

C PROGRAMMING

Người trình bày: Phạm Hồng Thi

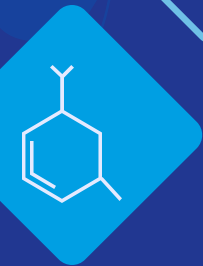




Table of contents

01

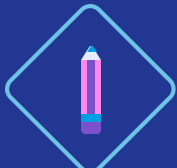
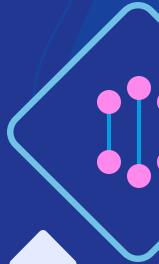
STRUCT, UNION, ENUM

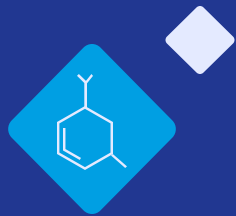
02

LINKER

03

MAKE FILE





01

STRUCT, UNION, ENUM



Cấu trúc người dùng tự định nghĩa



1. STRUCT: Definition

Kiểu dữ liệu do người dùng tự định nghĩa, tập hợp các biến (có thể khác loại) dưới một tên duy nhất.

```
1 struct structureName
2 {
3     dataType member1;
4     dataType member2;
5     ...
6 };
7
8 struct structureName name1, name2;
```


```
1 typedef struct structureName
2 {
3     dataType member1;
4     dataType member2;
5     ...
6 } name_t;
7
8 name_t name1, name2;
```

1. STRUCT: Access member

Hai toán tử để truy xuất tới các biến thành viên của kiểu struct trong C

- . => Toán tử truy xuất tới thành phần khi khai báo biến bình thường
- -> => Toán tử truy xuất tới thành phần khi khai báo biến con trỏ

```
4 ▸ typedef struct PhanSo{  
5     int tu;  
6     int mau;  
7 } PS;  
8  
9 ▸ void main(){  
10     PS ps1 = {1, 2};  
11     PS * ps_ptr = &ps1;  
12     printf("tu: %d \nmau: %d\n", ps1.tu, ps1.mau);  
13     printf("tu: %d \nmau: %d\n", ps_ptr->tu, ps_ptr->mau);  
14 }
```



tu: 1
mau: 2
tu: 1 |
mau: 2

1. STRUCT: Initialization

Danh sách khởi tạo

```
struct structure_name str = { value1, value2, value3 };
```

Danh sách khởi tạo chỉ định từng phần tử

```
struct structure_name str = { .member1 = value1, .member2 = value2, .member3 = value3 };
```

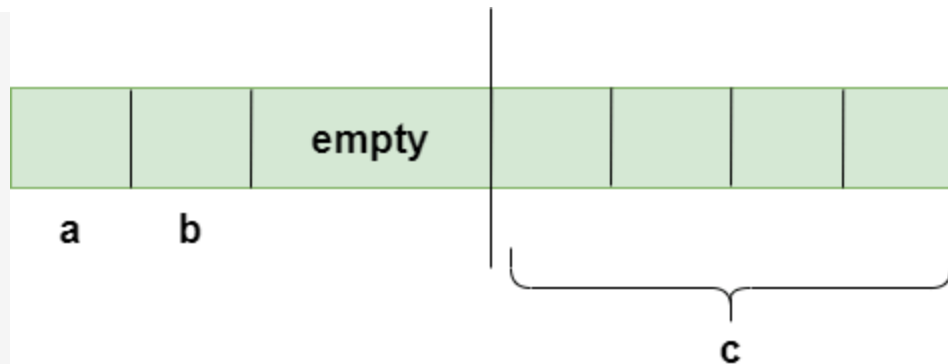
Toán tử gán

```
struct structure_name str;  
str.member1 = value1;  
str.member2 = value2;  
str.member3 = value3;  
.  
.  
.
```

1. STRUCT: PADDING

Để tối ưu hiệu suất cho CPU (đọc theo chu kỳ), trình biên dịch thực hiện chèn dữ liệu một cách tự động

```
2 ▾ typedef struct structa_tag {  
3     char a;  
4     char b;  
5     int c;  
6 } structA_t;  
7  
8 ▾ typedef struct structb_tag {  
9     char c;  
10    double d;  
11    int s;  
12 } structB_t;  
13  
14 int main()  
15 ▾ {  
16     printf("sizeof(structa_t) = %lu\n", sizeof(structA_t));  
17     printf("sizeof(structb_t) = %lu\n", sizeof(structB_t));  
18     return 0;  
19 }
```



