Chapter

1

Introducing C++

This is the Course Reader for CS 150, C++ Programming I, at Orange Coast College. CS 150 is a **second-semester** course for majors who have already taken a course covering variables, input and output, calculations, loops, decisions, functions and arrays.

The reader is short and concise, in line with Leonard Elmore's advice: "leave out the parts readers skip." It focuses on the parts you need to get your work done.

To get more details, consult one of these highly recommended texts.

- **The C++ Primer, 5th Edition. Lippman**, Lajoie and Moo
- **The C++ Programming Language, 4th Edition**, Bjarne Stroustrup

These are **professional** books to keep on your shelf after the course is over. Throughout the text you'll also find links pointing to additional reading, which is **optional**.

The C++ Programming Language

C++ is a **compiled**, high-level programming language. Compilers produce **native machine code** programs, which run directly on your hardware, without an interpreter (like Python) or intermediate software, such as the Java Virtual Machine. Even though C++ programs are often more efficient than their Java or Python counterparts, certain kinds of errors cannot easily be detected, so you will find that you must be more careful and detail-oriented when writing programs in C++.

C++ is also a **standardized language**, specified by the ISO, or International Standards Organization. Unlike **proprietary languages**, controlled by a single company, **anyone** may implement the C++ language without fear of lawsuits. There are three versions of C++ you should know about.

- **Pre-standard C++**: 1980-98. Often incompatible versions, vendor specific.
- **C++ 98**: the first standard version; it was updated in 2003 (**C++03**).
- **C++ 11**: the second **major** standard; it was updated in **C++14** and **C++ 17**.
- C++20, the latest major standard, is now **complete**. Most recent compilers offer a sampling of some of its new features.

In this class we will be using **C++ 17**. Compilers that implement C++17 include **Visual Studio 2019**, GNU **g++ 7** (or later), and **clang 6**+.

Hello World

On the first page of Dennis Ritchie's seminal programming book, The C **Programming Language,** he writes:

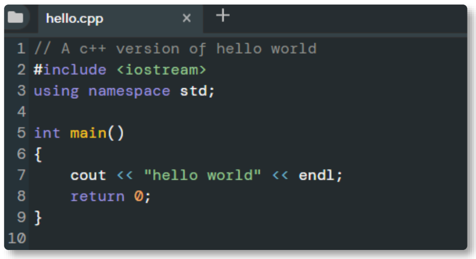
The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages:

Print the words: ***`hello, world`***

This is the big hurdle; to leap over it you have to be able to create the program text somewhere, compile it successfully, load it, run it, and find out where the output went. With these mechanical details mastered, everything else is comparatively easy.

I'm sure his last sentence was **somewhat** tongue-in-cheek, but let's follow his example and look at a C++ version of "Hello World" by clicking the little "running-man" to your left. Leave it open for the next few pages. (You must be logged into your GitHub account)

The Source Code



```
1  // A c++ version of hello world
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      cout << "hello world" << endl;
8      return 0;
9  }
10
```

When you click the link and open the CS50 Sandbox, you'll find the human-readable **source code** for this program, in the file named **`hello.cpp`**.

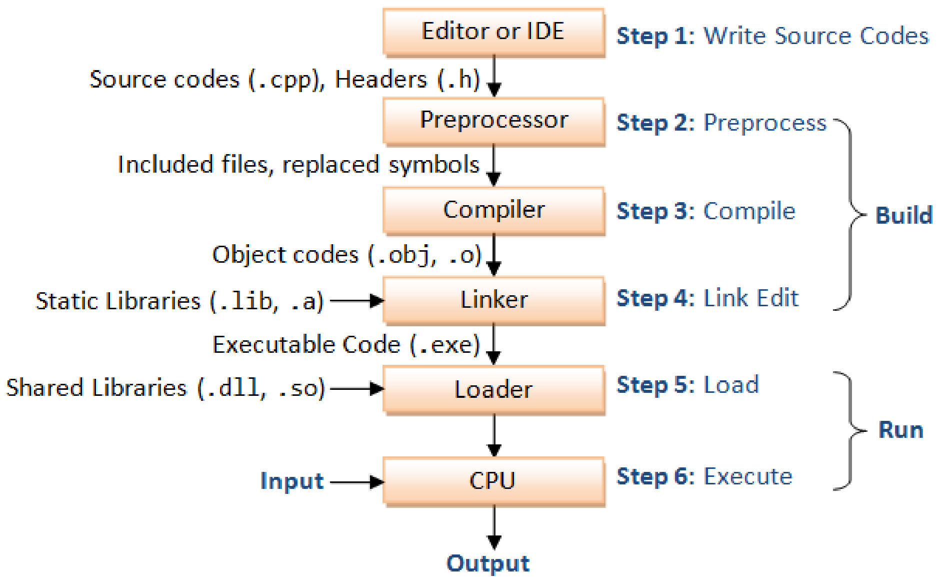
The **.cpp** extension is the **naming convention** that we will adopt. Let's review each line:

