第八章 語意分析

- 上一章的 plone 程式語言語法分析從樹根的非終端符號一直剖析到樹葉的終端符號，並做語法錯誤時的復原處理，但是還不夠，因為對於所使用的識別字有沒有宣告並沒有處理。程式裡使用到沒有宣告的識別字，就犯了語意（semantic）上的錯誤。
- 本章要探討語意上的錯誤要如何處理。

8.1 識別字結構

- 我們的目標是針對識別字的使用，因此必須對識別字做有效的管理，包括插入及搜尋的動作，選用 C 語言所提供的 struct 結構，這種結構的宣告格式如下：
- ***struct idobjTag { ... };***
- idobj 表示識別字結構（物件）的意思，Tag 表示標籤，整個表示識別字結構，「struct idobjTag」就是它的資料型態，屬於 C 語言的結構型態。在這個結構裡您可定義各種資料型態的欄位（field），或稱為成員（member），其架構如下：

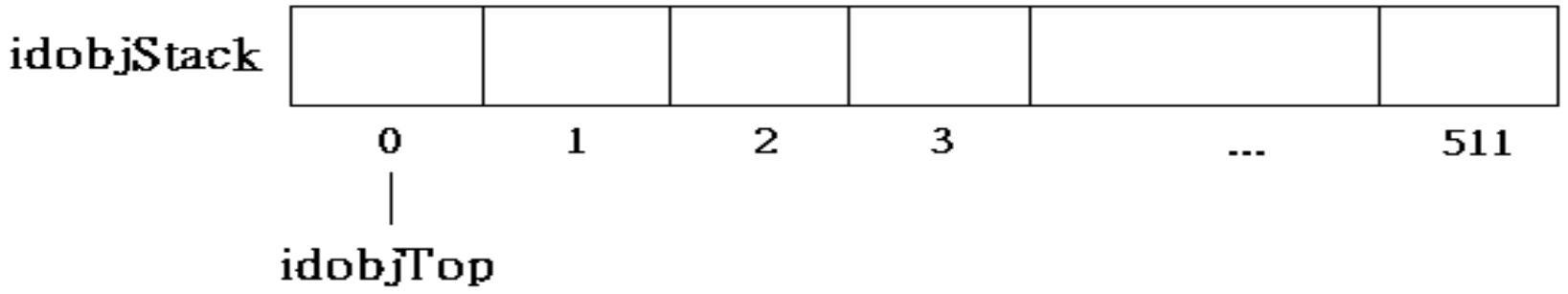
```
struct idobjTag
{
  char name[36];
  int sym;
  int attr;
  int level;
  //略
}
```

struct idobjTag 結構變數成員

- 包含四個結構變數成員（member）。
- ***第一個成員*** name 為識別字名稱，屬於字串（字元陣列）。***第二個成員*** sym 為該識別字在 sym.h 裡所定義的整數編號。***第三個成員*** attr 說明為該識別字宣告在 CONST 區或 VAR 區。宣告在 CONST 區其值為 symCONST，宣告在 VAR 區其值為 symVAR。***第四個成員*** level 為該識別字宣告在 PROCEDURE 區的層次。

8.2 識別字結構堆疊

- 一個程式裡頭通常會使用到許多的識別字，有必要將這些識別字結構儲存起來，並作適當的管理，我們可以選用堆疊（stack）、佇列（queue）、串列（list）、樹（tree）等結構，因為堆疊只在頂端出入，只需一個頂端的指標，較為簡單，選用陣列當堆疊使用，一切都朝最簡單的設計方式，簡單容易了解，也較不會犯錯。宣告如下：
- ***struct idobjTag *idobjStack[512];***
- ***int idobjTop = 0;***
- 識別字結構堆疊 idobjStack，堆疊頂端指標 idobjTop，所儲存的元素為識別字結構指標，空白的識別字結構堆疊如下圖所示。



