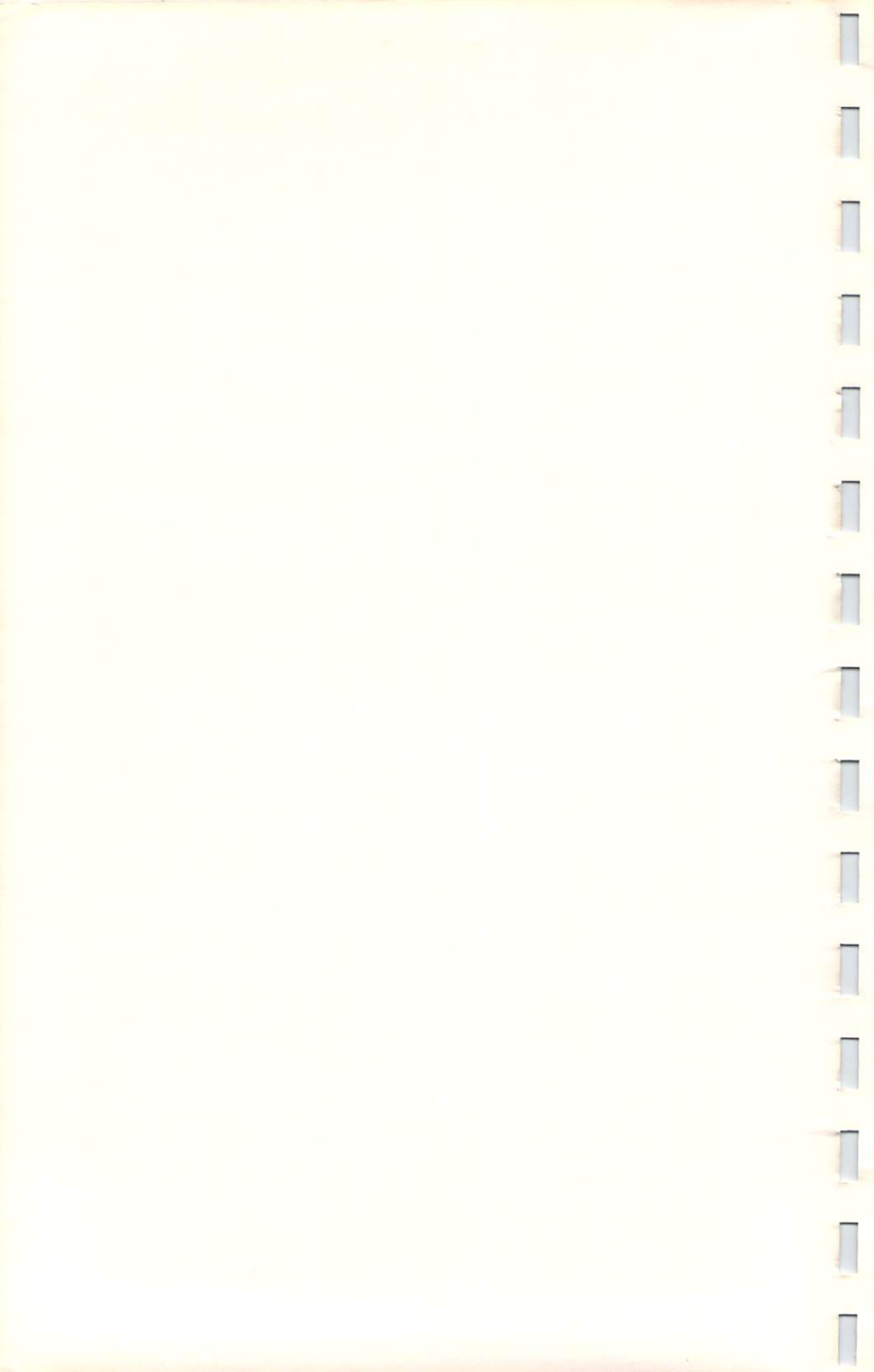


# Learn BASIC Programming in 14 Days on Your Commodore 64™

Gil M. Schechter

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
				1	2
				How to give instructions to the computer	H C
4	5	6	7	8	9
How to print on the screen	How to enter data into a program	How to control the flow of the program	Using loops	Saving programs and data on cassette tape	U
11	12	13	14	15	16
Poking and poking the memory	Sound and music	Print and poke graphics	Introduction to sprite graphics		
18	19	20			
25	26	27			

Commodore 64  
Commodore 64  
Commodore 64  
Commodore 64





**Learn BASIC Programming  
in 14 Days  
On Your Commodore 64™**

*Gil M. Schechter* has over twenty years experience in the computer field. He has worked in the areas of computer hardware and software design, artificial intelligence, and the application of computers to sales forecasting and market analysis.

Mr. Schechter received a degree in electrical engineering from the University of Florida in 1963, and has attended George Washington University and Johns Hopkins. He has taught several microcomputer courses to both children and adults, and is the author of *TI-99/4A: 51 Fun and Educational Programs* (SAMS #22192)



**Learn BASIC  
Programming in 14 Days  
On Your Commodore 64™**

**by  
Gil M. Schechter**

**Howard W. Sams & Co., Inc.**  
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

Copyright © 1984 by Gil M. Schechter

FIRST EDITION  
SECOND PRINTING—1984

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. While every precaution has been taken in the preparation of this book, the publisher assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-22279-5  
Library of Congress Catalog Card Number: 84-50052

Edited by *Diana Francoeur*  
Illustrated by *T. R. Emrick*

*Printed in the United States of America.*

## PREFACE

This book has been designed to help you learn how to program in BASIC on the Commodore 64. It is a friendly and easy-to-use "course" that you should be able to complete in about 14 days.

The first 8 days deal with acquiring fundamental skills, such as entering and editing a program, bringing data into a program, controlling the flow of the program, and saving programs and data on cassette tapes. The final 6 days deal with more advanced features, such as functions, arrays, PEEK and POKE, sound and music, and graphics.

The Question and Answer method has been used throughout. Complicated technical terms and "computerese" have been avoided. Over 130 programs and examples have been provided to give you hands-on experience.

Although written primarily for beginners (aged 13 and older), this book may also serve as a useful guide for those more-advanced programmers not familiar with the marvelous Commodore 64.

I hope you find this book both valuable and fun to use.

GIL M. SCHECHTER

## **ACKNOWLEDGEMENTS**

I would like to thank all those at Howard W. Sams & Co., Inc. who helped in the planning and production of this book. Also, I would like to thank my wife, Carole, for the long hours she spent in reviewing and preparing the manuscript, as well as offering many helpful suggestions.

## **DEDICATION**

*This book is dedicated to Carole, Jeff, and Lori.*

# CONTENTS

<b>How to Use This Book</b> .....	8
-----------------------------------	---

## PART I—FUNDAMENTAL SKILLS

<b>DAY 1.</b> How to give instructions to the computer .....	11
<b>DAY 2.</b> How to list and change a program .....	21
<b>DAY 3.</b> Working with numbers and strings .....	31
<b>DAY 4.</b> How to print on the screen .....	43
<b>DAY 5.</b> How to enter data into a program .....	55
<b>DAY 6.</b> How to control the flow of the program .....	67
<b>DAY 7.</b> Using loops .....	81
<b>DAY 8.</b> Saving programs and data on cassette tape .....	91

## PART II—MORE-ADVANCED FEATURES

<b>DAY 9.</b> Using functions .....	103
<b>DAY 10.</b> How arrays make programming easier .....	113
<b>DAY 11.</b> Peeking and poking the memory .....	125
<b>DAY 12.</b> Sound and music .....	133
<b>DAY 13.</b> Print and poke graphics .....	147
<b>DAY 14.</b> Introduction to sprite graphics .....	165

## APPENDICES

<b>Appendix A.</b> Commodore ASCII Codes .....	184
<b>Appendix B.</b> Screen Codes .....	186
<b>Appendix C.</b> Color Codes .....	188
<b>INDEX</b> .....	189

# HOW TO USE THIS BOOK

In order to run the examples and programs found in this book, you will need a Commodore 64 computer and a tv or monitor (preferably color, but not absolutely necessary). A Commodore cassette recorder will also be helpful, especially in the section for Day 8.

Whether your tv or monitor is in color or black and white, you will probably find that it is best to work with an all-white screen and dark letters. To turn your screen all-white, enter:

```
POKE 53280,1: POKE 53281,1
```

(hit RETURN.)

To print with dark letters:

Hold down the Commodore key ( **C** ) in the lower left corner of the keyboard and hit the 4 key.

This book is divided into 14 lessons. Each lesson can be covered in several hours. The exact amount of time you spend on each lesson will depend on such factors as your age and your experience, especially in using the BASIC programming language and working with computers, particularly the Commodore 64.

Everyone is different, so adjust the pace to your own needs, even if it means taking longer to complete the material covered. The quality of what you learn is far more important than how quickly you finish. Speed will increase as you gain experience, and what once seemed complicated will become very simple.

Before you begin a section, try to leaf through it and become as familiar with it as possible. Read all the questions. Look at the examples, drawings, and tables. Read the summary on the first page and the last page.

Next, go through the material in detail. Enter and run each example. Try to understand each program line. Try to change the examples and see the results. Use some of your own inputs in addition to the ones provided in the book.

After you complete the book, quickly skim all 14 lessons. Review anything that is not clear. When you finish, you should be well on your way toward a good understanding of Commodore BASIC.

Part I

**Fundamental Skills**





## Day 1

# HOW TO GIVE INSTRUCTIONS TO THE COMPUTER

In today's lesson you will learn:

- How to enter instructions into the computer.
- The difference between DIRECT MODE and PROGRAM MODE.
- How to enter and RUN a program.
- Your first 3 BASIC commands.



## **1** HOW DO I ENTER INSTRUCTIONS INTO THE COMPUTER?

When you turn on your computer, you will see this message on the screen:

```
**** COMMODORE 64 BASIC V2 ****
```

```
64 K RAM SYSTEM    38911 BASIC BYTES FREE
```

```
READY.
```

The word **READY** means that the computer is waiting for you to tell it what to do next.

Suppose you want the computer to print on the screen the words **HI THERE**. To do this, type in:

```
PRINT " HI THERE "
```

and hit the **RETURN** key. The computer will print on the screen:

```
HI THERE
```

The computer did exactly what you told it to do, but it did not do it until *after* you hit **RETURN**. Typing an instruction on the screen is not enough by itself. After you have finished typing your instruction, you must hit **RETURN**.

Do the same thing again, slowly. Type in **PRINT "HI THERE"**, but don't hit **RETURN** this time. As you type, notice the little blinking square. It is called a **CURSOR**. It tells you where you will be typing next on the screen. Every time you type a letter, the cursor moves one place to the right.

You can type 40 numbers, letters, or other characters on a line. If you type more than 40 characters, the cursor will automatically continue onto the next line. As far as the computer is concerned, however, your line ends when you hit **RETURN**.

Your cursor should now be to the right of the second quote mark after the word **THERE**. To erase part of what you just typed, hit the **INST/DEL** key several times. Notice that the cursor moves to the left every time you press this key. Now hold down the **INST/DEL** key. See how quickly letters are erased. Hold down **INST/DEL** until the entire line is erased.

Now let's clear the screen. Whenever you want to use the keyboard to clear the screen, do this:

Hold down the SHIFT key and hit the CLR/HOME key.

We will show this as:

**SHIFT** CLR/HOME

The reversed version of any key means that you should hold down that key while you hit another key. In this case, the other key is the CLR/HOME key.

## 2 HOW DO I USE THE CURSOR KEYS TO MOVE THE CURSOR?

Knowing how to move the cursor to the right place on the screen is very important. You will find that there are two cursor control keys at the lower right part of the keyboard. They look like this:



Using these keys, you can move the cursor to any position on the screen. The key on the left moves the cursor up or down. The key on the right moves the cursor left or right.

- To move the cursor down one line, press the left key one time.



- To move the cursor up one line, hold down SHIFT and press the left key one time.



- To move the cursor right one space, press the right key one time.



- To move the cursor left one space, hold down SHIFT and press the right key one time.



- If you hold down either of the cursor control keys, the cursor will move rapidly.

For practice, clear the screen and use the cursor control keys to help you type this pattern:

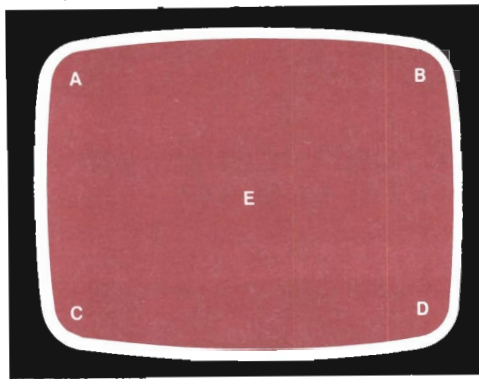


Fig. 1-1. Practice pattern for control of cursor.

### 3

## WHAT IS THE DIRECT MODE?

There are two ways for you to give instructions to the computer:

1. By using the DIRECT MODE.
2. By using the PROGRAM MODE.

Before, when you gave the computer the instruction PRINT "HI THERE", you used the DIRECT MODE. The DIRECT MODE is used to give the computer one or several short instructions, like the ones that follow.

Enter each instruction for practice. Remember to hit RETURN after entering each one.

```
PRINT " THIS IS MY COMPUTER"  
PRINT " THIS IS THE DIRECT MODE"  
PRINT 2+2
```

These PRINT commands tell the computer to print something on the screen. Later, we will learn how to give the computer other kinds of commands in DIRECT MODE.

When a computer performs an instruction, we say that the computer executes the instruction.

## 4

### WHAT IS PROGRAM MODE?

**PROGRAM MODE** is the second way to give the computer instructions. While **DIRECT MODE** is used to give the computer one or several short instructions, **PROGRAM MODE** is used to give the computer an *entire list of instructions*. This list is called a *program*.

In **DIRECT MODE**, the instructions do not have line numbers. In **PROGRAM MODE**, every instruction has a line number. The computer assumes that every instruction which starts with a number is in **PROGRAM MODE**.

In **DIRECT MODE**, the computer begins executing your instruction as soon as you hit **RETURN**. In **PROGRAM MODE**, when you hit **RETURN**, the instruction is not executed. Instead, it is stored in a special area of memory reserved for **BASIC** programs. To execute these stored instructions, type **RUN** and hit **RETURN**.

Now, for some practice. Enter this example of a **DIRECT MODE** instruction:

```
PRINT " HOW ARE YOU?" (hit RETURN)
```

Now try an example in **PROGRAM MODE**. Type **NEW** and hit **RETURN**. Then enter the following two instructions:

```
10 PRINT " HOW ARE YOU?" (hit RETURN)
20 PRINT " I AM FINE " (hit RETURN)
```

This creates a simple two-line program in memory. To execute it, type **RUN** and hit **RETURN**. On the screen, you should see:

```
HOW ARE YOU?
I AM FINE
```

Notice that in **PROGRAM MODE** the computer does not execute your program until you type **RUN** and hit **RETURN**.

## 5

### WHAT IS A COMPUTER PROGRAM?

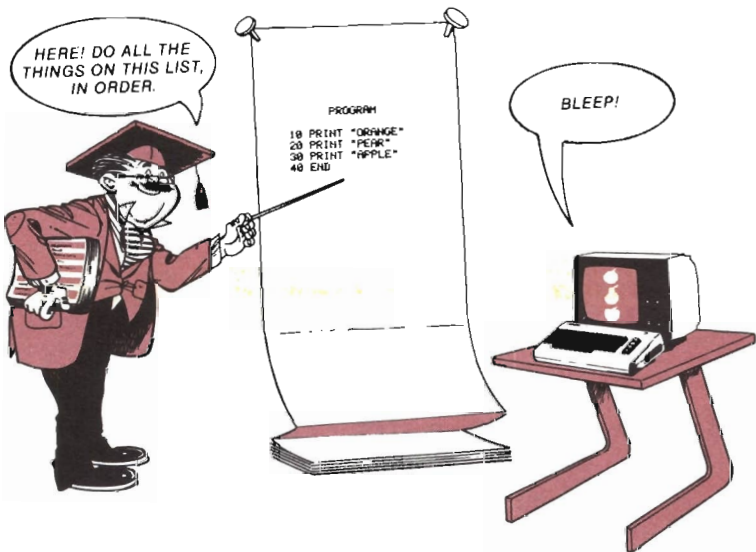
A *computer program* is a list of instructions to the computer. It tells the computer exactly what to do and in what order to do it. Look at the program on the next page, but do not type it in:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE"
30 PRINT "PEAR"
40 PRINT "APPLE"
50 END
```

There are a few things about this program that you should know:

1. It is written in a computer language called BASIC.
2. It has five program lines. Each line begins with a number.
3. The words REM, PRINT, and END are specially reserved words called *commands*. There are about 70 such BASIC commands which the Commodore 64 can understand. When you write a program, you use these commands to give instructions to the computer.
4. Each line of this program contains one BASIC statement. (For example: In line #20, PRINT "ORANGE" is a BASIC statement.) It is possible to put more than one statement on a line, as we shall see later.

You will notice that the line numbers go up in increments of ten (10, 20, 30, 40, 50). We number the lines this way because it allows us to



**A computer program is a list of instructions to the computer.**

“squeeze in” new lines between existing lines. If we wanted to add a line between line #10 and line #20, we could use any line number that falls between 10 and 20, such as line #15.

When we RUN a program, the computer will execute the lines in line-number order. The computer doesn't care what line numbers we use. All it cares about is the order of the line numbers. We could have numbered the lines 12, 73, 89, 107, and 305. The computer would then start with line #12, then do #73, and so on up to #305.

## 6 HOW DO I ENTER AND RUN THIS PROGRAM?

Here, again, is our short example program:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE"
30 PRINT "PEAR"
40 PRINT "APPLE"
50 END
```

To enter this program (or any other program) into the computer's memory, just follow these easy steps:

1. Clear the screen by holding down SHIFT and hitting the CLR/HOME key. You don't *have to* clear the screen to enter a program, but it's nice to start with a clean screen.
2. Type NEW and hit RETURN. NEW clears out any old programs in memory. You should *always* use NEW before you enter a new program.
3. Enter each line of your program, as follows:
  - Type the line number and a space.
  - Type in the entire line.
  - At the end of the line, hit RETURN. This puts the program line into a special area of memory reserved for BASIC programs.

To RUN your program, type RUN and hit RETURN. If there are no errors, you should see this on the screen:

```
ORANGE
PEAR
APPLE
```

These are the three words that you told your computer to print.

If instead of these three words, you saw a message that said:

```
?SYNTAX ERROR
```

just correct the problem as shown in Question #7.

**7**

## HOW DO I CORRECT PROGRAM ERRORS?

Nobody is perfect, and from time to time we all make mistakes. If you type in any of your program lines incorrectly, the computer will not be able to run your entire program. Instead, it will print a message on the screen, which will look like this:

```
?SYNTAX ERROR IN 20  
READY.
```

This means that there is something about line #20 that the computer could not understand. To correct the problem, just retype line #20 and hit RETURN. (Later, you will learn how to correct errors without retyping the entire line.) We used line #20 as an example. Errors can happen in line #20 or any other line of the program.

**8**

## HOW DO I STOP A PROGRAM BEFORE IT IS FINISHED?

To stop a program that is running, just hit the RUN/STOP key. Here is an example. Enter and RUN this program:

```
NEW  
10 PRINT "MY PROGRAM IS RUNNING"  
20 GOTO 10  
RUN
```

To stop this program, hit the RUN/STOP key. To continue the program, type CONT and hit RETURN.

You can also use the RUN/STOP key together with the RESTORE key. RESTORE returns the computer to the way it was when you first turned it on. To restore the computer to its original condition, hold down the RUN/STOP key and hit the RESTORE key:

**RUN/STOP** RESTORE



On the screen you should see:

READY.

Remember:

To stop a program, press the RUN/STOP key.

To continue the program, type CONT and hit RETURN.

To return the computer to the condition it was in when you first turned it on, hold down the RUN/STOP key and press the RE-STORE key.

## 9 WHAT ARE THE COMMANDS REM, PRINT, AND END USED FOR IN THIS PROGRAM?

REM stands for REMARK. Its only purpose is to put notes (or remarks) in the program. These notes make the program easier to understand when you or anyone else looks at it later. They tell you the purpose of the program and what different parts of the program do. When you run the program, the REM statements are ignored by the computer.

PRINT is used to print on the screen. We used it in our program to print the words ORANGE, PEAR, and APPLE. PRINT is probably the BASIC command you will use more than any other.

END is used to tell the program to come to a stop. It can be used in any part of the program, but it often appears on the last line. In some programs, it is not absolutely necessary to use END because the program stops anyway after executing the last line. However, if you want the program to stop automatically *before* it gets to the last line, then you must insert the END command earlier in the program.

These commands will become clearer as you use them throughout this book.

## 10 HOW CAN I PUT MORE THAN ONE PROGRAM STATEMENT ON THE SAME PROGRAM LINE?

You can put more than one program statement on the same program line by using a colon (:) to separate the statements. For example, our "Names of Fruits" program contains five statements, one on each line. Type NEW and enter this program:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE"
30 PRINT "PEAR"
40 PRINT "APPLE"
50 END
```

Type RUN and hit RETURN.

By using colons, you can put more than one statement on the same line, like this. Type NEW and enter:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE":PRINT "PEAR":PRINT"APPLE":END
```

Type RUN and hit RETURN. Running the program both ways will get the same result.

The same thing will also work in DIRECT MODE. If you type in:

```
PRINT " CAT " : PRINT " MOUSE " : PRINT " DOG "
```

and hit RETURN, the screen will show:

```
CAT
MOUSE
DOG
```

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. DIRECT MODE is used to give the computer one or several short instructions. PROGRAM MODE is used to give the computer an entire numbered list of instructions, which is called a *program*.
2. Before you enter a new program, you should always type NEW. This clears from memory any leftover old programs.
3. When you type in your program, it is stored in memory. To execute the program, type RUN and hit RETURN.



## HOW TO LIST AND CHANGE A PROGRAM

In today's lesson you will learn:

- How to LIST a program.
- How to add, delete, or replace a program line.
- How to edit a program line.



## 1 HOW CAN I LOOK AT MY PROGRAM WHEN IT IS IN THE COMPUTER'S MEMORY?

First, let's enter our example program that prints the names of fruits:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE"
30 PRINT "PEAR"
40 PRINT "APPLE"
50 END
```

Did you remember to type NEW before entering the program and to hit RETURN after each line? Now RUN this program to make sure that it works. If it doesn't, retype any bad lines.

Your program is now in a special area of memory that is reserved for BASIC programs. By using the LIST command, you can print the entire program, or any part of it, on the screen.

To look at the *entire* program, type LIST and hit RETURN. This is called *listing a program*. Listing a program is very helpful when you are trying to change it or when you are trying to find an error.

Some programs are so long that the entire program will not fit on the screen. When this happens, you will have to list only parts of the program at one time. Here are some examples that show you how this is done:

To list *only line #20*, type LIST 20 and hit RETURN.

To list all lines *up to line #30*, type LIST-30 and hit RETURN.

To list *line #20 and all lines after it*, type LIST 20- and hit RETURN.

To list *from line #20 through line #40*, type LIST 20-40 and hit RETURN.

Try each of these yourself.

Before you list a program, it is helpful to clear the screen. You don't *have to* do it, but it makes the listing easier to look at.

## 2 WHAT ARE THE DIFFERENT WAYS TO CHANGE A PROGRAM IN MEMORY?

When you have a program stored in the computer's memory, you may find that you need to change it for one of these two reasons:

1. You want to correct an error.
2. You want the program to do something different for you.

There are four ways to change a program in memory:

1. You can add an entire line.
2. You can remove an entire line.
3. You can replace an entire line.
4. You can edit parts of a line.

When you edit parts of a line, it means that you add, remove, or change one or several of the characters in that line, but the rest of the line stays the same.

You might ask at this point, "Is it better to replace an entire line or to edit it?" The answer to this question depends on the situation. If the line is short or if you have a great many changes to make, then it is usually better to replace it. If the line is long, or you have few changes, then it is probably better to edit it.

### **3 HOW DO I ADD A LINE TO MY PROGRAM?**

To add a line to your program in memory is very easy. All you have to do is type the line using a line number that *is not already in the program*. For instance, if you want to add a line between line #20 and line #30 of your program, you can use line #25 or any other line number between #20 and #30. The line number you use must not already be in your program.

Here is an example that will show you how you can add a line to our "Names of Fruits" program:

#### EXAMPLE

Suppose you want to add a new line to your program, between lines #20 and #30. You want the new line to print the word GRAPE. To do this, type:

```
25 PRINT " GRAPE " (hit RETURN)
```

When you hit RETURN, the line is added to your program in memory.

Now LIST your program to make sure the line has been added. On the screen you should see:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE"
25 PRINT "GRAPE"
30 PRINT "PEAR"
40 PRINT "APPLE"
50 END
```

RUN the program. The screen should show:

```
ORANGE  
GRAPE  
PEAR  
APPLE
```

## **4** HOW DO I REMOVE A LINE FROM MY PROGRAM?

To remove a line from a program in memory, just type the number of the line to be removed and hit RETURN.

### EXAMPLE

Suppose you want to remove line #25 from your program. To do this, type in:

```
25 (hit RETURN)
```

As soon as you hit RETURN, line #25 will be removed from the program in memory.

To make sure that line #25 has been removed, LIST the program. On the screen you should see:

```
10 REM NAMES OF FRUITS  
20 PRINT "ORANGE"  
30 PRINT "PEAR"  
40 PRINT "APPLE"  
50 END
```

Another word for "remove" is *delete*. In this example, we *deleted* line #25.

## **5** HOW DO I REPLACE A LINE IN MY PROGRAM?

To replace a line in the program with another line, just type the new line. Use the same line number as the line number of the line that you want to replace.

### EXAMPLE

Suppose you want to replace line #30. It now is:

```
30 PRINT "PEAR"
```

You want it to be:

```
30 PRINT "BANANA"
```

To do this, just type:

```
30 PRINT "BANANA" (hit RETURN)
```

As soon as you hit RETURN, the new line #30 will replace the old line #30.

LIST the program to make sure that the change has been made in memory. On the screen, you should see:

```
10 REM NAMES OF FRUITS
20 PRINT "ORANGE"
30 PRINT "BANANA"
40 PRINT "APPLE"
50 END
```

RUN the program to check it. The screen should show:

```
ORANGE
BANANA
APPLE
```

## 6 HOW DO I EDIT A LINE IN MY PROGRAM?

When you *edit* a line, it means that you are going to do one or more of the following:

1. Add one or several characters to a line in your program.
2. Delete (or remove) one or several characters from a line in your program.
3. Change one or several characters in a line that is in your program.

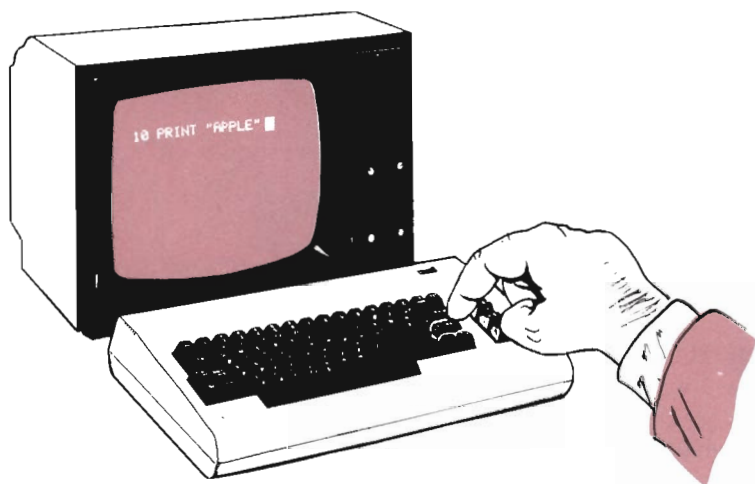
A good thing about being able to edit a line is that you can make a change in the line without having to retype it completely. Any time you want to edit a line in your program, you must follow the three steps described here:

1. Make sure that the line you want to edit is printed on the screen. If it isn't, then LIST the line. (You can LIST the line by itself or with other lines. The important point is that the line shows up on the screen.)

2. Using your cursor controls, move the cursor to the place where you want to make the change.
3. Make your change and hit RETURN. Just making the change on the screen is not enough. You must hit RETURN to put the change into memory.

If you have made many changes to your program, it is usually a good idea to LIST your program so that you can check it before you RUN it.

Questions #7, #8, and #9 will describe how to add, delete, or change characters in a program line.



**After you edit a program line on the screen, you must hit RETURN to put the change into memory.**

## **7** HOW DO I ADD CHARACTERS TO MY PROGRAM LINE?

Type in the following program:

```
NEW  
10 PRINT "APPLE"  
20 PRT "ORANGE"  
RUN
```

(Notice that the command PRINT in line #20 has been misspelled on purpose.)

When you RUN this program, you will see on the screen:



```
APPLE
?SYNTAX ERROR IN 20
READY.
```

Line #10 is correct and prints the word APPLE. Line #20 confuses the computer because it does not recognize the command PRT. To correct the error, you must add the characters "IN" after the "R."

To add the two letters "IN" after the "R" in line #20:

1. Move the cursor to the right of the "R" (over the "T") in line #20.
2. You want to add *two* letters, so you need to insert two spaces. To do this, hold down SHIFT and press the INST/DEL key two times. This will open two spaces after the "R" as the "T" shifts over to the right.
3. Type the letters "IN" and hit RETURN.

Now move the cursor down to a clear line, below any printing on the screen. Type RUN and hit RETURN. On the screen, you should see:

```
APPLE
ORANGE
```

Using the same procedure, can you change the word APPLE in line #10 to PINEAPPLE?

## **8 HOW DO I DELETE CHARACTERS FROM MY PROGRAM LINE?**

Type in the following program:

```
NEW
10 PRINNNT "TRAIN"
20 PRINT "AIRPLANE"
RUN
```

(Notice that the command PRINT in line #10 has been misspelled on purpose.)

When you RUN the program, you will see on the screen:

```
?SYNTAX ERROR IN 10
READY.
```

To correct line #10, you will have to delete two of the three "N's." Here are the steps to follow:

1. Move the cursor to the right of the last "N" (over the "T").
2. Press the INST/DEL key two times. This will remove two of the three "N's."
3. Hit RETURN to put the change into memory.

Now move the cursor down to a clear line, below any printing on the screen. Type RUN and hit RETURN. On the screen, you should see:

```
TRAIN  
AIRPLANE
```

Using the same procedure, can you change the program so that it prints the word PLANE instead of the word AIRPLANE?

## **9** HOW DO I CHANGE CHARACTERS IN MY PROGRAM LINE?

Type in the following program:

```
NEW  
10 PRINT "FRED HANSON"  
20 PRINT "JIM COLE"  
RUN
```

When you RUN the program, you will see on the screen:

```
FRED HANSON  
JIM COLE
```

To change HANSON to HANSEN, follow these steps:

1. Move the cursor over the character you want to change in line #10, which is the "O."
2. Type the correct character, which is "E."
3. Hit RETURN to put the change into memory.

Now move the cursor to a clear area of the screen, below any printing, and RUN the program. On the screen you should see:

```
FRED HANSEN  
JIM COLE
```

Using the same procedure, can you change the program to print the name KOLE instead of the name COLE?

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. BASIC programs are stored in a special area of the computer's memory. When you LIST a program, you actually print out whatever is stored in this special part of memory.
2. To add a line to your program, use a line number between two existing line numbers. To delete or replace a line, use the line number of the program line you are going to delete or replace.
3. Before you edit a line, it must appear on the screen. Making the change on the screen is not enough, however. You must hit RETURN to put the change into memory.

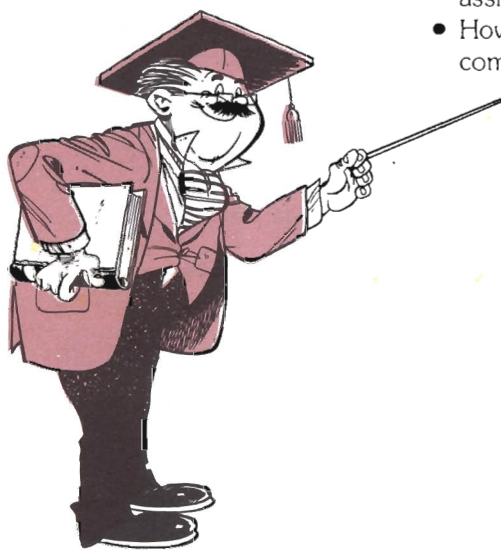




## WORKING WITH NUMBERS AND STRINGS

In today's lesson you will learn:

- The difference between numbers and strings.
- What a variable is and how to assign it a value.
- How to do math on the computer.



