

I. Cmake



1. Định nghĩa: Cmake là gì?

CMake là công cụ để từ các file mã nguồn, nó tạo ra bộ project tương ứng, kèm luôn công cụ biên tập code, macro, tạo các tệp và không gian làm việc gốc có thể được sử dụng trong môi trường trình biên dịch mà bạn chọn bằng phương pháp độc lập với trình biên dịch. Nó được thiết kế để hỗ trợ phân cấp thư mục và các ứng dụng phụ thuộc vào nhiều thư viện. Từ đó có thể build tiếp project đó.













Tóm lại: Chúng ta có một cấu hình và hoạt động cho many môi trường.

2. Phương pháp hoạt động:

- a) Nó **KHÔNG** thực sự biên dịch bất cứ điều gì. Để sử dụng CMake, chúng ta phải nói với nó (sử dụng các tệp cấu hình có tên CMakeLists.txt) những gì cần biên dịch, thư viện nào họ liên kết đến, thư mục nào có trong dự án.
- b) Ý tưởng của CMake là **viết một script có tên CMakeLists.txt** và chạy nó trên platform mà bạn muốn sinh project files, ta chạy trên Linux, CMake sẽ sinh Makefile ra để ta build project, chạy trên Windows, CMake sẽ sinh solution cho VS, và vân vân.

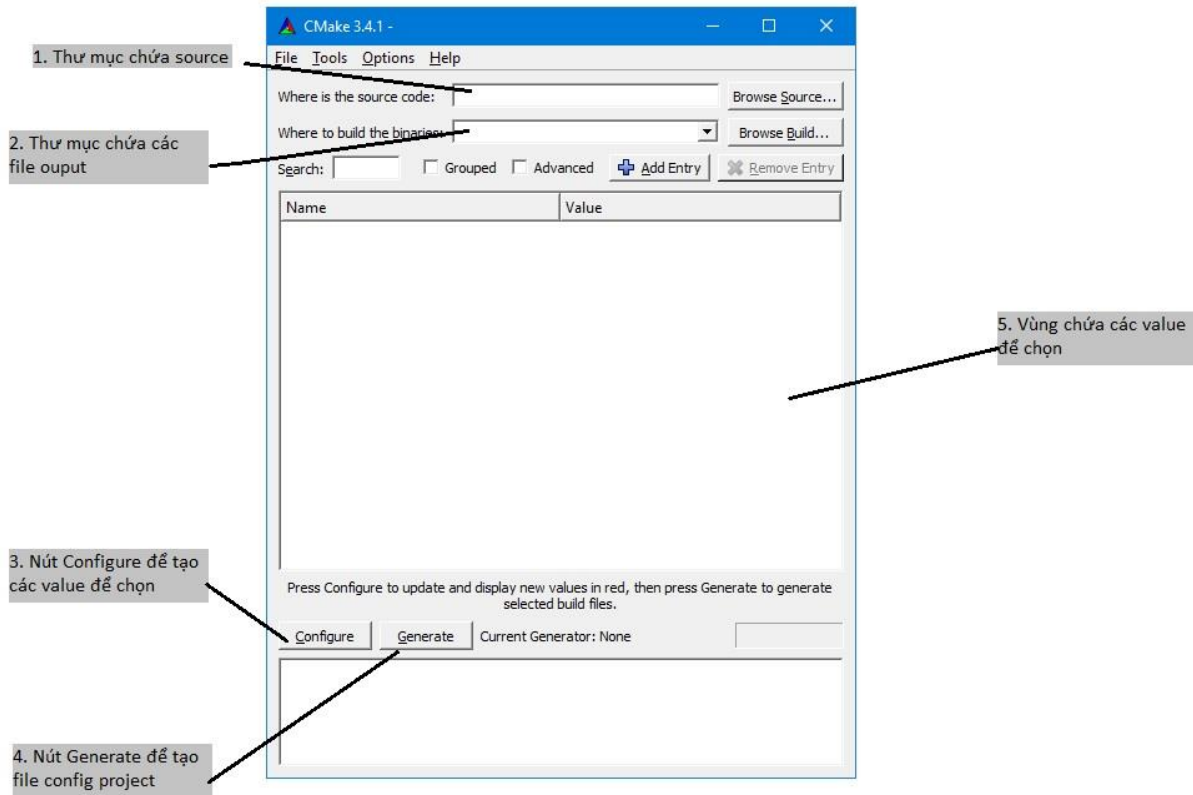
Name	Date modified	Type	Size
 Main.cpp	6/8/2018 4:48 PM	C++ source file	0 KB
 CMakeLists.txt	6/8/2018 4:56 PM	Text Document	1 KB

