Getting setup C++ at home

C++ is a compiled language. This means that your code is translated by a compiler into and executable file that your computer understands. There are multiple compilers for C++, the most common being g++ and clang++. At camp we are using g++.

There are also different standard versions of C++. The latest, as of Jan 2017, is C++14. Compiling according to the correct version is important as some features may not work under older standards. For example, iterating over an array using range-based loops only works in C++11 or later.

Your options:

1. Read this short guide to installing g++ for the Command Line Interface (CLI).
2. On the web. Visit http://cpp.sh for an online compiler. This does not give you as much control over the compiler, but is good in situations where access to an alternative is not an option.
3. Look into installing an IDE (integrated development environment). You're not going to learn as much this way because the IDE does everything for you. They can also be a pain to get setup and often create strange errors.

The Index
Installing and compiling c++ programs

Installing the g++ compiler

g++ is part of a compiler program that turns c++ code into executables to be run by the computer.

It is possible that g++ is already installed. Type `g++ --version` in the command line of your selected operating system. If some text and a version number is displayed, you have g++. Otherwise, if you see a message saying that the g++ command is not found then you need to install g++ as follows.

Linux

Open the terminal and type `sudo yum install gcc-c++`. This video https://www.youtube.com/watch?v=NNF4nevgjWo also shows how to compile and run c++ programs too.

Ubuntu

Using the package handling utility, enter the following commands individually:

1. `sudo apt-get update`
2. `sudo apt-get install build-essential`
3. `sudo apt-get install g++`

OS X (Mac)

You could just install Xcode, which also installs command line developer tools including a complier.

However, this is around 10gb to install. If you just want the command line developer tools (which is much smaller) then open terminal.app and enter `make`. You will be prompted to install developer tools. Select “install”. Optionally use `xcode-select --install` instead.

Note: the developer tools install LLVM clang (accessible by typing `clang++` instead of `g++`). You can still use the `g++` commands in terminal, but these link back to LLVM which is not GCC (what we normally use). There is nothing wrong with using clang while you are learning, but if you want to use GCC for continuity then follow these steps:

1. install the home brew package manager (http://brew.sh) by entering the single command found on their site.
2. enter `brew update` in the terminal.
3. then enter `brew install gcc` for the latest stable build of GCC, this may take a long long long time to install (usually stops on the bootstrap bit, which is normal).
4. `brew info gcc` will tell you the version installed. **e.g.** mine is version 5.3.0
5. Use this version number to access gcc g++ in the terminal. **e.g.** I type `g++-5`. where `g++` is followed by a dash “`-`“ and the version “`5`”. You can also look in the homebrew directory for the name.
6. Notice by typing, for example, `g++-5 --version` in the terminal it will say something like `homebrew GCC` instead of `Apple LLVM` when typing `g++ --version`. Just remember to type, for example, `g++-5` instead of `g++` when compiling.

Windows

Install Mingw. Cygwin is not recommend as its emulation makes it much slower. Go to http://www.mingw.org/wiki/Getting_Started and follow the instructions **carefully** to install the MinGW installation manager.

1. In the package selection step, mark `gcc-g++`, `developer-tools` (MSYS) and `base` (Basic MinGW install) for installation under the "Basic Setup" package.
2. Select Installation > Apply Changes and then click Apply
3. Go back to http://www.mingw.org/wiki/Getting_Started and the "After Installing You Should" section to follow the steps before running the `msys.bat` file mentioned in the instructions.
4. When you run the `msys.bat` script, a command line window should open, type `g++ --version` to check that the installation has succeeded. It should show you the version number.
5. Create a nice shortcut to the `msys.bat` file. :)
6. In the same directory as the `msys.bat` file, is another folder called home. Inside home should be a folder with the same name as your current computer user. This user folder is your root directory for MinGW. Put your code in there.
7. To compile, do `g++ main.cpp` where your code file is called main.cpp and kept in the MinGW root directory. Put this command (and all future commands) into the command line window that appears after running `msys.bat`.
8. A file called `a.exe` should be created in the MinGW root directory.
9. Type `a.exe` to run the program.

Compiling c++ code

Assuming you followed the instructions above to install g++, each operating system should respond to the same command when typed in their respective command lines. Just remember to use MinGW if you are using Windows.

```
g++ -o ./myOutputFileName.out ./myInputFileName.cpp
```

The `-o` flag indicates the path we want to save to **e.g.** `./myOutputFileName.out` will save the executable file as `myOutputFileName.out` in the current directory. If you leave this bit out, the executable with be saved in the root user directory as `a.out` (or `a.exe` in Windows). In that case, run the program by typing `a.out` (or `a.exe`).

The last path (`./myInputFileName.cpp`) is the source file (your program). **e.g**. the program is in the current directory with the name `./myInputFileName.cpp`.

Remember, `./` refers to the current directory, `/` refers to the root user directory. So assuming you save the executable in the current directory, enter `./myInputFileName.cpp` to run the program.

Lets be fancy

The train site uses the command `g++ -std=gnu++11 -O2 -o program.exe tmp/program.cpp`. It's the same, just with different settings. It makes sense that we should use these settings too. Therefore, as a full command, use the following for competitive programming…

```
g++ -std=gnu++11 -Wall -o ./myOutputFileName.out ./myInputFileName.cpp
```

It sets some options that configure the compiler how we want it, but it's a bit different again…

`-std=gnu++11` means use C++11, it has more cool stuff in it (like range-based loops).
`-O2` (that's a capital "o") optimises the compiled program as a release version. This makes compilation slower, but the program run faster. That's why we don't need it when debugging, because we don't really care about tiny speed differences at that point.
`-Wall` means show us all the pretty warnings. The train site does not need this extra info, but for us this is useful when debugging.