Tổng quan

CMake là công cụ quản lý xây dựng mã nguồn mở, giúp tự động hóa quá trình biên dịch và liên kết các dự án phần mềm. Dưới đây là hướng dẫn cơ bản về cách sử dụng CMake trong các dự án Nhúng.

Lưu ý rằng ở đây, tài liệu đã giả định người đọc có kiến thức về Makefile và sử dụng GNU Toolchain cơ bản như GCC, GDB.

Setup CMake trong dự án

Giả sử chúng ta có 1 cấu trúc dự án như sau;

```
project-root/
├── CMakeLists.txt
├── main.c
└── include/
```

Trong đó, `CMakeLists.txt` là tệp cấu hình chính của CMake.

Cấu trúc nội bộ CMakeLists.txt

Dưới đây là một ví dụ đơn giản về nội dung của `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.13) #Yêu cầu phiên bản CMake
tối thiểu
project(myprj C) #Đặt tên dự án và ngôn ngữ lập trình
add_executable(myprj main.c) #Chỉ định tệp nguồn để biên dịch
```

Best Practice trong sử dụng CMake

Nên bổ sung 1 thư mục `build` riêng biệt bên trong dự án để chứa các tệp biên dịch, giúp giữ cho thư mục gốc sạch sẽ:

```
project-root/
├── build/
```

```
├── CMakeLists.txt
├── main.c
└── include/
```

Biên dịch dự án với CMake

1. Redirect vào thư mục `build`:
   ```
   cd project-root/build
   ```
2. Chạy lệnh CMake để tạo các tệp Makefile:
   ```
   cmake ..
   ```
   Lúc này CMake sẽ đọc tệp `CMakeLists.txt` từ thư mục gốc và tạo các tệp cần thiết trong thư mục `build`.
3. Chạy lệnh `make` để biên dịch dự án: `make`

Sử dụng biến trong CMake

CMake hỗ trợ sử dụng biến để tùy chỉnh quá trình xây dựng. Ví dụ:

```
project(myprj C)
```

Khi cấu hình, CMake sẽ tồn tại biến `PROJECT_NAME` với giá trị "myprj" để sử dụng.
Một ví dụ sử dụng chính là khi bạn muốn đặt tên tệp thực thi dựa trên tên dự án:

```
add_executable(${PROJECT_NAME} main.c)
```

Import thư viện STATIC trong biên dịch với CMake

Giả sử có một thư viện đã có implement sẵn như sau:

```
project-root/
├── build/
```

```
├── CMakeLists.txt
├── main.c
├── lib.c
├── lib.h
└── include/
```

Bạn có thể thêm nó vào CMakeList.txt:

```
add_library(mylib STATIC lib.c) #Thêm thư viện tĩnh
```

Sau đó liên kết thư viện này với tệp thực thi:

```
target_link_libraries(${PROJECT_NAME} mylib)
```

Import với subdirectory trong CMake

Với cấu trúc dự án phức tạp hơn, ta có thể thiết kế cấu trúc thư mục như sau để tổ chức mã nguồn:

```
project-root/
├── build/
├── CMakeLists.txt
├── main.c
└── include/
    └── lib.h
    └── lib.c
```

Lúc này, ta có thể sử dụng lệnh `add_subdirectory` để thêm thư mục con vào quá trình biên dịch và loại bỏ `add_library` trong tệp gốc CMakeLists.txt:

```
add_subdirectory(include) #Thêm thư mục con
```

Trong thư mục con này cũng sẽ bổ sung 1 tệp CMakeLists.txt riêng:

```
project-root/
├── build/
├── CMakeLists.txt
├── main.c
└── include/
    ├── CMakeLists.txt
    └── lib.h
    └── lib.c
```

Ở tệp CMakeLists.txt trong thư mục con, ta sẽ khai báo thư viện:

```
add_library(mylib STATIC lib.c) #Thêm thư viện tĩnh
target_include_directories(mylib PUBLIC
${CMAKE_CURRENT_SOURCE_DIR})
#Chỉ định thư mục include,
# trong đó CMAKE_CURRENT_SOURCE_DIR là biến chỉ đến thư mục hiện tại
# PUBLIC để chỉ định rằng thư mục này sẽ được sử dụng bởi các mục tiêu liên kết với mylib, có thể sử dụng các từ khóa khác như PRIVATE, INTERFACE
#  PRIVATE chỉ sử dụng nội bộ trong thư viện, INTERFACE chỉ sử dụng cho các mục tiêu liên kết
```

So sánh giữa CMake chính và CMake phụ

Để có cái nhìn tổng quan hơn, ta có thể so sánh giữa CMake chính (gốc) và CMake phụ (thư mục con) như sau:

CMake chính

```
cmake_minimum_required(VERSION 3.13)
project(myprj C)
add_executable(${PROJECT_NAME} main.c)
add_subdirectory(include)
target_link_libraries(${PROJECT_NAME} mylib)
```

CMake chính sẽ là nơi thực hiện các bước cấu hình tổng thể của dự án, bao gồm:

- Đặt tên dự án và ngôn ngữ lập trình.
- Chỉ định tệp nguồn chính để biên dịch.
- Thêm các thư mục con thông qua `add_subdirectory`.
- Liên kết các thư viện với tệp thực thi chính.

CMake phụ

```
add_library(mylib STATIC lib.c)
target_include_directories(mylib PUBLIC
${CMAKE_CURRENT_SOURCE_DIR})
```

CMake phụ sẽ tập trung vào việc định nghĩa các thành phần cụ thể của dự án, bao gồm:

- Tạo các thư viện từ các tệp nguồn cụ thể.
- Chỉ định các thư mục include cần thiết cho thư viện.

Cụ thể hơn, CMake phụ là nơi cấu hình access modifier (PUBLIC, PRIVATE, INTERFACE) cho các thư viện, giúp kiểm soát cách các thư viện này được sử dụng trong toàn bộ dự án.