1. A comment, designed for the reader, but ignored by the compiler.
2. A preprocessor directive which makes the C++ library facilities available.
3. A namespace directive makes standard names available without qualification.
4. A blank line, used to make your code more readable.
5. The **`main`** function or entry point to your program. Each program has one **`main`**.
6. The **opening** brace. How you say begin the actions in C++.
7. Prints to the console, using the standard output object named **`cout`**.
8. Ends the **`main`** function, returning the value **`0`** back to the operating system.
9. The **closing** brace marks the end of **`main`**, matching the opening brace.

Program Mechanics

To create your own C++ programs, you follow a mechanical process, a well-defined set of steps called the **edit-build-run cycle**. In short you:

1. Create your **source code** using an editor
2. Build your executable using a **compiler toolchain**
3. Run your program using your operating system facilities

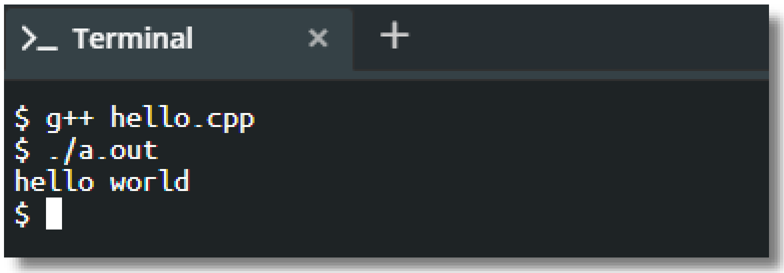


The Build Process

Once you have written your source code, the **build process** turns that source code into an executable. It involves several tools: (this is a more complete list than show above)

- **Preprocessor**—performs text substitution on your source code.
- **Compiler**— generates **assembly code** from the preprocessed source code.
- **Assembler**—converts assembly code into **object** or **native machine code**.
- **Linker**—combines machine-code modules into an **executable** that runs.
- **Make**—provides instructions for **building** each program.
- **Loader**—reads the executable from disk into memory and starts it running.
- **Debugger**—runs your program inside a controlled environment.

A **combination** a C++ compiler, linker, assembler and libraries, is known as a toolchain. The toolchain we'll use for this course is Harvard's **CS50 IDE**. The **CS50 Sandbox** (which you're using now), contains the same toolchain.



```
$ g++ hello.cpp
$ ./a.out
hello world
$ 
```

To use these tools on the **hello world** program:

- Switch to the **Terminal** window (or shell).
- Type `g++ hello.cpp` and press `ENTER`.
- This executes the build process.
- Type `./a.out` which triggers the run phase.

The name default Unix name, `a.out`, stands for assembled output.

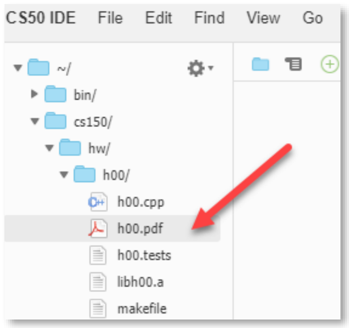
The tool `g++` is often called a compiler, but in fact, it is a **driver program**, which runs the different tools in the build process for you, one after another. If you like, you may use the alternative driver program, `clang++`, which often provides better error messages when things go wrong.

