Introduction to C++ Programming

C++ is one of the most popular programming languages in use in industry today. According to the TIOBE Programming Community Index, which can be accessed at this url:

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

at the time of writing C++ is the third most popular programming language, after Java and C++'s little brother, C. C++ is a general-purpose, multi-paradigm programming language, which just means that you can do many different types of programming in C++. You can write procedural programs, object-oriented programs, and generic programs in C++, and you can conceivably use all three paradigms in one program.

C++ is also considered to be a relatively hard computer programming language to use. The language has a wide range of features and many programmers only use a small portion of those features on a regular basis. Trying to learn the whole language at once is nearly impossible.

The goal of this book is to introduce you to just a small subset of the huge number of features found in C++, but include the main constructs and features you need to start writing useful C++ programs right away. You will learn how to write simple procedural programs to solve arithmetic problems; you will learn some techniques for manipulating textual data (strings); you will learn how to use decision constructs (branching) in your programs in order to selectively execute code; you will learn how to use repetition (loops) in your programs to repeatedly execute code over and over again; you will learn how to compose functions that enables you to modularize your programs; you will learn how to build data structures such as vectors and arrays so that you can store related data values in memory; you will learn how to process text files in order to work with real world data; and finally you will learn how to write object-oriented programs so that you can extend the C++ language with your own data types and also learn how to write programs that use objects to more accurately model the problems you are trying to solve.

One of the major features that separates this book from all the other books on C++ programming is the fact that we do not try to overwhelm you with facts and features. Each new programming construct and concept is presented clearly and concisely, showing you only those features you need to know to get you started programming in C++. When you finish this book you will not be an expert in the minutiae of C++, but you will be able to write useful programs in C++ and have a great foundation for learning those advanced features that are most important to you.

Finally, the only way you can learn how to program in C++ is to actually write C++ programs. To do that you need a C++ compiler. We choose to use the Microsoft Visual C++ Express compiler for the examples in this book. This compiler can be downloaded for free from Microsoft and contains everything you will need for this book and, for that matter, for more advanced C++ programming.

If you choose to use another C++ compiler, the only code you might have to change will be in the #include statements at the beginning of your programs. If you have trouble making

these changes, please contact the author for help.

You will get the best use of this book if you run all the programs, make changes to them, run them again, and don't be afraid to experiment. Once you've finished a chapter, don't move on to the next chapter until you've worked as many of the exercises as you have time to work. You can only learn to program by programming.

Chapter 1

Getting Started With Visual C++

C++ is a compiled programming language. What does that mean? It means that the programs you write in C++ are translated by the compiler into machine language and then executed. You don't have to understand the compilation process to begin learning how to program in C++. However, you do need to understand the steps your program has to go through from conception to execution. That is what we will be discussing in the chapter.

Downloading and Installing Visual C++ Express

Microsoft provides a free version of C++ for non-commercial purposes, Visual C++ Express (Visual C++). The current version is titled Visual C++ 2010 Express but we will leave out the year since the program is updated very often. You can find it at this url:

http://www.microsoft.com/express/Downloads/#2010-Visual-CPP.

Installing Visual C++ is an automatic process. You should not experience any problems with either the download or the installation, especially if you choose the defaults during installation. If you have any problems with the download or the installation, you can view a video on downloading and installing Visual C++ Express. You can find it at this url:

http://www.techbookspress.com/cpp

Visual C++ programs are organized as projects. From the opening window, you select New Project to start a new program. So click New Project.

The window lists the different types of projects you can create. We will work only with CLR Console Applications in this book, so make sure that selection is highlighted. Then move down to the bottom of the window where the Name text box is located.

Visual C++ Express will use the name you give your project as the means of storing the files for the project. For example, if we name a project Example, a folder with that name will be placed in the Location path shown in the text box below the Name text box. You can use any name you want for your projects, but be sure to use names that are meaningful so you can go back to a project later to work on it. We're going to write a simple “Hello, world!” program now, so enter “HelloWorld” as the name of the project. The name you enter is also added to the Solution name text box below Location. We'll talk more about this box in a later post. Press OK and your project is opened.

