1111 NTOUCSE 程式設計 1C 期中考參考答案

111/11/01 (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/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. [10] 下面的程式由鍵盤讀入多個(大於等於 3)小於 2^{31} 且兩兩不等的正整數，想要把第 3 大的數字列印出來，上課時講過要找到最大的數字的話，需要有一個變數來紀錄當一個數字一個數字讀進程式時，所看過資料中最大的數字，那麼要找到第 3 大的數字，是不是該用 3 個變數來紀錄到當下為止最大值 (max[0])、第 2 大(max[1])、以及第 3 大(max[2])的數值呢? 請完成下列的程式 (在答案卷上請標示程式第幾列作答，如果寫在題目卷上請在答案卷上註明)

```
01 #include <stdio.h>
02
03 int main()
04 {
05      int x, max[3]={-1,-1,-1};
06      while (1==scanf("%d", &x))
07      {
08          if (x>max[0])
09          {
10              max[2]=   max[1]   ;
11              max[1]=   max[0]   ;
12              max[0]=x;
13          }
14          else if (x>   max[1]   )
15          {
16              max[2]=max[1]   ;
17              max[1]=    x    ;
18          }
19          else if (x>max[2])
20              max[2]=x;
21      }
22      printf("%d\n", max[2]);
23      return 0;
24 }
```

2. 請閱讀下列程式並回答相關問題

```
01 #include <stdio.h>
02
03 int len_x = 8, total, x[] = {31, 57, 86, 75, 11, 52, 43, 17};
04
05 void sum()
06 {
07      for (int i=0; i<len_x; i++)
08          total += x[i];
09 }
10
```

```
11 int main()
12 {
13        total = 0;
14        sum();
15        printf("Total value is %d\n", total);
16        return 0;
17 }
```

(a) 程式中哪些變數是全域變數?[3] 哪些變數是區域變數? [2]

Sol:

全域變數: len_x, total, x

區域變數: i

(b) [5] 以此程式為例，使用全域變數最大的好處為何?

Sol:

呼叫函式 sum() 的時候不需要提供任何引數，函式內部會把資料陣列 x 裡面的資料加總起來，把總和寫到變數 total 裡面，讓 main() 函式可以列印出總和，撰寫的時候可以節省一些打字的時間，函式呼叫的時候可以節省一點點參數複製的時間，表面上似乎覺得生產力有所提昇。

(c) [5] 以此程式為例，使用全域變數有什麼壞處?

Sol:

程式可以正確執行，但是最大的壞處是隱藏了資料流，從第 14 列 sum()這個函式呼叫敘述裡看不出來函式需要哪些資料作為輸入，計算出什麼結果或是有些什麼影響，一定要去閱讀函式定義本體才能夠知道。此外從程式的複雜度來探討，所有的程式碼藉由全域變數耦合在一起，程式碼之間因此具有密切的關聯性，如此使得程式的正確性不易維持，偵錯、修改、與維護成本指數式地提昇，此種程式除了在競賽程式中「有意識地」使用之外，千萬不要變成你個人的習慣，千萬不要使用在公司的軟體產品中，或是使用在需要和他人合作開發的場合中，就算個人的生產力有所提昇，但是全體的生產力一定反比於個人的所得、指數式地向下沉淪。

(d) [5] 請以函式參數及區域變數修改此程式以避免使用全域變數?

Sol:

```
01 #include <stdio.h>
02
03 int sum(int x[], int len)
04 {
05        int total=0;
06        for (int i=0; i<len; i++)
07              total += x[i];
08        return total;
09 }
10
11 int main()
12 {
13        int len_x = 8, total, x[] = {31, 57, 86, 75, 11, 52, 43, 17};
14        total = sum(x, len_x);
15        printf("Total value is %d\n", total);
16        return 0;
17 }
```

3. (a) [10]請完成下面這個計算陣列中負數和奇數個數的程式：(在答案卷上請標示程式第幾列作答，如果寫在題目卷上請在答案卷上註明)