Perhaps even better, you can use the **make** program. Type `make hello` in the **shell** and it will run all of the tools, producing the executable output executable named `hello`. (Unix programs typically do not have an extension, like .exe.)

You can **run the program** by typing `./hello` in the shell. Unlike Windows, the current directory (or folder) is **not** in the executable path. The `./` means "run the `hello` program in the current directory; not some other `hello` program.

The CS 150 Homework

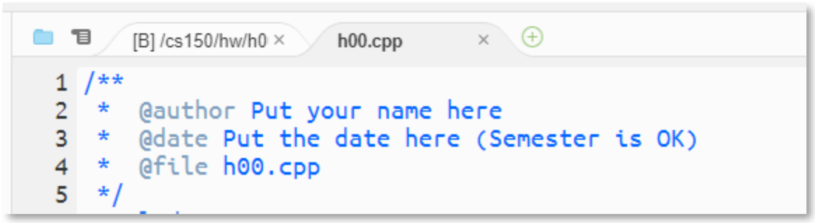
You'll complete your homework using the CS50 IDE. You will set up and configure your IDE in class. If you have not, then contact your instructor.



For each assignment, I'll provide you with a set of "starter files" which provide a framework for running and testing simple C++ programs.

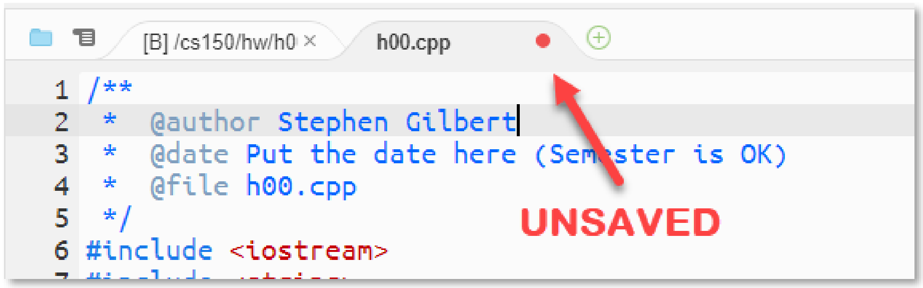
Open the folder for each homework problem by clicking the **arrow to the left of the folder**, to expand its contents. In the screenshot, the `cs150` folder has been expanded, as well as the `hw` folder it contains, and the `h00` folder with its contents.

Open the source code files (`.cpp` and `.h`) by double-clicking the icon in the file explorer. Each file will open in its own tab.



```
/**
 *  @author Put your name here
 *  @date Put the date here (Semester is OK)
 *  @file h00.cpp
 */
```

As you edit your source code, save each file before trying to compile, run or test.



```
/**
 *  @author Stephen Gilbert
 *  @date Put the date here (Semester is OK)
 *  @file h00.cpp
 */
#include <iostream>
```

To save your file, press **`CONTROL+S`**, or choose **`File->Save`** from the menu.

The Building Directory

Once you've entered and saved your source code, you're ready to **compile** and **link** it. This is called building the program, and is done by running a program named **`make`**.

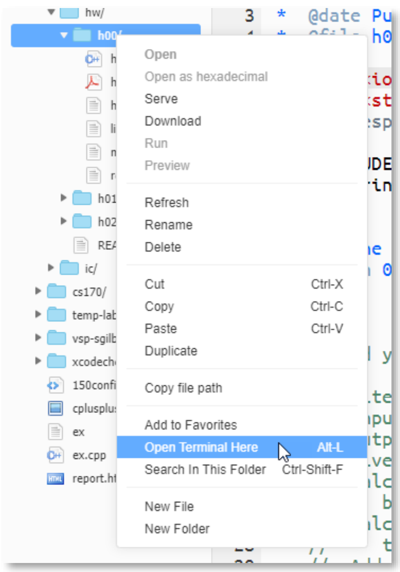
Before you can do **this**, you **must be in the correct folder**. This is called the current working directory.

