# 第十一章 plone 建碼程式

- 上一章為了突顯建碼的一些基本技巧，特地將 Plone 語言簡化，拿掉非終端符號 <ConstDeclaration>、<ProcDeclaration> 以及 <Statement> 裡面的非終端符號 <CompoundStatement>、<CallStatement>、<ReadStatement>、<WhileStatement>，現在我們要將拿掉的非終端符號補回來，構成完整的編譯程式 Plone。

# 11.1 區塊宣告

- 因為區塊 Block 裡頭可能含有程序宣告 ProcDeclaration，程序宣告裡頭一定包含區塊 Block，構成一個遞迴（recursive）的結構。為了分別區塊的深度，宣告一個 level 整數變數用來記錄遞迴的層次（深度）。在變數宣告 VarDeclaration 之後，ProcDeclaration 程序宣告之前建立一個本區塊開始執行的標籤，其程式碼如下：
- ***sprintf(buf, "_start%d:\n", labelCount);***
- ***fprintf(outfile, buf);***
- 所建立的目的碼如下，XX 為標籤計數。
- ***_startXX:***

# 11.2 常數宣告建碼

- 常數宣告建碼如下：
- 
- 識別字名稱1　DB　'字串1' , '$'
- 識別字名稱2　DB　'字串2' , '$'
- ...
- 識別字名稱n　DB　'字串n' , '$'

# 11.3 程序宣告建碼

- 程序宣告建碼的架構如下：
-
- ***JMP _goXX***
- ***程序名稱:***
- ***<Block>***
- ***RET***
- ***_goXX:***
-

# 程序宣告建碼

- 依據 Plone 的語法，我們含有程序的原始程式架構如下：
- ***ORG 100H***
- ***JMP _start1***
- ***//略***
- ***_start1:***
- ***JMP _go1***
- ***程序名稱:***
- ***<Block>***
- ***RET***
- ***_go1:***
- ***//略***

# 11.4 CALL敘述建碼

- CALL 敘述建碼如下：
- ***CALL 程序名稱***
- 程序名稱緊緊跟隨在保留字 CALL 之後，很容易取得。程式碼如下：

```
void CallStatement()
{
  if (strcmp(token->value, "CALL")==0)
  {
    token = nextToken();
    idobj=getIdobj(token->value);
    if (idobj != NULL)
    {
      sprintf(buf, "\tCALL\t%s\n", token->value);
      fprintf(outfile, buf);
    }
    Identifier();
  }
  /*略*/
}
```

# 11.5 WHILE敘述建碼

- WHILE 敘述
- **WHILE <Condition> DO**
- **<Statement>**
- 建碼如下：
- ***_goHOME:***
- ***<Condition>***
- ***若條件成立則跳至_goHEAD***
- ***JMP _goTAIL***
- ***_goHEAD:***
- ***<Statement>***
- ***JMP _goHOME***
- ***_goTAIL:***

**_goHOME 為 WHILE 敘述重複執行時第一個指令的位置，也就是判斷條件是否成立的位置，若條件 <Condition> 成立則跳至 DO 後面敘述 _goHEAD 處（這個跳躍指令由 <Condition>提供），否則跳出 WHILE 敘述，即 _goTAIL 處。其中 HOME、HEAD、TAIL 均為標籤計數，其計數隨時不同，以防重複。**

# 11.6 READ敘述建碼

- READ 敘述
- **READ ( 識別字 )**
- 建碼如下：
- ***readstr _buf***
- ***strtoi _buf，'$'，識別字***
- ***newline***

# 11.7 完整的 plone 程式碼

- 依據上述的建碼程序補成完整的 plone 編譯器, 程式如下。
- 【plone.c】
- /************************** plone.c **********************/
- #include <stdio.h>
- #include <stdlib.h>
- /*
- ** 自訂表頭檔
- */
- #include "scanner.h"
- #include "resword.h"
- #include "err.h"
- #include "followsym.h"
- #include "idobj.h"
- #include "idobjstack.h"
- //略

