William Paterson University
Department of Computer Science

Microsoft Visual C++ .NET Tutorial

Spring 2006 Release 1.0

Microsoft Visual C++ .NET Tutorial

Spring 2006 Release 1.0

I. Introduction

This tutorial teaches the basics of the Integrated Development Environment (IDE) of Microsoft Visual Studio .NET. It introduces in detailed steps how to enter, compile, link, and execute a C++ program in the IDE. Using two simple examples – a single file and a multiple-file program – the beginning students should be able to learn quickly the needed skills to carry out programming assignments in CS230, CS240, and other and other CS courses.

II. The IDE Environment

The IDE is Graphical User Interface or GUI-based command center where all the tools needed for software development are accessible through menus and tool bars. Although Microsoft .NET supports many different programming languages, we focus our discussion on C++.

III. Basic terms and definitions

1. Application: refers to application software or program that performs certain data processing task such as computing the GPAs of a list of students.
2. Console application: is character-based (none-graphical) application where textual input and output are displayed in a console window. Console window is also known as a DOS box, a window with a black background. When a console application is running, the program normally receives input from the keyboard and sends its output to the DOS box as streams of characters.
3. Solution, project, and file:
   a. Solution: In Visual Studio .NET, a “Solution” is synonymous with an application. In general, a “Solution” consists of one or more “Projects” that collectively perform a particular data processing task. For example, an accounting program or “Solution” may contain two projects; one processes account-payable and the other account-receivable.
   b. Project: a “Project” consists of one or more related files; together these files perform certain data processing tasks.
   c. File: Any named object or document stored in a folder is referred to as a file. Besides it name, a file may have an extension which identifies the type of the file. For example, files with .cpp extension denotes C++ source file, files with .h extension represent header files, and files with .exe extension are executable files. When a project is processed by the Visual C++ .NET, the system generated many intermediate files of various types for house-keeping purposes. For our purpose, but we are we will focus our attention on .cpp and .h files.

The relationship between Solution, projects, and files is shown in Figure 1.

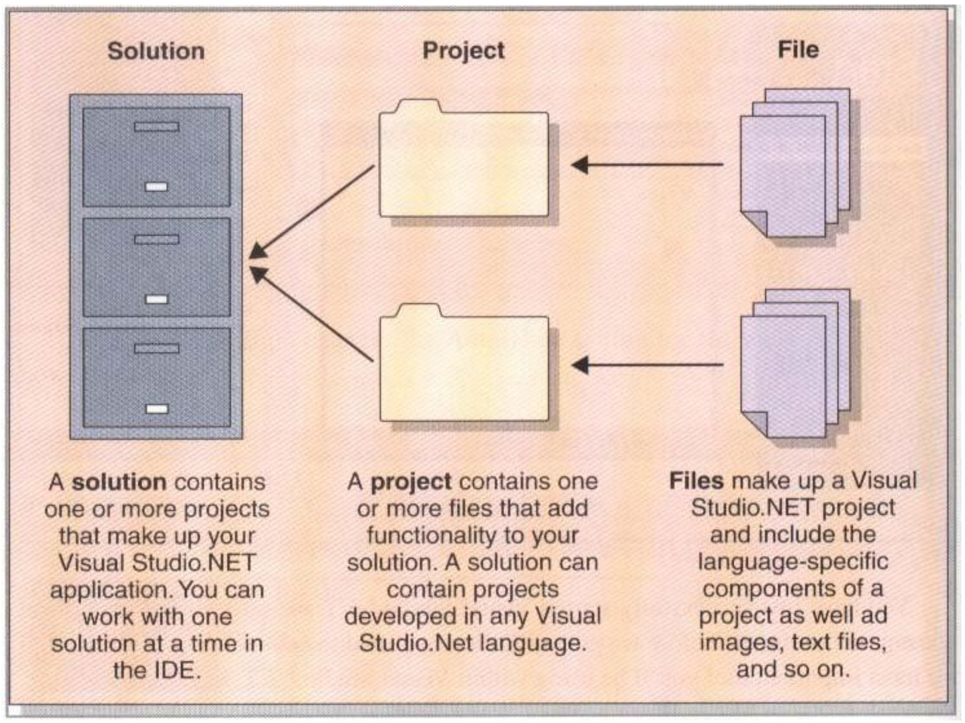

Solution
Project
File
A **solution** contains one or more projects that make up your Visual Studio.NET application. You can work with one solution at a time in the IDE.
A **project** contains one or more files that add functionality to your solution. A solution can contain projects developed in any Visual Studio.Net language.
Files make up a Visual Studio.NET project and include the language-specific components of a project as well ad images, text files, and so on.

Figure 1 – Relationships between Solution, Projects, and Files (Koneman)

IV. Notations used in the following tutorials:

1. Menu items appear in boldface. Two or more items selected or clicked in a row are separated by arrows "->", e.g., when we say click **File -> Save As…,** we mean click **File** then click **Save As…,** where **File** and **Save As…** are items in the **File** menu.
2. Words or phrases other than menu items that are displayed in windows or in dialog boxes appear in bold and italicized, e.g., ***Solution Explorer, New Project, Open Project***, etc.
3. Words entered into a dialog box by the user are italicized without being bolded, e.g., *HelloWorld* is the name entered by the user for the ***Solution*** as well as for the ***Project*** in our first tutorial.
4. Program code is shown is shown in `Courier New` font; all others are in Times New Roman font.

V. Summary of C++ program development steps: Assuming you have a new program to compile and run and the program is ready to be typed in, you need to go through the following steps. These steps may not make much sense to you at the moment but they will once you finished the tutorials. You may come back here for a review and print out this page as a quick reference.

1. Launching the Microsoft Visual Studio .NET
2. Setting the ***Profile*** for C/C++ program development
3. Creating an empty new C/C++ ***Project*** by selecting
   a. Select ***Win32*** as the ***Project Type***
   b. Select ***Win32 Console Project*** as the ***Template***

   c. Specify Disk drive and path; then enter a name for a folder in which all files associated with the project are to be stored.
4. Entering C++ source code into file or files
5. Compiling, linking, and executing the project
6. Printing the source code and the output in the console window
7. Debugging basics
   a. Syntax errors
   b. Runtime errors
   c. Logic errors

Tutorial I: Single-file project:

We use a simple program that displays a "Hello World" message on the screen as our demo. The source code will be entered into a file named *HelloWorld.cpp*.

1. <u>Launching Microsoft Visual Studio .NET</u>

From the desktop, double click on the icon of MS Visual Studio .NET. If the .NET icon is not on the desktop, click **start** (in the task bar of the Windows) **->** **Program ->** **Microsoft Visual Studio .NET 2003 -> Microsoft Visual Studio .NET 2003**. The system displays the main window of the Visual Studio .NET as shown is Figure 1.

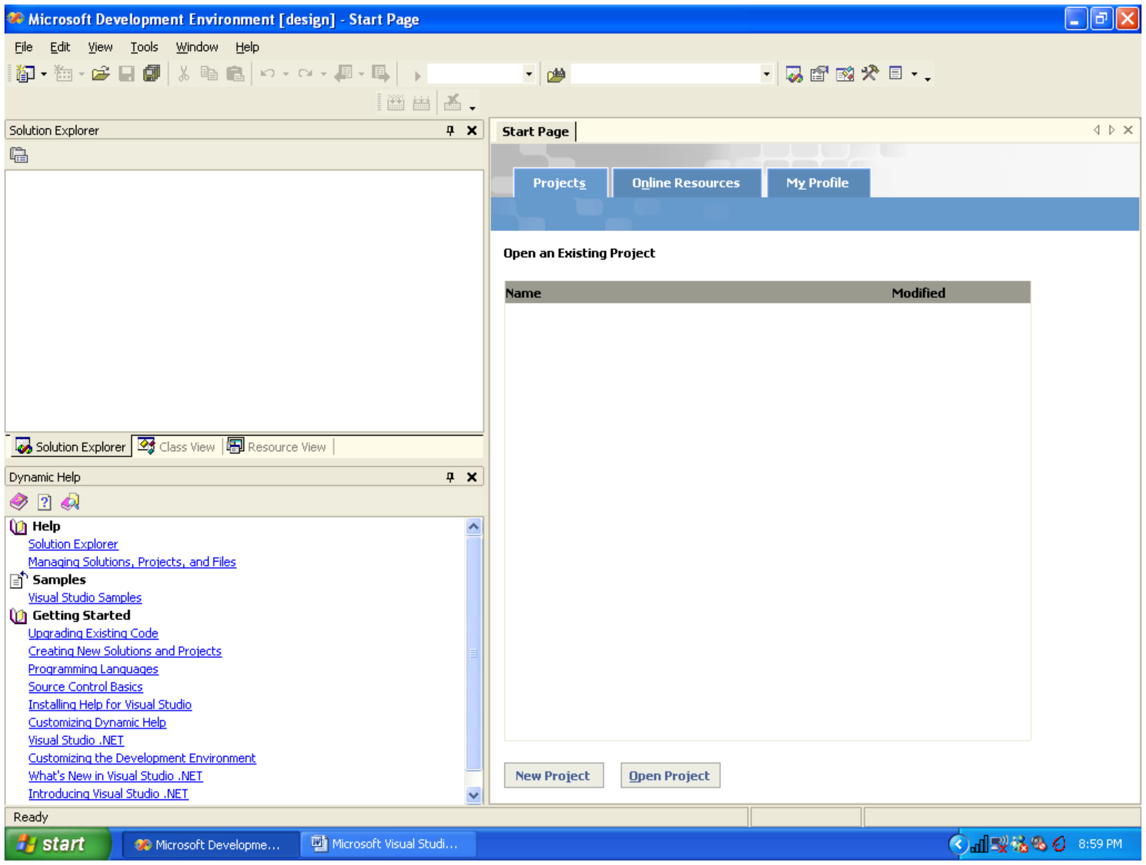


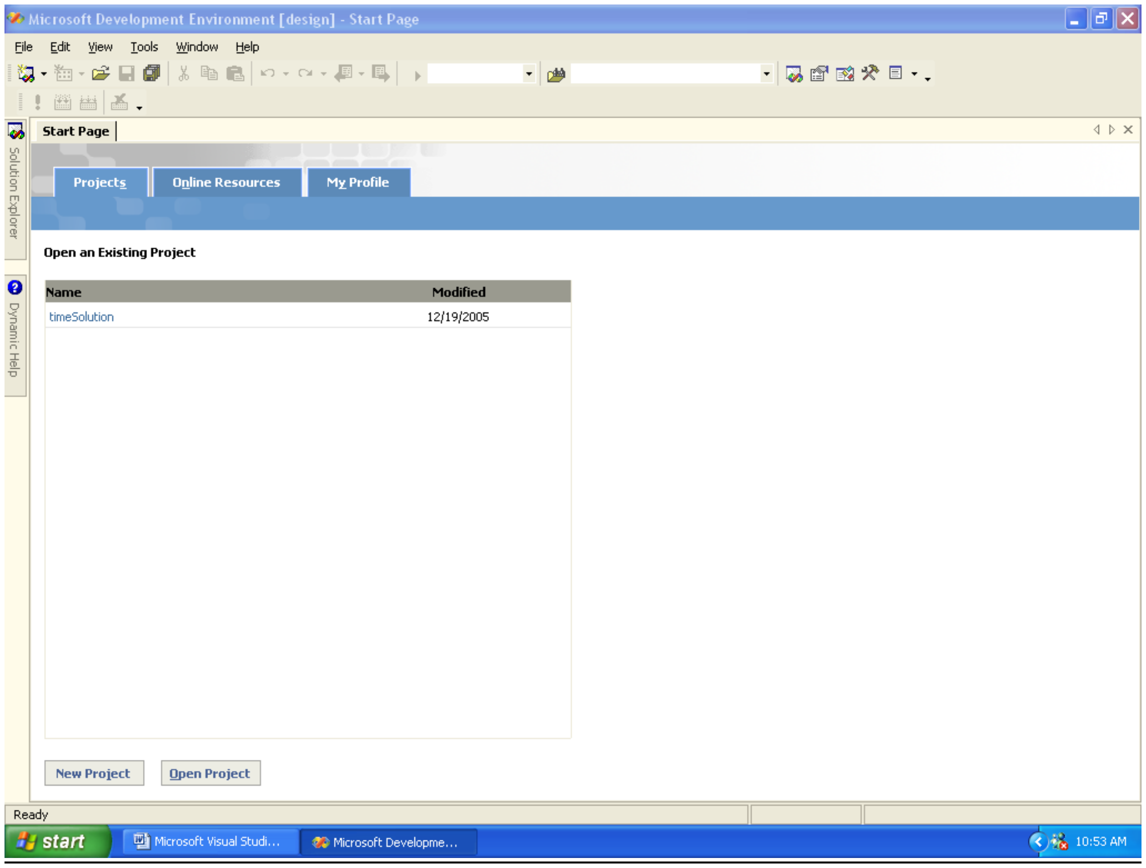
Figure 1 - The main .NET window

Note that the title bar displays ***Microsoft Design Environment [Design] – Start Page***; it will change once we enter a name for the project later. As shown, the main .NET window contains a title bar, a menu bar, a ***standard*** tool bar, and a ***build*** tool bar. New tools bars can be added to and existing ones removed from and the main window by simply right clicking anywhere in the tool bar area; from the pop-up menu, you may them select or de-select them. The ***standard*** and ***build*** tool bars used frequently. The ***debug*** tool bar, which is not shown in Figure 1, is useful to debug or locate errors in a program.

