|  |  |  |
| --- | --- | --- |
|  | Fundamentals of Computer Programming  Visual Studio IDE & Basic C++ constructs | Spring 2019 |

# **Objectives**

# By the end of this lab, the student should be able to:

* List basic C++ constructs.
* Write simple C++ programs.
* Differentiate between the roles of the editor, compiler, linker, debugger and IDE.
* Use Visual C++ IDE to edit, compile and run console applications.
* Differentiate between syntax, runtime and logical errors.
* Use debugger to trace programs and to locate and solve program errors.

# **Part I - Basic Programming tools**

# To develop software using C++ (or any compiled language) you need:

* A text **editor** to write the *source code* and save it into a *source file*.
* A **compiler** and a **linker** that accept source files, compile them and produce the corresponding machine instructions usually in the form of an executable file

In C/C++, the final output is produced in 3 stages as shown in Figure 1.1:

1. ***Preprocessing*** each source file

In this stage, the preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using #define with their values.

1. ***Compiling*** each preprocessed file to produce an object file

The compiler parses the source code and translates it into machine code.

In this stage, syntax errors or failed overload resolution errors are reported.

1. ***Linking*** these object files to produce the final output (the executable file).

The linker links all the object files by replacing the references to undefined symbols contained within them with the correct addresses. Each of these symbols can be defined in other object files or in libraries. If they are defined in libraries other than the standard library, you need to tell the linker about them.

In this stage, the most common errors are missing definitions or duplicate definitions.

|  |
| --- |
|  |
| Figure1-1 : Compilation Process |

**Debugger**

Sometimes, and **even if your program has no syntax errors**, it may not work as desired because of a logical error or a *run-time* *error*. In this case, the compiler does not produce error messages and you need to detect the part of the code causing the problem and fix it, a process known as *debugging*. A **debugger** is a computer program that enables the developer to monitor the execution of the program, run it step by step, monitor variables or even change their values during execution, go back in time … etc to help the developer detect logical and runtime problems in the code. More details about debugging will be explained in next lab. Interested reader is referred to <http://en.wikipedia.org/wiki/Software_bug>

A developer (programmer) can use the editor, the compiler and the debugger separately, but usually the developer uses a package that integrates these three tools as well as other tools, allowing the developer to use them and view their output through a single convenient graphical user interface. Such package is called an **integrated development environment** or an **IDE**.

This lab is concerned with editing, compiling and debugging *console applications* using Microsoft Visual Studio IDE.

# **C++ Basic Language Constructs**

* Constant and variable **declaration**.
* Assignment Statement.
* Arithmetic and logic operations including comparison.
* Input and output
* Comments.
* Conditional statements (if, switch-case )
* Loop statements (for, while, do-while)
* Arrays.
* Functions
* Built-in subroutines for input and output.
* Structures and Classes

In this lab, these features will be demonstrated for C++ language.

# **Part II – Visual Studio IDE**

# **Creating and Manipulating Projects**

Almost all IDEs let the developer organize his work into *projects*. A project contains a set of files to be compiled as well as compilation settings to produce output file (e.g. exe file).

Here are the steps that you should follow in this lab to make a project.

1. Open Microsoft Visual Studio
2. Select File🡪New 🡪Project
3. In the dialog box that will appear:
   1. Select Visual C++ 🡪 Win32 🡪 Win32 Console Application
   2. Type project name and path.
   3. Click “Ok”, select **“An empty project”** and Click “Finish”

Now the project is set up, you are ready to add a new source file to the project and write your program:

1. Select Project🡪"Add New Item" or Right Click on the newly created project🡪Add🡪"New Item"
2. In the dialog box that will appear:
   1. Select “C++ File” from the list.
   2. Type file name (e.g. program.cpp) and click “Ok”.