# 11.8 測試程式

- 撰寫一個 plone 語言的原始程式 test1101.pl 來測試 plone 編譯器程式是否正確，程式如下。

```
PROGRAM test1101;
CONST
  msg1=" keyin a number to n please: ",
  msg2=" n=",
  msg3=" 1+2+3+...+n=";
VAR
  n, sum;
BEGIN
  WRITE(msg1);
  READ(n);
  WRITE(msg2,n);
  sum := 0;
  WHILE n>0 DO
    BEGIN
      sum := sum+n;
      n := n-1;
    END;
  WRITE(msg3, sum);
END.
```

- 在「命令提示字元」視窗，先執行下列的編譯命令：
- **C:\plone\ch11> plone test1101.pl 1 <Enter>**
- 然後在命令提示「Command Prompt」視窗的命令列輸入批次檔，此批次檔 test1101.bat 為 plone 編譯完成後沒有錯誤時自動產生的，該批次檔的內容如下。
- nasmw test1101.asm -o test1101.com
- test1101.com
- test1101.asm 為 NASM 組合語言檔名，test1101.com 為組譯後的目的程式檔名，執行批次命令如下：
- C:\PLONE\CH11> test1101.bat <Enter>
- C:\PLONE\CH11> nasmw test1101.asm -o test1101.com
- C:\PLONE\CH11> test1101.com
- keyin a number to n please: 5 <Enter>
- n= 5
- 1+2+3+...+n= 15

# 11.9 檢驗所建程式指令

```
;************** test1101.asm ****************
;
        ORG    100H
        JMP    _start1
_intstr  DB     '      ','$'
_buf            TIMES   256 DB ' '
        DB     13,10,'$'
%include        "dispstr.mac"
%include        "itostr.mac"
%include        "readstr.mac"
%include        "strtoi.mac"
%include        "newline.mac"
msg1     DB     ' keyin a number to n please: ','$'
msg2     DB     ' n=','$'
msg3     DB     ' 1+2+3+...+n=','$'
n        DW     0
sum      DW     0
```

```
_start1:
        dispstr msg1
        readstr _buf
        strtoi  _buf, '$', n
        newline
        dispstr msg2
        itostr  n, _intstr, '$'
        MOV     DX, _intstr
        MOV     AH, 09H
        INT     21H
        newline
        PUSH    0
        POP     AX
        MOV     [sum], AX
```

```
_go2:
        PUSH    WORD [n]
        PUSH    0
        POP     BX
        POP     AX
        CMP     AX, BX
        JG      _go3
        JMP     _go4
```

```
_go3:
        PUSH    WORD [sum]
        PUSH    WORD [n]
        POP     BX
        POP     AX
        ADD     AX, BX
        PUSH    AX
        POP     AX
        MOV     [sum], AX
        PUSH    WORD [n]
        PUSH    1
        POP     BX
        POP     AX
        SUB     AX, BX
        PUSH    AX
        POP     AX
        MOV     [n], AX
        JMP     _go2
```

```
_go4:
        dispstr msg3
        itostr  sum, _intstr, '$'
        MOV     DX, _intstr
        MOV     AH, 09H
        INT     21H
        newline
        MOV     AX, 4C00H
        INT     21H
```

# 11.10 plone 編譯器測試

- 我們的編譯器程式 plone.c 總算經過掃描程式測試、語彙分析測試、語法分析測試、語意分析測試、建碼測試後才完成的，各個項目若測試不完全而隱藏著錯誤，這些錯誤會延伸到整個編譯器程式的，因此必須對編譯器程式做各種可能情況的測試，測試愈完整，錯誤就愈少。