You may use the UNIX command **`cd`** to switch to the correct folder. Assuming you are working on **`h00`**, the command would look like this:

```
cd
cd workspace/cs150/hw/h00
```

In the CS50 IDE, though it is probably easier to right-click the homework folder, and select Open a Terminal Here as shown in the screenshot on the right.

If you have your source code file open in the editor, you can just use the short-cut: **`ALT+L`**. That's what I usually do.



When you open a new terminal (as opposed to use the terminal at the bottom), it opens a new tab which hides your source code.

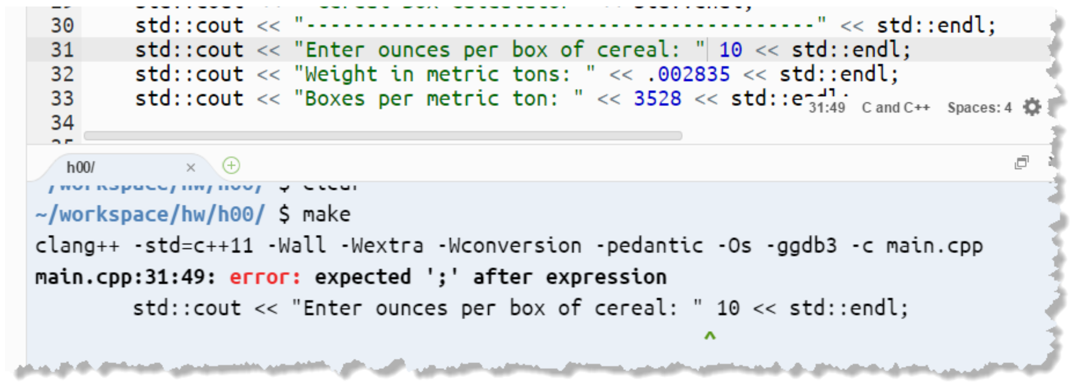
- You may right-click the new terminal tab and choose: **Split Pane in Two Columns** to move it to the right.
- You can **drag** the new terminal and **drop** it on the Console pane at the bottom.

Building Your Program

The `make` program is responsible for building your code. Switch to the terminal, type `make` and then press `ENTER`. This command **compiles** and **links** your program. If all is well, the prompt will reappear and there will be no other output.

No! Wait! Something Went Wrong!!!!

Of course it's possible that your code didn't compile successfully.



```
30     std::cout << "-----------------------------------------" << std::endl;
31     std::cout << "Enter ounces per box of cereal: " 10 << std::endl;
32     std::cout << "Weight in metric tons: " << .002835 << std::endl;
33     std::cout << "Boxes per metric ton: " << 3528 << std::e
34
```

```
~/workspace/hw/h00/ $ make
clang++ -std=c++11 -Wall -Wextra -Wconversion -pedantic -Os -ggdb3 -c main.cpp
main.cpp:31:49: error: expected ';' after expression
        std::cout << "Enter ounces per box of cereal: " 10 << std::endl;
                                                        ^
```

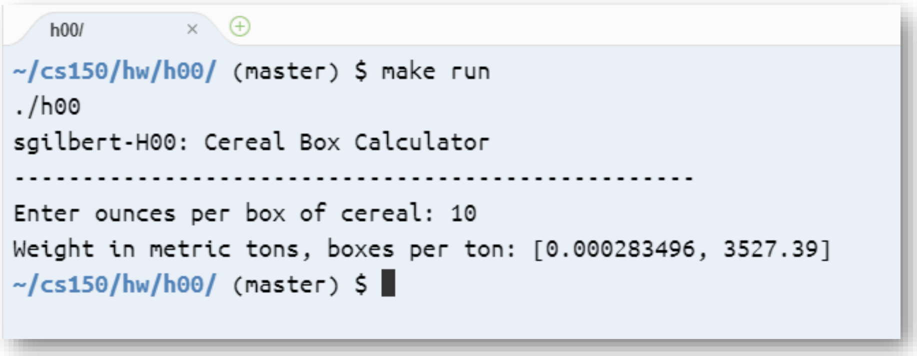
Syntax or compiler errors occur when you have broken one of the grammar rules of C++. If that occurs, you'll see an (often inscrutable) error message in the output pane instead. Follow these instructions exactly to fix them.

