國立屏東大學 資訊工程學系 程式設計(二)

20. 前置處理指令

前置處理係指在編譯之前所進行的程式碼處理動作，如figure 1所示，原始程式在編譯前可先經前置處理器(Preprocessor)進行程式碼的修改，然後才交由編譯器進行真正的編譯動作，最後才得到編譯好的目的檔(也就是可執行檔)。

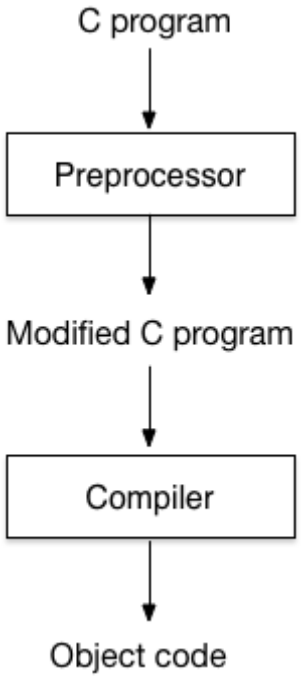

Fig. 1: 前置處理流程

同學通常不會發覺這個前置處理的動作，因為它是在我們對程式碼下達編譯命令時，和編譯的動作一併完成的。在C語言的程式碼中，存在以下三種前置處理的指令：

- 巨集定義
- 檔案引入
- 條件式編譯

在開始介紹這些前置處理指令前，先說明相關的規定：

- 所有前置處理指令(directives)永遠以「#」開頭。
- 在指令中可以「空白字元」或□tab□將出現在指令中的符號加以分隔
- 所有指令皆以「換行」結尾。若超過一行時可以「\」連接。
- 指令可以在程式任何地方出現。
- 註解可以與指令位於同一行

20.1 巨集定義(Macro Definition)

所謂的巨集(Macro)是一些程式碼的集合，由一個單一的巨集名稱所代表。一個已定義的巨集，可以在程式碼中用其名稱代表其程式碼的集合。就好比有時我們會以一些縮寫或代號，來替代較長的文字敘述，比方說我們可以用□USA□代替□The United States of American□一樣。不過在C語言中的巨集可以提供的功用遠超過縮寫的功用，我們將在以下的內容中加以介紹。

20.1.1 簡單巨集(Simple Macros)

簡單巨集顧名思義就是其定義內容較為單純的巨集(其實我們已在第6單元的常數定義中使用過)，其語法如下：

```
#define identifier replacement-list
```

其中identifier就是巨集的名稱□replacement-list則是我們要用以替換的內容。例如：

```
#define PI 3.1415
```

<note> 在定義簡單巨集時，常見的錯誤是加入了「=」與「;」。例如

```
#define PI=3.1415
或者
#define PI 3.1415;
```

</note>

這種型式的巨集，因為其用途多是將特定的數值以具有意義的名稱來代替，因此又稱為「具意義的常數(manifest constants)定義」，或簡稱為「常數定義」。在程式碼中使用常數定義巨集有以下的好處：

- 使程式更具可讀性
- 使程式更容易被修改
- 避免程式中的不一致(例如有時寫3.14有時寫3.1415)
- 可修改C語言的語法
  - 將「{」與「}」改成□Begin□與「End□□
  - 賦與型態別名，如#define BOOL int

20.1.2 參數式巨集(Parameterized Macros)

參數式巨集可以讓巨集接收參數，在替換時能依參數的內容動態地產生不同內容，其語法如下：

```
#define identifier(x1, x2, ..., xn) replacement-list
```

其中在identifier後所接的□(x1, x2, ..., xn)□即為該巨集的參數，與identifier間不可以有空白隔開。讓我們看看下面的例子：

```
#define MAX(x,y)   ((x)>(y)?(x):(y))
```

```
#define IS_EVEN(n) ((n)%2==0)
#define TOUPPER(c) ('a'<=(c)&&(c)<='z'?(c)-'a'+'A':(c))
#define newline()  printf("\n")
```

上述的例子很像是將函式定義為巨集，好處是省略了函式呼叫所須的參數傳遞與記憶體配置等成本。有以下的注意事項：

- 在replacement-list中使用參數時，最好在參數前後都加上「()」。例如：

```
#include <stdio.h>

#define MAX(x,y) x>y?x:y
int main()
{
  int i=2;
  int j=-5;

  printf("%d\n",MAX(i,MAX(3,6)));

//  2>3>6? 3: 6?2: 3>6? 3:6

  return 0;
}
```

又或者

```
#include <stdio.h>

#define TWICE(x) 2*x

int main()
{
  int i=2;
  int j=-5;

  printf("%d\n",TWICE(3+5));

  return 0;
}
```

發現了嗎？其中的TWICE(3+5)被代換為「2*3+5」，這明顯是個錯誤。

- 在巨集中參數不具備型態，換言之為泛型(generic)
  - 例如MAX(x,y)可以用於int, float, ...等不同型態
  - 但#define printINT(x) printf("%d\n",x)就可能會出問題
    - printINT(3); *正確* * *printINT(3/2);* 1.5 or 1?