- c) CMake thiết lập dự án theo **rootCMakeLists.txt** và thực hiện trong bất kỳ thư mục nào đã thực hiện cmake từ trong bảng điều khiển. Làm điều này **từ một thư mục không phải là gốc** của dự án của ta tạo ra cái được gọi là ngoài nguồn build, có nghĩa là các tệp được tạo trong quá trình biên dịch (tệp obj, tệp lib, tệp thực thi...) sẽ được đặt trong thư mục đã nói, tách biệt với mã thực tế

Hello World > Build >				Search Build
Name	Date modified	Type	Size	
 CMakeFiles	6/8/2018 5:10 PM	File folder		
 ALL_BUILD.vcxproj	6/8/2018 5:10 PM	VC++ Project	17 KB	
 ALL_BUILD.vcxproj.filters	6/8/2018 5:10 PM	VC++ Project Filte...	1 KB	
 cmake_install.cmake	6/8/2018 5:10 PM	CMAKE File	3 KB	
 CMakeCache.txt	6/8/2018 5:10 PM	Text Document	12 KB	
 HelloWorld.sln	6/8/2018 5:10 PM	Microsoft Visual S...	5 KB	
 HelloWorld.vcxproj	6/8/2018 5:10 PM	VC++ Project	28 KB	
 HelloWorld.vcxproj.filters	6/8/2018 5:10 PM	VC++ Project Filte...	1 KB	
 INSTALL.vcxproj	6/8/2018 5:10 PM	VC++ Project	11 KB	
 INSTALL.vcxproj.filters	6/8/2018 5:10 PM	VC++ Project Filte...	1 KB	
 ZERO_CHECK.vcxproj	6/8/2018 5:10 PM	VC++ Project	16 KB	
 ZERO_CHECK.vcxproj.filters	6/8/2018 5:10 PM	VC++ Project Filte...	1 KB	

3. Phân loại

Cmake có 2 loại: Command line và GUI: command line dùng dòng lệnh để truyền tham số, còn GUI thì bạn chỉ việc click và click



(Giao diện của Cmake GUI)

Giao diện của chương trình Cmake GUI:

- 1. Thư mục chứa source mà cta download về
 - 2. Thư mục chứa các file output, ở đây là các file config của project như là *.sln, *.vcxproj, vcxproj.filters,...
 - 3. Ấn nút Generate để tạo ra các file config với nhiều tùy chọn khác nhau trên các IDE. Chương trình sẽ đọc nội dung của file **CMakeLists.txt**, rồi kiểm tra các đường dẫn, tạo ra các tùy chọn khác,...
- Thí dụ C++ thì sẽ chọn phiên bản Visual Studio, tạo ra các mode Debug, Release, chọn các lib hỗ trợ,... thì tùy theo từng lib khác nhau mình sẽ viết kỹ hơn. Nếu bước này xảy ra lỗi thì phải kiểm tra để khắc phục.
- 4. Sau khi lựa chọn xong các option thì ấn nút Generate để tạo các file config cho project

4. Ưu điểm lớn nhất

Các tính năng mạnh của CMake

Những tính năng được gọi là mạnh của CMake, có thể kể đến 2 tính năng sau:

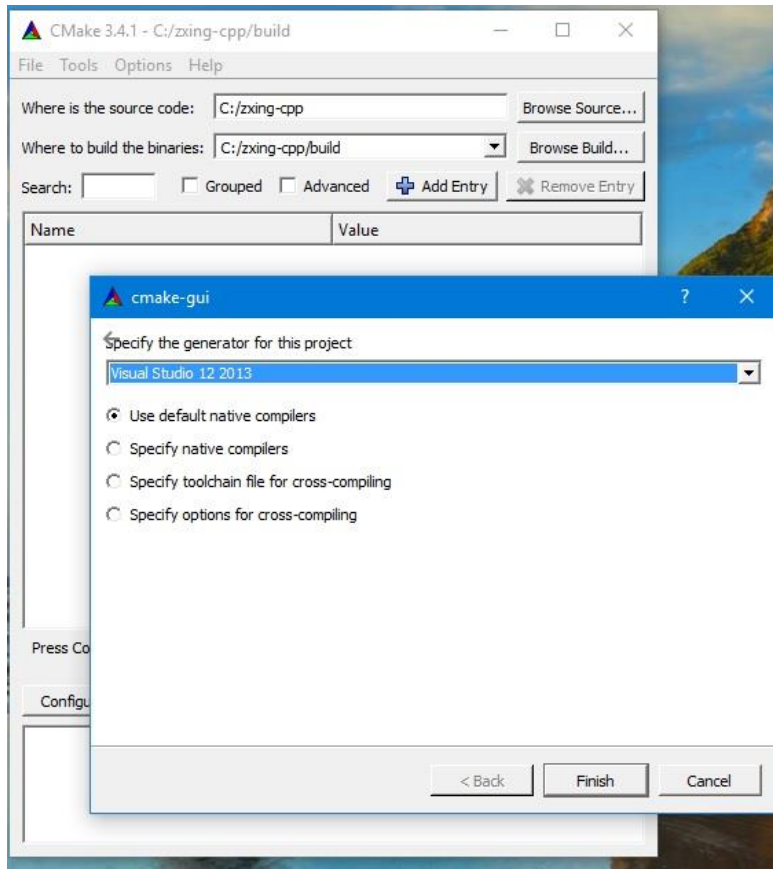
- Tạo ra kịch bản build cho ra hiệu suất làm việc cao trên các hệ thống nó support.

- Tạo ra kịch bản build mà tách riêng với source code, (tính năng này có từ Autotools), điều hành đặc biệt hữu ích khi ta muốn thực hiện nhiều kịch bản build trên cùng 1 source code.

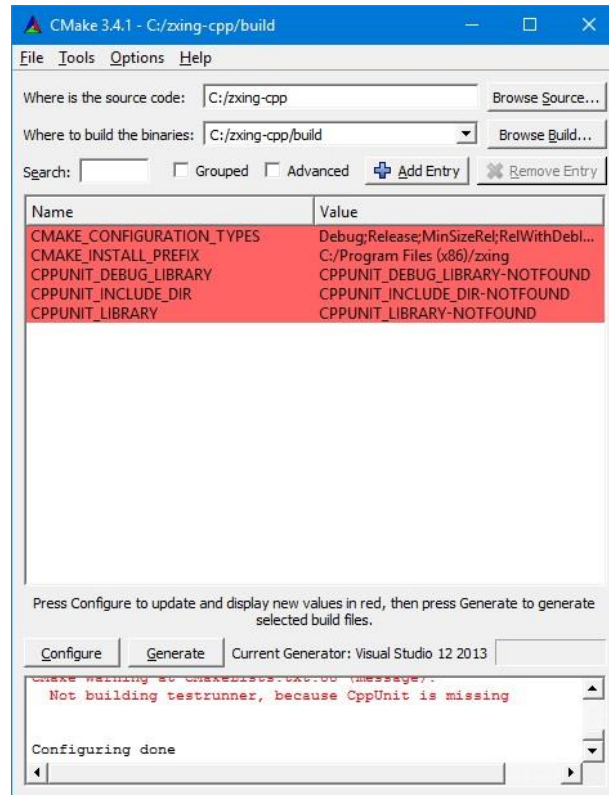
(Ta thấy trên Window, các project thường chứa luôn source code trong đó, nhưng CMake tạo ra project file mà các file source ở một chỗ khác)

