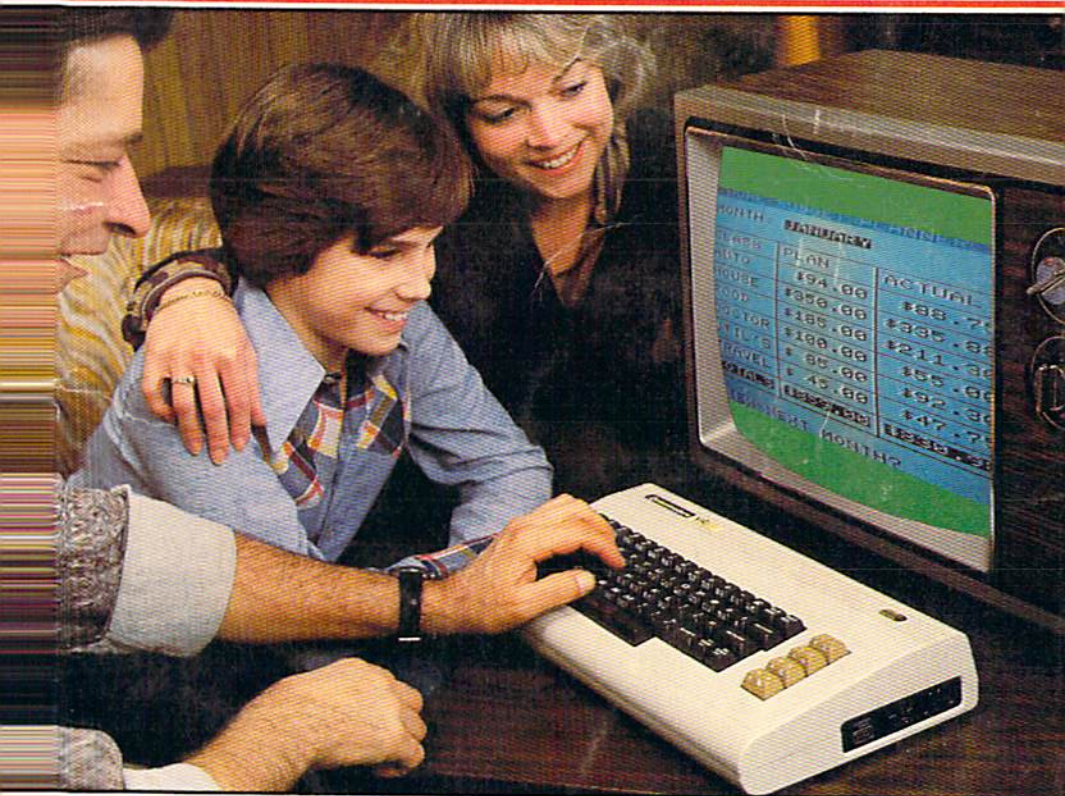


Personal Computing on the

VIC-20



a friendly computer guide

commodore
COMPUTER

Published by
COMMODORE INTERNATIONAL, LTD.
Computer Systems Division
950 Rittenhouse Road
Norristown, PA 19403
U.S.A.

Copyright © 1981 Commodore International and Avalanche Productions. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Commodore International.

PERSONAL
COMPUTING
ON THE
VIC20

A friendly computer guide

COMMODORE INTERNATIONAL
AVALANCHE, INC.

PREFACE

You are about to meet a friendly computer! Friendly in price, friendly in size, friendly to use and learn on and experience. Most important — you don't have to be a computer programmer, or even a typist, to use it!

If you're a first time computerist, this manual will provide an excellent introduction to computing. Unlike most instruction manuals, you don't have to read through this whole book to get to the "good stuff." After reading Chapter 1 (GETTING STARTED), you can go directly to a chapter that interests you and start reading. If you're interested in animation turn to Chapter 4. If you like music, try Chapter 5.

The first page of each chapter has a sample program to start you off. Just type the program exactly as shown ("Try Typing This Program") and see what happens. The rest of the chapter explains what you did, and shows you how to do more. Chapter 7 summarizes some important programming concepts, and explains the techniques used in the sample programs.

If you're an experienced programmer, you can use the VIC like any microcomputer. Familiarity with Commodore computers will help, since the BASIC and graphics are nearly identical to those used in the PET/CBM. Advanced reference material and programming information are included in the Appendix. For more sophisticated programming, see the VIC PROGRAMMERS REFERENCE GUIDE, available from your Commodore dealer.

If you're a noncomputerist and have no interest in programming, per se, you should check out the VIC's growing library of plug-in cartridges and program tapes. VIC cartridges plug directly into the back of the console and work automatically. Programs are also provided on cassette tapes for use with the Commodore Cassette Tape Recorder.

Cartridges and tapes include exciting "arcade-type" games such as "VIC INVADERS" as well as educational programs to help you develop special skills, and home utility programs to help you solve problems and perform calculations.

Peripherals and accessories for the VIC include the VIC tape cassette recorder, single disk drive, telephone modem and printer, to name a few. (See Appendix A)

Computers are becoming an increasingly important part of our everyday lives — in our homes, at school and in business. Those who become familiar with computers now will have an important advantage in the coming months and years. The VIC not only introduces you to the world of computing, but also gives you the features and flexibility you need to expand that world.

Enjoy your new world!

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
	PREFACE.....	II
	UNPACKING AND CONNECTING THE VIC20.....	V
One	GETTING TO KNOW YOUR VIC	1
	• Getting Started	3
	• Your First Computer Program.....	7
Two	USING THE SCREEN AND KEYBOARD	11
	• Your First Graphic Character.....	14
	• A Tour of the VIC20 Keyboard.....	17
	• Printing on the Screen.....	21
	• The VIC20 Calculator.....	24
	• Introduction to Color	25
Three	COLOR AND GRAPHICS	27
	• Programming in Color	30
	• The VIC Color Keys.....	32
	• Changing Screen and Border Colors	34
	• Screen & Border Color Combinations	37
	• Coloring the Screen.....	37
	• Screen Locations	39
	• Random Colors.....	40
	• Combining Sound and Color.....	45
	• Keyboard Graphics.....	47
	• Graphics in Headlines and Titles.....	48
Four	ANIMATION.....	51
	• Flying Birds.....	53
	• Bouncing Ball	57
	• Controlling the Cursor.....	60
	• Animating with POKEs and PEEKs	61

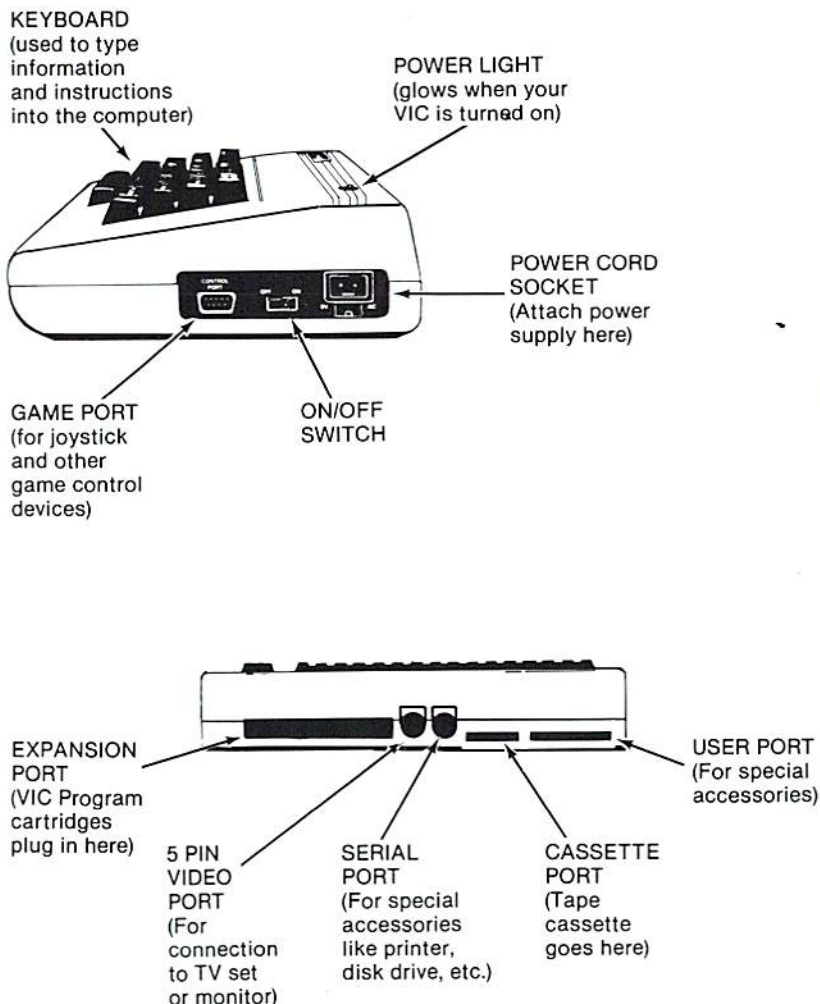
Five	SOUND AND MUSIC	67
	• Making Music	69
	• The Four Voices of VIC.....	71
	• The White Noise Generator.....	74
	• Using the VIC as a Piano	76
	• Playing Songs.....	78
	• A Few Words About Poke	80
Six	CONVERSING WITH YOUR VIC	81
	• What's Your Name?	83
	• Introducing Variables.....	86
	• Choose a Note.....	88
	• The GET Statement	89
Seven	INTRODUCTION TO PROGRAMMING	93
	• Your First BASIC Commands	95
	• Random Numbers	103

	TITLE	PAGE
APPENDIX		105
A.	VIC System Accessories.....	106
B.	Working With Tape Cassettes	109
C.	VIC BASIC Vocabulary.....	113
D.	BASIC Command Abbreviations	133
E.	Screen and Border Color Combinations.....	134
F.	Musical Notes	135
G.	Sample Sound Effects.....	136
H.	Screen Display Codes	139
I.	Screen Memory Map	143
J.	Ascii and Character (\$) Codes.....	145
K.	Deriving Mathematical Functions	148
L.	Pinouts for Input/Output Devices	150
M.	VIC Programs to Try	153
N.	Error Messages	160
INDEX		162

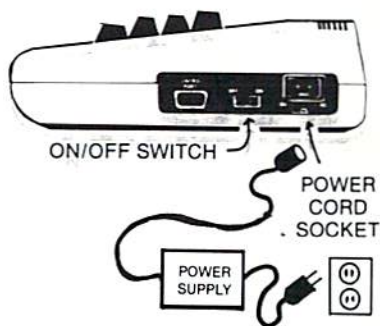
UNPACKING AND CONNECTING THE VIC 20

Welcome to computing! The following step-by-step instructions show you how to unpack the VIC, connect it to your television set and make sure it's working properly.

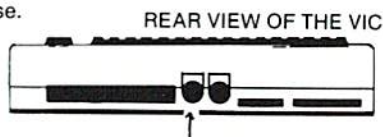
Let's begin by taking a quick look at the VIC 20:



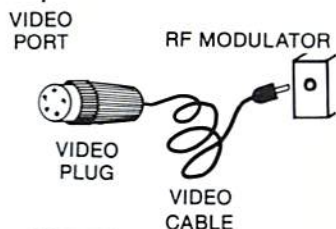
1. Check the contents of your VIC container. You should find the following items;
 - VIC 20 Personal Computer
 - Power Supply (large box with 2 cords coming out of it)
 - RF Modulator (small metal box) and short cable*
 - Video Cable
2. You will need 2 *electrical outlets* (sockets) — one for the VIC and one for your television set.
3. Position the *VIC and Television set* so you can use the keyboard comfortably while viewing the television screen...ideally, a tabletop or desk.
4. Find the *ON/OFF switch* on the right hand side of the VIC. Make sure it's in the "OFF" position.



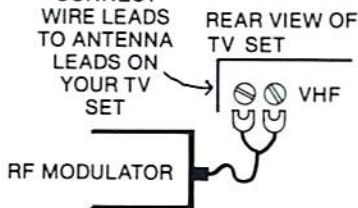
5. There are two cords coming out of the power supply box. Plug the power *supply cord* into an electrical outlet and plug the other cord into the *power cord socket* on the side of the VIC. NOTE: The power supply remains "on" while plugged in so you should unplug it when not in use.



6. Connect the *video cable* to the *back* of the VIC and to the *RF Modulator box*, as shown. Make sure to connect it to the *video port* and *not* to the 6-pin serial port, which is next to it.



7. Connect the *RF Modulator* to your television set. For this you'll need a *screwdriver*. The short *TV connector cord* runs from the RF Modulator box to the 2 *VHF Antenna leads* on the back of your television. Simply connect the two wires to the VHF leads and tighten the screws firmly.*

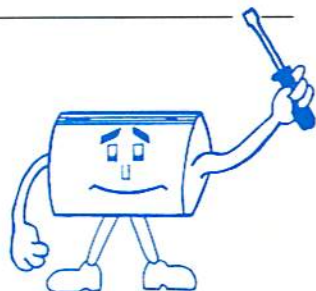


8. Turn on the TV set.

Note: Some VICs are provided with a "switchbox" which attaches between the RF modulator and TV set. The switchbox contains a switch with settings for "computer" and "TV" and should be set to "computer" when using the VIC.

TROUBLESHOOTING CHART

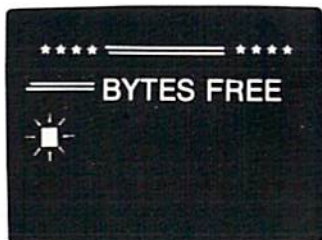
SYMPTOM	CAUSE	REMEDY
NO PICTURE (POWER LIGHT OFF)	VIC not "on"	Make sure power switch is in "on" position
	VIC not plugged in	Check power socket next to power switch
	Power supply not plugged in	Check connection with wall outlet
	Bad fuse in VIC	Take VIC to your Authorized Commodore Service Center for replacement of fuse*
NO PICTURE (POWER LIGHT ON) (Try turning VIC off for a few seconds, then back on)	TV on wrong channel	Check Channel 3 & 4 for picture
	Incorrect hookup	VIC hooks up to "VHF" terminals on TV
	Modulator not plugged in	Check connection at 5-pin Video Port
	Modulator on wrong channel	Flip switch on Modulator
	Video cable not connected	Check connection on modulator
PICTURE WITHOUT COLOR (TRY TV & MODULATOR ON CHANNEL 3 & 4)	Poorly tuned TV	Retune TV
PICTURE WITH POOR COLOR	Bad color adjustment on TV (see "picture without color")	Adjust color/hue/brightness controls on TV
PICTURE WITH EXCESS BACKGROUND NOISE	TV volume up too high (see "picture without color")	Adjust volume of TV
PICTURE OK, BUT NO SOUND	TV volume too low	Adjust volume of TV



*The VIC uses a 3 amp SLO-BLO fuse.

- 9.** Turn on the VIC (the red power light on the top of the computer should come on). If the power light does not, consult the accompanying troubleshooting chart.
- 10.** There is a switch on the RF Modulator for selecting either Channel 3 or 4. Choose the channel with the weaker reception in your area, and set both the TV and the Modulator to that channel. The fine tuning on your television may need some adjustment.

Here is what you should see on the screen — sometimes it takes a second or two to activate. If you don't get the following display on your screen, turn the computer off, wait a few seconds and turn it on again.



- 11.** Adjusting the color and tint/hue depends on the color controls provided on your television set — naturally, sets with better controls yield better color. Some sets show some colors better than others.
- 12.** If you have trouble with any of these steps, consult the accompanying troubleshooting chart.

NOW...you are ready to start using the VIC.

NOTE: You can use a *monitor* instead of a television set — in which case you can go directly from the VIC to the monitor cable, without the RF Modulator.

**IF ALL ELSE FAILS...CALL THE TOLL-FREE
COMMODORE TECHNICAL HOTLINE...**

1-800-523-5622

1

Getting to Know Your VIC

- Getting Started
- Your First
Computer Program

Try typing this program:

Type this program exactly as shown and see what happens!

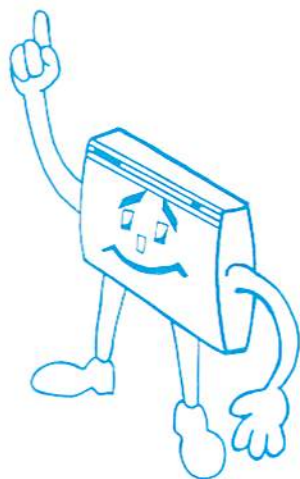
```
1 PRINT "VIC20" RETURN
2 GOTO 1 RETURN
RUN RETURN
```

This line tells the VIC to print what's between the quotation marks.

This line tells the VIC to go back to Line 1 and print it again.

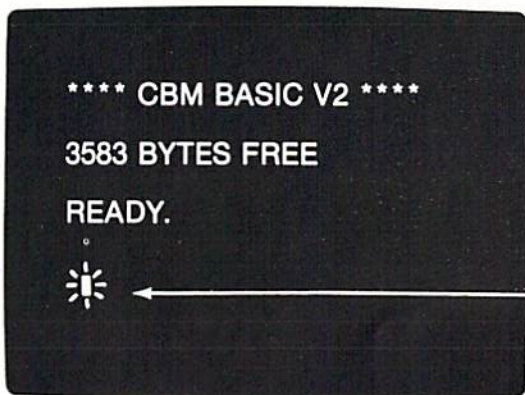
Typing the word RUN makes the program run.

To stop the program, press the **RUN STOP** key.



GETTING STARTED— EXPERIMENT A LITTLE

You made it! Your VIC is aglow with color and ready for you to tell it what to do. The dark blue, blinking rectangle, called the *cursor*, is a signal from the VIC that it is waiting for you.



VIC TIP:

If you type a character on the screen that you don't want, press the



key.

This key will erase the character immediately to the left of the blinking cursor.

Use this key as often as you like to delete unwanted characters.

Now, on with the tour! Begin by pressing the following keys:

P R I N T

See how the *cursor* moves over one position every time you press a key? The cursor tells you where the next character is going to appear on the screen. OK, now find the SHIFT key, it looks like this:



There are two of them, both the same.

Hold the **SHIFT** key down, and while it is being held, press:

"2

You can release the **SHIFT** key after you press the **"2** key. The screen should look like this:



you typed this

Pressing the **"2** key while holding down the **SHIFT** key caused the quotation mark to appear on the screen.* Let's continue. Now press the following keys:

R A I N B O W

Finally, hold down the **SHIFT** key, and press the **"2** key once again. The screen now shows:

Everything you typed is on this line.

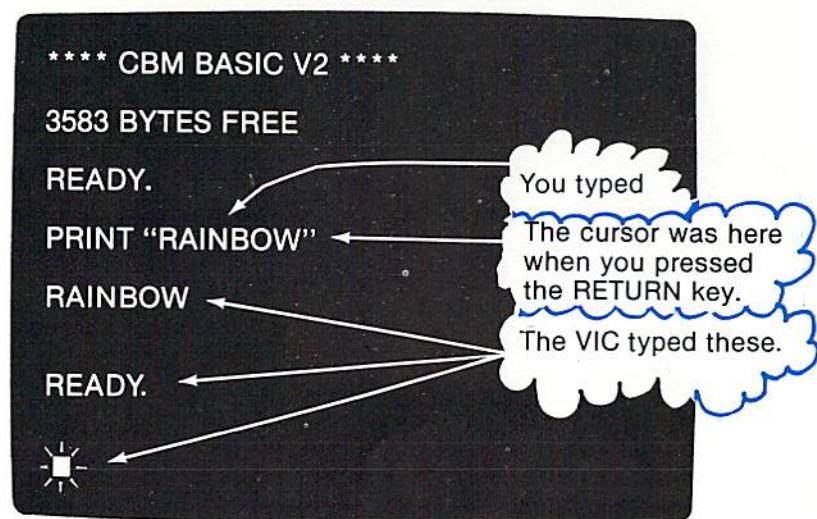


*Note: if the number 2 appeared on the screen instead of the " sign, you didn't hold down the shift key. Hit the **INST DEL** once to erase the 2 and try again.

Look for this key:

RETURN

Press the **RETURN** key and look at the screen.



Pressing the **RETURN** key told the VIC you were finished typing. The VIC then looked at what you typed, recognized that it was being asked to do something (actually, you told it to *print* something). The VIC then **PRINTed** everything between the two quotation marks (RAINBOW).

When the VIC finished **PRINTing** the word RAINBOW, it let you know by displaying the **READY** message and blinking the cursor.

It's your turn. Go ahead and enter some other **PRINT** messages for your VIC to display. Try these or make up your own:

P R I N T " R A I N B O W "

Press **RETURN** here

P R I N T " H E L L O "

P R I N T " K A T H Y "

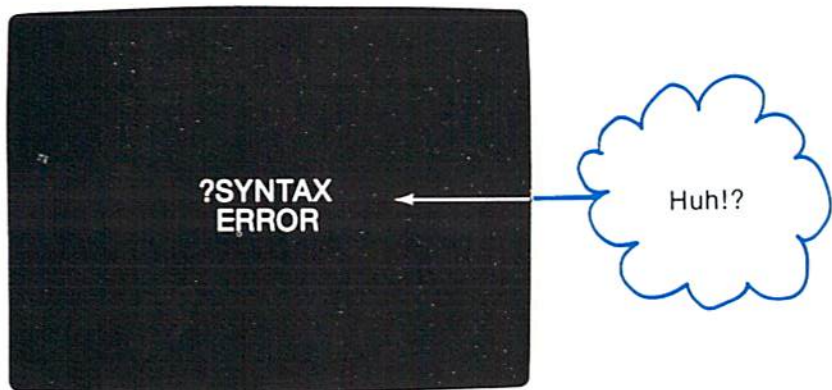
Remember, hold the **SHIFT** down to get this character.

RETURN here

and

RETURN again

Experiment by using characters other than letters between the quote marks: The VIC won't mind. Note: If you misspell the word PRINT, the VIC will let you know by displaying this message on the screen:



Don't worry. *There is absolutely no way you can hurt your VIC by typing at the keyboard* (unless, of course, you're an elephant), but, if you make a mistake, the VIC will help you out by calling attention to the error. These error messages and what they mean are listed in Appendix N. At this point, don't worry about the "SYNTAX ERROR" message. Just keep experimenting.

In no time your screen gets cluttered with all the stuff you've been typing. The VIC has a handy way to CLear up this clutter. To tell the VIC to "clear" the screen, do the following:

Hold down the  key and press

the  key.



IMPORTANT

The screen clears instantly; everything you and the VIC have typed disappears. You are left with a clean white display area and a blue cursor blinking away in the upper left hand corner.

Remember how you told the VIC to do this feat. Clearing up the screen is one of the more frequent commands you'll be using as you get to know your VIC.

YOUR FIRST COMPUTER "PROGRAM"

If your VIC performed well in displaying your messages, no matter how bizarre they might have been, then your computer is probably ready to do just about anything. Let's begin by "entering" your *first computer program*.

STEP 1: Clear the screen by holding down the **SHIFT** key and then pressing the **CLR HOME** key.

STEP 2: Type: **N E W** and press the **RETURN** key.

STEP 3: Type: **1 0 P R I N T " V I C 2 0 " ;** and press **RETURN**.
Use the **SHIFT** key for these.
semi-colon
SPACE

STEP 4: Type: **2 0 G O T O 1 0** and press **RETURN**.

Do not type out the letters of this word —press the space bar (long key on the bottom of the VIC keyboard).

When you finish, the screen looks like this:



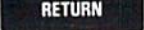
VIC TIP: EDITING MISTAKES IN A PROGRAM

If you make a mistake on a line, you have these *editing* options:

1. You can *retype a line* anytime and the VIC will automatically substitute the new line for the old one. For example, if your program looks like this:

```
10 PRINN "VIC20"  
20 GOTO 10
```

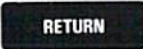




mistake


You can skip down by hitting  a few times and type:

```
10 PRINT "VIC20"
```



Now, the *new line* has replaced the old line and the program will "run". To make sure, type **L I S T**. Replacing lines in a program is also a quick and easy way to *experiment*.

2. You can *erase an unwanted line* by typing the *number* of that line and hitting . The entire line will be erased from memory.
3. You can *edit a line* by using the cursor keys to move to the character(s) in the line of a program you want to change, typing in a program over them and hitting . Note that quotation marks sometimes confuse the VIC—if you get unwanted characters after quote marks, go back to the beginning of the line and type it over.
4. The INST key (get it by typing  ) lets you insert characters by opening up spaces in a word or line you've already typed.
5. The DELETE key (just type ) erases characters immediately to the left of the cursor.

If everything looks all right to you, type the following word, and press  :

R U N

 here

The screen should fill with VIC20. At times, it looks like small animated letters traveling up the screen.

```
VIC20VIC20VIC20VIC20VIC20VIC
IC20VIC20VIC20VIC20VIC20VIC20
C20VIC20VIC20VIC20VIC20VIC20V
20VIC20VIC20VIC20VIC20VIC20V
JVIC20VIC20VIC20VIC20VIC20VI
VIC20VIC20VIC20VIC20VIC20VIC20
C20VIC20VIC20VIC20VIC20VIC20V
20VIC20VIC20VIC20VIC20VIC20V
0VIC20VIC20VIC20VIC20VIC20VI
VIC20VIC20VIC20VIC20VIC20VIC20
VIC20VIC20VIC20VIC20VIC20VIC20
VIC20VIC20VIC20VIC20VIC20VIC20
```

Want to slow down the program? Press the key on the left side of the keyboard marked:

CTRL

If you hold down the **CTRL** key, the program *slows down*.

Amazing! Your VIC is full of wonderful features. Here, with just a single key, you are telling the VIC to reduce how fast it is displaying stuff on the screen.

Yes, but how do you STOP the program? Good question. Look around the keyboard until you find this key:

**RUN
STOP**

Press the **RUN STOP** key. The program should stop, and the message:

BREAK IN 10

READY

should appear on the screen. (Don't worry, you didn't break the VIC—"break" means "stop" in VIC language). Also, the cursor should reappear. Did you notice that it was gone while it was printing?

Now, let's take a look at your program and see if it's still there. Try typing this:

L I S T

RETURN

Your program (lines 10 and 20) will be displayed on the screen. Now type RUN and the program will "run" again.

You have just been introduced to several aspects of the VIC that you will use in many of the later chapters. You have:

- PRINTed messages on the screen.
- CLearRed the screen (SHIFT CLR keys).
- Written your first program (VIC20) and created a moving display.
- Slowed down (ConTRoLed) the program (the CTRL key).
- STOPed the program with the STOP key (RUN/STOP key).
- LISTed the program.
- Learned some easy ways to edit what you type.

As you explore the chapters of this guide, you will find many uses for what you have seen here. Don't worry if you have unanswered questions at this point. Just go ahead and *experiment* and most of your questions will be answered as you go along.

This guide is designed so that you can go directly to *any chapter* that looks interesting to you. You do not have to read each chapter in order to get to know your VIC. Just be sure to start from the beginning of each chapter. You will find that our gradual introduction to each topic makes it easier for you to learn how to create adventures of your own. Enjoy!

