

程式設計

Ch12. Preprocessor

Chuan-Chi Lai 賴傳淇

Department of Communications Engineering
National Chung Cheng University

Spring Semester, 2024

Outline

- 1 回顧 – C 語言編譯流程
- 2 Include 前置處理器命令 (Include Preprocessor Directive)
- 3 Define 前置處理器命令 (Define Preprocessor Directive)
- 4 條件式編譯 (Conditional Compilation)
- 5 # 和 ## 運算子 (# and ## Operators)
- 6 事先定義的符號常數 (Predefined Symbolic Constants)
- 7 斷言 (Assertions)
- 8 C 語言專案

回顧 – C 語言編譯流程

- 前置處理器主要用於以下幾種情況：
 - 巨集定義：給定一個名稱，讓它在編譯時替換成某個字串。
 - 條件編譯：讓程式碼在特定情況下才能被編譯。
 - 包含文件：可以包含其他檔案的程式碼，以便在多個檔案之間共享程式碼。

Include 前置處理器命令 Include Preprocessor Directive

Include 前置處理器命令 (Include Preprocessor Directive)

- 當撰寫程式時，若將所有程式碼寫在同一份文件，那麼程式碼會變得攏長難以維護，所以可以將程式碼分類並寫在不同文件，透過 `include` 前置處理器命令將特定檔案複製到命令所在的位置使用。
- `include` 命令具有兩種格式:

```
1  #include <FileName>
2  #include "FileName"
```

Include 前置處理器命令 (Include Preprocessor Directive)

- 差異在於前置處理器開始搜尋位置不同，這兩種格式含入檔案的位置不同。
- 如果檔案名稱用角括號括起，則通常以實作環境相依的方式執行搜尋，通常在事先指定的編譯器和系統資料夾中進行。
- 如果檔案名稱用雙引號括起，前置處理器將從編譯檔案所在目錄開始搜尋指定的含入檔案。

Define 前置處理器命令 Define Preprocessor Directive

Define 前置處理器命令 (Define Preprocessor Directive)

- Define 命令可以建立符號常數 (以符號來表示常數) 以及巨集 (定義成符號的運算)。
- 當這一行出現在某個檔案時，之後所有該識別字 (identifier) 出現 (不在字串常數中) 的位置，都會在編譯之前自動取代成代換文字 (replacement-text)。
- define 命令的格式如下：

```
1 // Format
2 #define identifier replacement-text
3
4 // Example
5 #define MAX 100
6 #define swap(a, b) { \
7     int temp = a;    \
8     a = b;           \
9     b = temp;        \
10 }
```

Define 前置處理器命令 (Define Preprocessor Directive)

- 而 undef 則可以把已經定義的東西消除掉：

```
1  #include <stdio.h>
2  #define MAX 100
3  int main() {
4      #undef MAX
5      #if !defined(MAX)
6          printf("MAX is not defined\n");
7      #endif
8  }
```

C:\Projects\ch12_code\undef_ x + v

MAX is not defined

條件式編譯 Conditional Compilation

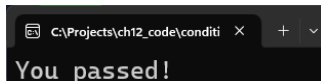
條件式編譯 (Conditional Compilation)

- 透過條件編譯，開發者可以根據特定條件來決定是否編譯特定段落程式碼。這可以用於刪除程式碼中的特定部分 (例如：調試程式碼)，或在不同平台或環境中訂製程式碼。
- 例如：假如你正在開發一套軟體，可以透過條件編譯在調試過程中啟用額外的日誌紀錄，並且在發布版本中禁用它。
- 因此條件編譯是一種靈活的工具，可以幫助開發者更好地控制程式碼的行為，並提高程式碼的可讀性以及可維護性。

條件式編譯 (Conditional Compilation)

- 基本上語法和 if-else 一樣，只是前面須加上井字號，並且 else if 需寫成 elif。
- 右圖中，由於 score 被定義為 65，符合第二個 if 的條件，因此輸出 "You passed!"。

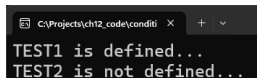
```
1 #include <stdio.h>
2 #define SCORE 65
3
4 int main() {
5     #if (SCORE == 100)
6         printf("Perfect score!\n");
7     #elif (100 > SCORE && SCORE >= 60)
8         printf("You passed!\n");
9     #elif (SCORE < 60 && SCORE >= 0)
10        printf("You failed!\n");
11    #else
12        printf("Invalid score!\n");
13    #endif
14 }
```



條件式編譯 (Conditional Compilation)