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

```
02
03 int calculate(____int []____, int,____int *____);
04
05 int main()
06 {
07        int x[] = {3, -5, 8, 7, -11, 5, 4, -1}, odd, negative;
08        odd = calculate(x, sizeof(x)/____sizeof(int)____, ____&negative____);
09        printf("# of odd numbers is %d\n", odd);
10        printf("# of negative numbers is %d\n", negative);
11        return 0;
12 }
13
14 int calculate(int data[], int len, ____int *____negative)
15 {
16        int i, odd=0;
17        *negative = 0;
18        for (i=0; i<len; i++)
19        {
20              ____*negative____ += ____data[i]<0____;
21              odd += ____data[i]%2!=0____; // 或是 data[i]%2
22        }
23        return odd;
24 }
```

程式執行所列印的結果如下：

```
# of odd numbers=6
# of negative numbers=3
```

(b) [2] 整數的型態在 C/C++ 語法中是 int，請問整數指標(整數變數的記憶體位址)的型態 C/C++ 語法該怎麼寫?

Sol:

int *

(c) [3] 請問第 17 列 *negative=0; 代表將程式中哪一個變數清除為 0?

Sol:

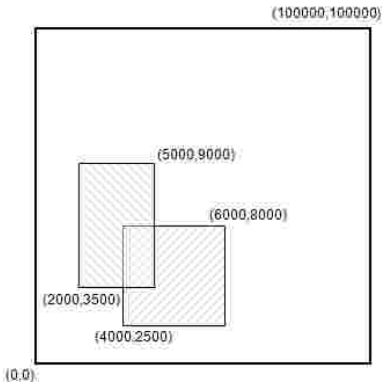
第 7 列 main 函式裡的 negative 變數

4. [10] 在一個 int domino[40000][2]; 的整數陣列變數中有 n 筆資料, 0≤n<40000, 第 i 筆資料包含兩個整數 (x, y) 代表撲克牌的兩張牌, 0≤x,y<52, domino[i][0] 存放的是 x，domino[i][1] 存放的是 y，如果兩筆資料 (x_1,y_1), (x_2,y_2) 中 ($x_1=x_2$ 且 $y_1=y_2$) 或是 ($x_1=y_2$ 且 $y_1=x_2$), 則我們說這兩筆資料是等效的，請完成下列程式計算在原來資料中**任取兩筆資料得到一對等效資料的機率**，下面這個程式裡需要計算 C(52,2)+52 種可能的資料中，每一種重複的次數(len, 23-27 列)，然後計算任取兩筆資料剛好是同一種的數量並且加總(count, 28 列)，除以任取兩筆資料的總數就是機率(41 列)。函式 numEquivDominoPairs(10-31 列) 計算任取兩筆資料得到一對等效資料的數量，為了比較有效率地運用 qsort 排序使得等效的資料接續在一起，首先把每一筆資料正規化(13-19 列)，讓 domino[i][0] ≤ domino[i][1]，提供給 qsort 一個比較兩個整數陣列的函式 comp (04-09 列，其中第 8 列在陣列第 1 個元素相等的情況下比對第 2 個元素)，然後再以一個迴圈 (23-27 列)計算相同資料重複的次數。(在答案卷上請標示程式第幾列作答，如果寫在題目卷上請在答案卷上註明)

```
01 #include <stdio.h>
02 #include <stdlib.h>
```

```
03
04 int comp(const void *a, const void *b)
05 {
06        int *arrayA = (int*) a, *arrayB = (int*) b;
07        int firstElemCompResult = arrayA[0] - arrayB[0];
08        return firstElemCompResult ==0 ?  ____arrayA[1] - arrayB[1]____: firstElemCompResult;
09 }
10 int numEquivDominoPairs(int dominoes[40000][2], int dominoesSize)
11 {
12        int i, tmp, count, len;
13        for (i=0; i<dominoesSize; i++)
14              if (dominoes[i][0]>dominoes[i][1])
15              {
16                    tmp =  ____dominoes[i][0]____;
17                    dominoes[i][0] =____dominoes[i][1]____;
18                    dominoes[i][1] = tmp;
19              }
20        qsort(___dominoes___, ___dominoesSize___, ___sizeof(int[2])___, ___comp___);
21        for (count=i=0; i<dominoesSize; i+=____len____)
22        {
23              len = 1;
24              while (i+len<dominoesSize &&
25                    dominoes[i][0]==dominoes[____i+len____][0] &&
26                    dominoes[i][1]==dominoes[____i+len____][1])
27                    len++;
28              count += len * (len-1) / 2;
29        }
30        return count;
31 }
32
33 int main()
34 {
35        int i, n, dominoes[40000][2];
36
37        scanf("%d", &n);
38        for (i=0; i<n; i++)
39              scanf("%d%d", &dominoes[i][0], &dominoes[i][1]);
40
41        printf("%f\n", numEquivDominoPairs(dominoes,n)*2.0/n/(n-1));
42
43        return 0;
44 }
```

