

# 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

- Exam rules :
1. **Close book, close** everything including quizzes, homeworks, assignments, reference materials, etc.
  2. You can answer the questions in **English** or in **Chinese**, in this **problem sheet**, the **answer sheet**, or **both**.
  3. You can use language features not taught in class if you feel necessary, but strictly limited in C or C++.
  4. **No mobile phone, pad, computer** or **calculator** is allowed. (Electronic) English dictionary is OK
  4. No peeping around! No discussion! No exchange of any material; **raise your hand if you have any question about the exam problems**
  5. If you turn in the paper earlier than the specified time, **leave the classroom immediately and quietly**
  6. Against any of the above rules will be treated as cheating in the exam and handled by school regulations.
  7. **Turn in BOTH the problem sheet with your name and id and answer sheet with your name and id.**

1. Please complete the following program, in which  $N = n_1n_2\dots n_d$  is a  $d$ -digit positive decimal integer. Except the range of the most significant digit  $n_1$  is  $\{1, \dots, 9\}$ , the range of all other digits  $n_i$  is  $\{0, 1, \dots, 9\}$ ,  $i = 2, 3, \dots, d$ . This program searches and prints all integers satisfying the condition  $N = n_1^d + n_2^d + \dots + n_d^d$ . For example if  $d$  is 4, the output is 1634↵ 8208↵ 9474↵

(a) [5] Because the program needs to calculate  $n_i^d$ ,  $n_i \in \{0, 1, \dots, 9\}$ ,  $d \in \{1, \dots, 9\}$  many times, in order to save some computation time, please use a 2 dimensional integer array `tab[10][10]` to save all pre-calculated values. Please use a two-layer loop structure to calculate  $n^d$  and save to `tab[n][d]`.

(b) [5] if the value of  $d$  is 3, please complete the following 3-layer loop to achieve the target (`tab[n][d]` is the result of part (a))

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

(c) [16] If the value of  $d$  is variable, for example 9 or even more, we should not use the  $d$ -layer for loop of part (b) to implement the solution. Instead, we should use a depth first search (DFS) algorithm implemented with a recursive function call. Please complete the following program (the logic of this program is equivalent to a multi-layer for loop program, yet the number of layers is variable), this program enumerates all positive 1-digit to 9-digit decimal integers, check, and print all numbers satisfying the condition  $N = n_1^d + n_2^d + \dots + n_d^d$ .

```
01 #include <stdio.h>
02 unsigned long tab[10][10];
03 void find(int d, int num[], int p) // num[0], ..., num[p-1] filled, num[p], ..., num[d-1] to fill
04 {
```

```

05     int i;
06     unsigned long v1, v2;
07     if (_____) // terminating condition, num[0],..., num[d-1] are digits enumerated
08     {
09         for (v1=i=0; i<d; i++)
10             _____ // calculate the integer N saved in array num[]
11         for (v2=i=0; i<d; i++)
12             _____ // calculate  $n_1^d + n_2^d + \dots + n_d^d$ 
13         if (_____) // satisfying the condition
14             printf("%u\n", v1);
15     }
16     else // try 1~9 or 0~9 as the p-th digit
17         for (num[p]=_____; _____; num[p]++)
18             find(d, num, _____); // recursive call
19 }
20 int main()
21 {
22     int d, num[11]; // num[0], num[1], ..., num[d-1] specified each digits of the integer
23     // the two layer for loop to pre-calculate tab[n][d] of part (a)
24     for (d=1; d<=9; d++)
25         find(d, num, _____);
26     return 0;
27 }

```

- (d) [6] After implementation and tests, the program in part (c) is found to be inefficient and requires some adjustment. The first adjustment is for lines 09~14 that compare the number  $N$  and calculated  $n_1^d + n_2^d + \dots + n_d^d$ : when  $n_1^d + n_2^d + \dots + n_d^d$  is calculated, it is not necessary to evaluate the positive integer  $N$  from the array num[]. It is possible to calculate each digit gradually from the value  $n_1^d + n_2^d + \dots + n_d^d$  and compared with digits stored in num[0], num[1], ..., num[d-1] one by one.

Whenever a comparison fails, this  $d$ -digit number can be dropped immediately

```

    for (v2=i=0; i<d; i++) // line 11 of part (c)
        // line 12 of part (c)
    for (i=d-1; i>=0; i--, _____)
        if (_____ != num[i]) return;
    if (_____) {
        for (i=0; i<d; i++)
            printf("%d", num[i]);
        printf("\n");
    }

```

- (e) [16] The execution time of part (d) is about 25% shorter than the code in part (c). It requires additional adjustment. No matter the methods of part (c) or part (d) enumerate all positive integers no larger than 999999999 and check their validity. Actually this might not be necessary, for example, the 100 integers of the form 978xx need not be enumerated after 97800 is tested since  $9^5 + 7^5 + 8^5 = 108624 > 978xx$ . Thus, it is not necessary to enumerate the other 99 integers of the form 978xx and test 979xx directly. In this way, we can trim the decision branch in depth first search to accelerate the program as soon as we find that the branch is not possible. Please complete the following program. Indeed the method of the following code could save up to 85% of computation

time.

```

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]=_____ ; _____ ; num[p]++) // line 17 of part(c)
11             {
12                 v1n = v1*10 + _____ ; // calculate  $N$  from num[0], ..., num[p]
13                 v2n = v2 + _____ ; // calculate  $n_1^d + n_2^d + \dots + n_p^d$ 
14                 if (_____ >= (_____) * tens[d-p-1]) continue; // terminate prematurely
15                 find(d, num, _____, _____, _____); // recursive call
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] specified each digits of the integer
24     // the two layer for loop to pre-calculate tab[n][d] of part (a)
25     for (d=1; d<=9; d++)
26         find(d, num, _____, _____, _____);
27     return 0;
28

