《程式語言》

試題評析	今年程式語言考題都以觀念爲主，雖沒有艱深的程式設計，但較爲注重考生的思考能力，因此作答的彈性較大。 第一題：考驗考生對電腦中亂數形成方式的了解，偏向程式設計的背景知識，需要有較多的實務經驗才好作答。 第二題：函數的傳址呼叫與傳參考呼叫，屬於較好拿分的程式設計知識。 第三題：費式數列考題，但不同於以往的是要提出解決遞迴重複呼叫的遞迴版本。必須使用動態程式設計的概念，對考生而言較具挑戰。 第四題：例外處理，屬於基本性的題目，也較好拿分。 第五題：也屬考古題，詳細比較動態繫結的優缺點，並與複載做比較，需對物件導向概念較爲熟悉才能寫出較完整的答案。 綜觀而論，今年題目以考古題居多，程度中等的同學應可拿到 50-60 分，程度較好的同學拿到 80 分以上也並非難事。
高分閱讀	第一題：金乃傑，程式語言考題補充，頁 17 (第 19 題)。 第二題：金乃傑，程式語言講義第二回，頁 9、《程式語言總複習》，頁 47-48。 第三題：金乃傑，程式語言講義第二回，頁 21-24。 第四題：金乃傑，程式語言講義第三回，頁 59、頁 62。 第五題：金乃傑，程式語言講義第四回》頁 18-22。

一、編寫程式有時需叫用（invoke）能產生亂數（random number）的副程式。請問為何叫用副程式以產生亂數前應先設定亂數種子（random seed）？作為亂數種子的值可以是固定數值也可以是系統時間，請各舉一例分別說明固定數值及系統時間之適用時機。(20 分)

答：

(一)使用亂數種子的原因：目前大部分電腦的運算都是來自於人類先定義好的規則，因此在電腦中沒有真正的亂數（亂數都是定義好的）。電腦產生亂數必須要透過亂數表（random table），在亂數表中從不同位置開始讀取就能得到不同的數值。而亂數表位置的索引就是所謂的亂數種子（整數值），透過亂數種子可以對照亂數表得出不同的亂數值，因此若亂數種子相同，得到的亂數也相同，故必須先設定亂數種子才能產生不同的亂數。

(二)亂數種子適用性比較如下表：

	固定數值	系統時間
特性	使用手動或系統設定（如遞增或其他運算方法產生）的亂數，若設定值相同，則產出的亂數也相同。	使用時間戳記作爲亂數種子，由於每次執行程式的時間必不相同，因此可以達到不同的亂數。
適用	作爲連續亂數輸出。由於系統時間是以秒爲單位，因此當連續產生亂數時，若使用系統時間建立亂數會使得同一秒內執行的敘述得到相同的亂數種子，造成輸出相同的亂數值。爲了避免此一現象，會使用固定數值的系統設定改變亂數種子。	作爲程式中的初始亂數，或建立單一的亂數元素。

二、程式語言中交換兩個變數時，常以暫存變數 tmp 協助交換。

(一)請以虛擬碼（pseudo code）寫出如何在主程式中交換兩個變數（v1 及 v2）。(5 分)

(二)以 C 語言的指標（pointer）寫法，寫出主程式如何叫用副程式，以交換兩個整數變數（v1 及 v2）。再寫出完整的副程式。副程式名稱為 swap，且無傳回值（return value）。（10 分）

(三)以傳參考叫用（call by reference）的方式，用 C++將交換兩個整數變數（ν1 及 ν2）的過程寫成完整的副程式。副程式名稱為 swap，且無傳回值。(5 分)

答：

(一)考慮以下虛擬碼：

```
int main(){
    int v1 = 5, v2 = 10;
    int temp = v1;
    v1 = v2;
    v2 = temp;
}
```