`sizeof(structa_t) = 8`
`sizeof(structb_t) = 24`


1. STRUCT: PACKING

Để tránh sự lãng phí bộ nhớ, ta sử dụng `#pragma pack(1)` để chỉ dẫn cho trình biên dịch đặt cấu trúc tự định nghĩa vào bộ nhớ một cách liên tục mà không căn chỉnh

```
1  #include <stdio.h>
2  #pragma pack(1)
3  struct Example {
4      char c;
5      int i;
6      double d;
7  };
8
9  int main() {
10     printf("Size of struct: %d\n", sizeof(struct Example));
11     return 0;
12 }
```

2 // #pragma pack(1) Size of struct: 16


Size of struct Example: 13



1. STRUCT: PACKING

Kết hợp sử dụng `#pragma pack(push,n)` và `#pragma (pop)` để điều chỉnh bộ đệm tùy mục đích

```
2~ struct A {  
3    char c;  
4    int i;  
5 };  
6 #pragma pack(push, 1)  
7~ struct B {  
8    char c;  
9    int i;  
10 };  
11 #pragma pack(pop)  
12 #pragma pack(push, 2)  
13~ struct C {  
14    char c;  
15    int i;  
16 };  
17 #pragma pack(pop)  
18~ int main() {  
19    printf("Size of struct A: %lu\n", sizeof(struct A));  
20    printf("Size of struct B: %lu\n", sizeof(struct B));  
21    printf("Size of struct C: %lu\n", sizeof(struct C));
```



Size of struct A: 8
Size of struct B: 5
Size of struct C: 6

1. STRUCT: PACKING

Sử dụng `__attribute__((packed))` để yêu cầu trình biên dịch đặt căn chỉnh tối đa về nhỏ nhất có thể

- 1 byte cho 1 biến
- 1 bit cho 1 trường bit


```
1  #include <stdio.h>
2  //#pragma pack(1)
3  typedef struct __attribute__((packed)) A{
4      char a;
5      int b;
6      char c;
7      short d;
8  } structA_t;
9
10 int main() {
11     printf("size of struct A: %d\n", sizeof
        (structA_t));
12     return 0;
13 }
```

→ size of struct A: 8

1. STRUCT: ALIGNED

Sử dụng `__attribute__((aligned(n)))` để yêu cầu trình biên dịch đặt căn chỉnh tối thiểu về một số byte cụ thể

```
1  #include <stdio.h>
2  // #pragma pack(1)
3  typedef struct __attribute__((aligned(32)))
4      A{
5      char a;
6      int b;
7      char c;
8      short d;
9  } structA_t;
10
11 int main() {
12     printf("size of struct A: %d\n", sizeof
13         (structA_t));
14     return 0;
15 }
```




size of struct A: 32

1. STRUCT: with pointer

Dịch chuyển con trỏ trong struct

```
1  #include <stdio.h>
2  // #pragma pack(1)
3  typedef struct struca_tag{
4      char a;
5      int b;
6      char c;
7      short d;
8  } structA_t;
9
10 int main() {
11     structA_t a = {'1', 2, 'x', 4};
12     structA_t *ptr = &a;
13     printf("size of ptr: %d\n", sizeof(ptr));
14     printf("character: %c\n", *((char*)ptr+8));
15     return 0;
16 }
```



size of ptr: 8
character: x

1. STRUCT: with function pointer

```
3 ▸ typedef struct {
4     float result;
5     float (*operation)(float a, float b);
6 } OP_t;
7
8 ▸ float Add (float a , float b){
9     return a+b;
10 }
11
12 ▸ float Multi ( float a , float b ){
13
14     return a*b ;
15 }
16 ▸ int main() {
17     OP_t op;
18     op.operation = &Add;
19     op.result = op.operation(3.33,2.22);
20     printf("result: %.3f\n",op.result);
21     return 0;
```

→ result: 5.550

1. UNION: Definition

Kiểu dữ liệu cho phép lưu trữ các kiểu dữ liệu khác nhau trong cùng một vị trí bộ nhớ.


