

1111 NTOUCSE 程式設計 1C 期末考

姓名：_____ 系級：_____ 學號：_____

111/12/27 (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/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. 請完成下列程式，其中 $N = n_1n_2\dots n_d$ 為一 d 位數十進位正整數，除了最高位數 $n_1 \in \{1, 2, \dots, 9\}$ 之外，每一位數 $n_i \in \{0, 1, \dots, 9\}, i = 2, 3, \dots, d$ ，此程式尋找並列印滿足 $N = n_1^d + n_2^d + \dots + n_d^d$ 的所有數字，例如 d 為 4 時輸出 1634↵ 8208↵ 9474↵

- (a) [5] 由於程式中需要不斷地運用 $n_i^d, n_i \in \{0, 1, \dots, 9\}, d \in \{1, \dots, 9\}$ 數值來檢查，因此請先運用整數二維陣列 `tab[10][10]` 紀錄預先計算的所有數值，請運用**兩層**迴圈計算 n^d 為 `tab[n][d]` 的數值

Sol:

```
for (n=0; n<=9; n++)
    for (tab[n][1]=n, d=2; d<=9; d++)
        tab[n][d] = tab[n][d-1] * n;           因為 n,d 範圍不大，也可以用 pow(n,d)+0.5
```

- (b) [5] 如果 d 是 3，請完成下列 3 層迴圈程式以達成目標 (`tab[n][d]` 為題(a)的結果)

```
int i0, i1, i2, d=3;
unsigned long v1, v2;
for (i0= 1; i0<= 9; i0++)
    for (i1= 0; i1<= 9; i1++)
        for (i2=0; i2<=9; i2++)
        {
            v1 = (i0*10+ i1)* 10 +i2;
            v2 = tab[i0][d]+ tab[i1][d] +tab[i2][d];
            if ( v1 == v2 )
                printf("%d%d%d\n", i0, i1, i2);
        }
```

- (c) [16] 如果 d 是可變的，例如 9 甚至更大，當然不能夠用上面的 d 層迴圈來實現，請完成下列遞迴程式的深度優先搜尋 (DFS)，此程式邏輯等效於層數可變的多層迴圈程式，程式列舉所有十進位 1~9 位數字、檢查並且列印所有符合條件的數字

```
01 #include <stdio.h>
02 unsigned long tab[10][10];
03 void find(int d, int num[], int p) // num[0], ..., num[p-1] 已填好，填入 num[p], ..., num[d-1]
04 {
05     int i;
06     unsigned long v1, v2;
07     if ( p >= d ) // 遞迴的結束條件，num[0],..., num[d-1] 為所列舉數字的每一位數
08     {
09         or p == d
```

```

09         for (v1=i=0; i<d; i++)           Horner's rule
10         v1 = v1*10+num[i] // 計算 num 陣列內代表之數值 N
11         for (v2=i=0; i<d; i++)
12         v2 += tab[num[i]][d] // 計算  $n_1^d + n_2^d + \dots + n_d^d$ 
13         if ( v1 == v2 ) // 符合指定條件
14             for (i=0; i<d; i++) printf("%u\n", v1);
15     }
16     else // 嘗試 1~9 或是 0~9 所有的數字作為第 p 位數
17         for (num[p]= (p == 0) ; num[p] <= 9 ; num[p]++)
18             find(d, num, p+1); // 遞迴呼叫
19     }
20 int main()
21 {
22     int d, num[11]; // num[0], num[1], ..., num[d-1] 為所列表數字的每一位數
23     // 題(a)之兩層迴圈預先計算 tab[n][d]
24     for (d=1; d<=9; d++)
25         find(d, num, 0);
26     return 0;
27 }

```

or (p==0?1:0)

- (d) [6] 實際測試以後題(c)的程式其實沒有什麼效率，需要調整一下，第一個調整是上面第 09~14 列比對 N 以及 $n_1^d + n_2^d + \dots + n_d^d$ 的程式片段，當 $n_1^d + n_2^d + \dots + n_d^d$ 計算出來以後，並不需要先把 N 計算出來，可以一位數一位數去比對 num[0], num[1], ..., num[d-1] 裡面的數字，任何時候只要一個數字比對失敗就可以提前確定這個 d 位數字是不符合的

```

for (v2=i=0; i<d; i++) // 題(c)第 11 列
    // 題(c)第 12 列
for (i=d-1; i>=0; i--, v2 /= 10 )
    if ( v2%10 != num[i]) return;
if ( v2 == 0 ) {           or !v2
    for (i=0; i<d; i++)
        printf("%d", num[i]);
    printf("\n");
}

```

- (e) [16] 題(d)的程式大概比題(c)少 25% 的執行時間，還需要進一步調整，不論是題(c)還是題(d) 把所有小於 999999999 的數字列舉出來，實際上應該不太需要，例如 978xx 這 100 個數字其實只需要列舉出 97800 以後就可以確定 $9^5 + 7^5 + 8^5 = 108624 > 978xx$ ，所以不需要列舉其它 99 個 978xx 的數字，直接測試 979xx，這樣子能夠在一發現不符合的時候就把深度優先的搜尋截斷以加快速度，請完成下列程式，實際測試這樣的作法大概可以比題(d)少 85% 的執行時間

```

01 #include <stdio.h>
02
03 unsigned long tab[10][10];
04 const int tens[]={1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000};
05
06 void find(int d, int num[], unsigned long v1, unsigned long v2, int p)
07 {
08     unsigned long v1n, v2n;
09     if (p<d)
10         for (num[p]= (p == 0) ; num[p] <= 9 ; num[p]++) // 題(c)第 17 列
11         {

