Writing your First C++ Program in Code::Blocks

Estimated Completion Time: 15 Minutes

Goal: The goal of these instructions is to create, write, and execute a C++ program in Code::Blocks.

Introduction
These instructions assume you have successfully installed Code::Blocks IDE (integrated development environment) as well as the GNU GCC Compiler on a Windows 10 computer.

It is a common tradition within the programming world that a programmer's first program is "Hello World". This program prints the text "Hello World!" to a window on the screen.

Successful completion of the program will display a window like this on the screen

TIPS:

As you go through the instructions, words in **BOLD** text are words on the screen that you will click on.

Anything in *ITALICS* is something you will need to type in.

If you are unable to read these instructions, go to www.website.com and download the PDF for these instructions

The instructions will be broken up into 3 parts

- Creating the Project
- Writing the Code
- Executing the Program

Terms to Know Before you Start

1. Project: (aka Program) a set of files required to run a C++ program

2. Compiler: A software tool that translates the code you write into something the computer can interpret. Understanding what a compiler does is not required for successful completion

3. Compile: Translates the code into the 0's and 1's that the computer will interpret

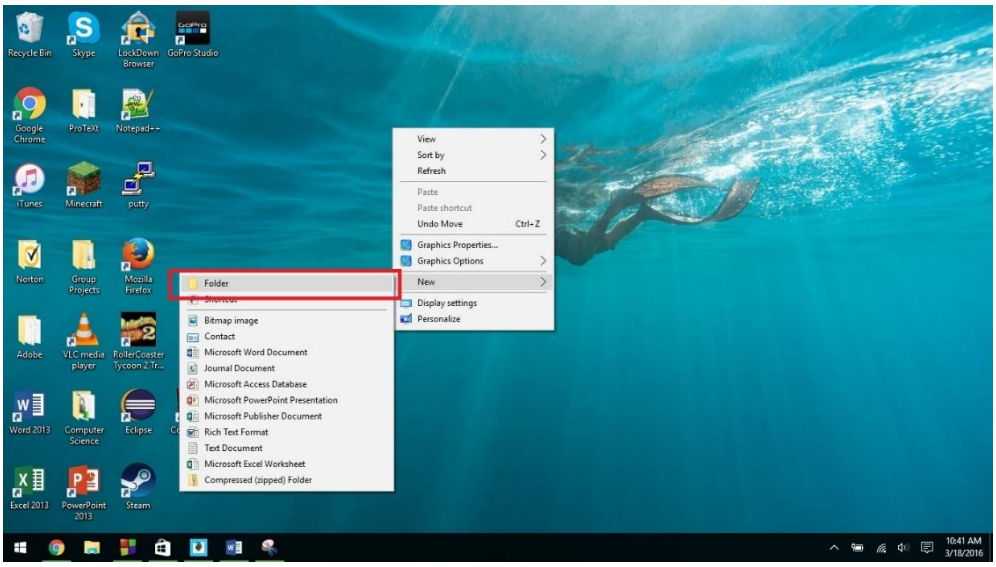
4. Execute/Execution: The process of compiling and running the program

5. IDE (Integrated Development Environment): A software that combines multiple steps in the process of creating executable code into one application. Understanding how the IDE makes an executable application is not necessary for successful completion

Creating the Project

1. On your desktop, create a new folder by right-clicking anywhere on the desktop, then clicking **New** -> **Folder** (Fig 1). Name the folder *Hello World*