⇒ Có thể lưu trữ dữ liệu khác “kiểu và kích thước” tại các thời điểm khác nhau.

```
9 union [union tag] {  
10     member definition;  
11     member definition;  
12     ...  
13     member definition;  
14 } [one or more union variables];
```

1. UNION: Size

Memory cấp phát cho union = kích thước thành viên lớn nhất

```
4 union Data1 {  
5     int i;  
6     float f;  
7     char str[20];  
8 };  
9  
10 struct Data2 {  
11     int i;  
12     float f;  
13     char str[20];  
14 };  
15  
16 int main( ) {  
17  
18     union Data1 data1;  
19     struct Data2 data2;  
20     printf( "Memory size occupied by Union : %ld\n", sizeof(data1));  
21     printf( "Memory size occupied by Struct : %ld\n", sizeof(data2));  
22 }
```




Memory size occupied by Union : 20
Memory size occupied by Struct : 28

1. UNION: Storage data

Lưu trữ duy nhất trường dữ liệu cuối cùng được đọc/ghi

```
4 union Data {  
5     int i;  
6     float f;  
7     char str[20];  
8 };  
9  
10 int main( ) {  
11  
12     union Data data;  
13  
14     data.i = 10;  
15     data.f = 220.5;  
16     strcpy( data.str, "C Programming");  
17  
18     printf( "data.i : %d\n", data.i);  
19     printf( "data.f : %f\n", data.f);  
20     printf( "data.str : %s\n", data.str);
```



```
data.i : 1917853763  
data.f : 4122360580327794860452759994368.000000  
data.str : C Programming
```


1. UNION: Storage data

```
4 union Data {  
5     short value1;  
6     int value2;  
7 };  
8  
9 int main( ) {  
10  
11     union Data data;  
12  
13     data.value2 = 16843012; //hex: 0101 0104  
14     //data.value1 = 1;      //hex: 0000 0001  
15  
16     printf( "data.value2 : %d\n", data.value2);  
17     printf( "data.value1 : %d\n", data.value1);  
18 }
```

```
13 data.value2 = 16843012; //hex: 0101 0104  
14 data.value1 = 1;        //hex: 0000 0001
```

data.value2 : 16843012
data.value1 : 260

data.value2 : 16842753
data.value1 : 1

Little Endian

High address
01
01
01
04

00
01


Value2 (4byte)

Value1 (2byte)

1. UNION: Application

Tạo một mảng gồm nhiều loại dữ liệu

```
1  #include <stdio.h>
2  // #pragma pack(1) //bỏ sung tiền chỉ thị
3  typedef union {
4      int a;
5      char b;
6      double c;
7  } data;
8
9  int main()
10 {
11     data arr[10];
12     arr[0].a=10;
13     arr[1].b = 'x';
14     arr[2].c = 10.191;
15     printf("size of arr: %d\n",sizeof(arr));
16     printf("arr[0]: %d\narr[1]: %c\narr[2]: %.3f\n",arr[0].a,arr[1].b,arr[2].c);
```



size of arr: 80
arr[0]: 10
arr[1]: x
arr[2]: 10.191

1. UNION: Application

Kết hợp union và struct để lưu dữ liệu thành ghi.

```
12 typedef union{
13     struct {
14         uint8_t DIV: 8;
15         uint16_t reverse1: 8;
16         uint16_t reverse2: 8;
17         uint8_t BCD: 1;
18         uint8_t BCP: 1;
19         uint8_t MSEL: 2;
20         uint8_t BCI: 1;
21         uint8_t BCS: 1;
22         uint8_t SYNC: 2;
23     } TCR2;
24     uint32_t value;
25 } check;
26 int main()
27 {
28     check check1;
29     check1.value = 0xFF0000FF;
30     printf("%ld\n", sizeof(check));
31     printf("%x\n", check1.value);           //0xFF0000FF
32     printf("%x\n", check1.TCR2.DIV);        //0xFF
33     printf("%x\n", check1.TCR2.reverse1);   //0x00
34     printf("%x\n", check1.TCR2.reverse2);   //0x00
35     printf("%x\n", check1.TCR2.BCD);        //0b01
36     printf("%x\n", check1.TCR2.MSEL);       //0b11
37     printf("%x\n", check1.TCR2.SYNC);       //0b11
```

Address: 4002_F000h base + 8h offset = 4002_F008h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SYNC								0							
W	BCS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0								DIV							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2Sx_TCR2 field descriptions

```
4
ff0000ff
ff
0
0
1
3
3
```

1. ENUM: Definition

Kiểu dữ liệu cố định tập hợp một nhóm các hằng số, chỉ cho phép biến nhận số lượng giá trị nhất định nào đó.


```
enum enum_name{int_const1, int_const2, int_const3, ..., int_constN};
```

Compiler sẽ tự động gán giá trị cho các phần tử không được khởi tạo giá trị.