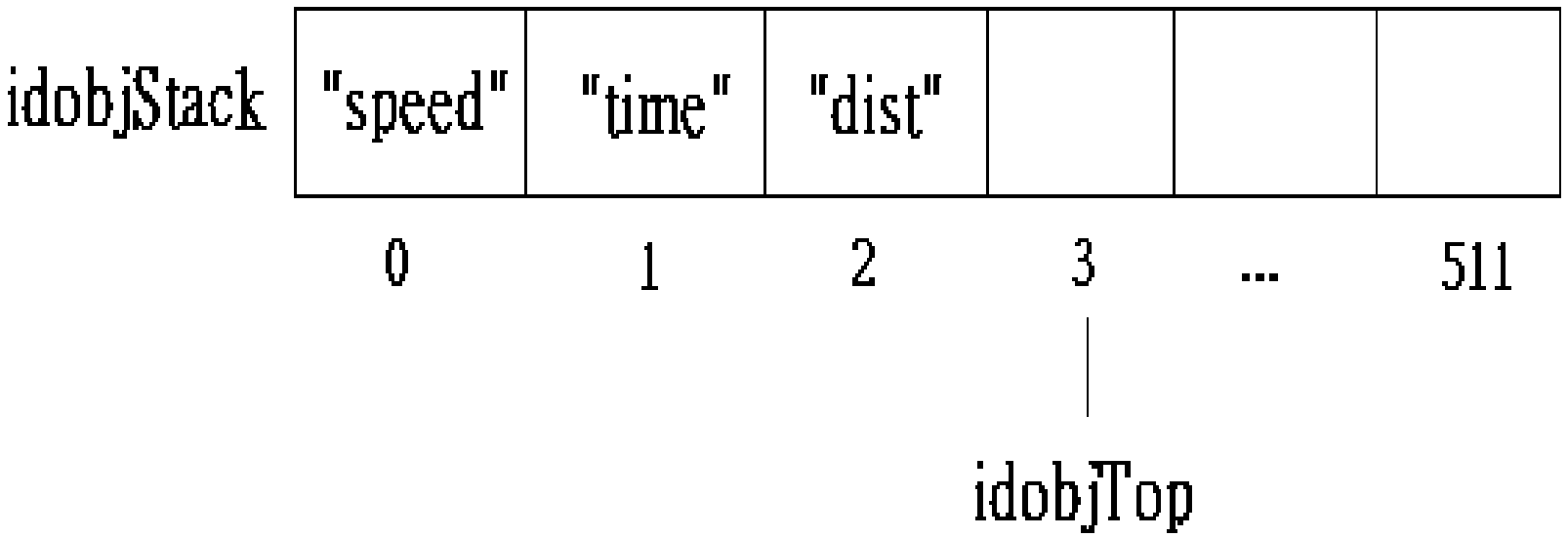
- 要將識別字結構指標 p 疊入堆疊頂端，其程式碼如下：
- ```
  void idobjpush(struct idobjTag *p)
  {
    idobjStack[idobjTop++]=p;
  }
  ```
- 要從堆疊頂端疊出頂端元素至識別字結構指標 p，其程式碼如下：
- ```
  struct idobjTag * idobjpop()
  {
    struct idobjTag *p=idobjStack[--idobjTop];
    return p;
  }
  ```

