CMake Quick Start from Scratch

This guide will show you how to use CMake to build a simple C project.

Why use CMake?

For multi-file projects, it can be a hassle to manually compile and link all the source files.

If we use `gcc` to compile a project with multiple source files, we might have to run the following commands:

```
gcc -Wall -o helloworld helloworld.c add.c
```

This can get tedious very quickly, especially for larger projects if source files are spread across multiple directories.

CMake can automate this process for us. It generates the necessary build files for the platform being used (e.g. `Makefile` for Linux, Visual Studio project files for Windows, etc.) so that you can build your project with a single command.

Prerequisites

Basic requirements:

- CMake installed on your system
- C compiler (`gcc` & `g++`, `clang` & `clang++` or `msvc`) installed on your system
- Terminal or command prompt
- Text editor (e.g. VS Code)

Recommended:

- `Ubuntu 22.04 LTS` on physical or virtual machine (like VMWare/WSL)
- `gcc` and `g++` installed on your system
- Visual Studio Code (VS Code) with `C/C++` extension pack (which includes `CMake Tools` extension)

Step 1: Create a CMakeLists.txt file

If you already have a project folder with CMakeLists.txt file, you can skip this step.

If you don't have a project, you can create a new directory for your project and add a few source files. For example, you could create a directory called `hello` and add the following files:

```
hello/
├── CMakeLists.txt
├── add.c
├── add.h
└── helloworld.c
```

Create a file named `CMakeLists.txt` in the root directory of your project. This file will contain the instructions for CMake to build your project.

Add the following content to the `CMakeLists.txt` file:

```
cmake_minimum_required(VERSION 3.0)

project(hello)

add_executable(hello_out helloworld.c add.c)

if(MSVC)
  target_compile_options(hello_out PRIVATE /W4)
else()
  target_compile_options(hello_out PRIVATE -Wall)
endif()
```

This file contains the following commands:

- `cmake_minimum_required(VERSION 3.0)`: This command specifies the minimum version of CMake required to build the project. In this case, we require version 3.0 or later.
- `project(hello)`: This command sets the name of the project. In this case, the project is called `hello`.
- `add_executable(hello_out helloworld.c add.c)`: This command tells CMake to create an executable called `hello_out` from the source files `helloworld.c` and `add.c`. You can add more source files here if needed. (No need to add header files here, since they are "included" in the source files)
- `if(MSVC) ... else() ... endif()`: This command sets compiler-specific options. In this case, we are setting the warning level to `/W4` (equivalent for `-Wall`) for MSVC (Microsoft Visual Studio) and `-Wall` for other compilers (e.g. g++).

In the example above, `helloworld.c` is a simple C++ program that prints "Hello, world!" to the console. You can create this file with the following content:

```
#include <stdio.h>

#include "add.h"

int main()
{
    printf("Hello, world! 1+1=%d\n", add(1, 1));
    return 0;
}
```

You can also create a header file called `add.h` with the following content:

```
#ifndef __ADD_H__
#define __ADD_H__

int add(int a, int b);

#endif
```

And a source file called `add.c` with the following content:

```
int add(int a, int b)
{
    return a + b;
}
```

Step 2: Configure the project

Open a terminal and navigate to the root directory of your project. Run the following command to configure the project:

```
mkdir build
cd build
cmake ..
```

This command tells CMake to generate the necessary build files for the project. By default, CMake will generate build files for the platform you are using (e.g. Makefiles for Linux) into the `build` directory (current directory).

The `..` argument tells CMake to look for the `CMakeLists.txt` file in the parent directory of the current directory (i.e. the root directory of your project)

Step 3: Build the project

After configuring the project, you can build it by running the following command (assuming you are still in the `build` directory, where the `CMakeCache.txt` file is located):

```
cmake --build .
```

This command tells CMake to build the project using the generated build files (in the current directory). After running this command, you should see an executable file called `hello_out` in the `build` directory of your project.

Step 4: Run the program

You can run the program by executing the following command from the `build` directory of your project:

```
./hello_out
```

This should print "Hello, world! 1+1=2" to the console.

VS Code Integration

If you are using VS Code, you can use the `CMake Tools` extension to build and run your project directly from the editor.

After installing the extension, open the root directory of your project in VS Code.

Normally VS Code will ask you to configure the project at the bottom-right corner of the window when you open the folder for the first time. You can click "Yes" if your CMake has been correctly installed. This will automatically configure the project for you.

If it doesn't, you can manually configure the project by pressing `Ctrl+Shift+P` to open the command palette, then search for `cmake configure` and press `Enter`.

After configuring the project, you can **build** it by pressing `F7`.

You can also **run or debug** the program by pressing `Ctrl+F5` (run) or `Shift+F5` (debug) respectively.

You can also use the `Build` / `run` (play) / `debug` (bug) button in the status bar at the bottom-left corner of the window to perform these actions.

Conclusion

In this guide, you learned how to use CMake to build a simple C++ project. You created a `CMakeLists.txt` file to specify the build instructions for the project, configured the project using CMake, built the project using the generated build files, and ran the resulting program.

References

- CMake Documentation
- Github Copilot (for generating this guide automatically)