LESSON 1
INTRODUCTION TO COMPUTERS AND C++ PROGRAMMING *WITH VISUAL STUDIO 2010*

PROF. JOHN P. BAUGH
PROFJPBAUGH@GMAIL.COM
PROFJPBAUGH.COM

CONTENTS

INTRODUCTION

This book and any accompanying materials serve as a first course in computer science, or a supplement to such a course. I welcome (and encourage) students and instructors alike to use and distribute this material as they see fit. I do ask that you reference my website and my name as the author of the materials, though.

This first lesson will give a brief overview of computers and programming, and introduce the C++ programming language. Although it will benefit any student desiring to learn C++, the examples and tutorials are done in the Visual Studio 2010 IDE (Integrated Development Environment), which is an industry standard IDE used extensively for C++ development (as well as development in other languages) on Windows.

ASSUMPTIONS

I assume that the reader is familiar with basic computer usage, typing, Internet usage, and has basic mathematic skills. Also, in order to follow along with examples, tutorials, and especially the video tutorials, I assume you have are using a Visual Studio IDE, preferably Visual Studio 2010 Professional. To obtain a copy of this IDE, you can either purchase it from a distributor, directly from Microsoft, or you can **obtain VS 2010 for free** if you have a .edu mailing address and your educational institution is enrolled with Dreamspark.

Go to Dreamspark.com, and find your educational institution and follow the instructions to download your copy of Visual Studio 2010. If you do not wish to purchase Visual Studio 2010 Professional, and do not have access to Dreamspark, then you can download the free Visual Studio 2010 Express edition, but note that some interface differences do exist between the edition I use, and the Express edition.

1.1 – COMPUTERS

A **computer** is an electromechanical device that can perform computations and make decisions at speeds millions, billions, or even trillions of times faster than a human being. The definition of what a computer is may be a little broader than you expect. Typically, people think of a personal computer such as a **desktop computer** or a **laptop** (also called a **notebook**), or possibly a **netbook** when they hear the term *computer*. These are in fact, computers. However, many other devices are also computers or contain computing components. **Cell phones** (from the dumb ones, up to the so-called **smart phones**,) **electronic readers** (such as the Kindle or Nook), **tablet computers** (such as the iPad, Nexus, or Galaxy Tab 2) are all computers, although they may have special purposes. Also, gaming systems such as **Xbox 360** or **PS3** are also computing devices. And this may surprise you, but your vehicle, especially if it is relatively new, is full of computing devices that control the vehicle and respond to maintenance problems. Even modern refrigerators, microwaves, and other appliances have significant computing capabilities at their cores and often digital interfaces for the user (that's you!)

Computers are made up of hardware and software. The **hardware** of a computer are the physical, tangible components (or at least components you could touch if you opened the case) such as the **motherboard** which internal components of the computer plug into, the **CPU** (**central processing unit**), which is the brain of the computer that performs calculations, the **memory**, which is also known as **main memory** or **RAM** (**random access memory**) that is responsible for storing data and programs to be used by the CPU and other processors like the **GPU** (**graphics processing unit**.) There are many more pieces of hardware that are used by computers, including other

devices with which the computer interacts (called **peripheral devices**) such as printers and scanners. The **software** of a computer consists of the code that runs on the hardware. The software is the intangible part of a computer. In other words, you can't touch it (respect to M.C. Hammer.) In a desktop computing environment, software is generally categorized as either **system software**, such as the **operating system** and other utilities related to the running of the computer itself and **application software**, which generally consists of the programs the users run for information, entertainment, and productivity (like games, web browsers, word processors, etc.)

1.2 - PROGRAMMING

Computer scientists and software engineers are primarily concerned with **software development**, so the software is of primary importance. However, if you study computer science and software engineering for long enough you will find that you must learn some about the hardware as well, but it is not the focus of this book, or most programming books in general.

When we want a computer to perform a set of operations or accomplish a task, we need to tell it what to do. This is the essence of **computer programming**. In C++, we give the computer instructions in the form of an **algorithm**, which is a well-defined, unambiguous, finite set of instructions to perform a task. That might seem like a very formal sounding definition, but it's necessary. An algorithm must be clear (well-defined), it can't leave room for confusion (unambiguous) and it must eventually stop running (finite.) Computers need this type of precision. As long as there aren't major problems, a computer only does what it's told to do.

