Assignment 1

20201601 예지훈

Problem 1

1. Requirement:

The problem 1 is about addition of the roman numbers. First the number of test case is given as input. For each test cases, there are two roman numbers which are 1000 or less. The program need to show the result of addition of two numbers for each test case. Also it have to show the same result in decimal arabic number system as output. There is a picture of the input and output example of the program below. The left box is input and the right box is output for the input test cases.

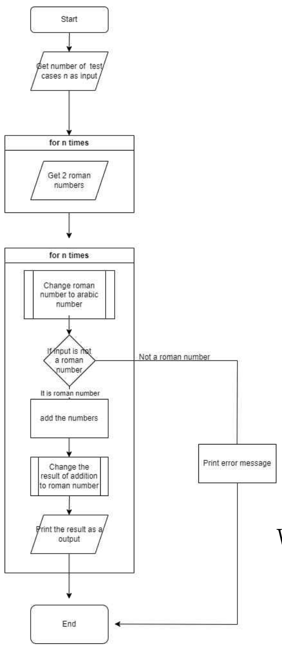
2 CMLXXX XIX CLVI XLIII	CMLXXX+ XIX=IM 980+ 19=999 CLVI+ XLIII=CIC 156+ 43=199

2. Analysis:

In this stage of system life cycle, the problems must be broken down into manageable pieces. The problem 1 could be broken into 3 small pieces with bottom up method. To change input roman number string into arabic decimal number, to add the numbers, and to change arabic number into roman number string.

3. Design:

The program is designed like this simple flowchart.

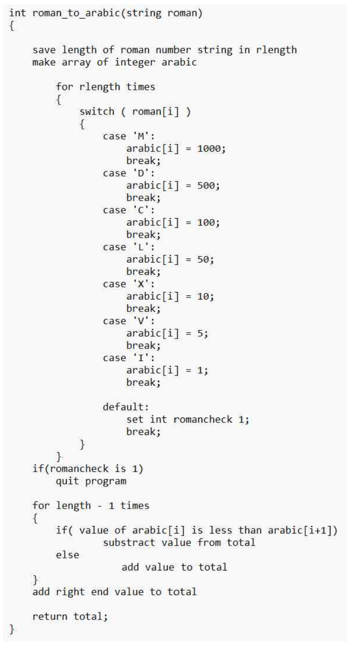


Start
Get number of test cases n as input
for n times
Get 2 roman numbers
for n times
Change roman number to arabic number
If input is not a roman number
Not a roman number
It is roman number
add the numbers
Change the result of addition to roman number
Print the result as a output
Print error message
End

When the program starts, it gets number of the test cases it will process as input. For each test case, the program get 2 roman numbers

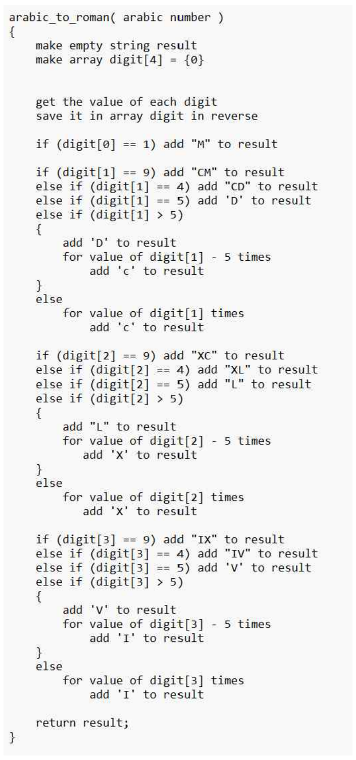
string. After it gets roman numbers for every test case, the program starts changing the roman number into arabic number one case at a time. But if test case include wrong alphabet or not a roman number, the program prints error message and quit the program. Otherwise it add up the test case which is changed into arabic number and save the result. Then program change the result of addition into roman number. Finally it prints out the result of the addition in both roman number and arabic number. The program does the same for remaining test cases.
To change the roman number into arabic number, the program used simple function roman_to_arabic(). The function reads roman number string one character by one character. It uses switch to get the value for each characters. Then it checked if the character right after the target character is bigger or smaller than the target character. If the target character is bigger or equal, we can just add the value of the charcter to total value. Otherwise the value of the character have to be substracted from the total value. The last character (in right end) don't have to be checked and should be added to total value.
However, to change the arabic number into roman number, the program get the number for each digit. For each digit the program use many conditional statements to match the right roman number characters. For example it match "CD" if the third digit of the number is number 4.
I didn't consider about the case that input roman number dose not follow official rule of roman number system because in the problem it said "Two roman numbers are given" so i assume wrong roman numbers would not be accepted as roman numbers and would not be given.

