



Exported for Brian O'Connell on Tue, 27 May 2025 19:33:25 GMT

# Programming 9 Pre-lab lesson - CLION/C++

There are questions included throughout this reading. They are required for participation in this assignment. You do not need to get 100% correctness for full credit but you do need to at least attempt each. There will also be a guided walkthrough activity for implementing a C++ program.

After completing this pre-lab, you should be able to recognize and understand:

- The parts/elements of a C++ Program
- Standard Data Types
- Input/Output Formatting
- Mathematical Operators
- Loops

This lesson will go over:

- CLION Basics
- Introducing C++
- Setting up a C++ program
- Input/Outputs
- C++ Mathematics
- Loops

## What is C++?

You're already familiar with a variant of C++, Arduino, but let's talk about the base-line programming language.

C++ is a general-purpose object-oriented programming (OOP) language. It is an extension of the C language and therefore it is possible to code C++ in a C-style, a structural or procedural method that you should be familiar with by now, or in an object-oriented style, a more advanced modularized style capable of Polymorphism, Abstract Data Types, Encapsulation, among others. We will focus on continuing the scripted C-style since object oriented is more advanced than our needs and too abstract to contextualize in the time we have.

C++ is one of the more popular languages currently, particularly due to its legacy use. It's utilized with system/application software, drivers, client-server applications, embedded firmware, and even serves as the foundation for the Arduino IDE and many of the available libraries for that hardware. It is a powerful, albeit somewhat complex, tool that will enable you more freedom and capabilities than MATLAB but at a cost of complexity.

## Differences with C++

- MATLAB is fundamentally designed as an engineering calculation tool
  - It is designed to do specific things very well and with relative ease
  - Even though it can do a lot of things and has many add-ons to help with that, it is still essentially limited in that scope
- C++ is a general purpose full features programming language
  - Can do everything MATLAB can and far more but has none of the added features that handle ambiguity or assumptions that MATLAB does

Let's look at the difference with a simple code example. Let's just print out the following lines:

```
Hello, world!  
This was created using ...
```

### In MATLAB:

1. `disp ('Hello, world!');`
2. `disp ('This was created using MATLAB.');`

### In C++:

1. `#include <iostream>`
2. `int main()`
3. `{`
4. `std::cout << "Hello World!\n";`
5. `std::cout << "This was created using C++.\n";`

6. `}`

C++ requires you to load the library necessary to print, encapsulate your code in a main function, and then compile that into its own program before running. MATLAB already has much of that assumed and allows you to run single lines of code without compiling a full application.

Even though Arduino is based on C++, it also has some things already taken care of. In Arduino, you always need a `setup()` and `loop()`. In C++, you always need a `main()`. It will also only run once.



### Debugging

[Show Correct Answer](#)[Show Responses](#)

Will your experience debugging MATLAB and Arduino help with C++?

**A**

No

**B**

Yes

## CLion

CLion is an Integrated Development Environment (IDE) created and distributed by JetBrains, a software development company located in the Czech Republic but with offices in Boston and other cities. It is a C and C++ editor with code analysis and debugging tools that will aid in your success with the C++ lab assignments. In the same way that the Arduino IDE is how you code and compile Arduino scripts, CLION will be our IDE for coding and compiling in C++

Before we get you started going over some of the C++ concepts and commands, you need to be able to run them to better engage with them. This next section will go through the following:

- Access or install CLion IDE
  - The tool we'll use for development of C++ applications
- Start a new script written in C++ using the CLion IDE
- Compile and Run that script
- Save your work and recover your work for C++ Assignments

- Properly set up and submit documents for C++ assignments

## Getting Started with CLion

As with all the software we will be using, it is accessible through VLab or available for download and installation on your personal computer. There are also pros and cons to each option

AS USUAL THOUGH, it's recommended to install the application locally if you can.

## Downloading and Installing CLion

### Pros

- Instant access
- Not susceptible to internet access
- Cross-platform so layout is similar across versions

### Cons

- Requires some hard drive space
- Requires minimum CPU power – Up to 2GB of RAM

## Accessing through VLab

### Pros

- All course instructions assume you're using the versions available on the VLAB
- You can save your files on your partition of the server, saving more hard drive space

### Cons

- Dependent on internet access
- If many students are accessing the server, it may slow down

## Access through VLab

- Connect to **VLab** through the VMWare Horizon Client
- Search for '**CLion**'
  - The full name of the application is **Jetbrains CLion**
    - There may be a date attached to the end of the name as some versions have included that

- Select and start the application
  - It may ask you to accept a user agreement when you start it up.
- Skip the install instructions and go to **Creating, Compiling, and Running a script**

# Install CLion Locally

If you plan to use VLab to access CLion, then skip this section.

**Do not try to install CLion on the virtual machine accessed through VLab.** It already exists there and you do not have administrative rights for installing software there anyways.

Follow the instructions in the FYELIC sharepoint for installing CLion:

<https://northeastern.sharepoint.com/sites/FYELIC/SitePages/FYELIC-Software.aspx>

You may have to install a compiler along with this though.

## Installing a Compiler

A compiler is needed to convert the code you write into machine language.

### For PC users: Install the MinGW compiler

For PCs we will use MINGW.

- Identify if you're using a 64 or 32-bit machine.
  - If you don't know, try using this video
  - or google search "Identify if my laptop is 64 or 32 bit"
- Go to <https://bit.ly/2kVR51m> and download and run the MINGW installer.
  - For some, they get a "The file as been downloaded incorrectly" error during install. In that case, use this compiler - <https://sourceforge.net/projects/mingw/>. This is a reduced version, more than enough for what we're doing.
- When you get to the settings if you are running a 64 bit version of windows (unless you have a very old PC, you probably are) change the Architecture to x86-64.
  - Leave everything else alone and hit next.
- Accept all other defaults to finish the install.

### For Mac Users: Update XCODE

Mac has a built-in development environment called XCODE which includes C++ compilers. We just need to make sure you have it activated and updated. XCODE can be obtained/updated for free

through the App Store if it is not.

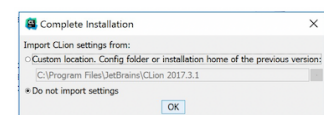
- Determine if XCODE is installed:
  - Open Spotlight Search
    - hit 'Cmd+Spacebar'
    - or click on the the magnifying glass icon on the top bar
- Type *terminal.app* which will open a terminal window.
- Click in the terminal window and type **g++** then, hit enter.
  - If you get something similar to:
    - **YourComputerName~ YourUserName\$ g++**
    - **clang: error: no input files**
  - Then XCODE Command Line Tools are installed
- If you get a different message, XCODE is not installed and you may be prompted to install.
- If you are not presented with the option, go to the **App Store** and get XCODE.

## Running CLion for the first time

Once you have installed CLion, you can run it. There is a bit of setup to do when we run CLion for the very first time. This is mostly the same on all platforms. For Windows, there is an extra setup to get a Unix based C++ compiler installed, but the CLion Wizard will guide you through it.

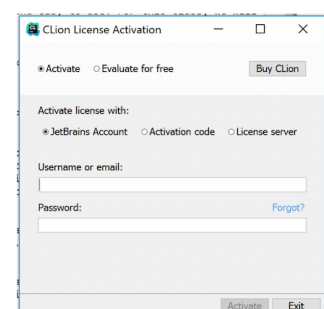
When you first run CLion, it will ask if you would like to import settings. You will see the following dialog box:

- Choose “Do Not Import Settings”.  
Click OK.
  - If necessary, close the next  
window until you see the ‘CLion License Activation’ window.

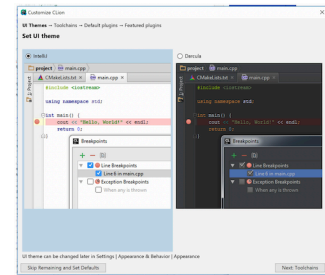


It will now ask you to activate your CLion License.

- Enter your JetBrains username and password for the account you created.
- Then Click on Activate.
- It may ask you to choose a theme.  
Pick one that you like.



- You can change it later by going to **Files / Settings** on Windows, or **CLion / Preferences** on a Mac.
- The remaining windows are generally not relevant for new programmers, so just click on the Skip Remaining and Set Defaults tab.



Your setup is now complete.

