**Visual C++**

Lecture #1  
Introducing Visual C++

**What is C++?**  
C++ is a general-purpose programming language created in 1983 by Bjarne Stroustrup. It is an enhanced version of the C programming which is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories and was originally created for developing applications to run in the Unix operating system. C++, as a direct descendant of C, preserved most fundamental features of the C programming language, yet C++ has many new features such as the ones that simplify memory management and garbage collection. C++ also has a much better support in object-oriented paradigm. Since its debut, C++ quickly became widely used in the software industry, and is possibly the most popular languages ever created.

The following is a simple “console” program written in standard C++ to display a text “Hello world!” in the console screen. In computing, a “console” usually means a combination of a display monitor and an input device (usually a keyboard and mouse pair) which allows a user to input commands and receive visual output from a terminal device or computer system.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello world!";
}
```

It is necessary to note that this course is not designed to discuss how to develop console applications using standard C++. This course will focus on developing simple Windows applications using Visual C++.

**What is Visual C++?**  
Visual C++ is a variation of the C++ language created and maintained by Microsoft as one of the supported programming languages of the .NET Platform. As a member of Visual Studio, Microsoft’s IDE (integrated development environment) product, Visual C++ can build applications written in C, C++, and C++/CLI programming languages. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML and CSS.

As a commercial product, Visual Studio package includes Visual Basic, Visual C#, Visual C++, and other languages. As of August 2018, the latest version is Visual Studio 2017, which supports Visual C++ with a full-featured code editor, compiler, project templates, designers, code wizards, a powerful and easy-to-use debugger, and other tools. Interestingly, on June 6, 2018 Microsoft announced Visual Studio 2019 (version 16) without announcing the release date.

In a nutshell, the Visual C++ IDE is one of the GUI tools provided by Microsoft’s Visual Studio package for developing a variety of C or C++ programs. Application developers, often use the Visual C++ IDE to create Windows-based and .NET-based applications with GUI (graphical user interface) to run in the Windows operating system environment. Developers of C or C++ also use the Visual C++ IDE to create traditional C and C++ applications.
The following is a simple GUI program written in Visual C++ to display “Hello world!” in a GUI-based message box, as shown in Figure 1. A later section will explain the code in detail.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("Hello World!");
}
```

Visual C++ applications can run on both 64- and 32-bit processors. Programmers can build applications to run on a variety of architectures including x86, x64, and or ARM in a standalone Windows PC, a Windows server, or Xbox as well as the cloud-based network.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>x86</th>
<th>x64</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows XP</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Windows Server 2003</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Windows Vista</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Windows Server 2008</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Windows 7</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Windows Server 2012 R2</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>Windows 8</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Windows 10</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

Students can download the “Visual Studio 2017 Community” version for free and use it throughout the course. When starting Visual studio for the first time, new users might be asked to sign in and provide some basic registration information. Users must provide a Microsoft account, a work, or a school account that best represents them. If a user does not have any of these accounts, the alternative is to create a Microsoft Outlook account for free and use it to complete the registration of Visual Studio.

Interestingly, since Visual Studio 2015, the installation wizard of Visual Studio does not automatically install the Visual C++ IDE nor the support of C++ console environment. It becomes an on-demand option. Students must manually configure the installation wizard in order to properly install Visual C++ and all necessary supports for Visual C++.

Starting from Visual Studio 2017, users of MacOS can download the “Visual Studio for Mac” version. However, due to the limitations of lab equipment, all lecture notes of this course are written with an assumption that all students will use Windows operating systems, particularly Windows 10 and 8.1. As of this version of lecture notes, no support will be provided to students using Mac computers.
Native code, managed code? What are they?

Programs written in **native code** will be compiled from a programming language directly to the machine language and the compiled code will be executed directly by the processor without any intermediary or translator. The following standard C++ code is a sample of native code.

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello world!";
}
```

![Machine level of codes](1011110111011011011011011111........)

Programs written in **managed code** is compiled in a way that the instructions require another program as intermediary to manage the machine codes before the processor executes them. The term “managed” largely reflects on how the .NET Framework implements memory management, garbage collection, and other advanced features to ensure the “resource-efficient” execution of a Visual C++ program.

According to Microsoft, the .NET Framework provides over twenty high-level programming features for Visual C++ programs to better manage resources. The “intermediary” is a unified set of libraries encoded into the so-called “Intermediate Language” (IL). During the compilation, ILs are compiled into native codes while Visual C++ codes are compiled as managed code. When executing, IL codes will collaborate with Visual C++ codes within a managed execution environment to ensure an efficient resource management including type safety, array bound and index checking, exception handling, and garbage collection.

In the following sample code, the bold-faced codes are statements that import libraries provided by the .Net Framework to automatically manage resources; therefore, the entire sample code is a managed code.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("Hello World!");
}
```

![Using Managed library](Using Managed library)

![Using managed namespace](Using managed namespace)

Prior to the debut of “managed code”, a standard C++ code is a native code that does not have built-in memory management mechanism. Programmers must write additional codes to manage physical memory and collect garbage. The quality of resource management is subject to the quality of the codes written by the programmer to “manage” resources. Lacking features like “reference counting” and garbage collection, native codes often have the weakness of insufficient capability to manage resources. For example, after execution, used physical memory of a native code may not be properly released or freed for a better allocation, unless the programmer provides sophisticated code to manage the resources.
Managed codes, on the other hand, utilize mechanisms of memory management and garbage collection (as well as other features) provided by the .NET Frameworks. All executed “managed codes” automatically inherit these mechanisms; therefore, their used resources will be freed and allocated by the runtime at the end of program’s life span.