- 下列的程式碼透過函式的參數建立一個識別字結構，並傳回其指標。
-
- ```
  struct idobjTag *newIdobj(char name[], int sym,
                            int attr, int level)
  {
    struct idobjTag *p =
       malloc(sizeof(struct idobjTag));
    strcpy(p->name, name);
    p->sym = sym;
    p->attr = attr;
    p->level = level;
    return p;
  }
  ```
  ```

- 下列的 getIdobj() 函式透過已知的識別字字串 id 找到相對應的識別字結構，並傳回其指標，找不到時傳回 NULL 值。
-
-
```
struct idobjTag *getIdobj(char *id)
{
  struct idobjTag *idobj=NULL;
  int i;
  for (i=0; i<idobjTop; i++)
  {
    if (strcmp(idobjStack[i]->name,id)==0)
      return idobjStack[i];
  }
  return idobj;
}
```

- 下列的 idobjstackToString() 函式將整個識別字結構堆疊的內容以文字方式呈現，並傳回。
-
-
```
char * idobjstackToString()
{
  static char str[1024];
  int i;
  strcpy(str, "");
  for (i=0; i<idobjTop; i++)
  {
    strcat(str, idobjToString(idobjStack[i]));
    strcat(str, "\n");
  }
  return str;
}
```

- 程式 idobjstacktest.c 所建立的識別字結構堆疊如下圖所示，請注意圖中所示 "speed"、"time"、"dist" 均代表該識別字名稱所屬的識別字結構指標。



8.2 CONST 宣告識別字

- 在 Plone 程式語言語法裡第四條規則 <ConstDeclaration> 常數非終端符號規定如下：
-
- 4. <ConstDeclaration> ::=
- CONST <Identifier>=<String>
- {,<Identifier>=<String>};
-
- 宣告識別字後面跟著等號，其後跟著一個字串，後面可跟著以逗號隔開的常數識別字宣告，最後以分號結束。

- 當辨認出第一個識別字時，執行如下的查詢該識別字是否存在識別字結構堆疊 idobjStack 裡頭，若是則顯示錯誤編號為 24，表示該識別字重複宣告。
-
- ```
  if (checkexist()) Error(24);
  ```
-
- 檢查是否存在的 checkexist() 方法，其程式碼如下：
-
- ```
  int checkexist()
  {
    idobj=getIdobj(token->value);
    if  (idobj==NULL)
      return 0;
    else
      return 1;
  }
  ```

- 檢查是否存在的 checkexist() 方法，其程式碼如下：
-
-
```
int  checkexist()
{
  idobj=getIdobj(token->value);
  if  (idobj==NULL)
    return 0;
  else
    return 1;
}
```
-
- 透過 getIdobj() 函式傳回符記識別字 token->value 相對應的識別字結構指標 idobj，若其值為 NULL 表示該識別字不存在，傳回 0 值，否則表示在識別字結構堆疊裡頭找到該符記識別字 token->value，傳回 1 值。

- 當辨認出識別字時，建立該識別字結構指標 idobj，並將它疊入識別字結構堆疊裡頭，其程式碼如下。
-
- idobj = newIdobj(token->value, token->sym,
- symCONST, level);
- idobjpush(idobj);

8.3 VAR 宣告識別字

- 在 Plone 程式語言語法裡第五條規則 <VarDeclaration> 變數宣告非終端符號規定如下：
-
- 5. <VarDeclaration> ::= VAR <IdentifierList> ;
- 15 <IdentifierList> ::= <Identifier>{,<Identifier>}
-
- VAR 宣告一個變數識別字，後面可跟著以逗號隔開的變數識別字，最後以分號結束。

- 當辨認出第一個識別字時，執行如下的查詢該識別字是否存在識別字結構堆疊裡頭，若是則顯示錯誤編號 25 該識別字重複宣告的訊息。
-
- ```
  if ( checkexist() ) Error(25);
  ```
-
- 當辨認出識別字時，建立該識別字結構指標 idobj，並將它疊入識別字結構堆疊裡頭，其程式碼如下。
-
- ```
  idobj = newIdobj(token->value, token->sym,
                   symVAR, level);
  idobjpush(idobj);
  ```

8.4 PROCEDURE 宣告識別字

- 在 Plone 程式語言語法裡第六條規則 <ProcDeclaration> 程序非終端符號規定如下：
-
- **6. <ProcDeclaration>::={PROCEDURE <Identifier>;<Block>;}**
- **3. <Block> ::= [<ConstDeclaration>]**
- **[<VarDeclaration>]**
- **[<ProcDeclaration>]**
- **<CompoundStatement>**

- PROCEDURE 宣告一個程序識別字，後面可跟著分號，其後跟著一個區塊，最後以分號結束。區塊裡又可宣告 PROCEDURE，構成一個遞迴（recursive）的結構，也就是說一個 PROCEDURE 裡又可宣告另一個 PROCEDURE 程序。

- 程序 PROCEDURE 宣告的主體是一個 while 迴圈，表示可宣告多個程序。當辨認出第一個識別字時，執行如下的查詢該識別字是否存在識別字 編號為 28 該識別字重複宣告的訊息。
-
- if （ checkexist() ） Error(28);
-
- 一個程序 PROCEDURE 裡包含一個 <Block> 區塊，該區塊又可包含一個程序，構成遞迴。也就是說在一個程序裡區塊可以出現好多次，一層一層往下深入，因此設計一個整體變數 level，往下深入一層時 level 值就增加一，往上淺出一層時就減一，這個值就記錄在識別字結構裡，存入識別字結構堆疊中。

- 非終端符號 Block 敘述的結構如下：
-
-
```
    void Block()
    {
      ++level;
      if (strcmp(token->value, "CONST")==0)
        ConstDeclaration();
      if (strcmp(token->value, "VAR")==0)
        VarDeclaration();
      if (strcmp(token->value, "PROCEDURE")==0)
        ProcDeclaration();
      CompoundStatement();
      --level;
    }
```

8.5 敘述裡的識別字

- 敘述裡的識別字都要透過 Identifier() 方法處理。

```
void Identifier()
{
  if  ( token->sym == symIDENTIFIER )
  {
    idobj=getIdobj(token->value);
    if (idobj == NULL)
      Error(26);
    token = nextToken();
  }
  else
    Error(21);
}
```

- 每一個敘述裡的識別字都要檢查是否存在識別字結構堆疊裡，若是則取出該識別字結構 idobj 備用，否則報告第 26 號錯誤，也就是該識別字未經宣告，屬於語意上的錯誤。

8.6 剖析程式 parser.c

- 依據前面的論述，修改原來的剖析程式如下。
- 【parser.java】

```
/********************** parser.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"
/*
** 自訂常數
*/
//略
```

8.7 測試程式 test81.pl

```
C:\plone\ch08> parser test81.pl 1 <Enter>
 1 PROGRAM test81;
 2 CONST
 3   msg1=" x=",
 4   msg2=" y=";
 5 VAR
 6   x, y;
 7 BEGIN
 8   x := 3;
 9   WHILE x>0 DO
10     BEGIN
11       y := x*3+6;
12       WRITE(msg1,x);
13       WRITE(msg2,y);
14       x := x-1;
15     END;
16 END.

