

MICROSOFT VISUAL C++ 6.0 IDE TUTORIAL **Creating Win32 Console-Mode Applications**

Jeffrey S. Franzone, Assistant Professor
Engineering Technology Department
University of Memphis

Abstract

Microsoft Visual C++ is a commonly used programming language and application environment in many computer science and computer engineering technology programs. Visual C++ can be used to teach both C and C++ and it boasts a highly powerful, but easy to use, development environment. One of the strengths of the Visual C++ product is the different types of applications that can be generated. Visual C++ supports Windows applications, DLLs, and console-mode applications, to name just a few.

Visual C++ 6.0 is currently used in the C/C++ programming curriculum in the Computer Engineering Technology Department at the University of Memphis. The department has chosen this product to teach C and C++ programming for three major reasons: 1) it is a professional and modern C/C++ programming application platform that is an industry standard, 2) it is relatively easy to use and its rich set of features and tools can be taught in parallel with C/C++ programming concepts and techniques, and 3) it supports console-mode programming.

Visual C++'s console-mode support makes it an ideal application platform for teaching C/C++ programming fundamentals in lower division programming courses. It is believed by many (and supported by many C/C++ programming textbooks) that console-mode programming is an easier environment to teach, analyze, and learn fundamental C/C++ programming concepts and standard programming techniques. Many of these concepts and techniques, when taught in a console-mode environment, are also more portable to other computer platforms.

The purpose of the *Microsoft Visual C++ 6.0 IDE Tutorial* is to support computer programming instructors who want to use the Visual C++ product to develop Win32 console-mode applications in introductory C/C++ programming courses without spending several weeks explaining the Visual C++ environment. The tutorial takes a visual, step-by-step approach in demonstrating how to create a Win32, console-mode application as well as covering introductory programming concepts such as the “*edit, compile, link, and run*” process and useful procedures such as printing source-code and program output. The tutorial is thorough and complete enough to be given as a lab or homework exercise or as a class exercise done in a laboratory setting.

I. Introduction

The Microsoft Visual C++ IDE Tutorial is organized under the following sections with a brief description of the concepts covered under each section. The complete tutorial is provided at the end of this paper.

1. OBJECTIVES

A very brief general summary of the purpose of the tutorial.

2. RESOURCES YOU WILL NEED

An outline of reader skills, materials, and software needed to successfully complete the tutorial.

3. INTRODUCTION TO C/C++ DEVELOPMENT

This section briefly discusses the edit, compile, link, and run development process for C/C++ programs, the types of files generated by each step, and the meaning of “console-mode”.

4. CREATING A PROJECT & WORKSPACE

This section uses visual screenshots to single-step the reader through the process of starting Visual C++ 6.0 and creating an empty Win32, console-mode application, an overview of the more important IDE commands and windows encountered in the development environment, the meaning of the terms “Project” and “Workspace”, and some of the important files generated by the creation of an empty Win32, console-mode application.

5. CREATING A SOURCE FILE

This section uses visual screenshots to show the reader the steps necessary to create a blank C/C++ source file and include it in the current project.

6. USING THE IDE EDITOR

This section instructs the reader to enter an example C program into the Document Window. Issues concerning saving files, opening and saving project files from a floppy-disk versus a hard disk, and generating a default workspace are also discussed.

7. COMPILING A PROGRAM

This section instructs the reader on the steps needed to compile a C/C++ source file and the purpose of the Output Window. To demonstrate how the compiler detects syntax errors, an error is purposely entered into the example program to generate a compilation error.

8. LINKING A PROGRAM

This section instructs the reader on the steps needed to link a C/C++ object file.

9. RUNNING A PROGRAM

This section instructs the reader on the steps needed to run a C/C++ executable file from within the Visual C++ development environment. Visual screenshots are used to demonstrate the running of the example program. Also, instructions are given on the steps needed to run applications from a windowed-screen to a full-screen and vice-versa.

10. PRINTING SOURCE CODE & PROGRAM OUTPUT

This section discusses the steps needed to print source code from within the Visual C++ development environment and how to use text or graphics screenshots to capture and print program output.

11. OTHER ITEMS OF INTEREST

This section consists of several subsections that describe useful shortcuts in the development process and common questions that arise when developing Win32, console-mode applications for the first time. The subsections include:

- a). Reopening a Project.
- b). Where do I find my program's executable file?
- c). Do I need Visual C++ to run my applications?
- d). Using the Build and Execute commands.
- e). IDE color schemes.

12. WHAT YOU HAVE LEARNED

This section provides a summary of the major concepts covered in the tutorial.

13. SOME CLOSING THOUGHTS

This section provides the reader with some helpful "wisdom" about learning the Visual C++ environment before undertaking serious study of C/C++ programming.

The following list is a summary of the concepts and techniques that are learned after completing the *Microsoft Visual C++ 6.0 IDE Tutorial*.

1. The four major steps in writing a C/C++ program.
2. The meanings of the terms "*Edit*", "*Compile*", "*Link*", and "*Run*".
3. The meanings of the terms "*source code*", "*object file*", and "*executable file*" and the file extensions associated with them.
4. The meanings of the terms "*console-mode*" and "*Win32 console-mode*" programs.
5. The four major IDE screen areas.
6. The steps required to create an empty *Project* for a Win32 Console Application.
7. The meanings of the terms "*Project*" and "*Workspace*".
8. The major files that are internally generated by Visual C++ when an empty Win32 Console Application Project is created.
9. The purpose, significance, and file extensions associated with the "*Project file*", "*Workspace file*", and "*Workspace Options file*".

10. The name and location of the folder where a program's executable and object files are stored.
11. How to create a new C/C++ source file and how to determine the appropriate file extension.
12. How to list project files in the *Project Window* and how to view the contents of project files in the *Document Window*.
13. How to manually enter a C/C++ source file into a blank document using the built-in editor.
14. The maximum line length that can be entered into a source document without text-clipping or word-wrapping when printed from the IDE.
15. How and when to save your source and *Workspace* files using the IDE.
16. The advantages of storing *Project* files on the local hard drive during development and when it is appropriate to store *Project* files on a floppy disk.
17. How to open a previously created *Workspace* from a floppy disk or local hard drive assuming all appropriate files exist and how to create a default *Workspace* when only the source file exists.
18. How to "*compile*" a C/C++ source file using the "*Compile...*" command.
19. The purpose of the *Output Window* and how it helps you detect syntax errors in your source code.
20. How to create an executable file by "*linking*" object code using the "*Build...*" command.
21. How to run your programs from within the IDE by using the "*Execute...*" command.
22. When the "*Press any key to continue*" message appears.
23. How to switch a running program from a "*windowed*" session to a "*full-screen*" session.
24. How to print a source file from within the IDE.
25. How to capture and print program output using a "*text screenshot*" or "*graphics screenshot*".
26. How to open a previously created Project without first starting the IDE and executing menu commands.
27. The meaning and significance of an executable file (standalone application).
28. How to compile and link a program with a single command (*Build...*) and how to compile, link, and run a program with a single command (*Execute...*).
29. The three shortcut command icons that are used to compile, link, and execute programs from within the IDE.
30. How and where to change the color scheme of the IDE environment and the effect it has on other Windows applications.

Many beginning C/C++ programming students who have used the tutorial in the Engineering Technology Department at the University of Memphis have provided valuable feedback to the effectiveness of the tutorial (and many have suggested additional screenshots and explanations that have been incorporated into the latest version). Many have commented positively on the tutorial's "attention-to-detail", leading the reader through a thorough, step-by-step process. The author has also observed that during the course of a semester, students have fewer questions concerning the use of the Visual C++ IDE environment. Based on student feedback, it is the

author's opinion that after students complete the tutorial, they are less frustrated with the programming process because they find themselves struggling less with the programming tools.

II. Conclusion

Microsoft's Visual C++ 6.0 Integrated Development Environment provides a rich-set of features for C/C++ program development. It is especially useful for developing Win32, console-mode applications that are reminiscent of the old DOS-style programs. Console-mode is generally the preferred methodology used to teach lower-division C and C++ programming courses because many console-mode techniques are portable to other platforms and intimate knowledge of the host operating system is not required.

The Microsoft Visual C++ 6.0 IDE Tutorial is designed to teach beginning C/C++ programmers how to create Win32, console-mode applications using the Visual C++ development environment. It can be particularly useful to instructors teaching a beginning C/C++ course because the tutorial single-steps the reader through the entire development process using numerous visual screenshots. Instead of spending valuable weeks covering the Visual C++ environment in a traditional lab setting, instructors can assign this tutorial as a first laboratory assignment or can cover it during a lecture period. Students will also find this tutorial as a good reference source.

Bibliography

None

JEFFREY FRANZONE

Jeffrey Franzone currently teaches in the Engineering Technology Department at the University of Memphis as an Assistant Professor. He teaches C, C++, Java, and microprocessor courses. He has 7 years industrial experience working as an engineering technologist both in hardware and software design and testing and 8 years teaching in Engineering Technology. Jeff received a Bachelor's degree in Electronics Engineering Technology at California State University at Long Beach in 1991 and received a Master's degree in Computer Technology at Arizona State University in 1996.