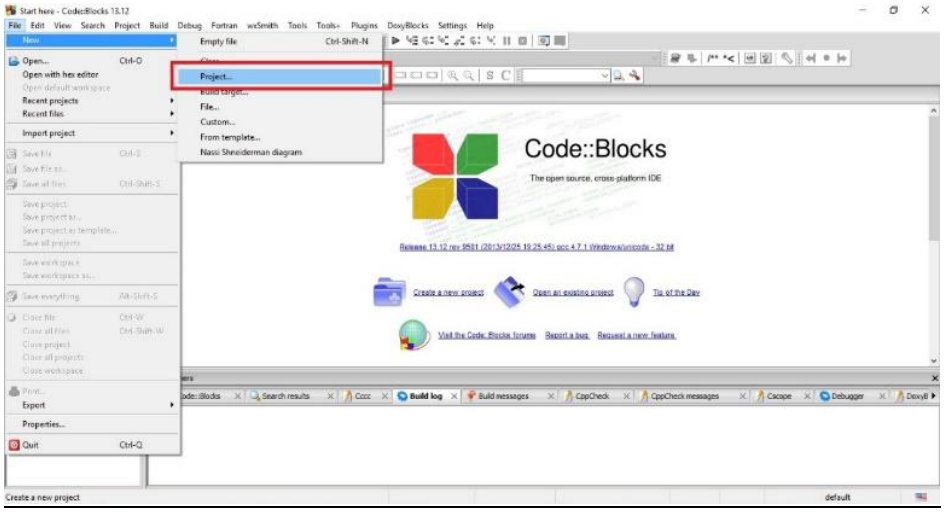
Figure 1



2. Double click the Code::Blocks icon  on the desktop which opens the application

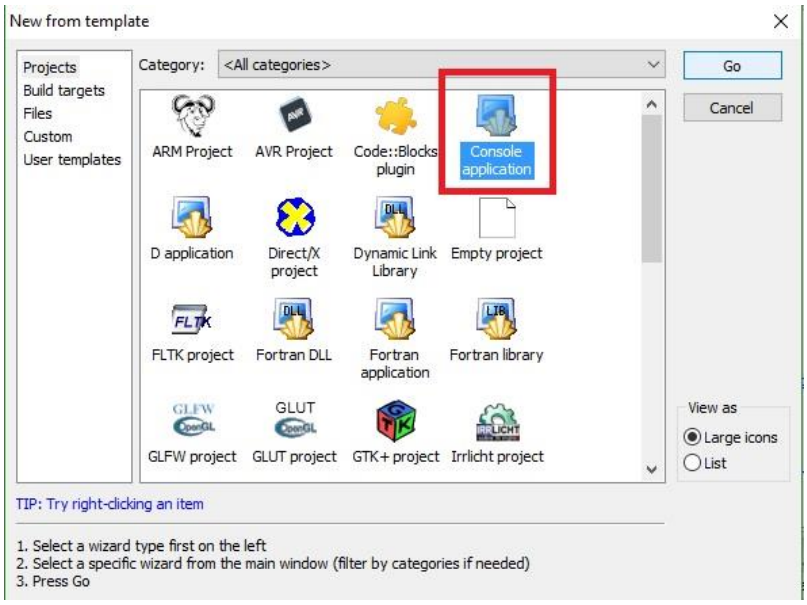
3. Once the application opens, open the new project window by going to the top left part of the screen and clicking **File** -> **New** -> **Project** (Fig 3)

Figure 3



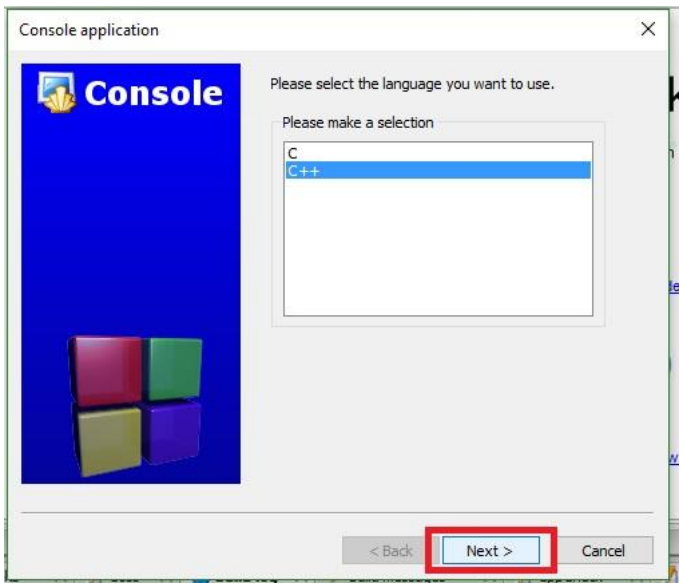
4. Once the new project window pops up, click the top right icon called **Console Application** then **Go** (Fig 4)

