Moving Beyond One File

Peter Mitrano

Outline

- Why multiple files? [5 min]
- How to split up the code [30 min]
  - headers versus source
  - declarations versus definitions
- How to compile with multiple files [20 min]
  - command line
  - libraries vs executables

[5 min break]

- Project structure
- CMake for compiling multiple files [15 min]
- CMake for dependencies [15 min]

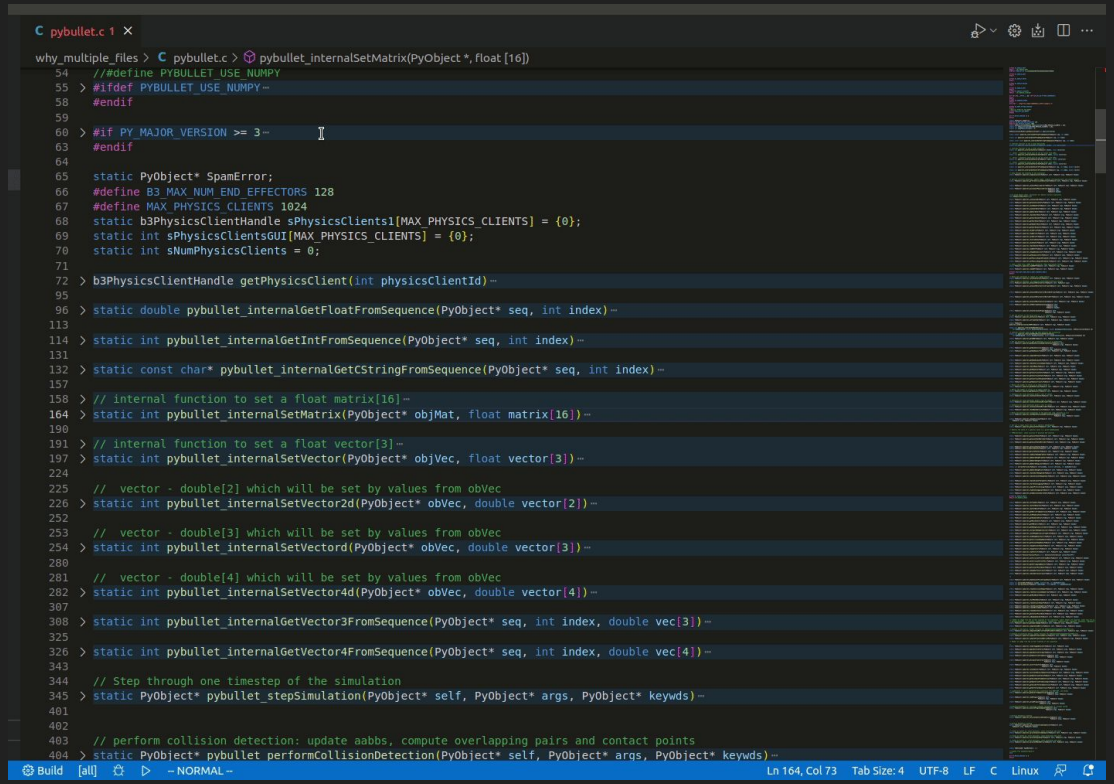
[5 min break]

