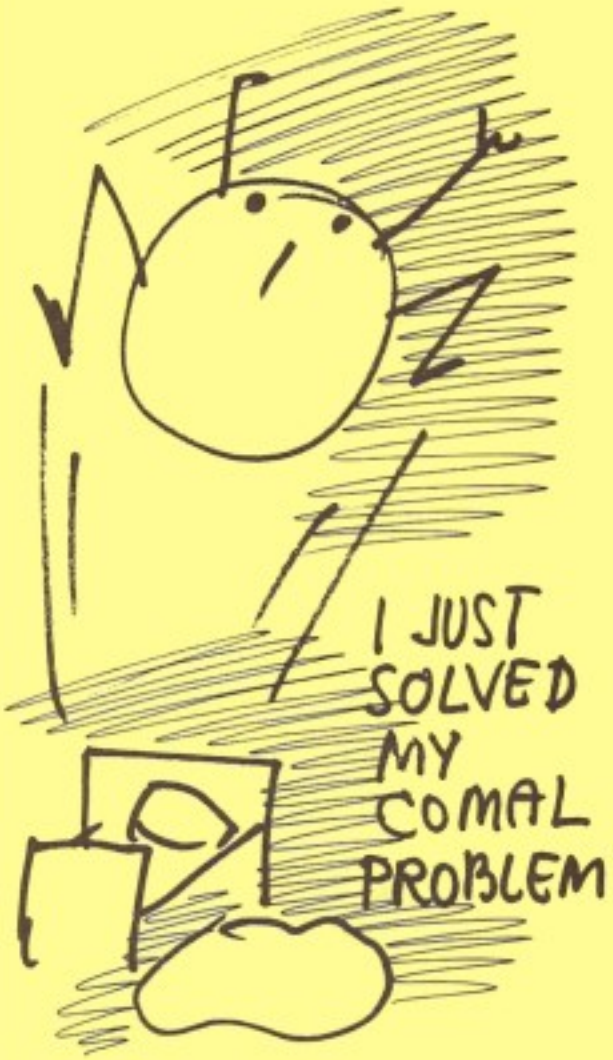
THE AMAZING ADVENTURES OF

CAPTAIN COMAL™

BOOK 6

COMAL WORKBOOK

BY GORDON SHIGLEY



WELCOME TO

Cover art by Frank Hejndorf

PREFACE

The WORKBOOK provides a simple and clear introduction to programming for secondary school students. The text seeks to accomodate students and home users who otherwise might shy away from computer programming as a discipline that is too technical, too mathematical. The WORKBOOK compliments THE COMAL COMPUTER TUTOR [also called TUTORIAL DISK], an interactive series of lessons that introduces programming with tutorials that contain programs which the student can "LIST" and "RUN" within each lesson. This innovation permits programs to "come alive" as immediate demonstrations that show the effect of new concepts, statements, and commands. The WORKBOOK takes up where each lesson ends, providing a hands-on tutorial not simply to a computer language, but to the fundamental concepts behind programming.

We take it for granted that COMAL is a superior language for instructional purposes. Variables that can be named without any practical limitation make COMAL programs self documenting. Most of the control structures available to Pascal or Ada, the speed of compiled code, the ability to run programs still in the developmental stages, and the interactive environment of simple but increasingly obsolete languages such as BASIC -- all these combine to produce a highly structured, easily programmed, powerful, and friendly language. Teachers no longer have to guess what a student's program will do. Nor will they (or the student) have to put up with the frustrating environment of Pascal, particularly for those just learning how to program. And the cost advantages, or the availability of the LOGO turtle haven't even been mentioned!

The WORKBOOK's design carries a central focus: activity. We pay more than lip service to the axiom that the way to learn programming is to write programs. Each exercise concentrates on having the student do something, observe the result, and apply the concept. A self-check section ends each exercise.

Even for students who use the WORKBOOK as an extra-credit or extracurricular project, these lessons provide insights and guidance for overcoming many of the bad habits that older languages seem to encourage.

Grateful thanks to my students in BASIC and Pascal, whose enthusiasm for COMAL enticed me to take the plunge!

Madison, WI.
August 6, 1984
Gordon D. Shigley

ISBN 0-928411-05-2

EXERCISE 1: GETTING STARTED

This exercise is probably the most difficult because everything you do will be something new. Don't worry. You will soon be able to do all of these steps in your sleep!

There are three pieces of equipment of "hardware" that you will be using. The first is the computer. It's the one with all the typewriter keys on it. There is an on-off switch on the right-hand side. Don't turn it on yet.

The second piece of "hardware" is the disk drive. It is long and narrow, has a green and red light on the front, and a narrow opening across the front guarded by a small handle that travels up and down. There is an on-off switch at the back of the disk drive. Don't turn it on yet.

The last piece of hardware is the monitor. It's the TV-like screen on which the computer displays its information. Now follow these steps:

1. Turn on the monitor, disk drive, and computer. The monitor should show this message:

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

 64K RAM SYSTEM  38911 BASIC BYTES FREE

 READY.
```

and there will be a flashing square under the R of READY. That flashing square is called the CURSOR.

2. Open the disk-drive door by gently pushing in the small handle and letting it go upwards. Pick up THE COMAL COMPUTER TUTOR [TUTORIAL DISK] disk so that the label is nearest you and facing upwards. Gently slide the disk all the way into the disk drive until it stops. (If it pops out, just push it back in until it stops and stays in.) Pull the handle back down and slightly toward you until it stays down.

3. Very carefully type this message on the keyboard of the computer (if you make a typing mistake, use the instant delete key marked INST DEL to make a correction):

```
LOAD "*",8
```

and "enter" this command into the computer by pressing the RETURN key. The monitor will respond with

```
SEARCHING FOR *
LOADING
READY.
```

4. Enter the RUN command by typing:

```
RUN
```

and pressing the RETURN key. On the monitor you will see an introductory message while the program loads. It takes just over a minute to load the COMAL language and the initial MENU of choices.

5. When the MENU screen appears you will see this message at the bottom:

```
ENTER NUMBER (Ø TO QUIT) -->
```

Press the number 1 (don't use the letter L for 1) and enter this choice as usual by pressing the RETURN key. The computer will direct the disk drive to locate this lesson on the disk, load it into the computer, and begin running it.

6. You can now see a flashing sign on the screen under the lesson title. Carry out this instruction and you are on your way! When you reach the end of lesson one, the MENU will automatically reappear. You may either repeat lesson one at this time, or go on to lesson two.

7. You may quit at any time by pressing the RUN-STOP key or entering a zero (Ø) as your MENU choice. When you quit the tutorial, you are placed into COMAL programming mode. You then can enter COMAL commands or write your own program. You can restart an interrupted lesson by entering the RUN command again (unless you have changed or deleted it). You can return to the MENU of the TUTORIAL DISK by entering this COMAL command:

```
CHAIN "HI"
```

EXERCISE 1 SELF-TEST

1. Name three pieces of "hardware."

_____ _____ _____

2. How are commands "entered" into the computer?

_____

3. What is the cursor?

_____

4. How do you enter the RUN command?

_____

5. How can you get a listing of your program statements?

_____

6. What would this program print on the screen?

```
PRINT "HELLO.  I LOVE YOU."
```

_____

7. Write a program that, when RUN, will say MERRY CHRISTMAS

_____

8. Write a two-line program that will say

```
HAPPY
THANKSGIVING
```

_____

_____

9. How can you stop a program that is running?

_____

10. How can you get it running again (what command will you enter?)

_____

EXERCISE 2: PROGRAMMING FOR OUTPUT

This exercise gives you practice in loading and using COMAL. You will be adding the AUTO, and NEW commands to the system commands that you already know -- the LIST and RUN commands. You will also be practicing the PRINT programming statement.

Practice starting the COMAL system. Turn on the computer (if it is alreaady on - turn it off, then back on again.) Place the COMAL SAMPLER disk into the disk drive, type LOAD"*",8 and enter this command as usual. Do that with the disk drive turned OFF and write the ERROR message that appears on the monitor.

This is the message you can expect any time you try to locate a program on the disk with the disk drive turned off. Remove the disk (never turn the drive on or off with a disk inserted). Now turn the disk drive on, insert the disk, and re-enter the LOAD"*",8 command. You should get the familiar:

```
SEARCHING FOR *
LOADING
READY.
```

When you load "*" the computer looks for the first programon the disk's catalog of programs. The first program happens to be BOOT C64 COMAL and you also could load it by typing LOAD "BOOT C64 COMAL",8 and enter the command with the RETURN key.

When you RUN the BOOT program, it automatically finds, loads, and begins to run the COMAL language. There are a number of messages displayed on the screen while COMAL is loading. The last message asks you to BACKUP THIS DISK WITH 1541 BACKUP FIRST. Your teacher has already done this for you, so you may ignore this instruction. All it means is that you may load and run a program that copies the entire group of programs on one disk to another blank disk -- that is, you may make a copy of the COMAL language disk. You should not use this program to make copies of copyrighted programs unless there is a specific message permitting you to do so. Go ahead and run the BOOT program. It will take about a minute to load COMAL.

As COMAL starts, it asks if you wish to have error messages. The error messages are part of a disk file. Reply Y if you want error messages, or N if not.

Then the last line that appears on the screen is

```
Help Demo Comal Graphics (H,D,C,G):-
```

To enter the COMAL programming mode, type the letter C and press RETURN [Note: early versions of the SAMPLER DISK should enter the letter P for Program mode instead of C].

Now enter the LIST command.

Wow! Look at all those program statement fly by! There is a quick way to clear these out of the computer's memory. Enter the NEW command followed by the RETURN key. The program has vanished. You can prove this. How? By entering the LIST command again. Do that now. Did any statements appear after you entered this command? (YES / NO) When there are no statements to list, nothing gets listed.

Now hold down the SHIFT key and while you are still holding it down, press the CLEAR HOME key -- it's in the upper right hand corner and marked CLR HOME. What happened?

_____

The CLEAR HOME key clears off anything that is on the screen, and puts the CURSOR in the "home" position -- top left-hand corner of the screen.

What two keys do you use to clear off the screen?

_____ _____

In some computer languages you have to type in line numbers for every statement. In COMAL line numbers appear automatically if you enter the AUTO command. Do that now. What numbers appear on the screen?

_____

Now type in this statement exactly as you see it here:

```
PRINT "HAPPY"
```

and "enter" this line as usual with the RETURN key. If you forget to include the word PRINT before typing "HAPPY" an error message will appear on the screen telling you that "happy" is not a statement. The CURSOR will stop where it thinks the error begins -- and you can correct the mistake right on the spot. Let's suppose that you didn't make any mistakes. What number appeared after you entered the first statement?

_____

Now type in this program statement:

```
PRINT "THANKSGIVING"
```

and "enter" it as usual. What number appears on the screen now?

_____

Don't type any more statements. Instead, just press the RETURN key one more time. Did any new numbers appear under your statements? (YES NO)

It is time to RUN your program. Enter the RUN command. What appears under the RUN command now?

_____

_____

_____

Important point: The PRINT statement tells the computer to print anything that appears in quotation marks. (The last message on the screen just lets you know the line number where the computer program ended.) Enter the LIST command. What two program statements appeared? Write them here (you may leave off line numbers).

_____

_____

This is a listing of your program statements. Do you remember how to clear off the screen? (You have to use two keys at the same time.) Try that now.

Now enter the LIST command again. Did your program statements reappear? (YES / NO) Important point: clearing off the screen is not the same as using the NEW command to clear a program from the computer's memory. Your program hasn't vanished. It is still there. Clear off the screen once more, then enter the RUN COMMAND. What two words appear under the RUN command?

_____

_____

Are there any quotation marks around either word? (YES / NO) LIST the program once more. Now are there quotation marks around each word that you asked the compuer to print? (YES / NO) Remember, the computer will print anything with the PRINT statement that you enclose in quotation marks.

Clear this program from the compuer memory and entering the NEW command. Now enter the RUN command. What appears on the screen?

_____

This message means that the program ended at line 0000. In other words, there was no program to run.

It's time for you to write a program of your own design. First clear off the screen again. Enter the AUTO command to starting the automatic line-numbering system. Now enter at least two PRINT statements that say anything you wish. Each PRINT statement may contain several words (such as PRINT "I LOVE MY DOG.").

After you have entered your second PRINT statement, be sure to press the RETURN key a second time to get out of the automatic line numbering system. Copy your two PRINT statements here:

_____

_____

Now RUN your program. What two lines appear under the RUN command?

_____

_____

Enter the NEW command to clear this program from the computer's memory. Clear off the screen, then enter the AUTO command. Now write another short program composed PRINT statements. RUN your program and LIST it several times. (Remember, you must press the RETURN key twice to get out of the automatic line-numbering system before you can enter direct commands such as RUN and LIST.)

Now enter this command: RENUM 10Ø and then LIST the program again. Are your statements renumbered, beginning with 10Ø? (YES / NO) Enter RENUM 5ØØ and then list the program once more. Now what number begins your program statements?

_____

Enter RENUM without any numbers and list the program again. Do your statements once more begin with 1Ø? (YES / NO) Try this one. Enter RENUM 10Ø,2 and list the program. What number begins your program?

_____

What is the number of the second line?

_____

Important point: the RENUM command renumbers program statements.

EXERCISE 2: SELF-TEST

1. What does LOAD"*",8 tell the computer to do?

_____

2. How do you clear out an existing program before beginning to write a new one?

_____

3. How do you clear off the screen at any time?

_____

4. What command do you enter to get automatic line numbering?

_____

5. How do you get out of the automatic line-numbering system?

_____

6. What will this program print on the screen when it is RUN?

```
PRINT "********"
PRINT "* O  O *"
PRINT "*  !!  *"
PRINT "* ==== *"
PRINT "********"
```

7. What command would you enter if you wanted to renumber all your program statements beginning with 900?

_____

8. What command would you use to renumber all your program statements beginning with 10 and going up by 2 each statement?

_____

9. Can you have more than one word for each PRINT statement? (YES

EXERCISE 3: OUTPUT OF VARIABLE VALUES

You should complete at least lesson 3 on THE COMAL COMPUTER TUTOR [TUTORIAL DISK] before beginning this exercise. If you have just finished lesson 3, press the RUN-STOP key and enter the NEW command. If you are loading COMAL from the COMAL SAMPLER disk, get into the Comal programming mode. Now clear off the screen and get into the auto line numbering system (by entering the AUTO command).

You can switch from upper case letters to lower case letters and back again by holding down the C= key (it's the one in the lower left-hand corner of the keyboard) and while holding this down gently tap the SHIFT key. Try that several times right now. Does the AUTO change to "auto" and back again? (YES / NO) You may leave the computer in either upper or lower case mode. All the programs in this exercise use upper case, but in COMAL, unlike some types of BASIC, programs run perfectly in either mode. Now enter this short program:

```
PRINT "THIS PROGRAM SHOWS THE INPUT STATEMENT."
```

Notice that after the T of INPUT the rest of the statement automatically goes to a new line. On the screen it should look like this:

```
0010 PRINT "THIS PROGRAM SHOWS THE INPUT
 STATEMENT."
0020
```

Hit the RETURN key a second time to get out of the AUTO system. Now RUN the program. Does the entire sentence fit on one line? (YES / NO)

LIST the program so far. Enter the command, AUTO 20

What line number appears?

_____

Now add this statement:

```
INPUT "WHEN WILL YOU GRADUATE?":YEAR
```

It doesn't make any difference if the R of YEAR doesn't fit on the first line. You already know it will fit when we RUN the program. Be sure there is a colon (:) before the word YEAR. The colon is an important part of the INPUT statement. If it is missing, the computer will think that you have made a mistake (and you have.) Now add this third PRINT statement:

```
PRINT "YOU WILL GRADUATE IN";YEAR
```

Make sure that a semicolon (;) appears before YEAR in this line. Now hit the RETURN key a second time to get out of the AUTO system, then LIST your whole program. Now RUN your program. Notice that the program stops after the question mark. It is waiting for

you to enter a number. Enter the year that you will graduate. What does the third line of the program print on the screen?

_____

Run the program again. This time enter a different number for YEAR. What number did you enter?

_____

What does the last line of the program now show?

_____

YEAR holds the number that the INPUT statement gets from the user. Since the number can vary each time the program is run, YEAR is called a VARIABLE. The value it holds can vary from one RUN of the program to the next.

LIST the program again. Let's use the CURSOR CONTROL KEYS (those are the last two keys in the bottom right-hand corner of the keyboard labeled CRSR) to move the cursor. First move the cursor to the right by pressing the key in the very right-hand corner 5 times. What happens?

_____

Now hold down the SHIFT key and press this same CRSR key another 5 times. Which way did the cursor move? (LEFT / RIGHT)

Do the same things for the other cursor key. Can you get the cursor to move up and down the screen without erasing the text? (YES / NO)

Using the cursor control keys, move the cursor over to the semicolon (;) that appears just before the variable YEAR. When the cursor is blinking right on top of the semicolon, type a comma (,) then press the RETURN key. Now move the cursor down past the last program program statement and LIST the program again. Did the comma replace the semicolon in your program listing? (YES / NO)

Clear off the screen and LIST the program again. Now RUN the program once more, enter any number, and look at the results. Is there a space between the word IN and the year that you entered? (YES / NO) Important point: A semicolon always leave a single space before it prints the variable's value. A comma does not. (The comma does great and glorious things that we will cover in another exercise!)

Use the cursor control keys to get past the output from your program RUN, all the way up to the comma before YEAR. Change the comma back to a semicolon, press RETURN, and get the cursor back down past the last program statement. Clear off the screen and LIST the program again.

We want to add some statements to the program without having to type the whole program over. Enter this command:

```
AUTO 40
```

and notice that the automatic line-number system starts with line-number 40. Go ahead and add these two statements:

```
PRINT YEAR;"IS WHEN I WILL GRADUATE."
PRINT "YES";YEAR;"IS WHEN I'M FREE."
```

Get out of the AUTO system as usual by pressing the RETURN key a second time. RUN the program. We would like to change the word YOU in the second statement to the word I. Here's how. LIST the program. Use the cursor control keys to get up to the Y of YOU in line 30. When the cursor is blinking over the Y type the letter I and then move the cursor two spaces to the right. Now press the INST DEL key (in the upper right-hand corner of the keyboard) two times. Did the OU of YOU DISAPPEAR? (YES / NO) Now hit the RETURN key and then get the cursor past the last program statement with the cursor keys. RUN the program again. Important point: you can change whole words by using the cursor control keys and the INST DEL key without having to retype a whole program or even a single program statement. Once you have made the change and entered it with the RETURN key, be sure to use the cursor control keys to get past the last statement on the screen.

Look at the last line of your output. Is there a space on either side the number you entered for YEAR? (YES / NO) Why?

By using semicolons you can print the value for a variable at the beginning of a print statement, in the middle of the statement, or at the end.

EXERCISE 3: SELF-TEST

1. What is the purpose of the NEW command?

_____

2. You have already typed in three program lines (10,20,30), and have hit the RETURN key twice to get out of the AUTO system. You have RUN the program. Now you want to continue adding statements with the AUTO system. What command do you enter?

_____

3. INPUT statements usually have a colon. Where does the colon appear in an INPUT statement? (Or, you may give an example.)

_____

4. What is the variable in this statement?

```
INPUT "ENTER THE TOTAL DISTANCE IN MILES: ":MILES
```

_____

5. What is the variable in this PRINT statement?

```
PRINT "TOTAL COST IS $",COST;"FOR THE REPAIR"
```

_____

7. How many spaces does the semicolon automatically provide?

_____

8. What keys can move the cursor without erasing text?

_____

9. What key moves the cursor and erases text?

_____

10. Why is a variable called a "variable"?

_____

11. Design and run a program that uses an INPUT statement.

EXERCISE 4: ASSIGNMENTS AND EXPRESSIONS

Start this exercise after you have completed lesson four of THE COMAL COMPUTER TUTOR [TUTORIAL DISK]. From Comal Programming mode enter the NEW command, clear off the screen, and enter the AUTO command. Now type in this program:

```
PRINT "THIS PROGRAM COMPUTES SALARY"
PRINT
INPUT "ENTER CURRENT WAGE PER HOUR: ":WAGE
PRINT
INPUT "ENTER NUMBER OF HOURS WORKED: ":HOURS
PRINT
SALARY := WAGE * HOURS
PRINT "GROSS SALARY IS $";SALARY
```

RUN the program. What is the name of the first varible?

_____

What is the name of the second variable?

_____

Notice the "assignment operator" (:=) between the variable SALARY and the variables WAGE and HOURS. It tells the computer to do what is on the right side of it, and to store the results in the variable that is on its left side, SALARY. The star (*) is used in place of x to tell the computer to multiply WAGE by HOURS. Most computers use * for multiplication to avoid confusion with the letter x. In this case the computer will multiply the number you gave it for WAGE by the number you gave for HOURS and store the result in the variable SALARY. Important point: The computer does what is on the right side of the assignment operator (:=) and and then stores the results in the variable that is on the left side of the assignment operator.

Why don't you try your hand at writing a program that includes an assignment expression. It can be as simple as a program that assigns the sum of the variables FIRST and SECOND to the variable SUM, such as:

```
SUM := FIRST + SECOND
```

(of course you will have to supply INPUT statements to get values for the variables) or it can be something more original. You may separate your INPUT statements with blank PRINT statements as in the example above if you wish. Begin by entering the NEW command and the AUTO command. Keep the program short. When you have one that does what you want, copy the program here:

EXERCISE 4: SELF-TEST

1. Name the variables in this assignment expression:

```
NECK'SIZE := HAT'SIZE / 2
```

_____

_____

2. What is the symbol for the "assignment operator"?

_____

3. Which part of the assignment statement stores the results of a calculation -- the variable on the right or left side of the assignment operator?

_____

4. Why might COST'OF'GAS be a better-named variable than COSTOFGAS?

_____

5. Would either variable work the same? (YES / NO)

6. Look at this program:

```
COUNT := Ø
INPUT "PLEASE ENTER A NUMBER: ":NUMBER
COUNT := COUNT + NUMBER
PRINT "THE COUNT NOW IS";COUNT
```

If you enter the number 2Ø, what would the count be? _____

7. If the first statements was COUNT := 1Ø what do you think the last line would print if your entered a 2Ø for the variable NUMBER?

8. Look at this program statement:

```
COUNT :+1
```

If the count was 2Ø before this statement, what do you think it would be after the statement is carried out? (You might want to try it out if you have any doubts!)

_____

EXERCISE 5: FUN WITH THE TURTLE

Load COMAL from the COMAL SAMPLER DISK and press D to get into the demo mode rather than the programming mode. You can see the "turtle" carrying out the commands that appear in the upper left-hand corner of the screen. Each of these commands is a separate "procedure" (more about them in the next exercise). After you have watched the turtle at work for a few demonstrations, press the RUN-STOP key. Did the demonstration stop? (YES / NO)

Now type in this command:

```
SETGRAPHIC 0
```

What happened when you entered this command?

This command activates the turtle graphics system. Now enter