The main window may contain several smaller windows and each serve a particular purpose. As shown in Figure 1, the three smaller windows displayed are the ***Solution***

Explorer, the ***Dynamic Help***, and the ***Start Page*** windows. The ***Solution Explorer*** window allows you to locate and work with individual files of you program while ***Dynamic Help*** allows you to get online help quickly. Smaller windows may be deleted from the main window. For example, you may choose to remove the ***Dynamic help*** window by clicking the ***Close*** button at the upper right corner of this window. If this window is not on the screen, you can display it by clicking **Help -> Dynamic Help**.

When many small windows are opened, the screen may be cluttered, especially if you do not have a big monitor. Instead of closing down some windows, you may want to use the ***Auto Hide*** feature associated with most of the small windows to avoid the cluttering of the screen. Notice that there is a "push pin" icon to the left of the ***Close*** button in the title bars of the ***Solution Explorer*** and ***Dynamic Help*** windows. When the push pin points downward, the associated window is displayed as shown in Figure 1. By clicking on it, the push pin points to the left and the window shrinks to an icon or "hidden" on the left of the main window, as shown in Figure 2. By moving the cursor over it, the window is restored; by moving the cursor away from the window, the window automatically shrinks to an icon.



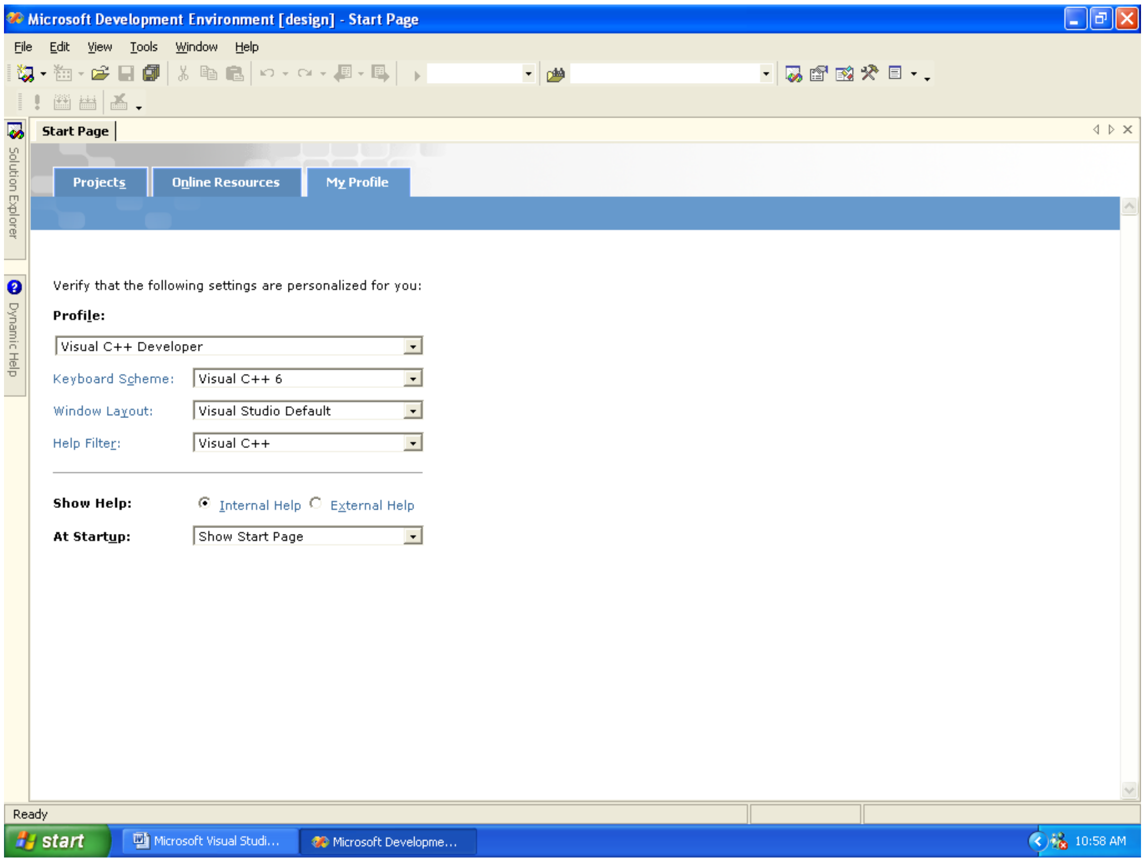
Open an Existing Project

Name	Modified
timeSolution	12/19/2005

Figure 2 - Demonstration of the ***Auto Hide*** feature where both the ***Solution Explorer*** and ***Dynamic Help*** windows are shrunk to icons.

2. Setting the ***Profile*** for C/C++ program development

Click the ***My Profile*** tab as shown in Figure 2. From top toward bottom of the drop-down lists, select ***Visual C++ Developer***; ***Visual C++ 6*** for ***Keyboard Scheme***; ***Visual C++ 6*** for ***Window Layout***; and ***Visual C++*** for ***Help Filter*** and the remaining ones as shown in Figure 3. Note that this is a one-time operation and there is no need to do it again if the settings have not been changed.



Verify that the following settings are personalized for you:

Profile:

Visual C++ Developer

Keyboard Scheme: Visual C++ 6

Window Layout: Visual Studio Default

Help Filter: Visual C++

Show Help: [✓ Internal Help] External Help

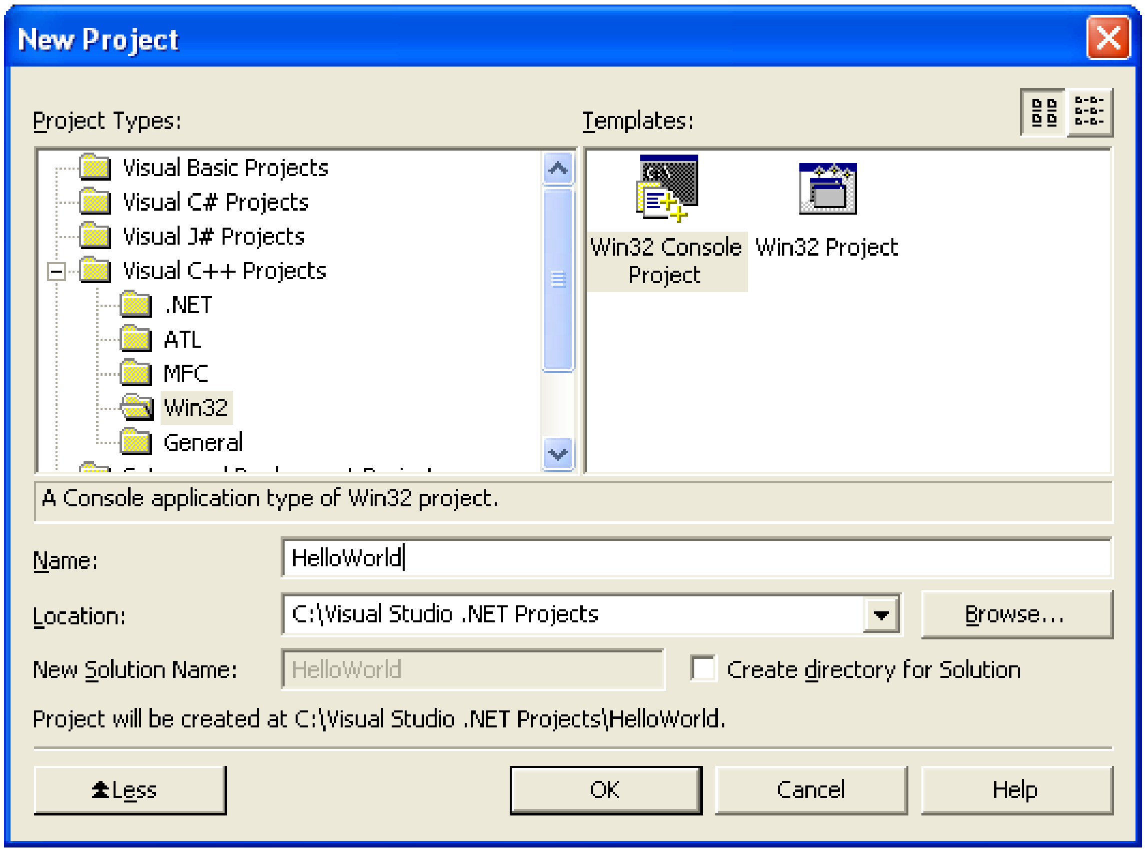
At Startup: Show Start Page

Figure 3 - Setting the ***Profile*** for C/C++ program development

3. Creating a Naming an Empty Project

Click the ***Projects*** tab in the ***Start Page*** window and then click the ***New Project*** button at the lower left of the window, the ***New Project*** dialog box as shown in Figure 4 is displayed. Click the ***More*** button at the lower left corner to make sure the dialog box is fully displayed. A dialog box is by itself a small window; it is used by the system to prompt user to enter information. In the ***Project Type*** pane on the left, open the ***Visual C++ Projects*** folder by clicking the "+" symbol to its left and select and ***Win32*** as ***Project Type.*** In the ***Templates*** pane on the left, select ***Win32 Console Project*** as the template. Remember that, as discussed in the introduction section, a console project is a project whose input and output are streams of characters and are displayed in the "DOS" box.

Enter *Hello World* as the name of project in the ***Name*** box and specify where you would like to have the project stored by directly entering drive label and path name in the ***Location*** box. Alternatively, you may use the ***Browse*** button to do the same. In this demo, we use the ***Browse*** button and select *C:\Visual Studio .NET Projects* as the place to store our project, where *C:* is the drive label and *Visual Studio .NET Projects* is the path name. A path name is actually a folder in which files are stored. The folder may have been previously created; if it was not, the system will be created automatically for the project. Note that you may choose any drive and name a folder any way you like. Also notice that we have left ***Create directory for Solution*** unchecked, meaning the **Solution** contains only a single **Project** (defined in the introduction section), and therefore, share the same folder. For very large software where the **Solution** may contain multiple **Projects**, you may want to check the box and enter a name for the **Solution**.



A Console application type of Win32 project.

Name: HelloWorld

Location: C:\Visual Studio .NET Projects

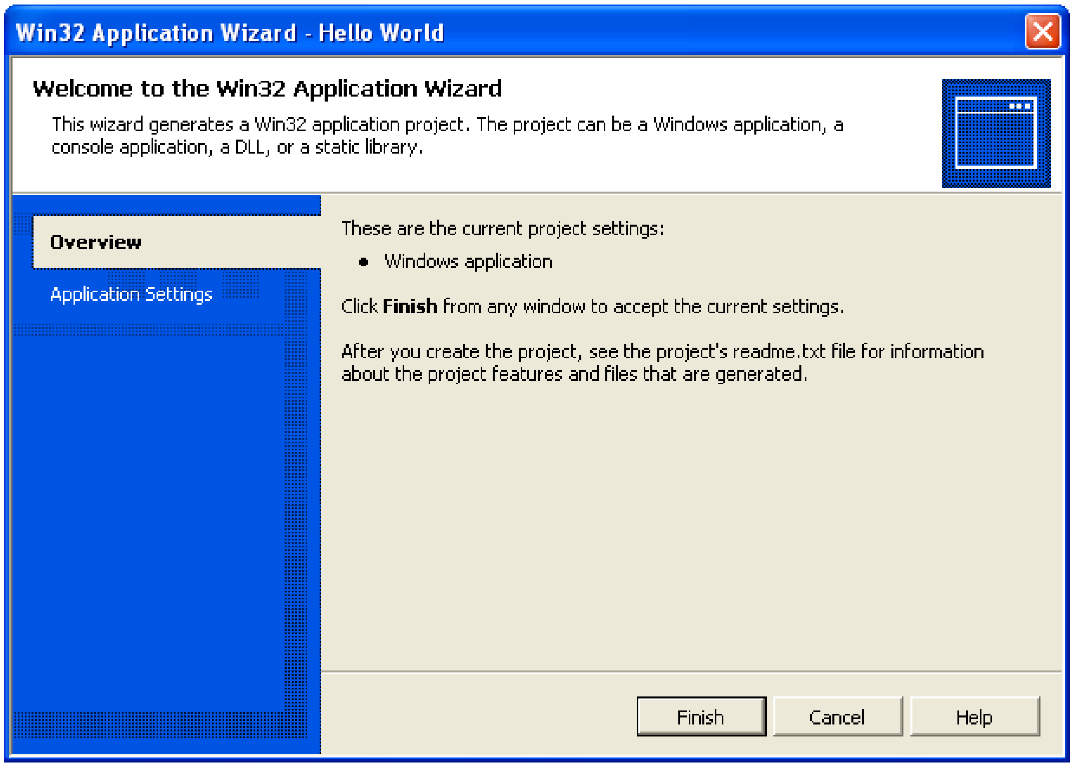
New Solution Name: HelloWorld

☐ Create directory for Solution

Project will be created at C:\Visual Studio .NET Projects\HelloWorld.

