



BASIC for Beginners

Lesson II

By David W. Ostler

In a previous article [September 1987, Page 26], I covered variables and some of their uses in programming. This time I will cover more of the most used commands that will help you become good programmers.

Remember the old saying "Practice makes perfect"? It is even more true in programming. If you know about a command and do not practice it, you will probably forget about the command when it's needed most. Therefore, try all of these commands at least 10 times, to entrench them in your mind. Also, remember this series just covers some of the commands found in the Color Computer BASIC language, not the commands pertaining to drawing and graphics generation. Many programmers do not program for graphics, so I concentrated on the commands common to many of the different BASIC languages.

My son is able to translate programs for the Apple and Commodore computers for use in his school work. Many of the commands translate directly, with few exceptions.

REM (')

The apostrophe (') or REM symbol is the famous remark statement. It notifies the computer that all characters following the symbol are not commands and should be ignored by the computer's command interpreter.

Use REM statements to place remarks within the program body itself to embed programming notes that explain the use or function of particular portions of a program. This helps when trying to debug a program, which is the act of finding the location of a problem (called a bug) that has made itself known by returning a wrong answer or causing the

Dave Ostler is an IC layout designer and the systems manager for a CAD main-frame system. He teaches CAD and electronics at Guilford Technical Community College. Dave is married and has three children, Avis, Chuck and Erik.

program to crash. It also allows a programmer to know where various parts of the program are stored within the body of the program. This is helpful when you have various loops within a program.

This practice is useful for beginners and for expert programmers who are working with a very intricate program.

Some programmers prefer to use the REM statement instead of the apostrophe symbol. The proper syntax for this command is `10 GOTO 5 'THIS LOOP STARTS THE PROGRAM OVER.` The computer recognizes only the `GOTO 5` command and ignores the rest of the line.

Remember that any character you place in a program will use up memory, whether it's a command or remark and text. So, use remarks only where you need them, but use them!

CLEAR

The CLEAR command notifies the command structure that you want to set up an area of memory reserved for variable storage. It is normally used as `CLEAR xxxx`, where `xxxx` is the amount of memory you want to reserve. Or it can be used as `CLEAR xxxx, yyyy`, where `xxxx` is still the amount of memory you want to reserve, but `yyyy` is an area of memory you desire to protect from overwriting. The `yyyy` area is usually used to protect the BASIC memory area from any variable storage.

The `xxxx` figure is usually obtained by trial and error, until a satisfactory balance between variable storage and program area is reached.

Also, when you clear memory within the body of a program, all variables are cleared out. All numeric variables will now equal zero, and all string variables will have nothing in them after a CLEAR command is issued. Therefore, it is normal practice to issue a CLEAR command early in a program, unless you want to clear out all variables at a certain point in a program.

PRINT@

The PRINT@ command makes a program appear professional. It orders the computer to print text or graphics characters to the screen at the location desired. For the screen locations that are available, see your manual for the PRINT@ worksheet page.

Proper syntax is `PRINT@ xxx,` where `xxx` is a numerical value between 0 and 511. This numerical value directly relates to the screen location as found in the PRINT@ worksheet. The command is usually used as `10 PRINT@`

`128, "THIS TEXT WILL BE PRINTED".`

This would have been printed at screen location 128, and the text would have ended at screen location 152. Another enhancement of the PRINT@ command is to use it with the CLS command and to use the semicolon delimiter. Try this line and see what result it has:

```
10 CLS3:PRINT@ 227, "THIS  
TEXT WILL BE PRINTED";
```

The screen would be blue with normal text starting at screen location 227 and ending at screen location 251. The only difference between this line and the other one above is that the screen is blue with the text centered within the blue screen. Notice that blue is showing even after the last character of text is printed. Take out the semicolon and see what effect it has on how the line is printed.

Here are tips that will help you center text on a line:

- 1) Count the number of characters you want printed on the line. (Remember, a line can be no longer than 32 characters.)
- 2) Subtract that number from 32.
- 3) Take the remainder from Step 2 and divide by 2.
- 4) Add the amount obtained in Step 3 to whatever screen location line you want to print that text.

STRINGS

The STRING\$ command is used to create a 1 to 255 character string made up of the same character. This is useful when trying to create a title page, border, etc., to enhance a program's appearance.