- Lab exercises

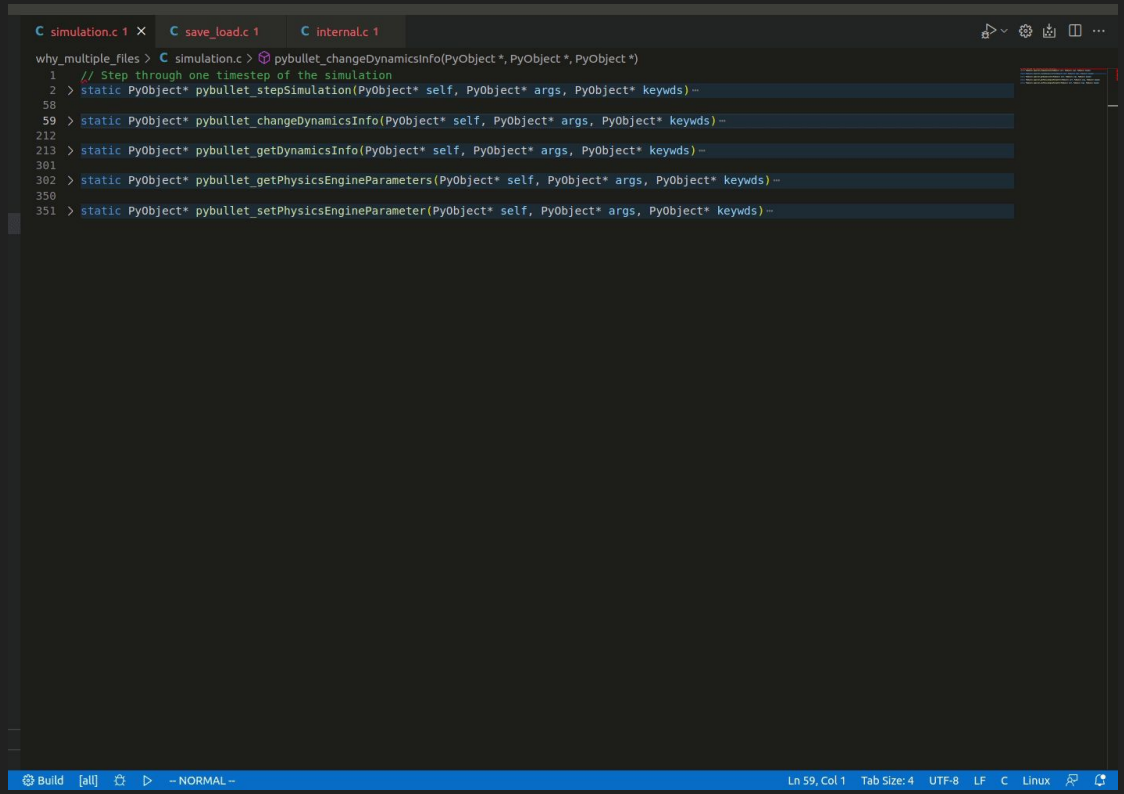
Why multiple files?

- Organization
  - Scrolling for ages is no fun
  - multiples files and folders make it easier mentally to handle the code
  - header files often belong in a separate folder

Why multiple files?



Why multiple files?



How to split up the code?

- Header Files (.h)
  - declarations

- Source Files (.cpp)
  - definitions

declarations.h

why_multiple_files > declarations.h > Robot > set_joint_configuration(std::ve

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

int const MAX_STEPS;

int add_int(int a, int b);

class Robot
{

public:
    Robot(std::string name);

    void set_joint_configuration(std::vector<double> q);
};

```

definitions.cpp

why_multiple_files > definitions.cpp > set_joint_configuration(std::vector<double

```cpp
#include "declarations.h"

int const MAX_STEPS = 100;

int add_int(int a, int b)
{
    return a + b;
}

Robot::Robot(std::string name) {
}

void Robot::set_joint_configuration(std::vector<double> q) {
}
```

How to split up the code?

Functions:

```
double sqrt(double x) { ... }
```

Becomes…

Declaration (in the .h):

```
double sqrt(double x);
```

Definition (in the .cpp):

```
double sqrt(double x) { ... }
```

How to split up the code?

What about classes?

Let's practice!

declarations.h

why_multiple_files > declarations.h > Robot > set_joint_configuration(std::ve

```
1   #include <vector>
2   #include <string>
3
4   int const MAX_STEPS;
5
6   int add_int(int a, int b);
7
8   class Robot
9   {
10
11  public:
12      Robot(std::string name);
13
14      void set_joint_configuration(std::vector<double> q);
15  };
16
```

definitions.cpp

why_multiple_files > definitions.cpp > set_joint_configuration(std::vector<double

```
1   #include "declarations.h"
2
3   int const MAX_STEPS = 100;
4
5   int add_int(int a, int b)
6   {
7       return a + b;
8   }
9
10  Robot::Robot(std::string name) {
11  }
12
13  void Robot::set_joint_configuration(std::vector<double> q) {
14  }
```

Compiling with multiple files

Building C++ code (even with only one file)

1. Compile
2. Link

You can compile one or more files into an executable using:

```
g++ main.cpp -o my_program
```

For multiple files, one executable:

```
g++ main.cpp lib1.cpp lib2.cpp -o my_program
```

Compiling with multiple files

Let's practice!

Compiling with multiple files - Libraries

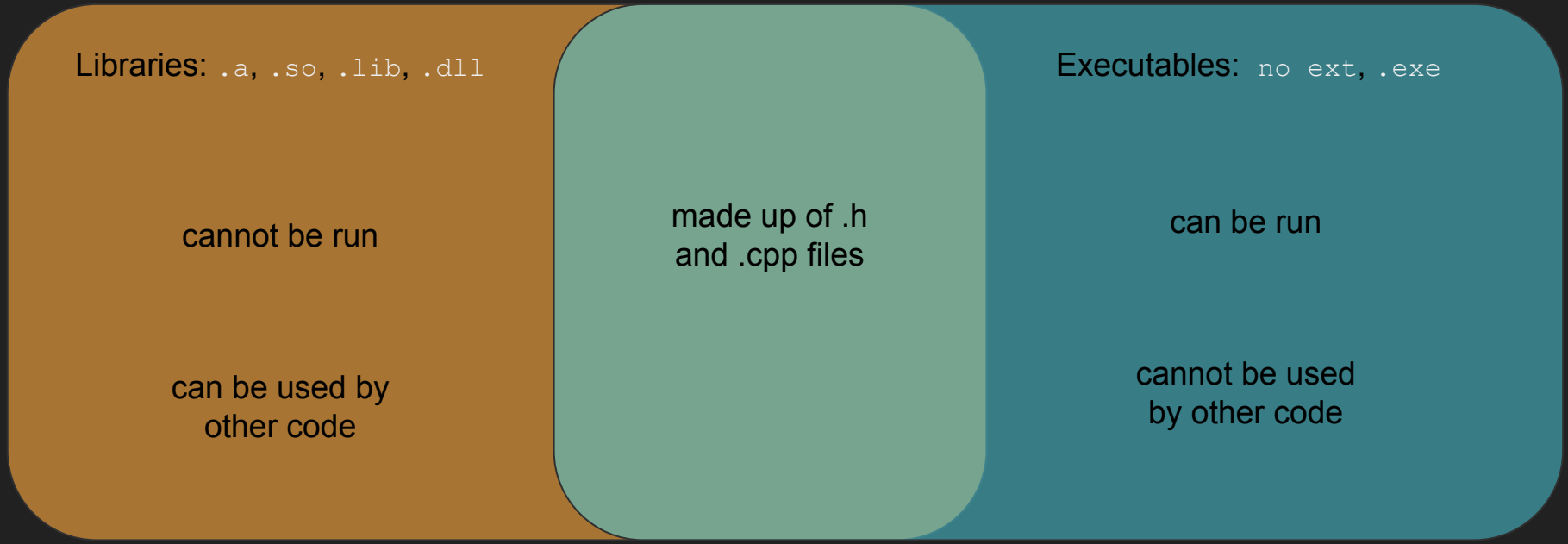
But usually we want to split up our code into libraries. Why?

- faster, otherwise you have to recompile all the code every time!
- reusable, you gave use other people’s libraries or they can use yours

So what is a library?

- A collection of compiled code that you don’t run directly, but use from other code

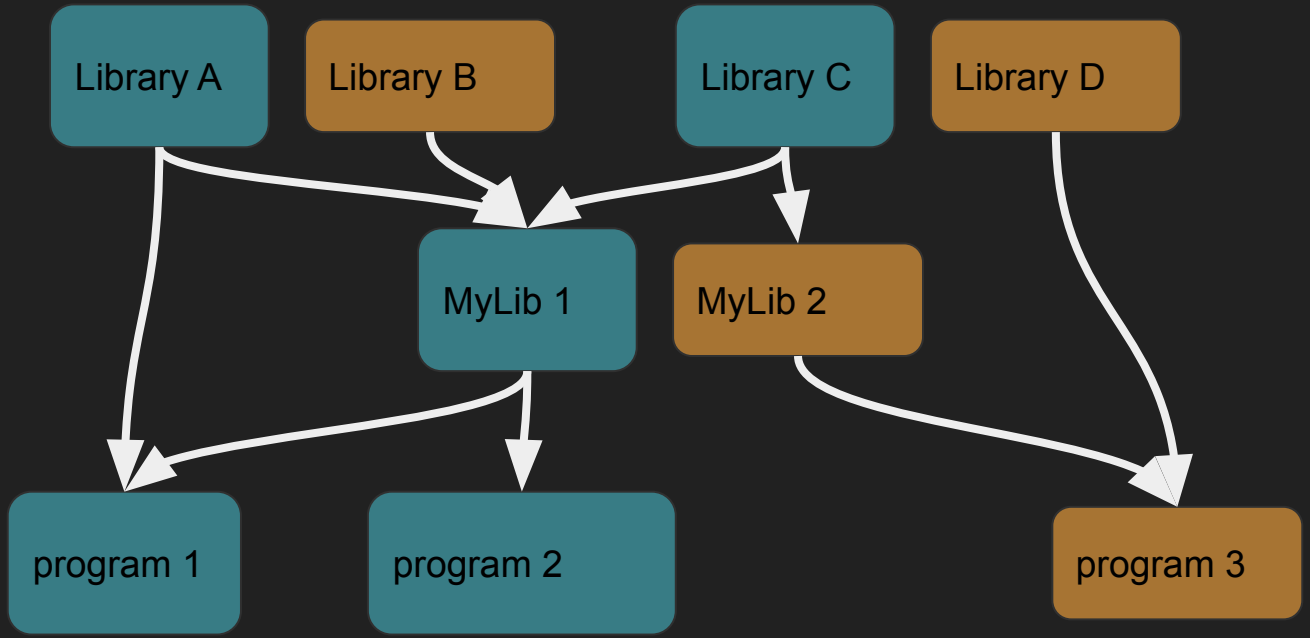
Compiling with multiple files - Libraries



Compiling with multiple files

Outside the simple coding of course, libraries are how everything works!

The illustration below is actually ***simpler*** than many C++ projects



Compiling with multiple files