Figure 4 - New Project dialog box.

Click the ***OK*** button, the .NET display the following ***Win32 Application Wizard –Hello World*** window as shown is Figure 5.



Welcome to the Win32 Application Wizard

This wizard generates a Win32 application project. The project can be a Windows application, a console application, a DLL, or a static library.

These are the current project settings:

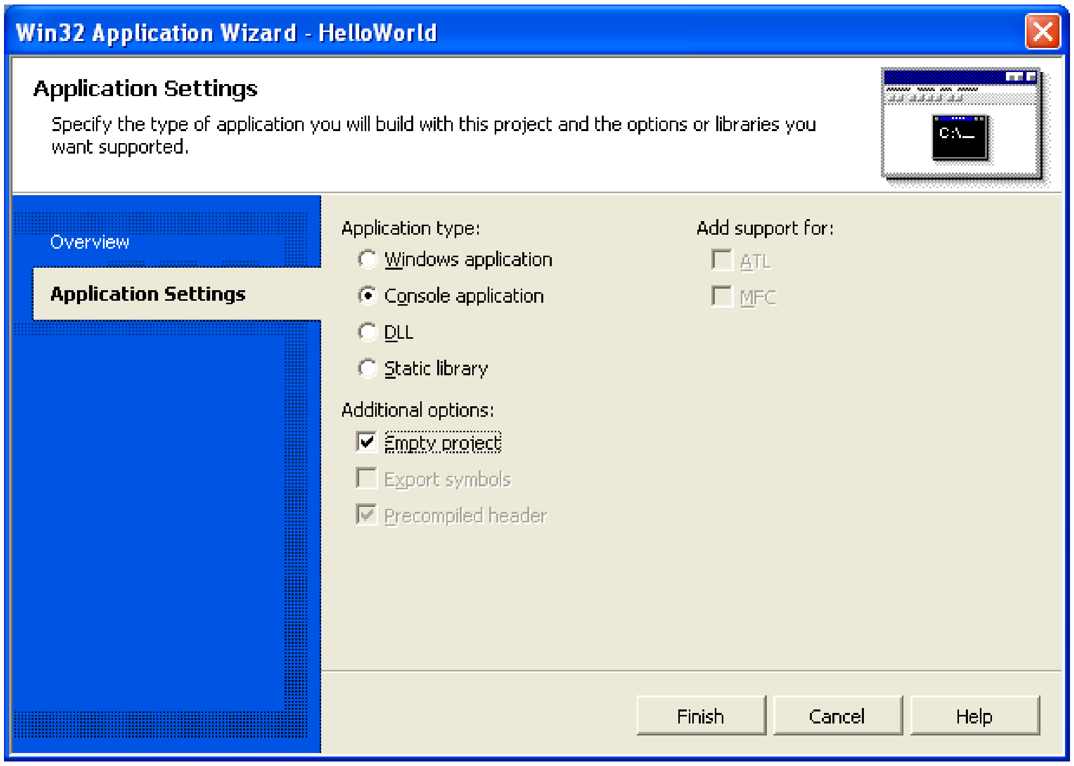
- Windows application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

Figure 5 - Win32 Application Wizard window - Overview

Click the Application Settings tab to display the window shown in Figure 6 and make sure that the ***Empty project*** box is checked!!!



Application Settings

Specify the type of application you will build with this project and the options or libraries you want supported.

Application type:

- ( ) Windows application
- (●) Console application
- ( ) DLL
- ( ) Static library

Additional options:

- [✓] Empty project
- [ ] Export symbols
- [✓] Precompiled header

Add support for:

- [ ] ATL
- [ ] MFC

Figure 6 - Win32 Application Wizard window – Application Settings.

Click the ***Finish*** button to display the main .NET window shown in Figure 7. Notice that the title bar of the main window is changed from ***Microsoft Design Environment [Design] – Start Page*** to ***HelloWorld – Microsoft Visual C++ [Design] – Start Page***. Also note that we have clicked the push pin icon in the title bar of the ***Solution Explorer*** window to turn off the ***Auto Hide*** feature so that the window is fully displayed. In the ***Solution Explorer*** window, there are 4 folders, the main *HelloWord* and three sub-folders: the ***Source Files***, ***Header Files***, and ***Resource Files*** folders which are empty at the moment. For single-file project, source code is always stored in the ***Source Files*** folder as a file with a .cpp extension. For multiple-file project, all files with .cpp extension are stored in ***Source Files*** folder and all programmer-defined header files are stored in the ***Header Files*** folder; ***Resource Files*** folder is not used for console applications.

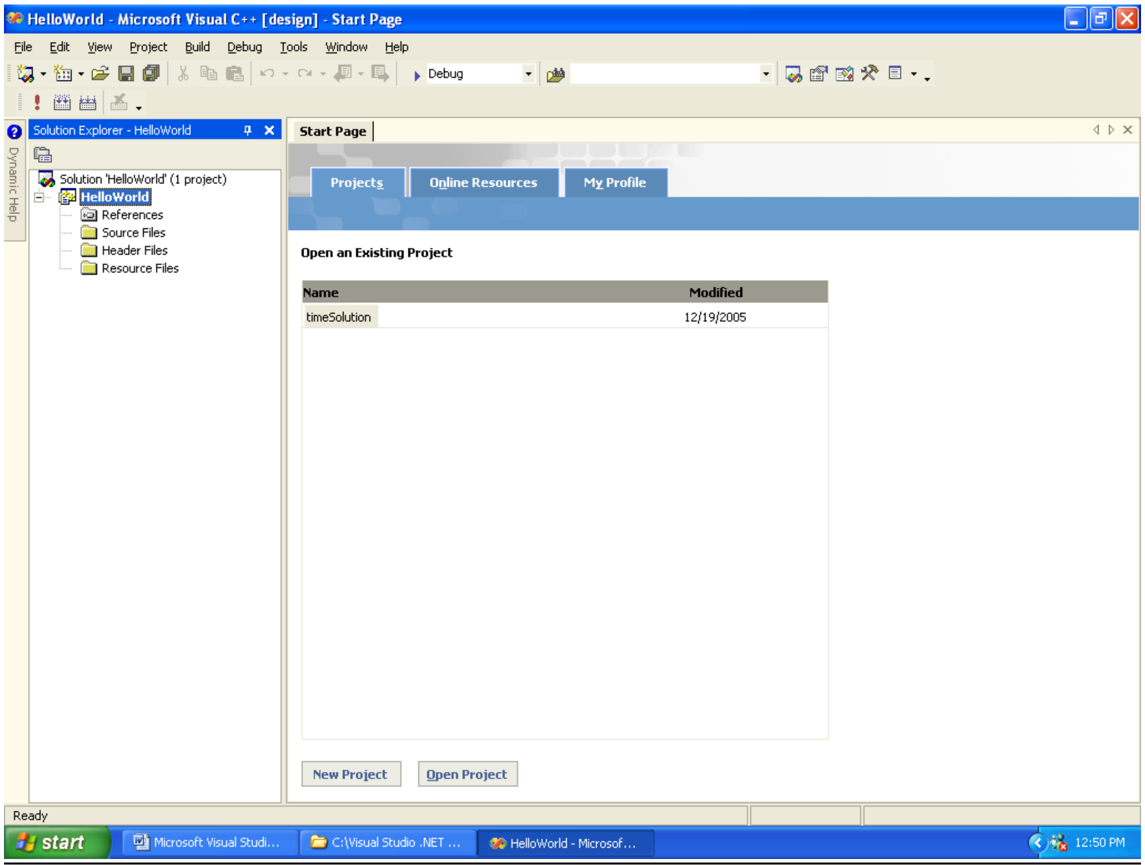


Figure 7 - Four folders have been created in the ***Solution Explorer*** and window

4. <u>Adding source file to the project and entering C++ source code for the file</u>

Right click anywhere on the ***Source Files*** folder, a pop-up menu is displayed as shown in Figure – 8 below.

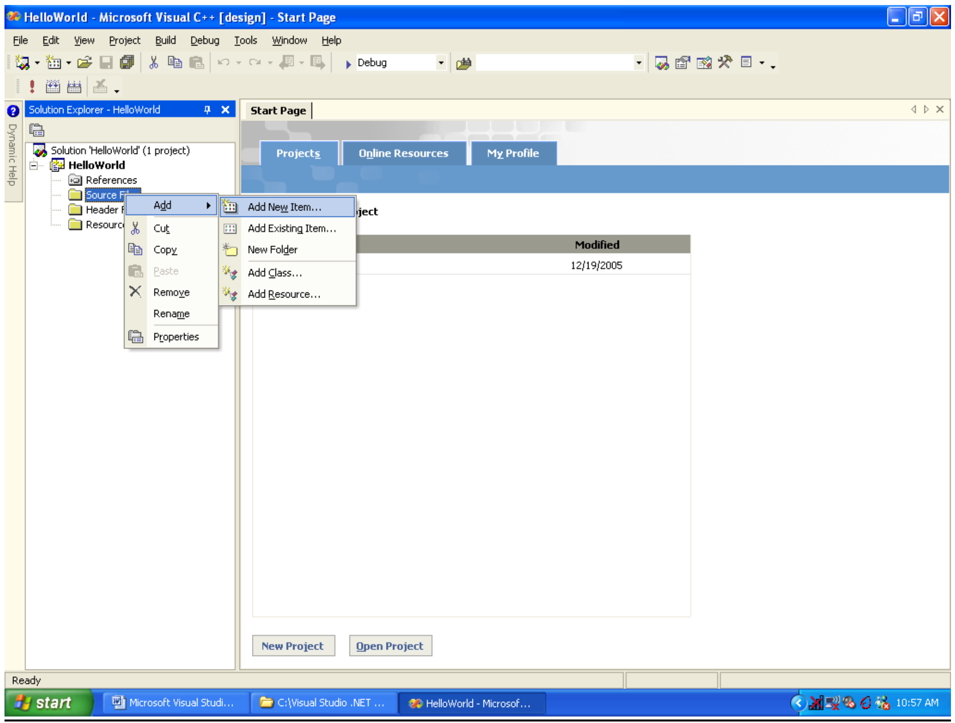
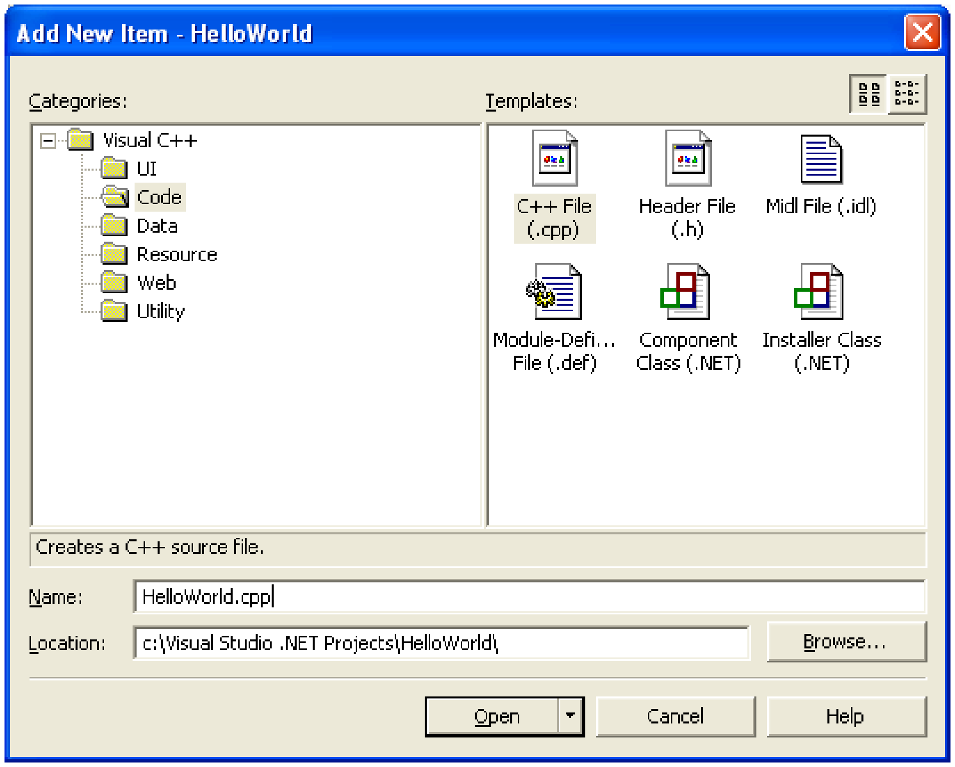


Figure 8 – Pop-up menu upon right clicking anywhere on ***Source Files***

Select ***Add*** in the pop-up menu and click ***Add New Item…***, the system displays the ***Add New Item – HelloWorld*** dialog box as shown in Figure 9 below. Select the ***Code*** folder in the ***Categories*** pane on the left and ***C++ File(.cpp)*** in the ***Templates*** pane on the right, then enter *HelloWorld.cpp* in the name box.