In general, standard C and C++ programs are native while C++/CLI and Windows Forms applications are managed. The term “Windows Forms” (or WinForms) is a set of graphical (GUI) libraries provided by the Microsoft .NET Framework as application programming interfaces (APIs) for building rich client applications to run in Windows desktop environments. Any application created using this platform is known as “Windows Forms applications” (or WFPs). A later lecture will discuss “Windows Forms application” in detail.

Microsoft introduced a managed extension for C++, known as “Managed C++” (MSVC++), to enhance the interoperability between native code and the Common Language Runtime (CLR) in a managed program. For a while, Microsoft also uses the term “Managed C++” to describe “Managed C++” codes since the release of Microsoft Visual C++ 2002. However, Microsoft deprecates managed C++ in 2004 and began promoting its successor C++/CLI (C++ modified for Common Language Infrastructure) which is complete revision that aims to simplify the Managed C++.

The following is the Managed C++ version of the “Hello world!” code as compared to the previous version written in standard C++.

```
using <mscorlib.dll>
using namespace System;

int main()
{
    Console::WriteLine(S"Hello, world!");
    return 0;
}
```

C++/CLI is currently the primary specification for “Managed C++” codes, so all lectures of this course is designed based on the C++/CLI specification. It is necessary to note that C++/CLI preserves many features of the deprecated “Managed C++”, such as the use of “System” namespace. The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions. For example, the Environment class which provides tools to obtain information about, and means to manipulate, the current environment and platform. The following is a C++/CLI code that uses the “OSVersion” property to get the current platform identifier and version number.

```
using <System.dll>
using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("OS Type: " + Environment::OSVersion);
}
```

By the way, the plus sign (+) in the following statement denote the “concatenation” operator. In programming, “concatenation” means to combine two or more values with at least one string literal to produce another single string literal. Any character sequence enclosed by a pair of double quotes, such as "OS Type: ", is a string literal.
MessageBox::Show("OS Type: " + Environment::OSVersion);

In a nutshell, a string concatenate any value produce a single string. A later lecture will discuss the concept of “concatenation” in detail.

\[
\text{string1 + Value1 + Value2 + \ldots.}
\]

\[\text{a new string}\]

With C++/CLI, programmers can easily “inherit” members of classes (or structures) from .Net Framework libraries. The following figure illustrates the processing flow. The “managed program” sends “managed codes” to CPU (or processor) with part of the instructions telling the CPU to reference libraries from the .Net Framework. Altogether, managed program and the .Net Framework collaboratively provide instructions for the CPU to generate the output and free the used resources.

The following demonstrates how to create an “instance” of the DateTime structure while “dt” is the identifier (name) of the instance. An “instance” of a class or structure is an abstract entity (known as an “object”) defined by the programmer to represent the class or structure in a given program. DateTime is a structure in the System namespace, and it provides many tools to manipulate date and time values. The “Today” property of DateTime structure, for example, gets the current date from the computer. By inheriting the “Today” property from the “DateTime” structure, the following code obtain the demanded date value from the .Net Framework (which is installed in a Windows computer).

```c++
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    DateTime dt = DateTime::Today;
    MessageBox::Show("Today is " + dt);
}
```

The above code also uses the “Show()” method provided by the “MessageBox” class of the .Net Framework to display the output. The plus sign (+) in the message box denotes the “concatenation” operator, not the “addition” operator. Any character sequence enclosed by a pair of double quotes, such as "Today is ", is a string literal. A combination of a string literal with other value produces a new string literal.

All managed codes must be compiled using the so-called “C++/CLI compiler” (cl.exe) with the “/clr” (short for common language runtime) option. The common language runtime is a .Net Framework-specific run-time environment that runs the managed code. By importing the .Net
libraries, the “C++/ CLI compiler” generates MSIL (Microsoft Intermediate Language) code to manage memory and garbage collection.

The following is a sample statement in the simplest form. It specifies how to compile a Visual C++ source file (named 1.cpp) with the “clr” option. A more advanced use of the “clr” option will be discussed in a later section.

```
C:\cis223>cl.exe /clr 1.cpp
```

It is necessary to note that any C++ code compiled without ‘clr’ support is treated as “unmanaged code”, and such applications are called “unmanaged programs.” Only those C++ code compiled with “/clr” option can be called managed code (or a “managed program”).

C++/CLI is a secular programming language, which means it can be used for managed or unmanaged or mixed-mode programming. A mixed code is the code consisting of both native and managed codes in one program. Typically, it is the manage codes that provide the structure to host some native codes. In the following, “std::cout” is a tool (provided by the Standard C++ language) that displays a text on a console screen, while “MessageBox::Show()” is a tool of the .Net Framework that displays a text in a GUI-based message box. The “iostream” is a console library while “System.dll” and “System.Windows.Forms.dll” are managed libraries. All C++ console codes are native codes.

```
#include <iostream>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
std::cout << "Hello console!"; //console
MessageBox::Show("Hello GUI!"); //managed
}
```

GUI and console code can co-exist in one single source file and work interactively. In the following example, `cin` (standard input stream) is a Standard C++ tool used to take inputs redirected by the extraction operator (>>), and then treat the input as string literal. When the user enters an integral value, such as 19, the two letters, 1 and 9, are passed to a variable of `int` type named “age”. The value of “age” is then displayed by a message box. The plus sign (+) in the message box denotes the “concatenation” operator, not the “addition” operator.

```
#include <iostream>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
int age;
std::cin >> age; //console
MessageBox::Show("Hello GUI!"); //managed
return 0;
}
```
```cpp
#include <iostream> // console
using namespace std;

int main()
{
    cout << "How old are you? "; // display a string on console
    int age;
    cin >> age; // save the entry in a variable
    MessageBox::Show("You are " + age + " years old.");
    return 0;
}
```

During the execution, the user will see a string message in the console asking “How old are you?”. The user also needs to reply in the console by entering some digits (e.g. 12). The final output is a message box that looks similar to:

```
You are 12 years old.
```