Although I won't belabor the issue, it's important to know why languages like C++ are important. Back in the day, if you wanted to tell a computer what to do, you had to directly speak *its* language, that is, what we call **machine language**. This type of language consists ultimately of the 1s and 0s (representing high and low electrical signals.) This code is very close to the hardware, and is extremely difficult to read and write. Because of these difficulties, programmers developed **assembly languages**, which use abbreviations that are closer to English or some other **natural language** (the term *natural language* is used extensively in computer science literature to distinguish spoken and written languages from programming languages.) Computers cannot understand assembly language directly (remember – they *only* understand their machine code.) Thus, the assembly language must be translated to machine code by a program called an **assembler**.

Assembly code might look something like the following:

LOAD NUM1
ADD NUM2
STORE RESULT

This is essentially taking a value in a **variable** called NUM1, adding that value to NUM2 and storing the result in the variable RESULT. This is much better than writing out the machine language instructions, but it is still tedious.

So, we've covered two broad categories of languages so far: machine language and assembly language. Note that machine languages are **machine dependent**. In other words, the code runs on the machines for which they were written.

After this brief overview, we can now talk about the category we're most interested in for our purposes: **high-level languages**. A high level language is much closer to English (or possibly another natural language) than machine languages or assembly languages. For example, to perform a similar calculation to the one we discussed in assembly language above, in C++, we would write:

```
result = num1 + num2;
```

In this one simple line of code we've done what took three lines in our assembly language example. This is obviously a very simple example. As you may imagine, sometimes a few lines of high-level language code could translate into hundreds of lines of assembly language code. High-level languages are much more readable and writable than are their assembly and machine counterparts.

Again, computers can't understand anything but their machine code, so some sort of translation program is needed to take C++ (or any high-level language) and convert it to machine code. A program that takes the source code of a high-level language and converts it to machine code is called a **compiler**. A compiler is a major part of an IDE, such as Visual Studio 2010.

As a note, many other high-level languages other than C++ exist. Examples of other high-level languages include Java, C#, Visual Basic, Python, PHP, Ruby, and Go. Obviously, many other languages exist.

1.3 – INTERFACE TYPES

There are many different types of programs that we could write in C++. A category of programs with which you are very familiar are **graphical user interfaces** or **GUIs** (pronounced *gooey*.) These programs usually use **event-driven** programming and wait on the user to interact with them by clicking or otherwise interacting with a **control** (also called a **widget**) such as a button or textbox. See ***Figure 1.3-1*** for the Calculator program that comes with Windows, which uses a GUI. Note that you interact with the application by clicking buttons, selecting menu options, etc.

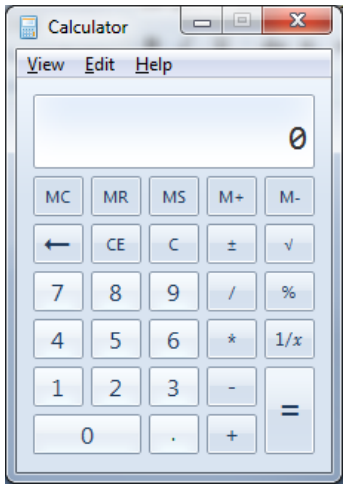


Figure 1.3-1: The Calculator Program

Although GUIs are great, a lot of system software, and software used to perform calculations and arguably, a large portion of C++ applications are not graphical in nature. Before graphical user interfaces and personal computing became popular, most programs were **command line interface** (CLI), also known as **console** programs. These programs allow the user to enter commands and data to the console. They process files and user commands without the need for graphical components. The commands are typically just special strings of text entered into the program from the keyboard, or file (or even sometimes databases and other sources.)

Now, before you get discouraged, C++ is used extensively in many graphical applications as well. It is well-known as a programming language used in popular video games and graphics applications. But the foundation of a computer scientist and software engineer's education is the essentials of a powerful programming language. Then, once these are mastered, you can venture into more complex territory with C++ such as graphics, complex database interaction, networking, etc. The sky is the limit. But let's learn to walk before we learn to run.

