CMake

Introduction

In the last two assignments we dealt with:

- writing tests using catch2
- code coverage using gcov, lcov

We saw that compiling, running the program and generating the coverage information takes a sequence of commands which can be tedious and error prone. In this assignment, we will use **CMake** to manage the build process of our program.

Minimal example for CMake

- This problem is meant to be a quick start for CMake.
- Create a directory called **problem1** and inside it create the following directories:
  - **src** to store all source files
  - **tests** to keep all source files for the tests
  - **include** to keep all header files
  - **build** to store the executable and the coverage information
- We also need a starting point for our program. Create a file `main.cpp` and put it in the **problem1** directory. The following is the code for `main.cpp`

```
#include "hello_world.h"
```

```
int main()
{
cout << "Hello World from main.cpp" << endl;
cout << print_hello_world(true);
return 0;
}
```

- As you can see, we included our own header file called `hello_world.h`. Usually, header files should only contain declarations. Let's create it in the **include** directory.

```
#include <iostream>

using namespace std;

bool print_hello_world(bool);
```

- This header file contains a function declaration `bool print_hello_world(bool);`, we need to define it. Let's define it in the **src** folder.
- Create a new file called `hello_world.cpp` in the **src** folder and use the following code.

```
#include "hello_world.h"
bool print_hello_world(bool print)
{
if (print)
{
cout << "Hello World from hello_world.cpp" << endl;
}
else
{
cout << "No Hello World from hello_world.cpp" << endl;
}
```

```
cout << "Hi world" << endl;
return true;
}
```

- We can now use *CMake* to compile and run this minimal example. For this, we need to create a new configuration file for *CMake*. Create a file called *CMakeLists.txt* in the root directory i.e., **problem1** directory with the following content.

```
# Required for compatibility reasons
cmake_minimum_required(VERSION 3.10)

# Name of the project and the release version
project(HelloWorldProject VERSION 1.0)

# This specifies the C++ standard version we want our project to be
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# We are including the header files required
include_directories(include)

# The following specifies the name of the executable - 'HelloWorldProject'
add_executable(HelloWorldProject main.cpp src/hello_world.cpp)
```

- We can now compile and run the project. For this:
  - Go to the **build** directory and run the following commands
  - `cmake ..`
  - `make`
  - Run the executable `./HelloWorldProject` which should print out the following:

```
Hello World from main.cpp
Hello World from hello_world.cpp
Hi world
```

- We created a very simple minimal example for using *CMake*. You can now make changes to the code and simply use `make` to build the project and create the executable. This makes the overall workflow much simpler.

Adding Tests

- Now let's integrate tests to our workflow. In a previous assignment, we included `catch.hpp` to write tests. Copy the provided `catch.hpp` to the **include** folder.
- Now let's write tests. Create a new file in the **tests** directory called `test_hello_world.cpp` and write the following content.

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

#include "hello_world.h"

TEST_CASE("Hello World from hello_world.cpp", "[hello_world]")
{
REQUIRE(print_hello_world(true) == true);
}
```

This tests the function `print_hello_world` written in the **src** folder. We need to integrate this to *CMake* configuration. For this we need to add the new lines to the *CMakeLists.txt* file. The updated configuration looks like the following:

```
cmake_minimum_required(VERSION 3.10)
# Name of the project and the release version
project(HelloWorldProject VERSION 1.0)

# This specifies the C++ standard version we want our project to be in
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED True)

# We are including the header files required
include_directories(include)

# The following specifies the name of the executable
add_executable(HelloWorldProject main.cpp src/hello_world.cpp)

# Added the following lines for integrating tests
add_executable(HelloWorldTest tests/test_hello_world.cpp src/hello_world.cpp)

enable_testing()

add_test(NAME HelloWorldTest COMMAND HelloWorldTest)
```

- Rest of the steps is similar as before:
  - Go to the **build** directory and run the following commands
  - `cmake ..`
  - `make`
  - Now run the tests using `./HelloWorldTest`
  - *Tip* You can also use the command `ctest` to run the tests instead of using the executable file `./HelloWorldTest`

Adding Code Coverage

The next step is to integrate code coverage to our minimal example. If you recall, for this we need to build our project using the flags `-fprofile-arcs -ftest-coverage`. Let's add it to *CMake*.

We can do so by adding the following lines to *CMakeLists.txt* ideally below other `set()` parts.

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fprofile-arcs -ftest-coverage")
set(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} -fprofile-arcs -ftest-coverage")
```

We can build and run the program which will now generate `.gcno` and `.gcda` files.

Integrating lcov and genHTML

Next, we add the `lcov` and `genhtml` command sequence as a task to *CMakeLists.txt*.

```
# Adding coverage task

find_program(LCOV_PATH lcov)
find_program(GENHTML_PATH genhtml)

add_custom_target(coverage
    COMMAND ${LCOV_PATH} --capture --directory . --output-file coverage.info
    COMMAND ${LCOV_PATH} --remove coverage.info '/usr/*' '*/tests/*' '*/cmake_tests/include/*' --output-file
coverage_filtered.info
    COMMAND ${GENHTML_PATH} coverage_filtered.info --output-directory coverage_report
    WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
    COMMENT "Creating coverage"
)
```

We can run this task using the command `make coverage`

Bonus

Since finding and deleting the `.gcno` and `.gcda` files can be tedious, we can also create a task to delete them as follows:

```
add_custom_target(coverage_clean
    COMMAND find . -name "*.gcda" -delete
    COMMAND find . -name "*.gcno" -delete
    WORKING_DIRECTORY ${CMAKE_BINARY_DIR}
    COMMENT "Deleted the coverage files"
)
```

This can be run using `make coverage_clean`