In the following sample code, there is a native function name `squareIt()` which calculates the square of an integer. Inside the main() function, the managed function `Show()` calls the `squareIt()` function by sending an integer and then including the returned value as the output.

```cpp
#using <System.dll> // Windows forms
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

#include <iostream> // console
using namespace std;

int squareIt(int x) //native function
{
    return x * x;
}

int main()
{
    cout << "Enter an integer: "; // display a string on console
    int x;
    cin >> x; // get a value as input
    MessageBox::Show("The square is " + squareIt(x));
    return 0;
}
```

Most of the textbooks use console application to teach basic programming concepts and skills because the console is relatively simple compared to the Windows desktop. However, it is the
instructor’s intention to gear the lecture discussion towards GUI-based code. Throughout the course, students will write managed codes in Visual C++. Yet, some of the first few examples are mixed codes.

From standard C++ code to GUI-based C++/CLI code

One concern raised by previous students is how the transition from console applications to GUI applications could have impact on the learning of programming skills. Many students worried about if their previously learned programming skills might not apply to the coding of GUI applications. Despite the execution environment, console or GUI environment, most of the basic programming skills (those can be acquired from any programming course) will apply to the coding part of application development.

In this section, the instructors will demonstrate how to apply the previously learned skills (if any) to the C++/CLI coding. By the way, for those who do not have any programming background, this course will guide them through the learning journey to acquire the basic programming skills.

The following is a simple console application written in Standard C++. For those who do not have experience in C++ programming (not a prerequisite for this course), the next few paragraphs will briefly explain the code for the sake of comparing Standard C++ code with its C++/CLI version. The learning objective of this course is placed on C++/CLI of Visual C++, not C++ programming.

```
Line 0  // C++ code
1    #include <iostream>
2    using namespace std;
3    int main()
4    {
5        cout << "Welcome to Visual C++!" << endl;
6        return 0;
7    }
```

The zero-th line is a “comment” for human reader to use as reference. It is not a computer statement and is ignored by the compiler.

The first line uses the “include” keyword (known as a “directive”) to tell the compiler to copy and paste source code from another file (known as “header” file) whose file name is “iostream”.

The second line uses the “using” directive to declare the use of the “std” namespace. In Standard C++, a “namespace” is a logical container containing tools for the programmers to use.

From the third to the seventh lines is a function named “main()” which will be executed by default during the program execution. A function in C++ typically encloses its body with a pair of curly braces: { and }.

The fifth line uses the “cout” function provided by the “std” namespace to display a text on the screen. It also uses the “endl” keyword to insert a new line.

The sixth line uses the “return” statement to send an integer 0 to indicate the success of execution.
Source codes of a console application written in standard C++, such as the following, can be rewritten in C++/CLI code to be converted to a GUI-based Window application. The orange lines show the replacement of codes. The structure is similar, the programming logic is the same, and no difference in programming skills.

```cpp
#include <iostream>
using namespace std;

int main()
{
    cout << "Welcome to Visual C++!" << endl;
    return 0;
}
```

The following is the GUI-based version of code. The standard C++ “header files” (iostream) are replaced with .Net Framework’s “dynamic link library” files (.dll). The “std” namespace is replaced by “System”.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("Welcome to Visual C++!");
    return 0;
}
```

Most user-interface classes in the “System::Windows::Forms” namespace provides a common set of classes you can use and derive from to build Windows Forms applications. It will be programmatically convenient to declare the use of the “System::Windows::Forms” namespace. The MessageBox::Show() method, for example, is a member of the System::Windows::Forms namespace that displays a message box. The following is a sample message box produced by the above code.

![Message Box Image]

The above examples demonstrate two different execution environments: console application and GUI application. While the complexity of coding increases from a console to a GUI application, and eventually to a Windows Forms application, whatever the new programmers need to learn is the how to implement the right set of tools provided by Standard C++, C++/CLI, and the .Net Framework. All fundamental things are the same in a sense. In the next few lectures, the instructor will guide students through the learning of basic programming skills using C++/CLI as language and Visual C++ as environment.

**Concept of escape sequence**

Programming languages use “wildcard character” to present special functions or to define a special meaning to be used within string literals. In other words, a “wildcard character” is a set of characters used as a “note” or “markup” in a program. For example, a pair of double
quotes (") is used to enclose a string literal, as shown below. Both the opening and closing quotes are not part of the string literal.

"How are you?"

Interestingly, the uses of “wildcard character” in programming frequently cause confusing. The following code simply causes syntax error because programming languages like C++ considers a pair of double quotes (") as notation to denote the beginning and the end of a string literal. According to the following code, there are two string literals, "He said, " and "", while How are you? are undefined objects to the program.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("He said, "How are you? "");
}
```

In the following, computer will treat characters enclosed by the first and the second occurrences of double quote (") as one single string literal.

"He said, "

From the third to the forth double quote ("), even though there is no character in between, forms another single string literal.

""

Anything between the above two string literals are expected to be objects of the program. However, they are not defined in the above program as objects. Undefined objects cause error message during compilation.

How are you?

Programmers can use escape sequences to resolve such “literal mis-representations”. Escape sequence releases a “wildcard character”, such as the double quotation mark ("), from its language-specific definitions. In other words, escape sequence forces the compiler to interpret a “wildcard character” as a regular ASCII (or Unicode) character. So, how would the programmer display the following string using escape sequence?

He said, "How are you?"