MICROSOFT VISUAL C++ 6.0 IDE TUTORIAL

Creating Win32 Console-Mode Applications

Written by Jeffrey Franzone
July 6, 2001

OBJECTIVES:

This tutorial will acquaint you with the Microsoft Visual C++ 6.0 *Integrated Development Environment* (IDE). You will enter a sample program into the editor and learn the commands needed to successfully *edit, compile, link, and run* a C/C++ “Win32 console-mode” program under the *Windows* environment.

RESOURCES YOU WILL NEED:

- 1). A 3½” floppy disk to save your source code and executable file (optional).
- 2). Access to Microsoft Visual C++ 6.0. Although the sample program given in this tutorial will run under any C/C++ compiler, this tutorial is specifically targeted for Microsoft Visual C++ 6.0. The author has chosen to write the sample program in C so that this tutorial can be used in any introductory C and C++ course. However, many of the concepts presented in this tutorial apply to both C and C++ projects and files. Any differences between the two languages are noted when applicable.
- 3). You do not necessarily need to understand any C code presented here because this tutorial does not require knowledge of how the sample program operates. It is merely intended to show you how to use the basic features of Microsoft Visual C++ 6.0’s IDE to write Win32 console-mode programs. However, if you do have some knowledge of C (even if you are taking a C++ course), the sample program may provide you a useful review of some C concepts.
- 4). Basic knowledge of computer usage and *Windows 95/98*. This tutorial assumes that you know how to operate a computer and are familiar with *Windows 95/98*. (**IMPORTANT**: STOP if you do not know how to operate a computer, how to create folders, how to navigate between folders and the computer’s directory structures, how to save files to the hard drive and floppy disk, etc. Consider taking an introductory computer literacy course before attempting programming.)
- 5). Printer access to print your source code and output results, if applicable.
- 6). **ANOTHER REMINDER**: This tutorial is only concerned with teaching you essential information about Microsoft Visual C++ 6.0’s IDE and the basic steps needed to develop a C/C++, Win32 console-mode program in this environment. It does not teach you how to program!

INTRODUCTION TO C/C++ DEVELOPMENT

To write a C/C++ program, several steps must be undertaken. There are four basic steps in this development process:

- 1). EDIT
- 2). COMPILE
- 3). LINK
- 4). RUN

The *EDIT* process refers to the actual writing of the C/C++ program into the editor. This implies that you have already formulated a software solution to a problem with outline pseudocode, a flowchart, or a combination of both. The program you type into the editor is called the “*source code*” which means that the program is written in a grammatical “*sentence-like*” and “*algebraic-like*” style that is easily read and understandable to humans. The file that contains C source code ends with a “.c” extension while files that contain C++ code end with a “.cpp” extension. It is important to realize that if you are writing programs that include any C++ code, you must save the file with a “.cpp” extension. In Microsoft Visual C++ 6.0, the “.cpp” extension automatically invokes the C++ compiler while the “.c” extension automatically invokes the C compiler. C++ code cannot be understood by the C compiler but the C++ compiler does understand C code.

The *COMPILE* process refers to the translation of “*source code*” to “*object code*”. As the C/C++ compiler begins its translation of each instruction, it checks every C/C++ statement for proper syntax. If a statement is not properly constructed, the compiler cannot translate it into object code. The compiler will then signal an error and you will have to correct the error before you can make a successful compile. When the compile process is successful, an *object file* (“.obj” extension) will be created.


If the compile process is successful, the *LINK* process is invoked. The LINK process takes the object file (or files) of your program and “joins” (or links) it together with system and library object files. Library files are usually provided by the compiler manufacturer, third-party vendors, or other programmers and provide added functionality to the C/C++ language. System object files are special instructions that are needed in your program so that the operating system knows how to load and run your program. If your program cannot be successfully linked, an error message will appear and you will have to resolve the error before continuing. The linker will create an *executable file* (“.exe” extension) if this process is successful.

Finally, if the link process is successful, you are ready to actually run the program and see if the program provides an adequate solution to the original problem. The programs we will be writing and running are called “*Win32 console-mode*” applications and are reminiscent of the old DOS-style, and still popular UNIX-style, programs that are run under a command-line prompt. *Console-mode is a term that refers to programs that do not incorporate the advanced windowing and graphics capabilities that are commonplace with applications developed under a graphical operating system such as Windows.* Consequently, console-mode applications have limited graphics and user-interface capabilities as compared to many Windows-type programs. However, console-mode programs do have

the advantages of easier application development (you don't have to know intimate details of the operating system) and faster program execution than equivalent Windows-type programs. A "Win32 console-mode" program does provide some basic windowing functionality and direct access to a limited set of Windows operating system features through calls to the Windows API (*Application Programming Interface*). However, this type of console-mode program can only be run on a Win32-compatible operating system, severely restricting its portability.

CREATING A PROJECT & WORKSPACE

STEP #1:

Start Microsoft Visual C++ 6.0 by mouse-clicking on the Visual C++ 6.0 startup icon  located on the *Windows 95/98* desktop or from the *Start / Programs* menu. When the program is started, you should see a screen similar to the one shown in Figure 1.

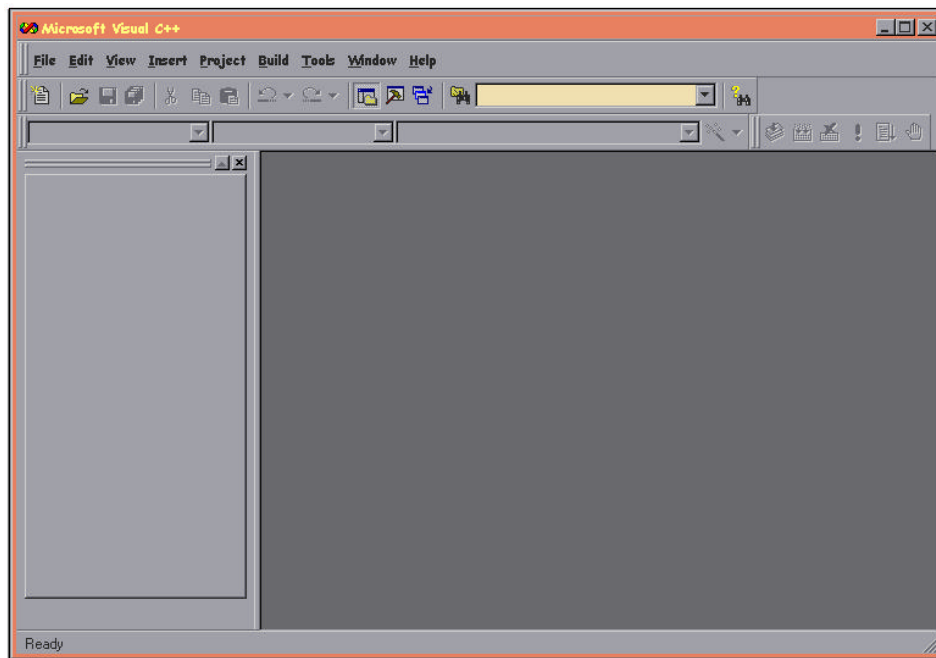
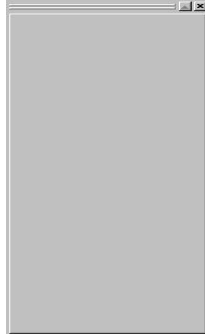


FIGURE 1. The Microsoft Visual C++ 6.0 *Integrated Development Environment*.

Note four items of immediate interest in Figure 1:

- a). The main *Menu Bar*  located just below the title bar. The *Menu Bar* contains many of the high-level commands that are typically used during the development of C/C++ programs. In this tutorial, you will not learn all of the commands available but you will learn some of the more important ones.

- b). The window to the left (and shown below as initially empty) is referred to as the *Project Workspace*. One of its main functions is to provide important information about the files contained in a programming project.



- c). The window to the right (and shown below as initially empty) is referred to as the *Document Window* or *Editor Window*. This window is where you will load, read, write, and/or modify C/C++ source code.



- d). *Shortcut Command Bars* are located below the *Menu bar* but above the *Project Workspace* and *Document Window* windows.



Each shortcut mimics a command from the *Menu Bar* and is represented by a unique icon. By placing the mouse cursor over a shortcut icon, Windows will display the name of the command at the bottom left of the icon (called a “balloon” message). There are several different types of shortcut bars available and they are “dockable”, meaning that they can be placed almost anywhere in the IDE environment. You can select and customize these bars by selecting the “Customize...” command from the *Tools* menu. Normally, the default *Shortcut Command Bars* are sufficient for most development projects.

Other important IDE bars are the *Title Bar* and the *Status Bar*.

- The *Title Bar* is located above the main *Menu Bar* and displays the name of the currently open project and any file currently open in the *Document Window*.



- The *Status Bar* (located at the bottom of the IDE window) displays brief information about a selected command, the current operating state of the IDE, and text cursor information for a file loaded into the *Document Window*.



STEP #2:

From the *Menu Bar*, select *File*, then “*New...*”. You should see a screen similar to Figure 2. (The “*Project name:*” edit box will be initially empty.)