Creates a C++ source file.

Name: HelloWorld.cpp

Location: c:\Visual Studio .NET Projects\HelloWorld\

Figure 9 – The ***Add New Item*** dialog box

Accept the default ***location*** *c:\Visual Studio .NET Projects\HelloWorld\* and click the ***Open*** button. The new *HelloWorld.cpp* file has now been added to the ***Code*** folder in the ***Solution Explorer*** folder and the empty file edit area appear on the right with ***HelloWorld.cpp*** shown in the tab. We are now ready to enter the C++ source code to the blank area.

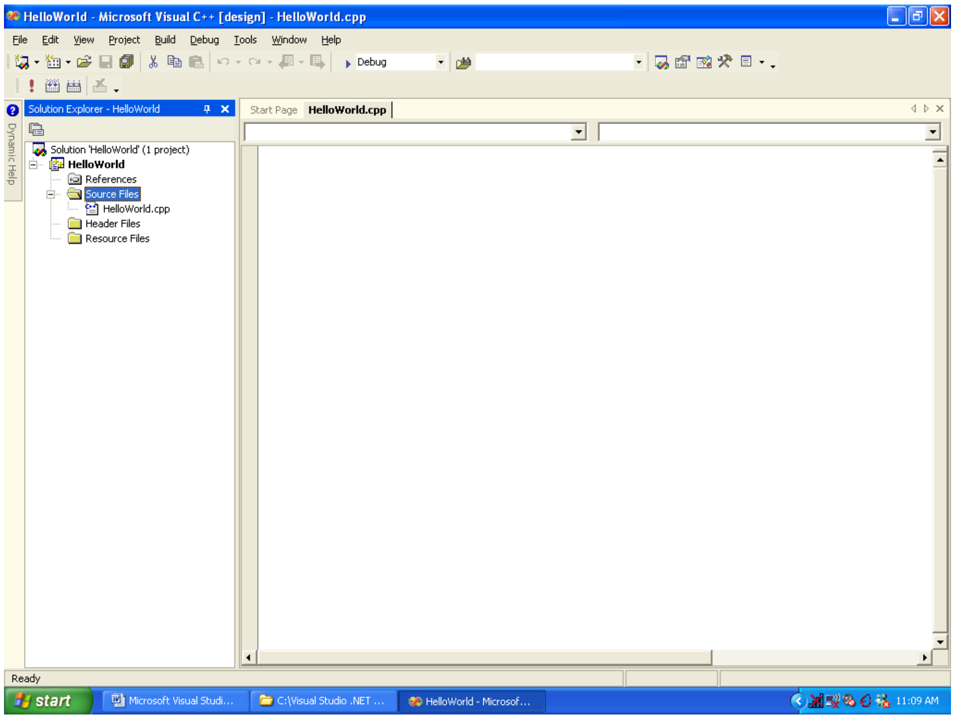
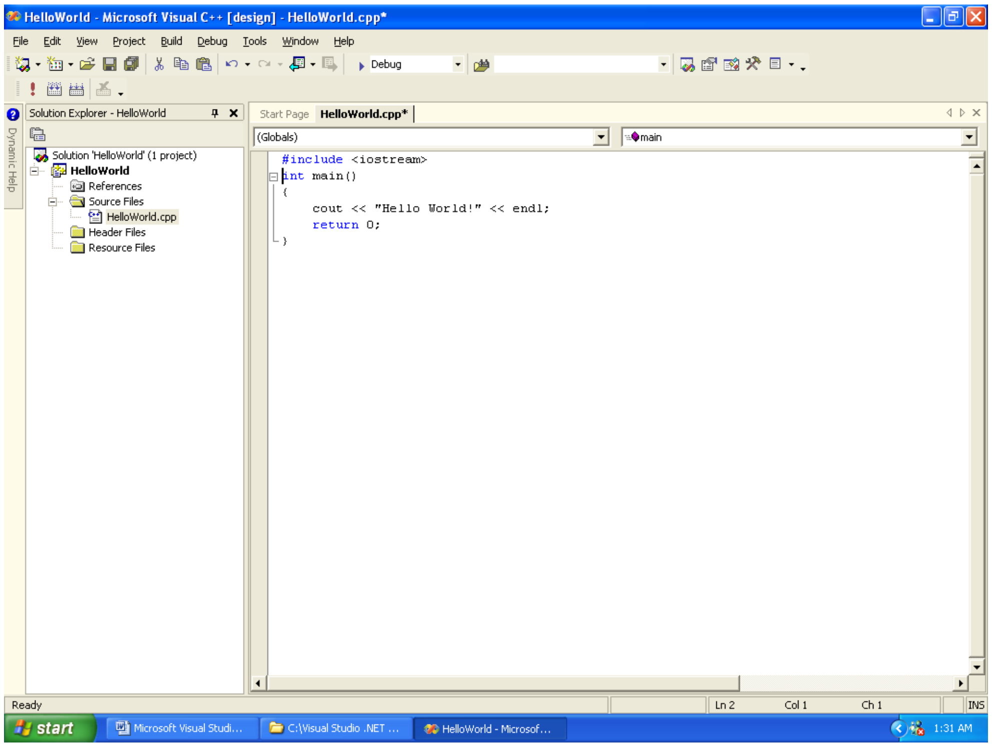

HelloWorld - Microsoft Visual C++ [design] - HelloWorld.cpp
File Edit View Project Build Debug Tools Window Help
Debug
Solution Explorer - HelloWorld
Solution 'HelloWorld' (1 project)
HelloWorld
References
Source Files
HelloWorld.cpp
Header Files
Resource Files
Start Page
HelloWorld.cpp
Dynamic Help
Ready
start
Microsoft Visual Studi...
C:\Visual Studio .NET ...
HelloWorld - Microsof...
11:09 AM

Figure 10 – The *HelloWorld.cpp* had been added to the ***Source Files*** folder

Enter source code as shown in Figure 11 below.

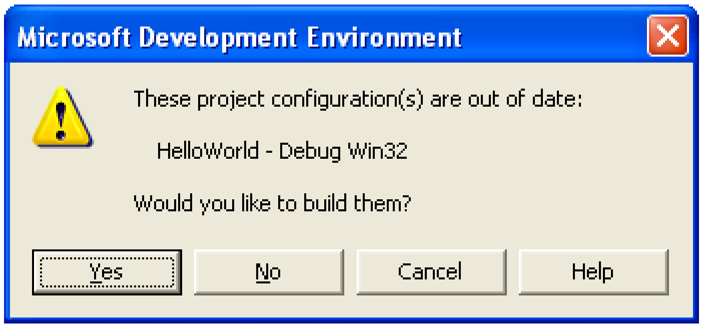


```
#include <iostream>
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Figure 11 – Source code has been entered in the code editing area

5. <u>Compiling, linking, and executing the project</u>

Compiling, linking, and executing are normally performed in three separate steps through menu manipulations. There is a simple way to do the three steps all at once by clicking the ***Start Without Debugging*** icon, a purple exclamation symbol in the ***Build*** tool bar. The **Build** tool bar is shown in both Figures 10 and 11; if it is not displayed, simply right click anywhere in the tool bar area to display the pop-up menu and then click on ***Build*** to select it. We now click on the purple exclamation icon, the system displays following ***Microsoft Development Environment*** dialog box:



Microsoft Development Environment

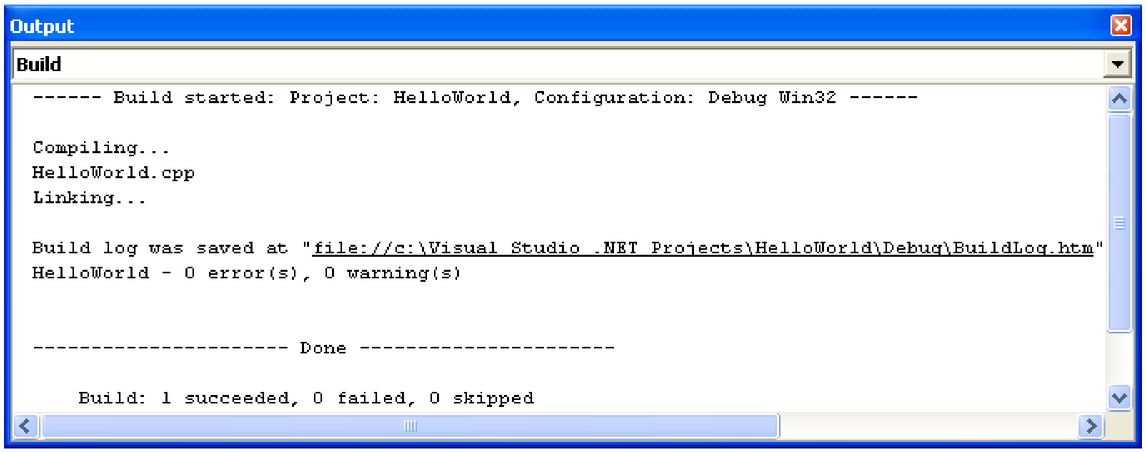
These project configuration(s) are out of date:

HelloWorld - Debug Win32

Would you like to build them?

Figure 12 - ***Microsoft Development Environment*** dialog box

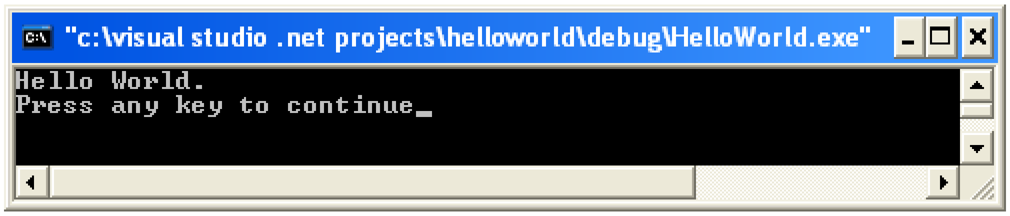
Click ***Yes*** button, the system goes through compile and link steps and produces a summary report about the build process in the ***Output*** window shown in Figure 13 below. As the message indicates, the build process is successful since both compiling and linking are without errors.



```
------ Build started: Project: HelloWorld, Configuration: Debug Win32 ------

Compiling...
HelloWorld.cpp
Linking...