Proper syntax for this command is `STRING$(xx,yy)`, where `xx` is the character desired. This character may be any of the ASCII characters, any of the graphics characters your computer can generate for screen use, or any characters your computer can send to various output devices, such as disk drives, printers, tape drives and modems. The `yy` value is the number of characters that you want to create.

SOUND

The SOUND command produces a tone from the speaker of the television or monitor. It can be used to notify you when input is needed or an error has been detected or made.

The proper syntax for this command is `SOUND x,y`, where `x` is a number between 1 and 255 and sets the pitch of

the tone. `Y` is a number between 1 and 255 and sets the length of the tone.

GOSUB

The GOSUB command forces the computer to jump to a defined line, which contains the desired subroutine, within the program. This is an unconditional loop that usually contains conditional loops nested within it; therefore, they are also called nested loops. A GOSUB subroutine must *always* end with the RETURN command. This command will force the computer back to the next command directly following the GOSUB command. The only exception is when the RETURN command is superceded by a GOTO or IF/THEN command.

The proper syntax for this command is `GOSUBxxxx`, where `xxxx` is the line number where the subroutine starts.

This command is useful when using a pause within a program, such as "Press any key to continue." You can place the GOSUB command at the end of the area where you want to pause the program. The program can then go to the subroutine and wait for the key press. After the key is pressed, it will return to the program command immediately after the GOSUB command.

Look at Listing 1 for an example of the GOSUB command.

CHR\$

The CHR\$ command converts a numerical value to a single character string. Use this when you want to send control codes to a software programmable printer or to print graphics characters to the screen or printer.

The proper syntax for this command is `CHR$(xxx)`, where `xxx` is the numerical value that is converted into a single character string.

PRINTUSING

The PRINTUSING command prints the text following it in the format that was selected. This format is specified by putting characters behind the PRINTUSING command. These characters can be found by looking in your manual under the PRINT command area.

The proper syntax for this command is `PRINTUSING "$###,###,##.##";B.` Assume a value of one million for the integer variable `B`. This particular format will print the integer variable `B` in the format of `$1,000,000.00`. Or, for a value of 10,000 for `B`, it will print `$10,000.00`. Note that no matter what the value is, it will be printed with two decimal places to the right of the period, and the dollar sign printed 12

spaces to the left. The commas will only be printed when the value is great enough to warrant it.

This command is useful when you want to print a numerical value or character string in a particular fashion. One use for the PRINTUSING command would be in a program that prints values in dollars and cents.

IF/THEN

The IF/THEN command tests variables to see if various conditions have been met. In standard BASIC the syntax is IF/THEN GOTO. But the Color Computer BASIC can shorten it by leaving off the GOTO command because it is assumed by the command interpreter.

Proper syntax is IF X = Y THEN 1000; when variable x equals the value of variable y, then the program will be forced to jump to Line 1000.

When multiple comparisons are to be made, you can use the ELSE command. The proper syntax for this use is IF X = Y THEN 1000 ELSE 2000 or IF X = Y THEN 1000 ELSE IF Y = Z THEN 5000 ELSE. . . . When x equals the value of y, then force a jump to Line 1000, or else force a jump to Line 2000.

You can see that you can compare

many different variables within a command line and keep memory requirements to a minimum.

Looking at Listing 1

Line 10 clears the screen, moves the cursor down two lines and prints the text.

Line 15 forces the program to go to the subroutine located at Line 1000.

At Line 1000, the cursor is moved down the screen four more lines, and the text "PRESS ANY KEY TO CONTINUE" is then printed.

At Line 1010, string variable B\$ is set equal to the key pressed. Only in this instance we want a key to be pressed to continue the program, and we don't care which key. If no key is pressed, this line will be repeated by the IF/THEN, ELSE command directly following the INKEY\$ command. (Note that IF/THEN, ELSE is a variation on the IF/THEN command. The ELSE command helps shorten up the command line so that multiple comparisons can be made.

Looking at Listing 2

Line 0 is a remarked line.

Line 5 clears 1,000 bytes of memory for variable storage, clears the screen

and prints the text at the specified locations.

Line 10 prints text at the location, allows the input of variable A and sounds a tone.

Line 20 prints a string of blanks at the location to clear out the previous text; then it prints new text at the same location, allows the input of variable B and sounds a tone.