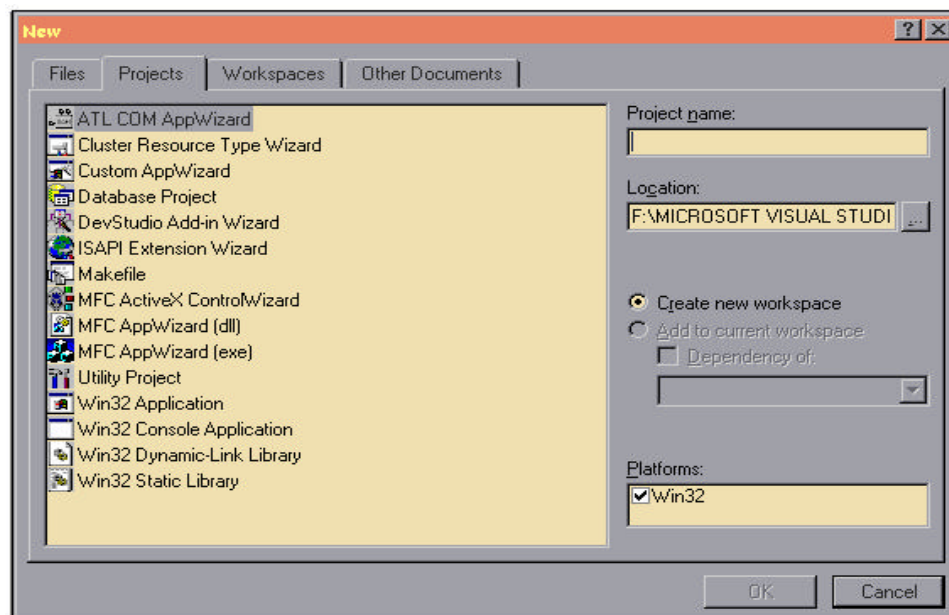


FIGURE 2. The *New Project* dialog box.

Under “*Project name:*” type a name that you want to call this project (for example, “*tutorial*”). Under “*Location:*” you can select the path (drive and directory structure) where you want to store the project if the default location is not desirable. Next, highlight the selection “*Win32 Console Application*” that appears in the list box on the left side of the window. Press *OK* and the window in Figure 3 should appear immediately.

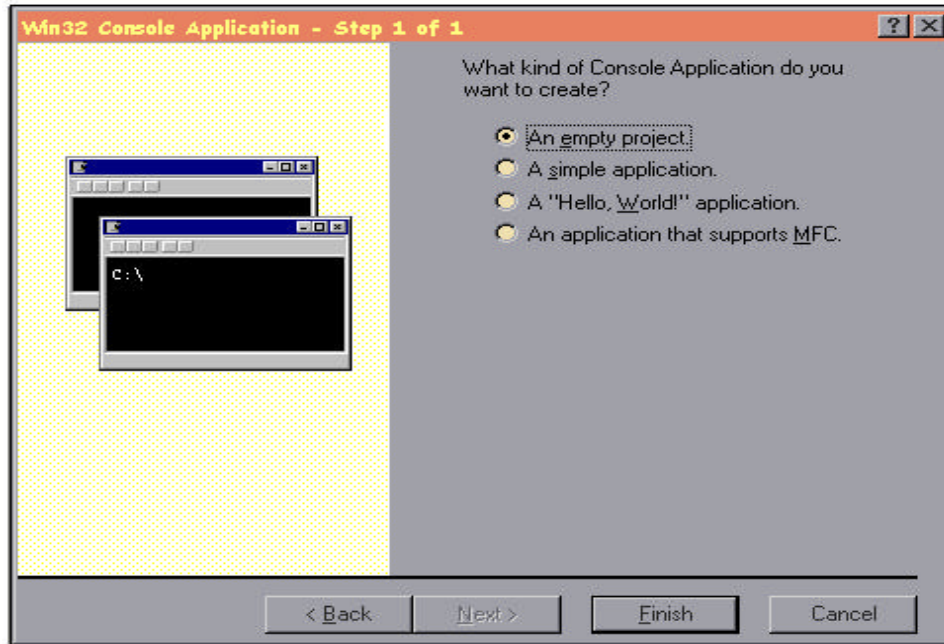


FIGURE 3. The *Win32 Console Application* Dialog box.

STEP #3:

You will see several choices that determine the type of Console Application you want to create. Keep the default choice “*An empty project*” selected, then press *Finish*, then *OK*. You should now see a screen similar to Figure 4.

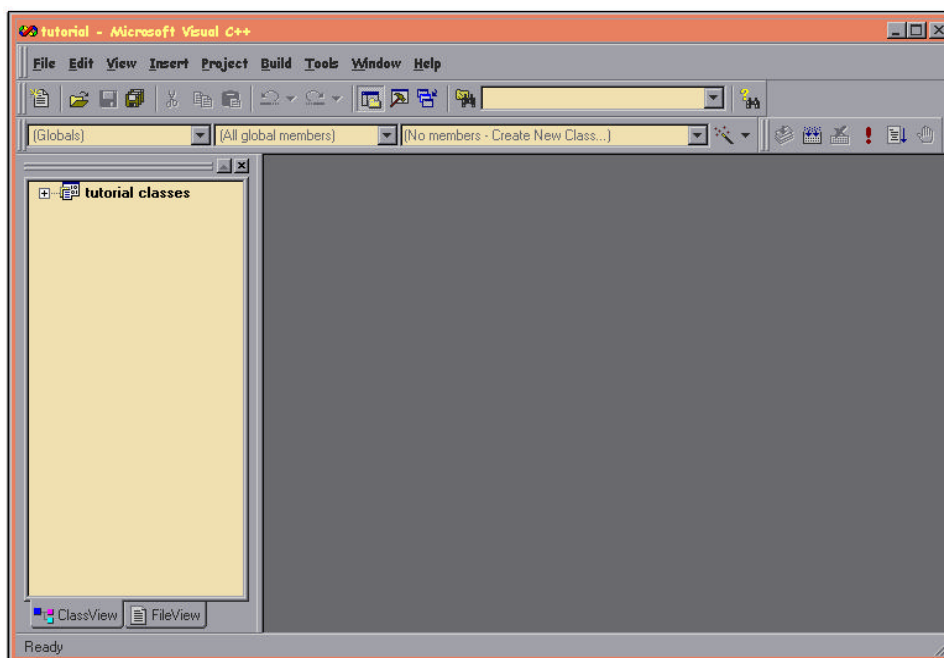


FIGURE 4. The default IDE *Workspace* after a *Project* has been created.

As seen in Figure 4, every *Project* has an associated *Workspace*. A Project's workspace includes, but is not limited to, the *Menu Bar*, *Shortcut Command Bars*, *Project Window*, *Document Window*, the size, visibility, and the placement of windows, project settings, and other tools and window sizes.

After the first three steps, Visual C++ automatically creates several important files located in your project folder ("*tutorial*" in this example). For now, you should be aware of three generated files: 1) a *Project* file (ProjectName.dsp) which stores information about how to build your program (application type, applicable program source files, header files, library files, etc., compiler and linker settings, and configuration settings for Debug and Release versions), 2) a *Workspace* file (ProjectName.dsw) which organizes all projects within a single environment (a *Workspace* file can include multiple projects) and loads the *Project* file and *Workspace Options* file associated with the current project, and 3) a *Workspace Options* file (ProjectName.opt) which stores environment settings such as visibility, size and placement of Workspace windows and many other default Workspace and Project settings. This allows you to arrange the IDE environment the way you want for each project you create. The next time you start your project, it will look exactly the same as you left it last. Or, the next time you create a new project, it will contain the project settings (such as compiler and linker settings) from the previous project. In this tutorial, the generated *Project* file is "*tutorial.dsp*", the *Workspace* file is "*tutorial.dsw*", and the *Workspace Options* file is "*tutorial.opt*" all stored under the "*tutorial*" Project folder.

Visual C++ also creates an empty folder called "*Debug*" under your *Project* folder. Later, you will see that this is the location that Visual C++ stores your program's executable and object files.

In the next several steps, you will compile, link, and run a C program from an example C source file you will enter into the editor. (**NOTE:** a simple C++ program can be easily substituted for the example C program.)

CREATING A SOURCE FILE

STEP #4:

To create a new C/C++ source file, select "*New...*" from the *File* menu. The *New Files* dialog box will appear as shown in Figure 5. Note that the project name you entered earlier will be automatically listed under the "*Add to project:*" edit box. Next, type a filename for the C or C++ program in the "*Filename:*" edit box using a filename extension of ".c" if you are writing only C code or ".cpp" if you writing C++ code or a combination of C/C++ code (for example, in this project, the C program file is called "*tutorialsrc.c*"). Then, highlight the selection labeled "*C++ Source File*" and press OK. You should see a window similar to the one shown in Figure 6.