Plone compile completed.
Error count : 0
```

- 測試程式 test81.pl 語法完全正確，執行時第二個引數輸入 1 表示除了編譯之外，還要輸出識別字結構堆疊的內容。如執行結果所示。
- 識別字結構堆疊內容如下：
- name:"msg1" sym:2 attr:30 level:0
- name:"msg2" sym:2 attr:30 level:0
- name:"x" sym:2 attr:31 level:0
- name:"y" sym:2 attr:31 level:0

8.8 測試程式 test82.pl

```
C:\plone\ch08> parser test82.pl 1 <Enter>
  1 PROGRAM test82;
  2 CONST
  3   msg1=" x=",
  4   msg2=" y=";
  5 VAR
  6   x, y;
  7 BEGIN
  8   x := 3;
  9   WHILE x!=0 DO
****          ^20  關係運算子錯誤
****          ^23  飛越至下一個敘述
 10    BEGIN
 11      y := x^3+6;
****               ^15  WHILE敘述錯誤,遺漏DO
 12      WRITE(msg1,x);
 13      WRITE(msg2,y);
 14      x := x-1;
 15    END;
****       ^0  必須跟著句點.

  Plone compile completed.
  Error count : 4
```

- 測試程式 test82.pl 刻意撰寫些錯誤的敘述，執行時第二個引數輸入 1 表示除了編譯之外，還要輸出識別字結構堆疊的內容。如執行結果所示。
- Plone compile completed.
- Error count : 4
-
- 識別字結構堆疊內容如下：
- name:"msg1" sym:2 attr:30 level:0
- name:"msg2" sym:2 attr:30 level:0
- name:"x" sym:2 attr:31 level:0
- name:"y" sym:2 attr:31 level:0

8.9 測試程式 test83.pl

- 測試程式 test83.pl 主要測試程序，包含 gcd 程序，語法完全正確，執行時第二個引數輸入 1 表示除了編譯之外，還要輸出識別字結構堆疊的內容。如執行結果所示。
- Plone compile completed.
- Error count : 0
- 識別字結構堆疊內容如下：
- name:"msg1" sym:2 attr:30 level:0
- name:"msg2" sym:2 attr:30 level:0
- name:"msg3" sym:2 attr:30 level:0
- name:"x" sym:2 attr:31 level:0
- name:"y" sym:2 attr:31 level:0
- name:"hcf" sym:2 attr:31 level:0
- name:"gcd" sym:2 attr:32 level:0
- name:"q" sym:2 attr:31 level:1
- name:"r" sym:2 attr:31 level:1

8.10 測試程式 test84.pl

- C:\plone\ch08> parser test84.pl 1 <Enter>
- 1 PROGRAM test84;
- 2 CONST
- 3 msg1=" x=",
- 4 msg3=" y=",
- 5 msg3="hcf=";
- **** ^24 CONST宣告常數重複
- 6 VAR
- 7 x,y,hcf,x;
- **** ^25 VAR宣告變數重複
- 8 PROCEDURE gcd;
- 9 VAR
- 10 q,r,hcf;
- **** ^25 VAR宣告變數重複

```
    11   BEGIN
     12     WHILE y > 0 DO
     13       BEGIN
     14         q := x/y;
     15         r := x-y*q;
     16         x := y;
     17         y := r;
     18       END;
     19     hcf := x;
     20     WRITE(msg3,hcf);
     21   END;
     22 BEGIN
     23   x := 12;
     24   y := 9;
     25   WRITE(msg1,x);
     26   WRITE(msg2,y);
    ****           ^26  識別字沒有宣告
     27   CALL gcd;
     28 END.

     Plone compile completed.
     Error count : 4
```

- 識別字結構堆疊內容如下：
- name:"msg1" sym:2 attr:30 level:0
- name:"msg3" sym:2 attr:30 level:0
- name:"msg3" sym:2 attr:30 level:0
- name:"x" sym:2 attr:31 level:0
- name:"y" sym:2 attr:31 level:0
- name:"hcf" sym:2 attr:31 level:0
- name:"x" sym:2 attr:31 level:0
- name:"gcd" sym:2 attr:32 level:0
- name:"q" sym:2 attr:31 level:1
- name:"r" sym:2 attr:31 level:1
- name:"hcf" sym:2 attr:31 level:1