Ngoại trừ phần tử đầu tiên trong **enum** được gán =0 , những hằng số khác sẽ được gán giá trị bằng phần tử trước nó cộng thêm 1.

1 2 4 5

```
3 enum Direction
4 {
5     UP = 1,    //assigned 1 by programmer
6     DOWN,     //assigned 2 by compiler
7     LEFT = 4, //assigned 4 by programmer
8     RIGHT      //assigned 5 by compiler
9 };
10 void main(){
11     printf("%d %d %d %d ",UP,DOWN,LEFT,RIGHT);
12 }
```



2. ENUM: Scope

Một kiểu dữ liệu không thể định nghĩa ở hai enum cùng phạm vi biên dịch

```
1 enum state {working, failed};
2 enum result {failed, passed};
3
4 int main() {
5     //enum result {failed, passed};
6     return 0; }
```



ERROR!

```
/tmp/UrGiFm9g2N.c:2:14: error: redeclaration of enumerator 'failed'
```

```
2 | enum result {failed, passed};
```

```
    |                ^~~~~~
```

```
/tmp/UrGiFm9g2N.c:1:22: note: previous definition of 'failed' with type 'enum state'
```

```
1 | enum state {working, failed};
```


```
    |                ^~~~~~
```

=== Code Exited With Errors ===

2. ENUM: Scope

Một kiểu dữ liệu không thể định nghĩa ở hai enum cùng phạm vi biên dịch

```
1  #include<stdio.h>
2  enum state {working, failed};
3  //enum result {failed, passed};
4
5  int main() {
6      enum result {failed, passed};
7      printf("failed: %d\npassed: %d",failed, passed);
8      return 0; }
```

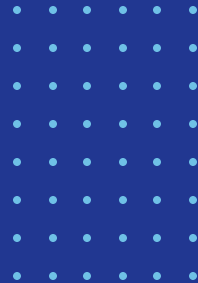
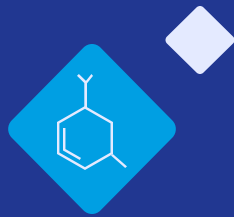


```
failed: 0
passed: 1
```

```
=== Code Execution Successful ===
```

Lợi ích so với sử dụng marco

- Tuân theo các quy tắc phạm vi biên dịch
- Các biến enum được tự động gán giá trị, dễ dàng theo dõi và sử dụng