```

or (p==0?1:0)

```

12         v1n = v1*10 + num[p]; // 計算 num[0], ..., num[p] 代表的整數
13         v2n = v2 + tab[num[p]][d]; // 計算  $n_1^d + n_2^d + \dots + n_p^d$ 
14         if (v2n >= (v1n+1)*tens[d-p-1]) continue; // 提前結束遞迴
15         find(d, num, v1n, v2n, p+1); // 遞迴呼叫
16     }
17     else if (v1==v2)
18         printf("%u\n", v1);
19 }
20
21 int main()
22 {
23     int d, num[11]; // num[0], num[1], ..., num[d-1]為所列舉數字的每一位數
24     // 題(a)之兩層迴圈預先計算 tab[n][d]
25     for (d=1; d<=9; d++)
26         find(d, num, 0, 0, 0);
27     return 0;
28

```

backtracking 節省的運算時間非常顯著

2. 下面程式根據有限的提示來尋找開啟寶藏可能的密碼，程式輸入 $n, r, 3 \leq n \leq 30, 0 \leq r \leq 9$ ，兩個整數以及一個 $n-1$ 位數的十進位數字 d ，程式輸出所有可能的不重複密碼。例如：

輸入：

4 5

138

輸出：

1238

1328

1382

2138

首先定義一個十進位整數的根如下：把這個整數每位數字加總起來，得到的整數位數變少，反覆操作，直到變成一位數字的整數，即為根，例如 $n=4$ ，數字 $d=138$ 第一次加總會變成 $1+3+8=12$ ，第二次加總再變成 $1+2=3$ ，3 稱為數字 138 的根，把這個十進位數字增加一位數 x ，例如 $x=2$ 可以得到 **2138**, **1238**, **1328**，或是 **1382**，使得此數字的根為指定的數字 $r=5$ ，當然增加其他數字 x 也可能符合要求，此時請由小到大將所有可能的密碼列印出來。

- (a) [8] 由於輸入的十進位數字可能有 29 位數，所以定義字元陣列 $d[31]$ 來存放輸入，由於增加一位數最多可以有 $10*30$ 種可能性，所以定義一個 $ans[300][32]$ 的二維字元陣列來存放所有符合的密碼數字串，第 12~20 列請計算每一個可能字串的根
- (b) [10] 第 21~28 列請在符合指定要求的根 r 時在 ans 陣列中新增 n 個密碼字串
- (c) [8] 上述字串需要由小到大排序，第 30 列請運用 `string.h` 裡面的 `strcmp()` 函式以及 `stdlib.h` 裡面的 `qsort` 函式排序
- (d) [6] 由於這些字串中可能有重複的字串，所以第 31~34 列除了列印第一個密碼之外，只有在與前一個字串不同時才列印

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

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

```

03 #include <stdlib.h>
04
05 int main()
06 {
07     int i, j, n, r, rp, x, sum, psum, na;
08     char ans[300][32], d[31];
09     while (3==scanf("%d%d%s", &n, &r, d))
10     {
11         na = 0; // 可能的密碼數字組
12         for (sum=i=0; i<n-1; i++)
13             sum += d[i]-'0'; // 計算 d 陣列的數字和 sum
14         for (x=0; x<10; x++) // 可能加入的一位數字 x
15         {
16             rp = sum+x;
17             do
18                 for (psum=rp, rp=0; psum > 0; psum/= 10)
19                     rp += psum%10;
20             while (rp>10); // 迴圈結束時 rp 為新增一位數字的根
21             if (rp==r) // 滿足指定的要求
22                 for (i=0; i<n; i++) // 共增加 n 組密碼數字, 放在字串 ans[na] 中
23                 {
24                     ans[na][i] = x+'0'; // 第 i 個字元為新增的字元
25                     for (j=0; j<n-1; j++) // 複製 d[] 陣列中其它 n-1 個字元
26                         ans[na][j+(j>=i)] = d[j];
27                     ans[na++][n] = 0; // 字串結束字元
28                 }
29             }
30             qsort(ans, na, sizeof(ans[0]), (int (*)(const void*, const void*))strcmp);
31             printf("%s\n", ans[0]);
32             for (i=1; i<na; i++)
33                 if (strcmp(ans[i], ans[i-1]) != 0)
34                     printf("%s\n", ans[i]);
35         }
36     return 0;
37 }

```

ans[1],...ans[i], or char[32]

