I've been writing about microcomputer systems in my spare time for well over eight years. For the past four years I've been writing mostly about the various systems from Commodore Business Machines while producing the monthly PETpourri column for MICROCOMPUTING magazine.

This book is a collection of some of my major programs, articles and programming ideas that have appeared over the years, plus a few never before seen. A fair amount of the material has been updated or re-written for the VIC-20 and Commodore-64 systems. All of the programs were reviewed with some getting added improvements or new features.

Since this is a collection of many different ideas over a long period of time, there is no specific theme to this book. However, I have tried to organize the material into useful sections that hopefully make it easier to digest. If you should have any comments concerning this book, my monthly column, or any articles; please include a self aaddressed stamped envelope if you expect a reply.

My thanks to the many people at Commodore for their support; plus my wife and son for putting up with all the late hours spent writing over the years.

C H A P T E R 1

COMMON SENSE PROGRAMMING

SAVING SPACE

If you're using a Commodore-64 system, the available RAM space for writing Basic programs is adequate for most programs you might try to write on your own. However, an unexpanded VIC-20 with only 3.5K of available RAM can put a serious limit on what you can do. Fortunately there are certain techniques you can use when writing your programs that might just let you squeak by without adding additional memory. Eventualy you'll probably want to add extra memory but even then these same techniques will let you "do more with less".

The most obvious space saving technique is to avoid using REMarks within your program. This saves a great deal of space but makes your program much harder to document. If you ever want to make changes or enhancements later, it may take quite a while to remember what each variable is used for and how the program works. Just to be safe you might want to jot down notes about the program while you're writing it and save them somewhere safe.

The next best space saving technique is to use multiple statements per line with a separating colon. There's a five-byte overhead associated with every Basic program line regardless of its length. The five bytes consist of two-bytes for the line number encoded in binary, two-bytes for a link

address that points to the next sequential Basic line, and a single byte used as the end-of-line indicator. Whenever you combine two program lines on a single line you save four bytes. You save the five byte overhead but lose another byte for the required separating colon.

Just be careful, it's not always legal or correct to combine program lines. You might change the program flow or create a part of a program line that might never get executed. Be especially careful around IF...THEN... and GOTO statements. Make sure the new program line still does what was intended.

Another way to cut down on memory useage is to delete all unnecessary spaces from within your program. Commodore Basic does not need any separating spaces within program statements. Key words in Basic are stored as coded single byte tokens when a program line is typed in. When the program is interpreted and executed the tokens are easily identified by their unique codes. Separating spaces are simply ignored and just waste space and the time it takes to process them.

As I just mentioned, Basic key words are stored as single byte tokens. All other text in a program line is stored just as it's typed, one character per byte. Thus you can save additional space by using small line numbers and short variable names throughout your program. By using line numbers

1, 2, and 3 instead of 1000, 2000, and 3000 you'd save 3 bytes on every GOTO or GOSUB involving these lines.

Don't forget to use variables to their fullest. Equate variables to the value of commonly used constant values. This especially applies to strings, since the string variable pointer will actually point to the text in the Basic program line where it's defined, instead of using variable space at the top of memory. Also, try to reuse variables wherever possible, rather than defining new variables every time you need to do some calculation. Rememer the less variable space you use the more program space you'll have.

Avoid using arrays when simple variables will work just as well, an array has more overhead associated with keeping track of possibly much more data. If you have to use an array, be sure to declare it's size and don't forget to use the zero element.

Watch the use of integer variables. Simple integer variables still take seven bytes per variable, but integer array elements save memory by using two bytes per array element. If you use simple integer variables like K%, you'll actually waster space by using percent signs each time you reference the variable. However, if this forces using INT(...) functions, you may be better off using the simple integer

variables in certain instances.

Don't forget that Basic sets a default value for each variable the first time encountered in executing a program, strings are set to a null string and numeric variables are set to zero. There's no need to initialize variables to these values since Basic does it for you.

Constant data used by a program can be read into an array from an external data file on tape or disk, rather than from data statements within the program. Alternately, the data can be used directly from the data statements without being placed into an array. By using the Restore command, the data can be reused any number of times.

You can also save space by omitting quotation marks around string element values in data statements. Quotes are only required if there are spaces or special characters like graphics, cursor controls, commas, colons, etc. within the data.

While on the subject of quotation marks, you can also omit closing quotation marks in any Print statement not followed by other items to be printed in the same statement, or not followed by a colon and another Basic statement. Basic will automatically add the closing quote and print the line.

Skip punctuation within multiple-item print statements whenever possible. If you're printing variables X, Y, and Z separated by some text

```
100 PRINT "LENGTH=";X;" WIDTH=";Y;" AREA=";Z
```

you could simply use:

```
100 PRINT "LENGTH="X" WIDTH="Y" AREA="Z
```

There's no need to use the separating semicolons since Basic will automatically assume a semicolon seperator between items.

Use TAB and SPC functions to avoid using extra spaces in Print statements. Just keep in mind that these functions actually move the cursor right the required number of spaces. You cannot use these functions to clear something from the screen since a space character is not printed.

Always use subroutines to perform common functions needed at various points in the program. If you're going to need a yess/no answer from a user at various points in your program, then make a subroutine to do it and call the subroutine whenever needed.

If you have nested FOR...NEXT loops with common exit points then combine the Next statements:

```
100 FOR X=1 TO 10
```

```
110 FOR Y=10 TO 100
120 FOR Z=2 TO 4
 .
 .
 .
200 NEXT Z,Y,X
```

Just be sure you get the variables listed in the correct order. Don't forget the innermost loop will terminate first.

Avoid using parentheses in expressions if not really needed for the proper interpretation of the statement. Remember the hierarchy of operations shown in the manuals.

I hope this information proves helpful in making your program fit in the available space. Generally, most of these space-saving techniques will also save execution time, so you're programs may even run faster as well.