# MAKEFILE

## GIỚI THIỆU

### Đặt vấn đề

#### Những vấn đề khi biên dịch

* Một chương trình đơn giản Chỉ có một vài file
* Một chương trình “không đơn giản”
* Nhiều dòng lệnh
* Nhiều module
* Nhiều người tham gia viết

#### Vấn đề xảy ra

* Khó quản lý một file lớn (cả người và máy).
* Mỗi thay đổi cần thời gian biên dịch lâu.
* Nhiều người lập trình không thể thay đổi cùng một file đồng thời
* Chương trình được phân ra thành nhiều module

#### Giải pháp

Chia project ra thành nhiều file.

#### Mục tiêu

* Chia project ra thành các modul một cách đúng đắn.
* Thời gian biên dịch phải ngắn nếu có sự thay đổi
* Dễ dàng bảo trì cấu trúc project

### Makefile

#### Makefile 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

#### Cấu trúc project

* Cấu trúc và sự phụ thuộc của project có thể được biểu diễn bằng một DAG (Directed Acyclic Graph)

**Ví dụ:**

* Chương trình chứa 3 file: **main.c**, **sum.c**, **sum.h**
* File sum.h được dùng bởi cả 2 file **main.c** và **sum.c**
* File thực thi là **sum**

### Một số ví dụ cơ bản

#### Ví dụ với makefile cơ bản.

Chương trình đơn giản in ra dòng Hello makefiles được viết thành 3 file.

|  |  |
| --- | --- |
| hellomake.c | |
| 1  2  3  4  5  6  7  8 | #include <hellomake.h>  int main() {  // call a function in another file  myPrintHelloMake();  return(0);  } |

|  |  |
| --- | --- |
| hellofunc.c | |
| 1  2  3  4  5  6  7 | #include <stdio.h>  #include <hellomake.h>  void myPrintHelloMake(void) {  printf("Hello makefiles!\n");  return;  } |

|  |  |
| --- | --- |
| hellomake.h | |
| 1  2  3  4 | /\*  example include file  \*/  void myPrintHelloMake(void); |

Để bắt đầu chúng ta cần có 3 file bỏ chung vào 1 thư mục là:

* hellomake.c tương ứng chương trình chính.
* hellofunc.c là file hàm in thông báo.
* hellomake.h là file header khai báo hàm in.

Thông thường chúng ta có thể compile code và xem kết quả một cách đơn giản bằng lệnh sau:

|  |  |
| --- | --- |
| 1  2 | $ gcc -o hellomake hellomake.c hellofunc.c -I.  $ ./hellomake |

Lệnh này sẽ thực hiện compile 2 file .c. -I. có nghĩa là include gcc sẽ thực hiện tìm kiếm trong thư mục hiện tại(.) để thêm file hellomake.h.

Nếu không có makefile thì mỗi compile chúng ta lại phải mở terminal lên và gõ lệnh gcc -o … vào, điều này trở nên quá rắc rối, đặc biệt là khi chúng ta add thêm nhiều file .c khác vào trong chương trình hoặc khi chúng ta sửa lại nội dung code trong các file .c

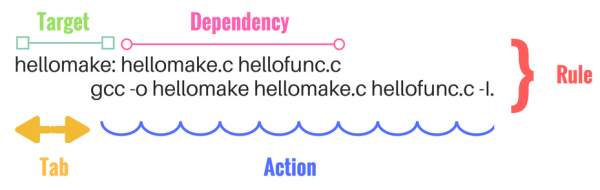
#### Vậy makefile sẽ khắc phục hạn chế ở trên như thế nào ?

##### Makefile 1

Trước tiên để dùng được makefile thì phải tạo file có tên là Makefile hoặc makefile trong thư mục chứa code hiện có với nội dung:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CC=gcc  CFLAGS=-I.  hellomake: hellomake.c hellofunc.c  $(CC) -o hellomake hellomake.c hellofunc.c -I. |

Một cấu trúc make file cơ bản sẽ có dạng cơ bản như sau:



Trong đó:

* **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 lùi vào 1 Tab so với Target
* **Target**: là file đích, nghĩa là file được hình thành sau khi quá trình make được thực hiện.

Vậy là trong thư mục chúng ta sẽ có 4 file: hellofunc.c hellomake.c hellomake.h và Makefile

Chạy chương trình bằng lệnh sau:

|  |  |
| --- | --- |
| 1  2 | $ make  $ ./hellomake |

##### Makefile 2

|  |  |
| --- | --- |
| 1  2  3  4  5 | CC=gcc  CFLAGS=-I.  hellomake: hellomake.o hellofunc.o  $(CC) -o hellomake hellomake.o hellofunc.o -I. |

Chúng ta tiếp tục phát triển thêm một chút bằng cách thêm CC và CFLAGS vào Makefile ở trên.

Trong đó:

* CC: là compiler C được sử dụng
* CFLAGS: là danh sách các flag của compiler.

Có một điểm khác nữa là thêm 2 file object là: hellomake.o và hellofunc.o trong dependency list và trong rule để make biết rằng đây là lần đầu tiên của quá trình biên dịch.

Với việc sử dụng makefile như trên thì đã có thể làm được các project nhỏ nhỏ rồi. Tuy nhiên vẫn còn thiếu dependency là các file include. Giả sử như ta có thay đổi trên file hellomake.h thì make sẽ không biên dịch lại file .c. Để khắc phục lỗi này thì ta cần phải thông báo cho make rằng tất cả các file .c đều bị phụ thuộc vào file .h, khi compile nhớ phải lưu ý nha.

##### Makefile 3

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CC=gcc  CFLAGS=-I.  DEPS = hellomake.h  %.o: %.c $(DEPS)  $(CC) -c -o $@ $< $(CFLAGS)  hellomake: hellomake.o hellofunc.o  gcc -o hellomake hellomake.o hellofunc.o -I. |

Makefile này sẽ tạo ra một macro là DEPS để chỉ ra file .h mà các file .c phụ thuộc vào. Ngoài ra sẽ có một định nghĩa về rule áp dụng cho tất cả các file .o, rule này thông báo rằng các file .o phụ thuộc vào các file .c và .h được định nghĩa trong macro là DEPS (dòng 5). Rule sẽ tạo ra file .o, make được dùng C compile được định nghĩa trong CC để compile các file c.(dòng 6)

Có một số lưu ý:

* -c là tạo ra các object file
* $@ là tạo ra output của quá trình biên dịch trong tập tin bên trái dấu :
* $< là thành phần đàu tiên trong danh sách của dependency và CFLAGS là macro đã được định nghĩa ở dòng 2

Tiếp tục bước cuối cùng là sử dụng các macro đặc biệt như $@ và @^ để lấy thông tin bên trái và bên phải của dấu : để làm cho quá trình biên dịch được tổng quát hơn, cụ thể là các file include sẽ được đưa vào trong macro DEPS, tất cả các object file được đưa vào macro OBJ như makefile 4

##### Makefile 4

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | CC=gcc  CFLAGS=-I.  DEPS = hellomake.h  OBJ = hellomake.o hellofunc.o  %.o: %.c $(DEPS)  $(CC) -c -o $@ $< $(CFLAGS)  hellomake: $(OBJ)  gcc -o $@ $^ $(CFLAGS) |

Ta sẽ thấy một số file được sinh ra sau quá trình make như hình dưới:

Graphical user interface, application, Word

Description automatically generated