```
FORWARD 70
```

What happened?

Now enter

```
LEFT 90
FORWARD 70
```

Did the turtle make a 90-degree turn then move forward again? (YES / NO)

Enter these same two commands again.

Did the turtle do what you expected? (YES / NO)

To complete drawing a box, enter the two commands one last time.

Now enter the command CLEAR

What happened?

When the turtle graphics system is activated, the CLEAR command clears off the screen. Enter the HOME command. Notice that the "home" position for the turtle is in the middle of the screen.

Now enter the command BACK 100

What happened?

When the demo program was running, one of the procedures was

called TREE. Once you have run a program, the procedures can be "called" directly. Try that now. Enter the procedure call:

```
TREE(50)
```

Entering this command told the computer to execute the procedure named TREE. 50 is the value that the procedure uses to determine how large a tree to draw. Enter this procedure call

```
TREE(20)
```

Did you get a smaller tree? (YES / NO)

How about asking for a tree of size 100. Did this procedure call do what you expected? (YES / NO)

We can plant a "rocket" in the middle of this tree by calling on another procedure in the demo program. Enter the procedure call:

```
ROCKET
```

While we are calling procedures, why don't you try:

```
SPINSQUARES(20)
```

The SPINSQUARES procedure also contains a variable that determines the size of the square. 20 is the value that you passed to this procedure.

Notice that this procedure continues on and on. Press the RUN-STOP key. This command not only stops the program, it cancels the turtle-graphics system. You should see a complete list of the commands that you have entered so far:

```
SETGRAPHIC 0
BACK 100
TREE(50)
TREE(20)
TREE(100)
ROCKET
SPINSQUARES(20)
```

You can get back into the turtle-graphics system by entering the SETGRAPHIC 0 command again [or just SETGRAPHIC will do it]. Do that. Then experiment a bit with the FORWARD, RIGHT, BACK, HOME, CLEAR, and perhaps some procedure calls such as SPINSQUARES(90), TREE(30), etc. You can stop the program with the RUN-STOP key at any time.

EXERCISE 5: SELF-TEST

1. What command activates the turtle-graphics system?

_____

2. In which direction does the FORWARD command move the turtle -- always upwards?

_____

3. If LEFT 90 can be used to make a square, what command do you think might help draw a triangle? (You might want to try your guess.)

_____

4. How do you get out of the graphics mode?

5. What was the purpose of the number in parentheses after the procedure TREE?

_____

6. The value inside the parentheses can vary from one call of the procedure to the next. What is the name given to programming structures that can hold various values?

_____

7. What is the function of the HOME command?

8. What do you think this program would do?

```
SETGRAPHIC 0
FORWARD 50
LEFT 120
FORWARD 50
```

9. If SETGRAPHIC 0 turns on the turtle, what do you think SETTEXT does? (Try it!)

10. Can you guess what the command hideturtle DOES?

_____

11. TURTLESIZE defines the size of the turtle from 0 to 10. Which number do you suppose makes the turtle smallest?

_____

12. As you might guess, there are quicker ways to get into the turtle-graphics system and get back to the regular text mode without having to enter SETGRAPHIC 0 and SETTEXT. There are four function keys on the right side of the keyboard. Which one automatically activates the graphics system? (f1 / f3). And which one returns the computer back to text mode? (f1 / f3) You may have

to experiment to answer this question.

13. The turtle can move without leaving a trail. Which command permits this (PENUP / PENDOWN). Verify your guess with experimentation.

14. The color of the border can change with BORDER commands. What colors result from

BORDER 5 _____

BORDER 7 _____

BORDER 14 _____

15. If HIDETURTLE hides the turtle, what do you suppose SHOWTURTLE does?

_____

EXERCISE 6: PROCEDURES FOR MODULARITY

Lesson 6 in THE COMAL COMPUTER TUTOR [TUTORIAL DISK] should be completed before starting this exercise. After you have entered [from COMAL] the NEW command and the AUTO command, type in this short program:

```
GET'NUMBERS
FIND'AVERAGE
PRINT'RESULTS
```

Get out of the automatic line-numbering system by tapping the RETURN key a second time, RUN this incomplete program, and write the ERROR MESSAGE that appears on the screen:

_____

_____

This is the typical error message that appears when you try to "call" or execute a procedure that doesn't exist. You already know that the procedure itself is just a list of statements grouped together to get a specific job done. Enter AUTO 40 and continue with:

```
//
PROC GET'NUMBERS
 INPUT "ENTER FIRST NUMBER: ":FIRST
 INPUT "ENTER SECOND NUMBER: ":SECOND
ENDPROC
```

Recall that the slash marks (//) in line 40 help to separate this first procedure from the statements that make up the "main" program. It will make the program listing easier to read.

Tap the RETURN key a second time to get out of the AUTO system and RUN the program. What question appears on the screen?

_____

Go ahead and enter a number. What number did you enter?

_____

When you press RETURN to enter this last number, a familiar ERROR MESSAGE should appear. Notice that the error message tells you where it found the error -- not at the line that calls the first procedure (there was nothing wrong with it). Rather, the error is at line 20. Important point: The computer carries out the statements of the main program, one at a time, until it reaches the last statement -- nothing new here. The first statement told the computer to "do" procedure GET'NUMBERS. You notice that this part of the program went just fine. The next statement of the main program was to do the procedure named FIND'AVERAGE. The computer couldn't find this procedure (because we have written its statements yet) and so the program stopped here with the familiar error message.

LIST the program again and enter the appropriate AUTO command so that you can continue entering program statements. Now add the code for these remaining two procedures:

```
//
PROC FIND'AVERAGE
 SUM := FIRST+SECOND
 AVERAGE := SUM / 2
ENDPROC
//
PROC PRINT'RESULTS
 PRINT
 PRINT "THE AVERAGE OF";FIRST;"AND";SECOND;"IS";
 PRINT AVERAGE
ENDPROC
```

After entering the last statement, RUN the program to test it. LIST the program again. Notice that when you LIST a program, all the statements inside a procedure are automatically indented. Enter the EDIT command. Are the statements indented now? (YES / NO) Many programmers prefer to make changes in a program that has been EDITed rather than LISTed, especially in long program statements that wrap around to the next line.

Let's add a mistake to the program. Enter the RENUM command, then LIST the program again. Notice that the last statement number is 190. Delete that statement by typing

```
DEL 190
```

and entering this command as usual. LIST the program again. Is statement 190 gone? (YES / NO) RUN the program again and write the ERROR MESSAGE that appears on the screen:

If you forget to include ENDPROC to end the procedure, this is the message that occurs. Remember, procedures need to start with the word PROC and end with the word ENDPROC.

Here is a new command. Enter the command CAT right now. What happened?

You can slow down the catalog listing by holding the control key (marked CTRL). Enter the CAT command once more, this time holding the CTRL key down. Notice that you cannot find the program that you have been working on. That doesn't mean the program has vanished. Enter the RUN command. Is your program still there? (YES / NO) After running the program enter this command

```
SAVE "AVERAGE"
```

Now get a catalog listing with CAT again. Your program should be the last one on the disk. You can remove programs from the catalog

listing with this command--try it:

```
DELETE "Ø:AVERAGE"
```

Now list the catalog again with the CAT command. Is your program gone? (YES / NO)

From now on you should save copies of your best programs on a blank diskette. If the disk is brand new, right out of the box, it will have to be "formatted" first. The teacher will show you how to do this or will give you a disk that is already formatted. If you want to format a new diskette now, follow these steps:

1. Open the disk drive door, slide a new disk in, and close the door.

2. Type:

```
PASS "NEWØ:GORDON,Ø1"
```

and enter this with the RETURN key. (Use your own name in place of GORDON and be sure to use a zero (Ø) not an O after the word NEW.)

3. The disk drive will whir for a while. When it stops and the red light goes out, the cursor will appear on the screen. If you want to see a catalog listing of the new, empty diskette, enter the CAT command.

4. The disk is ready to record programs. Remember, a program can be saved with

```
SAVE "AVERAGES"
```

using, of course, any name for your program that you wish.

It can be both loaded into the computer and started running with the command

```
CHAIN "AVERAGES"
```

If you want to just load the program, enter:

```
LOAD "AVERAGES"
```

after which you may either LIST the program or RUN it.

EXERCISE 6: SELF-TEST

1. What word begins all procedures?

_____

2. What word ends all procedures?

_____

3. What is the purpose of slash marks (//) between procedures?

_____

4. What error message will you get if the main program tries to find a procedure which does not exist (or which doesn't match the spelling in the main program!)

_____

5. What error message can you expect if you forget to open a procedure with the word PROC or forget to end it with ENDPROC?

_____

6. Look at this program. What is the first variable that needs a value?

```
ASK'THE'QUESTIONS
PRINT'THE'RESPONSES
//
PROC ASK'THE'QUESTIONS
INPUT "HOW OLD ARE YOU?":AGE
INPUT "WHAT IS YOUR HEIGHT IN INCHES?":HEIGHT
ENDPROC
//
PROC PRINT'THE'RESPONSES
PRINT "YOU ARE";AGE;"YEARS OLD."
PRINT "YOU ARE";HEIGHT;"INCHES TALL."
ENDPROC
```

_____

7. How many procedure calls does the main program have?

_____

8. Do you think this program has been LISTed or EDITed as shown above?

_____

9. What command would you enter to delete line number 5Ø of a program?

_____

10. The command DEL 10-40 will delete lines 10 through 40. What will DEL 150-200 do?

_____

11. How can you save a program named SALARY? (What command would you enter?)

_____

12. What command would you enter to get a catalog listing?

_____

13. What command would you enter to load a program named GAME?

_____

14. What command would erase LOUSY from the disk?

_____

EXERCISE 7: USING THE CASE STATEMENT

This exercise will give you practice using the CASE statement. You should have completed lesson seven in THE COMAL COMPUTER TUTOR [TUTORIAL DISK] before beginning this exercise.

You already know that the CASE statement begins with the word CASE and ends with the word ENDCASE. You understand how the CASE statement works. Enter this program:

```
DIM NAME$ OF 20
INPUT "WHAT IS YOUR NAME? ":NAME$
GET'FAVORITE'COLOR
SAY'GOODBYE
//
PROC GET'FAVORITE'COLOR
 DIM COLOR$ OF 10
 PRINT
 INPUT "WHAT IS YOUR FAVORITE COLOR? ":COLOR$
 CASE COLOR$ OF
 WHEN "RED"
  PRINT NAME$,", YOU LOVE ADVENTURE."
 WHEN "BLUE"
  PRINT NAME$,", YOU ARE LOYAL."
 WHEN "YELLOW"
  PRINT NAME$,", YOU ARE CHEERFUL."
 WHEN "GREEN"
  PRINT NAME$,", YOU LOVE THE OUTDOORS."
 WHEN "BLACK"
  PRINT NAME$,", YOU'RE EITHER CLASSY OR STRANGE."
 OTHERWISE
  PRINT NAME$,", I DON'T RECOGNIZE THAT COLOR."
 ENDCASE
ENDPROC
//
PROC SAY'GOODBYE
 PRINT
 PRINT "DON'T TAKE THIS SERIOUSLY."
 PRINT "GOODBYE FOR NOW."
ENDPROC
```

Notice that the main program consist of only four statements -- a DIM statement, an INPUT statement, and two procedure calls. The DIM stands for "dimension" and tells the computer to set aside enough space to hold the largest expected name for the variable NAME$. Most names are less than 20 characters long. The DIM statement for COLOR$ serves the same purpose. We don't expect the user to enter a color that is more than 10 characters long. Also notice that the variables that will accept letters instead of numbers (NAME$ and COLOR$) have a dollar sign as their last character. Do you think the following variable will accept letters or numbers?

```
INPUT "ENTER A COLOR: ":COLOR
```

How many letters can be used for the following variable?