Programmers will add a backslash (\) as prefix to the wildcard character to void the language-specific definition of the wildcard character, as shown below.

```cpp
MessageBox::Show("He said, \"How are you?\"");
```

The backslash forces the compiler to escape from the interpretation of a wildcard character next to the backslash in a string. Placing a backslash before the double quote mark forces the computer to treat the released double quote (\") as a double quote, not an indicator of beginning or end of a string literal. The following demonstrates the correct way to write the code.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
```
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("He said, \\"How are you?\\n");
}

The output looks:

While the backslash (\) can void the definition of a wildcard character, it is also used to re-
define a list of character combinations to represent a newline character, a tab, or a beep sound.
Invented by Bob Bemer, the following table lists the character escapes supported by regular expres-
sions in the .NET Framework.

<table>
<thead>
<tr>
<th>sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\t</td>
<td>Equals to a tab.</td>
</tr>
<tr>
<td>\v</td>
<td>Equals to a vertical tab.</td>
</tr>
<tr>
<td>\n</td>
<td>Equals to a new line.</td>
</tr>
</tbody>
</table>

The following code uses “\t” (tab key) to add indents. It also uses “\n” to add a new line. A “tab key” adds several blank spaces.

```c++
#using <System.dll>
#using <System.Windows.Forms.dll>
using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("City\tState\tZipCode\n" +
                     "Burbank\tCA\t91510\n" +
                     "Ingham \tMI\t48908\n" +
                     "Houston\tTX\t77003\n" );
}
```

The output looks.

Throughout this course, many sample code requires the use of “\n” to insert a new line and “\t” to insert blank spaces.

**What is a library?**

In programming, a code library refers to a collection of source codes or object codes that are
stored for a later reuse. The following table lists two individual C++ source files. The
“gameover.cpp” file is a program that defines a class of “Game Over” messages for other programs to use and display the defined text(s) in a message box.

```cpp
#include <System.dll>
#include <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

public ref class GameOver
{
    public:
        String^ M101 = "Game Over.");
        String^ M102 = "Game Over. Play Again.");
        String^ M103 = "Game Over. Power Down.");
};
```

The “lab1_5.cpp” file uses the “include” directive to import the source code of “gameover.cpp”. All the codes in the “gameover.cpp” will be copied and pasted to the “lab1_5.cpp” file. A later lecture will discuss about the coding of object-oriented paradigm in detail.

```cpp
#include "gameover.cpp"

int main()
{
    GameOver^ gl = gcnew GameOver;
    MessageBox::Show(gl->M101 + "\n" +
                     gl->M102 + "\n" +
                     gl->M103 );
}
```

Without the need to write the same code in “lab1_5.cpp”, the “lab1_5” program can reuse the source code written in “gameover.cpp” and display a message box with “Game Over.” in it.

In a large-scale programming project, library codes can significantly reduce project development time and improve the efficiency. The above figure illustrates the concept. In a game application, both “missle.cpp” and “gameover.cpp” contain reusable codes for the rest of programs (stage1.cpp, stage2.cpp, and so on) to use.

As discussed previously, a library in C++ is a collection of reusable code to be reused by another C++ source file. Like the Standard C++, the .NET Framework provide libraries. The following compares three types of commonly used libraries and illustrates how they function in Visual C++.

- **.h**: Traditional header file. It is a source file containing declarations (as opposed to .cpp .cxx etc. containing implementations). This type of libraries are typically designed for console applications.
- **.lib**: Static-link library that may contain code or just links to a dynamic library. They either declare the binary interface to a dynamic library (DLL) or contain the binary code of a library.
• `.dll`: Dynamic-link library. A dynamic link library (DLL) is a collection of small programs, which can be called upon when needed by the executable program (EXE) that is running.

The need to learn hand-coding without using IDEs

Many experienced programmers in the industry advocate the need for students to learn the basic programming knowledge and skills without the use of IDE (Integrated Development Environment). IDEs are simply software designed to develop programs. They generally provide a “code editing” environment with tools to facilitate the writing of code much quickly and efficiently. However, a sophisticated IDE with intelligent features like automatic error detection and correction can debug the code for students; therefore, students often miss the chance to manually debug the code. Such an intelligent tool may not be an ideal learning tool for students. They take away the opportunities for students to learn how to manually debug the code, which is a very important skill demanded by the software industry.

Visual Studio, for example, provides intelligent features to: (a) automatically generate codes and display only the outcome on screen, (b) automatically highlight typographic errors, (c) suggest a list of available functions, and (d) minimize the need to hand-write the code.

While an intelligent IDE can maximize the efficiency, effectiveness, and productivity of an experience programmer, using IDEs to learn programming often have critical disadvantages that must be addressed in a college-level programming course: (1) students typically acquire only the skills to use IDEs to assemble applications, not to be proficient in coding, (2) IDEs by default prevent students from seeing details of the source code and the structure of the program, (3) students tend to learn to build applications without knowing details of the language. Details of hidden codes should not be overlooked when learning a new language. Overall, using an IDE may hamper the learning of a new language.

Based on the industrial advisors’ recommendations, the instructor urges students to avoid learning Visual C++ with the use of IDE (particularly Visual Studio). Instead, students should learn to hand-code from scratch by manually entering every line of code to a generic (the simpler the better) text editor (such as Notepad), and then use their eyes and brains to debug the code, spot on the typos and errors, and seek for a better way to refine the code. Consequently, students will learn how an application is created and later excel in programming.

Visual Studio Developer Command Prompt

Visual Studio 2017 provides a handy command-line tool for programmers to build their applications. The tool is known as “Developer Command Prompt”, which is the only tool needed for students to hand-code, hand-debug, and hand-compile Visual C++ applications throughout this course.

```
** Visual Studio 2017 Developer Command Prompt v15.0.26403.3
** Copyright (c) 2017 Microsoft Corporation
******************************************************************************
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community>
```

Unlike the regular Windows Command Prompt, the Developer Command Prompt (previously known as Visual Studio Command Prompt) automatically sets the environment variables to access all coding-related supports provided by the .NET Framework, with the premise that these
supports have been properly installed along with the installation of Visual Studio. The following is the prompt of regular Windows Command Prompt. Do not use the regular Command Prompt to compile codes written in Visual C++!!!