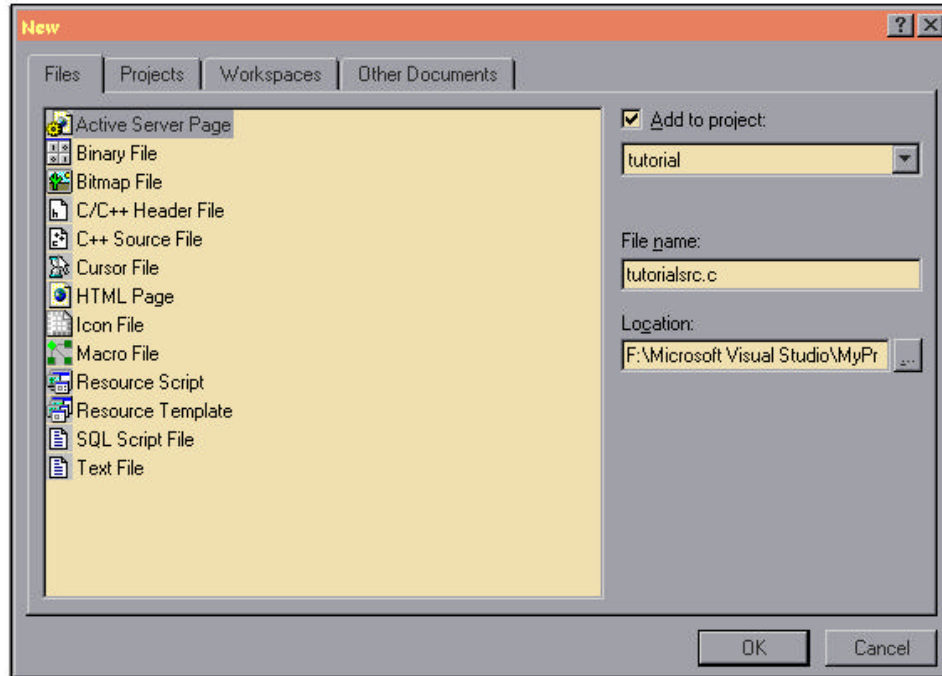


FIGURE 5. The *New Files* Dialog box.

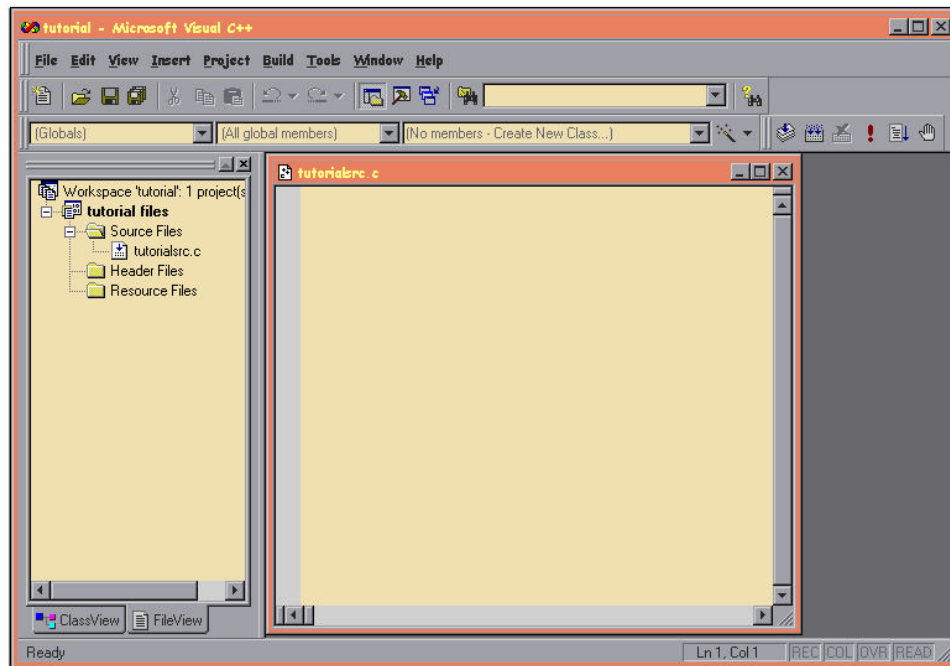



FIGURE 6. A blank document in the *Document Window*.

Referring to Figure 6, the *Document Window* will now contain an empty document with the name of the file you entered earlier (“*tutorialsrc.c*”) displayed in the title bar. The *Project Window* on the left shows that the C source file is stored under the “*Source Files*” folder under the “*tutorial*” project folder. To see this screen, first select the “*FileView*” tab located at the bottom of the Project Window. Then, click the small + sign located to the left

of the “*tutorial files*” icon. You will now see three folder icons named “*Source Files*”, “*Header Files*”, and “*Resource Files*”. Clicking on the small + sign next to the “*Source Files*” folder icon will open up a list of all the C or C++ source files that are contained in your project (there is only one C source file in this project). Any file contained in the *Project Window* can be viewed in the *Document Window* simply by double-clicking on the file. You can also expand the document to fill the entire *Document Window* by clicking on the small box icon  located near the far right of the document’s title bar. (**NOTE:** By the way, the information shown under “*tutorial files*” is stored in “*tutorial.dsp*”.)

USING THE IDE EDITOR

STEP #5:

Type the sample program (Listing 1) exactly as given below into the blank document. If you are new to programming, do not be concerned at this time if you do not understand why or how this program works. This program is only designed to get you comfortable with typing a real C program into the editor. Figure 7 shows a partial view of the sample program entered into the blank document. (**NOTE:** Do not type past column 95 in the document window or the printed output will not be aligned properly.)

```
/* Anything contained between these two symbol pairs */
/* are called "comments". Comments are ignored by the compiler */
/* but are very important in documenting a program so that
   humans can understand important information about the program.
*/
```

```
/* This is called the Programmer's Block */
```

```
/* Tutorialsrc.c
   Written by Jeffrey Franzone
   August 25, 1999
   Course number: CETH 2251
```

```

   The purpose of this program is to provide a sample C program
   that incorporates the basic elements of a more practical
   C program. This program takes input from the user, makes a
   decision based on a user's response, and then outputs a message.
*/
```

```
// The symbols // is another way to add comments. Everything after the // is a comment
// until the end of the line.
```

```
// Header files
#include<stdio.h>
```

```

/* The main() function */
main()
{
    char answer;

    /* This section of code is the INPUT module. */
    printf("\n\nDo you FANCY me, BABY!\n");
    printf("Type 'y' or 'n', BABY! --> ");
    scanf("%c", &answer);

    /* This section of code is the PROCESS ALGORITHM module and
       the OUTPUT module. The PROCESS ALGORITHM module makes a
       decision on what output should be displayed to the screen.
       The OUTPUT module actually displays a message to the screen.
    */
    if(answer == 'y')
        printf("\n\nOH YEAH, BABY! YEAH, YEAH, YEAH, BABY!!!\n\n");

    if(answer == 'n')
        printf("\n\nPerhaps you lack the proper motivation, BABY!\n\n");

    if(answer != 'y' && answer != 'n')
        printf("\n\nOH BEHAVE, BABY! You did not type the right letter.\n\n");
}

```

LISTING 1. A sample C program.

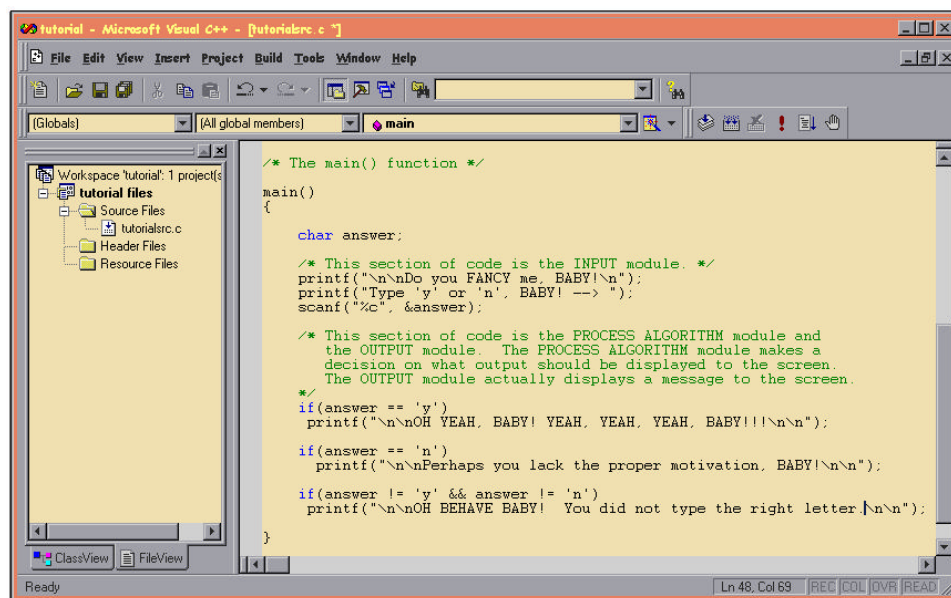





FIGURE 7. A partial view of the sample C program entered into the editor.


It is a good habit to always save your work at short, regular intervals. To save your source code, execute the “*Save*” command by selecting it either from the *File* menu, pressing the Ctrl + S keys, or selecting its shortcut icon . If you want to save your window and project settings and the files that are currently open, execute the “*Save All*” command from the *File* menu or its shortcut icon . (**NOTE:** The “*Save Workspace*” command only saves your window and project settings.)

It is recommended that you save all of your program files on the local hard drive rather than a floppy disk. The development tools run much slower when they have to read, write, and translate files that are stored on a floppy. Also, for larger projects, all of the files that Microsoft Visual C++ generates will not fit on a single floppy disk. This will usually prevent Visual C++ from creating an executable file. After you complete your program, copy your source and executable files to your floppy disk for safekeeping. Copying project and workspace files (and other internally generated Visual C++ files) are optional since these files can be easily generated automatically by Visual C++ when you recompile or rebuild your project. However, if you have room on your floppy, it is recommended for convenience that you save your *.dsp*, *.dsw*, and *.opt* files in addition to your source and executable files.