- 巨集的參數若為運算式，則該運算式可能會被運算多次
  - 例如n = MAX(i++, j);會被替換為n=((i++)>(j)?(i++):(j));其中i++被執行兩次。

#運算子

巨集中的參數若冠以「#」運算子，則會將該參數轉換為字串常值(string literal)其內容為該參數的名稱。我們將「#」運算子稱為「字串化(stringization)運算子。請參考以下的範例：

```
#define PRINT_INT(n) printf(#n "=%d\n", n)
```

在這個例子中，若在程式碼中出現PRINT(i/j);則會被代換為printf("i/j" "=%d\n", i/j);其中#n所轉換成的字串常值以紅色標示。這裡要特別說明的是，編譯器會自動將連續兩個字串常值合併為一個。

##運算子

這個「##」運算子被稱為「字符拼接(token-pasting)運算子」，可用以將參數的值與其它部份連接起來，例如：

```
#define MakeVar(n) var##n
```

在程式中MakeVar(1)就會變成var1MakeVar(2)變成var2等，依此類推，將其應用在變數的宣告，則以下的程式碼：

```
int MakeVar(1), MakeVar(2);
```

會轉換產生

```
int var1, var2;
```

我們也可以利用這個運算子，來設計可產生適用於不同型態的函式樣板(template)例如：

```
#define GENERIC_MAX(type)       \
type type##_max(type x, type y) \
{                               \
   return x > y ? x : y;        \
}
```

如果我們在程式中使用該巨集GENERIC_MAX(double)則可以產生以下的程式碼：

```
double double_max(double x, double y) { return x > y ? x : y; }
```

取消巨集定義

在程式碼中，如果使用#undef identifier則是將其定義移除。

預定義巨集

C語言已預先定義了一些巨集，例如：

- __LINE__：使用__LINE__來顯示所在的行號，型態為整數。
- __FILE__：程式檔名，型態為字串。
- __DATE__：當前日期，型態為字串。
- __TIME__：當前時間，型態為字串。
- __STDC__：是否以C89或C99編譯器編譯，型態為int，1表示編譯器符合C89或C99。

我們可以在程式中使用這些預先定義好的巨集，例如：

```
printf("%d %s %s %s %d\n", __LINE__, __FILE__, __TIME__, __DATE__,
__STDC__);
```

20.2 檔案引入(File Inclusion)

檔案引入即為我們時常使用的#include用以將所需的檔案載入。

20.3 條件式編譯(Conditional Compilation)

我們可以將特別的程式碼標示在#if與「#endif這兩個前置處理指令中，依#if的條件決定在編譯時是否要涵蓋在內。通常會先定義一個非0的常數做為條件，例如：

```
#define DEBUG 1

...

#if DEBUG
printf("value of x = %d\n", x);
printf("value of y = %d\n", y);
#endif

...
```

20.3.1 defined運算子

在前置處理指令中，除了「#」與「##」外，還有一個運算子□defined□□□defined□通常與□#if□搭配使用，例如：

```
#if defined(DEBUG)

或者

#if defined DEBUG
```

再搭配□-D□的編譯器參數，就可以分別編譯除錯版與發佈版的程式，例如：

```
[04:16 user@ws home] cc -DDEBUG someprog.c
```

20.3.2 #ifdef/#ifndef

這兩個指令其實等同於□#if defined□與「#if defined !□□

20.3.3 #elif/#else

如同□if]敘述一樣□□#if□與「#endif□前置處理指令也可以搭配□#elif□與「#else□指令，進行更複雜的條件式編譯。

20.4 inline函式

當我們在函式定義的前面加以一個□inline□的修飾字時，在編譯時，其函式呼叫就會改以其函式內容代替，在執行時省去了函式呼叫的跳躍與返回的成本□inline的方式類似於巨集定義，但較具彈性。

```
#include <stdio.h>

inline int foo(int x, int y)
{
  if(x>y)
    return x*y;
  else
    return x+y;
}

int main()
{

  printf("%d\n", foo(3,5));
  return 0;
}
```