# 11.10.1 READ/WRITE 測試

- 【測試01】
- 從鍵盤輸入一個整數至變數 n，計算兩倍值後輸出。
- 【readwrite.pl】

```
PROGRAM readwrite;
CONST
  msg1=" please key in a number: ",
  msg2=" number keyed in is ",
  msg3=" value doubled is ";
VAR
  n, d;
BEGIN
  WRITE(msg1);
  READ(n);
  WRITE(msg2,n);
  d := n*2;
  WRITE(msg3,d);
END.
```

- 【測試02】
- 從鍵盤輸入兩個整數至 m、n 變數，計算其和後輸出。
- 【readwrite2.pl】

```
PROGRAM readwrite2;
CONST
  msg1=" please key in two numbers: ",
  msg2=" numbers keyed in are ",
  msg3=" sum is ";
VAR
  m, n, sum;
BEGIN
  WRITE(msg1);
  READ(m,n);
  WRITE(msg2,m,n);
  sum := m+n;
  WRITE(msg3,sum);
END.
```

# 11.10.2 IF 測試

- **【測試03】**
- 從鍵盤輸入兩個整數至 a、b 變數，將較大者輸出。
- **【iftest.pl】**

```
PROGRAM iftest;
CONST
  msg1=" please key in a number (a): ",
  msg2=" please key in a number (b): ",
  msg3=" the bigger is ";
VAR
  a, b, bigger;
BEGIN
  WRITE(msg1);
  READ(a);
  WRITE(msg2);
  READ(b);
  IF a>=b THEN bigger:=a;
  IF b>a THEN bigger:=b;
  WRITE(msg3,bigger);
END.
```

```
PROGRAM iftest2;
CONST
  msg1=" please key in a number (a): ",
  msg2=" please key in a number (b): ",
  msg3=" the bigger is ";
VAR
  a, b, bigger;
BEGIN
  WRITE(msg1);
  READ(a);
  WRITE(msg2);
  READ(b);
  IF a>=b THEN
    BEGIN
      bigger:=a;
      WRITE(msg3,bigger);
    END;
  IF b>a THEN
    BEGIN
      bigger:=b;
      WRITE(msg3,bigger);
    END;
END.
```

# 11.10.3 WHILE 測試

- 【測試04】
- 從鍵盤輸入兩個整數至 a、b 變數，求出最大公約數後輸出。
- 【whiletest.pl】

```
PROGRAM whiletest;
CONST
  msg1=" keyin a number (a): ",
  msg2=" keyin a number (b): ",
  msg3="H.C.F =";
VAR
  a,b,c,q,r,hcf;
BEGIN
  WRITE(msg1);
  READ(a);
  WRITE(msg2);
  READ(b);
```

```
IF a<b THEN
  BEGIN
    c:=a;
    a:=b;
    b:=c;
  END;
 WHILE b > 0 DO
  BEGIN
    q := a/b;
    r := a-b*q;
    a := b;
    b := r;
  END;
 hcf := a;
 WRITE(msg3,hcf);
END.
```

# 11.10.4 CALL 測試

- 【測試05】
- 從鍵盤輸入一個溫度度數 deg。
- 1) 將 deg 視為華氏溫度，轉換為攝氏溫度 c。
- 2) 將 deg 視為攝氏溫度，轉換為華氏溫度 f。
- 【calltest.pl】
-

```
PROGRAM calltest;
CONST
  msg1 = " please keyin a degree (deg) : ",
  msg2 = " deg = ",
  msg3 = " c = ",
  msg4 = " f = ";
VAR
  deg, f, c;
```

```
PROCEDURE cTOf;
  BEGIN
    f := deg*9/5+32;
  END;
PROCEDURE fTOc;
  BEGIN
    c := (deg-32)*5/9;
  END;
BEGIN
  WRITE(msg1);
  READ(deg);
  CALL cTOf;
  WRITE(msg2,deg,msg4,f);
  CALL fTOc;
  WRITE(msg2,deg,msg3,c);
END.
```

