Header Approach

Mudream

以下即使不用編譯指令，用IDE的快捷編譯即可

重複使用

假如在同一份文件裡面有重複或類似的code，我們會包成一個函數。

那多份文件呢？

例如？

a.cpp

```
int foo(int a, int b){
    return a + a/b;
}
int main(){
    //...
}
```

b.cpp

```
int foo(int a, int b){
    return a + a/b;
}
int main(){
    //...
    //...
}
```

我們會發現都有相同的函數

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

假如是短code，那還好，可假如是...

- 很長的code呢？
- 常常需要維護更新的code？

感覺上面兩個都讓人很崩潰，能不能只需要寫或修改一次啊？

Header檔

還記得在寫Bmp時，有**引入**一個檔案嗎？

```
#include "xxx_header.h"
```

- 我們也可以寫個 Header檔 喔

```
*-exp1
|--a.cpp
|--b.cpp
|--foo_header.h
```

- `foo_header.h`

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

- `a.cpp`

```
#include "foo_header.h"
//...
```

語法

```
#include "[header_name]"
```

演示

Include Guard

情境

```
*-exp2
|--point.h
|--tri.h
|--rect.h
|--main.cpp
```

- tri定義三角形，引入`point.h`
- rect定義長方形，引入`point.h`
- main要用三角形和長方形，引入 `rect.h`, `tri.h`

point.h

```
struct point{ int x, y; };
```

tri.h

```
struct tri{ point pts[3];};
```

rect.h

```
struct rect{point pts[4];};
```

main.cpp

```
#include "tri.h"
#include "rect.h"
//...
```

編譯不過O.O

```
In file included from main.cpp:2:
In file included from ./tri.h:1:
./point.h:1:8: error: redefinition of 'point'
struct point{int x, y;};
       ^
./point.h:1:8: note: previous definition is here
struct point{int x, y;};
       ^
1 error generated.
```

原因是出在我們重複定義`struct point`兩次！！

怎麼辦QQ

Include Guard

```
#ifndef POINT_DEF_123
// if not define POINT_DEF_123
// POINT_DEF_123 就像是變數名稱
// 只是這裡的功能是判斷有沒有被定義過

#define POINT_DEF_123

struct point{int x, y;};

#endif
// 熟悉的if --- endif敘述
```

重複使用

以下只需一行編譯

複習：指令編譯

還記得怎麼用指令編譯一個檔案嗎？

```
$ g++ filename.cpp
```

- 下完這行指令就會出現一個`a.out`或`a.exe`
- 點開他即可執行

分離實做

```
*-split
|--foo.h
|--foo.cpp
|--main.cpp
```

`foo.h`

```
int foo1(int a, int b);
int foo2(int a, int b, int c);
```

`foo.cpp`

```
int foo1(int a, int b){
    return a-a/b;
}
int foo2(int a, int b){
    return a+a*a/b;
}
```

編譯？

```
$ g++ foo.cpp main.cpp
```

要把`foo.cpp`和`main.cpp`攪在一起編譯，要不然會找不到函數的實做

假如把`foo.cpp`拿掉會發生什麼事？

```
Undefined symbols for architecture x86_64:
...
```