# CTI 134

# M1T1 (Module 1, Tutorial 1): “Hello World”

First off, welcome to CSC 134, C++ Programming. We will start off with the traditional programmer welcome: Hello World.

A common exercise when learning a new programming language is to write a program that prints “Hello World” as output. (Why that phrase in particular? No idea. But it’s tradition.)

This is a simple problem, but it confirms that the programming language is is installed properly and that you are able to run a program. In this tutorial we’ll look at “Hello World” in several languages, and then you will tell your computer to print “Hello World” yourself in C++.

As you work through this tutorial you will see sections marked

**TO DO:** Follow these instructions when they appear!

They indicate steps you will need to complete for this assignment.

This tutorial consists of two parts, **Background** and **Assignment**. You don’t absolutely need to read the Background to be able to do the Assignment … but it’s helpful to know **why** you’re doing what you’re doing.

# Part I -- Background

# Computer Languages

Computers don’t speak fluent English. Even if they did, human languages are not precise enough to give exact instructions to follow.

Computers speak **binary**, a series of ones and zeros. These binary digits (or **bits**) are put together to form larger numbers, and each number has a specific meaning to a particular CPU. This is called **machine code**. In the early days of computing, humans hand-translated instructions into a series of ones and zeros and then entered those values into a computer by flipping a series of switches on and off. The computer replied by flashing a series of lights on and off.

Later systems used number displays to save space, but the underlying ones and zeros were still there.

Here’s one way to write “Hello, world!” in machine code:

48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21 0a 00 2e |Hello, world!...|

This was a difficult system to use, so programmers developed **programming languages** that were easier for us to ‘speak’ in. As these languages became more complex, a distinction was made between **low-level languages** and **high-level languages**. The “lower level” a language is, the closer it is to how the computer thinks internally.

The first low-level languages were **assembly language**. Instead of binary numbers for instructions, assembly language uses **mnemonics** – short phrases easier for humans to remember. Instead of memorizing the special number used to tell a processor to add two values, you might type add. Moving a value from one place to another in memory might be mov. Programmers used a tool called an assembler to translate this language into binary.

With that in mind, here’s “Hello World” in 8086 Assembly as someone might have in the 1980s. (This and all other examples are from the **Rosetta Code** website.)

# Hello World – 8086 Assembly (IBM PC compatible)

DOSSEG  
.MODEL TINY  
.DATA  
TXT DB "Hello world!$"  
.CODE  
START:  
 MOV ax, @DATA  
 MOV ds, ax   
 MOV ah, 09h ; prepare output function  
 MOV dx, OFFSET TXT ; set offset  
 INT 21h ; output string TXT  
  
 MOV AX, 4C00h ; go back to DOS  
 INT 21h  
END START

While this is easier to read than a series of ones and zeros, it’s not very programmer-friendly. A programmer still has to write down each step the computer has to take, and they are very small steps. For example, here’s part of 8086 assembly program to add two numbers.

# Add two numbers – 8086 Assembly

; This program adds two numbers together and stores the result

ASSUME CS:CODE,DS:DATA  
START:  
 MOV ax, @DATA  
 MOV ds, ax  
 MOV al, a ; get the first number a   
 MOV bl, b ; get the second number b  
 ADD al, bl ; add the two  
 MOV c, ax ; save result in c  
   
END START

Six lines to say c = a + b, and that’s not even the whole program.

In addition, every processor had a different set of instructions, and this meant a different assembly language dialect was used on each. A 6502 (Apple II) spoke a different language than an 8086 (IBM PC).

# High-level languages

You can tell someone to “Go to the store and pick up some milk” without having to tell them every step like “open the front door”, “close the front door”, and “stop at red lights”. This saves a lot of time. **High-level languages** allow programmers to bundle a lot of small steps into a simple abstraction in the same way.

Instead of specifying each step of addition, you might write x = 2 + 2. Because computers only speak binary, another translation program is needed to convert this language into assembly and then into native machine code. These translation programs are called **compilers** if they convert the entire program at once, and **interpreters** if they convert it while the program is running. C++ uses the compiler method.

High-level languages aren’t English, but they allow you to think in English-sized steps.

***To recap:* High-level Language** is interpreted into **assembly language** which is assembled into binary **machine code**.

# Types of high-level languages

There are many high-level languages, but the list of most commonly used ones is pretty short – Java, Python, C++ (“C plus plus”), C# (“C Sharp”) are examples. Swift is a newer language used for iOS development. All of these languages are taught at FTCC.

This course will teach you the basics of programming in **C++ ,** a language created by Bjarne Stroustrup.

Before we get into C++, we’ll write a very simple program in **Python**, a language created by Guido van Rossum. We won’t run this Python program, I’m just showing it to you so you can see what high level code looks like before we get into C++.

So what does Python code look like?