Now you will find “program.cpp” in the “Source Files” folder in the “Solution Explorer” window as shown in Figure 1-2. Double click it to start editing. Copy the following code:

|  |
| --- |
| **#include <iostream>**  **using namespace std;**  **int main()**  **{**  **cout << "Hello World !" << endl**  **return 0;**  **}** |
| Code 1-1 : Hello World Program |

Save the file. You have now completed the editing phase and are ready to compile your program and make an exe. Select Build🡪 Compile (or just press Ctrl+F7). If you copy and paste the code as it is, Visual C++ will report an error as in Figure 1-2. When you double click on the error message, an indicator shows you the line of code that caused the error. Correct it and compile again.

|  |
| --- |
|  |
| Figure 1-2 : Compilation Error |

Once you compile the code without any errors, you can run linker by selecting “Build🡪Build X”, where X is the name of your project.

**What does "Build" actually do?**

If your project has multiple source files, each file should be compiled first before running the linker to link all compiled error-free files. But actually when you select "Build X" as mentioned above, the IDE first compiles all source files then runs the linker if all files are error-free. The linker produces the executable file.

Now you can run the executable file (X.exe) to see the output by selecting "Debug🡪 Start Without Debugging" or press Ctrl + F5.

|  |
| --- |
| **C:\Users\Eman\AppData\Local\Microsoft\Windows\INetCacheContent.Word\download.jpgNotes:**   * If you cannot find solution explorer or error list, you can view them by:   View -> ‘Solution Explorer’  View -> ‘Error List’ |

# **Visual Studio "Solutions"**

Frequently, you need to create multiple projects related to the same product. These projects may need to use functions from each other. In this case it is a good idea to group these projects in a single *‘solution’*. When you create a project, it is placed in a solution. To create another project in the same solution, select File🡪New Project and in the dialog box that will appear to add a new project. Select "Add to Solution" or Right Click on the current solution from solution explore window then 🡪Add🡪"New Project".

When a solution contains multiple projects, one of them will be the *active project*. This is the project that will respond to build, debug and execute commands. It is shown in the solution explorer in boldface as shown in Figure 1-3. To change the active project, right-click the project you want to activate and select “Set as Startup Project”.

For example, Figure 1-3 shows the ‘solution explorer’ window. The solution name in the figure is ‘DS\_Lab1’. It contains 2 projects: ‘HelloWorld’ and ‘MinMaxAvg’. ‘MinMaxAvg’ is the active project (the **bold** one), so if you build/run, ’MinMaxAvg’ is the project that will be built/run.

|  |
| --- |
|  |
| Figure 1-3 : Multiple Projects |

Table 1-1 summarizes the most important file types used by Visual C++.

|  |  |  |
| --- | --- | --- |
| **File Type** | **Description** | **Example** |
| Solution | A logical container that can contain one or more projects. | Sol\_name.**sln** |
| Project | A logical container that contains one or more files that, when built, produces a single target. | Proj\_name**.vcxproj** |
| Target | A File that results from building a project. Such a file can be an executable application, a library or a component. | Proj\_name.**exe**  Proj\_name.**lib**  Proj\_name**.dll** |
| Source File | A file that needs to be compiled. The result of compilation constitutes a part of the target of the project. | File\_name.**cpp** |
| Object File | The result of the compilation of a single source file. This file contains the machine code corresponding to the source code as well as other information that helps in the linking process. | File\_name.**obj** |
| Header File | A special type of file, which contains source code that is not compiled alone but rather included by source files and hence compiled as part of those source files that include it. | File\_name.**h** |
| Table 1-1: File types used by Visual C++ | | |

You can find the previous files in the location that you choose when you created new project as shown in Figure 1-4. The .sln file is the solution file, so double click on it when you want to open the solution. ‘MinMaxAvg’ and ‘HelloWorld’ folders are 2 projects inside the solution. Each project contains its .cpp, .vcxproj …etc inside it.

|  |
| --- |
|  |
| Figure 1-4 : Solution Folder Examples |

When you want to copy a solution to give it to your friend or to submit it, you should copy the parent folder that contains all the folders and files shown in Figure 1-4. Note that if you copied the .sln file only, you will not be able to open the projects inside it because you don’t copy the projects themselves and the source files with it too.

The following files/folders are NOT needed when copying your code

* Any “Debug” folder
* ipch folder and .vs folder
* Files with extensions .ncb, .sdf

|  |
| --- |
| **C:\Users\Eman\AppData\Local\Microsoft\Windows\INetCacheContent.Word\download.jpgNotes:**   * .sln, .cpp, …etc are called file extensions. * To view file extensions, follow the steps shown in this link:   <https://helpx.adobe.com/x-productkb/global/show-hidden-files-folders-extensions.html> |

# **Part III – Debugging**

There are 3 types of errors: syntax, runtime and logical errors.

