

1141 NTOUCSE 程式設計 1C 期末考參考答案

姓名：_____ 系級：_____ 學號：_____

114/12/23 (二)

考試時間：13:20 – 16:00

- 考試規則：
1. 請闔上課本，不可參考任何文件包括小考、作業、實習、或是其它參考資料
 2. 你可以在題目卷上直接回答，可以使用鉛筆，但是請在答案卷上寫下題號，並且註明在題目卷上
 3. 有需要的話，可以使用沒有教過的語法，但是僅限於 C 語言
 4. 程式撰寫時請寫完整的程式碼，寫... 的分數很低 (程式裡有重複很多遍的敘述本來就是扣分的)
 5. 不可使用電腦、平板、電子紙、智慧手機、手錶、及工程型計算機
 6. 請不要左顧右盼！請勿討論！請勿交換任何資料！對於題目有任何疑問請舉手發問
 7. 如果你提早交卷，請迅速安靜地離開教室，請勿在走廊喧嘩
 8. 違反上述考試規則視為不誠實的行為，由學校依學務規章處理
 9. 請在題目卷及答案卷上都寫下姓名及學號，交卷時請繳交題目卷及答案卷

1. 我們在課程中說明程式設計時最主要的兩種重複架構是迴圈和遞迴，在你逐步接觸資訊領域的過程中，你會一直有一個疑問就是到底遞迴好還是迴圈好？這不是一個很容易回答的問題，和你的程式經驗有絕對的關聯，也牽涉到其它程式設計方法論例如 Functional Programming 的概念，我們的課程中希望用不同的練習來解答你心中的疑問，請運用我們課程中的範例來比較遞迴與迴圈的應用：

(a) [3] 請以目前主流的處理器架構來說明

Sol:

在目前的處理器架構、在程序化程式設計方法、或是在物件化程式設計方法中當然主要的重複架構是迴圈；在 Functional Programming 設計方法中，主要的重複架構則是遞迴

(b) [3] 以程式的複雜度來說 (變數的數量和程式列數)

Sol:

遞迴函式中只需要注意問題規模 $n-1$ 的解到問題規模 n 的解之間的關聯，程式常常比較簡短，需要顧及的資料變數與狀態變數比較少，在程式偵錯以及程式維護上有相當的好處，例如：計算階乘，計算 x 的 n 次方

(c) [3] 以自動運用多核、多執行緒、多機器的網路運算環境來考量

Sol:

用迴圈撰寫程式時，變數內的資料隨著迴圈執行過程不斷改變，因此所有的動作都有先後性，在 Functional Programming 設計方法中希望沒有隱藏的狀態改變，每一個遞迴呼叫以不同參數來定義其數值，有效地降低程式碼的先後關聯性，如此多執行緒或是分散式處理時編譯器可以做有效率的最佳化將工作分配到不同的核心、執行序、或是處理單元上。

(d) [3] 問題本身架構具有 recursion formula, 例如：

Sol:

費氏序列、帕司卡三角形計算組合數 $C(n,k)$ 、二元樹狀架構巡訪

(e) [3] 問題中「規模 n 問題的解」很直接可以用「規模 $1 \sim n-1$ 問題的解」來表示，例如：

Sol:

最大公因數、擴充歐幾里得法、河內塔、Palindrome 字串檢查、運用旋轉法列舉所有 $n!$ 種排列、二分搜尋法、快速排序、快速選擇法、合併排序法、或是分治法 (divide and conquer) 形式的問題解決方法非常適合使用遞迴來描述

(f) [3] 暴力以深度優先搜尋 (Depth First Search) 方法搜尋一個決策樹時，例如：

Sol:

可以用遞迴架構實作概念上可變層數的迴圈，例如：由 0000 到 9999 數數字、列舉所有 $n!$

排列、列舉 $C(n,k)$ 種組合，列舉所有分割 (partition, Restricted Growth String 數數)、Gray code 生成、老鼠走迷宮、速讀、八后問題、馬的走法、人狼羊菜安全渡河問題