Figure 4



5. Make sure C++ is highlighted in blue, then click **Next** (Fig 5)

Figure 5

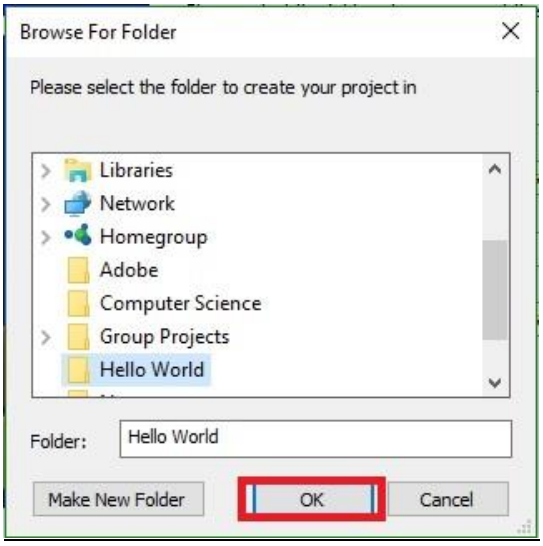


6. On the next screen, in the “project title” section, type *Hello World*

7. Then in the **Folder to create project in** section, click the ... icon on the right

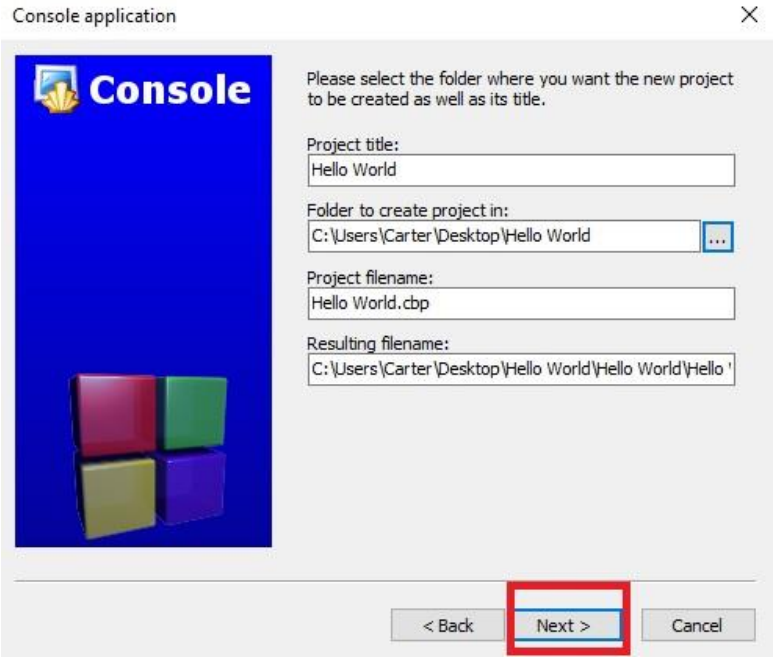
8. Navigate to the folder by going to **Desktop** -> **Hello World**. When **Hello World** is highlighted in blue, click ok (Fig 8)

