【資訊處理】

《程式語言》

試題評析

今年程式語言的考題有60%可說是基本分的題目，包括第一、三、五題，正常的情況下都應該要拿到分數，而第二及第四題則較具有挑戰性，也可能是分數高低的關鍵。第一題是簡單的程式設計問題，撰寫一個求Fibonacci number的遞迴函數，算是送分題；第二題要寫出可產生3之倍數的BNF文法，想要正確完整地導出來有一定的困難度存在；第三題考捷徑計算的問題，應該也不難；第四題考Ada或Java等並行程式語言的合作同步及競爭同步機制，除了要能了解其意義之外，如何將這兩種同步機制描述清楚或舉例說明，就要看考生的實力夠不夠了；最後一題要計算兩虛擬程式片段的時間複雜度，因這兩段程式都很簡單，故只要觀念會就寫得出來。綜合而言，今年的題目一般考生應可拿60分左右，程度較好者70~80分或更高都有希望。

一、在達文西密碼一書中有提及Fibonacci序列為1，1，2，3，5，8，13，21，....，即其定義為f (n)=f (n-1)+f (n-2)，n≧3，f (1)=f (2)=1，請以PASCAL，C，JAVA中任一種語言寫一程式來計算Fibonacci數f (n)，並請使用遞迴副程式(recursive call，即副程式呼叫本身)。（20分）

答：以C語言撰寫程式如下：

```
#include   <stdio.h>
int Fibonacci(int n);
void main()
{
     int     i;
     printf("前20個Fibonacci數為：\n");
   for ( i=1; i <= 20; i++ )
          printf("%d ", Fibonacci(i) );
     printf("\n");
}

int Fibonacci(int n)
{
     if ( (n == 1) || (n == 2) )
          return 1;
     else
          return Fibonacci(n-1) + Fibonacci(n-2);
}
```

執行結果：

```
前20個Fibonacci數為：
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

二、試寫出一BNF文法，使其產生二進位數字，且數字之值為3之倍數。（20分）

答：3倍數之文法如下

<N3>　→　<N3>0 | <N1>1 | 0

<N1> → <N2>0 | <N3>1 | 1
<N2> → <N1>0 | <N2>1

三、(一)若程式語言本身不提供捷徑計算，則下列虛擬指令程式片段會出現什麼錯誤訊息？（10分）

index :＝1；
while(index＜＝listlen) and (list【index】◇key) do index :=index+1；
（假設 list【1..listlen】為被查詢之陣列，而key為要查詢之值）

(二)試說明Ada程式語言預設為不提供捷徑計算之理由，並舉例說明之。（10分）

答：

(一)若程式語言不提供捷徑計算，則此程式片段在執行時有可能出現陣列柱標超出範圍(array index out of range)之錯誤。因爲當陣列list中找不到要查詢之key值時，index將會到達listlen+1，此時and的第一個條件（index <= listlen）已不成立，但因and沒有捷徑計算功能，仍會執行第二個條件（list[index] <> key），此index註標即已超出宣告範圍。

(二)Ada語言對於捷徑計算功能提供了較高的使用彈性，因其布林運算子是否要進行捷徑計算，可由使用者自行決定，其中預設的and和or運算子並不提供捷徑計算能力，但若使用者想執行捷徑計算時，可改用and then和or else，這兩個布林運算子皆具有捷徑計算能力。例如要解決前述程式片段的問題，在Ada語言中可改寫程式如下：