2

Using the Screen and Keyboard

- **Your First Graphic Character**
- **A Tour of the VIC 20 Keyboard**
- **Printing on the Screen**
- **The VIC 20 Calculator**
- **Introduction to Color**

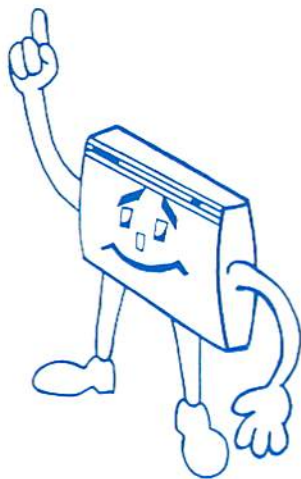
Try typing this program:

Type this program exactly as shown and see what happens!

```
10 PRINT "
    SHIFT CLR
    HOME
20 FOR T = 1 TO 300: NEXT
30 PRINT "your name"
40 FOR T = 1 TO 300: NEXT
50 GOTO 10
```

Type: **R** **U** **N** and hit **RETURN**

To stop the program, press the **RUN STOP** key.



USING THE SCREEN AND KEYBOARD

This chapter assumes that you've read and understood Chapter 1: *Getting to Know Your VIC*. If you have not, go back and read at least the last two sections which show you how to use the keyboard to control what the VIC prints on the screen.

To start, poise yourself before the VIC keyboard and type as follows, including the program line numbers and punctuation marks:

Hold down the **SHIFT** key and press the **CLR HOME** key.

N E W and press the **RETURN** key.

1 P R I N T " H E L L O **SPACE**

SPACE " ;

and press **RETURN**.

2 G O T O 1

and press **RETURN**.

R U N and press **RETURN**.

this means hit the long space bar

When you type RUN, the screen should fill with the word:

HELLO

The words *appear* to be moving up and sideways! Press the CTRL key to slow things a bit. The VIC is PRINTing the message several times near the bottom of the screen. When the screen fills, the contents of the screen are moved *up* to make room for more PRINTing. So, the upward movement is really happening. The "barber pole effect" is an "illusion" caused by the number of characters the VIC is putting in each line.

To stop this program, press this key: **RUN STOP**

Now, it's your turn. Type these two lines:

1 **SPACE** P R I N T " H E L L O YOUR NAME " ;

and press **RETURN**.

R U N and press **RETURN**.

Put your name here.

The semicolon means print everything next to each other.

Wow! Now that you're a TV star, what does your name do on the screen? The "illusion" of movement depends on the number of characters in the message.

Again, when you want to stop the action, press the **RUN STOP** key.

YOUR FIRST GRAPHIC CHARACTER

Clear the screen (hold down **SHIFT** and **CLR HOME**). Now type **SHIFT** AND **S**. You should get a blue *heart* on the

screen. Try it again. You have just typed your first *graphic character*.

Try typing other graphics. Now hold down the **G** key and type some graphics — this is the left side graphics set. Left side graphics are very good for designing business forms, charts and graphs. Typing the **SHIFT** and **G** keys at the same time lets you use upper and lower case letters. See Chapter 3 for an explanation.

SIZE OF THE SCREEN

How big is the screen on the VIC? Let's find out. Do this: Clear the screen and type the following:

N E W and press **RETURN**

1 P R I N T " ♥ " ;

RETURN

To type a *blue heart*, hold down **SHIFT** and press **S**

2 G O T O 1

RETURN

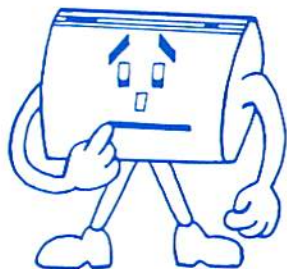
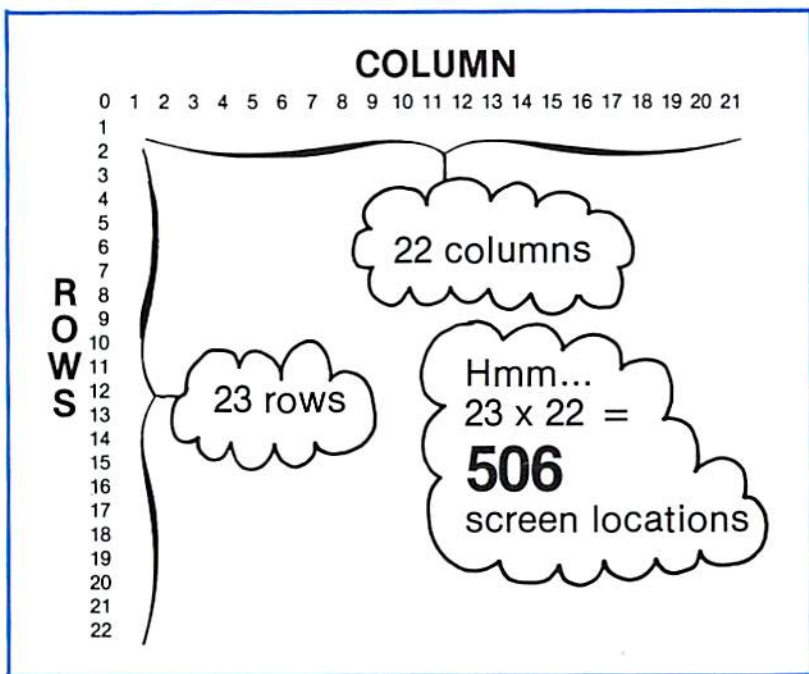
R U N

RETURN

The screen will fill with blue hearts! Count the number of hearts being printed across the screen. There are 22 of them in each row. The VIC has 22 PRINT positions *across* the screen. The positions across the screen are sometimes called *columns*. The VIC has 22 columns.



How many positions are there *down* the screen? Press the **CTRL** key to slow down the PRINTing. Holding the **CTRL** key causes the last four rows to "flash". The VIC has 23 rows down the screen.

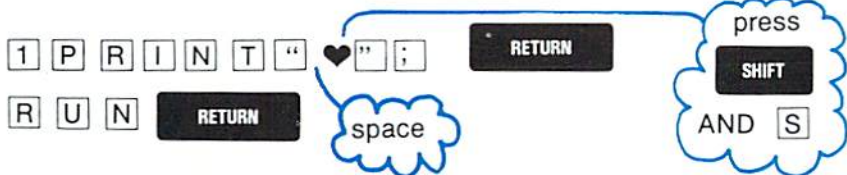


The VIC has 506 places on the screen for characters, letters, symbols, and so forth. You might say that the VIC can juggle 506 characters at the same time. Amazing!

VIC TIP

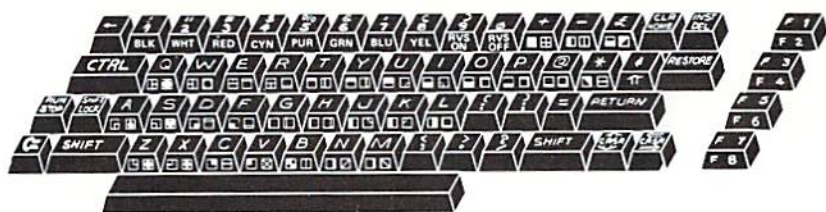
If the VIC has 22 columns across the screen, then any message whose length is an even divisor of 22 (messages 2 characters, 11 and 22 characters in length) causes the VIC to PRINT in nice neat columns. Messages of other lengths spill over to the next line. Test this assumption for yourself.

Stop the VIC's PRINTing of the neat columns of hearts by pressing the **RUN STOP** key. Then enter these two lines into the VIC.



You can change the way information is printed on the screen by putting spaces between the quotation marks. Another way is to use periods (or dots) instead of spaces. Try typing your name and 3 periods in the program at the beginning of this chapter.

A TOUR OF THE VIC20 KEYBOARD



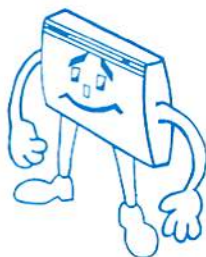
You have now used the keyboard to create and PRINT messages, put graphics characters on the screen, control the flow of what the VIC is doing (**CTRL** and **RUN STOP**), and possibly edit what you have typed (**INST DEL**).

Time now for an extended tour of the keyboard and what it can do for you. Consult the diagram above for the powerful and versatile set of VIC20 keys.

This is a "reset" key.

If you type the RUN/STOP key and the **RESTORE** key together, you completely reset the computer as if you just turned it on...with the benefit that any programs you had in the memory are retained and can be listed or run from the start.

Here is how
you can use
these keys:



SHIFT

SHIFT keys — The VIC keyboard is just like a typewriter and has two *shift* keys and a SHIFT LOCK key. The SHIFT key is used with other keys to type graphics characters and perform operations like clearing the screen.

CLR HOME

CLR-HOME key — Press this key and the cursor moves to the top left-hand corner of the screen (the “home” position). If you hold down the SHIFT key and press this key, the cursor still moves to the home position, but you also clear the screen.



CRSR keys — With the VIC, you can easily move the cursor up, down and sideways. The CRSR keys have a *repeat* feature that keeps the cursor moving until you release the key. Each key has a set of arrows that tell you the directions the key controls—up and down or sideways. To move the cursor down or to the right, you simply press the appropriate key. To move up or to the left, you must hold down the SHIFT key while pressing the appropriate CRSR key. It is significant to keep in mind that you can move the cursor over the tops of characters on the screen *without affecting those characters*.

RETURN

RETURN key — You press RETURN at the end of each line of instruction. Pressing this key tells the VIC to enter the line, or to execute the instruction(s). Sometimes it helps to think of RETURN as an ENTER key because this key actually enters the information or instruction into the computer.

CTRL

CTRL key — This key is used with the COLOR keys to select the *colors* that you create on the VIC screen. The key also provides you with the ability to define your own *control* commands that can be incorporated into any applications you might develop for the VIC. Some plug-in cartridges will make use of the Control key to perform special functions. The CTRL key works like the SHIFT key. You must hold it down while pressing the color key.



COLOR keys — You can change the colors of the characters displayed by simultaneously pressing the CTRL key and one of the 8 color/number keys on the top row of the keyboard. A shorthand notation for each color is shown on the face of the keys. The colors are black, white, red, cyan (light blue), purple, green, blue, and yellow. With these keys, you can set or change the color of letters, numbers and graphics displayed inside or outside a computer program. Once you “set” a color, everything you type will be in that color until you change colors again.



RVS ON and RVS OFF keys — You can *reverse* the images that the VIC puts on the screen by typing CTRL and RVS ON. Everything you type will then be reversed...for example, you can make the VIC display white characters on a blue background (the opposite of what it normally prints) by pressing CTRL and RVS ON. To get back to normal type CTRL and RVS OFF. Try it!



RUN-STOP key — Press this key to tell the VIC to *stop* what it is doing and return control back to you. When the VIC is *running* a program, you can stop the program with this key. Holding down the SHIFT and pressing this key, tells the VIC to begin *loading* information into memory from the optional tape cassette unit.



INST-DEL key — You can *insert* and *delete* characters from the line you are typing by pressing this key. When you press the key by itself (delete), the character that was immediately to the left of the cursor *disappears*. If you're in the middle of a line, the character to the left is deleted and the characters to the right automatically move in to close up the space. Holding down SHIFT and pressing this key, *opens* up a space in the line so you can *insert* a new character. This is very powerful for editing and correcting mistakes!



GRAPHICS & THE COMMODORE KEY — When you turn on the VIC, you're automatically in “graphics” mode which means you can type UPPER CASE letters and the more than 60 graphics you see on the keys. There are two graphics on each key. To get the graphic on the right side, simply hold down the SHIFT key and type the key with the graphic you want. To get the graphics on the left side, hold down the “COMMODORE” key (the little flag). In this way you can type UPPER CASE letters and the full graphics set at the same time! You can create pictures, charts, and designs by placing characters side by side or on top of each other (like building blocks).



UPPER/LOWER CASE and GRAPHICS keys — If you press the SHIFT and COMMODORE keys at the *same time*, you put the VIC into a *text mode*. You can then use the VIC like an ordinary typewriter, with full upper and lower case letters, plus all the graphics on the *left* side of the keys. The left side graphics are ideal for creating charts, graphs, and business forms. To get back into "Upper Case/Full Graphics" mode, press the SHIFT and COMMODORE keys together.




PROGRAMMABLE FUNCTION KEYS — The four tan keys on the right side of the console are not defined when you turn on the VIC. They can be assigned tasks or *functions* from within the applications that you create. By using these keys with and without SHIFT, you get a total of eight assignable function keys. Function keys will be mostly used with plug-in cartridges containing special programs, but computer programmers can assign these keys as well

SPECIAL KEYS — The VIC keyboard also contains special symbols not found on many typewriters, or even most computers. Examples are the English "pound" sign (£), pi (π), back arrow (←), up arrow (↑), greater/less than (> <), and brackets ([])

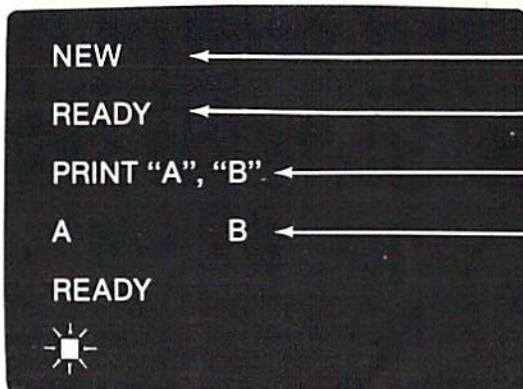
This concludes your tour of the VIC keyboard. Using only words, it's difficult to convey just how flexible and powerful the VIC keyboard is. The best way to discover everything the VIC can do is to begin your own "touring". Experiment with the keyboard. Try out the various upper/lower case features mentioned above. See what you can create with the rich VIC graphics set. The keyboard is your direct link to the VIC. Knowing the Keyboard will help you know your VIC 20.

Clear the screen and enter the following lines:

NEW 

PRINT "A", "B" 

The screen shows:



```
NEW
READY
PRINT "A", "B"
A      B
READY

```

you typed

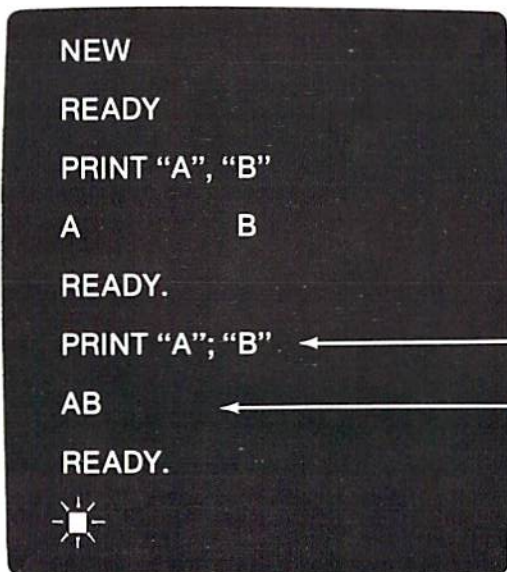
the VIC typed

then you typed

and the VIC displays

Now, enter this line and press  : PRINT "A"; "B"

The screen now shows:



```
NEW
READY
PRINT "A", "B"
A      B
READY.
PRINT "A"; "B"
AB
READY.

```

you typed

VIC shows

When the comma was used in the first PRINT statement, the VIC placed the letters on the screen, but separated them by several spaces. When the semicolon was used, the VIC displayed the two letters close together.

In the first case, the letters are exactly 11 spaces apart. That fact gives a clue to what's happening. The VIC divides the screen area into two equal parts.

When the VIC is PRINTing two messages or numbers separated by a *comma*, it puts the first item on the left side of the screen and the second on the right...

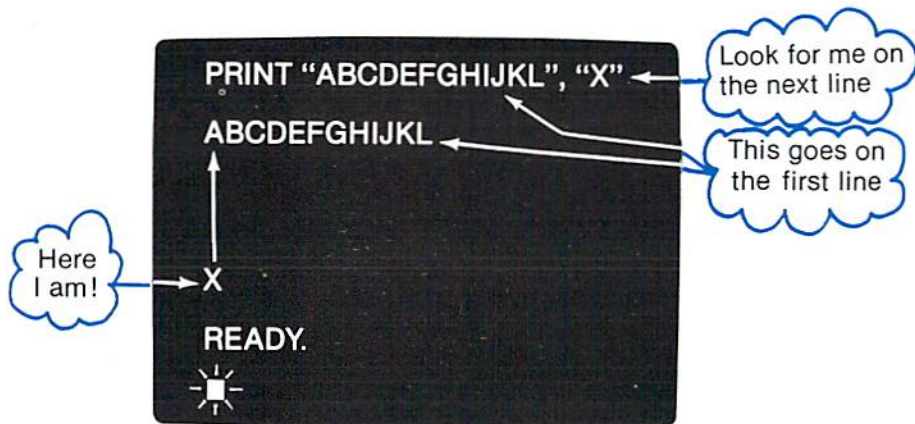
UNLESS

The *first* item is longer than 11 characters.

If the first item is less than (or equal to) 11 characters, the VIC PRINTs it and then moves to the center of the screen to display the second item. If the first item is longer than 11 characters, the second item appears on the next line. Clear the screen, and try this example:

```
PRINT "ABCDEFGHijkl", "X"
```

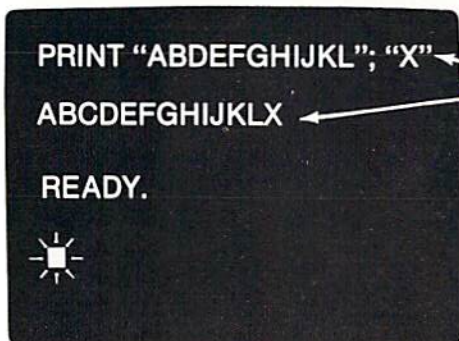
The screen will show:



The first part of the message is 12 characters long, so the "X" ends up on the next line. Repeat this example with a semicolon (;) between the two items.

PRINT "ABCDEFGHijkl"; "X"

Does your screen show this result?



Now I am on the same line

Get the idea? The VIC acts like a typewriter with an automatic *tab* set near the screen's center. When it sees the comma, it either "tabs" to the center of the screen or the beginning of the next line, whichever is next available.

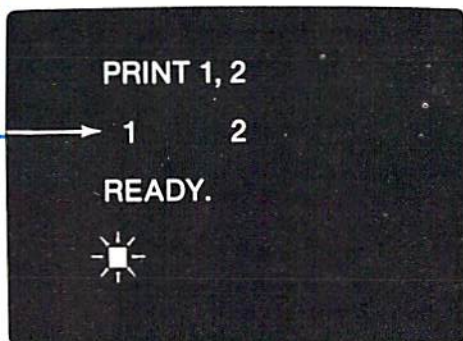
Clear the screen and type the following line into the VIC:

PRINT 1, 2

Aha! With numbers you can leave off quotation marks.

The screen shows:

space

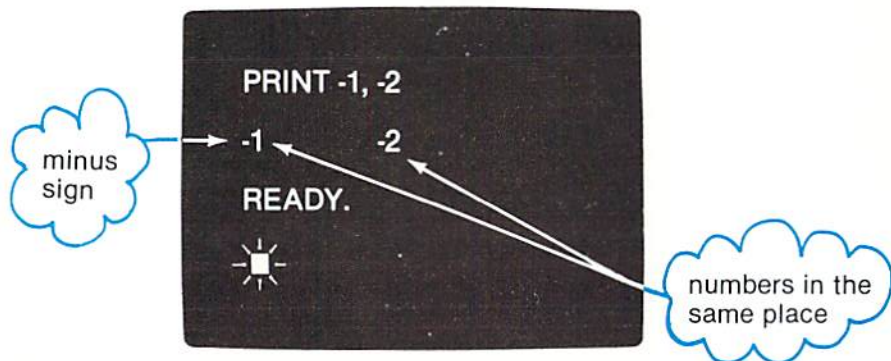


Do you see the space in front of the first number? When the VIC displays numbers, it leaves a space at the beginning for the *sign* of the number. If the number is positive, you see a blank space. If the number is negative, a *minus sign* (-) would appear on the screen.

Try it and see. Enter this line into the VIC:

PRINT -1, -2

Look at the screen and see what is displayed.



The numbers appear in the same places as in the previous example; they are preceded by the minus signs (-).

These few examples give you some idea of how the VIC can help you get your messages and information on the screen. The VIC has many other ways to assist you with this task, and you will learn what they are as you continue to use your new computer.

The VIC Calculator

The VIC can also be used as a 9 digit calculator. The + and - sign are used just like in mathematics. The VIC multiplication sign is the asterisk (*) and the division sign is the slash (/). Type these calculations and check the results. See Appendices C and K for more information.

The mathematical slash is the one on the ? key. The slash on the N key is a graphic symbol

PRINT 1 + 1

RETURN

PRINT 2*(4/2)

RETURN

PRINT 3 - 2

RETURN

PRINT 5000/5

RETURN

PRINT 5*2

RETURN

PRINT 2/3

RETURN

PRINT 6/3

RETURN

PRINT 3+3

RETURN

If you PRINT a calculation you should put it *outside* the quotation marks. Try these examples:

1 PRINT "2*(4/2)"

RETURN

1 PRINT "THE ANSWER IS"2*(4/2)

RETURN

VIC automatically performs the calculation outside the quotes and prints the result

The + sign is used for exponents. This means 3³ or 3x3x3

VIC prints everything inside the quotes

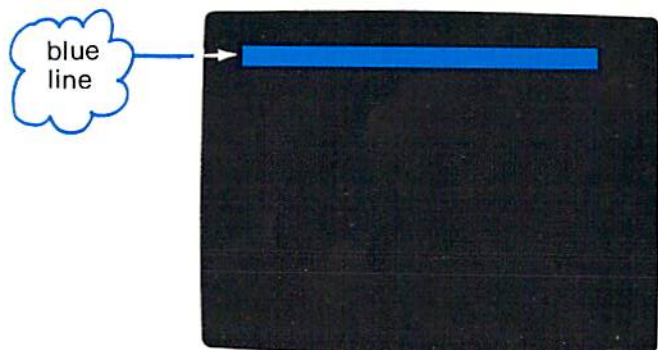
INTRODUCTION TO COLOR

The VIC can print letters, numbers and graphic symbols in 8 different colors. It can also print characters in *reverse*.

With the screen clear, hold down the CTRL key and press this key:



Now let go of the CTRL key and put your finger on the SPACE bar at the bottom on the keyboard. Hold the SPACE bar down. What happens? Is there a blue line being drawn across the screen?



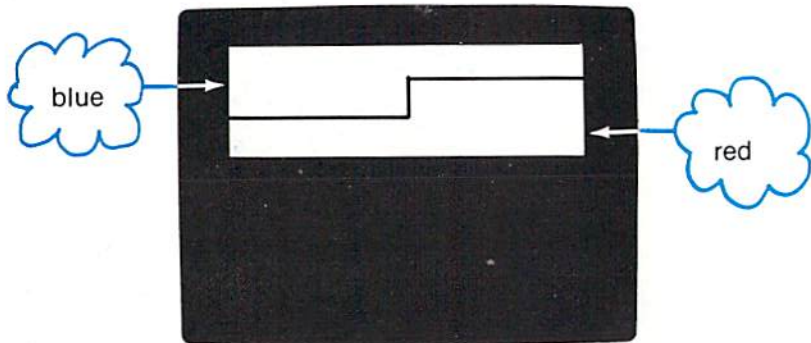
Hold the SPACE bar down as long as you want. As the cursor disappears off the right edge of the screen, it reappears on the left and the blue line starts forming a larger blue color bar.

Release the SPACE bar and do this:

Hold down the CTRL key and press the RED key.

the key with
3 on top

The cursor should now be *red*. Press and hold the SPACE bar once again. Does a new red color bar start to form? Yes! Well, keep painting!



Change to other colors as you feel like it. Make the color bars as thick or thin as you like. Enjoy this newly discovered ability of the VIC that puts a little color into your life.

Now, type **CTRL** **RVS OFF** and hit the space bar.

Nothing happens except blank spaces. Type **CTRL** **RVS ON**

and the color bar reappears. Try typing some letters in "reverse". Reverse letters make excellent headlines and are often used to highlight special words and numbers. You can also use reverse characters inside a program. For example, try this:

NEW

10 PRINT " **CTRL** **RVS ON** VIC20";

20 GOTO 10

RUN

hold this while typing this

don't forget to type **RETURN** after each line!

To get ready for the next chapter, type

RUN STOP

RESTORE

and type the word NEW and

RETURN

From now on, use this method to erase unwanted programs and start "NEW"

3

Color and Graphics

- Programming in Color
- The VIC Color Keys
- Changing Screen and Border Colors
- Screen and Border Color Combinations
- Coloring the Screen
- Screen Locations
- Random Colors
- Combining Sound and Color
- Keyboard Graphics
- Graphics in Headlines and Titles

Try typing this program:

Type this program exactly as shown and see what happens!

```
1FOR H = 1 TO 505
2PRINT " ♥ ";
3NEXT
4FORC = 8 TO 255 STEP 17
5POKE 36879, C
6FOR T = 1 TO 500: NEXT
7NEXT
8GOTO 4
RUN
```

This means
print 505
hearts on
the screen.

This changes
the value of
C (color) 17
steps at a time.

This is a time
delay loop that
tells VIC to
count to 500
before changing
colors again.

To stop the program, press the



key.



COLOR & GRAPHICS

The VIC and your color TV set give you the ability to put colors everywhere on the screen. When you first turn on the VIC, the border, the cursor, and any characters on the screen are already in *color*. But that's only the beginning. The VIC can display 8 cursor colors, 8 border colors and 16 screen colors!

Let's start by using the keys on the VIC's keyboard to make colors on the screen. Type any letter. The letter should appear in dark blue on a white screen. Now look at the top row of keys (the numbered keys from 1 to 8). Do you see the color names written on the front of each key? Now find the key marked **CTRL** on the left side of the keyboard.

Hold down the **CTRL** key and hit the key marked **YEL**.

Release the **CTRL** and **YEL** keys and type any letter on the keyboard. This letter should appear in yellow. Now hold down the **CTRL** key and hit another color, then type some letters. See how easy it is to change letter colors on the screen?

Now find the key marked **RVS ON**. Hold down the **CTRL** key and hit the **RVS ON**. Try typing some letters. All the letters you type (until you hit the **RETURN** key) appear in *reverse* on the screen, like a photographic negative. If you hold down **CTRL** and hit **RVS OFF**, the letters you type will be displayed normally.

PROGRAMMING IN COLOR

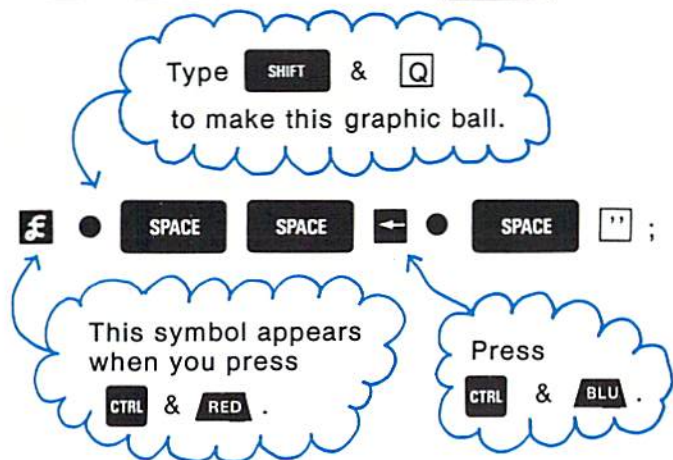
Now let's combine color control with a simple program command. Note that when you type **CTRL** and a color key inside quotation marks, a reverse graphic symbol appears. This is okay. Do this:

Hold down the **SHIFT** key and press the **CLR HOME** key.