- `#ifdef` 等同於 `#if defined`，可以用來判斷變數是否“有”被定義 (`#define`)。
- `#ifndef` 等同於 `#if !defined`，跟 `#ifdef` 剛好跟相反，用來判斷變數是否“沒有”被定義。

```
1  #include <stdio.h>
2  #define TEST1 1
3
4  int main() {
5      #ifdef TEST1 // If TEST1 is defined, print a message
6          printf("TEST1 is defined...\n");
7      #else
8          printf("TEST1 is not defined...\n");
9      #endif
10
11     #ifndef TEST2 // If TEST2 is not defined, print a message
12         printf("TEST2 is not defined...\n");
13     #else
14         printf("TEST2 is defined...\n");
15     #endif
16 }
```



```
C:\Projects\ch12_code\conditi x + v
TEST1 is defined...
TEST2 is not defined...
```


和 ## 運算子 (# and ## Operators)

和 ## 運算子
and ## Operators

和 ## 運算子 (# and ## Operators)

- # 為 Stringizing Operator，目的是把一個表示式 (statement) 變成字串：

```
1  #include <stdio.h>
2  #define a(s) printf(#s)
3
4  int main() {
5      a(abc\n);
6      a("abc");
7  }
```



C:\Projects\ch12_code\stringi: X +

abc
"abc"

和 ## 運算子 (# and ## Operators)

- ## 為 Token Pasting Operator，可以連接兩個 token，我們可以拿來作具有命名規則的資料結構：

```
1  #include <stdio.h>
2  #define make_struct(name) \
3      typedef struct { \
4          int name##_price; \
5      }name
6
7  int main() {
8      make_struct(suchi);
9      suchi Kura;
10     suchi Suchi;
11     suchi Suchiro;
12     Kura.suchi_price = 40;
13     Suchi.suchi_price = 30;
14     Suchiro.suchi_price = 35;
15 }
```

事先定義的符號常數 Predefined Symbolic Constants

事先定義的符號常數 (Predefined Symbolic Constants)

- ANSI C 提供事先定義的符號常數，而這些符號常數都是分別以兩個底線字元開始和結束：

符號常數	說明
__LINE__	目前原始碼的行數 (整數常數)
__FILE__	檔案的假設名稱 (字串)
__DATE__	原始程式檔編譯時的日期 (字串)
__TIME__	原始程式檔編譯時的時間 (字串)
__STDC__	指出編譯器是否支援標準 C

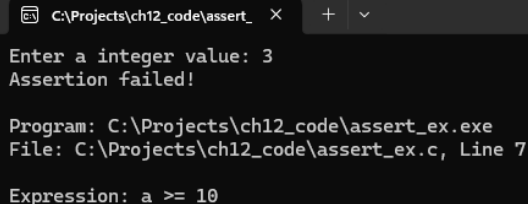
斷言

Assertions

斷言 (Assertions)

- assert() 會測試括號裡面的真偽，當括號裡面為 false 時，assert 將印出一段錯誤訊息並且呼叫 abort 函式來終止程式的執行：

```
1 #include <assert.h>
2 #include <stdio.h>
3 int main() {
4     int a;
5     printf("Enter a integer value: ");
6     scanf("%d", &a);
7     assert(a >= 10);
8     printf("Integer entered is %d\n", a);
9 }
```



C:\Projects\ch12_code\assert_ x + v

Enter a integer value: 3
Assertion failed!

Program: C:\Projects\ch12_code\assert_ex.exe
File: C:\Projects\ch12_code\assert_ex.c, Line 7

Expression: a >= 10

C 語言專案

- 當我們想用 C 語言開發專案的時候，假如我們將所有的程式碼全部放在同一份文件 (.c 檔) 中，那麼文件會變得非常冗長，而且程式碼也會變得很難維護。
- 因此我們可以將程式碼分散在不同文件中，並且透過 include 的方式將他們包含起來。
- 例如一個專案資料夾內有以下檔案：
 - main.c \\ 主程式
 - myFile.h \\ 標頭檔
 - myFile.c

- 先看主程式的部分，我們引入了“myFile.h”這個標頭檔，因此我們可以使用 myFile 裡面的程式碼 (這邊使用了 add 以及 print_int 函式)。

```
C main.c x
ch12_code > project > C main.c > ...
1  #include "myFile.h"
2
3  int main() {
4      int sum = add(5, 10);
5      print_int(sum);
6  }
7
```