```
DIM COLOR$ OF 4
INPUT "ENTER A COLOR: ":COLOR$
```

_____

Now enter this command

```
LIST 10-200
```

What is the first line number on the screen? _____

What is the last line number you see? _____

Enter this command

```
EDIT 10-200
```

Compare what is on the screen right now with the program listing. Use the cursor control keys to make any corrections necessary, then press the RETURN key for that line to make the correction permanent. If you wish, change some of the PRINT statements to say something you prefer. Here's another way to replace any program line: get the cursor to the bottom of the screen, type in the line number and the program statement you wish to use as a replacement, hit the RETURN key. When you LIST or EDIT the program, notice that the statement that you typed in at the bottom of the screen took the place of the one originally in the program. Important point: there are two ways to modify a program. If you use the cursor control keys, you can make changes right in the program listing. Or you can re-enter the program line at the bottom of the screen.

The OTHERWISE part of the CASE statement guards against surprise input. If you wish, delete this line number (with the DEL command), then RUN the program again. When it asks for a color, enter PINK and notice the error message. Important point: take advantage of COMAL's OTHERWISE part of the CASE statement to prevent your program from "crashing" from surprise input!

EXERCISE 7: SELF-TEST

1. How many characters can be used for the employee name according to this DIM statement?

```
DIM EMPLOYEE'NAME$ OF 25
```

_____

2. What is the advantage of including the OTHERWISE part of the CASE statement?

_____

3. What will LIST 40-120 do?

_____

4. What will DEL 40-120 do?

_____

5. What variable does this CASE structure test?

```
CASE CHOICE OF
WHEN 1
 PRINT "CORRECT!"
WHEN 2,3,4
 PRINT "SORRY.  YOU BLEW IT."
OTHERWISE
 PRINT "NO SUCH CHOICE!"
ENDCASE
```

_____

6. What variable does this CASE structure test?

```
DIM CHOICE$ OF 5
INPUT "GIVE ME YOUR ANSWER TRUE..":CHOICE$
CASE CHOICE$ OF
WHEN "Y","YES","YEAH"
 PRINT "I SEE THAT YOU AGREE."
WHEN "N","NO","NOPE"
 PRINT "I GUESS YOU DON'T AGREE."
OTHERWISE
 PRINT "I DON'T UNDERSTAND YOUR ANSWER."
ENDCASE
```

_____

7. Write a program that uses the CASE structure to do something that you wish. If you are really brave, try to program today's horoscope. Each alternative of the CASE statement could be many PRINT statements, or it could simply be a call to do a procedure that prints the information as in this part of a program:

```
DIM SIGN$ OF 10
INPUT "WHAT IS YOUR SIGN? ":SIGN$
CASE SIGN$ OF
WHEN "CAPRICORN"
 GET'CAPRICORN
WHEN "PICES"
 GET'PICES
etc.
etc.
```

You would then have procedures named GET'CAPRICORN, GET'PICES, etc. which would have the PRINT statements to produce the appropriate message. If you write such a program, be sure to save a working copy of it on your disk.

8. Name two ways that you could change a word in a print statement.

a)_____

b)_____

9. Can you include more than one test with the WHEN part of a case statement? (YES / NO)

10. How many tests are included in the WHEN part of this CASE statement?

```
CASE SCORE OF
WHEN 10,9,8,7
 PRINT "GUESS WHAT?  YOU PASSED!"
OTHERWISE
 PRINT "BLEW IT AGAIN..."
ENDCASE
```

_____

EXERCISE 8: USING THE FOR, REPEAT, AND WHILE STRUCTURES

If you have finished lesson eight of THE COMAL COMPUTER TUTOR [TUTORIAL DISK], clear off the screen, enter the NEW command, and get into the AUTO system. This exercise will focus on three ways of saving much programming time and space through the use of structures that permit you to repeat certain steps automatically.

It is wise to tell the computer to repeat a series of program statements:

1. When you know that a group of steps needs to be done a definite number of times.

2. When the steps needs to be repeated until a condition becomes true.

3. When you need to repeat steps only if a condition remains true.

Here's an example of the first type of repetition. The teacher tells you that you have to write I WILL NOT CHEW GUM IN SCHOOL 100 times. This program will do that for you:

```
FOR COUNT := 1 TO 100
 PRINT "I WILL NOT CHEW GUM IN SCHOOL"
ENDFOR
```

Enter this program and RUN it. That's somewhat better than having to write 100 separate PRINT statements, isn't it? In this program COUNT is the variable that control how many times the "loop" of PRINT statements is done. In fact, COUNT is the "loop control variable." It keeps track of how many repetitions of the step (or group of steps) have been completed. Add this statement to the program between lines 20 and 30:

```
25 PRINT "THIS IS LOOP NUMBER ";COUNT
```

Remember, to add such a statement, just get the cursor to the bottom of the screen and type in the line number and statement. Now LIST the program again. Does your listing show a line number 25 between lines 20 and 30? (YES / NO) Now enter the RENUM command and list the program again. Are the program statements numbered by 10 once more? (YES / NO) Statement number 30 asks the computer to print the current value in COUNT. Every time this loop is carried out, the value of COUNT increases by one. Run the program again and see.

What statement begins the FOR loop?

_____

What statement ends the FOR loop?

_____

Sometimes it is necessary to repeat a group of steps until a condition becomes true. You don't know ahead of time how many

times the steps have to be repeated. Passwords provide a good example of when this type of loop is useful. All that is needed is a cue for getting out of the loop. Look at this short program:

```
DIM WORD$ OF 30
REPEAT
 INPUT "PLEASE ENTER PASSWORD: ":WORD$
UNTIL WORD$="SECRET"
```

In this program SECRET is a password. Only if the user knows this password will the computer let him or her out of the loop. Enter the program and run it. Notice that it will accept words all day long until you enter the word that makes the UNTIL condition true. Most password-protected programs use some variation of this idea.

Here's another example to enter and run.

```
REPEAT
 INPUT "ENTER NUMBER ==> ":CHOICE
UNTIL CHOICE = 2
PRINT "THAT IS THE CORRECT CHOICE."
```

Again, notice that the loop will continue UNTIL a condition becomes true. Important point: the REPEAT..UNTIL structure contains an EXIT condition.

It may be dangerous to use a REPEAT..UNTIL loop for reading data. What is the EXIT condition was missing? The computer would not be able to leave the loop! The program would keep executing the loop forever. The WHILE statement contains an ENTRY condition -- the computer won't even enter the loop unless the entry condition is true. In this next program, EOD is the standard "end of data" test which COMAL and other higher languages recognize. The WHILE statement asks the program to retrieve values as long as there are more values to retrieve:

```
WHILE NOT EOD DO
 READ VALUE
 TOTAL:+VALUE
ENDWHILE
PRINT "THE TOTAL IS";TOTAL
DATA 23,34,45,56,67,78,89
```

Can you guess what would have happened if there was no data and we had used a REPEAT statement instead of a WHILE statement? With the WHILE statement, the program doesn't crash, because the loop is not even entered unless a condition is true -- in this case only as long as there is not an "end of data" signal. Our entry condition told the computer to enter this loop only if we were NOT at the END OF DATA.

Also notice the statement:

```
TOTAL:+VALUE
```

The assignment statement simply adds the value it found to the total. Another way to write this assignment statement (and it

would be perfectly acceptable) is:

```
TOTAL := TOTAL + VALUE
```

The computer would still do the same job -- take the current value of TOTAL and add VALUE to it. Languages such as BASIC and even Pascal use the second method. COMAL permits both methods.

EXERCISE 8: SELF-TEST

1. Write a program of your own design that illustrates the FOR loop. When you are satisfied that you have one that works well, save it on your diskette.

2. Write a program that demonstrates the REPEAT structure. Save a copy of your best work.

3. When should the FOR loop be used?

_____

4. Which type of loop has an exit condition?

_____

5. Which type of loop has an entry condition?

_____

6. Which statement would you prefer as a safeguard against unauthorized access to the rest of the program?

_____

7. Which type of loop is safer for reading input information?

_____

8. Write a program that uses the WHILE statement and the NOT EOD condition to read scores, average them, and print out the results. Remember, DATA statements must accompany any program that uses the READ statement.

9. Write a program that asks for the word that is to be used as the password for that day, then use an appropriate statement to guard against unauthorized use of the rest of your program. As soon as the password is chosen, have the program clear off the screen so the next user doesn't see it sitting there on the screen (it would not be a very useful password in that case!). This program fragment uses PRINT CHR$(147) which is the command to clear the screen. It may help you get started:

```
DIM PASSWORD$ OF 15
DIM WORD$ OF 15
INPUT "WHAT PASSWORD DO YOU WANT TO USE? ":WORD$
PASSWORD$:=WORD$
PRINT CHR$(147)
//
//  (put any program here)
```

Be sure to save a working copy of your work.

EXERCISE 9: USING THE IF STATEMENT

When you have finished lesson nine of THE COMAL COMPUTER TUTOR [TUTORIAL DISK], enter and run this program that uses the IF structure.

```
PRINT CHR$(147)
DIM REPLY$ OF 6
DIM PASSWORD$ OF 6
COUNT := 1
PASSWORD$ := "SENTRY"
REPEAT
 PRINT "THIS IS TRY #";COUNT
 INPUT "ENTER PASSWORD: ":REPLY$
 IF REPLY$=PASSWORD$ THEN
  PRINT CHR$(147)
  PRINT "ACCESS CONFIRMED."
  GET'INSTRUCTIONS
 ELSE
  COUNT:+1
 ENDIF
UNTIL (REPLY$=PASSWORD$) OR (COUNT=4)
IF REPLY$<>PASSWORD$ THEN PRINT "WARNING!  ACCESS DENIED!"
//
PROC GET'INSTRUCTIONS
 PRINT "HERE ARE THE INSTRUCTIONS..."
 //
 //  any program here
 //
ENDPROC
```

This program uses the IF structure to give the user three tries to enter the correct password. If after three attempts the user still hasn't entered the correct password, we suspect foul play. Another IF statement is used to control a PRINT statement that will say

```
WARNING!  ACCESS DENIED!
```

The first IF statement is made of three parts: IF, THEN, and ELSE. IF something is true, THEN we will do something, ELSE we will do something else. Suppose the user enter the correct password on the first try. The program will:

1. clear the screen
2. print the message ACCESS CONFIRMED
3. and then call on the procedure GET'INSTRUCTIONS

On the other hand, if the user misses on the first try, we will:

1. add 1 to the count
2. enter the loop for a second try.

Notice that after the third try COUNT will be 4. Look at the program again. Notice that the UNTIL part of the loop specifies either one of two conditions which will let the user out of the loop. What are they?

_____

OR

_____

What is the purpose of the PRINT CHR$(147) statement at the beginning of the program?

_____

RUN the program and enter three wrong passwords. What message appears?

_____

RUN the program again. This time enter two wrong passwords, then the word SENTRY. Does the program do what you expect? (YES / NO)

Look at the comma and period keys on the computer. The sign above the comma means "less than." The one above the period means "greater than." When both of these are used together, the computer interprets it as "not equal to." On the basis of the second IF statement, if REPLY$ is not equal to the password, the program will print a warning message. Notice that the IF statement has only a single action. It is short enough to fit on a single line. The usual ENDIF is not required for short, single-action IF statements.

As you can see, the IF structure increases the decision-making power of your programs. Using these ideas, you can begin to write programs that are quite sophisticated. Be sure to save copies of your best work.

EXERCISE 9: SELF-TEST

1. Space for how many characters is reserved with this expression?

```
DIM REPLY$ OF 70
```

_____

2. This program counts all the numbers entered which are greater than 10. Add the missing sign in the IF statement.