**Syntax errors** include errors in writing c++ statements or following the syntax rules of the language such as: forgetting a semicolon, using an identifier(variable) without declaring it …etc. They also include type checking part.

You can see these errors in “error list” window in visual studio (view -> “error list” if it’s not already viewed) and they’re shown in red line in the source files. Syntax errors are caught during compilation time (before running).

After fixing all syntax errors first, you can start running your program. **Runtime errors** appears during runtime when it reaches a statement that when executed causes the program to crash. For example, division by zero will cause runtime error.

When your program finishes the execution of all statements successfully (no runtime errors) but the output you get is not as expected, now you’re facing **Logical errors**. This means that for example you made a wrong calculation or you have problems in your designed algorithms.

Syntax errors are easy to handle because their descriptions are clear and you can google them if you want to know how to solve them. However, runtime and logical errors are harder to know their reasons. For runtime and logical errors, you need to use **Debugger**.

Debugging enables you, for example, to run your program statement by statement (Shortcut: F10). After each statement, you can visualize (watch) the values of all local variables to trace your program execution and find out where it goes wrong and fix it.

Note that you cannot start debugging before fixing all syntax errors first.

Visual C++ provides various debugging operations. They are summarized in Table 1-2:

| **Operation** | **Button** | **Shortcut** | **Description** |
| --- | --- | --- | --- |
| Start without Debugging | Icon_Execute | Ctrl+F5 | This is the same as running the exe (No debugging), except that the program pauses upon termination and prints “Press any key to continue …”. |
| Start Debugging | Icon_GO | F5 | The program runs under the debugger, meaning that it will pause at breakpoints, handles run-time errors in the debugger ... etc. When execution is paused, this command causes it to resume. |
| Stop Debugging | Icon_Stop | Shift+F5 | When the program is paused, this command stops it and closes the debugger. |
| Restart | Icon_Restart | Ctrl+Shift+F5 | When the program is paused, this command causes it to restart. |
| Breakpoints | Icon_BreakPnt | F9 | Breakpoints are “stop signs” in the code that causes the program to pause when reaching them. They are useful to detect whether some portion of code is executed at a particular situation. They are also useful when we want to focus on a certain portion of the code. |
| Run to Cursor | Icon_RuntoCrsr | Ctrl+F10 | This command causes the program to run/resume until it reaches the line of code where the typing cursor is located and then pause. However, execution will be paused if a breakpoint is met beforehand. |
| Step Over | Icon_StepOvr | F10 | This command causes the program to execute one line of the source code and then pause. |
| Step Into | Icon_StepInto | F11 | This command is similar to the above. However if the executed line calls a function, execution pauses at the beginning of this function. |
| Step out | Icon_StepOut | Shift+F11 | This command causes the program to execute until it exits the current function and then pause. |
| Table 1-2: Common Debugging Operations | | | |

The debugger also provides various tools to monitor the state of the program. Among them are:

1. Locals and Auto

The Locals window displays the values of local variables that are accessible in the current scope. The Auto window displays the values of variables used in the current statement and the previous statement.

1. Watch Windows

Watch windows display the value of user provided expressions, allowing the user to monitor these values.

1. Call Stack (will be clearer when you study functions in C++)

The call stack shows the path of function calls from the operating system to the current function. It is the lower right window in figure 3.

The three windows described above are shown in figure 3.

|  |
| --- |
|  |
| Figure 3 : Call Stack, Auto and Locals and Watch Windows |

# **Part IV – Practice problems**

Run code examples provided with the lab.

For the 2nd example:

After the first successful run, re-run again in debugging mode to learn how to use debugger tools.

# **Part V – Practice problems**

Solve the following problems from text book:

"Y. Daniel Liang, Introduction to Programming with C++, Pearson Educational Limited 2014."

**Chapter 2 – Problems:**

**1, 9, 6, 13, 17, 18, 19**