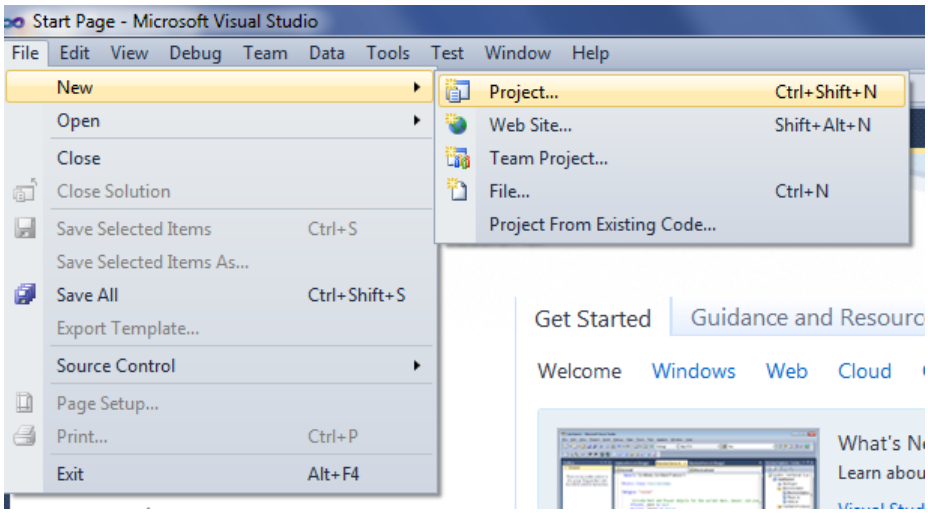
1.4 – OUR FIRST PROGRAM IN C++

Now let's get to the interesting stuff. Actually programming. Thanks for staying with me this long! The vast majority of code in this book should compile just fine on other platforms and with other IDEs. But, the examples in this book use Visual Studio 2010.

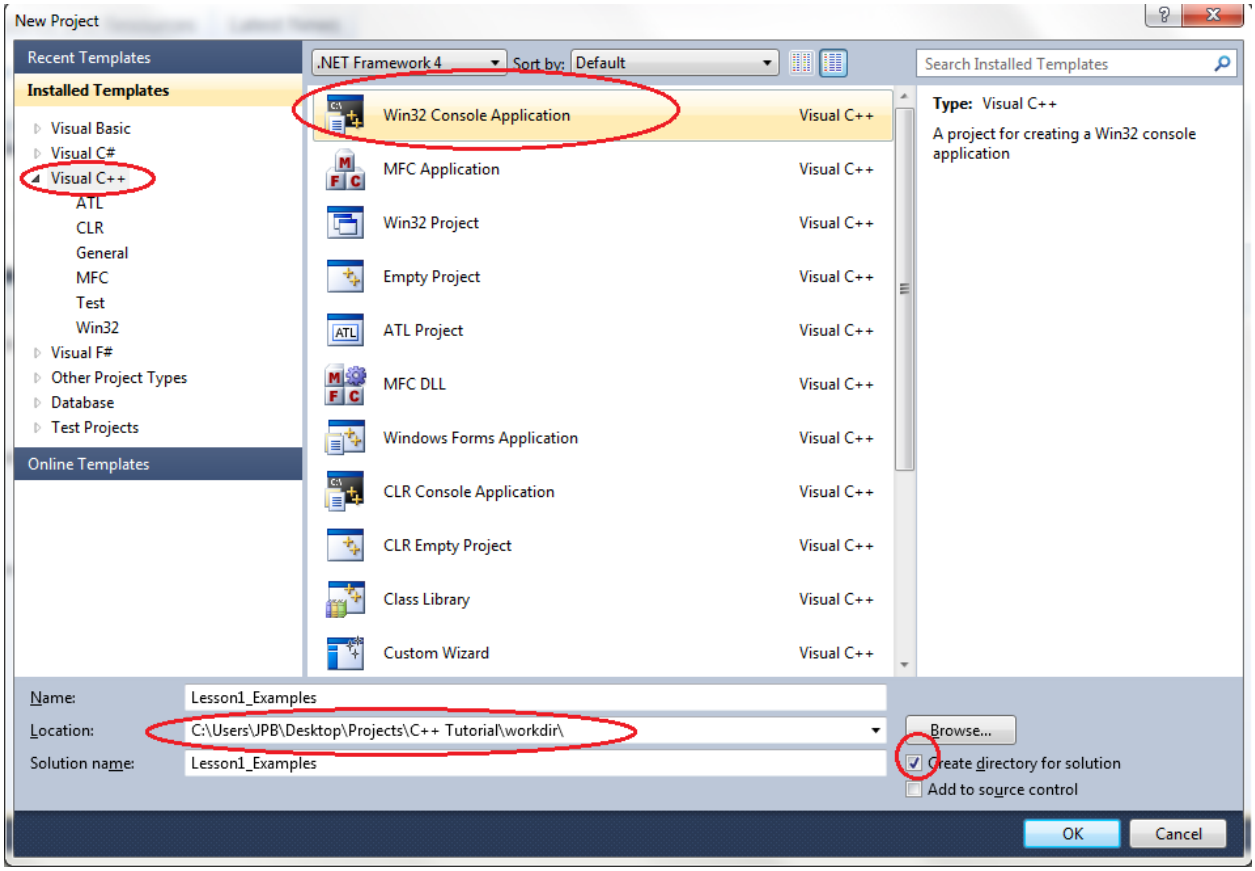
Let's get going. First we need to create a new **project** in Visual Studio 2010. Follow the following steps and you should be able to see the code work just great.