2. [12] 如下列運用多層迴圈來列舉所有 0~9 的可能性時，程式架構很直接，但是最大的問題是什麼？請撰寫一個遞迴函式完成相同功能？

```
1 int i1, i2, i3, i4;
2 for (i1=0; i1<=9; i1++)
3     for (i2=0; i2<=9; i2++)
4         for (i3=0; i3<=9; i3++)
5             for (i4=0; i4<=9; i4++)
6                 printf("%d %d %d %d\n", i1, i2, i3, i4);
```

Sol:

最大的問題在於 scaling，調整數字個數的時候需要修改程式

```
01 #include <stdio.h>
02
03 void countUp(int n, int data[], int pivot);
04
05 int main()
06 {
07     int data[10];
08     countUp(4, data, 0);
09     return 0;
10 }
11
12 void countUp(int n, int data[], int pivot)
13 {
14     int i;
15     if (pivot>=n)
16         for (i=0; i<n; i++)
17             printf("%d%c", data[i], " \n"[i==n-1]);
18     else
19         for (data[pivot]=0; data[pivot]<=9; data[pivot]++)
20             countUp(n, data, pivot+1);
21 }
```

這個遞迴函式可以在呼叫的時候指定任意的數字個數

3. (a) [8] 在 C 程式中該如何用指標語法讓一個一維整數陣列 `int array1D[12]` 同時也是一個二維的整數陣列(使用起來和 `int array2D[3][4]` 一模一樣)，如此相同的一段記憶體可以有兩個不同的存取方法？第 07 列的迴圈會印出 0 1 2 3 10 11 12 13 20 21 22 23。類似的概念也用在想要一個位元組一個位元組來看資料的內容時，我們也可以用指標語法讓這個 `array1D` 陣列看成是一個 48 個位元組的陣列，如此第 09 列的迴圈會印出 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00 0a 00 00 00 0b 00 00 00 0c 00 00 00 0d 00 00 00 14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00

```
01 int i, array1D[12];
02 _____ array2D_____ = _____ array1D;
03 _____ cbuf = _____ array1D; // 或是 array2D
04 for (i=0; i<3; i++)
05     for (j=0; j<4; j++)
06         array2D[i][j] = i*10+j;
07 for (i=0; i<12; i++)
08     printf("%d ", array1D[i]);
09 for (i=0; i<12*sizeof(int); i++)
```

```
10    printf("%02x ", cbuf[i]);
```

Sol:

`int (*array2D)[4] = (int (*)(4)) array1D;` 或是 `int (*const array2D)[4] = (int (*)(4)) array1D;` 都可以使得 `array2D` 成為和 `array1D` 成為使用相同記憶體區段但是型態不同的兩個陣列，如此同樣的陣列可以有兩種不一樣的存取方法，注意因為 `array1D` 的型態是 `int *`，`array2D` 的型態是 `int (*)(4)`，所以在 `array1D` 前面一定要有強制的型態轉換。

`char *cbuf = (char *) array1D;` 或是 `char *const cbuf = (char *) array1D;` 同樣地使得 `cbuf` 和 `array1D` 成為使用相同記憶體區段但是型態不同的兩個陣列，運用 `cbuf[i]` 可以一個一個位元組來存取。

- (b) [6] 很多人不太喜歡二維陣列 `int x[3][4]`；或是 `x` 這個指標的型態定義讓一個變數的型態分開在變數名稱的前後，C 語言中提供 `typedef` 的語法替型態定義別名，請替 `int[4]` 定義一個別名 `I4ARY`，再以 `I4ARY` 定義上面這個二維陣列 `x`，並且用常數指標替這個 `x` 陣列定義一個等效的名稱 `y`，如此下列兩個迴圈都可以分別用 `x` 和 `y` 操作相同的陣列

```
04 for (i=0; i<3; i++)
05     for (j=0; j<4; j++)
06         x[i][j] = i*10+j;
07 for (i=0; i<3; i++) {
08     for (j=0; j<4; j++)
09         printf("%d ", y[i][j]);
10     printf("\n");
```