## Creating, Compiling, and Running a script

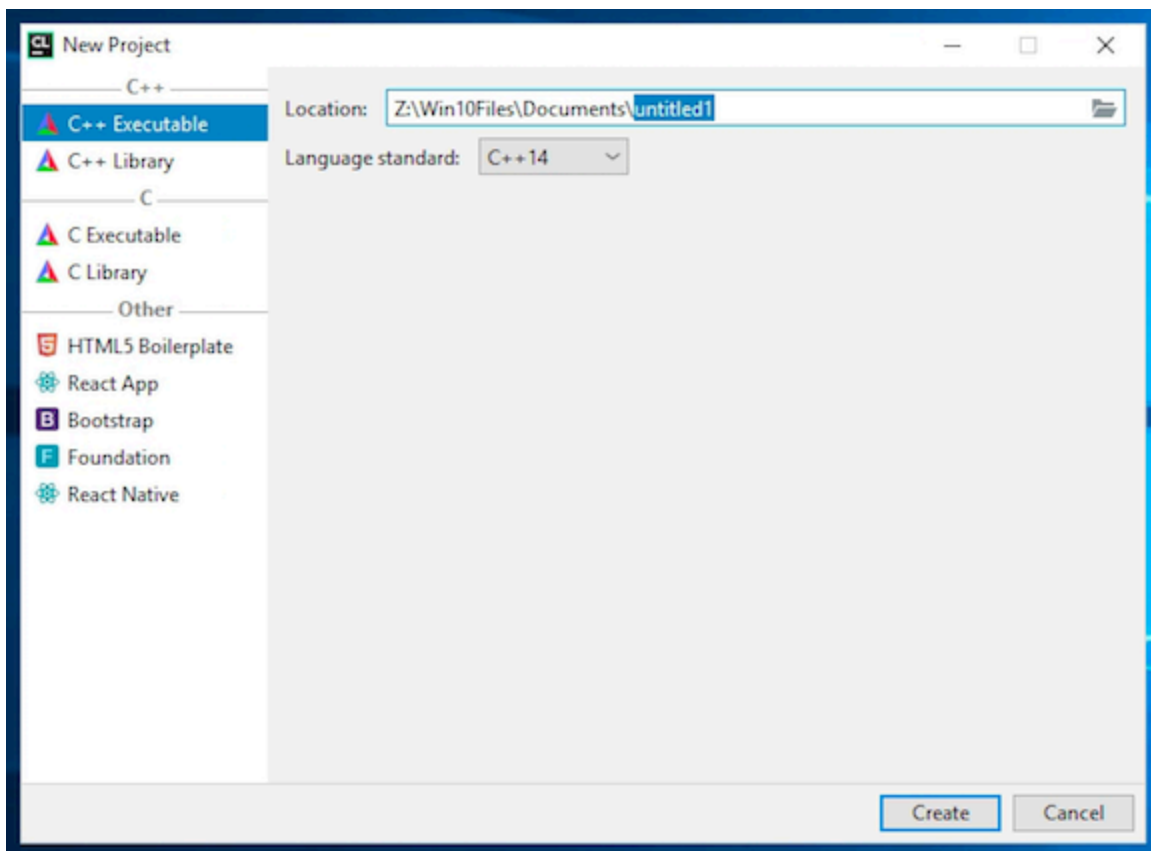
Now that you've gained access to a C++ development environment, let's actually make sure you know how to start a project and submit all the proper documents for the assignments.

## STEP 1 – Start your first C++ Projects

In order to write and run a C++ program on CLion, you need to create a project.

**NOTE:** The following images have old naming conventions. Make sure you use the current convention of **P8L1\_....**.

- Open **CLion** and then choose **File / New Project**
  - It may also give you this option when you first start the application



You will now see a screen where it asks you to choose where to save the project. The new project is called “untitled1” by default.

- Change the name using the standard naming convention. This also creates a location for your CLion project.
  - By default, CLion creates a folder called CLionProjects in
    - **C:\\Users\\<username>\\CLionProjects\\** under Windows
    - **Macintosh HD\\Users\\<UserName>\\ClionProjects\\** on a Mac and
    - **Z:\\Win10Files\\Documents\\** when using VLab

### Important Note:

You can locate your project anywhere by changing the file path here. However, wherever you choose to locate your project, it would be wise to implement some organization strategy here. One way to approach this would be to create a folder within CLionProjects for each assignment, and within this folder, create a separate project for each program within the assignment.



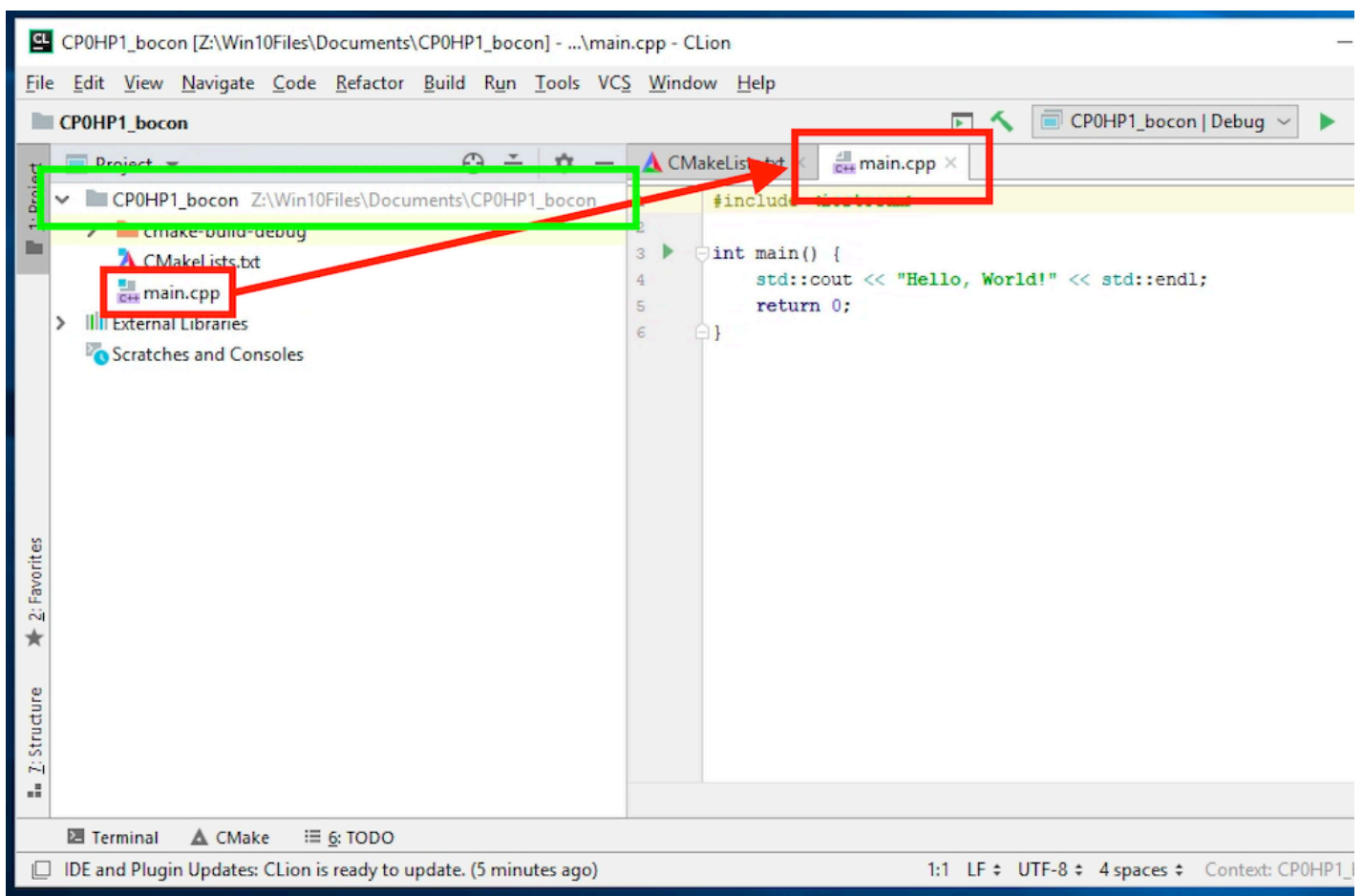


Image uses an old naming convention

Once you create a project, the project window will open with a short example program of *HelloWorld* in the editor window. The folder you created is displayed in [the Project View](#), the left side window and highlighted with a *green box*. **main.cpp** will be open in the editor, the right side window. The *red boxes* show the source file from the project and the filename of the navigation bar.

The CLion project window always opens with this program and you will replace the HelloWorld program with your own code. The project tree is shown in the left pane of the project window. The basic project will contain your **source code (main.cpp)**, a folder named **cmake-build-debug**, and a file named **CMakeLists.txt**.

### Important Note:

Note that the tab at the top of the window shows the program name as **main.cpp**. CLion is designed to accommodate large coding projects where programmers might need to create additional **.cpp** and header (**.h**) files. Those topics are beyond the scope of this course, but each CLion project must include one, and only one, main function (contained in the **main.cpp** file). In this course you will be writing all of your code in the **main.cpp** file. This is referred to as your **source code**. **Do not change** the name of **main.cpp** as it will create problems with the **MakeFile**,

which contains instructions for compiling the code and linking the necessary libraries to create an executable.

## STEP 2 - Create your C++ Program

Commenting in C++:

- For line comments:
  - Use `//`
  - Use **Ctrl+/ **(on PC) or **Cmd+/ **(on Mac) to line comment or uncomment a selection********
- For block comments
  - `/* ... */`
  - Use **Shift+Ctrl+/ **(on PC) or **Opt+Cmd+/ **(on Mac) to block comment or uncomment a selection********

Add a title block to the top of your **source code**.

- In a comment block, include the following information at the top of your **source code**.
  1. `/* Programming #<Lab #> Lesson Part <Part # of the Lab>`
  2. `* <First Initial>.<Last Name>`
  3. `* <Include names of those who you work with or receive significant help from, if applicable>`
  4. `* <Name of Script>`
  5. `*/ <Description of what the program will do.>`

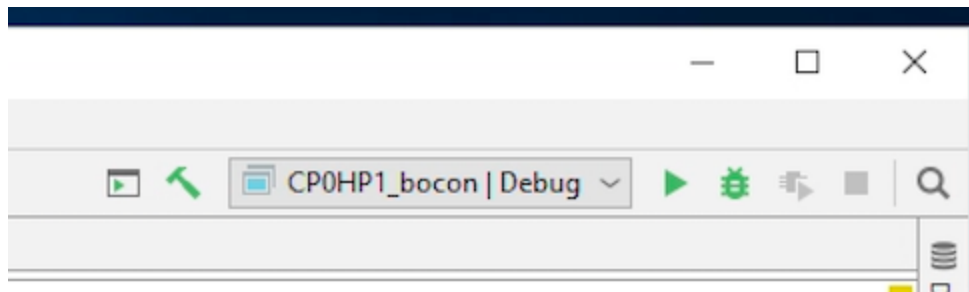
Update the code. We are going to update it to match your first MATLAB program so you can see the level of difference in complexity.