Build log was saved at "file://c:\Visual Studio .NET Projects\HelloWorld\Debug\BuildLog.htm"
HelloWorld - 0 error(s), 0 warning(s)


---------------------- Done ----------------------

    Build: 1 succeeded, 0 failed, 0 skipped
```

Figure 13 – The ***Output*** window showing a summary report of the Build process

Since there is no error in compiling and linking, the project is automatically executed and results produced and displayed in the console or DOS window as shown in Figure 14 below. Note that the console window displays results or outputs from the execution of your program *HelloWorld.exe* whereas the Output window of Figure 13 displays the results or outputs of the compilation and linking process.


"c:\visual studio .net projects\helloworld\debug\HelloWorld.exe"

```
Hello World.
Press any key to continue_
```

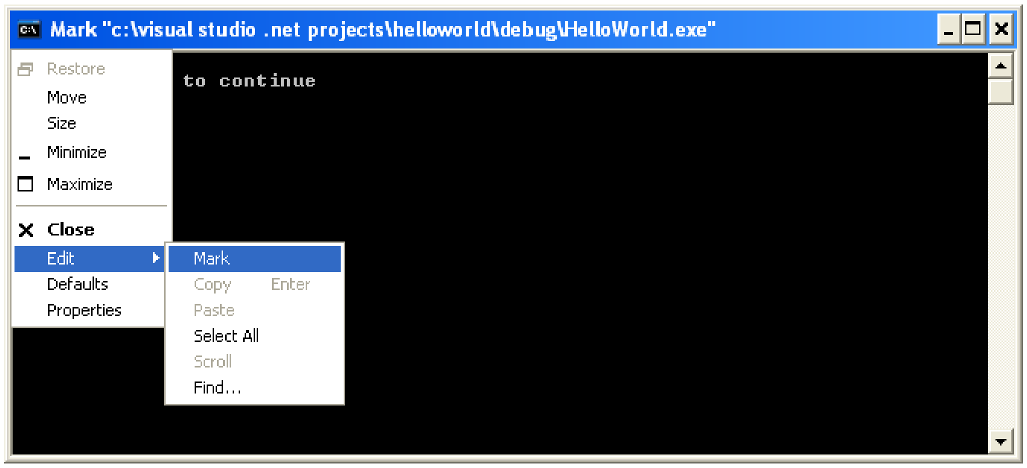
Figure 14 – Output from the execution of the *HelloWorld.cpp* program

5. Printing the source code and the output in the console window

To print the source code in the project editing window, simply click anywhere on the window then click on **File** -> **Print**… -> **OK**.

To print the output result in the console window involves the following steps:

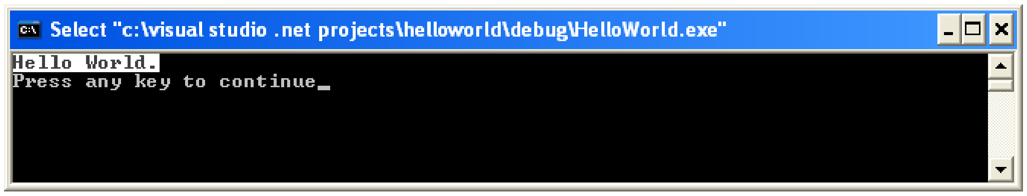
a. Click on the drive icon (C:\ in this demo) on the left end of the title bar of the console window to display the pop-up menu then click on **Edit** -> **Mark** as shown in Figure 15. A rectangular shaped marker is displayed.



```
to continue
```

Figure 15 – Display the marker in the console window.

b. Drag the marker to select the output as a block of text (“Hello World.” in this demo) as shown in Figure 16.


Select "c:\visual studio .net projects\helloworld\debug\HelloWorld.exe"