Figure 8



9. Once everything looks like Figure 9, click **Next**

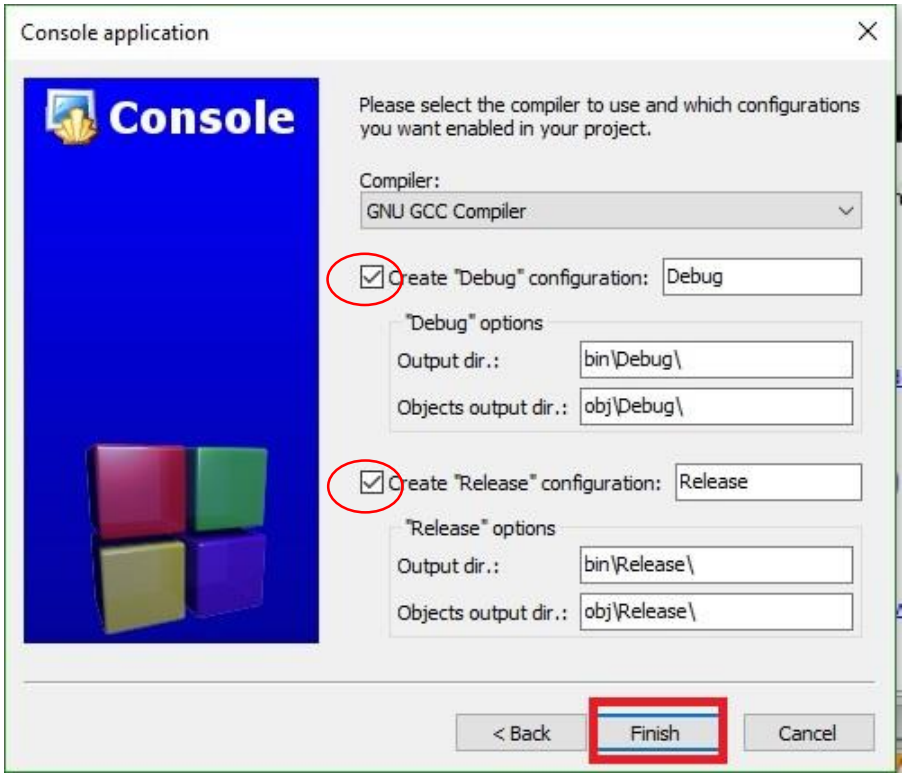
Figure 9



10. On the next window, make sure the compiler says “GNU GCC Compiler” and the “Create ‘Debug’ configuration” and the “Create ‘Release’ Configuration” boxes are checked, then click **Finish**. Your screen should look like Figure 10 before clicking **Finish**

Note: In most situations, the boxes will already be checked. You will likely just need to confirm that they are checked