You will find that the compilation, linking, and running of your programs will be much faster from the local hard drive than from a floppy. If you have previously saved project files, workspace files, and source files on a floppy, first make a temporary project folder on the local hard drive. Then, copy the necessary files from your floppy to the folder on the local hard drive. Next, start Visual C++. If you have all of your project and workspace files, open the *Workspace* file (ProjectName.dsw) by selecting the command “*Open Workspace...*” from the *File* menu. All of your previous program project and workspace settings should load and remain intact. If you only have your source file, open this file by executing the “*Open...*” command by selecting it either from the *File* menu, pressing the Ctrl + O keys, or selecting its shortcut icon . When you go to compile or build your program, Visual C++ will prompt you to build a default workspace. Select *Yes* and default *.dsp* and *.dsw* files will be created for your program.

COMPILING A PROGRAM

STEP #6:

To compile the source code into object code, execute the command “*Compile tutorialsrc.c*” either from the *Build* menu, pressing the Ctrl + F7 keys, or selecting its shortcut icon . Another window will appear at the bottom of the IDE as shown in Figure 8. This is the *Output Window* and Visual C++ will display status information concerning the compile and build process. The *Output Window* contains several tabs that determine the type of information displayed in the *Output Window*. The window can display debugging information (the *Debug* tab), build information (the *Build* tab, selected by default), and the results for string searches in source-files (*Find in Files* tabs).

If no syntax errors occur, you will see a message in the *Output Window* stating that no errors were found and that the compilation process was successful. If errors exist in the source code, the *Output Window* will list the line number in the source code where the error occurred as well as a brief description of the error. As shown in Figure 9, an error was purposely inserted into the sample program for illustration purposes. If you double-click on the error information line in the *Output Window*, Visual C++ will visually indicate with an arrow where in the document it thinks the error occurred. You must always eliminate syntax errors from your source code before the compilation process can be successfully completed.

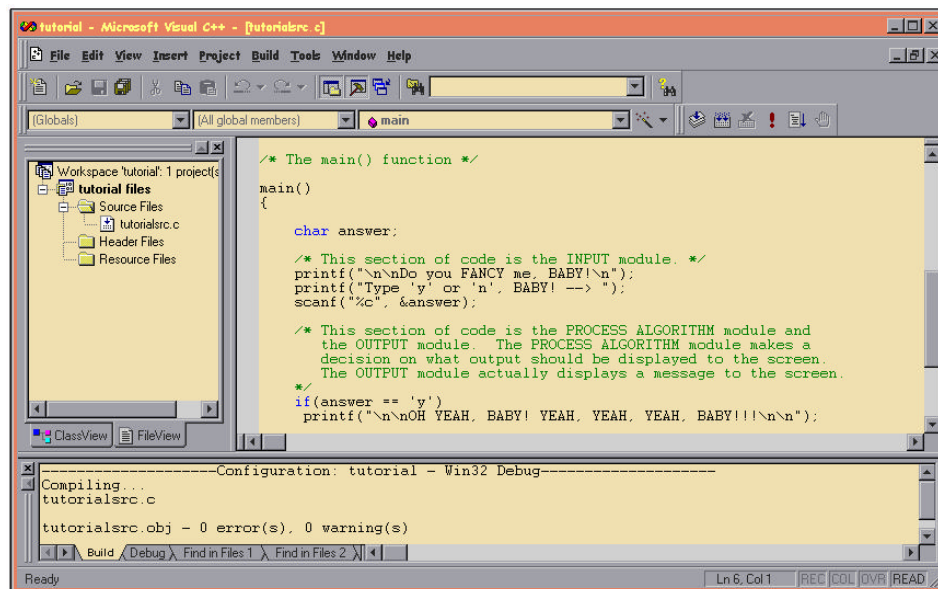


FIGURE 8. The *Output Window* shows the results of the compile process.

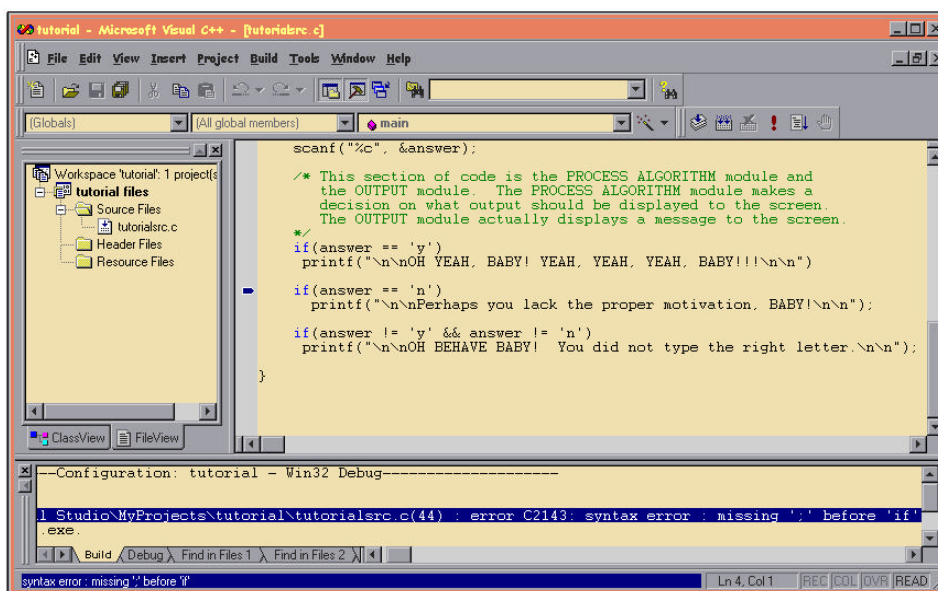



FIGURE 9. If a syntax error occurs, you can double-click the error line in the *Output Window*. Visual C++ will place an arrow in your document showing you where it thinks the error occurred.

LINKING A PROGRAM

STEP #7:

When the compilation is successful, you are then ready to “*link*” your program. When you link a C/C++ program, Visual C++ will create a file that is executable only under those versions of the *Windows* operating system that are Win32 compatible (Win95/98/NT) or *Windows’ DOS Emulation mode* compatible. By default, Visual C++ automatically chooses the name of the project as the filename and adds a “.exe” extension. For example, in this tutorial, the Project name is “*tutorial*”. Consequently, Visual C++ will create an executable file called “*tutorial.exe*” when you invoke the Linker. (**NOTE:** *The executable file is saved in the Debug folder under the program’s Project folder.*)

To link your program, execute the command “*Build tutorial.exe*” either from the *Build* menu, pressing the F7 key, or selecting its shortcut icon . As shown in Figure 10, status information will be presented in the *Output Window*. The same error procedures apply here as they do for the compilation process. If the link process is successful, you will see a message indicating zero errors in the *Output Window*.

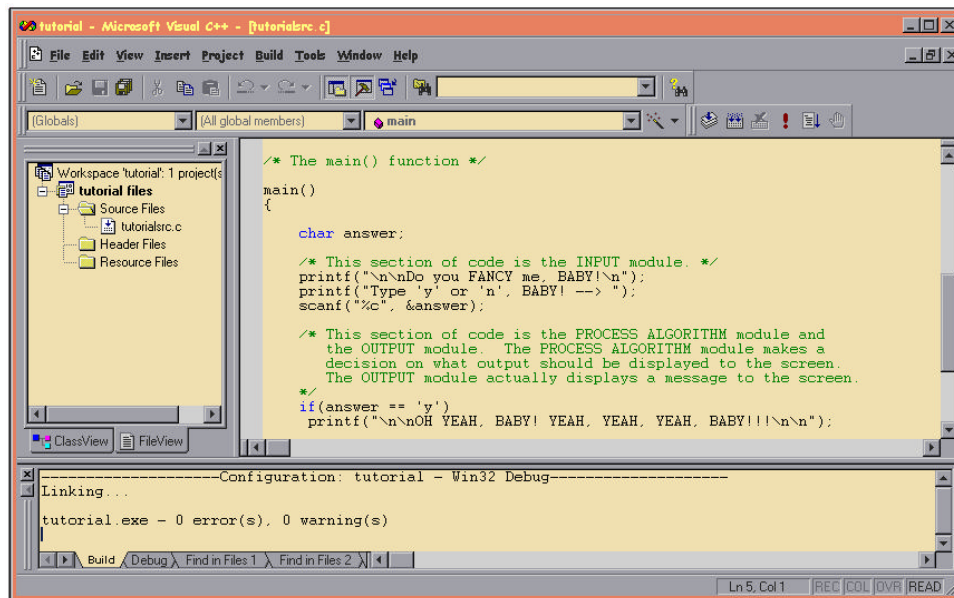



FIGURE 10. The *Output Window* shows the results of the linking process.

You have now created a *Project* file (with corresponding *Workspace* and *Workspace Options* files), typed a C program into the built-in editor, and compiled and linked a program. Now, you are finally ready to run the program and observe the results.