(二)考慮以下 C 語言程式碼：

```
void swap(int* v1, int* v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}

int main(){
    int v1 = 5, v2 = 10;
    printf("v1 = %d, v2 = %d\n", v1, v2);
    swap(&v1, &v2);
    printf("v1 = %d, v2 = %d\n", v1, v2);
}
```

輸出：

```
v1 = 5, v2 = 10
v1 = 10, v2 = 5
```

(三)考慮以下 C++程式碼：

```
void swap(int &v1, int &v2){
    int temp = v1;
    v1 = v2;
    v2 = temp;
}

int main(){
    int v1 = 5, v2 = 10;
    printf("v1 = %d, v2 = %d\n", v1, v2);
    swap(v1, v2);
    printf("v1 = %d, v2 = %d\n", v1, v2);
}
```

輸出：

```
v1 = 5, v2 = 10
v1 = 10, v2 = 5
```

三、費氏（Fibonacci）數列定義如下：

$F_0=0$；$F_1=1$；$n>1$時，$F_n=F_{n-1}+F_{n-2}$

（一）以遞迴（recursive）方式寫出可得出費氏數列中第 n 個數的虛擬碼。（5 分）

（二）若要得出F_5，則F_0、F_1、F_2、F_3、F_4各被重覆叫用幾次？（5 分）

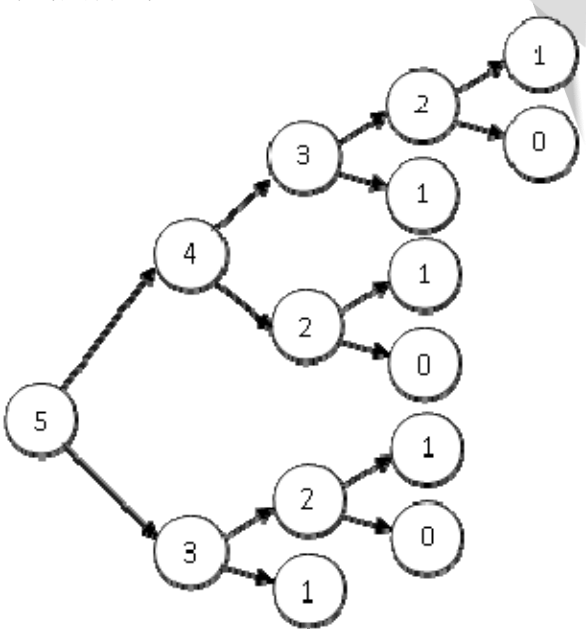
（三）為了避免因為遞迴叫用F_{n-1}和F_{n-2}所浪費的重覆計算時間，如何修改(一)中的虛擬碼，使得該函式仍是以遞迴方式進行，但計算過的F_{n-1}和F_{n-2}不須重覆計算？(10 分)

答：

(一)考慮以下虛擬碼：

```
3  int fib(int n){
4      if(n <= 0) return 0;
5      else if(n == 1) return 1;
6      else return fib(n-1)+fib(n-2);
7  }
8
9  int main(){
10     for(int i = 0; i < 10; i++) printf("%d ", fib(i));
11 }
```

(二)呼叫次數如下：



F0：3 次

F1：4 次

F2：3 次

F3：2 次

F4：1 次

(三)以動態程式規劃（Dynamic Programming）的概念，建立陣列暫存運算結果，避免重複函數呼叫。考慮以下虛擬碼：

```
int f[100];                          //建立暫存結果陣列
                                     //[利用全域變數會被設為0的特性]
                                     //[檢驗是否暫存陣列有更新過    ]
int fib(int n){
    if(n <= 0) return 0;
    else if(n == 1) return 1;

    if(f[n] != 0) return f[n];       //如果暫存結果陣列有值
                                     //，就不遞迴呼叫
    else{                            //否則一樣遞迴往下追蹤
        f[n] = fib(n-1)+fib(n-2);
        return f[n];
    }
}

int main(){
    printf("%d", fib(5));
}
```