# 1 WHAT IS THE DIFFERENCE BETWEEN NUMBERS AND STRINGS?

Before you can program a computer in BASIC, you must understand the important difference between numbers and strings.

*Numbers* are used for counting and math, and we are all familiar with them.

## Examples of Numbers

3	.07
42	400
134.5	7.7

Strings can be letters, words, or symbols (such as @\*?!).

## Examples of Strings

"CAT"	"F15"
"JOHN SMITH"	"XY! *?"
"ABCD"	"2147"

As you can see from the examples, strings must be enclosed in quotes.

Let's look carefully at the string "2147". This is not the same as the number 2147. The string "2147" could be an address, a license plate number, or part of a telephone number, whereas the number 2147 could be the price of a computer or the number of jelly beans in a jar.

You can use the number 2147 in a mathematical operation, such as addition, subtraction, multiplication, or division. You *cannot* use the string "2147" in a mathematical operation—you can use it only to identify something (like a house address, telephone number, or license plate).

It is all right to use:        PRINT 2147+3

It is NOT all right to use: PRINT "2147"+3

Try both and see what happens. (Don't worry; the computer won't blow up!)

We said before that strings can be letters, words, or symbols such as @\*!?, etc. In Commodore BASIC, strings can also include colors, special graphics, characters, and even cursor commands. We will cover these special strings in more detail later.

All data are made up of either numbers or strings. From now on, whenever we use the word *data*, we will mean either numbers or strings.



## 2

### WHAT IS A NUMERIC VARIABLE?

A numeric variable *stands for* a number. Here is a simple example to help you understand how numeric variables are used. Type NEW and enter this program:

```
10 AGE=18
20 PRINT AGE
30 AGE=25
40 PRINT AGE
```

Type RUN and hit RETURN. After you run this program, you should see on the screen:

```
18
25
```

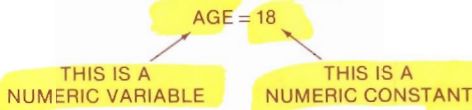
In this example, AGE is a numeric variable that can stand for different numbers. In line #10, we make AGE stand for the number 18. In

line #20, we print whatever AGE stands for at that time, which is 18. In line #30, we change AGE to 25. In line #40, we print whatever AGE stands for at that time, which is 25.

As you can see from this example, the numeric variable AGE can stand for many different numbers. In the preceding example, it stands for the number 18 at one time and for the number 25 at another time.

Because the number that AGE stands for can change (or vary), we call AGE a *variable*, and because AGE can stand only for numbers (but not for strings), we call it a *numeric variable*.

By the way, the number that we make AGE equal to is called a *numeric constant*. Any number is a numeric constant. In the following example, the number 18 is always equal to 18 (its value is constant). *Numeric constant* is a fancy term for number.



Now, let's look at some examples of other numeric variables besides AGE. In the program that follows, each numeric variable is underlined. *Do not enter or run this program; just look at it carefully.*

```
10 X=7
20 XYZ=100
30 Q1=3
40 WEIGHT=160
```

There are four numeric variables in this program. They are X, XYZ, Q1, and WEIGHT. Each one is made equal to a number (or *numeric constant*, if you prefer).

A numeric variable can be written as a single letter, several letters, a letter and number, or an entire word (such as WEIGHT). But no matter how you write it, the numeric variable has only one purpose—to stand for a number.

Beside *numeric variables* (which stand only for numbers), there are also such things as *string variables* (which stand only for strings). String variables will be explained next.

### 3

## WHAT IS A STRING VARIABLE?

A *string variable* is very similar to a numeric variable, except that it stands for a string and not a number. Here is a simple program to help you understand how string variables are used. Type NEW and enter this program:

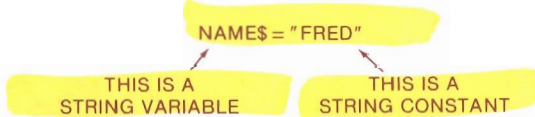


```
10 NAME$="FRED"  
20 PRINT NAME$  
30 NAME$="LARRY"  
40 PRINT NAME$
```

Type RUN and hit RETURN. After you run this program, you should see the names FRED and LARRY on the screen.

In this example, NAME\$ is a string variable that can stand for different strings. In line #10, we make NAME\$ equal to the string "FRED". In line #20, we print whatever NAME\$ happens to stand for at the time, which is the string "FRED". In line #30, we change NAME\$ to stand for the string "LARRY". In line #40, we print whatever NAME\$ happens to stand for at this time, which is the string "LARRY".

The string variable NAME\$ can stand for many different strings. In the preceding example, we made it stand for the strings "FRED" and "LARRY". Because the strings that NAME\$ stands for can change (or vary), we call NAME\$ a *variable*, and because NAME\$ can stand only for strings (and not numbers), we call it a *string* variable. The string that we made NAME\$ equal to is called a *string constant*.



The following program shows examples of other string variables. Do not enter or run this program; just look at it carefully:

```
10 A$="HOUSE"  
20 CI$="CAR"  
30 CITY$="NEW YORK"  
40 DESC$="TALL AND THIN"
```

All four string variables in this program are underlined.

A string variable can be written as a single letter, a letter and number, or several letters which form a word. All string variables must end with a \$ sign to tell the computer that they are string variables and not numeric variables. But, no matter how you write it, a string variable has only one purpose—to stand for a string.

## 4 WHAT ARE THE RULES FOR NAMING VARIABLES?

The person who writes a program is usually the one who makes up the names of the variables used in the program. Whenever you make

up the name of a variable for your program, you must follow these rules:

## Rules for Naming Variables

1. All names must start with a letter.
2. String variables must end with a \$ sign.
3. The computer looks only at the first two characters of the variable name. The variables CAR\$, CAT\$, and CAP\$ are considered the same variable. Therefore, different variables must not have the same first two characters.
4. You may not use variable names that contain in them any of the 70 BASIC commands. For example, you cannot use the variable name PRINTS because it contains the BASIC command PRINT.

Here are some examples of variable names. These names are all right to use for numeric variables:

AGE	F7
GRADE	N
TEMP	WEIGHT

These names are all right to use for string variables:

Q2\$	A1\$
NAMES\$	MONTHS\$
CITY\$	X\$

The following names are NOT all right, or can cause a problem:

3F	(Not all right because name must start with a letter)
NAME	(All right for numeric variable, but if string variable then it must end with \$ sign)
BATS } BALLS }	(All right, but will be considered the same variable)
RUNS	(Not all right. Contains the BASIC command RUN)

## 5 HOW DO I USE THE = TO ASSIGN A VALUE TO A VARIABLE?

The = sign is used to assign a value to a variable. We used it in one of our previous examples:

```
10 AGE=18
```

When we say AGE = 18, we make AGE equal to the number (or

numeric constant) 18. However, there are two other ways to do the same thing. We can say:

```
10 AGE=16+2
```

Or we can say:

```
10 X=18
20 AGE=X
```

In other words, when we use the = sign to assign a value to a **numeric variable**, that **value can be**:

1. A number (Example: AGE = 18)
2. A calculation (Example: AGE = 16 + 2)
3. Another variable (Example: 10 X = 18  
20 AGE = X)

In the same way, when we use the = sign to assign a value to a **string variable**, that **variable can be**:

1. A string (Example: NAME\$ = "FRED")
2. Another variable (Example: 10 A\$ = "FRED"  
20 NAME\$ = A\$)

The **BASIC command LET** is sometimes used together with the = sign. For example:

```
LET AGE=5
LET NAME$="GEORGE"
```

In Commodore BASIC, the LET command is not required and is usually not used.

## 6 HOW DO I GET THE COMPUTER TO ADD, SUBTRACT, MULTIPLY, AND DIVIDE?

Addition, subtraction, multiplication, and division are called *mathematical operations*. In BASIC, we use a symbol to tell the computer which mathematical operation we want it to do. These symbols are shown here:

<b>Mathematical Operation</b>	<b>Basic Symbol</b>
Add	+
Subtract	-
Multiply	*
Divide	/

Here are some short program examples. RUN each of these and see if you can get the right answer. To add  $3 + 4$ , enter this program:

```
NEW
10 S=3+4
20 PRINT S
RUN
```

After you have run the program, the screen should show 7.  
To subtract 5 from 10, enter and RUN this program:

```
NEW
10 D4=10-5
20 PRINT D4
RUN
```

The screen should show 5.  
To multiply 8 times 9, enter and RUN this program:

```
NEW
10 P=8*9
20 PRINT P
RUN
```

The screen should show 72.  
To divide 24 by 3, enter and RUN this program:

```
NEW
10 Q=24/3
20 PRINT Q
RUN
```

The screen should show 8.  
Do you realize that in these four examples we used four different numeric variables? What were they? Go to the head of the class if you said S, D4, P, and Q!

Let's take a look at the first of our short examples:

```
10 S=3+4
20 PRINT S
```

In this example, we make S equal to  $3 + 4$  and then print the value of S. Another way to do the same thing is to put the calculation right into the PRINT command as shown:

```
10 PRINT 3+4
```

Both ways will give the same answer.

By putting the calculation right into the PRINT command, you save writing a program line; but that doesn't mean it is the best way to do it in all cases. If you need the value of that same calculation in several different parts of the program, then it is better to make that value equal to a variable S. Then, whenever you need that value, you just use the variable S, and you don't need to recalculate every time. *This is a major reason why we use variables.*

You can also put the calculation into the PRINT command when you are in direct mode. Here are some examples to try:

```
PRINT 10-5
PRINT 8*9
PRINT 24/3
```

## **7** HOW CAN I CALCULATE POWERS WITH THE COMPUTER?

Calculating a number to a power is very easy to do on your computer. Just use the symbol  $\uparrow$  before the power.

Here are some examples. To calculate  $2^2$ , enter and RUN this program:

```
NEW
10 AREA=2  $\uparrow$  2
20 PRINT AREA
RUN
```

The screen should show 4.

To calculate  $3^3$ , enter and RUN this program:

```
NEW
10 VOLUME=3  $\uparrow$  3
20 PRINT VOLUME
RUN
```

The screen should show 27.

To calculate  $5^2 + 7$ , enter and RUN this program:

```
NEW
10 X=(5  $\uparrow$  2)+7
20 PRINT X
RUN
```

The screen should show 32.

## 8

## IN WHAT ORDER ARE MATHEMATICAL OPERATIONS PERFORMED?

Suppose one of your program lines has several mathematical operations in it, like this:

$$10 \quad X=4+6/2$$

Does this mean:

$$X=(4+6)/2$$

Therefore  $X = 5?$

Or does it mean:

$$X=4+(6/2)$$

Therefore  $X = 7?$

You can tell the computer *exactly* what you want it to do by putting parentheses around the operations you want done as a group. If you don't use parentheses, the computer will make the calculations in this order:

1. It will calculate all powers first.
2. Next, it will perform all multiplications and divisions.
3. Lastly, it will perform additions and subtractions.

Therefore,  $X = 4 + 6/2$  will be calculated as  $X = (4) + (6/2) = 7$ .

Here is another example. Suppose your program line is:

$$K=2 \uparrow 2+4*3$$

If you don't use parentheses ( ), the computer will make the calculations in this order:

1. Powers ( $2 \uparrow 2$ )
2. Multiplication and division ( $4*3$ )
3. Addition ( $2 \uparrow 2$ ) + ( $4*3$ )

Therefore,

$$\begin{aligned} K &= (2 \uparrow 2) + (4*3) \\ &= (4) + (12) = 16 \end{aligned}$$

If this is not what you want the computer to do, then you must tell it what to do by using parentheses. Suppose what you really wanted was  $K=(2^2 + 4)*3$ . Then use parentheses and enter your program line like this:

$$K = ((2 \uparrow 2) + 4)*3$$

The value of K will then be calculated as:

$$\begin{aligned} K &= (4 + 4) * 3 \\ &= 8 * 3 \\ &= 24 \end{aligned}$$

A good rule is: whenever you have several mathematical operations in the same program line, use parentheses to avoid confusion.

## 9 HOW CAN I ADD STRINGS TOGETHER TO FORM LONGER STRINGS?

The mathematical symbol + can be used to add strings together and in that way form longer strings. For example, try these lines in DIRECT MODE:

```
PRINT "ABC" + "DEF" should result in ABCDEF .  
PRINT "HOT" + " " + "DOG" should result in HOT DOG .
```

Notice that " " is considered a string.  
Now type NEW and enter this program:

```
10 A$="GEORGE"  
20 B$=" "  
30 C$="WASHINGTON"  
40 PRINT A$+B$+C$
```

Type RUN and hit RETURN. On the screen you should see:

```
GEORGE WASHINGTON
```

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. Numbers are used for counting and math. Strings are often used to identify something (name, license number, etc.).
2. *Numeric variables* stand for numbers. *String variables* stand for strings. The person who writes the program usually makes up the names of the variables.
3. If you are doing calculations and your BASIC statement has several variables in it, use parentheses to tell the computer how you want the calculations grouped together.





## Day 4

# HOW TO PRINT ON THE SCREEN

In today's lesson you will learn:

- How to clear the screen using the PRINT command.
- How to print on the screen in various ways.
- How to print reversed characters.
- How to move the cursor by using the PRINT command.



## 1 WHAT ARE THE DIFFERENT WAYS I CAN USE THE *PRINT* COMMAND?

PRINT is probably the BASIC command you will use more often than any other. In fact, you have already used it in some of the very simple examples we have had so far. PRINT is, of course, used to print data (numbers and strings) on the screen. The data can be printed in several different ways:

1. You can print data in standard columns. The screen is divided into four such standard columns.
2. You can print data items near each other.
3. You can print data a certain number of spaces apart. Here, you tell the computer how many spaces apart to print this data.
4. You can also print data at different TAB settings.

You can also use the PRINT command to print *reversed* characters. These are characters whose colors are printed backwards. For example, if you normally print black letters on a white background, then reversing the colors will print white letters on a black background.

In Commodore BASIC, you can also put cursor movements in the PRINT command so that your program is able to move the cursor to a certain spot, before you print on the screen.

Finally, one of the most important uses of the PRINT command is to clear the screen. So, let's begin here.

## 2 HOW CAN I USE THE *PRINT* COMMAND TO CLEAR THE SCREEN?

You have already learned how to clear the screen by using the keyboard. You just hold down the SHIFT key and hit the CLR/HOME key.

**SHIFT** CLR/HOME

To get a program to clear the screen, you use the same action with a PRINT command, like this:

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

First, type some letters and characters; then RUN this one-line program to see how it works. It should clear the screen.

LIST the program. It will show on the screen as:

```
10 PRINT "♥"
```

It may seem a bit strange, but in Commodore BASIC, certain commands, like this one, LIST as graphic symbols. The symbol for "SHIFT and CLR/HOME" lists as a heart. Unfortunately, this does not describe clearing the screen very well.

From now on in this book, the BASIC statement to clear the screen will be shown as:

```
PRINT "{SC}"
```

The SC stands for SCREEN CLEAR.

**Very Important:** Whenever you see a program line in this book that looks like this:

```
10 PRINT "{SC}"
```

enter it into the computer as:

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

When you list your program, it will show on the screen as:

```
10 PRINT "♥"
```

### 3 HOW DO I PRINT ON THE SCREEN IN STANDARD PRINT COLUMNS?

To help you understand what standard columns are all about, type NEW and enter this program:

```
10 PRINT "{SC}"
20 PRINT 5,6,7,8
30 PRINT 1,7,6,4
40 PRINT 4,3,1,20
```

Type RUN and hit RETURN.

After you run the program, you will see all the numbers printed in four columns on the screen. These four columns are the *standard print columns*.

The Commodore 64 can normally print up to 40 characters on a single line. These 40 characters are divided into the four standard print columns. Each of these columns is 10 characters long:

Column 1 starts in position 1.

Column 2 starts in position 11.  
Column 3 starts in position 21.  
Column 4 starts in position 31.

To print in these standard columns, use the PRINT command and put a *comma* between the data that you want to print in each column. Every time the PRINT command sees another comma, it will move to the start of the next standard print column.

Here are some examples for you to try. For each one, clear the screen; then type NEW and hit RETURN.

To print a single number in the first column, enter and RUN:

```
10 PRINT 4
```

To print four columns of numbers, enter and RUN:

```
10 PRINT 5,7,93,241
```

To print two columns of strings, enter and RUN:

```
10 PRINT "PENCIL", "ERASER"
```

To print the value of numeric variables in four columns, enter and RUN:

```
10 W=10:X=15:Y=20:Z=25
20 PRINT W,X,Y,Z
```

To print string variables in three columns, enter and RUN:

```
10 A$="HAT":B$="COAT":C$="GLOVES"
20 PRINT A$,B$,C$
```

You can mix numeric and string variables in the same PRINT command. As an example, enter and RUN:

```
10 N=5:A$="GIRLS"
20 PRINT "THERE ARE",N,A$
```

Using these standard print columns, you can print up to four columns of data on a single line. If you try to print more than four columns, the rest will show up on the next line. To prove this, type NEW and enter the following one-line program:

```
10 PRINT 101,102,103,104,105,106
```

Type RUN and hit RETURN. The 105 and 106 will print on the next line.

If you try to print very large numbers or strings in a standard column, you will quickly discover that not all of the 10 positions are used for printing. The longest single number that you can print in a standard column is seven digits long (like 1234567). The longest single string that you can print in a standard column is nine characters long (like "ABCDEFGHI").

The following program will show you how numbers and strings line up on the screen. Type NEW and enter:

```
5 PRINT "(SC)"
10 L$="1234567890"
20 PRINT L$+L$+L$+L$
30 N=1234567
40 PRINT N,N,N,N
50 Q$="ABCDEFGHI"
60 PRINT Q$,Q$,Q$,Q$
```

Type RUN and hit RETURN.

After you run the program, look carefully at the screen. You will see that numbers begin in position 2 of the standard column and run up to position 8. Strings begin in position 1 and run up to position 9. If you change the number 1234567 to a longer number, or if you change "ABCDEFGHI" to a longer string, they will no longer fit into a standard column.

#### **4 HOW DO I PRINT NUMBERS AND STRINGS NEAR EACH OTHER?**

You don't *have* to print in standard columns. Sometimes you want to print numbers and strings near each other. To do this, just put a semicolon (;) between the data that you want to print in this way. Let's try some examples. For each one, type NEW and hit RETURN.

To PRINT seven numbers, enter and RUN:

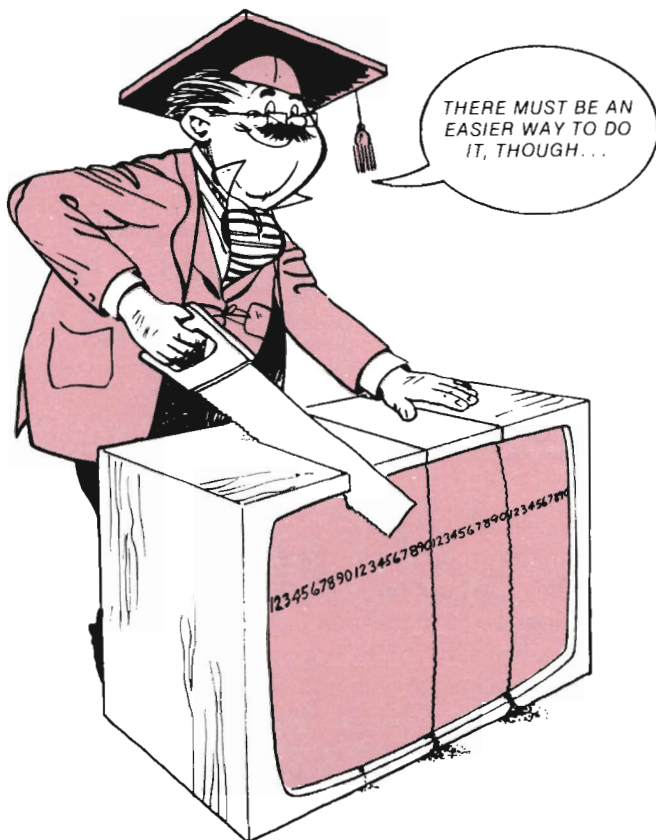
```
10 PRINT 1;2;3;7;23;19;874
```

Notice how many more numbers we can print on a single line this way. Also, notice that the computer automatically prints 2 spaces between the numbers.

To PRINT two strings, enter and RUN:

```
10 PRINT "HOT"; "DOG"
```

Notice that the computer prints no spaces between strings. If you want a space between "HOT" and "DOG", then use:



**The screen can be divided into four standard columns of 10 characters each.**

```
PRINT "HOT "; "DOG"
```

To PRINT four numeric variables, enter and RUN:

```
10 W=10 : X=15 : Y=20 : Z=25
20 PRINT W;X;Y;Z
```

To PRINT three string variables, enter and RUN:

```
10 A$="TICK": B$="TOCK": C$="  "
20 PRINT A$; C$; B$
```

Notice the " " is considered a string.

You can mix numbers and strings in the same PRINT command. As an example, enter and RUN:

```
10 X=2*3
20 PRINT "THE ANSWER=" ; X
```

You have now seen how you can print data items near each other by using a semicolon (;) in the PRINT statement. A big advantage of printing your data in this way is that you can usually print more data items on the same line than you can by using the standard columns.

Standard columns, however, do have an advantage of their own. When you print several lines in standard columns, the data items line up a lot better than they do when you print them using the semicolon.

Here is a program that shows how data items line up when printed using a semicolon. Type NEW and enter:

```
10 PRINT 10;20;30;40
20 PRINT 100;200;300;400
30 PRINT 1000;2000;3000;4000
```

Type RUN and hit RETURN.

## 5 HOW CAN I USE SPC (X) TO PRINT SPACES BETWEEN DATA?

You have already learned how to:

1. Print data items in standard print columns.
2. Print data items near each other.

Now you will learn how to separate the data items with a certain number of spaces by using SPC (X). The X stands for the number of spaces by which you want to separate the data.

Suppose that you want to print the letter "A" and the letter "B." To skip 3 spaces between A and B, enter and RUN:

```
10 PRINT "A" SPC (3) "B"
```

To skip 11 spaces between A and B, enter and RUN:

```
10 PRINT "A" SPC (11) "B"
```

Instead of putting a number in the parentheses (), you can use a variable. Suppose you want to print A, skip 7 spaces, print B, skip 11 spaces, and print C. To do this, type NEW and enter the following program:

```
10 S1=7
20 S2=11
30 PRINT "A" SPC(S1) "B" SPC(S2) "C"
```

Type RUN and hit RETURN. As you can see, by using SPC (X), you can skip different numbers of spaces between each data item.

## 6 HOW CAN I PRINT DATA AT DIFFERENT TAB POSITIONS?

TAB (X) lets you jump to a certain tab position on the print line before printing. The X stands for the number of the print position. Each print line in the Commodore 64 has 40 print positions.

To print ABC starting at print position 20 and to print DEF starting at print position 30, enter and RUN:

```
10 PRINT TAB (20) "ABC" TAB (30) "DEF"
```

You can also use numeric variables to tell the computer the tab positions that you want. To print the number 200 at print position 10, 350 at print position 20, and 777 at print position 30, type NEW and enter the following program:

```
10 T1=10
20 T2=20
30 T3=30
40 PRINT TAB(T1) 200 TAB(T2) 350 TAB(T3) 777
```

Type RUN and hit RETURN.

## 7 HOW DO I MAKE MY PROGRAM PRINT REVERSED CHARACTERS?

Reversed characters are those with their colors printed in reverse. If you normally print black characters on a white background, then reversing them will print white characters on a black background.

To print the string "ABCDEFGH I" normally, enter and RUN:

```
10 PRINT "ABCDEFGH I"
```

However, to print the same string "ABCDEFGH I" with "DEF" reversed, type NEW and enter:

```
10 PRINT "ABC (RV) DEF (RO) GHI"
```

```
{RV} = CTRL RVS/ON
{RO} = CTRL RVS/OFF
```



Type RUN and hit RETURN. *Note:* {RV} means hold down the CTRL key and hit the RVS/ON key (same as the 9 key). {RO} means hold down the CTRL key and hit the RVS/OFF key (same as the zero key).

Now let's try the same thing, but let's make the actions SCREEN CLEAR, CTRL RVS/ON, and CTRL RVS/OFF equal to string variables. To see how this works, type NEW and enter the following program:

```
10 SC$="{SC}"
20 R$="{RV}"
30 X$="{RO}"
40 PRINT SC$;"ABC";R$;"DEF";X$;"GHI"
```

```
{RV} = CTRL RVS/ON
{RO} = CTRL RVS/OFF
{SC} = SHIFT CLR/HOME
```

Type RUN and hit RETURN.

Reversed characters are used to make printing stand out. For a demonstration, type NEW and enter the following program:

```
10 PRINT "{SC}"
20 PRINT "{RV} HEADLINE {RO}"
30 PRINT
40 PRINT "NOTICE HOW {RV} REVERSE {RO} PRINTING"
50 PRINT "MAKES SOME WORDS {RV} STAND OUT "
60 PRINT
70 PRINT "ISN'T THAT NEAT?"
```

```
{RV} = CTRL RVS/ON
{RO} = CTRL RVS/OFF
{SC} = SHIFT CLR/HOME
```


Type RUN and hit RETURN.


## 8 HOW CAN I USE THE PRINT COMMAND TO CONTROL THE CURSOR'S POSITION ON THE SCREEN?

You have already learned how to use the two cursor control keys (marked CRSR) to move the cursor on the screen. Now you will learn how to get the PRINT command in your program to do the same thing.

For example, suppose you want to print an A, have the cursor move three spaces to the right, and then print a B. To do this, type NEW and enter the following program:

```
10 PRINT "A{CR}{CR}{CR}B"
```





```
{CR} = 
```

Type RUN and hit RETURN. Note: {CR} means press the  key one time.

After you run this program, you should see on the screen:


A space space space B

In future examples, you will see symbols just like {CR}. Here is what these symbols mean and how to type them in:

Symbol	Meaning	Keys to Type
{CR}	cursor right	
{CL}	cursor left	<b>SHIFT</b> 
{CD}	cursor down	
{CU}	cursor up	<b>SHIFT</b> 

Here is another program to try. Type NEW and enter:

```
10 PRINT "A{CD}{CD}{CD}{CL}B"
```

{CD} = 

{CL} = **SHIFT** 


Type RUN and hit RETURN. On the screen you should see:

A

B

And finally, try this example. Type NEW and enter:

```
10 PRINT "{CD}{CD}{CD}{CD}{CD}X{CD}Y{CU}X{CD}Y{CU}X"
```

{CD} = 

{CU} = **SHIFT** 

Type RUN and hit RETURN. On the screen you should see:

```
X   X   X
  Y   Y
```

## 9 HOW CAN I USE THE QUESTION MARK (?) IN PLACE OF THE PRINT COMMAND?

In Commodore BASIC you can use a question mark in place of the word "PRINT." Here are some examples to try in DIRECT MODE. Type:

```
? 777,12
```

On the screen you should see:

```
777          12
```

Now type:

```
? "ABCD", "WXYZ"
```

On the screen you should see:

```
ABCD          WXYZ
```

You can also use the ? in PROGRAM MODE. Type NEW and enter this program:

```
10 ? "THIS WAY IS FASTER"
```

```
20 ? "ISN'T IT?"
```

Type RUN and hit RETURN. On the screen you should see:

```
THIS WAY IS FASTER
ISN'T IT?
```

Notice in line #20 that only the first question mark (the one not in quotes) causes the computer to print. If you have a question mark enclosed in quotes, the computer assumes that it is part of the string that is in quotes and that it does not mean the same thing as a PRINT command.

Using the ? instead of the word "PRINT" will save you time in entering a program. When you list the program, all question marks (?) not in quotes will be changed to the word "PRINT." Prove this to yourself by LISTing the program you just ran.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

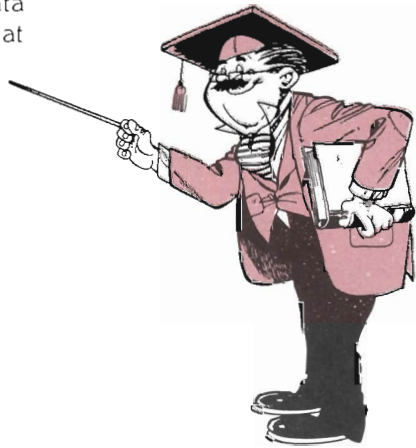
1. Whenever you see PRINT "{SC}" in a program, it stands for the instruction to clear the screen. Enter a clear-screen instruction as:

```
PRINT " SHIFT CLR/HOME "
```

When you LIST a program that contains PRINT "{SC}", it will appear on the screen as:

```
PRINT " ♥ "
```

2. To print in standard columns, use *commas* in your PRINT statement. To print data items near each other, use *semicolons* in your PRINT statement.
3. Use SPC(X) to skip X number of spaces between your data items. Use TAB(X) to print at certain tab positions.



## HOW TO ENTER DATA INTO A PROGRAM

In today's lesson you will learn:

- How to enter data into your program from the keyboard.
- How to read data using the READ command.
- How to use the GET command to find out which key on the keyboard is being pressed down.



**1****WHAT ARE THE DIFFERENT WAYS I CAN ENTER DATA INTO MY PROGRAM?**

There are several BASIC commands that you can use to enter data into your program. Here we will look at three of them: INPUT, READ, and GET.

INPUT is used to enter data from the keyboard. When the program comes to an INPUT command, it stops, prints a question mark on the screen, and waits for you to type in your input. After you type in your input and hit RETURN, the program continues. Use INPUT to enter small amounts of data from the keyboard.

READ is used to enter larger amounts of data. The READ command works together with the DATA command. You put your data in a DATA statement and READ assigns it to different variables.

GET is similar to INPUT because it enters data from the keyboard. But, it is different from INPUT because it doesn't stop and wait for you to type in data. Instead, it just reports which key, if any, is being pressed down at that moment.

**2****HOW CAN I USE THE INPUT COMMAND TO ENTER A SINGLE NUMBER INTO MY PROGRAM?**

Here is an example of the INPUT command. Type NEW and enter this program:

```
10 INPUT X
20 PRINT "YOU TYPED IN";X
```

Type RUN and hit RETURN.

When you run this program, a ? will appear on the screen. After you see the ?, type in the number 75 and hit RETURN. The computer will then print on the screen:

```
YOU TYPED IN 75
```

Here is how the program works. In line #10, the INPUT command tells the computer to do three things:

1. To print ? on the screen.
2. To stop the program from going any further until you type in a number and hit RETURN.
3. As soon as you hit RETURN, to make X equal to the number you typed in. Since X is a *numeric variable*, the computer

expects you to type in a number. (If you type in a letter or a string, you will get this error message: ? REDO FROM START)

In line #20 the PRINT command tells the computer to print on the screen the string "YOU TYPED IN" and, next to it, to print the value of X, which is 75.

For practice, RUN the program several times using different numbers.

## EXAMPLE

Suppose you want to INPUT a number, divide it by 2, and print the answer on the screen. Type NEW and enter the following program:

```
10 INPUT N
20 M=N/2
30 PRINT "HALF YOUR NUMBER IS";M
```

Type RUN and hit RETURN.

When you see the ?, type in 10 and hit RETURN. On the screen you should see:

```
HALF YOUR NUMBER IS 5
```

RUN the program several times using other numbers.

## 3 HOW CAN I USE THE INPUT COMMAND TO ENTER SEVERAL NUMBERS INTO MY PROGRAM?

The program that follows shows you how to enter several numbers into your program by using the INPUT command. Type NEW and enter:

```
10 INPUT A,B,C
20 PRINT "YOUR NUMBERS ARE",A;B;C
```

Type RUN and hit RETURN.

After you run this program, a ? will appear on the screen. When you see the ?, type in:

```
10,20,30 (hit RETURN)
```

The computer will then print on the screen:

```
YOUR NUMBERS ARE      10      20      30
```

Notice that when you type in several numbers next to the ?, you must type a comma between the numbers. You must also hit RETURN after your last number.

