:

DLOAD ""	LOADs the 1st program on the disk.
DLOAD "MYFILE"	LOADs a disk program called MYFILE.
DLOAD "SET"	LOADs the first program on the disk that begins with the letters SET.



## HOW TO HEADER A DISKETTE

HEADERING prepares your new BLANK disk for use.

**IMPORTANT: DO NOT HEADER A DISK THAT HAS ANYTHING ON IT UNLESS YOU WANT TO ERASE THE ENTIRE DISK. HEADERING ERASES EVERYTHING ALREADY ON A DISK.**

To HEADER a disk, enter this command exactly as it appears. Type your own disk name and disk id number, which are explained below.

HEADER "disk name", Ddrive# ,id RETURN

- The name you use is the name of the entire disk. Give the disk any name up to 16 characters.
- If you have a dual drive, add D0 or D1 to identify the drive number.
- The id is the letter I and any two letters and/or numbers, like I21, IR5, ISM, etc. Give the disk any id you want, but you should give every disk a different id.

## ARE YOU SURE?

As soon as you press RETURN, the C-264 asks ARE YOU SURE? This is to give you a last chance to change your mind.

To header a disk, type YES or Y and press RETURN. If you decide not to header the disk, type NO or N and press RETURN.

Here are some examples of the HEADERing command:

HEADER "LETTERS",I07

HEADER "FINANCES",D0,IS3

Now that you know how to HEADER a disk, you're ready to use disks to write and save programs on your C-264. Appendix B has more information about the HEADER command.

## HOW TO SAVE PROGRAMS ON DISKETTE

When you want to reuse a program you've written, be sure to **SAVE** it before you **LOAD** another program. If you don't you'll lose the program.

When you change a previously **SAVED** program, you have to **SAVE** it again if you want to keep the new version.

When you **reSAVE** a program, you are replacing the old version with the new one. If you want to keep both the old and the changed versions, you have to give the new one a different name when you **SAVE** it.

Follow these simple steps to save a program on disk:

**STEP 1.** Type **DSAVE "program name"**

**STEP 2.** Press **RETURN**. The disk drive makes a noise, and the computer displays this message when the program is saved:

SAVING "program name"  
OK  
READY.



Example: **DSAVE "MYPROG5"**

If the red light on the disk drive blinks after the **DSAVE** is finished, something went wrong. To find out what happened, type:

?DS\$

**RETURN**

## HOW TO FIND OUT WHAT PROGRAMS ARE ON A DISKETTE

When you **SAVE** programs on disk, the computer keeps a table of contents for that disk. You can display the table of contents to see what's on a disk. Just type this command:

**DIRECTORY**

**RETURN**

As soon as you press **RETURN**, your C-264 displays the name of each program on your disk.

You can also display just part of the table of contents:

**DIRECTORY "MY\*"**

**RETURN**

Lists every file on the disk that starts with the letters **MY**.

## GETTING SOFTWARE THROUGH TELECOMMUNICATIONS

If you have a modem, you can get access to huge amounts of software and other information from computer information services like CompuServe and The Source. Commodore supports its own service called the Commodore Information Network, which is available through CompuServe. Many programs are available on the Commodore user databases.

## CHAPTER **5**

### **BEGINNING BASIC**

- Doing calculations on your C-264.
- Immediate mode.
- Using variables.
- Inputting information.
- Using loops: FOR . . . NEXT

## DOING CALCULATIONS ON YOUR C-264

You can use PRINT to do more than just display what you put in quotation marks. You can use your C-264 like a simple calculator. Besides the standard + and - signs, your C-264 uses the \* sign for multiplication and the / sign for division and fractions. (Computers use the \* sign instead of an X for multiplication because a computer can't tell the difference between the letter X and the mathematical symbol X.)

### Basic Mathematical Operators

Addition	+
Subtraction	-
Division and fractions	/
Multiplication	*
Exponentiation	↑

### Basic Relational Operators

Greater than	>
Less than	<
Equals	=
Greater than or equal	>=
Less than or equal	<=

To calculate a problem, type PRINT and then the math problem. Don't put the problem in quotes.

Type this program:

NEW

1 PRINT 1 + 2, 2 - 1

2 PRINT 2 \* 2, 4 / 2

RUN

3 1

4 2

use the slash on the ? key

For the first time, PRINT didn't print exactly what you typed in the statement. Instead, the C-264 solved the calculations and PRINTed the answers. All you have to do to use PRINT to calculate is omit the quotation marks. Now try this:

NEW

1 PRINT "2 \* 3 + 1 = "

2 PRINT (2 + 7) / 3

RUN

2 \* 3 + 1 =

↑ 3

this space is left for the answer's sign

Since the calculation in line 1 is in quotes, the C-264 just PRINTs the problem as if it were any text: exactly as it appears between the quotation marks. The problem isn't solved, and no space is left for the numbers' sign.

Now move the cursor back to line 1 and change the line so it looks like this:

1 PRINT "2\*3 + 1 = ";2\*3 + 1  
RUN  
2\*3 + 1 = 7  
3

Annotations:

- don't forget the semicolon (points to the semicolon in line 1)
- this space is left for the answer's sign (points to the space before the semicolon in line 1)
- the answer for line 2 stays the same (points to the number 7 in line 2)

If you want to both PRINT the problem AND solve it you have to type it twice: once in quotes and once out of quotes.

**NOTE:** The C-264 doesn't accept commas as part of a number. For example, always type 109401 instead of 109,401. If you put a comma in a number, the C-264 thinks you mean two numbers (separated by the comma), so the C-264 would read 109 and 401 instead of 109401.

## FRACTIONS AND DECIMALS

You can write a fraction like this: 1/2  
or like this: .5

If you put a fraction in a PRINT statement, your answer will always be returned as a decimal or whole number. For example:

PRINT 139/493 + 5      **RETURN**  
5.28194726

Here's an example that uses pi (3.1415926535...), which represents the ratio of the circumference of a circle to its diameter. Just type the pi key.

**π**

PRINT π/374      **RETURN**  
8.39998036E-03

## SCIENTIFIC NOTATION

What did the C-264 mean by the E-03 part of the above answer ? The C-264 displays decimal numbers in the range - 999,999,999 to 999,999,999 in standard numerals, but numbers beyond this range are automatically displayed in scientific notation. This special notation lets the C-264 display large numbers in fewer digits.

Scientific notation takes this form:

number[.number]E<+ | - >number

only ONE digit is shown to the left of the decimal point

the third number is the number of places the decimal point is moved

For example:

$$20 = 2E + 1$$

the decimal point is moved 1 digit left

$$105000 = 1.05E + 5$$

the decimal point is moved 5 digits left

$$.0666 = 6.66E - 2$$

the decimal point is moved 2 digits right

## IMMEDIATE MODE

You can put any calculation in a program, or get an immediate answer by typing PRINT and the problem without a line number, like this:

PRINT 3-6

-3

PRINT 24/(6 + 2)

3

When you don't have a line number before a BASIC statement, you don't have to type RUN to tell the computer to follow the instruction. When you press RETURN, you get instant results. Not having a line number means the command should be acted on immediately; it's in IMMEDIATE, or DIRECT MODE. Having a line number means the statement is part of a BASIC program; it's in PROGRAM MODE.

You've used several other immediate mode commands: RUN, NEW, and LIST. You can use PRINT in immediate mode to instantly solve calculations or display messages.

PRINT "2 TO THE 3RD POWER EQUALS";213

2 TO THE 3RD POWER EQUALS 8

this arrow stands for exponentiation; get it by typing SHIFT and 0 the message is PRINTed and then the problem's solution is PRINTed

## TIPS ON ERRORS

If you make a mistake in your PRINT statement, use the INST/DEL keys to make corrections. You can change the line as much as you like before you press the RETURN key.

If you make a mistake that you don't catch before you press RETURN, your C-264 displays an error message to help you figure out what you did wrong. For example:

?SYNTAX ERROR

If you get this message, check over what you typed to see where you made a mistake. The computer is very picky, and it can't follow instructions that contain spelling errors or other mistakes. If you can't figure out your mistake, refer to Appendix A for more information about error messages. Once you find your mistake, you can use the cursor control keys to move the cursor over the error, and then use the INST/DEL keys to correct it.

### PROGRAMMING TIP

You can abbreviate PRINT (and many other BASIC commands.) The abbreviation for PRINT is the question mark. For example:

? 1/4\*3 is the same as PRINT 1/4\*3

Other BASIC abbreviations are listed in Appendix C.

### MORE ABOUT MATHEMATICAL CALCULATIONS

The last example shows that you can perform more than one calculation in one line. Try typing this:

PRINT 200 + 50/5

RETURN

Is the answer what you expected? Try this:

PRINT (200 + 50)/5

RETURN

The C-264 always solves problems in a certain order. Problems are solved from left to right, but within that general rule, some types of calculations are solved first. The order which the C-264 evaluates expressions is called the ORDER OF PRECEDENCE.

FIRST: C-264 checks for negative numbers (not subtraction, just negative numbers).

SECOND: C-264 solves any exponents.

THIRD: C-264 solves all multiplications and divisions, from left to right.

FOURTH: C-264 solves additions and subtractions, from left to right.

BUT: C-264 always solves any portion of the problem surrounded by parentheses first. You can even put parentheses within parentheses:  $36 * (12 + (A / 3))$ . The innermost parentheses are solved first.

Sometimes it's a good idea to put negative numbers in parentheses for clarity. For example, if you want to add 45 and  $-5$ , type it like this:  $45 + (-5)$ . The C-264 will understand it either way.



## USING VARIABLES

The example  $36 * (12 + (A/3))$  shows one of the most powerful features of a computer. When we used a letter instead of a number in a mathematical problem, we used a VARIABLE. A variable stands for a value:

```
10 A = 3
20 PRINT "TOTAL: "; A * 4
RUN
TOTAL: 12
```

There are three types of variables you can use:

Type	Symbol	Description	Examples	Sample Values
Integer	%	whole numbers	X%, A1%	15, 102, 3
Text string	\$	letters, numbers, and all other characters in quotes	X\$, MS\$	"TOTAL:", "\$65", "DAY 1", "CBM"
Floating point		real (decimal) or whole numbers	X, AB, T4	23.5, 12, 1.3E + 2

Always use the right variable type. If you try to do something like assign a word to an integer variable, your program won't work.

Besides using PRINT to add, subtract, multiply and divide, you can figure exponents and do advanced mathematical functions. Appendix B shows you how to use BASIC functions to quickly and simply figure square roots, cosines, and lots more.

### TRY TYPING THIS PROGRAM

```
NEW
10 INPUT "WHAT'S THE DATE";D$
20 INPUT "WHAT'S YOUR NAME";N$
30 SCNCLR
40 PRINT "  ON ";D$; ", ";N$; " LEARNED MORE ABOUT USING
THE C-264"
```

RUN

press the cursor-down arrow 5 times

When you RUN this program, type in answers to the questions WHAT'S THE DATE? and WHAT'S YOUR NAME?.

## INPUTTING INFORMATION

The last example showed you a new BASIC statement, and a new way to enter information. INPUT lets you enter words and numbers WHILE the program is running. The greatest advantage to this is the reusability INPUTting brings to programs.

For example, if the last program had the line PRINT "OCTOBER 31", you'd have to change the line before you executed the program whenever the date isn't October 31. With the INPUT statement, you don't change the program. Instead, every time you reuse it, you supply fresh information.

Try this program.

```
NEW
10 INPUT "KEEP GOING OR STOP";X$
20 PRINT "YOU TOLD THE C-264 TO ";X$
30 IF X$ = "STOP" THEN END
40 GOTO 10
RUN
KEEP GOING OR STOP? GO
YOU TOLD THE C-264 TO GO
KEEP GOING OR STOP? CONTINUE
YOU TOLD THE C-264 TO CONTINUE
KEEP GOING OR STOP? STOP
YOU TOLD THE C-264 TO STOP
```

You type in these words.  
The Commodore 264 prints  
the rest.

Here's what happens in this program:

Line 10 tells the computer to display a question, called a PROMPT, to tell you to INPUT a value for X\$. The program waits until you supply the value before continuing with the program.

Line 20 PRINTs a message and the value you INPUT for X\$.

Line 30 tells the computer to end the program immediately IF the value you INPUT for X\$ is STOP. When this happens, line 40 is not executed. If the value you INPUT is not STOP, the program doesn't end.

Line 40 tells the computer to go back to line 10 so you can INPUT another value for X\$.

You don't have to include a message in an INPUT statement. When you don't, the computer displays a question mark to tell you it's waiting for data INPUT. For example:

NEW

10 FOR C = 1 TO 3

20 INPUT X

30 PRINT "TOTAL: "; X \* 1.06

40 NEXT C

RUN

? 5.5

TOTAL: 5.83

? 1.00

TOTAL: 1.06

? 6.97

TOTAL: 7.3882

## USING LOOPS: FOR . . . NEXT

So far we've shown you several ways to make a program repeat some or all of its lines. GOTO and IF . . . THEN statements are two ways to control and repeat program execution.

FOR . . . NEXT statements let you create program LOOPS that control the number of times a segment of a program is executed. The FOR statement sets a limit on the number of times the loop will execute by assigning a range of values to a variable. FOR works like this:

FOR counter = first time TO last time  
do something until the last value for the counter is reached  
add the NEXT value to the counter

The NEXT statement marks the end of a FOR . . . NEXT loop. When the program reaches the NEXT statement, the computer sends control back to the FOR statement. The computer then increases the loop counter value by 1 (or whatever the value named in a STEP statement) and checks to see if the end limit of the loop has been reached. If the end limit has been exceeded, control is transferred to the statement following NEXT. If the end limit has not been reached, the loop continues.

NEW

```
10 FOR CT = 1 TO 4
```

```
20 PRINT "COMMODORE COMPUTERS  "; "COUNTER = "; CT
```

```
30 NEXT CT
```

```
40 PRINT "FINAL COUNTER VALUE = "; CT
```

RUN

```
COMMODORE COMPUTERS  COUNTER = 1
```

```
COMMODORE COMPUTERS  COUNTER = 2
```

```
COMMODORE COMPUTERS  COUNTER = 3
```

```
COMMODORE COMPUTERS  COUNTER = 4
```

```
FINAL COUNTER VALUE = 5
```

Ordinarily, every time a FOR . . . NEXT loop executes, the value of the counter increases by one. You can vary the amount of the increase by adding a STEP statement to the FOR statement:

FOR CT = 1 TO 4 STEP 2      this STEP adds 2 to the counter each time  
the loop executes, so this loop executes  
twice, not four times

FOR CT = 1 TO 4 STEP .5      this STEP adds .5 to the counter, so this  
loop executes seven times

FOR CT = 4 TO 1 STEP -1      a counter can run from a higher to a lower  
number IF you use a negative STEP

## CHAPTER 6

# USING GRAPHICS AND COLOR

- Keyboard colors
- Graphics characters
- Character animation
- Controlling colors
- High resolution graphics
- Points, lines, and labels
- Squares, circles, polygons, and painting
- Multi-color graphics

## KEYBOARD COLORS

You can change the color of the characters on the screen to improve readability or to find a color combination you like. Here's how to see how the different color characters look on your screen:

**STEP 1.** Hold down the CTRL key.


**STEP 2.** Press the 2 key while you're holding the CTRL key down. The cursor turns white.

**STEP 3.** Let go of CTRL and type some letters. Everything you type appears in white now.

HOLD CTRL WITH COLOR KEY	COLOR RESULT
1	BLACK
2	WHITE
3	RED
4	CYAN
5	PURPLE
6	GREEN
7	BLUE
8	YELLOW

Table 1: Colors using CTRL key

Using the CTRL key with the keys between 1 and 8 lets you choose the colors shown on the top of each color key.

Now hold down the  key. By typing on the keys between 1 and 8, the cursor changes one of the 8 colors printed on the bottom of each color key. All 16 colors can appear on the screen at the same time.




HOLD  WITH COLOR KEY	COLOR RESULT
1	ORANGE
2	BROWN
3	YELLOW-GREEN
4	PINK
5	BLUE-GREEN
6	LIGHT BLUE
7	DARK BLUE
8	LIGHT GREEN

Table 2: Colors using  key

## GRAPHIC CHARACTERS

Each letter key contains 2 different graphic characters, as do the @, -, \*, and £ keys. To print graphic characters, you must hold down the SHIFT or  key while you press the key for the graphic symbol you want.

When your C-264 is in uppercase/graphic mode, hold down SHIFT and press a letter key to display the graphic character on the right side of that letter key. These characters include the playing card suits, a solid and a hollow ball, and a set of lines and connecting characters that let you draw many different pictures on your screen.

Here are some examples to help you get used to these characters:


### Exercise 1: Large Circle

Step 1: Press down the SHIFT LOCK key. It should stay down.

Step 2: Hit the letter U then the letter I.

Step 3: Hit the  key.


Step 4: Hit the letter J then the letter K.

Step 5: Hit the  key.


### Exercise 2: Snake

Step 1: Press down the SHIFT LOCK key. It should stay down.

Step 2: Hit U, then I, then U, then I, then U, then I.

Step 3: Hit the  key.


Step 4: Hit K, then J, then K, then J, then K, then J.

Step 5: Hit the  key.

### Exercise 3: Crooked Line

Step 1: Press down the SHIFT LOCK key. It should stay down.

Step 2: Hit E, then D, then C, then \*, then F, then R.

Step 3: Hit the  key.

#### Exercise 4: Two Crosses

Step 1: Press down the SHIFT LOCK key. It should stay down.

Step 2: Hit M, then SPACE, then N, then SPACE, then —.

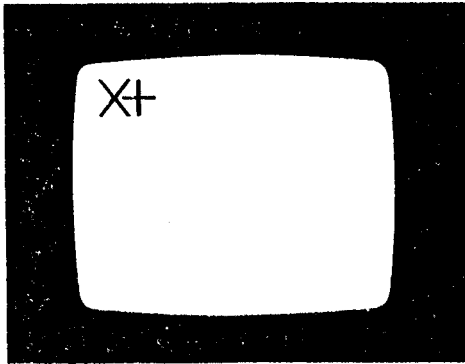
Step 3: Hit the **RETURN** key.

Step 4: Hit SPACE, then V, then SPACE, then \*, then +, then \*.

Step 5: Hit the **RETURN** key.

Step 6: Hit N, then SPACE, then M, then SPACE, then —.

Step 7: Hit the **RETURN** key.



When you are finished, press SHIFT LOCK again so it pops up.

Did you wonder why the computer doesn't say SYNTAX ERROR when you hit RETURN? After all, you had characters on the line that weren't commands that the computer can understand.

The reason is that the C-264 doesn't pay attention to the line you typed if you hold down SHIFT when you press RETURN. If you press RETURN without the SHIFT key, the computer tries to figure out what you mean when you're just drawing pictures.

So far we haven't talked about the graphic characters on the left side of the keys. These graphics work just like the right side characters, except that you hold down the **C** key instead of SHIFT. There is no **C** lock, so you must hold it down yourself.

You can print this set of graphics in either uppercase/graphic mode or upper/lowercase mode.

The left side graphic characters include lines and angles used for drawing charts and tables. For example, here's how to underline a word:

Step 1: Move the cursor just under the word you want to underline.

Step 2: Hold down the **C** key and the T key, which prints an underline graphic. Hold these two keys down until the word is underlined.



### Exercise 5: Half Bar

Step 1: Hold down the **C** key with one hand during the whole exercise.

Step 2: Hit D, then I, then I, then F.

Step 3: Hit the **RETURN** key.

### Exercise 6: Wedge

Step 1: Hold down the **C** key and hit T, then Y, then U.

Step 2: Hold down the **CTRL** key and hit 9.

Step 3: Hold down the **C** key and hit I, then O, then P, then @, then SPACE.

Step 4: Hit the **RETURN** key.

### Exercise 7: Window

Step 1: Hold down the **C** key with one hand during the whole exercise.

Step 2: Hit A, then R, then S, then **RETURN**.

Step 3: Hit Q.

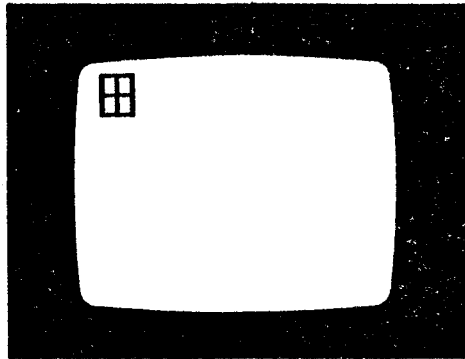
Step 4: Let go of **C** (it's OK, honest).

Step 5: Hold down **SHIFT** and hit +.

Step 6: Let go of **SHIFT** and hold down **C** again. Don't let go of it any more.

Step 7: Hit W, then **RETURN**.

Step 8: Hit Z, then E, then X, then **RETURN**.



## CHARACTER ANIMATION

Movies are really a sequence of still pictures. Each picture is a little different from the one that came before. The projector shows each picture for a very short time, then goes on to the next one. The scene becomes animated.

Computer animation works the same way. First the computer draws one picture, then it changes the picture slightly. The C-264 is fast enough to allow objects to move smoothly all around the screen in your games and practical programs.

You can't type fast enough to create animation. A movie is animated at a rate of 30 pictures per second. The changes must be fast enough to fool the eye. So you must use a program to draw a picture, wait for a split second, then change to a new picture.

To get the program to create pictures we use the PRINT statement with the graphic characters. The simplest type of animation involves alternating two characters to get the effect of movement.

### Exercise 8: Pulsing Ball

Type NEW and hit RETURN. Remember to hit RETURN after each line.

```
10 PRINT " [HOME] [SHIFT] Q "  
20 FOR L=1 TO 100  
30 NEXT  
40 PRINT " [HOME] [SHIFT] W "  
50 FOR L=1 TO 100  
60 NEXT  
70 GOTO 10
```

Type these keys.



Type RUN and hit [RETURN].

To get a more interesting effect, you can build a small picture from several graphic characters, then change a few of the characters while leaving others in the same place. This gives the effect of part of an object moving.

### Exercise 9: Jumping Jacks

Type NEW and hit RETURN. Remember to hit RETURN after each line.

```
10 PRINT " HOME SHIFT M SHIFT W SHIFT N "  
20 PRINT " SPACE C + SPACE "  
30 PRINT " SHIFT N SPACE SHIFT M "  
40 FOR L=1 TO 100: NEXT  
50 PRINT " HOME SPACE SHIFT W SPACE "  
60 PRINT " C T C + C T "  
70 PRINT " SPACE C G C G "  
80 FOR L=1 TO 100: NEXT  
90 GOTO 10
```

Type RUN and hit RETURN.

In both examples of animation so far, we've worked on only one area on the screen. The next step is to move the animated figure around. The TAB function helps when you want to move objects from the left edge.

### Exercise 10: Crawling Snake

Type NEW and hit RETURN. Remember to hit RETURN after each line.

```
10 PRINT " SHIFT CLR "  
20 PRINT TAB (A) " SHIFT U SHIFT I SHIFT U SHIFT I "  
30 PRINT TAB (A) " SHIFT K SHIFT J SHIFT K SHIFT J "  
40 FOR L=1 TO 100: NEXT  
50 PRINT " SHIFT CLR "  
60 PRINT TAB (A+1) " SHIFT I SHIFT U SHIFT I SHIFT U "  
70 PRINT TAB (A+1) " SHIFT J SHIFT K SHIFT J SHIFT K "  
80 FOR L=1 TO 100: NEXT  
90 GOTO 10
```

Type RUN and hit RETURN.

Using characters like the ball (SHIFT Q), you can play video games on the screen. To move a ball, just erase the ball and replace it at a new position.

## Exercise 11: Moving Ball

Type **NEW** and hit **RETURN**. Remember to hit **RETURN** after each line.

```
10 PRINT " SHIFT CLR "  
20 PRINT " SPACE SHIFT Q ⇐CRSR ";  
30 FOR L=1 TO 50: NEXT  
40 GOTO 20
```

Type **RUN** and hit the **RETURN** key. Hit the **STOP** key when you want the program to stop moving the ball.

## CONTROLLING COLORS

Separate colors can be put into each part of the screen. The border can be one color, the background a different one, and each character can have its own color. You already know how to set the character colors using the keyboard. The **COLOR** command adjusts the color of the other screen areas.

Turn the border of your screen red by typing the command **COLOR 4, 3** and pressing the **RETURN** key. The number 4 in the command stands for the border area, and color number 3 is red (the same number as on the key marked **RED**).

Now type **COLOR 0, 7** and hit **RETURN**. The screen background turns blue. The number 0 stands for the background, while the 7 is blue (also the same as the keyboard).

The first number after the word **COLOR** stands for the area on the screen you want to change. Area 0 is the background, 1 is the character color, 4 is the border. You'll learn about areas 2 and 3 when you get into multi-color graphics later in this chapter.

Area #	Area Name
0	Background
1	Character
2	multi-color 1
3	multi-color 2
4	Border

Table 3: Screen Area Numbers

Each color also has an adjustable brightness level, called the luminance. You can add a number from 0 (darkest) through 7 (brightest) after the color number to vary the color. Type COLOR 4, 3, 0 and hit RETURN. The border becomes a dark red. Type COLOR 4, 3, 7 and the border changes to a bright red.