```
chrome => /opt/google/chrome/chrome (interpreter => /lib64/ld-linux-x86-64.so.2)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
    libgobject-2.0.so.0 => /lib/x86_64-linux-gnu/libgobject-2.0.so.0
        libffi.so.7 => /lib/x86_64-linux-gnu/libffi.so.7
    libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3
    libnss3.so => /lib/x86_64-linux-gnu/libnss3.so
        libplc4.so => /lib/x86_64-linux-gnu/libplc4.so
        libplds4.so => /lib/x86_64-linux-gnu/libplds4.so
    libnssutil3.so => /lib/x86_64-linux-gnu/libnssutil3.so
    libsmime3.so => /lib/x86_64-linux-gnu/libsmime3.so
    libnspr4.so => /lib/x86_64-linux-gnu/libnspr4.so
    libatk-1.0.so.0 => /lib/x86_64-linux-gnu/libatk-1.0.so.0
    libatk-bridge-2.0.so.0 => /lib/x86_64-linux-gnu/libatk-bridge-2.0.so.0
    libcups.so.2 => /lib/x86_64-linux-gnu/libcups.so.2
        libgssapi_krb5.so.2 => /lib/x86_64-linux-gnu/libgssapi_krb5.so.2
            libkrb5.so.3 => /lib/x86_64-linux-gnu/libkrb5.so.3
                libkeyutils.so.1 => /lib/x86_64-linux-gnu/libkeyutils.so.1
                libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2
            libk5crypto.so.3 => /lib/x86_64-linux-gnu/libk5crypto.so.3
            libcom_err.so.2 => /lib/x86_64-linux-gnu/libcom_err.so.2
            libkrb5support.so.0 => /lib/x86_64-linux-gnu/libkrb5support.so.0
        libavahi-common.so.3 => /lib/x86_64-linux-gnu/libavahi-common.so.3
        libavahi-client.so.3 => /lib/x86_64-linux-gnu/libavahi-client.so.3
        libgnutls.so.30 => /lib/x86_64-linux-gnu/libgnutls.so.30
            libp11-kit.so.0 => /lib/x86_64-linux-gnu/libp11-kit.so.0
            libidn2.so.0 => /lib/x86_64-linux-gnu/libidn2.so.0
            libunistring.so.2 => /lib/x86_64-linux-gnu/libunistring.so.2
            libtasn1.so.6 => /lib/x86_64-linux-gnu/libtasn1.so.6
            libnettle.so.7 => /lib/x86_64-linux-gnu/libnettle.so.7
            libhogweed.so.5 => /lib/x86_64-linux-gnu/libhogweed.so.5
            libgmp.so.10 => /lib/x86_64-linux-gnu/libgmp.so.10
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1
    libgio-2.0.so.0 => /lib/x86_64-linux-gnu/libgio-2.0.so.0
        libgmodule-2.0.so.0 => /lib/x86_64-linux-gnu/libgmodule-2.0.so.0
        libmount.so.1 => /lib/x86_64-linux-gnu/libmount.so.1
            libblkid.so.1 => /lib/x86_64-linux-gnu/libblkid.so.1
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1
            libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0
libdrm.so.2 => /lib/x86_64-linux-gnu/libdrm.so.2
libdbus-1.so.3 => /lib/x86_64-linux-gnu/libdbus-1.so.3
