C++ 개발 환경 구성과 CMake

C++ with CMake

Sangkon Han

January 25, 2025

기본적인 CMake 사용법 및 CLI를 통한 CMake 이해

간단한 C++ 프로젝트

CMake를 사용해서 실행 파일을 만드는 과정을 통해 CMake의 기본적인 사용법을 연습하는 간단한 과정을 소개해드리겠습니다.

- 소스 코드: `src/main.cpp` 파일에 "Hello, World!"를 출력하는 C++ 코드를 작성합니다.
- CMake 설정: `CMakeLists.txt` 파일에 CMake 프로젝트 설정을 정의하고, 실행 파일 타겟을 생성하고 소스 파일을 연결합니다.
- 빌드: 명령 줄에서 CMake를 사용하여 프로젝트를 구성하고 빌드합니다.
- 실행: 빌드된 실행 파일을 실행하여 "Hello, World!"가 출력되는지 확인합니다.

C++ 코드와 CMake 작성

- `src/main.cpp`

이 파일은 표준 출력 스트림을 사용하여 "Hello, World!"를 출력하는 간단한 C++ 프로그램입니다.

```cpp
#include <iostream>

int main() {
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

- `CMakeLists.txt` 파일
  - `cmake_minimum_required(VERSION 3.21)`: 이 명령어는 합니다. 3.21 버전 이상에서 이 프로젝트를 빌드해야 합니다.
  - `project(...)`: 프로젝트 이름, 버전, 설명, 사용 언어 등을 설정합니다.
  - `add_executable(hello_world)`: `hello_world`라는 실행 파일 타겟을 생성합니다.

- `target_sources(hello_world PRIVATE src/main.cpp)`: `hello_world` 타겟에 소스 파일 `src/main.cpp`를 연결합니다. `PRIVATE` 키워드는 이 소스 파일이 현재 타겟에서만 사용됨을 의미합니다.

```
cmake_minimum_required(VERSION 3.21)

project(HelloWorld VERSION 1.0
  DESCRIPTION "A simple Hello World project with CMake"
  LANGUAGES CXX
)

add_executable(hello_world)
target_sources(hello_world PRIVATE src/main.cpp)
```

빌드 및 실행

다음은 명령 줄에서 프로젝트를 빌드하고 실행하는 단계입니다.

- 빌드 디렉토리 생성, `build` 디렉토리를 생성하고 이동합니다. 이 디렉토리에서 빌드 관련 파일들이 생성됩니다.

```
mkdir build
cd build
```

- CMake 구성, 상위 디렉토리(`..`)에 있는 `CMakeLists.txt` 파일을 사용하여 빌드 시스템을 구성합니다.

```
  cmake ..
```

- 프로젝트 빌드, 현재 디렉토리(`.`)에 있는 빌드 파일을 사용하여 프로젝트를 빌드합니다. 이 명령어는 실제 컴파일 및 링크 과정을 수행합니다.

```
cmake --build .
```

- 실행 파일 실행, `build` 디렉토리 안에 생성된 `hello_world` 실행 파일을 실행합니다.

```
  ./hello_world
```

정적 라이브러리

```
/
├── CMakeLists.txt
├── staticlib/
│   ├── CMakeLists.txt
│   ├── include/
│   │   └── lib/
│   │       └── add.h
│   └── src/
│       └── add.cpp
```

```
├─── sharedlib/
│    ├─── CMakeLists.txt
│    ├─── include/
│    │    └─── sharedlib/
│    │         └─── sharedlib.h
│    └─── src/
│         └─── sharedlib.cpp
└─── app/
     ├─── CMakeLists.txt
     └─── src/
          └─── main.cpp
```

정적 라이브러리 소스 파일

- lib/src/add.cpp

```cpp
#include "lib/add.h"

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

- lib/src/include/lib.h

```cpp
#pragma once
int add(int a, int b);
```

- dynamiclib/CMakeLists.txt
  - add_library(staticlib STATIC src/add.cpp): lib라는 정적 라이브러리 타겟을 생성하고, 소스 파일 src/add.cpp를 연결합니다. STATIC 키워드는 정적 라이브러리를 의미합니다.
  - target_include_directories(staticlib PUBLIC include): lib 타겟에 헤더 파일 검색 경로를 추가합니다. PUBLIC 키워드는 이 라이브러리를 사용하는 타겟도 이 경로를 사용할 수 있음을 의미합니다.

```cmake
add_library(staticlib STATIC src/add.cpp)
target_include_directories(lib PUBLIC include)
```

공유 라이브러리 예제

- ‘sharedlib/src/sharedlib.cpp

```cpp
#include "sharedlib/sharedlib.h"

int multiply(int a, int b) {
    return a * b;
}
```

- sharedlib/include/sharedlib/sharedlib.h

```
#pragma once
int multiply(int a, int b);
```

- sharedlib/CMakeLists.txt
- add_library(sharedlib SHARED src/sharedlib.cpp): sharedlib라는 공유 라이브러리 타겟을 생성하고, 소스 파일 src/sharedlib.cpp를 연결합니다. SHARED 키워드는 공유 라이브러리를 의미합니다.
- target_include_directories(sharedlib PUBLIC include): sharedlib 타겟에 헤더 파일 검색 경로를 추가합니다. PUBLIC 키워드는 이 라이브러리를 사용하는 타겟도 이 경로를 사용할 수 있음을 의미합니다.

```
add_library(sharedlib SHARED src/sharedlib.cpp)
target_include_directories(sharedlib PUBLIC include)
```

실행 파일 예제

- app/src/main.cpp

```
#include <iostream>
#include "lib/lib.h"
#include "sharedlib/sharedlib.h"

int main() {
    int a = 5, b = 3;

    std::cout << "Addition: " << add(a, b) << std::endl;
    std::cout << "Multiplication: " << multiply(a, b) << std::endl;

    return 0;
}
```

- app/CMakeLists.txt
  - add_executable(_app src/main.cpp): _app이라는 실행 파일 타겟을 생성하고, 소스 파일 src/main.cpp를 연결합니다.
  - target_link_libraries(_app PRIVATE lib sharedlib): _app 타겟에 lib와 sharedlib 라이브러리를 연결합니다. PRIVATE 키워드는 이 라이브러리들이 _app 타겟 내부에서만 사용됨을 의미합니다.
  - target_include_directories(_app PRIVATE ...): _app 타겟에 lib 및 sharedlib의 헤더 파일 검색 경로를 추가합니다.

```
add_executable(_app src/main.cpp)

target_link_libraries(_app
    PRIVATE
    lib
    sharedlib
```

```
)

target_include_directories(_app
    PRIVATE
    ${CMAKE_CURRENT_SOURCE_DIR}/../lib/include
    ${CMAKE_CURRENT_SOURCE_DIR}/../sharedlib/include
)
```

- CMakeLists.txt
  - add_subdirectory() 명령을 사용하여 각 하위 디렉토리에 있는 CMakeLists.txt 파일을 포함합니다.

```
cmake_minimum_required(VERSION 3.21)
project(LibsExample VERSION 1.0
    DESCRIPTION "A simple example demonstrating static and shared libraries"
    LANGUAGES CXX
)

add_subdirectory(lib)
add_subdirectory(sharedlib)
add_subdirectory(app)
```

빌드 및 실행

- 빌드 디렉토리 생성

```
mkdir build
cd build
```

- CMake 구성

```
cmake ..
```

- 프로젝트 빌드

```
cmake --build .
```

- 실행 파일 실행:

```
./app/_app
```

7. 실행 결과

```
Addition: 8
Multiplication: 15
```