RUNNING A PROGRAM

STEP #8:

To run your program, execute the command “*Execute tutorial.exe*” either from the *Build* menu, pressing the Ctrl + F5 keys, or selecting its shortcut icon . If you run the program three times with a different input response, you will see the output results displayed in a *Console Window* as shown in Figures 11, 12, and 13.

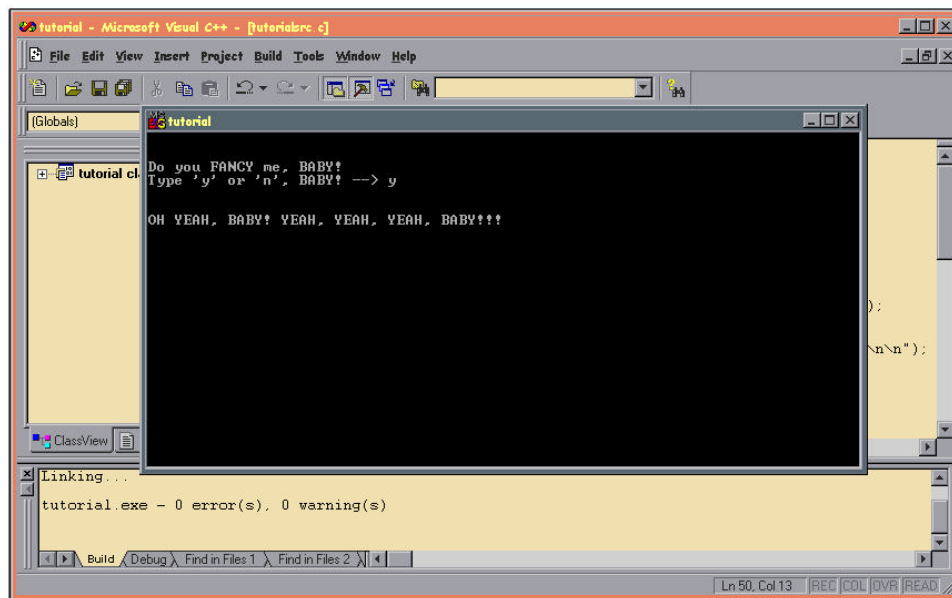


FIGURE 11. First output run of sample C program.

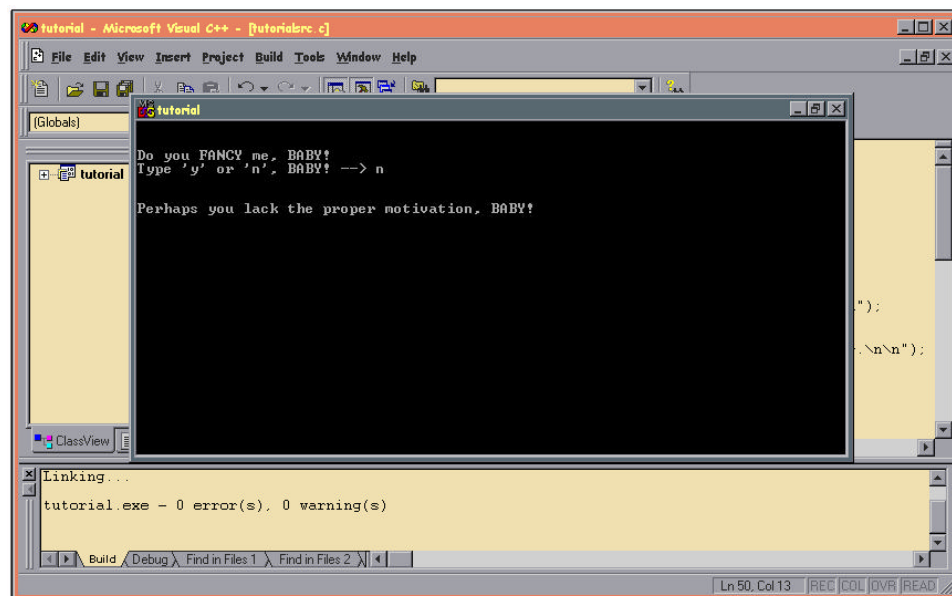


FIGURE 12. Second output run of sample C program.

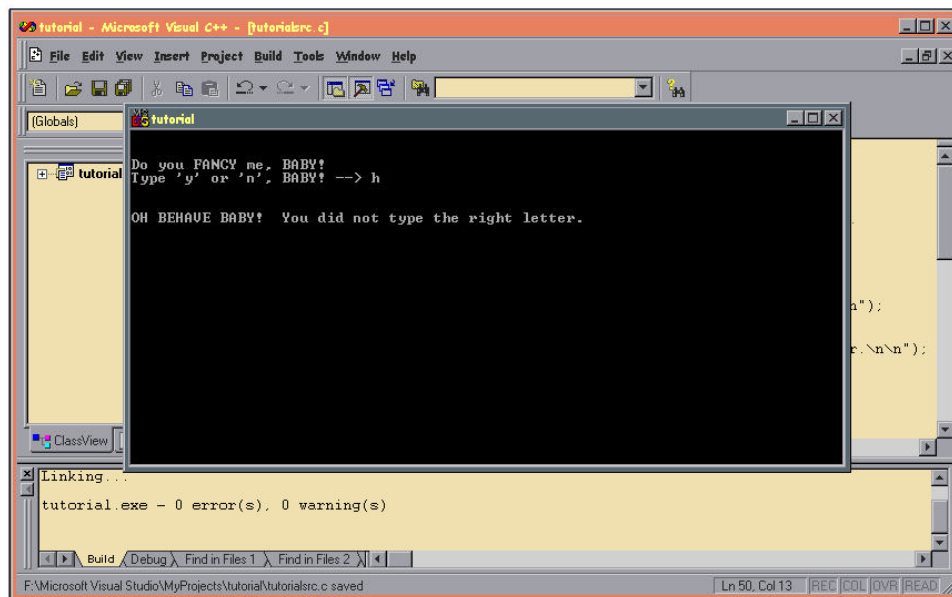


FIGURE 13. Third output run of sample C program.

When you run your C/C++ programs from within the Visual C++ IDE environment, you will see one additional item not shown in the last three figures. The very last line displayed in the *Console Window* will read *“Press any key to continue”*. This message is not *part of your C/C++ programs but something Visual C++ places into the Console Window* to pause the display. (If you run *“tutorial.exe”* from an MS-DOS prompt or icon, you will not see this message.)

One thing that you may want to do is run the program in *Full-screen mode* rather than *Window mode*. To do this, place the mouse cursor over any part of the *Console Window* and click the right mouse button once. The *Properties* Dialog box for the *Console Window* should appear as shown in Figure 14. Next, mouse click on the tab labeled *“Screen”*. Select the option *“Full-screen”*, press *Apply*, then *OK* (see Figure 15). A full-screen version of your program will appear. (**NOTE:** Sometimes, *Windows 95/98* will minimize the window on the taskbar. If this occurs, click the MS-DOS icon on the task bar and a full-screen version of your program will appear.)

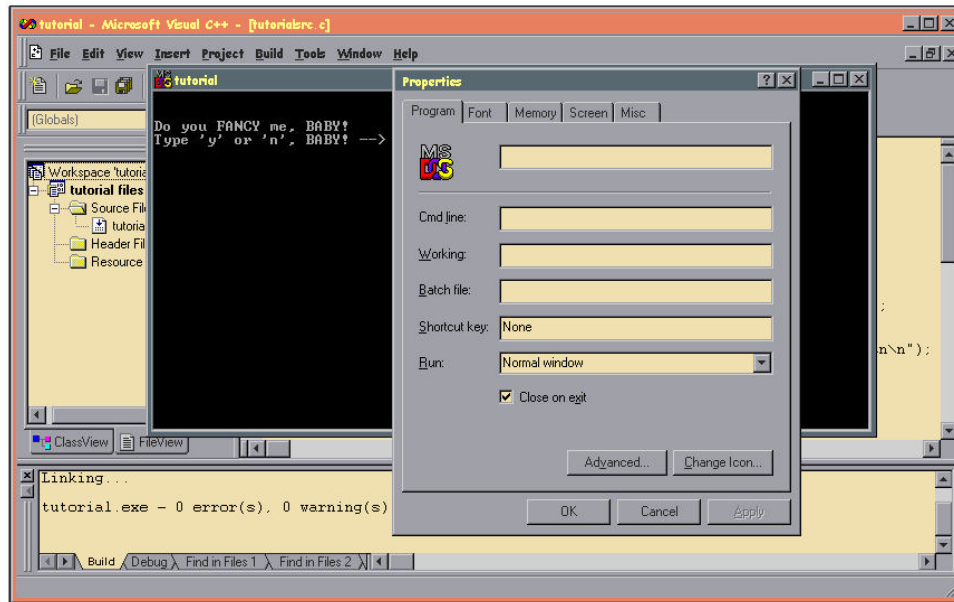


FIGURE 14. The *Properties* Dialog box of the *Console Window*.