```
Hello World.
Press any key to continue_
```

c. Click on the drive icon again to display the pop-up menu. In the menu, click **Edit** -> **Copy** to copy the selected block of text to the clipboard which it can be paste to any word processor such as Microsoft Word where it can be printed.

Tutorial II: Multi-file project:

<u>Multi-file or separate file project</u>: In the previous demo, the project consists of a single *HelloWorld.cpp* file. In practice, a C++ programming project may contain many .cpp and .h files. It is desirable to have these files separately compiled and debugged before assembling them into a project. In this tutorial, we demonstrate how separate .cpp and .h can be assembled, compiled, linked, and executed together. Although not as detailed as the previous tutorial, we decide to show these steps not to avoid confusion and the need to refer back frequently.

We use a *time* project for the demonstration. The *time* project displays time of the day in both standard (0 – 12 AM or PM) and military (0 – 23 without AM or PM) formats. The project contains three separate files: *time.h*, *time.cpp*, and *timeMain.cpp*. The header file *time.h* contains the declaration or interface of the time class; time.cpp contains code that implements all the member functions of the *time* class; and the *timeMain.cpp* is the driver function that uses the *time* class to display time in both military and standard formats. The detailed code is listed in the appendix I.

1. <u>Launching Microsoft Visual Studio .NET</u>:

   From the desktop, double click on the icon of MS Visual Studio .NET. If the .NET icon is not on the desktop, click **start** (in the task bar of the Windows) **-> Program -> Microsoft Visual Studio .NET 2003 -> Microsoft Visual Studio .NET 2003**. The system displays the main window of the Visual Studio .NET as shown is Figure 1.

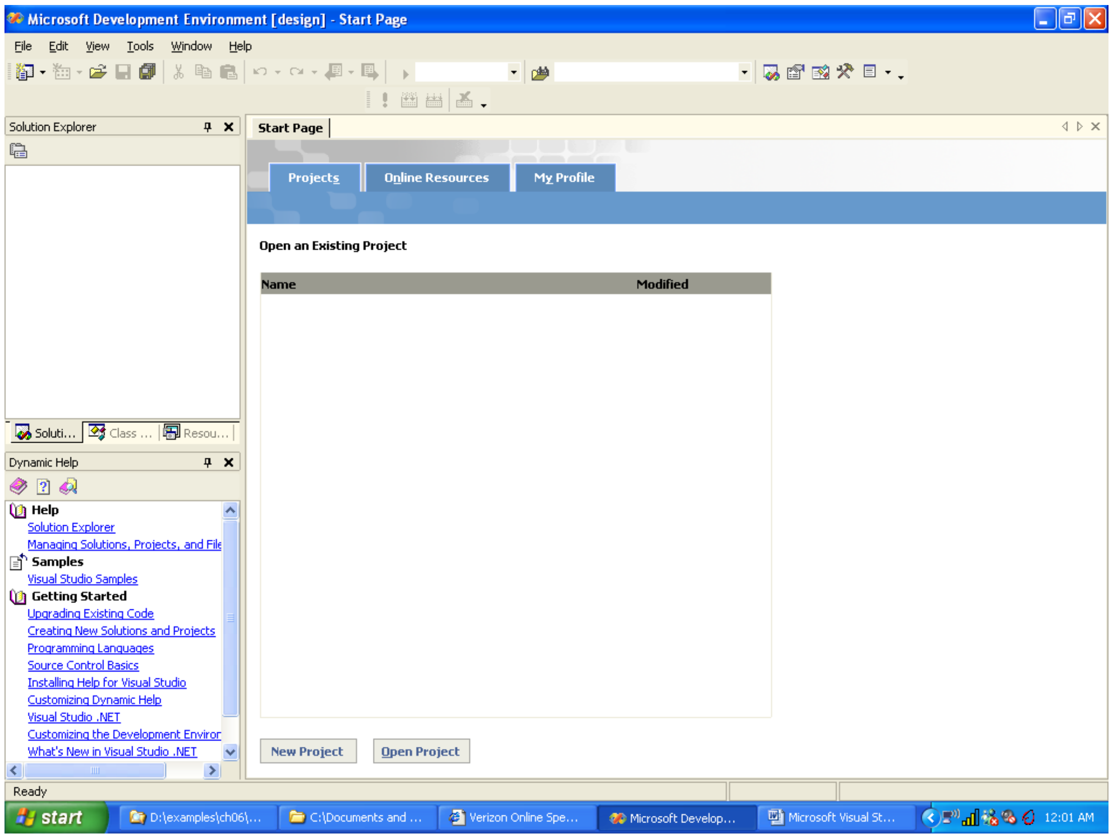
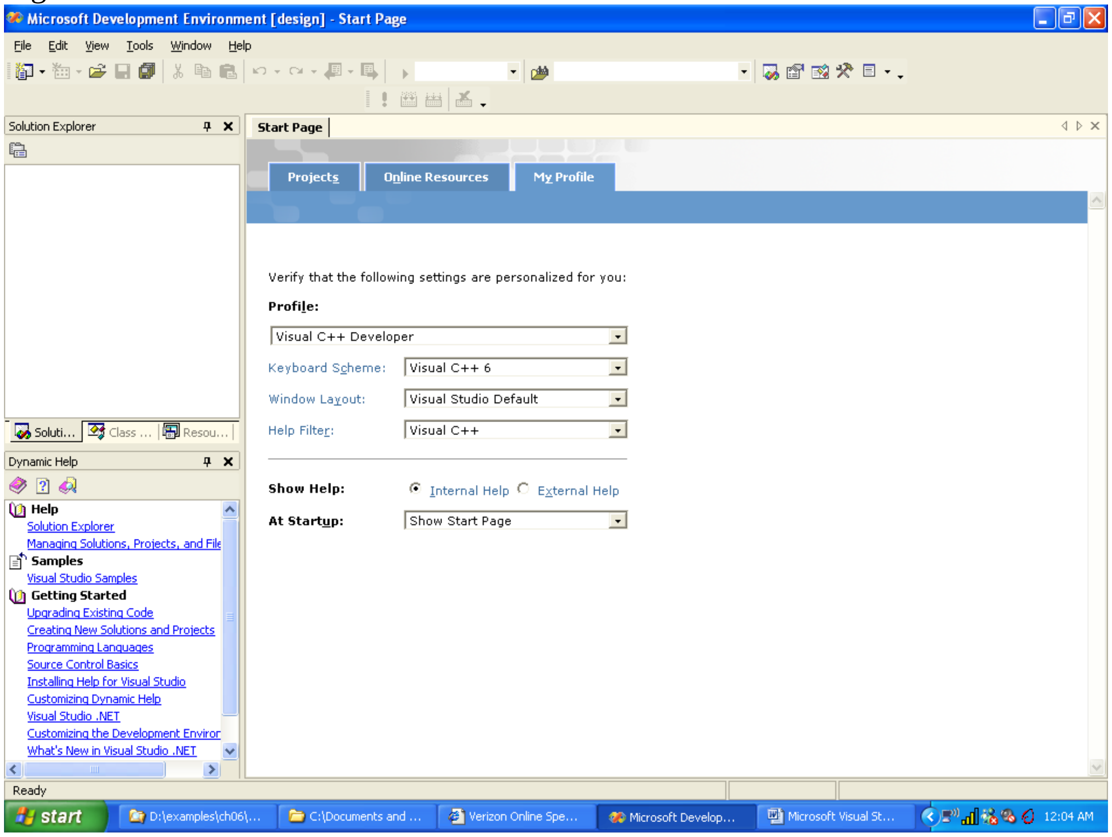


Figure 1 – The main window of Visual Studio . NET

2. <u>Setting the ***Profile*** for C/C++ program development</u>:

Click the ***My Profile*** tab as shown in Figure 2. From top toward bottom of the drop-down lists, select ***Visual C++ Developer***; ***Visual C++ 6*** for ***Keyboard Scheme***; ***Visual C++ 6*** for ***Window Layout***; and ***Visual C++*** for ***Help Filter*** and the remaining ones as shown in Figure 2.



Verify that the following settings are personalized for you:

Profile: Visual C++ Developer

Keyboard Scheme: Visual C++ 6

Window Layout: Visual Studio Default

Help Filter: Visual C++

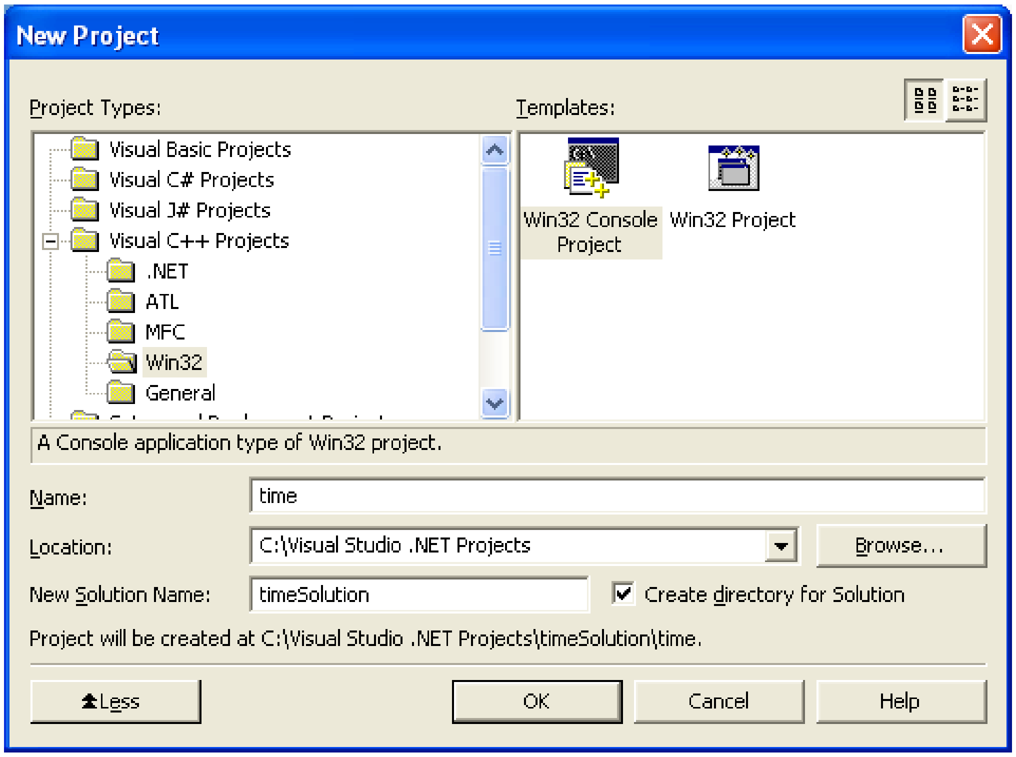
Show Help: [✓ Internal Help] External Help

At Startup: Show Start Page

Figure 2 – Setting the profile

3. <u>Creating an empty new Win32 Console Project</u>

Enter *time* as the name for the “Project” and *timeSolution* as the name for the “Solution”; specify *C:\Visual Studio .NET* as the drive and directory that will be used to stored all the files for the *time* “Project” and the *timeSolution* solution. As shown in Figure 3 below, the *time* project folder will be created at *C:\Visual Studio .NET\timeSolutioin\time*. It is noted that in this demo, the “Solution” (*timeSolution*) contains only one “Project” (*time*); in general, a complex “Solution” or application may contain more than one project.


New Project

Project Types:

- Visual Basic Projects
- Visual C# Projects
- Visual J# Projects
- Visual C++ Projects
  - .NET
  - ATL
  - MFC
  - Win32
  - General

Templates:

- Win32 Console Project
- Win32 Project

A Console application type of Win32 project.

Name: time

Location: C:\Visual Studio .NET Projects

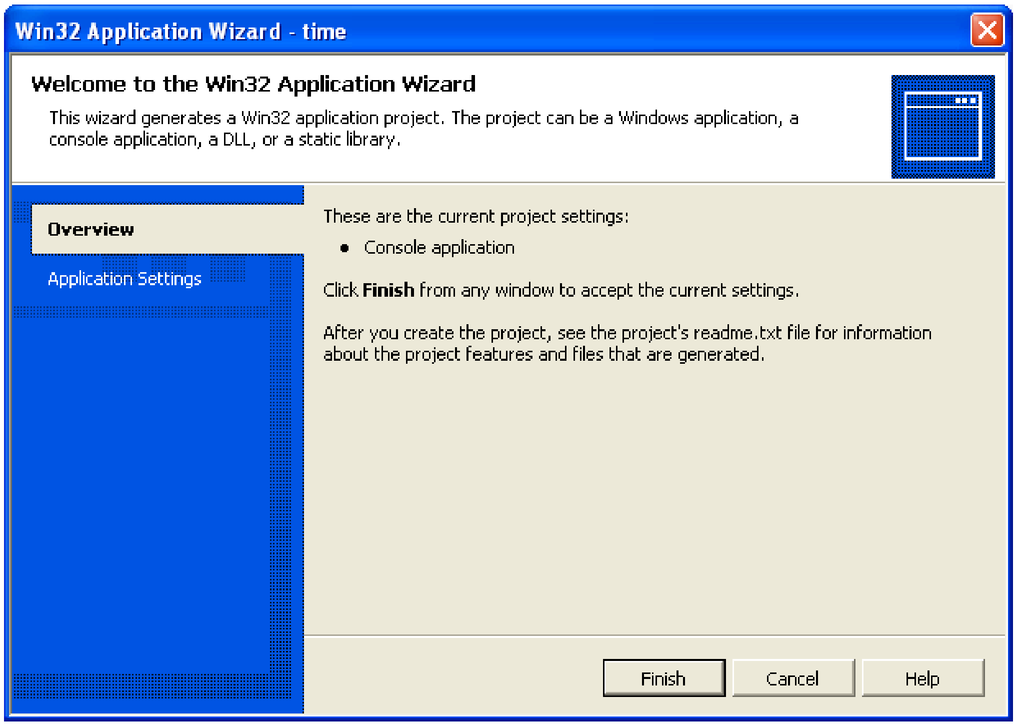
New Solution Name: timeSolution

[✓] Create directory for Solution

Project will be created at C:\Visual Studio .NET Projects\timeSolution\time.

Figure 3 –The ***New Project*** dialog box

Click the “Create directory for Solution” box and then the “OK” button, the system displays the ***Win32 Application Wizard – time*** dialog box as shown in Figure 3 below.


Win32 Application Wizard - time

Welcome to the Win32 Application Wizard

This wizard generates a Win32 application project. The project can be a Windows application, a console application, a DLL, or a static library.

These are the current project settings:

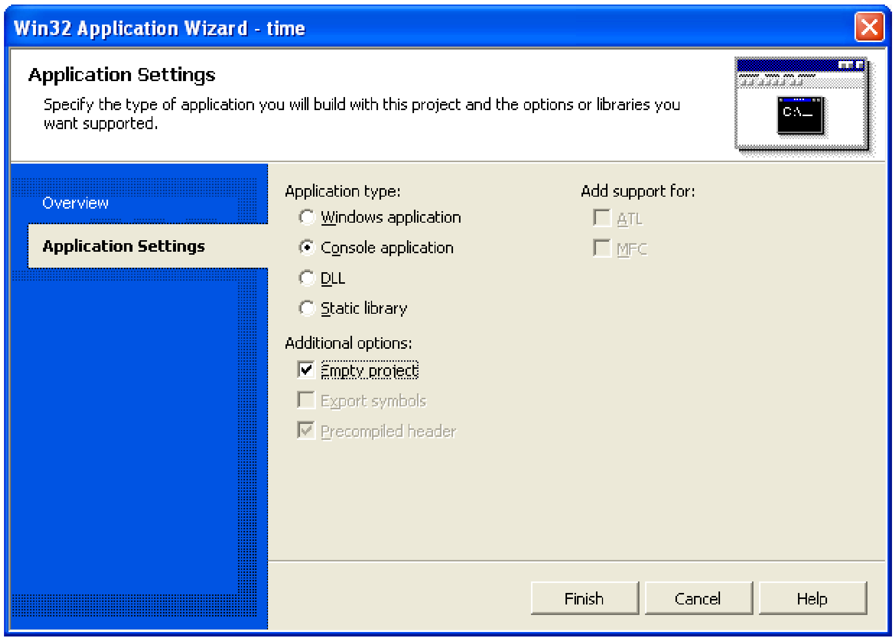
- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

Figure 3 – The ***Win32 Application Wizard – time*** dialog box

Click the "Application Settings" tab to display the dialog box shown in Figure 4 below.



Application Settings

Specify the type of application you will build with this project and the options or libraries you want supported.

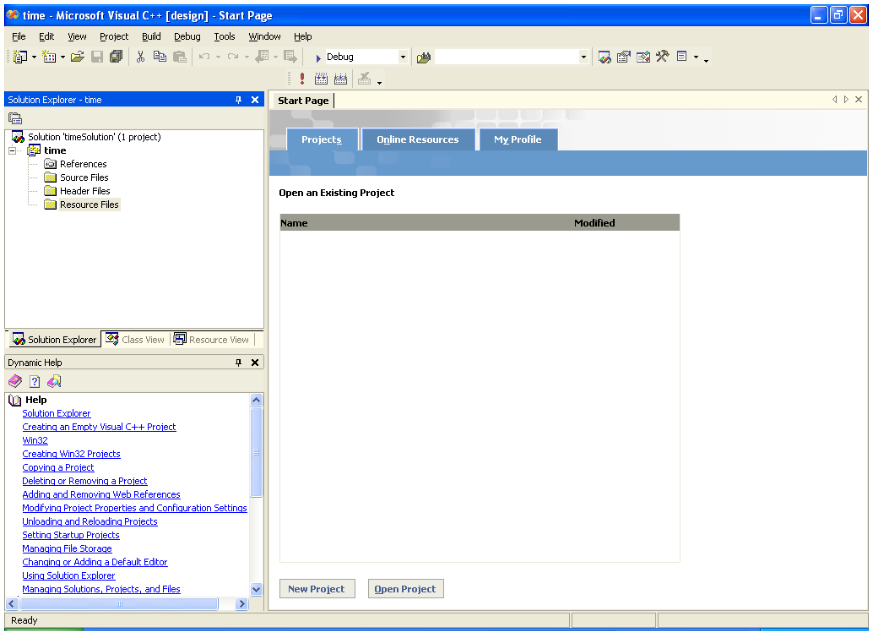
Application type:
( ) Windows application
(●) Console application
( ) DLL
( ) Static library

Additional options:
[✓] Empty project
[ ] Export symbols
[✓] Precompiled header

Add support for:
[ ] ATL
[ ] MFC

Figure 4 – The ***Application Settings*** tab of the ***Win32 Application Wizard***

Make sure the ***Empty project*** box is checked and then click the ***Finish*** button to display the main Visual Studio .NET window as shown in Figure 5 below. Let's take a look at the ***Solution Explorer - time*** pane on the left of the main window where the ***Solution Explorer*** tab has been clicked. Notice the hierarchical organization of the solution in the display: the *timeSolution* solution contains 1 project, namely, the *time* project and it contains three empty folders. We will create and store all .cpp files in the ***Source Files*** folder and all .h files in the ***Header Files*** folder as illustrated below.



Solution 'timeSolution' (1 project)
time
References
Source Files
Header Files
Resource Files

Open an Existing Project

Name	Modified

Help
Solution Explorer
Creating an Empty Visual C++ Project
Win32
Creating Win32 Projects
Copying a Project
Deleting or Removing a Project
Adding and Removing Web References
Modifying Project Properties and Configuration Settings
Unloading and Reloading Projects
Setting Startup Projects
Managing File Storage
Changing or Adding a Default Editor
Using Solution Explorer
Managing Solutions, Projects, and Files

Figure 4 – Three empty folders for the projects has been created

4. <u>Adding files to the newly created but empty project</u>

As mentioned in the beginning of this tutorial, the *time* application contains three files: *time.h*, *time.cpp*, and *timeMain.cpp*. We first add *time.cpp* file to the ***Source Files*** folder and then type in its source code. We then repeat the steps for the *timeMain.cpp* and the *time*.h files. Note that in general, all the .cpp files are added to the ***Source Files*** folder and all the .h files to the ***Header Files*** folder.

a. <u>Adding *time.cpp* file to the ***Source Files*** folder</u>

Right click anywhere on the ***Source Files*** and then click Add -> Add New Item… to display the following dialog box (you may click on **ADD Existing Item** should the file have previously been created) as shown in Figure 5.

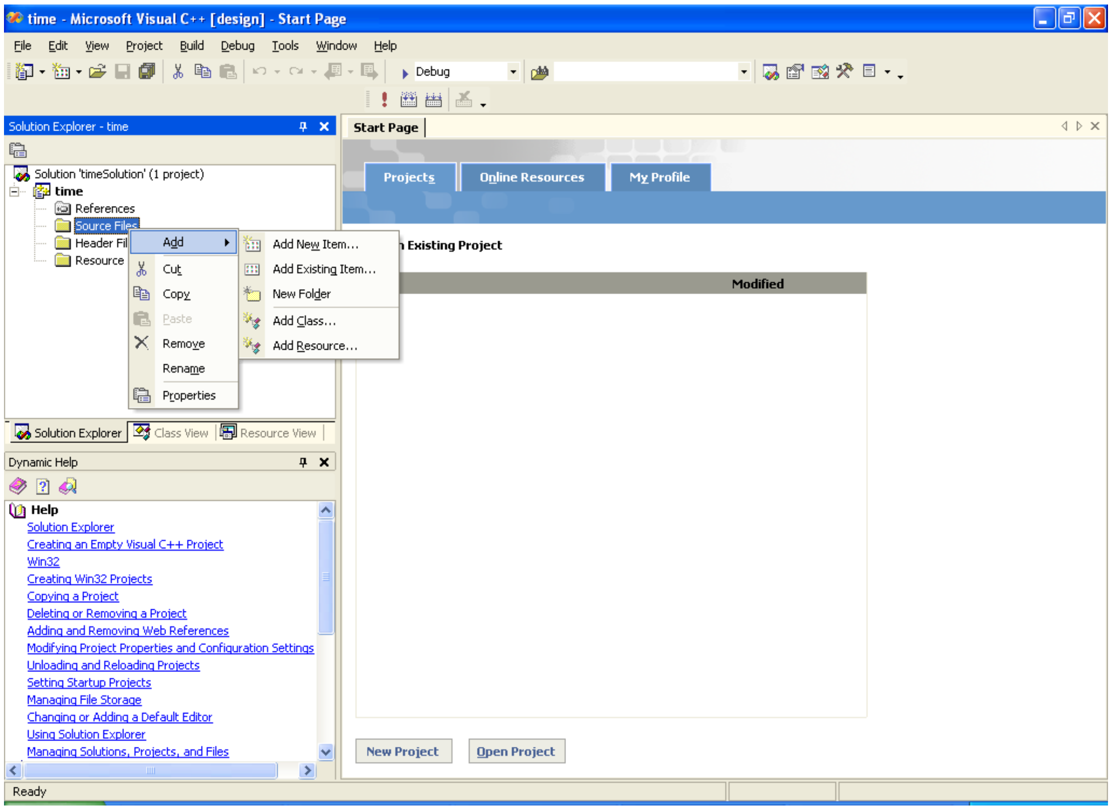
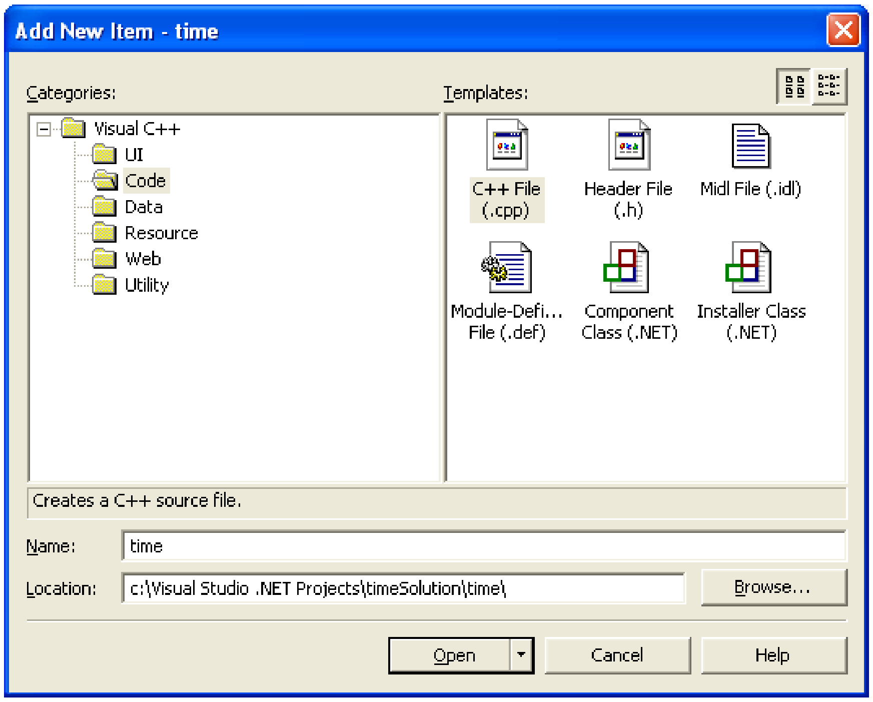


Figure 5 – Adding .cpp files to the ***Source Files*** folder

The system displays the Add New Time – time dialog box as shown in Figure 6 below. Click on ***Code*** icon in the left ***Categories*** pane and select ***C++ File (.cpp)*** in the right ***Template*** pane; enter *time* as the name for the file (do not type .cpp, as the system will automatically appends the .cpp extension).



Creates a C++ source file.

Name: time

Location: c:\Visual Studio .NET Projects\timeSolution\time\

Figure 6 – Select ***Categories*** and ***Templates*** and enter filename

Click the ***Open*** button the system display the screen shown in Figure 7. Notice that the *time.cpp* file appears under the ***Source Files*** folder. Also shown in Figure 7 is the blank area on the right for entering and editing the source code for the *time.cpp* file.

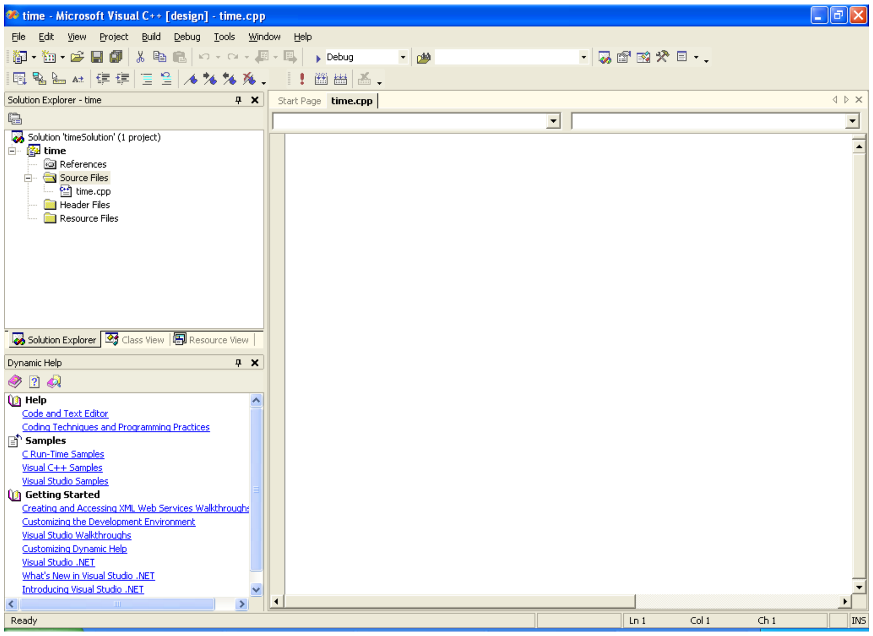
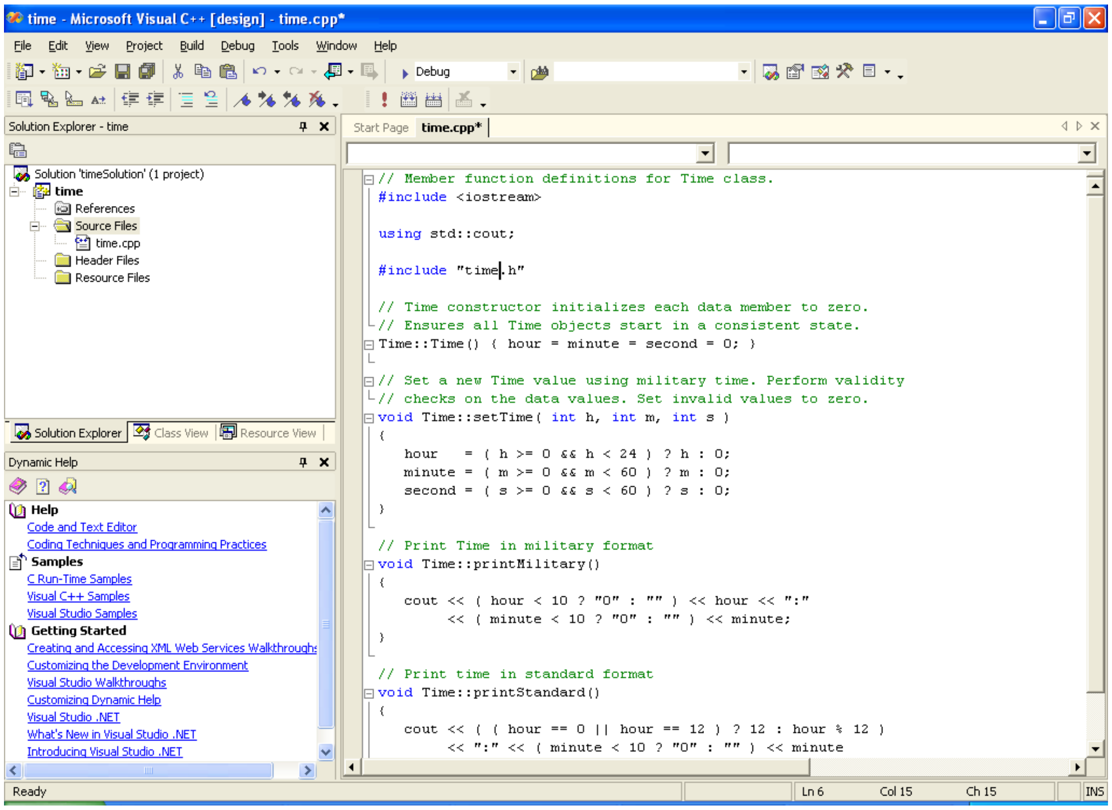

time - Microsoft Visual C++ [design] - time.cpp
File Edit View Project Build Debug Tools Window Help
Debug
Solution Explorer - time
Solution 'timeSolution' (1 project)
time
References
Source Files
time.cpp
Header Files
Resource Files
Solution Explorer
Class View
Resource View
Dynamic Help
Help
Code and Text Editor
Coding Techniques and Programming Practices
Samples
C Run-Time Samples
Visual C++ Samples
Visual Studio Samples
Getting Started
Creating and Accessing XML Web Services Walkthroughs
Customizing the Development Environment
Visual Studio Walkthroughs
Customizing Dynamic Help
Visual Studio .NET
What's New in Visual Studio .NET
Introducing Visual Studio .NET
Start Page
time.cpp
Ready
Ln 1
Col 1
Ch 1
INS

Figure 7 – Code entering/editing area on the right of the main .NET window.

Type in the C++ source code for *time.cpp* file as shown in Figure 8 below (only partially visible; the detailed code is provided in Appendix I).


time - Microsoft Visual C++ [design] - time.cpp*
File Edit View Project Build Debug Tools Window Help
Debug
Solution Explorer - time
Solution 'timeSolution' (1 project)
time
References
Source Files
time.cpp
Header Files
Resource Files
Solution Explorer
Class View
Resource View
Dynamic Help
Help
Code and Text Editor
Coding Techniques and Programming Practices
Samples
C Run-Time Samples
Visual C++ Samples
Visual Studio Samples
Getting Started
Creating and Accessing XML Web Services Walkthroughs
Customizing the Development Environment
Visual Studio Walkthroughs
Customizing Dynamic Help
Visual Studio .NET
What's New in Visual Studio .NET
Introducing Visual Studio .NET
Start Page
time.cpp*
Ready
Ln 6
Col 15
Ch 15
INS

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

using std::cout;

#include "time.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
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
```

