

《程式語言》

試題評析

今年程式語言的考題都算是普遍性的題目，也都不難，大多集中在比較性的問題，例如第一題考變數的 static scoping 與 dynamic scoping 之比較；第二題是 PASCAL 的 case 敘述與 C 的 switch 敘述之差異和優缺點比較；第五題則考 Java 的陣列和 C 語言陣列之差異性比較。其他第三題考 C 語言的指標特性，第四題考一個 LISP 的衍生語言 Scheme 之程式。基本上每一題都應該可以答得出來，差別在於能否寫得完整而已。因此今年的考題對一般考生而言，應可得六、七十分，程度好者拿個八、九十分並不困難。

命中事實：第一題於第二回講義第 11、12 頁及總複習第二回第 73 頁第二題；第二題於總複習第二回第 8 頁第一題；第三題於第二回講義第 110、111 頁；第四題於第五回講義第 10 頁；第五題於總複習第二回第 96 頁第三題。

一、(一)變數 (variable) 的使用可分為靜態範圍 (static scoping) 及動態範圍 (dynamic scoping)，請分別說明。(10分)

(二)下面的程式，是以像 PASCAL (PASCAL like) 的語言所寫的。如分別用靜態及動態範圍處理變數範圍，試問所得的值各為何？(10分)

```

Program main;
  Var x: real;
  Procedure sub1;
    Begin/*sub1*/
      Writeln(" x=", x)
    End;/*sub1*/
  Procedure sub2;
    Var x: real;
    Begin/*sub2*/
      X:= 3.6
    Sub1
    End;/*sub2*/
  Begin/*main*/
    x:= 6.3
  Sub2
  End/*main*/

```

答：

(一)

1. Static scoping :

對於自由變數之定義的決定方式，是從它所屬程式單元的上一層程式單元中尋找是否宣告此變數？若沒有，再到更上一層的程式單元去找，直到第一次發現即為其定義，或到了最外層都沒有找到即為未定義之變數。

2. dynamic scoping :

在尚未終結的程式單元中尋找自由變數之定義，檢查的順序是依照呼叫的相反順序進行，第一次出現的定義便繫結到此自由變數上；若找不到表示此變數未定義。

(二)

1. 採用 static scoping 輸出：

X=6.3

2. 採用 dynamic scoping 輸出：

X=3.6

二、請舉例詳細說明PASCAL的Case敘述與C語言的Switch敘述的(一)異、同，(10分)並(二)指出各有什麼優、劣點。(10分)

答：

(一)相同點：PASCAL的Case敘述C語言的switch敘述都是可從多條路徑中選擇一條出來執行的控制結構。

(二)語法格式之比較：

1.PASCAL的Case敘述格式：

```
case expression of
    constant_list_1: statement1;
    :
    :
    constant_list_n: statementn;
[else statementn+1];
end
```

2.C語言的switch的敘述格式：

```
switch(expression){
    case constant_expression_1: statement1;
    :
    :
    case constant_expression_n: statementn;
    [default ; statementn+1]
}
```

(三)兩者相異處包括

- 1.PASCAL case敘述的選擇子(即expression)可為任何有序的純量型態，如整數、字元、布林、列舉或子區間等型態；而C語言switch敘述的選擇子只能是整數型態。
2. PASCAL的case敘述中，與選擇子比對者可為多個常數組成的常數串列；而C語言的switch敘述中，與選擇子比對者只能是單一個常數運算式(即單一個常數值)。
3. PASCAL的case敘述中，所選擇的敘述執行後，會自動跳出此case敘述；而C語言的switch敘述中，所選擇的敘述執行後，不會自動跳出此switch，而是繼續往下執行，直到遇到break敘述或switch敘述結束為止。亦即前者具有隱含式跳出(implicit branch)功能，而後者沒有。

(四) PASCAL的case敘述：

- 1.優點：因具有隱含式跳出功能，故可靠性較高。
- 2.缺點：因所選擇的敘述執行後立即跳出，無法連續執行多條路徑之敘述，故彈性較低。

(五)C語言的switch敘述的優缺點恰與PASCAL的case敘述相反，即：

- 1.優點：彈性較高。
- 2.缺點：可靠性較低。

三、請舉例詳細說明C語言中，指標的特性。(20分)

答：

(一)C語言指標的宣告和使用

- 1.在變數宣告時，於變數名稱前面加上星號“*”，即可宣告指標。
- 2.在可執行敘述中，“*”代表查考運算，“&”可取得變數之位址。

```
【例】int *ptr; _____ ①
      int sum, value;
      ...
      ptr=&sum; _____ ②
      *ptr=*ptr+value; _____ ③
```

說明：1.敘述①宣告ptr為可指到整數記憶體的指標。

2.敘述②將sum的位址存入ptr中，即讓ptr指向sum。

3.敘述③將ptr所指記憶體的內容與value的值相加，並將結果存回ptr所指的記憶體中。在效果上相當於sum = sum + value。

(二)指標的算術運算：

1.指標在進行算術運算時是以指標所指個體的大小為運算的單位。

【例】若宣告

```
double *ptr;  
int index=5;
```

則執行

```
ptr=ptr+index;
```

ptr的值不是加上5，而是加上5個倍精度重數(double)的記憶體大小，亦即真正執行的動作如下：

```
ptr=ptr+index*sizeof(double)
```