Figure 10

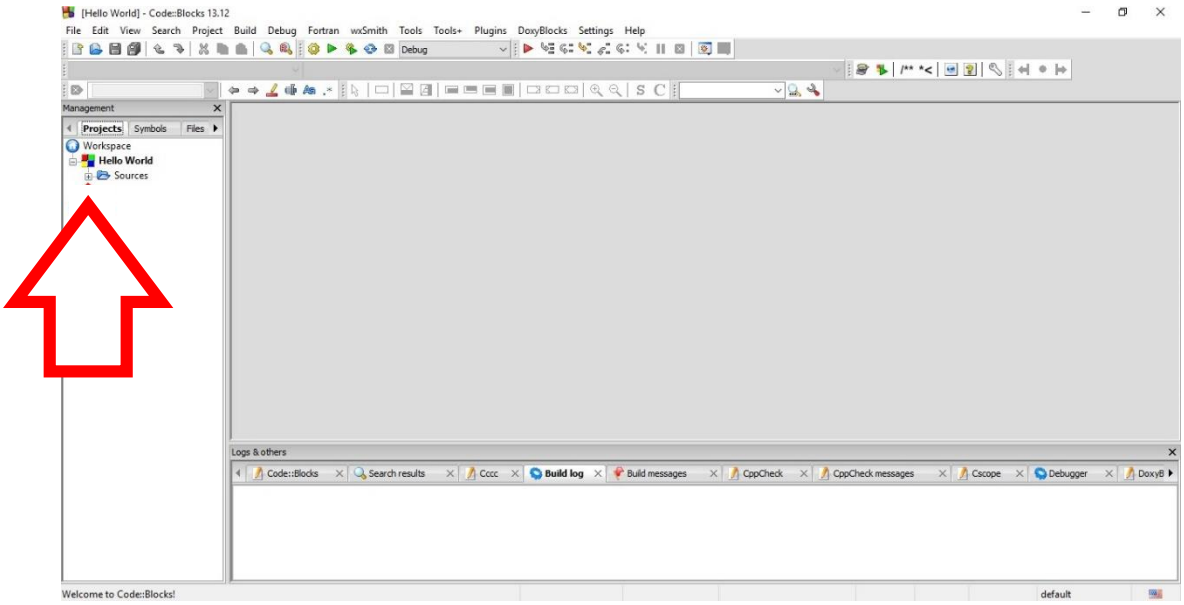


You have now created the project! Next we will write the code for Hello World

Writing the Code

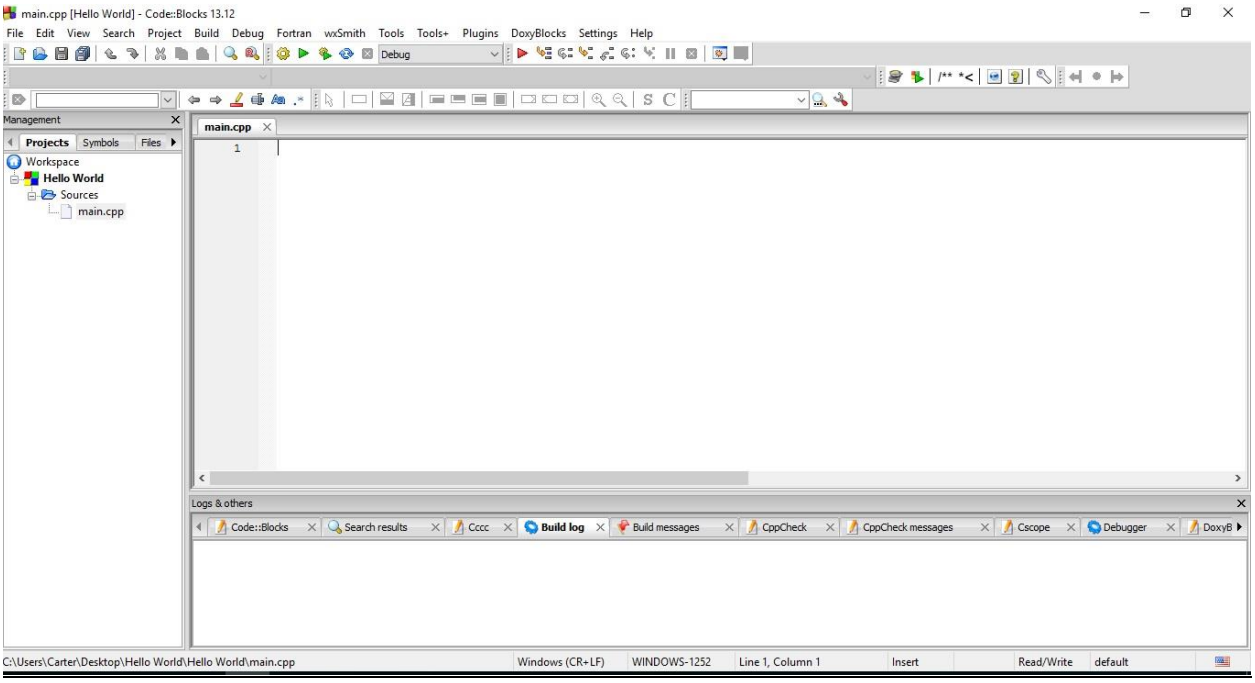
11. On the left side of the screen, click the plus icon to the left of the folder icon named "Sources" (Fig 11)