Type the letters:

N **E** **W** and press the **RETURN** key.

Then type: **1** **P** **R** **I** **N** **T** **"** **SPACE**



and press **RETURN**

Type: **2** **SPACE** **G** **O** **T** **O** **SPACE** **1**

and press **RETURN**

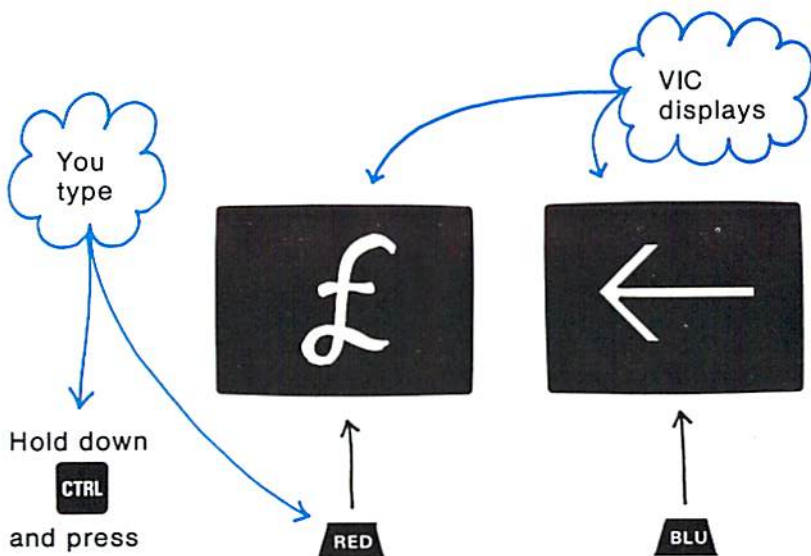
If you have trouble typing this example, flip back to the section called "Getting Started" in Chapter One. You can't hurt the VIC with anything you type, but you can get confused by certain key combinations. If you accidentally

leave the **SHIFT LOCK** key engaged, for example, the resulting display is difficult to decipher. If you make a mistake, hit

RETURN a few times and retype the entire line. The new line will automatically replace the old one. When you have the two lines shown above on the screen, type:

R **U** **N** and press **RETURN**.

As soon as you press that final **RETURN**, you should see hundreds of red and blue balls float by on your screen. How? It's easy with the VIC. Look back at the example above. See the two strange characters in the line that begins with "1"? They were created when you held the **CTRL** key and pressed the **RED** and **BLU** keys. The symbols that appear are VIC's shorthand to tell it to make the first ball red and the second one blue.



Nothing to it. When you are tired of red and blue, press STOP. If you like free form exploring, try retyping line 1.

Throw in some **CTRL** and color keys along with graphics

characters and letters or numbers. The VIC can handle it all and will give you an enlightening color display.

THE VIC COLOR KEYS

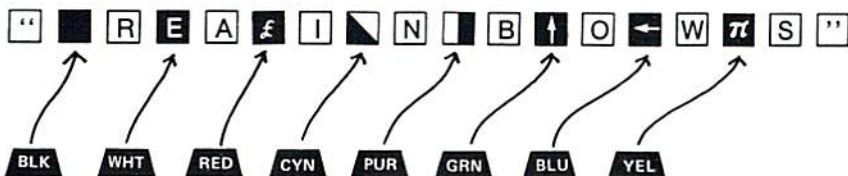
In the last example, you discovered that you can insert color controls into a PRINT message by using the **CTRL** and the keys whose faces are labeled:



These keys are the number keys 1 through 8. When you press these keys in the middle of a PRINT statement, a set of "strange" characters appear. To see what these

characters look like, press **RUN STOP** (if the program is still running), clear the screen, and enter:

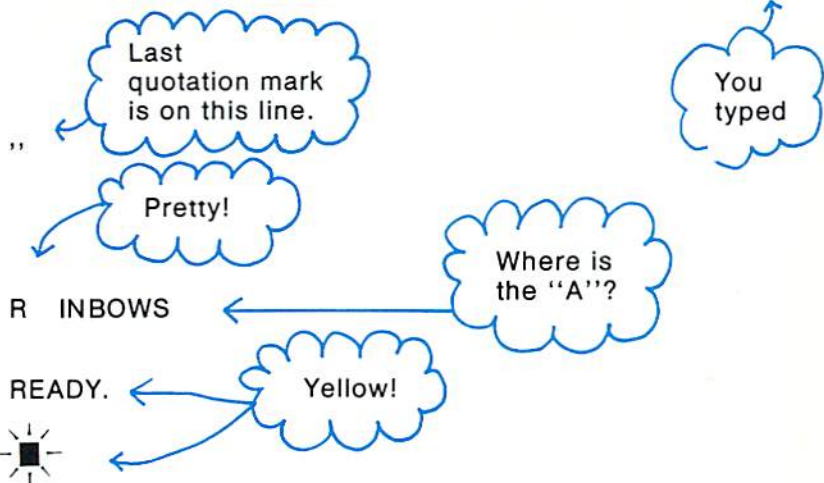
PRINT



Don't forget to put in the quotation marks or to use the **CTRL** key with the color keys. Your screen should show:

PRINT

“ ■ R E A ■ I ■ N ■ B ↑ O ← W ↻ S



Where is the letter “A”? Oh! The second color key is **WHT** and the background is . . . you guessed it . . . *white*. PRINTing a white letter “A” on a white background gives you a space in your RAINBOWS.

The other seven letters in the word RAINBOWS are each a different color. The last letter, S, is yellow. Note that the READY message is also in yellow, along with the cursor. *When color controls are put in PRINT messages, the VIC remembers the last color used and stays in that color.*

To change the cursor back to the regular blue color, hold down **CTRL** and press **BLU** . If you wish, try PRINTing some color messages on your own. You will get a chance to see more uses for the color control keys in just a bit. Right now, there is an important announcement . . .

EXTRA!! EXTRA!!
The VIC Color Show is on The Way!!

CHANGING SCREEN & BORDER COLORS

Now that you know how to change the colors of letters and graphics, we're going to show you how to change the screen and border colors. You can display 255 different screen and border color combinations. To make sure the VIC is ready for

what comes next, press **RUN STOP** and then **SHIFT**, **CLR HOME** to erase the screen. Next, type these lines into the VIC:

N **E** **W**


and press **RETURN**.

(Press **RETURN** at the end of each of the following lines.)

The left-most character on each line

1FOR X = 1 TO 255

2POKE 36879, X

3PRINT "  POKE 36879," X

4FOR T = 1 TO 1000: NEXT T

5NEXT X

Poke what?!

For this one, hold down **SHIFT** and press **CLR HOME**.

Add one to "X" — in other words — change "X" to the next number and do it over again.

This is a "time delay". VIC counts from 1 to 1000 (whew!) between changes.

When you have typed these lines, look them over and see if they match what is shown on this page. If there are some that don't match, retype those lines from the beginning. Use the **CRSR** to move back to any lines you want to retype.

Once you are satisfied that the lines are all right, get a piece of paper and a pen or pencil and place it nearby. Then type:

R U N and press **RETURN** .

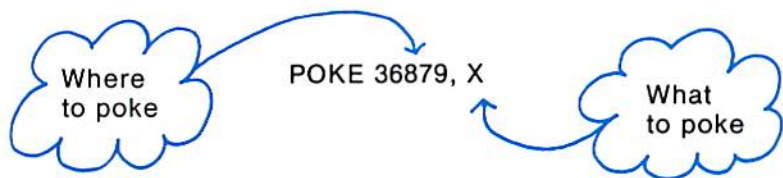
Your screen should begin to blush and flash. The border changes colors. The background changes colors. Even the small message at the top of the screen is changing colors. The VIC is displaying 255 different color combinations. (Syntax error in one of the lines? Retype that line and then type:

R U N and **RETURN** again.)

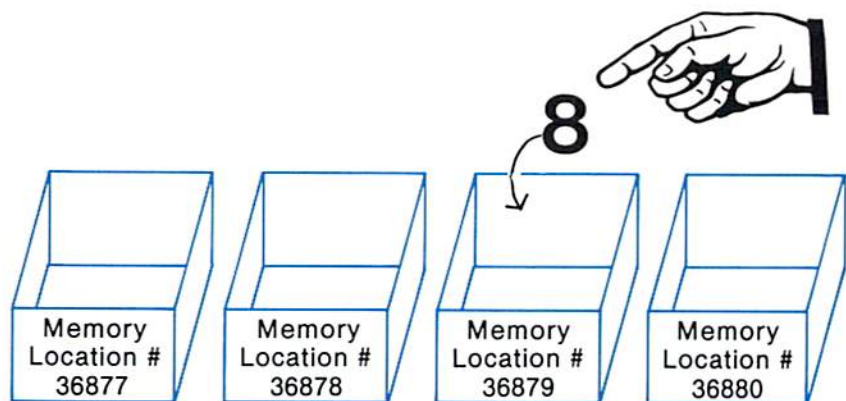
While the VIC is running, if you notice a particular set of colors that look interesting; write down the number of that combination. Only the number at the end of the printed message is changing. The "POKE 36879," stays the same. (Occasionally, you will not be able to read the message because it is the same color as the background. C'est la vie. The letters and numbers will reappear after a few additional flashes.) The VIC has eight border colors, sixteen background colors, and eight character colors. You can put characters in all eight colors over any background. That gives you a lot of combinations to explore.

If you want to restore the screen to its original colors, simply hold down the **RUN STOP** key and hit **RESTORE** .

Line two in the example above is responsible for making the VIC change colors. The line contains a POKE command. Every POKE command has two numeric values that the VIC uses:



The first number (in this case, 36879) is the location in memory (think of it as a little box labeled 36879) into which you are going to POKE (ie. place) the second number, X. The memory location 36879 happens to be where the VIC stores its information on what the screen's border and background colors should be. Each value of "X" corresponds to a different color combination the VIC can display. For this example, "X" starts with the value 1, then changes to 2, 3, and so forth, up to a final value of 255.



AN EXAMPLE OF POKE 36879, 8


To assist you in your quest for the *perfect* color combinations, here is a table of POKE values ("X" values) and the background and border colors they produce. The table gives you the POKE values to produce all combinations of these colors. The POKE values are in sequence with skips of eight (8) numbers. The missing numbers are the POKE values that cause the *reversal* of displayed characters.




SCREEN & BORDER COLOR COMBINATIONS

Screen	Border							
	BLK	WHT	RED	CYAN	PUR	GRN	BLU	YEL
BLACK	8	9	10	11	12	13	14	15
WHITE	24	25	26	27	28	29	30	31
RED	40	41	42	43	44	45	46	47
CYAN	56	57	58	59	60	61	62	63
PURPLE	72	73	74	75	76	77	78	79
GREEN	88	89	90	91	92	93	94	95
BLUE	104	105	106	107	108	109	110	111
YELLOW	120	121	122	123	124	125	126	127
ORANGE	136	137	138	139	140	141	142	143
LT. ORANGE	152	153	154	155	156	157	158	159
PINK	168	169	170	171	172	173	174	175
LT. CYAN	184	185	186	187	188	189	190	191
LT. PURPLE	200	201	202	203	204	205	206	207
LT. GREEN	216	217	218	219	220	221	222	223
LT. BLUE	232	233	234	235	236	237	238	239
LT. YELLOW	248	249	250	251	252	253	254	255

COLORING THE SCREEN

Time to combine the VIC's color and graphics features in a delightful color display. Again, you only have to enter a few lines into the VIC, and spectacular events begin to happen.

If any examples are still flapping or bouncing across the screen,  them. Clear the screen. Then, type these mysterious lines:

   (not too mysterious, yet)

1 L = INT(RND(1)*500) + 1

Generates a
random number from 1
through 500.

2 C = INT(RND(1)*8) + 1

Read about
these in
Chapter 7.

Random number
from 1 through 8

3 POKE 7680 + L, 160

Where is
this location?

4 POKE 38400 + L, C

And this
one?

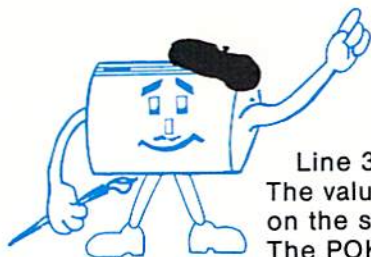
5 GOTO 1

Check the example to see if all the lines were typed as shown. If they are all right, clear the screen and type

R **U** **N** . Your screen should explode in tiny bursts of color.

The VIC has done it again! With only a few keystrokes, it is now dabbling in color everywhere. How is this being done?

In simple terms, lines 1 and 2 are randomly selecting where on the screen the color will go (L for *Location* maybe) and what color (C for, of course, *Color*). Line 1 generates a number from 1 through 500. Line 2 generates a number from 1 through 8.



Line 3 POKEs a character onto the screen. The value 7608 represents the *home* position on the screen; the top lefthand corner. The POKE value 160 is a solid block (essentially a reversed space).

Line 4 POKEs the color onto the character. The value 38400 represents the location in memory of the color component of the character in the *home* position.

SCREEN LOCATIONS

VIC TIP:

To POKE characters on the screen, you must POKE the screen location and the color at that location for each character. The screen locations start at 7680. The color locations begin at 38400. See Appendix for the screen memory map.

You should try POKEing other values onto the screen. Change the 160 in line 3 to any number from 0 to 255.

EXPERIMENT — and enjoy your VIC's colorful personality.

There are 506 possible screen locations (chart). You can place any word, letter, sentence, graphic, or whatever — whenever you want on the screen. Just imagine the screen consists of 506 boxes like this — each box has a number.

Enough POKEing around! Time to get back to the color control keys.

VIC TIP:

To restore your VIC to its normal border and background colors, type this POKE statement:

POKE 36879, 27 **RETURN**




or hold down the RUN/STOP key and hit **RESTORE**

RANDOM COLORS

You are now going to find out how you can let the VIC choose colors to put on the screen. First clear the screen, and type these lines (press **RETURN** after each line):

SPACE **CTRL & RED** **CTRL & BLU**

N **E** **W**

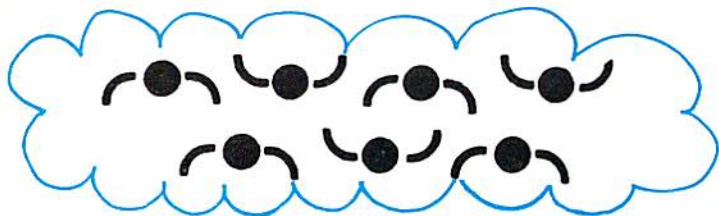
1 PRINT "    ";

2 GOTO 1

RUN

To get "birds":
Hold: Press:
SHIFT **J** **Q** **K**
SHIFT **U** **Q** **I**

Your screen should fill with red and blue birds. Wonder where they are flying to? They are really moving.



Let the birds fly for a while. When you want to stop their flight, press **RUN STOP** .

Now that you know how to create birds, clear the screen and type these lines: (Remember the **RETURN**)

N **E** **W**

1 A\$ = " **█** **E** **£** **▀** **▯** **↑** **←** **π** "

To type this line, hold down **CTRL** and press each of the color keys

(**BLK** **WHT** **RED** **CYN** **PUR** **GRN** **BLU** **YEL**).

2 N = INT(RND(1)*8) + 1

This line gives *random* whole numbers from one through eight.

3 B\$ = MID\$(A\$,N,1)

Color controls

Random value

Picks one color control out of A\$.

4 PRINT B\$ " **◡** **◡** ";

SPACE once.

5 GOTO 2

Look the lines over once they are entered. Check to see if they match what is shown above. When you are ready, clear

the screen and type **R U N** . If you have any errors, retype this problem line and **R U N** it again, Sam

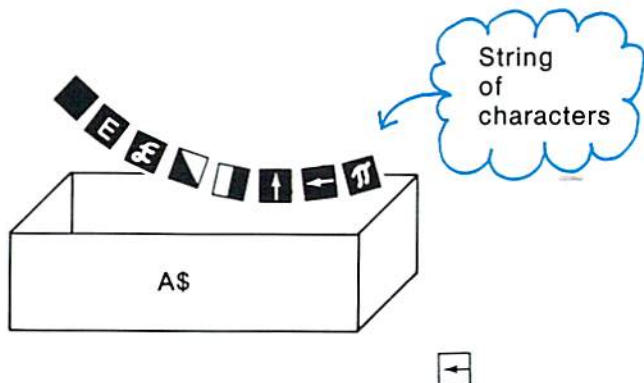
(excuse us, VIC!!). Birds! Birds! Everywhere!! In every color of the rainbow! Your screen should be humming (oops!) with flocks of "rainbow" birds. Notice that there are gaps in your flock. Actually, the VIC is PRINTing white birds in those places. If you want these "invisible" birds to appear, you must change the background color. (Remember how? You POKE a number into location 36879.)

Of course, if you change the background to another color, say blue, then all the blue birds will disappear. Try it and see for yourself. Press **RUN STOP** when you want to keep the birds from flying.

The last example used some interesting VIC features to get the birds on the screen and change their colors. Let's look at the example line-by-line. The first line is:

```
1 A$ = " ■ E £ ▽ □ ↑ ← π "
```

The color controls form a long *string* of characters and are put into the VIC's memory in a place called A\$ (pronounced "A string" or "A dollar sign"). You can think of A\$ as being like a box where strings of characters can be stored. The first position in the box A\$ contains the **BLK** control character. The last position contains the **YEL** control character.



The next line is:

$$2 N = \text{INT}(\text{RND}(1)*8) + 1$$

This line generates a *random* positive whole number from one through eight and puts the number into "N". "N" is the place in the VIC's memory that can be used to store numbers. The VIC knows that "N" is a place to store numbers since there is no dollar sign (\$) at the end of the name.

In line one, the location used to store the color controls was called A\$, and it has a dollar sign on the end. The VIC knows that locations whose names end with a dollar sign are to be used to store messages or strings of characters.

For more information on how **I** **N** **T** and **R** **N** **D** work together to generate random numbers, look at Chapter Seven in this book. For now, just be aware that this line is producing numbers from one through eight. See the number eight on the right side of the equal sign? That number controls how many random numbers are being generated. If you were to change the eight to a six, then any number from one through six would be produced.

Hmmm . . . moving right along now:

$$3 B\$ = \text{MID\$}(A\$,N,1)$$

This line creates a new box called B\$, and puts one color control character into it. A\$ contains all eight color control characters. "N" is a random number from one through eight. What does MID\$ do? It picks out one color control character from A\$ at the "N"th position in the string. This character is put into B\$. Yes, "Virginia", now we have three little boxes inside the VIC, one called A\$, one called "N" and one called B\$. Each time the VIC gets to line two, it is given a new value for "N". This determines which color MID\$ picks and puts into B\$.

A\$ = "  E £   ← π "

The fourth line is:

```
4 PRINT B$ "🐦 🐦";
```

This line tells the VIC to use the color it finds in the B\$ box and to draw the birdlike creatures with it. The command, PRINT B\$, creates the same effect as typing a specific color control character. By using B\$, however, we can have the VIC change the color character automatically. As "N" (line 2) changes, B\$ changes (line 3), and the birds change color.



The VIC's ability to generate random numbers can be combined with its color, sound, and graphics features in many interesting ways. For example, let's revisit the VIC Color Show, but randomly, and make some noise while we are at it.



In this last example, the VIC's color and sound are both randomly produced. Enter the following lines into the computer:

NEW

1POKE 36878, 3

Set volume.

2C = INT(RND(1)*255) + 1

Random number 1 through 255

3S = INT(RND(1)*50) + 175

Random number 175 through 224

4POKE 36879, C

Set color.

5POKE 36875, S

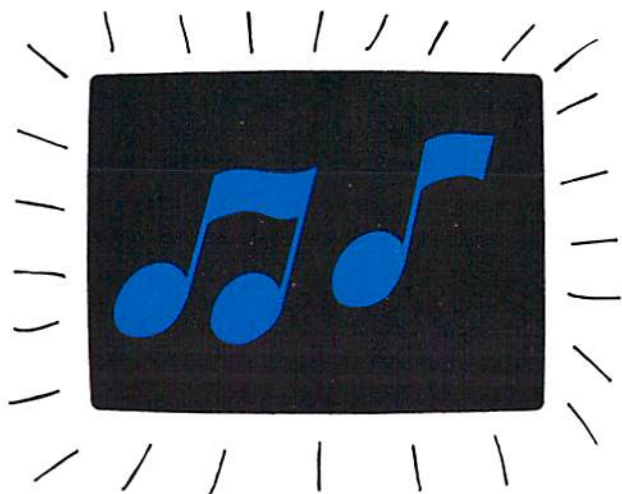
Play note.

6FOR T = 1 TO 100: NEXT T

Wait.

7GOTO 2

The bubble captions tell most of the story on this example. Chapter Five tells you more about music. If you are ready to run the example, clear the screen and type **R U N** , then press **RETURN** .



My, my!! What a wild bunch of sound and color combinations. Oh, well! You get what you play. You can't expect random music to be to everyone's taste.

When you want to halt this color and sound machine, press



. Then you need to type these two lines to shut off the note that is still playing and restore the VIC screen to its normal colors:

POKE 36875, 0

Stop
note.

POKE 36879, 27

Restore
colors.

Time for you to explore on your own. Experiment with the VIC's colors. Enjoy your colorful, musical VIC computer.

KEYBOARD GRAPHICS

One of the special features of your VIC 20 is the graphics keyset. Most of the keys have two graphic characters printed on the front. You can display these graphics on the screen, or, if you have a Commodore dot matrix printer, print them on paper along with all the other keyboard symbols.

To display the graphic on the left side of the key, hold down the Commodore key while pressing the graphic key, to display the right side graphic, hold down the **SHIFT** key and type the graphic you want.

VIC TIP:

Caution...if you push both the SHIFT and Commodore keys at the same time, you will switch to upper/lower case mode. In upper/lower case, only the left side graphics are available. To get back to upper case/graphics mode, push **SHIFT** and **C** together.

The easiest way to use VIC graphics is to type the **COMMODORE** **C** or **SHIFT** key, and the graphic symbol you want to display from the keyboard. For example, type the following:

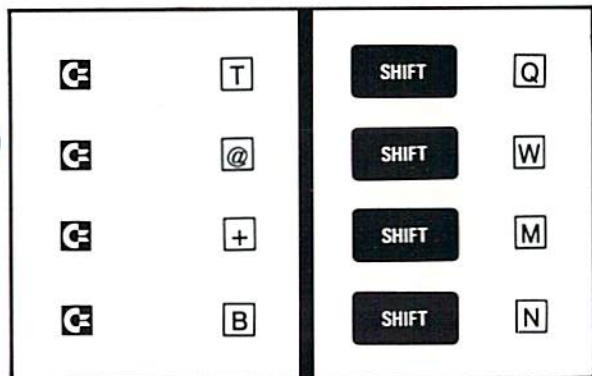
SHIFT **S**

You should see a *blue heart* printed on your screen. Try typing some other graphic keys. Here are some other keys you might want to try:

Left Side Graphics



Right Side Graphics

Note: Lines and bars come in different sizes & increments so you can make exactly the graphics you want.



Notice that the left side graphics are good for creating charts, graphs and business forms. Right side graphics are good for drawing illustrations, animations . . . even playing cards!

GRAPHICS IN HEADLINES & TITLES

Graphics aren't limited to cartoons and games. You can use a variety of special effects to enhance titles in charts and graphs, or highlight special words in programs that use a lot of text. The easiest way to highlight a word or phrase is to type it in REVERSE. Simply type  ,  , and the word you want.


For example:

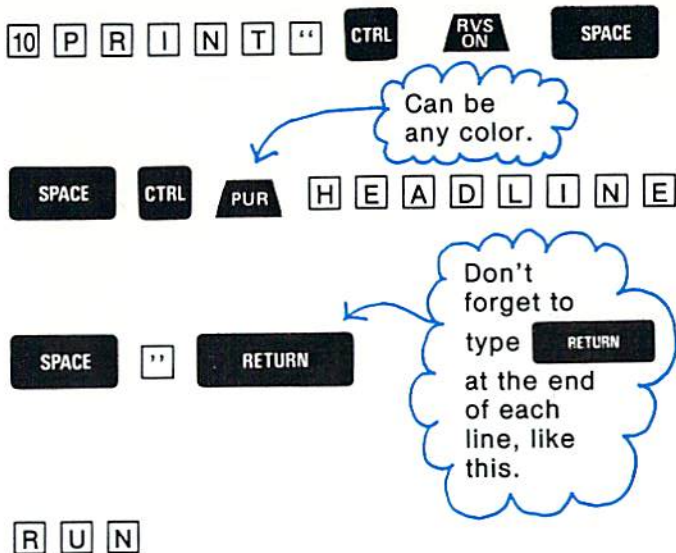
  

Your screen will show  .

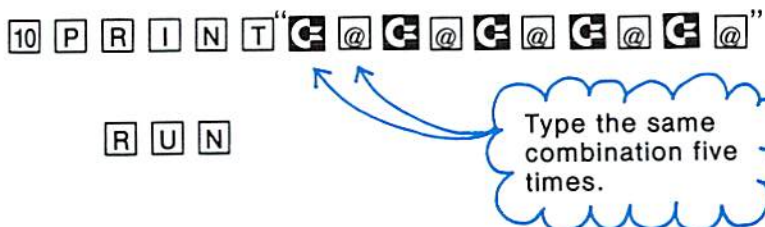
Hold down while typing.

Type a headline using the keyboard.

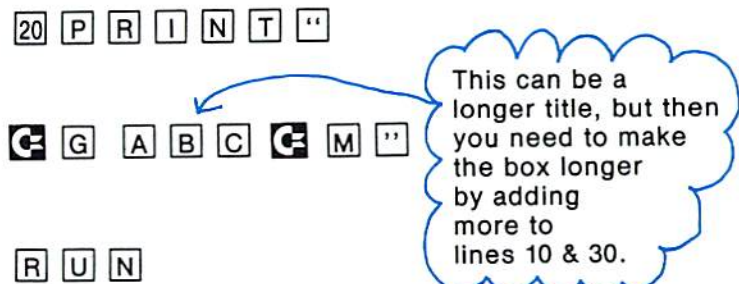
Now try some spaces to create a title bar . . .



Another way to highlight a word is to draw a graphic box around it. The technique is like drawing. Be sure to type exactly as shown. Let's do it step by step.



The screen will show a straight line. The next line includes your headline and a vertical line at each end to complete the sides of your "box".



See how we are building the headline box? Now to finish . . .

30 P R I N T " **C T C T C T** "

C T C T "



R U N

VIC TIP:

To edit your program, type LIST and press RETURN. Now you can go back and change a line that isn't exactly right, by using the CRSR and INST/DEL keys to move the error and retype it. After you make a correction or change, press RETURN to enter the change for that line . . . or . . . you can also retype any line any time and press RETURN to change it.

Here's another way to highlight a word or headline, by animating it so it appears to blink several times when it comes on the television screen . . .

N E W



10 FOR H = 1 to 250

20PRINT " **SHIFT CLR HOME** H E A D L I N E "

25FOR T = 1 TO 50: NEXT

30PRINT " **SHIFT CLR HOME CTRL RVS ON**

H E A D L I N E "

35FOR T = 1 TO 50: NEXT

40NEXT H

R U N

If the headline doesn't overlap evenly, try adjusting the spacing inside the quote marks. To speed up or slow down the blinking change the number in lines 25 and 35.