- Update the code to match the following:
  1. `#include <iostream> // Needed to communicate with console window`
  2. `#include <ctime> // For time functions`
  3. `int main() {`
  4. `std::cout << "Hello, World!" << std::endl; // Print to`
  5. `std::time_t now = std::time(0); // Get Current Timestamp`
  6. `char* dt = std::ctime(&now); // Convert it to characters`
  7. `std::cout << "The local date and time is: " << dt << std::endl;`
  8. `return 0;`

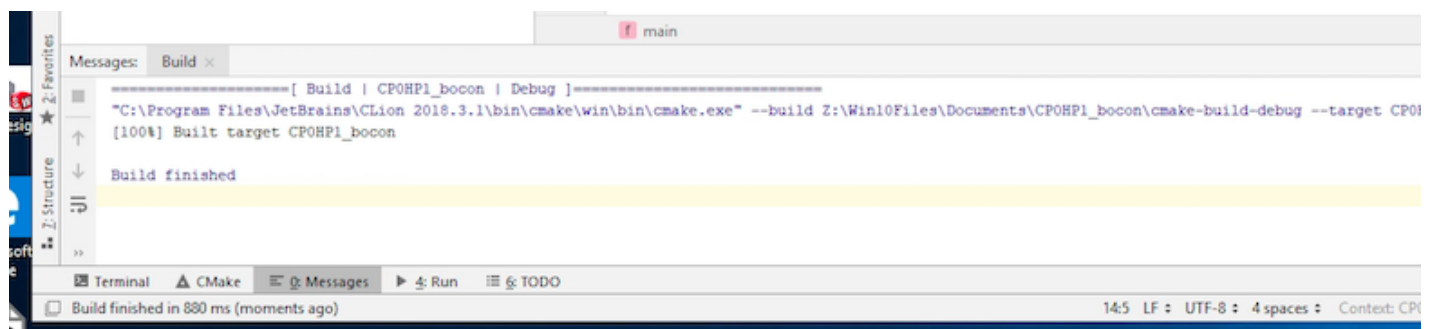
## STEP 3 - Run your first Program

Once you have completed writing your program, it's time to build and run your project. If your compiler is correctly installed, once you create your project, you should see at startup of a project a blue progress bar at the bottom of the editor with the message: *Loading CMake project.* followed by Loading symbols followed by Updating Symbols. This may take a few minutes, especially under Windows, but if you're busy typing your code, you will not notice this delay. When the process is complete, the **hammer** and arrow icons at the top right of the window will turn **green**.

If this doesn't work, see the troubleshooting section at the end of this chapter.



- Build your program by clicking on the **hammer**
  - This will open a **Build** window on the bottom
- If your program is error free, you should see something similar to the following screen:



- It should display something along these lines. It will vary slightly depending on operating system and version:

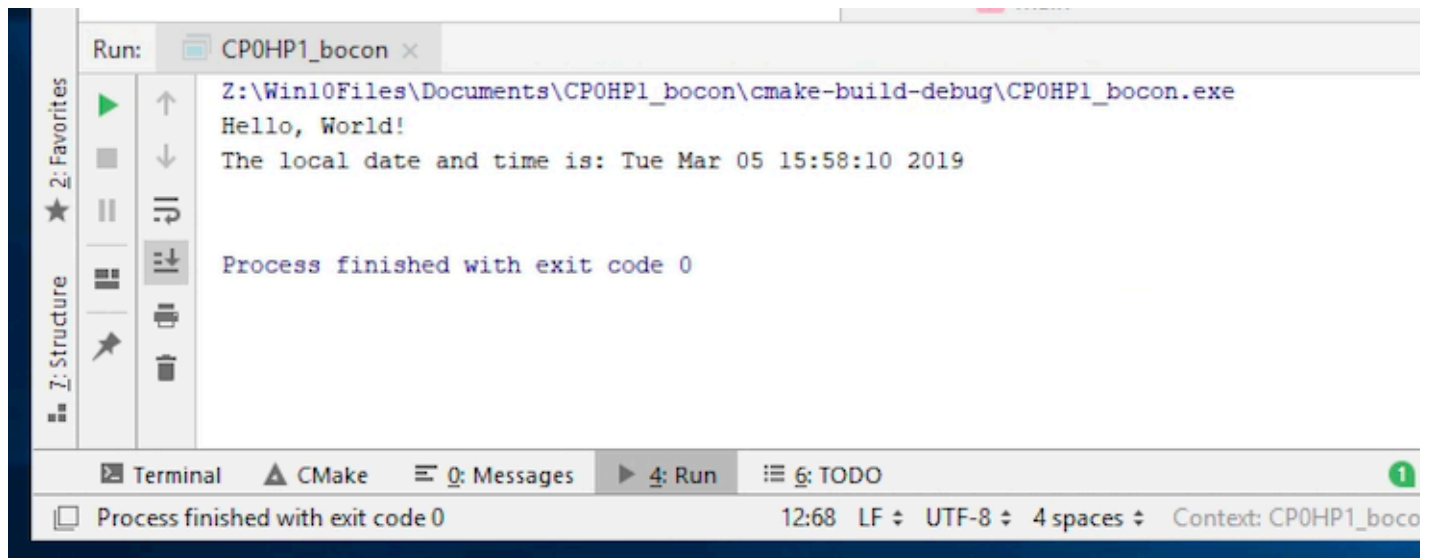
1. **[100%] Built target <Project Name>**

## 2. Build finished

- If not:
  - check that you copied the above code completely.
  - This window will also display the location of any errors, indicating the line number it occurred at.

Now that it's fully compiled, you can now run the program:

- Run your program by clicking on the **Green Arrow**
  - If you haven't built the program in a while, this will rebuild it before running.



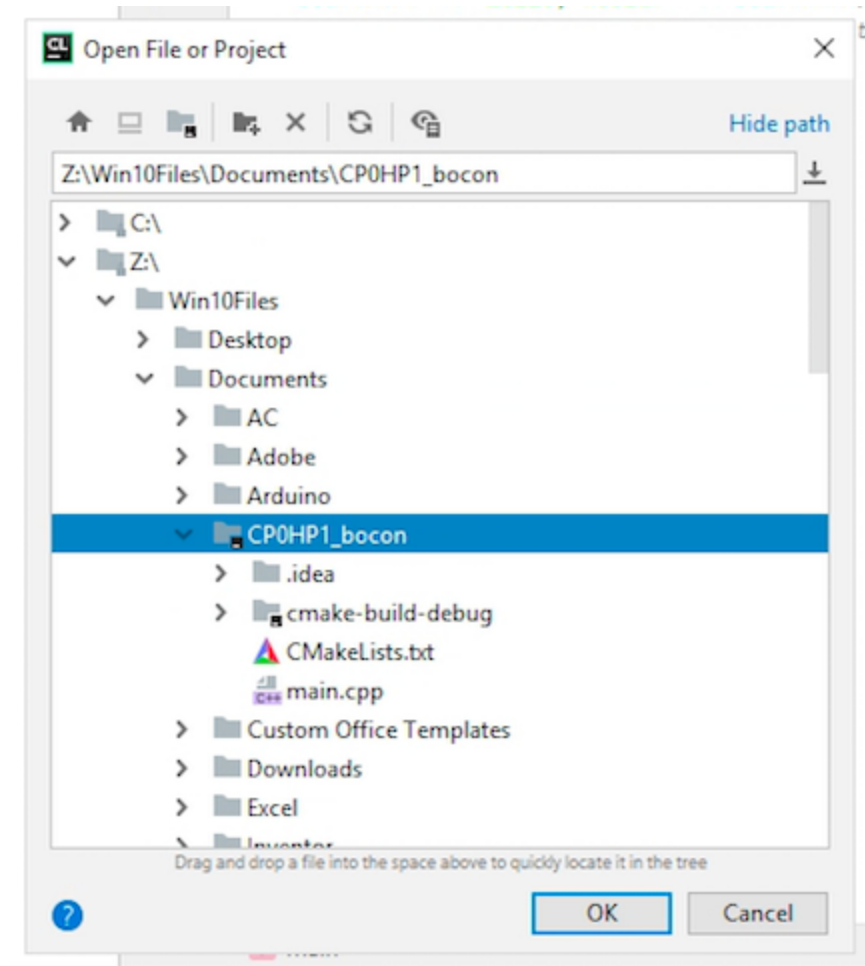
Your output will be displayed on the bottom of the screen in the console window, where the build results previously were.