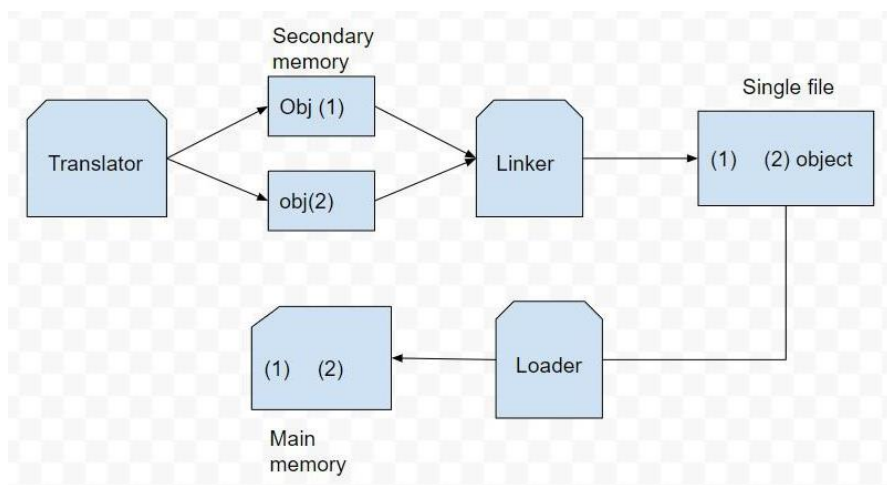


02

LINKER

Static library & Dynamic library

2. LINKER

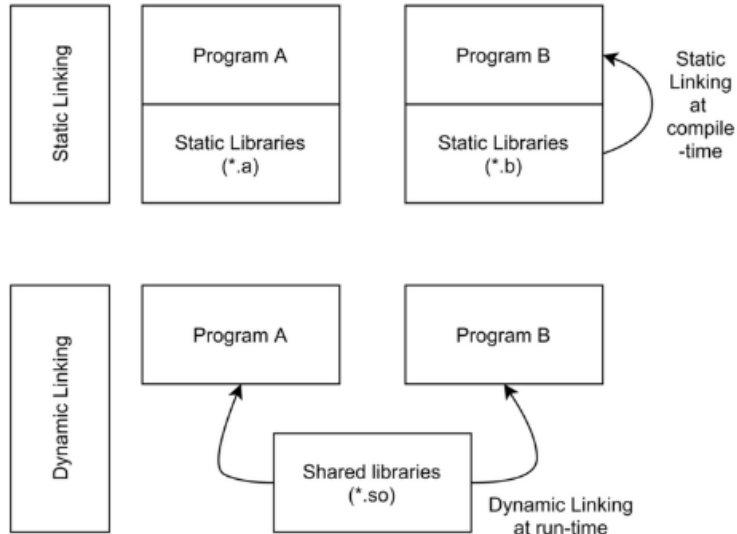


Linking là một quá trình trong hệ thống giúp liên kết các object module của chương trình thành một tệp object duy nhất để có thể load xuống bộ nhớ và thực thi.

Linking được thực hiện tự động bởi các chương trình gọi là linker.

Với một chương trình lớn, chúng ta có thể phân tách chúng ra thành các module nhỏ hơn để dễ dàng quản lý và sửa đổi, biên dịch riêng lẻ. Khi thay đổi một trong những module này ta chỉ cần biên dịch lại những thành phần khác

2. LINKER



Static linking được thực hiện trong quá trình biên dịch của chương trình. Nó tổng hợp các file object và các command-line arguments tạo ra một file object được liên kết đầy đủ, file này sẽ được tải xuống và thực thi.

Dynamic linking được thực hiện trong quá trình run-time của chương trình. Linker chỉ lưu thông tin tham chiếu đến các hàm trong thư viện liên kết động.

So sánh ưu và nhược điểm

Stactic library	Dynamic library
Kích thước chương trình tăng lên do trình biên dịch đưa dữ liệu từ thư viện vào bên trong mã nguồn	Kích thước chương trình không bị tăng lên do số lượng code bên trong thư viện => gọn nhẹ
Khi nâng cấp hay sửa đổi, muốn tận dụng những chức năng mới của thư viện ta cần biên dịch lại chương trình	Cho phép nhiều chương trình sử dụng một cách trực tiếp mà không cần biên dịch lại
Tốc độ chạy của chương trình nhanh hơn nhiều do không tốn thời gian mở các file thư viện động để đọc	Thời gian chạy lâu hơn do quá trình truy cập file, đọc file thư viện từ ổ cứng
Khi các file thư viện bị lỗi, chương trình không thể biên dịch được	Khi các thư viện bị lỗi, chương trình có thể chỉ báo lỗi trong lúc chạy

2. LINKER: Static library

Khởi tạo file thư viện với 2 hàm cộng trừ cơ bản, file main.c để sử dụng thư viện

```
$ cat math_functions.c
#include<stdio.h>

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

int subtract(int a, int b){
    return a - b;
}
```

```
cat main.c
#include<stdio.h>
#include"math_functions.h"

int main(){
    int x = 10, y = 5;
    printf("Sum: %d\n", add(x, y));
    printf("Subtract: %d\n",subtract(x, y));
    return 0;
}
```

Tạo file object của thư viện

```
gcc -c math_functions.c -o math_functions.o
```

Tạo file .a của thư viện tĩnh

```
$ ar rcs libmath.a math_functions.o
```

Link với tệp thư viện .a tạo file thực thi

```
$ gcc main.c -L. -lmath -o main
```