4

Animation

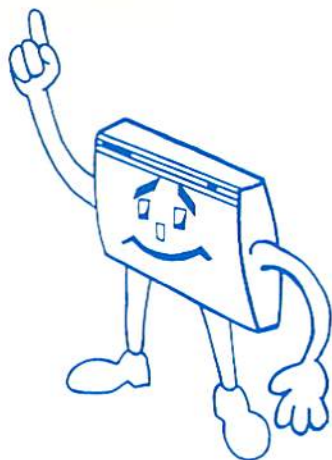
- Flying Birds
- Bouncing Ball
- Controlling the Cursor

Try typing this program:

Type this program exactly as shown and see what happens!

```
10 PRINT " CLR HOME ";
20 PRINT " □ ○ ▽ "
30 PRINT " □ ▣ □ "
40 PRINT " ▽ □ ▽ "
50 FOR T=1 TO 300: NEXT
60 PRINT " CLR HOME ";
70 PRINT " □ ○ □ "
80 PRINT " ▽ ▣ ▽ "
90 PRINT " □ □ □ "
100 FOR T=1 TO 300: NEXT
110 GOTO 10
```

To stop the program,
press the **RUN STOP** key.



FLYING BIRDS

This chapter shows you how to use the VIC's *graphic* abilities to create illusions of characters moving about the screen.

The illusion of movement you can create with the VIC is often referred to as *animation*. Animation can make any program — from games to business programs — fun and exciting.



Let's begin by entering the following lines:

Hold down the **SHIFT** key and press the **CLR HOME** key.

N E W and press the **RETURN** key.

1 **SPACE** **P R I N T** "   "



To get birds	
HOLD	PRESS
 SHIFT	J Q K
 SHIFT	U Q I

For more on "BIRDS" see Chapter 3, "Graphics"

and press **RETURN**.

2 **SPACE** **F O R** **SPACE** **T =**
1 T O 1 5 0 : N E X T T

Wait awhile

and press **RETURN**.

3 **SPACE** **P R I N T** "   "

See line 1

and press **RETURN**.





4 **SPACE** **F O R** **SPACE** **T =**
1 T O 1 5 0 : N E X T T

Wait again

and press **RETURN**.

5 **SPACE** **G O T O** **SPACE** **1**

WHAT YOU HAVE JUST TYPED SHOULD LOOK LIKE THIS:

```
1 PRINT "   "  
2 FOR T = 1 TO 150: NEXT T  
3 PRINT "   "  
4 FOR T = 1 TO 150: NEXT T  
5 GOTO 1
```

Look over what you have typed. Does it match what is shown? If it does not, re-enter the lines that are different. When everything matches, type:

R **U** **N** and press **RETURN**.

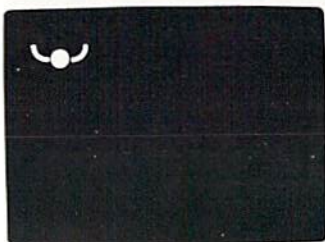
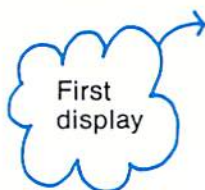
HINT: You can re-enter or change any line by typing it over. The second typing will automatically erase and replace the first. You can erase a line by typing the line no. only followed by **RETURN**.

Behold your first VIC illusion or animation. The "bird" creature is huffing and puffing up in the top left-hand corner of the screen. It seems to be using a lot of wing power, but there must be a strong headwind. The bird is getting nowhere.



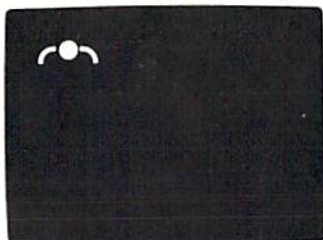
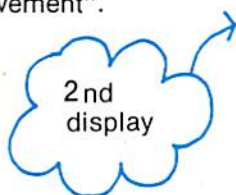
You, however, *are* getting places. Can you see how the bird appears to be actually moving its wings? Study the example for a moment. The secret to almost all animation on the VIC is given in this one example. How you can create this illusion of movement is easily described in a few simple steps. Here is a brief summary of how to make just about any character wink, blink, wave, nod, seem to move, and so forth.

First, display the character or combination of characters in one position on the screen.



Second, wait for a short period of time. A FOR-NEXT "time delay loop" makes this happen. Examples of "time delay" instructions are shown in lines 2 and 4 of the flying bird program. These lines tell the VIC to count from 1 to 150. The VIC can count to 150 quickly. Try changing 150 to 500 (or some other number) and see what happens.



Third, display the character again but with some part of it altered. Because of the previous delay, your eye is fooled and you think you have seen "movement".



Fourth, wait again. Based on what effect you are trying to achieve, this wait may be longer or shorter than the first delay loop. In the example, the delays between PRINTings were the same.

Fifth (the easy part), repeat the first four steps just mentioned. This is accomplished by the GOTO command which tells the VIC to go back to the first line and start over again. Thus, the program prints the bird with his wings "up", counts to 150, then prints the bird with his wings "down" and goes back to the beginning again.

In the first and third steps, where the character is being displayed, it is often necessary to make sure that no old character parts are left on the screen. In our bird example, this problem is taken care of by putting the screen clear character

( and  in each message. That character

clears the screen and homes the cursor so each version of the bird (wings up, wings down) appears in the upper left-hand corner.

You may want to experiment with this example some.

Press **RUN STOP** to stop the wing flappings and change the upper

counting limit (150) in lines 2 and 4. Try changing just one value, leaving the other at 150. Does your bird begin to soar? Try a value like 50 in both places. Does your bird fly faster? What about an upper limit of 500? 1000?

Hmmm...the bird gets rather s-l-o-w.


May Day!
May Day!



CLear Out for HOME

You already know that you can clear the screen by holding down **SHIFT** and pressing the **CLR HOME** key. From within

a program, you can tell the VIC to clear the screen. You do so by typing:

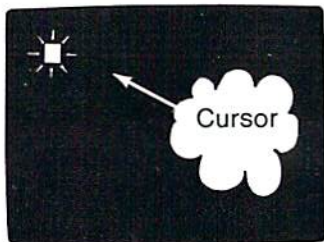
PRINT "  ";

This character appears when you hold **SHIFT** and press **CLR HOME**

When you are entering a PRINT message, if you press the two keys (**SHIFT** and **CLR HOME**) that you use to clear the

screen, the *reverse* heart image appears on in the message field. That symbol is a signal to the VIC that when the message is PRINTed, the screen is to be cleared at that place in the message, and the cursor sent *home*. Home is the upper left-hand corner of the screen.

It's nice to be...




Try the PRINT statement directly. Type it into the VIC. Does the screen clear?



Type: PRINT "  ";

Yes!
Works in the
immediate mode


Now, mess up the screen. Put some birds, characters, numbers, and graphics symbols all over the place. When your screen is sufficiently full, type this line:

PRINT "  ";

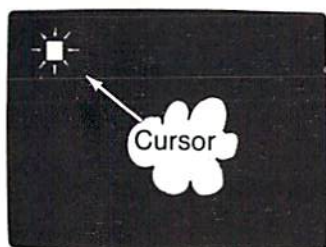
Do not hold  Press only 

Did you notice a difference from before with the other PRINT statement? Yes, this time the cursor went *home* but the screen did not clear. If you *don't* press the  key while typing  into the message field, the reverse S image appears

instead of the heart. The reverse S tells the VIC to *home* the cursor, but *do not* erase the screen.

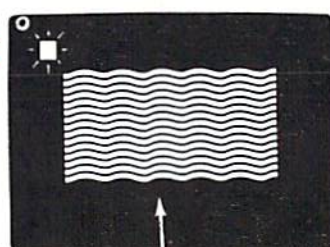
Type: 





Screen
clears

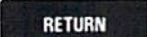
Cursor
moves
here




Screen
stays
cluttered

BOUNCING BALL

Clear the screen and enter the following into the VIC:

(Press  after typing each line.)

NEW
1 PRINT "  ";





Space

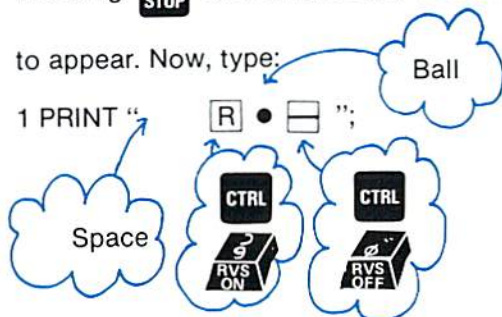
2 GOTO 1

RUN

When RUN is typed, the screen should fill with columns of blue balls. Let the balls flow by for awhile and then press **RUN STOP**.

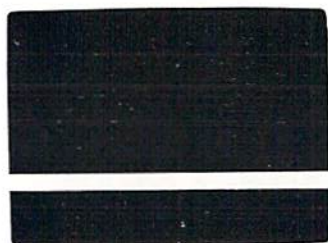
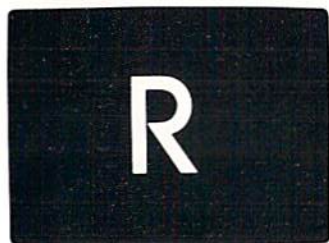
Pressing **RUN STOP** causes the BREAK and READY messages

to appear. Now, type:



The two reverse image characters are the symbols the VIC puts in a message field when you hold down **CTRL** and press

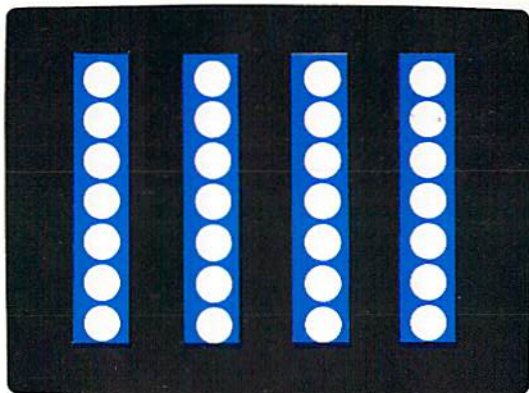
the **RVS ON** and **RVS OFF** keys.



After you have typed the new line, tell the VIC to RUN this example:

Type: RUN

What happened to the balls? Press the **CTRL** key to slow down the PRINTing. There are now columns of blue rectangles with white holes in them. Can you guess why?



Putting the **RVS ON** control character in front of the ball symbol, causes the VIC to *reverse* the colors being used to display each ball. The ball is normally blue and is surrounded by white background. The reversal made the background blue and the ball white.



Placing the **RVS OFF** control character after the ball, tells the VIC to reset the reversing of background and character colors. That is like reversing the reverse, and you end up back where you started. If the **RVS OFF** is not used, an interesting effect is produced.



the reverse balls if need be, and enter this line:

1 PRINT "

R

•

";

RVS ON

Space

Ball

Type: RUN

When **RVS OFF** is not used, all the spaces on the screen are reversed and become solid blocks. What an interesting pattern maker the VIC can become. How about designing your own textiles, or wallpaper, or floor tiles with the VIC? What can you think to do with the VIC?

THE CURSOR KEYS

The four cursor control keys let you move the cursor anywhere on the screen. You can put the cursor controls in PRINT statement message fields to help you position the characters and messages you are printing. Here is the bird example from the beginning of the chapter with the addition of a few cursor controls.

Clear the screen, and enter this new bird display:

NEW

1 PRINT " Q | ● | | | | ";



This line uses several cursor controls (**DOWN** , **RIGHT** , **LEFT**) to animate and move the bird image. You will see the movements when you RUN the entire example.

2 FOR T = 1 TO 150: NEXT T



3 PRINT " ● | | | | ";



This line contains three **LEFT** cursor control characters.

4 FOR T = 1 TO 150: NEXT T



5 PRINT " | | | | ";

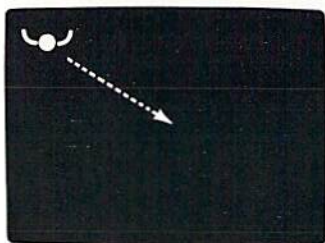


6 FOR T = 1 TO 150: NEXT T



7 GOTO 1

Type and check the example against the printed text. When you are ready, type RUN and watch the birdie.



The cursor control characters in this example moves the bird down and across the screen. Easy as bluebird pie!

Your turn again; why don't you invent another creature and m-o-v-e it across the screen.

ANIMATING WITH POKES AND PEEKS

So far you have learned to use the PRINT statement with the cursor controls and the color controls to move objects around the screen. This is probably the fastest way to move objects using BASIC, but it is not the smartest. Most arcade-style games require that the object moving around the screen reacts to other objects on the screen. The only practical way to program this is using the PEEK and POKE statements.

In order to use this technique, you must first understand the concept of memory-mapped video. Each position on the screen corresponds to a location inside the VIC's memory (RAM). Since there are 506 possible positions on the screen for characters, there are 506 locations in memory to hold the characters. And, since each location in memory can contain a number from 0 to 255, there are 256 possible values for each memory location. These are the 256 different characters that the VIC can make (see Appendix H). A number in a position in screen memory is a code for the character in that position.

Screen memory in the VIC normally begins at location 7680, and ends at location 8185. Location 7680 is the location of the upper left corner of the screen. Location 7681 is the position of the next character to the right of that one, and so on down the row. The next location following the last character of d row is the first character of the next row down.

Now let's say that we are controlling a ball bouncing on the screen. The ball is in the middle of the screen, column ten and row ten. (The upper left corner is column zero, row zero.) The formula for calculating the memory location on the screen is:

$$P = 7680 + X + 22 * Y$$

where X is the row and Y is the column. Therefore, the memory position of the ball is $7680 + 10 + 220$, or 7910. Clear the screen and type:

```
POKE 36879,8
```

This changes the color of the screen to all black. Now type:

```
POKE 7910,81
```

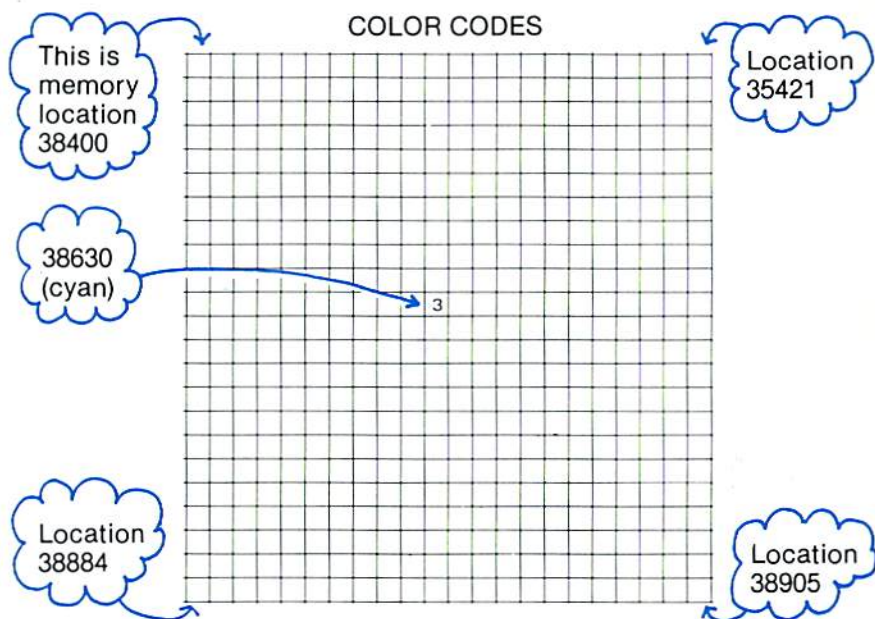
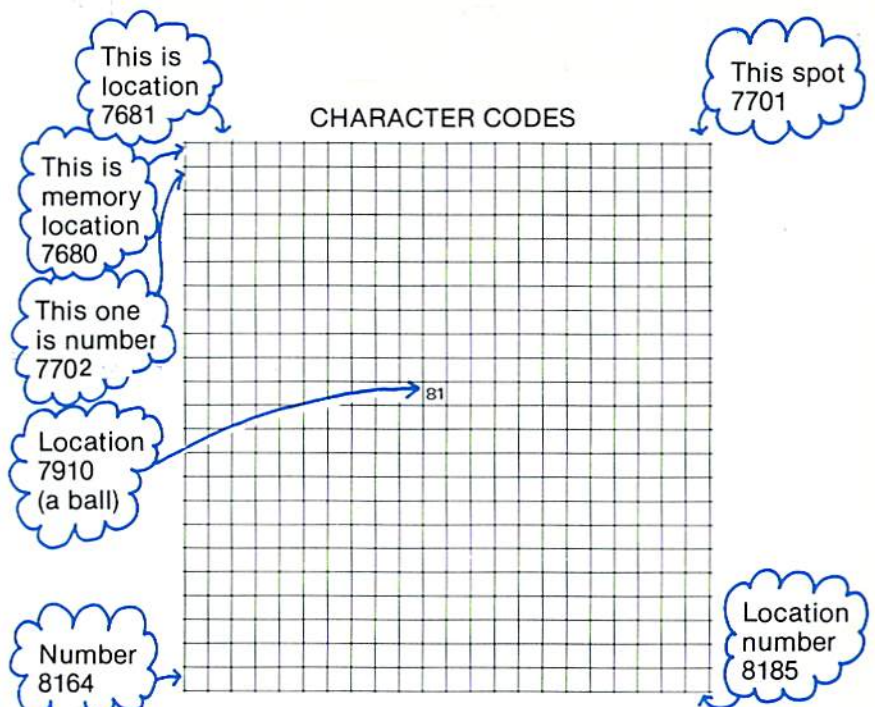
A ball appears in the middle of the screen! You have placed the character directly into screen memory without using the PRINT statement.

The ball that appeared was white. However, there is a way to change the color of an object at any location on the screen, also using POKES. Type:

```
POKE 38630,3
```

The ball's color changes to cyan. For every spot on the VIC's screen there are two memory locations, one for the character code and the other for the color code. The color memory map begins at location 38400 (top left-hand corner), and continues on for 506 locations. The color codes from 0 to 7 will give you the 8 colors numbered 1 to 8 on the keyboard. (Other numbers will give strange-looking results. See the VIC Programmer's Reference Guide for a complete explanation.)

SCREEN MEMORY MAPS



Here is a short program to bounce a ball around the screen, followed by a detailed explanation.

```
10 PRINT "  SHIFT  CLR HOME  "  
  
20 POKE 36879,9  
30 POKE 36878,15  
40 X = 1  
50 Y = 1  
60 DX = 1  
70 DY = 1  
80 POKE 7680 + X + 22*Y, 81  
90 FOR T = 1 TO 10: NEXT  
100 POKE 7680 + X + 22*Y, 32  
110 X = X + DX  
120 IF X = 0 OR X = 21 THEN DX = -DX: POKE 36876, 220  
130 Y = Y + DY  
140 IF Y = 0 OR Y = 22 THEN DY = -DY: POKE 36786, 230  
150 POKE 36876, 0  
160 GOTO 80
```

Line 10 clears the screen, and line 20 sets the color of the screen to black with a white border. Line 30 sets the loudness of the sound generators to the loudest level.

The X and Y variables used in lines 40 and 50 keep track of the current column and row position of the ball. The DX and DY used in lines 60 and 70 are the horizontal and vertical direction of the ball's movement. When a + 1 is added to the X value, the column is moved 1 to the right. When -1 is added to X, the column of the ball is moved 1 to the left. + 1 added to Y moves the ball down a row, and -1 added to Y moves the ball up a row.

Line 80 puts the ball character on the screen at its current position. Line 90 is a delay loop, leaving the ball on the screen just long enough for your eye to see it. Line 100 now erases the ball by putting a space (code of 32) where it was on the screen.

Line 110 adds the direction factor to X. Line 120 tests to see if the ball has reached one of the side walls, reversing the direction and beeping if there is a bounce. Lines 130 and 140 do the same for the top and bottom walls.

Line 150 turns off the sound, if any, to complete the bounce sound effect. Line 160 sends the program back to display and move the ball again.

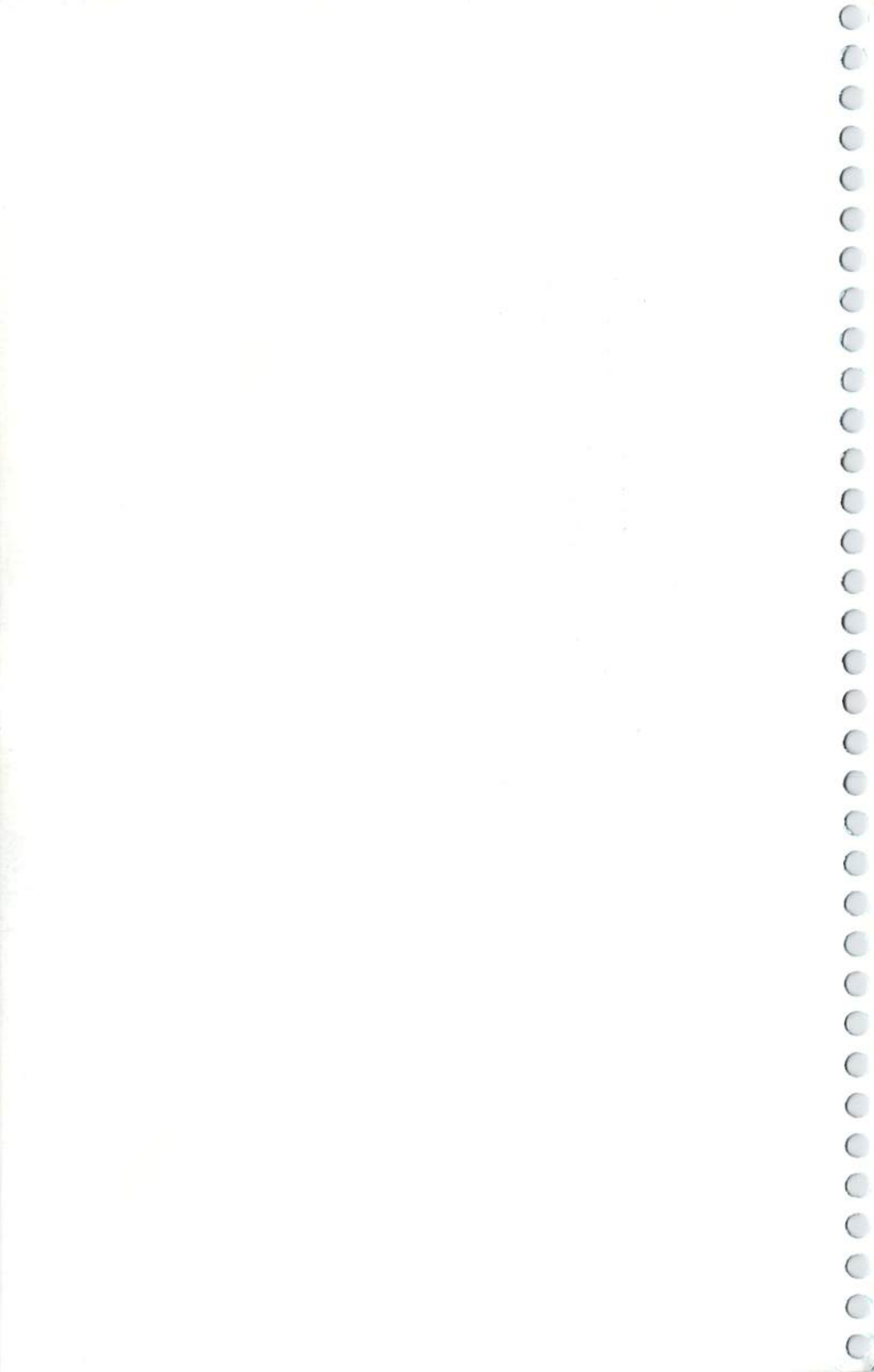
Play with this program a little. By changing the number in line 80 from an 81 to another character code, you can change the ball to any other character. If you change either DX or DY to 0, the ball will bounce straight instead of diagonally.

Now we can add a little more intelligence to this program. So far the only thing we check for is the X and Y values getting out of bounds for the screen. Add the following lines to the program shown above. (Just type these lines and they'll be added automatically)

```
32 FOR L = 1 TO 10
35 POKE 7680 + INT (RND (1) * 506), 102
37 NEXT
155 IF PEEK (7680 + X + 22 * Y) = 102 THEN DX = -DX: DY = -DY:
    POKE 36876, 180: GOTO 110
```

Lines 32 to 37 put 10 gray characters on the screen in random positions. Line 155 checks ("PEEKs") to see if the ball is about to bounce into a grey block, and reverses the ball's direction if so.

See Appendices H and I for more information on animation.



5

Sound and Music

- Making Notes
- The Four Voices of VIC
- The “White Noise” Generator
- Playing Songs
- Using the VIC as a Piano
- A Few Words about POKE

Try typing this program:

Type this program exactly as shown and see what happens!

NEW

```
10 PRINT "  SHIFT CLR HOME "
```

20 FOR I = 1 TO 5
30 POKE 36873 + I, 0
40 NEXT I
50 PRINT "WHICH VOICE (1-4)?"
60 PRINT "IF DONE, ENTER 0"
70 INPUT N
80 IF N = 0 THEN END
90 PRINT "WHICH PITCH (128-254)?"
100 INPUT P
110 POKE 36878, 4
120 POKE 36873 + N, P
130 FOR J = 1 TO 2000: NEXT J
140 GOTO 10

To stop the program, press the **RUN STOP** key.



MAKING MUSIC

You may not realize it, but you and your VIC can make music and sound effects! This chapter introduces you to the music and sounds that the VIC can make. It will teach you how to control these sounds and play any kind of music from Bach — to Rock. So put your candelabra on your TV set, and get ready for a concert!

First, let's find out how to play a note:

Hold down the **SHIFT** key and press the **CLR HOME** key.

N E W and press the **RETURN** key.

P O K E **SPACE** **3 6 8 7 8 , 3**

and press **RETURN** .

P O K E **SPACE** **3 6 8 7 5 , 2 2 5**

and press **RETURN** .

At this point, you should hear a tone coming from your TV speaker. If you don't, try adjusting the volume control on the TV in the normal way. Set it so that the tone is comfortably loud...like music on a TV show.

You've just played middle C, one of 128 notes in VIC's repertoire! To turn off the tone, type this:

P O K E **SPACE** **3 6 8 7 5 , 0**

Next, let's find out how many notes the VIC can sing:

Type **NEW** and press **RETURN**



This tells VIC you're ready to write a new program. It automatically erases the old one.

Now, type this program:

1 POKE 36878, 15

RETURN

2 FOR I = 128 TO 255

RETURN

3 POKE 36875, I

RETURN

4 FOR D = 1 TO 100

RETURN

5 NEXT D

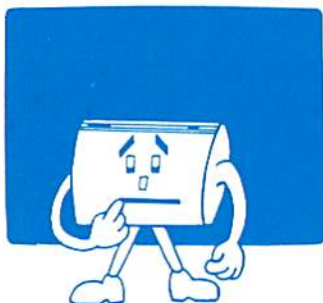
RETURN

6 NEXT I

RETURN

7 POKE 36875, 0

RETURN



Take a little time to look at the VIC's screen, and see if all of the lines on it look just like these. If any line looks wrong, just type it over again. Make sure you include the line number. VIC will automatically substitute the new line for your previous version. To get a listing of the lines in your program, type:

LIST and press

RETURN

If all the lines are OK, type:

RUN and press

RETURN

and you will hear all 128 notes that the VIC can sing with this voice. (If you don't, recheck the lines as you typed them.) If you want to hear the notes again, type:

RUN again, and press

RETURN

Now that you have heard the notes that your VIC can make, we want to explain a little bit about how it does all that.