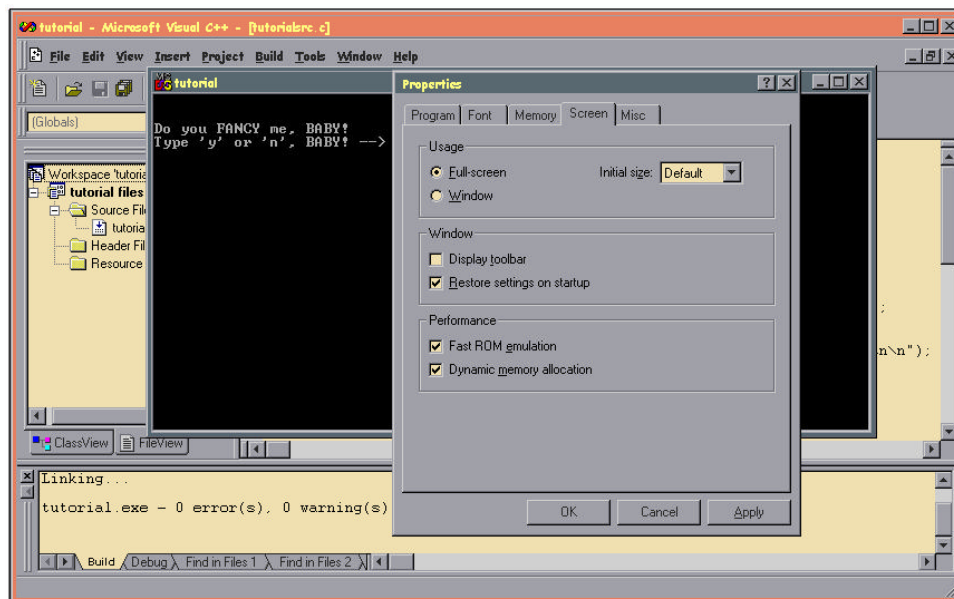


FIGURE 15. Selecting the *Full-screen* option.

The following two steps will explain how to print your source code and output results to hardcopy.

PRINTING SOURCE CODE & PROGRAM OUTPUT

STEP #9:

Printing your source code from the Visual C++ Editor is quite easy. Just execute the “*Print*” command from the *File* menu or press the Ctrl + P keys. Make sure that each line in the printed output is aligned exactly as shown in the editor window. If you do not exceed column 95 in the *Document Window*, the printed output will always be aligned properly. If the printed output is not aligned properly (for example, you see random “*word-wrapping*” or “*text-clipping*” of program text), you will have to go back to the editor and adjust the appropriate program text lines so that they do not exceed column 95. Random word-wrapping and text-clipping makes a program hard to read and looks sloppy.

STEP #10:

To print the output results when running the program from within Visual C++, from the command-line, or from an icon, run the program in either *Window* or *Full-screen* mode until the program reaches completion (or when the program pauses). To capture the program’s screen contents in text-mode (sometimes called a “*text screenshot*”), press **ALT + PRINT SCRN** on the keyboard. (**NOTE:** If the output runs in a *Console Window*, make sure that the window is active before pressing **ALT + PRINT SCRN** on the keyboard otherwise you will capture the entire Desktop screen.) The output will be copied to the Windows Clipboard. Then, open any word processor (such as Word or WordPad) or text editor (such as NotePad), and select *Paste* from the *Edit* menu to insert the output into a blank document. Now, you can print the output results to hardcopy.

Sometimes, you may want to capture the program’s output in graphics-mode. For example, many console-mode programs use colors and character-mode graphics (such as lines, boxes and unique character symbols) to embellish the normally boring black-and-white text output that is default behavior for console-mode programs. Capturing a program’s output in text-mode results in the loss of this information. To capture the program’s screen contents in graphics-mode (sometimes called a “*graphics screenshot*”), the program must be run in a window and not full-screen. With the window active, press **ALT + PRINT SCRN** to copy the *Console Window* into the Windows Clipboard. Next, open a word processor and paste the contents of the Clipboard into the text document. Do not use a text editor such as NotePad. Text editors only recognize common ASCII text characters. They do not recognize graphics elements and nonstandard ASCII symbols. (**NOTE:** capturing a program’s output in full-screen mode results in text-mode behavior. You will lose any graphics information from the program’s output.)

Figures 16 and 17 show the results for capturing program output from the sample program using the techniques described above. Figure 16 shows the “*graphics screenshot*” method while Figure 17 shows the “*text screenshot*” method.

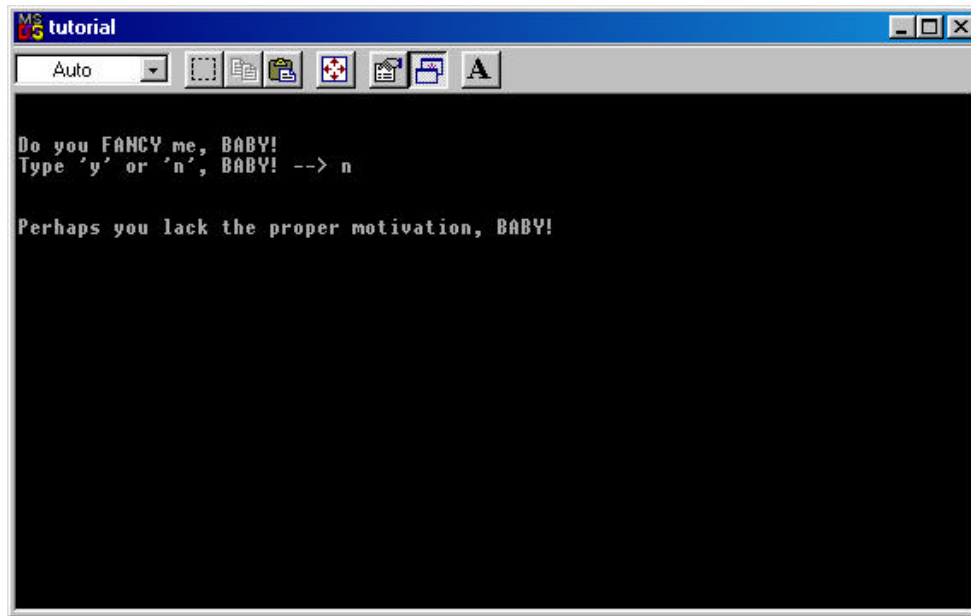


FIGURE 16. The “*graphics screenshot*” method for capturing program output.

Do you FANCY me, BABY!
Type 'y' or 'n', BABY! --> n

Perhaps you lack the proper motivation, BABY!

FIGURE 17. The “*text screenshot*” method for capturing program output.

OTHER ITEMS OF INTEREST

REOPENING A PROJECT:

As previously mentioned, once you create a *Project*, a *Workspace* is automatically created for you by Visual C++. The next time you start Visual C++, select the “*Open Workspace...*” command from the *File* menu. Navigate to the folder that stores the *Project* you want to open. In this directory, you will see a file that has the name of your *Project* but ends with a “.dsw” extension. This is the *Workspace* file and contains all the information needed for Visual C++ to open your *Project*. (For example, in this tutorial, the *Project* name is “*tutorial*” and is stored in a project folder called “*tutorial*”. Navigating to this folder reveals a file called “*tutorial.dsw*”. This is the *Workspace* file and should be selected to open the *Project*.) Once you create a *Project*, you can always open up the *Project* using the “*Open Workspace...*” command. The *Workspace* file internally calls the *Project* file “*tutorial.dsp*”, in this example) so that all of your compiler and linker options remain intact.

Opening up a *Project* using the above procedure can be tedious and time-consuming since you must initiate several steps. A shortcut does exist that greatly simplifies the process of opening a specified Project. Assuming your *Project* (.dsp), *Workspace* files (.dsw and .opt), and program source files are located in a *Project* folder on the local hard drive, simply navigate to the *Project* folder and locate the *Workspace* file icon. Double-click the *Workspace* file icon to initiate the following sequence.

1. Visual C++ IDE will open.
2. The *Workspace* file will call the *Project* file and the *Workspace Options* file to load project information and IDE settings.
3. The *Document Window* loads the source file you were last editing during a previous *Project* session.

WHERE DO I FIND MY PROGRAM'S EXECUTABLE FILE?:

When you create a *Project*, Visual C++ automatically creates a *Debug* folder inside your *Project* folder. When you successfully build (or link) a program using the *Build* command, you will find the executable file, the object code file, and other intermediate files in the *Project's Debug* folder. By default, the executable file is associated with the following icon



DO I NEED VISUAL C++ TO RUN MY APPLICATIONS?:

You do not need to execute your application from within Visual C++'s IDE. An executable file contains all of the instructions necessary for your application to run “on its own” in a Win32-compatible operating system without Visual C++. During program development, the *Execute* command allows you to quickly run your program for testing without having to leave the IDE environment. Once your application is finalized, you can distribute the .exe to other users or run it on other computers.

USING THE BUILD AND EXECUTE COMMANDS:

There are several shortcuts that can be used to build and/or run a program. In this tutorial, four distinct steps have been emphasized in developing C/C++, Win32 console-mode programs using Microsoft Visual C++ 6.0's IDE. They are *Edit*, *Compile*, *Link*, and *Run*. The three specific commands needed to compile, link, and run C/C++ programs have been shown. However, as a convenience to the programmer, Visual C++ does not require you to execute three separate commands. Instead, you can execute the *Build* command, rather than the *Compile* command, to force the compiler to automatically invoke the compile command (if needed) and the link command in one operation. Likewise, you can

select the *Execute* command to force the compiler to automatically invoke the compile and link commands (if needed) and to run the program in one operation.