1. Start **Visual Studio 2010**

2. Go to **File→New →Project**...

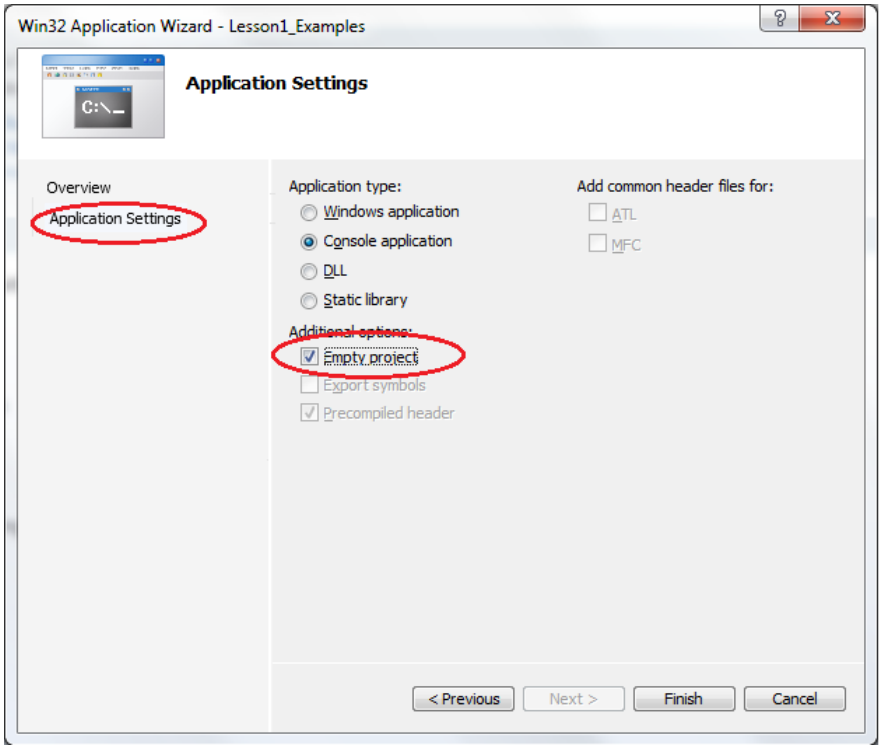


3. The **New Project dialog box** pops up. Make sure that you are under the correct language (Visual C++) and that you have Win32 Console Application selected.
   Note that you can use Browse... to select where you want your solution folder created. Keep note of where you put the solution. I like to keep my solutions organized so, I created a set of folders and subfolders to contain my code. If you don't change this location, the default Visual Studio 2010 location will be used. Also make sure you give the project/solution a useful name, and that you have *Create directory for solution* selected



