Introduction to Computers and Programming LAB-Quiz2 2014/12/24

Time: 2.5 hrs

※Please create a new folder. Name the folder as: Student ID-Name (XXXXXXX-○○○).

※No Internet. No discussions.

※The class is for <u>C language</u>, so do not use C++.

※If any of your program cannot be compiled, you will get zero score for the question.

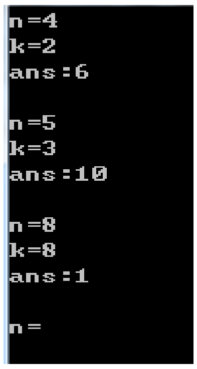
※Before you use any variables, make sure you have assigned values to them. Some IDE's, such as Dev-C++, may not automatically initialize the variables.

※Your programs will be checked (by a tool) for the programming integrity. Be honest with your own works.

※Your output must comply with the sample output format.

1. (10pts) Write a program to calculate the number of *k*-combinations from a set S of n elements.

$$C_k^n = \binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots1} = \frac{n!}{k!\,(n-k)!}$$



```
n=4
k=2
ans:6

n=5
k=3
ans:10

n=8
k=8
ans:1

n=
```

2. (15pts) Turns in city road

You will be given a city map as a 100 by 100 integer array. The elements in the array will be 1 or 0. Each element has four neighbors, except the element on the corner (3 neighbors) or on the side (2 neighbors).

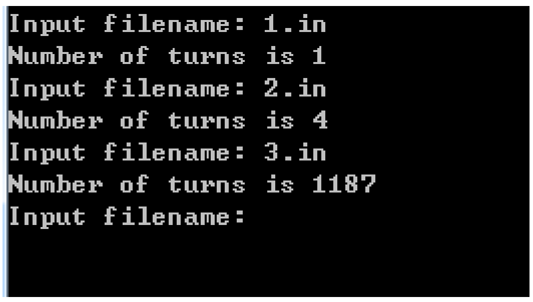
The element is a turn if

1. **This element is 1, not 0.**
2. **It has exactly two neighbors are 1, other neighbors are 0.**
3. **This element and its two neighbors do not form a straight line (horizontal or vertical).**

$$\begin{array}{c|cccc} x^y & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & 1 & 1 & 1 & 0 \end{array}$$

In right graph example(4*4 map array), blue ones are turns. But map[0][2], map[2][0] and map[3][1] aren't turns because they and their 2 neighbors form a straight line.

In quiz2_2.c, the storage of map[100][100] was already prepared, and you should ONLY modify the *count_turn* function to get the answer.

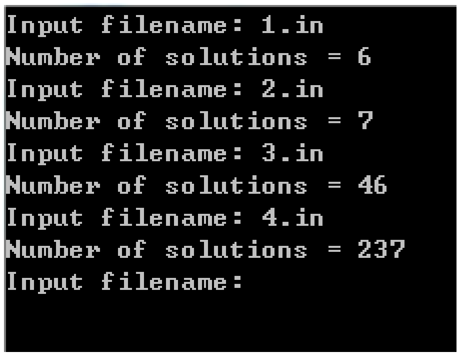


```
Input filename: 1.in
Number of turns is 1
Input filename: 2.in
Number of turns is 4
Input filename: 3.in
Number of turns is 1187
Input filename:
```

3. (15pts) Number selection (recursive problem)

Write a program to count the number of ways to pick at least k number out of N positive numbers, so that the sum of the picked numbers is no more than M. For example, given four numbers 1, 3, 5, and 7. There are 6 ways to pick at least 2 numbers, so their sum is no more than 10. The solutions are {1, 3}, {1, 5}, {1, 7}, {3, 5}, {3, 7}, and {1, 3, 5}. There are totally 6 solutions, so output 6.

In quiz2_3.c, the storage of k, N, M and the array arr[20] were already prepared for you.
Note: 1 <= k <= N, 1 <= N <= 20, 1<=M<=5000000

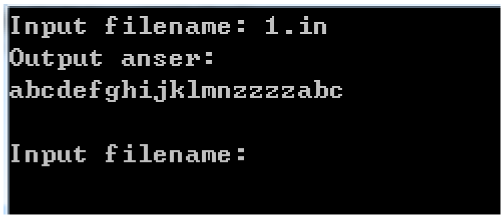


```
Input filename: 1.in
Number of solutions = 6
Input filename: 2.in
Number of solutions = 7
Input filename: 3.in
Number of solutions = 46
Input filename: 4.in
Number of solutions = 237
Input filename:
```

4. (15pts) Fusion

Write a program to do string “fusion”. The fusion is similar to string concatenation, but it will combine the suffix of the first string and the prefix of the second string if they are the same. For example, if you fuse “abcd” with “cdef”, then you will get “abcdef” because “cd” is repeated, but if you fuse “abc” with “def”, then you will get “abcdef”. Also the fusion will try to fuse as many characters as possible. For example, if you fuse “abccc” with “cccde” then you will get “abcccde”, not “abccccde”, nor “abcccccde”.