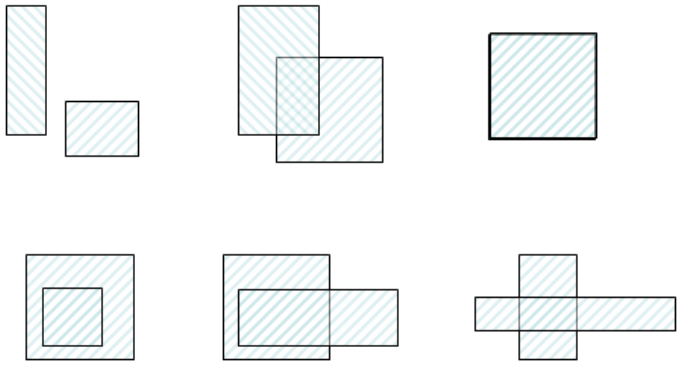
5. 右圖座標平面上 (0,0) 到 (100000,100000) 範圍中有兩個矩形，矩形的左下角座標為 (x_1,y_1)、右上角座標為 (x_2,y_2)，一個點 (x,y) 如果滿足 $x_1 \le x < x_2$ 及 $y_1 \le y < y_2$ 則稱此點在矩形內部，請撰寫程式計算兩個矩形內部重疊部份面積以及只有在單一矩形內部的面積，右圖中重疊區域的左下角座標為 (4000,3500)，右上角座標為 (5000,8000)，依照上面的定義，面積為 (4999-3999) * (7999-3499) = 4500000，只在單一矩形內部的面積則為兩個矩形面積個別扣除上述重疊區域後面積的總和，請依照下列步驟分析並撰寫程式：



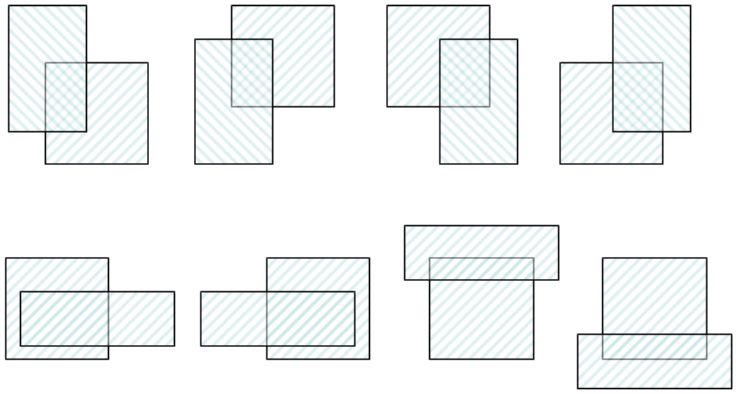
(a) [10] 直接根據上面範例實作程式，似乎問題不大，但是題目並沒有限制兩個矩形的幾何關係只能是這一種啊! (如果把兩個矩形各有一角在另一矩形內都看成是同類的) 也有可能兩個矩形沒有任何重疊 (把所有沒有重疊的兩個矩形都看成同一種)、或是還有其他可能性呢?

請將上圖之外另外五種可能性畫出來?