```
Microsoft Windows [Version 10.0.17134.167]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Users\user>
```

Starting with the Visual Studio 2015, the 64-bit versions of Visual Studio provide two command prompts: the **Developer Command Prompt** for 32-bit tools, and the **Visual Studio x64 Win64 Command Prompt** for 64-bit tools. Since most PCs and Laptops are 64-bit based, students should use the 64-bit version of Developer Command Prompt for a better compatibility. In most cases, it is marked as “**Developer Command Prompt for Visual Studio**”.

Throughout this course, students will hand-code Visual C++ source code and compile it with Visual C++ compiler in the Developer Command Prompt. Students will use a generic text editor (such as Microsoft Notepad) to write the source code. Visual C++ files may be suffixed by *.cpp, *.cc, or *.c. As a principle, all source files appear in this course will be suffixed by *.cpp for the sake of simplification. The following figure illustrates how to save the source code as a file named “**lab1_1.cpp**” as a generic text file in Notepad.

The **cl.exe** is a command-line compiler tool that controls the Microsoft C and C++ compilation and linkage. It can be run only on operating systems that support Microsoft Visual Studio and the .NET Framework. As discussed previously, all managed codes written in Visual C++ must be compiled with the use of “**clr**” option. The following illustrates how to compile a source file written in Standard C++ named “lab1_1.cpp”.

```
cl.exe /clr lab1_1.cpp
```

or simply,

```
cl /clr lab1_1.cpp
```

The “**clr**” option is not required for C++ code written in Standard C++. The following illustrates how to compile a source file written in Standard C++ named “std_c.cpp”.

```
C:\Users>cl.exe std_c.cpp
```

or simply,
The following is a completed sample code of a generic Windows-based application. It is a managed code. This program uses a .NET Framework method, `Show()`, provided by the `MessageBox` class to display the specific string “Welcome to CIS223!”. The `Show()` method will call the .Net Framework to build a message box with the string in it. The next lecture will explain the code and the concept of “Windows Form Application” in detail. By the way, students will use the “`Show()`” method very frequently throughout the entire course.

```cpp
#include <System.dll>
#include <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("Welcome to Visual C++!");
}
```

The following figure is a screen shot of Notepad with the above source codes. The file name is “lab1_1.cpp”.

Students will then launch the Developer Command Prompt. The following is a sample screen shot.

By default, users will be directed to a directory similar to the following.

```
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community>
```

or

```
C:\Program Files\Microsoft Visual Studio\2017\Community>
```

In this course, the instructor recommends students to create a “C:\cis223” directory and save all the Visual C++ source files (or projects) in that directory. Then, use the “cd” (short for Change Directory) command to change to that directory by typing `cd c:\cis223` and pressing [Enter]. It is necessary to note that “cd” is a DOS command for changing from current directory to a destination directory in the Windows Command Prompt environment.

```
C:\Program Files\Microsoft Visual Studio\2017\Community>cd c:\cis223
```
The following illustrates how to compile a Visual C++ source file (named “lab1_1.cpp”) to a self-executable program “lab1_1.exe” using the “/clr” option of the “cl.exe” compiler. The Visual C++ compiler will generate one “.obj” (short of “object code”) file that contains MSIL codes rather than machine executable instructions, while the linker will create one “.exe” file which is self-executable in the Windows environment.

C:\cis223> cl.exe /clr lab1_1.cpp
Microsoft (R) C/C++ Optimizing Compiler Version 15.00.21022.08
for Microsoft (R) .NET Framework version 4.00.50727.3053
Copyright (C) Microsoft Corporation. All rights reserved.
lab1_1.cpp
Microsoft (R) Incremental Linker Version 10.00.21022.08
Copyright (C) Microsoft Corporation. All rights reserved.
/out:lab1_1.exe
lab1_1.obj

It is necessary to reiterate that all sample codes presented in this course requires the “/clr” (indicating the Common Language Runtime Compilation) option because they contain .NET Framework classes and .NET libraries that are written for the “Common Language Runtime”. In the Developer Command Prompt, students have options to type either cl.exe /clr lab1_1.cpp or simply cl /clr lab1_1.cpp and press [Enter] to start the compilation.

C:\cis223> cl.exe /clr lab1_1.cpp
or,

C:\cis223> cl /clr lab1_1.cpp

The following two lines indicate that the “cl” compiler has created two new files: lab1_1.obj and lab1_1.exe. The “.obj” file contains the object codes which are instructions written in machine language. The “.exe” file is a self-executable file and is the one a human user should use to execute the program.

/out:lab1_1.exe
lab1_1.obj

Students can type dir lab1_1.* and press [Enter] to check if lab1_1.exe is created successfully, as shown below. The “dir” (short for directory) command is a command used for listing directory contents.

C:\cis223> dir lab1_1.*
Volume in drive C has no label.
Volume Serial Number is 84AF-8E2B
Directory of C:\cis223
01/01/2015  10:05 PM          218  lab1_1.cpp
01/01/2015  10:05 PM          23,824  lab1_1.exe

Once the executable file is created successfully, users can type lab1_1.exe (or simply lab1_1) and press [Enter] to execute the program.

C:\cis223> lab1_1.exe

Or, simply

C:\cis223> lab1_1
The following is a sample output of the above statement. A rectangular GUI “form” (known as “message box”) created by the MessageBox::Show() method will pop up with a text in it.

It is necessary to note that the “cl.exe” compiler, by default, creates a “DOS-based” .exe file. The term “DOS-based” program means that it must be executed in a Command Prompt (or a DOS environment). According to Microsoft, the default “Output File Format” of “cl.exe” is set to be “Console”. With this format, the Windows operating system must first emit a Command Prompt (a console emulator), and then use that Command Prompt to run the “console” program.

The following figure illustrates the sequence of executions with an assumption that a user attempts to execute a “DOS” command using “Windows Explorer”.

In the above figure, the dashed line indicates that there is a way to bypass the Command Prompt. The “cl.exe” provides the “/subsystem” option for specifying the environment for the executable. When the subsystem is set to be “windows”, the compiled application does not require a console. The following uses the “subsystem” option to forces the “cl” compiler to create a Windows-based application.