4. Refinement and coding:



```
int roman_to_arabic(string roman)
{

    save length of roman number string in rlength
    make array of integer arabic

        for rlength times
        {
            switch ( roman[i] )
            {
                case 'M':
                    arabic[i] = 1000;
                    break;
                case 'D':
                    arabic[i] = 500;
                    break;
                case 'C':
                    arabic[i] = 100;
                    break;
                case 'L':
                    arabic[i] = 50;
                    break;
                case 'X':
                    arabic[i] = 10;
                    break;
                case 'V':
                    arabic[i] = 5;
                    break;
                case 'I':
                    arabic[i] = 1;
                    break;

                default:
                    set int romancheck 1;
                    break;
            }
        }
    if(romancheck is 1)
        quit program

    for length - 1 times
    {
        if( value of arabic[i] is less than arabic[i+1])
                substract value from total
        else
                   add value to total
    }
    add right end value to total

    return total;
}
```

Above is pseudocode of function used for changing roman number into arabic number. The program use data type string to save each roman numbers. In the function roman_to_arabic() it make array that same length with roman number string to store value for each roman number alphabets. Used case statement to get numeric value of each alphabet and add up or substract the values depending comparision of its right-next value.



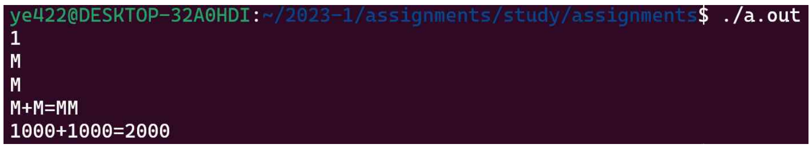
```
arabic_to_roman( arabic number )
{
    make empty string result
    make array digit[4] = {0}


    get the value of each digit
    save it in array digit in reverse

    if (digit[0] == 1) add "M" to result

    if (digit[1] == 9) add "CM" to result
    else if (digit[1] == 4) add "CD" to result
    else if (digit[1] == 5) add 'D' to result
    else if (digit[1] > 5)
    {
        add 'D' to result
        for value of digit[1] - 5 times
            add 'c' to result
    }
    else
        for value of digit[1] times
            add 'c' to result

    if (digit[2] == 9) add "XC" to result
    else if (digit[2] == 4) add "XL" to result
    else if (digit[2] == 5) add "L" to result
    else if (digit[2] > 5)
    {
        add "L" to result
        for value of digit[2] - 5 times
           add 'X' to result
    }
    else
        for value of digit[2] times
           add 'X' to result

    if (digit[3] == 9) add "IX" to result
    else if (digit[3] == 4) add "IV" to result
    else if (digit[3] == 5) add 'V' to result
    else if (digit[3] > 5)
    {
        add 'V' to result
        for value of digit[3] - 5 times
            add 'I' to result
    }
    else
        for value of digit[3] times
            add 'I' to result

    return result;
}
```

The code above is pseudocode about function that changes arabic number to roman number. It made empty string to store roman number alphabets and length 4 array initialized by 0. That is for storing each number of digits. In this problem maximum input is 1000, so 4 length array is enough. In the function, it gets the value for each digits by continuously getting modulo of 10 and dividing the number by 10. It stores each digit in array in reverse to unite the notation. For each digits the program use 5 conditional statement to allocate the roman alphabet. It returns result string when the work is finished.

5. Verification:

The program does not used particular set of algorithm to get the output. Thus tested the program with 3 test cases to verify the program.

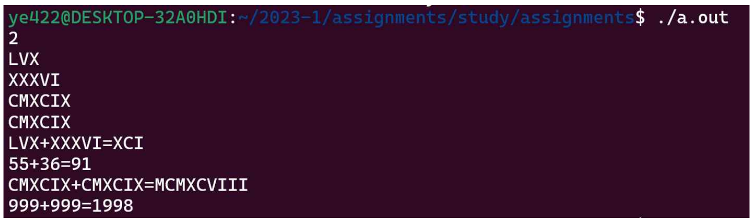
(1) Case with maximum value



```
ye422@DESKTOP-32A0HDI:~/2023-1/assignments/study/assignments$ ./a.out
1
M
M
M+M=MM
1000+1000=2000
```

The program shows right answer when the maximum value is given.

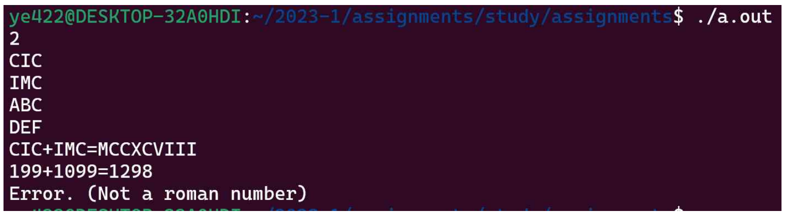
(2) Normal cases



```
ye422@DESKTOP-32A0HDI:~/2023-1/assignments/study/assignments$ ./a.out
2
LVX
XXXVI
CMXCIX
CMXCIX
LVX+XXXVI=XCI
55+36=91
CMXCIX+CMXCIX=MCMXCVIII
999+999=1998
```

The program give right answers to each test case.