Sol:

```
typedef int (*I4ARY)[4];
I4ARY x[3];
I4ARY *const y = x;
```

- (c) [6] 請寫出將二維陣列 `int x[3][4]`；傳入函式 `int func(int y[3][4])` 時常見的三種等效的參數宣告方法

Sol:

```
int (*y)[4]
int (*const y)[4]
int y[][4]
```

- (d) [6] 這四種方法不論是哪一種，在函式裡面陣列元素存取的方法都一模一樣是 `y[i][j]`，請問下面程式執行的時候會印出什麼？請解釋原因（假設為記憶體位址為 64 位元）

```
01 #include <stdio.h>
02 void func(int y[3][4])
03 {
04     printf("%d %d", sizeof(y), sizeof(*y));
05 }
06 int main()
07 {
08     int x[3][4];
09     printf("%d ", sizeof(x));
10     func(x);
11     return 0;
12 }
```

Sol:

48, 8, 16

編譯器看 sizeof(x) 的時候 x 的型態是 int[3][4]，sizeof(y) 時 y 是一個指標，型態是 int (*)[4]

4. (a) [5] 請問使用二分搜 binary search 搜尋陣列 int data[1000];中指定資料的先決條件為何?

Sol:

陣列中的資料需要先排好順序，分治法才能正確將問題縮小，當 data[mid]<target 時，在右半邊搜尋，當 data[mid]>target 時，在左半邊搜尋。

(b) [5] 請完成下列迴圈搜尋陣列中指定的資料並傳回該元素的註標，沒有找到的話回傳 -1

```
01 int binarySearch(int target, int data[], int left, int right)
02 {
03     int mid;
04     while (left <= right)
05     {
06         mid = (left+right)/2;
07         if (target == data[mid])
08             return _____;
09         else if (target > data[mid])
10             _____;
11         else
12             _____;
13     }
14     return -1;
15 }
```

Sol:

```
01 int binarySearch(int target, int data[], int left, int right)
02 {
03     int mid;
04     while (left <= right)
05     {
06         mid = (left+right)/2;
07         if (target == data[mid])
08             return mid;
09         else if (target > data[mid])
10             left = mid + 1;
11         else
12             right = mid - 1;
13     }
14     return -1;
15 }
```

(c) [5] 請完成下列 lower_bound()函式搜尋陣列中指定資料並傳回該元素第一次出現的註標，在沒有找到的時候傳回小於指定資料的元素中最大的元素最後一次出現的註標，此函式相當於傳回「小於等於指定元素的陣列元素註標」例如: data 資料陣列如下

0	1	2	3	4	5	6	7	8	9
1	2	2	3	3	3	5	6	6	8

lower_bound(3,data, 0, 9) 得到 3，lower_bound(4,data, 0, 9) 得到 6

```
01 int lower_bound(int target, int data[], int left, int right)
02 {
03     int mid, result = _____;
```

```

04     while (left <= right)
05         if (data[mid]=(left+right)/2] _____ target)
06             _____;
07         else
08             _____, result = _____;
09     return result;
10 }

```

Sol:

```

01 int lower_bound(int target, int data[], int left, int right)
02 {
03     int mid, result = ____right+1____;
04     while (left <= right)
05         if (data[mid]=(left+right)/2]____<____target)
06             ____left = mid + 1____;
07         else
08             ____right = mid - 1____, result = ____mid____;
09     return result;
10 }

```

在 C++ STL 函式庫中也有這個函式 lower_bound(begin, end, target)

- (d) [3] 如何運用題(c)的 lower_bound()函式來得到第一個大於指定數值(target)元素的註標，以上述 data 陣列為例，target 為 3 時需要得到 6，target 為 4 時也是得到 6

Sol:

呼叫 lower_bound(target+1, data, left, right)

- (e) [3] 如何運用題(c)的 lower_bound()函式來得到小於等於指定數值最後一次出現的註標，以上述 data 陣列為例，target 為 3 時需要得到 5，target 為 4 時也是得到 5

Sol:

呼叫 lower_bound(target+1, data, left, right)-1

- (f) [3] 如何運用題(c)的 lower_bound()函式來得到小於指定數值最後一次出現的註標，以上述 data 陣列為例，target 為 3 時需要得到 2，target 為 4 時需要得到 5