```
COUNT:=0
REPEAT
 INPUT "ENTER A NUMBER: ":NUMBER
 IF NUMBER __ 10 THEN COUNT:+1
UNTIL NUMBER = 0
```

3. What is the exit condition for this loop?

_____

4. Name the two variables:

_____

_____

5. We want to add a statement that will tell us how many of the numbers which were entered were over 10. Write the PRINT statement that would give this information:

_____

6. Is the IF statement in this program "compound" or "simple"?

_____

7. Why does the IF statement need no ENDIF?

_____

8. How could you change this program to count all the numbers that were NOT exactly equal to 10? (Write the statement here.)

_____

9. How could you change the program to count all the numbers less than 10?

_____

EXERCISE 10: COLORS,TAB, AND ZONE

Enter the NEW command and clear the screen. What color is the background on the screen right now?

_____

What color is the border?

_____

Enter this direct command:

```
BORDER 6
```

Now what color is the border? _____

Enter PENCOLOR 1

What color is the cursor now? _____

Enter this command

```
BACKGROUND 113
```

What error message appears on the screen?

_____

Important: the colors for background, border, and pencolor range from 0 through 15.

Enter this short program from Comal Today, an excellent newsletter for those interested in COMAL:

```
BACK'COLOR:=0
REPEAT
 BACK'COLOR:=(BACK'COLOR+1) MOD 16
 BACKGROUND BACK'COLOR
 PRINT CHR$(147)
 FOR PENS:=1 TO 15
  PENCOLOR PENS
  PRINT "PENCOLOR";PENS;"BACKGROUND";BACK'COLOR
 ENDFOR
 WHILE KEY$=CHR$(0) DO NULL
UNTIL TRUE = FALSE
```

This program combines many of the statements we have been studying and throws in a few that are still new. The MOD function, for example, is not introduced until lesson 12. Notice first of all that the whole program is wrapped in a REPEAT..UNTIL structure. But what is the exit condition? UNTIL true = false??? Since true will never equal false this program will run until someone presses the RUN-STOP key or pulls the plug!

The PRINT CHR$(147) is no mystery. Why don't you enter that command directly right now...

```
PRINT CHR$(147)
```

What happened when you pressed the RETURN key?

_____

LIST the program again. Let's add a modification that will make the border the same color as the background. What variable holds the number for the background color?

_____

We can use this same variable in a statement for the border. Add this line number and statement:

```
45 BORDER BACK'COLOR
```

Now list the program again. Is line 45 in the right place? (YES/NO).

RUN the program again. Does the border color now match the background color? (YES / NO).

How could you renumber the program?

_____

What is the name of the variable that is holding the values for PENCOLOR (look at the FOR statement).

_____

What is the lowest value for this FOR loop?

_____

What is the highest value for this loop?

_____

Notice that when you run this program some color combinations are are unreadable, others are quite clear. You may want to jot down those combinations that you find particularly clear and pleasant.

Look at the WHILE statement. KEY$ can detect when a key is pressed. The statement:

```
WHILE KEY$=CHR$(Ø) DO NULL
```

tells the computer to do nothing as long as no key is pressed. As soon as a key (any key)is pressed, CHR$ is no longer Ø. (This is a handy program line to remember even if you don't fully understand it at present!). Be sure to save a copy of your best modification of this program.

Enter the NEW command and clear the screen. Now enter this short

program:

```
PRINT TAB(5),"HELLO"
PRINT TAB(15),"HELLO"
```

RUN the program. What is the purpose of the number in parentheses?

Enter and run this program:

```
PRINT CHR$(147)
FOR COUNT:=1 TO 20
 PRINT TAB(15),"C O M A L"
ENDFOR
```

There should be no surprises here. Now make these changes:

```
N:=1
PRINT CHR$(147)
FOR COUNT:=1 TO 20
 PRINT TAB(N),"C O M A L"
 N:+1
ENDFOR
```

After adding these changes you may renumber the program lines with the RENUM command if you wish. Try the program out now. Notice that the variable inside the parentheses of TAB starts out as 1 and grows to 20 because N keeps getting increased by 1 with every loop. Design a program that uses some of these effects.

Enter and run this program:

```
PRINT CHR$(147)
ZONE 5
WHILE NOT EOD DO
 READ VALUE
 PRINT VALUE,
ENDWHILE
DATA 12,23,34,45,56,67,78,89
DATA 34,34,45,56,67,78,89,90
```

Notice that ZONE 5 spaces the output five spaces apart because of the comma at the end of the PRINT statement. Change the ZONE to 10 and run the program again. Did you expect this? Try a ZONE of 20 and run the program again.

The ability to add color to your border, background, and characters, as well as space your output with the TAB and ZONE commands gives added versatility to your programming skills.

EXERCISE 10: SELF-TEST

1. What border, background, and pencolor do you think provides a clear and readable combination?

BORDER _____

BACKGROUND _____

PENCOLOR _____

2. How do you "insert" program statements in a program?

_____

3. Here is a small program:

```
PRINT "HAPPY",
PRINT "BIRTHDAY"
```

Notice the comma after HAPPY. What will this program print when you run it?

_____

4. If you add the command ZONE 10 at the beginning of the program, what will it print when you run it?

_____

5. If there is no ZONE command, how many spaces is a comma worth?

___

6. What is the purpose of this statement?

```
WHILE KEY$=CHR$(0) DO NULL
```

_____

7. Look at this program:

```
JOE := 0
BACKGROUND JOE
```

What will the program do?

_____

8. What will this program do?

```
FOR COUNT := 0 TO 15
 BORDER COUNT
ENDFOR
```

_____

9. The border colors flash by so quickly, it's hard to even see them... How could we slow things down a bit? Which do you think is the better modification?

A)
```
FOR COUNT := Ø TO 15
 BORDER COUNT
ENDFOR
FOR DELAY :=1 TO 5ØØ DO NULL
```

B)
```
FOR COUNT := Ø TO 15
 BORDER COUNT
 FOR DELAY := 1 TO 5ØØ DO NULL
ENDFOR
```

1Ø. Write a program that selects a good border, background, and pencolor, one which illustrates the TAB as well as ZONE functions. Save a copy of your best program.

* *

Remember, the best way to learn programming is to write programs. These lessons and tutorials have, we hope, been useful in providing ideas for your own programming efforts. There is, of course, much yet to learn. Lessons 11-2Ø continue where these left off. Good luck as you gain more knowledge and use your new skills!

EXERCISE 11 - SOPHISTICATED BOOLEAN TESTS

Boolean operators such as AND, OR, and NOT can perform tests that otherwise would require a complicated series of IF statements. Before we look at an illustration of these operators, let's take a look at how the individual letters of a word can be examined--we'll need that skill in our program. Enter these two lines:

```
DIM WORD$ OF 5
INPUT "ENTER A FIVE-LETTER WORD: ":WORD$
```

You can get out of the AUTO mode by pressing the RETURN key a second time. Now RUN the program. Notice that after you enter a five-letter word, the program ends as expected. What five-letter word did you enter? _____

Now enter this command:

```
PRINT WORD$(1)
```

What letter appeared in response to your command? ___

Enter this command:

```
PRINT WORD$(3)
```

Is WORD$(3) the same letter as the third letter of your five-letter word? (YES / NO). Important point: it is possible and easy to select a particular letter from a word. We will need to do that so that we can test to see if the first and last letter is the same--one of the tests to see if a word is a "palindrome." (A palindrome is a word that is spelled the same forwards and backwards.) This program will illustrate the AND operator because we need to know if the first AND last letter of a word are the same. Enter the AUTO 30 command and continue with the program as follows:

```
PRINT WORD$
IF (WORD$(1)=WORD$(5)) AND (WORD$(2)=WORD$(4)) THEN
 PRINT WORD$;"IS A PALINDROME."
ELSE
 PRINT WORD$;"IS NOT A PALINDROME."
ENDIF
```

RUN the program. Test it with the word MADAM or another palindrome of your choice. Notice that the use of the AND operator made an otherwise complicated test much easier by testing for two conditions at the same time. Incidently, also notice that we surrounded each condition with parentheses. You don't have to do this, but it is good programming practice, especially when you have three or more conditions being tested at the same time.

When you are ready to go on, enter the NEW and AUTO commands, then

```
PRINT CHR$(147)
```

That statement, of course, will clear the screen when we run this new program which illustrates the function of another boolean operator, the OR operator. Continue with

```
DIM REPLY$ OF 1
ASK
ANSWER
PRINT "END OF PROGRAM"
```

Notice that the main program simply clears the screen, and then calls on two procedures, ASK and ANSWER. Don't run the program yet. Continue with the procedures:

```
PROC ASK
 INPUT "IS IT HOT (Y/N)? ":REPLY$
 IF REPLY$="Y" THEN
  HOT:=TRUE
 ELSE
  HOT:=FALSE
 ENDIF
```

No mystery here. We simply ask a question and then set HOT to either TRUE or FALSE depending on the reply. The other two questions for this procedure are similar:

```
 INPUT "IS IT HUMID (Y/N)? ":REPLY$
 IF REPLY$="Y" THEN
  HUMID:=TRUE
 ELSE
  HUMID:=FALSE
 ENDIF

 INPUT "IS IT RAINING (Y/N)? ":REPLY$
 IF REPLY$="Y" THEN
  RAINING:=TRUE
 ELSE
  RAINING:=FALSE
 ENDIF
ENDPROC
```

That is the end of procedure ASK. Now comes the fancy IF statement in procedure ANSWER.

```
PROC ANSWER
 IF ((NOT HOT) AND HUMID) OR RAINING THEN
  PRINT "LOUSY DAY FOR BIKE RIDING."
 ELSE
  PRINT "LET'S GO BIKING."
 ENDIF
ENDPROC
```

Before you RUN this program, take another look at the IF statement. Notice that both NOT HOT as well as HUMID must be true, or RAINING must be true in order for LOUSY DAY FOR BIKE RIDING to be printed. For example, if it is hot but it is not humid, it will still be a good day for bike riding. RUN the program now and try it out for a hot day that is not humid, but that is rainy.

(Believe it or not, it is possible for the humidity to be low when it first starts raining!) What response do you get?

Try the program again for not hot, not humid, and not rainy conditions. What is the response this time?

This time specify hot, but neither humid nor rainy conditions. Now what is the response?

Very complex conditions can be tested with a single IF statement when all three boolean operators are used at the same time. To make it clear what is being tested, be sure to put parentheses around the conditions. The program will work without the parentheses, but it may not work as you had planned!

Exercise 11 - SELF TEST

1. What command would test to see if the first and second letter of a word were the same?

---

2. It is a good idea to separate the conditions that you are testing from the boolean operators such as AND, OR, and NOT. How do you "separate" each condition?

---

3. Write a program that will

1) clear the screen
2) dimension the variable SEX$ to accept six letters
3) ask if user is male or female
4) ask for an age
5) test both sex and age. If the sex if female and the age is between 10 and 15, then print the message ALLOWED IN GIRL'S LOCKER ROOM, otherwise it prints STAY OUT OF LOCKER ROOM! Be sure to end your IF statement with an ENDIF.

4. Write a program that will print the message

    ENJOY THE MOVIE

IF there is not homework, OR the homework is already done.

EXERCISE 12 - MOD AND DIV OPERATORS

When you have finished the tutorial on MOD and DIV assignment operators, clear out the memory with NEW and enter this direct statement:

```
PRINT 21 DIV 5
```

What number appears under the PRINT statement? ___

How many times will 5 go into 21? ___

Now type in this direct statement:

```
PRINT 21 MOD 5
```

What number appears under the PRINT statement? ____

What is the remainder after dividing 21 by 5? ____

Important point: MOD will supply the remainder of any division problem. DIV will supply the exact number of times one number will go into another. But can this information ever be useful? Yes. All calendar programs that keep track of the number of days in a month must account for 29 days in February every 4 years. Enter this short program:

```
INPUT "WHAT YEAR IS THIS?  ":YEAR
CASE (YEAR MOD 4) OF
WHEN Ø
 PRINT "FEBRUARY HAS 29 DAYS."
WHEN 1,2,3
 PRINT "FEBRUARY HAS 28 DAYS."
ENDCASE
```

If you divide the year by 4, the remainder will be only one of four possible values--Ø, 1, 2, 3. But there are only two possible answers for the question of how many days to expect in February--there are either 28 or 29. Here the CASE statement combines with the MOD function to give us the same response for three answers, and another response for the other possible answer. This program, in fact, is a part of the instructions inside those watches that display the calendar for each month of the year.

Let's try an application for the DIV statement. It estimates the outside temperature based on the number of chirps a cricket makes.