1. Scroll up to the first error that appears in the output window and make note of the file name and line numbers.
2. The second line of the error message attempts to show you where the compiler got confused. In this case, it is right before the literal number **10**.
3. Go to the text editor and **fix the problem** and them save and build again. This is where things get tricky. The compiler **doesn't know** what you **intended** to write, so the solution it recommends is often incorrect. The actual solution in this case is to add the output operator (`<<`) which we've forgotten.

Once you've corrected and saved your source code, build again to see if you've fixed the problem. You **can't go onto the next step** until there are no errors.

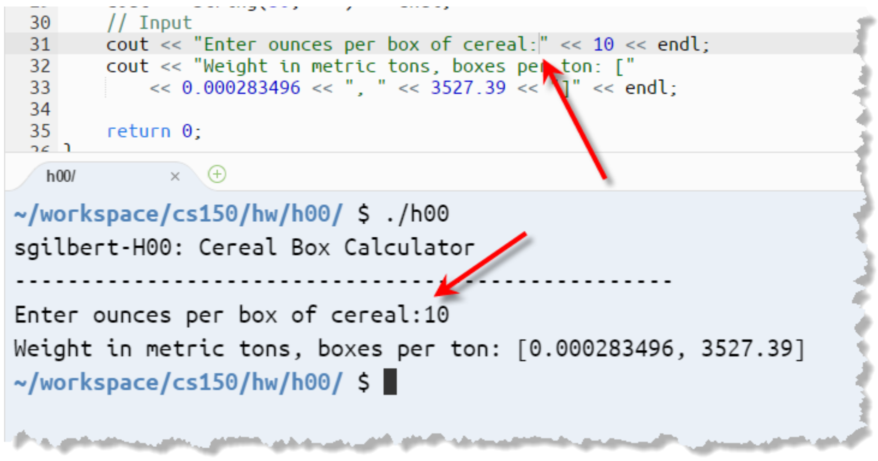
Running Your Program

Type make run and press **ENTER** to **compile**, **link** and **run** your program.



```
~/cs150/hw/h00/ (master) $ make run
./h00
sgilbert-H00: Cereal Box Calculator
--------------------------------------------------
Enter ounces per box of cereal: 10
Weight in metric tons, boxes per ton: [0.000283496, 3527.39]
~/cs150/hw/h00/ (master) $ 
```

Logic errors or **bugs** occur when your program **doesn't do what it is supposed to do**. If the output of the program looks like this that shown below, then you have a **logic error**, because you removed a space that was intended to appear in the output according to the homework specification.



```
    // Input
    cout << "Enter ounces per box of cereal:" << 10 << endl;
    cout << "Weight in metric tons, boxes per ton: ["
        << 0.000283496 << ", " << 3527.39 << "]" << endl;

    return 0;
```

```
~/workspace/cs150/hw/h00/ $ ./h00
sgilbert-H00: Cereal Box Calculator
-----------------------------------------------------
Enter ounces per box of cereal:10
Weight in metric tons, boxes per ton: [0.000283496, 3527.39]
~/workspace/cs150/hw/h00/ $ 
```