```
cl.exe /clr FileName.cpp /link /subsystem:windows /ENTRY:main
```

The “/ENTRY” option is a Visual C++ linker options that specifies which function in the program should be treated as the starting point. The following illustrates the syntax, where `function` is a function that specifies a user-defined starting address (or starting point) for an .exe file or DLL.

```
/ENTRY:function
```

In C++, the `main()` function is the language-designated starting point of all C++ programs. Visual C++ preserves this convention; therefore, it is necessary to specify to start with the `main()` function, as shown below.

```
/ENTRY:main
```

Throughout this course, students will compile almost all Visual C++ codes by issuing a statement similar to the following.

```
cl.exe /clr lab1_1.cpp /link /subsystem:windows /ENTRY:main
```

**Universal Windows Platform**

Visual Studio 2015 introduced the **Universal Windows Platform** (UWP) which is an API created by Microsoft and first introduced in Windows 10. The purpose of this platform is to help develop universal apps that run on Windows 10, Windows 10 Mobile, Xbox One and HoloLens without the need to be re-written for each.
UWP is relatively new. It is an extension of the Windows Runtime (WinRT). Unlike C++/CLI, WinRT applications natively support both the x86 and ARM processors and run inside a sandboxed environment to allow greater security and stability. Since it is only available in Windows 10 and Windows 10 Mobile operating systems, applications built with UWP do not run on earlier Windows versions.

Interestingly, UWP is not a platform for developing simple applications, it is the Microsoft's newest software development platform for experienced developer to build rich applications for Windows 10 devices. To learn UWP, students should have a solid foundation of programming with some programming experiences in web developing, software developing, and tool developing. However, UWP is a promising direction for students to move towards after completing this course.

Just for fun!!! Try the following program and observe its result. Is it an ASCII art?

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

void main()
{
    String^ str="";
    str+= "    |" + "\n"
    str+= "    |  (o)   (o)  " + "\n"
    str+= "    C           _) " + "\n"
    str+= "    |   ,______| " + "\n"
    str+= "   /______" + "\n"
    str+= "  /" + "\n"
    str+= " /" + "\n"
    MessageBox::Show(str);
}
```

The output looks:

![ASCII art](image)

Review Question

1. Which statement is true about Visual C++?
   A. It is a general-purpose programming language.
   B. It is an integrated development environment (IDE) for the building application for Linux operating systems.
   C. It is a graphical user interface specially created for building native C++ programs.
   D. It is a language specially developed for building iOS applications.

2. Which is not the CPU architecture supported by Visual C++?
   A. x86
3. Which is not an acceptable file extension for Visual C++?
   A. *.cpp
   B. *.cl
   C. *.cc
   D. *.c

4. In terms of Visual C++, CLR stands for __.
   A. common linker runtime
   B. common language runtime
   C. C++ linker runtime
   D. C++ language runtime

5. What will the following code product?
   
   ```cpp
   MessageBox::Show("She said, 'It's right?\n");
   ```
   
   A. She said, It's right?
   B. She said, "It's right?"
   C. She said, 'It's right?'
   D. She said, \"It's right?\n"

6. Which statement is true about the CL.exe tool?
   A. It does not support the common language runtime (CLR).
   B. It only supports Visual C# and Visual C++.
   C. It only work with native C++ code, not managed Visual C++ code.
   D. Programmers can build C and C++ applications on the command line with it.

7. Which can compile a generic Visual C++ Windows Forms application named "lab1.cpp"?
   A. cl.exe /clr lab1.cpp
   B. cl.exe /clr:win lab1.cpp
   C. cl.exe /clr:exe lab1.cpp
   D. cl.exe /clr:winexe lab1.cpp

8. Which of the following will create a message box in Visual C++?
   A. MessageBox::Show("Welcome to CIS223!");
   B. Message::Show("Welcome to CIS223!");
   C. MessageBoxShow("Welcome to CIS223!");
   D. MessageShow("Welcome to CIS223!");

9. The plus sign (+) of the following code denote the __ operator.
   
   ```cpp
   MessageBox::Show("OS Type: " + Environment::OSVersion);
   ```
   
   A. Addition
   B. Append
   C. Concatenation
   D. Combination

10. Given the following code segment, which statement is true?
    
    ```cpp
    cl.exe /clr lab1.cpp /link /subsystem:windows /ENTRY:main
    ```
    
    A. "lab1.cpp" is a console application.
B. The starting point of "lab1.cpp" is the ENTRY() function.
C. The /subsystem option specifies that "lab1.cpp" is an operating system tool.
D. It disables the Console Window.
Preparation #1: the “Hide extensions for known file types” option.

(For Windows 10/8 Users Only)
1. Launch the Windows Explorer.
2. Click “View”.
3. Check the “File name extension” option.
4. Proceed to Preparation #2 now.

(For Windows 7/Vista Users Only)
1. Launch the Windows Explorer.
2. Click “Folders and search options”
3. Click the View tab, and then uncheck the “Hide extensions for known file types” option.
4. Click OK. Proceed to Preparation #2 now.

Preparation #2: Installation
1. Use Internet Explorer to visit https://www.visualstudio.com/vs/community/ and download the latest version of Visual Studio Community for Windows Desktop. As of August 2018, it is the “2017” version. The file name of the installer is something similar to vs_community_8a112csafsh2122ljdk392cess12.exe.
2. Use Windows explorer to locate the .exe file, and then double click it to launch it after the downloading completes.