Inside the VIC, the Video Interface Chip (which gives the VIC its name) handles both sound and picture. It lets your VIC send sounds to the speaker of your TV set. The volume of sound can be adjusted by the volume control on the TV or from the VIC's keyboard.

THE FOUR VOICES OF VIC

Your VIC has four voices...that is, it can "sing" four different notes to you at the same time! You might think of them as soprano, alto, tenor, and noise. Each of the voices has a particular "speaker control number". By using this number, we can turn the speaker "on" and use it to create a musical note or sound effect. We use the word POKE to do this.

The VIC's four speaker numbers are:

36874 (speaker 1 - music)

36875 (speaker 2 - music)

36876 (speaker 3 - music), and

36877 (speaker 4 - noise).

The volume control number is 36878.



36874 ... (alto)



36875 ... (tenor)



36876 ... (soprano)



36877 ... (noise)
speaker 1



36878 ... (volume)

To play a note, type POKE and the speaker number, a comma, and a number representing the note you want to play.

For example, type:

POKE 36874, 128 and press

RETURN

You've just sounded note #128 using speaker #36874, the first (lowest) voice. From now on, let's call this speaker S1 (short for speaker 1); the second, S2; the third, S3; and the fourth, S4. S2 sings higher notes than S1 does, and S3 sings higher than S2. S4 sings with a buzz in its voice...this "voice" creates "white noise" used for sound effects.

By the way, to turn off a voice, POKE 0 into it, like this:

POKE 36874, 0 and press **RETURN**

This might seem a little complicated if you're not a "computer musician" yet, so we've made it easier for you. You can save a lot of typing if you use the following short program to translate the speaker numbers into shorter numbers.

Type this on your VIC exactly as shown:

NEW

S1 = 36874

RETURN

S2 = 36875

RETURN

S3 = 36876

RETURN

S4 = 36877

RETURN

V = 36878

RETURN

This allows us to refer to the speaker numbers by the abbreviations S1, S2, S3, S4, and V (volume). Before proceeding, check to make sure that each line has been entered correctly. Now you are ready to make sounds the easy way!

POKE V, 10

RETURN

This sets the volume

This POKEs 10 into the location that controls volume. The volume control can store any value between 0 and 15. The higher the number, the louder the volume.

POKE S1, 195
POKE S2, 215
POKE S3, 231

Experiment a bit with this method of producing sounds. It's much easier, isn't it? When you want to stop the sound (to let your poor ears rest), POKE 0 into the speakers you want to turn off, like this:

POKE S1, 0
POKE S2, 0
POKE S3, 0

Here is a chart of the values which can be POKEd into the speakers to get various notes: (**Note:** numbers below 128 produce "silence"):

TABLE OF MUSICAL NOTES

NOTE	VALUE	NOTE	VALUE
C	135	G	215
C#	143	G#	217
D	147	A	219
D#	151	A#	221
E	159	B	223
F	163	C	225
F#	167	C#	227
G	175	D	228
G#	179	D#	229
A	183	E	231
A#	187	F	232
B	191	F#	233
C	195	G	235
C#	199	G#	236
D	201	A	237
D#	203	A#	238
E	207	B	239
F	209	C	240
F#	212	C#	241


By POKeing values into the first three voices (S1-S3), you can even play tunes. ("White noise" isn't really appropriate to this

kind of thing...unless you dream of playing first-chair computer with Pink Floyd!) Unfortunately, this POKE process is very slow and tedious for a human...but it's child play for your VIC. So let's put the POKES in a program and let the computer do the work! Anyone for "Flight of the Bumble Bee"? How about the "Maple Leaf Rag"?

THE "WHITE NOISE" GENERATOR

The fourth voice, S4, is numbered 36877. We've called it "noise" because it's actually a "white noise" generator, used primarily for special effects. Try making the buzz of an airplane:

POKE V, 6
POKE S4, 130



a very low tone

It's a four-engine prop job, right? Now, how about the wind whipping by the wings of a sailplane?

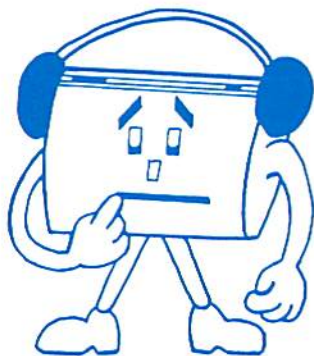
POKE S4, 240

Don't forget to turn on the volume with:

POKE V,4

or whatever volume setting you like best. Now let's turn off the sound:

POKE S4, 0



PLAYING SONGS

Using the VIC speakers and table of musical note values, you can make up your own songs, or transcribe tunes from a songbook. The following songwriting program shows you how to do it:

Type:

```
NEW  
10 S2 = 36875  
20 V = 36878
```

REM means this line is not an instruction, but a REMark or comment to yourself or others

```
100 REM READ AND PLAY LOOP  
110 POKE V,15
```

set volume

```
120 READ P  
130 IF P = -1 THEN 200  
140 READ D
```

end of melody marker

VIC will look for information

```
150 POKE S2,P  
160 FOR N=1 TO D: NEXT N  
170 POKE S2,0  
180 FOR N=1 TO 20: NEXT N  
190 GOTO 120
```

plays pitch P for duration D

silence is golden

back for more

```
200 REM IF NOTE -1 THEN STOP  
210 POKE V,0  
220 END
```

another REMark

```
300 DATA 225, 1000, 228, 1000, 231, 1000  
310 DATA 232, 1000, 235, 1000, 237, 1000  
320 DATA 239, 1000, 240, 1000  
330 DATA -1
```

pitch values

duration values

the end

In the first group of instructions then, we've set up the POKE locations for the speaker(s) we're using, in this case speaker 2 and volume, and entered our abbreviations. The next group of instructions begins at line 100.

Line 100 contains a REMark which explains what this section is supposed to do. It's called a "loop" because the section will read and play one note, then "loop" back to the beginning and do it again for another note. Line 110 turns up the volume.

Now we'll tell the VIC to find out what note to play:

VICTIP:

Programs don't have to start with line #1, or be numbered by 1. Most programs start at 10 and go up in increments of 10. This way you can go back and add extra lines in between if you want. For example, you could add a line 11, 12, etc. between lines 10 and 20.

Line 120 tells the VIC to look through the program and READ information — call it P — about what note to play. This information is contained in a "mystery statement" that we haven't written yet. Similarly, line 140 tells the VIC to READ information — call it D — about the duration of the note.

Notice especially line 130. The function of this line is to stop the program when the last note has been read. Without some kind of "end of melody" marker, the program would try to read notes that haven't been written, and make an error. Line 130 says that when the VIC reads this marker, a value of -1, it should not try to play this note, but go to an ending module at line 200. We must remember to place -1 at the end of our "mystery statement."

Now we'll have the VIC actually play the note, cut it off, and go back for another:

Line 150 simply POKEs the note we READ in line 120 into voice 1, while line 160 creates a delay for the duration we READ in line 130. Similarly, lines 170 and 180 turns off voice 1 for a short period. Line 190 sends the VIC back to line 120 to READ the next note.

Now we have to write our ending section. Remember that our "end of melody" marker, -1, sends the program to line 200.

Line 210 turns the voice off, and line 220 tells the VIC to stop performing the instructions in this program.

Even though we've written the ending section, we're not quite done. We still have to write our "mystery statements" to tell the program what notes to READ. These mystery statements are called DATA statements because they contain information, or data. DATA statements can be located anywhere in a program. Whenever the VIC encounters a READ instruction, it looks around for a DATA statement to READ.

This module contains the DATA for a C major scale. Line 300 contains the first three notes. The first number is the POKE value for the first note, 225 — low C. The second number sets its duration, 1000 — about 1 second. Line 310 contains the next three notes, and line 320 the last two. The values themselves are taken from the table of musical notes above.

If the lines look correct, you're ready to RUN. You should hear a fairly accurate C major scale! If you have a "clinker" or two, try adjusting the values in your DATA statements, starting at line 300.

Once again: You can put DATA statements anywhere in your program. They will be READ one by one, starting with the lowest line number, and working through each DATA statement from the beginning to the end.

Try substituting DATA statements to play other selections. For example, here's an old family favorite:

```
300 DATA 225, 360, 225, 360, 225, 240
310 DATA 228, 120, 231, 360, 231, 240
320 DATA 228, 120, 231, 240, 232, 120
330 DATA 235, 720, 240, 360, 235, 360
340 DATA 231, 360, 225, 360, 235, 240
350 DATA 232, 120, 231, 240, 228, 120
360 DATA 225, 480
370 DATA -1
```



use as many lines as you like to play longer selections but for the purpose of this program, stick to 3 notes per line.

Or if you prefer classics:



```
300 DATA 217, 400, 213, 400, 223, 400
310 DATA 227, 200, 234, 200, 230, 400
320 DATA 227, 200, 234, 200, 230, 400
330 DATA 223, 400, 227, 400, 217, 400
340 DATA 213, 600, -1
```

THE VIC AS PIANO

Finally, here's a program that lets you play the VIC's keyboard like a piano:

NEW

```
10 REM STORE SOUND REGISTERS
20 S2 = 36875
30 V = 36878
40 POKE S2, 0
```

Abbreviates the voice registers we'll need and turns them off

```
100 REM STORE B MAJOR SCALE
110 FOR N = 1 TO 8
120 READ A (N)
130 NEXT N
```

Reads B major scale from lines 140-160

```
140 DATA 223, 227, 230
150 DATA 231, 234, 236
160 DATA 238, 239
```

Contains POKE values for B major scale

```
200 REM PLAY KEYBOARD
210 POKE V, 3
```

Turns on the volume

Finds out what key is being pressed

```
220 GET A$: IF A$ = " " THEN 220
230 N = VAL (A$)
```

Ends the program if you've pressed "0" or "9"

```
240 IF N = 0 OR N = 9 THEN 300
```


250 POKE S2, 0	Brief silent interval
260 FOR T = 1 TO 25: NEXT T	between notes...Shhhhh
270 POKE S2, A (N)	Plays the tone
280 GOTO 220	and returns to
	look for another
300 REM ENDING MODULE	
310 POKE S2, 0	Turn off the sound
	before you go

Now, when you type RUN (and press **RETURN**), you can play tunes on your VIC. The keys in the top row with numbers on them control the various notes:

```

*****
  1     2     3     4     5     6     7     8
  DO   RE   MI   FA   SOL  LA   TI   DO
*****

```

The VIC will keep playing the note you hit last until you hit another note. When you're done, press either 0 or 9, and it will turn off. To start the VIC piano again, just reRUN the program.

Try the following (sing along if you wish):

```

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  8
9

```

OR:

```

3  3  4  5  5  4  3  2
1  1  2  3  3  2  2
3  3  4  5  5  4  3  2
1  1  2  3  2  1  1
0

```

Take it away, Ludwig!

A FEW WORDS ABOUT POKE

The command POKE lets you deal with your VIC on a completely new level. POKE allows you to find a particular memory location and change what is stored there. Since this command operates directly on the VIC's memory, it is possible to make mistakes by POKEing values into the wrong locations, or wrong values into the right locations! We want to repeat what we told you way back in Chapter One: *There is no way you can hurt the computer by typing on the keyboard...* not even with POKE. But you can cause the VIC to just go away someplace and sulk, cutting you off from any contact. For example, if you're ready to end this session anyway, try typing:

```
POKE 788, 0
```

You may find that the only way you can regain communication with a computer that has been "insulted" in this way is to type RUN/STOP and RESTORE. If the "crash" is serious, you may have to turn it off and turn it on again. This doesn't harm the computer, but it does mean that whatever program you were working on will be lost. Even if this should happen to you, a little typing time is normally the extent of the loss. But it does suggest that you should be careful of what you POKE. It also underlines the value of accessories like disk drives and data cassettes, which can store programs, as well as printers, which can at least give you a program listing that would help you reconstruct a lost program.

We recommend that you make a brief study of POKE in Appendix C before you begin to POKE around indiscriminately in your VIC. At least try the examples we give you in this book. Be careful typing those long numbers, and double check your work before you run your programs. A computer that has been POKEd in the wrong place may well reason that turnabout is fair play, and simply "turn you off."

In this chapter, you've learned how to make Bach and Beethoven sit up and take notice. You know how to drive your friends crazy and make them long for the good old days when your VIC was a quiet, mild-mannered little creature that kept pretty much to itself. So what if the VIC isn't quite as talented as Beverly Sills or a Steinway concert grand? You and VIC can still make beautiful music together! Maestro, please...

6

Conversing With Your VIC

- **What's Your Name?**
- **Introducing Variables**
- **Choose a Note**
- **The GET Statement**

Try typing this program:

Type this program exactly as shown and see what happens!

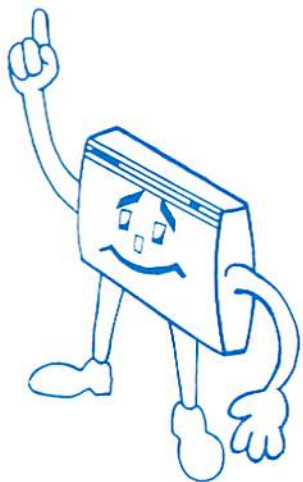
```
10 INPUT"DEGREES FAHRENHEIT";F
20 PRINT F"DEGREES F."
30 PRINT"IS"(F-32)*5/9"DEGREES C."
40 PRINT ←
50 GOTO 10
```

Typing the word *PRINT* by itself on a line adds a blank line when the program is run. Try it with and without this line

To stop the program, press the **RUN STOP** key.

and hit

RESTORE



Good morning VIC. Your mission (and we know you are going to accept it) is to perform miracles whenever we touch just a few keys!

Yes, your VIC is ready, able and willing to respond to your touch like no other computer ever made. So let's buckle down to some non-stop fun!

Hold down the **SHIFT** key and press the **CLR HOME** key.

Type the following keys:

N E W and press the

RETURN key.

This is the number one, not "L"

Press the space bar, not the letters

Don't forget the ; semicolon!

1 I N P U T
" W H A T ' S **SPACE**
Y O U R **SPACE**
N A M E " ; A \$

and the **RETURN** key.

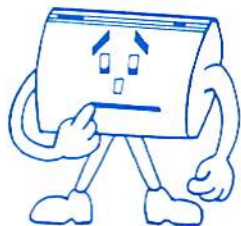
2 P R I N T "

H I , **SPACE** **" A \$**

and the **RETURN** key.

3 G O T O 2 and the **RETURN** key.

Now press **R U N** and **RETURN**



And the VIC responds:

```
1 INPUT "WHAT'S YOUR NAME";A$
```

```
2 PRINT " HI, " A$
```

```
3 GOTO 2
```

```
RUN
```

```
WHAT'S YOUR NAME? ←
```

Notice that the computer automatically adds the question mark (?) to show you that it's waiting for an input.

At which point you type in the customary response or if your name happens to be VIC you can type:

V **I** **C** and

RETURN

And whoopee! Your name splashed all over the screen! To slow down the show just press the **CTRL** key. When you

are ready to leave stardom behind, press the **RUN STOP** key.

To enter another name just type RUN and you will see the same *prompt* again. The prompt in this case is:
WHAT'S YOUR NAME?

Prompts are questions directed to you in an effort to get information into the VIC. The VIC has a limited number of ways to get information from the human world. Perhaps the most useful method for the computer to collect information from our keyboard is the INPUT statement. Let's go through the steps of our program. Here's what we told VIC to do:

First, display the message "WHAT'S YOUR NAME" on the screen and then wait for you to put in (or INPUT) characters from the keyboard. Take the response and name it "A\$". In our example "VIC" becomes A\$. This is a kind of shorthand for the computer.

Second, print the word "HI" followed by whatever was typed at the keyboard. In our example we print HI VIC.

Third, go back to line number 2.

The second and third steps combine to make the VIC continuously print the message "HI" all over the screen. If we changed line 3 to read:

```
3 GOTO 1
```

then the message HI VIC would print on the screen only once and the message prompt "WHAT'S YOUR NAME" would appear again. This alteration makes the VIC appear to have amnesia!

Whenever we use the INPUT statement, our program holds everything while awaiting a response from you. It is important to note that VIC will wait forever or until the **RETURN** key is pressed, whichever comes first.

In our program we created a friendly prompt. Unless we tell the VIC what we want our input message to say, we will just get a simple "?" which does not tell us much. So we will try to build prompts which suggest what type of input is required.

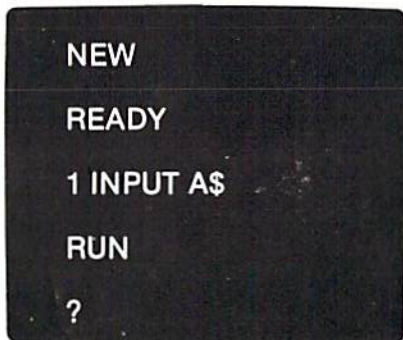
Press **RUN STOP** and **RESTORE** at the same time.

Type NEW and **RETURN**

```
1 INPUT A$
```

When we type RUN the VIC screen looks like:

The question mark alone raises more questions than it answers! So let's try this:



```
NEW
READY
1 INPUT A$
RUN
?
```

```
1 PRINT "MAY I HAVE YOUR NAME":INPUT A$
```

```
2 PRINT "WHAT IS YOUR FAVORITE FOOD":INPUT B$
```

```
3 PRINT
```

```
4 PRINT "THANK YOU,"A$" FOR YOUR POLITE ANSWER"
```

```
5 PRINT "WE WILL GIVE YOU SOME SPACE "B$" SPACE
      AS A GESTURE OF OUR APPRECIATION"
```

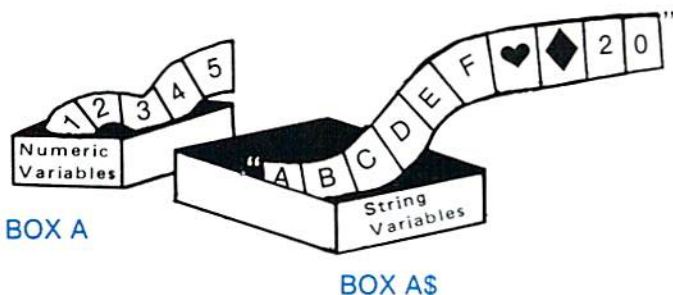
After typing RUN and **RETURN** you will see that treating messages with respect can result in a treat for your efforts!

INTRODUCING VARIABLES

Many of the programs used in this book use variables to simplify or strengthen the program.

Variables are very useful because they can be used to represent numbers, formulas, graphic symbols, words, phrases—even whole sentences. Examples of variable names are: X, AB, S2, X\$, AB\$, S2\$. The easiest way to explain the power of variables is to tell you that these simple variables can each be used to represent up to 255 characters!

There are two kinds of variables: numeric variables and string variables. Numeric variables are used to store numbers (actually, numeric values). String variables can be used to store all types of characters (numbers, letters, graphics, cursor controls, color controls, etc.).



Variables are like storage cabinets inside the computer. To tell the VIC that cabinet is for numbers or values, we must use a special name. Numeric variable names may be one or two characters long and may be one letter, two letters, or a letter and a number. Here are some examples of numeric variable names:

X AB S2 C2 AA ZX

String variable names may be one, two or three characters long (including the \$ sign), must always begin with a letter from A to Z and have a dollar (\$) sign at the end. Here are some string variable examples:

X\$ AB\$ S2\$ C2\$ AA\$ ZX\$

Here's a short program that shows one way to use variables:

```
10 A$ = "VIC 20"  
20 PRINT "HELLO," A$  
  
RUN
```

The VIC will display HELLO, VIC 20. Why? Because in Line 10 we told the VIC that the variable A\$ is the same as "VIC 20". Now try this:

```
10 A = 2  
20 B = 3  
30 C = 4  
  
40 PRINT A * B * C
```

In this example, A*B*C is the same as 2*3*4 because we "stored" the numbers 2, 3, and 4 in the variables A, B, and C.

Here's a final example that uses two kinds of variables with INPUT statements. The INPUT statement allows the person running the program to define what the variable will stand for like this:

```
10 PRINT "WHAT WORD DO YOU WANT X$ TO STAND FOR":  
   INPUT X$  
  
20 PRINT "WHAT NUMBER DO YOU WANT X TO STAND FOR":  
   INPUT X  
  
30 PRINT "NOW X$ STANDS FOR" X$  
  
40 PRINT "AND X STANDS FOR" X  
  
RUN
```

CHOOSE A NOTE

For this INPUT example we will tinker a little with VIC sound.

Type:

NEW

10 INPUT"HOW HIGH A NOTE";H

20 IF H=0 THEN 90

30 INPUT"HOW LONG A NOTE";L

40 POKE 36878,15

50 POKE 36875,H

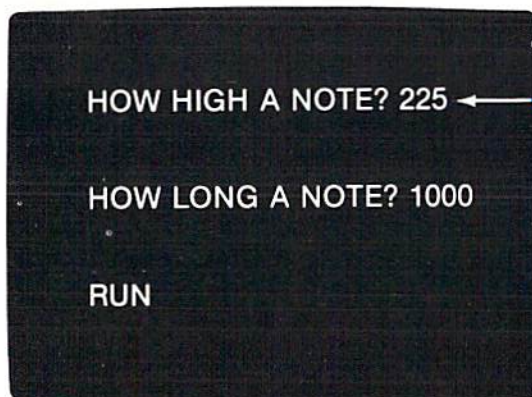
60 FOR T = 1 TO L:NEXT

70 POKE 36878,0

80 GOTO 10

90 END

After you press **RETURN** the screen should look like this:



This letter is a number variable and is used to define or input a number

If a note keeps playing, hit **RUN STOP** **RESTORE**

Type a number from 128 to 254

Press the **RETURN** key and listen to the pitch and duration of the note. After each note you can create another and gain an appreciation for how these "mysterious" POKES translate into sound.

The first two lines of "make a note" (10 and 20) give us the flexibility necessary to vary the pitch in line 50 and 60 respectively. Line 30 is our "door" out when we are finished experimenting (just type 0 (zero) in response to HOW HIGH A NOTE?). Line 70 turns off the voice which was activated earlier in line 40.

If you want all the noteworthy details on making music with your VIC, turn to chapter five.

THE GET STATEMENT

Now that you have mastered the INPUT statement, we will move on to a fancier way of getting information from the keyboard.


The GET statement is used to get characters from the keyboard one character at a time. In fact, the person RUNNING the program need not even hit the RETURN key! Here's how the statement looks!

```
10 GET A$
```

How do we make a program stop and wait for something to be typed? We put the program in a loop with an IF...THEN statement checking for an answer.

```
10 GET A$
```

```
20 IF A$ = "" THEN 10
```



No space
between
the quotes

What is the use of this? For a very simple application, the little 2 line program above will allow your program to pause until the operator hits a key on the keyboard. This is helpful in freezing a display on the screen until the person has read it and wants to go on.

Here is an expanded application for the GET statement:

```
10 GET A$
```

```
20 IF A$ = "" THEN 10
```

```
30 IF A$ = "A" THEN PRINT "CHICKEN SOUP"
```

```
40 IF A$ = "B" THEN PRINT "SPAGHETTI"
```

50 IF A\$ = "C" THEN PRINT "STEAK AND EGGS"

60 GOTO 10

RUN

When this program has been typed in and RUN, it will wait for the operator to hit any key. If the key was the letter A, the words, CHICKEN SOUP appear on the screen. The letter B makes the word SPAGHETTI appear, and the word C makes the words STEAK AND EGGS show up. You now have the VIC typing whole words with only 1 keystroke!

Now you will get the longest program so far, a practical example of the GET and PRINT statements used to give you a computerized *recipe file*. Don't be alarmed by the size of this program. It uses mostly simple PRINT statements but the lesson here is how the A\$ variable is used to stand for several whole phrases, and how the GET command

10 PRINT "   PLEASE PICK A CHOICE"

20 PRINT "FROM THE MENU:"

30 PRINT

40 PRINT "A...CHICKEN SOUP"

50 PRINT "B...SPAGHETTI"

60 PRINT "C...STEAK & EGGS"

200 GET A\$:IFA\$ = "" THEN 200

210 IF A\$ = "A" THEN 500

220 IF A\$ = "B" THEN 700

230 IF A\$ = "C" THEN 900

490 GOTO 200

500 PRINT "   MIKE'S CHICKEN SOUP"

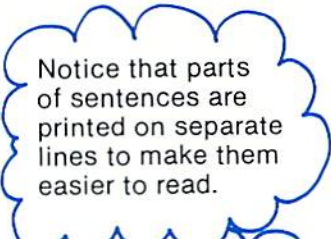
510 PRINT

520 PRINT "TAKE 1 CHICKEN. KILL"

Try typing your own recipes! All you have to do is change the titles and recipe information...

This means if you hit anything else, nothing happens and the program keeps waiting for you to type A, B or C.

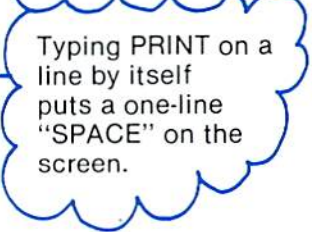
530 PRINT "AND PLUCK. REMOVE"
540 PRINT "GIBLETS. BOIL 4 QTS"
550 PRINT "WATER IN A LARGE POT."
560 PRINT "ADD CHICKEN. BOIL"
570 PRINT "2 HOURS, OR UNTIL"



Notice that parts of sentences are printed on separate lines to make them easier to read.

580 PRINT "HOUSE SMELLS GOOD."

590 PRINT



Typing PRINT on a line by itself puts a one-line "SPACE" on the screen.

600 PRINT "HIT ANY KEY TO GO ON"

610 GET A\$:IF A\$ = "" THEN 610

620 GOTO 10

700 PRINT "   MA'S SPAGHETTI"

710 PRINT

720 PRINT "BROWN 1 LB. GROUND"

730 PRINT "BEEF, WITH 1 ONION"

740 PRINT "AND 1 GREEN PEPPER."

750 PRINT "ADD 1 LG. CAN TOMATO"

760 PRINT "PUREE, 6 OZ. CAN TOM."

770 PRINT "PASTE, 6 OZ. WATER,"

780 PRINT "3 CLOVES GARLIC, SALT"

790 PRINT "& PEPPER, RED PEPPER,"

800 PRINT "OREGANO. SIMMER 1 HR"

810 PRINT "& SERVE WITH COOKED"

820 PRINT "NOODLES."

830 GOTO 590

900 PRINT " **SHIFT** **CLR HOME** STEAK AND EGGS"
910 PRINT
920 PRINT "TAKE 1 COOKED STEAK"
930 PRINT "AND COOKED EGGS."
940 PRINT "SERVE TOGETHER WITH"
950 PRINT "BEVERAGE."
960 GOTO 590

If you typed the program correctly and typed RUN, the screen should come up with the following display:

**PLEASE PICK A CHOICE
FROM THE MENU:**

**A...CHICKEN SOUP
B...SPAGHETTI
C...STEAK & EGGS**

Now the VIC is waiting for you to hit a key. If you type anything other than A, B, or C, nothing happens at all (Line 200 does this). If you hit the A, you get the recipe for Mike's Chicken Soup. Pretty terrible, huh? You can tell Mike is a bachelor.