Line 30 forces the program to go to the subroutine located at Line 1000.

Lines 100 to 130 all sound a tone, perform mathematical manipulation of variables A, B and C, then force a jump to Line 500.

Line 140 sounds a tone and jumps to Line 700.

Line 200 prints a string of blanks at the location to clear out the previous text, then prints new text at the same location, allows the input of variable B and sounds a tone.

Line 305 forces the program to go to the subroutine located at Line 2000.

Lines 310 to 340 all sound a tone, perform mathematical manipulation of variables A, B and C, then force a jump to Line 500.

Line 350 sounds a tone and forces a jump to Line 700.

Clearbrook Software Group

(604)853-9118



Information Management System



CSG IMS is THE full featured relational database manager for the Color Computer and OS9. The comprehensive structured application language makes CSG IMS the ideal development tool for sophisticated file-intensive applications.

- Interactive access to databases and quick queries.
- CSG IMS includes a recursive compiled language supporting program modules with full parameter passing.
- User defined screen and report formats.
- Record, index and file size almost unlimited.
- Text, BCD floating point (14 digits), short and long integer and date types.

CSG IMS for CoCo2/3 OS9 L1/2 (single user) \$169.95
 CSG IMS for OS9 L2 or 68000(multi user) \$495.00
 CSG IMS demo with manual \$30

ERINA - Symbolic User Mode Debugger for OS9

ERINA is a must for all serious assembler and C software developers. It lets you find bugs quickly by displaying the machine state and instructions being executed. You can set address and register break points, dump, search and change memory, assemble and disassemble code and many other things to numerous to mention. This program will pay for itself over and over by the time you save solving your bugs.

Requires 80 column display, OS9 L1/2 \$69.00

SERINA - System Mode Debugger for OS9 L2

SERINA is a debugger for OS9 system modules (device drivers, file managers, etc.). It allows you to trace execution of any system module, set break points, assemble and disassemble code and examine and change memory. There are special provisions for executing code with critical timing loops and for accessing I/O registers. A must for system programmers.

Requires CoCo3, OS9 L2, 80 col. terminal connected to /T1 or /T2 \$139.00

MSF - MSDos File Manager for CoCo 3/OS9 Level 2

MSF is a file manager which allows you to use MSDos disks directly under OS9. You don't have to change the format of the data before using it!

Requires CoCo 3, OS9 L2, SDISK3 driver \$45.00

Shipping: N. America - \$5, Overseas - \$10

Clearbrook Software Group

P.O. Box 8000-499
 Sumas, WA 98295



OS9 is a trademark of Microware Systems Corp., MSDos is a trademark of Microsoft Corp.

Line 500 clears the screen and prints text at the specified location.

Line 510 sets up a FOR/NEXT command loop and sets a value for Integer B. Note that the loop will count to five before going on to another part of the program.

Lines 520 to 590 first print graphics characters at the locations specified, then set up a timing loop so the graphics character will be displayed for a desired amount of time before the next character. See your manual for the characters that can be printed to the screen.

Line 600 sounds a tone and informs the FOR/NEXT command set up in Line 510 that the next value of B should be counted.

Line 610 clears the screen, sounds a tone and prints text at the location; immediately following the text, the integer variable C is printed out in the format set up by the PRINTUSING command.

Line 620 forces the program to jump to Line 300.

Line 700 clears the screen, prints text at the location, allows the input of variable A, sounds a tone and forces a jump to Line 20.

Lines 1000 to 1070 comprise a subroutine that prints text at various locations. They then use the INKEY\$ to determine which key has been pressed and send the program to the appropriate line to perform the proper mathematical manipulation. These lines are 100 to 130.

Lines 2000 to 2070 make up a subroutine that prints text at various locations, then uses the INKEY\$ to determine which key has been pressed and send the program to the appropriate line to perform the proper mathematical ma-

nipulation. These lines are 310 to 340.

(Questions or comments regarding this tutorial may be directed to the author at 901 Ferndale Blvd., High Point, NC 27260. Please enclose an SASE when writing for a reply.) □

Programming Exercises

Utilizing the methods presented, write a program that allows you to enter your name, street address, city, state and ZIP. This program should also allow you to call a subroutine that prints up a menu that lets you recall each variable entered in turn.