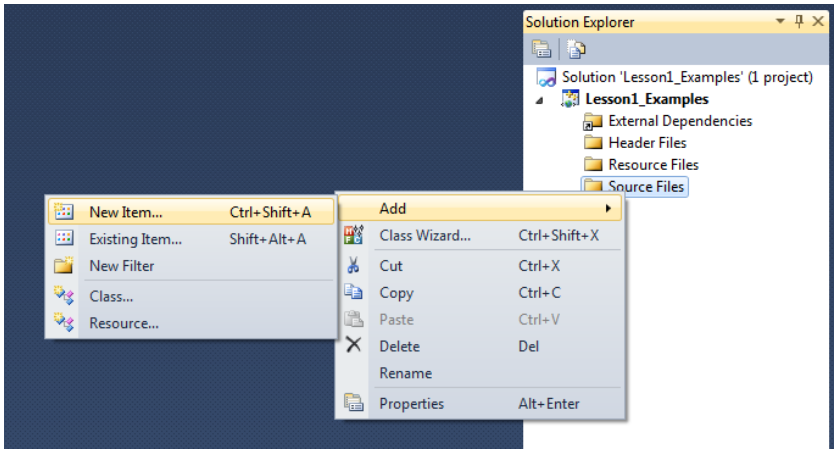
4. Click OK

5. The Application Wizard will pop up, and you should click on Application Settings and then check Empty project

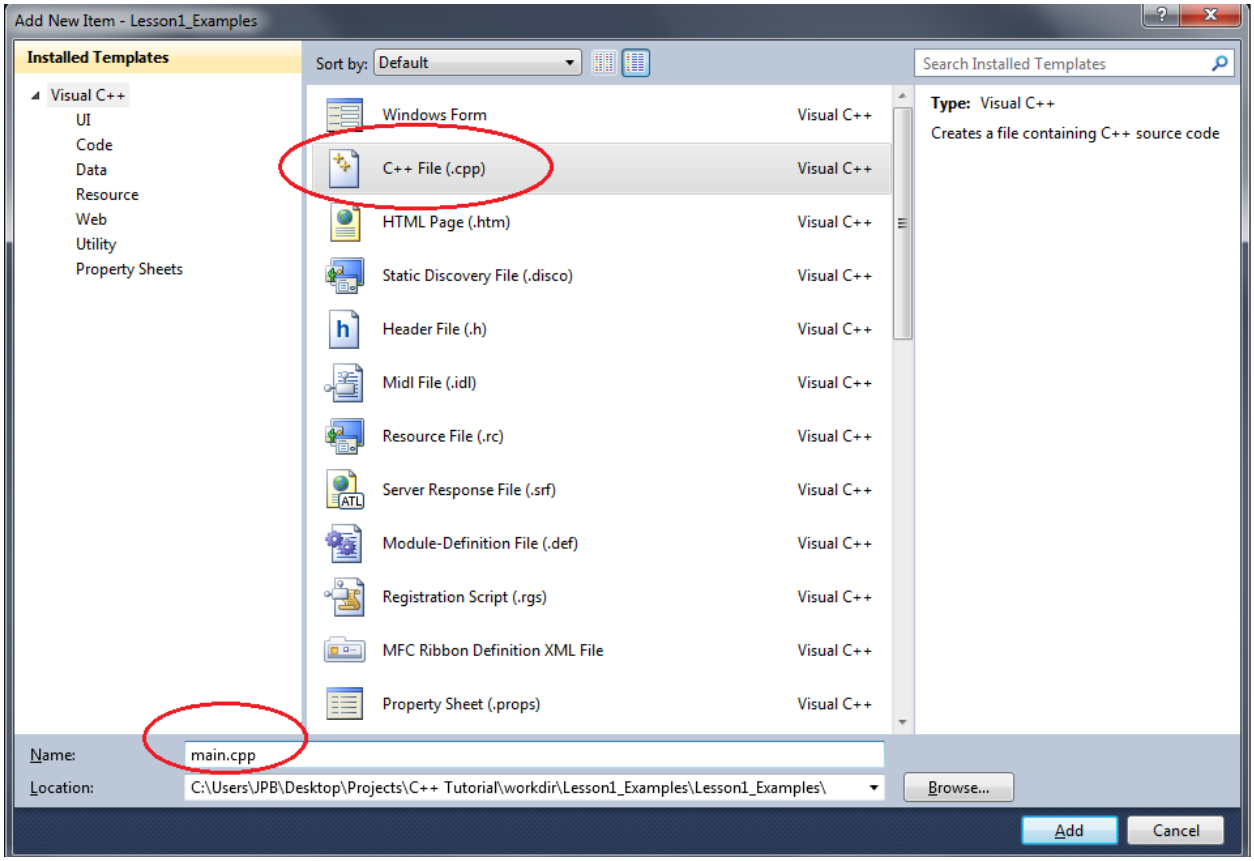


6. Click Finish

7. Now we have our empty project created. But we need to add a source code file so that we can actually type some code.
   You should see the Solution Explorer. Right-click Source Files, go to Add→New Item



8. Make sure that you have C++ File (.cpp) selected as the file type and then give the file a useful name. For example, I named mine main.cpp. You could call it lesson1.cpp or whatever you'd like as long as it is a legitimate file name.