    libsystemd.so.0 => /lib/x86_64-linux-gnu/libsystemd.so.0
        librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1
        liblzma.so.5 => /lib/x86_64-linux-gnu/liblzma.so.5
        liblz4.so.1 => /lib/x86_64-linux-gnu/liblz4.so.1
        libgcrypt.so.20 => /lib/x86_64-linux-gnu/libgcrypt.so.20
            libgpg-error.so.0 => /lib/x86_64-linux-gnu/libgpg-error.so.0
libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6
libX11.so.6 => /lib/x86_64-linux-gnu/libX11.so.6
libXcomposite.so.1 => /lib/x86_64-linux-gnu/libXcomposite.so.1
libXdamage.so.1 => /lib/x86_64-linux-gnu/libXdamage.so.1
libXext.so.6 => /lib/x86_64-linux-gnu/libXext.so.6
libXfixes.so.3 => /lib/x86_64-linux-gnu/libXfixes.so.3
libXrandr.so.2 => /lib/x86_64-linux-gnu/libXrandr.so.2
    libXrender.so.1 => /lib/x86_64-linux-gnu/libXrender.so.1
libgbm.so.1 => /lib/x86_64-linux-gnu/libgbm.so.1
    libwayland-server.so.0 => /lib/x86_64-linux-gnu/libwayland-server.so.0
libxcb.so.1 => /lib/x86_64-linux-gnu/libxcb.so.1
    libXau.so.6 => /lib/x86_64-linux-gnu/libXau.so.6
    libXdmcp.so.6 => /lib/x86_64-linux-gnu/libXdmcp.so.6
        libbsd.so.0 => /lib/x86_64-linux-gnu/libbsd.so.0
libxkbcommon.so.0 => /lib/x86_64-linux-gnu/libxkbcommon.so.0
libpango-1.0.so.0 => /lib/x86_64-linux-gnu/libpango-1.0.so.0
    libfribidi.so.0 => /lib/x86_64-linux-gnu/libfribidi.so.0
    libthai.so.0 => /lib/x86_64-linux-gnu/libthai.so.0
        libdatrie.so.1 => /lib/x86_64-linux-gnu/libdatrie.so.1
    libharfbuzz.so.0 => /lib/x86_64-linux-gnu/libharfbuzz.so.0
        libfreetype.so.6 => /lib/x86_64-linux-gnu/libfreetype.so.6
            libpng16.so.16 => /lib/x86_64-linux-gnu/libpng16.so.16
        libgraphite2.so.3 => /lib/x86_64-linux-gnu/libgraphite2.so.3
libcairo.so.2 => /lib/x86_64-linux-gnu/libcairo.so.2
    libpixman-1.so.0 => /lib/x86_64-linux-gnu/libpixman-1.so.0
    libfontconfig.so.1 => /lib/x86_64-linux-gnu/libfontconfig.so.1
        libuuid.so.1 => /lib/x86_64-linux-gnu/libuuid.so.1
    libxcb-shm.so.0 => /lib/x86_64-linux-gnu/libxcb-shm.so.0
    libxcb-render.so.0 => /lib/x86_64-linux-gnu/libxcb-render.so.0
libasound.so.2 => /lib/x86_64-linux-gnu/libasound.so.2
libatspi.so.0 => /lib/x86_64-linux-gnu/libatspi.so.0
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
ld-linux-x86-64.so.2 => /lib64/ld-linux-x86-64.so.2
```

A note on #include…

When you write #include <vector> …

It copies and pastes the code from a file called “vector” into your code exactly where you wrote #include

5 min break

Next up:

- Project structure
- CMake

Project Structure