Figure 11



12. Double click the file called **main.cpp** and the file will open on the screen (Fig 12)

Figure 12



13. Type the text **EXACTLY** as seen in Figure 13 below. Below Figure 13 is a brief explanation what the code means.

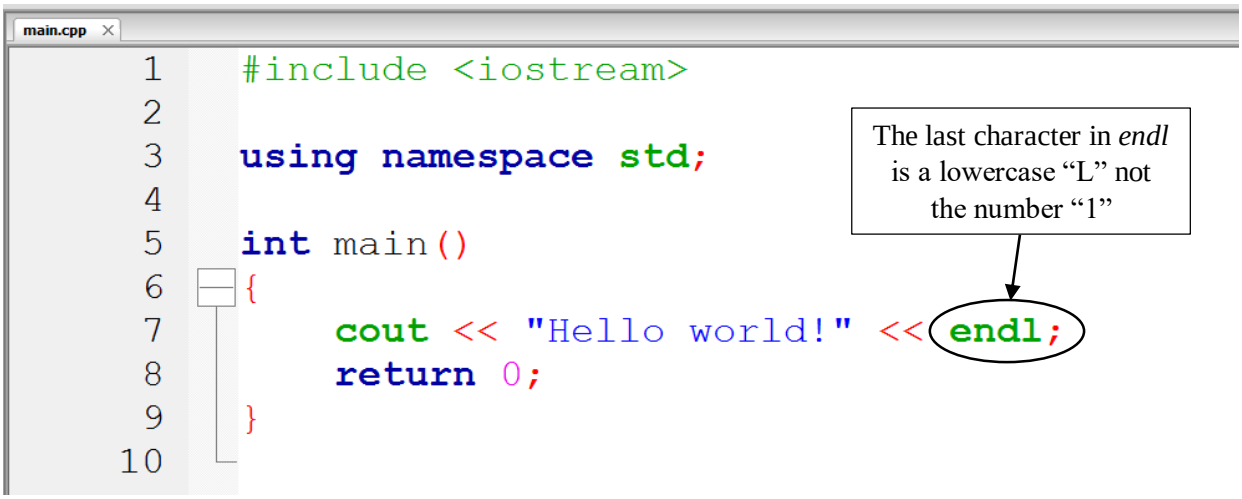
Notes: *The code is case sensitive. Use the same capitalization as seen in Figure 13*

To move to a new line, press the "Enter" key

When typing "(" or "{", Code::Blocks will automatically place a ")" or "}"

You do not need line 10 (the line after the "}")

Figure 13



What Do These Words Mean?

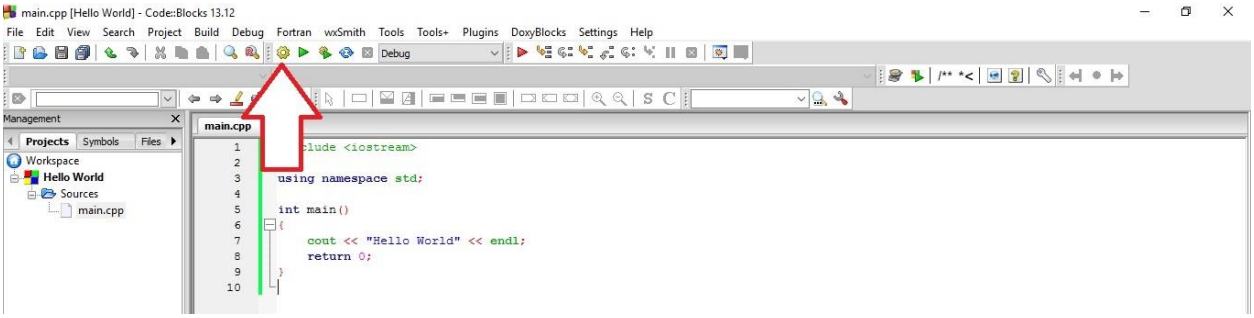
- Line 1: This imports a library (code written by other people) so that you can interact with the console window
- Line 3: This tells the computer that you are using the standard version of each word in the code
- Line 5: The word "main" tells the computer it needs to run the part of code in the curly braces "{}" when you run the program
- Line 7: This is the part of the code that prints the text to the console
  - cout: Console OUTput, tells the computer to write the next text to the console
  - "Hello World!": The text inside the quotes is what is written to the screen
  - endl: This is the end of the line (the equivalent of hitting "Enter" after typing)
- Line 9: Tell (or return to) the computer that there are 0 errors running the program

