Chapter 14: Example Programs

WHY THIS CHAPTER ?

Although every program in this tutorial has been a complete program, each one has also been a very small program intended to teach you a very specific principle of programming in C. It would do you a disservice to leave you at that point without introducing you to a few larger programs to illustrate how to put together the constructs you have learned to create a major program. This chapter contains four programs of increasing complexity, each designed to take you into a higher plateau of programming, and each designed to be useful to you in some way.

This course was originally written for use on an IBM-PC or compatible running the PC-DOS operating system. Some of the example programs in this chapter will not compile and execute properly if you are using some other computer or operating system, but the techniques may still be useful to you. It would be advantageous to you to spend some time studying these programs regardless of what computer you are using.

DOSEX will illustrate how to make DOS system calls and will teach you, through self-study, how the system responds to the keyboard. WHATNEXT reads commands input on the command line and will aid you in setting up a variable batch file, one that requests an operator input and responds to the input by branching to a different part of the batch file.

LIST is the C source code for the program that you used to print out the C source files when you began studying C with the aid of this tutorial. Finally we come to VC, the Visual Calculator, which you should find to be a useful program even if you don't study its source code. VC uses most of the programming techniques we have studied in this course and a few that we never even mentioned such as separately compiled subroutines.

We will examine the example programs one at a time. However, we will not give a complete explanation of any of them because you have been studying C for some time now and should be able to read and understand the details of these programs on your own.

dosex.c - The DOS Example Program

The copy of DOS that you received with your IBM-PC or compatible has about 80 internal DOS calls that you can use as a programmer to control your peripheral devices and read information or status from them. Some of the earlier IBM DOS manuals, DOS 2.0 and earlier, have these calls listed in the back of the manual along with how to use them. Most of the manuals supplied with compatible computers make no mention of the DOS calls even though they are extremely useful. These calls can be accessed from nearly any programming language but they do require some initial study to learn how to use them. This program is intended to aid you in this study.

Display the program on your monitor or print it out for reference. It is merely a loop watching for a keyboard input or a change in the time. If either happens, it reacts accordingly. In line 32, the function **kbhit()** returns a value of 1 if a key has been hit but not yet read from the input buffer by the program.

Look at the function named **get_time()** for an example of a DOS call. An interrupt 21(hex) is executed after setting the AH register to 2C(hex) = 44(decimal). The time is returned in the CH, CL,

and DH registers. Refer to the DOS call definitions in your copy of DOS. If the definitions are not included there, Peter Norton's book, "Programmers Guide to the IBM PC" is recommended as a good reference manual for these calls and many other programming techniques. If you read your documentation, you will probably find many useful functions available with your compiler that are included as a convenience for you by your compiler writer.

Another useful function is the **pos_cursor()** function that positions the cursor anywhere on the monitor that you desire by using a DOS interrupt. In this case, the interrupt used is 10(hex) which is the general monitor interrupt. This particular service is number 2 of about 10 different monitor services available. This function is included here as another example to you.

The next function, service number 6 of interrupt 10(hex) is the window scroll service as implemented in the function **scroll_window()**. This function should be self explanatory following the previous discussion of this example program.

In the dosex.c program, the cursor is positioned and some data is output to the monitor, then the cursor is hidden by moving it to line 26 which is not displayed. After you compile and run the program, you will notice that the cursor is not visible on the monitor. This can be done in any program, but be sure to return the cursor to any position where it is in view before returning control to the DOS operating system. DOS does not like to have a hidden cursor and may do some strange things if it is hidden.

Some time spent studying this program will be valuable to you as it will reveal how the keyboard data is input to the computer. Especially of importance is how the special keys such as function keys, arrows, etc. are handled. Also note that this program uses full prototype checking and is a good example of how to use it. It is a good example of the modern method of function definitions.

whatnext.c - The Batch File Interrogator

This is an example of how to read the data on the command line following the function call. Notice that there are two variables listed within the parentheses following the **main()** call. The first variable is a count of words in the entire command line including the command itself and the second variable is a pointer to an array of pointers defining the actual words on the command line. The names **argc** and **argv** are used by nearly all C programmers for these two variables.

When this program is executed, the question on the command line, made up of some number of words, is displayed on the monitor and the program waits for the operator to hit a key. If the key hit is one of those in the last word of the command line, the position number of the character within the group is returned to the program where it can be tested with the ERRORLEVEL command in the batch file. You could use this technique to create a variable AUTOEXEC.BAT file or any other batch file can use this for a many way branch. Compile and run this file with TEST.BAT for an example of how it works in practice. You may find this technique useful in one of your batch files and you will almost certainly need to read in the command line parameters someday.

An interesting alternative would be for you to write a program named **would.c** that would return a 1 if a Y or y were typed and a zero if any other key were hit. Then your batch file could have a line such as;

```
Would you like to use the alternative method ? (Y/N)
```

DOS will use **Would** as the program name and ignore the rest of the statement except for displaying it on the screen. The user can then respond to the question on the monitor with a single keyhit. Your batch file would then respond to the 1 or 0 returned and either run the alternative part of the batch file or the primary part whatever each part was.

```
Would you like to use primary ? (Y/N)
IF ERRORLEVEL 1 GOTO PRIMARY
(secondary commands)
GOTO DONE
:PRIMARY
(primary commands)
:DONE
```

list.c - The Program Lister

This program is actually composed of two files, list.c and listf.c that must be separately compiled and linked together with your linker. There is nothing new here and you should have no trouble compiling and linking this program by reading the documentation supplied with your C compiler.

