1041 國立台灣海洋大學資訊工程系 1C 程式設計 期末考參考解答

姓名：_____

系級：_____

學號：_____

105/01/06 (三)

考試時間：**13:20 – 16:20**

請儘量回答，總分有 **122**，看清楚每一題所佔的分數再回答

考試規則：1. 不可以翻閱參考書、作業及程式
2. 不可以使用任何形式的電腦 (包含手機、計算機、相機以及其它可運算或是連線的電子器材)
3. 請勿左顧右盼、請勿交談、請勿交換任何資料、試卷題目有任何疑問請舉手發問 (看不懂題目不見得是你的問題,有可能是中英文名詞的問題)、最重要的是隔壁的答案不見得比你的好，白卷通常比錯得和隔壁一模一樣要好
4. 提早繳卷同學請直接離開教室，請勿逗留喧嘩
5. 違反上述任何一點之同學一律依照學校規定處理
6. 繳卷時請繳交***簽名過之試題卷及答案卷***

1. 有下列的使用者自訂結構型態

```
struct data {
    char name[10];        // 姓名
    char gender;          // 性別
    char mathscore;       // 數學成績
};
```

a. [2] 請定義一個這種結構型態的指標 students

Sol:

```
struct data *students;
```

b. [2] 請用 malloc() 函式配置 100 個 struct data 型態元素的陣列，記錄在 students 變數裡

Sol:

```
students = (struct data *) malloc(sizeof(struct data)*100);
```

c. [4] 請檢查上面步驟是否有配置成功，有的話請用一個迴圈將所有 100 個學生的 name 欄位的資料都清除為空字串

Sol:

```
if (students!=NULL)
    for (int i=0; i<100; i++)
        students[i].name[0] = 0;
```

d. [2] 請用 free() 函式釋放上面用 malloc() 取得的記憶體

Sol:

```
free(students);
```

e. [2] 請問要使用 malloc() 和 free() 需要引入哪一個標頭檔案?

Sol:

stdlib.h 或是 malloc.h

2. 在 stdlib.h 裡面有一個 qsort() 的工具程式，它會用『快速排序法』來把一整個陣列裡的資料依照指定的比較大小方法來排順序，例如上題中 100 個學生的 students 陣列，就可以用 name 欄位來排大小 (假設 name 欄位裡面都是英文的資料)，程式如下:

```
#include <stdlib.h>
#include <string.h> // 使用 strcmp(str1, str2) 函式, str1 < str2 則函式回傳 -1, str1 == str2 則回傳 0, 否則回傳 1
struct data {
…
```

```
};
int compare(const void *a, const void *b) {
      return strcmp((*(struct data *)a).name, (*(struct data *)b).name);
}
int main() {
    …
     qsort(students, 100, sizeof(struct data), compare);
    …
}
```

a. [3] 請解釋上面 compare() 函式裡 a, b 兩個指標所指的東西對 qsort() 來說代表什麼資料

Sol:

在 qsort() 函式排序時，需要以一個區塊為單位來比較大小，如果需要的話交換其順序，例如上面程式裡是大小為 sizeof(struct data) 的連續記憶體為單位，也就是以一個完整的結構為單位，每次 qsort()需要比對兩塊資料的大小時，它就呼叫 compare()，把兩塊資料的記憶體位址傳遞進來，其中 a 就是第一塊的記憶體位址，b 是第二塊的記憶體位址

b. [3] 請解釋 compare() 函式裡為什麼不能寫 (*a).name 而要寫 (*(struct data *)a).name

Sol:

但是 qsort() 並不曉得使用者的 struct data 的結構型態，qsort() 是一個函式庫裡已經寫好的函式，所以它只能傳遞 const void * 型態的記憶體位址，compare() 函式裡面需要自己去告訴編譯器這個記憶體位址是 struct data * 的型態，如此編譯器才允許程式裡去存取 name, gender, mathscore 等欄位，也就是 *(struct data*)a 其實代表一個結構變數，(*(struct data*)a).name 就是它的 name 欄位，也可以寫成 ((struct data *)a)->name

c. [3] 上面是依照字典順序由小到大排序，請問怎麼改成由大到小排序

Sol:

```
int compare(const void *a, const void *b) {
      return strcmp((*(struct data *)b).name, (*(struct data *)a).name);
}
```

d. [6] 如果要改成主要是用 mathscore 由大到小排序，當成績相等時才用姓名由小到大排序該怎麼改

Sol:

```
int compare(const void *a, const void *b) {
      struct data *pa = (struct data *) a, *pb = (struct data *) b;
      if (pa->mathscore == pb->mathscore)
            return strcmp(pa->name, pb->name);
      else
            return pb->mathscore - pa->mathscore;
}
```