- myFile.h 的內容是對 myFile.c 做原型宣告，我們在 myFile.c 實作了 add 以及 print_int 兩個函式，因此我們也需要在 myFile.h 做這兩個函式的原型宣告。

```
C myFile.h x
ch12_code > project > C myFile.h > print_int(int)
1  int add(int, int);
2  void print_int(int);
3
```

C 語言專案

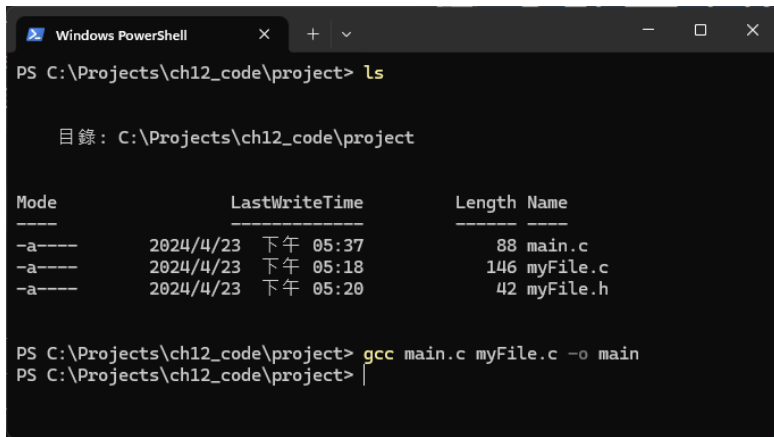
- myFile.c 的內容則是實作 add 以及 print_int 這兩個我們所需的函式。

```
C myFile.c x
ch12_code > project > C myFile.c > print_int(int)
1  #include <stdio.h>
2  #include "myFile.h"
3
4  int add(int a, int b) {
5      return a + b;
6  }
7
8  void print_int(int num) {
9      printf("%d\n", num);
10 }
```

- 開發好這些所需的程式文件後，我們需要將這些文件編譯起來，以下示範的兩行 GCC 指令意思不一樣，但結論上是一樣的，看實際情況擇一即可。

C 語言專案

- 第一種，我們直接指定所需文件做編譯。
- 但假如文件數量很多的時候，一個一個指定的話，指令會變得很冗長，因此可以使用第二種方式。



```
Windows PowerShell
PS C:\Projects\ch12_code\project> ls

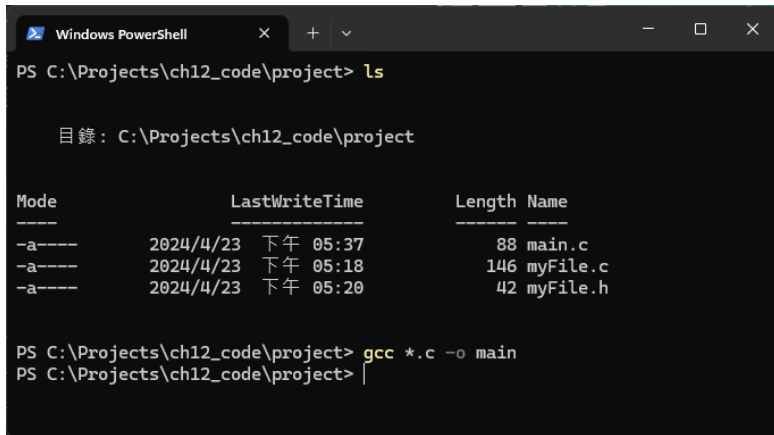
目錄: C:\Projects\ch12_code\project

Mode                LastWriteTime         Length Name
----                -
-a----            2024/4/23 下午 05:37             88 main.c
-a----            2024/4/23 下午 05:18            146 myFile.c
-a----            2024/4/23 下午 05:20             42 myFile.h

PS C:\Projects\ch12_code\project> gcc main.c myFile.c -o main
PS C:\Projects\ch12_code\project> |
```

C 語言專案

- 第二種，我們將所有當前資料夾副檔名為.c 的檔案都編譯進來，所以不需要全部檔名打出來。



```
Windows PowerShell
PS C:\Projects\ch12_code\project> ls

目錄: C:\Projects\ch12_code\project

Mode                LastWriteTime         Length Name
----                -
-a----            2024/4/23 下午 05:37             88 main.c
-a----            2024/4/23 下午 05:18            146 myFile.c
-a----            2024/4/23 下午 05:20             42 myFile.h

PS C:\Projects\ch12_code\project> gcc *.c -o main
PS C:\Projects\ch12_code\project> |
```

Q & A