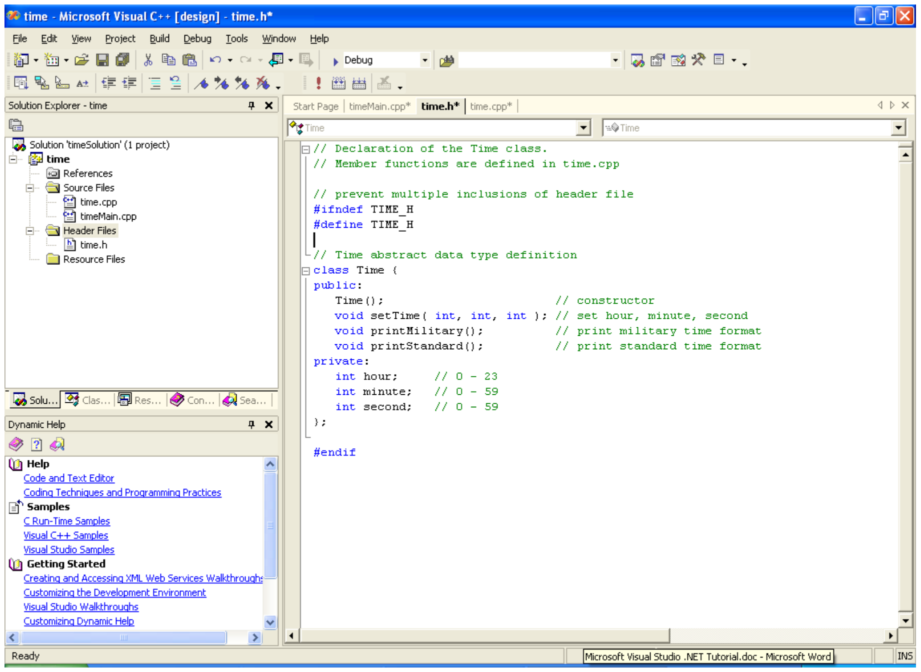
Figure 8 – Source code for the *time.cpp* has been entered as shown.

b. <u>Adding *timeMain.cpp* file to the ***Source files*** folder</u>:

Repeat the above steps *timeMain.cpp* to the "Source files" folder and then type in its code.

c. <u>Adding *time.h* file to the ***Header Files*** folder</u>:

Repeat the above steps shown to add *time.h to* the ***Header Files*** folder and then type in the code. The following screen shows all three files have been properly added to their respective folders. In the code entry area, the *time.h* tab has been clicked so the window shows the code of the *time.h file*. Now we have ready to compile, link, and execute our program.