3. 所謂的迴文 (Palindrome) 是指一個字串去掉標點符號和空白以後由前面看和後面看是一模一樣的字串(大小寫視為相同)，例如: "A man, a plan, a canal, Panama!", "Amor, Roma", "race car", "taco cat", "Was it a car or a cat I saw?"，或是 "No 'x' in Nixon"，請依照下面要求撰寫程式來檢查一個字串是不是迴文 (如果一個字串中沒有任何大寫或是小寫的英文，視為不是迴文)

a. [15] 首先請運用 gets() 由鍵盤讀取一個不超過 100 個字元的字串到字元陣列中，然後請寫一小段程式把不是 'a'~'z' 也不是 'A'~'Z' 的都去掉，並把所有字母轉換成大寫字母，請寫一個**迴圈**判斷是不是迴文，執行結果如下

輸入: A man, a plan, a canal, Panama!<enter>

輸出:是迴文

輸入: No 'y' in Nixon<enter>

輸出:不是迴文

Sol:

```
#include <stdio.h>
#include <string.h>

int main()
{
    int i, len, lenOriginal;
    char buf[101];
    gets(buf);
    lenOriginal = strlen(buf);
    for (i=len=0; i<lenOriginal; i++)
        if ((buf[i]>='A')&&(buf[i]<='Z'))
            buf[len++] = buf[i];
        else if ((buf[i]>='a')&&(buf[i]<='z'))
            buf[len++] = buf[i]-'a'+'A';
    buf[len] = 0;
    for (i=0; i<len/2; i++)
        if (buf[i]!=buf[len-i-1])
            break;
    if ((len<=0)||(i<len/2))
        printf("不");
    printf("是迴文\n");
    return 0;
}
```

b. [15] 請修改上題判斷迴文的部份 (不需要寫取讀字串, 刪除標點與轉換大小寫), 撰寫一個**遞迴**函式 int isPalindrome(int start, int end, char str[]) 來判斷 (回傳 1 代表是迴文, 0 代表不是迴文, 請在 main() 裡面列印)

Sol:

```
int isPalindrome(int start, int end, char buf[]);
    …
int main() {
    …
    if (!isPalindrome(0, strlen(buf)-1, buf))
        printf("不");
    printf("是迴文");
    …
}
int isPalindrome(int start, int end, char buf[])
{
    if (start >= end)
        return 1;
    if (buf[start] != buf[end])
        return 0;
    else
        return isPalindrome(start+1, end-1, buf);
}
```

c. [5] 請修改上題的遞迴函式成為 int isPalindrome(int size, char str[]), 其中 size 代表需要判

斷字串的長度

Sol:

```
int isPalindrome(int size, char str[]);
        …
int main() {
        …
    if (!isPalindrome(strlen(buf), buf))
        printf("不");
    printf("是迴文");
        …
}
int isPalindrome(int size, char str[])
{
    if (size <= 1)
        return 1;
    if (str[0] != str[size-1])
        return 0;
    else
        return isPalindrome(size-2, str+1);
}
```

```
5
SATOR
AREPO
TENET
OPERA
ROTAS
```

右圖的資料代表二維的迴文，第一列的數字 N 代表接下來是 NxN 的迴文,可以看到迴文的第五列是第一列反轉過來的字串，第四列是第二列反轉過來的字串，第三列是自己反轉過來的字串，如果是偶數列的話就不會有自己反轉的這一列)，假設輸入都是 'A'~'Z' 之間的字母，每一列最多只有 20 個字元

d. [15] 請用 gets() 配合 sscanf() 讀入這些資料，用**迴圈**撰寫判斷二維迴文的程式

Sol:

```
#include <stdio.h>

int main()
{
    int i,j, size;
    char buf[21], matrix[20][21];
    gets(buf);
    sscanf(buf, "%d", &size);
    for (i=0; i<size; i++)
        gets(matrix[i]);

    for (i=0; i<(size+1)/2; i++)
        for (j=0; j<size; j++)
            if (matrix[i][j] != matrix[size-1-i][size-1-j])
            {
                printf("不是迴文\n");
                return 0;
            }
    printf("是迴文\n");
    return 0;
}
```

e. [10] 請撰寫**遞迴**函式 int isPalindrome2D(char matrix[][21], int nrow, int ncol) 來判斷