Sol:



主要是上面這六類，注意上面的矩形都是包括左邊界、下邊界，但是不包括右邊界、上邊界的，其中第二類和第四類還可以細分如下，有可能影響到計算面積的程式（和程式的寫法有關）



(b) [5] 請撰寫一個函式 long long **area**(int rect[4])，計算並且回傳一個矩形的面積，因為座標範圍 (0,0) 到 (100000,100000) 的矩形面積有可能超過 int 型態變數的範圍，所以需要使用 long long 型態，我們用一個四個整數的陣列 rect 來存放一個矩形的 x_1,y_1,x_2,y_2 四個座標值

Sol:

```
long long area(int rect[4])
{
    long long width = rect[2]-rect[0], height = rect[3]-rect[1];
    return width*height;
}
```

請特別注意型態，如果寫 return (rect[2]-rect[0])*(rect[3]-rect[1]); 還是會發生溢位錯誤，因為兩個整數相乘一般情況下就只是 32 位元整數的運算，雖然原本就有可能超過 32 位元，如果系統因此就使用 64 位元來計算，計算完了以後存放到 32 位元的變數時還是會發生溢位，所以系統不會自動幫你用 64 位元執行乘法，除非是 CPU 預設的模式，所以需要自行做型態的轉換 return ((long long)(rect[2]-rect[0]))*((long long)(rect[3]-rect[1]));，如此二元運算子 * 才會是 64 位元的 *，下列程式可以簡單地測試是否需要強迫轉換型態：

```
#include <stdio.h>
int main()
{
    long long x;
    int y = 10000000;
```

```
    x = 10000000*10000000;
    printf("%lld\n", x);        // 276447232
    x = 10000000LL*10000000;
    printf("%lld\n", x);        // 100000000000000
    x = y*y;
    printf("%lld\n", x);        // 276447232
    x = (long long)y*y;
    printf("%lld\n", x);        // 100000000000000
    x = ((long long)y)*y;
    printf("%lld\n", x);        // 100000000000000
    x = (long long)(y*y);
    printf("%lld\n", x);        // 276447232
    return 0;
}
```

(c) [5] 以上圖為例，當程式讀取兩個矩形的四組座標 (x_1,y_1)-(x_2,y_2), (x_3,y_3)-(x_4,y_4) 到陣列變數 int rect[2][4]; 裡面以後 (變數 rect[0][0], rect[0][1], rect[0][2], rect[0][3] 分別存放 x_1,y_1 及 x_2,y_2, 變數 rect[1][0], rect[1][1], rect[1][2], rect[1][3] 分別存放 x_3,y_3 及 x_4,y_4)，主要目標是計算重疊區域的寬 $\min(x_2,x_4) - \max(x_1,x_3)$ 及重疊區域的高 $\min(y_2,y_4) - \max(y_1,y_3)$，如此就可以算出重疊區域的面積，存放到變數 **overlappedArea** 中，再運用兩個矩形的面積就可以計算單一矩形覆蓋部份的面積，存放到變數 **nonOverlappedArea** 中，請撰寫兩個函式 int **min**(int x, int y) 以及 int **max**(int x, int y) 來計算兩個數字 x, y 中較大的數字以及較小的數字，並且呼叫這兩個函式撰寫程式計算上述重疊區域的寬及高，並且寫幾列程式計算出 overlappedArea 以及 nonOverlappedArea

Sol:

```
int min(int x, int y) { return x<y?x:y; }
int max(int x, int y) { return x>y?x:y; }

int i;
long long overlappedArea, nonOverlappedArea, rectArea[2];
for (i=0; i<2; i++) rectArea[i] = area(rect[i]);
int overlappedRect[] = { max(rect[0][0],rect[1][0]), max(rect[0][1],rect[1][1]),
                         min(rect[0][2],rect[1][2]), min(rect[0][3],rect[1][3]) };
overlappedArea = area(overlappedRect);
nonOverlappedArea = rectArea[0] + rectArea[1] - 2*overlappedArea;
```