Color Key#	Color	Color Key#	Color
1	BLACK	9	ORANGE
2	WHITE	10	BROWN
3	RED	11	YELLOW-GREEN
4	CYAN	12	PINK
5	PURPLE	13	BLUE-GREEN
6	GREEN	14	LIGHT BLUE
7	BLUE	15	DARK BLUE
8	YELLOW	16	LIGHT GREEN

Table 4: Color Numbers

In short, the COLOR command looks like this:

COLOR area, color, luminance

Here is a quick program to show you all the C-264's colors:

#### Exercise 12: Luminance Bars

First type NEW and hit RETURN. Don't forget to hit RETURN after each program line.

```

10 COLOR 0, 7, 7
20 FOR M=0 TO 7
30 FOR N=1 TO 2
40 FOR L=1 TO 16
50 PRINT "  CTRL  HVS  " ;
60 READ A
70 COLOR 1, A, M
80 PRINT "  ";
90 NEXT
100 PRINT
110 RESTORE
120 NEXTN,N
130 COLOR 1, 2, 4
200 DATA 7,14,4,13,6,16,11,8,10,9,3,12,5,15,2,1

```

Now type RUN and hit RETURN. You will see a bright blue screen with each of the other 15 colors shown at each luminance level.

## HIGH RESOLUTION GRAPHICS

The C-264 screen contains 25 rows of 40 characters each, or 1000 total character positions on the screen. Each character is formed out of single dots, with 8 rows of 8 dots each making an entire character. Your screen has a total of 320 dots on each row, and 200 rows, or 64,000 dots all together. The Commodore 264 lets you control every dot.

Using normal graphics, you have limited control over the individual dots. You must use the 128 characters built into the C-264, which lets you create many pictures. But think of how many you could create if you could control each dot by itself!

The high resolution graphics ability of the C-264 lets you do just that. You can use commands that let you draw and erase dots, lines, circles, and other shapes.

There is one limit to high-res graphics. The C-264 can still only use one color in each character position. That is, each 8 by 8 space on the screen where characters usually go can still only have one color (aside from the background color). You can use different colors for each different character position, but only one color within that position.

### Exercise 13: Color Clashes

Start by typing NEW and hitting RETURN. Hit the RETURN key after typing each line. After the program is complete type RUN and hit RETURN.

```
10 COLOR 0,1
20 GRAPHIC 1,1
30 FOR L=2 TO 16
40 COLOR 1,L,2
50 DRAW 1,0,L*12 TO 319,L*12
60 DRAW 1,L*18,0 TO L*18,199
70 NEXT
80 FOR L=1 TO 5000:NEXT
90 COLOR 1,2,3
100 GRAPHIC 0
```

Notice that the colors change near the intersections.

To switch from normal graphics (also called Text Mode) to high-res, just type the command GRAPHIC 2,1 and hit RETURN. The screen goes blank and the cursor reappears near the bottom of the screen. The C-264 divides the screen into 2 separate sections: the top for graphics and the bottom five lines for text. If you don't want the bottom five lines for text, you can use the command GRAPHIC 1,1, but you won't be able to see any commands you type.

You can switch back and forth from graphics to text using the GRAPHIC command. The command GRAPHIC 0 switches the screen back to text, while GRAPHIC 2 switches back to high-res without erasing the screen. Adding ,1 after the command erases the screen.

In general, the GRAPHIC command looks like this:

GRAPHIC mode, clear

this part is optional.

Mode Number	Effect
0	Text
1	High-res
2	High-res + text
3	Multi-color
4	Multi-color + text

Table 5: Graphic mode numbers

Clear Number	Effect
0	Don't clear screen
1	Clear screen

Table 6: Clear numbers for GRAPHIC command

There is another way to clear the high-resolution screen. The command SCNCLR will erase the screen without changing the graphic mode.

Once you use high-resolution graphics, the computer loses 10K of memory, which is set aside for a more detailed screen. When you are through using graphics, you can reclaim this memory by using the command GRAPHIC CLR. In most programs it won't make a difference, but it's a good habit to get into.

## POINTS, LINES, AND LABELS

Type the commands `GRAPHIC 2,1: DRAW 1,0,0` and hit RETURN. Look closely at the upper left corner of the screen. The C-264 drew a black dot there.

In the `DRAW` command, the first number is either 1 (draw a dot) or 0 (erase a dot). The next two numbers are for the row and column positions for the dot. So if you wanted to draw a dot at row 17, column 20, just type `DRAW 1,17,20`. To erase the same dot type `DRAW 0,17,20`.

The `DRAW` command can also draw a line between any two points. Just add the word `TO` and the coordinates of the other end, like this: `DRAW 1,1,1 TO 100,100`. This draws a line from 1,1 to 100,100.

If you are used to drawing graphs in math, you might get a little confused at first while using the computer. The coordinate system in the C-264 is upside down from what you're used to. In math the 0,0 point would be at the lower left corner of the screen, but on the computer it is the upper left corner. You'll get used to the system in the computer as you practice.

Once you have put a point or line on the screen, you can draw a line from it to any other point like this: `DRAW 1 TO 150,50`. This draws a line from the last point drawn to row 150, column 50. If your program uses a lot of `DRAW TO` commands, you could position the first dot at a position on the screen by using the `LOCATE` command, as in `LOCATE 100,100`.

The `DRAW` command can have several forms, such as:

COMMAND	RESULT
<code>DRAW color source, row, column</code>	POINT
<code>DRAW color source, row, column TO row, column</code>	LINE
<code>DRAW color source TO row, column</code>	LINE FROM LAST POINT

Table 7: Versions of the `DRAW` Command. Color source is 0 for background, 1 for foreground.

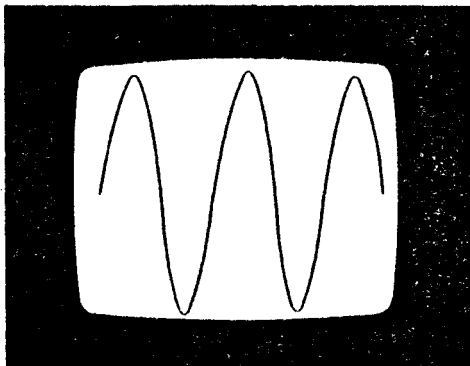
To erase points or lines on the screen, use the `DRAW` command followed by the number 0. If you created a point with `DRAW 1,1,1`, you can erase it with `DRAW 0,1,1`. A line created with `DRAW 1,1,1 TO 100,100` is erased by `DRAW 0,1,1 TO 100,100`.



#### Exercise 14: Sine Curve

Type NEW and hit RETURN. Remember to hit the RETURN key after each line, then type RUN.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 LOCATE 0,100
50 FOR X=1 TO 319
60 Y= INT( 100+ 99* SIN( X/20))
70 DRAW 1 TO X,Y
80 NEXT
90 FOR L=1 TO 5000
100 NEXT
110 GRAPHIC 0
```



### Exercise 15: Sine Plot

Don't type NEW after exercise 9. Just change line 70 to:

```
70 DRAW 1, X, Y
```

This program plots the same curve using points instead of lines.

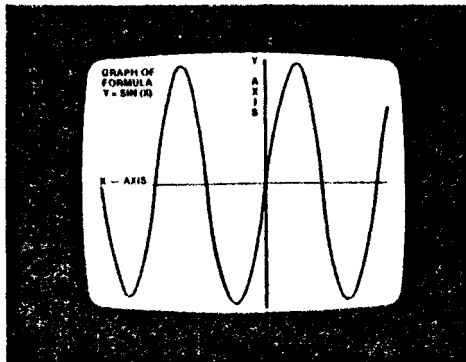
### USING TEXT IN GRAPHS

Your graphs will be more attractive if you use words to label them. You can use the CHAR command to mix text right into a high resolution drawing. For instance, the command CHAR 1,0,5"HELLO" puts the word HELLO into the sixth row at the left edge of the screen. The first number after the word CHAR is either 1 (for draw) or 0 (for erase). The next two numbers are the column and row where the text will appear.

### Exercise 16: Sine Graph With Labels

Leave exercise 14 or 15 in the computer: don't type NEW. Add these lines:

```
81 CHAR 1,0,0,"GRAPH OF": CHAR 1,0,1,"FORMULA"  
82 CHAR 1,0,2,"Y=SIN(X)"  
83 DRAW 1,0,100 TO 319,100: DRAW 1,189,0 TO 189,199  
84 CHAR 1,0,12,"X-AXIS": CHAR 1,22,0,"Y"  
85 CHAR 1,22,2,"A": CHAR 1,22,3,"X"  
86 CHAR 1,22,4,"I": CHAR 1,22,5,"S"
```



## SQUARES, CIRCLES, POLYGONS, AND PAINTING

### DRAWING RECTANGLES

Using the DRAW command, you can draw pictures by plotting many dots or lines. To draw a square, you can use the command DRAW 1,0,0 TO 100,0 TO 100,100 TO 0,100 TO 0,0.

The C-264 includes a command to make it easier to draw squares and other rectangular shapes. The BOX command lets you pick the points of 2 opposite corners of the square. To duplicate the same box as in the above example, just use BOX 1,0,0,100,100. The number 1 again means that you want to draw and not erase. The next four numbers are the coordinates of the box's opposite corners, (0,0) at the upper-left corner and (100,100) near the middle of the screen.

The BOX command can form any rectangle just by changing the corners. You can even rotate the box by specifying an angle (in degrees) after the last coordinate, like this: BOX 1,50,50,100,100,45. The box rotates 45 degrees clockwise.

If you would like to draw a solid box instead of just the outline, you just add a comma 1 after the angle. A solid box at the center of the screen is shown as BOX 1,100,50,220,150,,1. Notice that you need a comma to hold the place of the angle, even though you don't want the box rotated.

Some typical forms of the BOX command are:

COMMAND	EFFECT
BOX1, row1, column1, row2, column2	Outline
BOX1, row1, col1, row2, col2, angle	Rotated
BOX1, row1, col1, row2, col2, , fill	Solid box
BOX0, row1, col1, row2, col2, angle, fill	Erase area of screen

Table 8: Forms of the BOX Command

### Exercise 17: Rotating Boxes

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 2,1
40 A = RND(1)* 20+ 10
50 FOR L=0 TO 359 STEP A
60 BOX 1, 100, 30, 220, 130, L
70 NEXT
80 FOR L=1 TO 2000: NEXT
90 GRAPHIC 0,1
```

## Exercise 18: Colored Boxes

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```
5 TRAP 60
10 GRAPHIC 2,1
20 DEF FNA(X)= INT( RND(1)* X)
30 COLOR 1, FNA(15)+1
40 BOX 1, FNA(320), FNA(160), FNA(320), FNA(160),, 1
50 GOTO 30
60 COLOR 1,2,3: GRAPHIC 0
```

Hit RETURN and type RUN. Hold down the STOP key to end the program.

This program draws different colored squares all over the screen. You'll notice some parts of the screen changing when other parts near them change. The reason for this was discussed in section 5 earlier in this chapter.

## DRAWING CIRCLES

The C-264 also has commands for drawing circles. Like the BOX command, we can vary the shape of the circle to form an oval (also called an ellipse), and we can rotate the oval. We can also just draw a section of the shape (called an arc).

This command draws a circle in the center of the screen: CIRCLE 1,160,100,50. This tells the C-264 to draw a circle with its center at row 160 and column 100, with a radius of 50. This actually produces an oval, since the dots on the screen are taller than they are wide. To change this to a real circle you must add a separate number to tell that the height is different from the width, like this: CIRCLE 1,160,100,50,42.

The C-264 can also draw a square, triangle or other polygon using the CIRCLE command. Just tell the computer how many degrees to go between points on the circle, like this: CIRCLE 1,160,100,50,42,, ,120. This command draws a triangle, since each side is 120 degrees. A simple formula to get the angle for a polygon with N sides is  $360/N$ .

### Exercise 19: Polygons

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```
10 GRAPHIC 2,1
20 INPUT"HOW MANY SIDES";A
30 IF A<2 OR A>100 THEN PRINT "DON'T BE RIDICULOUS":
   GOTO 20
40 CIRCLE 1,160,80,40,33,,,,360/A
50 GOTO 20
```

You can choose to draw only an arc instead of a whole circle. The CIRCLE command accepts the starting and ending angles in degrees, right after the height number. The command CIRCLE 1,160,100,50,42,90,180 will display only the lower right section of the circle.

To rotate an oval, add the angle of clockwise rotation after the command, like this example: CIRCLE 1,160,100,100,20,, ,30.

The usual forms of the CIRCLE command are:

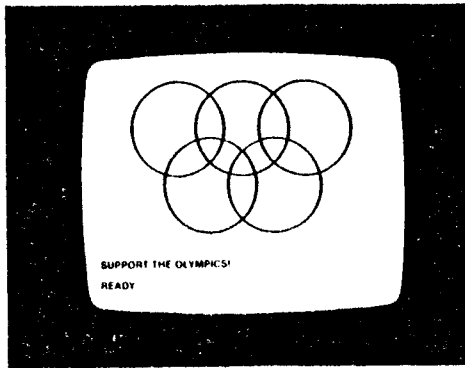
COMMAND	EFFECT
CIRCLE color source, center row, center column, radius	Oval
CIRCLE color source, c-row, c-col, width, height	Circle/oval
CIRCLE color source, c-row, c-col, wid, ht, start, finish	Arc
CIRCLE color source, c-row, c-col, width, height, , ,angle	Rotate oval
CIRCLE color source, c-row, c-col, wid, ht, , , ,point angle	Polygon

Table 9: Forms of the CIRCLE Command

### Exercise 20: Olympics

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```
10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 2,1
40 FOR L=1 TO 5
50 Y=50
60 IF L=2 OR L=4 THEN Y=100
70 X= L*35 + 50
80 CIRCLE 1,X,Y,50,42
90 NEXT
100 PRINT "SUPPORT THE OLYMPICS!"
```



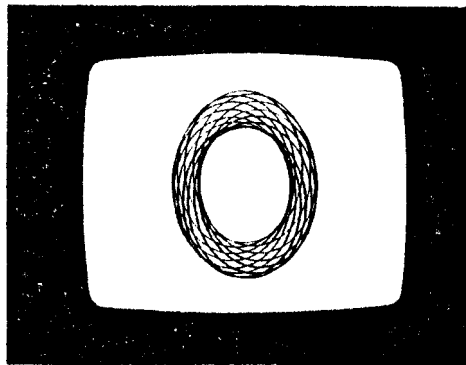
### Exercise 21: Rotating Circles

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```

10 COLOR 0,1
20 COLOR 1,2
30 GRAPHIC 1,1
40 A = RND(1)* 20+ 10
50 FOR L=0 TO 359 STEP A
60 CIRCLE 1, 160, 100, 80, 40,,, L
70 NEXT
80 FOR L=1 TO 2000: NEXT
90 GRAPHIC 0,1

```



The BOX command allows you to create solid blocks instead of outlines. But it doesn't let you color in irregular areas on the screen. That's where the PAINT command comes in.

In the last exercise, we created the 5-ring symbol of the Olympic games. By adding some PAINT statements to the program we can color in only the areas between the rings.

#### Exercise 22: Olympic Painter

Type in the program in Exercise 20, then add these lines:

```
110 FOR L=0 TO 2
120 PAINT 1,120 + 35 * L, 75
130 NEXT
```

The PAINT command will fill in any enclosed area up to the boundaries formed by any lines drawn on the screen. If there are no drawn lines, the screen is filled right to the edge.

### MULTI-COLOR GRAPHICS

The Commodore 264 high resolution graphics give you control over a great many dots on the screen, but you have seen that the ability to put colors close together is limited. Most high-res programs will probably only use one color.

To get around this, the C-264 has an in-between graphics mode called multi-color graphics. In multi-color graphics you can control half as many dots on each row as in high-res because each dot is twice as wide. You get 160 dots on each row, and you still get 200 rows.

To begin using multi-color graphics, refer to table 5 earlier in this chapter. You will see that the multi-color screen without text is GRAPHIC 3 and the multi-color screen with 5 lines of text is GRAPHIC 4.

Now look at table 3 on page 54, the color area table. You will see two areas that we haven't used yet, areas 2 and 3. These areas hold two extra colors. You can use any of the three colors (1, the text color; 2, an extra color; and 3, another extra color). These colors do not interfere with each other on the screen the way the high-res colors do in exercise 13.

### Exercise 23: Multi-color Olympics

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```
10 COLOR 0,1
20 GRAPHIC 4,1
30 FOR L=1 TO 5
40 Q=L: IF Q>3 THEN Q=Q-3
50 COLOR Q,L+1
60 Y=50
70 IF L=2 OR L=4 THEN Y=100
80 X= L*18 + 25
90 CIRCLE Q, X, Y, 25, 42
100 NEXT
110 PRINT "SUPPORT THE OLYMPICS!"
```

Color area 3, the second of the multicolor areas, has a special ability that none of the others has. Once you have drawn on the screen using area 3, you can change that color everywhere it appears on the screen by using the COLOR command. If you set the color with COLOR 3,5 and draw using that color, your graphics will appear in purple. If you then change the color with COLOR 3,6, all the purple areas would change to green. This doesn't work with any other area.

The Commodore 264 Programmer's Reference Guide contains more information about graphics.

### Exercise 24: Neon Sign

Type NEW then hit RETURN. Don't forget to hit the RETURN key after each line. Type RUN and hit RETURN at the end.

```
10 COLOR 0,1
20 GRAPHIC 3,1
30 COLOR 3,1
40 TRAP 200
50 DRAW 3,10,10 TO 10,100: DRAW 3,10,55 TO 30,55
60 DRAW 3,30,10 TO 30,100: DRAW 3,50,10 TO 80,10
70 DRAW 3,65,10 TO 65,100: DRAW 3,50,100 TO 80,100
80 FOR L=0 TO 7
90 COLOR 3,2,L
100 FOR M=1 TO 100: NEXT
110 NEXT
120 COLOR 3,1
130 FOR M=1 TO 100: NEXT
140 GOTO 80
200 GRAPHIC 0: COLOR 1,2,7
```



## CHAPTER 7

# MAKING SOUND AND MUSIC ON THE C-264

- Introduction
- Making some music
- The C-264 music machine

•

## INTRODUCTION

Here is a short program to make music on the C-264. Just type the program exactly as it looks, and remember to press RETURN at the end of each line. When the program is entered, type RUN and then press RETURN. When the program asks you to enter a number, type any number from 0 to 1023 and press RETURN.

```
10 VOL7
20 DO
30 INPUTX
40 SOUND1,X,10
50 LOOP UNTIL X=0
```

Here's how to play a note on the C-264. If you typed in the program at the beginning of this chapter, press the 0 key and press RETURN to stop the program.

Example #1:

First:

Type VOL 8 and press **RETURN**

Second:

Type SOUND 1,266,60 and press **RETURN**

You should hear a note play for about a second and then stop. If you don't hear anything, turn up the volume of your television or monitor and try it again.

These two steps are the only commands that you need to know to play music on your C-264. Let's look at what these two commands do.

## VOL

The VOL command controls the volume of the notes that the C-264 will play. Think of the first three letters of the word 'volume' to remember the VOL command. The number that comes after VOL is the setting for the volume. Think of the VOL command as a volume knob on the C-264. When the knob points to zero, the volume is off and you won't hear anything. When the knob points to 8, the volume is turned up all the way, and the C-264 will play as loud as it can.

Try the first example again and use a different number after the command VOL. As the number you choose gets smaller, the note that is played will get quieter.

## SOUND

The SOUND command tells the C-264 everything about the note you want to play. The SOUND command has three numbers after it that describe the note you want to play.

## VOICE

The first number after the word SOUND can be a 1, 2 or 3. This number selects which of the two voices you want to use. Just like a choir has different voices, so does the C-264. Each of the voices is a little different, and you have to choose the one that best fits what you are trying to play.

Voice 1 — This voice plays tones. You select this with a 1 following the SOUND command. It can go very high, and pretty low too.

Voice 2 — This voice can be used for tones or for noise. Use a 2 after the SOUND command to select tones, and a 3 to select noise.

You can use noise to make sound effects like thunder and rain.

## NOTE

The second number after the word SOUND is the note value. This can be any number from 0 to 1023, and it tells the C-264 how low or high a note to play. As the numbers get bigger, the notes get higher.

Noise is "white" noise only in the range 600-940. You can use register values outside this range to create other interesting sound effects.

Here is a chart that shows all of the notes in one scale, along with the note value to use. For example, if you want to play an 'E', you would look for 'E' on the chart and then read across to 854. The 854 would be the note value to use in the SOUND command. There is a complete chart of notes for the C-264 in Appendix G.

NOTE	SOUND REGISTER VALUE	ACTUAL FREQUENCY (HZ)
A	770	440.4
B	798	494.9
C	810	522.7
D	834	588.7
E	854	658
F	864	699
G	881	782.2

Try putting the following program into the C-264:

```

NEW
10 VOL 7          -----turn on volume
20 X=0
30 DO
40 SOUND 1,X,5      -----play note
50 X=X+5
60 LOOP UNTIL X=1020
70 VOL 0          -----turn off volume
80 END

```

Type this program in exactly as it is shown. Don't forget to press RETURN after each line. When you finish typing it in, and you have checked for mistakes, type RUN and press RETURN. The C-264 will show off how low and high it can go and play many notes in between.

## DURATION

The third number after the word SOUND is called the duration. This tells the C-264 how long to play the note. This number can be anything from 0 to 65535. This number is a timer, except instead of ticking one number each second, it ticks 60. A duration of 60 will keep the note on for one second. You don't need to understand this right now, just remember that the bigger the number, the longer the note will stay on. In fact if you use 65535, the note will stay on for over 16 minutes.

## MAKING SOME MUSIC

Now that we've looked at the commands that the C-264 uses for making sound, let's make some music. Here are a couple of programs to type:

The first program turns the keys from 1 through 8 into a piano. Type in the program and then type RUN.

### Color Piano

```
5 SCNCLR
6 FORX=1TO8:READN(X):NEXT
10 VOL7
20 DO
30 GETA$:IFA$=""THEN30
35 A=ASC(A$):IFA<49ORA>56THEN50
36 N=A-48
40 SOUND1,N(N),5
45 COLOR0,N,3
50 LOOP UNTIL A=32
55 VOL0:COLOR0,2,7
100 DATA169,262,345,383,453,516,571,596
```

Press numbers 1 through 8 to get notes. The screen even changes colors with the different notes. When you finish playing, you can press the space bar to stop the program.

Here are the numbers to press for a familiar song:

### Twinkle, Twinkle, Little Star

```
1 1 5 5 6 6 5
4 4 3 3 2 2 1
5 5 4 4 3 3 2
5 5 4 4 3 3 2
1 1 5 5 6 6 5
4 4 3 3 2 2 1
```

Here is a program that plays a song by reading a list of DATA statements. The DATA statements are in pairs. The first number is the note value for the SOUND command and the second number is the duration for the SOUND command.

### Row Boat

```
10 VOL 7
20 DO
30 READ X,Y
40 SOUND1,X,Y
45 SOUND1,1022,5
50 LOOP UNTIL X=0
60 END
100 DATA169,45,169,45,169,30
110 DATA262,15,345,45,345,30
120 DATA262,15,345,30,383,15
130 DATA453,60,596,45,453,45
140 DATA345,45,169,45,453,30
150 DATA383,15,345,30,262,15
160 DATA169,60
200 DATA0,0
```

This program plays notes going up and down scales at different speeds, and displays some color bars along with them.

Hold down CTRL and press each color key once. Then hold down **C** and press each color key once. Then type a quote.