5. Ví dụ :

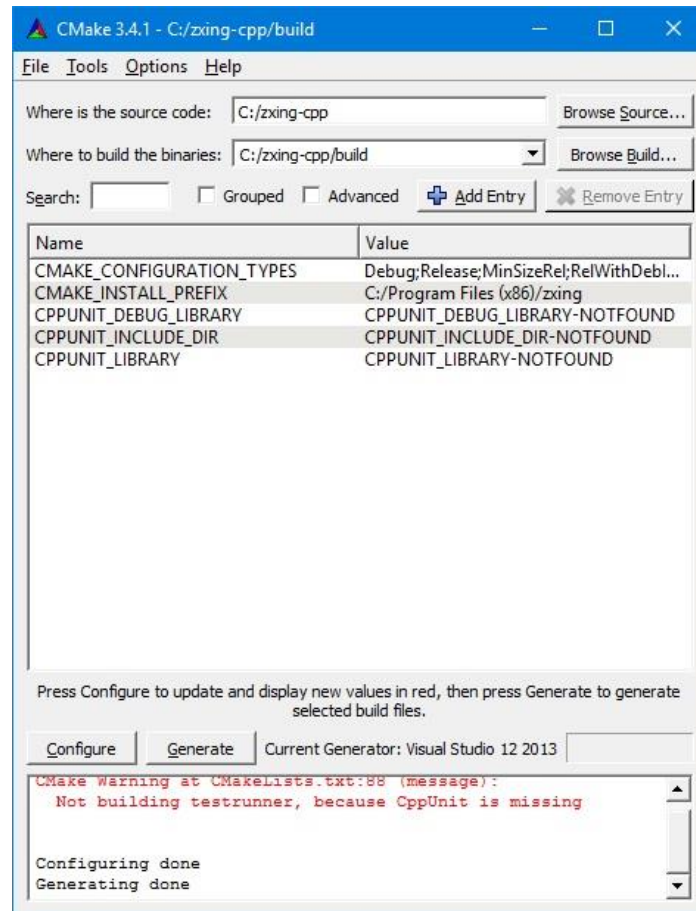
1. Đầu tiên download source code [Zxing cpp](#) về. Giải nén ra 1 thư mục nào đó, mình chọn thư mục **C:\Zxing-cpp**
2. Mở chương trình Cmake GUI, chọn thư mục chứa source là **C:\Zxing-cpp**
3. Chọn thư mục chứa file output là **C:\Zxing-cpp\build**. Thư mục này các bạn nên chọn nằm trong thư mục source hoặc ngay bên cạnh cho tiện.



(Lưu output ở thư mục build trong thư mục gốc)



(Phần bôi đỏ là chọn các Value để Configue)



(Sau khi ọc nội dung của file **CMakeLists.txt**, rồi kiểm tra các đường dẫn, tạo ra các tùy chọn khác, nếu bước này xảy ra lỗi phải sửa lại ngay, bước này là chọn xong các option và đã Generate các file config để build project)

II. Makefile

1) Make file là gì?