Sol:

呼叫 lower_bound(target, data, left, right)-1

5. 如下的氣泡排序(bubble sort)演算法可以看成是 n-1 個回合，每一回合把 [i,n-1] 範圍中最小的元素透過交換的動作移動到正確的位置上：

```

01 for (i=0; i<n-1; i++)
02     for (j=n-2; j>=i; j--)
03         if (x[j]>x[j+1])
04             swap(x, j, j+1);

```

過程中比較大的元素只是慢慢地靠向後方，並沒有特別針對它們處理，所以有人想說如果一個回合把範圍中最小的元素往前搬動，一個回合把範圍中最大的元素往後搬動，能不能提昇演算法的效率，請完成下列程式：

- (a) [5] 上述迴圈中每一回合資料的範圍是陣列的第 i 個到陣列的第 n-1 個元素，現在如果是一次往前、一次往後的話，資料的範圍應該是 [(i+1)/2, (n-1-i/2)]，第二層迴圈的方向需要隨著回合數改變

```

01 for (i=0; i<n-1; i++)

```

```

02     if (i%2==0)
03     {
04         for (j_____ ; j_____ ; j--)
05             if ( x[j]>x[j+1])
06                 swap(x, j, j+1);
07     }
08     else
09     {
10         for (j_____ ; j_____ ; j++)
11             if (x[j]>x[j+1])
12                 swap(x, j, j+1);
13     }

```

Sol:

```

01 for (i=0; i<n-1; i++)
02     if (i%2==0)
03     {
04         for (j_____ = n-2-i/2 _____ ; j_____ >= (i+1)/2 _____ ; j--)
05             if (x[j]>x[j+1])
06                 swap(x, j, j+1);
07     }
08     else
09     {
10         for (j_____ = (i+1)/2 _____ ; j_____ <= n-2-i/2 _____ ; j++)
11             if (x[j]>x[j+1])
12                 swap(x, j, j+1);
13     }

```

- (b) [5] 題(a)的實作裡有看到重複出現的程式碼，雖然已經用函式 swap() 把資料交換的程式碼包起來，使得重複的部份看起來不太多，但是重複會造成偵錯以及維護的困難，我們需要不斷練習把共通的部份抽離，運用變數表示有變化的部份，請完成下列合併的程式

```

01 for (i=0; i<n-1; i++)
02 {
03     start = i%2 ? _____ : _____;
04     end = i%2 ? _____ : _____;
05     dir = i%2 ? _____ : _____;
06     for (j=_____ ; j_____ ; j_____ )
07         if (x[j]>x[j+1])
08             swap(x, j, j+1);
09 }

```

Sol:

```

01 for (i=0; i<n-1; i++)
02 {
03     start = i%2 ? _____ (i+1)/2 _____ : _____ n-2-i/2 _____;
04     end = i%2 ? _____ n-2-i/2 _____ : _____ (i+1)/2 _____;
05     dir = i%2 ? _____ 1 _____ : _____ -1 _____; // dir = i%2*2-1;
06     for (j=_____ start _____ ; j_____ != end+dir _____ ; j_____ += dir _____ )
07         if (x[j]>x[j+1])
08             swap(x, j, j+1);
09 }

```

- (c) [5] 題(b)的實作裡如果省掉 start, end, 和 dir 三個變數的話，程式可以寫成

```

01 for (i=0; i<n-1; i++)
02     for (j=_____; j_____; j_____)
03         if (x[j]>x[j+1])
04             swap(x, j, j+1);

```

Sol:

```

01 for (i=0; i<n-1; i++)
02     for (j= i%2 ? (i+1)/2 : n-2-i/2 ; j != (i%2 ? n-2-i/2 : (i+1)/2) + (i%2*2-1) ;
03         j += (i%2*2-1) )
04         if (x[j]>x[j+1])
05             swap(x, j, j+1);

```

(d) [5] 如果增加變數去紀錄交換的次數的話，這個數字和原本 bubblesort 演算法紀錄到的有什麼關係？

Sol:

是相同的，就是這個序列的逆序數。