Now that you have written the code, we are going to execute the program

Executing the Program

14. In the top part of the screen, find the icon that looks like  and click that icon (Fig 14)

Figure 14

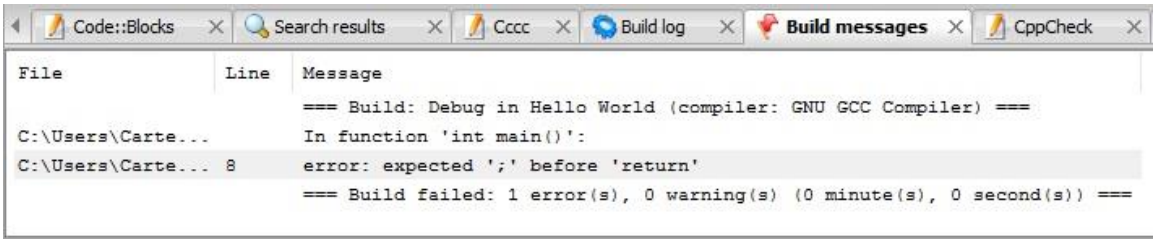


15. Look at the 2 figures below and compare them to the bottom of your screen. More specifically, look at the last line of output generated by the complier.

Note: *The last line of text in the window of the bottom of the screen will say how many errors the compiler finds. 0 errors is successful compilation. It is possible to have more than 1 error*

Figure 15a: Successful Compilation

Figure 15b: Error Example



- If it looks similar to Figure 15a, your code was successfully complied.
  Skip ahead to step 17.

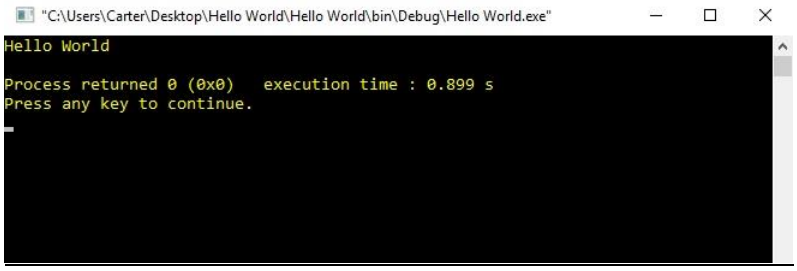
- If you see a screen similar to Figure 15b, go to step 16. There is an error in your code.

16. Check to make sure your code looks **EXACTLY** like Figure 13 on page 7, and click icon again. Continue this step until the bottom of your screen looks like Figure 15a, indicating no errors

Note: The error message will attempt to guess what is wrong with the code and give you a line number to check. This is not always correct, but it can give you a starting point for your errors

17. Click the icon which is located just to the right of the icon. A pop up window will display "Hello World" plus information below about the execution of the program (Fig17)

Figure 17



18. Exit the console window by clicking the "x" in the top right part of the window

19. Congratulations! You have written your first C++ program. If you are finished, continue to step 22. If you would like to play with this program some more, continue to step 20

20. Modify the blue text in between the quotation marks on line 7 and type whatever you want.

Note: If you attempt to put quotation marks or the character "\" (backslash) in between the quotation marks you will very likely get errors. Also do not hit the Enter key when typing between the quotes

21. Once you have modified the text, go back to step 14 and perform steps 15 – 20

22. Once you are finished working, you can exit out of Code::Blocks by clicking the X in the top right part of the screen. Your code is saved each time you compile, so you do not need to save the file

If you have any problems or would like to continue learning about Code::Blocks, visit wiki.codeblocks.org for more information.

To: Sara Kelm
From: STUDENT
Subject: Usability Test Report
Date: 4/1/16

The purpose of this document is to report the process of writing my instructions, and then revising them based on findings through usability testing.

Summary of Test Findings

The purpose of this project was to learn how to effectively write instructions and perform usability tests. This, in turn, would teach us how to revise our instructions based on feedback from users. The main goal of the usability tests was to figure out the strengths and weaknesses of the instructions I wrote. This allows the writer of the instructions to see how well people comprehend the instructions. The writer also learns where people have difficulties understanding the instructions and performing the tasks properly.