```
INPUT "NUMBER OF CHIRPS IN 1Ø SECONDS? ":CHIRPS
CHIRPS'PER'MINUTE:=CHIRPS * 6
TEMP:=(CHIRPS'PER'MINUTE+4Ø) DIV 4
PRINT "CURRENT TEMPERATURE IS";TEMP
```

Notice how this program makes use of the DIV operator. First it calculates the number of chirps per minute by multiplying the number of chirps in 1Ø seconds by 6--after all, trying to count between 2ØØ and 3ØØ chirps for a whole minute could get a little tiresome! Next the program adds 4Ø to this number and, with the

DIV operator, calculates the nearest whole number of this sum divided by 4. (That came from a formula someone must have spent many nights in the field trying to calculate!) It's easy, of course, to find whole numbers with the DIV operator. RUN the program and enter 40 for the number of chirps in 10 seconds. What should the current temperature be? _____

Try other numbers. Can you discover how many chirps a minute to expect when the outside temperature is 80?

EXERCISE 12 - SELF TEST

1. Which operator should be used if you need to find the quotient of a division without any remainder?

_____

2. Which operator should be used to find the remainder of any division problem?

_____

3. What is 7777 div 1ØØØ?

_____

4. What is 7777 div 1ØØ?

_____

5. How would you find the first number of a any 4-digit number?

_____

6. Write a program that determines a secret code number that is derived from adding the first number in a 4-digit number to the first two digits of that same number, which is added to the first three digits of that same number. Try your program on the number 4552 and see if you get 5Ø4.

7. A country holds single-term election once every 6 years. 1988 is an election year. Write a program that tells how far away the election is for any given year.

8. Reverse the cricket program to find out the number of chirps to expect at any given outside temperature (to the nearest ""whole"" chirp, of course.)

EXERCISE 13 - FUNCTIONS AS SUBPROGRAMS

You have already seen some of the built-in functions in operation if you have finished the tutorial for this topic. Let's try a few. Remember, a "function" finds and returns some value. If, for example, you want the absolute value of a number returned to you, the ABS function does the job. Enter:

```
PRINT ABS(-342)
```

What number appears under the PRINT statement? _____

Enter:

```
PRINT ABS(342)
```

Does the same number appear? (YES / NO). Similarly, when you ask the computer to:

```
PRINT LEN("MARCH")
```

it will find out how long this word is, and return it to you. Try it. What number appears under the PRINT statement? _____

The LEN function returns the length of a word or sentence. Try it again with a whole sentence. What sentence did you use the LEN function on?

_____

What was the length of that sentence as the LEN function determined and returned it?

_____

Do you think the LEN function always returns the exact number of characters and spaces between quotation marks? (YES / NO).

Look at this function:

```
PRINT RND(1,100)
```

What number appears under the PRINT statement when you enter this direct command?

_____

That is a random number between 1 and 100. Use the SHIFT keys and the CRSR keys to move the cursor up three spaces so that it is once again sitting on the P of PRINT. Now press the RETURN key again. Notice that the number changes. It has returned another random number between 1 and 100. You might want to try that several times. The RND function finds a random number between the two "arguments" that you supply in parentheses. If the arguments are 1 and 100, then it will find a random value between 1 and 100. If the arguments are (50,500), then it will find a random number between 50 and 500. Look at the arguments in this function call:

```
PRINT RND(1,10)
```

This means that the computer will find a random number between _____ and _____.

The function SQR returns the square root of the argument that is passed to it. For example, SQR(16) returns a 4. Try it. Enter PRINT SQR(16). What number appears under the PRINT statement?

_____

What happens, however, if you need a function that doesn't exist? Just tailor make your own! That's right. Say, for example, that you needed a function that would find the hypotenuse of a right triangle given the two legs. The hypotenuse will be the square root of the number that results from adding the squares of each leg. This is how the function HYPOTENUSE could be invented:

```
FUNC HYPOTENUSE(X,Y)
 RETURN SQR(X*X + Y*Y)
ENDFUNC
```

That is all there is to it. This function will expect two numbers for its arguments whenever the main program calls on it. For example:

```
PRINT HYPOTENUSE(3,4)
```

With this call of the function, the arguments 3 and 4 are passed to the function. The function will square 3 to give 9, square 4 to give 16, add the squares together to get 25, and find the square root of 25. Try it yourself. Enter:

```
FUNC HYPOTENUSE(X,Y)
 RETURN SQR(X*X + Y*Y)
ENDFUNC
PRINT HYPOTENUSE(3,4)
```

Does it return a 5? (YES / NO). If you wanted, you could supply a whole series of Xs and Ys with a FOR loop--and while we are at it, print them all out in columns with the ZONE command. (Remember, the ZONE command makes a comma worth as many spaces as you specify.) Here's the whole program:

```
FUNC HYPOTENUSE(X,Y)
 RETURN SQR(X*X + Y*Y)
ENDFUNC
//
ZONE 20
FOR X:=1 TO 10
 FOR Y:=1 TO 10
  PRINT USING "##.##": HYPOTENUSE(X,Y),
 ENDFOR Y
ENDFOR X
```

When you RUN this program notice that it prints out 100 hypotenuse

values (every X value will be matched with each of the 10 Y values) and the whole series will be neatly printed out in two columns separated by 20 spaces because of the comma at the end of the PRINT statement.

Here's a practical application of a tailor made function. Pretend that you run the ZZZ automobile club. You need to calculate the cost of gas for each customer who plans a trip and who can give you his usual gas mileage. The function will divide the round-trip distance by the customer's usual mileage to give the number of gallons of gas that will likely be needed. It then takes the number of gallons and multiplies it by the current cost of gas per gallon to come up with an estimated cost. Here's the function and an accompanying program that uses the function:

```
FUNC COST(DISTANCE,GAS'PRICE,MILEAGE)
 RETURN (DISTANCE/MILEAGE *GAS'PRICE)
ENDFUNC
//
PRINT CHR$(147)
INPUT "ENTER ROUND-TRIP DISTANCE: ":D
INPUT "ENTER EXPECTED GAS PRICE PER GALLON: ":G
INPUT "ENTER USUAL GAS MILEAGE: ":M
PRINT
PRINT "EXPECTED GAS COST WILL BE";
PRINT USING "$###.##":COST(D,G,M)
```

The PRINT USING "$###.##": part of the last statement automatically rounds off the result to a dollars and cents format. (If you expected answers to be more th $999.99, you could change this to PRINT USING "$####.##":, or even more.) Notice that the arguments when the COST function is called are in the same order that this new function expects to get values--we wouldn't want it to accidently figure the cost of a 5000-mile trip by using $5000 for the cost of gas per gallon! Just remember that the arguments used in the function call must be in the same order as you built the function. Enter the program, run it, add any embellishments that you find useful.

EXERCISE 13 - SELF TEST

1. Here's a function that adds any two numbers.

```
FUNC ADD(FIRST,SECOND)
 RETURN FIRST + SECOND
ENDFUNC
PRINT ADD(3567,3546)
```

In this function call what is the argument that is passed to FIRST?

_____

2.
```
FUNC HYP(X,Y)
 RETURN SQR(X*X + Y*Y)
ENDFUNC
//
PRINT HYP(25,34)
PRINT HYP(12,32)
PRINT HYP(60,90)
```

In the third call of the HYP function, which argument is passed to Y?

_____

3.
```
FUNC DISTANCE(RATE,TIME)
 RETURN RATE * TIME
ENDFUNC
//
PRINT DISTANCE(60,1.25)
```

What number is passed to RATE?

_____

4. Here's a function that finds the square of a number.

```
FUNC SQUARE(NUMBER)
 RETURN NUMBER * NUMBER
ENDFUNC
```

Can you write a function that finds the cube of a number?

_____

_____

_____

5. Invent a function called INTEREST that accepts three values (AMOUNT, RATE, YEARS), which returns simple interest. Try it out with INTEREST(2000,0.112,20) or a call with some other arguments. If you wish, add a program that will input the amount into A, the interest rate into R, and the years into Y, then call INTEREST(A,R,Y) similar to the cost-of-gas program.

EXERCISE 14 - PROCEDURES WITH PARAMETERS

You already know that a large programming job can be broken down into smaller, easy-to-program modules called procedures. You have also looked at functions as subprograms. Each time a function was called, some "arguments" were used in parentheses. These were the values passed to the function so that it could use them to calculate and return some value. For example,

```
COST(D,G,M)
```

passed the values in three variables to the function COST which then used them to calculate and return a value. It was, of course, expecting three values:

```
FUNC COST(DISTANCE,GAS'PRICE,MILEAGE)
 RETURN (DISTANCE/MILEAGE * GAS'PRICE)
ENDFUNC
```

We can do the same thing with procedures--but with a great advantage. Instead of just returning a single value each time it is called, a procedure can return several things. Look how similar to a function this procedure with two "parameters" is!

```
PROC SHOW(NUMBER,SYMBOL$)
 FOR COUNTER:=1 TO NUMBER PRINT SYMBOL$,
ENDPROC
```

And you might expect the procedure call to use arguments in parentheses just as the function call did:

```
SHOW(SCHOLARSHIPS,"#")
```

This procedure call tells the computer to use the number in SCHOLARSHIPS and the symbol "#" for the procedure' s two parameters--NUMBER, and SYMBOL$. If SCHOLARSHIPS contained the number 5, for example, procedure SHOW would print

```
#####
```

across the screen. (They will print next to each other because of the comma at the end of the procedure's PRINT statement.) OK. Get ready to enter a program which graphs the number of scholarships for three different schools over a period of three years (1982-1984).

```
DIM SYMBOL$ OF 1
PRINT CHR$(147)
PRINT "MERIT SCHOLARSHIPS IN THREE SCHOOLS"
PRINT
FOR YEAR:=1982 TO 1984
 FOR SCHOOL:=1 TO 3 DO
  READ SCHOLARSHIPS
  CASE SCHOOL OF
  WHEN 1
   SHOW(SCHOLARSHIPS,"%")
  WHEN 2
   SHOW(SCHOLARSHIPS,"#")
  WHEN 3
   SHOW(SCHOLARSHIPS,"&")
  ENDCASE
 ENDFOR SCHOOL
 PRINT YEAR
 PRINT
ENDFOR YEAR
PRINT "%=EAST HIGH   #=WEST HIGH   &=LAKELAND"
//
PROC SHOW(NUMBER,SYMBOL$)
 FOR COUNTER:=1 TO NUMBER PRINT SYMBOL$,
 PRINT
ENDPROC
//
DATA 26,33,22 // DATA FOR 1982
DATA 20,35,26 // DATA FOR 1983
DATA 16,30,28 // DATA FOR 1984
```

Notice that because of the two FOR loops, the computer looks for three numbers (the number of scholarships for school 1, school 2, and school 3) three times (1982, 1983, 1984).

So, what are parameters? Nothing more than variables for a procedure. They accept arguments in the same way a function does. Unlike functions, however, a procedure can use and return more than a single value.

EXERCISE 14 - SELF TEST

1. What are the arguments in this FUNCTION call?