This can be lengthened and modified very easily for your own use. To add items to the menu, just add a PRINT statement after line 60, add a new IF statement after line 230, and add the recipe wherever there is room at the end. The last line of your recipe should be the line GOTO 590, which tells the person RUNNING the program to hit a key to continue. This will keep the recipe on the screen until they are through with it.

You can use the program we just described for more than recipes, of course. How about a name and address file? Instead of a "menu" use last names with initials. Instead of recipes, use the person's name, address and phone number. Can you think of other uses for GET and INPUT based programs? This is the true power of computing—being able to tailor what the computer does to your own needs.

7


Introduction to Programming

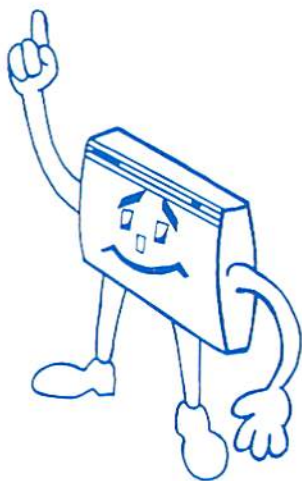
- Your First BASIC Commands
- Random Numbers

Try typing this program:

Type this program exactly as shown and see what happens!

```
10 PRINT "   "  
20 PRINT CHR$(205.5 + RND(1));  
30 GOTO 20
```


To stop the program, press the  key.



YOUR FIRST BASIC PROGRAMS AND HOW THEY WORK

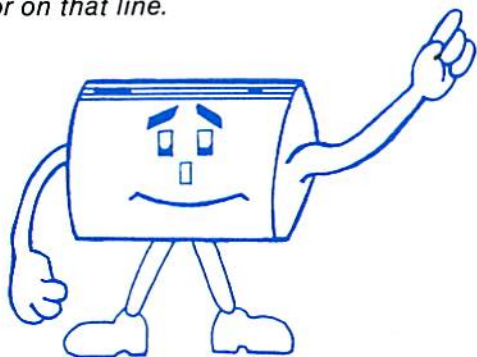
Until now, you've been patient and typed in several programs without understanding how they work. This chapter will explain what those tricky little programs were all about, and get you further along the road to programming your VIC.

PROGRAM 1: Your Name In Lights (Chapter 2)

```
10 PRINT "   "  
20 FOR T = 1 TO 300: NEXT  
30 PRINT "your name"  
40 FOR T = 1 TO 300: NEXT  
50 GOTO 10
```


This program and the ones to come are taken from the "sample programs" beginning each chapter.

There doesn't seem to be much to this little program, but once you understand what's going on here, you have the key to doing animation. Line 10 is the PRINT command, with the character meaning "clear the screen" inside the quote marks. If you tried to type this without the quotes, your program line disappeared from the screen before you could finish typing it. *The VIC only recognizes a new line when you hit the RETURN key with the cursor on that line.*



VIC TIP: QUOTES

Let's talk a little more about what happens when you hit the quote key. The first time you hit the quote, something funny happens. If you hit HOME, CLR, cursor up, down, left, or right, you get a *reversed graphic character*. This also happens on the VIC when any of the color control keys are pressed. You see, quotation marks are used in computer programs and the VIC recognizes quote marks as a Programming Command. Therefore, when you hit a color or cursor control key after a quote mark, the VIC displays a special code to designate that

operation. For example when you type " and 



" you get a reverse heart on the screen. If you see

this in a program you know it means clear screen. Other symbols stand for other operations. Once you have hit the quote key for the *second* time, any cursor controls and color keys will work normally.

The INSERT key causes a similar effect. When this key is hit, every space created on the line will act as if it was in quote mode. All cursor control characters will appear as if they were inside quote marks. In addition, the DELETE key will produce a special reversed graphic character in these spaces, which will have the effect of deleting characters when the program is listed, and printing DELETEDs on the screen when PRINTed.

Line 20 of the program is called a *time delay loop*. This is actually two BASIC statements on one line, separated from each other by the colon (:). All that happens here is that the VIC will count from 1 to 300, without doing anything else. This serves to slow down the program a little. (Try deleting the lines 20 and 40 and RUNning the program. It blinks too fast!)

In line 30, you typed your name inside the quotes (At least we hope you did.) This caused your name to be printed on the screen. There was nothing else on the screen because line 10 already cleared it off.

Line 40 is another delay, to give your name time to be on the screen long enough to see.

Line 50 causes the program "go to" line 10 as the next line to be executed and briefly clears the screen.

After your name is displayed on the TV screen, there is a delay of a second, after which the screen is erased. After another second, the name is displayed again in the same position. Again, it is erased, and so on. Because the letters appear in the same position on the screen, your eye believes that they are blinking on and off.

Experiment with this program. You can make the delays between displaying and erasing your name longer or shorter by changing the number 300 in lines 20 and 40.

PROGRAM 2: A Lot of Heart
(Chapter 3)

```
10 FOR H = 1 TO 505
20 PRINT " ♥ ";
30 NEXT
40 FOR C = 8 TO 255 STEP 17
50 POKE 36879, C
60 FOR T = 1 TO 500: NEXT
70 NEXT
80 GOTO 40
```

This program provides you with a colorful display of hearts. It introduces the use of punctuation marks in PRINT statements and the use of POKE to change the screen and border colors.

Line 10 sets up a loop that counts from 1 to 505. We want 505 hearts to appear on the screen, because there are 506 spaces on the screen. If we PRINTed the 506th character, the screen would be forced to roll up one line (*scrolling*), and there would be *less* hearts on the screen than before.

Line 20 PRINTs the heart character on the screen. The semi-colon (;) after the last quote has an important effect. You see, after a normal PRINT statement, the VIC will automatically perform 2 operations — move the cursor back to the beginning of the line (called a *carriage return*), and move the cursor down to the next line (called a *linefeed*). The punctuation mark at the end of the line will cancel the return and linefeed, so that the next thing PRINTed will appear immediately to the right of the last thing PRINTed.

Line 30 just completes the delay loop. As long as the value of H is 505 or less, the program will print hearts on the screen. When the 505th is printed, the program continues with the line after this one (line 40).

Line 40 establishes a new loop and sets up line 50, which changes the screen and border colors. C is defined as a series of numbers from 8 to 255, which increase in increments of 17. Every time line 70 (NEXT) is hit, 17 is added to the previous value of C and the sum is used for the new value. This causes a cycle of colors to be selected including black border with black screen, white border with white screen, etc., for all 8 border colors. Then the numbers in C go beyond the values for those colors and pick 7 different colors for the screen. (See Appendix I for a list of color numbers)



















Line 50 is the statement that actually changes the color: The value contained in variable C is stored in memory location 36879. This is actually a location on the VIC chip itself, not in your normal area of memory.

Line 60 is a delay loop. If this line is removed, the colors will change fast enough to give you a headache.

Line 70 completes the loop started in line 40. Notice that the line could have read 70 NEXT C, but doesn't have to be that way.

Line 80 sends the program back to cycle through the colors again. It will run forever, unless you press the STOP key, or turn off the VIC. Typing POKE 36879, 27 after you hit STOP will restore the normal colors of the screen.

PROGRAM 3: Exercising VIC Person (Chapter 4)

```
10 PRINT " CLR HOME ";
20 PRINT "    "
30 PRINT "    "
40 PRINT "    "
50 FOR T = 1 TO 300: NEXT
60 PRINT " CLR HOME ";
70 PRINT "    "
80 PRINT "    "
90 PRINT "    "
100 FOR T = 1 TO 300: NEXT
110 GOTO 10
```

This program is similar to the first program we did, called *Your Name In Lights*. However, instead of drawing an image and then blanking it out, like the first program did, this one draws a complete picture, pauses, and replaces the image with another complete picture. The head and body of the VIC person stays in the same position while the arms and legs change places. This gives the illusion of movement from one position to the next.

Lines 10 and 60 bring the cursor to the upper-left corner of the screen, which is known as the *home position*. This forces the image of the VIC person to be displayed in the same screen position each time.

Lines 20, 30, and 40 will "draw" each line of the VIC person's first image.

Lines 50 and 100 are delay loops, just to give the picture enough time on the screen.

Lines 70, 80, and 90 draw the second image on the screen. This takes place so fast that you can't see the transition — your eye sees the change from one to the other as instantaneous.

PROGRAM 4: Choose A Note
(Chapter 5)

NEW

```
10 PRINT "  SHIFT  CLR HOME  "  
20 FOR I = 1 TO 5  
30 POKE 36873 + I, 0  
40 NEXT I  
50 PRINT "WHICH VOICE (1-4)?"  
60 PRINT "IF DONE, ENTER 0"  
70 INPUT N  
80 IF N = 0 THEN END  
90 PRINT "WHICH PITCH (128-254)?"  
100 INPUT P  
110 POKE 36878, 4  
120 POKE 36873 + N, P  
130 FOR J = 1 TO 2000: NEXT N  
140 GOTO 10
```

These lines clear the screen and turn off all voices

These lines allow you to select a voice

this line ends the program

These lines allow you to select a pitch

These lines poke the note and volume, and provide a 2-second delay

Now we go back to the beginning

This means the VIC wants you to input a number

When you've checked that all the lines are correct, try running this program (type RUN and press RETURN). It will let you select a voice and a pitch, and play the tone you've chosen for about two seconds. The sound shuts off, and the program asks you for another voice and pitch. This program is a musical experimenter's delight, so be sure to give it a try. When you wish to stop the program, enter 0 as a voice selection. You may suspect there's a problem when you select pitch 254 in voice 3 and hear nothing. Actually, it's not an error — this note is just too high for human ears. (you might test it out on your dog though!)

PROGRAM 5: Temperature Conversion
(Chapter 6)

```
10 INPUT "DEGREES FAHRENHEIT"; F
20 PRINT F "DEGREES F."
30 PRINT "IS" (F-32)*5/9 "DEGREES C."
40 PRINT
50 GOTO 10
```

This introduces you to the INPUT statement, which allows your program to stop what it is doing and request necessary information from the operator (the person who is RUNNING the program).

Line 10 causes the message DEGREES FAHRENHEIT? to appear on the screen. The words inside the quotes of an INPUT statement work just like the PRINT statement. However, the last word will always be followed by the question mark character, and the program will wait at this point for more information.

Line 20 prints the value of F, which is what was just typed in (inputted). Line 30 prints the result of the conversion calculation. In line 40, the word PRINT alone on a line causes a blank line to appear on the screen.

Finally, line 50 makes the program go back to the beginning and request more information to start again. The original question will be asked again, and you can have more temperatures converted. If you are finished, hold down the STOP key and hit the RESTORE key. There is no other way to tell this program you are through.

PROGRAM 6: Random Maze (Chapter 7)

```
10 PRINT "   "  
20 PRINT CHR$(205.5 + RND(1));  
30 GOTO 20
```

This is a neat little program that prints pseudo-mazes all over the screen. As you may expect, line 20 is the key here.

The CHR\$ function will give you a character based on a code number from 0 to 255. Every character that the VIC can put on the screen is encoded this way (see Appendix H). To find out the code for any character, just type PRINT ASC("X") where X is

include quotes

no quotes

the character you're checking. Then type PRINT CHR\$(X) where X is the number the VIC gave you. See how it works?

Now try typing PRINT CHR\$(205); CHR\$(206). This should print the two right side graphic characters on the M and N keys. These are the two characters that the program is using for the "mazes".

By adding the formula $205.5 + \text{RND}(1)$, the VIC will pick a random number between 205.5 and 206.5. There is a fifty-fifty chance of the number being above or below 206. When the CHR\$ function works, it will ignore any fractional values. Therefore, half the time the character with code 205 is PRINTed, and the other half character code 206 PRINTs.

If you'd like to experiment with this program, try changing the 205.5 by adding or subtracting a couple of tenths from it. This would give either character a greater chance of being PRINTed.

MORE ABOUT RANDOM NUMBERS

The random number function is one of the most useful and enjoyable aspects of BASIC, allowing you to program all sorts of games of chance.

The line $X = \text{RND}(1)$ will cause the VIC to select a random number between 0 and 1, not inclusive, to be placed into X. This results in a range of possible values for X:

$$0 < X < 1$$

When you work with random numbers, it is best to keep in mind that you will generate a range of numbers, to see how calculations effect the whole range. For example, if you wanted to get a set of possible values between 0 and 3, you could just multiply X by three. The new range is:

$$0 < X < 3$$

If you needed to pick a number from 10 to 20, how would you perform a calculation to change the range? First, you would add 10 to the number picked, to change the range to

$$10 < X < 11$$

By multiplying the random number by 10 before adding 10, the range becomes:

$$10 < X < 20$$

So the formula for a random number between 10 and 20 is:

$$X = \text{RND}(1) * 10 + 10$$

So far, we have learned how to change the range of possible results for the random number. However, the result of the function will contain messy decimal places, which are not desirable for things like rolling dice or picking a number from 1 to 10. The function used to clean this up is the INT function. This will chop off all decimal places from the number. The formula for a random number from 10 to 20, with the INT function added, becomes: $X = \text{INT}(\text{RND}(1) * 10 + 10)$

The range of possible results is now:

$$10 \leq X \leq 19$$

But wait! The upper limit of the range has dropped from 20 to 19 in this case. Why? Because before the range was always less-than 20. The INT function will strip off any decimal places from a number greater than 19 and less than 20 to result in a 19. On the other end of the range, any results between 10 and eleven are truncated down to an even 10. If we still needed to get a range of numbers from 10 to 20, the formula should become:

$$X = \text{INT}(\text{RND}(1) * 11 + 10)$$

The random number is multiplied to expand the range, and added to move the range.

The general formula for a set of random numbers in a certain range is:

$$X = \text{INT}(\text{RND}(1) * a) + b$$

Where a represents the number of possible results and b is the lowest number in the range.

APPENDICES

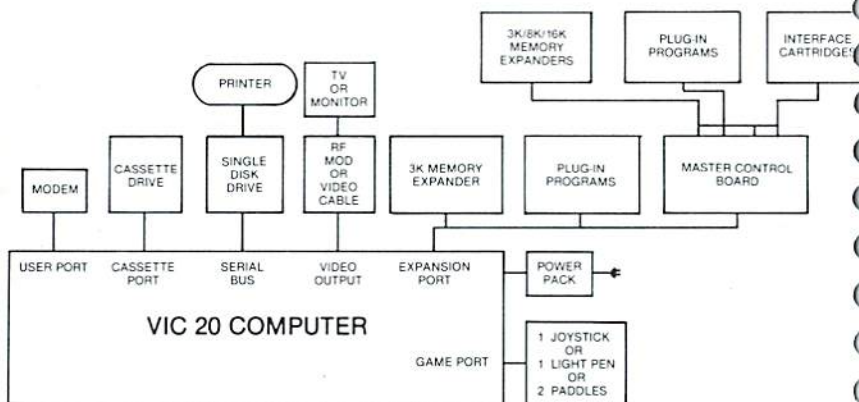
APPENDIX A: VIC ACCESSORIES — A QUICK INTRODUCTION*

This is a beginner's user manual, so we are not going to spend a lot of time and space telling you about the various peripherals that plug into the slots on the back of your VIC.

Each of the VIC peripherals will have its own instruction manual, which will tell you how to connect it to the VIC and how to use it to do interesting things.

Commodore has designed the VIC to grow with your needs and has an overall plan of how the various peripherals will fit together in and on the VIC. Here is the map of that plan:

THE COMMODORE VIC 20 PERSONAL COMPUTER SYSTEM*



SPECIAL PLUG-IN PROGRAMS

Super Expander Cartridge — 3K Added Memory (converts VIC to 8K)
High Resolution Graphics & Plotting Commands
Pre-assigned Function Keys

Programming Aid Cartridge — Programmer's "tool kit"
Machine Language Monitor
Pre-assigned function keys (prog commands)
User-assignable function keys

*Note: Peripherals described in this section scheduled for sale during late 1981.

TAPE CASSETTE RECORDER

The first peripheral you will probably get will be the Commodore tape recorder. The tape recorder can store several thousand characters (letters and numbers) on an ordinary cassette tape. You can store quite a few long programs on tape and load them back fairly quickly, without having to type them in every time. (See Appendix B - Working With Tape Cassettes)

The VIC Tape Cassette plugs into the Cassette Interface slot on the back of the VIC with its own special plug — it does not need any special interface. Commodore provides a variety of computer programs on tape for use with the cassette recorder.

MASTER CONTROL PANEL

You can plug a Master Control Panel in to the Expansion Port on the back of the VIC. This controller lets you use more than one cartridge at one time. It has six slots and accepts program cartridges, memory expanders and includes an IEEE-488 interface which lets you use Commodore PET/CBM peripherals and other IEEE devices.

VIC SINGLE FLOPPY DISK DRIVE

The VIC Single Floppy Disk Drive can store up to 170,000 characters (letters and numbers) and move a very long program in or out of the VIC's memory in a fraction of a second. No long waiting for a tape drive to find, read and load a program. This device connects to the VIC's serial port.

IEEE-488 INTERFACE CARTRIDGE

IEEE-488 is a universal scientific standard that lets you use Commodore PET/CBM peripherals such as disk drives and printers, as well as scientific instruments and tools.

SERIAL PRINTER

The VIC has a Serial Interface for peripherals which use serial connections to communicate with the computer. With Commodore's special dot matrix printer that uses this serial connection you can print your program listings and results on paper.

GAME PORT

The VIC Game Port allows you to hook up joysticks, light pens, and paddles, so that you can easily play exciting arcade type games without having to use the keyboard.

TELECOMMUNICATIONS

You may have heard about programs that let your personal computer talk to other computers and get information about the Stock Market, Business, News and other things — the VIC has this ability already built in.

The VIC's User Port includes an RS232 interface which lets you obtain or exchange information over the telephone using an inexpensive telecommunications *modem*.

SOFTWARE

In addition to all of this hardware, the VIC has a lot of interesting software.

Some software programs are stored in hardware cartridges which plug in to the back of the VIC and are ready to run as soon as the power is turned on. Some are on tape, some are on disk, and many are included in the VIC Learning Series Book and cartridge sets which let you teach yourself computing and other subjects at home.

This has been just a short overview of accessories available to you.

**KEEP IN TOUCH WITH YOUR
COMMODORE DEALER TO FIND OUT
ABOUT THE NEWEST DEVELOPMENTS!**

“APPENDIX B” USING THE CASSETTE RECORDER

The cassette recorder acts as the VIC's “memory”. Without this device, the VIC will forget any program you typed in as soon as the power goes off (or someone uses the NEW command). You can also use the recorder for programs you purchase. Commands used with programs on tape are SAVE, LOAD, and VERIFY.

The VIC can also “remember” the values of variables and other items of data, collected in a group called a file. The amount of data stored in a file can be very large compared to the amount of RAM in the VIC, because the VIC can operate on a small piece of the file at a time. Statements used with data files are OPEN, CLOSE, PRINT#, INPUT#, and GET#. The system variable ST (status), is used to check for tape markers.

PROGRAM STORAGE

Let's say you have just finished creating a new program on the VIC. You will want to use the program again at some other time, so now you must store it on the tape. Type the word SAVE. If you want the program to have a name, type a quote mark (") and the program name. The name can contain graphics and cursor controls, and can be up to sixteen characters long. Then just hit the RETURN key, whether you gave it a name or not.

If your recorder is plugged into the back of the VIC, and none of its buttons are pressed down, the following message will appear on the screen:

PRESS PLAY AND RECORD ON TAPE

All you have to do now is put a blank cassette tape (any decent audio tape will do) in the recorder. Hold down the RECORD key and hit PLAY on the recorder. The tape should start moving, and the following message will appear on the screen:
OK

SAVING name

What the VIC is doing now is recording, with a very fast series of high and low sounds, your program in RAM onto the tape. The program is not being disturbed at all, it is just being copied. The VIC actually records the program twice, just in case your blank tape isn't perfect. When the program is all on the tape, the VIC stops the recorder (all by itself!) and gives the READY message.

Now you may wish to check the tape to make sure that the SAVED program is correct. After all, you are risking much of your valuable time to an uncertain piece of thin magnetic tape. It is better to be safe and check the program.

The command to check the tape is called VERIFY. First, rewind the tape back to the beginning. Now type the word VERIFY. If your program had a name, you may type a quote mark and the name. If you want to VERIFY the first program on the tape, you can just leave off the program name. Next just hit the RETURN key.

Now, if none of the keys on the recorder are pressed, the VIC will tell you this:

PRESS PLAY ON TAPE

Be very sure that the RECORD button is NOT down when you do this. That would result in erasing any program or data that may have been on the tape.

When you've pressed PLAY, this message appears:

SEARCHING FOR name

From this point on, the VIC will search for the program you have specified on the tape. If it finds any programs or data files, it will report the fact with the message:

FOUND name

If the name of the program matches, or at least matches as many letters as you gave (which is why giving no name checks the first program), the next message is:

VERIFYING

When the program is completely read on the tape, the VIC gives you the verdict. You may see the message:

OK

READY.

This means that the program on the tape was identical with the program you have in RAM currently.

If the programs did not match, you'll see the message:

?VERIFY ERROR

READY.

One of the extra uses of the VERIFY command is to position the tape right after the last program, so you can add a new program to the tape. Just VERIFY using the name of the last program on the tape. You will end up with a VERIFY ERROR, but the tape will be right where you want it. Tricky, right?

Now let's say that you've come back to the VIC and you want to use that program you stored away. The command for this is LOAD. Type the word LOAD, and you can follow it with a program name (enclosed in quotes) if you like. The VIC says:

PRESS PLAY ON TAPE

You must press only the PLAY key on the recorder. The VIC will now say:

SEARCHING FOR name

Just as in VERIFY, the VIC lets you know of any programs or files encountered until the matching name. Then this message appears:

FOUND name

LOADING

When the program is loaded, the VIC says:

OK

READY.

Now the program is ready to be LISTed, RUN, or any other operation, just as if you had just typed it all in.

PROGRAM 1: WRITE-TO-TAPE

10 PRINT "   WRITE-TO-TAPE PROGRAM "

20 OPEN 1,1,1, "DATA FILE"

30 PRINT "NOW TYPE DATA TO BE"

40 PRINT "STORED OR TYPE   STOP  

50 PRINT

60 INPUT "DATA";A\$

70 PRINT #1,A\$

80 IF A\$ <> "STOP" THEN 50

90 PRINT

100 PRINT "CLOSING FILE"

110 CLOSE 1

PROGRAM 2: READ-TAPE USING INPUT#

```
10 PRINT" SHIFT CLR HOME READ-TAPE PROGRAM"  
20 OPEN 1,1,0,"DATA FILE"  
30 PRINT"FILE OPEN"  
40 PRINT  
50 INPUT#1,A$  
60 PRINT A$  
70 IF A$ = "STOP" THEN END  
80 GOTO 40
```

PROGRAM 3: READ-TAPE USING GET#

Lines 10 to 40 same as PROGRAM 2

```
50 GET#1, A$  
60 IF A$ = "" THEN END  
70 PRINT A$,ASC(A$)  
80 GOTO 50
```

APPENDIX C: VIC BASIC

This manual has given you an introduction to the BASIC language, just enough for you to get a feel for computer programming and some of the vocabulary involved. This appendix gives a complete list of the rules (SYNTAX) of the VIC BASIC 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 VIC by just typing in programs, and that the best way to learn computing is by doing.

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 VIC BASIC commands is provided in the VIC Programmer's Reference Guide, available where you bought your VIC.

1. VARIABLES & OPERATORS

a. VARIABLES

The VIC 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 -10^{38} to $+10^{38}$, 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 + 11.

Integer variables, are used when the number will always be between +32767 and -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 VIC can make. An example of a string variable is "VIC20."

Variable names may consist of a single letter, a letter followed by a number, or two letters.

An integer variable is specified by using the *percent (%) sign* after the variable name. String variables have the *dollar sign (\$) 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 which is which. 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 variable 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 meaning which row and the second number in the parentheses meaning which column.

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

There are three variable names which are reserved for use by the VIC, and may not be used for a normal purpose. These are the variables ST, TI, and TI\$. ST is a status variable which relates to input/output operations. The value of ST will change if there is a problem loading a program or data from the tape or disk. A more detailed explanation of ST is in the VIC BASIC Programmer's Reference Manual.

TI and TI\$ are variables which relate to the real-time clock built into the VIC. The variable TI is updated every 1/60th of a second. It starts at 0 when the VIC is turned on, and is reset only by changing the value of TI\$.

TI\$ is a string which is constantly updated by the system. The first two characters contains the number of hours, the 3rd and 4th characters are the number of minutes, and the 5th and 6th characters are the number of seconds. This variable can be given any value (so long as all characters are numbers), and will be updated from that point automatically.

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

This clock is erased when the VIC is turned off, and starts at zero when the VIC is turned back on.

b. OPERATORS

The arithmetic operators include the following signs:

- + addition
- subtraction
- * multiplication
- / division
- ↑ raising to 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, VIC BASIC allows you to isolate a calculation by putting parentheses around it. Operations enclosed in parentheses will take place before other operations. Be sure that your formulas have the same number of left parentheses (as right parentheses), or your program will get a SYNTAX ERROR message when run.

There are also operators for equalities and inequalities:

=	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.

EXAMPLE:

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.

2. COMMANDS

CONT (Continue)

This command is used to re-start the execution of a program which has been stopped by either using the STOP key, a STOP statement, or an END statement within the program. The program will re-start at the exact place from which 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), or if the program halted due to an error, or if you caused an error before trying to re-start the program. The message in this case is CAN'T CONTINUE ERROR.

LIST

The LIST command allows you to look at lines of a BASIC program that have been typed or LOADED into the VIC'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 CTRL key or STOPPED by hitting the key marked RUN STOP). If you follow the word LIST with a line number, the VIC will only show you that line number. If you type LIST with 2 numbers separated by a dash, the VIC will show all lines between the first and 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 until 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 10— Shows only from line 10 until the end.
- LIST 10 Shows only line 10.
- LIST—10 Shows lines from the beginning until line 10.
- LIST 10—20 Shows lines from 10 to 20, inclusive.

LOAD abbrev. L sh O

This is the command to use when you have a program stored on cassette tape or on disk, and you want to use it. If you type just the LOAD and hit the RETURN key, the VIC will find the first program on the cassette tape and bring it into memory, to be RUN, LISTed, or whatever. 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 any quotes) and a number or numeric variable, which acts as a device number to determine where the program is coming from. If there is no number given, the VIC assumes device #1, which is the cassette tape recorder.

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

EXAMPLES:

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

The LOAD command can be used within a BASIC program to find and RUN the next program on a tape.

NEW

This command erases the entire program in memory, and also clears out any variables that may have been used. Unless the program was previously 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 VIC gets to this line, the program is erased and everything stops. It is useful if you want to leave everything neat when the program is done.

RUN

Once a program has been typed into memory or LOADED, the RUN command is used to make it start working. If there is no number following the command RUN, the computer will start with the lowest numbered program line. If there is a number, that becomes the line number where the program starts from.

EXAMPLES:

- RUN Starts program working from lowest line number.
- RUN 100 Starts program at line 100.
- RUN X UNDEFINED STATEMENT ERROR (you must always type RUN by itself or with a line number - *not* with a letter).