Here is how the program works. In line #10, the INPUT command tells the computer to do three things:

1. To print a ? on the screen.
2. To stop the program from going any further and to wait until you type in your numbers and hit RETURN.
3. To make A equal to the first number that you type in, B equal to the second number that you type in, and C equal to the third number that you type in.

Since A, B, and C are *three numeric variables*, the computer expects you to type in *three numbers*. If you type in fewer than three numbers, the computer will print ?? until you have typed in all three numbers. If you type in more than three numbers, the computer will accept the first three and ignore the extra numbers. RUN the program several times with different numbers.

Now try the following example:

#### EXAMPLE

Suppose you want to INPUT four numbers and print their SUM.

Type NEW and enter the following program:

```
10 INPUT W,X,Y,Z
20 T=W+X+Y+Z
30 PRINT "TOTAL=" ;T
```

Type RUN and hit RETURN.

When you see the ?, type in:

10,20,30,40 (hit RETURN)

On the screen you should see:

TOTAL=100

## **4** HOW CAN I USE THE INPUT COMMAND TO ENTER A SINGLE STRING INTO MY PROGRAM?

The INPUT command can be used to enter a single string, as shown in the following program. Type NEW and enter:



```
10 INPUT A$
20 PRINT "YOU TYPED IN ";A$
```

Type RUN and hit RETURN.

When you see the ?, type in the string LINDA and hit RETURN. The computer will print on the screen:

```
YOU TYPED IN LINDA
```

Notice that the string LINDA does not need to be in quotes when you are using the INPUT command. (A string must be in quotes, however, when you use it with the = sign, as in A\$ = "LINDA".)

Here is how the program works. In line #10, the INPUT command makes A\$ equal to the string LINDA. Since A\$ is a string variable, the INPUT command expects you to type in a string. In line #20 the PRINT command tells the computer to print on the screen the message "YOU TYPED IN" and, next to it, to print the value of A\$, which is equal to LINDA. RUN the program several times using different strings as inputs.

## **5 HOW CAN I USE THE INPUT COMMAND TO ENTER SEVERAL STRINGS INTO MY PROGRAM?**

Entering several strings is not much more difficult than entering a single string. Type NEW and enter this program:

```
10 INPUT A$,B$,C$,D$
20 PRINT
30 PRINT A$,B$,C$,D$
```

Type RUN and hit RETURN.

When the ? appears on the screen, type in these four strings:

```
BIKE, CAR, TRAIN, PLANE (hit RETURN)
```

Note that you must put a comma between each string so that the computer knows where one ends and where the next one begins.

This program works like the other programs which used INPUT. In line #10, the INPUT command tells the computer to expect four strings because A\$, B\$, C\$, and D\$ are string variables. When you type in BIKE, CAR, TRAIN, PLANE, the INPUT command makes:

```
A$=BIKE
B$=CAR
C$=TRAIN
```

D\$=PLANE

In line #20, the PRINT command (with nothing after it) prints a blank line. This is done to make the screen output look better. In line #30, the values of the four variables are printed on the screen.

## **6** WHEN I USE THE *INPUT* COMMAND, HOW CAN I GET THE COMPUTER TO REMIND ME WHAT DATA I NEED TO TYPE IN?

Let's take another look at the last program. Type NEW and enter:

```
10 INPUT A$,B$,C$,D$
20 PRINT
30 PRINT A$,B$,C$,D$
```

Type RUN and hit RETURN.

When you RUN this program and see the ?, you will need to type in four strings such as:

JOHN, HENRY, DORIS, NANCY

It would be helpful if the computer printed a reminder message, in addition to the ?, to tell you that you need to enter four names.

There are two different ways to get the computer to print this reminder message:

1. By using a PRINT command before the INPUT in line #10.
2. By putting the reminder message right into the INPUT command.

Try the following program to see how you can use the PRINT command to print a reminder message. Type NEW and enter:

```
5 PRINT "ENTER ANY 4 NAMES"
10 INPUT A$,B$,C$,D$
20 PRINT
30 PRINT A$,B$,C$,D$
```

Type RUN and hit RETURN.

Now enter and RUN this next program to see how you can put the reminder message right into the INPUT command. Type NEW and enter:

```
10 INPUT "ENTER ANY 4 NAMES";A$,B$,C$,D$
20 PRINT
30 PRINT A$,B$,C$,D$
```

Type RUN and hit RETURN.

Note: Another name for a reminder message is a *prompt*.

## 7

### HOW DOES THE READ COMMAND WORK?

The READ command is used to bring in data that are stored in DATA statements. The following is an example of a program that uses READ. Each READ command is used as part of a READ statement. Lines 10, 20, and 30 are READ statements. Type NEW and enter:

```
10 READ A
20 READ B
30 READ C
40 PRINT "A= ";A
50 PRINT "B= ";B
60 PRINT "C= ";C
70 DATA 23,-7,72
```

Type RUN and hit RETURN. On the screen you should see:

```
A=23
B=-7
C=72
```

Notice that A equals the first number in the DATA statement, B equals the second number, and C equals the third number. That is because the first READ brought in the 23 and assigned it to A. The second READ brought in the -7 and assigned it to B, and the third READ brought in the 72 and assigned it to C.

How does READ know which number to bring in next? It uses a *pointer*. At the beginning of the program, the pointer starts out pointing to the 23. After the first READ, the pointer moves to the -7. After the second READ, the pointer moves down to the 72, etc. Every time a new number is read in, the pointer moves over and points to the next number.

It doesn't matter to the pointer how the data are arranged in the DATA statements. All the data can be contained on one DATA statement, or the data can be spread out over several DATA statements.

Instead of line #70, we could have:

```
70 DATA 23
80 DATA -7
90 DATA 72
```

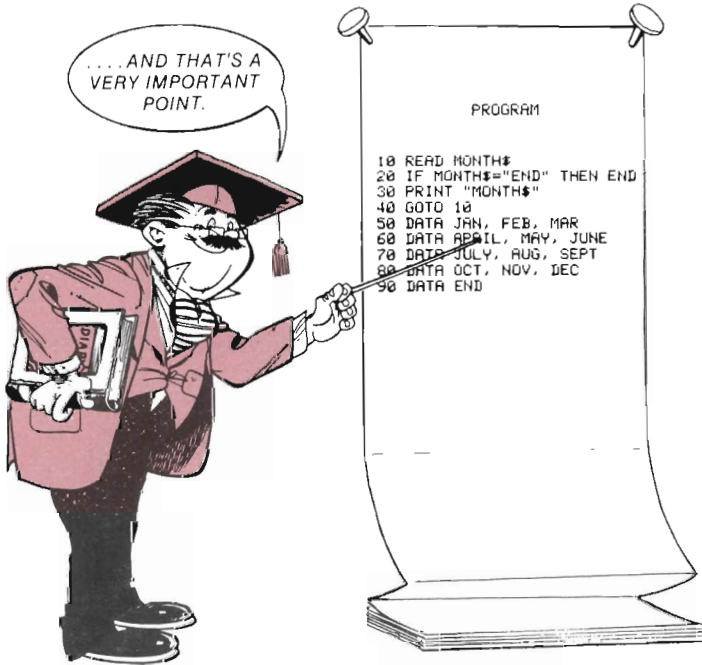
Or we could have:

```
70 DATA 23, -7
80 DATA 72
```

Or we could have:

```
70 DATA 23
80 DATA -7,72
```

The arrangement of the data doesn't matter to the pointer. It goes on its merry way and points to the next data item each time there is a READ, no matter how the DATA statements are arranged.



**A pointer keeps track of which item was read in last by any READ statement in the program.**

## 8

**CAN A READ STATEMENT BRING IN MORE THAN ONE DATA ITEM AT A TIME?**

Yes, a READ statement certainly can bring in more than one data item at a time.

Try this program. Type NEW and enter:

```
10 READ A$,B,C$
20 READ D,E$,F
30 PRINT F,E$,D,C$,B,A$
40 DATA ONE,2,THREE
50 DATA 4,FIVE,6
```

Type RUN and hit RETURN. The screen should show:

```
6   FIVE   4   THREE
2   ONE
```

Here are some important things to notice about this program:

1. We can read many data items with a single READ statement.
2. We can read in both numbers and strings with the same READ statement.
3. The first variable is A\$. Therefore, the first data item *must* be a string (ONE). The second variable is B. Therefore, the second data item *must* be a number (2), etc. In other words, the data in the DATA statement must match the variables that are in the READ statement.
4. If the READ statement contains three variables, then the pointer moves down three data items at a time. At the beginning of the program, the pointer points to ONE. After line #10 is completed, the pointer moves to the 4 in the second DATA statement.

Now try the following example:

Suppose you want to read in and print the name, age, and weight of each player on a basketball team. Type NEW and enter the following program:

```
10 READ N$,A,W:PRINT N$,A,W
20 READ N$,A,W:PRINT N$,A,W
30 READ N$,A,W:PRINT N$,A,W
40 READ N$,A,W:PRINT N$,A,W
50 READ N$,A,W:PRINT N$,A,W
60 DATA JOHN,18,160
70 DATA HARRY,19,162
80 DATA TED,17,158
90 DATA MARK,18,159,LARRY,17,161
```

Type RUN and hit RETURN.

Notice that in line #90, two players share the same DATA statement. This is perfectly all right. The pointer keeps track of what to read next, and it doesn't matter whether all the data are in one DATA statement line or in several data statements.

You can see in the example that 5 READ statements were used to read in the names of the 5 players. Actual programs are not written in this awkward and inefficient way. In actual programs, the same READ statement is used over and over again as part of a loop.

We will see how this works shortly, when we discuss loops in detail.

## **9 HOW CAN I USE THE RESTORE COMMAND TO RESET THE POINTER TO THE BEGINNING?**

Remember the pointer? Every time the computer reads in a data item from your DATA statement, the pointer moves down to the next data item. But, what if we want the computer to go back to the beginning and read the same data again? Will the data be lost forever? No, we have not burned our bridges! The RESTORE command can be used to move the pointer back to the first data item.

Try this example to see how it's done. Type NEW and enter:

```
10 READ M1$,M2$,M3$
20 PRINT M1$,M2$,M3$
30 RESTORE
40 READ M1$,M2$,M3$
50 PRINT M1$,M2$,M3$
60 DATA JAN,FEB,MAR
70 DATA APR,MAY,JUN
```

Type RUN and hit RETURN.

On the screen you should see:

```
JAN    FEB    MAR
JAN    FEB    MAR
```

The RESTORE command in line #30 returns the pointer to JAN, so that the next READ command in line #40 reads in JAN, FEB, MAR instead of APR, MAY, JUN.

If you delete line #30, there will be no RESTORE command and the READ command in line #40 will read in APR, MAY, JUNE instead.

**10****HOW CAN I USE THE GET COMMAND TO FIND OUT WHICH KEY IS BEING PRESSED DOWN?**

You can use the GET command to tell you which key is being pressed down on the keyboard. When you use it in a program, it looks like this:

```
10 GET A$
```

Whenever your program comes to a GET command, it looks to see which key is down. It then assigns the character for that key to A\$. For example: If Q is being pressed down when the program is at line #10, then A\$ = "Q". If no key is being pressed down, then A\$ has no value.

**EXAMPLE**

Suppose you want to press down different keys on the keyboard and have them print on the screen like a typewriter. Type NEW and enter the following program:

```
10 GET A$  
20 PRINT A$;  
30 GOTO 10
```

Type RUN and hit RETURN. Now type on the keyboard. The characters you typed will show on the screen.

In line #30, the GOTO command (which we will look at more closely later on) sends the program back to line #10.

To stop this program, hit RUN/STOP.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. Whenever you use the INPUT command, the data that you type on the keyboard must match the variables in the INPUT command. For example, if your statement looks like this:

```
INPUT N, A$, X
```

you must type a number first, a string second, and then another number.

Separate your inputs with a comma, like this:

```
7, GEORGE SMITH, 28
```

2. The data in the DATA statement must match the variables in the READ statement.

```
READ NAME$, CITY$, AGE  
DATA SAM JONES, NEW  
YORK, 48
```

In this example, NAME\$ AND CITY\$ expect strings, while AGE expects a number.

3. A *pointer* keeps track of the last item that was read by any READ statement in the program. You can set this pointer back to the beginning by using the RESTORE command.





## HOW TO CONTROL THE FLOW OF THE PROGRAM

In today's lesson you will learn:

- How to jump to a line number by using the GOTO and ON GOTO commands.
- How to call a subroutine by using the GOSUB and ON GOSUB commands.
- How to use the powerful IF command.



**1****WHAT IS PROGRAM FLOW AND HOW CAN I CONTROL IT?**

*Program flow* concerns the order in which program lines are performed (or executed) by the computer. Usually the computer executes the lines in order of their line number. It starts with the lowest line number and works on each line until it reaches the highest line number.

But, there are some BASIC commands that tell the computer not to follow the usual order. These commands tell the computer something like this: "Don't go to the next line number. Instead go to line #300." By using these kinds of commands, you can control the flow of your program.

Here is a list of BASIC commands that you can use to control program flow:

1. GOTO
2. ON GOTO
3. GOSUB
4. ON GOSUB
5. IF

**2****HOW CAN I USE THE GOTO COMMAND TO JUMP TO A LINE NUMBER IN MY PROGRAM?**

The GOTO command tells the program to jump to a certain line number. Here is an example of the GOTO command as it might appear in a program:

```
10 GOTO 300
```

After the program does line #10, it does not go on to the next line but jumps directly to line #300. By using the GOTO command, you have changed the normal flow of the program.

**EXAMPLE**

Suppose you want the computer to execute lines #10 and #20. After executing #20, you want the computer to jump to line #50. Type NEW and enter the following program:

```
10 PRINT "A"  
20 PRINT "B"  
30 GOTO 50  
40 PRINT "C"  
50 PRINT "D"
```

Type RUN and hit RETURN.

When you run this program, the screen will show:

```
A  
B  
D
```

The "C" does not print because line #40 has been skipped.

## EXAMPLE

Suppose you want to type a message on the keyboard and have the computer print it on the screen. You also want the computer to assign and print a message number with each such message. Type in NEW and enter the following program:

```
10 PRINT "ENTER MESSAGE"  
20 INPUT M$  
30 N=N+1  
40 PRINT:PRINT "MESSAGE # ";N  
50 PRINT "-----"  
60 PRINT M$:PRINT:PRINT  
70 GOTO 10:REM LOOP BACK TO #10
```

Type RUN and hit RETURN.

When you see the ?, type in your message and hit RETURN. Here are some messages to type:

```
I AM LEARNING BASIC  
COMPUTERS ARE FUN  
I LIKE THIS BOOK
```

Your message will be printed on the screen with a message number.

To stop the program, hold down RETURN and hit RUN/STOP *rapidly* several times. (When you are trying to stop a program that is busy looking for a keyboard input to an INPUT command, you must use this procedure. Pressing the RUN/STOP key by itself will not work well.)

Notice that in line #70, the GOTO command sends the

program back to line #10. This creates a *loop*, since the program goes around and around until you hit RUN/STOP.

In line #30, the value of N increases by 1, each time through the loop. This produces the message numbers, which are printed in line #40.

### 3

## HOW IS ON GOTO DIFFERENT FROM GOTO?

The GOTO command tells the program to jump to a line number. For instance, GOTO 225 tells the program to jump to line #225 and do that line next.

The ON GOTO command is similar. Its form is:

```
ON X GOTO 120,180,320
```

This means:

```
IF X=1 GOTO 120  
IF X=2 GOTO 180  
IF X=3 GOTO 320
```

In other words, the line number that the computer jumps to depends on the value of X. If X=0 or if X is greater than the number of choices following the ON GOTO command (three in this case), then the whole line is ignored, and the program goes to the next line number.

In the preceding example, there are three line numbers after the ON GOTO command. However, you can use more numbers or fewer, depending on your needs.

These forms of the ON GOTO command are also all right to use:

```
ON R GOTO 25,130  
ON AGE GOTO 100,200,300,500
```

## EXAMPLE

Suppose you want to input the number 1, 2, or 3. If the number is 1, then you want to print "ONE"; if 2, then print "TWO"; and if 3, then print "THREE." This printing is to be done in tab position 30. Type NEW and enter the following program:

```

10 INPUT "ENTER 1,2,OR 3";N
20 ON N GOTO 40,50,60
30 GOTO 10
40 PRINT TAB(30);"ONE":GOTO 10
50 PRINT TAB(30);"TWO":GOTO 10
60 PRINT TAB(30);"THREE":GOTO 10

```

Type RUN and hit RETURN.

Notice that if you input a number that is larger than 3, line #20 is ignored. The flow goes to line #30, which sends the program back to line #10.

To end the program, hold down RETURN and hit RUN/STOP *rapidly* several times.

## 4 WHAT IS A SUBROUTINE, AND HOW DO I USE THE GOSUB COMMAND?

A *subroutine* is a special part of a program, one that is not in the regular program flow. When you need to use a subroutine, you can call it into action by using the GOSUB command.

Here is an example. Type NEW and enter the following program:

```

10 REM MAIN PROGRAM
20 PRINT:PRINT "GEORGE"
30 GOSUB 500:REM CALL SUBROUTINE TO UNDERLINE
40 END
500 REM SUBROUTINE TO UNDERLINE
510 PRINT "-----"
520 RETURN

```

Type RUN and hit RETURN.

There are several things about this example that you should know:

1. The program has two parts: a **main program** (lines 10–40) that prints the name GEORGE and calls the subroutine to underline; and a **subroutine** (lines 500–520) that underlines.
2. A REM command identifies the main program and the subroutine. The REM also tells what the subroutine does. Remember, there may be several subroutines in the program, and it is important to know the purpose of each.
3. The subroutine has a RETURN command in line #520. There must be a RETURN command in all subroutines. The RETURN command tells the program to go to the line number that *follows* the GOSUB command which called it. Since the GOSUB is in

line #30, the RETURN will send the program to line #40 *after* the subroutine is done.

## EXAMPLE

Suppose you want to print three names and underline each name after you print it. Type NEW and enter the following program:

```
5 PRINT "{SC}"
10 PRINT "GEORGE"
20 GOSUB 500
30 PRINT "HARRY"
40 GOSUB 500
50 PRINT "SAM"
60 GOSUB 500
70 END
500 REM SUBROUTINE TO UNDERLINE
510 PRINT "-----"
520 PRINT:PRINT
530 RETURN
```

Type RUN and hit RETURN.

In this example, we wanted to underline several times and to do so at different points in the program. A subroutine allows us to do this quickly and efficiently. Every time we need to underline, we just say GOSUB 500.

## 5 HOW IS ON GOSUB DIFFERENT FROM GOSUB?

The GOSUB tells the program to jump to a subroutine starting at some line number. For instance, GOSUB 800 tells the program to jump to a subroutine that starts at line #800.

The ON GOSUB command is similar. Its form is:

```
ON X GOSUB 700,800,900,1000
```

This means:

- IF X=1, then jump to subroutine at line #700.
- IF X=2, then jump to subroutine at line #800.
- IF X=3, then jump to subroutine at line #900.
- IF X=4, then jump to subroutine at line #1000.

In other words, the line number of the subroutine that the program jumps to depends on the value of X. This is very similar to the way the ON GOTO command works.

If X=0 or if X is greater than the number choices following the ON GOSUB command (four in this case), then the whole line is ignored and the program goes to the next line number.

## EXAMPLE

Suppose you want your program to convert yards to feet or convert yards to inches. You want to display these choices. You then want to enter the choice and the number of yards you wish to convert. Type NEW and enter the following program:

```
10 PRINT "{SC}"
20 PRINT "1-YARDS TO FEET"
30 PRINT "2-YARDS TO INCHES"
40 PRINT:PRINT:PRINT
50 PRINT "ENTER CHOICE #,YARDS"
60 INPUT CHOICE,Y
70 PRINT "{SC}"
80 ON CHOICE GOSUB 100,200
90 PRINT:PRINT:PRINT:GOTO 20
100 REM YARDS TO FEET
110 F=Y*3
120 PRINT:PRINT Y;"YARDS=";F;"FEET"
130 RETURN
200 REM YARDS TO INCHES
210 I=Y*36
220 PRINT:PRINT Y;"YARDS =" ;I;"INCHES"
230 RETURN
```

Type RUN and hit RETURN.

## 6 WHAT DOES THE IF COMMAND DO AND HOW DOES IT WORK?

The IF command is one of the most powerful and useful BASIC commands. It has two parts:

**Part 1** asks a question.

**Part 2** tells the computer what to do if the answer to the question is TRUE. (If the answer to the question is NOT TRUE, then the rest of the line is ignored, and the program flow goes to the next line number.)

Let's look more closely at an example of the IF command:

```
IF X=5 THEN PRINT "FIVE"
```

Part 1                      Part 2

Notice that Part 1 starts with IF and asks a question. Part 2 starts with THEN and tells the computer what to do. IF and THEN are always used together.

Part 1 asks: "Is it TRUE X = 5?" Part 2 says: "If it is TRUE, THEN print the string "FIVE".

In Part 1, you ask a question by comparing two numbers or two strings. Here are the kinds of comparisons you can make and the symbol for each:

<b>Comparison</b>	<b>Symbol</b>
Equal	=
Less than	<
Greater than	>
Less than or equal	<=
Greater than or equal	>=
Not equal	< >

In Part 2, you tell the computer what to do if the answer to the question in Part 1 is TRUE. The BASIC commands that are usually used in Part 2 are:

1. PRINT
2. =
3. GOTO
4. GOSUB
5. POKE
6. END

Here are some examples of the IF command as it might be used in a program:

1. IF N=5 THEN PRINT Q\$
2. IF J < 7 THEN R=53.2
3. IF J > 23 THEN GOTO 150 } either way is correct\*
- or,
- IF J > 23 THEN 150 }
4. IF NAME\$ <= "E" THEN GOSUB 700
5. IF X\$ >= Y\$ THEN GOSUB 550
6. IF SEX\$ <> "MALE" THEN END

\* Notice that GOTO is optional and can be omitted. You can simply put the line number right after THEN. For example, both of the following statements are correctly written, and both do the same thing:



```
IF X=5 THEN GOTO 300
IF X=5 THEN 300
```

## 7 HOW CAN I USE THE IF COMMAND TO COMPARE NUMBERS?

The IF command can be used to compare two numbers to see if they are equal or if one is greater than the other. Here are some examples: (Don't enter these.)

To see if X equals 5, use:

```
IF X=5 THEN PRINT "FIVE "
```

To see if A is less than B, use:

```
IF A < B THEN GOTO 120
```

To see if SCORE is greater than 100, use:

```
IF SCORE > 100 THEN WINS=WINS+1
```

To see if AGE is less than or equal to 21, use:

```
IF AGE <= 21 THEN GOSUB 200
```

To see if WEIGHT is greater than or equal to 2000, use:

```
IF WEIGHT >= 2000 THEN POKE 1020,7
```

To see if PAY is not equal to zero, use:

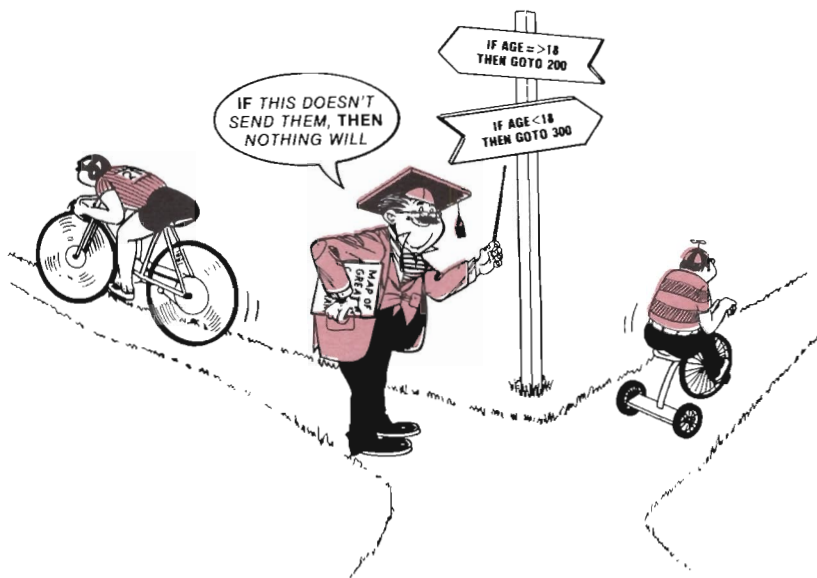
```
IF PAY < >0 THEN END
```

### EXAMPLE

Suppose you want the computer to compare your age with the ages of your friends, and to tell you who is older or younger. Type NEW and enter the following program.

```
10 PRINT "(SC)"
20 INPUT "ENTER YOUR AGE";Y
30 INPUT "ENTER A FRIEND'S NAME AND AGE";N$,A
40 IF A<0 THEN END
50 IF A=Y THEN PRINT "YOU ARE THE SAME AGE"
60 IF A<Y THEN PRINT N$;" IS YOUNGER"
70 IF A>Y THEN PRINT N$;" IS OLDER"
80 PRINT "-----"
90 PRINT:PRINT
100 GOTO 30
```

Type RUN and hit RETURN. To stop the program, enter a minus age for your friend.



## 8

### HOW DOES THE IF COMMAND WORK WHEN THERE IS MORE THAN ONE STATEMENT ON THE SAME LINE?

Take a look at this short program. Type NEW and enter:

```
10 INPUT X
20 IF X=5 THEN PRINT "FIVE":PRINT "SIX":PRINT "SEVEN":
   END
30 PRINT "NEXT LINE"
```

Type RUN and hit RETURN.

You might ask this question: "If X does NOT equal 5, I know that the string "FIVE" will not be printed. But, will the strings "SIX" and "SEVEN" be printed or will the program go to line #30?"

The answer is: "If X does NOT equal 5, the program will go right to line #30. Everything else on line #20 will be ignored."

#### EXAMPLE

Suppose you want to enter a number from 1 to 10. If the number you enter is 5, then you want to print this message:

```
    ** RIGHT **  
    YOU GUESSED IT  
    NICE GOING
```

If the number you enter is greater than 10, then you want the program to end.

Type NEW and enter the following program:

```
10 PRINT "<SC>"  
20 INPUT "ENTER ANY NUMBER 1-10";X  
30 IF X>10 THEN PRINT "GAME IS OVER";END  
40 IF X=5 THEN PRINT "** RIGHT **":PRINT "YOU GUESSED  
    IT":PRINT "NICE GOING"  
50 PRINT:GOTO 20
```

Type RUN and hit RETURN.

Notice that if  $X = 5$ , then all three statements on line #40 are executed. But, if  $X$  does not equal 5, then *none* of the statements on line #40 are executed and the program proceeds to line #50.

## 9 HOW CAN I USE THE IF COMMAND TO COMPARE STRINGS?

The IF command can be used to compare strings. These strings can be made up of numerals, letters, special characters such as !, \*, and #, or even graphics symbols such as ♥ or ♦. Each of these characters is represented in the computer by a number code. This number code, called ASCII, is a standard used by most computer manufacturers.

For example, the ASCII number code for the letter "A" is 65, and for the letter "B," 66. Therefore, we can say that A is less than B because the ASCII value 65 is less than the ASCII value 66. Of course, strings that are exactly the same will have the same ASCII value and will be equal.

Please look at Appendix A. You will see all the numerals, letters, symbols, and graphics characters that are used by the Commodore 64. Each of these has an ASCII value.

When we compare strings, we are actually comparing their ASCII values. The following example will show you how this works.

### EXAMPLE

Suppose you want to compare the numeral "1" with the letter "A":

IF "1" < "A" THEN PRINT "FIRST IS LOWER"

Since the ASCII value of "1" is 49 and the ASCII value of "A" is 65, the comparison is TRUE, and the computer will print "FIRST IS LOWER."

## EXAMPLE

Suppose you want to be able to compare any two characters and to print whether the first is lower or higher than the second. Type NEW and enter the following program:

```
5 PRINT "(SC)"
10 PRINT "ENTER ANY 2 CHARACTERS (EXAMPLE A,B)"
20 INPUT A$,B$:IF A$="END" THEN END
30 IF A$<B$ THEN PRINT "FIRST IS LOWER"
40 IF A$>B$ THEN PRINT "FIRST IS HIGHER"
50 IF A$=B$ THEN PRINT "THEY ARE SAME"
60 PRINT "-----":PRINT
70 GOTO 10
```

Type RUN and hit RETURN. Now compare the characters and strings shown in the first column below. Your results should match the ones in the second column.

<b>Compare</b>	<b>The First Is</b>
A,B	LOWER
R,D	HIGHER
4, 2	HIGHER
!, #	LOWER
(, %	HIGHER
AB,A	HIGHER
AB,ABC	LOWER
AB,AB	SAME
♥,♦	LOWER
π,↑	HIGHER
π,ππ	LOWER

To end the program, type END,A next to the ?. The program will then end in line #20.

## 10 HOW CAN I USE LOGICAL OPERATORS (AND, OR) WITH THE IF COMMAND?

Both AND and OR are called logical operators, and both can be used with the IF command.

Our first example of the IF command was:

```
IF X=5 THEN PRINT "FIVE"
```

When the computer executes this statement, it checks to see if the value of X equals 5. If it *does equal* 5, then the computer will print the string "FIVE".

The AND operator lets us write this statement as:

```
IF X=5 AND Y=5 THEN PRINT "FIVE"
```

Here the computer checks to see if both X = 5 AND Y = 5. If *both* are equal to 5, then it prints the string "FIVE".

The OR operator works similarly. It lets us write this statement as:

```
IF X=5 OR Y=5 THEN PRINT "FIVE"
```

In this case, either X must equal 5 or Y must equal 5 in order for the computer to print the string "FIVE".

## EXAMPLE

Suppose you want to enter the ages of two people. If one of the people is under the age of 10, then you want the computer to print "AT LEAST ONE IS A KID." If *both* are at least 18 years old, then you want the computer to print "BOTH CAN DRIVE." Type NEW and enter the following program:

```
10 PRINT "(SC)"
20 INPUT "ENTER TWO AGES-EXAMPLE 18,21";A,B
25 PRINT
30 IF A<0 AND B<0 THEN END
40 IF A<10 OR B< 10 THEN PRINT "AT LEAST ONE IS A KID"
50 IF A>=18 AND B>=18 THEN PRINT "BOTH CAN DRIVE"
60 PRINT:PRINT "LET'S TRY AGAIN"
70 GOTO 20
```

Type RUN and hit RETURN. When you run this program, enter two ages every time you see the ?.

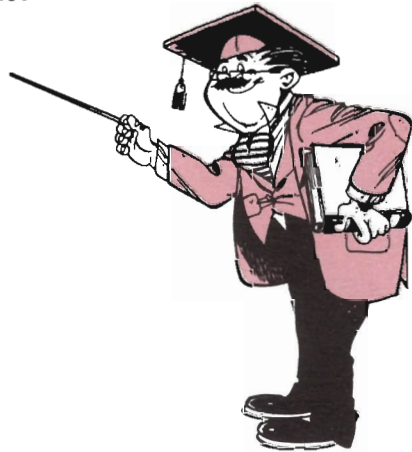
Try these ages and some of your own:

```
5,10
15,18
18,21
```

To end the program, enter both ages as negative numbers.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. Use a subroutine if you need to repeat the same procedure at different points in the program.
2. A RETURN command is required in every subroutine. The RETURN command tells the program to go back to the line number that follows the GOSUB command which called the subroutine.
3. The IF command has two parts: **Part 1** compares two numbers or two strings to see if one is smaller than, greater than, or equal to the other. **Part 2** tells the computer what to do if the answer to the comparison in Part 1 is TRUE. If there are other statements on the same line following Part 2, all are ignored if the comparison made in Part 1 is not TRUE.



## Day 7

# USING LOOPS

In today's lesson you will learn:

- How to create loops using GOTO and FOR-NEXT.
- How to use a GOTO loop to read data.
- How to create a NESTED LOOP.



# 1

## WHAT IS A LOOP AND HOW DO I USE IT IN MY PROGRAM?

You have already seen examples of loops in some of the earlier lessons. Now, let's take a closer look at them.

A *loop* is a part of a program that repeats itself many times. The following programs all contain examples of loops.

To create a loop that goes on and on without stopping, type NEW and enter the following program:

```
10 PRINT "THIS IS A LOOP THAT GOES ON AND ON"  
20 GOTO 10
```

Type RUN and hit RETURN. In line #20, the GOTO command causes the program to loop back to line #10. The program continues returning to line #10 until you either press RUN/STOP or turn off the computer.