- Makefile là một file dạng script chứa các thông tin:
- Cấu trúc project (file, sự phụ thuộc)
- Các lệnh để tạo file
- Lệnh make sẽ đọc nội dung Makefile, hiểu kiến trúc của project và thực thi các lệnh

2) Targets và Prerequisite(Hiểu như là input và output)

```
1 | target: prereq1 prereq2
2 |     commands
```

Makefile

- Target: file bạn cần make
- prerequisites hay dependent: là những file cần phải tồn tại trước khi target được tạo thành công
- command: là shell command tạo ra target từ prerequisites

ví dụ:

```
1 | foo.o: foo.c foo.h
2 |     gcc -c foo.c
```

Target ở đây là foo.o nằm trước dấu hai chấm, prerequisites là foo.c và foo.h, command là gcc -c foo.c, trước command là **tab**

3) Rule trong makefile:

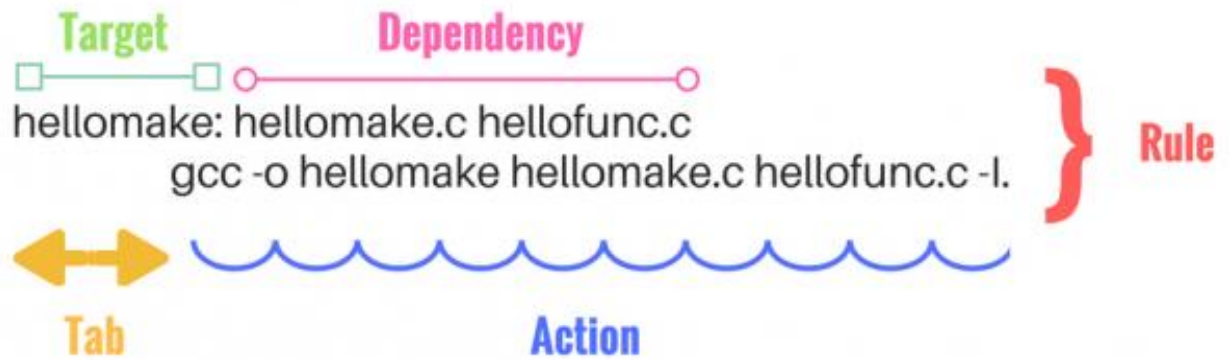
Trong đó mỗi rule sẽ xác định một target. Mỗi file target này thì phụ thuộc vào việc thiết lập prerequisite. Khi được yêu cầu update target, make sẽ thực thi các command script của các rule nếu prerequisite của các prerequisite này có sự thay đổi.

Vì target của một rule có thể là tham chiếu như một prerequisite trong rule khác, tập hợp các target và prerequisite tạo thành một chuỗi hoặc biểu đồ phụ thuộc (dependency graph). Xây dựng và xử lý dependency graph này để cập nhật target là tất cả những gì make phải làm.

Bởi vì rule rất quan trọng trong make, do đó có rất nhiều rule khác nhau: link dưới là tìm hiểu rất kĩ về rule

<https://hocarm.org/rule-trong-makefile/>

TÓM LẠI:



- **Rule**: các rule cần thực hiện khi compile
- **Dependency**: là các file cần thiết để tạo ra target
- **Action**: là câu lệnh compile để tạo ra Target từ Dependency. Action được thụt lùi vào 1 Tab (phím tab trên bàn phím) so với Target
- **Target**: là file đích, nghĩa là file được hình thành sau khi quá trình make được thực hiện.

Tìm hiểu kĩ hơn về Makefile 1,2,3 ở đây:

<https://hocarm.org/makefile-lagi/#M%E1%BB%99ts%E1%BB%91v%C3%AD%E1%BB%A5c%C6%A1b%E1%BA%A3n>

III. Ưu và nhược điểm của CMake và Makefile

- Tiện ích **cmake** và **Makefiles** cung cấp một hệ thống build mà chúng ta có thể sử dụng để quản lý việc compile và re-compilation của một chương trình được viết bằng ngôn ngữ bất kỳ. Việc sử dụng Makefiles đôi khi lại có thể trở thành một công việc phức tạp trong trường hợp project mà chúng ta build có nhiều sub directories hoặc sẽ phải triển khai trên nhiều nền tảng khác nhau.
- Để khắc phục điều đó thì CMake ra đời. CMake là một công cụ sinh Makefile đa nền tảng. Nói đơn giản thì CMake sẽ tự động tạo ra Makefiles cho project của chúng ta. Ngoài ra thì nó cũng làm được

nhiều hơn nhưng trong khuôn khổ bài viết thì mình sẽ chỉ tập trung vào việc tự động sinh Makefiles cho các project C/C++.