## STEP 4 - Save and Re-Open your project

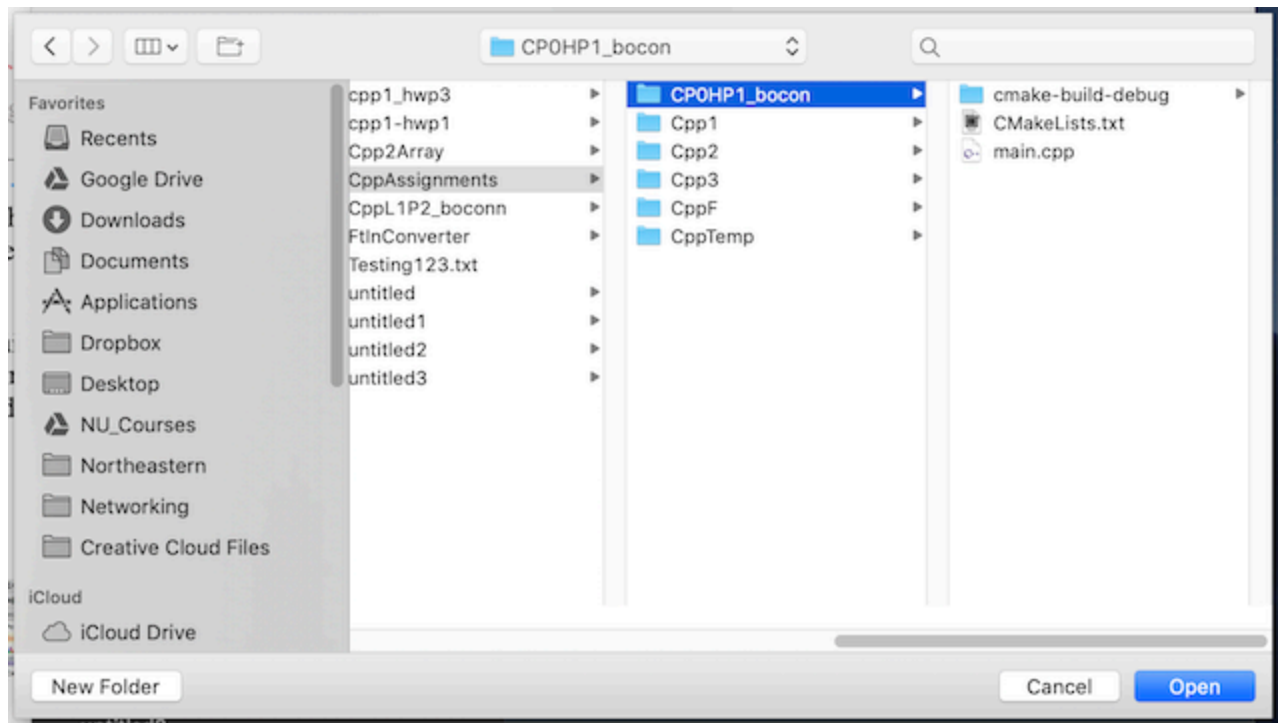
We'll go over reopening C++ Projects because it's a bit non-intuitive. You **are NOT** opening a specific file but rather **you ARE opening a folder** with CLion that contains the project files. That's the major non-intuitive part. After selecting the tool "open" in clion, you navigate to the folder containing your script and open that, not the script itself.

- Save your project (Hopefully this isn't the first time you've done so)
  - This saves all files in the project
  - **Command + S** (On Mac) or **Ctrl + S** (On PC)
- Close your project
- You may have to reopen **CLion**.

- Select **Open** from the "Welcome to CLion" window or go to **File / Open**
  - **Make sure you're selecting the project folder to open**
  - On PC, notice that there is an indicator on the file icon that shows that this is a useable project file.



'Open File or Project' window for on PC



'Open File or Project' window for on Mac

Most students accidentally try opening the **main.cpp** file and not the **project file**. That will open up the **main.cpp** file but without any of the other source files required to actually build and run the program. If you do this, close the CLion window and then remember to open the **Project File** itself.



Run your code

[Show Correct Answer](#)

[Show Responses](#)

Where, in the Editor window, do you click to run your code? Select the one the option that's always there, not the context specific one.

# Dissecting a C++ Program

At a basic level, a C++ Program has 4 major elements to consider in it's construction:

1. **Libraries & Directives**
2. **Declared Variables**
3. **Main()**
4. **Functions**

The libraries & directives determine what your program knows. In MATLAB, everything was pre-loaded. In C++ you have to state what information and capabilities your program should have. It's similar to having to enable a specific language when setting up your laptop or smart phone. The autocorrect won't recognize Swiss German or Farsi words if those capabilities aren't enabled and the appropriate dictionaries loaded.

Like in Arduino, you must declare variables before they can be used. MATLAB is like working in a fully stocked kitchen where you don't have to worry about not having the ingredients you may need because they're all there already. C++ and Arduino are like a small kitchenette. There are no ingredients to begin with. You need to make sure any ingredients are there before you start.

The main program is your plans, the blueprints to program. You should be familiar with the step by step programming requirements of this and experienced with having a specific space for that code from Arduino. Functions are another space for custom, repeated instructions but we will get into that in later labs.

# Libraries & Directives

As stated, these are how you establish what your program will know. A **library** is a modular component of reusable code. These can be used to integrate previously built and tested programming resources into your program. These include libraries for communication, mathematics, formatting, and other functionalities. A **C++ library** consists of one or more header files and an object **library**. A **directive** is a command to include that information in the programming, to be inserted into the program file during preprocessing when the program is compiling. This is done using the command **#include** and the header file name of the desired library.

In this lesson we will use the following libraries:

## Input/Output Stream Library

Command: **#include <iostream>**

Provides functionality to use an abstraction called *streams* to perform input and output operations between the program and the active console window, allowing information transfer from a user to and from the program.

Enables the use of the following [stream functions](#) (Just listing those we will use):

- **cin** - Standard input stream
- **cout** - Standard output stream

## Input/Output Manipulators Library

Command: **#include <iomanip>**

Provides functionality to allow for the parametric manipulation of *streams* from the program to the console window, allowing for format control of any outputs.



Enables the use of the following [manipulator functions](#):

- **setprecision()** - Set decimal precision (function )
- **setw()** - Set field width (function )

## Standard Library

Command: **using namespace std;**

The format is different because this is not preloading a specific functional library but establishing that the default language library for your program will be the standard library. So instead of having to specify the standard library to reference for every standard command you can just use the command. Without it, you'd have to preface every standard command with **std::** to differentiate it from commands with matching names in different libraries. Even though we will only be using standard library, C++ requires that clarification for each command unless you include this blanket statement. For instance, a bluetooth communications library might also have a **cout** command so you'd have to use **bth::cout** and **std::cout** to differentiate them in your program.

For instance, without it your code would look like:

```
std::cout << std::fixed << std::setprecision(2);
```

But with it, that line of code simplifies to:

```
cout << fixed << setprecision(2);
```

Please note, this is not a best practice. It will however make our introduction to C++ easier by eliminating an opportunity for error.

### Question

In C++, why is it necessary to include specific libraries and directives within a program, unlike in MATLAB?

- A** To load the required dictionaries for language autocorrection.
- B** To ensure the program has the correct ingredients before use.
- C** To enable automatic kitchen stocking of tools.

D

To build the main program blueprint.

 HINT

 EXPLANATION


Show Responses



Show Answer

# Declared Variables

## Data Types

Like in Arduino, before we can use a variable it must be declared with its data type. For review, here are the [default data types](#) available in C++ without including any extra libraries. Please note that the range is dependent on the bits of information assigned. C++ distributes that differently than the Arduino does because one runs on a CPU and the other on a microcontroller.

### Integers:

data type	range	calculated range
<b>short</b>	$-(2^{15})$ to $2^{15} - 1$	-32,768 to 32,767
<b>unsigned short</b>	0 to $2^{16} - 1$	0 to 65,536
<b>int</b>	$-(2^{31})$ to $2^{31} - 1$	-2,147,483,648 to 2,147,483,647
<b>unsigned int</b>	0 to $2^{32} - 1$	0 to 4,294,967,295
<b>long</b>	$-(2^{63})$ to $2^{63} - 1$	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<b>unsigned long</b>	0 to $2^{64} - 1$	0 to 18,446,744,073,709,551,615

### Floating Point Numbers:

Data Type	Range	Precision
<b>float</b>	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	6 digits
<b>double</b>	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$	15 digits
<b>long double</b>	$\pm 3.36 \times 10^{-4932}$ to $\pm 1.18 \times 10^{4932}$	36 digits

## Characters:

data type: **char**

Used like in other languages to handle single characters, ie letters and the like. This is stored as an integer but each is interpreted by the operating system as an ASCII character. For the equivalent representations, [see this chart](#). You just need to enter the character using single quotes.

## Boolean:

data type: **bool**

Binary representation of True/False. Stored as a 0 and 1 but interpreted as false(0) or true(1) by the operating system.

## Declaring Variables

Every variable needs to be declared before its first use. This lets the compiler set aside enough space for the variable in the computer memory and set how it will be interpreted. The syntax is to write the type and then the name. You can declare them individually or multiple if they are the same data type:

Declaring variables individually:

```
int a;
int b;
int c;
```

Declaring multiple variables of the same type simultaneously:

```
int a, b, c;
```

Variables declared outside of `Main()` are considered global and can be used anywhere. THIS IS NOT BEST PRACTICE because if you combine your code with others or use certain libraries, you may be accidentally overwriting things because you used the same variable names as someone else did. Variables declared in a function, including `main()`, are local and only accessible within that function. This is the preferred usage although for the scale and scope of what we're doing, global variables are allowed unless otherwise specified and useful for just beginning with C++.

## Initialization

In the above, they are created with NO initial value. Essentially they are empty containers. The compiler has still set aside and labeled those containers but without information in them. You can later set them equal to a value but if you try and use them when empty, you will get an error. You can declare and initialize them simultaneously though.

Examples:

- `int n;`
  - `n` is now a signed integer
  - can be positive or negative
  - Given no initial value
- `unsigned long m = 256;`
  - `n` is an unsigned integer
  - can only be positive
  - Given an initial value
- `float i,j;`
  - `i` and `j` are decimals, created simultaneously
  - Can only be done with no initial values
- `bool k=0;`
  - `k` declared as a Boolean
  - Initialized as false



Which type

Show Correct Answer

Show Responses

The value 132.54393402 can be represented in full using which data type?