```
my_project/
    include/
        my_project/
            utilities.h
            logging.h
            geometry/
                shapes.h
                area.h
    src/
        geometry/
            shapes.cpp
            area.cpp
        kinematics.cpp
        main.cpp
```

CMake

compile commands

```
g++ main.cpp lib.cpp -o my_program
```

CMake

```
add_executable(my_program main.cpp lib.cpp)
```

CMake

compile commands

```
g++ lib.cpp -o lib.a

g++ main.cpp lib.a -o my_program
```

CMake

```
add_library(lib lib.cpp)

add_executable(my_program main.cpp)

target_link_libraries(my_program PUBLIC lib)
```

CMake

The real power is when the project gets big, and has external dependencies! … also, it works on Windows!

```
my_project/
      include/
            my_project/
                  utilities.h
                  logging.h
                  geometry/
                        shapes.h
                        area.h
      src/
            geometry/
                  shapes.cpp
                  area.cpp
            kinematics.cpp
            main.cpp
```

compile commands

```
g++ src/geometry/shapes.cpp src/geometry/area.cpp
     -o build/libgeometry.a


g++ src/kinematics.cpp -o build/libkinematics.a


g++ main.cpp -o main
```

CMake

The real power is when the project gets big, and has external dependencies! … also, it works on Windows!

compile commands

```
g++ src/geometry/shapes.cpp
    src/geometry/area.cpp
    -o build/libgeometry.a
g++ src/kinematics.cpp
    -o build/libkinematics.a
g++ main.cpp -o main
```

CMake

```
add_library(geometry src/geometry/shapes.cpp
    src/geometry/area.cpp)
add_library(kinematics src/kinematics.cpp)
add_executable(main main.cpp)
target_link_libraries(main geometry kinematics)
```

CMake

two commands you always need at the top of the main CMakeLists.txt:

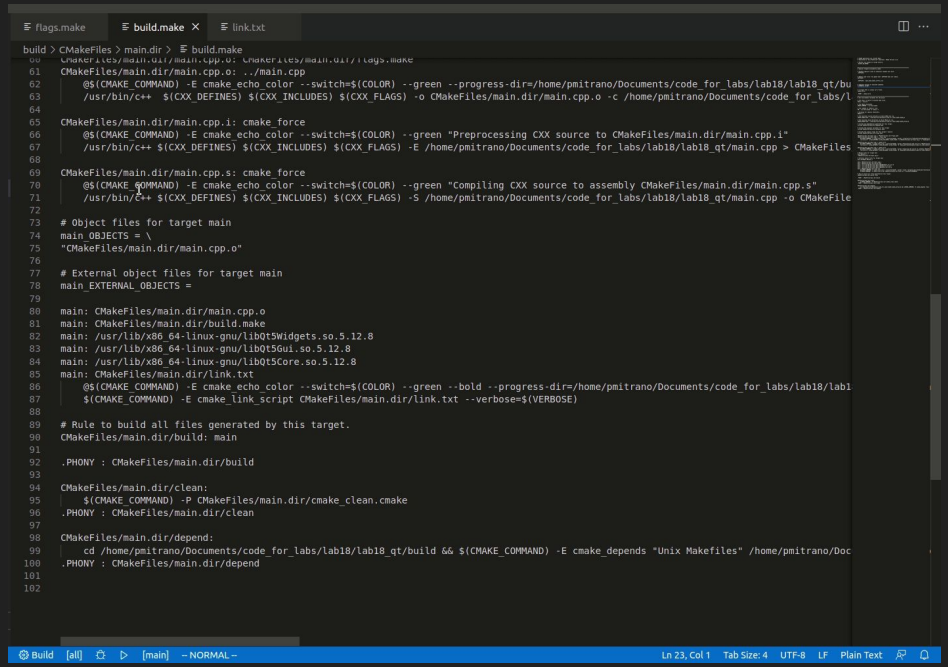
```
cmake_minimum_required(3.10)   # this is a version number

