WPUNJ Department of Computer Science

MS Visual C++ 6.0 Student User Manual

Table of Contents

Page No

Program Development with Microsoft Visual C/C++ 6.0

Introduction

This short manual is created mainly for the beginning students in CS230 Computer Science I. It introduces the **Integrated Development Environment** (**IDE**) of Microsoft Visual C++ 6.0 and shows how a C++ program is developed in such an environment.

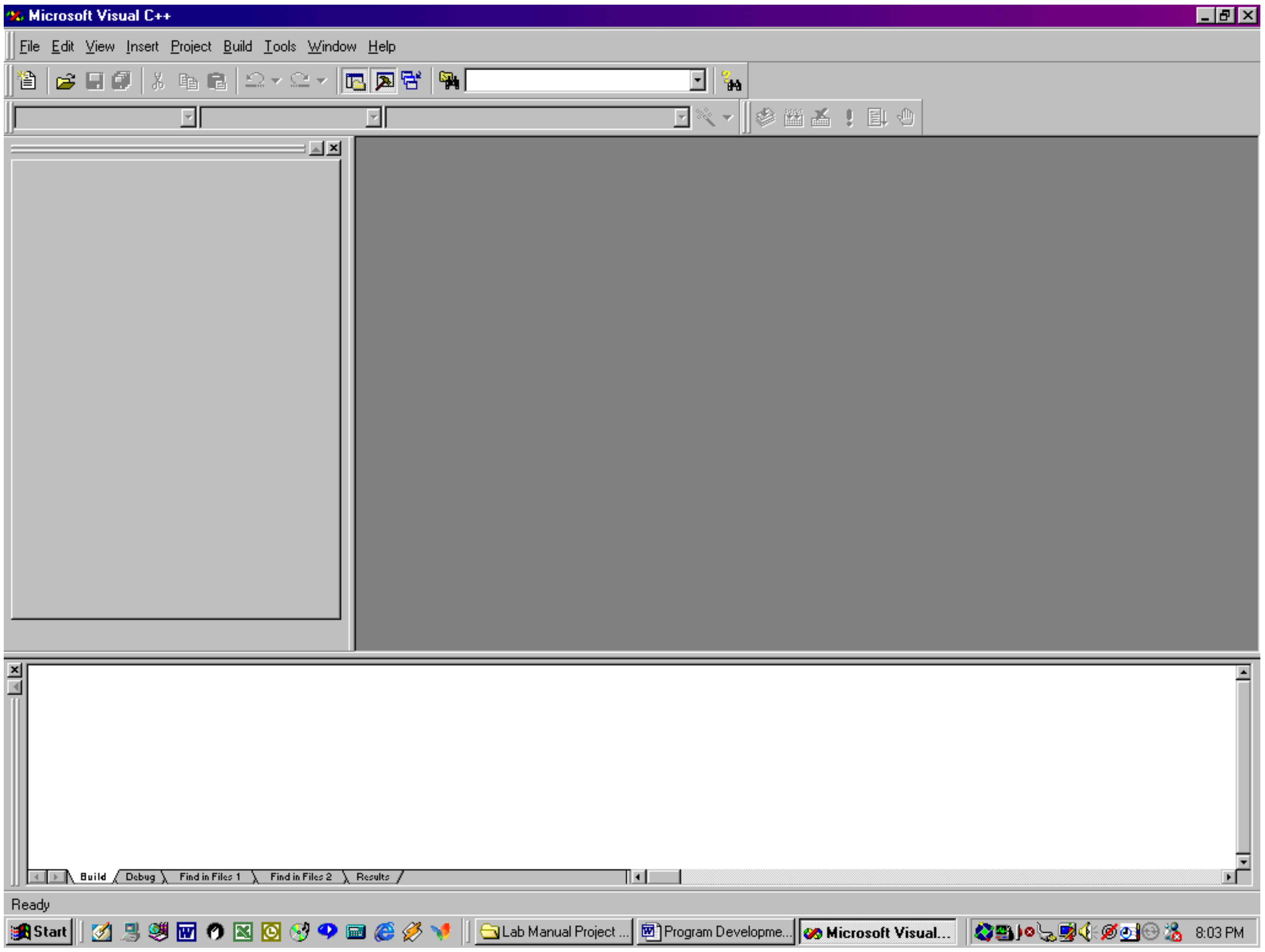
Two simple programs are chosen to allow students to focus on the detailed steps of how to enter, compile, and execute a C/C++ program rather than on the programs themselves. The first is the well-known "Hello World" single-file program; single-file in the sense that the source code entered by the user is contained in a single file with a *.cpp* file extension. The program prints literally the "Hello World" string. The second is a multiple-file program where the source code consists of two *.cpp* files and a user-defined header file with a *.h* file extension. The program displays a given time of the day in both standard (hour ranges between 1 and 12 with AM and PM to distinguish morning and afternoon) and military format (hour ranges between 0 and 23). For your convenience, we list the source code for both programs in the Appendix.

It is noted that in MS Visual C/C++ 6.0, a program is referred to as a **project** and we use these terms interchangeably in this document. As we will see in the following examples, a **project** is actually a directory or folder that contains several sub-directories or sub-folders used to store various files belonging to the **project,** including those entered by the programmer and intermediate files created by the system. For example, all the *.cpp* files are stored in **Source Files** sub-folder and all the user-defined *.h* files are stored in the **Header Files** sub-folder. The basic operations introduced in these manual will certainly make it easier for you to explore and learn more advanced features of the MS Visual C++ **IDE.**

Your feedback is important to us to continually improve this menu. Please forward your comments and suggestions by email to Dr. E. Hu at hue@wpunj.edu.

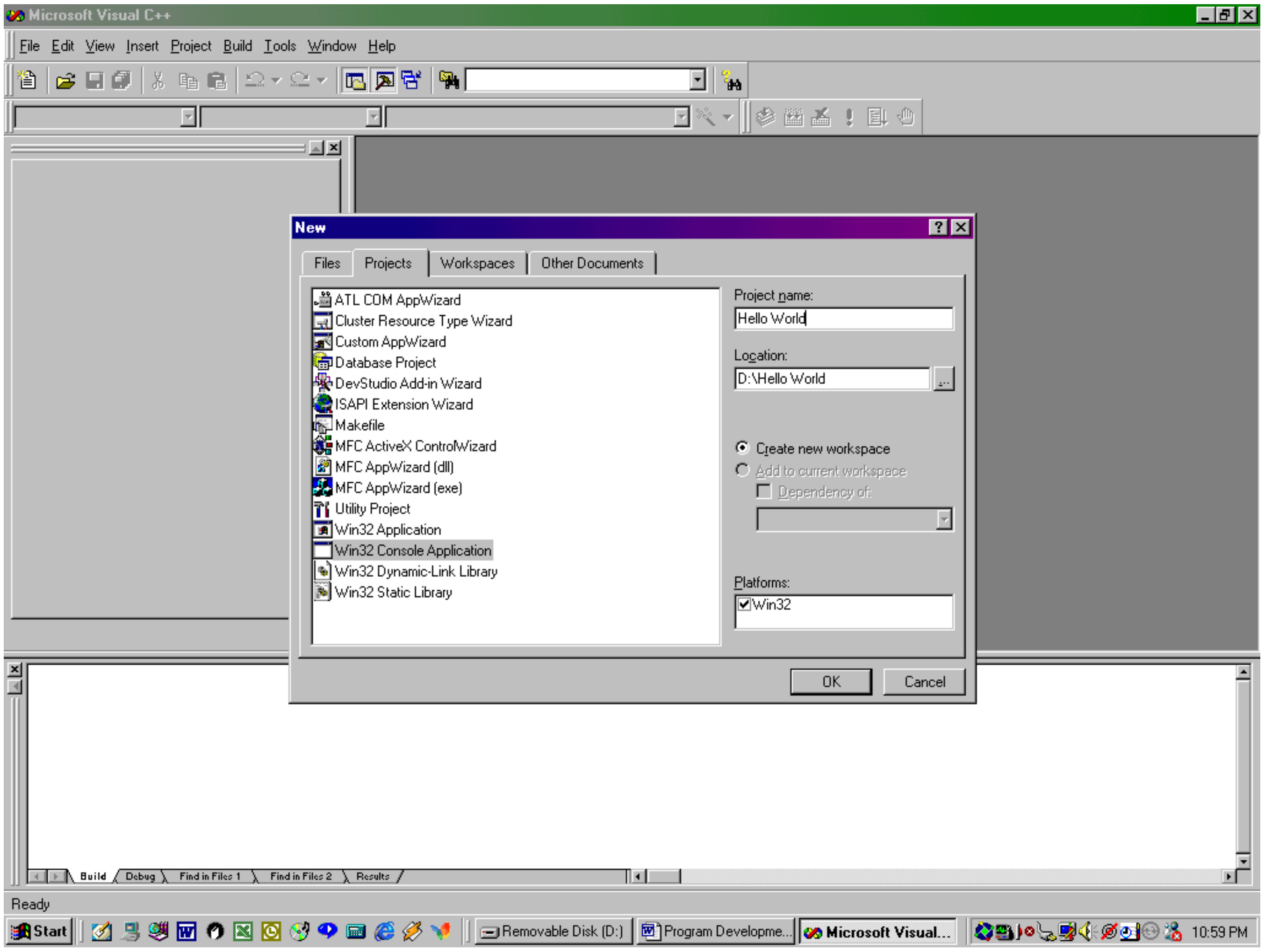
Single-file Project: The *Hello World* Program

Step 1: Launch the MS Visual C/C++ 6.0 software from task bar. The main window of MS Visual 6.0 should be similar to what is displayed below.


Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
Build Debug Find in Files 1 Find in Files 2 Results
Ready
Start
Lab Manual Project ...
Program Developme...
Microsoft Visual...
8:03 PM

Note: Hereafter, all system defined terms including menu items such as **File** will appear in bold and all entries made by programmers such as a file name are in italicized.

Step 2: In the Menu bar, click on **File** then click on **New…** In the **New** dialog box shown below, click on the **Projects** tab then select **Win32 Console Application**. In the **Location box**, select the disk drive (e.g., the D- drive in this demo) that you will use to store your project. In the **Project name** box, enter project name of your choice, e.g., *Hello World* is chosen for this demo. A folder named *Hello World* will be created on the disk in the drive you have chosen (the D drive in this example). This folder will be used by the system to store all files that are associated with the *Hello World* project. Click on the **OK** button.