```
10 A$=""
20 VOL7
30 DO
40 D=INT(RND(0)*5)+2:REM DURATION
50 S=INT(RND(0)*300)+700 :REM START
60 R=INT(RND(0)*(1020-S)):REM RANGE
70 P=INT(RND(0)*30)+5 :REM STEP
80 T=SGN(RND(1)-.5):IFT=0THEN80
90 FORZ=STOS+T*RSTEPP*T
100 SOUND1,Z,D
110 Y=(ZAND15)+1:FORX=1TOD
120 PRINT"C";MIDS(A$,Y,1);" ";
130 NEXT:NEXT
140 LOOP
```

press CTRL and RVS ON

The last program is a little longer. This is the "GREAT C-264 MUSIC MACHINE".

### The C-264 Music Machine

```
5 GOSUB1000
6 FORX=1TO9:READN(X):NEXT
8 CHAR1,8,1,"*THE GREAT MUSIC MACHINE*"
10 VOL7
20 DO
30 GETA$:IFA$=""THEN30
35 A=ASC(A$):IFA<49ORA>57THEN50
36 N=A-48
40 SOUND1,N(N),4
45 GSHAPENS,150,8*(6+(9-N)),4
46 FORZ=1TO50
47 GSHAPENS,150,8*(6+(9-N)),4
50 LOOP UNTIL A=32
55 VOLO:GRAPHICO:SCNCLR
60 END
100 DATA345,383,453,516,571,596,643,685,704
1000 GRAPHIC1,1
1010 FORY=60TO124STEP16
1020 DRAW 1,100,Y TO 200,Y
1030 NEXT
1040 A$="FEDCBAGFE"
1050 FORX=1TO9:C=13
1060 IFINT(X/2)=X/2THENC=14
1070 CHAR1,C,X+6,MID$(A$,X,1),0
1075 CHAR1,C+10,X+6,RIGHT$(STR$(10-X),1)
1080 NEXT
1090 FORX=1TO8:FORY=11TO16:DRAW1,X,Y:NEXT:NEXT
1100 Y=1:X=8:DRAW1,8,16 TO X,Y
1110 SSHAPENS,1,1,8,16
1120 GSHAPENS,1,1,4
1130 RETURN
```

When you press a key from 1 through 9, the note will be played, and a note will appear on the staff on the correct line.

As this chapter has shown, it's not hard to write your own sound program. The programs in this chapter just begin to show the music capabilities of the C-264. Don't be afraid to try new sounds and noises and create your very own masterpiece.



## CHAPTER 8

# BASIC TRICKS

## CONTROLLING THE SPEED OF YOUR PROGRAMS (FOR . . . NEXT)

Did you ever want to SLOW DOWN a program? You can control the speed of your program by using a special two-part command called a time delay loop. Time delay loops use the FOR . . . NEXT command to tell the computer to pause and count to some number before continuing, slowing down your program.

Here's what a time delay loop looks like — you can insert a line like this almost ANYWHERE in your program to slow it down:

FOR T = 1 TO 500: NEXT

You can use a different variable than T but we use T because it helps us distinguish Time loops from other loops in our programs.

These two numbers can be any two numbers. The computer will "count" from the first number to the second, and the larger the difference, the longer the time delay. For example 1 TO 100 is a much shorter time delay than 1 TO 1000.

Let's try an example to see how it works. Type the word NEW and press the RETURN key (to erase any previous programs), then type the following program exactly as shown:

```
10 PRINT"  COMMODORE"  
20 PRINT"  TECHNOLOGY"  
30 GOTO10
```

RETURN

RETURN

RETURN

Now type the word RUN and press the RETURN key. That's too fast! Let's slow it down. First, let's decide where we want the program to slow down. After the word COMMODORE appears on the screen we want to make the computer pause so we can read the word, so a good place to put a time delay would be right after LINE 10 . . . at a new line numbered 15 so it fits between Line 10 and Line 20.

We also want some time to read the second message, which is the word TECHNOLOGY, so let's put a time delay after Line 20, on a new line numbered 25 so it fits between LINE 20 and LINE 30. Type these two lines exactly as shown (the computer will automatically insert them into your program):

```
15 FOR T = 1 TO 500: NEXT  
25 FOR T = 1 TO 500: NEXT
```

RETURN

RETURN

Type the word RUN and press the RETURN key. To stop your program, hold down the RUN/STOP key. Then type the word LIST and press RETURN to see the complete program. You'll see FOR. . . NEXT loops used as time delays in various programs in this manual. Remember, whenever you see the variable T used in a FOR. . . NEXT loop in this manual, it is being used as a time delay.

**TIP:** To make the messages flash faster, try changing the number 500 in Lines 15 and 25 to the number 100.

### **AN EASY SOUND EFFECT (FOR . . . NEXT . . . STEP)**

FOR . . . NEXT . . . STEP can be used creatively in sound effects. In this program we are going to use a FOR . . . NEXT loop with a negative step, so we can count down from a high number to a lower one.

```
10 VOL7
20 FOR S= 1000 TO 700 STEP - 25
30 SOUND1,S,1
40 NEXT
```

Type RUN and press RETURN. Hear the sound effect? The key is LINE 20, where we used a number range from 1000 to 700 going DOWN THE SCALE. Then we STEPPed - 25 numbers at a time. Finally, in LINE 30 we played each note very fast by setting the DURATION to 1. Experimenting with different number and duration values can give you some very interesting effects.

### **PROGRAMMING TIP:**

You can make a long program shorter by combining more than one command on a line. For example, you can write the musical sound effect program on ONE LINE, by separating the various commands with colons (:), like this:

```
10 VOL7:FOR S= 1000 TO 700 STEP - 25 :SOUND1,S,1 :NEXT
```

What this does is create a ONE-LINE musical tone or sound effect that you can easily insert into your BASIC programs where you want a quick musical prompt. This technique is called "crunching" your program. Crunching makes the program use less memory, run faster, and is the way to get the most out of the memory capability of your computer.

## UNDERSTANDING RANDOM NUMBERS

Take 10 pieces of paper and write a number from 1 to 10 on each piece. Next, put the 10 pieces of paper into a hat or other container where you can't see them. Now, cover your eyes and draw out one piece of paper. What number did you get? That number is a RANDOM number. Now put the number back into the hat, mix up the papers and draw again. Each time you draw a number, put that number back in the hat so there are always 10 pieces of paper to choose from. Keeping 10 numbers in the hat means you always have 10 random numbers in the hat. When you take a number, you DON'T know what number is going to come up next, but you DO know that the number will be between 1 and 10. This is the basis of RANDOM NUMBERS.

In programming, random numbers usually have a RANGE. This means there's an UPPER LIMIT and a LOWER LIMIT to the numbers you can draw. In our hat example, the range of number is 1 to 10. The lower limit is "1" and the upper limit is "10", which means that ANY NUMBER FROM 1 TO 10 CAN COME UP AT RANDOM WHEN YOU DRAW.

Now let's see how your Commodore computer handles random numbers, and some of the fun things you can do with them. Here's a program that generates 5 completely RANDOM numbers:

```
10 FOR X = 1 TO 5: PRINT RND(X): NEXT
```

**RETURN**

These random numbers are all rather complex, with several places on the right side of the decimal point . . . but most uses for random numbers require WHOLE NUMBERS. You can make your numbers come out as whole numbers (without decimal places) by using the INTeger function, which cuts off all the digits on the RIGHT side of the decimal point. The next section gives you a formula for using random numbers.

## AN EASY RANDOM NUMBER FORMULA

Here's a simple formula for generating random numbers in any range you want. You can use this formula almost anywhere you would use a variable or number in your program.

**INT(RANGE\*RND(1)) + LOWER LIMIT**

The INT command tells the computer to cut off any decimal places and only give you whole numbers like 1, 45, or 320, instead of numbers like 1.223, 45.6677, or 320.59. Whole numbers are easier to work with when using random numbers.

LOWER LIMIT in our formula means the lowest number you want the computer to choose from.

RANGE means how many numbers are in the total group.

For example, if you want to choose a random number from 1 to 5, the LOWER LIMIT will be 1 and the RANGE will be 5. If you want to choose a random number from 15 to 20, the LOWER LIMIT will be 15 and the RANGE will be 5 because you are still choosing from a total group of 5 numbers. If you're choosing numbers from 2 to 100, the LOWER LIMIT is 2 and the RANGE is 98. See how it works? Let's try out a program:

```
10 PRINT INT (5*RND(1)) + 1
```

Type RUN and press RETURN. Type RUN again . . . and again. Each time you get a RANDOM number from 1 to 5. Now let's PRINT 15 random numbers with the LOWER LIMIT 1 and the RANGE 5 . . . note that all 15 numbers we choose will be selected at random from between 1 and 5:

```
10 FOR X= 1TO15
20 PRINT INT (5*RND(1)) + 1
30 NEXT
```

Type RUN and press RETURN.

The easiest way to use this formula is to make it into a user defined function as follows:

```
10 DEF FNR(X)= INT(X*RND(1)) + 1
```

This gives us random numbers in the range from 1 to X.

EXAMPLE using a defined function:

```
10 DEF FNR(X)= INT(X*RND(1)) + 1
20 DO
30 COLOR1,FNR(16),5
40 PRINT "THE SEARCH GOES ON"
50 LOOP
```

Using the defined function saves memory space when you use the function more than once, and makes your programs easier to read.

## GRAPHICS PROGRAM USING COLOR, GRAPHIC, SCNCLR, AND CIRCLE

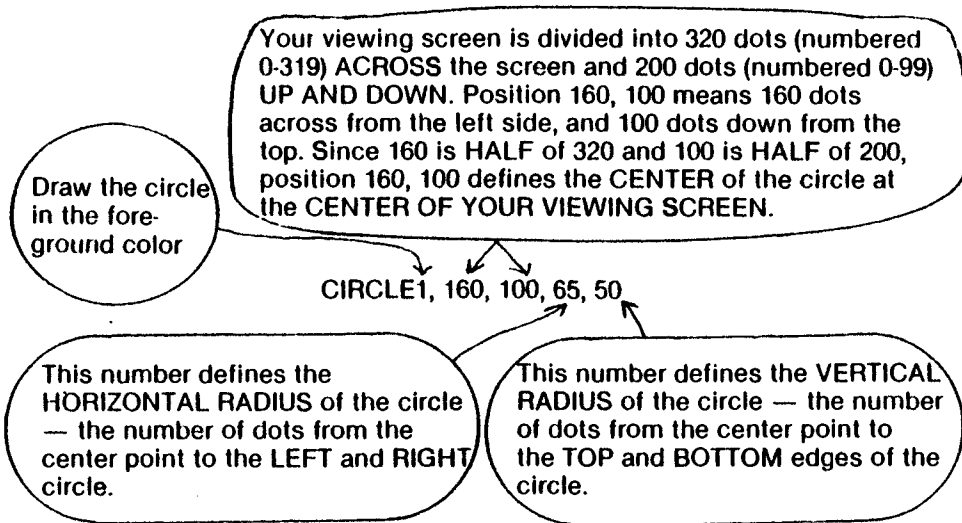
This program draws a circle on your screen:

```
10 GRAPHIC1,1
20 COLOR0,14: COLOR1,1: COLOR4,1
30 CIRCLE1,160,100,65,50
```

LINE 10 sets the computer to GRAPHIC1 "high resolution" graphics and clears the graphic screen of any previous graphics that might have been drawn there. It's always a good idea to clear the screen at the beginning of any graphic picture, so the new picture doesn't get mixed up with a previous picture.

LINE 20 defines the COLORS we're going to use. COLOR0 is the background screen color. COLOR1 is the LINE DRAWING COLOR. COLOR4 is the border color. The second number in each COLOR command sets the actual color. Colors range from 1 to 16 (and are shown on Page 48). In our example, we set the background (COLOR0) to 14, which is light green. We set the DRAWING COLOR (COLOR1) to 1, which is black, and we also set the border (COLOR4) to 1, which is black.

LINE 30 is the CIRCLE COMMAND — here's what the numbers mean:



## CONCENTRIC CIRCLES (COLOR, GRAPHIC, SCNCLE, CIRCLE)

One of the fun things you can do to impress your friends is to use your computer's graphic capabilities to create interesting patterns and designs. This program creates beautiful concentric circles with exquisite optical effects.

10 COLOR1,7,2

Drawing color(1) is blue(7), luminance is dark (2)

20 GRAPHIC1,1

Graphic 1 = high resolution

30 X = 1:Y = 1

Variables X and Y both equal 1

35 DO UNTIL Y = 49

Start the loop

40 CIRCLE1,160,100,X,Y

Circle1 means draw the circle in the same color as the DRAWING COLOR, which is blue. 160,100 tells the computer where to put the CENTER of the circle, which happens to be the center of the screen. X is the horizontal radius, Y is the vertical radius. We defined them both as 1 in Line 30 and change them in Line 50.

50 X = X + 2:Y = Y + 2

In Line 40 above, a small circle was drawn with an X-Y radius of 1. Now we increase the radius by 2 and go back and draw a slightly larger circle, again and again, each circle a little larger until we have a series of concentric circles.

60 LOOP

70 GETKEY AS\$: GRAPHIC0:END

Wait for user to hit a key before going back into text mode.

Note in Line 10 we set the line drawing color to dark blue. COLOR1 means drawing color and 7 means set it to blue, but there's a THIRD number here! We didn't show you this feature in the previous example, but if you add a THIRD NUMBER to your COLOR command, you can designate LUMINANCE. Luminance can be thought of as the LIGHTNESS or DARKNESS of a color. In other words, you can set luminance levels from 0 (darkest) to 7 (brightest). We wanted a very deep blue so we chose a luminance level of 2. Try changing the luminance levels and see what happens to the color.

## **CIRCLE PATTERNS**

Here are a couple of one line programs that show off the circle command.

```
10 GRAPHIC1,1:FOR T = 1 TO 180 STEP6: CIRCLE1,80,80,60,60 , , ,  
    T,120:NEXT
```

and

```
10 GRAPHIC1,1:FOR T = 1 TO 180 STEP6: CIRCLE1,80,80,60,60 , , ,  
120,I:NEXT
```

## **PRINTING THE CURSOR IN SCREEN POSITIONING**

The key to positioning a word or graphic image where you want it on the screen . . . or even making it MOVE in animated programs . . . is using the CHAR statement. While it is possible to do it with strings containing the cursor movement keys, you will soon agree that the CHAR statement is much easier to use:

```
10 CHAR 1,3,2,"3 SPACES RIGHT, 2 SPACES DOWN"
```

Here's a bit more complicated case of cursor positioning:

```
15 CHAR 1,0,10,"10 SPACES DOWN";  
20 FOR T = 1 TO 2000: NEXT  
25 CHAR1,0,5,"5 SPACES UP";  
30 FOR T = 1 TO 2000: NEXT  
35 CHAR 1,0,0,"HOME POSITION";  
40 FOR T = 1 TO 2000: NEXT  
45 CHAR 1,10,0,"10 SPACES RIGHT";  
50 FOR T = 1 TO 2000: NEXT  
55 CHAR 1,12,12,"12 DOWN, 12 LEFT";  
60 END
```

You can use this same technique to PRINT GRAPHIC SYMBOLS anywhere using the keyboard graphics explained in Chapter 6.



## CHANGING COLORS

When you turn on your C-264, the screen is BLUE and the characters on the screen are BLACK. As the last program showed, you can change these colors. Your C-264 has 16 colors each with 8 luminance levels (except black), so there are a lot of different color combinations. Here's a program that shows you all the combinations by changing the background color to each one.

```
10 DO
20 FOR C = 1 TO 16
30 FOR L = 0 TO 7
40 COLOR 0,C,L
50 FOR X = 1 TO 100:NEXT X
60 NEXT L:NEXT C
70 LOOP
RUN
```

Here's how to change the colors to a combination you like:

1. Choose the colors you want from this list:

- |          |                 |
|----------|-----------------|
| 1 black  | 9 orange        |
| 2 white  | 10 brown        |
| 3 red    | 11 yellow-green |
| 4 cyan   | 12 pink         |
| 5 purple | 13 blue-green   |
| 6 green  | 14 light blue   |
| 7 blue   | 15 dark blue    |
| 8 yellow | 16 light green  |

You may select any of the 8 luminance levels for each of these colors.

2. To change your screen color, type this:

COLOR 0,X,Y      but instead of an X, type in the number of the color you want, and instead of Y, select the luminance you want.

For example:

```
10 COLOR 0,2,0
```

sets the background to a dark shade of grey.

3. To change the color of the characters, type this:

COLOR 1,X,Y      but instead of an X, type in the number of the color you want, and instead of Y, select the luminance you want.

For example:

```
10 COLOR 1,3,7
```

sets the characters printed to red.

## USING THE SEMICOLON

The semicolon is very important in BASIC. Sometimes it might be the answer to one of those sticky programming problems.

The semicolon prevents BASIC from moving the cursor to the beginning of the next line after executing a statement. A common use is when you CLEAR THE SCREEN.

Try this:

```
10 PRINT"<SHIFT> <CLR/HOME>"  
20 PRINT"TITLE"
```

RETURN

RETURN

Type RUN and press RETURN. What happens? The word TITLE appears in the upper left corner of your screen . . . but it's ONE LINE DOWN FROM THE TOP! Whenever you CLEAR the screen, the next item you PRINT will automatically appear on the second line from the top of the screen UNLESS YOU USE A SEMICOLON.

Now, add a semicolon to the end of Line 10, like this:

```
10 PRINT"<SHIFT> <CLR/HOME>";
```

RETURN

Type the word LIST and press RETURN to see the new program, then type the word RUN and press RETURN. The word TITLE appears on the TOP line! Of course, the same technique works with a real title, and that's one of the semicolon's programming uses.

You can eliminate the need for the semicolon and save yourself some programming space if you include the CLEAR SCREEN command in quotation marks along with the message you're PRINTing, like this:

```
10 PRINT"<SHIFT> <CLR/HOME> TITLE"
```

This automatically PRINTs the word TITLE on the first line because the CLEAR SCREEN command and word you want to PRINT are combined on the same line.

## PRINTING GRAPHICS SIDE-BY-SIDE

Let's explore another use of the semicolon, this time with graphics. Type the word **NEW** and press **RETURN**. We want to **PRINT** 40 hyphens across the screen, to make an attractive border or dividing line between two parts of a program we're writing. There are two ways. One way is to type all 40 symbols in Line 20, like this:

```
10 PRINT"<SHIFT> <CLR/HOME>";
```

```
20 PRINT"....."
```

Another way to do exactly the same thing is to use a **FOR . . . NEXT** loop, like this:

```
10 PRINT"<SHIFT> <CLR/HOME>";  
20 FORX = 1TO40:PRINT" — ";NEXT  
30 PRINT
```

Type the graphic  
symbol here

Run this program and 40 graphic lines appear across your screen. The semicolon tells the computer to **PRINT** each of the 40 hyphens in the **FOR . . . NEXT LOOP** right **NEXT TO EACH OTHER** on the screen. Without the semicolon, the hyphens would **PRINT** in a vertical column, each one on a separate line. Try removing the semicolon and **RUN** the program. Try substituting some of your computer's other graphic symbols instead of the hyphen.

## COMBINING LONG PRINT MESSAGES

Using a **BASIC** program to **PRINT** a long instruction or sentence on the screen can be difficult because each program line is limited to 80 **COLUMNS** (two lines) on your screen and many messages are longer than that. Type **NEW** and **RETURN**, then enter this program:

```
10 PRINT"<SHIFT> <CLR/HOME> NOW IS THE TIME TO BUY A NEW  
COMMODORE DISK DRIVE AND PRINTER FOR YOUR COMM"  
20 PRINT"ODORE SYSTEM"
```

**RETURN**

**RETURN**

**RUN** the program. Too bad — you had to continue your sentence on a second program line (Line 20) but the lines don't match. You can make the two **PRINT** statements run together by adding a **SEMICOLON** at the end of Line 10, like this:

```
10 PRINT"SHIFT CLR/HOME NOW IS THE TIME TO BUY A NEW  
COMMODORE DISK DRIVE AND PRINTER FOR YOUR COMM";  
20 PRINT"ODORE SYSTEM"
```

**RETURN**

**RETURN**

Now **RUN** the program and you'll see that the last part of the word "**COMMODORE**" is automatically attached to the beginning of the word.

## INPUTS WITHOUT QUESTION MARKS

Here's a really handy programming technique — using INPUT statements WITHOUT question marks. Traditional INPUT statements have built-in question marks which typically appear on the NEXT LINE after the "prompt" or question, as in this sample program:

```
10 PRINT"<SHIFT> <CLR/HOME> ENTER YOUR NAME";INPUT N$
20 PRINT"<SHIFT> <CLR/HOME> YOUR NAME IS "N$
```

In Line 10 we CLEAR the screen, PRINT a prompt message, then provide an INPUT VARIABLE which we call N\$ (the variable we chose is arbitrary; it could just as easily be another legal string variable name like A\$, XY\$ or NN\$). In Line 20 we CLEAR the screen again and PRINT another message, this time using the name that was input. Here, N\$, which is the name the user typed in, is PRINTed because N\$ equals whatever the user typed in. But what if you don't want to use a question mark for the INPUT data? What if you want to have the user type in a LIST OF ITEMS (like an inventory list, for example, or some business data)? Here's a very helpful technique:

```
10 OPEN3,0
20 PRINT"<SHIFT><CLR/HOME> NAME:"; : INPUT#3,N$
30 PRINT CHR$(13) "YOUR NAME IS "N$
40 CLOSE3
```

Line 10: We open device number 0, which is the keyboard. The keyboard has a device number just like a Datasette, disk drive or printer (the Datasette device number is 1, the single disk device number is number 8 and the printer is number 4). Opening the keyboard lets you get input while side-stepping some of the built-in functions like the automatic question mark which appears when you usually program an INPUT statement.

Line 20: Here we CLEAR the screen again and PRINT a prompt message which is NAME: — then we provide for the INPUT of the name which we designate N\$. INPUT#3 is the same as our usual INPUT statement except we are now INPUTTING to the screen so we add the # as shown.

Line 30: The name which was typed in (N\$) is included at the end of our simple PRINT statement.

## USING COMMODORE BUSINESS GRAPHICS

Your Commodore computer has a full set of keyboard graphics that include an excellent set of business graphics. These special lines and blocks let you create a wide variety of charts and graphs. Here's a quick lesson in how to set up your screen using these graphics:

To get this symbol  
hold down the shift  
key and press the  
"P" key

Hold down the  
"Commodore key"  
and press the  
"Y" key 38 times

```
10 PRINT"<SHIFT> <CLR/HOME>]←"  
20 PRINT"]<SPACE> TITLE OF YOUR CHART"  
30 FOR X = 1 TO 20:PRINT"  
]":NEXT  
40 PRINT"HOME" SPC(241) CATEGORY 1:  
50 PRINT SPC(200) CATEGORY 2:
```

## **MID\$, LEFT\$, RIGHT\$, and INSTR**

Three useful functions which are seldom explained in user's guides are the **STRING FUNCTIONS**. These functions let you define a string of characters, letters, numbers or graphic symbols inside **QUOTATION MARKS**, and then work with any part of the information contained in that string. The best way to describe how these functions work is to give you an example:

```
10 SCNCLR
20 C$ = "HELLOGOODBYE"
30 PRINT LEFT$(C$,5)
40 PRINT RIGHT$(C$,7)
50 PRINT MID$(C$,4,3)
60 PRINT "GOOD STARTS IN POSITION"; INSTR(C$,"GOOD")
70 MID$(C$,4,3) = "RUG":PRINT MID$(C$,4,3)
```

Type **RUN** and press **RETURN**. There are three words displayed. They all came from the **C\$** string letters we defined in **LINE 20**.

**LINE 10** **CLEARs** the screen.

**LINE 20** defines the string variable **C\$** as a group of letters contained between the quotation marks. These letters could also be numbers or graphic symbols.

**LINE 30** shows how **LEFT\$** works. **PRINT LEFT\$(C\$,5)** means: **PRINT** the first 5 characters contained in **C\$**. The first 5 characters form the word **HELLO**.

**LINE 40** shows how **RIGHT\$** works. **PRINT RIGHT\$(C\$,7)** means: **PRINT** the last 7 characters contained in **C\$**. The last 7 characters form the word **GOODBYE**.

**LINE 50** shows how **MID\$** works. **PRINT MID\$(C\$,4,3)** means: **PRINT** three characters from **C\$**, starting with the **FOURTH CHARACTER FROM THE LEFT**. The fourth character is an **L**. **PRINTing** three characters from that position displays the word **LOG**.

**LINE 60** shows how you can use the **INSTR** function to search for a word in another string.

**LINE 70** shows how you can change the definition of the middle of a string using the **MID\$** function.

## A MORE COMPLEX PROGRAM (GOSUB, LEFT\$)

Here's a different construction of the program we just ran. Pay close attention to this example because it may have applications to your own programs:

```
10 PRINT" ENTER YES OR NO AND PRESS RETURN":VOL 7
20 INPUTA$
30 IF LEFT$(A$,1)="Y"THEN GOSUB 1000
40 IF LEFT$(A$,1)="N"THEN GOSUB 2000
50 PRINT"PROGRAM WOULD CONTINUE FROM HERE ..."
900 END
1000 PRINT"GOOD": SOUND 1,800,40:RETURN
2000 PRINT"SORRY": SOUND 1,200,40:RETURN
```

The GOSUB commands at the end of LINES 30 and 40 tell the computer to jump to THE SUBROUTINE contained on the lines shown. For example, GOSUB1000 means go to Line 1000 and perform the action there. In this case, LINE 1000 displays the message GOOD and plays the sound effect. The RETURN command accompanies the GOSUB. Actually RETURN tells the computer to go back to the end of the line where it came from and continue on through the program. So if the user types a Y for yes, the message and tone in LINE 30 are provided, then the program RETURNS to Line 30 and continues down through Line 40 (nobody pressed N so nothing happens in Line 40) . . . to Line 50 where this message is displayed. The GOSUB and RETURN features in LINE 40 work just like LINE 30.

The END command is needed between the part of the program with the GOSUB and the LINES containing the actual subroutine. Lines with subroutines like those in LINES 1000 and 2000 should always be relatively high program line numbers so you have space to fit your program.

We can improve the program a bit. By now you shouldn't need the detailed explanations, so here's the program:

```
10 PRINT" ENTER YES OR NO":VOL 7
30 GETKEY A$: IF A$="Y"THEN GOSUB 1000: ELSE GOSUB 2000
50 PRINT"PROGRAM WOULD CONTINUE FROM HERE ..."
900 END
1000 PRINT"GOOD": SOUND1,800,40: RETURN
2000 PRINT"SORRY": SOUND1,200,40: RETURN
```

## CREATING A MENU

One of the most common beginnings for programs is a menu. A menu is just a list of options that you choose from. Here is a simple program that prints a menu on the screen and then prompts you for your choice.