四、以 C++或 JAVA 為例，說明程式語言如何提供異常處理（exception handling）的機制。並根據所描述的機制，分別就強固性（robust）、可讀性及可維護性，說明異常處理的機制對應用程式的重要性。(20 分)

答：

(一)異常處理指的是當系統偵測到軟體或硬體引發的不正常事件或錯誤（異常）時，會主動執行異常處理程式（exception handler）之程式單元來處理此一例外狀況。考慮以下 Java 程式：

```
public class exception {
    public static void main(String[] args) {
        int i = 0, arr[] = {2, 3, 5, 7, 11};
        try{
            while(true){
                System.out.println(arr[i++]);
            }
        }catch(Exception e){
            System.out.println("Catched Integer exception!!!");
        }
    }
}
```

說明：Java 的異常處理敘述必須被 try-catch 包住，其中 try 中是可能發生異常的程式碼，catch 爲發生例外後的處理動作，當在 try 區塊中發生無法順利執行的敘述，便會直接跳到 try 後的 catch 區塊進行處理。

(二)異常處理對程式的影響：

1.增加程式強固性（robust）：避免程式因發生例外而中止執行。如上例中，由於第六行中的 while 迴圈會不斷執行，若不做異常處理，會發生陣列索引超出範圍的問題，使程式中止執行，甚至造成系統當機。但使用異常處理，可以透過 catch 接住 try 區域中所發生的異常現象，進行問題的補救措施，使程式能繼續正常執行。

2.增加程式可讀性：將一般敘述與錯誤處理指令分離，簡化程式設計。如上例，正常執行的程式碼寫在 try 區域中、異常處理的程式碼寫在 catch 中，如此程式設計師可以輕易的分辨程式碼中那些敘述在正常狀況下不會執行，以節省追蹤（trace）程式碼的成本。

3.提高程式之可維護性：可以依照錯誤種類動態選擇處理方法。在 Java 中可以定義多種異常現象，只需要透過多個 catch 連接即可，如此可以將處理不同異常的程式碼分開設計，提升維護的彈性。

五、何謂晚期繫結（late binding）？晚期繫結相對於早期繫結（early binding）有何優、缺點？以物件導向（object oriented）程式的多形（polymorphism）和複載（overloading）為例，兩者的繫結時機相同或不同？解釋其相同或不同之原因。(20 分)

答：

(一)晚期繫結又稱爲動態繫結，指程序間傳遞訊息時以動態的方式決定執行的操作，達成相同程式碼有多種執行模式。

(二)晚期繫結的優缺點比較如下：

1.優點：

(1)執行彈性高：相同的程式碼即使經過編譯，會依照執行時的訊息產生多種執行模式，增加程式執行的彈性。

(2)大量減少重複撰寫：使用動態繫結達成的虛擬函式，可以在繼承自父類別的函式（函式 A）中呼叫自己 override 的其他父類別函式（函式 B），減少父類別中該函式（函式 B）的修改或爲了些許差異 override 父類別函式（函式 A）的功能。

2.缺點：

(1)可讀性降低：由於程式會因爲執行時期的訊息而改變執行的模式，因此在程式碼的檢視中產生複雜的關聯，增加程式設計師程式碼追蹤時的困難度，造成超出預期的執行結果。

(2)效能低：由於在執行時才決定呼叫的操作，因此程式碼除了無法透過編譯器的完全優化，還會增加執行時期尋找呼叫對象的時間，降低程式的執行效能。

(三)多型與複載繫結時間不同，多型的繫結時間是在執行時期，但複載卻是在程式編譯時期。多型使用虛擬函數來達成，以 C++爲例，虛擬函數必須要在宣告前加上 virtual 關鍵字，使用虛擬函數，當程式在物件中進行訊息傳遞時，才會依照訊息的型態動態選擇對應的執行模式。但複載卻是在編譯的時候就對類別中能使用的方法重新定義，因此屬於編譯時期的繫結。