- 【測試06】
- 從鍵盤輸入兩個整數至 a、b 變數，求出最大公約數 hcf 及最小公倍數 lcm 後輸出。
- 【calltest2.pl】
- PROGRAM calltest2;
- CONST
- msg1=" keyin a number (a): ",
- msg2=" keyin a number (b): ",
- msg3="H.C.F =",
- msg4="L.C.M =";
- VAR
- a,b,c,hcf,lcm;

```
PROCEDURE hcfproc;
  VAR
    big, small, q, r;
  BEGIN
    big := a;
    small := b;
    WHILE small > 0 DO
      BEGIN
        q := big/small;
        r := big-small*q;
        big := small;
        small := r;
      END;
    hcf := big;
  END;
PROCEDURE lcmproc;
  BEGIN
    lcm := a*b/hcf;
  END;
```

```
BEGIN
  WRITE(msg1);
  READ(a);
  WRITE(msg2);
  READ(b);
  IF a<b THEN
    BEGIN
      c:=a;
      a:=b;
      b:=c;
    END;
  CALL hcfproc;
  WRITE(msg3,hcf);
  CALL lcmproc;
  WRITE(msg4,lcm);
END.
```

# 11.10.5 CONST 及 VAR 宣告測試

- 【測試07】
- 　設計一個只有 CONST 宣告的程式，測試一下常數識別字。
- 【consttest.pl】
- 　PROGRAM consttest;
- 　CONST
- 　　msg = "This is a CONST message!!";
- 　BEGIN
- 　　WRITE(msg);
- 　END.

- 【測試08】
- 設計一個含有 PROCEDURE 程序宣告的程式，從程序裡當然可存取區域變數，試試看從主程式裡能否存取區域變數。
- 【vartest.pl】

```
PROGRAM vartest;
CONST
  msg1=" b from PROCEDURE is ",
  msg2=" b from PROGRAM   is ";
VAR
  a;
PROCEDURE proc;
  VAR
    b;
  BEGIN
    b := 123;
    WRITE(msg1,b);
  END;
BEGIN
  CALL proc;
  WRITE(msg2,b);
END.
```

# 11.10.6 PROCEDURE 宣告測試

- 【測試09】
- 請設計一個程序宣告裡頭又有程序宣告的程式，並測試之。
- 【hcfmain.pl】
- PROGRAM hcfmain;
- CONST
- msg1=" keyin a number (a): ",
- msg2=" keyin a number (b): ",
- msg4="L.C.M =";
- VAR
- a,b,c,hcf,lcm;

```
PROCEDURE hcfproc;
 CONST
  msg3="H.C.F =";
 VAR
  big, small;
 PROCEDURE nextdivide;
  VAR
   q, r;
  BEGIN
   q := big/small;
   r := big-small*q;
   big := small;
   small := r;
  END;
 BEGIN
  big := a;
  small := b;
  WHILE small > 0 DO CALL nextdivide;
  WRITE(msg3, big);
  hcf := big;
 END;
```

```
PROCEDURE lcmproc;
  BEGIN
    lcm := a*b/hcf;
  END;
BEGIN
  WRITE(msg1);
  READ(a);
  WRITE(msg2);
  READ(b);
  IF a<b THEN
    BEGIN
      c:=a;
      a:=b;
      b:=c;
    END;
  CALL hcfproc;
  CALL lcmproc;
  WRITE(msg4,lcm);
END.
```

# 11.11 測試結論

- 本章這個版本的 plone 編譯器程式由前面幾個測試的結果得知它是有缺陷的，缺陷如下：
- ***1. 所有的識別字，不管是程序之****內或之外，均不能****重複，否則會****產生語法錯誤。***
- ***2. 從程序外面可以取得程序****內之區域變數。*
- 其實這兩個缺陷在我們當初訂出設計目標時就已經埋下原因，因為我們為了簡化，只對於識別字做單純的管理，並沒有分整體（global）變數或區域（local）變數的關係。