Chạy file thực thi chương trình

```
intern@cntd-sv101:~/thiham/testcode/staticLib$ ./main
Sum: 15
Subtract: 5
```

2. LINKER: Dynamic library

Khởi tạo file thư viện với 2 hàm cộng trừ cơ bản, file main.c để sử dụng thư viện

```
$ cat math_func.c
#include<stdio.h>

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

int subtract(int a, int b){
    return a - b;
}
```

1. Cờ -fpic cho phép tạo ra file thư viện có thể được tải và chạy ở bất kỳ vị trí nào trong bộ nhớ

```
$ gcc -c math_func.c -fpic
```

2. Tạo file .so của thư viện động

```
$ gcc -shared math_func.o -o libmath.so
```

```
cat main.c
#include<stdio.h>
#include"math_functions.h"

int main(){
    int x = 10, y = 5;
    printf("Sum: %d\n", add(x, y));
    printf("Subtract: %d\n", subtract(x, y));
    return 0;
}
```

3. Link với tệp thư viện .so tạo file thực thi

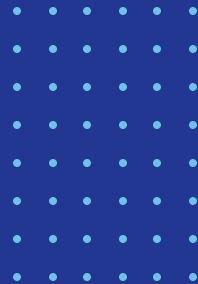
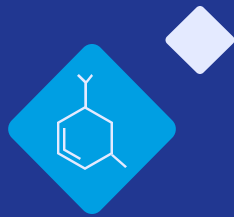
```
$ gcc main.c -o main -L. -lmath
```

***Chú ý:** Cần thay đổi biến môi trường là đường dẫn thư mục hiện tại "LD_LIBRARY_PATH."

```
export LD_LIBRARY_PATH=$PWD:$LD_LIBRARY_PATH
```

Chạy file thực thi chương trình

```
intern@cintd-sv101:~/thipham/testcode/staticLib$ ./main
Sum: 15
Subtract: 5
```



03

MAKE FILE

3. MAKE FILE

Đặt vấn đề :

- Một project được chia làm nhiều file, nhiều module và nhiều người tham gia viết nhằm đảm bảo việc bảo trì cấu trúc trở nên dễ dàng.
- Sau khi được build lần đầu, mỗi khi có thay đổi sẽ mất thời gian để build lại toàn bộ chương trình.

=> makefile có khả năng phát hiện file bị thay đổi và chỉ biên dịch lại file đó giúp tiết kiệm thời gian, dễ dàng quản lý.

3. MAKE FILE

– Makefile là một tập script chứa các thông tin : Cấu trúc project (file, dependence), và các lệnh để tạo ra file.



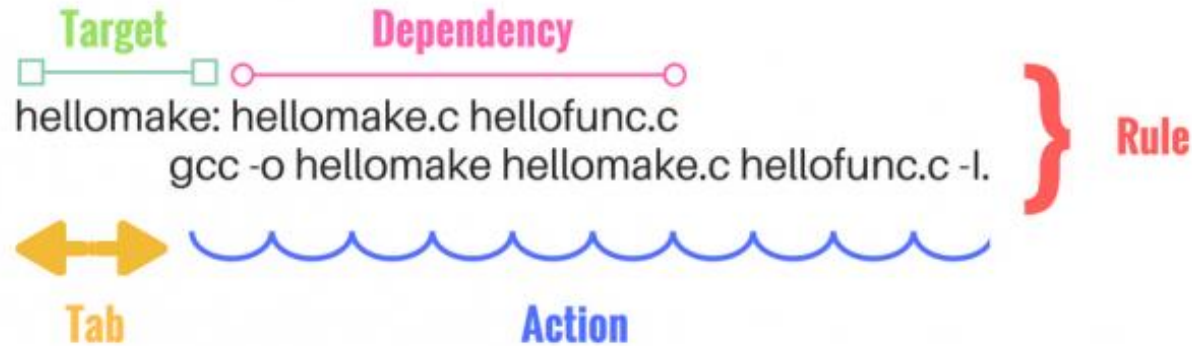
Rule: các rule cần thực hiện khi compile

Dependency: các file cần thiết để tạo ra target

Action: câu lệnh compile tạo ra **Target** từ **Dependency**, phải được thụt vào 1Tab so với **Target**

Target: là file đích được hình thành sau khi quá trình make được thực hiện

3. MAKE FILE



=> Tạo ra file thực thi hellomake từ hai file hellomake.c và hellofunc.c

- Tên của makefile nên đặt là **Makefile/makefile** để chỉ cần lệnh **make**, trình biên dịch sẽ tự động tìm đến **Makefile/makefile** để thực hiện.
- Đặt tên bất kỳ thì dùng lệnh **"make -f name"**.
- Trong cùng một thư mục trình biên dịch ưu tiên tìm kiếm makefile trước.