To create a loop that repeats itself five times and then stops, type NEW and enter the following program:

```
5 PRINT "(SC)"  
10 N=N+1  
20 PRINT N,"THIS IS A GOTO LOOP"  
30 IF N=5 THEN END  
40 GOTO 10
```

Type RUN and hit RETURN. In line #10, N increases by 1 each time. It counts how many times we have gone around the loop. In line #20, the program prints N and then prints the string "THIS IS A GOTO LOOP". In line #30, the program ends if N = 5. In line #40, the program loops back to line #10.

To create a loop that also repeats itself five times but uses the FOR-NEXT command instead of the GOTO command, type NEW and enter this program:

```
10 FOR N=1 TO 5  
20 PRINT N,"THIS IS A FOR-NEXT LOOP"  
30 NEXT N
```

Type RUN and hit RETURN. Notice that when you use a FOR-NEXT loop, you don't have to count the number of times the loop has gone around. The FOR-NEXT command does it for you.



## 2

### HOW DO I CREATE A LOOP TO READ DATA?

One of the most important uses of loops is to read data using the READ command. Here is an example. Type NEW and enter this program:

```
10 READ A:IF A=-9999 THEN END
20 PRINT A
30 GOTO 10
40 DATA 5,15,25,35,45,55,-9999
```

Type RUN and hit RETURN.

In line #10, the computer reads the data in the DATA statement and assigns each value, in turn, to the variable A. Then the computer tests A to see if A is equal to -9999. If A is equal to -9999, then it knows that the last number has been read and it ENDS the program. In line #20, the computer prints A, which is the number read in line #10. In line #30, the program loops back to line #10. In line #40 are kept the data to be read in. The only purpose of the -9999 is to tell the computer when the last number has been read. The number -9999 was chosen because it is so far away from the other numbers that it cannot be mistaken for data.

#### EXAMPLE

Suppose you want to read a list of people's names and ages and then you want to print out the names of all who can vote. (A person can vote if he or she is 18 years or older.)

Type NEW and enter the following program:

```
10 PRINT "(SC)"
20 PRINT:PRINT "PEOPLE WHO CAN VOTE"
30 PRINT "*****"
40 READ N$,A
50 IF A<0 THEN END
60 IF A>18 THEN PRINT N$
70 GOTO 40
500 DATA JOHN SMITH,17
510 DATA PHIL BLANK,20
520 DATA PAT ANDERSON,35
530 DATA ELANOR MILLER,16
540 DATA JUDY STAR,12
550 DATA END,-1
```

Type RUN and hit RETURN.

After you run the program, the screen should show the names of the two people who can vote: Phil Blank and Pat Anderson. Notice how the GOTO command in line #50 creates a loop that keeps reading names until a minus age is found.

### **3 HOW DOES THE FOR-NEXT LOOP WORK AND WHEN SHOULD I USE IT?**

You have just seen an example of how a loop is created using GOTO. The GOTO loop should be used whenever you don't know how many times you want the loop to repeat itself. In a previous example, the loop was repeated until the last data item was found. (The last data item was -9999.)

But, suppose you know exactly how many times you want the loop to repeat itself. In this case, the FOR-NEXT loop is usually best and easiest to use. To create a FOR-NEXT loop that will repeat itself three times, type NEW and enter the following program:

```
10 FOR N=1 TO 3
20 PRINT "PASS NUMBER ";N
30 NEXT N
```

Type RUN and hit RETURN.  
The screen should show:

```
PASS NUMBER 1
PASS NUMBER 2
PASS NUMBER 3
```

Try changing line #10 to :

```
10 FOR N=1 TO 5
```

Now, RUN the program again. On the screen you should see:

```
PASS NUMBER 1
PASS NUMBER 2
PASS NUMBER 3
PASS NUMBER 4
PASS NUMBER 5
```

Let's take a closer look at the FOR-NEXT loop. LIST the program you now have in memory. It should look like this:

```
10 FOR N=1 TO 5
20 PRINT "PASS NUMBER ";N
30 NEXT N
```

Type RUN and hit RETURN.

Line #10 says to start at  $N = 1$  and increase  $N$  by 1 each time until  $N = 5$ . Line #20 says to print "PASS NUMBER" and the value of  $N$  at that particular time. Line #30 says to check to see if  $N$  is less than 5. If it is less than 5, then send the program back to line #10. If  $N = 5$ , then go to the next line number.

A FOR-NEXT loop can have many statements. The first statement *always* contains the FOR command, and the last statement always contains the NEXT command.

#### EXAMPLE

Suppose you want to multiply the number 3 by all the numbers from 5 to 10 and you want to print the result. Type NEW and enter the following program:

```
10 FOR M=5 TO 10
20 PRINT 3*M
30 NEXT M
```

Type RUN and hit RETURN.

#### **4** IN THE FOR-NEXT LOOP, HOW CAN I INCREASE THE COUNT BY MORE THAN 1, WITH EACH PASS?

By adding the command STEP in the FOR-NEXT loop, you can increase the count by more than one with each pass.

To create a loop that increases from 0 to 20 by twos, type NEW and enter the following program:

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

Type RUN and hit RETURN. The screen should show all even numbers between 0 and 20.

#### EXAMPLE

Suppose you want to print every fifth number from 50 to

500, and you want to print all the numbers near each other (not in standard columns). Type NEW and enter the following program:

```
10 FOR NUM=50 TO 500 STEP 5
20 PRINT NUM;
30 NEXT NUM
```

Type RUN and hit RETURN.

## **5** IN A FOR-NEXT LOOP, CAN I MAKE THE NUMBERS DECREASE AS WELL AS INCREASE?

Yes! In a FOR-NEXT loop, you can get the numbers to start high and go lower.

To go from 30 to -10 in steps of -5, type NEW and enter the following program:

```
10 FOR X=30 TO -10 STEP -5
20 PRINT X
30 NEXT X
```

Type RUN and hit RETURN.

### EXAMPLE

Suppose you want to print all the temperatures from 100 down to -20 in steps of -10 degrees. You want to underline all temperatures below zero. Type NEW and enter the following program:

```
10 FOR TEMP=100 TO -20 STEP -10
20 PRINT TEMP;"DEGREES"
30 IF TEMP<0 THEN PRINT "-----"
40 NEXT TEMP
```

Type RUN and hit RETURN.

## **6** IN A FOR-NEXT LOOP, CAN I USE VARIABLES INSTEAD OF NUMBERS IN THE FOR STATEMENT?

Yes. Variables are often used in the FOR statement. Here is the same example as the one in Question #5, except that

the example will use variables instead of numbers. To go from 30 to -10 in steps of -5 (using variables for 30, -10 and -5), type NEW and enter the following program:

```
10 A=30:B=-10;C=-5
20 FOR X=A TO B STEP C
30 PRINT X
40 NEXT X
```

Type RUN and hit RETURN.

## EXAMPLE

Suppose you want to input a high temperature, a low temperature, and a step temperature. You then want to print out all temperatures from high to low in steps. Type NEW and enter this program:

```
10 PRINT "{SC}"
20 INPUT "ENTER HI,LO,STEP";H,L,S
30 IF H=-9999 THEN END
40 PRINT:PRINT:PRINT "** TEMPERATURE SCALE **":PRINT
50 FOR T=H TO L STEP S
60 PRINT T
70 NEXT T
80 INPUT "HIT RETURN TO CONTINUE";R$
90 GOTO 10
```

Type RUN and hit RETURN. Try this program with a different HI, LO, STEP each time. Enter the STEP temperature as a minus number.

Notice that in line #80, the INPUT command is used to hold the printing on the screen. If you hit RETURN, the program will go to line #90, which will loop back to line #10. In line #10, the screen will clear and the program will start a new loop. To end the program, enter a high temperature of -9999 (a very unlikely number for a temperature).

## 7

### CAN I HAVE A LOOP INSIDE A LOOP?

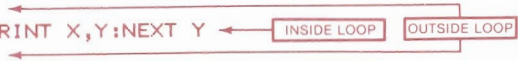
Yes. A loop inside a loop is called a *nested* loop. (It probably should have been called a *nasty* loop because it can be confusing to understand.) Anyway, nested loops can be very useful to you, so let's try to learn how they work.

To create a nested loop, type NEW and enter the following program:

```

5 PRINT "{SC}"
10 PRINT "X", "Y"
20 FOR X=1 TO 2
30 FOR Y=1 TO 3:PRINT X,Y:NEXT Y
40 NEXT X
50 END

```



Type RUN and hit RETURN.

When you run this program, the screen should show:

X	Y
1	1
1	2
1	3
2	1
2	2
2	3

Line #10 prints the “X” and “Y” column headings. Line #20 is the FOR statement for the outside loop. It begins by making X = 1.

Line #30 has three statements in it. They are: FOR Y = 1 to 3, PRINT X,Y and NEXT Y. These make up the inside loop. Therefore, with the outside loop holding at X = 1, the inside loop make Y = 1, Y = 2, and Y = 3. On each pass, the values of X and Y are printed by the PRINT X,Y statement.

When the inside loop is finished, the program goes to line #40, which now makes X = 2 and sends the program back to line #20. The entire procedure is then repeated for X = 2.

Since the outside loop only calls for X = 1 and X = 2, the program ends in line #50 after the X = 2 pass.

The same program can be written another way. Type NEW and enter this program:

```

5 PRINT "{SC}"
10 PRINT "X", "Y"
20 FOR X=1 TO 2
30 FOR Y=1 TO 3
40 PRINT X,Y
50 NEXT Y
60 NEXT X
70 END

```



Type RUN and hit RETURN.

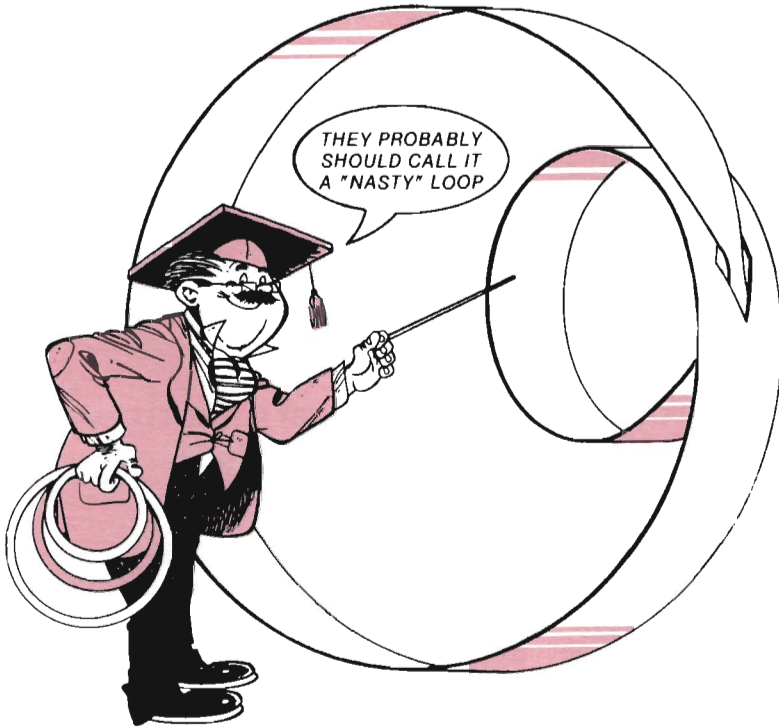
### EXAMPLE

There are seven days in a week. Suppose that you want to print the week number and day number for the first three

weeks of the month. Type NEW and enter the following program:

```
10 PRINT "(SC)"
20 FOR WEEK=1 TO 3
30 FOR DAY=1 TO 7
40 PRINT "WEEK#=" ; WEEK, "DAY#=" ; DAY
50 NEXT DAY
60 NEXT WEEK
```

Type RUN and hit RETURN.



**A nested loop is a loop inside a loop.**

**8**

## **DO I HAVE TO NAME THE VARIABLE EACH TIME I USE THE NEXT COMMAND?**

No. It's not necessary to name the variable each time, but it's a good idea. Here are two different ways to write the same loop.

### Method 1

```
10 FOR K=1 TO 5  
20 PRINT K  
30 NEXT K
```

### Method 2

```
10 FOR K=1 TO 5  
20 PRINT K  
30 NEXT
```

Notice line #30 in Method 2. The variable K is not used with the NEXT command.

In Commodore BASIC, Method 2 is all right to use. However, in many other computers, you must write the program *with* the variable included in the NEXT command, as shown in Method 1.

Method 2 is sometimes used when you have a nested loop (a loop inside a loop). Here is an example.

Type NEW and enter the following program:

```
10 FOR X=1 TO 2  
20 FOR Y=1 TO 3  
30 PRINT X,Y  
40 NEXT: NEXT
```

Type RUN and hit RETURN.

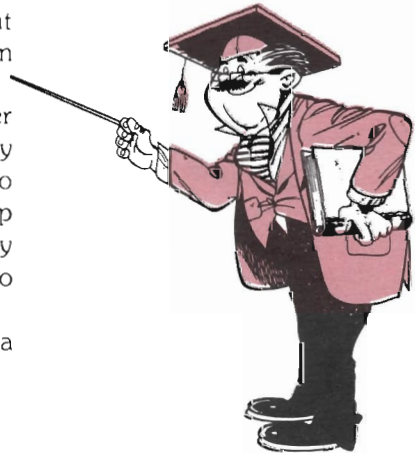
A better way to write line #40 would be:

```
40 NEXT X: NEXT Y
```

However, in Commodore BASIC, either way will work.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. A *loop* is a part of a program that repeats itself many times. One of the most important uses of a loop is to read in data.
2. Use a GOTO loop whenever you don't know how many times you want the loop to repeat. Use a FOR-NEXT loop if you know exactly how many times you want the loop to repeat.
3. A loop inside a loop is called a *nested loop*.

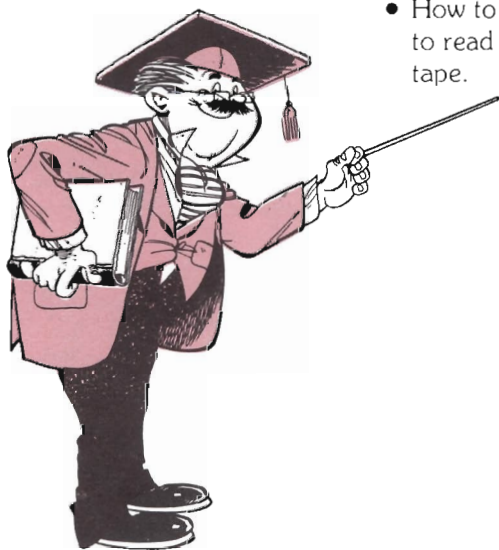




## SAVING PROGRAMS AND DATA ON CASSETTE TAPE

In today's lesson you will learn:

- How to save programs on tape, make sure that they have been recorded properly, and then load them back into your computer.
- How to record data files on tape.
- How to program your computer to read data files stored on tape.



## **1** WHAT KINDS OF DEVICES CAN BE HOOKED UP TO THE COMMODORE 64?

Many different types of devices can be hooked up to the Commodore 64. The most common ones are shown here, along with their device number and use.

<b>Device</b>	<b>Device No.</b>	<b>Use</b>
CASSETTE	1	Store programs and data
MODEM	2	Permit telephone communications
SCREEN	3	Display programs and data
PRINTER	4-5	Print programs and data
DISK DRIVE	8-11	Store programs and data

The *device number* is used in your program to tell the computer which device you want to call into action.

Today's lesson covers Device 1, the cassette recorder. Your cassette recorder can be used to:

1. Store *programs*.
2. Store *data* (such as a list of friends' names and phone numbers) that can be read by a program.

Programs and data are recorded on cassette tape in units called *files*. Each program is a separate file. A collection of related data is also a file. The following list shows the kinds of files that can be recorded on cassette tape.

### **Examples Of Files**

1. A program that lets you play a space arcade game.
2. A program to teach math.
3. Data containing a list of friends' names and phone numbers.
4. Data containing a list of baseball players and their batting averages.

The cassette recorder most commonly used with the Commodore 64 is the C2N Cassette Unit from Commodore. For that reason, our discussion will center around this model.

## **2** HOW CAN I SAVE MY PROGRAMS ON CASSETTE TAPE?

Saving programs on the Commodore C2N Cassette Unit is very easy. First, let's make up a test program that we want to save. Type NEW and enter the following program:

```
10 REM MY FIRST CASSETTE PROGRAM
20 PRINT "GRAPE"
30 PRINT "PLUM"
40 PRINT "LEMON"
50 END
```

LIST the program and RUN it to make sure that it works as it should.

Now put a blank tape into your cassette recorder. Rewind the tape to the beginning. Press the counter button on the tape counter so that three zeros appear in the counter window. You have now *zeroed* the tape counter.

To save your program under the file name "MYFIRST," do the following:

1. Enter SAVE "MYFIRST" and hit RETURN. The computer will print on the screen:

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

2. On the cassette recorder, press down the RECORD button and the PLAY button at the same time until they both stay down. The screen will go blank for a short time and, then, will return with the message:

```
OK
SAVING MYFIRST
READY.
```

3. On the cassette recorder, press the STOP key.

Your program should now be saved on tape. The next step is to VERIFY it, which means to check that the program on tape matches the program in memory that you are trying to save.

### **3 HOW DO I VERIFY MY PROGRAM TO MAKE SURE IT IS RECORDED PROPERLY?**

Is your program recorded properly on the tape? VERIFY will help you find out.

To verify the program that you just entered:

1. Rewind the tape to the beginning.
2. Enter VERIFY "MYFIRST" and hit RETURN. The computer will print on the screen:

```
PRESS PLAY ON TAPE
```

3. On the cassette recorder, press the PLAY button. The screen will go blank for a short time and then return with the message:

```
OK
SEARCHING FOR MYFIRST
FOUND MYFIRST
```

4. Hit the **C** key one time. (This key is called the Commodore key.) The screen will go blank for a short time and then return with the message:

```
VERIFYING
OK
READY.
```

This means that the program on the tape matches the program in memory, and the recording is good.

5. On the cassette recorder, hit the STOP button. If the program on tape does not match the program in memory, you will see on the screen:

```
? VERIFY ERROR
READY.
```

This means that something is wrong and that you should try to repeat the recording.

## **4 HOW CAN I LOAD MY PROGRAM FROM CASSETTE TAPE?**

You have now saved your program under the file name "MYFIRST" and used VERIFY to make sure that the recording is good.

Suppose that at a later time you want to find your program on the tape and load it into memory so that you can run it. First, find the cassette tape with your program on it. Then rewind the tape to the beginning. Zero the tape counter by pressing the button to the right of the counter. You are now ready to load the program.

To LOAD the program "MYFIRST":

1. Type LOAD "MYFIRST" and hit RETURN. The computer will print on the screen:

```
PRESS PLAY ON TAPE
```

2. On the cassette recorder, press down the PLAY button. The computer will print on the screen:

OK  
SEARCHING FOR MYFIRST  
FOUND MYFIRST

3. Press **C** key (Commodore key). The screen will go blank for a short time and will return with the message:

LOADING  
READY.

Your program has now been loaded from tape into the computer's memory. On the cassette recorder, press the STOP key. You are now ready to LIST or RUN your program.

## **5 DO I HAVE TO ASSIGN A NAME TO EACH OF MY PROGRAM FILES?**

You *do not* have to assign a name to each program file. However, a file name makes it easier to identify and find your file on the tape.

In the previous examples, we saved our program under the name "MYFIRST" by using the command SAVE "MYFIRST". Later we loaded in the program by using the command LOAD "MYFIRST".

If for some reason you do not want to assign a name to a file, just use the command:

SAVE

This will record the program with no file name.

To load in the program, use the counter to position the tape right in front of your file and then enter the command:

LOAD

Follow the screen prompts and the next program on the tape will be loaded in.

## **6 HOW DO I RECORD DATA ON CASSETTE TAPE?**

To record a file of data on cassette tape, you need to learn three BASIC commands: OPEN, PRINT#, and CLOSE.

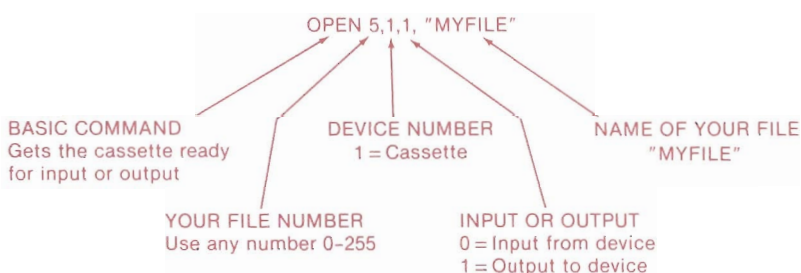
OPEN gets the cassette tape ready for either input or output. (You must tell it which.)

PRINT# is similar to a PRINT command, except that it writes the data on the cassette tape instead of the screen.

CLOSE tells the computer that you are finished recording your file. If you don't use the CLOSE command, some of the data that you want to record may be lost.

Let's take a closer look at each one of these commands.

To open a file on which you want to record data, use:



With this OPEN command you say to the computer: "The file in my program, which I call file #5, is on cassette tape. I want to record on this file (output to it)."

Here is an example of a BASIC command that tells the computer to *print*, or write, on the file:

```
PRINT#5, N$; ", " ; PH$
```

The PRINT#5 command tells the computer: "Record this data on my file #5. The data to be recorded are N\$, a comma, and PH\$." In this example, N\$ stands for a person's name and PH\$ stands for a phone number. A comma is to be recorded between them so that the recording on tape will look like this:

```
JOHN,257-3075
```

The computer knows that your file #5 is to be recorded on cassette tape because of the OPEN 5,1,1,"MYFILE" command. The first 1 tells the computer to record on device number 1 (the cassette recorder).

To close file #5, use:

```
CLOSE 5
```

The CLOSE command tells the computer: "CLOSE my file #5. I am finished with it. Clear out any leftover data." (If you don't use the CLOSE command, some of your data may be lost.)

## EXAMPLE

Suppose you want to record a file of names and phone

numbers on cassette tape. Type NEW and enter the following program:

```
10 REM OUTPUT TO TAPE
20 OPEN 5,1,1,"PHONE"
30 PRINT
40 INPUT "ENTER NAME,PHONE #";N$,PH$
50 IF N$="END" GOTO 999
60 PRINT#5,N$;" ";PH$
70 GOTO 30
999 CLOSE 5:END
```

Type RUN and hit RETURN.

The following message will appear on the screen:

PRESS RECORD & PLAY ON TAPE

When you press these two buttons, the screen will go blank for a short period of time and then this message will appear on the screen:

```
OK
ENTER NAME , PHONE# ?
```

After the ? mark, type in these names and phone numbers:

```
GEORGE ,917-2432
HARRY ,214-2446
JIM ,483-8104
STEVE ,761-1122
END ,0
```

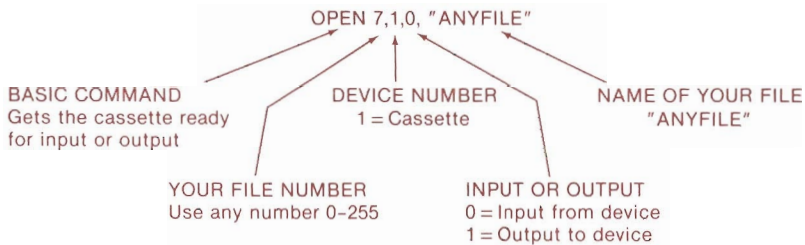
The END,0 breaks your program out of the loop in line #50, and the program jumps to line #999. In line #999, file #5 is closed and the program ends.

Write down the tape-counter reading so that you know where your data are stored on the tape. Your list of four phone numbers is now recorded on tape under the file name "PHONE." Next, we will show you how to read the data on this file and print it on the screen.

## **7** HOW DO I READ DATA RECORDED ON CASSETTE TAPE?

To read a data file recorded on cassette tape, you must use these commands: OPEN, INPUT#, CLOSE, and STATUS.

The OPEN command for reading a data file is:



With this OPEN command, you say to the computer: "The file in my program, which I call file #7, is on cassette tape. I want to read in data from this file."

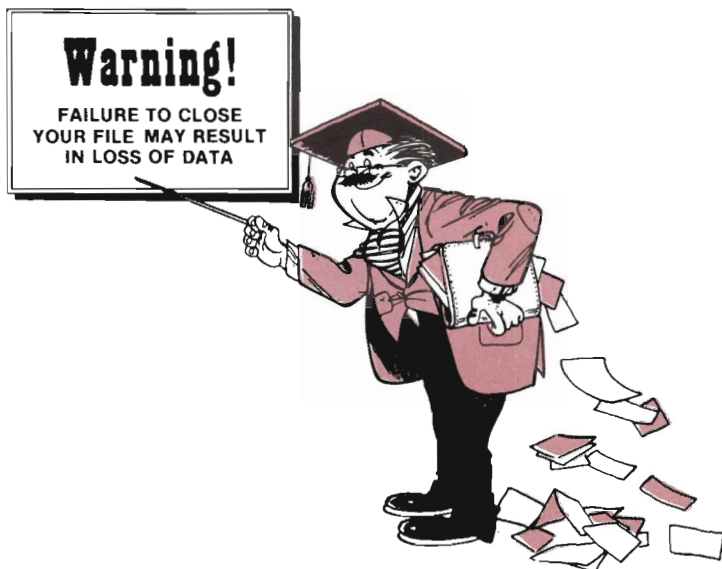
The INPUT# command is similar to INPUT. However, instead of entering data from the keyboard, as INPUT does, it enters data from the device that you have opened with the OPEN command.

The INPUT command looks like this:

**INPUT#7, A\$, N**

It tells the computer: "Bring in one string and one number from my file #7. Make the string equal to A\$ and make the number equal to N."

The CLOSE command tells the computer that you are now finished with the file. When you are finished with any open file, you should always close it. If you don't, some of your data may be lost.





STATUS is a reserved BASIC word that tells whether all the data recorded on the file have been read. STATUS = 0 means that there are more data to read and that the program should loop back to the READ statement.

## EXAMPLE

Suppose you want to read in and print our previously recorded file of names and telephone numbers. Type NEW and enter the following program:

```
10 REM INPUT FROM TAPE
20 OPEN 7,1,0,"PHONE"
30 INPUT#7,N$,PH$
40 PRINT N$,PH$
50 IF STATUS=0 THEN 30
60 CLOSE 7:END
```

Type RUN and hit RETURN.

When you run this program, the screen will go blank for a short period of time. Then you should see on the screen:

GEORGE	917-2432
HARRY	214-2446
JIM	483-8104
STEVE	761-1122

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. To OPEN a cassette file on which you want to record data, use:

```
OPEN 5,1,1,"PHONE"
```

The 5 is your own reference number for this file. (You can use any number between 0 and 255.) To write on the file, you will use a PRINT#5 command.

The 1 in the middle tells the computer that you want to record on Device 1, which is the cassette recorder.



The *I* at the right tells the computer that you are opening this file for output. (You are going to record data on this file.)

"PHONE" is the name that you are going to assign to this file.

2. To OPEN a cassette file that has data on it, and that you want to read in, use:

```
OPEN 7,1,0,"PHONE"
```

The 7 is your reference number for this file. (You can use any number from 0 to 255.) To input data from this file, you will use an INPUT#7 command.

The *I* tells the computer that you want to read data in from device 1, which is the cassette recorder.

The 0 tells the computer that you are opening this file for input.

"PHONE" is the name of the file on which the data are to be found.



Part II

**More-Advanced Features**



Day 9

## USING FUNCTIONS

In today's lesson you will learn:

- What functions are.
- How functions can help you in writing a program.
- How to use specific functions.



# 1

## WHAT ARE FUNCTIONS?

*Functions* are built-in operations that can be used to make your programming easier.

The LEN function is a fairly easy one to understand. You can use it to *find out the number of characters in a string*. For example, if you want to find out the number of characters in the string "ALICE", type NEW and enter the following program:

```
10 N=LEN("ALICE")
20 PRINT N
```

Type RUN and hit RETURN. The screen should show 5.

Line #10 of this program contains the function LEN ("ALICE"). The string "ALICE", whose characters we want to count, is placed in parentheses right after LEN. We call "ALICE" the *argument* of the function LEN. LEN ("ALICE") has a value of 5. If we say N=LEN ("ALICE"), then we also make N equal to 5.

The DIRECT MODE is handy for trying to understand functions. Instead of our little program, we could have just typed the DIRECT MODE command:

```
PRINT LEN ("ALICE ")
```

Try this yourself. Practice it with other strings in place of "ALICE".

The function LEN counts characters, so it will always give a number. Many other functions give numbers. These are called *numeric functions*.

Another group of functions, called *string functions*, gives strings. String functions must end with a \$. An example of this kind of function is LEFT\$.

LEFT\$ can be used to tell us which characters make up the left side of a string. For example, to find out the left four characters of the string "TELEVISION", type NEW and enter the following program:

```
10 Q$=LEFT$("TELEVISION",4)
20 PRINT Q$
```

Type RUN and hit RETURN. The screen should show TELE.

The same results can be obtained in DIRECT MODE by typing in:

```
PRINT LEFT$ ("TELEVISION",4)
```

**Important rule:** Those functions that give a number (like LEN) can

be made equal *only* to a numeric variable. Those functions that give a string (like LEFT\$) can be made equal *only* to a string variable.

It is all right to say:

```
X=LEN ("CAR")
L$=LEFT$ ("BOAT",3)
```

It is not all right to say:

```
X$=LEN ("CAR")
L=LEFT$ ("BOAT",2)
```

Functions are a very important part of BASIC. You should take the time to learn about them. But, if you are a little confused at this point, don't worry. The examples on the following pages will clear things up.

## 2 WHAT ARE SOME OF THE FUNCTIONS COVERED IN THIS SECTION AND WHAT DO THEY DO?

Below is a list of the functions covered in this section, including a brief description of what each one does. Remember that those functions that end in a \$ are used to give a string. All others are used to give a number.

### Function

LEN

ABS

INT

SGN

ASC

CHR\$

LEFT\$

RIGHT\$

MID\$

STR\$

VAL

RND

### Lets Us Find Out The:

Number of characters in a string

Absolute value of a number

Integer part of a number

Sign of a number

ASCII number of a character

Character of an ASCII number

Left side of a string

Right side of a string

Middle part of a string

String value of a number

Number value of a string

Random number

## 3 HOW CAN I USE THE LEN FUNCTION TO FIND OUT THE NUMBER OF CHARACTERS IN A STRING?

Although the LEN function was covered in Question #1, here is another example:

```
PRINT LEN ("BOOK")
```

After you enter this statement in DIRECT MODE, the screen should show 4 because there are four characters in the string "BOOK".

## EXAMPLE

Suppose you want to input a name and have the computer print the number of letters in the name. Type NEW and enter the following program:

```
10 INPUT "ENTER NAME";Q$:IF Q$="END" THEN END
20 N=LEN(Q$)
30 PRINT "THE NUMBER OF CHARACTERS=";N
40 PRINT:GOTO 10
```

Type RUN and hit RETURN.

Note in line #20 that you can put a variable like Q\$ in the parentheses (). Q\$ is equal to the string that you input. To end this program, type in END next to the ?.

## 4 WHAT CAN THE FUNCTIONS ABS, INT AND SGN DO FOR ME?

**ABS** lets you find out the *absolute value* of a number:

The absolute value of -17 is 17.

The absolute value of -327 is 327.

The absolute value of 98 is 98.

The absolute value of any number is the number itself. Therefore, PRINT ABS (- 17) gives 17.

**INT** lets you find out the *integer value* of a number:

The integer value of 71.43 is 71.

The integer value of 247.38 is 247.

The integer value is the whole number portion. Therefore, PRINT INT (247.43) gives 247.

**SGN** lets you find out the *sign* (+ or -) of a number:

If the number is +, then SGN gives +1.

If the number is -, then SGN gives -1.

If the number is 0, then SGN gives 0.

Therefore, PRINT SGN (300) gives +1; PRINT SGN (- 307) gives -1; and PRINT SGN (0) gives 0.



## EXAMPLE

Suppose you want to input a number and print its absolute value and integer value. If the number is a negative number, you want the computer to print the message "NEGATIVE NUMBER" and END the program. Type NEW and enter the following program.

```
10 INPUT "ENTER NUMBER";X
20 PRINT "ABSOLUTE VALUE=";ABS(X)
30 PRINT "INTEGER VALUE=";INT(X)
40 IF SGN(X)=-1 THEN PRINT "NEGATIVE NUMBER":END
50 PRINT
60 GOTO 10
```

Type RUN and hit RETURN.