Sol:

```
#include <stdio.h>

int isPalindrome2D(char matrix[][21], int nrow, int ncol);

int main()
{
    int size;
    char matrix[20][21];
      …
    if (!isPalindrome2D(matrix, size, size))
        printf("不");
    printf("是迴文\n");
      …
}

int isPalindrome2D(char matrix[][21], int nrow, int ncol)
{
    int j;
    if (nrow <= 0)
        return 1;
    for (j=0; j<ncol; j++)
        if (matrix[0][j] != matrix[nrow-1][ncol-1-j])
            return 0;
    return isPalindrome2D(&matrix[1], nrow-2, ncol);
}
```

這個遞迴函式也有使用迴圈，就好像在寫迷宮的遞迴程式，在每一次函式執行過程中也會用一次的迴圈，主要是考慮整個函式的可讀性；也可以把它寫成完全沒有迴圈的形式，

```
if (!isPalindrome2D(matrix, 0, size*size-1, size))
    printf("不");
printf("是迴文\n");
…
int isPalindrome2D(char matrix[][21], int start, int end, int size)
{
    if (start >= end)
        return 1;
    if (matrix[start/size][start%size] == matrix[end/size][end%size])
        return isPalindrome2D(matrix, start+1, end-1, size);
    else
        return 0;
}
```

這個函式基本上也就是把整個二維陣列拉直了來看而已，或是

```
    if (!isPalindrome2D(matrix, 0, 0, size))
        printf("不");
    printf("是迴文\n");

int isPalindrome2D(char matrix[][21], int row, int col, int size)
{
    if (row*size+col >= size*size/2)
        return 1;
```

```
        if (matrix[row][col] == matrix[size-1-row][size-1-col])
            return isPalindrome2D(matrix, row+(col+1)/size, (col+1)%size, size);
        else
            return 0;
    }
```

還有很多其它的寫法，沒有標準答案

4. 三個柱子的河內塔遞迴函式範例如下

```
01 void move(char from_peg, char to_peg, char aux_peg, int nDisks) {
02     if (nDisks == 1) {
03         printf("1 %c %c\n", from_peg, to_peg);
04     } else {
05         move(from_peg, aux_peg, to_peg, nDisks - 1);
06         printf("%d %c %c\n", nDisks, from_peg, to_peg);
07         move(aux_peg, to_peg, from_peg, nDisks - 1);
08     }
09 }
```

a. [5] 請寫出函式呼叫 move('A', 'C', 'B', 3) 在螢幕上的輸出

Sol:

1 A C
2 A B
1 C B
3 A C
1 B A
2 B C
1 A C

b. [5] 這個函式移動的次數一定是最少的，如果是只有一個碟子時，只要移動 $a_1 = 1$ 次，如果只有兩個碟子時，只要移動 $a_2 = 3$ 次，…，如果有 n 個碟子時，你知道最大的碟子至少要移動 1 次，但是在移動它之前，要把上面 n-1 個碟子都先移到輔助的柱子上 (由 from_peg 到 aux_peg)，那麼最少要移動 a_{n-1} 次，然後才能移動最大的碟子 (由 from_peg 到 to_peg)，然後再把輔助的柱子上面的 n-1 個碟子移到目標柱子上，又需要 a_{n-1} 次,總共需要移動 $a_n = 2\,a_{n-1} + 1$ 次，請寫出 a_n 的通式 (用 n 表示出來)

Sol:

$a_1 = 1, a_2 = 1 + 2, a_3 = 1 + 2 + 2^2, a_4 = 1 + 2 + 2^2 + 2^3, \ldots, a_n = 2^n - 1$

c. [5] 如果我們禁止 A 與 C 兩個柱子之間直接搬移，也就是說要由 A 搬到 C 只能先由 A 搬到 B 然後再由 B 搬到 C，所以最大的碟子 (第 n 個) 要由 A 搬到 C 至少要搬兩次，在過程裡還要先把 n-1 個碟子由 A 先搬到 C，把最大的碟子由 A 搬到 B，再把 n-1 個碟子由 C 搬回 A，把最大的碟子由 B 搬到 C，再把 n-1 個碟子由 A 搬到 C，參考題 b. 請寫出最少需要移動幾次 b_n 的 recursion 表示法