<b>A</b>	double
<b>B</b>	void
<b>C</b>	int
<b>D</b>	bool
<b>E</b>	float

# main()

The **main** function is required by all C++ programs and it is how the compiler knows where to start its function. This is like **setup()** and **loop()** in Arduino. When Arduino compiles, it actually runs your program as:

```
main(){  
  setup()  
  while(1)  
  {  
    loop()  
  }  
}
```

Once compiled, all C++ programs start their execution with **main()**, independently of its location within the source code. The instructions contained within this function's definition will always be the first ones to be executed in any C++ program, regardless of there being other functions. For that same reason, all C++ programs require a main function.

## return

Functions, including **main()**, have a data type associated with any information they output. You will typically see main set up as **int main()** with a **return 0** at the end of the main code. The

return command will automatically stop the run although it is not necessary. The program will end when it gets to the end of `main()` anyways. All the return does is send an integer code to the operating system. No output or a 0 indicates exit success. There are ways to return a warning if something went wrong but we won't get into that at this level.

# Functions

This refers to functions specifically included in your program not as part of a library or `main()`. We'll get into writing custom functions in another lab.

## Writing C++ programs

Before we begin, let's go over some things to watch out for/common debugging issues.

- C++ only has limited default functionality. Only math understood: +, -, \*, /
- Spelling matters!
- Use `;` at the end of lines.
- Again, use semi-colons at the end of lines
- Be careful of **truncation**
  - number that is too big to fit in its allocated storage. Think the issues with storing `millis()`
- Variable names:
  - start with a letter (OR an underscore)
  - consist of only letters, digits, underscore
  - no keywords (like `int` or `return`)
  - case sensitive

Let's take a look at a simple program:

```
1. #include <iostream>
2. using namespace std;
3. int main ()
4. {
5.     char x;
6.     cout<< "Hello, world!," << endl
7.         << "C programming is fun!" << endl;
8.     cin >> x;
9.     cout << "You typed '" << x << "'." << endl;
```

```
10.     return 0;
11. }
```

Let's add comments to better understand it.

```
1. // Directives and Libraries
2. #include <iostream> // input/output stream library for communication
3. using namespace std; // Set a default reference for all commands
4. // No global variables
5. // Main function created
6. int main () // Starts the main function, required for a C++ program
7. {
8.     char x; // Created a local variable, only usable in main()
9.     // Prints information to the console (We'll go over these commands
        next)
10.    cout << "Hello, world!," << endl // No semi-colon so the compiler
        interprets this line and the next as a single action.
11.    << "C++ programming is fun!" << endl;
12.    cin >> x; // Looks for information FROM the console to load into x
13.    cout << "You typed '" << x << "'." << endl; // Prints back to
        console
14.    return 0; // Stops the function, returning a 0.
15. }
```

The program establishes what the program knows and how to interpret things with lines 2 & 3.

The `main()` function is created, a requirement in a C++ program. It uses the `cout` command to print information to the console and `cin` to take an input from the console. You should have enough experience with basic program structures by now though so let's just go over the specific commands.

## iostream commands

`iostream` will be commonly used for outputting information and inputting information.

Examples of all of these were used in the main example above.

### output

- `cout` - standard output stream
  - Opens an output stream to the default destination, the console window for CLION.

- `<<` - Insertion operator
  - Writes data to the indicated output stream
- `endl` - End line
  - A data input manipulator that inserts a new-line character and *flushes* the stream.

## input

- `cin` - standard input stream
  - Opens an input stream from the default destination, the console window for CLION.
- `>>` - Extraction operator
  - Reads data from the indicated input stream
  - Requires data to have been fed to that stream. In our case that is typically typed in the console and then hit enter.

## iomanip commands

`iostream` will be commonly used for formatting information as it is outputted.

- `setprecision(x)` - Sets decimal precision for the output format of floating point notation
  - Defines the number of numerical characters to display as `x`.
  - Default floating point notations considers all numerical characters, not including the decimal.
- `fixed` - sets output format to fixed floating point notation
  - THIS IS ACTUALLY PART OF `iostream` but typically used in conjunction with `setprecision`.
  - Fixed considers only numerical characters after the decimal.

To see the difference, run the following example:

```
1. #include <iostream>      // I/O stream
2. #include <iomanip>       // I/O Manipulators
3. using namespace std;
4. int main () {
5.     double f =543.1415987654;
6.     // default IDE system precision
7.     cout << f << endl;
8.     // Manipulating precision of default floating point notation
9.     cout << setprecision(5) << f << endl;
10.    cout << setprecision(9) << f << endl;
11.    cout << fixed;
12.    // Manipulating precision of fixed floating point notation
```



```
13.     cout << setprecision(5) << f << endl;
14.     cout << setprecision(9) << f << endl;
15.     return 0;
16. }
```

# Practice Problems

## Foot to Inch Convertor:

Practice using this fill in the blank example. Copy this to CLION, fill in the blanks, and try to run it. Complete the fill in the blank question below to show your understanding.

[https://raw.githubusercontent.com/boconn7782/CourseCode/main/CPP/P9L\\_Demo](https://raw.githubusercontent.com/boconn7782/CourseCode/main/CPP/P9L_Demo)

```
1. Fill in the blanks for the following code:
2. #include _____ // to use input & output
3. #include _____ // to use fixed, set precision
4. using _____ ; // to use standard c++ functions
5. int main ()
6. { // don't forget starting bracket
7.     _____ foot, inch; // declare vars: decimal!
8.     _____ << "Enter length: "; // display a user prompt
9.     cin >> foot; // get input from user
10.    inch = 12* foot; // calculate the conversion
11.    // decimal numbers to 2 places
12.    cout<<fixed<< _____;
13.    // display results
14.    cout<< foot << " ft = " << inch << " in" << _____;
15. }
```

Fill in the blanks for the following code segments. It's broken down into the 3 parts of a program.

### Libraries & Directives:



## Foot to Inch Convertor: Libraries & Directives

[Show Correct Answer](#)[Show Responses](#)

```
#include  // to use input & output -  
#include  // to use fixed, set precision  
using  ; // to use standard c++ functions  
...
```

## Declaring Variables:



## Foot to Inch Convertor: Variables

[Show Correct Answer](#)[Show Responses](#)

```
...  
  
int main () // alternative: int main (void)  
  
{ // don't forget starting bracket  
  
   foot, inch; // declare vars: decimal!  
  
  ...
```



## Type of variables

[Show Correct Answer](#)[Show Responses](#)

Because of where the variables are declared, what type are they and where are they usable?

**A**

Local and only usable in Main()

**B**

Global and usable in all applications

**C**

Local to this program and usable in any other functions in this program

**D**

Global but only usable in this program

**main():****Foot to Inch Convertor: main()**[Show Correct Answer](#)[Show Responses](#)

...

`int main ()``{ // don't forget starting bracket`

...

`<< "Enter length: "; // display a user prompt``>> foot; // get input from user``inch = 12* foot; // calculate the conversion``// Display decimal numbers to 2 places``cout<<fixed<<``;``// display results``cout<< foot << " ft = " << inch << " in" <<``;``} // don't forget ending bracket`

# CMATH

The **cmath** library provides access to a set of common mathematical operations and transformations. These include the following types of functions:

- Trigonometric & Hyperbolic
- Exponential & logarithmic
- Power
- Rounding & remainder
- Comparison
- and other types

These enable calculations exponential, absolute value, square root, natural logarithm, and more. They do not enable extra operator commands like using the '^' for a power function. For those, you will need to use the proper function call to perform that calculation, using **pow(a, b)** instead of **a^b** for instance.

The following are the more common commands :

Function	Description
<b>pow(a, b)</b>	Computes <b>a</b> to the power of <b>b</b> : <b>a^b</b>
<b>sqrt(a)</b>	Finds the square root of <b>a</b> , if <b>a &gt;= 0</b>
<b>abs(a)</b>	Finds the absolute value of <b>a</b> , integers only
<b>fabs(a)</b>	Finds the absolute value of <b>a</b> , all types
<b>exp(a)</b>	Computes e to the power <b>a</b> : <b>e^a</b>
<b>log(a)</b>	Finds the <i>natural</i> log of <b>a</b> (base is e)
<b>log10(a)</b>	Finds the <i>common</i> log of <b>a</b> (base is 10)
<b>ceil(a)</b>	Rounds <b>a</b> down to the nearest integer
<b>floor(a)</b>	Rounds <b>a</b> down to the nearest integer

This is very different from MATLAB in that you need to load the library to enable your program to handle this math while MATLAB has these capabilities pre-loaded. You also need to use the function calls for each rather than being able to write them out in a closer to natural or handwritten format.



Say it's 11PM and the homework is due at 11:59PM. You're pretty sure you won't get marked late if you can just submit it by morning. You need to perform one last calculation that's not listed in Lesson 2. You're pretty sure it's made available by CMATH but you don't know what the function command would be best to use or can't remember them all enough to know. What would be the smartest and most efficient thing for you to do?

**A**

Message Prof. O'Connell and hope he's up and that he'll pause his video game (currently Star Wars: Outlaws) or put down his book (Still on the most recent Stormlight Archive novel) to tell you the command outright.

**B**

Email Prof. O'Connell to ask for an extension after the assignment was already due.

**C**

Use an AI resource to ask about 'CMATH' and use whatever command it tells you to.

**D**

Google 'CMATH' and find a resource that lists all the available commands for that library. Read through and determine the right command.

**E**

Plan to use FYELIC the next day and accept the late penalty.

**F**

Make guesses at what the command might be to try and discover it through trial and error.