Here are some numbers to try:

20.775  
.804  
- 7.3

A negative number will end the program.

## 5 WHAT IS THE ASCII CODE?

ASCII stands for American Standard Code for Information Interchange. It is a standard code used by most computer manufacturers. Every character that your computer uses has a code number. For example, the ASCII code for A is 65, and the ASCII code for X is 88.

Let's take a look at a short table of these codes:

Character	Its ASCII Code
A	65
B	66
X	88
Comma	44
CLR/HOME	147
Red	28
Cursor down	17
♥ (heart)	115

As you can see from our little table, even graphics characters (like the heart), cursor commands, and colors have an ASCII code number. A complete table of ASCII codes is shown in Appendix A.

**6****HOW DO I FIND OUT THE ASCII CODE OF A CHARACTER?**

Another function to the rescue! The ASC function will help you find out the ASCII code for any character.

To find out the ASCII code for X, enter:

```
PRINT ASC ("X")
```

The screen should show 88.

To find out the ASCII code for 7, enter:

```
PRINT ASC ("7")
```

The screen should show 55. Now, wasn't that easy?

**7****HOW DO I CHANGE FROM AN ASCII CODE TO A CHARACTER?**

Changing from an ASCII code to a character is the reverse of what we did in Question #6. The function CHR\$ is used to change from the ASCII code into the character that it stands for. For example, to find out the character having an ASCII code of 88, enter and run:

```
PRINT CHR$ (88)
```

The screen should show X.

To find out the character whose code is 55, enter and run:

```
PRINT CHR$ (55)
```

The screen should show 7.

**EXAMPLE**

Suppose you want to clear the screen, print 2 hearts, move the cursor down, and print 2 "A's." Type NEW and enter the following program:

```
10 PRINT CHR$(147):REM CLEAR SCREEN
20 PRINT CHR$(115);CHR$(115):REM PRINT 2 HEARTS
30 PRINT CHR$(17):REM CURSOR DOWN
40 PRINT CHR$(65);CHR$(65):REM PRINT 2 A'S
```

Type RUN and hit RETURN. Notice in line #10 that the statement PRINT CHR\$ (147) is another way to clear the screen, because 147 is the ASCII code for CLR/HOME (see Appendix A).

## 8

**HOW CAN I FIND OUT WHICH CHARACTERS MAKE UP DIFFERENT PARTS OF A STRING?**

There are three functions that can be used to find out the different parts of a string:

LEFT\$ finds out characters on the left side of a string.

RIGHT\$ finds out characters on the right side of a string.

MID\$ finds out characters in the middle of a string.

Suppose you have a long string that looks like this:

```
"ABCDEFGHI "
```

To pull out the three left characters, type NEW and enter the following program:

```
10 Q$="ABCDEFGHI "
20 PRINT LEFT$(Q$,3)
```

Type RUN and hit RETURN. The screen should show ABC.

To pull out the five right characters, type NEW and enter the following program:

```
10 X$="ABCDEFGHI "
20 PRINT RIGHT$(X$,5)
```

Type RUN and hit RETURN. The screen should show EFGHI.

To pull out the four middle characters, starting with the third character (which is the C), type NEW and enter the following program:

```
10 C$="ABCDEFGHI "
20 PRINT MID$(C$,3,4)
```

Type RUN and hit RETURN. The screen should show CDEF.

**EXAMPLE**

Suppose you want the computer to read the names of a group of states and then print those that either begin with an "A" or end with an "A." Type NEW and enter the program:

```
10 PRINT "<SC>"
20 READ X$:IF X$="END" THEN END
30 IF LEFT$(X$,1)="A" OR RIGHT$(X$,1)="A" THEN PRINT X$
40 GOTO 20
50 DATA TEXAS,GEORGIA,ARKANSAS
60 DATA ALABAMA,WASHINGTON,NEW YORK
70 DATA MARYLAND,LOUISIANA,END
```

Type RUN and hit RETURN.

**9****HOW CAN I CHANGE A STRING TO A NUMBER OR A NUMBER TO A STRING?**

There are times when you want to change a string of numerals (like "1147") into a number so that you can use the numbers in a mathematical operation. The VAL function changes a string to a number. For example, to change the string "1147" to the number 1147 and add 100 to it, type NEW and enter the following program:

```
10 N=VAL("1147")
20 PRINT N+100
```

Type RUN and hit RETURN. The screen should show 1247.

At other times, you may want to change a number (like 427) into a string so that it can be combined with another string (like "XYZ"). The STR\$ function will change a number to a string. For example, to change the number 427 to a string and combine it with the string "XYZ", type NEW and enter the following program:

```
10 A$=STR$(427)+"XYZ"
20 PRINT A$
```

Type RUN and hit RETURN. The screen should show 427XYZ, which is the new string.

**10****WHAT ARE RANDOM NUMBERS AND HOW ARE THEY USED IN PROGRAMS?**

Random numbers are numbers that cannot be predicted. These are random numbers: 1, 44, 18, 143, 7 . . . . These are NOT random numbers: 1, 2, 3, 4, 5, 6 . . . . You can use random numbers to make up games where chance plays a part. The random numbers can simulate flipping a coin or rolling a pair of dice.

Let's see how random numbers are created. To create and print a random number between 0 and 1, enter and run:

```
10 PRINT RND (1)
```

To create 20 random numbers between 0 and 1, type NEW and enter the following program:

```
10 FOR N=1 TO 20
20 R=RND(1)
30 PRINT R
40 NEXT N
```

Type RUN and hit RETURN.

To create 20 integers (whole numbers) between 0 and 9, type NEW and enter the following program:

```
10 FOR N=1 TO 20
20 R=INT( RND(1)*10)
30 PRINT R
40 NEXT N
```

Type RUN and hit RETURN. The random numbers that will be printed will not include the number 10. That's because the highest number created by RND(1) is .99999 and, therefore, INT (.99999\*10) = 9.

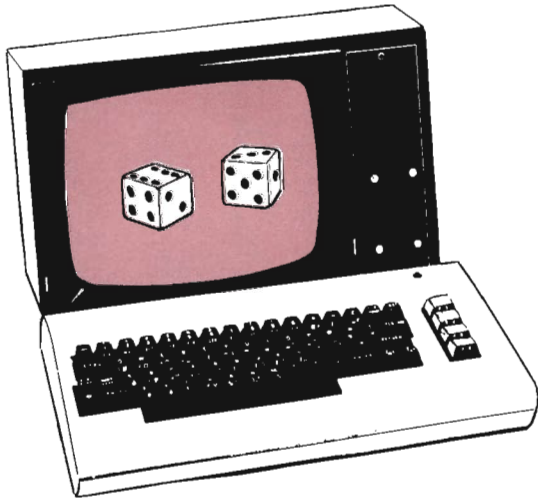
To create 20 integers from 1 to 10 (including the number 10), type NEW and enter the following program:

```
10 FOR N=1 TO 20
20 R=INT( RND(1)*10)+1
30 PRINT R
40 NEXT N
```

Type RUN and hit RETURN. To include the number 10, we had to add 1 to R. This gave us a range of 1 to 10.

To simulate the flip of a coin, we need random numbers between 1 and 2. The 1 will stand for heads; the 2 will stand for tails. To get a range of 1 to 2, enter and run:

```
10 PRINT INT (RND(1)*2)+1
```



**You can use random numbers to simulate flipping a coin or rolling a pair of dice.**

## EXAMPLE

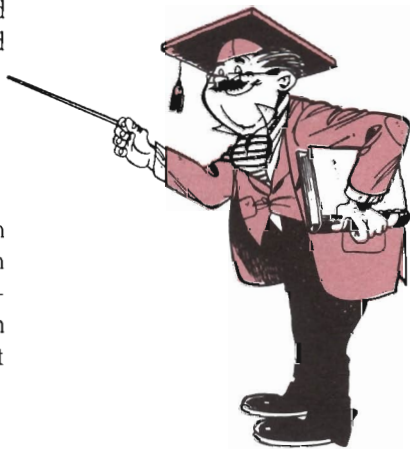
Suppose you want the computer to “flip a coin” and to print either HEADS or TAILS with each “flip.” If five heads or five tails are flipped in a row, you want the program to END. You also want the program to print the number of flips that it took to get five heads or five tails in a row. Type NEW and enter the following program:

```
10 R=INT(RND(1)*2)+1
20 IF R=1 THEN PRINT "HEADS":T=0:H=H+1
30 IF R=2 THEN PRINT "TAILS":H=0:T=T+1
40 FLIPS=FLIPS+1
50 IF H=5 OR T=5 THEN PRINT "# OF FLIPS=";FLIPS:END
60 GOTO 10
```

Type RUN and hit RETURN.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

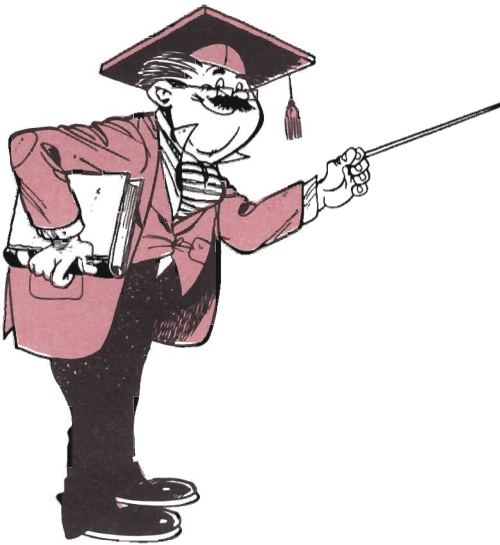
1. There are two kinds of functions: those that give a number (called *numeric functions*) and those that give a string (called *string functions*).
2. Numeric variables can be set equal to *numeric functions*. String variables can be set equal to *string functions*.
3. ASCII stands for American Standard Code for Information Interchange. It is used by computer manufacturers to assign a code number to different characters.



## HOW ARRAYS MAKE PROGRAMMING EASIER

In today's lesson you will learn:

- What an ARRAY is.
- How to use ARRAYS to help you write your program.
- What the difference is between ONE-DIMENSIONAL and TWO-DIMENSIONAL ARRAYS.



# 1

## WHAT IS AN ARRAY?

An *array* is a special kind of variable. It can be used to stand for numbers or strings.

Here is an example that should help you understand how arrays are used. Suppose you have three boxes of cookies:

Box #1 has five cookies in it.  
Box #2 has seven cookies in it.  
Box #3 has two cookies in it.

There are two ways to represent these boxes in a program. The first way is to use letter variables, like A, B, and C. We could say that Box #1 is A, Box #2 is B, and Box #3 is C. Therefore, our program would contain these lines:

```
10 A=5  
20 B=7  
30 C=2
```

The second way is to use *arrays*. Our program would then have these lines:

```
10 BOX (1)=5  
20 BOX (2)=7  
30 BOX (3)=2
```

What we have done is set up an array called BOX (N). In this example, N can be either 1, 2, or 3 because there are three boxes. You can think of the array BOX (N) as three different variables with the same name but a different number. The three different variables are BOX (1), BOX (2), and BOX (3). These variables are also called *elements* of the array BOX (N).

Using arrays instead of regular variables has some very important advantages in certain situations, as we will soon see.

# 2

## WHY SHOULD I BOTHER TO LEARN ABOUT ARRAYS?

That's a good question. After all, arrays are not the easiest concept to understand, and learning to use them will take a little work on your part. But, *arrays make writing some programs much easier, and they make many programs much shorter.*



Suppose you have 10 boxes of cookies. Each box has 18 cookies in it. If you used regular variables to represent each box, then your program might start out something like this:

```
10 A=18
20 B=18
30 C=18
.
.
.
Up to J=18
```

This method is very long and awkward. (What if you had one thousand boxes? How long and awkward would that be?) Using arrays, your program would look much simpler, as shown here.

```
10 FOR N=1 TO 10
20 BOX(N)=18
30 NEXT N
```

As you can see, the FOR-NEXT loop does all the hard work. Instead of having to say BOX (1)=18, BOX (2)=18, BOX (3)=18, etc., you can just say BOX (N)=18 and let the FOR-NEXT loop change N from 1 to 10.

**When to use arrays:** If you have many of the same kind of variable (like 1,000 boxes of cookies), it is easier to use arrays than to call each box by a different variable name.

### 3 HOW DO I GET DATA INTO AND OUT OF AN ARRAY?

The simplest way to get data into and out of an array is to work with each element, *one at a time*. Type NEW and enter the following program:

```
10 BOX(1)=5
20 BOX(2)=7
30 BOX(3)=2
40 PRINT BOX(3),BOX(1),BOX(2)
```

Type RUN and hit RETURN. On the screen you should see:

```
2      5      7
```

But to really make the best use of arrays, you should put them in a loop. Type NEW and enter this program:

```

10 FOR I=1 TO 9:READ BOX(I):NEXT I
20 FOR K=1 TO 9:PRINT BOX(K);:NEXT K
30 DATA 5,7,2,3,22,15,21,10,1

```

Type RUN and hit RETURN. On the screen you should see:

```

5 7 2 3 22 15 21 10 1

```

Notice that line #10 causes data to be read directly into the array BOX(I) and, in line #30, to be printed out onto the screen.

## EXAMPLE

Suppose you work for five days and earn a different amount of money each day.

Day No.	You Earned
1	\$10
2	\$20
3	\$30
4	\$40
5	\$50

You want to put the amount you earned each day into an array called DAY (N). Then, for each day, you want to print the day number and the amount earned. At the end, you want to print the total earned. Type NEW and enter the following program:

```

10 REM READ NUMBERS
20 FOR N=1 TO 5
30 READ DAY(N)
40 NEXT N
50 REM PRINT NUMBERS ON SCREEN
55 PRINT "{SC}"
60 FOR N=1 TO 5
70 PRINT "DAY # ";N, DAY(N)
90 T=T+DAY(N)
100 NEXT N
110 PRINT:PRINT:PRINT "TOTAL=",T
120 DATA 10,20,30,40,50

```

Type RUN and hit RETURN.

## 4

### CAN STRINGS BE PUT INTO ARRAYS?

Yes, definitely! Moving strings into and out of an array is no more difficult than moving numbers. You must follow these two rules, however:

1. Your *string array* must have a \$ sign after its name, to tell the computer it is a string array.
2. Only strings can be moved into a string array. If you try to move a number into it, you will get an error message.

To read in and print 10 names, type NEW and enter the following program:

```
10 FOR N=1 TO 10:READ NAME$(N):NEXT N
20 FOR N=1 TO 10:PRINT NAME$(N):NEXT N
30 DATA JOHN,FRANK,PETE,HOWARD,ED
40 DATA EVA,ILENE,JODY,ELISE,TRUDI
```

Type RUN and hit RETURN.

## EXAMPLE

Suppose you want to put the names of the first five presidents of the United States into an array called NAME\$(N). You then want to be able to input a number from 1 to 5 and have the computer tell you the name of that president. Type NEW and enter the following program:

```
10 REM PRESIDENTS OF THE US
20 N=1
30 READ NAME$(N):IF NAME$(N)="END" THEN 50
40 N=N+1:GOTO 30
50 PRINT "(SC)"
60 PRINT:PRINT
70 INPUT "ENTER 1-5";X:IF X<1 OR X>5 THEN END
80 PRINT "-----"
90 PRINT "PRESIDENT #";X;" IS:";NAME$(X)
100 GOTO 60
110 DATA GEORGE WASHINGTON
120 DATA JOHN ADAMS
130 DATA THOMAS JEFFERSON
140 DATA JAMES MADISON
150 DATA JAMES MONROE
160 DATA END
```

Type RUN and hit RETURN.

## 5 WHAT IS THE DIM COMMAND AND WHY DO I NEED IT?

Each array that you use must have a certain amount of space reserved for it in memory. The purpose of the DIM command is to tell the computer how much space to reserve. DIM stands for dimension.

In your program, the DIM command must be placed *before* the array is used. The best place to put it is near the beginning of the program. You need only one DIM command to list all the arrays you plan to use.

The DIM command is easy to use. If you have a numeric array called BOX (N) with 20 elements, use:

```
10 DIM BOX (20)
```

If you have a numeric array called DAY (N) with 30 elements, use:

```
20 DIM DAY (30)
```

If you have a string array called NAME\$ (B) with 150 elements, use:

```
10 DIM NAME$ (150)
```

If you plan to use all of them in the same program, then you can put them all in the same statement like this:

```
15 DIM BOX (20), DAY (30), NAME$ (150)
```

If the highest element number in your array is 10 or less, you *don't have to* use a DIM command. The computer will automatically reserve the memory space for you. For example, DIM X (10) is not necessary. The Commodore 64 will reserve X (0) through X (10) for you automatically. But if you need to use X (11) or greater, then you must use DIM.

If your array contains more elements than you reserved in your DIM for that array, the program will not run. If you reserve more than you need, the program will run, but you will waste memory space. This may not make any difference, however, if you do not run short of memory space.

## **6 HOW CAN I COPY DATA FROM ONE ARRAY INTO ANOTHER?**

Sometimes you will want to copy data from one array into another. This is very easy to do. You can copy the data one element at a time, like this:

```
10 BOX (5) = Q (7)
```

Here we have copied the data that was in the array element Q (7) into the array element BOX (5). Q (7) is not changed—only its data have been copied.

Or, you can copy many elements by using a loop, as shown in this example.

```
10 FOR I=5 TO 10
20 BOX(I)=Q(I)
30 NEXT I
```

Here, we have copied elements 5 through 10 from ARRAY Q (I) into ARRAY BOX (I).

## EXAMPLE

Suppose you have an array called N (X). In N (X), you want to put the first 20 numbers in a list. You also have an array called T (X). In T (X), you want to put all the numbers in N (X) that are negative. Finally, you want to print out N (X) and T (X).

When you run the following program, you will see that the ARRAY T (X) contains all the negative numbers that are in the ARRAY N (X). Type NEW and enter this program:

```
5 PRINT "(SC)"
8 DIM N(20),T(20)
10 FOR X=1 TO 20:READ N(X):NEXT X
20 FOR X=1 TO 20
30 IF N(X)<0 THEN T(X)=N(X)
40 NEXT X
45 PRINT "N(X)", "T(X)":PRINT
50 FOR X=1 TO 20:PRINT N(X),T(X):NEXT X
60 DATA 5,7,23,-4,-7,14,10,5
70 DATA -21,3,8,-4,7,21,14,-1
80 DATA 10,21,-3,14
```

Type RUN and hit RETURN.

## 7

## WHAT ARE TWO-DIMENSIONAL ARRAYS?

In order to explain two-dimensional arrays, let's look at some boxes of candy. Suppose you have four boxes of candy, and you want to arrange them on a table top. Here are two ways to arrange them. In the first way, you can put all four boxes in a single row like this:

1      2      3      4

In a program, this arrangement is represented by a one-dimensional array with four elements. These are:

BOX (1)   BOX (2)   BOX (3)   BOX (4)

In the second way, you can put the four boxes in two rows and two columns, like this:

		Column	
		1	2
Row	1	1,1	1,2
	2	2,1	2,2

In a program, this arrangement is represented by a two-dimensional array with four elements.

```
BOX (1,1)    BOX (1,2)
BOX (2,1)    BOX (2,1)
```

The first number is the row and the second number is the column.

Here is another example. If you have six boxes, they can be arranged as follows:

1. All six in a row.
2. Two rows of three columns.
3. Three rows of two columns.

In a program, the arrangement of all six in a row can be represented as a one-dimensional array with six elements:

```
BOX (1)    BOX (2) . . . BOX (6)
```

You can represent the arrangement of *two rows of three columns* as a two-dimensional array:

```
BOX (1,1)  BOX (1,2)  BOX (1,3)
BOX (2,1)  BOX (2,2)  BOX (2,3)
```

You can represent the arrangement of *three rows of two columns* as a different two-dimensional array:

```
BOX (1,1)  BOX (1,2)
BOX (2,1)  BOX (2,2)
BOX (3,1)  BOX (3,2)
```

In each case, there are six elements.

## **8** HOW DO I USE THE *DIM* COMMAND WITH TWO-DIMENSIONAL ARRAYS?

The DIM command is also used to reserve memory space for two-dimensional arrays. For each two-dimensional array in your DIM



**Two-dimensional arrays have many rows. One-dimension arrays are all in one row.**

statement, you must tell the computer the highest number of rows and columns used in that array.

For example, if you have an array called HOUSE (R,C), with 10 rows and 20 columns, then use:

```
15 DIM HOUSE ( 10, 20 )
```

If you have a string array called CITY\$ (R,C), with 15 rows and 200 columns, then use:

```
15 DIM CITY$ ( 15, 200 )
```

If you have both the array HOUSE (R,C) and the array CITY\$ (R,C) in the same program, then use:

```
10 DIM HOUSE ( 10, 20 ) , CITY$ ( 15, 200 )
```

If you have an array like BOX (9,10) where neither the highest row number nor the highest column number is greater than 10, then you don't need to put that array into a DIM statement. The computer will automatically reserve the memory space for you.

## 9 HOW DO I GET DATA INTO AND OUT OF A TWO-DIMENSIONAL ARRAY?

Getting data into and out of a two-dimensional array is similar to getting it into and out of a one-dimensional array. You can do it in one of two ways:

1. With one element at a time.
2. With a loop.

Here is an example of one element at a time. Type NEW and enter the following program:

```
10 BOX(1,1)=7
20 BOX(2,1)=10
30 PRINT BOX(1,1),BOX(2,1)
```

Type RUN and hit RETURN. On the screen you should see:

```
7      10
```

Now, here is an example with a loop. Suppose you have a two-dimensional array that looks like this:

		<b>Column</b>		
		<b>1</b>	<b>2</b>	<b>3</b>
<b>R</b>	<b>1</b>	11	22	33
<b>o</b>	<b>2</b>	44	55	66

It has six elements arranged in two rows of three columns each. To read this data into a two-dimensional array and then to print out the elements in (ROW 1, COL 1), (ROW 2, COL 2), and (ROW 1, COL 3), type NEW and enter the following program:

```
10 FOR R=1 TO 2
20 FOR C=1 TO 3
30 READ BOX(R,C)
40 NEXT C
50 NEXT R
60 PRINT BOX(1,1),BOX(2,2),BOX(1,3)
70 DATA 11,22,33,44,55,66
```

Type RUN and hit RETURN. Notice the nested loop in lines #10 through #50. This loop first makes R = 1 (for Row 1) and reads in data for columns 1, 2, and 3. Next, it makes R = 2 (for Row 2) and again reads in data for columns 1, 2, and 3.



## EXAMPLE

A student who graduated from high school (years 10, 11, and 12) received his math grades every quarter. Here are the grades he received:

		Quarter			
		1	2	3	4
Year	10	A	A	C	B
	11	B	C	D	C
	12	C	B	A	B

You want this data stored in the computer so that you can type in the year and quarter and have the computer give you the student's grade. For example, if you type in year 11, quarter 2, the computer should print on the screen the grade C.

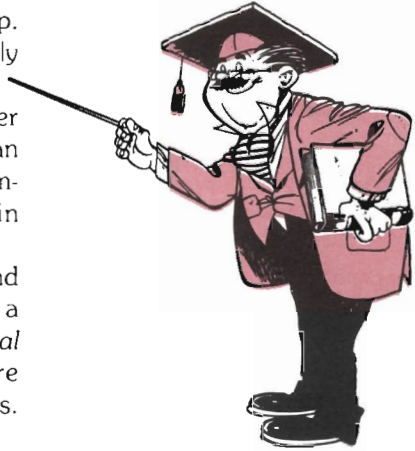
To END the program, type in any year other than 10, 11, or 12, or type in any quarter other than 1, 2, 3, or 4. Now type NEW and enter this program:

```
10 REM GRADES
20 PRINT "(SC)"
30 DIM GRADE$(12,4)
40 FOR Y=10 TO 12
50 FOR Q=1 TO 4
60 READ GRADE$(Y,Q)
70 NEXT Q
80 NEXT Y
90 INPUT "ENTER YEAR,QUARTER";Y,Q
100 IF Y<10 OR Y>12 THEN END
105 IF Q<1 OR Q>4 THEN END
110 PRINT GRADE$(Y,Q)
120 PRINT
130 GOTO 90
500 DATA A,A,C,B,B,C,D,C,C,B,A,B
```

Type RUN and hit RETURN.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON:

1. An easy way to get data into or out of an array is to use a loop. A FOR-NEXT loop is especially handy for this purpose.
2. If the highest element number in your array is greater than 10, you must use a DIM command to reserve space in memory for your array.
3. *One-dimensional* arrays stand for data that are arranged in a single row. *Two-dimensional* arrays stand for data that are arranged in rows and columns.



## PEEKING AND POKING THE MEMORY

In today's lesson you will learn:

- How to put numbers into the computer's memory by using the POKE command.
- How to look into the computer's memory by using the PEEK command.
- How to calculate where in memory your BASIC program is located.



## 1

**HOW ARE THE COMMANDS POKE AND PEEK USED?**

POKE is used to put a number into a memory location. That number must be between 0 and 255. To POKE the number 151 into memory location 3000, enter:

```
POKE 3000, 151
```

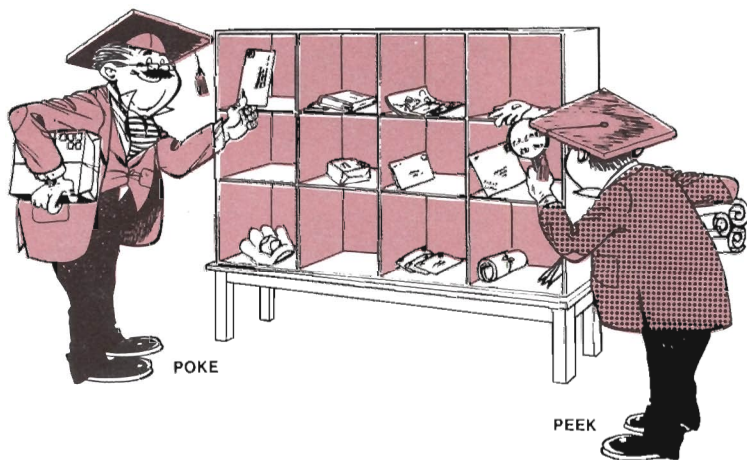
PEEK is used to look into a memory location. To look into location 3000 (the place where we just POKED the number 151), enter:

```
PRINT PEEK (3000)
```

On the screen you should see 151. By using the POKE and PEEK commands, you have put a number into one of the memory locations on the Commodore 64 and then read it back out.

**Caution:** Some memory locations are used to store programs or data. Others are used to control the computer. If you POKE a number into a control location, the computer may begin to act strangely. It is even possible that the keyboard will stop working. If this should happen, just turn off the computer and then turn it back on. The problem should disappear, and the computer should operate normally. However, any programs that you had in memory at the time will be lost.

You cannot hurt the computer in any way by using the POKE or PEEK commands.



**POKE is used to put a number into a memory location. PEEK is used to look into a memory location to find out what number is stored there.**

**2****HOW CAN I USE POKE TO TURN OFF THE KEYBOARD?**

Remember our caution. If you POKE a number into the wrong memory location, strange things may happen. Here is an example. Turn off the keyboard by entering:

```
POKE 649,0
```

Your keyboard will not work now. Try it.

The surest way to fix this kind of problem is to turn off the computer and turn it back on again. However, if you have any programs in memory at the time, they will be lost. *Sometimes* you can fix your problem by hitting RUN/STOP and RESTORE. Try it, and you will see that it works *in this case*. If you have a program in memory, it will not be lost. While this is a better way to fix your problem, it will not always work.

**3****HOW CAN I USE POKE TO CREATE A SECRET PROGRAM THAT CANNOT BE LISTED?**

To keep someone else from listing your program, put this statement in it. (It will also disable the RUN/STOP key.)

```
POKE 808, 225
```

**EXAMPLE**

Suppose you want to keep other people from being able to list the following program. Type NEW and enter:

```
10 PRINT "APPLE"  
20 PRINT "ORANGE"  
30 PRINT "BANANA"  
40 POKE 808,225
```

Type RUN and hit RETURN.

Now, try to LIST your program. Bet you can't do it! (Try betting a friend that he or she can't do it either.) To be able to list again, enter:

```
POKE 808, 237
```

## 4 HOW CAN I USE POKE TO PRINT REVERSE CHARACTERS?

In the lesson for Day 4, we used the keyboard to print reversed characters. Now we will see how POKE can be used to do the same thing.

To print reversed characters, use:

```
POKE 199, 1
```

To return to normal characters, use:

```
POKE 199, 0
```

### EXAMPLE

Suppose you want to print "ABC" normally, then "DEF" reversed, then "GHI" normally. Type NEW and enter the following program:

```
10 PRINT "ABC";  
20 POKE 199,1:PRINT "DEF";  
30 POKE 199,0:PRINT "GHI"
```

Type RUN and hit RETURN.

## 5 HOW CAN I USE POKE TO MAKE KEYS AUTOMATICALLY REPEAT WHEN I PRESS THEM DOWN?

Press the key for the letter "K." You will see that one "K" is printed on the screen every time you press down this key. But, with the right POKE, you can get many "K's" on the screen when you press down the K key. To do this, enter:

```
POKE 650, 128
```

Now, press the K key, or any other key, and watch how the letters repeat. To go back to normal, enter:

```
POKE 650, 0
```

## 6 HOW CAN I USE POKE TO CHANGE THE COLOR OF THE SCREEN AND BORDER?

To change the screen color, POKE a color code between 0 and 15 into memory location 53280. To change the border

color, POKE a color code between 0 and 15 into memory location 53281. A list of these color codes is shown in Appendix C.

Here are some examples to try:

To get a blue border and white screen, use POKE 53280,6: POKE 53281,1.

To get a green border and a red screen, use POKE 53280,5: POKE 53281,2.

To get an all-white screen, use POKE 53280,1: POKE 53281, 1. The all-white screen is what we have been using since the beginning of the book and it is the combination we will continue to use.

## **7 HOW CAN I USE PEEK TO FIND OUT THE LINE NUMBER AND COLUMN NUMBER OF THE CURSOR'S LOCATION ON THE SCREEN?**

The line number of the cursor is always kept in memory location 214. The column number is always kept in memory location 211. To find out the line number of the cursor, enter:

```
PRINT PEEK (214)
```

To find out the column number of the cursor, enter:

```
PRINT PEEK (211)
```

### **EXAMPLE**

Suppose you want the computer to start in the upper left corner of the screen and to print three lines. On the third line, in the sixth column, you want to print five characters and an X. You then want to print the line number and column number of the X. Type NEW and enter the following program:

```
10 PRINT "{SC}"
20 PRINT "1":PRINT "2":PRINT "32345X";:L=PEEK(214):
  C=PEEK(211)
30 PRINT:PRINT "X IS ON LINE=";L,"COLUMN=";C
```

Type RUN and hit RETURN.

## 8 HOW CAN I USE PEEK TO FIND OUT THE MEMORY LOCATION OF MY BASIC PROGRAM?

When you type in your BASIC program, it is stored in memory starting at memory location 2049. In order to run your program, the computer must know this location. It looks into its own memory locations 43 and 44 to find out where BASIC begins. By using PEEK, you too can find out.

To find out where your BASIC program starts, enter:

```
PRINT PEEK ( 44 ) * 256+PEEK ( 43 )
```

On the screen you will see 2049.

How did we get to 2049? Let's take a closer look. Enter this line:

```
PRINT PEEK ( 44 ) , PEEK ( 43 )
```

On the screen you will see:

```
8      1
```

This means that the number at memory location 44 is an 8. The number at memory location 43 is a 1.

Both of these memory locations are used to calculate where BASIC begins. Memory location 44 is called the *high byte*, and memory location 43 is called the *low byte*. To calculate any memory location using a two-byte address, use the formula:

$$\text{Hi Byte} * 256 + \text{Low Byte}$$



## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON:

1. POKE is used to put a number into a memory location. PEEK is used to look into a memory location to see what number is stored there.
2. The highest number you can POKE into any memory location on your Commodore 64 is 255.
3. You can find out where in memory your BASIC program begins by PEEKing into memory locations 44 and 43. To calculate and print the starting address, use:

```
PRINT PEEK ( 44 ) * 256 +  
PEEK ( 43 )
```



### **IMPORTANT NOTE**

The programs in the lessons for Days 12, 13, and 14 are longer and more complicated than the preceding ones, and make frequent use of the POKE command. If you make a mistake when typing in your program, a POKE to the wrong memory location may result. While this will not cause any permanent damage to your computer, it is possible that the computer may begin to act strangely after you run your program.