Note: Use the commands found in this series to dress up your program any way you want. Feel free to experiment and have fun trying new things. A good way to learn new methods of programming is to find a program in which you like the way something is done, and examine the program to see how it is put together.

(See Page 174 for a possible solution to this exercise.)

Hints and Tips

When you program, you will find shortcuts to entering loops and variables. Each character within a program takes up memory in your computer, even the line numbers and spaces in the program. The overhead that the program uses cannot be eliminated. Therefore, you can minimize memory usage

by combining lines. That will result in fewer line numbers and, therefore, a smaller program.

When programming, always number your lines in increments of 10 or 100 so that if you need to edit the program you can do so without changing the program flow drastically. Nothing puts a damper on programming like having to rewrite a program because you numbered the lines 1, 2, 3, 4 and 5 instead of 10, 20, 30, 40 and 50, which would allow plenty of room to make enhancements.

When you want to print one character string or text immediately following another character string or text; you must place a semicolon directly after the string value or text.

This short program will print the text in this manner:

```
THIS IS 1, 2, 3
```

```
10 CLS:PRINT"THIS IS 1,";  
20 PRINT"2,";  
30 PRINT"3."  
40 END
```

 □

Listing 1: GOSUB

```
10 CLS:PRINT:PRINT"THIS IS AN EX  
AMPLE OF THE GOSUB COMMAND. PLE  
ASE NOTE THAT THIS IS LINE 10."  
15 GOSUB1000  
20 CLS:PRINT:PRINT"THIS IS THE S  
ECOND PART OF THE GOSUB COMMAND  
PLEASE NOTE THAT THIS IS NOW L  
INE 20."  
25 GOSUB1000  
30 CLS:PRINT:PRINT"THIS IS THE T  
HIRD PART OF THE GOSUB COMMAND  
PLEASE NOTE THAT THIS IS NOW L  
INE 20. ALSO NOTE THAT THIS IS  
THE LAST PART OF THIS DEMO ALS  
O YOU MAY DO THIS TYPE OF THING  
MANY, MANY TIMES. USING THE SAM  
E GOSUB AREA."  
35 GOSUB1000  
40 CLS:PRINT:PRINT:PRINT:PRINT:P  
RINT" THIS DEMO IS ENDED. REBOO  
TING TO BASIC AT THIS TIME.
```

```
"  
45 FORX=1TO1000STEP1:NEXTX:CLS:E  
ND  
1000 PRINT:PRINT:PRINT:PRINT" P  
RESS ANY KEY TO CONTINUE"  
1010 A$=INKEY$:IFA$=""THEN1010EL  
SE1020  
1020 RETURN
```

Listing 2: COCOCALC

```
0 'THE COCO CALCULATOR HAS BEEN  
WRITTEN TO DEMONSTRATE COMMANDS.  
THIS PROGRAM IS TO BE USED WITH  
THE BASIC PROGRAMMING COURSE  
WRITTEN BY DAVID W. OSTLER, COPY  
RIGHT 1987  
5 CLEAR1000:CLS:PRINT@32,"WELCOM  
E TO THE COCO CALCULATOR":PRINT@  
96," PLEASE ENTER AMOUNTS YOU WA
```

```

NT      THE CALCULATOR TO WORK ON
"
10 PRINT@224,"FIRST AMOUNT";:INP
UTA:SOUND200,1
20 PRINT@224,STRING$(20,32):PRIN
T@224,"NEXT AMOUNT";:INPUTB:SOU
ND200,1
30 GOSUB1000
100 SOUND200,2:C=A+B:GOTO500
110 SOUND200,2:C=A-B:GOTO500
120 SOUND200,2:C=A*B:GOTO500
130 SOUND200,2:C=A/B:GOTO500
140 SOUND200,2:GOTO700
300 PRINT@224,STRING$(20,32):PRI
NT@224,"NEXT AMOUNT";:INPUTB:SO
UND200,1
305 GOSUB2000
310 SOUND200,2:C=C+B:GOTO500
320 SOUND200,2:C=C-B:GOTO500
330 SOUND200,2:C=C*B:GOTO500
340 SOUND200,2:C=C/B:GOTO500
350 SOUND200,2:GOTO700
500 CLS:PRINT@141,"WORKING"
510 FORB=1TO5STEP1
520 PRINT@236,CHR$(162);:PRINT@2
43,CHR$(161)
530 FORX=1TO5STEP1:NEXT
540 PRINT@236,CHR$(168);:PRINT@2
43,CHR$(164)
550 FORX=1TO5STEP1:NEXT
560 PRINT@236,CHR$(164);:PRINT@2
43,CHR$(168)
570 FORX=1TO5STEP1:NEXT
580 PRINT@236,CHR$(161);:PRINT@2
43,CHR$(162)
590 FORX=1TO5STEP1:NEXT
600 SOUND199,1:NEXTB
610 CLS:SOUND20,5:PRINT@64,"TOTA
L EQUALS:";:PRINTUSING"$###,###,
###.##";C