**G**

Use an AI resource and work through understanding 'CMath' better and determine the proper command

## C++ Loops

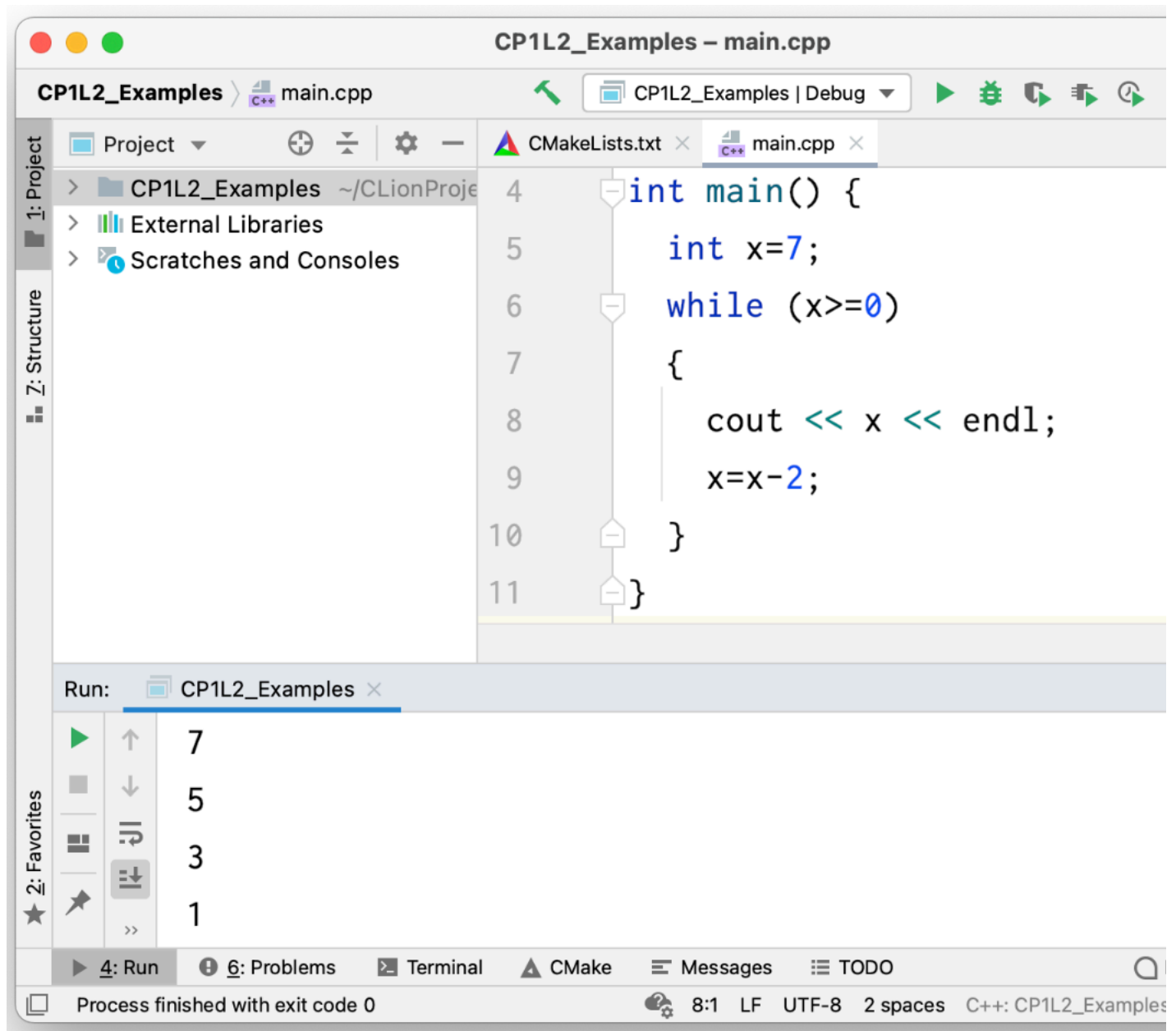
This will focus primarily on just the new syntax for C++ and provide some example. The programming concepts of loops, specifically of the following type, has already been covered, but the syntax and overview is repeated here as a reminder and for your reference.

### While Loops

- `while (expression){`
- `action(s);`
- `}`

For a **while** loop, the expression is assessed BEFORE each through the action(s). Each action requires a semi-colon after it and, in the case of multiple actions, they must be contained by braces (also referred to as curly brackets { }). If the expression is false, the loop will not run. For the loop to end, the expression must at some point become false. This is typically due to something occurring in the action(s).

## Examples:



The screenshot shows the CLion IDE interface with a project named "CP1L2\_Examples". The main.cpp file is open, displaying the following C++ code:

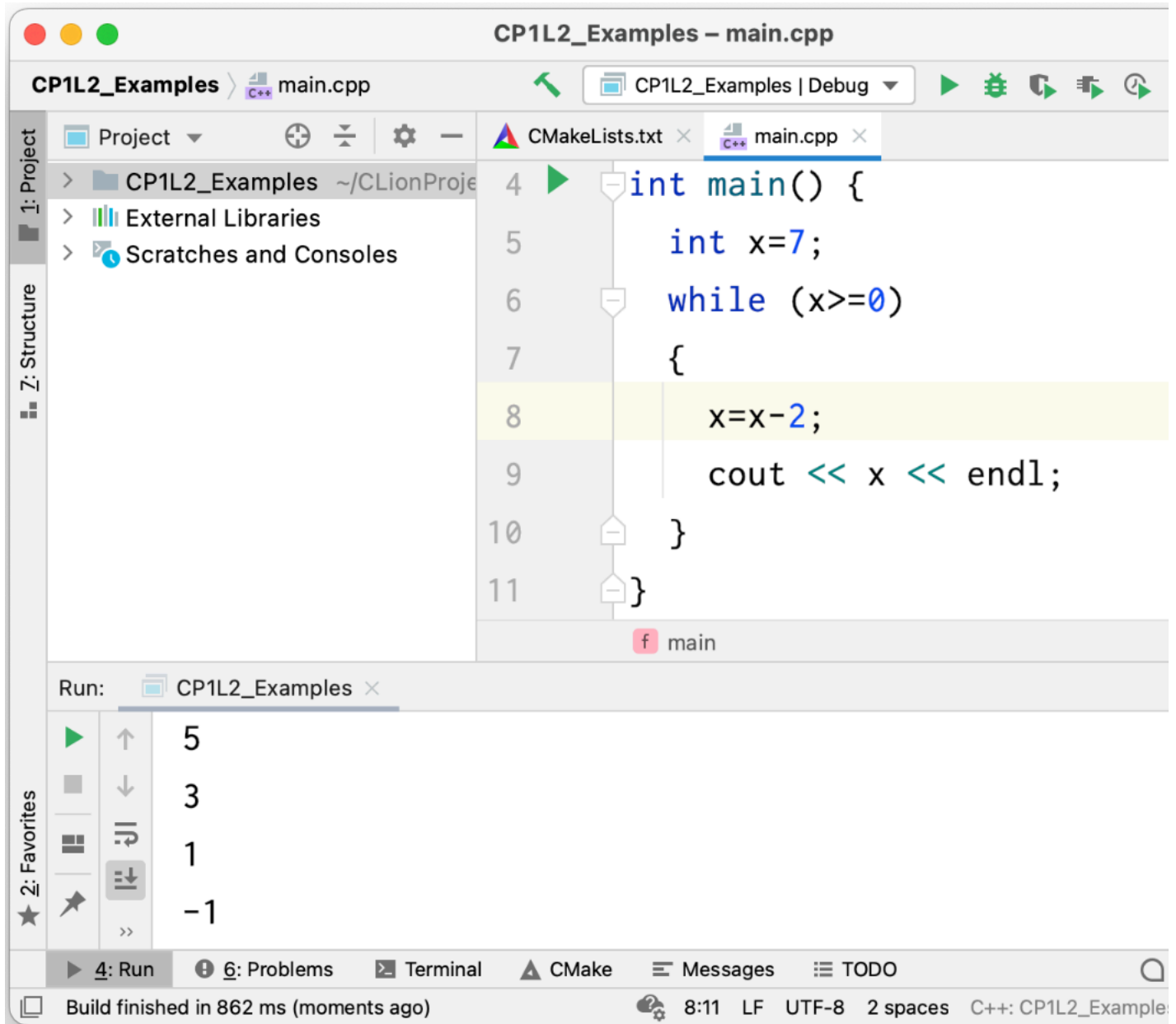
```
1 int main() {  
2     int x=7;  
3     while (x>=0)  
4     {  
5         cout << x << endl;  
6         x=x-2;  
7     }  
8 }
```

The code is executed, and the output is shown in the Run console:

```
7  
5  
3  
1
```

The status bar at the bottom indicates "Process finished with exit code 0".

Example 1: Output before action



Example 2: Output After Action

The previous 2 examples show the importance of **order of operations**. Once  $x$  is no longer greater than or equal to 0, the loop will not run again, but it will continue to complete the loop.