```
RND(100,150)
```

_____ _____

2. What is the argument in this FUNCTION call?

```
PRINT "THE TANGENT IS";TAN(NUMBER)
```

_____

3. What are the arguments in this PROCEDURE call?

```
DRAWBAR(20,"$")
```

_____ _____

4. What value will be passed to the procedure from this procedure call?

```
REVERSE'LETTERS("ABCDE")
```

_____

5. Here is a procedure as well a as procedure call.

```
PROC DRAWBAR(LENGTH,CHARACTER$)
 FOR COUNT:=1 TO LENGTH DO PRINT CHARACTER$,
ENDPROC
//
DRAWBAR(15,"$")
```

What value is passed to LENGTH?

_____

What value is passed to CHARACTER$?

_____

How many characters will the FOR loop print?

_____

6. Can you modify the SCHOLARSHIPS program to compare the number of students in three classes or some other variable at your school? Try it. Be sure to save a copy of your best work.

7. This procedure uses parameters for two numbers which it switches:

```
PROC SWITCH (FIRST,SECOND)
 HOLD:=FIRST
 FIRST:=SECOND
 SECOND:=HOLD
ENDPROC
```

And here is a procedure call:

```
SWITCH(F,S)
```

Which argument is used for FIRST?

_____

Which argument is used for SECOND?

_____

Write a procedure call that uses the arguments HIS'SCORE and MY'SCORE.

_____

EXERCISE 15 - INTRODUCTION TO ARRAYS

Enter this short program after finishing the two tutorials on ARRAYS.

```
PRINT CHR$(147)
ZONE 10
DIM SCORES(64)
FOR COUNT:=1 TO 64 DO
 SCORES(COUNT):=RND(80,100)
ENDFOR
//
FOR LOOK:=1 TO 64 PRINT SCORES(LOOK),
```

Be sure to include the comma at the end of the last line. Before you run the program, what is the purpose of the first line?

_____

How many spaces is the comma worth at the end of the last statement?

_____

What statement is responsible for making the comma worth that many spaces?

_____

Run the program now. Why are the random numbers between 1 and 64 arranged into four columns?

_____

Look at the variable SCORES. Even though there is only a single variable name, there are 64 different scores--SCORES(1), SCORES(2), SCORES(3)... SCORES(64). What variable is increasing from 1 to 64 to produce this many SCORES?

_____

Instead of using SCORES with COUNT to store 64 different values, we could use 64 separate assignment statements:

```
SCORE1:=RND(80,100)
SCORE2:=RND(80,100)
SCORE3:=RND(80,100)
SCORE4:=RND(80,100)
SCORE5:=RND(80,100)
ETC.
ETC.
```

but it would be extremely boring to write such a program even if it would do the same thing. A single FOR loop saves a lot of programming!

Look at the second FOR loop--the one that prints out the scores

generated by the first one. The counter variable is LOOK. It could just as well have been any other variable name. Enter the EDIT command and change the first LOOK variable, but not the second one, to a single letter, I. Run the program and write the error message that appears:

_____

Enter the EDIT command again and look at the last statement. Notice that by changing the counter variable from LOOK to I, but leaving the "index" variable for SCORES as LOOK, the program had no scores it could find by the name of SCORES(LOOK). Why? Because LOOK has a value of zero and there is no SCORES(Ø) to print! Change the index variable from LOOK to I, so that when I goes from 1 to 64, SCORES will go from SCORES(1) to SCORES(64) right along with it. Be sure you have a comma at the end of this statement, then run the program again. The program should generate and then print out all its random scores between 8Ø and 1ØØ.

Arrays let you use a single name in combination with an index variable to create, print out, or find any value in a whole list or array of values. Add this line to your program:

```
90 print scores(63)
```

Run the program again. What is the very last score printed on the screen?

_____

Is it the same as the 63rd score that is at the bottom of column three? (YES / NO). Change that last program line to

```
90 PRINT SCORES(1)
```

Now when you run the program check to see if the last value is the same as the first value. Is it the same? (YES / NO).

Now look at line 4Ø of the program. What is the "index variable" in this line?

_____

Important point: it is the number of the index variable, not its name, that is important in creating and listing arrays. COUNT can go from 1 to 64 for just as easily as a variable such as I.

Arrays of words ("strings" of letters or even single letters) behave exactly like arrays of numbers. Look at this program which uses RND(65,9Ø) for random numbers between 65 and 9Ø for the CHR$ codes of keyboard characters. (Every letter on the keyboard has a CHR$ number--the letter A is 65, the letter Z is 9Ø.) Other than this, the program is very similar to the previous number-array program.

```
PRINT CHR$(147)
ZONE 10
DIM LETTER$(64) OF 1
FOR COUNT:=1 TO 64
 LETTER$(COUNT):=CHR$(RND(65,90))
ENDFOR
//
FOR I:=1 TO 64 PRINT LETTER$(I),
```

Enter this program and notice the DIM statement. For words (or letters) we not only have to let the computer know how many items the array is to hold, we need to indicate how large the longest word will be. In this case no "string" is going to be longer than 1 character since we are only using single letters. Be sure the last line has a comma, then run the program.

Do you get four columns of letters? (YES / NO). If you wish, add

```
90 PRINT LETTER$(1)
```

and compare this letter with the first one that the program generates.

Take look at this short variation of the program in the tutorial:

```
DIM ITEM$(4) OF 7
FOR I:=1 TO 4
 READ ITEM$(I)
 PRINT ITEM$(I);"IS DATA ITEM NUMBER";I
ENDFOR
//
DATA "SUSAN","JOE"
DATA "DIANE","STEPHEN"
```

Enter and run this program--you may want to substitute some names of your own choice for the DATA items. The DIM statement tells the computer to reserve space for 4 names, each of which will be no more than 7 characters in length. (If a data item is longer than 7 characters, only the first 7 will be printed.)

In summary, a "string" array is just like an array or list of numbers. You "dimension" it with a DIM statement by including information about how long the longest string of characters will be as well as indicating how many items will be going by a single variable name.

Exercise 15 - SELF TEST

1. How many numbers will this numeric array hold?

```
DIM ID'NUMBER(2400)
```

_____

2. How many words will this string array hold?

```
DIM EMPLOYEE'NAME$(160) OF 25
```

_____

3. What is the length of the largest employee name in the above DIM statement?

_____

4. Look at this program fragment

```
INPUT "ENTER NAME: ":LOG'ON'NAME$
FOR I:=1 TO 200 DO
 IF NAME$(I)=LOG'ON'NAME$ THEN
  PRINT "CLEARED."
  INPUT "ENTER ID NUMBER: ":NUMBER
  FOR J:=1 TO 200
   IF ID(J)=NUMBER THEN
    PRINT "LOG ON COMPLETE"
    CHAIN "MENU"
   ENDIF
  ENDFOR J
 ENDIF
ENDFOR I
```

What is the counter variable for the first FOR loop?

_____

What is the index variable for the first FOR loop?

_____

What is the index variable for the second FOR loop?

_____

If the counter variable did not match the index variable, what type of an error message would appear?

_____

Do you think MENU is a procedure, a function, or a separate program?

_____

What two conditions must be true before MENU will load and run?

6. Look at these two DIM statements.

```
DIM LOG'ON'NAME$ OF 25
DIM NAME$(200) OF 25
```

Which statement dimensions an array?

_____

How many letters will the first DIM statement accept?

_____

EXERCISE 16 - INTRODUCTION TO FILES

Information such as a person's name and address may be called a RECORD when it is stored in a computer. A collection of records is called a FILE. There are two basic types of files, SEQUENTIAL and RANDOM ACCESS. Both types can store the same kind of information. What makes them different from each other is the way the computer looks for and finds records. In a sequential file the computer looks at all the records, beginning with the first, until it runs across the one it is looking for. In a random access file the computer goes directly to the record that it wants to find.

The difference between the two types is similar to the difference between a selection of music on a recorded tape, and the same selection on a phonograph record. If you wanted to play the 8th selection on a tape recorder, you would first have to advance the tape past the first seven selections. But if you wanted to play the 8th selection on a phonograph record, you could simply pick up the arm of the record player and move it over to the beginning of the 8th band. You don't even have to guess which way is faster!

Both types of computer files are useful. Let's first look at a program that creates and then reads a SEQUENTIAL file. (IMPORTANT: don't run this program until you have a blank diskette that has been "formatted" to save programs already in the disk drive! You may use the disk that you have been saving other programs on.)

```
OPEN FILE 2,"NUMBERS",WRITE
FOR RECORDNUMBER:=1 TO 100 DO
 RECORD:=RND(80,100)
 WRITE FILE 2: RECORD
ENDFOR
CLOSE
ZONE 10
OPEN FILE 2,"NUMBERS",READ
FOR RECORDNUMBER:=1 TO 100 DO
 READ FILE 2: RECORD
 PRINT RECORD,
ENDFOR
CLOSE
```

Notice what this program does. First it opens a file that goes by a number and a name. This particular file happens to use 2 and the name NUMBERS. Also notice that this is a file to which we plan to WRITE some records.

There will be 100 records. Each one will consist of a random number between 80 and 100. As the computer generates these records it will save them to the file as NUMBERS on the disk. The file is then closed.

The next part of the program reads from the file that has just been created. Zone 10 will make the comma at the end of the PRINT statement worth 10 spaces so that the next part of the program, which READS from the NUMBERS file, will print these in columns 10 spaces apart. Then this file is also closed.

OK. Check your program listing. If the statements appear correct, put a disk for saving programs into the drive and close the door. Then RUN the program. It will take a few seconds to create 100 random numbers and write them on the disk, then open the file again and read and print each number that the first part of the program created, but soon you will see the numbers that were stored on the disk.

Why store information such as this on the disk? For a very simple reason. A disk can hold far more information than will fit into the computer at any one time. You could have a single disk full of records and several separate programs that could read those records and print out only the ones that you need. For example, one program could open a file that existed on a disk, read each record, and print only the names of the people taking a particular class. A different program could open the same file and pull out all those records that met some other condition. Both programs could do this even if there were too many records to fit inside the temporary memory in the computer (the program workspace).

EXERCISE 16 - SELF TEST

1. OPEN FILE 3,"NAMES",WRITE

What is the number of this file? _____

What is the name of the file? _____

Will information be taken out of this file or put into this file according to this OPEN statement?

_____

2. OPEN FILE 2,"EMPLOYEES",READ

Does this file already exist? (YES / NO)

Change this OPEN statement to one that would be able to put new information into the file:

_____

3. What is the difference between sequential access and random access files?

_____

_____

4. Look carefully at this program segment. Can you spot the error?

```
DIM NAME$ OF 15
OPEN FILE 2,"NAMES",READ
FOR I:=1 TO 1Ø
  INPUT "ENTER A NAME: ":NAME$
  RECORD:=NAME$
  WRITE FILE 2: RECORD
ENDFOR
CLOSE
```

What word needs to be changed?

_____

What should it be instead?

_____

5. Write the code that would read all the names that had been entered:

EXERCISE 17 - RANDOM ACCESS FILES

This exercise will use most of the concepts covered in all the previous lessons to produce an inventory program. It will introduce a final concept for this workbook: random access files. An inventory program should be able to create inventory categories that you want and let you tell how many of these items you already have on hand. Then you should be able to add to the the stock, subtract from it, and display any record in the file. This program adds two additional features: it lets you scroll through the entire inventory and it will also list all the items that fall below a "reorder" level that you set when first entering a new item. It doesn't have many "bells or whistles," but it does show how to open and use random access files. Answer the questions that accompany the program's explanation.