(3) Wrong cases



```
ye422@DESKTOP-32A0HDI:~/2023-1/assignments/study/assignments$ ./a.out
2
CIC
IMC
ABC
DEF
CIC+IMC=MCCXCVIII
199+1099=1298
Error. (Not a roman number)
```

Even if the input does not follow rule of roman number the program still can add the numbers. But if the input is not a roman number it prints error message and does not add the numbers.

The time complexity of this program is $\Theta(n)$, n is combined length of the two roman strings since the program should check value of each roman number alphabet.

Problem 2

1. Requirement

The smart rat problem is problem that requires program to get the maximum amount of food the rat can get when it move along the minimum distance to get out the square structure. As an input, first there is number of test cases. Then the program gets the column and row of the grid. Finally it gets the number of the food in each room by one row at a time. Then it does this process again til it gets information about every test case. When every test case are entered, he program prints the maximum amount of the food the rat can get when it move along the minimum distance to get out the square structure for each case.

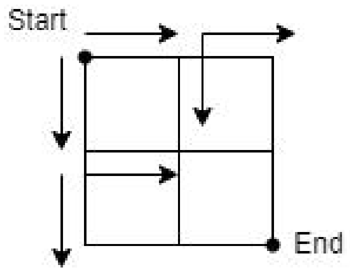
Thus the program need to calculate the maximum amount of the food the rat can get.

2. Analysis

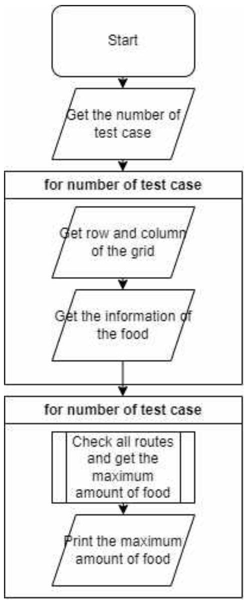
The method this program use to calculate the amount of the food is simple. Test every possible route the rat can go through and compare the amount of food it get during the route with current maximum amount of food. If the current amount of food is bigger, it will be the new maximum amount of food.

3. Design

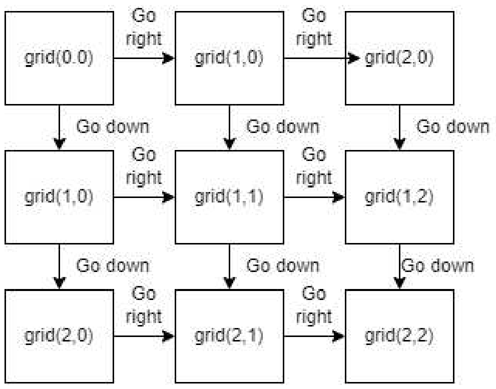
Then how to check every possible route is the problem. This program use top-down method for this problem. Suppose the rat is at any point of the grid, the rat can move to only two or less directions, which is right and down. Since the rat should move through the shortest route possible, the rat can only move to right and down at the grid. As long as it move through those directions it can move through the shortest route. The picture below shows rat can only move to two or less directions. Depending the position of the rat it can move to only one direction.



Thus the program can make function that sums up the amount of food the rat have eaten at current position. The function should be recursive to sums up the amount of food all the way to the end. There are two functions. One considers situation of the rat going right, and the other considers situation of going down. Each function should call those two functions recursively depending on the position of the rat. The function continues until the rat gets to the end point of the grid. The picture below is the flowchart of the program.


Start
Get the number of test case
for number of test case
Get row and column of the grid
Get the information of the food
for number of test case
Check all routes and get the maximum amount of food
Print the maximum amount of food

The picture below is diagram of checking every possible shortest route for 3X3 size grid.



At each position of the grid, Go right and go down function should be called. Thus I made each function recursive so that they can keep calling themselves until the reach the end point. At each call the function gets current position, size of the grid, information about the grid and the current amount of food as argument.

4. Refinement and coding

Before coding, I made brief pseudocode of the program.