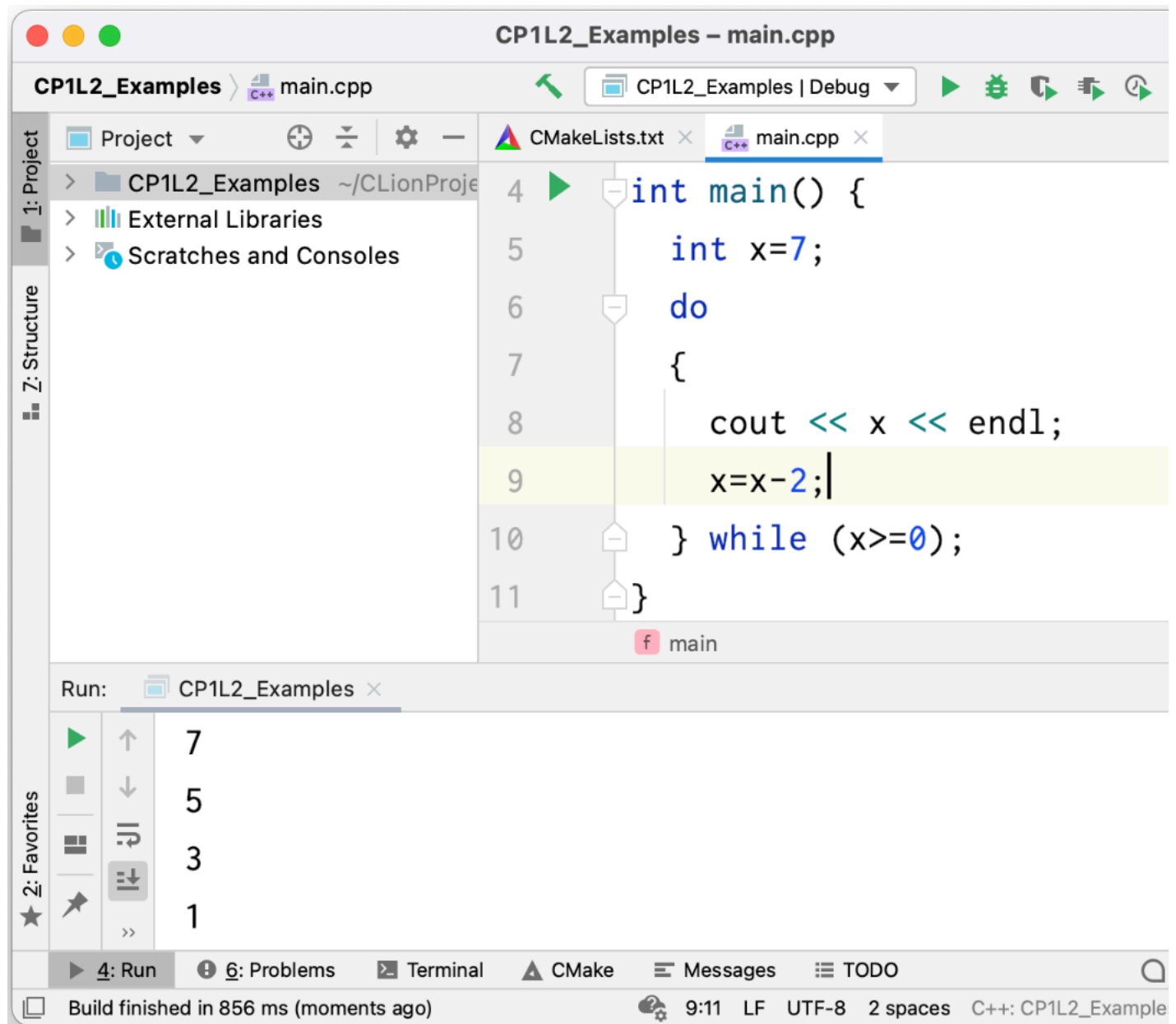
## Do While Loops

We have not done a do-while loop yet but, conceptually it's very similar to a while loop. It just differs in when the question is asked.

- `do {`
- `action(s);`
- `} while (expression);`

For a **do-while** loop, the expression is assessed AFTER each loop through the action(s). Each action requires a semi-colon after it and, in the case of multiple actions, they must be contained by braces (also referred to as curly brackets `{ }`). If the expression is false, the loop will at least run once because of that post loop evaluation. For the loop to end, the expression must at some point become false. This is typically due to something occurring in the action(s).

## Examples:



```
CP1L2_Examples – main.cpp
CP1L2_Examples > main.cpp
CMakeLists.txt x main.cpp x
1: Project
  > CP1L2_Examples ~/CLionProject
  > External Libraries
  > Scratches and Consoles
2: Structure
3: Favorites
4: Run
5: Problems
6: Terminal
7: CMake
8: Messages
9: TODO
Build finished in 856 ms (moments ago) 9:11 LF UTF-8 2 spaces C++: CP1L2_Example

4 int main() {
5     int x=7;
6     do
7     {
8         cout << x << endl;
9         x=x-2;
10    } while (x>=0);
11 }
```

Run: CP1L2\_Examples x

7  
5  
3  
1

Example 1: Output before action



The screenshot shows an IDE window titled "CP1L2\_Examples - main.cpp". The main editor displays the following C++ code:

```
4 int main() {  
5     int x=7;  
6     do  
7     {  
8         x=x-2;  
9         cout << x << endl;  
10    } while (x>=0);  
11 }
```

The output window at the bottom shows the results of the program execution:

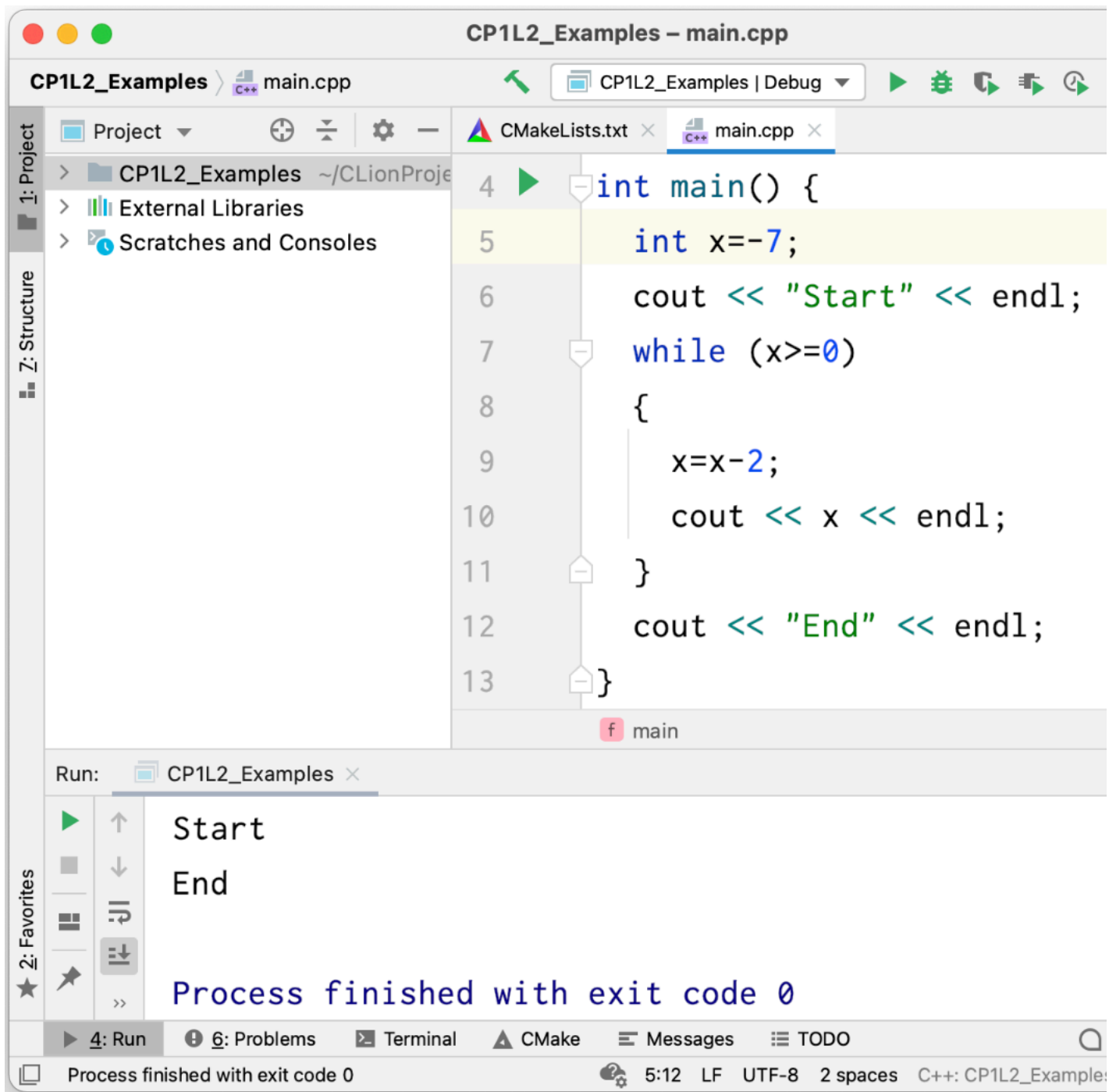
```
5  
3  
1  
-1
```

The status bar at the bottom indicates "Build finished in 213 ms (moments ago)" and "C++: CP1L2\_Example".

Example 2: Output after action

The previous 2 examples also show the importance of order of operations but also that once x is no longer greater than or equal to 0, the loop will not run again.

The following 2 show what happens when the statement is false before the loop begins.



Example 1: False statement going into a While Loop

```
CP1L2_Examples – main.cpp
CP1L2_Examples > main.cpp
CMakeLists.txt x main.cpp x
1: Project
  > CP1L2_Examples ~/CLionProject
  > External Libraries
  > Scratches and Consoles
2: Structure
3: Favorites
4 int main() {
5     int x=-7;
6     cout << "Start" << endl;
7     do
8     {
9         x=x-2;
10        cout << x << endl;
11    } while (x>=0);
12    cout << "End" << endl;
13 }
f main
Run: CP1L2_Examples x
Start
-9
End
4: Run 6: Problems Terminal CMake Messages TODO
Process finished with exit code 0 5:12 LF UTF-8 2 spaces C++: CP1L2_Example
```

Example 1: False statement going into a Do-While Loop

In the WHILE example, the loop does not run because the expression is checked before hand. In the DO-WHILE example, it runs once since the expression is checked at the end.