Project name: Hello World

Location: D:\Hello World

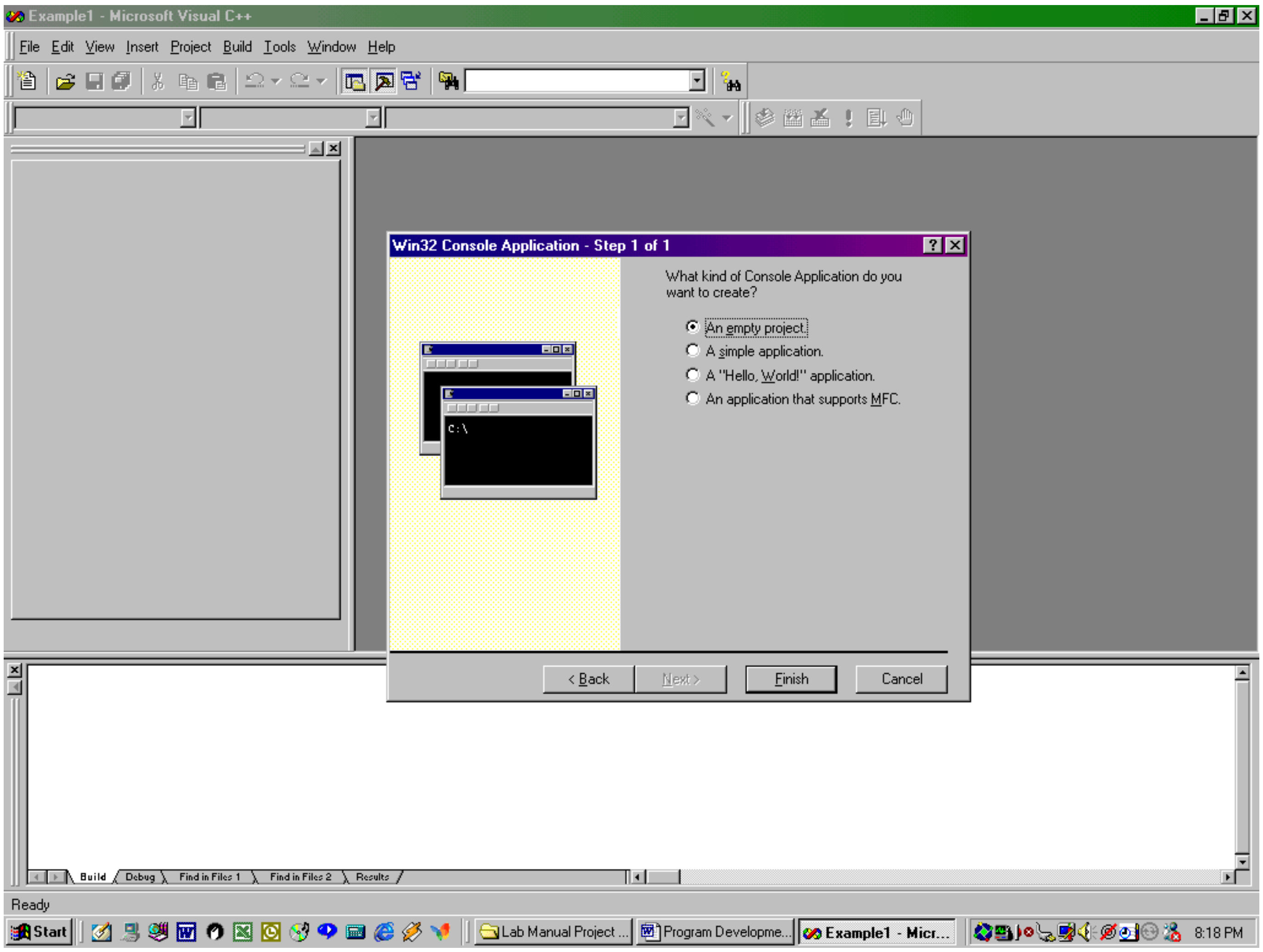
(●) Create new workspace

( ) Add to current workspace

[ ] Dependency of:

Platforms: [✓] Win32

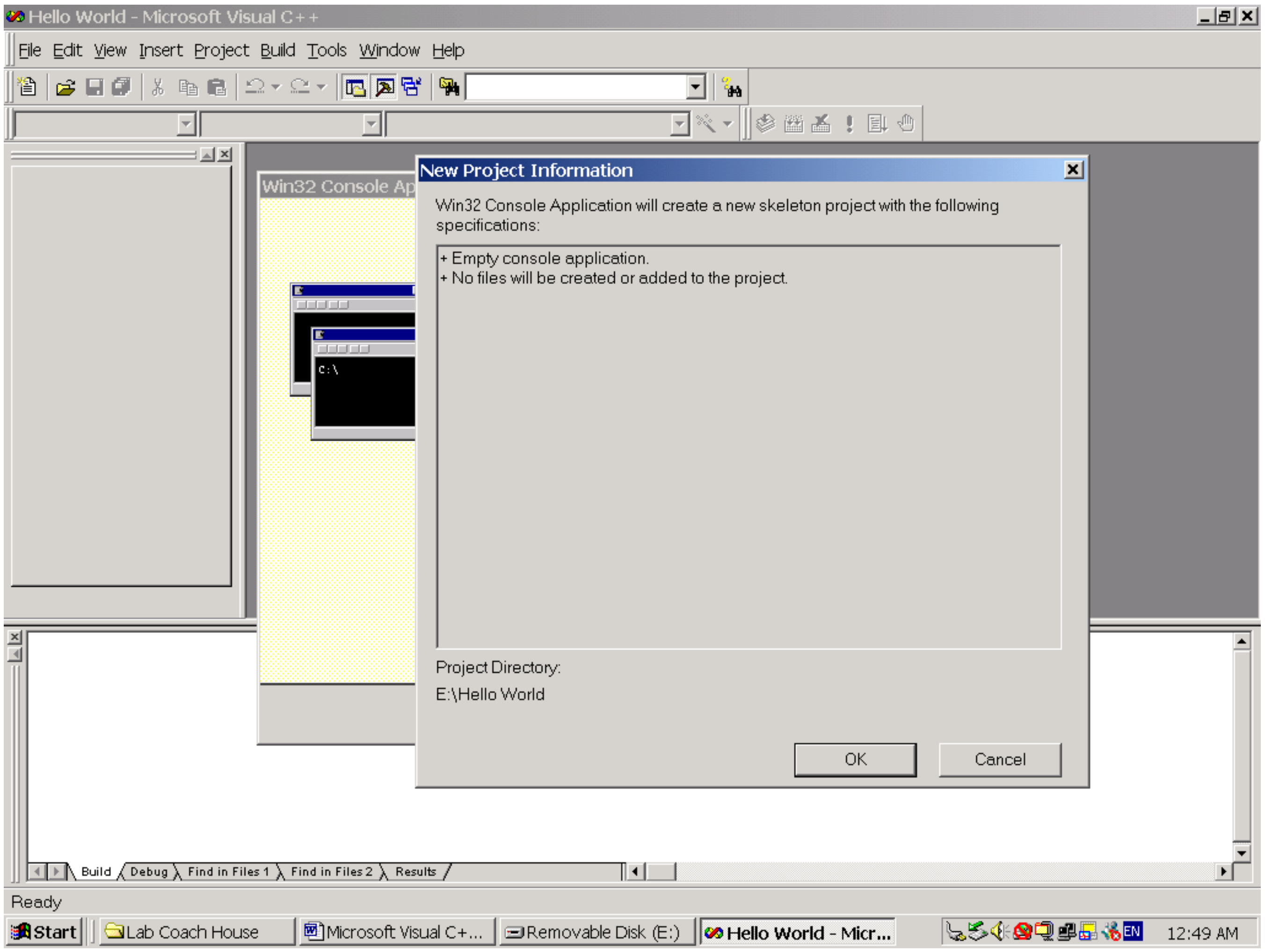
Step 3a: The system displays the window shown below. Click on the **Finish** button in the **Win32 Console Application** dialog box.



What kind of Console Application do you want to create?

- [✓ An empty project.]
- A simple application.
- A "Hello, World!" application.
- An application that supports MFC.

Step 3b: The system displays the **New Project Information** dialog box. Click on the **OK** button.

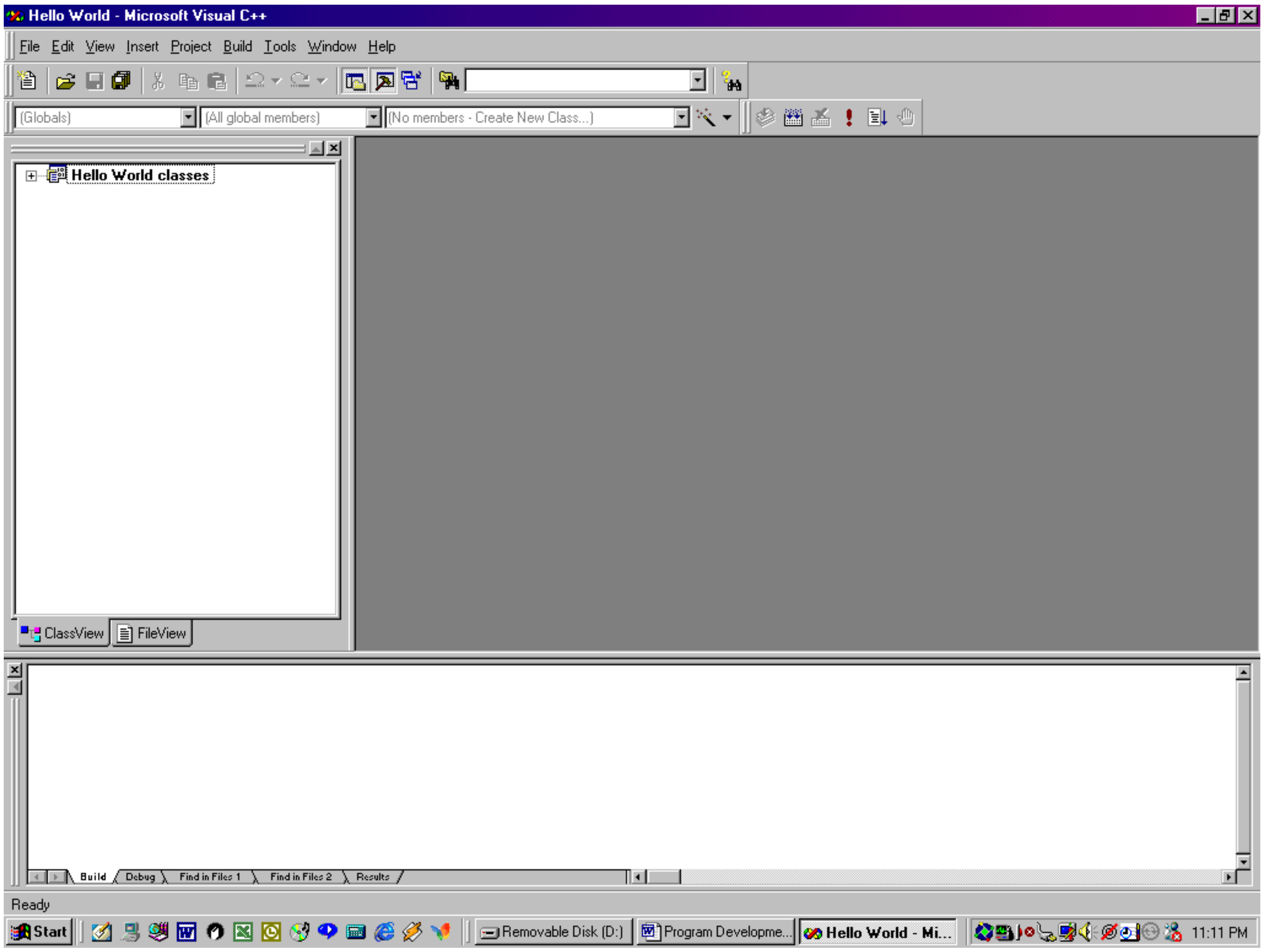


Win32 Console Application will create a new skeleton project with the following specifications:

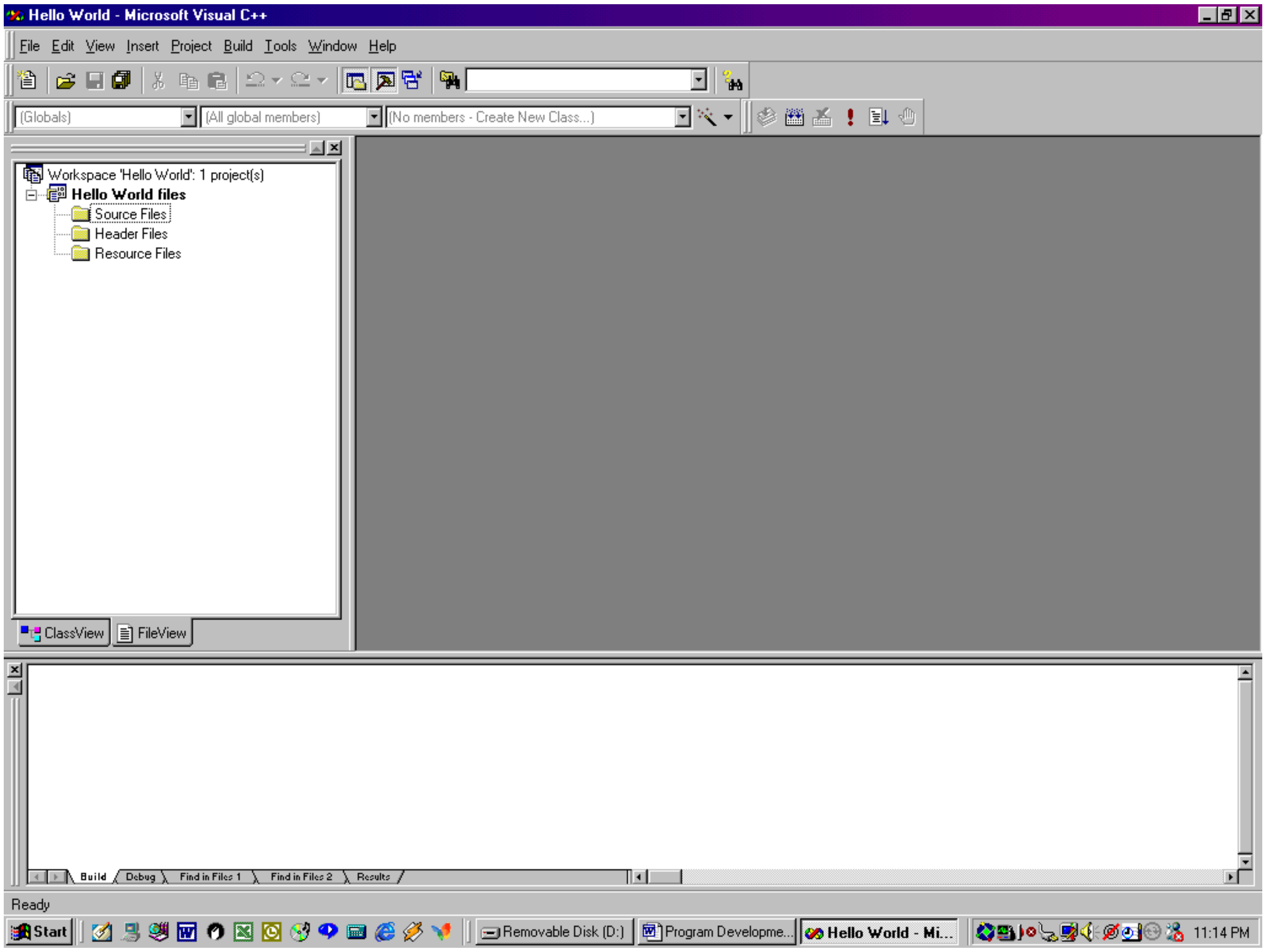
+ Empty console application.
+ No files will be created or added to the project.

Project Directory:
E:\Hello World

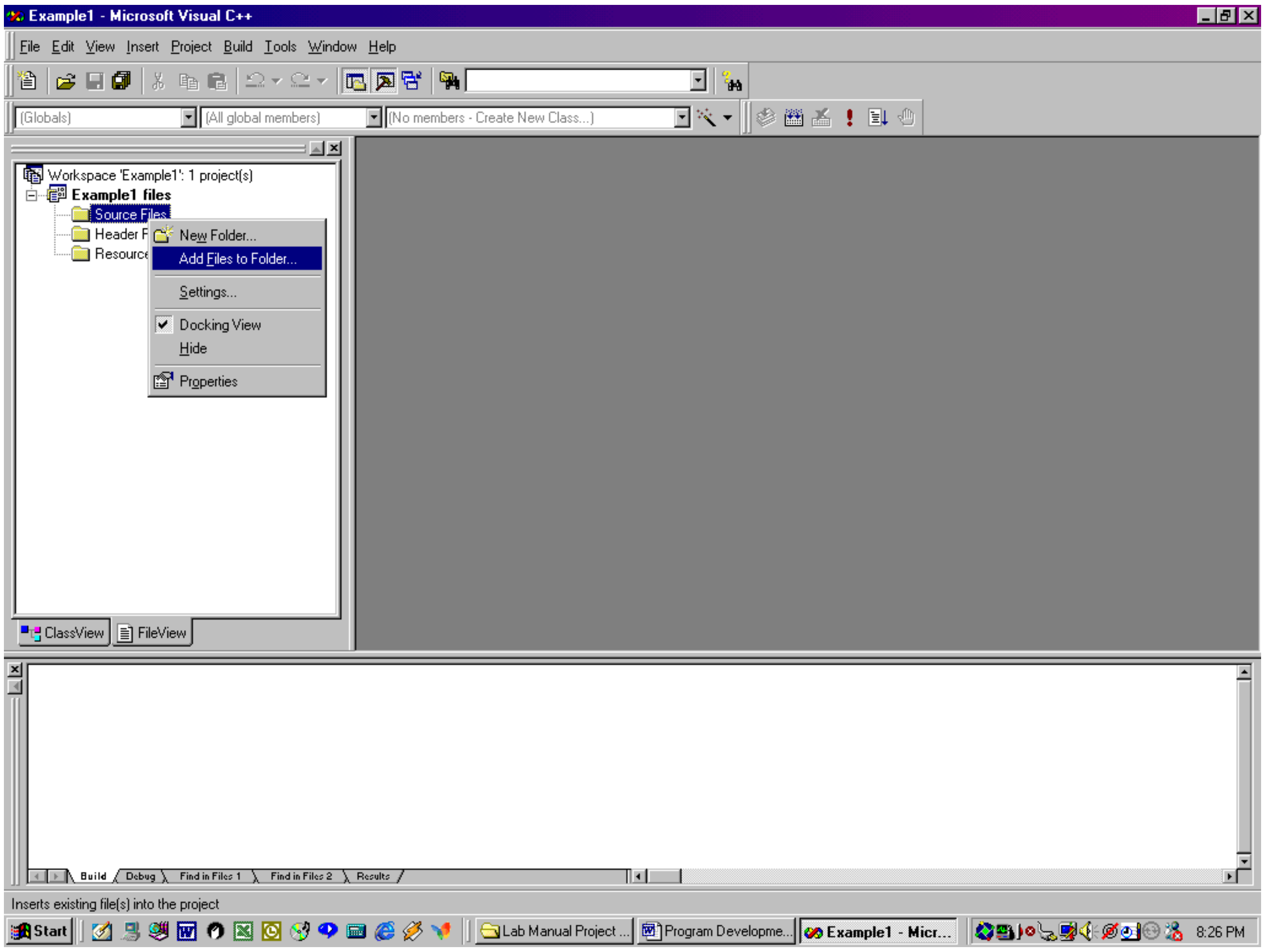
Step 4: On the left pane, the system displays the **ClassView** folder. Select the **FileView** folder by clicking on the **FileView** tab.



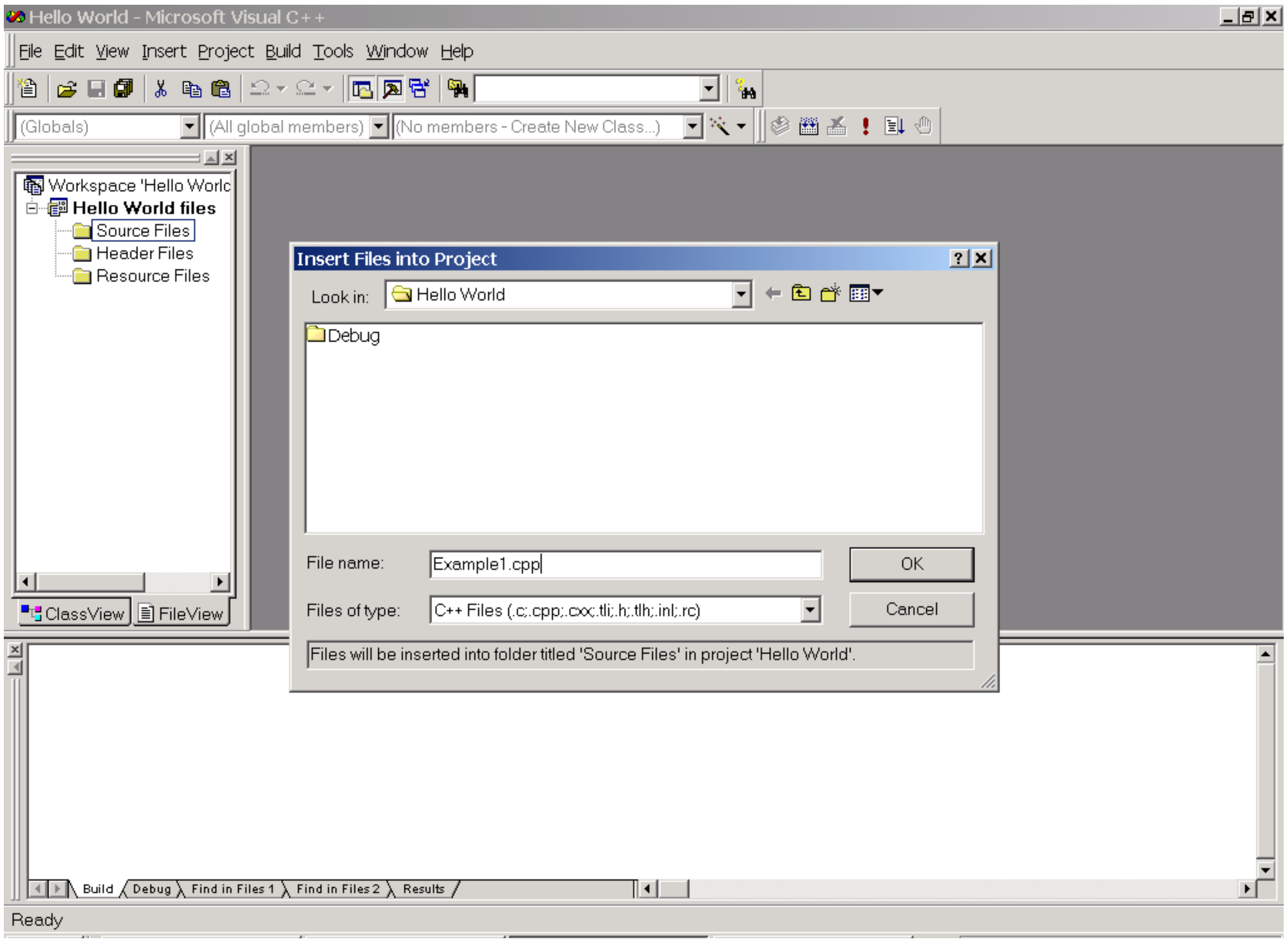
Step 5: The system now displays the **FileView** of the *Hello World* project on the left pane. Click on the **+** button to display three sub-folders – the **Source files, Header Files,** and **Resource Files** folders – for the *Hello World* project as shown below. Note that all three folders are empty because we have not yet “add” any files to these folders. We are now ready to add our empty *Example1.cpp* (filename is arbitrarily chosen) source file to the **Source Files** folder and then enter C++ code into this file.



Step 6a: Right click on the **Source File** folder and select **Add Files to Folder…** in the popup menu**.**



Step 6b: In the **File name** box of the **Insert Files into Project** dialog box, enter the filename of your choice (*Example1.cpp* in this case. Note that it is essential to enter the file extension *.cpp*). Click on the **OK** button to add the *Example1.cpp* file to the **Source Files** folder of the project.