```

```

620 GOTO300
700 CLS:PRINT@224,"FIRST AMOUNT"
;:INPUTA:SOUND200,1:GOTO200
1000 PRINT@297,"DO YOU WANT TO"
1010 PRINT@330,"(A) DD"
1020 PRINT@362,"(S)UBTRACT"
1030 PRINT@394,"(M)ULTIPLY"
1040 PRINT@426,"(D)IVIDE"
1050 PRINT@458,"(E)ND"
1060 G$=INKEY$:IFG$=""THEN1060EL
SEIFG$="A"THEN100ELSEIFG$="S"THE
N110ELSEIFG$="M"THEN120ELSEIFG$=
"D"THEN130ELSEIFG$="E"THEN140ELS
E1060
1070 RETURN
2000 PRINT@297,"DO YOU WANT TO"
2010 PRINT@330,"(A) DD"
2020 PRINT@362,"(S)UBTRACT"
2030 PRINT@394,"(M)ULTIPLY"
2040 PRINT@426,"(D)IVIDE"
2050 PRINT@458,"(E)ND"
2060 G$=INKEY$:IFG$=""THEN2060EL
SEIFG$="A"THEN310ELSEIFG$="S"THE
N320ELSEIFG$="M"THEN330ELSEIFG$=
"D"THEN340ELSEIFG$="E"THEN350ELS
E2060
2070 RETURN

```

Mouse Tales By Logan Ward



"I cannot imagine the CoCo 3 without ADOS-3; it would not be a complete machine."

The RAINBOW, July 1987

You've moved up to a CoCo 3. A powerful new machine. Now, it's time to give BASIC a shot in the arm, with ADOS-3. Wouldn't it be nice to turn on your machine and be greeted by an 80-column display, in the colors of your choice, with your own custom startup message? To run routinely at 2 MHz (double speed) without having to slow down for disk and printer operations? This and much, much more is possible with ADOS-3, our CoCo 3 adaptation of the acclaimed original ADOS, which shares the original's virtual 100% compatibility with commercial software. After customizing ADOS-3 using the provided configuring utility, you can have it burned into an EPROM that plugs into the Disk BASIC ROM socket, or just use it in RAM as a disk utility. (EPROM + burning will cost \$15-20; we provide information concerning how you can have this done.) Supports double-sided drives (35, 40, or 80 tracks). FAST and SLOW commands, auto line number prompts, RUNM command, keystroke macros, arrow-key scroll through BASIC programs, auto-edit of error line, and many more valuable features.

"ON A SCALE OF 1 TO 10, I RATE ADOS-3 A SOLID 15." RAINBOW, 7/87
 Disk . . . \$34.95 Original ADOS for CoCo 1 or 2 . . . \$27.95 (See 6/87 RAINBOW review)
 Original ADOS plus ADOS-3 . . . \$50.00

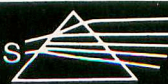
THE PEEPER

ML program tracer that multitasks with the target program. An excellent learning tool for the ML novice; an invaluable debugging aid for the expert. CoCo 1, 2, or 3 compatible.

Disk . . . \$23.95 Assembler source listing . . . Add \$3.00

MONITOR CABLES for CoCo 3

Magnavox 8CM515/8CM505/8CM643 . . . \$19.95 Sony KV1311CR . . . \$29.95

SPECTROSYSTEMS  11111 N. Kendall Drive,
 Suite A108
 Miami, Florida 33176
 (305) 274-3899 Day or Eve

No delay on personal checks • Please add \$2.00 shipping • Sorry no credit cards or COD's