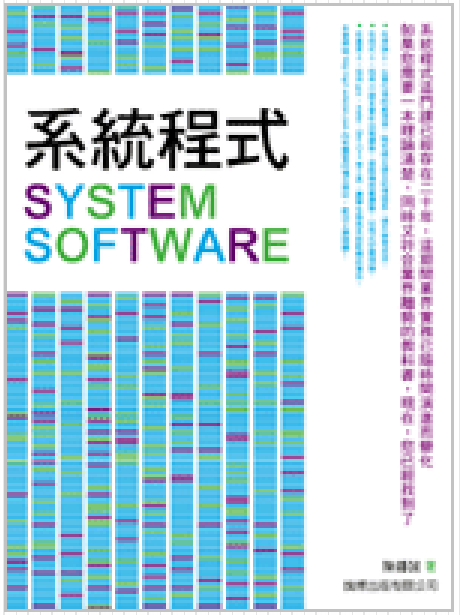
第 6 章、巨集處理器

作者：陳鍾誠

旗標出版社



第 6 章、巨集處理器

- 6.1　組合語言的巨集
- 6.2　巨集處理的演算法
- 6.3　實務案例：C 語言的巨集

簡介：巨集處理器

- 巨集處理器 (Macro Processor) 乃是一種方便程式撰寫者使用, 避免重複撰寫程式的工具

- 在程式被編譯前, 巨集處理器會先將程式當中的巨集展開, 然後再交給編譯器或組譯器進一步處理。

6.1 組合語言的巨集

▶範例 6.1 具有巨集的組合語言 - 展開前與展開後的狀況

(a). 巨集展開前

```
1   MAX   MACRO   &X, &Y, &Z
2         LD      R1, &X
3         LD      R2, &Y
4         CMP     R1, R2
5         JLE     $ELSE
6         ST      R1, &Z
7         JMP     $END
8   $ELSE:
9         ST      R2, &Z
10  $END:
11        MEND
12
13        MAX(A, B, C)
14        MAX(C, D, E)
15        RET
16
17  A:    WORD    5
18  B:    WORD    3
19  C:    RESW    1
20  D:    WORD    7
21  E:    RESW    1
22
23
24
25
26
27
```

(b) 巨集展開後

```
// MAX(A, B, C)
      LD      R1, A
      LD      R2, B
      CMP     R1, R2
      JLE     $ELSE1
      ST      R1, C
      JMP     $END2
$ELSE1:
      ST      R2, C
$END2:
// MAX(C, D, E)
      LD      R1, C
      LD      R2, D
      CMP     R1, R2
      JLE     $ELSE3
      ST      R1, E
      JMP     $END4
$ELSE3:
      ST      R2, E
$END4:
      RET

A:    WORD    5
B:    WORD    3
C:    RESW    1
D:    WORD    7
E:    RESW    1
```

巨集處理器的動作

- 巨集展開
  - 將巨集的內容嵌入到呼叫行上，並整個展開
- 參數取代
  - 將巨集參數取代為呼叫參數
- 標記編號
  - 為標記加上編號, 以避免重複的狀況。