3. 請回答下列程式相關問題

- (a) [5] 寫了一個學期的程式，你一定曉得如果資料沒有正確地讀入記憶體中，辛苦設計的程式根本沒有機會表現，下列程式片段加入 #include 敘述以及 main() 函式，可以順利編譯，但是執行起來輸入 D3 時，都沒有列印出任何結果就當掉了，請問為什麼有這樣的表現？該如何修改？

```

char c; int n;
scanf("%1[A-Z]%d", c, &n);
printf("%c%d\n", c, n);

```

Sol:

沒有任何結果就直接當掉時，最常見的原因就是記憶體存取的錯誤，在上面這一段程式中 `scanf` 的格式轉換指令 `%l[A-Z]` 要求把資料寫入字元陣列中，因此程式需要改成 `char c[2];` 以及 `printf("%c%d\n", c[0]` 才不會出現記憶體存取的錯誤，如果只是改成 `scanf("%l[A-Z]%d", &c, &n);` 還是一樣會產生記憶體存取的錯誤，格式轉換指令 `%l[A-Z]` 和 `%s` 一樣都要求把資料寫入字元陣列，`%c` 則要求寫入一個字元變數，將 `scanf` 敘述改為 `scanf("%c%d", &c` 雖然可以避免錯誤，但是 `scanf` 並不會檢查字元在 A-Z 之間，邏輯上並不相同。

- (b) [5] 下列程式編譯的時候發生錯誤，錯誤訊息是 `error: cannot convert 'int (*)[3]' to 'int**' for argument '3' to 'int sumMatrix(int, int, int**)'`，請問這個訊息的意義是什麼？該如何修改程式呢？

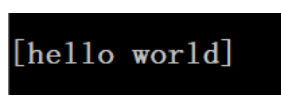
```
#include <stdio.h>
int sumMatrix(int n, int m, int **data) {
    int i, j, sum=0;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            sum += data[i][j];
    return sum;
}
int main() {
    int A[][3]={{1,2,3},{4,5,6}};
    printf("%d\n", sumMatrix(2, 3, A));
}
```

Sol:

在 `main()` 中宣告 `A` 為單位元素為 3 個整數陣列的陣列，因此 `A` 的數值是陣列的第一個元素的記憶體位址，型態則為 `int (*)[3]`，因此函式參數需要寫成 `int (*data)[3]` 或是 `int[][3]`，並不是 `int**`，如果呼叫的時候用強制型態轉換 `sumMatrix(2,3,(int**)A)`，雖然可以避免編譯錯誤，但是執行的時候 `data[i][j]` 會產生記憶體存取的錯誤。

- (c) [5] C/C++ 的字串是用字元陣列來存放的，請說明下列這個程式執行起來為什麼會視窗裡看到右圖的結果？請做最少的修改讓列印的結果是 `[hell]`？

```
char buf[]={'h', 'e', 'l', 'l'};
int j=0x00646c72, k=0x6f77206f;
printf("[%s]\n", buf);
```


Sol:

猜測 `printf` 原來想要列印的是 `[hell]\n`，但是 `buf` 字元陣列缺了字串的結束字元 `'\0'`，所以 `printf` 只能在記憶體裡一路往下印，直到遇見結束字元 `'\0'` (也就是位元組 `0x00`) 為止，剛好記憶體裡接下來的位元組是 `0x6f, 0x20, 0x77, 0x6f, 0x72, 0x6c, 0x64, 0x00`，也就是 `"o world\0"` 的 ASCII 編碼，所以列印出有點不可思議的 `[hello world]`，至於為什麼 `k` 和 `j` 這兩個整數變數剛好安排在 `buf` 陣列後面和編譯器內部運作有關，不是語法指定的範疇。想要恢復成正常的表現的話只要加上字串的結束字元 `'\0'` 就可以了，也就是 `char buf[]={'h', 'e', 'l', 'l', '\0'};`。

- (d) [5] 軟體業的技術人員需要和同事一起合作編碼，有一天你看到下面一段同事寫的程式，同事告訴你這段 C 語言的程式可以編譯也可以執行，你需要了解這段程式才有辦法完成你自己的工作，你該怎麼辦？

```
void util(int x[]) {
    for (int i=2; i<n-m; i++)
        x[i] = 2*x[i+m]-x[i-2];
}
```

- i). 假設 $n=10, m=5$ 來了解這段程式
- ii). 跟他爭執說這一段程式沒有辦法編譯
- iii). 把 m, n 改為區域變數並且設定資料
- iv). 他一定用了全域的變數，請他修改成參數以避免後續不可避免的錯誤

Sol:

iv，在多人合作的環境中使用全域變數是一個絕對的災難，沒有特別好的理由的話，一律是需要移除的，如果你有習慣使用全域變數，請在有意識的情況下使用(例如競賽程式)，否則對你也會是一個災難，在軟體專案中遇見一個會不自覺使用全域變數的同事基本上就是一個不定時炸彈，很難共事，而且這個炸彈還會很得意地展現他所提供的很多簡潔作法喔。基本上全域變數可以將許多毫無關聯的程式碼串起來，使得程式的複雜度指數式地提高，所以在許多年來軟體工程的實作裡，全域變數是第一個需要從軟體專案裡移除的惡魔，經驗足夠的專案管理人都不會犧牲自己以及整個群體來遷就你的惡習。