# Hello World – Python 3

**print (**"Hello world!"**)**

Python is a lot easier to read than assembly. While it’s still not English, it uses commands that are close enough to guess at. To print “It’s only a flesh wound”, you can write:

print (“It’s only a flesh wound”)

To ask someone what their favorite color is (and save it in memory in a location called favorite), you can write:

favorite = input (“What is your favorite color?”)

In this course we will learn a few commands at a time, and build on them in each module to write increasingly complex programs.

C++ commands will look different from Python, but the ideas behind them are the same.

**Part 2 – Assignment**

# Writing your own “Hello World”

C++ has a little more setup involved than Python, but once you get going, it’s not much more complicated than what you’ve already seen.

We will use the online website <http://repl.it>.

**TO DO:** Go to <http://repl.it>. When you visit the site, click “New Repl”, and choose “C++”.

# Writing a Program

You’ll now see three windows in your browser.

On the left is the **files** window. We’re only working with main.cpp for now, nothing needed to do here.

In the middle is an **editor window**, which has some code already showing up in it. This window works like a standard text editor.

On the right is a black window with white text (it may be blank when you start). This is the **output window** where your program runs.

Let’s look at the sample code that repl.it provided.

#include <iostream>

int main() {

std::cout << "Hello World!\n";

}

Look, they already wrote “Hello World” for you. It looks a little different from Python, but you can guess that **std::cout** is something like **print**.

We will make some changes to this program, but we’re halfway done.

**TO DO:** Type the following code into your repl.it window where it says “main.cpp”, at the top of the file above the code already there:

// M1T1

// Hello World

// My Name

// The Current Date

Lines starting with // are **comments** – the computer will ignore them. Humans have to be able to read code to work on it, so comments are very important even though they’re not used by the compiler.

Replace “My Name” and “The Current Date”. This is a **comment header block** and all your programs must include them. Correct filenames and comment blocks are necessary to receive full credit for any assignment.

# Running the Program

**TO DO:** Using the button above the editor window, select **Run** (it looks like a Play button.) The output window will update with new output. Mine looks like this:

Hello World

**Making Some Changes**

Typing std::cout every time is a bit wordy. We can tell the C++ compiler that we want to assume we’re using the std library.

**TO DO:** After the line that says #include <iostream> add this line:

using namespace std;

And now remove the std:: from that cout line. Run the program again to make sure it works.

**Customizing the Program**

Let’s add one more thing: have the program greet you by name.

We’re going to need somewhere in memory to store your name so that the program knows it. This is called a **variable**. A variable that holds plain text is known as a **string** (like a string of letters and numbers).

**TO DO:** Add this code after your cout statement, but before the closing curly brace (That’s a } symbol).

string name = "Your Name";

cout << "Nice to meet you, " << name << endl;

Now run your program, it should print something like

Hello, World!

Nice to meet you, John Doe

(If it’s not showing your real name, change that line of code.) It’s worth mentioning that << is used to separate the things you’re printing, and that “\n” and endl both mean “new line”. (NOTE: This is “end-lower case L”, not “end-number one” – with the font it can be hard to tell.)

**Finishing Up**

If your program doesn’t run, go back and check for typos.

Here’s the full program to compare with yours.

// Comment block goes here

#include <iostream>

using namespace std;

int main() {

cout << "Hello World!\n";

string name = "Your Name";

cout << "Nice to meet you, " << name << endl;

}

Before you save your work, make sure that your comment block and the “Nice to meet you” line both use your actual name.

Finally, you’ll want to save your main.cpp file.

**TO DO:** In the left Files window, there are three dots next to a folder icon. Click the dots, and select “Download as zip”. This will download main.cpp as a Zip file. Rename this file to **M1T1\_lastname.zip**. (Obviously use your own last name.)

Finally, you’ll want to screenshot your work.

**TO DO:** You should **take a screenshot** of the repl.it window with your conversation with C++ now. Follow either set of steps:

Method One: Snipping Tool

You can also just upload the image by itself. To do that, use Snipping Tool, take a new snip of the browser window, and click the disk icon to save the file as something like **M1T1\_lastname.png**. (The linked video in Blackboard will show how to do this, or you can use Method Two, which is slower but easier to explain in text.)

Method Two: Copy / Paste + MSWord

This is a good backup if you’re having issues with Snipping Tool.

1. With your repl.it browser window selected, press **Alt + PrintScreen.** This will screenshot just the window and not your entire desktop. You can also use the Windows Snipping tool or another method you feel comfortable with.
2. Open MS Word and paste into a new document. Your screenshot will appear in the document.
3. Save this document with a filename in the format **M1T1\_lastname.docx** (replacing “lastname” with your last name). You should save the file somewhere you can easily find it when it’s time to submit your work, for example on your flash drive in a directory named CSC134.

# Submitting Your Work

In order to get full credit for the assignment, you’ll need to attach both files (your screenshot and the .zip file) and submit the assignment to Blackboard.

You can attach any number of files before you press Submit, and some assignments will require you to include multiple files in this way. Ask your instructor for assistance if you have issues submitting the assignment.