```
index := 1;
while ( index <= listlen ) and then ( list[index] /= key ) loop
    index = index + 1;
end loop;
```

當陣列list中找不到要查詢之key值時，and then的第一個條件不成立，則立即產生false值，不會再執行第二個條件，故將不會發生陣列柱標超出範圍之錯誤。

四、一個平行程式語言（concurrent programming language）較非平行程式語言在設計上須增加競爭（competition）及合作（cooperation）同步（synchronization）機制，試以Ada或Java程式語言分別說明之。（20分）

答：以Ada語言爲例。

(一)合作同步(cooperation synchronization)可要求一個任務A必須等待另一任務B完成某項特定工作後，A才能繼續執行。在Ada中要進行合作同步，可在任務(task)的accept子句前面加上警衛(when)子句，則只有當when條件成立時，此accept入口才能開啓(open)接受訊息，並進行會晤(rendezvous)，若when的條件不成立，則accept入口將關閉(close)而無法接受訊息。透過when的條件的適當設定，可讓不同任務輪流執行工作。

例如下列Buf_Task中宣告一整數空間buffer，並定義兩個入口：DEPOSIT與FETCH，可分別由Producer任務與Consumer任務呼叫來存放資料與取出資料，兩者以合作同步方式互動如下：先由Producer呼叫Buf_Task之DEPOSIT存放資料至buffer中，設定buffer_full爲true，再由Consumer呼叫FETCH取出buffer中的資料並傳回使用，此時設定buffer_full爲false，再輪回Producer工作，依序循環下去。

```
task Buf_Task;
    entry DEPOSIT(item : in integer);
    entry FETCH(item : out integer);
end Buf_Task;
task body Buf_Task is
    buffer : integer;
    buffer_full : boolean := false;
    loop
        select
```

```
            when not buffer_full =>
                accept DEPOSIT(item : in integer) do
                    buffer := item;
                    buffer_full := true;
                end DEPOSIT;
            or
            when buffer_full =>
                accept FETCH(item : out integer) do
                    item := buffer;
                    buffer_full := false;
                end DEPOSIT;
            end select;
        end loop;
    end Buf_Task;

    task Producer;
    task Consumer;
    task body Producer is
        new_value : integer;
        begin
            loop
                -- produce new value --
                Buf_Task.DEPOSIT(new_value);
            end loop;
        end Producer;
    task body Consumer is
        stored_value : integer;
        begin
            loop
                Buf_Task.FETCH(stored_value);
                -- consume stored value --
            end loop;
        end Consumer;
```

(二)競爭同步(competition synchronization)是指允許兩個或多個任務競爭使用共享資料（沒有規定先後順序），但必須互斥使用而非同時使用。在Ada中要進行競爭同步，可在管理共享資料的任務中提供多個同時開啓的入口點，而由欲存取共享資料的任務競爭送訊息給這些開啓的入口點，只要有任務已送訊息進入某一入口點，其他想存取共享資料的任務皆必須先等待，待先前任務離開之後，再安排等待中的任務進入存取共享資料。

例如修改前例中的Buf_Task任務，令其包含一個長度100的整數陣列作爲緩衝區，並設計成環狀佇列(circular queue)，其body task程式如下：

```
    task body Buf_Task is
        buffer : array(1..100) of integer;
```

```
        filled : integer range 0..100 := 0;
        next_in, next_out : integer range 1..100 := 1;
        loop
            select
            when filled < 100 =>
                accept DEPOSIT(item : in integer) do
                    buffer(next_in) := item;
                end DEPOSIT;
                next_in := (next_in mod 100) + 1;
                filled := filled + 1;
            or
            when filled > 0 =>
                accept FETCH(item : out integer) do
                    item := buffer(next_out);
                end DEPOSIT;
                next_out := (next_out mod 100) + 1;
                filled := filled – 1;
            end select;
        end loop;
    end Buf_Task;
```

如此將可讓多個Producers (Producer1, …, Producerm，格式皆如前例之Producer)及多個Consumers (Consumer1, …, $Consumer_n$，格式皆如前例之Consumer)同時作業，共用這100個整數的緩衝區，只要緩衝區還有空位，即可讓Producer進去存放資料，而只要緩衝區內仍有資料，即可讓Consumer進去取出資料，但任何時刻皆只允許一個任務進入（互斥存取）。

五、試算出下列虛擬指令之時間複雜度：（每小題10分，共20分）

(一)
```
for i＝0 to n do
begin
    j＝i；
    while j ＞0 do j = j / 2；
end
```

(二)
```
for i＝0 to n do
begin
    j＝i；
    while j ＞0 do j = j -1；
end
```

答：

(一)$O(n\log_2 n)$

(二)$O(n^2)$