6.2 巨集處理的演算法

- 通常分為兩輪 (2-Pass) 進行處理

- 第一輪：定義巨集
  - 將巨集內容儲存到記憶體表格中。

- 第二輪：展開巨集
  - 在呼叫時展開巨集，並進行參數取代與標記編號等動作。

單層巨集處理器的演算法

▶圖 6.1 單層巨集處理器的演算法

演算法	說明
Algorithm MacroProcessor	單層巨集處理器
Input sourceFile, expandFile	輸入原始程式、輸出展開檔
inFile = open(sourcFile)	開啟輸入檔 （原始程式）
outFile = create(expandFile)	建立輸出檔 （展開程式）
pass1()	第一輪：定義巨集
pass2()	第二輪：展開巨集
End Algorithm	
Function pass1	第一輪：定義巨集
while (not inFile.isEnd)	當輸入檔未結束時
line = getLine()	讀取一行
if isMacroDefine(line)	如果是巨集定義
macro = readMacro();	讀取整個巨集
macroTable[macro.name] =macro	記錄到巨集表當中
end while	
Function pass2	第二輪：展開巨集
inFile.goTop();	回到輸入檔開頭
while (not inFile.isEnd)	當輸入檔未結束時
line = getLine()	讀取一行
op = opCode(line)	取得指令部分
search macroTable for op	看看是否為巨集呼叫
if found	如果是巨集呼叫
macroCall = parseMacroCall(line);	剖析巨集呼叫
macro = macroTable[macroName];	取得巨集內容
body = replace macroCall.args in macro.body	取代內容中的參數
replace label with label+id in body	為標記加上編號
write body to outFile	將取代後的內容輸出
else	如果不是巨集呼叫
write source line to outFile	將該指令直接輸出
end while	
End Function	

6.3 實務案例：C 語言的巨集

- 在 C 語言的設計中, 有兩種巨集宣告方式
- 方式一：
  - 使用 #define 指令宣告巨集函數
- 方式二：
  - 利用 inline 指令, 讓一般函數改為巨集函數
- 方式一較為常用，本節將以#define為例

C 語言的巨集展開

- 使用 gcc 加上 –E 參數，可以將巨集展開

▶範例 6.2 具有巨集的 C 語言 - 展開前與展開後的狀況

指令：gcc -E macro.c -o macro_E.c

(a) 展開前：macro.c

```
#define max(a, b) (a>b?a:b)
#define min(a, b) (a<b?a:b)

int main() {
    int x = max(3, 5);
    int y = min(3, 5);
    printf("max(3, 5)=%d,
      min(3, 5)=%d\n", x, y);
}
```

(b) 展開後：macro_E.c

```
int main() {
    int x = (3>5?3:5);
    int y = (3<5?3:5);
    printf("max(3, 5)=%d,
      min(3, 5)=%d\n", x, y);
}
```

條件式展開

- 不加 DEBUG：gcc -E macroDebug.c -o MacroDebug_E.c
- 有加 DEBUG：gcc -E -D_DEBUG_ macroDebug.c -o MacroDebug_DEBUG_E.c

▶範例 6.3 具有條件式巨集的 C 語言 - 展開前與展開後的狀況

(a) 檔案：MacroDebug.c

```
#ifdef _DEBUG_
 int bugs = 0;
 #define error(msg) {printf(msg);bugs++;}
#endif

#define max(a, b) (a>b?a:b)
#define min(a, b) (a<b?a:b)

int main() {
   int x = max(3, 5);
   int y = min(3, 5);
   printf("max(3, 5)=%d\n", x);
   printf("min(3, 5)=%d\n", y);
#ifdef _DEBUG_
   if (x!=5)
      error("max(3, 5)");
   if (y!=3)
      error("min(3, 5)");
   printf("共有 %d 個錯誤", bugs);
#endif
}
```

(b) 檔案：MacroDebug_E.c

```
int main() {
   int x = (3>5?3:5);
   int y = (3<5?3:5);
   printf("max(3, 5)=%d\n", x);
   printf("min(3, 5)=%d\n", y);
}
```

(c) 檔案：MacroDebug_DEBUG_E.c

```
int bugs = 0;

int main() {
   int x = (3>5?3:5);
   int y = (3<5?3:5);
   printf("max(3, 5)=%d\n", x);
   printf("min(3, 5)=%d\n", y);

   if (x!=5)
      {printf("max(3, 5)");bugs++;};
   if (y!=3)
      {printf("min(3, 5)");bugs++;};
   printf("共有 %d 個錯誤", bugs);

}
```

習題

- 6.1 請說明巨集處理器的輸入、輸出與功能為何？
- 6.2 請說明巨集處理器會如何處理巨集參數？
- 6.3 請說明巨集處理器在展開標記時會產生甚麼問題, 應如何解決？
- 6.4 請使用 gcc 工具將範例 6.2 展開, 觀察展開後的檔案, 並說明展開前後的對應關係。
- 6.5 請使用 gcc 工具將範例 6.3 展開, 觀察展開後的檔案, 並說明展開前後的對應關係