3. MAKE FILE: Example

```
$ cat abc.mk
hellomake:
    @echo "hello, im in abc.mk\n"
```

```
$ cat makefile
TARGET = test
include abc.mk

target1: test.c
    gcc test.c -o test

target2:
    gcc -c test.c -o test.o

clean:
    rm -f $(TARGET)
```

Nếu không lựa chọn target cụ thể, makefile được chạy mặc định tại rule đầu tiên

```
intern@cntd-sv101:~/thipham/testcode/test$ make
hello, im in abc.mk
```

Muốn chạy rule mong muốn chỉ cần thêm trực tiếp rule đó vào đằng sau make

```
intern@cntd-sv101:~/thipham/testcode/test$ make target2
gcc -c test.c -o test.o
```

3. MAKE FILE: Example

Một vài phép gán cơ bản

```
$ cat Makefile
var3 = "aaa"

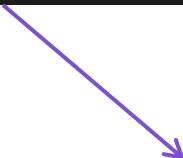
var = "bbb"
var1 = $(var)
var2 := $(var)
var3 ?= $(var)

var = "ccc"
rule3:
    @echo "$($(var1)) --- $($(var2)) --- $($(var3))"
rule1:
    @echo "hi im rule1"
rule2:
    @echo "hi im rule2"
```

“=” Phép gán đệ quy: mỗi khi biến var thay đổi, var1 thay đổi theo

“:=” Phép gán trực tiếp: var2 chỉ nhận giá trị một lần duy nhất ngay sau thời điểm lệnh gán được thực thi

“?=” Phép gán hoặc: var3 được kiểm tra xem có giá trị hay chưa, nếu rỗng sẽ nhận giá trị của var



```
intern@cntd-sv101:~/thiham/testcode/testMakefile2$ make
ccc --- bbb --- aaa
```

3. MAKE FILE: Example

```
$ cat Makefile
CC = gcc #cross compiler
CFLAGS = -I. #flag -I. to include all file *.h
INC_FILES = hello.h
OBJ = main.o hello.o
.PHONY: hellomake rule clean

%.o: %.c $(INC_FILES)
    gcc -c -o $@ $< $(CFLAGS)

hellomake: $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS)

rule:
    @echo "hello ae"

clean:
    rm -f *.o hellomake
```

```
hello.c hello.h main.c Makefile
intern@cntd-sv101:~/thipham/testcode/testMakefile2/cc$ make
gcc -I. -c -o main.o main.c
gcc -I. -c -o hello.o hello.c
gcc -o hellomake main.o hello.o -I.
intern@cntd-sv101:~/thipham/testcode/testMakefile2/cc$ ./hellomake
xin chào:
```

```
intern@cntd-sv101:~/thipham/testcode/testMakefile2/cc$ make
make: 'hellomake' is up to date.
intern@cntd-sv101:~/thipham/testcode/testMakefile2/cc$
```

Khi không sử dụng .PHONY

Trong folder có file trùng tên với target -> không chạy được rule đó, chương trình trả về như hình dưới => cần dùng biến .PHONY để đảm bảo thực thi action khi tên rule bị trùng

=> Kết quả được target mới ghi đè lên target đã có

3. MAKE FILE: Example

```
$ cat Makefile
CC = gcc #cross compiler
CFLAGS = -I. #flag -I. to include all file *.h
INC_FILES = hello.h
OBJ = main.o hello.o
.PHONY: hellomake rule clean

%.o: %.c $(INC_FILES)
    gcc -c -o $@ $< $(CFLAGS)

hellomake: $(OBJ)
    $(CC) -o $@ $^ $(CFLAGS)

rule:
    @echo "hello ae"

clean:
    rm -f *.o hellomake
```

```
hello.c hello.h main.c Makefile
intern@cmt-d-sv101:~/thipham/testcode/testMakefile2/cc$ make
gcc -I. -c -o main.o main.c
gcc -I. -c -o hello.o hello.c
gcc -o hellomake main.o hello.o -I.
intern@cmt-d-sv101:~/thipham/testcode/testMakefile2/cc$ ./hellomake
xin chao:
```