2.指標的算術運算經常用來處理陣列資料。

(三)指標與陣列：

1.由於C和C++的陣列註標都從0開始，而陣列名稱代表陣列的起始位址，因此欲讓指標指向陣列的開始之處，有兩種方式：

- (1)將陣列的首元素之位址指定給指標。
- (2)將陣列名稱指定給指標。

【例】int *ptr;

```
int list[10];
```

```
ptr=&list[0]
```

相當於

```
ptr=list;
```

2.當指標指向一陣列的開頭，陣列元素的存取即可透過指標來進行，此時指標可當做陣列名稱使用，而陣列名稱亦可當指標使用。

【例】續上例，當ptr指向list開頭，則以下四種寫法在意義上完全相同：

1.list[index]

2.*(ptr+index)

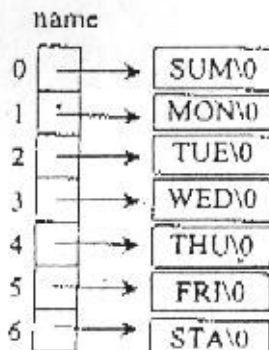
3.*(list+index)

4.ptr[index]

3.若以指標做為陣列元素，可構成指標陣列。

【例1】int *sale[10];/*sale為由10個指向整數記憶體的指標所組成之陣列*/

```
char *name[]={"SUN","MON","TUE","WED","THU","FRI","SAT"};  
則name的記憶體配置如下圖
```



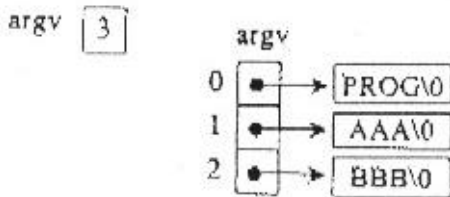
【例2】若在主程式main中有宣告參數如下：

```
main(int argc, char *argv[ ])
```

且在命令列鍵入下列命令（令執行檔名為PROG.EXE）：

```
>PROG AAA BBB
```

則在PROG執行時，命令列的引數個數會儲存在argc，而引數本身則儲存在指數陣列argv中，如下圖所示：



(四)指向函數的指標：

1.由於函數名稱儲存的是該函數的起始位址，因此可如陣列名稱一般，將函數的位址存入指標中，再透過指標來呼叫該函數。

2.欲宣告指向函數的指標，其格式如下：

```
type (*fun_ptr)( );
```

其中fun_ptr是一個指標變數的名稱，這個指標可指向一個傳回型態為type之函數。

3.若已將函數名稱（代表其位址）指定給fun_ptr，則可以透過該指標來進行函數呼叫，格式如下：

```
(*fun_ptr) ( 實際參數串列 )
```

【例】若已定義一個函數如下：

```
double capital_sum(int year, float rate){ ...};
```

則下列程式碼：

```
double(*fp)( ), sum; ⇒宣告fp為一指標，它可指向傳回double值的函數。
```

```
...
```

```
fp=capital_sum; ⇒將capital_sum的位址存入fp中。
```

```
...
```

```
sum=(*fp)(3, 0.15); ⇒進行函數呼叫，相當於執行sum
                    =capital_sum(3, 0.15);
```

四、屬函數型程式語言（Functional Programming language）的Scheme語言，是LISP的衍生語言之一。mapcar函數可提供一個簡單型式的Apply-to-all，定義如下：

```
(DEFINE(mapcar fun lis)
  (COND
    ((NULL? Lis) ( ))
    (ELSE(CONS(fun(CAR lis))(mapcar fun(CDR lis))))
  )
)
```

試問：

(一)(mapcar sub1 '(50 60 70))的執行結果如何？（10分）

(二)(mapcar (lambda (n) (* n n)) (3 4 5 6))的結果又如何呢？（10分）

答：

(一)執行結果為 (49 59 69)

(二)執行結果為 (27 64 125 216)

五、Java語言中的陣列（array）和C語言中的陣列語法類似，但在設計上仍有不同之處。試問，其主要著眼點在那裡？請說明。（20分）

答：

(一)Java的陣列屬於參考型態，在宣告陣列時並不具有記憶體，必須以new指令動態配置記憶體給它。例如以下敘述可讓陣

列a擁有10個整數之記憶體

```
int a[ ]  
a=new int[10];
```

或

```
int a[ ]=new int[10];
```

而C的陣列並不屬於參考型態，在宣告陣列時即具有記憶體。

例如以下宣告即可讓陣列a擁有10個整數之記憶體。

```
int a[10];
```

(二)Java的陣列在宣告時，中括號[]可放在陣列名稱之前或之後，例如以下兩組敘述之意義相同

```
int a[ ]=new int[10];  
int b[ ][ ]=new int[10][10];
```

或

```
int[ ] a=new int[10];  
int[ ][ ] b=new int[10][10];
```

而C的陣列在宣告時，中括號[]只能放在陣列名稱之後。

(三)Java沒有提供指標型態，因此無法透過指標之運算來存取陣列元；但C有提供指標型態，可透過指標之運算來存取陣列元素，例如

```
int a[10],*pa;  
pa=a;
```

則*(pa+5)可代表a[5]這個陣列元素。