This is the main window you will use to write your C++ programs. The panel to the left lists all the files that make up your project. We'll talk more about this pane in future posts. The panel to the right is the code editor where you will enter C++ code. You'll notice that some code is already written. For now, erase the line that starts “Console::WriteLine(...”. Now, below the line that reads “using namespace System;”, add a blank line and write the following two lines below the blank line:

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

Then, above the line that reads “return 0;”, add the following line of code:

```
cout << "Hello, World!" << endl;
```

That last word is “end ell”, not “end 1”, but many beginning C++ programmers mistake the “ell” for a “1”, and it causes the next thing to be printed to start on a new line. In other words, it forces the compiler to perform a newline command.

Now the code in the code editor window should look like this:

```
#include “stdafx.h”
using namespace System;

#include <iostream>
using namespace std;

int main(<System::String ^> ^args)
{
   cout << "Hello, World!" << endl;
   Console::ReadKey();
   return 0;
}
```

When your code looks like this, press the green arrow in the toolbar to run your program, though the tool tip says "Start Debugging." These are really the same thing. You will be asked if you want to rebuild the project. Answer "Yes" to this question. You should then see a console window with the phrase “Hello, World!”. Press any key to close the window.

If all this worked correctly, congratulations on writing your first C++ program. If it didn't work correctly, go back over the steps listed above to see what you did wrong.

How the “Hello, World!” Program Works

Let's examine the Hello, World! program line-by-line to get a basic understanding of what is going on. The first line is:

```
#include "stdafx.h"
```

This line is called a preprocessor directive, and it is used to let the compiler know to treat the contents of the specified file as if they were included with the other source code in the program. The file, stdafx.h, is called a *header* file and is used to store reusable source code that any C++ program can use. The use of header files allows the main language to be as short and concise as possible by allowing us to only call in extra functionality via a header file when we need it. We'll discuss header files more specifically below.

The next line is:

```
using namespace System;
```

Namespaces allow you to group files like header files under a common name. Files that have similar uses will be stored in the same namespace. The header file specified in the first line, stdafx.h, is stored in the namespace System so we have to tell the compiler to look for the file in the System namespace.

The next two lines are:

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

This is another example of a preprocessor directive. The first one above was called so that we can have access to the Console class for keeping the command prompt open when we display data in it. The iostream library allows us to write data to the console window using the

standard output stream, cout. As you can see, iostream exists in a different namespace, the std namespace, which is why we have to issue a separate directive for that file.

The following line:

```
int main(<System::String ^> ^args)
```

tells the system that we are now defining a function called main. Every C++ program must have a function named main, as the compiler looks for this function as the beginning point of a program. The word int before the function name indicates that the main function returns an integer value. For now, just ignore everything inside the parentheses. This is called the parameter list for the function and we won't be using it for now.

The next line consists of just the single character, {, called a curly brace. This character signifies the beginning of a block. } signifies the end of a block. We'll have more to say about blocks later but for now it's enough to know that a function must be defined within a block.

After the block is opened, the next line is the first line of our program:

```
 cout << "Hello, World! << endl;";
```

In this line we're saying that we want the phrase “Hello, World!” to be displayed to the standard output device, which is by default the console, and is defined as the cout object. The << operator, called the “put to” operator, specifies that the data that follows it is to be “put to” the standard output device. In simple terms, this line will cause the phrase “Hello, World!” to be displayed in a command prompt window. We've already discussed what endl does. Each piece of C++ code that ends in a semicolon is called a *statement*. Think of statements in C++ like sentences in English. The compiler checks for correct grammar in a C++ program by analyzing the statements you write.

The next line:

```
Console::ReadKey();
```

uses another object that represents standard output, Console, and calls a method or function, ReadKey(), to leave the command prompt window open until the user presses any key. If we don't use this line, the command prompt window will disappear as soon as “Hello, World!” is written to it.

Next we see:

```
return 0;
```

which is required at the end of the main function to indicate that the program has successfully completed. This line is known as a *return statement*. Every program we write will have this line at the end of the main function even though Windows doesn't really do anything with this value.

Finally, we end the block and the program by writing a closing curly brace:

```
}
```

Don't worry if you don't understand everything discussed in this post. We will be covering all these things again and again as we dig deeper into learning how to program in C++. The next chapter will discuss two of the fundamental building blocks of the C++ language – data types and variables.

Exercises

1. Be sure you can get the "Hello, World!" program working on your computer.
2. Modify the “Hello, World!” program so that it says hello to you.