Should this happen, you can restore normal operation by turning the computer off and then back on again. When you turn your computer off, however, the program you typed in will be lost. It is therefore recommended that you save each program *before* you run it. Then, if you lose your program in memory, you will not need to retype the entire program. After you restore normal operation, just load your saved program back in, LIST it, find and correct your error, and RUN it again. If you wish, you may then save the corrected version of the program.

## Day 12

# SOUND AND MUSIC

In today's lesson you will learn:

- How the computer produces sound and music.
- How to create many different kinds of sounds by controlling **VOLUME**, **FREQUENCY**, **ADSR**, and **WAVEFORM**.
- How to play the tune "Jingle Bells" on the computer.



# 1

## HOW DOES THE COMMODORE 64 PRODUCE SOUNDS AND MUSIC?

Sounds and music are produced on the Commodore 64 by a special chip called the SID (SOUND INTERFACE DEVICE). SID has many exciting features. Here are just some of them:

- It has three voices, which you can use one at a time or all at once.
- It has a volume adjustment, which you can control from your program.
- It can produce sounds in a wide range of frequencies (or pitches).
- It lets you control how fast your volume rises and falls. (This is called ADSR.)
- It can produce sounds by using any of four different waveforms.

In fact, SID has so many features that an entire book could be written about them. However, in this section, we will merely introduce you to SID and try to cover those basic features you will need in order to use SID in most of your programs. Also, to keep things simple, all of our sound programs will use VOICE 1 only.

Even with these limitations, using SID can be a bit complicated. But, we will discuss each of these features one at a time and let you experiment with them and listen to the different sounds. Once you have learned the basics of using SID, you may wish to try your hand at the more-advanced features. *The Commodore 64 Programmer's Reference Guide* (Howard W. Sams & Co., Inc.) gives an excellent description of the SID chip and how to use it.

# 2

## WHICH MEMORY LOCATIONS ARE USED TO CONTROL SID?

You can control SID by using your POKE command. With the right POKES into the area between memory locations 54272 and 54300, you can produce the sounds that you want.

All of the programs that we will use in this section are based on controlling these four factors:

1. Volume
2. Frequency
3. ADSR
4. Waveform

As an example, let's look at volume. You can adjust the volume of your sound by POKEing any number between 0 and 15 into memory location 54296. The higher the number that you POKE, the louder the volume. To adjust the volume to the loudest level, use:

POKE 54296, 15 (Don't enter this.)

Another way to do the same thing is to use:

10 S=54272 (start of SID)

20 POKE S+24, 15 (same as POKE 54296, 15)

(Don't enter these statements either.)

What we have done in the second example is to set S equal to memory location 54272, which is the start of SID. This makes memory location 54296 equal to  $S + 24$ . (That's because  $54272 + 24 = 54296$ .)

In fact, you will see that it is much easier to POKE all of SID's memory locations as  $S +$  (some small number), instead of using the larger memory location number. For example, memory location 54273 is the same as  $S + 1$ , and 54278 is the same as  $S + 6$ . This is the method we will use in all of our sound programs.

Here, then, are the memory locations you will need to POKE in order to produce the sound that you want:



**By POKEing the right memory locations for SID, you can produce the sounds that you want; you can even produce music.**

## To Control

VOLUME  
FREQUENCY  
ADSR  
WAVEFORM

## POKE Memory Location

S + 24  
LOW = S HIGH = S + 1  
AD = S + 5 SR = S + 6  
S + 4

Notice that you have to POKE *two* locations in order to control frequency. There is a LOW location and a HIGH location. Also, notice that you have to POKE two locations to control ADSR. One location controls AD and the other controls SR. More about these later.

### 3

## HOW DO I PRODUCE A SIMPLE TONE?

You can produce a simple tone by using the following program.

### EXAMPLE

Suppose you want your computer to produce a simple tone using VOICE 1. Type NEW and enter this program:

```
10 REM SIMPLE TONE PROGRAM
20 S=54272:REM START OF SID
30 POKE S+24,15:REM VOLUME
40 POKE S,75:POKE S+1,34:REM FREQUENCY
50 POKE S+5,0:POKE S+6,240:REM ADSR
60 WF=32:REM WAVEFORM=SAWTOOTH
70 POKE S+4,WF+1:FOR T=1 TO 800:NEXT:REM ON
80 POKE S+4,WF:REM OFF
```

Type RUN and hit RETURN.

After you run this program, you will hear the tone. Try running it several times.

Although this program does nothing more than produce a short tone, it provides us with a simple example to study and to use for experiments. Let's become familiar with this program by trying to understand the key lines. The REM statements in the program describe what each line does.

Line #30 sets *volume* at loudest level. Line #40 sets a *low frequency* value of 75 and a *high frequency* value of 34 to produce a C note. Line #50 sets ADSR, with AD = 0 and SR = 240, to produce a long (SUSTAIN) tone. Line #60 contains *waveform* #32, which is a sawtooth wave. Line #70 turns the waveform ON and keeps it ON while the loop counts from T = 1 to T = 800. You can hear the tone during this

time. Line #80 turns the waveform OFF so that you cannot hear the tone anymore.

Lines #30, #40, #50, and #60 control volume, frequency, ADSR, and waveform. Each of these will be discussed in detail, so don't worry if you feel a bit confused at this point.

First, let's look at volume. Volume is very easy to understand. You can control it by POKEing memory location  $S + 24$  with *any* number between 0 and 15. The closer the number is to 15, the louder the volume. The closer the number is to 0, the lower the volume.

If you would like to try to experiment with different volumes, change the 15, in line #30, to lower levels such as 10, 5, and 0. Run the simple tone program at each level.

#### **4 HOW CAN I CHANGE THE FREQUENCY OF THE TONE IN MY PROGRAM?**

The tone program that you just ran always plays a C note. This is because the frequency (or pitch) of the tone that it produces is always the same. However, if you want to play songs on the Commodore 64, you will need to learn how to change the frequency so that you can play many different notes.

In our simple tone program, the frequency is determined by line #40 when we POKE a number into location  $S$  and another number into location  $S + 1$ . The number that we POKE into location  $S$  is called the LOW number, and the number that we POKE into location  $S + 1$  is called the HIGH number. Both numbers are needed in order to produce a single note.

Table 12-1 shows you the HIGH and LOW numbers that must be POKED in order to produce different notes.

The words HIGH and LOW can be confusing because they do not describe the size of the number that is being poked. For example, to produce the C note, LOW = 75 and HIGH = 34. In other words, the HIGH number of 34 is lower than the LOW NUMBER of 75. This is perfectly normal. HIGH and LOW refer to different POKE locations and not to the actual size of the number.

Now that you understand how different notes can be played on the Commodore 64, let's try some experiments. First, however, let's change our simple tone program in order to make it easier to input HIGH and LOW numbers.

**Table 12-1. HIGH and LOW Numbers Needed to Produce Certain Notes**

NOTE	LOW (POKE S)	HIGH (POKE S+1)
C	75	34
D	126	38
E	52	43
F	198	45
G	97	51
A	172	57
B	188	64

### EXAMPLE

Suppose you want to change the simple tone program so that you can play different notes. Type NEW and enter the following program:

```

10 REM INPUT FREQUENCY
15 PRINT "{SC}"
20 S=54272:REM START OF SID
30 POKE S+24,15:REM VOLUME
40 INPUT "ENTER FREQ LOW,HI";FL,FH
42 IF FL+FH=0 THEN END
45 POKE S,FL:POKE S+1,FH:REM FREQUENCY
50 POKE S+5,0:POKE S+6,240:REM ADSR
60 WF=32:REM WAVEFORM=SAWTOOTH
70 POKE S+4,WF+1:FOR T=1 TO 800:NEXT:REM ON
80 POKE S+4,WF:REM OFF
100 GOTO 15

```

Type RUN and hit RETURN.

As you run the program, input the LOW and HIGH numbers for each note in Table 12-1. To end the program, enter 0,0 and your program will END in line #42.

## 5

### WHAT IS ADSR AND HOW DO I USE IT?

ADSR is short for ATTACK/DECAY and SUSTAIN/RELEASE. It measures how fast the volume rises and falls.

You can think of a sound wave as having four parts, as shown in Fig. 12-1. During the ATTACK (A) part of the sound wave, the volume rises to its highest level. After reaching its highest level, the



volume falls back a bit during the DECAY (D) part of the wave. After settling back, the sound wave stays at the same volume during the SUSTAIN (SU) portion of the wave. (Note: be careful not to use the variable S for SUSTAIN since S stands for the start of SID.) During the last part of the sound wave, RELEASE (R), the volume fades out until you cannot hear it anymore.



**Fig. 12-1. The four different parts of a sound wave.**

Now for the fun part! You can create many interesting sounds by giving different numbers to A, D, SU, and R. For each one you can use any number between 0 and 15. The closer a number is to 15, the slower the rise or fall in volume or the longer the SUSTAIN portion. Numbers closer to zero produce a faster rise or fall in volume or a shorter SUSTAIN.

For example, to produce a sound with the shortest ATTACK, make  $A = 0$ . This means that your sound will reach its highest volume very quickly. To produce a sound with the longest ATTACK, make  $A = 15$ . This means that your sound will reach its highest volume very slowly. Or, if you wish, you can use any value of A between 0 and 15.  $A = 8$  can be used for medium ATTACK.

DECAY, SUSTAIN, and RELEASE work the same way.  $D = 0$  produces the fastest drop in volume;  $D = 15$  produces the slowest drop in volume.  $SU = 0$  produces the shortest SUSTAIN;  $SU = 15$  produces the longest SUSTAIN.  $R = 0$  produces the fastest RELEASE and the sound dies out quickly.  $R = 15$  produces the longest RELEASE, and the sound dies out very slowly.

Once you have given A, D, SU, and R each a number between 0 and 15, you need to follow these steps in your program.

1. Calculate a combined ATTACK/DECAY number (AD) as follows:

$$AD = (A \times 16) + D$$

2. Calculate a combined SUSTAIN/RELEASE number as follows:

$$SR = (SU \times 16) + R$$

- POKE AD into memory location S + 5 and POKE SR into memory location S + 6. Use:

```
POKE S+5,AD
POKE S+6,SR
```

The best way to appreciate ADSR is to listen to it. The following program is our simple tone program, which has been changed to make it easy to input ADSR.

#### EXAMPLE

Suppose you want to change the simple tone program to make it easy to input values for A, D, SU, and R. Type NEW and enter the following program:

```
10 REM INPUT ADSR
15 PRINT "(SC)"
20 S=54272:REM START OF SID
30 POKE S+24,15:REM VOLUME
40 POKE S,75:POKE S+1,34:REM FREQUENCY
50 INPUT "ENTER A,D,SU,R (0 TO 15)";A,D,SU,R
52 IF A+D+SU+R=0 THEN END
53 AD=(A*16)+D:SR=(SU*16)+R
55 POKE S+5,AD:POKE S+6,SR:REM ADSR
60 WF=32:REM WAVEFORM=SAWTOOTH
70 POKE S+4,WF+1:FOR T=1 TO 1000:NEXT:REM ON
80 POKE S+4,WF:REM OFF
90 GOTO 15
```

Type RUN and hit RETURN. When you run the program, use the five values in Table 12-2 for ADSR.

**Table 12-2. Values for A, D, SU, and R to Produce Interesting Sound Effects**

<b>A</b>	<b>D</b>	<b>SU</b>	<b>R</b>
0	10	0	0
10	0	0	0
0	0	15	0
0	0	15	15
3	6	9	12

Repeat these five values several times. Once you get the “feel” of controlling ADSR, experiment with some of your own numbers and try to create some interesting sound effects. To END the program, enter 0,0,0,0 and the program will end in line #52.

Notice that in line #53, AD and SR are calculated using the preceding formulas.

## **6 WHICH WAVEFORMS CAN BE USED TO CREATE A TONE?**

You can create a tone by using any of four different waveforms. Each of these four waveforms has an identification number (ID#) to identify it.

Fig. 12-2 shows the four different waveforms, the ID# for each one, and the waveform picture.

Waveforms are turned ON and turned OFF. The sound starts just as you turn the waveform ON and ends just as you turn it OFF. To turn ON a waveform:

```
POKE S+4, ID#+1
```

To turn OFF a waveform:

```
POKE S+4, ID#
```

For example, to turn ON and OFF a triangle wave, use:

```
POKE S+4, 17          (turn ON)
```

```
POKE S+4, 16          (turn OFF)
```

To turn ON and OFF a sawtooth wave, use:

```
POKE S+4, 33          (turn ON)
```

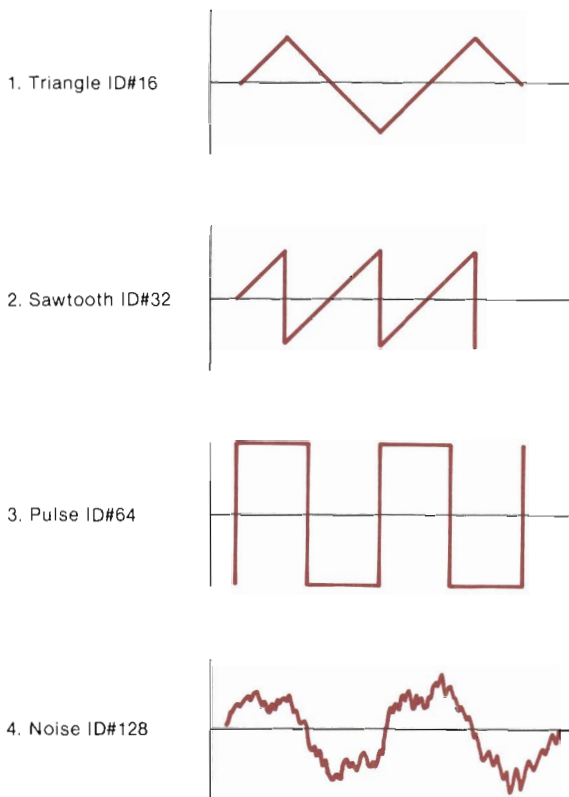
```
POKE S+4, 32          (turn OFF)
```

Once you turn on a waveform, you will probably want to keep it on for a short period of time before you turn it off. You can do this by putting a delay loop between TURN ON and TURN OFF. With this delay loop in place, the previous example of a sawtooth wave would actually look like this:

```
POKE S+4, 33          (turn ON)
```

```
FOR T=1 TO 800:NEXT   (delay)
```

```
POKE S+4, 32          (turn OFF)
```



**Fig. 12-2. Waveforms.**

As before, the best way to understand the different waveforms is to listen to them. In the following example, we have changed our simple tone program to make it easy to input the different waveforms.

#### EXAMPLE

Suppose you want to change the simple tone program to make it easy to hear different waveforms. Type NEW and enter the following program:

```

10 REM INPUT WAVEFORM
15 PRINT "(SC)"
20 S=54272:REM START OF SID
30 POKE S+24,15:REM VOLUME
40 POKE S,75:POKE S+1,34:REM FREQUENCY
50 POKE S+5,0:POKE S+6,240:REM ADSR
60 INPUT "ENTER WAVEFORM T=16 S=32 P=64 N=128";WF

```

```

70 IF WF=0 THEN END
73 IF WF=64 THEN POKE S+3,10:POKE S+2,150:REM PULSE
  WIDTH
75 POKE S+4,WF+1:FOR T=1 TO 800:NEXT:REM ON
80 POKE S+4,WF:REM OFF
90 GOTO 15

```

Type RUN and hit RETURN. As you run the program, input the different ID numbers for each waveform that you want to hear. For example, to hear a sawtooth wave, enter 32. To END the program, enter 0 and the program will end in line #70.

Notice that in line #73 we check for WF = 64 (PULSE waveform). If we use the PULSE waveform, then we must also enter a *pulse width* into S + 2 and S + 3. We do this by using:

```

POKE S+3,10
POKE S+2,150

```

The pulse width controls the timing of the PULSE waveform.

We will not discuss in great detail the setting of the pulse width, other than to say these numbers work pretty well. You might want to experiment with your own pulse widths. If you do, then S + 2 can be POKEd with any number between 0 and 255 and S + 3 with any number between 0 and 15.

## 7 HOW DO I PLAY A SONG ON THE COMMODORE 64?

So far we have used our simple tone program to learn about controlling volume, frequency, ADSR, and waveforms. Now, let's put this knowledge to use and run a program that will play a song for us.

### EXAMPLE

Suppose you want your program to play the song "Jingle Bells." Type NEW and enter the following program:

```

1 REM JINGLE BELLS
3 PRINT "(SC)"
4 INPUT "ENTER WAVEFORM T=16 S=32 P=64 N=128";WF
6 PRINT
7 INPUT "ENTER A,D,SU,R";A,D,SU,R
8 AD=(A*16)+D:SR=(SU*16)+R
9 PRINT:PRINT

```

```

10 S=54272:REM START OF SOUND CONTROL
20 FOR I=S TO S+24:POKE I,0:NEXT:REM CLEAR SOUND CONT
30 POKE S+24,15:REM VOL
40 POKE S+5,AD:REM AD
50 POKE S+6,SR:REM SR
60 READ N$:IF N$="END" THEN END:REM NEXT NOTE
63 GOSUB 500:REM CONVERT NOTE TO FREQ
66 IF N$="P" THEN FOR I=1 TO 100:NEXT:PRINT " ":GOTO 60
68 PRINT N$;
70 POKE S+1,FH:POKE S,FL:REM FREQ HI&LO
75 IF WF=64 THEN POKE S+3,7:POKE S+2,150:REM SET PULSE
    WIDTH
80 POKE S+4,WF+1:FOR T=1 TO 300:NEXT:REM ON
90 POKE S+4,WF:REM OFF
95 GOTO 60
500 REM SUB TO CONVERT NOTE TO FREQ
510 IF N$="A" THEN FH=57:FL=172
520 IF N$="B" THEN FH=64:FL=188
530 IF N$="C" THEN FH=34:FL=75
540 IF N$="D" THEN FH=38:FL=126
550 IF N$="E" THEN FH=43:FL=52
560 IF N$="F" THEN FH=45:FL=198
570 IF N$="G" THEN FH=51:FL=97
600 RETURN
700 DATA E,E,E,P,E,E,E,P
710 DATA E,G,P,C,D,E,P,P
720 DATA F,F,F
730 DATA F,F,E,E,P
740 DATA E,G,G
750 DATA E,D,C
760 DATA END

```

Type RUN and hit RETURN.

Try different waveforms by entering the waveform ID# (16, 32, 64, or 128). Try different values for A, D, SU, and R, and notice the difference in the way the song sounds. Start with some of the values shown in Table 12-2, such as 0,10, 0, 0.

The "Jingle Bells" program in the example is similar to our simple tone program, except for the way the frequencies are handled. In this program, the actual musical notes are read in from the DATA statements in line #60. The subroutine in line #500 converts any note from A to G into the two frequency values needed to produce the note. These values are then poked into S and S + 1 in line #70.

The letter "P" in the DATA statements means that you want to pause between notes. In line #66, if N\$ = "P", then we pause while the FOR-NEXT counts to 100. In line #68, the program prints the note on the screen. In line #66, the program skips to the next line every time it reads a P (for pause).

## 8

**HOW DO I READ IN FREQUENCIES  
FROM DATA STATEMENTS?**

In the "Jingle Bells" program, we read in the name of each note and converted it to a frequency. This method makes it easy to enter a song. At times, however, you may want to enter frequencies directly from the DATA statements. This is easy to do, as the next example shows.

**EXAMPLE**

Suppose you want to play some notes by reading in their frequencies directly from DATA statements. Type NEW and enter the following program:

```
1 REM FREQ IN DATA STATEMENTS
3 PRINT "{SC}"
4 INPUT "ENTER WAVEFORM T=16 S=32 P=64 N=128";WF
10 S=54272:REM START OF SID
20 FOR I=S TO S+24:POKE I,0:NEXT:REM CLEAR SID
30 POKE S+24,15:REM VOL
40 POKE S+5,72:REM AD
50 POKE S+6,40:REM SR
55 PRINT:PRINT "LOW","HIGH":PRINT
60 READ FL,FH:IF FL<0 THEN END:REM NEXT NOTE
68 PRINT FL,FH
70 POKE S,FL:POKE S+1,FH:REM FREQ HI&LO
75 IF WF=64 THEN POKE S+3,7:POKE S+2,150:REM SET PULSE
   WIDTH
80 POKE S+4,WF+1:FOR T=1 TO 300:NEXT:REM ON
90 POKE S+4,WF:REM OFF
95 GOTO 60
700 DATA 75,34
710 DATA 126,38
720 DATA 52,43
730 DATA 198,45
740 DATA 97,51
750 DATA 172,57
760 DATA 188,64
770 DATA -1,-1
```

Type RUN and hit RETURN.

Run this program several times with different waveforms. You will hear the notes C, D, E, F, G, A, B; and on the screen you will see the LOW and HIGH frequency number for each note.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. You can produce different types of sounds by controlling volume, frequency, ADSR, and waveform. *Volume* controls loudness. *Frequency* controls pitch. *ADSR* controls how fast the volume rises and falls. *Waveform* controls the shape of the wave that produces the sound.
2. A sound waveform has four parts: ATTACK, DECAY, SUSTAIN, and RELEASE. These are called ADSR for short. During ATTACK, volume rises to its highest level. During DECAY, volume falls back a bit. During SUSTAIN, volume remains at the same level. During RELEASE, volume fades out until you cannot hear the sound any longer.
3. Four different types of waveforms can be produced: *triangle*, *sawtooth*, *pulse*, and *noise*.





## PRINT AND POKE GRAPHICS

In today's lesson you will learn:

- How to produce PRINT GRAPHICS and POKE GRAPHICS.
- How to animate your graphics.
- How to control your graphics with a joystick.
- How to create your own arcade-type video game.



# 1

## WHAT ARE THREE WAYS TO PRODUCE GRAPHICS ON THE COMMODORE 64?

You can produce graphics on the Commodore 64 in three ways:

1. You can use the PRINT command to print graphics characters on the screen. We will call this method PRINT GRAPHICS.
2. You can use the POKE command to display graphics characters in any location on the screen. We will call this POKE GRAPHICS.
3. You can use SPRITES, which are large graphics characters that the computer creates from your own description of the character. We will call this method SPRITE GRAPHICS.

This lesson will cover PRINT GRAPHICS and POKE GRAPHICS. SPRITE GRAPHICS will be covered in the next lesson.






**Important Note:** This lesson contains many examples of programs that produce graphics on the screen. These graphics can be seen best on an all-white screen. To turn the screen all white, enter:

```
POKE 53280,1: POKE 53281,1 (hit RETURN)
```

Use the all-white screen throughout the entire lesson.

# 2

## HOW DO I PRODUCE PRINT GRAPHICS BY USING THE KEYBOARD GRAPHICS CHARACTERS?

You probably have noticed by now that many of the keys on the keyboard have two graphics characters drawn on the side of the key. For example, on the Z key, the two characters are  and . To display the , hold down SHIFT and press Z. To display the , hold down  and press Z.

Now, let's see how these keyboard graphics characters can be displayed by using the PRINT command. To print five diamonds, type NEW and enter the following program:

```
10 PRINT "{SC}"  
20 PRINT "◆◆◆◆◆":REM DIAMOND =SHIFT Z
```


Type RUN and hit RETURN.


If you want to PRINT these diamonds at a certain spot on the

screen, then you can add cursor movements. Also, if you want your diamonds to appear in color, you can use your color keys. To display five red diamonds starting in the third column of the fourth row, type NEW and enter this program:

```
10 PRINT "{SC}"
20 PRINT "{RD}"
30 PRINT "{CR}{CR}{CD}{CD}{CD}♦♦♦♦♦":REM DIAMOND
   = SHIFT Z
```

{RD} = **CTRL** RED




{CR} = 

{CD} = 


Type RUN and hit RETURN.

### EXAMPLE

Suppose you want to PRINT the picture of a plane by using the following keyboard graphics characters:

Part of Plane	Character	Key Used
Tail	>	>
Body		SHIFT C
Nose	♦	SHIFT Z
High wing	REVERSE 	RVS/ON <b>Ctrl</b> *
Low wing		SHIFT £

Type NEW and enter the following program:

```
10 REM DRAW PLANE USING KEY CHARACTERS
20 PRINT "{SC}"
30 A$=" {RV}␣{RO} " :REM SPACE/RVS-ON/CMDB */RVS-OFF/
   SPACE
40 B$=">-♦":REM >/SHIFT C/SHIFT Z
50 C$="  " :REM SPACE/SHIFT £/SPACE
60 PRINT A$
70 PRINT B$
80 PRINT C$
```

{RV} = **CTRL** RVS/ON

{RO} = **CTRL** RVS/OFF

Type RUN and hit RETURN.

### 3 HOW DO I PRODUCE PRINT GRAPHICS BY USING CHR\$?

The keyboard graphics characters that you see on the keys can be printed on the screen by using the CHR\$ command. Before, when we used the keyboard graphics characters, we printed a diamond character by using the statement:

```
PRINT " ♦ " ( ♦ = SHIFT Z)
```

You can do exactly the same thing by using the statement:

```
PRINT CHR$(122)
```

This works because 122 is the code for the diamond character. Each character that can be printed on the Commodore 64 has such a code. Look at Appendix A. It gives codes for all letters and numbers, all the graphics characters, cursor movements, colors, and CLR/HOME (the command to clear the screen).

Now, let's repeat one of the examples we just did in Question #2, but let's do it by using CHR\$ instead of the keyboard graphics characters. Here are the codes you will need to know:

Command	Code
Cursor right	29
Cursor down	17
Diamond	122
Color red	28
CLEAR/HOME	147

To display five red diamonds, starting in the third column of the fourth row, type NEW and enter this program:

```
10 PRINT CHR$(147):REM CLR/HOME
20 PRINT CHR$(28)
30 D$=CHR$(122):CR$=CHR$(29):CD$=CHR$(17)
40 PRINT CR$;CR$;CD$;CD$;CD$;D$;D$;D$;D$;D$
```

Type RUN and hit RETURN.

#### EXAMPLE

Suppose you want to draw a picture of a plane by using CHR\$.

Plane Part	Character	Code
Tail	>	62
Body	☐	96
Nose	◆	122
High wing	REVERSE ☐	127
Low wing	◼	169
REVERSE ON		18
REVERSE OFF		146
SPACE		32

Type NEW and enter the following program:

```

10 REM DRAW PLANE USING CHR$ CODES
20 PRINT CHR$(147):REM CLEAR SCREEN
30 A$=CHR$(32)+CHR$(18)+CHR$(127)+CHR$(146)+CHR$(32):
  REM HIGH WING
40 B$=CHR$(62)+CHR$(96)+CHR$(122):REM BODY OF PLANE
50 C$=CHR$(32)+CHR$(169)+CHR$(32):REM LOW WING
60 PRINT A$
70 PRINT B$
80 PRINT C$

```

Type RUN and hit RETURN.

## 4 HOW DO I PRODUCE POKE GRAPHICS?

POKE GRAPHICS are produced by poking codes into certain locations of memory. The codes that we POKE to produce POKE GRAPHICS *are not* the same as the CHR\$ codes we just used. The codes used with POKE GRAPHICS are called SCREEN CODES and are shown in Appendix B. For example,

The CHR\$ code for diamond is 122  
 The SCREEN CODE for diamond is 90

Now, suppose we want to use POKE GRAPHICS to display a *red diamond* in the upper left corner of the screen. To do this, type NEW and enter the following program:

```

10 PRINT "(SC)"
20 POKE 1024,90:REM CODE FOR DIAMOND IS 90
30 POKE 55296,2:REM CODE FOR RED IS 2

```

Type RUN and hit RETURN.

In line #20, we poked SCREEN CODE 90 (which draws a diamond)

into memory location 1024 (which is the memory location that tells the computer what to display in the upper left corner of the screen). In line #30, we poked a 2 into memory location 55296. This memory location tells the computer which color to make the character that we poked into memory location 1024.

At this point, you are probably thinking to yourself, "How am I ever going to remember all these memory locations?" Actually, it's not as difficult as it may seem because there are a few tricks to make your job easier. Here is one way to make things simpler—write your program like this next one. Type NEW and enter:

```
10 PRINT "(SC)"
15 SM=1024
20 POKE SM,90:REM CODE FOR DIAMOND IS 90
30 POKE SM+54272,2:REM CODE FOR RED IS 2
```

Type RUN and hit RETURN.

Notice that the memory location that controls the color of 1024 is 55296. In line #30, we used POKE SM + 54272,2 instead of POKE 55296,2. Since in line #15 we make SM = 1024, then POKE SM + 54272,2 and POKE 55296,2 do exactly the same thing (since  $1024 + 54272 = 55296$ ). The variable SM stands for SCREEN MEMORY.

In fact, the memory location that controls a character's color is *always* higher by 54272\*. The number 54272 is therefore a very important number to remember and will be used in *all* examples of POKE GRAPHICS. By using SM + 54272, we do not need to figure out the memory location that controls color; we just add 54272 to whatever memory location holds our poked SCREEN CODE.

To see how the COLOR MEMORY location can be found by using 54272, change line #15 to SM = 1063 and run the program again. This time the diamond will appear in the upper right corner. Its color will still be red because in line #30 we poked a 2 into memory location  $1063 + 54272$ .

## EXAMPLE

Suppose you want to draw the picture of a plane, using POKE GRAPHICS. Here are the SCREEN CODES you will need to know:

---

\*NOTE: Unless you take special action to change it, the memory location that controls a character's color will always be higher by 54272. Experienced programmers sometimes change this number, but beginners are not advised to do so and the procedure will not be covered in this book.

Plane Part	Character	Screen Code
Tail	>	62
Body	☐	67
Nose	◆	90
High wing	REVERSE ☐	95 + 128
Low wing	◼	105

Notice that the SCREEN CODE for the high wing is 95. To reverse this character, or any other, add 128 to the SCREEN CODE.

Now type NEW and enter the following program:

```

10 REM DRAW PLANE USING POKE
20 PRINT "(SC)"
30 PL=1164:REM STARTING LOCATION OF PLANE
40 POKE PL,90:POKE PL-1,67:POKE PL-1-40,95+128:POKE PL-1
  +40,105:POKE PL-2,62
50 CM=PL+54272
60 POKE CM,0:POKE CM-1,0:POKE CM-1-40,0:POKE CM-1+40,0:
  POKE CM-2,0

```

Type RUN and hit RETURN.

## 5 WHAT ARE SCREEN MEMORY AND COLOR MEMORY?

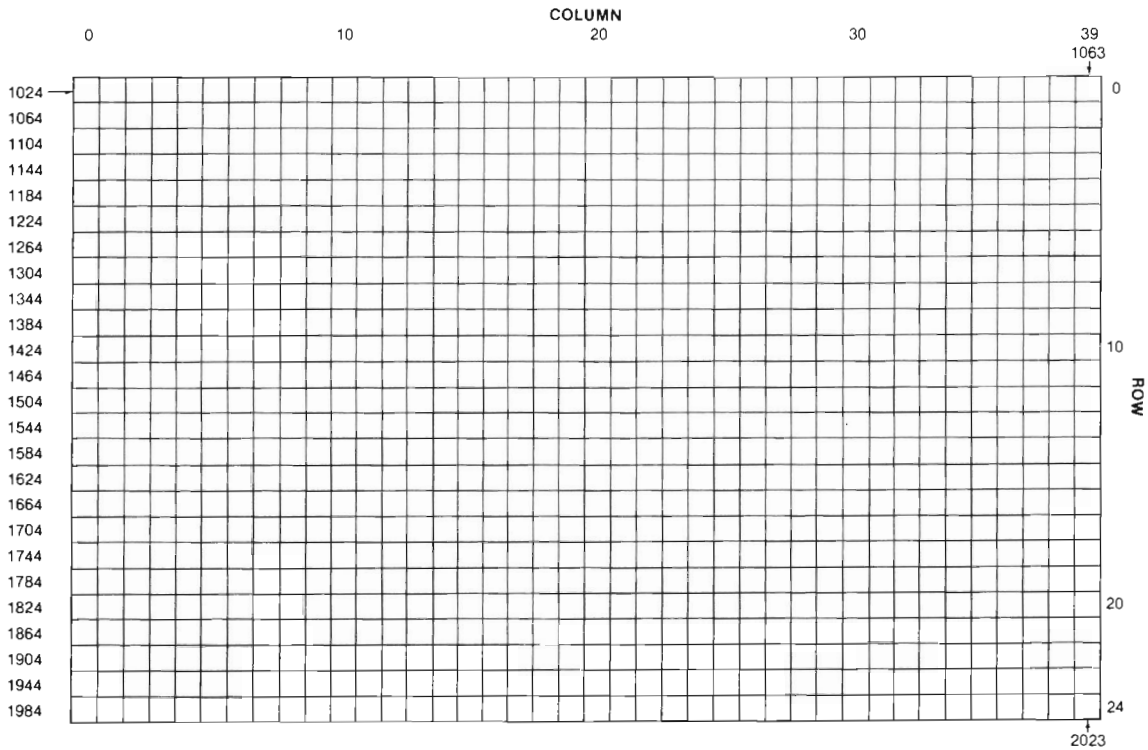
In Question #4, we used POKE GRAPHICS to draw a red diamond in the upper left corner of the screen. We did this by poking a 90 (the SCREEN CODE for the diamond character) into memory location 1024 and then poking a 2 (the COLOR CODE for red) into memory location 55296.