In quiz2_4_sample.c, num and arr[100][100] were already prepared, and the final answer will not be over 10000 characters.



```
Input filename: 1.in
Output anser:
abcdefghijklmnzzzzabc

Input filename:
```

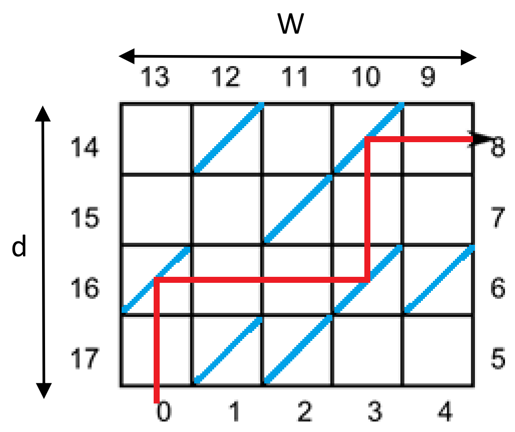
5. (15pts) House mirror

We divide a room into *d* by *w* spaces (i.e. denoted by a *d*×*w* array), where you can place a mirror or not. For example, in file 1.in, a room is divided into 4×5 spaces, where 1 indices a mirror, and 0 indicates no mirror.

File 1.in:

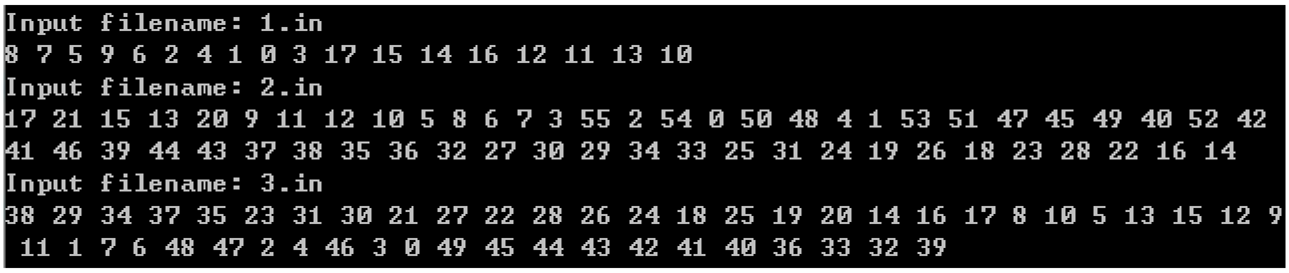
```
4 5
0 1 0 1 0
0 0 1 0 0
1 0 0 1 1
0 1 1 0 0
```

All the mirrors from 1.in are arranged as in the following figure.



We install the windows along the perimeter of the room, for example, there are 18 windows in 1.in, denoted by 0 to 17. In general, we label each window from 0 to 2(*w*+*d*) -1. Based on the arrangement of the mirrors in file 1.in, when we look into **window number 0**, we will see the view of **window number 8**, illustrated by the red line in figure above. Now given all the mirror positions, please compute the view for **each window** (0…2(*w*+*d*) -1), so when input is 1.in, you should output the view from 0 to 17 window.

Note: 1<=w<=100, 1<=d<=100



```
Input filename: 1.in
8 7 5 9 6 2 4 1 0 3 17 15 14 16 12 11 13 10
Input filename: 2.in
17 21 15 13 20 9 11 12 10 5 8 6 7 3 55 2 54 0 50 48 4 1 53 51 47 45 49 40 52 42
41 46 39 44 43 37 38 35 36 32 27 30 29 34 33 25 31 24 19 26 18 23 28 22 16 14
Input filename: 3.in
38 29 34 37 35 23 31 30 21 27 22 28 26 24 18 25 19 20 14 16 17 8 10 5 13 15 12 9
 11 1 7 6 48 47 2 4 46 3 0 49 45 44 43 42 41 40 36 33 32 39
```