While or Do While, when is the question.

[Show Correct Answer](#)

[Show Responses](#)

There are two examples snippets of code below. They represent the two available implementations of While loops available in C++. Match what will happen when you run each.

Assume that 'Action' is a snippet of code that functions and makes NO change to the value of X.

Premise			Response	
1	X=2; do {Action} while(X>=15);	→	A	Will never perform "Action"
2	X=2; while(X>=15) {Action};	→	B	Will perform "Action" infinite times
			C	Will cause an error
			D	Will perform "Action" once

## For Loops

- `for {expression1; expression2; expression3}`
- `{`
- `action(s)`
- `}`

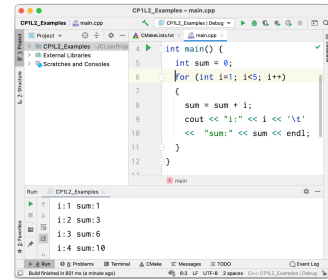
A FOR loop starts with 3 expressions:

- **Expression 1 - Initialization**
  - Creates the counter variable to be used throughout the loop
  - Commonly created as an integer and always includes setting its initial value
  - Typically something like `int i = 0`
    - Typically just an arbitrary simple variable name since it's just a counter, likely `i`, `j`, `k`, `l`, or another single letter
    - Initial value depends on use case, but commonly 0 or 1
- **Expression 2 - End Condition**
  - Defines the condition for continuing the loop. This is evaluated before the start of each iteration, stopping the iteration if it is false.
    - Typically a conditional associated with the variable from expression 1.
- **Expression 3 - Iteration**
  - Updates the counter to its new value
  - Executes after each iteration, just before the next.

# Examples

## Example 1:

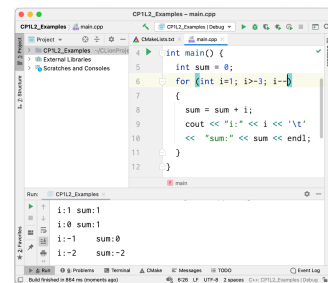
In example 1, **i** is initialized as an integer at a value of 1. Each loop is checked to see if **i** is still less than 5. After each loop, **i** is increased by 1 (**i++** translates to **i=i+1**).



Example 1: Using i++

## Example 2:

The incrementation does not have to always be positive.

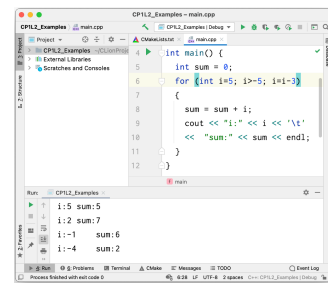


Example 2: Using i--

In example 2, **i** is initialized as an integer at a value of 1. Each loop is checked to see if **i** is still greater than -3. After each loop, **i** is decreased by 1 (**i--** translates to **i=i-1**).

## Example 3:

The  
incrementation  
does not have to  
always be by a  
single unit  
increment but  
does have to be  
constant.



```
1 int main() {  
2     int sum = 0;  
3     for (int i=5; i>=-5; i--){  
4         sum = sum + i;  
5         cout << "i:" << i << "\t"  
6         << "sum:" << sum << endl;  
7     }  
8 }
```

Run: CP12\_Examples

```
1 i:5 sum:5  
2 i:4 sum:7  
3 i:3 sum:6  
4 i:2 sum:2
```

Example 3: Using  $i = i - 3$

In example 2, **i** is initialized as an integer at a value of 5. Each loop is checked to see if **i** is still greater than -5. After each loop, **i** is decreased by 3.



Spot the errors

Show Correct Answer

Show Responses



Multiple answers: Multiple answers are accepted for this question

Which of the following for loops will NOT work? Select all that won't work

**A** for (i=5; i=10; i++) {...}

**B** for (i=5; i<=10; i++) {...}

**C** for (i=0, i<10, i++) {...}

**D** for (i=0; i<10; i++); {...}

**E** for (i=0; i<10; i++) {...}

```
F    for {i=0; i<10; i++} {...}
```

Now it's time for the walkthrough activity where you can fully create a program and submit it.

Please note that the video and the example code refer to this as part 2. I eliminated the part 1 so this is now your part 2 and I didn't want to completely reshoot the video for the one reference to Part 2 in the title block. So you are going to submit this as part 1.

## P9L1: Geometry Calculator

We will create a new C++ program that:

- Starts a loop that can only be exited by a user input
  - The program should ask the user if they want to calculate again...
  - And exit when the user finally answers “No”.
  - It should run at least once.
- Asks user for radius of a circle and
- Calculates the circumference & area.
- The program should print out the answer.

Requirements:

- For variables use data type: double
  - **pi** is not a built-in variable so you'll have to create it: **double pi=3.1415926535898;**
    - Use the version of pi here, to 14 decimal places
- Set precision to 3
- Test the program several times with 3 different inputs for radius before exiting:
  - Integers (examples: 1, 2, 3...)
  - Floating point (examples: 2.5, 22.678, 0.508...)
  - Scientific Notation (examples: 3.25e2, 6.25e10...)

**Create Pseudocode**

This is pretty straight forward for this part:

1. **PROGRAM GeometryCalculator**
2. **DO**
3.     **Ask user for a radius value**
4.     **Calculate the Circumference –  $2 \times \pi \times r$**
5.     **Calculate the Area –  $\pi \times r^2$**
6.     **Output the values using a set precision of 3 and scientific notation**
7.     **Ask user if they would like to calculate again**
8. **WHILE user wants to calculate again**
9. **ENDDOWHILE**
10. **END**

## Build your program

Start a new project using CLION. Use most of your Pseudocode as your first comments to help scaffold your program. Don't forget your title block as well:

1. **/\* Title Block \*/**
2. **// Ask user for a radius value**
3. **// Calculate the Circumference –  $2 \times \pi \times r$**
4. **// Calculate the Area –  $\pi \times r^2$**
5. **// Output the values using a set precision and scientific notation**
6. **// Ask user if they would like to calculate again**

You can also use this starter code to help ensure you have some of the elements not covered by the general comment scaffold:

[https://raw.githubusercontent.com/boconn7782/CourseCode/main/P9L1\\_Starter](https://raw.githubusercontent.com/boconn7782/CourseCode/main/P9L1_Starter)

1. **/\* Title Block \*/**
2. **// Libraries & Directives**
3. **#include... // Load the necessary libraries**
4. **using namespace std; // To simplify code**
5. **// Declare Variables**



```

6. double pi=3.1415926535898;
7. double r, c, a; // radius, circumference, area
8. string ans("Yes");
9. // Main Program
10. int main()
11. {
12. // Introduce and explain the program for the user
13. cout << ...
14. do{
15.     // Ask user for a radius value
16.     // Calculate the Circumference - 2*pi*r
17.     // Calculate the Area - pi*r^2
18.     // Output the values using a set precision and scientific
        notation
19.     // Ask user if they would like to calculate again
20.     cout << "Would you like to make another calculation?" << endl;
21.     cout << "Enter 'Yes' or 'No'";
22.     cin >> ans;
23. }
24. while(ans!="No");
25. }

```

## Set up your parts of the program:

### 1. Libraries & Directives

We have requirements that will need certain libraries:

- Again, we need to **input** and **output** information from the command window
  - Therefore use **iostream**
- Formatting requirements again:
  - Therefore use **iomanip**
- New this time, we have to do more than just basic math
  - Area will require a power function
    - Therefore use **cmath**

### 2. Declare Variables

- **Pi** isn't built-in so we'll need to add that

- We'll need to store some information:
  - The radius, the area, and the circumference
  - So we need 3 **variables** and we'll need to make they are **doubles** since **pi** is a **double**
- We also need to store the user response.
  - Required response to look for is a **"No"** which will be a **string**

### 3. Main Program

- We know what we want to do in our main program, we just need to use the proper syntax
  - `int main() { ... }`
- We want to repeat based on user responses so we'll need a `do { ... } while (...)` loop in `main{...}`
- We can't just call `r^2`. We need to use `pow(r,2)`

### 4. Functions

- Nothing is requiring a separate function

### Integrate this into your code:

```

1. /* Title Block */
2. // Libraries & Directives
3. #include... // Load the necessary libraries
4. using namespace std; // To simplify code
5. // Declare Variables
6. double pi=3.1415926535898;
7. double r, c, a; // radius, circumference, area
8. string ans("Yes");
9. // Main Program
10. int main()
11. {
12. // Introduce and explain the program for the user
13. cout << ...
14. do{
15.     // Ask user for a radius value
16.     // Calculate the Circumference - 2*pi*r
17.     // Calculate the Area - pi*r^2
18.     // Output the values using a set precision and scientific
    notation

```

```
19.     // Ask user if they would like to calculate again
20.     cout << "Would you like to make another calculation?" << endl;
21.     cout << "Enter 'Yes' or 'No'";
22.     cin >> ans;
23. }
24. while(ans!="No");
25. }
```

### Keep building out your code:

- We need to get information from the user so use `cout <<` to ask the user and then use `cin >>` to load their response into variables `r` and `n2`.
- Use basic math operators and the `cmath` functions to calculate `c` and `a`
- Use `scientific` and `setprecision(#)` to control the output format.
- Use `cout <<` to output your results. Give it a label so the user knows what information you are giving them.

## Walkthrough Video

The following video walks through this. I've added to the start code to include the do-while code to simplify things for you, particularly if you want to attempt this without following through with the video. In the video, it's manually added later. It also focuses just on the code elements for expediency so you still need to include an introduction to your program and ensure you have a full title block.

### Video

Please visit the textbook on a web or mobile device to view video content.