project(my_project_name)
```

CMake

The real power is when the project gets big, and has external dependencies! … also, it works on Windows!

Look how much you'd have to write to do use a library like Qt…



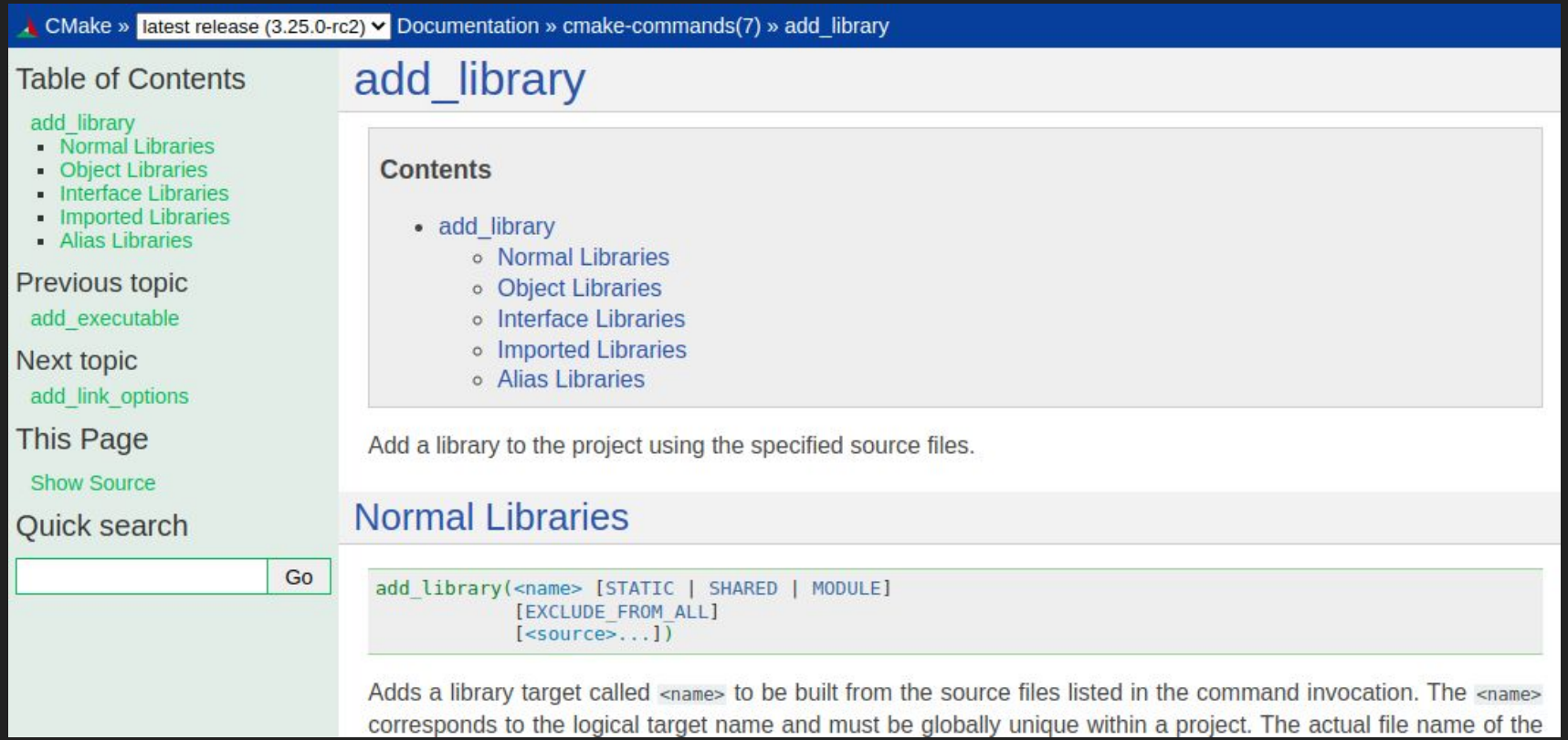
Include Paths

```
my_project/
    include/
        my_project/
            utilities.h
            logging.h
            geometry/
                shapes.h
                area.h
    src/
        geometry/
            shapes.cpp
            area.cpp
        kinematics.cpp
        main.cpp
```

```
#include <include/my_project/utilities.h>
#include <my_project/utilities.h>
#include <../utilities.h>
#include <../my_project/utilities.h>
```

Which should we use?

Reading the CMake Documentation



CMake in VSCode

- We will go over this once together
- There are tutorials online if you forget
  - https://code.visualstudio.com/docs/cpp/cmake-linux

5 minute break

Next up:

- Lab exercises

Homework:

- Homework 5
- Read Type deduction for objects using the auto keyword – Learn C++