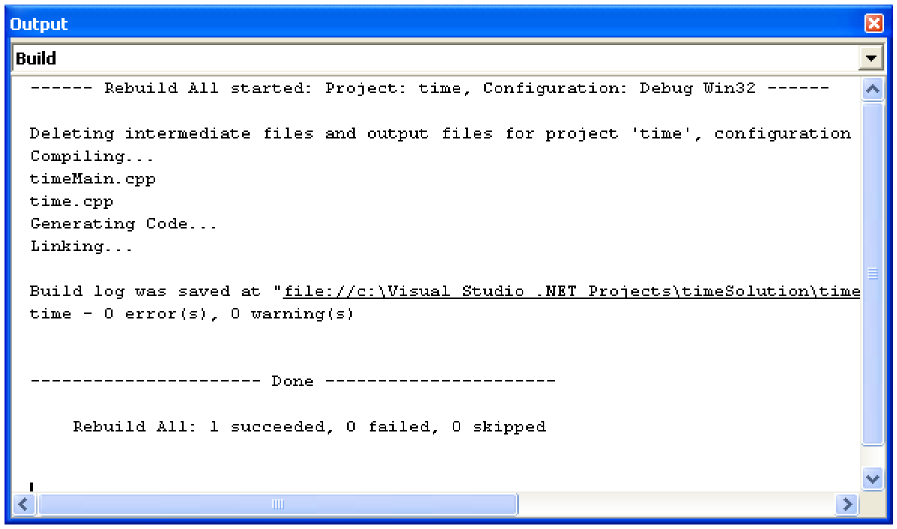
```
// Declaration of the Time class.
// Member functions are defined in time.cpp

// prevent multiple inclusions of header file
#ifndef TIME_H
#define TIME_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
```

Figure 9 – All three files has been created; source code for the *time.h* is displayed.

5. <u>Compiling, linking, and executing the three-file project</u>

a. To build or compile the Solution:

Click **Build Solution** in **Build** menu or press F7, the system displays a floating ***Output*** window shown below (Note: depending on how it is configured the ***Output*** window may be docked right next to the .cpp and .h files. The window displays the messages about the compilation and linking process. If your project contains syntax and linkage errors, diagnostic messages are displayed. In our demonstration, two .cpp and one .h files are successfully compiled and the system has produced the *time.exe* load module or file for execution.

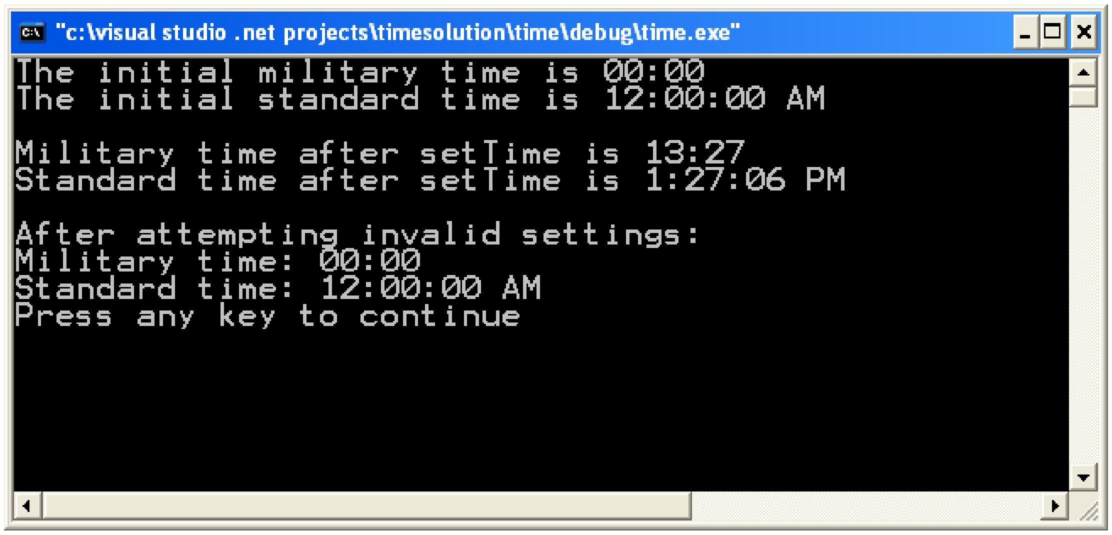


```
------ Rebuild All started: Project: time, Configuration: Debug Win32 ------

Deleting intermediate files and output files for project 'time', configuration
Compiling...
timeMain.cpp
time.cpp
Generating Code...
Linking...

Build log was saved at "file://c:\Visual Studio .NET Projects\timeSolution\time
time - 0 error(s), 0 warning(s)


---------------------- Done ----------------------

    Rebuild All: 1 succeeded, 0 failed, 0 skipped
```

Figure 10 – The ***Output*** window shows successful compilation and linking

b. To execute the load module:

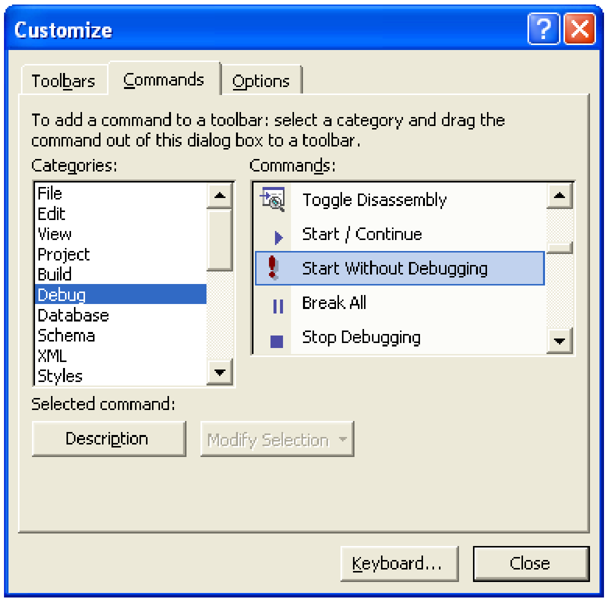
Click **Debug -> Start Without Debugging** or press **Ctrl+F5**, the load module (*time.exe* file) is executed and the result is displayed in the console or DOS box as shown below:



```
"c:\visual studio .net projects\timesolution\time\debug\time.exe"
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
```

Figure 11 – The console or DOS window displays the result of execution of *time.exe* file.

Note that as demonstrated in tutorial I, steps 5.a and 5.b can be consolidated into a single step by clicking the ***Start Without Debugging*** icon, an exclamation symbol in the ***Build*** tool bar. Again, if the ***Build*** tool bar is not displayed; simply right click anywhere in the tool bar area and then click ***Build*** in the pop-up menu to activate it. It is possible that the ***Start Without Debugging*** icon is not present in the ***Build*** tool bar. To display it, right click the tool bar area and then click **Customize…** The system displays the ***Customize*** dialog box shown in Figure 12 below:



Customize

Toolbars | Commands | Options

To add a command to a toolbar: select a category and drag the command out of this dialog box to a toolbar.

Categories: File, Edit, View, Project, Build, Debug, Database, Schema, XML, Styles

Commands: Toggle Disassembly, Start / Continue, Start Without Debugging, Break All, Stop Debugging

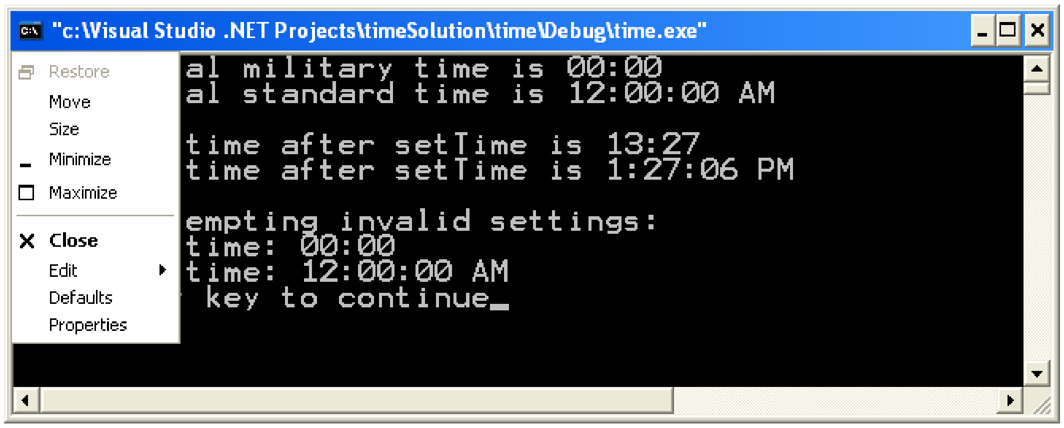
Selected command:

Figure 12 – The Customize window.

In the box, click the ***Commands*** tab and select ***Debug*** in the ***Categories*** pane on the left and locate the exclamation mark in the ***Commands*** pane on the right; drag the exclamation symbol and drop it in the ***Build*** tool bar.

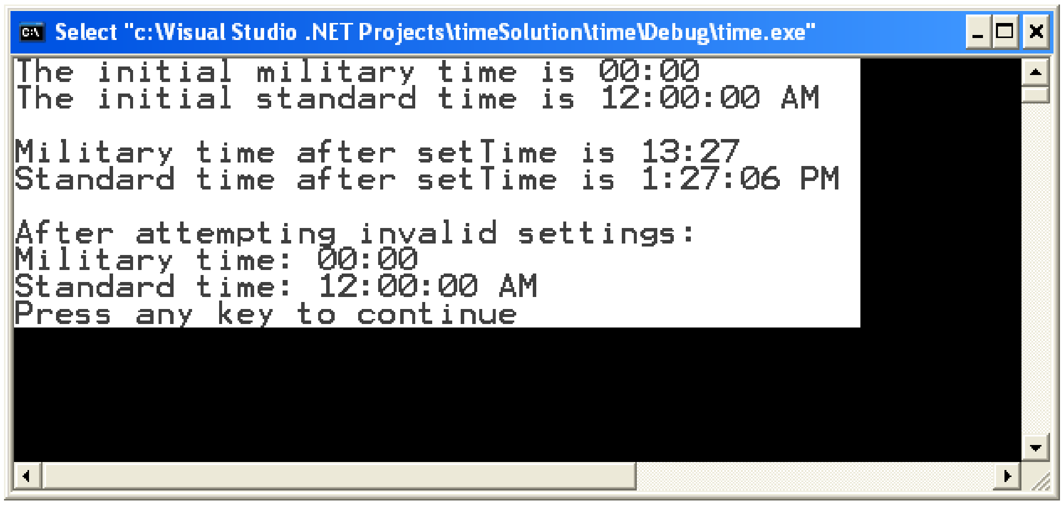
6. Printing the source files and the output results in the console window:

a. To print the source files: click **File -> Print** in the menu bar and click to print all .h and .cpp files.
b. To print the output displayed in the DOS box: click the DOS icon at the left end of the title bar to show the popup menu below.


"c:\Visual Studio .NET Projects\timeSolution\time\Debug\time.exe"
Restore
Move
Size
Minimize
Maximize
Close
Edit
Defaults
Properties

```
al military time is 00:00
al standard time is 12:00:00 AM

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

empting invalid settings:
time: 00:00
time: 12:00:00 AM
 key to continue_
```

Click **Edit -> Mark** to display a little blinking rectanglar marker at the upper left corner in the console or DOS window. Drag the rectangle to mark or highlight the block of text as shown in below, cut and paste it to a blank MS Word or WordPad document and print the document from there.


Select "c:\Visual Studio .NET Projects\timeSolution\time\Debug\time.exe"

```
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
```

Appendix

The source code for *time.h*, *time.cpp*, and *timeMain.cpp* used in Tutorial II (Deitel & Deitel)

time.h file:

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

// prevent multiple inclusions of header file
#ifndef TIME_H
#define TIME_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
```

time.cpp file:

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

using std::cout;

#include "time.h"

// Time constructor initializes each data member to zero.
// Ensures all Time objects start in a consistent state.
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" );
}
```

***timeMain.cpp file*:**

```
// Driver for Time1 class
// NOTE: Compile with time1.cpp
#include <iostream>

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

#include "time.h"

// Driver to test simple 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;
}
```