SAVE

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 machine will attempt to store the program on the cassette tape. It has

no way of checking that there is already a program on 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 VIC will give the program that name, so it may be more easily located and retrieved in the future. The name may be followed by a comma (after the quotes) and a number or numeric variable. This number tells the VIC which device on which to store the program. Device number 1 is the tape drive, and #8 is the disk. After the number there can be a comma and a second number, which is either 0 or 1. If the second number is 1, the VIC will put an END-OF-TAPE marker after your program. If you are trying to LOAD a program and the VIC finds one of these markers, you will get a FILE NOT FOUND ERROR.

EXAMPLE:

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

VERIFY

This command causes the VIC 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 the VIC will write *after* the last program on the tape. All you do is tell the VIC 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 VIC 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).

EXAMPLE:

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

3. STATEMENTS

CLOSE

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 Only file #2 is closed.

CLR

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

CMD

CMD sends the output which normally would go to the screen (i.e. PRINT statements, 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.

EXAMPLE:

OPEN 1,4 OPENS 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 LIST that you typed!

To start sending back to the screen normally, just CLOSE the file.

DATA

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 quote marks, 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.

EXAMPLE OF A DATA STATEMENT:

DATA 100,200,FRED,"HELLO, MOM",,3.14,abc123

Since the program never needs to actually execute a DATA statement in order to read the information, it is a good idea to put your DATA statements as close to the last line of the program as possible. This will help your programs run faster.

DEF FN (Define Function)

This command allows you to define a complex calculation as a function with a short name. In the case of a long formula that is used several times during a program, this can save lots of space.

The name you give the function will be the letters FN and any legal variable name (1 or 2 characters long). 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)
```

```
20 PRINT FNA(7)
```

Asterisk is used as multiplication sign

The No. 7 is inserted where X is in the formula

DIM (Dimension an array)

Before you get to use arrays of variables, unless there are 11 or fewer elements, 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.

EXAMPLE:

```
10 DIM A$(40),B7(15),CC%(4,4,4)
```

41 Elements

16 Elements

125 Elements

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 a good idea to keep DIMs near the beginning of the program.

END

When the program hits a line with the END statement, the program stops RUNNING as if it ran out of lines. You may use the CONT command to re-start the program.

FOR...TO...STEP

This statement works with the NEXT statement to set up a section of the program that repeats for a set number of times. You may just want the VIC to count up to a large number so the program will pause for a few seconds, or you may need something counted. These are among the most commonly used statements in BASIC.

The format of the statement is as follows:

FOR (loop variable name) = (start of count) TO (end of count). The loop variable is a variable which will be added or subtracted to 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 VIC. When the program reaches a line with the command NEXT, it adds one 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.

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. Do you see why it works? If not, try re-reading the paragraph before the example again, and tracing through the program one step at a time on paper.

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 later loop to start is the earlier 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.

Not correct:

```
10 FOR L = 1 TO 100
20 FOR A = 5 TO 11 STEP 2
30 NEXT L
40 NEXT A
```

GET

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 received, and the program continues. 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 were 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.

EXAMPLE:

```
10 GET A$: IF A$ = "" THEN 10
```

This line waits for a key to be struck. Typing any key will continue the program.

GET#

Used with a previously OPENed device or file to input one character at a time.

EXAMPLE:

```
GET#1,A$
```

GOSUB

This statement is like the GOTO statement, except that the VIC 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. This is useful if there is a routine in your program that occurs several times in different parts of the program. Instead of typing the same over and over, you type it once and GOSUB to it from the different parts of the program. 20 GOSUB 800 means go to the subroutine beginning at line 800 and execute it.

GOTO or GO TO

When a statement with the GOTO command is reached, the next line to be executed will be the one with the line number following the word GOTO.

IF...THEN

The IF...THEN statement lets your VIC analyze a situation and take two possible courses of action depending on the outcome. If the expression being evaluated is found to be true, the statement following the word THEN is executed. This may be a line number, which will cause the VIC to GOTO that line of the program. It may also be any other BASIC statement or statements. If the expression is false, then the next line (*not* the next statement on the same line) is executed instead.

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 the *relational operators* (=, <, >, <=, >=, <=, >=, AND, OR, NOT). If the result is found to be true, it has a value of -1, and a value of 0 if false. See the section on relational operators for an explanation of how this works.

INPUT

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.

EXAMPLE:

```
10 INPUT"PLEASE TYPE A #";A
20 INPUT"AND YOUR NAME";A$
30 INPUT B$
40 PRINT"BET YOU DIDN'T KNOW  WHAT I WANTED!"
```

INPUT#

This works like INPUT, but takes the data from a previously OPENed file or device.

LET

The word LET itself is hardly ever used in programs, since it is optional, but the statement is the heart of all BASIC programs. The variable name which is to get 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"
```

NEXT

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

This command can make the GOTO and GOSUB commands into special versions of the IF statement. The word ON is followed by a formula, 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, negative, or larger than the list of line numbers, the next line executed will be the statement following the ON statements.

EXAMPLE:

```
10 INPUT X
20 ON X GOTO 10,50,50,50
30 PRINT "NOPE!"
40 GOTO 10
50 PRINT "YUP!"
60 ON X GOTO 10,30,30
```

OPEN

The OPEN statement allows the VIC to access devices such as the cassette recorder and disk for data, a printer, or even the screen of the VIC. The word OPEN is followed by a number, which is the number to which all other BASIC statements will refer. This number is from 1 to 255. There is usually a second number after the first, separated by a comma. This is the device number, 0 for the VIC screen, 1 for the cassette recorder, 4 for the printer, 8 for the disk. It is a good idea to use the same reference number as the device number, which makes it easy to remember which is which. Following the second number may be a third number, separated again by a comma, which is the secondary address. In the case of the cassette, this may 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 buffer, or channel, number. In the printer, the secondary addresses become different types of commands. See the VIC Programmers' Reference Manual for more on these. There may also be a string following the third number, which would be a command to the disk drive or the name of the file on tape.

EXAMPLE:

- 10 OPEN 1,0 OPENS the SCREEN as a device.
- 20 OPEN 2,1,0,"D" OPENS the cassette for reading, file to be searched for is named D.
- 30 OPEN 3,4 To use the printer.
- 40 OPEN 4,8,15 OPENS the data channel on the disk.

See also: CLOSE, CMD, GET#, INPUT#, and PRINT# statements, system variable ST, and Appendix B.

POKE

The POKE command is always followed by two numbers, or formulas. The first number is a location inside the VIC's memory. There could be locations numbered from 0 to over 65000. Some of these, like the ones described in the chapters on sound and colors, can be used easily in your programs. Some, however, are used by the VIC itself, to keep track of your programs and so on. Experimenting with the POKE statement will probably result in some interesting effects. If something happens and you can't stop it, just turn the VIC off and on again, or hold down the RUN/STOP key and hit RESTORE.

The second number is a value from 0 to 255, which will be placed in the memory location, replacing any value that was there previously.

EXAMPLE:

10 POKE 36879,8

20 POKE 9*16↑ 3 + 15,27

PRINT

The PRINT statement is the first one most people learn to use, but there are lots of 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. Variable names outside of quotes will have the value they contain printed. Functions will have their values printed also. Punctuation marks are used to help format the data neatly on the screen. The comma is used to divide the screen into 2 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:

10 PRINT "HELLO"

20 PRINT "HELLO, "A\$

30 PRINT A + B;

50 PRINT J;

60 PRINT A,B,C,D

See also: POS(), SPC(), TAB() functions.

PRINT#

There are a few differences between this statement and the PRINT. First of all, the word 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$,B$
```

READ

This statement is used to get information from DATA statements into variables, where they may be used. Care must be taken to avoid reading strings where the READ statement wants a number, which will give you a TYPE MISMATCH ERROR.

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. The word REM may be followed by any text, although use of graphic characters will give strange results (see the VIC PROGRAMMER'S REFERENCE GUIDE for more info.)

RESTORE

When executed in a program, the pointer to which item in a DATA statement will be read next is reset to the first item in the list. This gives you the ability to re-READ the information. The word RESTORE stands by itself on the line.

RETURN

This statement is always used in conjunction with the GOSUB statement. When the program hits a RETURN statement, it will go to the statement immediately following the GOSUB command. If no GOSUB was previously issued, there is a RETURN WITHOUT GOSUB ERROR. There is nothing following the word RETURN.

STOP

This statement will halt the program. A message, BREAK ERROR IN LINE xxxx, where xxxx is the line number containing the STOP. The program can be re-started by using the CONT command. The STOP statement is used for debugging a program.

SYS

The word SYS is followed by a decimal number or numeric variable in the range 0-65535. 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 allow parameter passing.

WAIT

The WAIT statement is used to halt the program until the contents of a location in memory changes 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

ABS(X) (absolute value)

The absolute value returns the value of the number, without its sign (- or +). The answer is always positive.

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.

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.

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 = INT(X*100 + .5)/100 Rounds to the nearest penny.

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.

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(-1), 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, the result will be the same random number as the last one. A negative value for X will re-seed the generator. The use of the same negative number for X will result in the same sequence of "random" numbers.

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 Simulate 2 dice.

100 X = INT(RND(1)*1000) + 1 Number from 1-1000.

100 X + INT(RND(1)*150) + 100 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 a 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 1 and 2. The parameter X is passed to the machine language program, which will return another number back to the BASIC program. See the VIC PROGRAMMER'S REFERENCE MANUAL for more details on this, and on machine language programming.

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.

LEFT\$(X\$,X)

This will return a string containing the leftmost X characters of X\$.

LEN(X\$)

Returned will be 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\$.

RIGHT\$(X\$,X)

This will return the rightmost X characters in X\$.

STR\$(X)

This will contain a string which is identical to the PRINTed version of X\$.

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 VIC 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(" 12A13B") X = 12

10 X = VAL("RIUO17*") X = 0

10 X = VAL("- 1.23.23.23") X = - 1.23

c. OTHER FUNCTIONS

FRE(X)

This function returns the number of unused bytes available in memory, regardless of the value of X.

POS(X)

This function returns the number of the column (0-21) at which the next PRINT statement will begin on the screen. X may have any value, and is not used.

SPC(X)























































































This is used in the PRINT statement to skip X spaces forward.



















TAB(X)

This is used in the PRINT statement. The next item to be printed will be in column number X.

APPENDIX D: ABBREVIATIONS FOR BASIC KEYWORDS

As a time saver when typing in programs and commands, VIC BASIC allows the user to abbreviate most keywords. The abbreviation for the word PRINT is a question mark. The abbreviations for the other words are made by typing the first one or two letters of the key word, followed by the SHIFTeD next letter of the word. If the abbreviations are used in a program line, the keyword will LIST in the longer form. Note that some of the keywords when abbreviated include the first parenthesis, and others do not.

Command	Abbreviation	Looks like this on screen	Command	Abbreviation	Looks like this on screen
AND	A  N	A 	PRINT#	P  R	P 
NOT	N  O	N 	READ	R  E	R 
CLOSE	CL  O	CL 	RESTORE	RE  S	RE 
CLR	C  L	C 	RETURN	RE  T	RE 
CMD	C  M	C 	RUN	R  U	R 
CONT	C  O	C 	SAVE	S  A	S 
DATA	D  A	D 	STEP	ST  E	ST 
DEF	D  E	D 	STOP	S  T	S 
DIM	D  I	D 	SYS	S  Y	S 
END	E  N	E 	THEN	T  H	T 
FOR	F  O	F 	VERIFY	V  E	V 
GET	G  E	G 	WAIT	W  A	W 
GOSUB	GO  S	GO 	ABS	A  B	A 
GOTO	G  O	G 	ASC	A  S	A 
INPUT#	I  N	I 	ATN	A  T	A 
LET	L  E	L 	CHRS	C  H	C 
LIST	L  I	L 	EXP	E  X	E 
LOAD	L  O	L 	FRE	F  R	F 
NEXT	N  E	N 	LEFT\$	LE  F	LE 
OPEN	O  P	O 	MID\$	M  I	M 
POKE	P  O	P 	PEEK	P  E	P 
PRINT	?	?	RIGHT\$	R  I	R 



Command	Abbreviation	Looks like this on screen	Command	Abbreviation	Looks like this on screen
RND	R  N	R 	STR\$	ST  R	ST 
SGN	S  G	S 	TAB (T  A	T 
SIN	S  I	S 	USR	U  S	U 
SPC (S  P	S 	VAL	V  A	V 
SQR	S  Q	S 			

APPENDIX E SCREEN & BORDER COLOR COMBINATIONS

You can change the screen and border colors of the VIC anytime, in or out of a program, by typing

POKE 36879, X

where X is one of the numbers shown in the chart below. POKE 36879, 27 returns the screen to the normal color combination, which is a CYAN border and white screen.

Try typing POKE 36879, 8. Then type   and you have white letters on a totally black screen! Try some other combinations. This POKE command is a quick and easy way to change screen colors in a program.

SCREEN	BORDER							
	BLK	WHT	RED	CYAN	PUR	GRN	BLU	YEL
BLACK	8	9	10	11	12	13	14	15
WHITE	24	25	26	27	28	29	30	31
RED	40	41	42	43	44	45	46	47
CYAN	56	57	58	59	60	61	62	63
PURPLE	72	73	74	75	76	77	78	79
GREEN	88	89	90	91	92	93	94	95
BLUE	104	105	106	107	108	109	110	111
YELLOW	120	121	122	123	124	125	126	127
ORANGE	136	137	138	139	140	141	142	143
LT. ORANGE	152	153	154	155	156	157	158	159
PINK	168	169	170	171	172	173	174	175
LT. CYAN	184	185	186	187	188	189	190	191
LT. PURPLE	200	201	202	203	204	205	206	207
LT. GREEN	216	217	218	219	220	221	222	223
LT. BLUE	232	233	234	235	236	237	238	239
LT. YELLOW	248	249	250	251	252	253	254	255

APPENDIX F TABLE OF MUSICAL NOTES

APPROX. NOTE	VALUE	APPROX. NOTE	VALUE
C	135	G	215
C#	143	G#	217
D	147	A	219
D#	151	A#	221
E	159	B	223
F	163	C	225
F#	167	C#	227
G	175	D	228
G#	179	D#	229
A	183	E	231
A#	187	F	232
B	191	F#	233
C	195	G	235
C#	199	G#	236
D	201	A	237
D#	203	A#	238
E	207	B	239
F	209	C	240
F#	212	C#	241

SPEAKER COMMANDS:	WHERE X CAN BE:	FUNCTION:
POKE 36878, X	0 to 15	sets volume
POKE 36874, X	128 to 255	plays tone
POKE 36875, X	128 to 255	plays tone
POKE 36876, X	128 to 255	plays tone
POKE 36877, X	128 to 255	plays "noise"

APPENDIX G: 20 SOUND EFFECTS FOR THE VIC-20

Here are some sample routines to use as a guide for creating sounds to enhance your programs. You may type them into your VIC-20 either by themselves or inside other programs. Of course, these are not nearly all of the possible sounds that your VIC-20 can play, so feel free to use your creativity.

The sound effects listed here will make a program pause for however long they take to be completed. It is possible to put these effects into a program in a way that does not stop whatever animation may be running, and this topic is discussed in detail in the VIC-20 Programmer's Reference Manual.

Remember to use line numbers when you type these routines into the computer. The numbers are not shown here in order to avoid confusion when you enter them into your programs.

#1: SCALES

```
POKE 36878,15
FOR L = 250 TO 200 STEP - 2
POKE 36876,L
FOR M = 1 TO 100
NEXT M
NEXT L
FOR L = 205 TO 250 STEP 2
POKE 36876,L
FOR M = 1 TO 100
NEXT M
NEXT L
POKE 36876,0
POKE 36878,0
```

#2: COMPUTER MANIA

```
POKE 36878,15
FOR L = 1 TO 100
POKE 36876,INT(RND(1)* 128) + 128
FOR M = 1 TO 10
NEXT M
NEXT L
POKE 36876,0
POKE 36878,0
```

#3: EXPLOSION

```
POKE 36877,220
FOR L = 15 TO 0 STEP - 1
POKE 36878,L
FOR M = 1 TO 300
NEXT M
NEXT L
POKE 36877,0
POKE 36878,0
```

#4: BOMBS AWAY

```
POKE 36878,10
FOR L = 230 TO 128 STEP - 1
POKE 36876,L
FOR M = 1 TO 20
NEXT M
NEXT L
POKE 36876,0
POKE 36877,200
FOR L = 15 TO 0 STEP - .05
POKE 36878,L
NEXT L
POKE 36877,0
```

#5: RED ALERT

```
POKE 36878,15
FOR L = 1 TO 10
FOR M = 180 TO 235 STEP 2
POKE 36876,M
FOR N = 1 TO 10
NEXT N
NEXT M
POKE 36876,0
FOR M = 1 TO 100
NEXT M
NEXT L
POKE 36878,0
```

#6: LASER BEAM

```
POKE 36878,15
FOR L = 1 TO 30
FOR M = 250 TO 240 STEP - 1
POKE 36876,M
NEXT M
FOR M = 240 TO 250
POKE 36876,M
NEXT M
POKE 36876,0
NEXT L
POKE 36878,0
```

#7: HIGH-LOW SIREN

```
POKE 36878,15
FOR L = 1 TO 10
POKE 36875,200
FOR M = 1 TO 500
NEXT M
POKE 36875,0
POKE 36876,200
FOR M = 1 TO 500
NEXT M
POKE 36876,0
NEXT L
POKE 36878,0
```

#8: BUSY SIGNAL

```
POKE 36878,15
FOR L = 1 TO 15
POKE 36876,160
FOR M = 1 TO 400
NEXT M
POKE 36876,0
FOR M = 1 TO 400
NEXT M
NEXT L
POKE 36878,0
```

#9: PHONE RINGING

```
POKE 36878,15
FOR L = 1 TO 5
FOR M = 1 TO 50
POKE 36876,220
FOR N = 1 TO 5
NEXT N
POKE 36876,0
NEXT M
FOR M = 1 TO 3000
NEXT M
NEXT L
POKE 36878,0
```

#10: BIRDS CHIRPING

```
POKE 36878,15
FOR L = 1 TO 20
FOR M = 254 TO 240 + INT (RND(1)* 10) STEP - 1
POKE 36876,M
NEXT M
POKE 36876,0
FOR M = 0 TO INT(RND(1)* 100) + 120
NEXT M
NEXT L
```

#11: WIND

```
POKE 36878,15
POKE 36874,170
POKE 36877,240
FOR L = 1 TO 2000
NEXT L
POKE 36874,0
POKE 36877,0
POKE 36878,0
```

#12: OCEAN WAVES

```
POKE 36877,180
FOR L = 1 TO 10
D = INT(RND(1)*5)*50 + 50
FOR M = 3 TO 15
POKE 36878,M
FOR N = 1 TO D
NEXT N
NEXT M
FOR M = 15 TO 3 STEP - 1
POKE 36878,M
FOR N = 1 TO D
NEXT N
NEXT M
NEXT L
POKE 36878,0
POKE 36877,0
```

#13: VANISHING UFO

```
POKE 36878,15
FOR L = 130 TO 254
POKE 36876,L
FOR M = 1 TO 40
NEXT M
NEXT L
POKE 36878,0
POKE 36876,0
```

#14: UFO LANDING

```
POKE 36878,15
FOR L = 1 TO 20
FOR M = 220-L TO 160-L STEP - 4
POKE 36876,M
NEXT M
FOR M = 160-L TO 220-L STEP 4
POKE 36876,M
NEXT M
NEXT L
POKE 36878,0
POKE 36876,0
```

#15: UFO SHOOTING

```
POKE 36878,15
FOR L = 1 TO 15
FOR M = 200 TO 220 + L*2
POKE 36876,M
NEXT M
NEXT L
POKE 36878,0
POKE 36876,0
```

#16: WOLF WHISTLE

```
POKE 36878,15
FOR L = 148 TO 220 STEP .7
POKE 36876,L
NEXT L
FOR L = 128 TO 200
POKE 36876,L
NEXT L
FOR L = 200 TO 128 STEP - 1
POKE 36876,L
NEXT L
POKE 36878,0
POKE 36876,0
```

#17: RUNNING FEET

```
POKE 36878,15
FOR L = 1 TO 10
POKE 36874,200
FOR M = 1 TO 10
NEXT M
POKE 36874,0
FOR M = 1 TO 100
NEXT M
NEXT L
POKE 36878,0
```

#18: TICK—TOCK

```
POKE 36878,15
FOR L = 1 TO 10
POKE 36875,200
FOR M = 1 TO 10
NEXT M
POKE 36875,0
FOR M = 1 TO 300
NEXT M
POKE 36874,200
FOR M = 1 TO 10
NEXT M
POKE 36874,0
FOR M = 1 TO 300
NEXT M
NEXT L
POKE 36878,0
```

#19: DOOR OPENING

```
POKE 36878,15
B = 0
FOR L = 128 TO 255 STEP 11
POKE 36874,L
FOR M = 1 TO 10
NEXT M
B = B + 1
IF B = 3 THEN B = 0: POKE 36874,0
NEXT L
POKE 36874,0
POKE 36878,0
```

#20: BLIPS