(d) [5] 設計到這個地方以後，有很多種思考的方式可以繼續下去，例如可以計算一個矩形有幾個頂點在另外一個矩形內，去判斷兩個矩形究竟是題 (a) 六種可能性的哪一種? 然後再依照個別狀況去計算重疊區域面積，如此需要撰寫函式 int ptInRect(int point[2], int rect[4]) 來判斷一個點是否在矩形內部，撰寫函式 int cornersInRect(int rect1[4], int rect2[4])來計算一個矩形的四個頂點有幾個在另一個矩形內，回傳 0, 1, 2, 4 中一個數值，在 main 裡面則呼叫上述函式來計算 count1 = cornersInRect(rect1, rect2); 以及 count2 = cornersInRect(rect2, rect1);，並且運用 count1 及 count2 判別兩個矩形是屬於哪一種可能性。不過請別急，耐心往下看，因為是考試，用上面這樣子寫很直接，沒有寫錯的話一定可以得到答案，但是寫起來程式比較長，會有點辛苦，這裡希望你實作一個比較簡潔的方法來計算重疊部份的面積：首先寫一個函式 int **isOverlapped**(int rect1[4], int rect2[4]) 來判斷兩個矩形是否有交集，(提示：判斷的準則如：第一個矩形的右側邊界座標小於等於第二個矩形的左側邊界座標。。。)，沒有重疊的話，overlappedArea = 0。

Sol:

```
int isOverlapped(int rect1[4], int rect2[4])
```

```
{
    return !(rect1[2]<=rect2[0] || rect1[0]>=rect2[2] || rect1[3]<=rect2[1] || rect1[1]>=rect2[3]);
}
```

(e) [5] 接下來是一個很關鍵的觀察，令測資中兩個矩形的四組座標為 (x_1,y_1) , (x_2,y_2), (x_3,y_3), (x_4,y_4)，不論是題 (a) 兩個矩形有重疊的五種狀況中哪一種，兩個矩形的左側邊界 x_1,x_3 中比較大的就是重疊矩形的左邊界，右側邊界 x_2,x_4 中比較小的就是重疊矩形的右邊界，所以重疊區域的寬還是 $\min(x_2,x_4)-\max(x_1,x_3)$；同樣地兩個矩形的下方邊界 y_1,y_3 中比較大的就是重疊矩形的下方邊界，上方邊界 y_2,y_4 中比較小的就是重疊矩形的上方邊界，所以重疊區域的高還是 $\min(y_2,y_4)-\max(y_1,y_3)$，請運用題 (c), (d) 所完成的函式計算 overlappedArea 和 nonOverlappedArea

Sol:

```
int i;
long long overlappedArea=0, nonOverlappedArea, rectArea[2];
for (i=0; i<2; i++) rectArea[i] = area(rect[i]);
if (isOverlapped(rect[0], rect[1])
{
    int overlappedRect[] = { max(rect[0][0],rect[1][0]), max(rect[0][1],rect[1][1]),
                             min(rect[0][2],rect[1][2]), min(rect[0][3],rect[1][3]) };
    overlappedArea = area(overlappedRect);
}
nonOverlappedArea = rectArea[0] + rectArea[1] - 2*overlappedArea;
```

另外一種寫法是不測試兩個矩形是否有重疊，直接以上式計算重疊矩形 overlappedRect，只有邊長都大於 0 的時候，才需要計算重疊矩形的面積如下：

```
int overlappedRect[] = { max(rect[0][0],rect[1][0]), max(rect[0][1],rect[1][1]),
                         min(rect[0][2],rect[1][2]), min(rect[0][3],rect[1][3]) };
if ((overlappedRect[0]>=overlappedRect[2]) || (overlappedRect[1]>=overlappedRect[3]))
    overlappedArea = area(overlappedRect);
```

6. 請指出並且更正下列程式片段中的語法錯誤或是語法陷阱造成的邏輯/功能錯誤

(a) [6] 下列程式希望能夠在同學得到 80 分以上和 60 分以下兩種情況下印出訊息，但是執行時不是這樣的結果，請問輸入 75 會印出什麼？輸入 50 會看到什麼？請做最小的修改來得到預期的效果？

```
01    int score;
02    scanf("%d", &score);
03    if (score>=60)
04        if (score>=80)
05            printf("Congratulation, you get an A.\n");
06    else
07        printf("I am sorry that you failed.\n");
```

Sol:

輸入 75 時應該會列印 I am sorry that you failed.