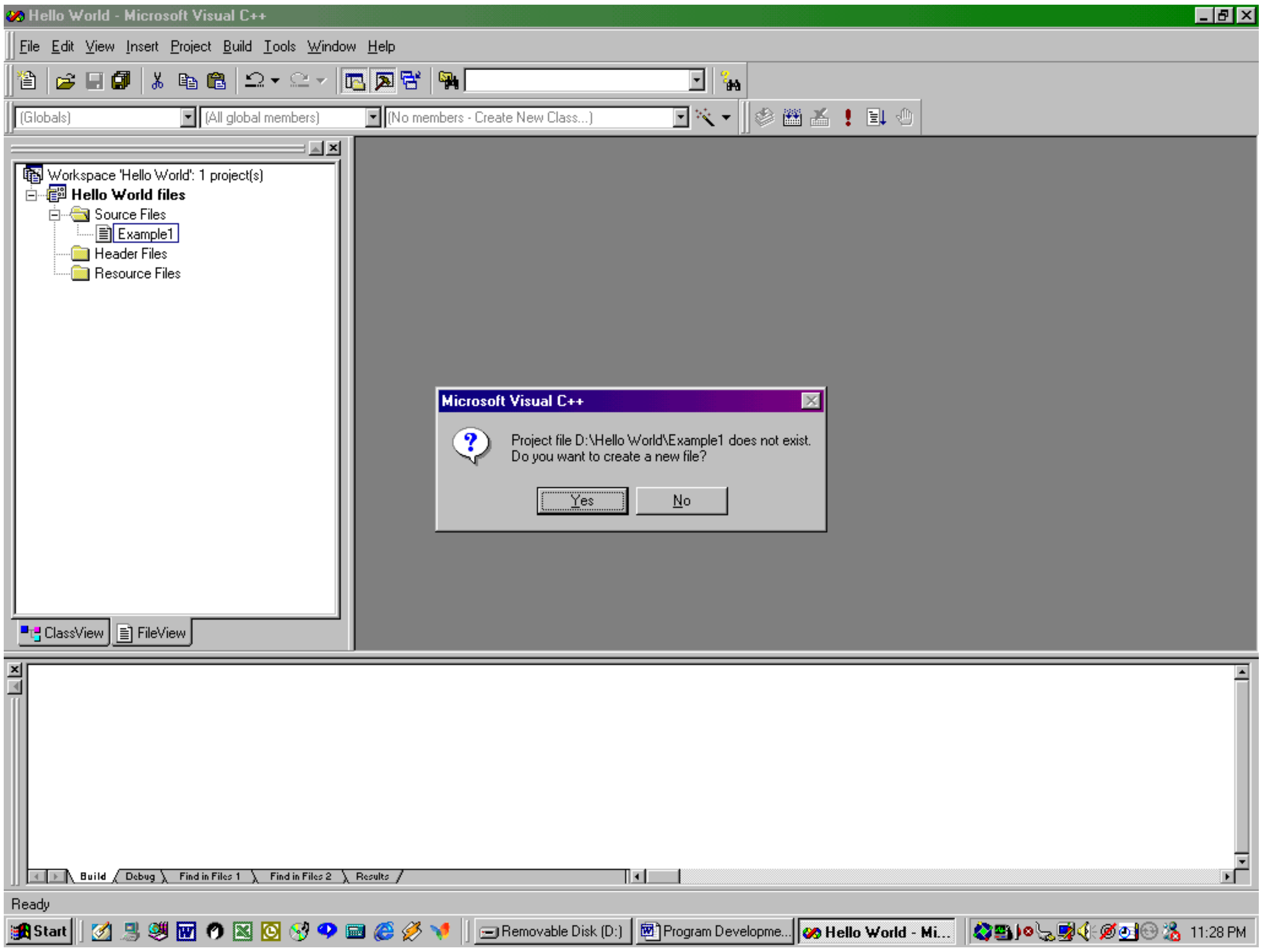


File name: Example1.cpp

Files of type: C++ Files (.c;.cpp;.cxx;.tli;.h;.tlh;.inl;.rc)

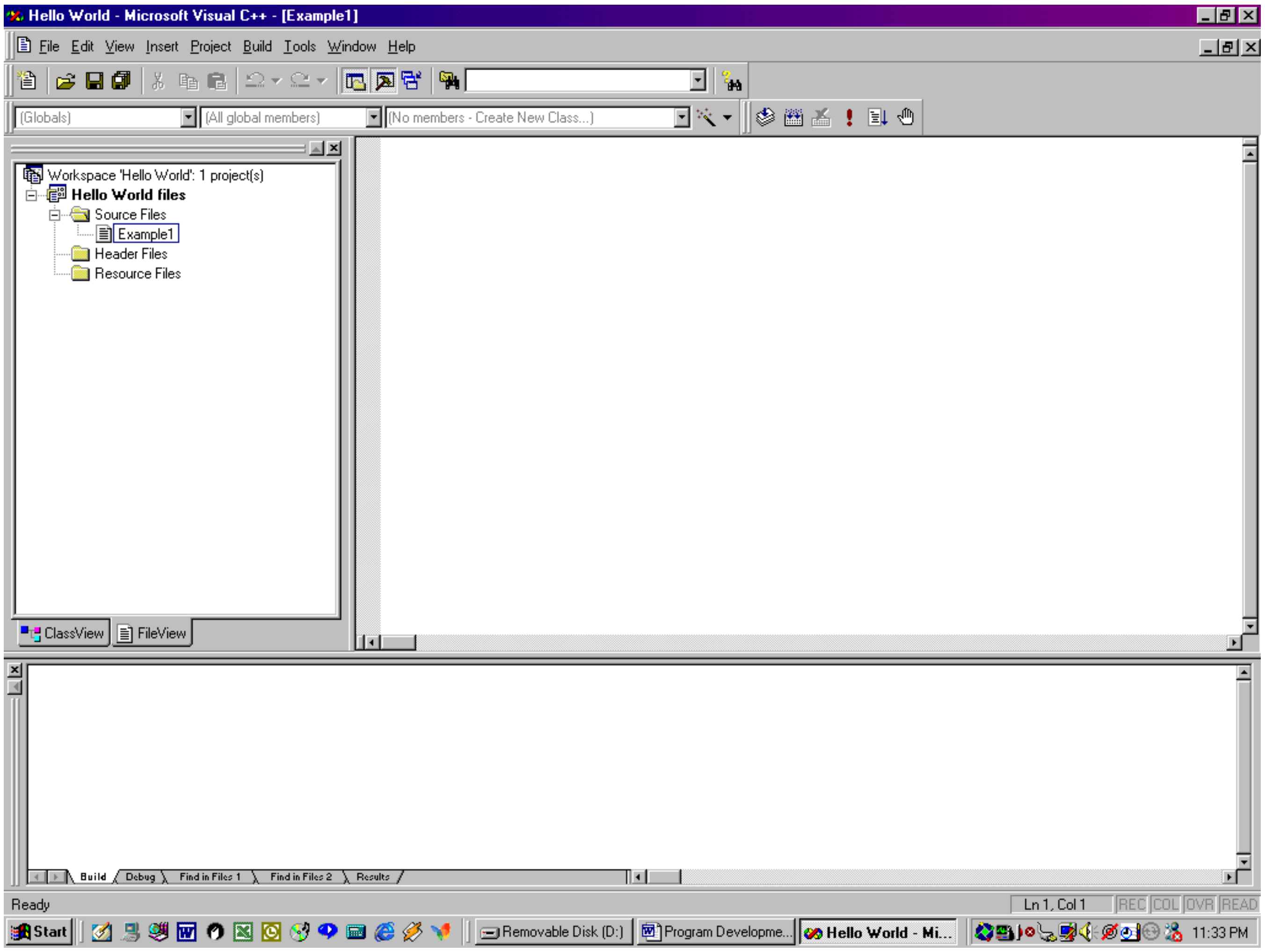
Files will be inserted into folder titled 'Source Files' in project 'Hello World'.

Step 6c: Right click the + button on the left of the **Source Files** folder to show the inserted *example1.cpp* file. Double click the presently empty *Example1.cpp* file, a **Microsoft Visual C++** dialog box is displayed, asking if you want to create a new file (*Example1.cpp*). Click on **Yes** to display the editing window.

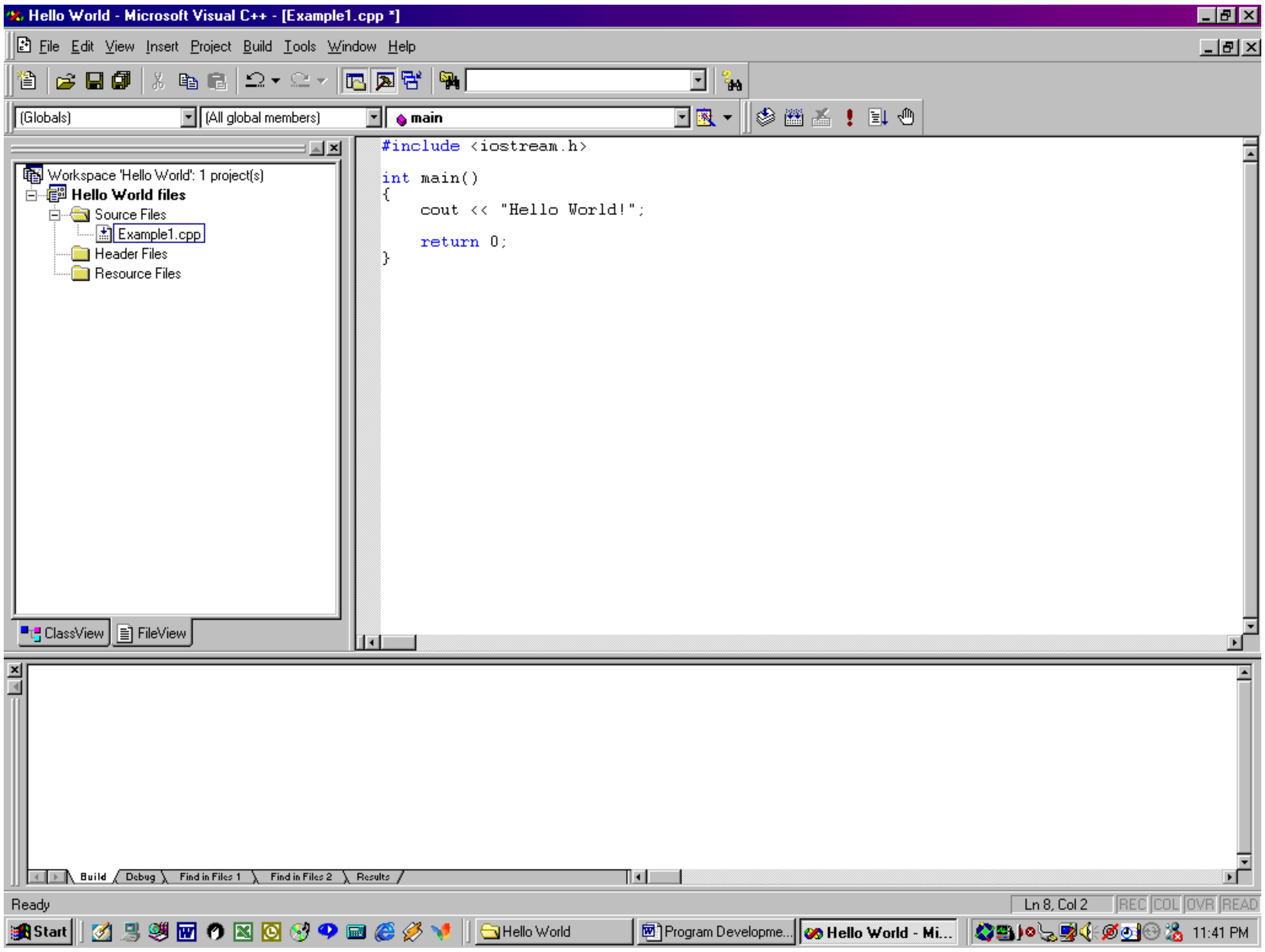


Project file D:\Hello World\Example1 does not exist.
Do you want to create a new file?

Step 6d: An empty editing window appears to the right, waiting for you to enter the source code for the *Example1.cpp* file.