```
10 SCNCLR:PRINTSPC(18)"MENU"
20 PRINT"[DOWN][DOWN][DOWN]"
30 PRINT"1. MAKE A CIRCLE"
40 PRINT"2. MAKE A SOUND"
50 PRINT"3. END"
60 PRINT"[DOWN][DOWN]"
70 INPUT"CHOOSE A NUMBER (1-3)";A
80 ON A GOSUB 1000,2000,3000
90 FORZ=1TO1000:NEXT
100 GRAPHIC0,1:GOTO10
1000 GRAPHIC1,1
1010 CIRCLE1,160,100,50
1020 RETURN
2000 VOL7
2010 SOUND1,200,60
2020 RETURN
3000 SCNCLR:END
```

Lines 10 through 60 display the title MENU and the three selections.

Line 70 prompts you for a number from 1 to 3. Line 80 uses a new command called ON . . GOSUB. This command will GOSUB 100, 200 or 300 depending on the value of A. It will GOSUB 100 ON A= 1, GOSUB 200 ON A= 2, and GOSUB 300 ON A= 3.

Line 90 makes the program wait for a little while and then go back to the beginning.

Lines 100 through 300 make the circle, the sound, and end the program.



## CATCHING ERRORS

This program demonstrates a very powerful tool that is built into the C-264. This tool is called ERROR TRAPPING. This means that if an error occurs in a program you can keep the program from crashing. Type the following program into your computer:

```
20 INPUT "TYPE ANY NUMBER";A
30 PRINT "32/A = "32/A
40 PRINT "THE END"
50 END
```

Run this program a couple of times with different numbers to be sure that it works. Now enter a zero as the number. You get an error and the program crashes, right? Add these lines to the program:

```
10 TRAP 1000
1000 IF ER = 20 THEN PRINT "DO NOT ENTER ZERO!":RESUME20
1010 PRINT ERR$(ER)" ERROR IN LINE"EL
```

Now run the program and enter a zero. The program now traps the error before the program can crash. Here is what the commands are:

Line 10 sets the trap and says that if an error occurs, go to line 1000.

Line 1000 checks which error occurred (IF ER = 20). If the error is DIVIDE BY ZERO, a warning message is printed. The next command is RESUME. This starts the program back up again. The number after the RESUME is the line number where the program will restart.

Line 1010 prints out the error and line number of any other errors that occur.

## COMMODORE FUNCTION KEYS

The COMMODORE FUNCTION KEYS at the top of your COMMODORE 264 keyboard are among the most useful features of your computer.

Here are just a few of the ways the COMMODORE FUNCTION KEYS can be used:

- Software developers use these keys to define special functions that make their software easier to use. Most COMMODORE SOFTWARE uses these keys, including the HELP key, which gives you instructions if you make a mistake or have a question while using a software program.
- You can use these keys to LOAD, SAVE, RUN and edit your own BASIC programs because the keys are programmed when you turn on the COMMODORE 264.
- Most important — you can program these keys yourself to meet your own needs as you program.

Those three functions of the keys are explained elsewhere. But how do you use them in your own programs? Having to check for an entire message will bog your program down. Fortunately the answer is easy. We can reprogram the function keys to match the function keys on the Commodore 64 and VIC-20. (This also makes program conversions from those machines easier).

To reprogram the keys, put the following line into your program:

```
10 FOR K = 1 TO 8:KEYK, CHR$(K + 132): NEXT
```

That's all there is to it. Now whenever you type a function key, it sends a non-printing character, from 133 to 140, just like the Commodore 64. To check for this in a program, you can use this method;

```
20 GETKEY$:IFASC(A$)= 133THENPRINT"FUNCTION KEY 1 HIT":GOTO20
30 IFASC(A$)>133ANDASC(A$)<141THENPRINT"SOME OTHER FUNCTION
KEY HIT"
40 GOTO 20
```

After your program is done, you will have to redefine the keys again if you want them to say directory, dload, etc. You can do this by hand, in a program, or by resetting the C-264.

## **APPENDICES**

## APPENDIX A: ERROR MESSAGES

These error messages are printed by BASIC. You can also PRINT the messages through the use of the ERR\$( ) function.

### ERROR # ERROR NAME

1	TOO MANY FILES	:There is a limit of 10 files OPEN at one time.
2	FILE OPEN	:An attempt was made to open a file using the :number of an already open file.
3	FILE NOT OPEN	:The file number specified in an I/O statement must :be opened before use.
4	FILE NOT FOUND	:Either no file with that name exists (disk) or an end- :of-tape marker was read (tape).
5	DEVICE NOT PRESENT	:The required I/O device was not available.
6	NOT INPUT FILE	:An attempt was made to GET or INPUT data from :a file that was specified as output only.
7	NOT OUTPUT FILE	:An attempt was made to send data to a file :that was specified as input only.
8	MISSING FILE NAME	:An OPEN, LOAD, or SAVE to the disk drive :generally requires a file name.
9	ILLEGAL DEVICE NUMBER	:An attempt was made to use a device improperly :(SAVE to the screen, etc.)
10	NEXT WITHOUT FOR	:Either loops are nested incorrectly, or there :is a variable name in a next statement that :doesn't correspond with one in a FOR.
11	SYNTAX	:A statement is unrecognizable by BASIC. This :could be because of missing or extra :parenthesis, misspelled keyword, etc.
12	RETURN WITHOUT GOSUB	:A RETURN statement was encountered when no :GOSUB statement was active.
13	OUT OF DATA	:A READ statement was encountered, but there :is no data left unREAD.
14	ILLEGAL QUANTITY	:A number used as the argument of a function :or statement is outside the allowable range.
15	OVERFLOW	:The result of a computation is larger than the :largest number allowed (1.701411833E + 38)
16	OUT OF MEMORY	:Either there is no more room for program and :program variables, or there are too many DO, :FOR, or GOSUB statements in effect.
17	UNDEF'D STATEMENT	:A line number was referenced that doesn't exist :in the program.
18	BAD SUBSCRIPT	:The program was trying to reference an element of :an array out of the range specified by the :DIM statement.
19	REDIM'D ARRAY	:An array can only be DIMensioned once. If an :array is referenced before that array is DIM'd, an :automatic DIM (to 10) is performed.

20	DIVISION BY ZERO	:Division by zero is mathematically not allowed.
21	ILLEGAL DIRECT	:INPUT or GET statements are only allowed within a program.
22	TYPE MISMATCH	:This occurs when a number is used in place of a string or vice-versa.
23	STRING TOO LONG	:A string can contain up to 255 characters.
24	FILE DATA	:Bad data was read from a tape file.
25	FORMULA TOO COMPLEX	:Simplify the expression (break into 2 parts or use fewer parentheses).
26	CAN'T CONTINUE	:The CONT command will not work if the program was never RUN, there was an error, or a line has been edited.
27	UNDEF'D FUNCTION	:A user defined function was referenced that was never defined.
28	VERIFY	:The program on tape or disk does not match the program in memory.
29	LOAD	:There was a problem loading. Try again.
30	BREAK	:The stop key was hit to halt program execution.
31	CAN'T RESUME	:A RESUME statement was encountered when a TRAP statement is not in effect.
32	LOOP NOT FOUND	:The program has encountered a DO statement and cannot find the corresponding LOOP.
33	LOOP WITHOUT DO	:The LOOP was encountered when there was no DO statement active.
34	DIRECT MODE ONLY	:This command is allowed only in direct mode, not from a program.
35	NO GRAPHICS AREA	:A graphics command (DRAW, BOX, etc.) was encountered before the GRAPHIC command was executed.
36	BAD DISK	:An attempt failed to HEADER a diskette, either because the quick header method (no ID) was attempted on an unformatted diskette, or the diskette is bad.

## DESCRIPTION OF DOS ERROR MESSAGES

These error messages are returned through the DS and DS\$ reserved variables.

**NOTE:** Error message numbers less than 20 should be ignored with the exception of 01, which gives information about the number of files scratched with the SCRATCH command.

- 20: **READ ERROR (block header not found)**  
The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.
- 21: **READ ERROR (no sync character)**  
The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment of the read/writer head, no diskette is present, or unformatted or improperly seated diskette. Can also indicate a hardware failure.
- 22: **READ ERROR (data block not present)**  
The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or sector request.
- 23: **READ ERROR (checksum error in data block)**  
This error message indicates that there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.
- 24: **READ ERROR (byte decoding error)**  
The data or header has been read into the DOS memory, but a hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate grounding problems.
- 25: **WRITE ERROR (write-verify error)**  
This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.
- 26: **WRITE PROTECT ON**  
This message is generated when the controller has been requested to write a data block while the write protect switch is depressed. Typically, this is caused by using a diskette with a write a protect tab over the notch.
- 27: **READ ERROR (checksum error in header)**  
The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.
- 28: **WRITE ERROR (long data block)**  
The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear within a pre-determined time, the error message is generated. The error is caused by a bad diskette format (the data extends into the next block), or by hardware failure.
- 29: **DISK ID MISMATCH**  
This message is generated when the controller has been requested to access a diskette which has not been initialized. The message can also occur if a diskette has a bad header.
- 30: **SYNTAX ERROR (general syntax)**  
The DOS cannot interpret the command sent to the command channel. Typically, this is caused by an illegal number of file names, or patterns are illegally used. For example, two file names may appear on the left side of the COPY command.
- 31: **SYNTAX ERROR (invalid command)**  
The DOS does not recognize the command. The command must start in the first position.

- 32: SYNTAX ERROR (invalid command)  
The command sent is longer than 58 characters.
- 33: SYNTAX ERROR (invalid file name)  
Pattern matching is invalidly used in the OPEN or SAVE command.
- 34: SYNTAX ERROR (no file given)  
The file name was left out of a command or the DOS does not recognize it as such. Typically, a colon (:) has been left out of the command.
- 39: SYNTAX ERROR (invalid command)  
This error may result if the command sent to command channel (secondary address 15) is unrecognized by the DOS.
- 50: RECORD NOT PRESENT  
Result of disk reading past the last record through INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning.
- 51: OVERFLOW IN RECORD  
PRINT# statement exceeds record boundary. Information is truncated. Since the carriage return which is sent as a record terminator is counted in the record size, this message will occur if the total characters in the record (including the final carriage return) exceeds the defined size.
- 52: FILE TOO LARGE  
Record position within a relative file indicates that disk overflow will result.
- 60: WRITE FILE OPEN  
This message is generated when a write file that has not been closed is being opened for reading.
- 61: FILE NOT OPEN  
This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored.
- 62: FILE NOT FOUND  
The requested file does not exist on the indicated drive.
- 63: FILE EXISTS  
The file name of the file being created already exists on the diskette.
- 64: FILE TYPE MISMATCH  
The file type does not match the file type in the directory entry for the requested file.
- 65: NO BLOCK  
This message occurs in conjunction with the B-A command. It indicates that the block to be allocated has been previously allocated. The parameters indicate the track and sector available with the next highest number. If the parameters are zero (0), then all blocks higher in number are in use.
- 66: ILLEGAL TRACK AND SECTOR  
The DOS has attempted to access a track or block which does not exist in the format being used. This may indicate a problem reading the pointer to the next block.
- 67: ILLEGAL SYSTEM T OR S  
This special error message indicates an illegal system track or sector.
- 70: NO CHANNEL (available)  
The requested channel is not available, or all channels are in use. A maximum of five sequential files may be opened at one time to the DOS. Direct access channels may have six opened files.

71: **DIRECTORY ERROR**

The BAM does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem reinitialize the diskette to restore the BAM in memory. Some active files may be terminated by the corrective action. NOTE: BAM = Block Availability Map

72: **DISK FULL**

Either the blocks on the diskette are used or the directory is at its entry limit. DISK FULL is sent when two blocks are available on the 1541 to allow the current file to be closed.

73: **DOS MISMATCH (73, CBM DOS V2.6 1541)**

DOS 1 and 2 are read compatible but not write compatible. Disks may be interchangeably read with either DOS, but a disk formatted on one version cannot be written upon with the other version because the format is different. This error is displayed whenever an attempt is made to write upon a disk which has been formatted in a non-compatible format. (A utility routine is available to assist in converting from one format to another.) This message may also appear after power up.

74: **DRIVE NOT READY**

An attempt has been made to access the Floppy Disk Drive without any diskette present.



## APPENDIX B: BASIC 3.5 COMMANDS, STATEMENTS, AND FUNCTIONS

This manual has given you an introduction to the BASIC language, to give you a feel for computer programming and some of the vocabulary involved. This appendix gives a complete list of the rules (SYNTAX) of the BASIC 3.5 language, along with a concise description of each. You are encouraged to experiment with these commands, remembering that you can't do any permanent damage to the C-264 just by typing in programs, and that the best way to learn computing is by doing.

This appendix provides formats and brief explanations and examples of the BASIC 3.5 commands and statements. It is not intended to teach BASIC. Appendix M lists tutorial books that help you learn BASIC.

This appendix lists commands and statements in separate sections. Within the sections, the commands and statements are listed in alphabetical order. In most cases, commands can be used as statements in a program if you prefix them with a line number. You can use many statements as commands by issuing them in direct mode (i.e., without line numbers).

This appendix is divided into sections according to the different types of operations in BASIC. These include:

1. Variables and Operators: describes the different types of variables, legal variable names, and arithmetic and logical operators.
2. Commands: describes the commands used to work with programs, edit, store, and erase them.
3. Statements: describes the BASIC program statements used in numbered lines of programs.
4. Functions: describes the string, numeric, and print functions.

The commands in each section are listed alphabetically for convenience. A fuller explanation of C-264 BASIC commands is provided in the C-264 Programmer's Reference Guide, available from your Commodore dealer or your local bookstore.

### 1. VARIABLES & OPERATORS

#### a. VARIABLES

The C-264 uses three types of variables in BASIC. These are: normal numeric, integer numeric, and string (alphanumeric) variables.

Normal numeric variables, also called floating point variables, can have any value from (superscript) - 10 to (superscript) + 10, with up to nine digits of accuracy. When a number becomes larger than nine digits will show, as in  $10^{10}$  or  $10^{-10}$ , the computer will display it in scientific notation form, with the number normalized to 1 digit and eight decimal places, followed by the letter E and the power of ten by which the number is multiplied. For example, the number 12345678901 will be displayed as 1.23456789E + 10.

Integer variables are used when the number will always be from + 32767 to - 32768, and without fractional parts. Integer variables require less memory space than floating point variables, but the difference probably would not be substantial unless used in a large quantity such as an array (see below). An integer variable would be a number like 5, 10, or - 100.

String variables are those used for character data, which may contain numbers, letters, and any other character that the C-264 can make. An example of a string variable is "C-264".

## VARIABLE NAMES

Variable names may consist of a single letter, a letter followed by a number, or two letters. Variable names may be longer than 2 characters, but only the first two are significant.

An integer variable is specified by using the percent (%) sign after the variable name. String variables have the dollar sign (\$) after their names.

### EXAMPLES:

Numeric Variable Names: A, A5, BZ

Integer Variable Names: A%, A5%, BZ%

String Variable Names: A\$, A5\$, BZ\$

Arrays are lists of variables with the same name, using an extra number to specify the elements of the array. They are defined using the DIM statement, and may contain floating point, integer, or string variables. The array variable name is followed by a set of parentheses ( ) enclosing the number of the variables in the list.

EXAMPLES: A(7),BZ%(11),A\$(87)

Arrays may have more than one dimension. A two dimensional array may be viewed as having rows and columns, with the first number identifying the row and the second number in the parentheses identifying the column.

EXAMPLES: A(7,2),BZ%(2,3,4),Z\$(3,2)

## RESERVED VARIABLE NAMES

There are seven variable names which are reserved for use by the C-264, and may not be used for another purpose. These are the variables DS, DS\$, ER, EL, ST, TI, and TI\$. It is also illegal to use TO, IF, and any names that contain command names within them, such as SRUN, RNEW, or XLOAD.

ST is a status variable for input and output (except normal screen/keyboard operations). The value of ST depends on the results of the last input/output operation. A more detailed explanation of ST is in the C-264 Programmer's Reference Guide, but in general, if the value of ST is 0 the operation was successful.

TI and TI\$ are variables that relate to the real-time clock built into the C-264. The system clock is updated every 1/60th of a second. It starts at 0 when the C-264 is turned on, and is reset only by changing the value of TI\$. The variable TI gives you the current value of the clock in 1/60ths of a seconds.

TI\$ is a string that reads the value of the real-time clock as a 24 hours clock. The first two characters of TI\$ contain the hour, the 3rd and 4th characters are the minutes, and the 5th and 6th characters are the seconds. This variable can be set to any value (so long as all characters are numbers), and will be automatically updated as a 24 hour clock.

EXAMPLE: TI\$ = "101530" sets the clock to 10:15 and 30 seconds (AM)

The value of the clock is lost when the C-264 is turned off. It starts at zero when the C-264 is turned back on, or when the value of the clock exceeds 235959.

The variable DS reads the disk drive command channel, and returns the current status of the drive. To get this information in words, PRINT DS\$. These status variables are used after a disk operation, like a DLOAD or DSAVE, to find out why the red error light on the disk drive is blinking.

ER, EL, and ERR\$ are variables used in error trapping routines. They are usually only useful within a program. ER returns the last error encountered since the program was RUN. EL is the line where the error occurred. ERR\$ is a function which allows your program to print one of the BASIC error messages. PRINT ERR\$(ER) will print out the proper error message.

ER = 1 2 no error

## CONVENTIONS IN FORMATS

The following conventions are used in the formats of the BASIC commands and statements:

- **KEYWORDS**, also called **RESERVED WORDS**, appear in uppercase letters. **YOU MUST ENTER THESE KEYWORDS EXACTLY AS THEY APPEAR**. However, many keywords have abbreviations that you can also use (see Appendix C).  
Keywords are words that are part of the BASIC language that your computer knows. Keywords are the central part of a command or statement. They tell the computer what kind of action you want it to take. These words cannot be used as part of variable names.
- **ARGUMENTS**, (also called parameters), appear in lowercase letters. Arguments are the parts of a command or statement that you select; they complement keywords by providing specific information about the command or statement. For example, a keyword tells the computer to load a program, while an argument tells the computer which specific program to load and in which drive the disk containing the program is located. Arguments include filenames, variables, line numbers, etc.
- **SQUARE BRACKETS [ ]** show **OPTIONAL** arguments. You select any or none of the arguments listed, depending on your requirements.
- **ANGLE BRACKETS ( < > )** indicate that you **MUST** choose one of the arguments listed.
- **VERTICAL BAR ( | )** separates items in a list of arguments when your choices are limited to those arguments listed, and you can't use any other arguments. When the vertical bar appears in a list enclosed in **SQUARE BRACKETS**, your choices are limited to the items in the list, but you still have the option not to use any arguments.
- **ELLIPSIS ( . . . )**, a sequence of three dots, means that an option or argument can be repeated more than once.
- **QUOTATION MARKS ( " " )** enclose character strings, filenames, and other expressions. When arguments are enclosed in quotation marks in a format, you must include the quotation marks in your command or statement. Quotation marks are not conventions used to describe formats; they are required parts of a command or statement.
- **PARENTHESES**. When arguments are enclosed in parentheses in a format, you must include the parentheses in your command or statement. Parentheses are not conventions used to describe formats; they are **REQUIRED** parts of a command or statement.
- **VARIABLE** means any valid BASIC variable name, such as X, A\$, or T%.
- **EXPRESSION** means any valid BASIC expression, such as A + B + 2 or .5\*(X + 3).

## BASIC OPERATORS

The arithmetic operators include the following signs:

- + addition
- subtraction
- \* multiplication
- / division
- ↑ raising to a power (exponentiation)

On a line containing more than one operator, there is a set order in which operations always occur. If several operators are used together, the computer assigns priorities as follows: First, exponentiation. Next, multiplication and division, and last, addition and subtraction. If you want these operations to occur in a different order, C-264 BASIC allows you to give a calculation a higher priority by placing parentheses around it. Operations enclosed in parentheses will be calculated before any other operation. You have to make sure that your equations have the same number of left parentheses as right parentheses, or you will get a SYNTAX ERROR message when your program is run.

There are also operators for equalities and inequalities, called relational operators:

- = is equal to
- < is less than
- > is greater than
- <= or = > is less than or equal to
- >= or = > is greater than or equal to
- <> or = <> is not equal to

Finally, there are three logical operators:

AND  
OR  
NOT

These are used most often to join multiple formulas in IF . . . THEN statements. When they are used with arithmetic operators, they are evaluated last (i.e., after + and -). When used with arithmetic operators, they are evaluated with the lowest priority (last).

### EXAMPLES:

IF A = B AND C = D THEN 100	requires both A = B & C = D to be true.
IF A = B OR C = D THEN 100	allows either A = B or C = D to be true.
A = 5: B = 4: PRINT A = B	displays a value of 0
A = 5: B = 4: PRINT A > B	displays a value of - 1
PRINT 123 AND 15: PRINT 5 OR 7	displays 11 and 7

## GRAPHIC STATEMENT INFORMATION

There are a few concepts that apply to all of the bit map graphics statements. First is the concept of the Pixel Cursor (PC). The PC is similar to the cursor in text mode: it is the position where the next dot will be drawn. The PC however, is invisible. All drawing commands use the PC. In addition, the LOCATE command lets you reposition the PC without drawing anything.

Wherever you use X,Y coordinates in a drawing command, you can use RELATIVE coordinates based on the current value of the PC. Merely precede your coordinates with + or -. A plus sign before the X value moves the PC to the right. A minus sign before the X value moves the PC to the left. Likewise, a minus sign before the Y coordinate moves the PC up, while a plus sign moves the PC down. For example:

LOCATE + 100, - 25                      moves the PC right 100 pixels and up 25.

DRAW1, + 10, + 10 to 100, 100      draw a line 10 pixels right and 10 pixels  
below the current value of the PC to the absolute point 100, 100.

You can also specify a distance and angle relative to the current PC by separating the two parameters by a semicolon.

For example:

LOCATE 50;45                      will move the PC from its current location  
by a distance of 50 dots at an angle of 45 degrees.

## 2. BASIC COMMANDS

### AUTO

AUTO number

Turns on the automatic line numbering feature which eases the job of entering programs by typing the line numbers for you. As you enter each program line and press RETURN the next line number is printed on the screen, with the cursor in position to begin typing that line. AUTO with 0 or NO ARGUMENT turns off auto line numbering, as does RUN. This statement is executable only in direct mode.

EXAMPLE:

AUTO 10                      automatically numbers line in increments of ten  
AUTO 50                      automatically numbers line in increments of fifty  
AUTO                          turns OFF automatic line numbering

### BACKUP

BACKUP D drive # To D drive # (ON Unit#)

This command copies all the files on a diskette to another diskette. You can copy onto a new diskette without first using the HEADER command to format the new diskette because BACKUP also formats diskettes. You should always backup disks in case the original is lost or damaged.

This command can only be used with dual disk drives.

NOTE: Because the BACKUP command also headers diskettes, it destroys any information already stored on the diskette onto which you are copying information. Therefore, be careful when you use this command. If you're copying onto an old diskette, make sure it doesn't contain any programs you wish to keep. See also the COPY command.

Examples:

BACKUP D0 TO D1                      Copies all the files from the disk in drive 0 to the disk in drive 1.  
BACKUP D0 TO D1, ON U9      Copies all files from drive 0 to drive 1 in disk drive unit 9.

NOTE: After you issue a BACKUP command, the computer asks ARE YOU SURE? Type Y and press RETURN to execute the BACKUP. Press any other key and RETURN to cancel this command.

## CONT

(Continue)

This command is used to re-start the execution of a program that has been stopped by either using the STOP key, a STOP statement, or an END statement within the program. The program will resume execution where it left off. CONT will not work if you have changed or added lines of the program (or even just moved the cursor to a program line and hit RETURN without changing anything), if the program stopped due to an error, or if you caused an error before trying to re-start the program. The error message in this case is CAN'T CONTINUE ERROR.

## DELETE

DELETE — line number —

Deletes lines of BASIC text. This command is executable only in DIRECT mode.

### EXAMPLES:

DELETE 75                      Deletes line 75.


DELETE 10 — 50              Deletes lines 10 through 50 inclusive.

DELETE — 50                 Deletes all lines from the beginning of the program up to and including line 50.

DELETE 75—                 Deletes all lines from 75 on to the end of the program.

## DIRECTORY

DIRECTORY [Ddrive#[,Uunit#[,“pattem”]]]

Displays a disk directory on the C-264 screen. Use CTRL-S to pause the display. Use the  key (the Commodore key) to slow it down. The DIRECTORY command cannot be used to print a hard copy. You must load the disk directory, destroying the program currently in memory, to do that.

### EXAMPLES:

DIRECTORY                    List all files on the disk.

DIRECTORY U9,“AB\*”         List all files on disk drive unit 9 (8 is the default) starting with the letters AB.

DIRECTORY D0,  
“FILE?.BAK”                 The ? is a wildcard that matches any single character in that character position.

### NOTE: TO PRINT A COPY ON PAPER USE THE FOLLOWING:

LOAD“\$0”,8:OPEN4,4:CMD4:LIST:PRINT#4:CLOSE 4

## DLOAD

DLOAD “filename” [,Ddrive#[,Uunit#]]

This command loads a program from disk into current memory. (Use LOAD to load programs on tape.) You must supply a program name.

### EXAMPLES:

DLOAD “BANKRECS”           Searches the disk for the program BANKRECS and LOADs it.

DLOAD (A\$)                 LOADs a program from disk whose name is in the variable A\$. You will get an error if A\$ is empty.

The DLOAD command can be used within a BASIC program to find and RUN another program on disk. This is called chaining.

## DSAVE

DSAVE "filename" [,Ddrive#[,Unit#]]

This command stores a program on disk. (Use SAVE to store programs on tape.) You must supply a program name.

### EXAMPLES:

DSAVE "BANKRECS"      SAVes the program BANKRECS to disk.

DSAVE (A\$)            SAVes to disk a program whose name is in the variable A\$.

DSAVE "PROG 3",DO,U9    Saves the program PROG 3 to the disk drive with a unit number (Serial bus address) of 9.

## HEADER

HEADER "diskname" [,Id code [,Ddrive#[,ON Unit#]]]

Before you can use a diskette for the first time you must format it with the HEADER command. If you want to erase an entire diskette for reuse, you can use the HEADER command. This command divides the disk into sections called blocks, and it creates a table of contents, called a directory or catalog, on the disk. The diskname can be any name up to 16 characters long. The id number is any 2 characters. Give each disk a unique id number. Be careful when you HEADER a disk because the HEADER command erases all stored data. Giving no ID number allows you to perform a quick header. The old id will be used. You can only use the quick header method if the disk was previously formatted, since the quick header only clears out the directory rather than formatting the disk.

### EXAMPLES:

HEADER "MYDISK",I23

HEADER "MYDISK" D9, I23

HEADER "RECS",IR5,U8,D1

## HELP

The HELP command is used after you get an error in your program. When you type HELP, the line where the error occurred is listed, with the portion containing the error displayed in reverse video.

## KEY

KEY [key no., string]

There are eight function keys available to the user on the C-264 computer: four unshifted and four shifted. C-264 allows you to define what each key will do when it is pressed.

KEY without any parameter specified will give a listing displaying all the KEY assignments. The data you assign to a key will be typed out when that function key is pressed. The maximum length for all the definitions together is 128 characters. Entire commands or a series of commands can be assigned to a key. For example:

KEY 7, "GRAPHIC0" + CHR\$(13) + "LIST" + CHR\$(13)

will cause the computer to select text mode and list your program whenever the 'F7' key is depressed (in direct mode). The CHR\$(13) is the ASCII character for RETURN. Use CHR\$(34) to incorporate a double quote into a KEY string.

The keys may be redefined in a program. For example:

10 KEY2,"TESTING" + CHR\$(34);KEY3,"NO"

10 FORK = 1 TO 8:KEY K, CHR\$(K + 132):NEXT    define the function keys like the Commodore 64 and VIC-20.

To restore all function keys to their default values, reset the C-264 by turning it off and on, or press the RESET button.

## LIST

LIST [1st line] [[last line]]

The LIST command lets you look at lines of a BASIC program that have been typed or LOADED into the C-264's memory. When used alone without any numbers following it, you will see a complete listing of the program on your screen (which may be slowed down by holding down the **[C]** key, paused by CTRL — S, or STOPped by hitting the key marked RUN STOP). If you follow the word LIST with a line number, the C-264 will only show you that line number. If you type LIST with 2 numbers separated by a dash, the C-264 will show all lines from the first to the second line number. If you type LIST followed by a number and just a dash, it will show all the lines from that number to the end of the program. And if you type LIST, a dash, and then a number, you will get all the lines from the beginning to that line number. Using these variations, you can examine any portion of a program, or bring lines to the screen for modification.

### EXAMPLES:

LIST	Shows entire program.
LIST 100—	Shows only from line 100 until the end.
LIST 10	Shows only line 10.
LIST—100	Shows lines from the beginning until line 100.
LIST 10—200	Shows lines from 10 to 200, inclusive.

## LOAD

LOAD ["filename"[,device#[,relocate flag]]]

This is the command to use when you want to use a program stored on cassette tape or on disk. If you type just LOAD and hit the RETURN key, the C-264 will blank the screen. You can rewind the tape at this point if you need to. Then press play. The C-264 will now start looking for a program on the tape. When it finds one, the C-264 will print FOUND filename. You can hit the **[C]** key to load, or the spacebar to keep searching on the tape. Once the program is loaded, you can RUN, LIST, or change it.

You can also type the word LOAD followed by a program name (which is most often a name in quotes (" ")). The name may be followed by a comma (outside of the quotes) and a number (or numeric variable) which acts as a device number to determine where the program is stored (disk or tape). If there is no number given, the C-264 assumes device 1, which is the cassette tape recorder.

The other device commonly used with the LOAD command is usually the disk drive, which is device #8.

### EXAMPLES:

LOAD	Reads in the next program on tape.
LOAD "HELLO"	Searches tape for a program called HELLO, and LOADS if found.
LOAD A\$	Looks for a program whose name is in the variable A\$.
LOAD "HELLO",8	Looks for the program called HELLO on the disk drive.

The LOAD command can be used within a BASIC program to find and RUN another program on disk or tape. This is called chaining.

A RELOCATE FLAG of 0 tells the C-264 whether to load the program at the start of the BASIC program area and a flag of 1 tells where it was SAVED from. This is generally only important to a machine language program.



## NEW

### NEW

This command erases the entire program in memory and clears out any variables that may have been used. Unless the program was stored somewhere, it is lost until you type it in again. BE CAREFUL when you use this command!

The NEW command can also be used as a statement in a BASIC program. When the C-264 gets to this line, the program is erased and everything stops. This is not especially useful under normal circumstances.

## RENUMBER

RENUMBER new starting line, increment, old starting line

The new starting line is the number of the first line in the program after renumbering. It defaults to 10.

The increment is the spacing between line numbers, i.e. 10, 20, 30 etc. It also defaults to 10.

The old starting line number is the line number in the program where renumbering is to begin. This allows you to renumber a portion of your program. It defaults to the first line of your program.

This command can only be executed from direct mode.

### EXAMPLES:

RENUMBER 20, 20, 1      Starting at line 1, renumbers the program. Line 1 becomes line 20, and other lines are numbered in increments of 20.

RENUMBER, , 65      Starting at line 65, renumbers in increments of 10. Line 65 becomes line 10.

Before you issue a RENUMBER command, be sure all line numbers are defined, the range of numbers is valid, and there is enough memory. It's a good idea to save your program before RENUMBERing in case an error occurs.

## RUN

RUN [line number]

Once a program has been typed into memory of LOADED, the RUN command makes it start working. RUN clears all variables in the program before starting program execution. If there is no number following the command RUN, the computer will start with the lowest numbered program line. If there is a number following the RUN command execution starts at that line. RUN may be used within a program.

### EXAMPLES:

RUN      Starts program working from lowest line number.

RUN 100      Starts program at line 100.

*increment > 1  
checks new numbers, ascends  
retroactively*

## SAVE

SAVE "filename" [,device# [,EOT Flag]]

This command will store a program currently in memory on a cassette tape or disk. If you just type the word SAVE and hit RETURN, the C-264 will attempt to store the program on cassette tape. It has no way of checking if there is already a program at that spot, so be careful with your tapes. If you type the SAVE command followed by a name in quotes or a string variable name, the C-264 will give the program that name, so it may be more easily located and retrieved in the future. If you want to specify a device number for the SAVE, follow the name by a comma (after the quotes) and a number or numeric variable. Device number 1 is the tape drive, and 8 is the disk. After the number on a tape command, there can be a comma and a second number, which is either 0 or 1. If the second number is 1, the C-264 will put an END-OF-TAPE marker after your program. If you are trying to LOAD a program and the C-264 finds one of these markers, you will get a FILE NOT FOUND ERROR.

### EXAMPLE:

SAVE	Stores program to tape without a name.
SAVE "HELLO",8	Stores on disk with the name HELLO.
SAVE A\$	Stores on tape with name in variable A\$
SAVE "0'HELLO",8	Stores on disk with name HELLO
SAVE "HELLO",1,2	Stores on tape with name HELLO and follows and END- OF- TAPE marker after the program.

## SCRATCH

SCRATCH "file name" ,Ddrive number ,Uunit number

Deletes a file from the disk directory. As a precaution, you will be asked "Are you sure" before the C-264 completes the operation. Type a Y to do the SCRATCH. Type an N to cancel the operation.

## VERIFY

VERIFY "filename" ,device# ,relocate flag

This command causes the C-264 to check the program on tape or disk against the one in memory. This is proof that the program you just SAVED is really saved, in case your tape is bad or something isn't working. This command is also very useful for positioning a tape so that C-264 will write after the last program on the tape. All you do is tell the C-264 to VERIFY the name of the last program on the tape. It will do so, and tell you that the programs don't match (which you already knew). Now the tape is where you want it, and you can store the next program without any fear of erasing an old one.

VERIFY without anything after the command causes the C-264 to check the next program on tape, regardless of its name, against the program now in memory. VERIFY followed by a program name (in quotes) or a string variable will search the tape for that program and then check. VERIFY followed by a name and a comma and a number will check the program on the device with that number (1 for tape, 8 for disk). The relocate flag is the same as in the LOAD command.

### EXAMPLE:

VERIFY	Checks the next program on the tape.
VERIFY "HELLO"	Searches for HELLO, checks against memory.
VERIFY "HELLO",8,1	Searches for HELLO on disk, then checks.

### 3. BASIC STATEMENTS

#### BOX

BOX[cs,] a1,b1 , a2,b2 [, angle] [,paint]

cs ..... Color source (0-3)(default is 1, foreground)  
a1,b1 ..... Corner coordinate (scaled)  
a2,b2 ..... Corner opposite a1,b1 (scaled) (default is the PC)  
angle ..... Rotation in clockwise degrees (default is 0 degrees)  
paint (0 or 1) ..... Paint shape with color (default is 0, no painting)

This command allows you to draw a rectangle of any size anywhere on the screen. Rotation will be about the center of the rectangle. The Pixel Cursor (PC) is left at a2,b2 after the BOX statement is executed.

#### EXAMPLES:

BOX 1, 10,10, 60,60                Draws the outline of a rectangle.  
BOX , 10,10, 60,60, 45,1        Draws a filled, rotated box (i.e., a diamond).  
BOX , 30,90,,45,1                Draws a filled, rotated polygon.

#### CHAR

CHAR [color source#],x,y ,string[,reverse flag]

color source ..... 0 - 3 (default is 1)  
x ..... Character column (0 - 39)  
y ..... Character row (0 - 24)  
string ..... String to print  
reverse ..... Reverse field flag (0 = off, 1 = on)

Text (alphanumeric strings) can be displayed on the screen at a given location by the CHAR command. Character data is read from C-264 character ROM area. You supply the x and y coordinates of the starting position and the text string you want to display. Color source and reverse imaging are optional.

The string is continued on the next line if it attempts to print past the right edge of the screen. When used in TEXT mode, the string printed by the CHAR command works just like a PRINT string, including reverse field, cursors, flash on/off, etc. These control functions inside the string do not work when the CHAR command is used to display text in GRAPHIC mode.

**NOTE:** To display foreground color in multi-color mode 1 background, use color source = 0 and reverse flag = 0. To display foreground in multi-color 2, use color source = 0 and reverse flag = 1.

## CIRCLE

CIRCLE [cs][,a,b,],xr[,yr][,sa[,ea[,angle[,inc]]]]

cs	Color source (0 - 3)(default is 1)
a,b	Center coordinate (scaled) (defaults to the Pixel Cursor PC)
xr	X radius (scaled)
yr	Y radius (default is xr)
sa	Starting arc angle (default 0)
ea	Ending arc angle (default 360)
angle	Rotation in clockwise degrees (default is 0 degrees)
inc	Degrees between segments (default is 2 degrees)

With the CIRCLE command you can draw a circle, ellipse, arc, triangle or an octagon. The final coordinate will be on the circumference of the circle at the ending arc angle. Any rotation will be about the center. Note that setting the Y radius equal to the X radius will not draw a circle since the X and Y coordinates are scaled differently. Arcs are drawn from the starting angle clockwise to the ending angle. The segment increment controls the coarseness of the shape and hence the speed at which it is drawn. The increment value should be chosen so that it is a modulus of the arc length.

### EXAMPLES:

CIRCLE , 160,100,65,10	Draws an ellipse.
CIRCLE , 160,100,65,50	Draws a circle.
CIRCLE , 60,40,20,18,,,,45	Draws an octagon.
CIRCLE , 260,40,20,,,,,90	Draws a diamond.
CIRCLE , 60,140,20,18,,,,,120	Draws a triangle.

## CLOSE

CLOSE file number

This command completes and closes any files used by OPEN statements. The number following the word CLOSE is the file number to be closed.

### EXAMPLE:

CLOSE 2    Logical file 2 is closed.

## CLR

CLR

This command will erase any variables in memory, but leaves the program itself intact. This command is automatically executed when a RUN or NEW command is given, or when any editing is performed.

*CLEARs reserved variables, too*

## CMD

CMD filename[,print list]

CMD sends the output which normally would go to the screen (i.e., PRINT statement, LISTS, but not POKEs into the screen) to another device instead. This could be a printer, or a data file on tape or disk. This device or file must be OPENed first. The CMD command must be followed by a number or numeric variable referring to the file.

### EXAMPLES:

OPEN 1,4	OPEN device #4, which is the printer.
CMD 1	All normal output now goes to the printer.
LIST	The LISTing goes to the printer, not the screen — even the word READY.
PRINT#1	Set output back to the screen.
CLOSE 1	Close the file.

## COLLECT

COLLECT [Ddrive#][,U unit#]

Use this command to free up space allocated to improperly closed files and delete references to these files from the directory.

EXAMPLE:

COLLECT D0

## COLOR

COLOR source#, color#, luminance#

Assigns a color to one of the 5 color sources:

Number	Source
0	background
1	foreground
2	multicolor 1
3	multicolor 2
4	border

Colors you can use are in the range 1 - 16 (BLACK, WHITE . . .). As an option, you can include the luminance level 0 - 7, with 0 being lowest and 7 being highest. Luminance defaults to 7.

## COPY

COPY [Ddrive#],[ "source file" TO [Ddrive#],[ "other file" ][,Uunit#]

COPYs a file on the disk in one drive to the disk in the other on a dual disk drive only, or creates a copy of a file on the same drive (with a different file name).

COPY D0, "TEST" to D1, "TEST PROG"

Copies TEST from drive 0 to drive 1, naming it TEST PROG only on drive 1.

COPY D0, "STUFF" to D1, "STUFF"

Copies STUFF from drive 0 to drive 1

COPY D0 to D1

Copies all files on drive 0 to drive 1.

COPY "WORK.PROG" TO "BACKUP"

Copies WORK.PROG. as a program called BACKUP on the same drive.

## DATA

DATA list of constants

This statement is followed by a list of items to be used by READ statements. The items may be numbers or words, and are separated by commas. Words need not be inside of quotes unless they contain any of the following characters: SPACE, colon, or comma. If two commas have nothing between them, the value will be READ as a zero for a number, or an empty string. Also see the RESTORE statement, which allows the C-264 to reread data.

EXAMPLE:

DATA 100,200,FRED,"HELLO,MOM",,3,14,ABC123

## DEF FN (Define Function)

DEF FN name (variable) = expression

This command allows you to define a complex calculation as a function. In the case of a long formula that is used several times within a program, this can save a lot of space.

The name you give the function will be the letters FN and any legal numeric variable name. First you must define the function by using the statement DEF followed by the name you have given the function. Following the name is a set of parentheses ( ) with a numeric variable (in this case X) enclosed. Then you have an equal sign, followed by the formula you want to define. You can "call" the formula, substituting any number for X, using the format shown in line 20 of the example below:

### EXAMPLE:

```
10 DEF FNA(X) = 12*(34.75 - X/.3) + X
```

```
20 PRINT FNA(7)
```

The number 7 is inserted each place  
X is located in the formula given in the DEF  
statement.

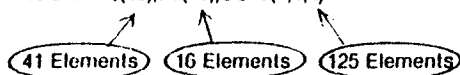
## DIM

DIM variable (subscripts) [, variable(subscripts) . . .]

Before you can use an array of variables, unless you want 11 or fewer elements in the array, the program must first execute a DIM statement for that array. The statement DIM is followed by the name of the array, which may be any legal variable name. Then, enclosed in parentheses, you put the number (or numeric variable) of elements in each dimension. An array with more than one dimension is called a matrix. You may use any number of dimensions, but keep in mind that the whole list of variables you are creating takes up lots of room, and it is easy to run out of memory if you get carried away. To figure the number of variables created with each DIM, multiply the total number of elements in each dimension of the array. (Each array starts with element 0.)

### EXAMPLE:

```
10 DIM A$(40),B7(15),CC%(4,4,4)
```



You can dimension more than one array in a DIM statement by separating the arrays by commas. Be careful not to let the program execute a DIM statement for any array more than once, or you'll get an error message. It is good programming practice to place DIM statements near the beginning of the program.

## **DO/LOOP/WHILE/UNTIL/EXIT**

**DO** [UNTIL boolean argument | WHILE boolean argument]  
[Statements]  
**[EXIT]**

**LOOP** [UNTIL boolean argument | WHILE boolean argument]

Performs the statements between the DO statement and the LOOP statement. If no UNTIL or WHILE modifies either the DO or the LOOP statement, execution of the intervening statements continues indefinitely. If an EXIT statement is encountered in the body of a DO loop, execution is transferred to the first statement following the LOOP statement. DO loops may be nested, following the rules defined for FOR-NEXT loops.

If the UNTIL parameter is used, the program will continue looping until the boolean argument is satisfied (becomes TRUE). The WHILE parameter is basically the opposite of the UNTIL parameter: the program continues looping as long as the boolean argument is TRUE. An example of a boolean argument is  $A = 1$  or  $G > = 65$ .

**EXAMPLE:**

```
DO UNTIL X = 0 OR X = 1
```

```
  :
```

```
    LOOP
```

```
DO WHILE A$ = " ":GET A$:LOOP
```

## **DRAW**

**DRAW** [color source#], a1, b1] [TO a2, b2 . . .]

With this command you can draw individual dots, lines, and shapes. You supply color source (0-3), starting (a1, b1) and ending points (a2, b2).

**EXAMPLES:**

a dot:	DRAW 1, 100, 50
lines:	DRAW , 10,10 TO 100,60
	DRAW , TO 25,30
a shape:	DRAW, 10,10 TO 10,60 TO 100,60 TO 10,10

## **END**

When the program finds an END statement, the program stops RUNNING immediately. The CONT command can continue the program at the statement following the END statement.

## FOR ... TO ... STEP

FOR variable = start value TO end value [STEP increment]

This statement works with the NEXT statement to set up a section of the program that repeats for a set number of times.

The loop variable is a variable which will be added to or subtracted from during the program. The start of count and end of count are the limits to the value of the loop variable.

The logic of the FOR statement is as follows. First, the loop variable is set to the start of count value. The end of count value is saved for later reference by the C-264. When the program reaches a line with the command NEXT, it adds the STEP increment (default = 1) to the value of the loop variable and checks to see if it is higher than the end of loop value. If it is not higher, the next line executed is the statement immediately following the FOR statement. If the loop variable is larger than the end of loop number, then the next statement executed will be the one following the NEXT statement. Also see the NEXT statement.

### EXAMPLE:

```
10 FOR L = 1 TO 10
20 PRINT L
30 NEXT L
40 PRINT "I'M DONE! L = "L
```

This program will print the numbers from one to ten on the screen, followed by the message I'M DONE! L = 11.

The end of loop value may be followed by the word STEP and another number or variable. In this case, the value following the STEP is added each time instead of one. This allows you to count backwards, by fractions, or any way necessary.

You can set up loops inside one another. This is known as nesting loops. You must be careful to nest loops so that the last loop to start is the first one to end.

### EXAMPLE OF NESTED LOOPS:

```
10 FOR L = 1 TO 100
20 FOR A = 5 TO 11 STEP 2
30 NEXT A
40 NEXT L
```

This FOR ... NEXT loop is "nested" inside the larger one.

## GET

GET variable list

The GET statement is a way to get data from the keyboard one character at a time. When the GET is executed, the character that was typed is received. If no character was typed, then a null (empty) character is returned, and the program continues without waiting for a key. There is no need to hit the RETURN key, and in fact the RETURN key can be received with a GET.

The word GET is followed by a variable name, usually a string variable. If a numeric variable is used and any key other than a number was hit, the program would stop with an error message. The GET statement may also be put into a loop, checking for an empty result, which will wait for a key to be struck.

This command can only be executed within a program.

### EXAMPLE:

```
10 GET A$:IF A$ = "" THEN 10
```

This line waits for a key to be struck.  
Typing any key will continue the program.



## GETKEY

GETKEY variable list

The GETKEY statement is very similar to the GET statement. Unlike the GET statement, GETKEY waits for the user to type a character on the keyboard.

This command can only be executed within a program.

EXAMPLE:

10 GETKEY A\$

This line waits for a key to be struck.  
Typing any key will continue the program.

## GET#

GET# file number,variable list

Used with a previously OPENed device or file to input one character at a time. Otherwise, it works like the GET statement.

This command can only be executed within a program.

EXAMPLE:

GET#1,A\$

## GOSUB

GOSUB line number

This statement is like the GOTO statement, except that the TED remembers where it came from. When a line with a RETURN statement is encountered, the program jumps back to the statement immediately following the GOSUB. The target of a GOSUB statement is called a subroutine. A subroutine is useful if there is a routine in your program that can be used by several different portions of the program. Instead of duplicating the section of program over and over, you can set it up as a subroutine, and GOSUB to it from the different parts of the program. Also see the RETURN statement.

EXAMPLE:

20 GOSUB 800

means go to the subroutine  
beginning at line 800 and execute it.

800 PRINT "HI THERE":RETURN

## GOTO or GO TO

GOTO line number

After a GOTO statement is executed, the next line to be executed will be the one with the line number following the word GOTO. When used in direct mode, GOTO line number allows you to start execution of the program at the given line number without clearing the variables.

EXAMPLE:

10 PRINT "COMMODORE"

The GOTO in line 20 makes line 10 repeat continuously until you press RUN/STOP.

20 GOTO 10

## GRAPHIC

GRAPHIC <mode [,clear option] | CLR>

This statement puts the C-264 in one of its 5 graphic modes:

mode	description
0	normal text
1	high-resolution graphics
2	high-resolution graphics, split screen
3	multicolor graphics
4	multicolor graphics, split screen

When executed, GRAPHIC 1 - 4 allocates a 10K bit-mapped area, and the start of the BASIC text area is moved above the HI-RES area. This area remains allocated even if the user returns to TEXT mode (GRAPHIC 0). If a second argument is given in the GRAPHIC command, and if that argument equals 1, the requested screen is also cleared. Executing a GRAPHIC CLR command de-allocated the 10K bit-mapped area, and makes it available once again for BASIC text and variables.

## IF ... THEN ... ELSE

IF expression THEN then-clause [:ELSE else-clause]

IF ... THEN lets the computer analyze a BASIC expression preceded by IF and take one of two possible courses of action. If the expression is true, the statement following THEN is executed. This expression may be any BASIC statement. If the expression is false, the program goes directly to the next line, unless an ELSE clause is present. The expression being evaluated may be a variable or formula, in which case it is considered true if non-zero, and false if zero. In most cases, there is an expression involving relational operators (=, <, >, <=, >=, <>, AND, OR, NOT).

The ELSE clause, if present, must be in the same line as the IF-THEN part. When an ELSE clause is present, it is executed when the THEN clause isn't executed. In other words, the ELSE clause executes when the IF expression is FALSE.

EXAMPLE:

```
50 IF X>0 THEN PRINT "OK":ELSE END
```

Checks the value of X. If X is greater than 0, the THEN clause is executed, and the ELSE clause isn't. If X is less than 0, the ELSE clause is executed and the THEN clause isn't.

## INPUT

INPUT [prompt string;] variable list

The INPUT statement allows the computer to get data into a variable from the person running the program. The program will stop, print a question mark (?) on the screen, and wait for the person to type the answer and hit the RETURN key.

The work INPUT is followed by a variable name or list of variable names separated by commas. There may be a message inside of quotes before the list of variables to be input. If this message (called a prompt) is present, there must be a semicolon (;) after the last quote of the prompt. When more than one variable is to be INPUT, they should be separated by commas when typed in. If you press the RETURN key without INPUTting a value, the INPUT variable retains the previous value. This statement can only be executed within a program.

EXAMPLE:

```
10 INPUT"PLEASE TYPE A NUMBER";A
20 INPUT "AND YOUR NAME";A$
30 INPUT B$
```

## INPUT#

INPUT# filename, variable list

This works like INPUT, but takes the data from a previously OPENed file or device. No prompt string is allowed. This command can only be used in program mode.

## LET

[LET] variable = expression

The word LET itself is hardly ever used in programs, since it is optional. The variable name which is to get the value or the result of a calculation is on the left side of the equal sign, and the number or formula is on the right side.

EXAMPLE:

```
10 LET A = 5
20 B = 6
30 C = A*B + 3
40 D$ = "HELLO"
```

## LOCATE

LOCATE x-coordinate, y-coordinate

The LOCATE command lets you put the pixel cursor (PC) anywhere on the screen. The PC is the current location of the starting point of the next drawing. Unlike the regular cursor, you can't see the PC, but you can move it with the LOCATE command. For example:

LOCATE 160, 100

positions the PC in the center of the high resolution screen. You won't see anything until you actually draw something. You can find out where the PC is at any time by using the RDOT(0) function to get the X-coordinate and RDOT(1) to get the Y-coordinate. The source of color of the dot at the PC can be found from the value of RDOT(2). In all drawing commands where a color option is present, you may select a value from 0 to 3, corresponding to the background, foreground, multicolor 1, or multicolor 2 as the source of the color.

## MONITOR

MONITOR

This command takes you out of BASIC into the built-in machine language monitor program. The monitor is used to develop, debug, and execute machine language programs more easily than from BASIC. See the section on monitor commands for more information. (When in the monitor, typing an X and hitting RETURN will get you back to BASIC.)

## NEXT

NEXT [variable, ..., variable]

The NEXT statement is always used in conjunction with the FOR statement. When the program gets up to a NEXT statement, it goes back to the FOR statement and checks the loop. (See FOR statement for more detail.) If the loop is finished, execution proceeds with the statement after the NEXT statement. The word NEXT may be followed by a variable name, or a list of variable names, separated by commas. If there are no names listed, the last loop started is the one being completed. If the variables are given, they are completed in order from left to right.

EXAMPLE:

```
10 FOR L = 1 TO 10:NEXT
20 FOR L = 1 TO 10:NEXT L
30 FOR L = 1 TO 10:FOR M = 1 TO 10:NEXT M,L
```

## ON

ON expression <GOTO|GOSUB> line numbers

This command can make the GOTO and GOSUB commands into special versions of the IF statement. The word ON is followed by an arithmetic expression, which is evaluated into a number. The word GOTO or GOSUB is followed by a list of line numbers separated by commas. If the result of the calculation is 1, the first line in the list is executed. If the result is 2, the second line number is executed, and so on. If the result is 0 or larger than the list of line numbers, the next line executed will be the statement following the ON statements. If the number is negative, an illegal quantity error will result.

### EXAMPLE:

```
10 INPUT X:IF X<0 THEN 10 When X = 1, ON sends control to the first line number in the list
20 ON X GOTO 10,50,50,70 (10). When X = 2, ON sends control to the second line (30), etc.
30 PRINT "TOO HIGH": GOTO10
50 PRINT "TOO LOW"
70 END
```

## OPEN

OPEN file number, device # [,secondary address ,"filename ,type ,mode"]

The OPEN statement allows the C-264 to access devices such as the cassette recorder and disk drive, a printer, or even the screen of the C-264. The word OPEN is followed by a logical file number, which is the number to which all other BASIC statements will refer. This number is from 1 to 255. There is always a second number after the first, called the device number. Device number 0 is the C-264 keyboard, 3 is the C-264 screen, 1 is the cassette recorder, 4 is the printer, 8 is the disk (usually). It is often a good idea to use the same file number as the device number. Following the second number may be a third number called the secondary address. In the case of the cassette, this can be 0 for read, 1 for write, and 2 for write with end-of-tape marker at the end. In the case of the disk, the number refers to the channel number. In the printer, the secondary addresses are used to command the printer. See the C-264 Programmer's Reference Manual or the manual for each specific device for more information on secondary addresses. There may also be a string following the third number, which could be a command to the disk drive or the name of the file on tape or disk. The type and mode refer to disk files only. (File types are prg,seq, and rel; modes are read and write.)

### EXAMPLES:

```
10 OPEN 3,3          OPENS the SCREEN as a device.
10 OPEN 1,0          OPENS the keyboard as a device.
20 OPEN 1,1,0,"DOT"  OPENS the cassette for reading, file to be searched for is named DOT.
```

```
OPEN 4,4            OPENS a channel to use the printer.
OPEN 15,8,15        OPENS the command channel on the disk.
```

5 OPEN 8,8,12,"0:TESTFILE,SEQ,WRITE" creates a sequential disk file for writing.

See also: CLOSE, CMD, GET#, INPUT#, and PRINT# statements, reserved variables ST, DS, and DS\$.

## PAINT

PAINT [color source] [,a,b] [,mode]

Color source ..... (0-3) (default is 1, foreground)  
a,b ..... starting coordinate, scaled (default is at PC)  
mode ..... 0 = paint an area defined by the color source selected  
              1 = paint an area defined by any non-background source

The PAINT command lets you fill an area with color. It will fill in the area around the specified point until a boundary of the same color (or any foreground color, depending on which mode you have chosen) is encountered:

The final PC will be at the starting point (a,b). Note that if the starting point is already the color of color source you name (or any non-background color when mode 1 is in effect) nothing will happen.

### EXAMPLE:

```
10 CIRCLE,160,100,65,50
20 PAINT,160,100
```

fills in the circle with color

## POKE

POKE location, value

The POKE command allows you to change any value in the C-264 RAM memory, and lets you modify many of the C-264 Input/Output registers. POKE is always followed by two numbers, (or expressions). The first number is a location inside the C-264 memory. This could have any value from 0 to 65535. (Some of these locations can make the C-264 do unusual things.) The second number is a value from 0 to 255, which will be placed in the location, replacing any value that was there previously.

### EXAMPLE:

```
10 POKE 28000,8           Sets location 28000 to 8
20 POKE 28*1000,27        Sets location 28000 to 27.
```

## PRINT

PRINT [printlist]

The PRINT statement is the major output statement in BASIC. While the PRINT statement is the first BASIC statement most people learn to use, there are many subtleties to be mastered here as well. The word PRINT can be followed by any of the following things:

Words inside of quotes  
Variable names  
Functions  
Punctuation marks

The words inside of quotes are often called literals because they are printed literally as they are typed in. When variable names are outside of quotes, the values they contain are printed.

Functions will have their values printed also. Punctuation marks are used to help format the data neatly on the screen. The comma divides the screen into 4 columns, while the semicolon doesn't leave any space at all. Either mark can be used as the last symbol in the statement. This results in the next thing PRINTed coming out as if it were continuing the same PRINT statement.

### EXAMPLE:

	RESULT
10 PRINT "HELLO"	HELLO
20 A\$ = "THERE":PRINT "HELLO,"A\$	HELLO,THERE
30 A = 4:B = 2:PRINT A + B	6
50 J = 41:PRINT J:PRINT J - 1	41 40
60 C = A + B:D = A - B:PRINT A;B;C,D	4 2 6 2

See also: POS(), SPC(), and TAB() FUNCTIONS.

## PRINT#

PRINT# filename, print list

There are a few differences between this statement and PRINT. First of all, PRINT# is followed by a number, which refers to the device or data file previously OPENed. The number is followed by a comma, and a list of things to be PRINTed. The comma and semicolon have the same effect on adding spaces as they do in the PRINT, but some devices may not work with TAB and SPC.

EXAMPLE:

```
100 PRINT#1,"HELLO THERE!";A$,B$
```

## PRINT USING

PRINT [#filename] USING format list; print list ;

These statements let you define the format of the string and numeric output you want to print. Put the format you want in quotes. Then add a semicolon and a list of what you want printed in the format. The list can be variables or the actual values you want printed. For example:

```
10 PRINT USING "$##.##";13.25,X,Y
20 PRINT USING "###>#";"CBM",A$
```

CHARACTER	NUMERIC	STRING
Pound Sign (#)	X	X
Plus ( + )	X	
Minus ( - )	X	
Decimal Point (.)	X	
Comma (,)	X	
Dollar Sign (\$)	X	
Four Carets (ffff)	X	
Equal Sign (=)		X
Greater Than Sign (>)		X

The pound sign (#) reserves room for a single character in the output field. If the data item contains more characters than you have # in your format field, the following occurs:

\* For a numeric item, the entire field is filled with asterisks (\*). No numbers are printed.

For example:

```
10 PRINT USING "####",X
```

For these values for X, this format displays:

```
A = 12.34      12
A = 567.89     568
A = 123456     ****
```

\* For a STRING item, the string data is truncated at the bounds of the field. Only as many characters are printed as there are pound signs (#) in the format item. Truncation occurs on the right.

The plus ( + ) and minus ( - ) signs can be used in either the first or last position of a format field, but not both. The plus sign is printed if the number is positive. The minus sign is printed if the number is negative.

If you use a minus sign and the number is positive, a blank is printed in the character position indicated by the minus sign.

If you don't use either a plus or minus sign in your format field for a numeric data item, a minus sign is printed before the first digit or dollar symbol if the number is negative, and no sign is printed if the number is positive. This means that you can print one character more if the number is positive. If there are too many digits to fit into the field specified by the # and + / - signs, then an overflow occurs and the field is filled with asterisks (\*).

A decimal point (.) symbol designates the position of the decimal point in the number. You can only have one decimal point in any format field. If you don't specify a decimal point in your format field, the value is rounded to the nearest integer and printed without any decimal places.

When you specify a decimal point, the number of digits preceding the decimal point (including the minus sign, if the value is negative) must not exceed the number of # before the decimal point. If there are too many digits an overflow occurs and the field is filled with asterisks (\*).

A comma (,) lets you place commas in numeric fields. The position of the comma in the format list indicates where the comma appears in a printed number. Only commas within a number are printed. Unused commas to the left of the first digit appear as the filler character. At least one # must precede the first comma in a field.

If you specify commas in a field and the number is negative, then a minus sign will be printed as the first character even if the character position is specified as a comma.

#### EXAMPLES:

FIELD	EXPRESSION	RESULT	COMMENT
### +	-.01	0.01 -	Leading zero added.
### -	1	1.0	Trailing zero added.
####	-100.5	-101	Rounded to no decimal places.
####	-1000	****	Overflow because four digits and minus sign cannot fit in field.
###.	10	10.	Decimal point added.
#\$##	1	\$1	Leading \$ sign.

A dollar sign (\$) symbol shows that a dollar sign will be printed in the number. You must specify at least one # before the dollar sign or else the dollar sign will not float. If you specify a dollar sign without a leading #, the dollar sign is printed in the position shown in the format field. If you specify at least one # before the dollar sign, the dollar sign floats to be placed just before the number. If you specify commas and/or a plus or minus sign in a format field with a dollar sign, your program will print a comma or sign before the dollar sign.

The four carets (↑↑↑↑) symbol is used to specify that the number is to be printed in E + format. You must use # in addition to the ↑↑↑↑ to specify the field width. The ↑↑↑↑ can appear either before or after the # in the format field.

You must specify four carets (↑↑↑↑) when you want to print a number in E — format (scientific notation). If you specify more than one but fewer than four carets, you will get a syntax error. If you specify more than four carets only the first four are used. The fifth caret is interpreted as a no text symbol.

An equal sign (=) is used to center a string in the field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is centered in the field. If the string contains more characters than can be fit into the field, the right most characters are truncated and the string fills the entire field.

A greater than sign (>) is used to right justify a string in a field. You specify the field width by the number of characters (# and =) in the format field. If the string contains fewer characters than the field width, the string is right justified in the field. If the string contains more characters than can be fit into the field, the right-most characters are truncated and the string fills the entire field.

## PUDEF

### PUDEF 1 through 4 characters

PUDEF lets you redefine up to 4 symbols in the PRINT USING statement. You can change blanks, commas, decimal points, and dollar signs into some other character by placing the new character in the correct position in the PUDEF control string.

Position 1 is the filler character. The default is a blank. Place a new character here when you want another character to appear in place of blanks.

Position 2 is the comma character. Default is a comma.

Position 3 is the decimal point.

Position 4 is the dollar sign.

### EXAMPLES:

10 PUDEF "*"	PRINTs * in the place of blanks.
20 PUDEF "@"	PRINTs @ in place of commas.
30 PUDEF ".,"	PRINTs decimal points in place of commas, and commas in place of decimal points.
40 PUDEF ".,£"	PRINTs English pound sign in place of \$. Other signs are the default values. Be sure to type them in like this or your computer assumes you want to PRINT blanks instead of default symbols.

## READ

### READ variable list

This statement is used to get information from DATA statements into variables, where the data can be used. Care must be taken to avoid reading strings where the READ statement expects a number, which will give you a TYPE MISMATCH ERROR.

## REM

### REM remark

The REMark is just a note to whoever is reading a LIST of the program. It may explain a section of the program, give information about the author, etc. REM statements in no way effect the operation of the program, except to add to its length (and therefore slow it down). The word REM may be followed by any text, although use of graphic characters will give strange results.

## RENAME

RENAME "old name" TO "new name" [,Ddrive number ,Unit number]

Used to rename a file on a diskette.

### EXAMPLE:

RENAME "TEST" TO "FINALTEST",D0	Changes the name of the file TEST to FINALTEST.
---------------------------------	---

## RESTORE

### RESTORE [line number]

When executed in a program, the DATA statement pointer (indicating which item will be read next) is reset to the first item in the list. This gives you the ability to re-READ the information. If a line number follows the RESTORE statement, the pointer will be set to that line. Otherwise the pointer will be reset to the first DATA statement in the program.



## RESUME

RESUME [line-number | NEXT]

*Resets back if error occurred*

Used to return to execution after TRAPPING an error. With no arguments, RESUME will attempt to re-execute the statement in which the error occurred. RESUME NEXT will resume execution on the line following the statement on which the error occurred, the RESUME line-number will GOTO the line-number and begin execution there.

*Clears ERR.*

## RETURN

This statement is always used with the GOSUB statement. When the program hits a RETURN statement, it will go to the statement immediately following the last GOSUB command executed. If no GOSUB was previously issued, then a RETURN WITHOUT GOSUB ERROR will occur.

## SCALE

SCALE <1|0>

The scaling of the bit maps in multicolor and high resolution modes can be changed with the SCALE command. Entering:

SCALE 1

turns scaling on. Coordinates will then be scaled from 0 to 1023 in both X and Y rather than the initial values, which are:

	X	Y
multicolor mode . . . . .	0 to 159	0 to 199
high resolution mode . . . . .	0 to 319	0 to 199
high res. split screen mode . . . . .	0 to 319	0 to 159
multicolor split screen mode . . . . .	0 to 159	0 to 159

Scaling can be turned off by entering SCALE 0.

## SCNCLR

SCNCLR

Clears the current screen, whether graphics, text, or both (split screen).

## SOUND

SOUND voice number, frequency, duration

This statement produces a SOUND using one of three voices with a frequency based on the range 0 - 1023 for a duration of 0 - 65535 60ths of a second.

V	Voice
1	voice 1 (tone)
2	voice 2 (tone)
3	voice 2 (white noise)

After executing a SOUND instruction, BASIC will resume execution. If, however, a SOUND for voice N is requested, and the previous SOUND for the same N is still playing, BASIC will wait for the previous SOUND to complete. SOUND with a duration of 0 is a special case. It will cause BASIC to abort the current SOUND for that voice immediately, regardless of the duration remaining on the previous SOUND. See the MUSIC NOTE TABLE for the frequency values that correspond to real notes.

## SSHAPE / GSHAPE

SSHAPE and GSHAPE are used to save and restore rectangular areas of multicolor or high resolution screens using BASIC string variables. The command to save an area is:

SSHAPE strvar, a1,b1 [,a2,b2]

strvar ..... String name to save data in  
a1,b1 ..... Corner coordinate (scaled)  
a2,b2 ..... Corner coordinate opposite (a1,b1) (default is the PC)

Because BASIC limits string lengths to 255 characters, the size of the area you may save is limited. The string size required can be calculated using one of the following (unscaled) formulas:

$L(\text{multi-color mode}) = \text{INT} ( (\text{ABS}(a1-a2) + 1) / 4 + .99 ) * (\text{ABS}(b1-b2) + 1) + 4$   
 $L(\text{hi-res}) = \text{INT} ( (\text{ABS}(a1-a2) + 1) / 8 + .99 ) * (\text{ABS}(b1-b2) + 1) + 4$

The shape is saved row by row. The last four bytes of the string contain the column and row lengths less one (i.e.:  $\text{ABS}(a1-a2)$ ) in low/high byte format (if scaled divide the lengths by 3.2 (X) and 5.12 (Y)).

Now, the command to draw a saved shape to any area of the screen:

GSHAPE string [, [a,b] [,mode] ]

string ..... Aggregate with shape to draw  
a,b ..... Top left coordinate telling where (scaled — the default is the PC)  
mode ..... Replacement mode:  
0/ place shape as is (default)  
1/ place field inverted shape  
2/ OR shape with area  
3/ AND shape with area  
4/ XOR shape with area

## STOP

### STOP

This statement will halt the program. The following message appears: BREAK IN LINE xxxx, where xxxx is the line number containing the STOP. The program can be re-started at the statement following STOP if you use the CONT command. The STOP statement is usually used while debugging a program.

## SYS

SYS address

The word SYS is followed by a decimal number or numeric variable in the range 0 to 65535. The program will at this point begin executing the machine language program starting at that memory location. This is similar to the USR function, but does not pass a parameter. See the C-264 Programmer's Reference Guide for information about machine language programs.

## TRAP

TRAP line-number

When turned on, TRAP intercepts all error conditions (including STOP KEY) except "LINE NUMBER NOT FOUND". In the event of any execution error, the error flag is set, and execution is transferred to the line number named in the TRAP statement. The line number in which the error occurred can be found by interrogating the variable EL, and the specific error condition is contained in ER. The string function ERR\$(ER) will give the error message corresponding to any error condition ER.

**NOTE:** An error in a TRAP routine cannot be trapped. The RESUME statement can be used to resume execution. TRAP with no line-number argument will turn off error TRAPPING. *It does not*

*exit from error routine, but not if ER has changed*

## **TRON**

### **TRON**

TRON is used in program debugging. This statement begins trace mode. When you are in trace mode, as each statement executes, the line number of that statement is printed.

## **TROFF**

### **TROFF**

This statement turns trace mode off.

## **VOL**

VOL volume level

Sets the current volume level. V may be from 0 - 8, with 8 being maximum volume, and 0 being off. VOL affects all voices.

## **WAIT**

WAIT address, value 1[,value 2]

The WAIT statement is used to halt the program until the contents of a location in memory change in a specific way. The word WAIT is followed by a number, which is the memory address being checked. Then comes a comma, and another number. There may be another comma and a third number as well. These last two numbers must be within the range 0-255.

The contents of the memory location are first exclusive-ORed with the third number, if present, and then logically ANDed with the second number. If the result is zero, the program goes back to that memory location and checks again. When the result is non-zero, the program continues with the next statement.

## **4. FUNCTIONS**

### **a. NUMERIC FUNCTIONS**

ABS(X) (absolute value)

The absolute value returns the positive value of the number.

ATN(X) (arctangent)

Returns the angle, measured in radians, whose tangent is X.

COS(X) (cosine)

Returns the value of the cosine of X, where X is an angle measured in radians.

DEC(hexadecimal-string) *- needs quotes*

Returns decimal value of hexadecimal-string ( 0 < hexadecimal-string < FFFF)

EXP(X)

Returns the value of the mathematical constant e (2.71827183) raised to the power of X.

FNxx(x)

Returns the value of the user-defined function xx created in a DEF FNxx statement.

**INSTR (string 1, string 2 [,starting position] )**

Returns position of string 2 in string 1 at or after starting-position. Starting-position defaults to the beginning of string 1. If no match is found, a value of 0 is returned.

**INT(X) (integer)**

Returns the truncated value of X, that is, with all decimal places to the right of the decimal point removed. The result will always be less than or equal to X. Thus, any negative numbers with decimal places will become the integer less than their current value.

If the INT function is to be used for rounding up or down, the form is INT(X + .5).

**EXAMPLE:**

$X = \text{INT}(X * 100 + .5) / 100$  Rounds to the nearest penny.

**JOY (n)**

When n = 1 . . . . Position of joystick #1

n = 2 . . . . Position of joystick #2

Any value of 128 or more means the fire button is also depressed. The direction is indicated as follows:

fire = 128 +

		1		
	8		2	
7		0		3
	6		4	
		5		

**LOG(X) (logarithm)**

This will return the natural log of X. The natural log is log to the base e (see EXP(X)). To convert to log base 10, simply divide by LOG(10).

**PEEK(X)**

This is used for finding out the contents of memory location X, in the range of 0-65535, giving a result from 0-255. This is often used in conjunction with the POKE statement.

**RCLR (N)**

Returns current color assigned to source N (0 through 4)

(0 = background, 1 = foreground, 2 = multicolor 1, 3 = multicolor 2, 4 = border).

**RDOT (N)**

Returns characteristics of current position of the pixel cursor (PC) at XPOS/YPOS.

N = 0 for XPOS

1 for YPOS

2 color source

**RGR (N)**

Returns current graphic mode (dummy argument).

**RLUM (N)**

Returns current luminance level assigned to source N.

### RND (X) (random number)

This function will return a random (or nearly so) number between 0 and 1. This is useful in games, to simulate dice rolls and other elements of chance, and is also used in some statistical applications. The first random number should be generated by the formula  $RND(-TI)$ , to start things off differently every time. After this, the number in X should be a 1, or any positive number. If X is zero, RND is seeded from the hardware clock and random numbers are generated. A negative value for X will seed the random number generator from X and give random numbers. The use of the same negative number for X will result in the same sequence of random numbers. A positive value gives random numbers based on the previous seed.

To simulate the rolling of a die, use the formula  $INT(RND(1)*6 + 1)$ . First the random number from 0-1 is multiplied by 6, which expands the range to 0-6 (actually, greater than zero and less than six). Then 1 is added, making the range 1-under 7. The INT function chops off all the decimal places, leaving the result as a digit from 1 to 6. To simulate 2 dice, add two of the numbers obtained by the above formula together.

#### EXAMPLE:

100 X =  $INT(RND(1)*6) + INT(RND(1)*6) + 2$   
100 X =  $INT(RND(1)*1000) + 1$   
100 X =  $INT(RND(1)*150) + 100$

Simulate 2 dice.  
Number from 1-1000.  
Number from 100-249.

### SGN(X) (sign)

This function returns the sign, as in positive, negative, or zero, of X. The result will be + 1 if positive, 0 if zero, and - 1 if negative.

### SIN(X) (SINE)

This is the trigonometric sine function. The result will be the sine of X, where X is an angle in radians.

### SQR(X) (square root)

This function will return the square root of X, where X is as positive number or 0. If X is negative, an ILLEGAL QUANTITY ERROR results.

### TAN(X) (tangent)

The result will be the tangent of X, where X is an angle in radians.

### USR (X)

When this function is used, the program jumps to a machine language program whose starting point is contained in memory locations 760 and 761. The parameter is passed to the machine language program, which will return another number back to the BASIC program. See the C-264 PROGRAMMER'S REFERENCE MANUAL for more details on this, and on machine language programming.

### VAL(X\$)

This function converts the string X\$ into a number, and is essentially the inverse operation from STR\$. The string is examined from the left-most character to the right, for as many characters as are in recognizable number format. If the C-264 finds illegal characters, only the portion of the string up to that point is converted.

#### EXAMPLE:

10 X = VAL("123.456")	X = 123.456
10 X = VAL("3E03")	X = 3000
10 X = VAL("12A13B")	X = 12
10 X = VAL("RIU017")	X = 0
10 X = VAL("-1.23.23.23")	X = - 1.23

## **b. STRING FUNCTIONS**

### **ASC(X\$)**

This function will return the ASCII code of the first character of X\$.

### **CHR\$(X)**

This is the opposite of ASC, and returns a string character whose ASCII code is X.

### **ERR\$(N)**

Returns string describing error condition N (see TRAP).

### **HEX\$(N)**

Returns a 4 character string containing the hexadecimal representation of value N ( $0 < N < 65536$ )

### **LEFT\$(S\$,X)**

This will return a string containing the leftmost X characters of X\$.

### **LEN(X\$)**

Returns the number of characters (including spaces and other symbols) in the string X\$.

### **MID\$(X\$,S,X)**

This will return a string containing X characters, starting from the Sth character in X\$. MID\$ can also be used on the left side of assignment statement.

MID\$ may also be used as a pseudo-variable as well as a function. MID\$(string-variable, starting-position, length) = source-string.

Reassigns values of positions (starting-position) through (starting-position + length) of source-string to the characters of string-variable in corresponding locations. Length defaults to the length of string-variables, and an error results if (starting-position + length) is greater than the length of source-string.

#### **EXAMPLE:**

```
10 A$ = "THE DOG IN THE HAT":  
20 PRINT A$  
30 MID$(A$,5,3) = "CAT"  
40 PRINT A$
```

### **RIGHT\$(X\$,X)**

This will return the rightmost X characters in X\$.

### **STR\$(X)**

This will return a string which is identical to the PRINTed version of X\$.

#### **EXAMPLE:**