IV. Bazel là gì?

Bazel là một công cụ tự động hóa việc xây dựng và kiểm tra phần mềm. Các tác vụ xây dựng được hỗ trợ bao gồm chạy trình biên dịch và trình liên kết để tạo ra các chương trình và thư viện thực thi, đồng thời lắp ráp các gói có thể triển khai cho Android, iOS và các môi trường mục tiêu khác. Bazel tương tự như các công cụ khác như Make, Ant, Gradle, Buck, Pants và Maven.

1. Ưu điểm Bazel:

Bazel được thiết kế để phù hợp với cách phát triển phần mềm tại Google. Nó có các tính năng sau:

- Hỗ trợ đa ngôn ngữ: Bazel hỗ trợ nhiều ngôn ngữ, và có thể mở rộng để hỗ trợ các ngôn ngữ lập trình tùy ý.
- Ngôn ngữ xây dựng cấp cao: Các dự án được mô tả bằng ngôn ngữ BUILD, một định dạng văn bản ngắn gọn mô tả một dự án như một tập hợp các thư viện nhỏ, các tệp nhị phân và các bài kiểm tra được kết nối với nhau. Ngược lại, với các công cụ như Make, bạn phải mô tả các tệp riêng lẻ và các lệnh gọi trình biên dịch.
- Hỗ trợ đa nền tảng: Có thể sử dụng cùng một công cụ và các tệp BUILD giống nhau để xây dựng phần mềm cho các kiến trúc khác nhau và thậm chí cả các nền tảng khác nhau. Tại Google, chúng tôi sử dụng Bazel để xây dựng mọi thứ từ các ứng dụng máy chủ chạy trên các hệ thống trong trung tâm dữ liệu của chúng tôi đến các ứng dụng khách chạy trên điện thoại di động.
- Khả năng tái tạo: Trong các tệp BUILD, mỗi thư viện, thử nghiệm và nhị phân phải chỉ định hoàn toàn các phụ thuộc trực tiếp của nó. Bazel sử dụng thông tin phụ thuộc này để biết những gì phải được xây dựng lại khi bạn thực hiện thay đổi đối với tệp nguồn và tác vụ nào có thể chạy song song. Điều này có nghĩa là tất cả các bản dựng đều tăng dần và sẽ luôn tạo ra cùng một kết quả.

- Có thể mở rộng: Bazel có thể xử lý các bản dựng lớn; tại Google, thông thường một máy chủ nhị phân có 100 nghìn tệp nguồn và các bản dựng không có tệp nào được thay đổi sẽ mất khoảng ~ 200ms.

2. Ví dụ Bazel:

Đúng. Để biết một ví dụ đơn giản, hãy xem:

<https://github.com/bazelbuild/bazel/blob/master/examples/cpp/BUILD>

3. Hạn Chế của Bazel:

- Bazel thông minh về bộ nhớ đệm. Điều này có nghĩa là nó không tốt cho việc chạy các hoạt động xây dựng có kết quả đầu ra không được lưu vào bộ nhớ đệm(khác với Cmake) . Ví dụ: không nên chạy các bước sau từ Bazel:
 - Bước biên dịch tìm nạp dữ liệu từ internet.
 - Bước kiểm tra kết nối với phiên bản QA của trang web của bạn.
 - Bước triển khai thay đổi cấu hình đám mây của trang web của bạn.
- Nếu quá trình xây dựng của bạn bao gồm một vài bước dài, tuần tự, Bazel có thể không giúp được gì nhiều. Chúng ta sẽ nhận được nhiều tốc độ hơn bằng cách chia các bước dài thành các mục tiêu nhỏ hơn, rời rạc mà Bazel có thể chạy song song.