輸入 50 時應該沒有列印任何東西

應該可以加上一對左右大括號把第 04 列和第 05 列括起來，表現才會如題目所述，也就是

```
01    int score;
02    scanf("%d", &score);
03    if (score>=60)
04    {
05        if (score>=80)
```

```
06                  printf("Congratulation, you get an A.\n");
07      }
08      else
09              printf("I am sorry that you failed.\n");
```

(b) [3] 下列程式希望由鍵盤輸入任意一個字元並且列印出對應的 ASCII 編碼：

```
01 int ch=-1;
02 scanf("%c", &ch);
03 printf("%c(%d)", ch, ch);
```

```
M:\>ex1-1
a
a(-159)
```

執行時輸入字元 a, 列印出來的結果如上圖，請問為什麼印出來的不是 97? 該如何修改?

Sol:

scanf("%c", &ch); 會把一個字元讀到所傳入的位址處，如果 ch 是一個單位元組 char 型態的變數，那麼資料寫進去以後 printf("%c(%d)", ch, ch); 會印出 a(97)，但是如果 ch 是四位元組 int 型態的變數，&ch 是第一個位元組的記憶體位址，scanf 會把資料寫進這個位元組，另外三個位元組的資料則維持不變，以這個例子來說因為 int ch=-1; 會在 ch 變數裡寫入 0xFFFFFFFF 四個位元組，執行完第 02 列的 scanf 同時如果在鍵盤上輸入字元 a 以後，在 Little Endian 的機器上就是 0xFFFFFF61 (十進位 -159)，在 Big Endian 的機器上就得到 0x61FFFFFF，請注意：看到程式這個表現，你千萬不要得到一個結論說把第一列的 int ch=-1; 改成 int ch=0; 不就沒有問題了嗎? 題目裡面用 -1 (全部 32 位元都是 1) 是在強調說不論 4 個位元組的記憶體裡有什麼東西，其中 3 個位元組的資料都會留存下來。你得到的結論應該是：使用 scanf("%c", &ch); 時變數 ch 一定要是 char 型態。

(c) [3] 下列程式片段想要找出 10 筆資料中的最大值? 但是有的時候會沒有找到最大值? 請問為什麼? 該如何修改?

```
01 int i, max, data;
02 for (i=0; i<10; i++)
03 {
04      scanf("%d", &data);
05      if (data>max)
06          max = data;
07 }
08 printf("Maximum is %d\n", max);
```

Sol:

因為變數 max 沒有給適當的初始值，這裡要求最大值，應該要給一個很小的數字，例如最小的 32 位元整數是 0x80000000 (十進位 -2^{31})，應該寫成 int max=0x80000000; 或是 limits.h 裡面的 INT_MIN 或是 1<<31 或是 -2147483648 或是改成

```
05          if (i==0 || data>max)
```

或是改成

```
02 scanf("%d", &max);
03 for (i=1; i<10; i++) {
```

(d) [3] 請問下列程式片段列印的結果會有什麼異常的現象? 該怎麼修改?

```
01 double x=0.3;
02 for (x=0.3; x != 1.0; x+=0.1)
03      printf("%f\n", x);
```

Sol:

應該要看到右圖的狀況，而且完全不會停止地繼續增加下去，這幾乎是一個

```
0.300000
0.400000
0.500000
0.600000
0.700000
0.800000
0.900000
1.000000
1.100000
1.200000
1.300000
1.400000
1.500000
1.600000
```

無窮迴圈了，想要只印到 1.0 的話，應該改成 for (x=0.3; x<=1.05; x+=0.1)，
或是另外運用整數變數來控制迴圈執行的次數，for (int i=3; i<=10; i++)
主要是因為浮點數比對相等的話，需要精確到 10^{-15} 都完全一樣才會是相等的，
十進位的 0.1 在 2 進位表示法裡面又是一個無法精確表示的數字，所以 x!=1.0
是一個很難滿足的條件，x 的數值到了 1.0 附近時，也許是 1.00000000000001，
也許是 0.999999999999999，要剛好等於 1.0 的機會很低。