```
// INVENTORY PROGRAM
// Author: G. Shigley
// Written January 31, 1985
//
DIMENSION
REPEAT
 MENU
UNTIL FALSE
```

The // means that the INVENTORY PROGRAM statement:

1. Opens a file by that name
2. Is just a comment
3. Is an assignment statement

There are two procedures for this main part of the program. Name them:

_____

_____

That is the main program. It has two jobs: to dimension some variables that the program needs, and then to call on the MENU procedure. The MENU will be the central procedure to which each of the other procedures will return. Since we always want to return the REPEAT...UNTIL loop is a handy way of repeating the MENU procedure forever. Here's the code for the DIMENSION procedure:

```
PROC DIMENSION
 DIM ITEM$ OF 10
 DIM REPLY$ OF 1
 NUMBER'OF'RECORDS:=10
ENDPROC
```

Certainly not very complicated. Each item of the inventory is referred to by the variable name ITEM$. How many characters, according to this procedure, will the computer bother to remember as the name for each item? _____

The number of records for this inventory program is contained in

the variable:

_____.

Of course we could use larger numbers to dimension ITEM$ or for the number of records, but 10 characters for the name of inventory items and 10 items in our inventory will be enough to demonstrate the program. Let's take a look at the code for the MENU procedure.

```
PROC MENU
 REPEAT
  PRINT TAB(10);"INVENTORY PROGRAM"
  PRINT
  PRINT "1. ADD A NEW KIND OF ITEM"
  PRINT "2. DISPLAY   BY PART#"
  PRINT "3. ADD TO STOCK"
  PRINT "4. SUBTRACT FROM STOCK"
  PRINT "5. LIST ITEMS BELOW REORDER LEVEL"
  PRINT "6. INITIALIZE FILE"
  PRINT "7. SCROLL THROUGH INVENTORY"
  PRINT "8. QUIT"
  PRINT
  INPUT "NUMBER OF YOUR CHOICE? ":CHOICE
 UNTIL CHOICE > 0 AND CHOICE < 9
```

The TAB part of the statement tells the computer to move the cursor over 10 spaces from the left edge of the screen and then to print INVENTORY PROGRAM. Now look at the REPEAT..UNTIL statements. The program will print the list of choices and stop for some input from the user. What do you think will happen if the use enters a number that is either more than 8 or less than 1?

1. The program will "crash"
2. The menu will pop back up
3. The program will end

If you picked number 2, you are correct. The rest of the MENU procedure is nothing more than a CASE statement.

```
 CASE CHOICE OF
 WHEN 1
  CREATE'NEW'ENTRY
 WHEN 2
  DISPLAY'BY'NUMBER
 WHEN 3
  ADD'TO'STOCK
 WHEN 4
  SUBTRACT'FROM'STOCK
 WHEN 5
  LIST'REORDER
 WHEN 6
  INITIALIZE
 WHEN 7
  SCROLL
 WHEN 8
  END
 OTHERWISE
```

```
  NULL
 ENDCASE
ENDPROC MENU
```

That's the end of the procedure. If the user enters "1" as his or her selection, the computer will do procedure CREATE'NEW'ENTRY. What procedure will it do if the user enters a "7"?

So, the CASE statement determines which procedure to do, and the REPEAT..UNTIL simply guarantees that the user enters a selection that the CASE statement can use.

To add a new kind of item to the inventory we must open a file. After letting the program know the maximum size of each record (80 characters is more than long enough for our purposes), we will request some information with INPUT statements, and then write it onto the disk in a random file:

```
PROC CREATE'NEW'ENTRY
 OPEN FILE 2, "INVENTORY", RANDOM 80
 INPUT "INVENTORY NUMBER? :"NUMBER
 INPUT "NAME OF ITEM? ":ITEM$
 INPUT "CURRENT QUANTITY? ":QUANTITY
 INPUT "REORDER LEVEL? ":REORDER'LEVEL
 INPUT "PRICE? ":PRICE
 WRITE FILE 2, NUMBER: ITEM$,QUANTITY,REORDER'LEVEL,PRICE
 CLOSE FILE 2
ENDPROC
```

The user inputs four pieces of information and the computer then writes these to a random access file named INVENTORY. Notice how easy it is to make a random access file. All you have to add is the word RANDOM! Since this program will look up inventory items by their inventory number, we "attach" the four pieces of information in FILE 2 to NUMBER with WRITE FILE 2, NUMBER. How many records can we write? As many as we wish or as many as will fit on a disk! Writing information to a random file certainly appears easy. How can we find any record by its inventory number? Look:

```
PROC DISPLAY'BY'NUMBER
 REPEAT
  INPUT "INVENTORY NUMBER? ":NUMBER
 UNTIL NUMBER <= NUMBER'OF'RECORDS
 OPEN FILE 2, "INVENTORY", RANDOM 80
 PRINT "======================================="
 READ FILE 2, NUMBER:ITEM$,QUANTITY,REORDER'LEVEL,PRICE
 PRINT "DESCRIPTION..... ":ITEM$
 PRINT USING "QUANTITY ON HAND.###":QUANTITY
 PRINT USING "REORDER LEVEL...###":REORDER'LEVEL
 PRINT USING "UNIT PRICE $###.##":PRICE
 PRINT "======================================="
 CLOSE
 PRINT "PRESS ANY KEY FOR MENU"
 WHILE KEY$=CHR$(0) DO NULL
ENDPROC
```

The PRINT USING lets us line up the numbers without any other fancy commands. The last statement should be familiar. What will happen while the user does not press a key?

Notice that this procedure opens up the file and reads the information from any record that the user wants to see. As soon as this procedure is finished, the program continues (which means it starts running the MENU program again because we told it to run MENU forever.)

You can probably guess how we could add to the stock of a particular inventory item. Open the file, get out the particular record we want, change the number that tells how many of these items we have, write the new information back onto the disk and close the file. Here's the code:

```
PROC ADD'TO'STOCK
 OPEN FILE 2, "INVENTORY",RANDOM 80
 REPEAT
  INPUT "INVENTORY NUMBER? ":NUMBER
 UNTIL NUMBER <= NUMBER'OF'RECORDS
 READ FILE 2, NUMBER:ITEM$,QUANTITY,REORDER'LEVEL,PRICE
 PRINT "DESCRIPTION: ";ITEM$
 PRINT "QUANTITY ON HAND: ":QUANTITY
 PRINT "AMOUNT TO ADD? ":ADD
 QUANTITY:+ADD
 WRITE FILE 2, NUMBER: ITEM$,QUANTITY,REORDER'LEVEL,PRICE
 CLOSE FILE 2
ENDPROC
```

The statement that gets the job done, of course, is QUANTITY:+ADD. We look at the old value of QUANTITY, add to it the amount the user has entered under the variable name ADD, then write the record over using this new amount for the QUANTITY variable. In summary, adding to the stock of a particular inventory item is no more difficult than than opening the file, reading the desired record, incrementing its quantity by the amount you wish to add, and then writing the information back to the file.

You might guess that subtracting from stock is identical to the procedure for adding except the quantity is decreased. Which statement do you think would accomplish this if the variable holding the amount to subtract was called SUBTRACT?

1. QUANTITY:=QUANTITY - SUBTRACT
2. QUANTITY:-SUBTRACT
3. SUBTRACT:-QUANTITY

The answer is either of the first two. Here's the code for SUBTRACT'FROM'STOCK:

```
PROC SUBTRACT'FROM'STOCK
 OPEN FILE 2, "INVENTORY",RANDOM 80
 REPEAT
  INPUT "INVENTORY NUMBER? ":NUMBER
 UNTIL NUMBER <= NUMBER'OF'RECORDS
 READ FILE 2,NUMBER: ITEM$,QUANTITY,REORDER'LEVEL,PRICE
 PRINT "DESCRIPTION: ";ITEM$
 PRINT "QUANTITY ON HAND: ";QUANTITY
 INPUT "AMOUNT TO SUBTRACT? ":SUBTRACT
 QUANTITY:-SUBTRACT
 WRITE FILE 2,NUMBER:ITEM$,QUANTITY,REORDER'LEVEL,PRICE
 CLOSE FILE 2
ENDPROC
```

Listing all the items that have fallen below the reorder level depends on a IF...THEN statement. We look at the quantity currently in the record. If it is below the reorder level, we have that record printed on the screen. We might as well use PRINT USING statements for a clean look.

```
PROC LIST'REORDER
 ZONE 10
 OPEN FILE 2, "INVENTORY", RANDOM 80
 PRINT
 PRINT TAB(3);"ITEMS TO BE REORDERED"
 PRINT
 PRINT "======================================="
 PRINT "DESCRIPTION   QUANTITY   REORDER  UNIT"
 PRINT "OF ITEM       ON HAND    LEVEL    PRICE"
 PRINT "======================================="
 FOR NUMBER:=1 TO NUMBER'OF'RECORDS DO
  READ FILE 2, NUMBER:ITEM$,QUANTITY,REORDER'LEVEL,PRICE
  IF QUANTITY <= REORDER'LEVEL THEN
   PRINT ITEM$,
   PRINT USING "########":QUANTITY,
   PRINT USING "########":REORDER'LEVEL,
   PRINT USING "$####.##":PRICE
  ENDIF
 CLOSE FILE 2
 PRINT
 PRINT "END OF LISTING.  PRESS ANY KEY."
 WHILE KEY$=CHR$(0) DO NULL
 ZONE 0
ENDPROC
```

Scrolling through the inventory would be the same procedure without the IF...THEN statement. That is, we would look at all the records, not just those whose quantity was below the reorder level:

```
PROC SCROLL
 OPEN FILE 2, "INVENTORY",RANDOM 80
 FOR NUMBER:=1 TO NUMBER'OF'RECORDS DO
  READ FILE 2,NUMBER:ITEM$,QUANTITY,REORDER'LEVEL,PRICE
  PRINT "======================================="
  PRINT "DESCRIPTION: ";ITEM$
  PRINT "ON HAND:     ";QUANTITY
  PRINT "REORDER AT:  ";REORDER'LEVEL
  PRINT USING "UNIT PRICE $###.##":PRICE
  PRINT "======================================="
  INPUT "HIT RETURN FOR NEXT RECORD":REPLY$
  PRINT
 ENDFOR NUMBER
 CLOSE FILE 2
 PRINT
 PRINT "PRESS ANY KEY FOR MENU"
 WHILE KEY$=CHR$(0) DO NULL
ENDPROC
```

It is easy to destroy a file and start all over. But a warning message should let the user know what is about to happen, just to be sure that he or she really wants to start over with a new inventory.

```
PROC INITIALIZE
 INPUT "DESTROY THE FILE (Y/N)? ":REPLY$
 IF REPLY$="Y" THEN
  DELETE "0:INVENTORY"
  OPEN FILE 2,"0:INVENTORY",RANDOM 80
  ITME$="NONE"; ZERO=0
  FOR NUMBER:=1 TO NUMBER'OF'RECORDS DO
   WRITE FILE 2, NUMBER: ITEM$,ZERO,ZERO,ZERO
  ENDFOR
  CLOSE FILE 2
 ENDIF
ENDPROC
```

The command DELETE "0:INVENTORY" will delete from the disk the file that was named INVENTORY. Then it will create the new INVENTORY file and fill it with dummy records.

Well, that's the whole inventory program. Congratulations for getting all the way to the end.

TABLE OF CONTENTS

ISBN 0-928411-05-2