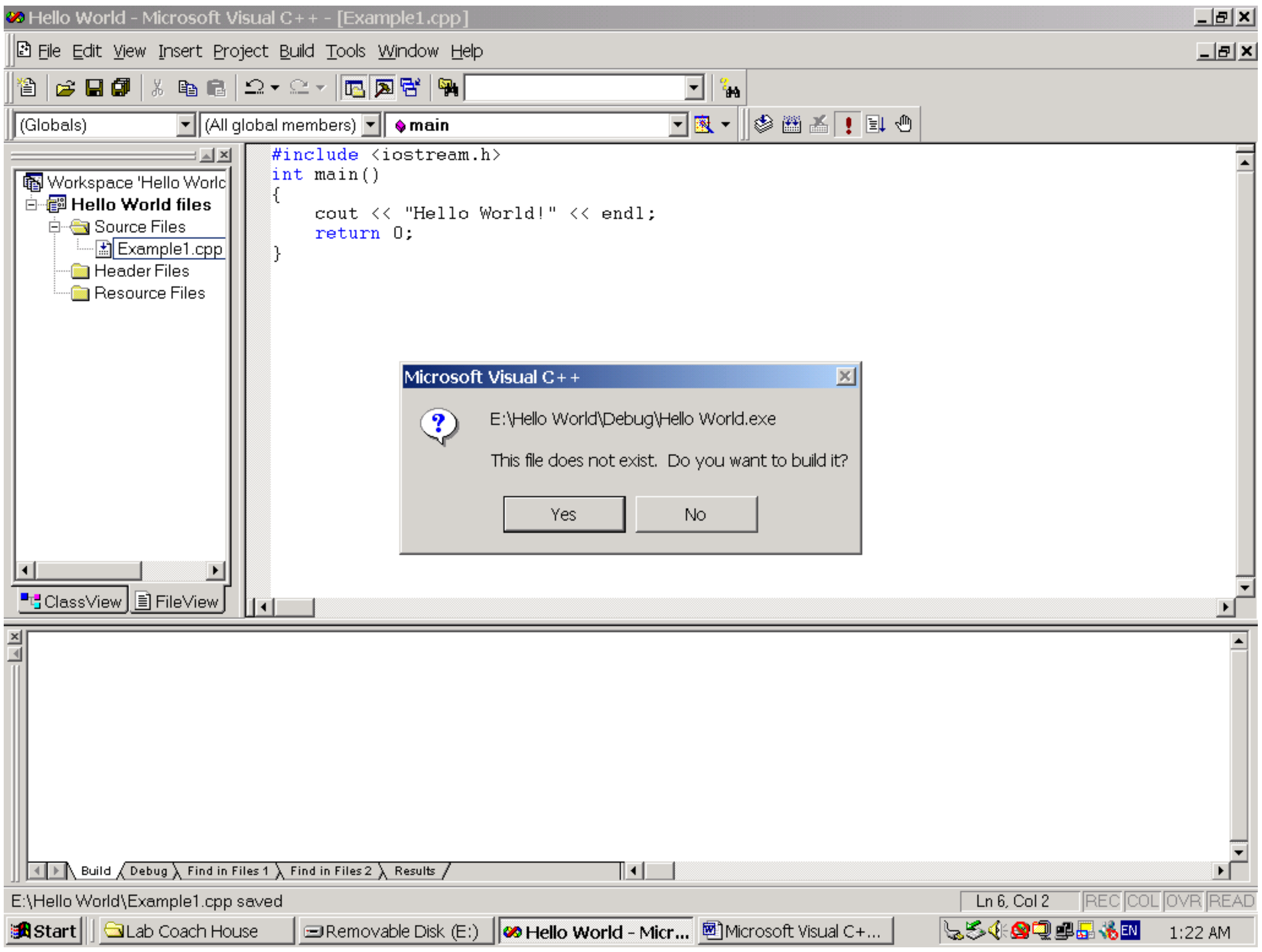
Step 7: Enter the C/C++ source code for *Example1.cpp* file in the editing window as shown. After the code is entered, you are ready to compile and/or execute your program.



```
#include <iostream.h>

int main()
{
    cout << "Hello World!";

    return 0;
}
```

Step 8a: In order to execute your program, it is necessary to go through three steps: compile, link, load, and execute. Normally, these steps performed separately especially when you have a large project that involves multiple files and you want to compile and debug each file separately. For simple projects such as most assignments in CS230, you may want to use the one-click operation that consolidates all four steps together automatically by simply clicking the **Execute Program** icon (a red exclamation symbol) in the tool bar as shown below. Click on it and then click on **Yes** in the dialog box. If there is no warning or error in your project, the system displays the output window (shown in step 8d, the output window has a black background) which displays the result of the execution. If there are warnings and errors, the system displays the diagnostic messages in the message window at the bottom of the screen.

Note that you may choose to skip steps 8b and 8c, which show how compile, link, load, and execute steps are performed separately.



```
#include <iostream.h>
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Microsoft Visual C++

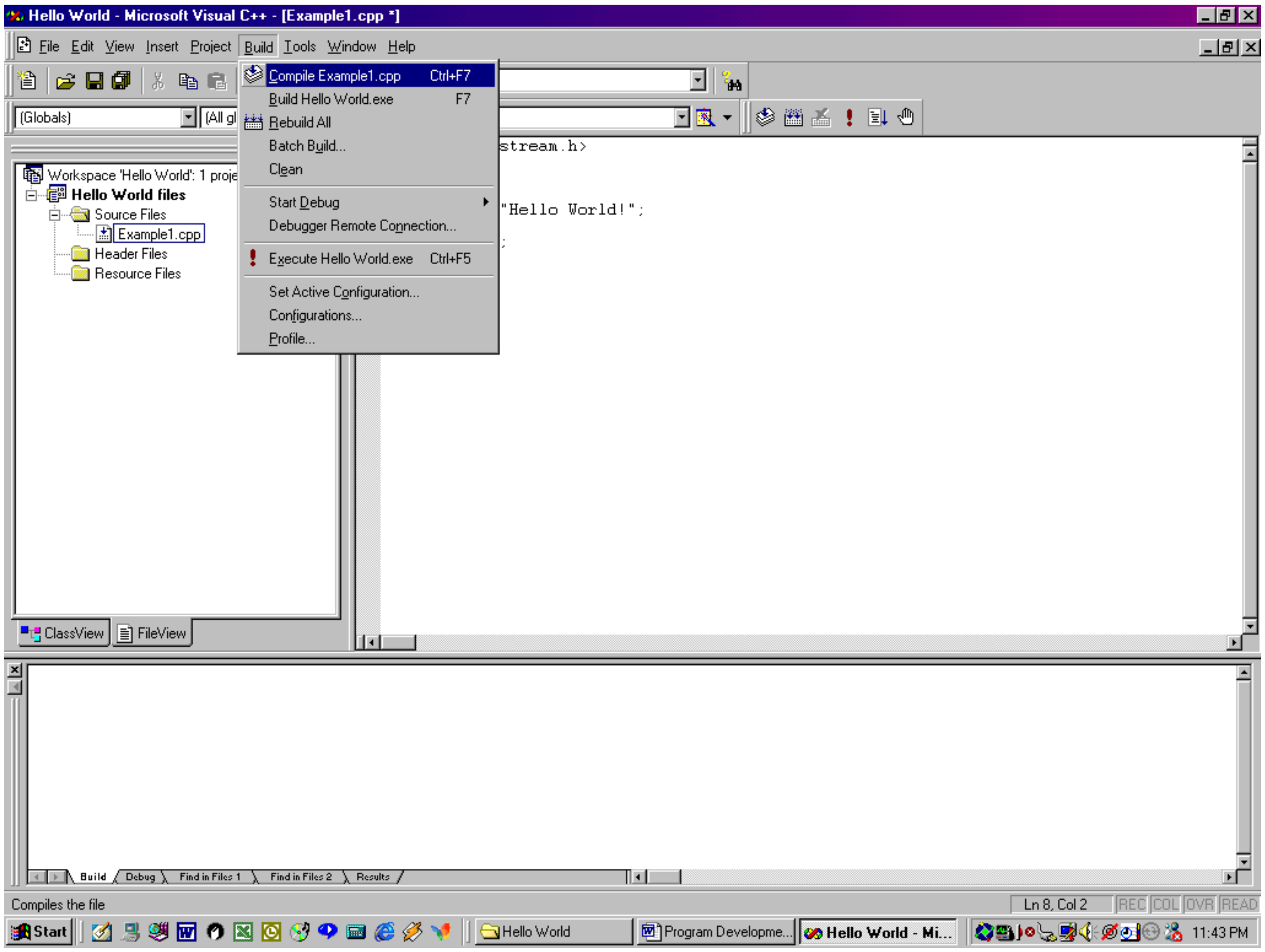
E:\Hello World\Debug\Hello World.exe

This file does not exist. Do you want to build it?

Yes No

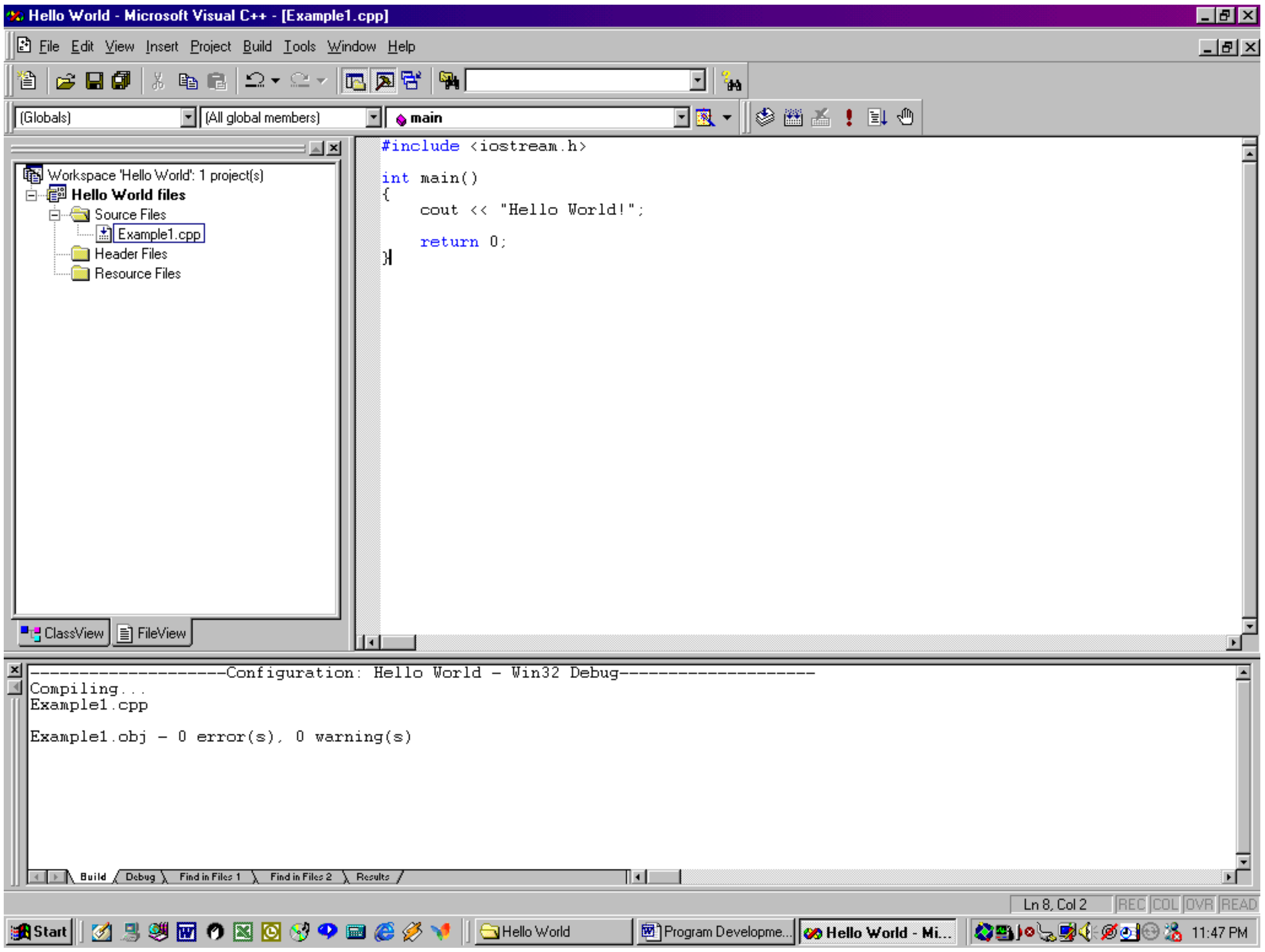
E:\Hello World\Example1.cpp saved

Step 8b: In the menu bar, click on **Build**, and then click on **Compile** *Example1.cpp***.** Alternatively, you may click on the **Compile** icon on the **Build MiniBar** tool bar (these icons are next to the exclamation symbol, move the cursor over them and use the tool tips to find out what each of these icons is supposed to do).



