Lab 18: Multiple Files & CMake

1. **Splitting declaration and definitions:** During lecture, we will walk through the exercise of splitting the following code into two files, one called `definitions.cpp` and one called `definitions.h`.

```
#include <string>
#include <vector>

class Robot
{
public:
    Robot(std::string const & name) : name_(name) {}

    void set_joint_configuration(std::vector<double> const &q) { q_ = q; }

    std::vector<double> get_joint_configuration() const { return q_; }
private:
    std::string name_;
    std::vector<double> q_;
};
```

2. **Reorganizing code into multiple files:** In this exercise we will practice reorganizing some code and compiling it manually.

   a. Download this cpp file and put it in a folder called `lab18_library`. Next, split it into 3 files: `main.cpp` `lib.cpp` and `lib.h`. The main file should contain the `main` function, and the library code should be split into declaration and definition. The declaration goes in the `.h` file, and the definition goes in the `.cpp` file. Make sure to add the right `#include` directives.

   b. Figure out the compile commands (e.g. `g++ main.cpp`) that you should use to compile all the files together into one executable. Run these commands in the terminal. The final executable should be called `main` and should have no file extension.

   c. Run your executable, it should print “8”.

d. Change the compiler commands to put the output executable, called `main`, in a folder called `build`. This is a good idea because it separates the source code from the compiled code, which is more organized, makes packaging the compiled code easier, and makes tracking files with git easier.

3. **Translating compile commands into CMake [20 minutes]:**
   a. Take the reorganized code you wrote for question 2 and copy everything into a new folder `lab18_library_cmake` so you don't mix up any of the files. Then, write a `CMakeLists.txt` to replace the commands you use to compile the code.
   b. Create your build directory: `mkdir build`
   c. Change directory to the build directory: `cd build`
   d. Run the cmake configure step: `cmake ..` (note `..` refers to the parent directory, which is where our `CMakeLists.txt` file is)
   e. Compile the code: `make`. It's worth noting that this sequence of commands can be used almost universally to compile any project written with cmake!
   f. See your code as built, then run it (think, where did the final executable go?)
   g. Now delete the build folder (`rm -r build`) and use VSCode to configure, compile, and run this code. (HINT: bring up the command palette and type "CMake")

4. **Include Paths:** When your write `#include "some/path"` (or `<some/path>`) you may need to tell the compiler where to look for those files. To do this, we use what's called the "include path". When compiling directly with the `g++` command, you can do this by passing `-I path/to/directory/`, and with CMake you can do it by adding `target_include_directories`. In this exercise, you will practice making sure the `#include` paths and the include path are correct.
   a. download this zip file
   b. Try to compile using cmake. This will fail, because the includes are missing.
   c. Look at what the include path has been set to in the `CMakeLists.txt` file, then add the necessary `#include` directives to the following files: `main.cpp`,

`kinematics.cpp`, `shapes.cpp`. Once the code compiles and runs without error, you're done.

5. **Using an external library with CMake [10 minutes]:** In this example you will write code that uses the Qt library, and you write the CMake code to compile it. Qt is *the* library for writing GUIs in C++, and is popular and well maintained.
    a. First, make sure the qt library is installed, by running the command: `sudo apt install qt5-default`
    b. Make a directory `lab18_qt` and copy the following code into a file called `main.cpp`:

```
#include <QtWidgets>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QWidget window;

    window.setWindowTitle("My First Window!");
    window.show();

    return app.exec();
}
```

c. Create the `CMakeLists.txt` file. Feel free to look at the instructions Qt provides for using cmake: https://doc.qt.io/qt-5/cmake-get-started.html. Most major packages will provide instructions on using their package with CMake, and Qt does a good job of this. **Make sure to completely and carefully read the text, don't just copy the cmake code!** Save the output you get from running the command `cmake ..` (or use the VSCode commands!) , you will be asked about it on the quiz. Once you run it, you should see a window pop up!