3. Click Continue, and then wait until the following windows appear.

4. Be sure to select the following three options: (1) Universal Windows Platform development, (2) .NET desktop development, and (3) Desktop development with C++.

5. Click Install to start the installation.
6. Wait till the following window appears. Click Restart to reboot.

Preparation #3: Searching for the Developer Command Prompt  
Windows 10  
1. Open the Start menu, press the Windows logo key on the keyboard.  
2. On the Start menu, enter dev. This step will bring a list of installed apps.  

Windows 8.1  
1. Go to the Start screen, press the Windows logo key on the keyboard.  
2. On the Start screen, press CTRL + TAB to open the Apps list and then enter V. This will bring a list of installed apps.  

4. In the prompt, type cl.exe (“l” as in “large”) and press [Enter]. If “cl” is not recognized, as shown below in the “Abnormal output”, see Appendix B for solution.

<table>
<thead>
<tr>
<th>Normal output</th>
<th>Abnormal output</th>
</tr>
</thead>
</table>
| **Visual Studio 2017 Developer Command Prompt v15.0.26403.7**  
Copyright (c) 2017 Microsoft Corporation  
C:\Program Files\Microsoft Visual Studio\2017\Community>cl.exe  
Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23918 for x86  
Copyright (C) Microsoft Corporation. All rights reserved.  
usage: cl [ option... ] filename... [ /link linkoption... ] | **Visual Studio 2017 Developer Command Prompt v15.0.26403.7**  
Copyright (c) 2017 Microsoft Corporation  
C:\Program Files\Microsoft Visual Studio\2017\Community>cl.exe  
"cl" is not recognized as an internal or external command,  
operable program or batch file.  
C:\Program Files (x86)\Microsoft Visual Studio 14.0> |

Learning Activity #1: A GUI App  
1. Under the C:\ drive, create a directory named C:\cis223 if it does not exist.  
2. Launch the Developer Command Prompt. Do not use regular Windows Command Prompt.  
3. In the Developer Command Prompt, type cd c:\cis223 and press [Enter] to change to the C:\cis223 directory.  
   
   C:\Program Files\Microsoft Visual Studio\2017\Community>cd c:\cis223  
   C:\cis223>  
   
4. Type notepad lab1_1.cpp and press [Enter] to use Notepad to create a new file named lab1_1.cpp.  
5. Click Yes to create.
6. Type in the following source codes (no typo allowed, case sensitive). The objective is to display a text in a pop-up message box.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    MessageBox::Show("Welcome to Visual C++!");
}
```

7. Save the file and exit back to the Visual Studio 201n (Developer) Command Prompt.

8. Type `cl /clr lab1_1.cpp /link /subsystem:windows /ENTRY:main` and press [Enter] to compile.

```cmd
C:\cis223>cl /clr lab1_1.cpp /link /subsystem:windows /ENTRY:main
Microsoft (R) C/C++ Optimizing Compiler Version 15.00.21022.08
for Microsoft (R) .NET Framework version 2.0.50727.3053
Copyright (C) Microsoft Corporation. All rights reserved.

lab1_1.cpp
Microsoft (R) Incremental Linker Version 9.00.21022.08
Copyright (C) Microsoft Corporation. All rights reserved.

/out:lab1_1.exe
/subsystem:windows
/ENTRY:main
lab1_1.obj
```

9. If no error, type `lab1_1.exe` (or simply `lab1_1`) and press [Enter] to test the program. The output looks:

![Welcome to Visual C++!]

10. Download the “assignment template” and rename it to lab1.doc if necessary. Capture a screen shot similar to the above figure and paste it to the Word document named lab1.doc (or .docx).

**Learning Activity #2: Detecting the operating system**

1. Launch the Developer Command Prompt (not the regular Command Prompt) and change to the C:\cis223 directory.

2. Type `notepad lab1_2.cpp` and press [Enter] to use Notepad to create a new source file called lab1_2.cpp with the following contents. The objective is to reference the “Environment” class of the .Net Framework (which is installed in the computer) to display the version of operating system.

```cpp
#using <System.dll>
#using <System.Windows.Forms.dll>
```
using namespace System;
using namespace System::Windows::Forms;

int main()
{
MessageBox::Show("OS Type: " + Environment::OSVersion);
}

3. Type `cl /clr lab1_2.cpp /link /subsystem:windows /ENTRY:main` and press [Enter] to compile.

4. Type `lab1_2.exe` and press [Enter] to test the executable file. A sample output looks:

5. Capture a screen shot similar to the above figure and paste it to the Word document named `lab1.doc` (or `.docx`).

Learning Activity #3: Display the current date
1. Launch the Developer Command Prompt (not the regular Command Prompt) and change to the C:\cis223 directory.

2. Type `notepad lab1_3.cpp` and press [Enter] to use Notepad to create a new source file called `lab1_3.cpp` with the following contents. The objective is to reference the “DateTime” structure provided by the .Net Framework to display the current date.