```
stream.h>

"Hello World!";
;
```

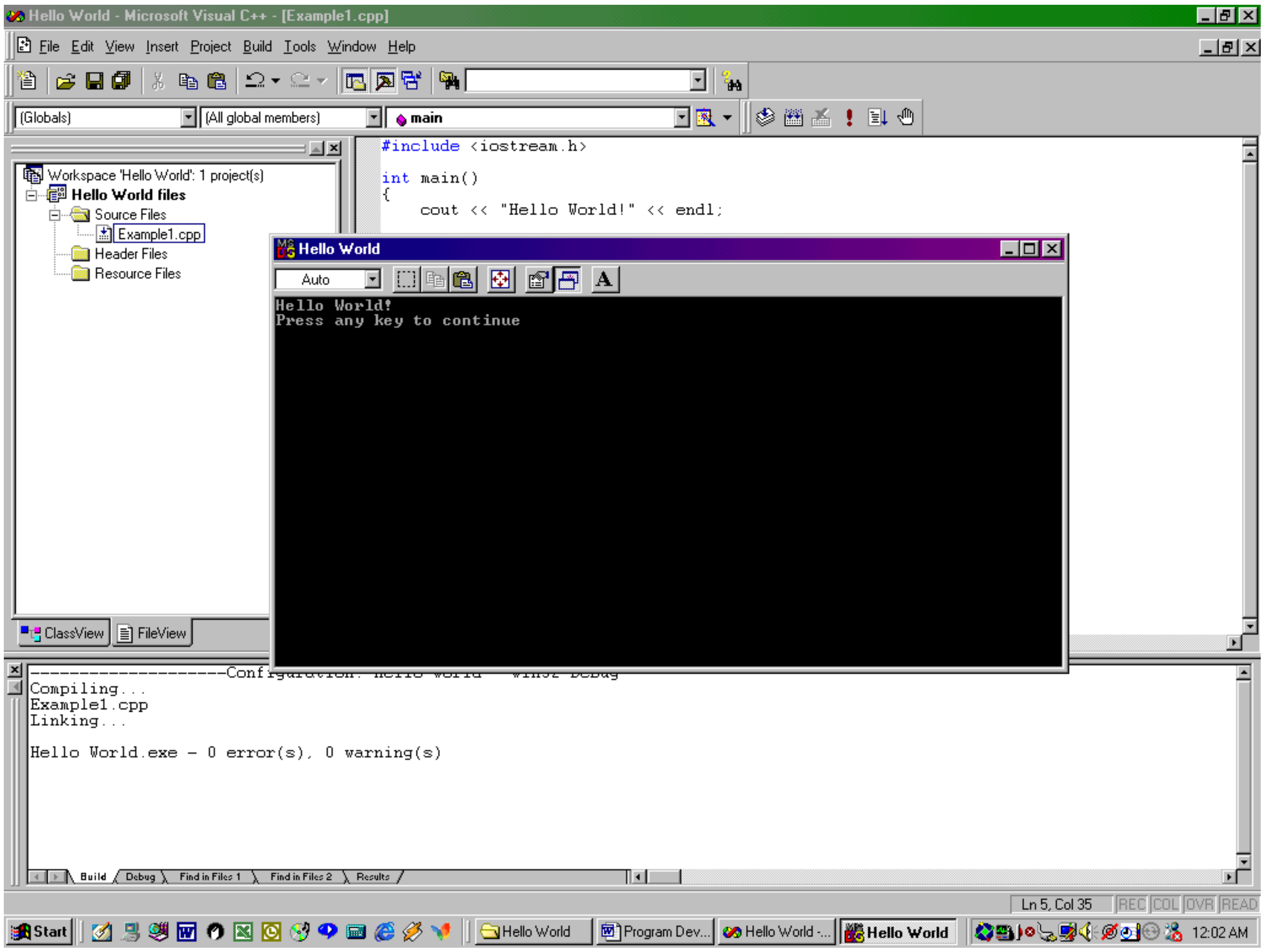
Step 8c: As a result of the compilation, the system displays the total numbers of errors and warnings in the message window at the bottom of the main window. If there are warnings and/or errors (none in this compilation), diagnostic messages are also displayed in the same window. Note that the **Build** tab of the message window has been selected.



```
#include <iostream.h>

int main()
{
    cout << "Hello World!";

    return 0;
}
```

```
--------------------Configuration: Hello World - Win32 Debug--------------------
Compiling...
Example1.cpp

Example1.obj - 0 error(s), 0 warning(s)
```

Step 8d: To execute the program: from the menu bar, click on **Build**, and then click on **Execute** *Hello World.exe* in the pull down menu**.** The results are displayed in the *Hello World* output window as shown on the right.

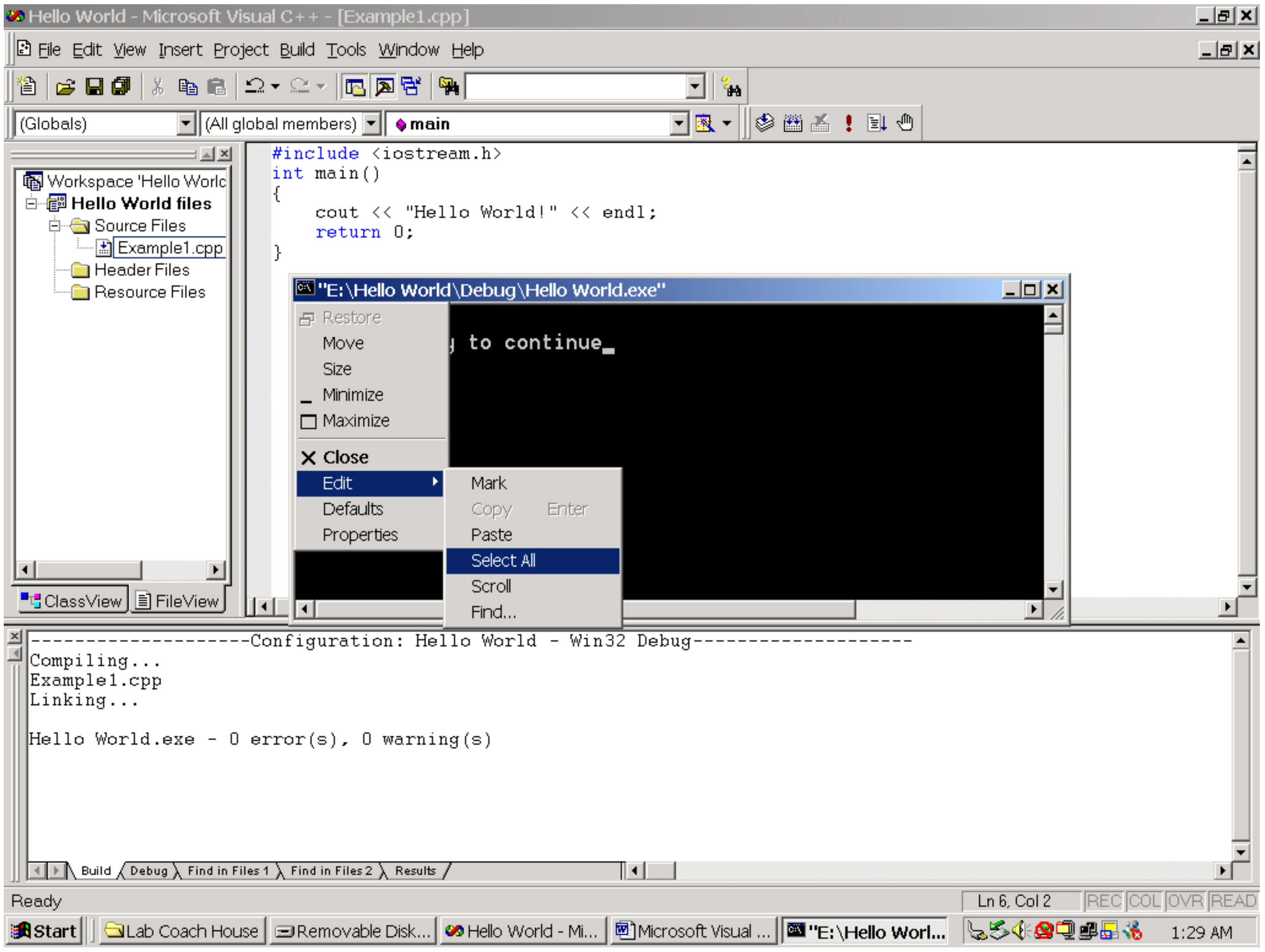


```
#include <iostream.h>

int main()
{
    cout << "Hello World!" << endl;
```

```
Hello World!
Press any key to continue
```

```
--------------------Configuration: Hello World - Win32 Debug
Compiling...
Example1.cpp
Linking...

Hello World.exe - 0 error(s), 0 warning(s)
```

Step 9a: To print the source code in the editing window is straightforward: click anywhere in the source code to activate the editing window. Click on **File** and then select **Print** in the pull down menu. Since there is no print menu associated with the output window, to print the output requires some cut and paste. Activate the output window by clicking anywhere in the output window. Click on the icon at the upper left corner of the title bar of the output window, a pull down menu is displayed as shown below. Click on **Edit**, then choose **Select All**.

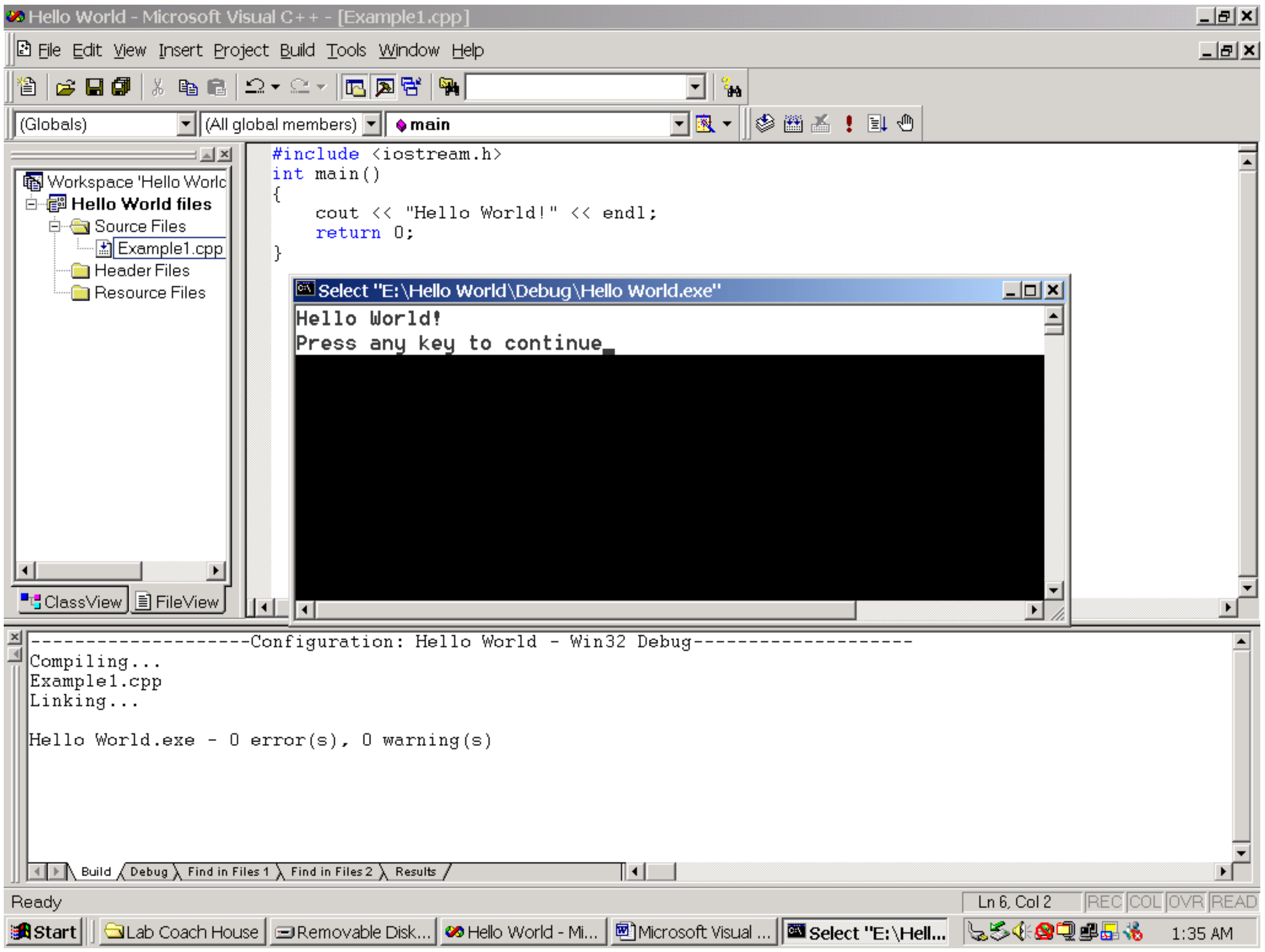


```
#include <iostream.h>
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

```
y to continue_
```

```
--------------------Configuration: Hello World - Win32 Debug--------------------
Compiling...
Example1.cpp
Linking...

Hello World.exe - 0 error(s), 0 warning(s)
```

Step 9b: The selected part of the screen contains the entire output as shown below with white background, which can be pasted to any text editor such as MS notepad or Word for printing. Note that you may also select part of the output for printing by selecting **Mark** then select the part of the output by dragging the pointer across it.



```
#include <iostream.h>
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

```
Hello World!
Press any key to continue
```

```
--------------------Configuration: Hello World - Win32 Debug--------------------
Compiling...
Example1.cpp
Linking...