```
A$ = STR$(X)
```

### c. OTHER FUNCTIONS

#### **FRE(X)**

This function returns the number of unused bytes available in memory. X is a dummy argument.

#### **POS(X)**

This function returns the number of the column (0-39) which the next PRINT statement will begin on the screen. X is a dummy argument.

#### **SPC(X)**

This is used in the PRINT statement to skip over X spaces. X can have a value from 0-255.


































#### **TAB(X)**

This is used in the PRINT statement. The next item to be printed will be in column number X. X can have a value from 0-255. For screen output, TAB (x) is the same as SPC (x).

#### **PI ( $\pi$ )**

The PI symbol, when used in an equation, has the value of 3.14159265.

## APPENDIX C: BASIC 3.5 ABBREVIATIONS

KEYWORD	ABBREVIATION	TYPE
ABS	a  B	function—numeric
ASC	a  S	function—numeric
ATN	a  T	function—numeric
AUTO	a  U	command
BACKUP	b  A	command
BOX	b  O	statement
CHAR	ch  A	statement
CHRS	c  H	function—string
CIRCLE	c  I	statement
CLOSE	cl  O	statement
CLR	c  L	statement
CMD	c  M	statement
COLLECT	col  L	command
COLOR	co  L	statement
CONT	c  O	command
COPY	co  P	command
COS	none	function—numeric
DATA	d  A	statement
DEC	none	function—numeric
DEF FN	d  E	statement
DELETE	de  L	command
DIM	d  I	statement
DIRECTORY	di  R	command
DLOAD	d  L	command
DO	none	statement
DRAW	d  R	statement
DSAVE	d  S	command
END	e  N	statement
ERR\$	e  R	function—string
EXP	e  X	function—numeric
FOR	f  O	statement
FRE	f  R	function—numeric
GET	g  E	statement
GETKEY	getk  E	statement
GET#	none	statement
GOSUB	go  S	statement
GOTO	g  O	statement



GRAPHIC	g	SHIFT	R	statement
GSHAPE	g	SHIFT	S	statement
HEADER	he	SHIFT	A	command
HEX\$	h	SHIFT	E	function—string
IF...GOTO		none		statement
IF...THEN...ELSE		none		statement
INPUT		none		statement
INPUT#	i	SHIFT	N	statement
INSTR	in	SHIFT	S	function—numeric
INT		none		function—numeric
JOY	j	SHIFT	O	function—numeric
KEY	k	SHIFT	E	command
LEFT\$	le	SHIFT	F	function—string
LEN		none		function—numeric
LET	l	SHIFT	E	statement
LIST	l	SHIFT	I	command
LOAD	l	SHIFT	O	command
LOCATE	lo	SHIFT	C	statement
LOG		none		function—numeric
LOOP	lo	SHIFT	O	statement
MID\$	m	SHIFT	I	function—string
MONITOR	m	SHIFT	O	statement
NEW		none		command
NEXT	n	SHIFT	E	statement
ON...GOSUB	on...go	SHIFT	S	statement
ON...GOTO	on...g	SHIFT	O	statement
OPEN	o	SHIFT	P	statement
PAINT	p	SHIFT	A	statement
PEEK	p	SHIFT	E	function—numeric
POKE	p	SHIFT	O	statement
POS		none		function—numeric
PRINT	?			statement
PRINT#	p	SHIFT	R	statement
PRINT USING	?us	SHIFT	I	statement
PUDEF	p	SHIFT	U	statement
RCLR	r	SHIFT	C	function—numeric
RDOT	r	SHIFT	D	function—numeric
READ	r	SHIFT	E	statement
REM		none		statement
RENAME	re	SHIFT	N	command
RENUMBER	ren	SHIFT	U	command
RESTORE	re	SHIFT	S	statement
RESUME	res	SHIFT	U	statement

RETURN	re	SHIFT	T	statement
RGR	r	SHIFT	G	function—numeric
RIGHT\$	r	SHIFT	I	function—string
RLUM	r	SHIFT	L	function—numeric
RND	r	SHIFT	N	function—numeric
RUN	r	SHIFT	U	command
SAVE	s	SHIFT	A	command
SCALE	sc	SHIFT	A	statement
SCNCLR	s	SHIFT	C	statement
SCRATCH	sc	SHIFT	R	command
SGN	s	SHIFT	G	function—numeric
SIN	s	SHIFT	I	function—numeric
SOUND	s	SHIFT	O	statement
SPC(	s	SHIFT	P	function—special
SQR	s	SHIFT	Q	function—numeric
SSHAPE	s	SHIFT	S	statement
STatus	st	SHIFT	A	reserved—numeric variable
STOP	s	SHIFT	T	statement
STR\$	st	SHIFT	R	function—string
SYS	s	SHIFT	Y	statement
TAB(	t	SHIFT	A	function—special
TAN		none		function—numeric
TI		none		reserved—numeric variable
TI\$		none		reserved—string variable
TRAP	t	SHIFT	R	statement
TROFF	tro	SHIFT	F	statement
TRON	tr	SHIFT	O	statement
UNTIL	u	SHIFT	N	statement
USR	u	SHIFT	S	function—special
VAL		none		function—numeric
VERIFY	v	SHIFT	E	command
VOL	v	SHIFT	O	statement
WAIT	w	SHIFT	A	statement
WHILE	w	SHIFT	H	statement

## APPENDIX D: TEDMON

### INTRODUCTION

TEDMON is a built-in machine language program which lets you easily write machine language programs. TEDMON includes a machine language monitor, a mini assembler, and a disassembler.

Machine language programs written using TEDMON can run by themselves, or be used as very fast 'subroutines' for BASIC programs since TEDMON has the ability to coexist peacefully with BASIC.

### TEDMON COMMANDS

A ASSEMBLE	Assemble a line of 6502 code
C COMPARE	Compare two sections of memory and report differences.
D DISASSEMBLE	Disassemble a line of 6502 code.
F FILL	Fill memory with the specified byte.
G GO	Start execution at the specified address.
H HUNT	Hunt through memory for all occurrences of certain bytes.
L LOAD	Load a file from tape or disk.
M MEMORY	Display the hexadecimal values of memory locations.
R REGISTERS	Display the 6502 Registers.
S SAVE	Save to tape or disk.
T TRANSFER	Transfer code from one section of memory to another.
X EXIT	eXit TEDMON.

### USING TEDMON

1. Enter TEDMON by typing:  
MONITOR

TEDMON will respond by displaying the 6502 registers and flashing the cursor. The cursor is your prompt that lets you know that TEDMON is waiting for your commands.

### COMMAND DESCRIPTIONS

COMMAND: A (ASSEMBLE)

PURPOSE: Enter a line of assembly code.

SYNTAX: A <address> <opcode mnemonic> <operand>

<address> A four-digit hexadecimal number indicating the location in memory to place the opcode.

<opcode mnemonic> A standard MOS assembly language mnemonic eg. LDA, STX, ROR, etc.

<operand> The operand, when required, can be of any of the legal addressing modes. (For zero-page modes a 2 digit hex number whose value is less than \$100 is required. For non-zero page addresses 4 digit hex numbers are required.)

A RETURN is used to indicate the end of the assembly line. If there are any errors on the line, a question mark is displayed to indicate an error, and the cursor moves to the next line. The screen editor can be used to correct any errors on the line.

After a line of code is successfully assembled, the assembler will print a prompt containing the next legal memory location for an instruction, so A and the line number do not have to be typed more than once when typing assembly language programs into the TED.

**EXAMPLE:**

```
.A 1200 LDX #$00  
.A 1202
```

**COMMAND: C (COMPARE)**

**PURPOSE:** Compare two areas of memory

**SYNTAX:** C <address 1> <address 2> <address 3>

<Address 1> is a hex number indicating the start address of the area of memory to compare against.

<Address 2> is a hex number indicating the end address of the area of memory to compare against.

<Address 3> is a hex number indicating the start address of the other area of memory to compare with.

If the two areas of memory are the same, then TEDMON will print a RETURN and flashing cursor, indicating that the second area of memory is the same as the first. The addresses of any bytes in the two areas which are different are printed out on the screen.

**COMMAND: D (DISASSEMBLE)**

**PURPOSE:** Disassemble machine code into assembly language mnemonics and operands.

**SYNTAX:** D <address> <address 2>

<address> A hexadecimal number setting the address to start the disassembly.

<address 2> An optional hexadecimal ending address of code to be disassembled.

The format of the disassembly is only slightly different than the input format of an assembly. The difference is that the first character of a disassembly is a comma rather than an A (for readability).

A disassembly listing can be modified using the screen editor. Make any changes to the mnemonic or operand on the screen, then hit a carriage return. This will enter the line and call the assembler for further modifications.

A disassembly can be paged. Typing a D will cause the next page of disassembly to scroll onto the screen.

**EXAMPLE:**

```
D 1000 1400  
. 1000 LDA #$00  
. 1002 ???  
. 1003 BNE $11030
```

**COMMAND: F (FILL)**

**PURPOSE:** Fill a range of locations with a specified byte.

**SYNTAX:** F <address 1> <address 2> <byte>

<address 1> The first location to fill with the <byte>

<address 2> The last location to fill with the <byte>

<byte value> A 2 digit hexadecimal number to be written

This command is useful for initializing data structures or any other RAM area.

**EXAMPLE:** F 0400 0518 EA

Fills memory locations from \$0400 to \$0518 with \$EA (a NOP instruction).

**COMMAND: G (GO)**

**PURPOSE:** Begin execution of a program at a specified address.

**SYNTAX:** G<address>

<address> An optional argument specifying the new value of the program counter and address where execution is to start. When <address> is left out execution will begin at the current Program Counter. (The current PC can be viewed using the R command.)

The GO command will restore all registers (displayable by the R command) and begin execution at the specified starting address. Caution is recommended in using the GO command.

**EXAMPLE:** G 140C

Execution begins at location \$140C.

**COMMAND: H (HUNT)**

**PURPOSE:** Hunt through memory within a specified range for all occurrences of a set of bytes.

**SYNTAX:** H <address 1> <address 2> <data>

<address 1> beginning address of hunt procedure

<address 2> ending address of hunt procedure

<data> data to search for may be hexadecimal or an ASCII string. An ASCII string is specified by preceding the first character with a single quote, eg, 'STRING. Data may be a single or multiple element argument.

**EXAMPLE:**

H C000 FFFF 'READ            Search for ASCII string

READ

H A000 A101 A9 FF 4C        Search for data \$A9, \$ff, \$4C.

**COMMAND: L (LOAD)**

**PURPOSE:** Load a file from cassette or disk.

**SYNTAX:** L "<filename>", <device>

<filename> is any legal C-264 filename.

<device> is a two-digit byte indicating the device to load from.

01 is cassette

08 is disk    (or 09, etc.)

The Load command causes a file to be loaded into memory. The starting address is contained in the first two bytes of the file (a PGM file). In other words, the Monitor Load command always loads a file into the same place it was saved from. This is very important in machine language work, since few programs are completely relocatable. The file will be loaded into memory until the EOF is found.

**EXAMPLE:**

L "SCREEN", 01            reads a file from cassette.  
L "TANK", 08            reads a file from disk drive.

**COMMAND: M (MEMORY DISPLAY)**

**PURPOSE:** Display memory as a hexadecimal and ASCII dump within the specified address range.

**SYNTAX:** M <address 1> <address 2>

<address 1> First address of memory dump. Optional. If omitted one page will be displayed. The first byte will be the last address specified.

<address 2> Last address of memory dump. Optional. If omitted one page will be displayed. The first byte will be the data of <address 1>.

Memory is displayed in the following format:

>A048 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE

Memory content may be edited using the screen editor. Move the cursor to the data to be modified and type the desired correction and hit return. If there is a bad RAM location or an attempt to modify ROM, an error flag (?) will be displayed.

An ASCII dump of the data is displayed in REVERSE (to contrast the dump with other data displayed on the screen) to the right of the hex data. When a character is not printable, it will be displayed as a reversed period (.).

As with the Disassembly command, you can page down by typing M and RETURN.

**EXAMPLE:**

M 1C00

>1C00 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C08 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C10 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C18 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C20 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C28 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C30 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C38 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C40 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C48 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C50 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE  
>1C58 41 E7 00 AA AA 00 98 56 45 :A! . \* . . VE

**COMMAND: R (REGISTER DISPLAY)**

**PURPOSE:** Show important 6502 registers. The program status register, the program counter, the accumulator, the X and Y index registers and the stack pointer are displayed.

## SYNTAX: R

Note that the stack pointer is displayed without its implied 8th bit.

Since the 8th bit of the stack pointer has been mentioned, it is appropriate to point out a bug in the 6502. When a PHP instruction is executed, the stack 8th bit of the stack pointer is OR'ed into the status byte and is stored on the stack with bit 4 (the break flag!) always set. For 99.9% of all applications this makes no difference. When this bug does turn up, it causes problems very difficult to track down.

## EXAMPLE: R

```
PC SR AC XR YR SP
; 1002 01 02 03 04 F6
```

## COMMAND: S (SAVE)

PURPOSE: Save the contents of memory onto tape or disk.

SYNTAX: S "<filename>",<device>,<address 1>,<address 2>

<filename> Any legal C-264 filename. To save the data, <filename> must be enclosed in double quotes. Single quotes are illegal.

<device> Two possible devices are cassette and disk. To save on cassette, use device 01. The device number of the C-264 disk drive is usually 08. However, this can be changed (i.e. when using more than one disk. See C-264 DISK MANUAL)

<address 1> Starting address of memory to be saved.

<address 2> Ending address of memory to be saved + 1. All data up to but not including the byte of data at this address will be saved.

The file created by this command is a program file. That is the first two bytes contain the starting address <address 1> of the data. The file may be recalled using the L command.

EXAMPLE: S "GAME", 08, 0400, 0C00

Saves memory from \$0400 to \$0BFF onto disk.

## COMMAND: T (TRANSFER)

PURPOSE: Transfer segments of memory from one memory area to another.

SYNTAX: T <address 1> <address 2> <address 3>

<address 1> Starting address of data to be moved

<address 2> Ending address of data to be moved

<address 3> Starting address of new location (where the data will go)

Data can be moved from low memory to high memory or vice-versa. Additional memory segments of any length can be moved forward or backward any number of bytes (i.e., shifted)

EXAMPLE: T 1400 1600 1401

Shifts data from \$1400 up to and including \$1600 one byte higher in memory.

**COMMAND: V (VERIFY)**

**PURPOSE:** Verify a file on cassette or disk with the memory contents.

**SYNTAX:** V "<filename>", <device>

<filename> is any legal C-264 filename.

<device> is a hex number indicating the device the file is on.

The Verify command compares a file to memory contents. If the file does not match memory, **ERROR** is printed.

**EXAMPLES:** V "SCREEN", 8

V "FILE 7", 1

**COMMAND: X (eXit)**

**PURPOSE:** Exit to BASIC

**SYNTAX:** X

When the X command is given, the machine stack pointer is set to the current stack pointer value (see the R command). If this is modified in any way, after exiting to BASIC the BASIC CLR command should be used to reset the stack pointer.



## APPENDIX E: CONVERTING STANDARD BASIC PROGRAMS TO COMMODORE BASIC 3.5

If you have programs written in a BASIC other than Commodore BASIC, some minor adjustments may be necessary before running them on the Commodore 264. We've included some hints to make the conversion easier.

### STRING DIMENSIONS

Delete all statements that are used to declare the length of strings. A statement such as `DIM A$(I,J)`, which dimensions a string array for J elements of length I, should be converted to the Commodore BASIC statement `DIM A$(J)`.

Some BASICs use a comma or ampersand for string concatenation. Each of these must be changed to a plus sign, which is the Commodore BASIC operator for string concatenation.

In Commodore BASIC, the `MID$`, `RIGHT$`, and `LEFT$` functions are used to take substrings of strings. Forms such as `A$(I)` to access the Ith character in A\$, or `A$(I,J)` to take a substring of A\$ from position I to J, must be changed as follows:

Other BASIC	Commodore BASIC
<code>A\$(I) = X\$</code>	<code>MID\$(A\$,I,J) = X\$</code>
<code>A\$(I,J) = X\$</code>	<code>MID\$(A\$,I,J) = X\$</code>

### MULTIPLE ASSIGNMENTS

To set B and C equal to zero, some BASICs allow statements of the form:

```
10 LET B = C = 0
```

Commodore BASIC would interpret the second equal sign as a logical operator and set `B = -1` if `C = 0`. Instead, convert this statement to:

```
10 C = 0 : B = 0
```

### MULTIPLE STATEMENTS

Some BASICs use a backslash (/) to separate multiple statements on a line. With Commodore BASIC, separate all statements by a colon (:).

### MAT FUNCTIONS

Programs using the MAT functions available on some BASICs must be rewritten using `FOR` . . . `NEXT` loops to execute properly.

# APPENDIX F C-264 MEMORY REGISTER MAP

REG		:DB7	:DB6	:DB5	:DB4	:DB3	:DB2	:DB1	:DB0
0	\$FF00	TIMER # 1 RELOAD VALUE, BITS 0:7 (LOW)							
1	\$FF01	TIMER # 1 RELOAD VALUE, BITS 8:15 (HIGH)							
2	\$FF02	TIMER # 2 RELOAD VALUE, BITS 0:7 (LOW)							
3	\$FF03	TIMER # 2 RELOAD VALUE, BITS 8:15 (HIGH)							
4	\$FF04	TIMER # 3 RELOAD VALUE, BITS 0:7 (LOW)							
5	\$FF05	TIMER # 3 RELOAD VALUE, BITS 8:15 (HIGH)							
6	\$FF06	:TEST	:FCM	:BMM	:BLANK	:# ROWS	:Y2	:Y1	:Y0
7	\$FF07	:RVS OFF	:PAL	:FREEZE	:MCM	:# COLS	:X2	:X1	:X0
8	\$FF08	KEYBOARD LATCH							
9	\$FF09	:IRQ	:IT3	:NC	:IT2	:IT1	:ILP	:IRAS	:NC
10	\$FF0A	:NC	:EI-T3	:NC	:EI-T2	:EI-T1	:EILP	:EI-RAS	:RC8
11	\$FF0B	:RC7	:RC6	:RC5	:RC4	:RC3	:RC2	:RC1	:RC0
12	\$FF0C	:NC	:NC	:NC	:NC	:NC	:NC	:C9	:CUR8
13	\$FF0D	:CUR7	:CUR6	:CUR5	:CUR4	:CUR3	:CUR2	:CUR1	:CUR0
14	\$FF0E	:SND1:7	:SND1:6	:SND1:5	:SND1:4	:SND1:3	:SND1:2	:SND1:1	:SND1:0
15	\$FF0F	:SND2:7	:SND2:6	:SND2:5	:SND2:4	:SND2:3	:SND2:2	:SND2:1	:SND2:0
16	\$FF10	:NC	:NC	:NC	:NC	:NC	:NC	:SND2:9	:SND2:8
17	\$FF11	:SND-REL	:NOISE	:V2-SEL	:V1-SEL	:VOL3	:VOL2	:VOL1	:VOL0
18	\$FF12	:NC	:NC	:BMB2	:BMB1	:BMB0	:R BANK	:S1:9	:S1:8
19	\$FF13	:CB5	:CB4	:CB3	:CB2	:CB1	:CB0	:SCLOCK	:STATUS
20	\$FF14	:VM4	:VM3	:VM2	:VM1	:VM0	:NC	:NC	:NC
21	\$FF15	BKGD0	:NC	:LUM2	:LUM1	:LUM0	:COLOR3	:COLOR2	:COLOR1
22	\$FF16	BKGD1	:NC	:LUM2	:LUM1	:LUM0	:COLOR3	:COLOR2	:COLOR1
23	\$FF17	BKGD2	:NC	:LUM2	:LUM1	:LUM0	:COLOR3	:COLOR2	:COLOR1
24	\$FF18	BKGD3	:NC	:LUM2	:LUM1	:LUM0	:COLOR3	:COLOR2	:COLOR1
25	\$FF19	BKDG4	:NC	:LUM2	:LUM1	:LUM0	:COLOR3	:COLOR2	:COLOR1
26	\$FF1A		:NC	:NC	:NC	:NC	:NC	:BRE9	:BRE8
27	\$FF1B		:BRE7	:BRE6	:BRE5	:BRE4	:BRE3	:BRE2	:BRE1
28	\$FF1C		:NC	:NC	:NC	:NC	:NC	:NC	:VL8
29	\$FF1D		:VL7	:VL6	:VL5	:VL4	:VL3	:VL2	:VL1
30	\$FF1E		:H8	:H7	:H6	:H5	:H4	:H3	:H2
31	\$FF1F		:NC	:BL3	:BL2	:BL1	:BL0	:VSUB2	:VSUB1
62	\$FF3E	ROM SELECT							
63	\$FF3F	RAM SELECT							

ADDRESS	CONTENTS	NOTES
	< ..... >	
\$FFFE-FFFF	< IRQ VECTOR > *	
\$FFFC	< RES VECTOR > *	
\$FFFA	< NMI VECTOR (NOT USED) > *	ROM BANK HIGH (cont)
	< > *	
\$FFB4-FFFF	< KERNAL JUMP TABLE > *	
	< >	
\$FF00-FF3F	< TED CHIP > *	
	< > *	
\$FE00-FEFF	< DMA DISK SYSTEM > *	TED Chip and I/O
\$FDE0-FDEF	< > *	space appear in
\$FDD0-FDDF	< CARTRIDGE BANK PORT > *	all memory maps.
\$FD10-FD1F	< 6529 PARALLEL PORT > *	
\$FD00-FD0F	< ACIA > *	
	< >	
\$FCD0-FCFF	< >	ROM banking routines
	< >	(appears in all ROM maps)
\$D800-FCFF	< > *	
	< > *	
\$D000-D7FF	< CHARACTER ROM > *	ROM BANK HIGH
	< > *	
\$C000-D7FF	< MORE BASIC > *	
	< > *	
\$8000-BFFF	< BASIC >	ROM BANK LOW
	< >	
\$4000-FFFF	< RAM, ALSO START OF BASIC TEXT >	
	< AREA WHEN HIRES GRAPHICS ARE USED >	
\$2000-3FFF	< BIT MAP SCREEN DATA >	
	< >	
\$C1C00-1FFF	< HIRES SCREEN VIDEO MATRIX >	
	< >	
\$1800-1BFF	< HIRES SCREEN ATTRIBUTE BYTES >	
	< >	
\$1000	< BASIC TEXT AREA (BIT MAP OFF) >	
	< >	
\$0C00-0FFF	< TEXT VIDEO MATRIX >	
	< >	
\$0800-0BFF	< TEXT ATTRIBUTE BYTES >	
	< >	
\$0000-07FF	< SYSTEM STORAGE >	
	< ..... >	

\* NOTE: In the 64K RAM system, RAM goes from \$0000-\$FCFF, and from \$FF40-\$FFFF.

## APPENDIX G: MUSICAL NOTE TABLE

NOTE	SOUND REGISTER VALUE	ACTUAL FREQUENCY (HZ)
A	7	110
B	118	123.5
C	169	130.8
D	262	146.8
E	345	164.7
F	383	174.5
G	453	195.9
A	516	220.2
B	571	246.9
C	596	261.4
D	643	293.6
E	685	330
F	704	349.6
G	739	392.5
A	770	440.4
B	798	494.9
C	810	522.7
D	834	588.7
E	854	658
F	864	699
G	881	782.2
A	897	880.7
B	911	989.9
C	917	1045
D	929	1177
E	939	1316
F	944	1398
G	953	1575

The above table contains the sound register values of four octaves of notes. The sound register values are used as the second parameter of the SOUND command. To use the first note in the table (A - sound register value 7) use the 7 as the second number after the SOUND command — SOUND 1,7,30.

Use the following formula to find the sound register values for frequencies other than those in the table:
















$$\text{SOUND REGISTER VALUE} = 1024 - (111860.781/\text{FREQUENCY})$$

Both the table of sound register values and the above formula are for NTSC televisions. This is the television standard used throughout the United States and all of Canada. If you are in a country where PAL is the television standard, you should use the following formula to calculate new sound register values for the entire table:









$$\text{SOUND REGISTER VALUE} = 1024 - (111840.45/\text{FREQUENCY})$$

## APPENDIX H: ASCII AND CHR\$ CODES

This appendix shows you what characters will appear if you PRINT CHR\$(X), for all possible values of X. It will also show the values obtained by typing PRINT ASC("X"), where X is any character you can type. This is useful in evaluating the character received in a GET statement, converting upper/lower case, and printing character based commands (like switch to upper/lower case) that could not be enclosed in quotes.

PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$	PRINTS	CHR\$
	0		17	"	34	3	51
	1		18	#	35	4	52
	2		19	\$	36	5	53
	3		20	%	37	6	54
	4		21	&	38	7	55
	5		22	.	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
DISABLES  	8		25	*	42	;	59
ENABLES  	9		26	+	43		60
	10	ESCAPE	27	,	44	=	61
	11		28	-	45		62
	12		29	.	46	?	63
RETURN	13		30	/	47	@	64
SWITCH TO LOWER CASE	14		31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
D	68		97		126		155
E	69		98		127		156
F	70		99		128		157
G	71		100		129		158
H	72		101		130		159
I	73		102		131		160
J	74		103		132		161
K	75		104		133		162
L	76		105		134		163
M	77		106		135		164
N	78		107		136		165
O	79		108		137		166
P	80		109		138		167
Q	81		110		139		168
R	82		111		140		169
S	83		112		141		170
T	84		113		142		171
U	85		114		143		172
V	86		115		144		173
W	87		116		145		174
X	88		117		146		175
Y	89		118		147		176
Z	90		119		148		177
[	91		120		149		178
£	92		121		150		179
]	93		122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
	184		186		188		190
	185		187		189		191

**CODES**            **192-223**  
**CODES**            **224-254**  
**CODE**              **255**

**SAME AS**            **96-127**  
**SAME AS**            **160-190**  
**SAME AS**            **126**

## APPENDIX I: SCREEN DISPLAY CODES

The following chart lists all of the characters built into the Commodore character sets. It shows which numbers should be POKEd into screen memory (locations 1024-2023) to get a desired character. Also shown is which character corresponds to a number PEEKed from the screen.

Two character sets are available, but only one set at a time. This means that you cannot have characters from one set on the screen at the same time you have characters from the other set displayed. The sets are switched by holding down the **SHIFT** and **C** keys simultaneously.

From BASIC, PRINT CHR\$(142) will switch to upper case/graphics mode and PRINT CHR\$(14) switches to upper/lower case mode.

Any number on the chart may also be displayed in REVERSE. The reverse character code may be obtained by adding 128 to the values shown.

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	T	t	20	(		40
A	a	1	U	u	21	)		41
B	b	2	V	v	22	*		42
C	c	3	W	w	23	+		43
D	d	4	X	x	24	,		44
E	e	5	Y	y	25	-		45
F	f	6	Z	z	26	.		46
G	g	7	[		27	/		47
H	h	8	£		28	0		48
I	i	9	]		29	1		49
J	j	10	↑		30	2		50
K	k	11	←		31	3		51
L	l	12	SPACE		32	4		52
M	m	13	!		33	5		53
N	n	14	"		34	6		54
O	o	15	#		35	7		55
P	p	16	\$		36	8		56
Q	q	17	%		37	9		57
R	r	18	&		38	:		58
S	s	19	'		39	;		59



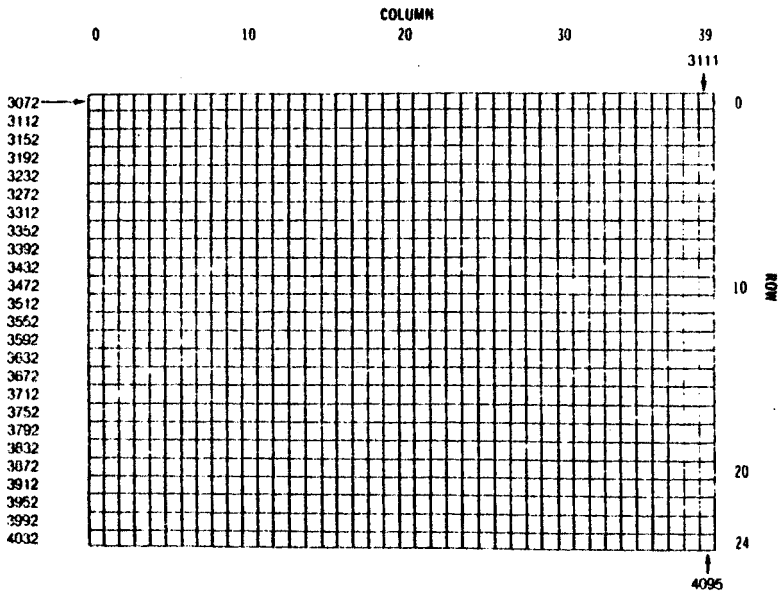
SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
<		60		T	84			108
=		61		U	85			109
>		62		V	86			110
?		63		W	87			111
		64		X	88			112
	A	65		Y	89			113
	B	66		Z	90			114
	C	67			91			115
	D	68			92			116
	E	69			93			117
	F	70			94			118
	G	71			95			119
	H	72	SPACE		96			120
	I	73			97			121
	J	74			98			122
	K	75			99			123
	L	76			100			124
	M	77			101			125
	N	78			102			126
	O	79			103			127
	P	80			104			
	Q	81			105			
	R	82			106			
	S	83			107			

Codes from 128-255 are reversed images of codes 0-127.

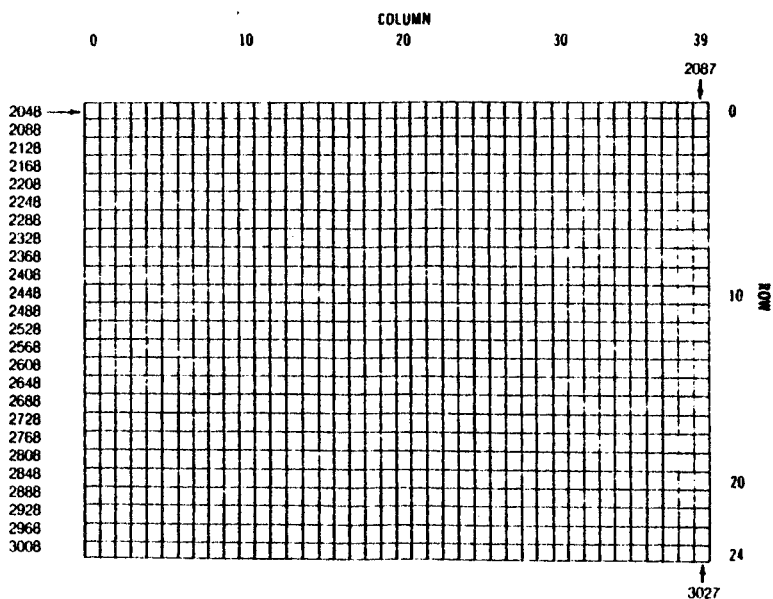
## APPENDIX J: SCREEN AND COLOR MEMORY MAPS

The following charts list which memory locations control placing characters on the screen, and the locations used to change individual character colors, as well as showing character color codes.

### SCREEN MEMORY MAP



## COLOR MEMORY MAP



The actual values to change a character's color are:

- |          |                 |
|----------|-----------------|
| 1 BLACK  | 9 ORANGE        |
| 2 WHITE  | 10 BROWN        |
| 3 RED    | 11 YELLOW-GREEN |
| 4 CYAN   | 12 PINK         |
| 5 PURPLE | 13 BLUE-GREEN   |
| 6 GREEN  | 14 Light BLUE   |
| 7 BLUE   | 15 DARK BLUE    |
| 8 YELLOW | 16 Light GREEN  |

The luminance of the color is selected by multiplying the luminance value (0-7) by 16, and added to the color. To make the character flash, add 128 to the color value also.

## APPENDIX K: DERIVING MATHEMATICAL FUNCTIONS

Functions that are not intrinsic to BASIC 3.5 may be calculated as follows:

FUNCTION	BASIC EQUIVALENT
SECANT	$\text{SEC}(X) = 1/\text{COS}(X)$
COSECANT	$\text{CSC}(X) = 1/\text{SIN}(X)$
COTANGENT	$\text{COT}(X) = 1/\text{TAN}(X)$
INVERSE SINE	$\text{ARCSIN}(X) = \text{ATN}(X/\text{SQR}(-X^2 + 1))$
INVERSE COSINE	$\text{ARCCOS}(X) = -\text{ATN}(X/\text{SQR}(-X^2 + 1)) + \pi/2$
INVERSE SECANT	$\text{ARCSEC}(X) = \text{ATN}(X/\text{SQR}(X^2 - 1))$
INVERSE COSECANT	$\text{ARCCSC}(X) = \text{ATN}(X/\text{SQR}(X^2 - 1)) + (\text{SGN}(X) - 1) * \pi/2$
INVERSE COTANGENT	$\text{ARCOT}(X) = \text{ATN}(X) + \pi/2$
HYPERBOLIC SINE	$\text{SINH}(X) = (\text{EXP}(X) - \text{EXP}(-X))/2$
HYPERBOLIC COSINE	$\text{COSH}(X) = (\text{EXP}(X) + \text{EXP}(-X))/2$
HYPERBOLIC TANGENT	$\text{TANH}(X) = \text{EXP}(-X)/(\text{EXP}(X) + \text{EXP}(-X))$
HYPERBOLIC SECANT	$\text{SECH}(X) = 2/(\text{EXP}(X) + \text{EXP}(-X))$
HYPERBOLIC COSECANT	$\text{CSCH}(X) = 2/(\text{EXP}(X) - \text{EXP}(-X))$
HYPERBOLIC COTANGENT	$\text{COTH}(X) = \text{EXP}(-X)/(\text{EXP}(X) - \text{EXP}(-X))$
INVERSE HYPERBOLIC SINE	$\text{ARCSINH}(X) = \text{LOG}(X + \text{SQR}(X^2 + 1))$
INVERSE HYPERBOLIC COSINE	$\text{ARCCOSH}(X) = \text{LOG}(X + \text{SQR}(X^2 - 1))$
INVERSE HYPERBOLIC TANGENT	$\text{ARCTANH}(X) = \text{LOG}((1 + X)/(1 - X))/2$
INVERSE HYPERBOLIC SECANT	$\text{ARCSECH}(X) = \text{LOG}((\text{SQR}(-X^2 + 1) + 1)/X)$
INVERSE HYPERBOLIC COSECANT	$\text{ARCCSCH}(X) = \text{LOG}((\text{SGN}(X) * \text{SQR}(X^2 + 1))/X)$
INVERSE HYPERBOLIC COTANGENT	$\text{ARCCOTH}(X) = \text{LOG}((X + 1)/(X - 1))/2$

## APPENDIX L: PROGRAMS TO TRY

