Concepts of C++ Programming
winter term 2023

Technical University of Munich
CIIP Group

Exercise sheet 2

2023-10-26

Due date: 2023-11-02 16:59

The goal of this exercise sheet is to get you used to `CMake` and the basics of C++ .

Don't desperate when dealing with `CMake` - it's important to understand how C++ projects and its dependencies are organized, to **prevent chaos** and ensure long-time maintainability in bigger, real world projects.

Exercise 1:

The goal is to build the project by creating the missing `hw02/CMakeLists.txt` file, and discovering `doctest` with `CMake`.

After updating your C++ tasks `git` repo with the latest homework, you'll see that we now have added files for `CMake`. Instead of building files manually (like in the last homework), we now have a proper build system.

Before your written code is compiled, you run `cmake` so it can check dependencies and then generate all the build files. All the resulting build files are stored in a separate directory, usually at `your-repo/build`.

Follow these steps to build your project. Make sure to **read and understand error messages**:

- `cd your-repo-root` Enter your `tasks` repo
- `mkdir build` Create the directory where **build files** will be stored in
- `cd build`
- `cmake ..` **Generate** the build files. This **will fail** for these reasons:
  - `doctest` could not be found: follow the doctest setup section
  - `tasks/hw02` does not contain a CMakeLists.txt file: you need to create it!

Have a look at the last part of this exercise to see how you can run your code afterwards.

i) Set up `doctest` for CMake usage

In the last homework we manually specified `doctest`'s include path with `-I some-path`. This time, we instruct `cmake` to automatically include the `doctest` location.

We use `cmake`'s `find_package` feature to locate `doctest`.

In case your system has a **proper package manager** (Linux, macOS Homebrew, ...), just install `doctest` there, then **you're done** with this step:

- Install on Ubuntu/Debian: `doctest-dev`, Arch: `doctest`, Gentoo: `dev-cpp/doctest`, Homebrew: `doctest`, ...
- If `doctest` is installed like this, `cmake` will automatically discover its system-wide installation when running `cmake ..`. So easy!

If you did/could not install `doctest` properly (`cmake ..` still complains about not finding `doctest`), you have to build `doctest` manually:

- You should have the repo from the last homework.
  If not, get it with `git clone` `https://github.com/doctest/doctest`
- Inside the `doctest` project, the build tool is also `cmake`, so the build steps in there are **pretty much the same** as with your homework!
- To build and install `doctest`, execute this in the `doctest` project root directory:
  - `cd doctest-project-root` - go to the doctest git repo
  - `mkdir build install` - create two directories
  - `cd build`
  - `cmake ..` - run cmake and generate build files for `doctest`
  - `make` - compile the `doctest` code
  - `make install DESTDIR=../install` - install doctest to this directory
- Now you have installed `doctest` into `doctest-project-root/install/`, so we can tell the homework's `cmake` configuration about the `doctest` location so it can use it:
  - switch to your task repo: `cd your-task-repo/build`
  - Now we can try to build the homework again, but append the `doctest` installation path prefix to the `cmake ..` invocation:
    `cmake .. -Ddoctest_ROOT=doctest-project-root/install/usr/local/`
  - If the `doctest` finding succeeds, you've taken this hurdle:
    `cmake` will now complain that `tasks/hw02/CMakeLists.txt` is missing.

ii) Create your Homework's `CMakeLists.txt`

The `cmake` execution will now fail at a later step:
```
The source directory ...tasks/hw02/ does not contain a CMakeLists.txt file.
```

So you have to create this file!

Needed ingredients:

- The lecture slides ("Important CMake commands")
- This week's exercise slides
- The CMake documentation

With these ingredients, create `hw02/CMakeLists.txt` so it can compile the `hw02/` C++ files:

- Define a **library** named *hw02* which contains the code of `combinatorics.cpp`
- Register `${CMAKE_CURRENT_SOURCE_DIR}` as **include directory** for *hw02* so headers in that directory can be included via `<headername.h>`
- Define an **executable** named *runhw02* for the code of `run.cpp`
- **Link** the library *hw02* into *runhw02*

There's no need to modify the `your-task-repo/CMakeLists.txt` and anything within the *tests/* directory. On the contrary - doing so will trigger pipeline failure!

iii) Execute the freshly built code

If your build system is now happy, you can continue to actually compile your code:

- `make` Build your C++ homework code
  - Use `make VERBOSE=1` to see exact compiler invocations!
- `./hw02/runhw02` Execute your code

You can also build our testing code:

- `make testhw02` Build our test code
- `./tests/hw02/testhw02` Run our test code

Exercise 2:

Implement a program to compute permutations and combinations

In `combinatorics.cpp`, complete the implementation of the `permutation` and `combination` functions.

$$P(n,k) = \frac{n!}{(n-k)!} \qquad C(n,k) = \frac{n!}{k!\,(n-k)!}$$

- To execute your `run.cpp`, build (`make runhw02`) and run `./hw02/runhw02`
- All the tests in `./tests/hw02/testhw02` should pass - this is checked in the pipeline.
- Commit and push your changes and verify the pipeline is passing!