Sol:

$b_n = 3\,b_{n-1} + 2$ $\quad (b_1 = 2, b_2 = 2\,(1 + 3), b_3 = 2(1 + 3 + 3^2), b_4 = 2(1 + 3 + 3^2 + 3^3), \ldots, b_n = 3^n - 1)$

d. [10] 請修改上面的 move() 函式來完成題 c. 的遞迴程式
右圖是一個簡單的執行範例 move('A', 'C', 'B', 2);

```
1 A B
1 B C
2 A B
1 C B
1 B A
2 B C
1 A B
1 B C
```

Sol:

```
#include <stdio.h>

void move(char from_peg, char to_peg, char aux_peg, int nDisks);
```

```
int main()
{
	int nDisks;
	printf("請輸入碟片數: ");
	scanf("%d", &nDisks);
	move('A', 'C', 'B', nDisks);
	return 0;
}

void move(char from_peg, char to_peg, char aux_peg, int    nDisks)
{
	if (nDisks == 1)
	{
		if (aux_peg=='B')
		{
			printf("1 %c %c\n", from_peg, aux_peg);
			printf("1 %c %c\n", aux_peg, to_peg);
		}
		else
			printf("1 %c %c\n", from_peg, to_peg);
	}
	else
	{
		if (aux_peg=='B')
		{
			move(from_peg, to_peg, aux_peg, nDisks - 1);
			printf("%d %c %c\n", nDisks, from_peg, aux_peg);
			move(to_peg, from_peg, aux_peg, nDisks - 1);
			printf("%d %c %c\n", nDisks, aux_peg, to_peg);
			move(from_peg, to_peg, aux_peg, nDisks - 1);
		}
		else
		{
			move(from_peg, aux_peg, to_peg, nDisks - 1);
			printf("%d %c %c\n", nDisks, from_peg, to_peg);
			move(aux_peg, to_peg, from_peg, nDisks - 1);
		}
	}
}
```

5. [10] 實習裡寫了快速排序法，基本的架構如下

```
01 void quick_sort(int array[], int istart, int iend) {
02       if (iend-istart > 1) {
03            int pivot = place_midst(array, istart, iend);
04            quick_sort(array, istart, pivot-1);
05            quick_sort(array, pivot+1, iend);
06       }
07       else if (iend-istart == 1)
08            if (array[istart]>array[iend]) swap(array, istart, iend);
09 }
10 int place_midst(int array[], int istart, int iend) { // 將 array[istart] 放到正確位置 array[pivot]
11       …                                              // 而且比較小的放在前半，大於或是等於的放在後半
12 }
```

假設 place_midst() 函式是可以正常運作的，它會回傳那個正確的位置 pivot，你現在需要修改上面的 quick_sort() 函式變成 int find_kth_element(int array[], int istart, int iend, int k)，我們只需要找到由小到大算起來前 k 個元素就好了，函式回傳第 k 個元素的數值，請用遞迴的二分法來完成

例如下面程式可以印出前 3 個元素

```
int array[] = {3, 1, 4, 9, 6, 2, 7}, kth, n = 7;
kth = find_kth_element(array, 0, n-1, 3);
for (int i=0; i<2; i++)
      printf("%d ", array[i]);
printf("%d\n", array[2]);
```

Sol:

```
int find_kth_element(int array[],    /* input/output - array to sort                          */
                     int istart,     /* input - starting element of the array to consider */
                     int iend,       /* input - ending element of the array to consider    */
                     int k)          /* find the k-th non-decreasing element                  */
{
      if (iend-istart > 1)
      {
            int pivot = place_midst(array, istart, iend);
            if (pivot==k-1)
                  return pivot;
            else if (pivot>k-1)
                  return find_kth_element(array, istart, pivot-1, k);
            else
                  return find_kth_element(array, pivot+1, iend, k);
      }
      else if (istart==k-1)
            return istart;
      else // if (iend==k-1)
            return iend;
}
```