%.o: %.c \$(INC_FILES) mô tả các file .o phụ thuộc vào file .c và các tệp tiêu đề
\$@ biến đại diện cho target trong rule hiện tại
\$< biến đại diện cho phần tử đầu trong list dependency
\$^ biến đại diện cho toàn bộ phần tử trong list dependency

3. MAKE FILE: Example

Build thư viện tĩnh / động

```
intern@cntd-sv101:~/thipham/testcode/testMakefile1$ make static_build
gcc -c -o main.o main.c -I. -g -Wall
gcc -c hello.c -o hello.o -fPIC
ar rcs lib_static.a hello.o
gcc -I. -g -Wall -o static_build main.o lib_static.a -L. lib_static.a
intern@cntd-sv101:~/thipham/testcode/testMakefile1$ ls
hello.c hello.h hello.o lib_static.a main.c main.o makefile static_build
intern@cntd-sv101:~/thipham/testcode/testMakefile1$ ./static_build
Xin chào
```

Build thư viện tĩnh sử dụng make static_build

```
intern@cntd-sv101:~/thipham/testcode/testMakefile1$ make dynamic_build
gcc -c -o main.o main.c -I. -g -Wall
gcc -c hello.c -o hello.o -fPIC
gcc -shared -o lib_dynamic.so hello.o
gcc -I. -g -Wall -o dynamic_build main.o lib_dynamic.so -L. lib_dynamic.so
```

Build thư viện động sử dụng make dynamic_build

```
cat makefile
CC = gcc #cross compiler

LFLAGS = -L.
CFLAGS = -I. -g -Wall #flag -I. to include all file .h
INC_FILES = hello.h
#OBJ = main.o hello.o

.PHONY: hellomake rule clean

%.o: %.c $(INC_FILES)
    $(CC) -c -o $@ $< $(CFLAGS)

#static lib
static_build: main.o lib_static.a
    $(CC) $(CFLAGS) -o $@ $^ $(LFLAGS) lib_static.a
lib_static.a: hello.o
    ar rcs lib_static.a hello.o
#hello.o: hello.c
#    $(CC) -c hello.c -o hello.o

#dynamic lib
dynamic_build: main.o lib_dynamic.so
    $(CC) $(CFLAGS) -o $@ $^ $(LFLAGS) lib_dynamic.so
lib_dynamic.so: hello.o
    $(CC) -shared -o lib_dynamic.so hello.o
hello.o:
    $(CC) -c hello.c -o hello.o -fPIC
clean:
    rm -f *.o *.a *.so static_build dynamic_build
```

3. MAKE FILE

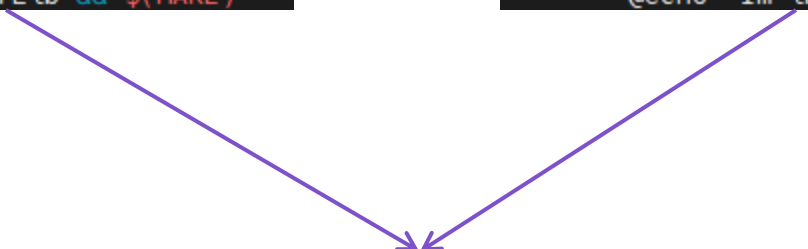
Gọi make file ở tệp con: giả sử một tệp con tên **anotherLib** có makefile muốn gọi trong makefile hiện tại

Thêm rule mới với action:

```
callmake:  
  cd anotherLib && $(MAKE)
```

Makefile trong tệp con:

```
rule1:  
  @echo "Im in anotherLib\n"
```



```
intern@cntd-sv101:~/thipham/testcode/testMakefile1$ make callmake  
cd anotherLib && make  
make[1]: Entering directory '/home/intern/thipham/testcode/testMakefile1/anotherLib'  
Im in anotherLib  
make[1]: Leaving directory '/home/intern/thipham/testcode/testMakefile1/anotherLib'
```

THANK FOR LISTENING