Notice the simplicity of the program named list.c. It can be read and understood completely in a very short period of time. Unfortunately, it doesn't do very much, because it passes the work off to the functions. Since it breaks the overall problem into many smaller problems, it makes it much simpler to understand exactly what is happening. You should have no problem understanding this file. If you wish to understand exactly what each of the functions do, you can review them in the listf.c file, and its accompanying listf.h header file.

Pay particular attention to the organization of the listf.h file and the way it is associated with the listf.c file. Any program that wishes to use the services provided by the listf.c file can #include the listf.h header file since it has prototypes for all of the functions, and all #defines that may be needed for use with the file. It would be a good plan for you to begin dividing your source into two files in this manner for two reasons. First, it is good programming practice, and secondly, it tends to look like the style used in object oriented programming. Because object oriented programming does so much for software development, it must be considered as a viable option for the future, and you should begin leaning in its direction.

It would be to your advantage to compile, link, and run this program to prepare you for the next program which is composed of 6 separate files which must all work together.

vc.c - The Visual Calculator

This program finally ties nearly everything together because it uses nearly every concept covered in the entire Coronado Enterprises C tutorial. It is so big that I will not even try to cover the finer points of its operation. Only a few of the more important points will be discussed.

Example program ------> **VC.DOC**

The first thing you should do is go through the tutorial for VC included in the file VC.DOC. There are several dozen steps for you to execute, with each step illustrating some aspect of the Visual Calculator. You will get a good feel for what it is capable of doing and make your study of the source code very profitable. In addition, you will probably find many ways to use the Visual Calculator to solve problems involving calculations where the simplicity of the problem at hand does not warrant writing a program.

Notice that the structure definitions, used in the various parts of the program, are defined in the file vc.h. During program development, if it became necessary to change one of the structures slightly, it was not necessary to change it in several different files. Only the file vc.h required modification, and since it was then included in the other source files they were all automatically updated. Notice that the transcript data is stored in a doubly linked list with the data itself being stored in a separate dynamically allocated character string. This line is pointed to by the pointer **lineloc**.

For ease of development, the similar functions were grouped together and compiled separately. Thus, all of the functions involving the monitor were included in the file named video.c, and all of the functions involving the data storage were grouped into the file.c collection. Dividing your program in a way similar to this should simplify debugging and future modifications.

Two styles of coding are presented in this program. video.c contains 18 functions which may be called by other parts of the program, and all prototypes are stored in video.h in the manner defined for the last example program, list.c. The other files have their prototypes all gathered together in vc.h so that including this file brings all prototypes into the including program. The first method is the preferred method, but either gets the job done.

Of special interest is the function named **monitor()**. This function examines the video mode through use of a DOS command and if it is a 7, it assumes it is a monochrome monitor, otherwise it assumes a color monitor. The colors of the various fields are established at this time and used throughout the program. Most of the data is written directly to the video memory, but some is written through the standard BIOS routines.

WHERE DO I GO FROM HERE ?

Although the C programming language is actually a very small language, considering the number of constructs that are available, it is extremely powerful. Having completed this course, you are ready to begin a very interesting journey in programming, and the best way to get started is to actually use the language. Make up a program that you would find useful and begin to develop it immediately. Because C is so new to you, you will find roadblocks as you progress, but with the plethora of books available on nearly any part of software development, you will find help along the way and will achieve great satisfaction as you complete each step.

Because the software development world is leaning heavily toward C++ and object oriented programming, I highly recommend that you begin a study of these topics, but not until you gain a little experience with C. C++ is actually a new way to package C code in such a way that errors in the code are reduced, and the program is much more understandable and maintainable. C++ uses all of the operations available in C and adds many new and extremely useful operations. C++ is also a much more complex language than C, but the benefits far outweigh the effort required to learn it.

Because many programmers have no idea of where to begin using a new programming language, a few recommendations are in order for first projects. These projects are selected in such a way that they can be useful but not too ambitious for beginning. They are not trivial and there are no solutions to them in the ANSWERS directory.

1. Write a program to store up to 100 names, addresses, and phone numbers of your friends and relatives. This list can be stored to a file, read from the file, or printed for a hardcopy listing in a format that could be easily folded and carried in a ladies purse.
2. Improve the above program so that it will store any number of names by storing the data in a linked list. The list should be sortable.
3. Write a program to play TIC-TAC-TOE interactively.
4. Write a program that reads in a C source file and counts the lines that have comments only, those that are blank, and the total number of lines in the file. The results can be listed on the monitor. The most interesting result will be the number of lines of non-comment source code (NCSC) which is calculated easily from the above information.
5. Improve the above program to read in multiple source files and combine the results to achieve the NCSC lines for an entire project.

By this time you should be developing some ideas for another project. The best advice I can give you toward gaining experience with C is to simply program in C.

Index Previous . . . . . *C Tutorial*

The Webwizard