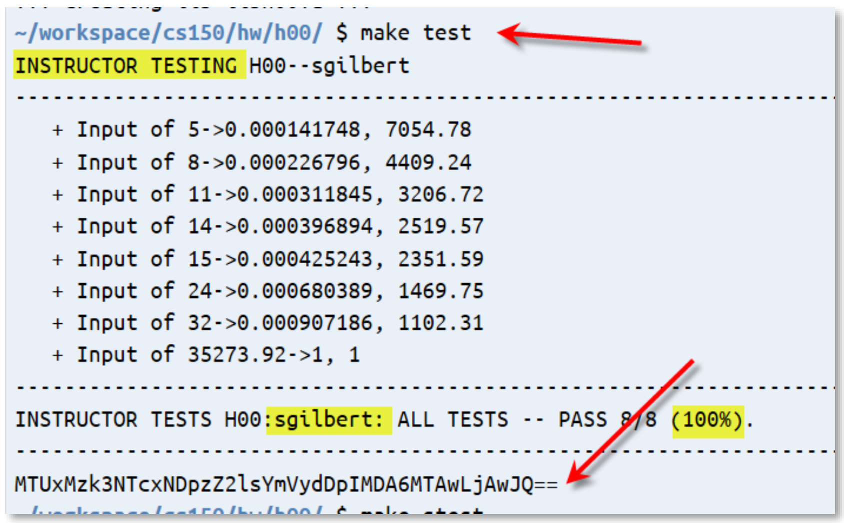
Testing Your Program

To see if the program is correct, you need to test it. Testing a program means that you **compare** the output which your program produces (called the actual output) to the output which was specified, (called the expected output).

With the CS150 testing framework, there are two kinds of tests: **instructor tests** (which your instructor has written), and **student tests**, which you can write yourself.

Instructor Tests

To run the instructor tests, type **make test** in the terminal. The test checks to see that you have the correct output for different inputs. Here is the testing output for **H00**.



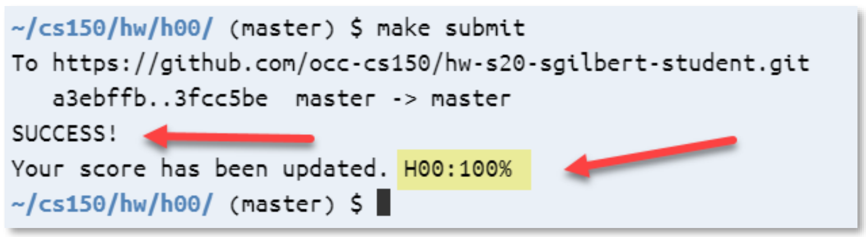
```
~/workspace/cs150/hw/h00/ $ make test
INSTRUCTOR TESTING H00--sgilbert
-------------------------------------------------------------------
   + Input of 5->0.000141748, 7054.78
   + Input of 8->0.000226796, 4409.24
   + Input of 11->0.000311845, 3206.72
   + Input of 14->0.000396894, 2519.57
   + Input of 15->0.000425243, 2351.59
   + Input of 24->0.000680389, 1469.75
   + Input of 32->0.000907186, 1102.31
   + Input of 35273.92->1, 1
-------------------------------------------------------------------
INSTRUCTOR TESTS H00:sgilbert: ALL TESTS -- PASS 8/8 (100%).
-------------------------------------------------------------------
MTUxMzk3NTcxNDpzZ2lsYmVydDpIMDA6MTAwLjAwJQ==
```

At the bottom of the run is the score. Make sure that **your ID** correctly appears, and that the assignment displayed is also correct. The final line is the **completion code**.

For the optional student testing, you'll find instructions in each assignment handout.

Submitting your Assignment

To submit each problem for credit, type **make submit** from the terminal. You'll receive a confirmation if your submission is accepted.



```
~/cs150/hw/h00/ (master) $ make submit
To https://github.com/occ-cs150/hw-s20-sgilbert-student.git
   a3ebffb..3fcc5be  master -> master
SUCCESS!
Your score has been updated. H00:100%
~/cs150/hw/h00/ (master) $
```

The CS 150 Homework Console allow you to check your past scores and see about future deadlines. Log in with your student login name (e.g. **sgilbert**) and your student ID (including the capital C. e.g. **C0223456**).

Finish Up

- Complete the Reading Exercises (**REX**) that accompany this chapter.
- Complete `H00` using the **CS50 IDE**. You'll find the instructions on Canvas.
  a. Over the weekend you'll usually have two assignments. During the week (Monday to Wednesday, for instance), you'll have only one.
  b. Make sure you submit the assignment using `make submit`.
  c. Make sure you check the CS150 Homework Console to see that your scores got reported, before the beginning of the next lecture.
- Take the pre-class reading quiz. You have two attempts.

See you in class or on the Canvas discussion board.