Below are the pseudocodes about main function and subsequent go_down and go_right function.

```
main
{
     get the number of test cases n

     for n times
     {
          get number of the colomn and row of the grid
          generate new grid
          if ( size is 0 )
               print "wrong input!"

          for number of colomns
               for number of rows
                    get information of food at each position of grid

          go_down(current position, size of grid, current amount of food)
          go_right(current position, size of grid, current amount of food)
          save the result
     }

     for n times
          print the result of each test case
}
```

main function calls function for two directions

```
go_down(current position, size of grid, current amount of food)
{
    add up the amount of food

     if rat reached end position
     {
          if current amount of food is bigger than current maximum amount of food
               current amount of food is new maximum
     }
     else if rat can't go down anymore
          go_right(new position, size of grid, current amount of food)

     else if rat can't go right anymore
          go_down(new position , size of grid, current amount of food)
     else if  rat can go anywhere
     {
          go_right(new position, size of grid, current amount of food)
          go_down(new position, size of grid, current amount of food)

}
```

function for going one grid down

```
go_right(current position, size of grid, current amount of food)
{
     add up the amount of food

     if rat reached end position
     {
          if current amount of food is bigger than current maximum amount of food
               current amount of food is new maximum
     }
     else if rat can't go down anymore
          go_right(new position, size of grid, current amount of food)

     else if rat can't go right anymore
          go_down(new position, size of grid, current amount of food)
     else if  rat can go anywhere
     {
          go_right(new position, size of grid, current amount of food)
          go_down(new position, size of grid, current amount of food)
     }
}
```

function for going one grid right

First the program gets the number of test cases to run. For that number of cases the function gets the size of the grid and makes it. I used 2 dimensional array to generate the grid. Then the main function gets information about the amount of the food at each grid. The main function calls functions for two directions because there are two ways to start the route at the start point. In the function for going one grid down and one grid left, there is conditional statement to check if the function should call itself or the other direction function depending on the position of the rat. At end point rat doesn't have to move, if the rat reaches bottom of the grid it should only move to right, or if the rat reaches wall of the grid it have to move down. If rat doesn't reach anything yet, it can move into two directions. At each case, the function calls the direction function recursively with the new position and updated amount of food the rat have eaten by now as parameters.

If the rat reaches end point, the function compares the amount of food the rat have eaten by now with the current maximum. The maximum number is saved as global variable. If the amount is bigger than max number til now the max number will change to new maximum.

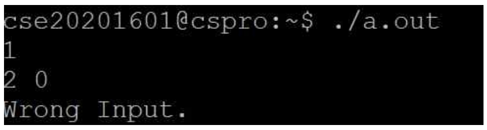
When the functions for two directions check all routes they will terminate. The main function prints the maximum amount the rat had eaten for each test cases.

5. Verification

The time complexity of the program is the picture below.

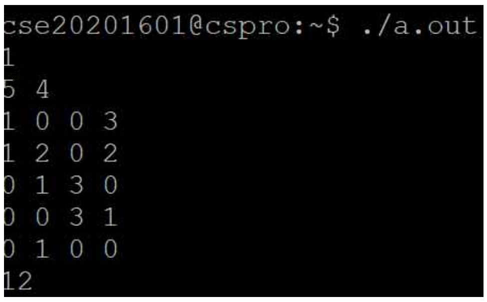
$$\Theta(2^{row+column})$$

Since the program runs recursive function that checks for two directions at each point of the grid, the time complexity is heavily influenced by sum of number of row and column. Thus the program would not be able to process the case of grid size of (100,100) since the computation would be so much for the computer.
I tested the program with 3 test cases.
First case is the case that the user enters wrong input such as 0 to grid size.



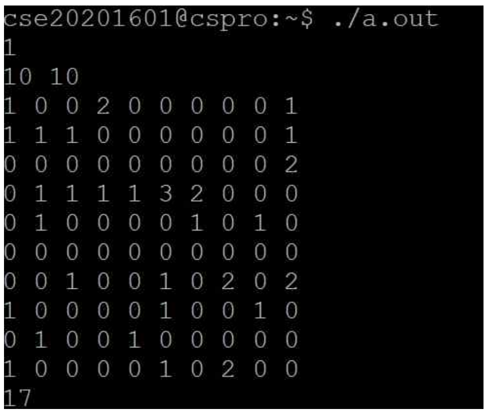
```
cse20201601@cspro:~$ ./a.out
1
2 0
Wrong Input.
```

The second case is normal case



```
cse20201601@cspro:~$ ./a.out
1
5 4
1 0 0 3
1 2 0 2
0 1 3 0
0 0 3 1
0 1 0 0
12
```

The last case is case with bigger grid.



```
cse20201601@cspro:~$ ./a.out
1
10 10
1 0 0 2 0 0 0 0 0 1
1 1 1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 2
0 1 1 1 1 3 2 0 0 0
0 1 0 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 2 0 2
1 0 0 0 0 1 0 0 1 0
0 1 0 0 1 0 0 0 0 0
1 0 0 0 0 1 0 2 0 0
17
```

The program processed these cases with no problem.