```csharp
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

int main()
{
    DateTime dt = DateTime::Today;
    MessageBox::Show("Today is " + dt);
}
```


4. Type `lab1_3.exe` and press [Enter] to test the executable file. A sample output looks:

5. Capture a screen shot similar to the above figures and paste it to the Word document named `lab1.doc` (or `.docx`).

Learning Activity #4: Console vs. GUI codes
1. Launch the Developer Command Prompt (not the regular Command Prompt) and change to the C:\cis223 directory.

2. Type `notepad lab1_4.cpp` and press [Enter] to use Notepad to create a new source file called `lab1_4.cpp` with the following contents.
#using <System.dll>
#using <System.Windows.Forms.dll>

using namespace System;
using namespace System::Windows::Forms;

#include <iostream> // console code

int main()
{
    std::cout << "Hello console!"; // console code
    MessageBox::Show("Hello GUI");
}

3. Type cl /clr lab1_4.cpp and press [Enter] to compile the source file to create the executable object file.

4. Type lab1_4.exe and press [Enter] to test the executable file. A sample output in the console looks:

C:\cis223>lab1_4
Hello console!

5. There should also be a pop-up window that look similar to the following.

6. Capture a screen shot similar to the above figures and paste it to the Word document named lab1.doc (or .docx).

Learning Activity #5: A generic Windows Form

1. Launch the Developer Command Prompt (not the regular Command Prompt) and change to the C:\cis223 directory.

2. Type notepad gameover.cpp and press [Enter] to use Notepad to create a new source file called lab1_5.cpp with the following contents.

    #using <System.dll>
    #using <System.Windows.Forms.dll>

    using namespace System;
    using namespace System::Windows::Forms;

    public ref class GameOver
    {
        public:
            String^ M101 = "Game Over.";
            String^ M102 = "Game Over. Play Again.";
            String^ M103 = "Game Over. Power Down.";
    }

3. Type notepad lab1_5.cpp and press [Enter] to use Notepad to create a new source file called lab1_5.cpp with the following contents. The objective is to reuse the code defined in “gameover.cpp”.

    #include "gameover.cpp"

    int main()
{GameOver^ g1 = gcnew GameOver;
    MessageBox::Show(g1->M101 + "\n" +
    g1->M102 + "\n" +
    g1->M103 );
}

4. Type `cl /clr lab1_5.cpp /link /subsystem:windows /ENTRY:main` and press [Enter] to compile the source file to create the executable object file.

5. Type `lab1_5.exe` and press [Enter] to test the executable file. A sample output looks:

![Sample Output Image]

6. Capture a screen shot similar to the above figures and paste it to the Word document named `lab1.doc` (or `.docx`).

**Submittal (Please read the instructions carefully)**

1. Complete all the 5 learning activities and the programming exercise in this lab.

2. Create a .zip file named `lab1.zip` containing ONLY the following self-executable files. (See Appendix C for instructions)
   - Lab1_1.exe
   - Lab1_2.exe
   - Lab1_3.exe
   - Lab1_4.exe
   - Lab1_5.exe
   - Lab1.doc (or `lab1.docx`, or `.pdf`) [You may be given zero point if this Word document is missing]

3. Log in to course web site (e.g. Canvas or Blackboard) and enter the course site.

4. Upload the zipped file to **Question 11** of Assignment as response.

**Programming Exercise 01:**

1. Create a new directory named `C:\cis223` if it does not exist.

2. Launch the Developer Command Prompt (not the Windows’ Command Prompt). You should see something similar to:

   ```
   C:\Program Files\Microsoft Visual Studio\2017\Community>
   ```

3. Type `cd c:\cis223` and press [Enter] to change to the `C:\cis223` directory. The prompt now looks:

   ```
   C:\cis223>
   ```

4. Type `notepad ex01.cpp` and press [Enter] to use Notepad to create a new text file named `ex01.cpp`.

5. In Notepad, add the following two heading lines (place them before your codes and replace `[YourFullNameHere]` with your full name).

   ```
   //File Name: ex01.cpp
   //Programmer: [YourFullNameHere]
   ```
6. Write codes that will create a generic GUI-based Windows application displaying a text message “YourFullName completed Exercise 01!” (Be sure to replace YourFullName with yours.).

7. Compile the source file to create an executable file (By default the file name is 01.exe). Instructions are available in the lecture note. The following figure is a sample output (be sure to use your full name). Points will be deducted if your code display “Jennifer Lopez….”.

8. Download the “programming exercise template” and rename it to ex01.doc if necessary. **Copy your source code to the file** and then capture the screen shot(s) similar to the above one and paste it to the Word document named “ex01.doc” (or .docx). You can also save the document as .pdf file.

9. Compress the source file (ex01.cpp), executable code (ex01.exe), and Word document (ex01.doc) to a .zip file named “ex01.zip”.

**Grading criteria:**
You will earn credit only when the following requirements are fulfilled. No partial credit is given.
- You successfully submit both source file and executable file.
- Your source code must be fully functional and may not contain syntax errors in order to earn credit.
- Your executable file (program) must be executable to earn credit.

**Threaded Discussion**
Note: Students are required to participate in the thread discussion on a weekly basis. Student must post at least two messages as responses to the question every week. Each message must be posted on a different date. Grading is based on quality of the message.

Question: Class, the topic for this week's lecture is "Introduction to Visual C++". In your opinion, what type of applications can be developed using Visual C++? Please provide at least one example. [There is never a right-or-wrong answer for this question. Please feel free to express your opinion.]

**Appendix A: How to compress files in Windows OS**
1. Open the Windows Explorer and change to the directory that has all the following files (e.g. C:\cis223).
2. lab1_1.exe, lab1_2.exe, lab1_3.exe, lab1_4.exe, and lab1_5.exe
3. Highlight all the five files, and then right click the highlighted area as shown below:
4. Select “Send to”, and then “Compressed (zipped) folder”. This step will create a .zip file.
5. Right click the lab1_x.zip file and then select “Rename”.
6. Change the file name to “lab1.zip”.

Appendix B:

<table>
<thead>
<tr>
<th>Symptom</th>
<th>C:\Program Files\Microsoft Visual Studio\2017\Community&gt;cl.exe 'cl' is not recognized as an internal or external command, operable program or batch file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cause</td>
<td>The C++ tools are not being installed by default in Visual Studio 2017.</td>
</tr>
<tr>
<td>Remedy</td>
<td>Locate the “Visual Studio Installer” and launch it. Then follow the instructions on Preparation #2.</td>
</tr>
</tbody>
</table>