The memory location 1024 (the one we poked for the diamond character) is in an area of memory called SCREEN MEMORY. SCREEN MEMORY consists of the 1000 memory locations between 1024 and 2023. If you want a certain character to appear anywhere on the screen, then you can POKE its SCREEN CODE into SCREEN MEMORY. *Where* your character will appear on the screen depends on where in SCREEN MEMORY, between locations 1024 and 2023, you POKE your SCREEN CODE.

There is another part of memory, called COLOR MEMORY, which controls the color of the characters on the screen. COLOR MEMORY consists of the 1000 memory locations between 55296 and 56295. If you want your character to be a certain color, then you can POKE its COLOR CODE into the correct COLOR MEMORY location. The COLOR MEMORY location is *always* the SCREEN MEMORY location + 54272.

A diagram of SCREEN MEMORY is shown in Fig. 13-1. A diagram

Fig. 13-1. SCREEN MEMORY map.





of COLOR MEMORY is shown in Fig. 13-2. Both diagrams are arranged like the screen (40 columns by 25 rows). This makes it easy to relate the location in SCREEN MEMORY and COLOR MEMORY to the actual position on the screen. Also, notice that corresponding locations in SCREEN MEMORY and COLOR MEMORY are separated by 54272.

SCREEN CODES are poked into SCREEN MEMORY. Appendix B lists these SCREEN CODES. SCREEN CODE numbers range between 0 and 255. If you use a SCREEN CODE between 0 and 127, the character will appear on the screen in its normal form. If you want that same character to appear reversed (the same way it does with REVERSE ON), then add 128 to the SCREEN CODE. For example, the SCREEN CODE for A is 1, while the SCREEN CODE for a reversed A is 1 + 128, or 129.

COLOR CODES are poked into COLOR MEMORY. The Commodore 64 can display a character in any of 16 colors. These colors and their color codes are given below and again in Appendix C.

<b>Color</b>	<b>Color Code</b>	<b>Color</b>	<b>Color Code</b>
Black	0	Orange	8
White	1	Brown	9
Red	2	Light Red	10
Cyan	3	Gray 1	11
Purple	4	Gray 2	12
Green	5	Light Green	13
Blue	6	Light Blue	14
Yellow	7	Gray 3	15

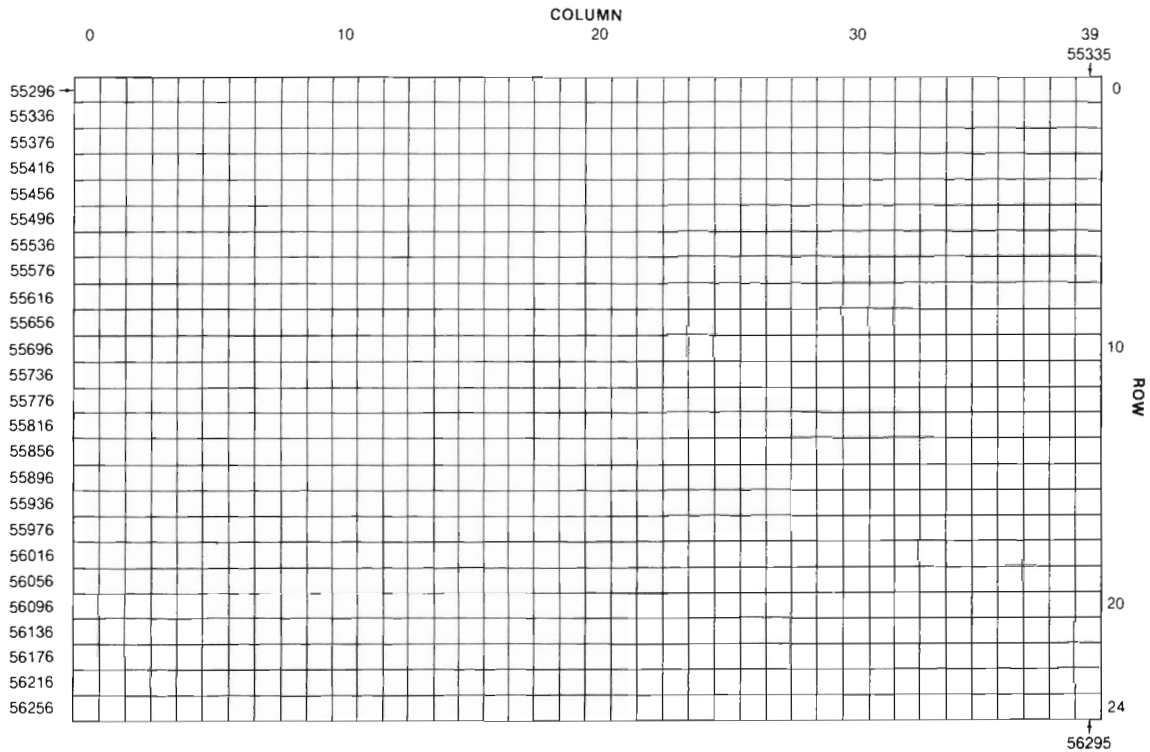
## **6 HOW CAN I USE POKE GRAPHICS TO DRAW LARGE PICTURES ON THE SCREEN?**

Up to now, we have drawn all our pictures by POKEing graphics characters, one at a time. This method is all right as long as you are working with small pictures, like the plane. But, to draw a large picture in this way would require more time and patience than most of us have.

Fortunately, two BASIC commands are very helpful in drawing large pictures:

1. FOR-NEXT
2. READ

Fig. 13-2. COLOR MEMORY map.



Look back at Fig 13-1. Suppose you wanted to draw a horizontal solid green line between 1154 and 1174. The next program shows you an easy way to do this, using a FOR-NEXT loop. (The solid line is drawn with a reversed-space character whose SCREEN CODE is 32 + 128.) Type NEW and enter this program:

```
10 PRINT "(SC)"
20 FOR SM=1154 TO 1174
30 CM=SM+54272
40 POKE SM,(32+128):POKE CM,5
50 NEXT SM
```

Type RUN and hit RETURN.

Next, let's draw a vertical blue line between 1074 and 1954, by using this program. Type NEW and enter:

```
10 PRINT "(SC)"
20 FOR SM=1074 TO 1954 STEP 40
30 CM=SM+54272
40 POKE SM,(32+128):POKE CM,6
50 NEXT SM
```

Type RUN and hit RETURN.

Notice the STEP 40 in line #20. From Fig 13-1, you can see that to draw a vertical line you have to POKE your character into every 40th position on the screen.

Now, suppose you wanted to draw three vertical lines made up of the diamond character. These lines are to run from 1154 to 1954, from 1164 to 1964, and from 1174 to 1974. Here is a case where the READ command can be used to bring in these values from DATA statements, as shown in the next program. Type NEW and enter:

```
10 PRINT "(SC)"
20 READ F,T:IF F<0 THEN END
30 FOR SM=F TO T STEP 40
40 CM=SM+54272:REM COLOR MEMORY LOCATION
50 POKE SM,90:POKE CM,2:REM DRAW
60 NEXT SM
70 GOTO 20
80 DATA 1154,1954
90 DATA 1164,1964
100 DATA 1174,1974
110 DATA -1,-1
```

Type RUN and hit RETURN.

## EXAMPLE

Suppose you want to draw several large faces on the screen using SCREEN CODES 81 through 95 and you want

to use different colors for each face. Type NEW and enter the following program:

```
3 REM DRAW MANY DIFFERENT FACES
4 COL=1
5 PRINT "(SC)":POKE 53280,1:POKE 53281,1
6 FOR CH=81 TO 95:RESTORE:GOSUB 10:NEXT CH:END
10 COL=COL+1
20 READ F,T:IF F<0 THEN 70:REM FROM-TO NUMBERS
30 FOR X=F TO T
40 POKE X,CH:POKE X+54272,COL:REM HORIZONTAL LINES
50 NEXT X
60 GOTO 20
70 FOR I=1234 TO 1824 STEP 40
75 POKE I,CH:POKE I+54272,COL:REM LEFT VERTICAL LINE
80 NEXT I
85 FOR I=1254 TO 1854 STEP 40
90 POKE I,CH:POKE I+54272,COL:REM RIGHT VERTICAL LINE
95 NEXT I
100 DATA 1234,1254
110 DATA 1358,1359,1369,1370
120 DATA 1398,1399,1409,1410
130 DATA 1484,1485,1524,1525
140 DATA 1638,1639,1649,1650
150 DATA 1678,1690,1718,1730
160 DATA 1834,1854
170 DATA -1,-1
200 RETURN
```

Type RUN and hit RETURN.

## 7 HOW CAN I CREATE ANIMATION ON THE SCREEN?

An animated drawing is one that appears to move. To animate your drawings on the screen, you:

1. Draw your picture on the screen.
2. Next, you erase it.
3. Then, you draw it again in another spot on the screen, a short time later.

The following program is a very simple example of animation. When you RUN this program, you will see a ball move across the screen from left to right. Type NEW and enter:

```
10 REM SIMPLE EXAMPLE OF ANIMATION
20 PRINT "(SC)"
30 SM=1504:REM STARTING LOCATION
40 FOR I=3 TO 39
50 POKE SM,32:REM ERASE
```

```

60 SM=1504+I:REM NEW LOCATION
70 POKE SM,81:REM DRAW IN NEW LOCATION
80 CM=SM+54272:REM CALC LOCATION IN COLOR MEM
90 POKE CM,2:REM COLOR CODE 2=RED
100 FOR K=1 TO 70:NEXT K:REM SLOW DOWN
110 NEXT I

```

Type RUN and hit RETURN.

In line #30, SM is the SCREEN MEMORY location of the ball; the starting location is 1504. Line #50 erases the ball by POKING SCREEN CODE 32 (blank space) into SM. In line #60, SM is increased by 1. This is the new location of the ball. Line #70 draws the ball (SCREEN CODE = 81) in the new location. In line #80, CM is the COLOR MEMORY location that controls the color of the ball. In line #90, COLOR CODE 2 (red) is poked into CM. We use a loop in line #100 to slow down the movement of the ball.

Now, let's use the same method to draw a plane moving across the screen. This program will be similar to the previous one, except we have to draw and erase each of the five graphics symbols that make up the plane. Type NEW and enter the following program:

```

5 REM MOVING PLANE
10 PRINT "(SC)"
15 PL=1144:REM STARTING LOC OF PLANE
20 FOR I=3 TO 40
28 POKE PL,32:POKE PL-1,32:POKE PL-1-40,32:POKE PL-1+40,
  32:POKE PL-2,32
29 PL=1144+I:REM NEW LOCATION
30 POKE PL,90:POKE PL-1,67:POKE PL-1-40,95+128:POKE PL-1
  +40,105:POKE PL-2,6:
40 CM=PL+54272:REM CALC LOC IN COL MEM
42 POKE CM,0:POKE CM-1,0:POKE CM-1-40,0:POKE CM-1+40,0:
  POKE CM-2,0
60 NEXT I
70 GOTO 20
80 END

```

Type RUN and hit RETURN. Notice that we draw each graphics character in line #30, color it black in line #42, and erase it in line #28.

## EXAMPLE

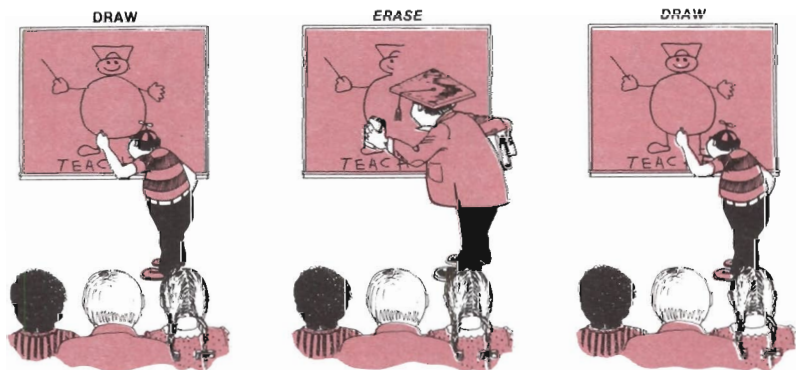
Suppose you would like to create an arcade game. In this game, a plane flies across the screen while you try to shoot it down. You already know how to create the moving plane. What you need now is some way to draw a picture of a gun that fires straight up. When the bullet hits the plane, you want the computer to print the word "HIT." Type NEW and enter the following program:

```

3 REM DRAW GUN SHOOTING PLANE
5 POKE 53280,1:POKE 53281,1:REM WHITE SCREEN & BORDER
10 PRINT "(SC)"
15 PL=1161:REM STARTING LOC OF PLANE
30 POKE PL,90:POKE PL-1,67:POKE PL-1-40,95+128:POKE
  PL-1+40,105:POKE PL-2,6:
40 CM=PL+54272
42 POKE CM,0:POKE CM-1,0:POKE CM-1-40,0:POKE CM-1+40,0:
  POKE CM-2,0
490 REM DRAW GUN
500 G=2000
501 POKE G,32:POKE G+1,32:POKE G-1,32:POKE G-40,32
520 POKE G,102:POKE G-1,102:POKE G+1,102:POKE G-40,90
530 GC=G+54272
540 POKE GC,2:POKE GC-1,2:POKE GC+1,2:POKE GC-40,2
600 REM FIRE GUN
610 FOR Y=G-80 TO 1084 STEP -40
620 POKE Y,65:POKE Y+54272,2:POKE Y,32
625 IF PEEK(Y-40)<>32 THEN PRINT " ", " ", " ", "HIT"
630 NEXT Y
640 END

```

Type RUN and hit RETURN. Notice line #620, which is used to draw the moving bullet. In this line, we draw the bullet (SCREEN CODE = 65), color it red, and erase it by POKE-ing a blank space (SCREEN CODE = 32).



To create animation, draw your picture on the screen, erase it, and draw it again.

## 8

### HOW CAN I USE THE JOYSTICK TO MOVE OBJECTS ON THE SCREEN?

Video games are a lot more fun if you can use a joystick to move objects on the screen. Next is a simple program that will show you

how a joystick works. If you have a joystick, plug it into PORT 2 (nearest the ON/OFF switch). Then type NEW and enter the following program:

```
10 REM JOYSTICK VALUES
20 J=PEEK(56320) AND 15:FB=PEEK(56320) AND 16
30 PRINT J,FB
40 GOTO 20
```

Type RUN and hit RETURN.

You should now see two columns of numbers being printed on the screen. The first is the value of the variable J, which tells you the position of the joystick. When the joystick is straight up, the value of J is 15. If you move the joystick forward (north), then J = 14. If you move the joystick toward you (south), then J = 13, etc. The second column is the value of the variable FB, which tells you whether the FIRE BUTTON is up or being pressed down. When it is up, FB = 16. When you press the FIRE BUTTON down, FB = 0.

Fig. 13-3 shows the values of J and FB at various positions. To get the value of J, we calculate:

```
J=PEEK ( 56430 ) AND 15
```

Since the binary number for 15 is 00001111, PEEK (56320) AND 15 gives us the value of the *first half* (first four bits) of memory location 56320. But, this is exactly what we need, since the value of the first half is what indicates the joystick's position.

To get the value of FB, we calculate:

```
FB=PEEK ( 56320 ) AND 16
```

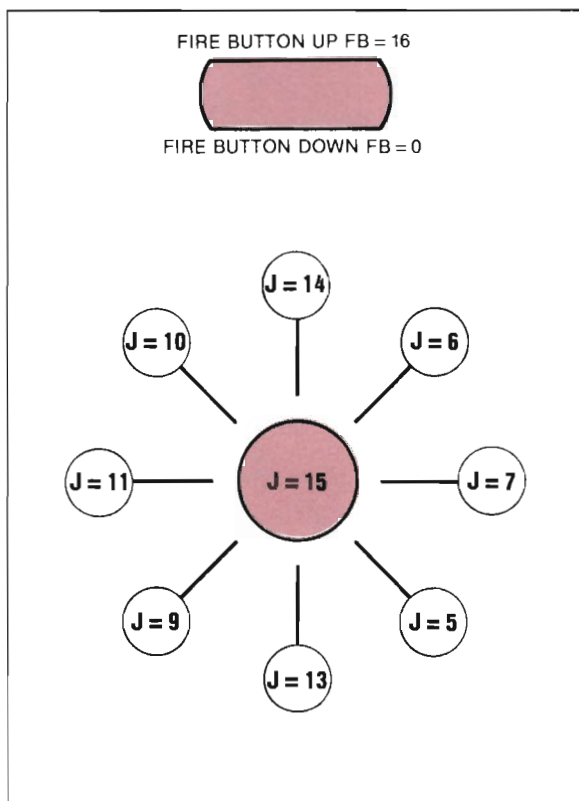
Since the binary number for 16 is 00010000, PEEK (56320) AND 16 gives us the value of only the fifth bit of memory location 56320. Again, this is exactly what we need, since the position of the fifth bit of memory location 56320 indicates the position of the FIRE BUTTON.

With our joystick plugged into PORT 2, we have used memory location 56320 to tell us the position of the joystick and FIRE BUTTON. If we switch the joystick to PORT 1 (nearest the front of the keyboard), then we must use J = PEEK (56321) AND 15 and FB = (56321) AND 16.

## EXAMPLE

Suppose you want to draw a gun that can be used in an arcade game. You want to be able to move the gun left and right with the joystick and to use the FIRE BUTTON to fire. Type NEW and enter the following program:





**Fig. 13-3. Values of J and FB for various positions of the joystick and FIRE BUTTON.**

```

5 POKE 53280,1:POKE 53281,1:REM WHITE SCREEN & BORDER
10 PRINT "(SC)"
490 REM DRAW GUN
500 G=2000:REM ORIG POSITION OF GUN
501 POKE G,32:POKE G+1,32:POKE G-1,32:POKE G-40,32
510 GOSUB 800:REM MOVE GUN
520 POKE G,102:POKE G-1,102:POKE G+1,102:POKE G-40,90
530 GC=G+54272
540 POKE GC,2:POKE GC-1,2:POKE GC+1,2:POKE GC-40,2
550 IF FB=0 THEN GOSUB 600
560 GOTO 501
600 REM FIRE GUN
610 FOR Y=G-80 TO 1084 STEP -40
620 POKE Y,65:POKE Y+54272,2:POKE Y,32
625 IF PEEK(Y-40)<>32 THEN PRINT " ", " ", "HIT"
630 NEXT Y

```



```

640 RETURN
800 REM MOVE GUN
810 J=PEEK(56320) AND 15:FB=PEEK(56320) AND 16
820 IF J=7 THEN G=G+2:IF G>2020 THEN G=2020
830 IF J=11 THEN G=G-2:IF G<1990 THEN G=1990
840 RETURN

```

Type RUN and hit RETURN.

## 9 HOW CAN I USE POKE GRAPHICS TO CREATE AN ARCADE GAME?

You can create your own arcade-type video game by putting together the ideas learned in this lesson.

### EXAMPLE

Suppose you want to create an arcade-type video game. In this game, a plane moves across the screen from left to right. You have a joystick-controlled gun that will shoot at the plane whenever you press the FIRE BUTTON.

If you hit the front or the rear of the plane, then you earn 100 points. If you hit the middle of the plane, you earn 300 points. The object of the game is to reach 1,500 points with as few shots as possible. Type NEW and enter the following program:

```

3 REM ARCADE GAME
5 POKE 53280,1:POKE 53281,1:REM WHITE SCREEN & BORDER
10 PRINT "(SC)"
12 PRINT "(CD)(CD)(CD)(CD)":REM MOVE DOWN 4 LINES FOR
SCORE
15 G=2004:PL=1144:REM START LOC OF GUN AND PLANE
20 FOR I=3 TO 40
22 GOSUB 500:REM DRAW AND MOVE GUN
25 REM DRAW AND MOVE PLANE
28 POKE PL,32:POKE PL-1,32:POKE PL-1-40,32:POKE PL-1+40,
32:POKE PL-2,32
29 PL=1144+I
30 POKE PL,90:POKE PL-1,67:POKE PL-1-40,95+128:POKE PL-1
+40,105:POKE PL-2,82
40 CM=PL+54272
42 POKE CM,0:POKE CM-1,0:POKE CM-1-40,0:POKE CM-1+40,0:
POKE CM-2,0
45 FB=PEEK(56320)AND 16:REM CHECK FIRE BUTTON
46 IF FB=0 THEN GOSUB 600:REM FIRE BUTTON IS DOWN
60 NEXT I
70 GOTO 0
500 REM DRAW GUN
501 POKE G,32:POKE G+1,32:POKE G-1,32:POKE G-40,32

```

```

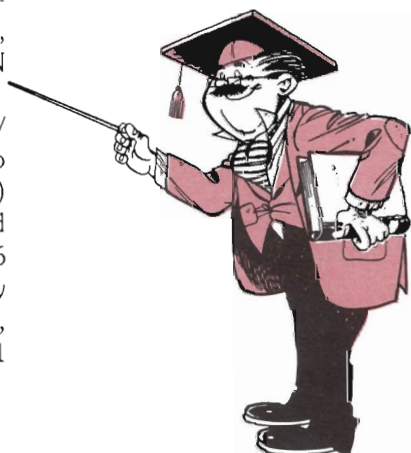
510 GOSUB 800:REM SEE IF GUN MOVED
520 POKE G,102:POKE G-1,102:POKE G+1,102:POKE G-40,90
530 GC=G+54272
540 POKE GC,2:POKE GC-1,2:POKE GC+1,2:POKE GC-40,2
550 RETURN
600 REM FIRE GUN
605 SF=SF+1:REM COUNT SHOTS FIRED
610 FOR Y=G-80 TO 1084 STEP -40
620 POKE Y,65:POKE Y+54272,2:POKE Y,32
625 IF PEEK(Y-40)<>32 THEN H=H+100:PRINT H:PRINT "(CU)";
628 IF H=1500 THEN 900
630 NEXT Y
640 RETURN
800 REM MOVE GUN
810 J=PEEK(56320) AND 15:FB=PEEK(56320) AND 16
820 IF J=7 THEN G=G+2:IF G>2020 THEN G=2020:REM MOVE GUN
RIGHT
830 IF J=11 THEN G=G-2:IF G<1990 THEN G=1990:REM MOVE GUN
LEFT
840 RETURN
900 PRINT "*** GAME OVER ***"
910 PRINT:PRINT "SHOTS FIRED=";SF
999 END

```

Type RUN and hit RETURN.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

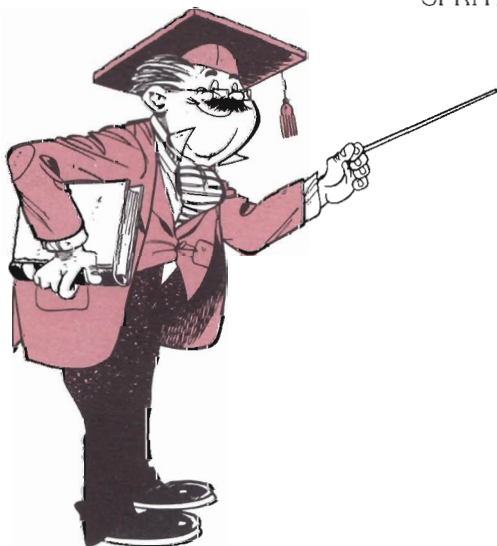
1. If you PRINT a graphics character, use the keyboard key for that character or the CHR\$ code. If you POKE a graphics character, use the SCREEN CODE.
2. To find the COLOR MEMORY location of your character, add 54272 to its SCREEN MEMORY location.
3. To find the position of a joystick/FIRE BUTTON plugged into PORT 2, use: J = PEEK (56320) AND 15 for the joystick and FB = PEEK (56320) AND 16 for the fire button. If the joystick is plugged into PORT 1, then use PEEK location 56321 instead of 56320.



## INTRODUCTION TO SPRITE GRAPHICS

In today's lesson you will learn:

- How to create and display a SPRITE.
- How to change a SPRITE's color, size, and position on the screen.
- How to create animation using SPRITE GRAPHICS.



**Reminder:** This lesson contains several examples of programs that produce graphics on the screen. These graphics can be seen best on an all-white screen. To turn the screen all-white, enter:

```
POKE 53280,1:POKE 53281,1 (hit RETURN)
```

Use the all-white screen throughout this entire section. Also, for reasons already discussed, we suggest that you save each SPRITE program *before* you RUN it.

## 1 WHAT IS A SPRITE?

A *SPRITE* is a large graphics character that the computer creates from your own description. SPRITES are very useful in programming video games because the computer does a lot of the work required to draw the SPRITE and to move it around the screen.

The best way to understand a SPRITE is to see one. For a demonstration, type NEW and enter SPRITE PROGRAM #1:

```
5 REM SPRITE PROGRAM #1 (SQUARE FACE)
10 PRINT "(SC)"
20 GOSUB 500:REM READ IN SHAPE DESCRIPTION
25 REM -----
30 REM SPRITE CONTROL
40 V=53248:REM START OF VIDEO
50 POKE V+39,2:REM COLOR
55 POKE V+23,0:REM EXPAND VERTICAL
58 POKE V+29,0:REM EXPAND HORIZONTAL
60 POKE V+21,1:REM TURN ON SPRITE
70 POKE V+1,100:REM Y POSITION
80 POKE V+16,0:POKE V,150:REM X POSITION
90 END
99 REM -----
500 REM READ IN SPRITE DESCRIPTION
505 POKE 2040,13:REM POINTER TO SPRITE DESCRIPTION
510 FOR N=0 TO 62
520 READ SD:POKE 832+N,SD
530 NEXT N
540 RETURN
695 REM -----
700 REM SPRITE DESCRIPTION
710 DATA 255,255,255
720 DATA 128,0,1
730 DATA 128,0,1
740 DATA 128,0,1
750 DATA 128,0,1
760 DATA 128,0,1
770 DATA 128,0,1
780 DATA 131,0,193
```

```
790 DATA 131,0,193
800 DATA 128,0,1
810 DATA 128,0,1
820 DATA 128,24,1
830 DATA 128,24,1
840 DATA 140,0,49
850 DATA 140,0,49
860 DATA 140,0,49
870 DATA 143,255,241
880 DATA 143,255,241
890 DATA 128,0,1
900 DATA 128,0,1
910 DATA 255,255,255
```

Type RUN and hit RETURN. On the screen you should see one red SPRITE in the shape of a smiling square face.

Actually, up to eight SPRITES can be displayed on the screen at one time. However, the program needed to do this can be long and complicated. To keep things simple in this example, we used just a single SPRITE whose only purpose was to introduce you to the subject of SPRITES. *The Commodore 64 Programmer's Reference Guide* (Howard W. Sams & Co., Inc.) gives a detailed explanation of SPRITES and should be very useful to those who wish to learn more about SPRITE GRAPHICS.

Before we go on, let's turn off the SPRITE on the screen so that it will not interfere when you LIST a program. To turn off the SPRITE, enter:

```
POKE V+21,0 (hit RETURN)
```

Should you want to turn it ON for some reason, use:

```
POKE V+21,1 (hit RETURN)
```

Note that these commands will work only after a SPRITE program has been run.

## **2 HOW DO I WRITE A PROGRAM THAT DISPLAYS A SPRITE ON THE SCREEN?**

Look carefully at the preceding SPRITE PROGRAM #1, which displayed the red SPRITE on the screen. Although this program may look a bit complicated, it can be broken up into three sections. These sections are separated by dashes (lines #25, #99, and #695).

In order to write a program that displays a SPRITE on the screen, you will have to learn how each of these sections works. Here is what each section does in the program:

SPRITE DESCRIPTION (lines #700-#910) contains code numbers that describe to the computer how the SPRITE should look.

READ IN SPRITE DESCRIPTION (lines #500-#540) reads the SPRITE DESCRIPTION into the computer's memory. In line #505, a pointer tells the computer where the SPRITE DESCRIPTION is located in memory.

SPRITE CONTROL (lines #30-#90) controls the SPRITE's color, size, and position on the screen.

### 3

## HOW DO I CREATE THE SPRITE DESCRIPTION?

The first step in writing a SPRITE program is to create the SPRITE DESCRIPTION, which tells the computer how the SPRITE should look. You can do this in three steps:

1. Draw a picture of your SPRITE on a SPRITE DESCRIPTION CHART. Fig. 14-1 shows how this was done for SPRITE PROGRAM #1. (Blank SPRITE DESCRIPTION CHARTS are provided in this book.)
2. Using the SPRITE DESCRIPTION CHART, calculate the code numbers that will describe the SPRITE to the computer.
3. Transfer these code numbers to DATA statements within your program.

Look again at the SPRITE DESCRIPTION CHART in Fig. 14-1. It is organized just like a SPRITE. It is made up of 504 tiny squares. There are 24 columns and 21 rows ( $24 \times 21 = 504$ ). The 24 columns are divided into 3 sections. At the top of each section is a row of code numbers. These are the *column codes*, which are used to describe the SPRITE.

The shape of the SPRITE is drawn on the chart by lightly shading in some of the squares. Next, the column code numbers are written into the shaded squares. Finally, the numbers in each row are added together within each section, and the total is written in the corresponding box to the left of the chart.

As an example, let's look at row number 8. In the *left* section of row 8, the sum of the shaded squares is  $128 + 2 + 1 = 131$ . Therefore, a 131 is written in the left box of row number 8. In the *middle* section of row 8, there are no shaded squares and the sum is 0. Therefore, a 0 is written in the middle box. In the *right* section of row 8, the sum of the

Fig. 14-1. SPRITE DESCRIPTION CHART showing smiling square face.

DATA	LEFT BOX	MIDDLE BOX	RIGHT BOX	R O W	LEFT SECTION										MIDDLE SECTION										RIGHT SECTION										COLUMN CODE #	
					128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1								
DATA	255	255	255	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
DATA	128	0	1	2	128																														1	
DATA	128	0	1	3	128																														1	
DATA	128	0	1	4	128																														1	
DATA	128	0	1	5	128																														1	
DATA	128	0	1	6	128																														1	
DATA	128	0	1	7	128																														1	
DATA	131	0	193	8	128							2	1																						1	
DATA	131	0	193	9	128							2	1																						1	
DATA	128	0	1	10	128																														1	
DATA	128	0	1	11	128																														1	
DATA	128	24	1	12	128													16	8																1	
DATA	128	24	1	13	128													16	8																1	
DATA	140	0	49	14	128																														1	
DATA	140	0	49	15	128																														1	
DATA	140	0	49	16	128																														1	
DATA	143	255	241	17	128																														1	
DATA	143	255	241	18	128																														1	
DATA	128	0	1	19	128																														1	
DATA	128	0	1	20	128																														1	
DATA	255	255	255	21	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1



shaded squares is  $128 + 64 + 1 = 193$ . Therefore, a 193 is written in the right box.

After all 63 boxes on the chart have been filled in with code numbers, the code numbers are transferred to DATA statements in the program. For example, the code numbers that describe row 8 are 131,0,193. These show up in SPRITE PROGRAM #1 as line #780, which is DATA 131,0,193.

You have now seen how code numbers can be used to describe to the computer what the SPRITE should look like. The next step is to read these code numbers into memory.