As seen in Figure 18, there are also three shortcut command icons that can be used to compile, build, or execute a program from within the IDE.

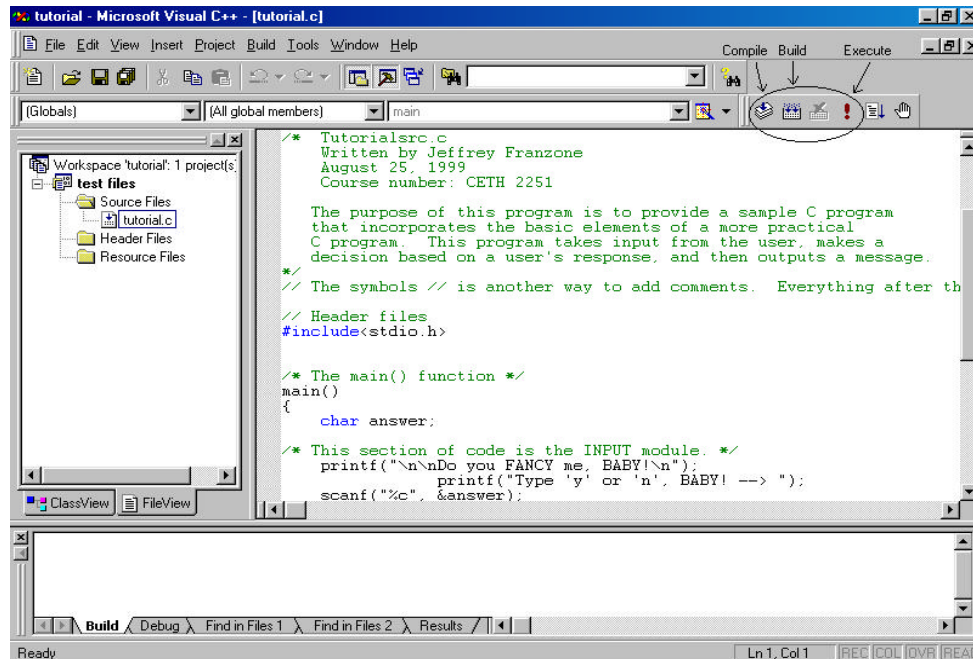


FIGURE 18. Shortcut command icons for the *Compile*, *Build*, and *Execute* commands.

IDE COLOR SCHEMES:

Color schemes for the common IDE window components (such as window borders, title, status, and menu bars, pop-up (or pop-down) menus, scrollbars, edit boxes, dialog boxes, etc.) are set by the Windows operating system and apply to all Windows applications. You cannot individually set a color scheme for the common window components of the IDE from within the IDE. All color scheme information is handled by Windows and is stored in the Registry. You can however, change the current Windows color scheme by selecting the *Appearance* tab from the *Display Properties Dialog Box*. You can invoke this dialog box by either executing the *Display* application found under the *Start / Settings / Control Panel* folder or by placing the mouse cursor anywhere on the Windows Desktop, single-clicking the right mouse button, selecting the “*Properties...*” command from the pop-up menu, and then selecting the *Appearance* tab. Keep in mind that whatever changes you make to the color scheme will cause all Windows applications to have the same color scheme.

Figure 18 shown above illustrates the effects of a color scheme change to the IDE using the above methods. Note that the background color of the *Project Window*, *Output Window*, *Document Window*, and text edit and list boxes have all changed from a tan color to white and that the border color enclosing the IDE screen has changed from orange to light gray.

Although you cannot change the color schemes of common window components, the IDE does allow you to change the foreground and background colors of text, keywords,

identifier names, debugging constructs, etc., as well as changing font types and sizes displayed in the various IDE windows. Simply select the *Format* tab from the *Tools / Options...* dialog box as shown in Figure 19 to see the available options.

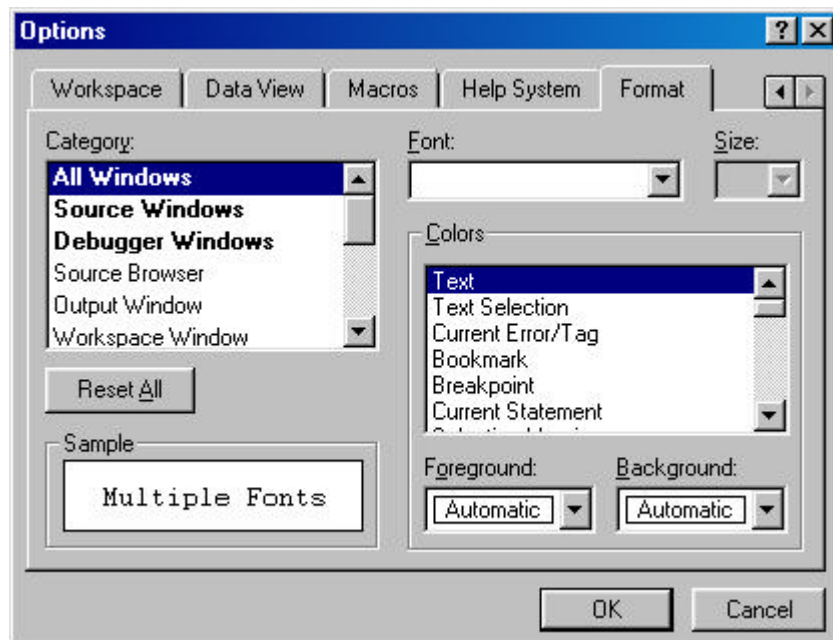


FIGURE 19. The *Format* tab selected from the *Tools / Options...* dialog box contains some limited options in changing color schemes for text, keywords, debugging constructs as well as changing font types and sizes.

WHAT YOU HAVE LEARNED!

With this tutorial, you should have learned the ideas listed below. Please review applicable sections of the tutorial for those items you inadvertently passed over or do not understand.

1. The four major steps in writing a C/C++ program.
2. The meanings of the terms “*Edit*”, “*Compile*”, “*Link*”, and “*Run*”.
3. The meanings of the terms “*source code*”, “*object file*”, and “*executable file*” and the file extensions associated with them.
4. The meanings of the terms “*console-mode*” and “*Win32 console-mode*” programs.
5. The major IDE screen areas.
6. The steps required to create an empty *Project* for a Win32 Console Application.
7. The meanings of the terms “*Project*” and “*Workspace*”.
8. The major files that are internally generated by Visual C++ when an empty Win32 Console Application Project is created.
9. The purpose, significance, and file extensions associated with the “*Project file*”, “*Workspace file*”, and “*Workspace Options file*”

10. The name and location of the folder where a program's executable and object files are stored.
11. How to create a new C/C++ source file and how to determine the appropriate file extension.
12. How to list project files in the *Project Window* and how to view project files in the *Document Windows* using the *Project Window*.
13. How to manually enter a C/C++ source file into a blank document using the built-in editor.
14. The maximum line length that can be entered into a source document without random text-clipping or word-wrapping when printed from the IDE.
15. How and when to save your source and *Workspace* files using the IDE.
16. The advantages of storing *Project* files on the local hard drive during development and when it is appropriate to store *Project* files on a floppy disk.
17. How to open a previously created *Workspace* from a floppy disk or local hard drive assuming all appropriate files exist and how to create a default *Workspace* when only the source file exists.
18. How to "*compile*" a C/C++ source file using the "*Compile...*" command.
19. The purpose of the *Output Window* and how it helps you detect syntax errors in your source code.
20. How to create an executable file by "*linking*" object code using the "*Build...*" command.
21. How to run your program from within the IDE by using the "*Execute...*" command.
22. When does the "Press any key to continue" message appear?
23. How to switch the running program from a "*windowed*" session to a "*full-screen*" session.
24. How to print a source file from within the IDE.
25. How to capture and print program output using a "*text screenshot*" or "*graphics screenshot*".
26. How to open a previously created Project without first starting the IDE and executing menu commands.
27. The meaning of an executable file (standalone application) and that it can be run on different computers without Visual C++.
28. How to compile and link a program with a single command (*Build...*) and how to compile, link, and run a program with a single command (*Execute...*).
29. The three shortcut command icons that are used to compile, link, and execute programs from within the IDE.
30. How and where to change the color scheme of the IDE environment and the effect it has on other Windows applications.

SOME CLOSING THOUGHTS

The author cannot stress this point enough. Please complete this tutorial carefully and repeat as many times as needed until you feel comfortable with the software (creating projects, printing, arranging windows, saving, capturing screenshots, etc.) and the process of editing, compiling, linking, and running the sample C program. Do it as many times as needed until you can visualize in your mind the basic steps needed to develop C/C++ programs using Visual C++'s IDE. One of the biggest mistakes beginning programmers make is to not take seriously the tools they will need to learn in order to successfully develop software programs. As you begin to learn the details of the C/C++ language, you will need to use the tools and concepts presented here to formulate and test your own programs. This is a challenging task in-and-of itself. Learn the idiosyncrasies of the software tools so that they can help you develop programs rather than hinder you.

REFERENCES

1. **Microsoft Visual C++ 6.0** contained in **Microsoft Visual Studio 6.0, Professional Edition**, 1998, Microsoft Corporation.
2. **MSDN Library Visual Studio 6.0**, 1998, Microsoft Corporation.