9. Click Add

10. Now you will see that we have a main.cpp file under the Source Files in the Solution Explorer. Now we can actually start coding!

In the new file that is open, main.cpp, we can add the following code. Don't worry if you don't understand it right now. This is just to get your feet wet. We'll learn more together as we go along.

```
1  //Lesson 1:  Hello World
2  #include <iostream>
3  using namespace std;
4
```

```
5   int main()
6   {
7           cout<<"Hello World!"<<endl;
8
9           return 0;
10  }
```

Code Sample 1.4-1

The code in ***Code Sample 1.4-1*** shows quite a bit for being so short. The first thing, on line 1, is a **comment**, which is code that the compiler ignores. Comments are used to write reminders and other useful information in the code for the programmer to read (either yourself or another programmer that works with your code later on.) On the line 2 and 3, a part of the compiler called the **preprocessor** will include the **library** called `iostream`, so we can print to the console. Libraries consist of code that has been written so that it can be reused.

Writing even simple things to the console is actually quite complicated if you had to code everything from scratch. But luckily enough, the developers of the C++ language have provided libraries like `iostream` to take a lot of the difficulties away. The `using` directive, without getting into too many complexities, is there so that we can write statements like `cout` (on line 7) without telling the compiler what namespace it belongs to. Otherwise, we would have to write `std::cout` every time we wanted to use `cout`. For now, you can just write this on faith and know that it works.

Later on we see the header of the `main` function. The `main` function in C++ console applications is the **entry point** of the program. When you execute the program, `main` is the starting point.

Notice the syntax. It may look very foreign to you right now, but will become natural as you learn the language. The body of the `main` function exists on lines 6 – 10. This body consists of the `cout` statement on line 7, and the `return` statement on line 9. The body is delimited by the opening and closing curly braces, { and }.

I'll just state that the return statement on line 9 simply returns the value 0 to the operating system to tell it that there were no problems. We'll learn more about return statements later in this book.

The interesting part of the code is on line 7. This is where we actually wrote some code to print the character string "Hello World" to the console. You use the `cout`, pronounced "see out" variable, which means "console out" to indicate that it will print to the console. Then, it is followed by two less than signs (<<, known as the **stream insertion operator** if you'd like to impress your friends.) Then we have a **string literal**, which is a string of characters enclosed in double quotes.

Then, << again and the variable `endl`, which stands for **end line** and causes the insertion point to move to the next line.

So, what does the code do?

Let's build our project and then run the code and see what happens.

1. Go to **Build→Build Solution**
2. If there are any errors, you need to correct them, but there shouldn't be if you copied the code correctly.
3. Now, the code has been compiled into machine language in a format known as an **executable file**. Now we can run the program from within Visual Studio by going to **Debug→Start without Debugging** or by hitting Ctrl + F5 on the keyboard.

The output is as follows:

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

Output for Code Sample 1.4-1

The "Press any key to continue..." is actually added by the Visual Studio environment and is not part of the code we wrote. In fact, the only thing that we see as evidence that we wrote code is the "Hello World!" Printed to the console.

Congratulations are in order! You've written your first program in C++!

1.5 – SUMMARY

In this chapter, you learned about the basics of computers, hardware, software, and programming concepts. We explored the different general categories of programming languages, including machine languages, assembly languages, and high-level languages. C++ is a high-level language.

To convert C++ into machine language, we use a compiler. For our purposes, we learned to use Visual Studio 2010 IDE (integrated development environment), of which a compiler is a major part.

We learned how to print a text string to the console.

EXERCISES

1. Change the code in your Hello World example to print your name to the console
2. What happens if you remove the `<<endl` (make sure to leave the semi-colon `;`)?
3. What happens if you have `<<endl<<endl;` at the end of your string? How is it different from just having one single `<<endl;` ?
4. What happens if you remove the semi-colon, `;`, from the end of the `cout` statement line? What does the compiler do? Does it allow you to compile the program?