For the first round of usability testing, Professor Kelm was my tester. My second round of testing was performed by Alan Zhang and Matthew Santiago. I also performed another usability test with my friend Allison Aguirre.

The main trends I discovered were that the users could successfully complete the task on their own with minimal issues. One problem I discovered was that there was initial confusion when users read step 15. However, after re-reading that step, they were able to figure out what to do. Also, users had some slight confusion on small parts of writing the code, but they were able to write it successfully.

Instruction Overview

The instructions required users to create, write, and execute a C++ program using the Code::Blocks IDE (Integrated Development Environment). These instructions were designed for people that have experience using a computer and have already successfully installed Code::Blocks and the GNU GCC complier on a Windows 10 machine. People likely reading these instructions would be interested in programming, and they are doing these instructions to learn how to do basic programming in this IDE. The users would likely be in a low-stress environment where finishing quickly is not a necessity. Users would also be more focused on doing everything exactly right, instead of doing it quickly

Because of this, the instructions have screenshots for just about every step. Every screen, window, and icon has a picture associated with it. This allows for users to visually confirm what they read in the step. Also, there are definitions and explanations for certain terms throughout the instructions. People reading these instructions will likely not be knowledgeable in programming terms, but they would want to know what certain words mean.

Methods

In usability testing, testers performed the instructions based on the way they were written. As mentioned earlier, this allowed me to see where users had problems understanding the instructions. My first tester was Professor Kelm. She does not have any programming experience, so she could not skip steps or assume things based on prior knowledge. In between rounds of in class testing, I had Allison Aguirre test my instructions. She has about one hour of programming experience, so she wasn't able to interpret much based on prior experience either. For the second round of testing, I had Alan Zhang test my instructions. He is an electrical engineer and has some experience in programming and the Code::Blocks IDE. I also had Matthew Santiago, a computer science major, test my instructions. Both testers had no problem performing my instructions. Unfortunately, because of their prior experience, I couldn't determine if prior experiences aided them in performing the instructions. This is one of my major limitations in my project. Having two testers with experience in my instructions limited my number of completely valid tests.

Discussion of Findings

I noticed that most people were able to perform the task successfully in 10-12 minutes. The longest step was writing the code. This isn't a surprise and there isn't much that could have been done to change this. The step was already broken down into the smallest possible action without breaking it up into too many pieces. I noticed that most people would have slight confusion after reading a step, but that confusion would go away when they saw the figure associated it. This justified including all the figures in my instructions.

Also when writing the code, I only had 40% of the people encounter a compile error. This was lower than I expected. I would also classify this error as a "minor error" instead of a "fatal error" because typos are very common, and typing code is very different from writing sentences. Because of this, I included an error correction step within the instructions. In my testing, Allie went through the steps in dealing with a compile error and was able to fix her code. Alan used his prior knowledge of programming to fix his error. Because of that, I was unable to figure out if my step 16 was clear. Nobody had any other errors that would be considered a fatal error.

Revisions

The feedback I ended up receiving was mostly about minor things. I would attribute this to the fact that I spent a lot of time making sure the first draft of my instructions was thorough. For example, 2 people wondered if a character they were typing out was the number one (1) or a lowercase "L". This encouraged me to put a note clarifying the character. Also, a couple people suggested that I have a note saying the "Enter" key moves to a new line. Once again, these are minor things, but if 2 people suggested it, it seemed important enough to make a note about it.

My other biggest area of feedback came with step 15. As I mentioned earlier, there was some initial confusion when users read that step. Initially, I chose to reword it based on Professor Kelm's usability test. I modified that step so that users were supposed to look at both of the

pictures before comparing their output. After that modification, I had other problems with the step. Matthew said that there were too many words in that step and it needed to be broken up. So for my next revision, I had the comparing part in one paragraph, then had two bullet points telling the users what to do next below the figures. This allows the users to read the comparison statement, move down the page to see the two figures they need to compare the output too, then continue moving down the page to figure out where to go from there. I also changed the word "continue" to "skip ahead" so that users understand that they will be skipping/not performing step 16.