```
5 GRAPHIC 3, 1: GRAPHIC 0, 1
10 INPUT "SHOULD I CLEAN UP MY MESS"; A$
20 INPUT "SHOULD I ROTATE"; B$
30 INPUT "SHOULD I VARY MOTION"; C$
40 INPUT "SHOULD I PICK THE START"; D$
50 IF A$="Y" THEN DIM A(3,200)
60 DEF FNA(X)= INT( RND(1) * X)
70 IF D$="Y" THEN X1=FNA(80)+80: X2=FNA(80)+80: Y1=FNA(100)+100
75 IF D$="Y" THEN Y2=FNA(100)+100
80 IF D$<>"Y" THEN X1=80: X2=80: Y1=100: Y2=100
90 GRAPHIC 3: FOR L=1 TO 3: COLOR L, FNA(15)+2, FNA(8): NEXT
100 IF C1<1 THEN COLOR FNA(3)+1, FNA(15)+2, FNA(8): C1= FNA(40)+20
110 IF C2<>0 THEN 140: ELSE XA=FNA(11)-5: XB=FNA(11)-5: YA=FNA(13)-6
115 YB=FNA(13)-6
120 IF C$="Y" THEN C2=FNA(10)+5
130 IF B$="Y" THEN XB=-XA: YB=-YA
140 IF C3<1 THEN C=FNA(3)+1: C3=FNA(10)
145 IF A$="Y" THEN DRAW 0, A(0,P), A(1,P) TO A(2,P), A(3,P)
150 X1= X1+ XA: X2= X2+ XB: Y1= Y1+ YA: Y2= Y2+ YB
160 IF X1<0 OR X1>159 THEN XA= -XA: X1= X1+XA
170 IF X2<0 OR X2>159 THEN XB= -XB: X2= X2+XB
180 IF Y1<0 OR Y1>199 THEN YA= -YA: Y1= Y1+YA
190 IF Y2<0 OR Y2>199 THEN YB= -YB: Y2= Y2+YB
200 DRAW C, X1, Y1 TO X2, Y2
210 IF A$="Y" THEN A(0,P)= X1: A(1,P)=Y1: A(2,P)=X2: A(3,P)=Y2: P= P+1
215 IF A$="Y" THEN IF P>200 THEN P=0
220 C1= C1-1: C2= C2-1: C3= C3-1: GOTO 100
```

## SOUND EFFECTS

### Wolf Whistle

```
10 VOL7
20 FORL=400TO800STEP20
30 SOUND1,L,3:NEXT
40 FORL=300TO600STEP40
50 SOUND1,L,3:NEXT
60 FORL=600TO300STEP-40
70 SOUND1,L,3:NEXT
```

### Computer Maniac

```
10 VOL7
20 FORL=1TO100
30 SOUND1,INT(RND(0)*500)+400,4
40 NEXT
```

## Telephone

```
10 VOL7
20 FORL=1TO5
30 FORM=1TO60
40 SOUND1,466,1
50 SOUND1,1020,1
60 NEXT
70 FORZ=1TO2000:NEXT
80 NEXT
```

## Busy Signal

```
10 VOL7
20 FORL=1TO15
40 SOUND1,466,20
50 SOUND1,1020,15
80 NEXT
```

## Bubbles

```
10 VOL7
20 GRAPHIC1,1
30 FORM=1TO50
40 GOSUB80
50 SOUND1,900-R*20,(YR/2)+50
60 CIRCLE1,X,Y,R,YR
70 NEXT:GRAPHIC0:END
80 X=INT(RND(0)*280)+20
90 Y=INT(RND(0)*160)+20
100 R=INT(RND(0)*40)+5
110 YR=R/1.3
120 RETURN
```

## Zap Beam

```
10 VOL7
20 FORM=1TO 20
30 FORL=900TO850STEP-10
40 SOUND1,L,1
50 NEXT
60 FORL=850TO900STEP10
70 SOUND1,L,1
80 NEXT
100 NEXT
```

## Music Lines

```
10 VOL7
15 X1=0:Y1=0
20 GRAPHIC1,1
30 GETA$:IFA$=" "THENGRAHICO:END
40 GOSUB80
45 FORL=1TODSTEP2
50 SOUND1,X*3,5
55 SOUND2,Y*3,5
60 DRAW1,X,Y
65 X=X+2*DX:Y=Y+2*DY
70 NEXT:GOTO30
80 X=X1:X1=INT(RND(0)*280)+20
90 Y=Y1:Y1=INT(RND(0)*160)+20
100 A=X1-X:B=Y1-Y:D=SQR(A*A+B*B)
110 DX=A/D:DY=B/D
120 RETURN
```

## **APPENDIX M: BOOK LIST**

### **BOOKS FOR COMMODORE PRODUCTS**

#### **Commodore Bookware**

- VIC 20 Programmer's Reference Guide
- Commodore 64 Programmer's Reference Guide
- Mastering Your VIC 20
- Four VIC 20 Computer Books:
  - VIC Revealed, Nick Hampshire
  - VIC Games, Nick Hampshire
  - VIC Graphics, Nick Hampshire
  - Stimulating Simulations for the VIC, C. W. Engel

#### **COMPUTE! Books**

- Programming The PET/CBM, R. West
- Machine Language For Beginners
- COMPUTE!'s First Book of VIC Games
- Creating Arcade Games On The VIC
- COMPUTE!'s First Book of Commodore 64 Games
- COMPUTE!'s Reference Guide To 64 Graphics
- Creating Arcade Games On The 64

#### **DATAMOST INC.**

- Kids & The VIC, Edward H. Carlson

#### **Dell Publishing Co., Inc.**

- Games for Your VIC 20, Alastair Gourlay

#### **Dilithium Press**

- A PET for Kids, Sharon Boren
- A PET in the Classroom: Activity Workbook
- More Than 32 BASIC Programs for the Commodore 64, Tom Rugg, Phil Feldman, Gene Moore
- 32 BASIC Programs for the PET Computer

#### **ELCOMP Publishing, Inc.**

- Tricks for VICs, Sam D. Roberts
- More on the 64, H. C. Wagner
- The Great Book of Games (for the Commodore 64)



**Hayden Book Co.**

The 6502 Software Gourmet Guide & Cookbook, Robert Findley  
I Speak BASIC to My VIC, Aubrey B. Jones, Jr.

**Reston Publishing Company, Inc.**

VIC BASIC: A User-Friendly Guide, Ramon Zamora, Don Inman, Bob  
Albrecht, Dymax  
25 Advanced Games for the PET/CBM, Larry Hatch

**Sybex Computer Books**

Your First VIC 20 Program, Rodney Zaks  
Programming the 6502, Rodney Zaks  
The VIC 20 Connection, James W. Coffron

## INDEX

- Abbreviations for BASIC statements 42, 130-132
- Animation 52-54
- Arrays 100, 112
- Assigning data
  - DATA ... READ statements 111, 122
  - INPUT 43-45, 86, 116
  - GET 114
  - LET 117
- AUTOMatic renumbering 103
- BACKUP command 103
- BASIC
  - abbreviations 42, 130-132
  - commands 99-108, 130-132
  - converting to Commodore BASIC 139
  - functions 99, 125-132
  - statements 99, 109-125, 130-132
- Branching programs
  - GOSUB 89-90, 115
  - GOTO 17, 115
  - ON ... GOSUB/GOTO 90, 118
  - RETURN 123
- Built-in software 14, 30
- Calculations
  - addition 38
  - decimals 39
  - division 38
  - execution order 42
  - exponentiation 41
  - fractions 39
  - mathematical operators 38, 102
  - multiplication 38
  - parentheses 42
  - PRINT statement 38-41
  - relational operators 38, 102
  - scientific notation 42
  - subtraction 38
- Cartridges
  - installing 4, 30-31
  - loading 30-31
- Cassette tapes
  - recorder 32-33
  - LOADing 32, 106
  - SAVEing 33, 108
  - software 32
- CHAR statement 60, 82, 109
- CHR\$ codes 143-145
- CHR\$ function 92, 128
- Clearing
  - CLR command 110, 116
  - graphics modes 57, 116
  - graphics screen 57, 116
  - memory 3
  - screen 11, 23
- CLOSE statement 110
- CLR command 110, 116
- CMD command 110
- Color
  - areas 54
  - background 54, 83
  - border 54, 83
  - changing 11-12, 27, 48, 54-55, 83
  - COLOR command 17, 55, 111
  - filling areas 61, 109, 119
  - keys 11-12
  - luminance 55, 83
  - memory map 149
  - PAINT 65
  - screen 54-55
  - source 54-55, 109, 111
- Commands (See BASIC commands)
- Commas
  - in PRINT statements 25-26
  - separating numbers 39
  - vs. semicolons 25-26
- Commodore key 11-12, 48
- Connecting the computer 3-7
- CONT command 104
- Control (CTRL) key 11, 48
- COPY statement 111
- Copying diskettes 103
- Cursor
  - controlling movement 10, 82
  - cursor keys 10, 20
  - in PRINT statements 43
- Debugging
  - CONT 104
  - DS\$ 34, 36
  - STOP 124
  - RESUME 123
  - TRAP 91, 124
- DEF FN statement 112, 151
- Defining function keys 14-15, 92, 105
- Defining functions in programs 112, 151
- Delete
  - command 104
  - editing 13, 21
  - files from a diskette 108
  - key 10, 21, 41
  - letters in a word 10, 21
  - lines in a program 19, 104
- DIM statement 112
- Dimensioning an array 112
- Direct mode (See Immediate mode)
- DIRECTORY command 36, 104
- Diskettes
  - COPY statement 111
  - DIRECTORY command 36, 104
  - disk drives 4, 34

- DLOAD command 34, 104
- DS\$ function 34, 36, 96-98, 100
- duplicating 103, 111
- formatting 35, 105
- HEADER command 35
- listing a directory 36, 104
- loading 34
- SAVEing 36, 105
- table of contents 36, 104
- DLOAD 34, 104
- DOPEN 34
- DS\$ 34, 36, 96-98, 100
- DSAVE 36, 105
- Duplicating diskettes 103
- Editing
  - INSert key 10, 21, 41
  - INSert mode 13, 21
  - DELeTe key 10, 21, 41
  - DELeTe command 104
  - RENUMBER command 103, 107
- END statement 113
- Errors
  - Debugging statements 91
  - Disk errors 34, 36, 96-98
  - Messages explained 94-98
- ESCAPE functions 13
- ESCAPE key 13
- EXP function 125
- Flash mode
  - accessing 12
  - keys 12
  - using in PRINT statements 27-28
- FOR ... TO ... STEP 46, 76-77, 114
- Formatting diskettes 35, 105
- Formatting output
  - PRINT USING 120-121
  - PUDEF 122
  - print zones 25-26
  - punctuation 25-26, 83, 86
- Function keys 14-15, 92, 105
- GET statement 114
- GETKEY statement 114
- GET# statement 115
- GOSUB 89-90, 115
- GOTO 17, 115
- Graphics
  - BOX 61, 109
  - CIRCLE 62-65, 80-82, 110
  - clearing 57
  - COLOR 17, 55
  - DRAW 58, 113
  - exercises 49-66
  - GRAPHIC command 57, 116
  - high resolution 56-65
  - keys 13-14, 49-53
  - modes 9-11, 57, 65-66
  - multicolor modes 57, 65-66
  - PAINT 61, 119
  - printing graphic symbols 9-11, 22, 85
  - SCNCLR 57, 80-81
  - uppercase/graphic mode 9-11
- HEADER command 35, 105
- HELP key 14-15, 105
- High resolution graphics 56-65
- IF ... THEN ... ELSE 116
- Immediate mode 41
- INPUT statement 43-45, 86, 116
- Insert
  - editing 13, 21
  - key 10, 21, 41
  - mode 13, 21
- Insert mode
  - accessing 10, 13, 21
  - key 10, 21
- Installing the computer 3-7
- INSTR function 88, 126
- INT function 79, 126, 127
- Integer variables 43, 99-100
- Joy sticks 4, 126
- KEY command 14-15, 92, 105
- Key redefining 14-15, 92, 105
- Keyboard
  - function keys 14-15
  - special keys 9-14
- LEFT\$ function 88-89, 128
- LET statement 117
- LIST statement 19, 106
- Loading
  - cartridges 30-31
  - cassettes 32, 106
  - diskettes 34, 104
  - DLOAD command 34, 104
  - LOAD command 32, 106
- LOCATE
  - command 103, 117
- Loops
  - DO ... LOOP ... WHILE/UNTIL 113
  - GOSUB 89-90, 115
  - GOTO 17, 115
  - FOR ... TO ... STEP 46, 76-77, 114
  - IF ... THEN ... ELSE 116
  - ON ... GOSUB/GOTO 90, 118
- Luminance 55, 83, 149
- Machine Language Monitor 117, 133-138
- Mathematical functions 150
- Mathematical operators 38, 102
- Memory maps 140-141, 148-149
- MID\$ function 88, 128, 139
- Modems 5, 36
- Modes
  - flash 12
  - graphics 54, 57

- Insert 13, 21
- multicolor 57, 65-66
- quote 13
- reverse 12-13, 27-28
- uppercase/graphic 9-11
- upper/lowercase mode 9-11
- Monitor
  - connecting to computer 6
  - machine language 117, 133-138
- Music
  - duration of notes 70
  - SOUND statement 69-70, 142
  - voices 69
  - volume 69-70, 125
- NEW command 17, 107
- NEXT statement 117
- Numbers
  - calculating 38-44
  - exponentiation 41
  - execution order in multiple calculations 42
  - mathematical operators 38, 102
  - $\pi$  39
  - relational operators 38, 102
  - signs (+ and -) 38
- ON ... GOSUB 90, 118
- ON ... GOTO 118
- OPEN command 118
- PEEK function 126
- Pi 39, 129
- Pixel cursor
  - in graphics modes 103, 117
  - LOCATE statement 103, 117
  - positioning 103, 117, 126
- POKE statement 119
- PRINT
  - calculations 38
  - displaying messages 17, 119
  - formatting output 25-26, 60, 82, 85-86, 109, 119
  - in immediate mode 41
  - in program mode 41
  - print zones 25-26, 85
  - punctuation 25-26
- PRINT USING 120-121
- PRINT# USING 120-121
- Print zones 25-26
- Program flow control (See Loops)
- Programming
  - BASIC commands 99-108
  - BASIC functions 99, 125-132
  - BASIC statements 99, 109-125
  - function keys 14-15, 92, 105
  - machine language monitor 133-138
  - mode 41
- PUDEF 122
- Quote mode
  - accessing 9, 20
  - keys 9
  - using in PRINT statements 17, 20, 23
- Random numbers 78-79, 127
- READ statement 122
- Relational operators 38, 102
- REM statement 122
- RENAME command 122
- RENUMBERING program lines 103, 107
- Reset button 3
- RESTORE 122
- RESUME 123, 124
- Resuming program display 11, 123
- RETURN 9, 123
- Reverse mode
  - accessing 12
  - keys 12-13
  - using in PRINT statements 27-28
- RIGHT\$ function 88, 128
- RND function 78-79, 127
- RS-232 port 4
- RUN command 10, 107
- SAVE command 33, 108
- Saving programs
  - cassettes 33, 108
  - disk errors 34, 36, 96-98
  - diskettes 36
  - DSAVE 36, 105
  - SAVE 33, 108
- SCNCLR statement 57, 80-81
- SCRATCH command 108
- Screen
  - clearing 11, 23, 57
  - clearing graphics modes 57, 80-81
  - display codes 146-147
  - LIST command 19, 106
  - memory map 148
  - program display 25
  - resuming display 11, 123
  - size 23-24
  - slowing display 11
  - windowing 13
- Screen area numbers chart 148
- Semicolons
  - in PRINT statements 25-26, 84
  - vs. commas 25-26
- Setting up the computer 3-7
- Slowing program display 11
- Software
  - built-in 14, 30
  - cartridges 30
  - cassettes 30
  - diskettes 30
  - LOADING 30-34
  - saving your own 33, 36
- Sound effects 69, 151-153
- SPC function 129
- STOP statement 124
- Stopping program display 10

- String functions 128
- Subroutines 89-90, 115, 123
- SYS statement 124
- TAB function 129
- Text
  - in graphics 60
  - in PRINT statements 17, 39
  - string functions 128
- TI\$ function 100
- TRAP statement 91, 124
- Troubleshooting 7
- TV
  - antenna types 5
  - channel selection 3, 6
  - hookup 5-7
  - switch box 2-6
- Uppercase/graphics modes
  - accessing 11
  - printing graphics 9-11
  - SHIFT key 9
- Upper/lowercase graphics
  - accessing 11
  - printing graphics 9-11
  - SHIFT key 9
- Variables
  - floating point 43, 99-100
  - integer 43, 99-100
  - text string 43, 99-100
  - types 43, 99-100
  - see also Assigning data
- VERIFY command 108
- Voices 69
- Volume 69-70, 125
- WAIT statement 125
- Windowing 13