```

2. The following program finds possible passwords to a treasure based on limited hints. The program input is two integers  $n, r$ ,  $3 \leq n \leq 30, 0 \leq r \leq 9$ , and one  $(n-1)$ -digit decimal integer  $d$ , the outputs are all non-repeating possible passwords in alphabetical ascending order. For example,

**input:**

4 5

138

**output:**

1238

1328

1382

2138

first we define the **root** of a positive decimal integer  $d$  as: sum up each digits of this integer and result to an integer with fewer digits, the above step is repeated until the result is a single digit number which we call it the root of  $d$ . For example,  $n=4$ , the decimal integer  $d=138$ . The first summation of its digits is  $1+3+8=12$ , the second summation is  $1+2=3$ , and 3 is the root of 138. Now the hint of the treasure add another digit  $x$  to  $d$  such that the number has  $n$  digits. For example  $x=2$ , we obtain four new integers **2**138, **12**38, **132**8, or **1382** such that the root of these new integers are the specified root  $r=5$ . Note that

there can be more legal digits  $x$  that satisfying the requirement. At last the password integers are printed out in ascending lexicographical order.

- (a) [8] Since the input decimal integer could have at most 29 digits, we define a character array `d[31]` to save the input string. Consider all possible digits  $x$  and one more digit password strings, we define a two dimensional character array `ans[300][32]` to save at most  $10*30$  possibilities of password strings. Please calculate the root of a possible digit string in line 12~20
- (b) [10] Please add  $n$  possible password strings to array `ans[300][32]` in line 21~28 whenever a digit  $x$  makes the root of the extended string the specified root  $r$
- (c) [8] The output contains all strings in ascending order (numerically or alphabetically). Please use `strcmp()` function in `string.h` and `qsort()` function in `stdlib.h` to sort the outputs in line 30
- (d) [6] To avoid outputting the same password, please complete the codes in line 31~34 to output the first password and all other passwords whenever they differ from their immediate preceding element

```
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; // all possible password strings
12         for (sum=i=0; i<n-1; i++)
13             _____; // calculate the sum of numbers in array d
14         for (x=0; x<10; x++) // for each possible digit x
15         {
16             rp = sum+x;
17             do
18                 for (psum=rp, rp=0; _____>0; psum/=_____)
19                     rp += _____;
20             while (rp>10); // rp is the root of an extended password string after the loop
21             if (_____) // satisfying the required r
22                 for (i=0; i<n; i++) // n passwords are added to string ans[na]
23                 {
24                     ans[na][_____] = _____; // the i-th character is the new one
25                     for (j=0; j<n-1; j++) // copy the remaining n-1 characters in array d[]
26                         ans[na][j+_____] = d[j];
27                     ans[na++][_____] = 0; // terminating characters for a C string
28                 }
29         }
```

```

30      qsort(ans, _____, sizeof(_____), (_____)strcmp);
31      printf("%s\n", ans[0]);
32      for (i=1; i<na; i++)
33          if (strcmp(ans[_____], ans[_____])_____)
34              printf("%s\n", ans[i]);
35      }
36      return 0;
37 }

```

3. Please answer the following programming related problems:

- (a) [5] After learning programming for a semester, you must know that if the input data were not read into variables correctly, your design has no chance to show its capability. The following piece of program with suitable #include preprocessor command and the main() function can be successfully compiled. However, as we input from keyboard “D3”, the program aborts without any output. Please explain the reason for this performance? How do you make the correction?

```

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

```

- (b) [5] The following program has compilation errors. The error message is error: cannot convert 'int (\*)[3]' to 'int\*\*' for argument '3' to 'int sumMatrix(int, int, int\*\*)'. Please explain the meaning of the message? How can you correct the program?

```

#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));
}

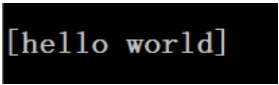
```

- (c) [5] Strings of C/C++ are stored in character arrays. Please explain why the following program can have the output shown in the lower-right figure. Please make the minimal modification to change the output string as [hell]?

```

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

```



- (d) [5] A software practitioner need to cooperate with his associates. One day you are given the

following piece of program. The author tells you the this piece of C program can be compiled and executed correctly. Now you need to understand this piece of program in order to finish your job, what should you do?

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

- i). Assume that  $n=10$  and  $m=5$  to understand the program
- ii). Argue with him that this piece of program cannot be compiled successfully
- iii). Change  $m, n$  to local variable and assign values accordingly.
- iv). He must use global variables. Ask him to modify the program and use function parameters to specify the value  $n$  and  $m$  such that possible future errors can be avoided.