```
POKE 36878,15
POKE 36876,220
FOR L = 1 TO 5
NEXT L
POKE 36876,0
FOR L = 1 TO 500
NEXT L
POKE 36876,200
FOR L = 1 TO 5
NEXT L
POKE 36876,0
FOR L = 1 TO 500
NEXT L
POKE 36878,0
```

APPENDIX H: SCREEN DISPLAY CODES

The following chart lists all of the characters built-in to the VIC20 character sets. They show which numbers should be POKEd into screen memory (locations 7680 to 8185) to get a desired character. Also, it shows what character corresponds to a number PEEKed from the screen.

The 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 COMMODORE keys simultaneously. This actually changes the 2 bit in memory location 36869, which means that the statement `POKE 36869, 240` will set the character set to upper case, and `POKE 36869, 242` switches to lower case.

If you want to do some serious animation, you will find that it is easier to control objects on the screen by POKeIng them into screen memory (and erasing them by poking a 32, which is the code for a blank space, into the same memory location), than by PRINTing to the screen by using cursor control characters.

Any number shown on the chart may also be displayed in REVERSE. Reverse characters are not shown, but the reverse of any character may be obtained by adding 128 to the numbers shown.

NOTE: SEE SCREEN MEMORY MAP APPENDIX.

If you want to display a heart at screen location 7800, find the number of the character you want to display there (in this case a heart) in this chart...the number for the heart is 83...then type in a POKe statement with the number of the screen location (7800) and the number of the symbol (83) like this:

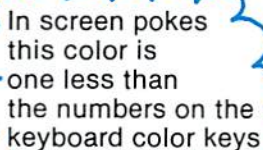
```
POKE 7800, 83
```

A white heart should appear in the middle area of the screen. Note that it will be invisible if the screen is white. Try changing the position by changing the larger number, or type in different symbols using the numbers from the chart.

If you want to change the COLOR of the symbol being displayed, consult the accompanying chart which lists the COLOR NUMBERS for EACH MEMORY LOCATION. In other words, to get a different colored symbol at a particular location, this requires another POKe command.

For example, to get a red heart, type the following:

POKE 38520, 2



In screen pokes
this color is
one less than
the numbers on the
keyboard color keys

This changes the color of the symbol at location 7800 to red. If you had a different symbol there, that symbol would now be red. You can display any character in any of the available colors by combining these two charts. These POKE commands can be added in your programs and are very effective especially in animation — and also provide a means to PEEK at certain locations if you are doing sophisticated programming such as bouncing a ball, which requires this information.

SCREEN CODES

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	U	u	21	*		42
A	a	1	V	v	22	+		43
B	b	2	W	w	23	,		44
C	c	3	X	x	24	—		45
D	d	4	Y	y	25	.		46
E	e	5	Z	z	26	/		47
F	f	6	[27	∅		48
G	g	7	£		28	1		49
H	h	8]		29	2		50
I	i	9	↑		30	3		51
J	j	10	←		31	4		52
K	k	11	SPACE		32	5		53
L	l	12	!		33	6		54
M	m	13	“		34	7		55
N	n	14	#		35	8		56
O	o	15	\$		36	9		57
P	p	16	%		37	:		58
Q	q	17	&		38	;		59
R	r	18	'		39	<		60
S	s	19	(40	=		61
T	t	20)		41	>		62

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
?		63		T	84			106
		64		U	85			107
	A	65		V	86			108
	B	66		W	87			109
	C	67		X	88			110
	D	68		Y	89			111
	E	69		Z	90			112
	F	70			91			113
	G	71			92			114
	H	72			93			115
	I	73			94			116
	J	74			95			117
	K	75	SPACE		96			118
	L	76			97			119
	M	77			98			120
	N	78			99			121
	O	79			100			122
	P	80			101			123
	Q	81			102			124
	R	82			103			125
	S	83			104			126
					105			127

APPENDIX I: SCREEN MEMORY MAPS

Use this appendix to find the memory location of any position on the screen. Just find the position in the grid and add the numbers on the row and column together. For example, if you want to poke the graphic "ball" character onto the center of the screen, add the numbers at the edge of row 11 and column 11 ($7900 + 10$) for a total of 7910. If you poke the code for a ball (81, see Appendix H) into location 7910 by typing `POKE 7910,81`, a white ball appears on the screen. To change the color of the ball (or other character), find the corresponding position on the color codes memory map, add the row and column numbers together ($38620 + 10$, or 38630) for the color code and type a second poke statement. For example, if you poke a color code into this location, `POKE 38630,3` the ball will change color to cyan. Note that when `POKEing`, the character color numbers are one less than the numbers on the color keys—as shown below.

Abbreviated List of Color Codes:

Code	Color
0	Black
1	White
2	Red
3	Cyan
4	Purple
5	Green
6	Blue
7	Yellow

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7680																						
7702																						
7724																						
7746																						
7768																						
7790																						
7812																						
7834																						
7856																						
7878																						
7900																						
7922																						
7944																						
7966																						
7988																						
8010																						
8032																						
8054																						
8076																						
8098																						
8120																						
8142																						
8164																						

PAGE 1: SCREEN CHARACTER CODES

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
38400																						
38422																						
38444																						
38466																						
38488																						
38510																						
38532																						
38554																						
38576																						
38598																						
38620																						
38642																						
38664																						
38686																						
38708																						
38730																						
38752																						
38774																						
38796																						
38818																						
38840																						
38862																						
38884																						






































PAGE 2: COLOR CODES MEMORY MAP

APPENDIX J: 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		16	SPACE	32	Ø	48
	1	CRSR ↓	17	!	33	1	49
	2	RVS ON	18	"	34	2	50
	3	CLR HOME	19	#	35	3	51
	4	INST DEL.	20	\$	36	4	52
WHT	5		21	%	37	5	53
	6		22	&	38	6	54
	7		23	.	39	7	55
	8		24	(40	8	56
	9		25)	41	9	57
	10		26	*	42	:	58
	11		27	+	43	;	59
	12	RED	28	,	44	<	60
RETURN	13	CRSR →	29	-	45	=	61
SWITCH TO LOWER CASE	14	GRN	30	.	46	>	62
	15	BLU	31	/	47	?	63

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
@	64	U	85		106		127
A	65	V	86		107		128
B	66	W	87		108		129
C	67	X	88		109		130
D	68	Y	89		110		131
E	69	Z	90		111		132
F	70	[91		112	f1	133
G	71	£	92		113	f3	134
H	72]	93		114	f5	135
I	73	↑	94		115	f7	136
J	74	←	95		111	f2	137
K	75		96		117	f4	138
L	76		97		118	f6	139
M	77		98		119	f8	140
N	78		99		120	SHIFT	141
O	79		100		121	RETURN	142
P	80		101		122	SWITCH TO UPPER CASE	143
Q	81		102		123	BLK	144
R	82		103		124	CRSR	145
S	83		104		125	RVS OFF	146
T	84		105		126	CLR HOME	147

PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS	PRINTS	CHRS
	148		159		170		181
	149		160		171		182
	150		161		172		183
	151		162		173		184
	152		163		174		185
	153		164		175		186
	154		165		176		187
	155		166		177		188
	156		167		178		189
	157		168		179		190
	158		169		180		191

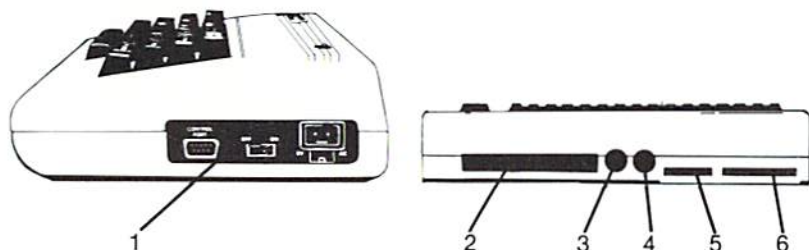
APPENDIX K: DERIVING MATHEMATICAL FUNCTIONS

Functions that are not intrinsic to VIC BASIC may be calculated as follows:

FUNCTION	VIC 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	$\text{TANH}(X) = \text{EXP}(-X)/(\text{EXP}(X) + \text{EXP}(-X))^2 + 1$
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	$\text{COTH}(X) = \text{EXP}(-X)/(\text{EXP}(X) - \text{EXP}(-X))^2 + 1$
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	$\text{ARCCOTH}(X) = \text{LOG}((X + 1)/(X - 1))/2$

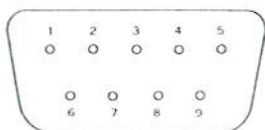
APPENDIX L: PINOUPS FOR INPUT/OUTPUT DEVICES

Here is a picture of the I/O ports on the VIC:



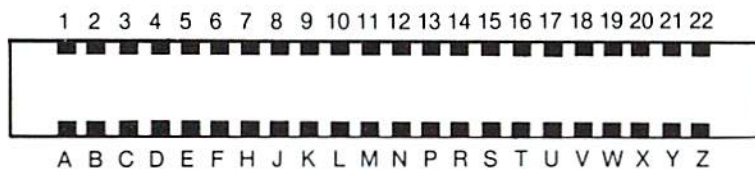
- | | |
|---------------------|----------------------|
| 1) Game I/O | 4) Serial I/O (disk) |
| 2) Memory Expansion | 5) Cassette |
| 3) Audio and Video | 6) User Port (modem) |

1) GAME I/O



PIN #	TYPE	NOTE
1	JOY0	MAX. 100mA
2	JOY1	
3	JOY2	
4	JOY3	
5	POT Y	
6	LIGHT PEN	
7	+5V	
8	GND	
9	POT X	

2) MEMORY EXPANSION



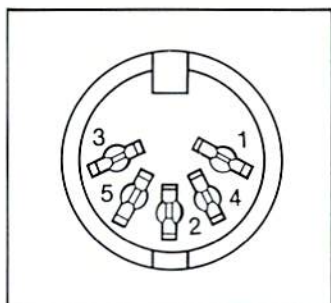
PIN #	TYPE
1	GND
2	CD \emptyset
3	CD1
4	CD2
5	CD3
6	CD4
7	CD5
8	CD6
9	CD7
1 \emptyset	$\overline{\text{BLK1}}$
11	$\overline{\text{BLK2}}$

PIN #	TYPE
12	$\overline{\text{BLK3}}$
13	$\overline{\text{BLK5}}$
14	$\overline{\text{RAM1}}$
15	$\overline{\text{RAM2}}$
16	$\overline{\text{RAM3}}$
17	VR/W
18	CR/W
19	$\overline{\text{IRQ}}$
2 \emptyset	NC
21	+5V
22	GND

PIN #	TYPE
A	GND
B	CA \emptyset
C	CA1
D	CA2
E	CA3
F	CA4
H	CA5
J	CA6
K	CA7
L	CA8
M	CA9

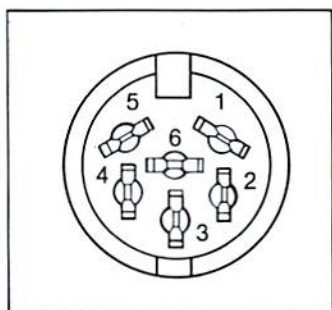
PIN #	TYPE
N	CA1 \emptyset
P	CA11
R	CA12
S	CA13
T	I/ \emptyset 2
U	I/ \emptyset 3
V	S \emptyset 2
W	$\overline{\text{NMI}}$
X	$\overline{\text{RESET}}$
Y	NC
Z	GND

3) AUDIO/VIDEO



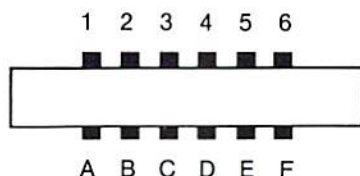
PIN #	TYPE	NOTE
1	+ 5V REG	10mA MAX
2	GND	
3	AUDIO	
4	VIDEO LOW	
5	VIDEO HIGH	

4) SERIAL I/O



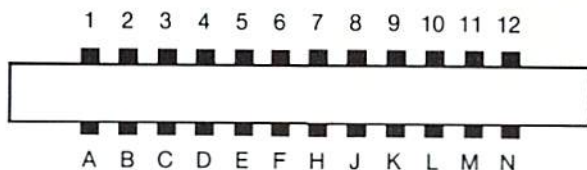
PIN #	TYPE
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET

5) CASSETTE



PIN #	TYPE
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SWITCH

6) USER I/O



PIN #	TYPE	NOTE	PIN #	TYPE	NOTE
1	GND		A	GND	
2	+5V	100mA MAX.	B	CB1	
3	RESET		C	PB0	
4	JOY0		D	PB1	
5	JOY1		E	PB2	
6	JOY2		F	PB3	
7	LIGHT PEN		H	PB4	
8	CASSETTE SWITCH		J	PB5	
9	SERIAL ATN IN	100mA MAX.	K	PB6	
10	+9V		L	PB7	
11	GND		M	CB2	
12	GND		N	GND	


```

174 POKEE,32:POKEE-1,32:POKEE-2,32:K=K+1
176 IFE=ITHEM=0:GOTO180
178 E=E+J: POKEE,62:POKEE-1,42:POKEE-2,60
179 IFJ=1THEN182
180 IFINT<<(8186-E)/22>>=22-K-ANDF=0THEN F=1:G=E+21:M=21:GOTO183
181 GOTO183
182 IFINT<<(8098-E)/22>>=A-KANDF=0THENF=1:G=E+23:M=23
183 IFF=0THEN125
184 POKEG,32:G=G+M
186 IFFEEK<G><>32THEN700
187 IFG>7680+22*21THENF=0:GOTO500
189 POKEG,81:GOTO125
220 IFA<0THENA=0
221 IFA>15THENA=15
222 PRINTTAB(A) " |"
225 PRINTTAB(A) " 3 4"
230 PRINTTAB(A) " 0000"
235 PRINT"000";GOTO135
300 PRINTTAB(A) "
305 PRINTTAB(A) "
310 PRINTTAB(A) "
315 PRINT"00";RETURN
400 PRINTPEEK<197>:GOTO400
500 POKEG,66:POKEG+1,78:POKEG-1,77:POKEG-20,46:POKEG-21,46:POKEG-22,46
510 POKEG-23,46:POKEG-24,46
520 FORAA=1TO100:NEXT
530 POKEG,32:POKEG+1,32:POKEG-1,32:POKEG-20,32:POKEG-21,32:POKEG-22,32
535 POKEG-23,32:POKEG-24,32
590 GOTO125
600 POKEC,160:POKEC+1,160:POKEC-1,160:POKEC+22,160:POKEC-22,160
601 L=0

```

```

610 POKEVN,128+100
611 FORGG=15TO0STEP-1:POKEVA,GG:FORGH=1TO70:NEXT:NEXT
615 B=0:D=0
616 POKEC,32 :POKEC+1, 32:POKEC-1, 32:POKEC+2, 32:POKEC-2,32:POKEC+3,32:POKEC-3
,32
617 POKEC-22,32:POKEC+22,32
640 E=E+22+J:POKEVA,15:POKEVN,0:
645 POKEE,62:POKEE-1,42:POKEE-2,60
646 FORO=248TO253:POKEVN-1,0:NEXT:FORO=253TO248STEP-1:POKEVN-1,0:NEXT
647 POKEE,32:POKEE-1,32:POKEE-2,32
650 IFE<7680+20*22THEN640
651 E=E+J
652 POKEE+22,62:POKEE+21,42:POKEE+20,60: POKEE+0F,4:POKEE+0F-1,4:POKEE+0F-2,4
653 POKEE+22+0F,0:POKEE+21+0F,0:POKEE+20+0F,32
654 POKEVN-1,0:POKEVN,128:FORQ=1TO20:POKEVA,15-INT(Q/1.33)
655 POKEE,223:POKEE-1,223:POKEE-2,223:FORO=1TO80:NEXT
656 POKEE,233:POKEE-1,233:POKEE-2,233:FORO=1TO80:NEXT
657 NEXT:POKEE,32:POKEE-1,32:POKEE-2,32:POKEE+22,32:POKEE+21,32:POKEE+20,32 :
658 PRINT "7680"
659 DU=DU+1:PRINT "7680JUFOS"DU"TANKS"DT:PRINT "XXXXXXXXXXXXXXXXXXXX"
660 GOTO125
700 POKEVN,128:L=0
701 A=A+1: FORKL=1TO200:POKEVA,15-INT(KL/13):
704 PRINTTAB(A) "XXXXXXXX"
705 PRINTTAB(A) "ER"
715 PRINT "DU";
720 PRINTTAB(A) "XXXXXXXX"
725 PRINTTAB(A) "ER"
735 PRINT "DU";
740 NEXT
745 PRINTTAB(A) " "

```



```

36 IFPEEK(8178)=160THENPRINT"THE MOON BASE DESTROYED !":GOTO500
40 GETA$:IFA$<>" "ANDG=0THENG=1:S=7680+15+22*21
50 IFG=0THEN80
55 POKES,32:S=S-22
60 IFS<746THENG=0:GOTO21
70 IFPEEK(S)=160THENPOKES,32:G=0:T=T+1:W=1:POKE9*16*3+13,128+000:U=15:GOTO80
71 IFPEEK(S-1)=160THENG=0:POKES-1,32:T=T+1:W=1:POKE9*16*3+13,128+000:U=15:GOTO8
0
75 POKES,81
80 F=1:REM CHECK MET
81 IFT=12THENPRINT"***METEOR DESTROYED***":FORRR=1TO2500:NEXT:LL=LL+44:C=LL
:GOTO2
82 FORE=0TO44STEP22
84 FORD=C+ETOC+3+E:IFPEEK(D)=32THENB(F)=32
86 F=F+1:NEXT:NEXT
90 GOTO21
500 POKE9*16*3+13,128+5
505 POKE9*16*3+14,5:FORRR=1TO300:NEXT
510 FORA=15TO0STEP-1
511 POKE9*16*3+14,A
520 FORRR=1TO500:NEXT
530 NEXT
540 FORRR=1TO2000:NEXT:RUN

```

```

1 REM"XXXXXXXXXX"
2 REM"XXXXXXXXXX"      ROCKET COMMAND      BY DUANE LATER
10 VI=9*1613:OF=38400-7680:PRINT"J"
11 FORA=38400+22T038400+22*23:POKEA,0:NEXT
15 C=7680+22*20+15
20 POKEVI+15,6+128+64+32+8
30 PRINT"***** ROCKET COMMAND ***";
35 PRINT"      HIT ANY KEY"
40 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
50 PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
60 A=7680+22*22+15
70 GET A$:IFA$<" "ANDB=0THENB=1:POKEC+22,32:D=C:C=C-1:K=1:POKE9*1613+13,128+125
L=16
71 IFK=1THENL=L-1:POKE9*1613+14,L
72 IFL=0THENK=0:POKE9*1613+13,0
75 IFC=8121THENPRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
NEXT:RUN
80 IFB=0THEN110
85 POKED,32:D=D-22
90 IFPEEK(D)=60ORPEEK(D)=62THENGOSUB500:GOTO70
100 IFC<7680+88THENB=0:GOTO110.
105 POKED,30
110 IFH=0THEN200
115 IFH=0THEN70
120 POKEF,32:F=F+E
125 IFF=1THENH=0:GOTO70
130 IFPEEK(F)=30THENGOSUB500:GOTO70
140 POKEF,G:GOTO70
200 H=1
205 IFINT(RND(1)*2)=1THENE=-1:F=7702+(INT(RND(1)*10)+6)*22:I=F-22:G=60:  GOTO115

```



```

210 E=1:F=7680+(INT(RND(1)*10)+6)*22:I=F+22:G=62:GOTO115
500 B=0:H=0
501 SC=SC+10:PRINT"SCORE="SC
502 POKEF+0F,4:POKEF+1+0F,4:POKEF-1+0F,4
503 POKEF+0F+22,4:POKEF-22+0F,4
510 POKEF,160:POKEF+1,160:POKEF-1,160:POKEF+22,160:POKEF-22,160
521 POKE9*16+13,128+35
522 FORY=16TO0STEP-1
523 POKE9*16+13+14,Y
524 FORP=1TO80:NEXT:Y
530 POKEF,32:POKEF+1,32:POKEF-1,32:POKEF+22,32:POKEF-22,32
531 POKEF+0F,0:POKEF+1+0F,0:POKEF-1+0F,0
532 POKEF+0F+22,0:POKEF-22+0F,0
533 POKE9*16+13,0
540 FORGH=FTOF+22*16STEP22
544 II=PEEK(GH):POKEGH,G:FOR00=1TO60:NEXT
546 POKEGH,II:NEXT
800 RETURN
999 GOTO70
1000 POKE9*16+13,128+125
1001 FORY=16TO0STEP-1
1005 POKE9*16+13+14,Y
1010 NEXT:POKE9*16+13,0
1020 RETURN

```

APPENDIX N: ERROR MESSAGES

This appendix contains a complete list of the error messages generated by the VIC, with a description of the causes.

BAD DATA...String data was received from an open file, but the program was expecting numeric data.

BAD SUBSCRIPT...The program was trying to reference an element of an array whose number is outside of the range specified in the DIM statement.

CAN'T CONTINUE...The CONT command will not work, either because the program was never RUN, there has been an error, or a line has been edited.

DEVICE NOT PRESENT...The required I/O device was not available for an OPEN, CLOSE, CMD, PRINT#, INPUT#, or GET#.

DIVISION BY ZERO...Division by zero is a mathematical oddity and not allowed.

EXTRA IGNORED...Too many items of data were typed in response to an INPUT statement. Only the first few items were accepted.

FILE NOT FOUND...If you were looking for a file on tape, and END-OF-TAPE marker was found. If you were looking on disk, no file with that name exists.

FILE NOT OPEN...The file specified in a CLOSE, CMD, PRINT#, INPUT#, or GET#, must first be OPENed.

FILE OPEN...An attempt was made to open a file using the number of an already open file.

FORMULA TOO COMPLEX...The string expression being evaluated should be split into at least two parts for the system to work with.

ILLEGAL DIRECT...The INPUT statement can only be used within a program, and not in direct mode.

ILLEGAL QUANTITY...A number used as the argument of a function or statement is out of the allowable range.

LOAD...There is a problem with the program on tape.

NEXT WITHOUT FOR...This is caused by either incorrectly nesting loops or having a variable name in a NEXT statement that doesn't correspond with one in a FOR statement.

NOT INPUT FILE...An attempt was made to INPUT or GET data from a file which was specified to be for output only.

NOT OUTPUT FILE...An attempt was made to PRINT data to a file which was specified as input only.

OUT OF DATA...A READ statement was executed but there is no data left unREAD in a DATA statement.

OUT OF MEMORY...There is no more RAM available for program or variables. This may also occur when too many FOR loops have been nested, or when there are too many GOSUBs in effect.

OVERFLOW...The result of a computation is larger than the largest number allowed, which is $1.70141884E + 38$.

REDIM'D ARRAY...An array may only be DIMensioned once. If an array variable is used before that array is DIM'd, an automatic DIM operation is performed on that array setting the number of elements to ten, and any subsequent DIMs will cause this error.

REDO FROM START...Character data was typed in during an INPUT statement when numeric data was expected. Just re-type the entry so that it is correct, and the program will continue by itself.

RETURN WITHOUT GOSUB...A RETURN statement was encountered, and no GOSUB command has been issued.

STRING TOO LONG...A string can contain up to 255 characters.

SYNTAX...A statement is unrecognizable by the VIC. A missing or extra parenthesis, misspelled keywords, etc.

TYPE MISMATCH...This error occurs when a number is used in place of a string, or vice-versa.

UNDEF'D FUNCTION...A user defined function was referenced, but it has never been defined using the DEF FN statement.

UNDEF'D STATEMENT...An attempt was made to GOTO or GOSUB or RUN a line number that doesn't exist.

VERIFY...The program on tape or disk does not match the program currently in memory.

INDEX

A

Abbreviations, BASIC commands 133
Accessories 106, 109
Addition 24, 115
AND operator 115
Animation 50, 51-66, 99, 139, 143
Arithmetic Operators 24, 116
Arithmetic Formulas 24, 115, 123, 148
Arrays 114-120
ASC function 131, 145
ASCII character codes 145
ATN function 129

B

BASIC

abbreviations 133
commands 115
operators 115
statements 119
variables 86, 113

Buffer 110

C

Calculations 24
Cassette tape recorder 107, 109
Cassette port 106
CHR\$ function 102, 131, 145
CLR statement 119
CLR / HOME key 6, 18
Clock 114
CLOSE statement 110, 119
Color
 Keys 19, 32
 Memory map 63, 143-144
 Screen and Border 33, 36, 38, 39, 134
Commands, BASIC 115
Commodore key (see graphics key)
Connecting the VIC to TV / monitor V
CONT command 115
CTRL key 18
CRSR keys 18, 60
Correcting errors 8, 50
Cursor 3, 18, 60
COSine function 129

D

DATA statement 79, 119
Data, saving & retrieving from tape 109
DEFine statement 120
Delay Loop 55, 78, 96
DELeTe key 3, 8, 19
DIMension statement 120
Division 115, 160
Duration (see FOR....NEXT)

E

Editing programs 8, 50
END statement 121
Equal, not-equal-to sign 115
Equations 115
Error Messages 6, 160
Expansion port 106, 149
EXPOnent function 129
Exponentiation 115

F

Files, cassette tape 109
FOR statement 121
FOR....NEXT loop 121
FRE function 132
Functions 129

G

Games to try 153
Game controls 108
Game port 108, 149
GET statement 89, 122
GET# statement 123, 111
Getting started V, 3
GOSUB statement 123
GOTO statement 123
Graphic keys 14, 19
Graphic symbols 14, 142, 146-147
Greater than 115

H

Hyperbolic functions 148

I

IEEE-488 Interface 107
IF...THEN statement 123
INPUT statement 84, 124
INPUT# statement 111
INSert key 3, 8, 19
INTeger function 129
Integer variables 113
I / O pinouts 149
I / O ports 106, 149

J

Joysticks 108

K

Keyboard 17-20

L

LEFT\$ function 131
LENGth function 131
Less than 115
LET statement 124
Line numbers 78
LIST command 8, 9, 50, 116
LOAD command 109, 116
Loading programs on tape 109
LOGarithm function 130
Loops, time delay 55, 78, 96
Lower case characters 20

M

Mailing label program 92
Mathematics
 formulas 24, 115, 123, 148, 160
 function table 148
 symbols 115
Memory 36, 80, 125, 126, 130
Memory expansion 107
MID\$ function 43, 131
Modulator, RF VI
Multiplication 115
Music
 pitch 68
 sound effects 135
 table of notes 73
 VIC piano 75
 writing songs 77

N

Names
 program 109
 variable 86
Nested Loops 122
NEW command 7
NEXT statement 125
Noise 71, 74
NOT operator 115
Numbers 23, 24, 115, 123
Numeric variables 86, 113

O

ON statement 125
OPEN statement 126
Operators
 Arithmetic 24, 114
 Logical 115
 Relational 115

P

Parentheses 115
PEEK function 130
Peripherals 106
Pi 20
POKE statement 36, 80, 125
Ports, I / O 149
POS function 132
PRINT statement 5, 21, 127
PRINT# 127
Programs
 editing 8, 50
 line numbering 79
 loading / saving on tape 109
Prompt 84

Q

Quotation marks 96

R

Random numbers 40, 43, 103
READ statement 79, 128
REMark statement 128
Reserved Words 86
Reset (see Restore Key)
Restore key 17, 26
RESTORE statement 128
Return key 18
RETURN statement 128
RIGHT\$ function 132
RND function 40, 43, 130
ROCKET COMMAND program 153
RUN command 117
RUN / STOP key 19

S

SAVE command 109, 117
Saving programs on tape 109
Screen memory maps 63, 143-144
Serial bus 107
SGN function 130
Shift key 18
SINe function 131
Sound effects 135
SPC function 132
SQR function 131
ST system variable 86
Stop key 19
STOP command 128
String variables 42, 86, 113
STR\$ function 132
Subscripted Variables 114
Subtraction 115
SYS statement 128
Syntax Error 6

T

TAB function 132
TAN function 131
Tape Cassette operation 109
TI variable 114
TI\$ variable 114
Time, setting VIC clock 114
TO in BASIC statements 121
Tones 70
TV, connecting the VIC V

U

Upper / lower case mode 20
USR function 131
User-defined function (DEF) 120

V

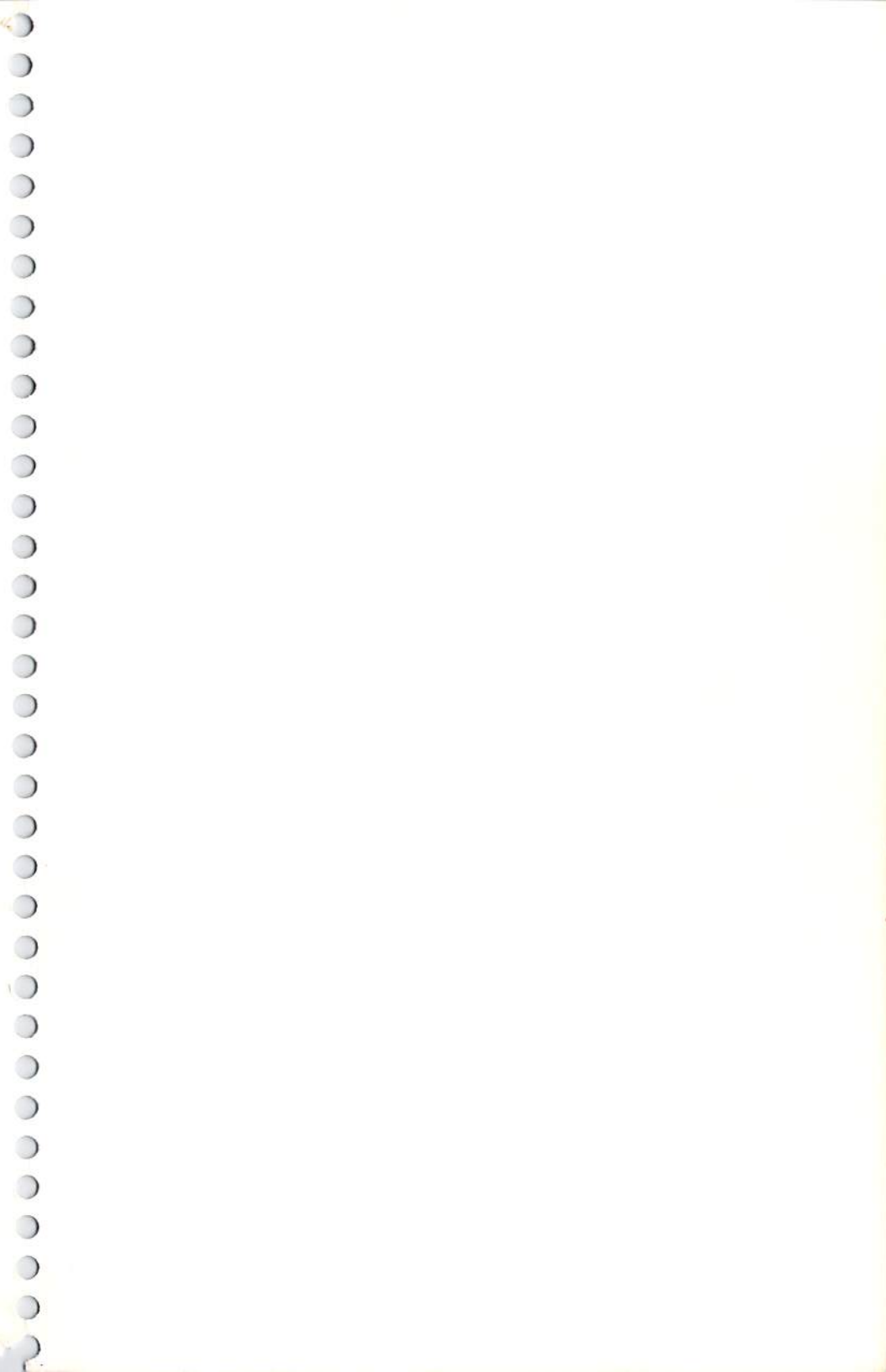
VALue function 132
Variables
 Array 114
 Floating point 113
 Integer 113
 Numeric 86, 113
 String 42, 86, 113
VERIFY command 109, 118
VICTIPS 3, 8, 16, 39, 40, 47, 50, 78, 96
VIC person (see animation)
Video port / connections V, 150
Volume 69, 71, 72

W

WAIT statement 129
Writing to tape 109

Y

Your Name in Lights (program) 95



A "USER FRIENDLY" COMPUTER

The new VIC computer is designed to be the most user friendly computer on the market...friendly in price, friendly in size, friendly to use and expand.

With the VIC, Commodore is providing a computer system which helps almost anyone get involved in computing quickly and easily...with enough built-in expansion features to let the system "grow" with the user as his knowledge and requirements become more sophisticated.

VIC owners who wish to learn more about computing should ask their Commodore dealer about these other self-teaching and reference materials:

- **VIC LEARNING SERIES...**a library of self-teaching books and tapes/cartridges which help you learn about computing and other subjects. Volume I in the VIC Learning Series is called "Introduction to Computing...On the VIC". Volume II is called "Introduction to BASIC Programming". Subsequent titles will include Animation, Sound and Music, and more.
- **VIC PROGRAMMER'S REFERENCE GUIDE...**a comprehensive guide to the VIC20 Personal Computer, including important information for new and experienced programmers alike.
- **VIC-PROGRAM TAPES, CARTRIDGES AND DISKS...**a growing library of recreational, educational and home utility programs which let you use the VIC to solve problems, develop learning skills, and play exciting television games. These easy-to-use programs require no previous computer experience.



commodore
COMPUTER