#### **4 HOW DO I READ MY SPRITE DESCRIPTION CODE NUMBERS INTO MEMORY?**

Look back at SPRITE PROGRAM #1. The subroutine in lines #500-#540 reads the 63 code numbers in the DATA statements and then POKES them into memory locations 832-894. The computer does not automatically know that the code numbers are stored in memory starting at location 832. To find out where the code numbers are stored, the computer looks into memory location 2040 and multiplies the number that it finds there by 64.

In line #505, we POKE a 13 into location 2040. Therefore, when the computer finds a 13 in location 2040, it calculates the starting location of the code numbers as  $13 \times 64 = 832$ . Had we stored the code numbers starting in location 896, we would have to POKE a 14 into location 2040 because  $14 \times 64 = 896$ .

Location 2040 "points" to the starting location in memory where the code numbers are kept. For this reason, location 2040 is called a *pointer*.

#### **5 HOW DO I CONTROL THE COLOR OF THE SPRITE?**

A SPRITE's color is one of several factors controlled by a chip called the VIDEO INTERFACE CONTROLLER. We will call it the video chip, for short.

You can control the video chip by POKING various memory locations, starting with location 53248. For example, the memory location that controls the SPRITE's color is 53287. To display your SPRITE in the color that you want, you will have to POKE a COLOR CODE into



53287. The same COLOR CODES that were used for POKE GRAPHICS are also used for SPRITE GRAPHICS. (COLOR CODES are listed in Appendix C.)

Therefore, to display your SPRITE in green, you would use:

```
POKE 53287,5                                (Don't enter this.)
```

A better way to do the same thing is to use:

```
V=53248  
POKE V+39,5                                (Don't enter these yet.)
```

This works because  $53248 + 39 = 53287$ .

The second method is the one we used in SPRITE PROGRAM #1 and is the one we will use from now on. It is similar to the method we used for controlling the SID chip in the lesson for Day 12 on sounds and music. Now RUN SPRITE PROGRAM #1. You should see the red SPRITE smiling at you on the screen.

You can change the SPRITE's color to green directly on the screen by using the DIRECT MODE instruction:

```
POKE V+39,5                                (hit RETURN)
```

Your SPRITE should now be green. (If you are not hooked up to a color television set or a color monitor, you will have to use your imagination.)

To return the SPRITE's color to red, enter:

```
POKE V+39,2                                (hit RETURN)
```

These DIRECT MODE commands will only work *after* you run your SPRITE program.

In the previous example, we changed the SPRITE's color by using the DIRECT MODE. You can of course *change the program* itself so that the SPRITE is originally displayed in a green color. The following example shows how to change the SPRITE display color in a program.

#### EXAMPLE

Suppose you want SPRITE PROGRAM #1 to display the SPRITE in a blue color. To do this, change line #50 to:

```
50 POKE V+39,6
```

RUN the program again. The color of the SPRITE should be blue. Now, change line #50 to:

50 POKE V+39,2

RUN the program again to return the SPRITE color to red.

## **6 HOW DO I CONTROL THE SIZE OF THE SPRITE?**

SPRITES come in only two sizes—regular and two times regular. You can make your SPRITE twice as tall by expanding its size in the vertical direction. RUN SPRITE PROGRAM #1 and then enter:

```
POKE V+23,1 (hit RETURN)
```

To return the SPRITE to regular size, enter:

```
POKE V+23,0 (hit RETURN)
```

You can make your SPRITE twice as wide by expanding its size in the horizontal direction. To do this, enter:

```
POKE V+29,1
```

To return to normal size, enter:

```
POKE V+29,0
```

Now, look at lines #55 and #58. These lines POKE a zero in locations V + 23 and V + 29, which means that the SPRITE will be displayed in regular size. You *do not have to* put these lines into the program if you are going to POKE a zero. They are put in this program merely for reference. Only if you POKE a 1 into these locations do you need to put them into the program. In the following example, we will display our SPRITE double-size by POKEing a 1.

### EXAMPLE

Suppose you want SPRITE PROGRAM #1 to display a SPRITE that is twice as tall and twice as wide as regular size. To do this, change lines #55 and #58 to:

```
55 POKE V+23,1  
58 POKE V+29,1
```

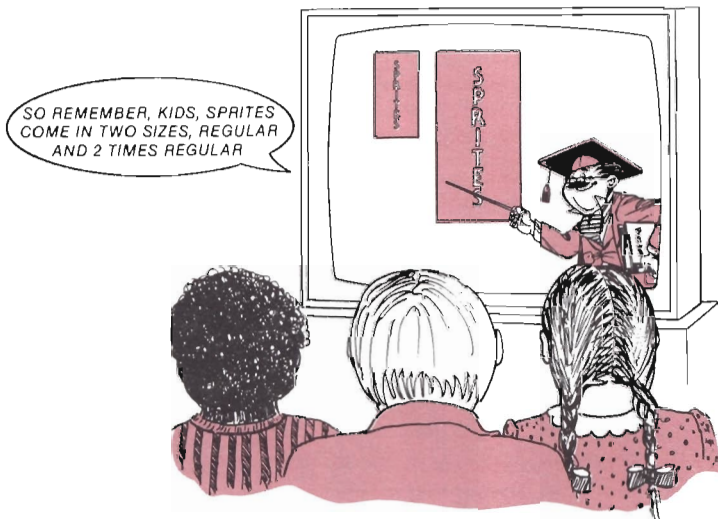
Now RUN the program to display your double-sized SPRITE.

To return the SPRITE to original size, change lines #50 and #58 back to:

```
55 POKE V+23,0
```

```
58 POKE V+29,0
```

RUN the program again to display the SPRITE's normal size.



## 7 HOW DO I CONTROL THE SCREEN POSITION OF MY SPRITE?

You can control your SPRITE's vertical (or Y) position by POKEing *any number* between 0 and 255 into memory location  $V + 1$ . Numbers closer to zero will move the SPRITE to the top of the screen. Numbers closer to 255 will move the SPRITE to the bottom of the screen.

RUN SPRITE PROGRAM #1. After the SPRITE appears on the screen, you can move it up or down. To move the SPRITE up, enter:

```
POKE V+1,70 (hit RETURN)
```

To move the SPRITE down, enter:

```
POKE V+1,200 (hit RETURN)
```

Experiment with other numbers between 0 and 255 to understand how the number affects the SPRITE's position on the screen.

Controlling the SPRITE's horizontal (or X) position is a bit more complicated because two memory locations are involved. These are

V + 16 and V. You can POKE into V any number between 0 and 255. Numbers closer to zero will move the SPRITE to the left. Numbers closer to 255 will move the SPRITE to the right.

RUN SPRITE PROGRAM #1. After the SPRITE appears on the screen, you can move it left or right. To move the SPRITE left, enter:

```
POKE V,40 (hit RETURN)
```

To move the SPRITE right, enter:

```
POKE V,255 (hit RETURN)
```

Experiment with other numbers between 0 and 255 to see how the number affects the SPRITE's position on the screen.

Notice that when you POKE the number 255 into location V, the SPRITE moves only about three-quarters of the way to the right. Since 255 is the highest number that can be poked into any memory location, how can you get the SPRITE past this "three-quarters point" and *all the way* to the right side?

The answer is that to get past the "three-quarters point," you must first POKE a 1 into location V + 16, and then begin POKEing numbers into V all over again from 0 to 255. For example, with the SPRITE still on the screen, enter:

```
POKE V+16,1: POKE V,60 (hit RETURN)
```

The SPRITE should now be close to the right side of the screen. With a 1 in location V + 16, you can POKE any number into V, from 0 to 255. However, any number higher than 87 will move the SPRITE off the screen to the right.

Now look at SPRITE PROGRAM #2 which follows. SPRITE PROGRAM #2 is the same as SPRITE PROGRAM #1, except that we have changed lines #5 and #80 and we have added lines #73, #75, #77, and #79. Change SPRITE PROGRAM #1 into SPRITE PROGRAM #2. You should see:

```
5 REM SPRITE PROGRAM #2 (SQUARE FACE)
10 PRINT "(SC)"
20 GOSUB 500:REM READ IN SHAPE DESCRIPTION
25 REM -----
30 REM SPRITE CONTROL
40 V=53248:REM START OF VIDEO
50 POKE V+39,2:REM COLOR
55 POKE V+23,0:REM EXPAND VERTICAL
58 POKE V+29,0:REM EXPAND HORIZONTAL
60 POKE V+21,1:REM TURN ON SPRITE
```

```

70 POKE V+1,100:REM Y POSITION
73 FOR X=0 TO 511
75 IF X<256 THEN POKE V+16,0:POKE V,X:REM X POSITION
77 IF X>255 THEN POKE V+16,1:POKE V,X-256:REM X POSITION
79 PRINT "PEEK (V+16)=";PEEK(V+16),"X=";X
80 NEXT
90 END
99 REM -----
500 REM READ IN SPRITE DESCRIPTION
505 POKE 2040,13:REM POINTER TO SPRITE DESCRIPTION
510 FOR N=0 TO 62
520 READ SD:POKE 832+N,SD
530 NEXT N
540 RETURN
695 REM -----
700 REM SPRITE DESCRIPTION
710 DATA 255,255,255
720 DATA 128,0,1
730 DATA 128,0,1
740 DATA 128,0,1
750 DATA 128,0,1
760 DATA 128,0,1
770 DATA 128,0,1
780 DATA 131,0,193
790 DATA 131,0,193
800 DATA 128,0,1
810 DATA 128,0,1
820 DATA 128,24,1
830 DATA 128,24,1
840 DATA 140,0,49
850 DATA 140,0,49
860 DATA 140,0,49
870 DATA 143,255,241
880 DATA 143,255,241
890 DATA 128,0,1
900 DATA 128,0,1
910 DATA 255,255,255

```

Notice that lines #73-#80 form a loop in which X increases from 0 to 511. In line #75, as long as X is less than 256, then V + 16 contains a 0, and we POKE the value of X into V. In line #77, when X becomes greater than 255, we POKE a 1 into V + 16 and X - 256 into V. In line #79, we PRINT the number that is in V + 16 and the value of X.

Type RUN and hit RETURN.

On the screen you will see the SPRITE move from left to right as the values of V + 16 and X are printed on the screen. Notice how the value of V + 16 changes from a 0 to a 1 when X becomes greater than 255.

## 8

## HOW DO I CREATE ANIMATION USING SPRITE GRAPHICS?

In SPRITE PROGRAM #2, our square-faced SPRITE moved from left to right across the screen. Let's now use the same method to create an animated picture of a plane.

### EXAMPLE

Suppose you want to use SPRITE GRAPHICS to create an animated picture of a plane moving from left to right across the screen. The shape of the plane is shown in the SPRITE DESCRIPTION CHART in Fig. 14-2.

*Note:* An easy way to create the program is to start with SPRITE PROGRAM #2, remove line #79, change line #5, and change all the DATA statements from line #710 to line #910. The new DATA statements should correspond with the SPRITE DESCRIPTION CHART in Fig. 14-2.

Type NEW and enter the following program:

```

5 REM SPRITE PROGRAM #3 (PLANE)
10 PRINT "(SC)"
20 GOSUB 500:REM READ IN SHAPE DESCRIPTION
25 REM -----
30 REM SPRITE CONTROL
40 V=53248:REM START OF VIDEO
50 POKE V+39,2:REM COLOR
55 POKE V+23,0:REM EXPAND VERTICAL
58 POKE V+29,0:REM EXPAND HORIZONTAL
60 POKE V+21,1:REM TURN ON SPRITE
70 POKE V+1,100:REM Y POSITION
73 FOR X=0 TO 344
75 IF X<256 THEN POKE V+16,0:POKE V,X:REM X POSITION
77 IF X>255 THEN POKE V+16,1:POKE V,X-256:REM X POSITION
80 NEXT
90 END
99 REM -----
500 REM READ IN SPRITE DESCRIPTION
505 POKE 2040,13:REM POINTER TO SPRITE DESCRIPTION
510 FOR N=0 TO 62
520 READ SD:POKE 832+N,SD
530 NEXT N
540 RETURN
695 REM -----
700 REM SPRITE DESCRIPTION
710 DATA 0,16,0
720 DATA 0,24,0
730 DATA 0,28,0
740 DATA 0,30,0

```

	LEFT BOX	MIDDLE BOX	RIGHT BOX	R O W	LEFT SECTION								MIDDLE SECTION								RIGHT SECTION								C O L U M N C O D E #						
					128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1							
DATA	0	16	0	1												16																			
DATA	0	24	0	2												16	8																		
DATA	0	28	0	3												16	8	4																	
DATA	0	30	0	4												16	8	4	2																
DATA	128	31	0	5	128											16	8	4	2	1															
DATA	64	31	128	6		64										16	8	4	2	1	128														
DATA	32	31	192	7			32									16	8	4	2	1	128	64													
DATA	16	31	224	8				16								16	8	4	2	1	128	64	32												
DATA	8	31	240	9					8							16	8	4	2	1	128	64	32	16											
DATA	7	255	252	10						4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4									
DATA	7	255	255	11							4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1						
DATA	7	255	252	12							4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4								
DATA	8	31	240	13					8							16	8	4	2	1	128	64	32	16											
DATA	16	31	224	14						16						16	8	4	2	1	128	64	32												
DATA	32	31	192	15							32					16	8	4	2	1	128	64													
DATA	64	31	128	16								64				16	8	4	2	1	128														
DATA	128	31	0	17	128											16	8	4	2	1															
DATA	0	30	0	18												16	8	4	2																
DATA	0	28	0	19												16	8	4																	
DATA	0	24	0	20												16	8																		
DATA	0	16	0	21												16																			

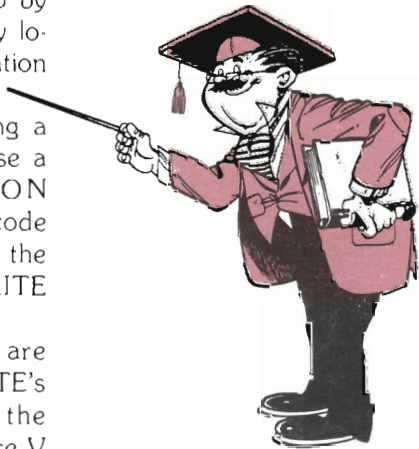
Fig. 14-2. SPRITE DESCRIPTION CHART showing plane.

```
750 DATA 128,31,0
760 DATA 64,31,128
770 DATA 32,31,192
780 DATA 16,31,224
790 DATA 8,31,240
800 DATA 7,255,252
810 DATA 7,255,255
820 DATA 7,255,252
830 DATA 8,31,240
840 DATA 16,31,224
850 DATA 32,31,192
860 DATA 64,31,128
870 DATA 128,31,0
880 DATA 0,30,0
890 DATA 0,28,0
900 DATA 0,24,0
910 DATA 0,16,0
```

Type RUN and hit RETURN. When you run this program, you should see the plane move from left to right across the screen.

## IMPORTANT THINGS TO REMEMBER ABOUT TODAY'S LESSON

1. SPRITES are created and controlled by the VIDEO INTERFACE CONTROLLER chip. You can control this chip by POKEing various memory locations, starting with location 53248.
2. The first step in creating a SPRITE program is to use a SPRITE DESCRIPTION CHART to calculate the code numbers that describe to the computer how the SPRITE should look.
3. Two memory locations are used to control a SPRITE's horizontal position on the screen. These locations are V and V + 16.





**SPRITE DEFINITION CHART**

	LEFT BOX	MIDDLE BOX	RIGHT BOX	R O W	LEFT SECTION							MIDDLE SECTION							RIGHT SECTION							COLUMN CODE#
					128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	
DATA				1																						
DATA				2																						
DATA				3																						
DATA				4																						
DATA				5																						
DATA				6																						
DATA				7																						
DATA				8																						
DATA				9																						
DATA				10																						
DATA				11																						
DATA				12																						
DATA				13																						
DATA				14																						
DATA				15																						
DATA				16																						
DATA				17																						
DATA				18																						
DATA				19																						
DATA				20																						
DATA				21																						



**SPRITE DEFINITION CHART**

	LEFT BOX	MIDDLE BOX	RIGHT BOX	R O W	LEFT SECTION 128 64 32 16 8 4 2 1	MIDDLE SECTION 128 64 32 16 8 4 2 1	RIGHT SECTION 128 64 32 16 8 4 2 1	COLUMN CODE#
DATA				1				
DATA				2				
DATA				3				
DATA				4				
DATA				5				
DATA				6				
DATA				7				
DATA				8				
DATA				9				
DATA				10				
DATA				11				
DATA				12				
DATA				13				
DATA				14				
DATA				15				
DATA				16				
DATA				17				
DATA				18				
DATA				19				
DATA				20				
DATA				21				





# APPENDICES

# APPENDIX A

## Commodore ASCII Codes

CODE	PRINTS	CODE	PRINTS	CODE	PRINTS	CODE	PRINTS
0		31		62	>	93	]
1		32		63	?	94	↑
2		33	!	64	@	95	←
3		34	"	65	A	96	
4		35	#	66	B	97	
5		36	\$	67	C	98	
6		37	%	68	D	99	
7		38	&	69	E	100	
8	DISABLES	39	.	70	F	101	
9	ENABLES	40	(	71	G	102	
10		41	)	72	H	103	
11		42	*	73	I	104	
12		43	+	74	J	105	
13		44	,	75	K	106	
14		45	-	76	L	107	
15		46	.	77	M	108	
16		47	/	78	N	109	
17		48	0	79	O	110	
18		49	1	80	P	111	
19		50	2	81	Q	112	
20		51	3	82	R	113	
21		52	4	83	S	114	
22		53	5	84	T	115	
23		54	6	85	U	116	
24		55	7	86	V	117	
25		56	8	87	W	118	
26		57	9	88	X	119	
27		58	:	89	Y	120	
28		59	;	90	Z	121	
29		60	<	91	[	122	
30		61	=	92	£	123	

CODE	PRINTS	CODE	PRINTS	CODE	PRINTS	CODE	PRINTS
124		141	SHIFT RETURN	158	YEL	175	
125		142	SWITCH TO UPPER CASE	159	CYN	176	
126		143		160	SPACE	177	
127		144	BLK	161		178	
128		145	CRSR	162		179	
129	Orange	146	RVS OFF	163		180	
130		147	CLR HOME	164		181	
131		148	INST DEL	165		182	
132		149	Brown	166		183	
133	f1	150	Lt. Red	167		184	
134	f3	151	Grey 1	168		185	
135	f5	152	Grey 2	169		186	
136	f7	153	Lt. Green	170		187	
137	f2	154	Lt. Blue	171		188	
138	f4	155	Grey 3	172		189	
139	f6	156	PUR	173		190	
140	f8	157	CRSR	174		191	



































Codes 192-223 same as 96-127  
Codes 224-254 same as 160-190  
Code 255 same as 126

# APPENDIX B

## Screen Codes

CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
0	@	31	←	62	>
1	A	32	<b>SPACE</b>	63	?
2	B	33	!	64	<input type="checkbox"/>
3	C	34	"	65	<input type="checkbox"/>
4	D	35	#	66	<input type="checkbox"/>
5	E	36	\$	67	<input type="checkbox"/>
6	F	37	%	68	<input type="checkbox"/>
7	G	38	&	69	<input type="checkbox"/>
8	H	39	,	70	<input type="checkbox"/>
9	I	40	(	71	<input type="checkbox"/>
10	J	41	)	72	<input type="checkbox"/>
11	K	42	*	73	<input type="checkbox"/>
12	L	43	+	74	<input type="checkbox"/>
13	M	44	,	75	<input type="checkbox"/>
14	N	45	-	76	<input type="checkbox"/>
15	O	46	.	77	<input type="checkbox"/>
16	P	47	/	78	<input type="checkbox"/>
17	Q	48	0	79	<input type="checkbox"/>
18	R	49	1	80	<input type="checkbox"/>
19	S	50	2	81	<input checked="" type="checkbox"/>
20	T	51	3	82	<input type="checkbox"/>
21	U	52	4	83	<input type="checkbox"/>
22	V	53	5	84	<input type="checkbox"/>
23	W	54	6	85	<input type="checkbox"/>
24	X	55	7	86	<input checked="" type="checkbox"/>
25	Y	56	8	87	<input checked="" type="checkbox"/>
26	Z	57	9	88	<input checked="" type="checkbox"/>
27	[	58	:	89	<input type="checkbox"/>
28	£	59	;	90	<input type="checkbox"/>
29	]	60	<	91	<input type="checkbox"/>
30	↑	61	=	92	<input checked="" type="checkbox"/>



CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
93		105		117	
94		106		118	
95		107		119	
96	<b>SPACE</b>	108		120	
97		109		121	
98		110		122	
99		111		123	
100		112		124	
101		113		125	
102		114		126	
103		115		127	
104		116			

Codes from 128-255 are reverse images of codes 0-127

# APPENDIX C

## Color Codes

0 Black	8 Orange
1 White	9 Brown
2 Red	10 Light Red
3 Cyan	11 Gray 1
4 Purple	12 Gray 2
5 Green	13 Light Green
6 Blue	14 Light Blue
7 Yellow	15 Gray 3

# Index

## A

- Abbreviations for commands, list of, 52
- ABS function, 106-107
- Adding program lines, 17
- Addition in BASIC, 37-38
- ADSR
  - definition of, 134, 135, 139
  - how to use, 139-141
  - memory location to POKE, 136
  - values for, 141
- AND, with IF command, 78-79
- Animation, 158-164
  - with SPRITE GRAPHICS, 176-177
- Arcade games; see video games
- Argument; see function
- Arrays, 114-115, 117-118, 120-121
  - copying data, 118-119
  - definition of, 114
  - moving data in and out of, 115-116, 121-123
  - one-dimensional, 119-120, 121
  - strings in, 116-117
  - two-dimensional, 119-123
- ASC function, 108
- ASCII and CHR\$ codes,
  - definition of, 77, 107, 108
  - list of, 184-185
  - number values in, 77-78
- Asterisk, as multiplication symbol, 37-38
- ATTACK; see ADSR, sound wave

## B

- BASIC
  - commands, 16
  - mathematical operations in, 37-38
  - symbols, 37-38

## C

- Calculation of power in BASIC, 38-39
- Cassette
  - files; see files
  - recorder, 92, 93
  - tape, 93-94, 97-99
    - loading programs from, 94-95
    - recording data on, 95-97
    - saving programs on, 92-93

## Characters

- adding in program line, 26-27
- changing in program line, 28
- deleting from program line, 27-28
- reversing, 44, 50-51, 128, 155
- CHR\$ function, 108, 150-151
- Clearing the screen
  - BASIC statement for, 45
  - with
    - CHR\$ function, 108
    - PRINT command, 44-45
    - SHIFT and CLR/HOME keys, 12-13
- CLOSE command, 96, 98
- CLR/HOME key, clearing screen with, 13
- Color, memory location for controlling, 152
- COLOR
  - CODES, 153, 155, 171, 188
  - MEMORY, 152, 153-155, 156
- Commodore key, 94
- Commodore 64 BASIC commands; see specific command
- Comparison symbols, 74
- CONT command, 18, 19
- Continuing program, 18-19
- CRSR keys; see cursor control keys
- Cursor, 12, 13-14
  - control keys
    - editing with, 26
    - moving cursor with, 13-14
  - controlling position of
    - cursor control keys, 13-14
    - PRINT command, 51-53
  - definition of, 12
  - POKE command to turn off, 128-129

## D

- Data, 46, 47, 49-50, 62, 64-65
  - commands for entering, 56
  - definition of, 32
  - in arrays, 115-116, 118-119, 121-123
- DATA
  - commands, 56
  - statement, 56, 62, 145-146
- DECAY; see ADSR, sound wave
- DIM command, 117-118, 120-121

Direct mode  
  definition of, 14  
  use of question mark in, 53  
Division in BASIC, 37-38

## E

Edit, 25-26  
END command, 19  
Entering a program, 17  
Equals sign, 36-37

## F

Files  
  assigning names to, 95  
  closing, 96-97, 98  
  definition of, 92  
  opening, 95-96, 97-98  
  reading, 97-99  
  recording, 95-97  
  saving, 92-93  
FIRE BUTTON; see video games  
FOR-NEXT  
  command  
    in loops, 82  
    in POKE graphics, 155, 157-158  
  loop, 82-90  
    FOR command in, 85, 86-87  
    increasing count in, 85-86  
    NEXT command in, 85, 89-90  
    numbers in, 86  
    STEP command in, 85-86  
    when to use, 84  
Frequency, 134, 135  
  changing tone, 137-139  
  memory location to POKE, 136  
Functions, 104-112; see also numeric  
  functions, string functions  
  definition of, 104-105

## G

Games; see video games  
GET command, 56, 65-66  
GOSUB command, 68, 71-73  
GOTO  
  command, 68-70, 74-75, 82, 84  
  loop, 83-84  
Graphics on Commodore 64, 148-164

## H

High byte; see memory location

## I

IF command, 68, 73-77  
  BASIC commands used in, 74  
  mathematical symbols used in, 74  
  parts of, 73-75  
  with multiple statements, 76-77  
INPUT command, 56-61  
INPUT# command, 98-99  
INST/DEL key, 12, 27-28

Instructions, entering and executing, 12-13, 14, 15  
INT function, 106-107

## J

Joystick; see video games  
"Jingle Bells" song, 144-145

## K

Keyboard graphics characters, 148-149

## L

LEFT\$ function, 109  
LEN function, 104-106  
LET command, 37  
LIST command, 22  
Logical operators, 78-79  
Loops  
  definition of, 70, 82  
  delay, 142-143  
  FOR-NEXT, 84-90, 157  
  GOTO, 83-84  
  nested, 87-88  
  uses of, 83-84  
  with arrays, 119-122  
Low byte; see memory location

## M

Mathematical operations  
  order of performance in, 40-41  
  parentheses used to group, 40-41  
  types of, 37-39  
Memory location  
  formula for calculating, 130  
  poking wrong, 126, 127, 134  
MID\$ function, 109  
Multiplication in BASIC, 37-38  
Music, creating on computer, 134-146  
Musical notes; 138, 145-146; see also tone

## N

Nested loop, 87-89  
NEW command, 17  
NEXT command, 85, 89-90  
Noise waveform; see waveforms  
Number(s); see numeric constant  
  IF command used to compare, 75-76  
  random, 110-112  
  STR\$ function used with, 110  
Numeric  
  constant, 34  
  functions  
    ABS, 106-107  
    ASC, 108  
    INT, 106-107  
    LEN, 104, 105-106  
    SGN, 106-107  
    VAL, 110  
  variable, 33-34

## O

- ON GOSUB command, 68, 72-73
- ON GOTO command, 68, 70-71
- OPEN command used to
  - open file, 96
  - read file, 97-98
- OR, with IF command, 78-79

## P

- PEEK command, 126, 129, 130
  - Plus sign, 37-38, 41
  - Pointer, 62, 63, 170
  - POKE
    - command, 126-129, 135-144
      - controlling SID chip with, 135-144
      - creating nonlistable program with, 127
      - making keys automatically repeat with, 128
    - printing
      - graphics with, 148, 151-160
      - reverse characters with, 128
    - turning off
      - cursor with, 128-129
      - keyboard with, 127
    - used with video chip, 170
  - graphics, 148
    - creating arcade game with, 163-164
    - diamond, 151-152
    - drawing large pictures on screen with, 155, 157-158
  - FOR-NEXT, READ commands used in, 155, 157
  - plane, 152-153
  - SCREEN CODES, MEMORY, 151, 152
- PRINT
- command, 14, 19
    - clearing screen with, 44-45
    - inserting cursor movements with, 44, 51-53
  - printing
    - blank line with, 60
    - data with, 44
    - graphics with, 148-151
    - reversed characters with, 44
  - putting calculations into, 39
  - with
    - comma, 46
    - question mark, 53
  - graphics
    - CHR\$ used to produce, 150-151
    - diamond, 148-149, 150
    - keyboard graphics characters used to produce, 148-149
    - plane, 149, 151-152
- PRINT# command, 95-96
- Printing
- in standard print columns, 45-47
  - spaces between data, 49-50
  - with data items near each other, 47-49

- Program(s), 15, 16, 23, 68, 71
  - and data, 92-95
  - editing line in, 25-26
  - errors, correcting, 18
  - line(s)
    - adding characters to, 26-27
    - changing characters in, 28
    - deleting characters from, 27-28
    - multiple statements in, 19
  - listing, 22
  - mode
    - definition of, 15
    - use of question mark in, 53
  - removing line from, 24
  - replacing line in, 24
  - SPRITE, 166-168, 174-175
  - stopping, 69
  - using
    - arrays in, 114-115
    - loops in, 82-90
- Prompt; see reminder message
- Pulse waveform; see waveforms
- Pulse width, 143-144

## Q

- Question mark, as substitute for word "PRINT", 53
- Quotation marks, use of in strings, 32, 41

## R

- Random numbers, 110-112
- READ
  - command
    - function of, 56, 61-62
    - in POKE graphics, 155, 157
    - with loops, 83
    - statement, 61, 62-63
- READY message, 12
- RELEASE; see ADSR, sound wave
- REM command, 19
  - identifying program parts with, 71
  - in subroutine, 71
- Reminder message, in INPUT, PRINT commands, 61
- RESTORE
  - command, resetting pointer with, 64-65
  - key, 18
    - with POKE command, 127
    - with RUN/STOP key, 18, 51
- Restoring screen to original colors, 51
- RETURN
  - command
    - in subroutine, 71-72
    - used with RUN/STOP key to stop program, 69
  - key
    - adding program line with, 23
    - CONT command used with, 18, 19
    - editing with, 25
    - LIST command used with, 22

Return—Contd.

key

removing program line with, 24

Reversed characters, 44

printing, 50-51, 128

SCREEN CODES for, 155

RIGHT\$ function, 109

Rules for naming variables, 36

Running a program, 17

RUN/STOP key

POKE command used with, 127

RESTORE key used with, 18-19, 51

returning screen to original color with,  
51

used to stop program, 18-19, 69

## S

Sawtooth waveform; see waveforms

Screen, all-white, 148, 166

SCREEN

CODES, 151, 152, 153, 155

list of, 186-187

MEMORY, 152, 153-155

diagram, 154

Semicolon, separating data with, 47-49

SGN function, 106-107

SID chip

features of, 134-135

memory locations used to control,  
135-136

Sound, using computer to create, 134-146

Sound wave, parts of, 139-141

SPC(X) command, 49-50

SPRITE

chart for, 169

controlling

color of, 170-172

position of, 173-176

size of, 172-173

definition of, 166

DESCRIPTION, 168-170

chart, 169

reading code numbers into memory,  
170

GRAPHICS, 176-177

programs, 166-168, 174-175

Standard print columns, 45-47

STATUS command, 98

STEP command, in FOR-NEXT loop, 85-86

Stopping a program, 18-19

String(s)

adding together, 41

compared with numbers, 32

String—Contd.

comparing ASCII values of, 77-78

constant, 35

functions

CHR\$, 108

LEFT\$, 109

MID\$, 109

RIGHT\$, 109

STR\$, 110

\$ used in, 104, 105

IF command used to compare, 77-78

in arrays, 116-117

quotation marks with, 32

variable(s), 34-35

\$ sign in, 35

STR\$ function, 110

Subroutine

definition of, 71

GOSUB command in, 71-72

ON GOSUB command in, 72-73

REM command in, 71

RETURN command in, 71-72

Subtraction in BASIC, 37-38

SUSTAIN; see ADSR, sound wave

## T

TAB function, 44, 50

Tone(s)

changing frequency of, 137-139

created from waveforms, 141-144

producing short, 136-137

Triangle waveform; see waveforms

## V

VAL function, 110

Variables

assigning values to, 36-37

naming, 36

numeric, string, 34

Video

chip, 170

games, 160-164

Virgule, as BASIC symbol for division, 37-38

Volume, 134, 135

adjusting, 135-136

memory location to POKE, 136, 137

## W

Waveforms, 134, 135

ID numbers for, 142

memory location to POKE, 136

turning on and off, 141-144

types of, 141-142



# Learn BASIC Programming in 14 Days on Your Commodore 64™

Learn how to program in BASIC on your Commodore 64. It's fun and easy with this mini course written especially for beginners 13 years and older.

- 14 easy to understand lessons
- Simple question and answer format
- No complicated technical terms or "computerese"
- Over 130 programs and examples for hands on experience
- Clear explanations of entering and editing a program, printing, using loops and arrays, saving data on cassette tapes, creating sound and music, working with color and animation, designing graphics, and more!

**Howard W. Sams & Co., Inc.**  
4300 West 62nd Street, Indianapolis, Indiana 46268 U.S.A.