Hello World.exe - 0 error(s), 0 warning(s)
```

Some Observations of the *Hello World* Example

1. The project name you have entered in Step 2 causes the system to create a folder of the same name. Although it is possible to store your files in different drives and folders, it is strongly advisable that you centrally store all your files in this folder for ease of access. If such a folder is already there, the system will use the existing one for the project.
2. An empty single source file named *Example1.cpp* was first added to the **Source Files** folder and its code subsequently entered. In a multiple-file project, the above process may be repeated to add any number of additional source files to the **Source Files** folder. Similarly, any number of user-defined header files (*.h* file extension) may be added to the **Header Files** folder. Note that both the **Source Files** and **Header files** folders are sub-directories or sub-folders in the project folder. Also note that it is not necessary to use the editor provided by Visual C++ to enter code into files; any ASCII editors will do.
3. It is possible to store physically all the *.cpp* and *.h* files in different folders and drives for multiple-file example. These files may have been created earlier and are reused in other project! However, all the files that are to be used by a project or program must be "added" or associated to the project. In our second example, we assume all files (two source files *Example2.cpp* and *Time.cpp,* and a *Time.h* header file) have already been created and stored in a folder named Time on the D-drive.
4. After you have successfully executed the program, the project folder contains more than 10 files, including many intermediate files created by the system to "build" your project. These files occupy more than 1 MB of the storage space even for the simply Hello World program with a few line of source code! For larger projects, a single floppy disk may not capacity to store the entire project.

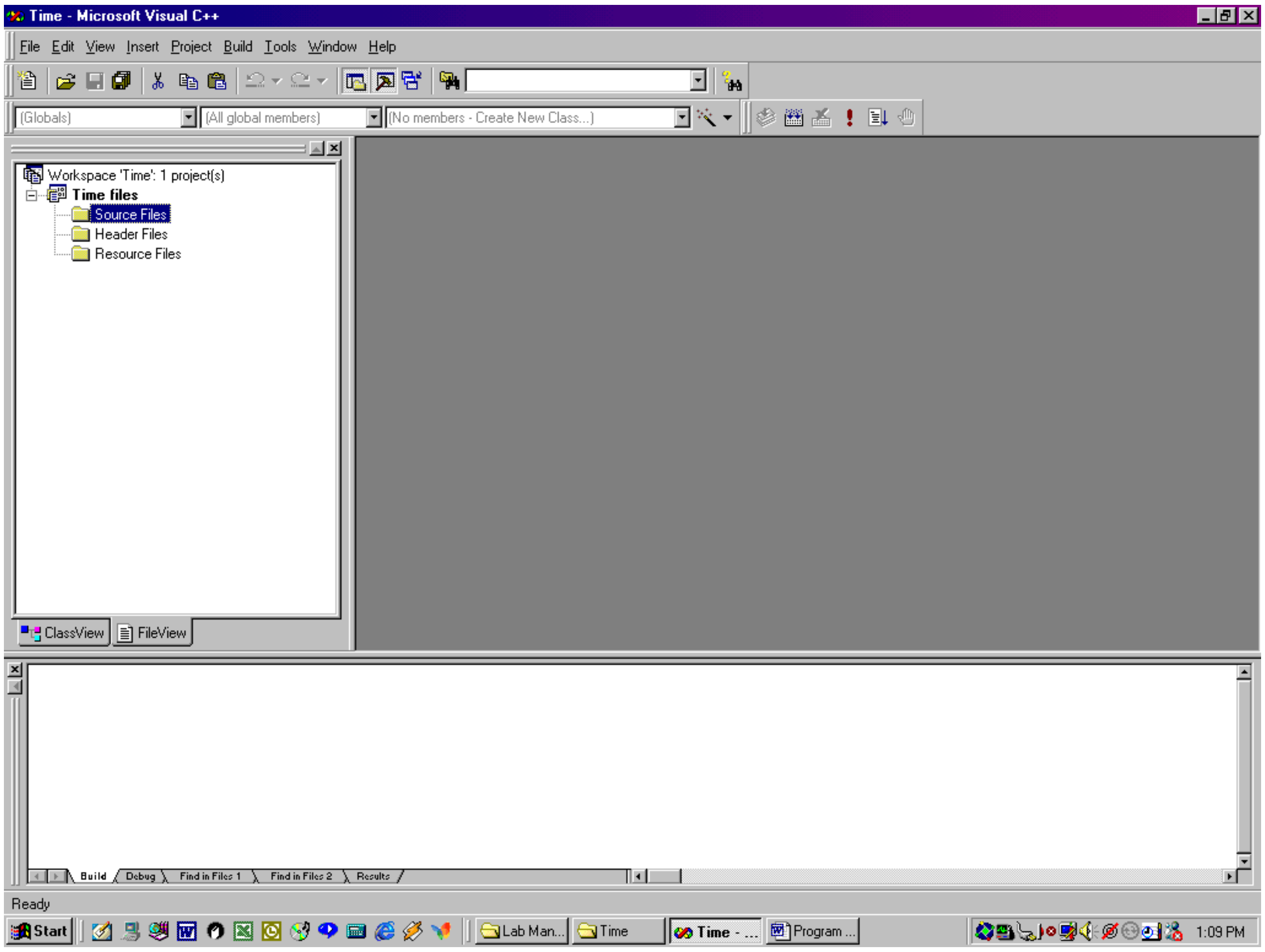
A Multiple-File Project: The *Time* Program

In this demonstration, the project or program that prints a given time of the day in both military as well as standard time formats. the project contains three files created by the programmer: two source files (Example2.cpp and Time.cpp) and a header file (Time.h). We assume that these files have already been created (you may use any ASCII editor) and stored in a project folder named Time on the D-drive. Based on Observation 2 above, we will name our multiple-file project Time and use this same Time folder to store all the files.

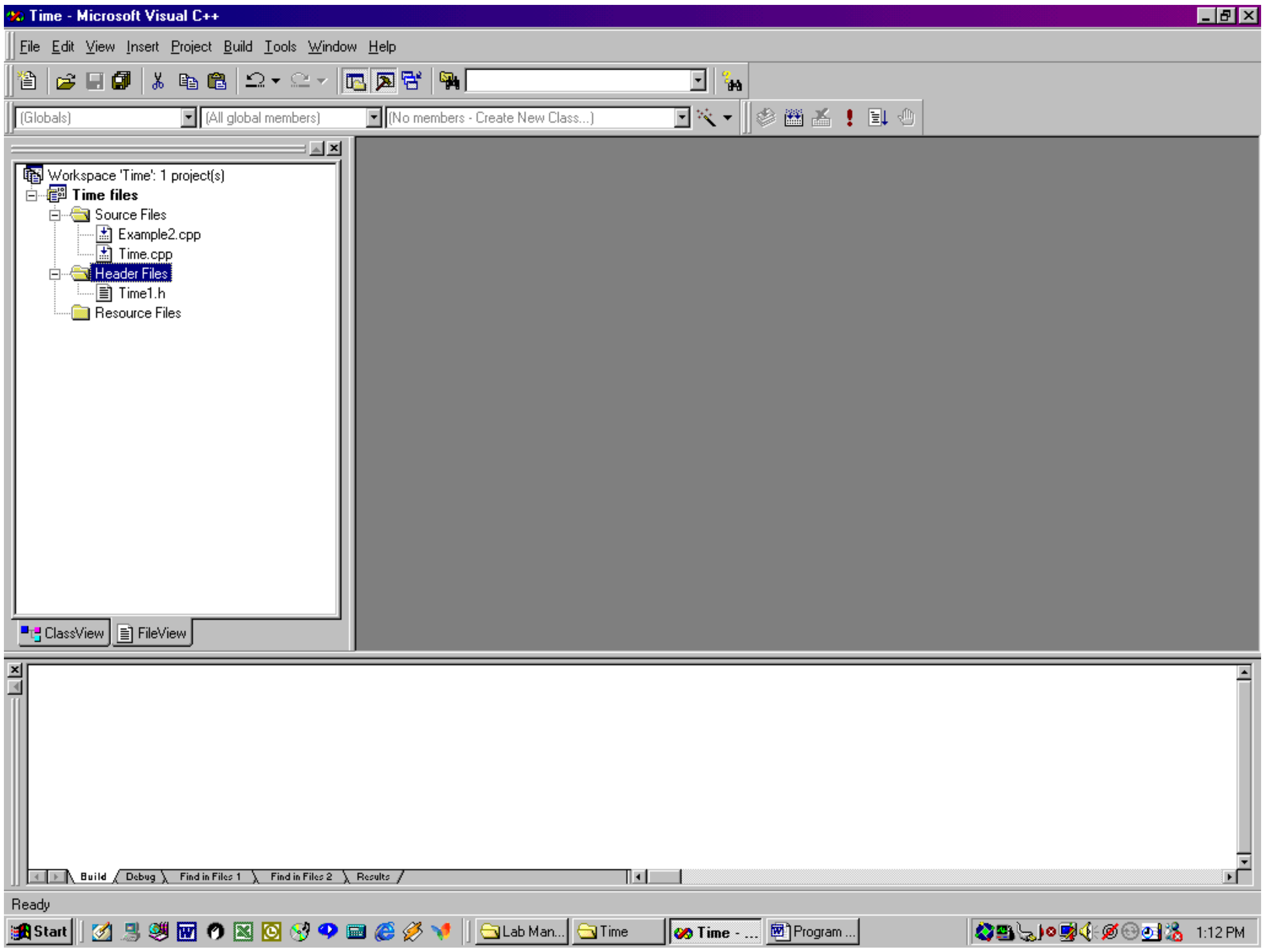
Step 1: Same as previous example: launch MS Visual C/C++ 6.0 from task bar.

Step 2: Same as previous example: click on **File** in the menu bar then click on **New…** Click on the **Projects** tab then select **Win32 Console Application**. In the Location box, select the disk drive (the D drive in this case) and in the Project box, enter project name (Time in this case). The Time folder is already there will be used by the system to store all the files for the Time project. Although all three files – Example2.cpp, Time.cpp, and Time.h – are already in the Time folder, the system does not recognize them as files that should be included in the project, as reflected by the fact that both the *Source Files* and the *Header Files* folders are empty. We must "add" these files to appropriate folder to associate them with the Time project!

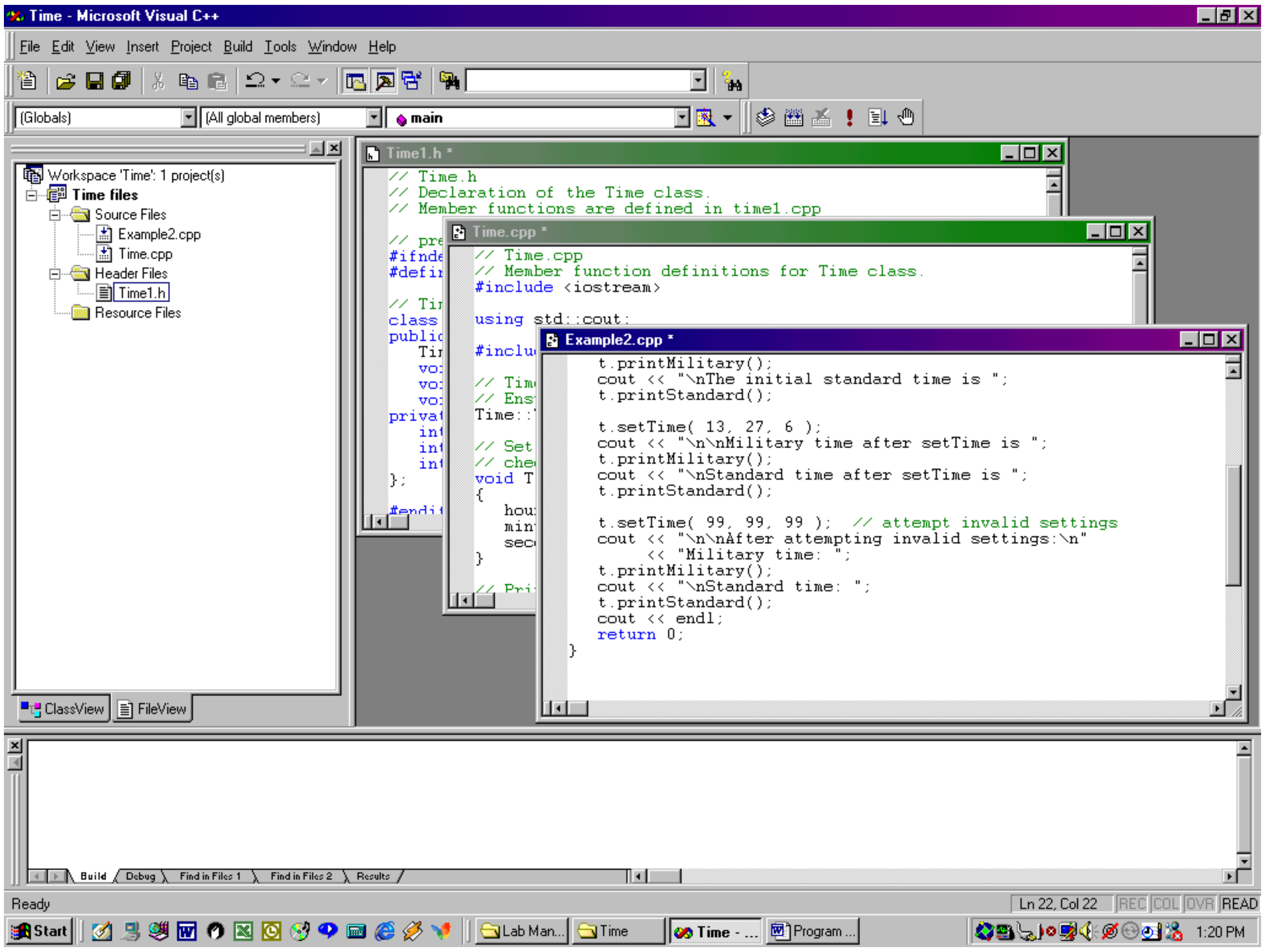
Go through Steps 3, 4, and 5 as you did in the previous example.



Step 6: Right click on the Source Files folder and add Time.cpp and Example2.cpp source files to the folder. Add Time.h to the Header Files folder. Expand the *Source Files* and *Header Files* folders to display their contents as shown below.



Step 7: Double clicking on the file name of each of these files opens the file in the edit windows. Note that three files are arranged in cascade. You may want to tile them vertically or horizontally with the **Window** menu in the menu bar.

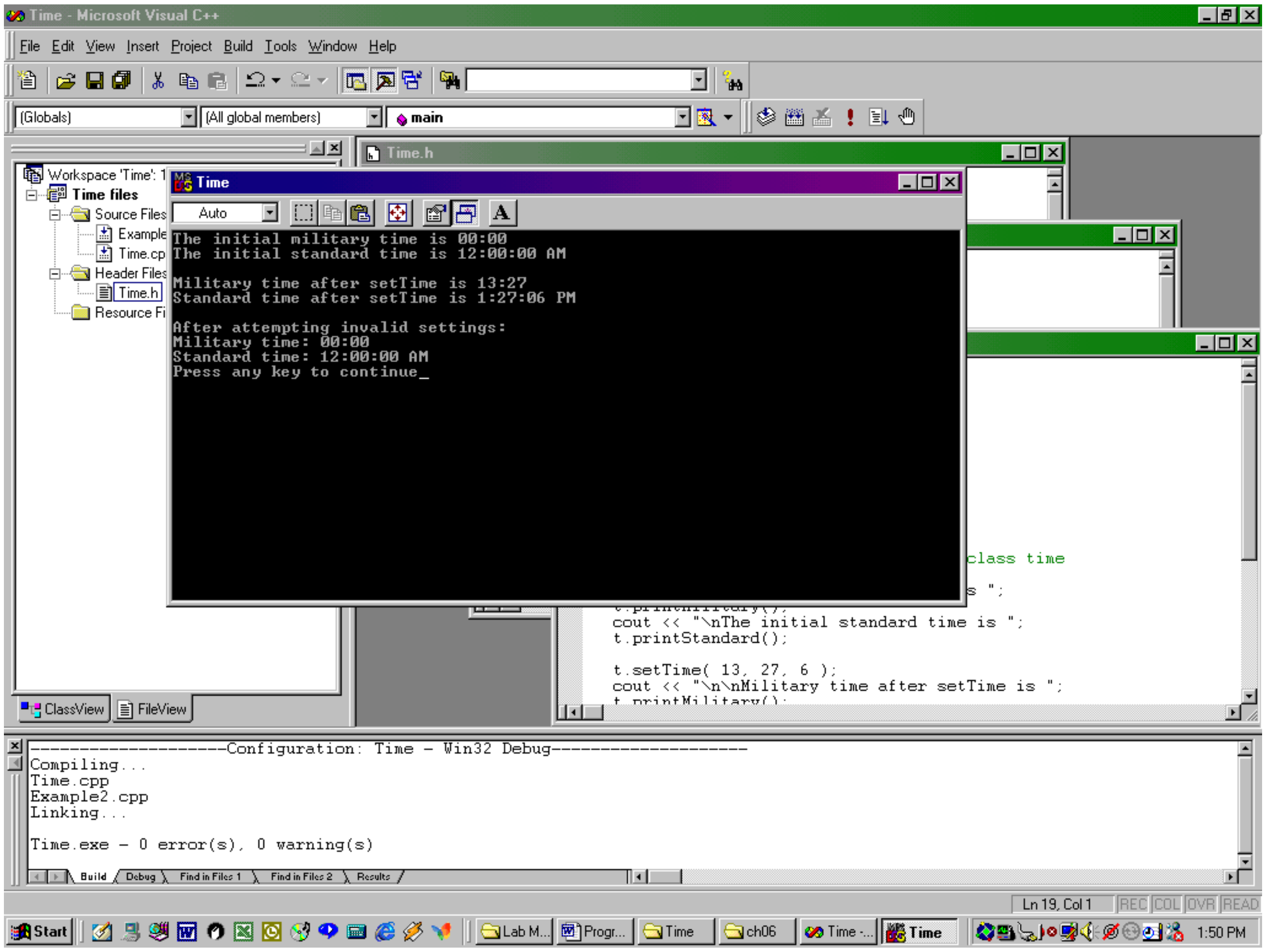


```
// Time.h
// Declaration of the Time class.
// Member functions are defined in time1.cpp
```

```
// Time.cpp
// Member function definitions for Time class.
#include <iostream>

using std::cout;
```

```
    t.printMilitary();
    cout << "\nThe initial standard time is ";
    t.printStandard();

    t.setTime( 13, 27, 6 );
    cout << "\n\nMilitary time after setTime is ";
    t.printMilitary();
    cout << "\nStandard time after setTime is ";
    t.printStandard();

    t.setTime( 99, 99, 99 );  // attempt invalid settings
    cout << "\n\nAfter attempting invalid settings:\n"
         << "Military time: ";
    t.printMilitary();
    cout << "\nStandard time: ";
    t.printStandard();
    cout << endl;
    return 0;
}
```

Step 8: We are now ready to compile, link, and execute the project. You may do each of these phases separately as we did in Example 1 or you can consolidate all three phases into one step by clicking on the Execute Program icon (an exclamation mark) in the tool bar. The system displays a dialog box "The file does not exist (Time.exe), Do you want to build it?", click Yes. If there is no error, as is the case here, the system displays the output result in a Time (name of the project) window as shown below.



```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
Press any key to continue_
```

```
class time
s ";
cout << "\nThe initial standard time is ";
t.printStandard();

t.setTime( 13, 27, 6 );
cout << "\n\nMilitary time after setTime is ";
```

```
--------------------Configuration: Time - Win32 Debug--------------------
Compiling...
Time.cpp
Example2.cpp
Linking...

Time.exe - 0 error(s), 0 warning(s)
```

Step 9: To print the source code and the output by following the same procedure as described in Step 11 of the previous example.

Appendix: The Source Code

Single-file Demo: The Hello World Program

File name: Hello World.cpp

```
#include <iostream.h>
int main()
{
      cout << “Hello World!” << endl;
      return 0;
}
```

Multiple-file Demo: the Time Program

File 1: Time.h

```
// Time class header file
// Member functions are defined in time.cpp
// prevent multiple inclusions of header file
#ifndef TIME1_H
#define TIME1_H

// Time abstract data type definition
class Time {
public:
   Time();                          // constructor
   void setTime( int, int, int ); // set hour, minute, second
   void printMilitary();          // print military time format
   void printStandard();          // print standard time format
private:
   int hour;     // 0 - 23
   int minute;   // 0 - 59
   int second;   // 0 - 59
};
#endif
```

File 2: Time.cpp

```
// Implementation file for Time.h
// Member function definitions for Time class.
#include <iostream>

using std::cout;

#include "time.h"

// Time constructor initializes each data member to zero.
Time::Time() { hour = minute = second = 0; }

// Set a new Time value using military time. Perform validity
// checks on the data values. Set invalid values to zero.
void Time::setTime( int h, int m, int s )
{
   hour   = ( h >= 0 && h < 24 ) ? h : 0;
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

// Print Time in military format
void Time::printMilitary()
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ":"
        << ( minute < 10 ? "0" : "" ) << minute;
}

// Print time in standard format
void Time::printStandard()
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
        << ":" << ( minute < 10 ? "0" : "" ) << minute
        << ":" << ( second < 10 ? "0" : "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

File 3: Example2.cpp (main function – the driver)

// Driver for Time class

```
#include <iostream>

using std::cout;
using std::endl;

#include "time.h"

// Driver to test class Time
int main()
{
   Time t;                        // instantiate object t of class time

   cout << "The initial military time is ";
   t.printMilitary();
   cout << "\nThe initial standard time is ";
   t.printStandard();

   t.setTime( 13, 27, 6 );
   cout << "\n\nMilitary time after setTime is ";
   t.printMilitary();
   cout << "\nStandard time after setTime is ";
   t.printStandard();

   t.setTime( 99, 99, 99 );       // attempt invalid settings
   cout << "\n\nAfter attempting invalid settings:\n"
        << "Military time: ";
   t.printMilitary();
   cout << "\nStandard time: ";
   t.printStandard();
   cout << endl;
   return 0;
}
```