Introduction to Computers and Programming LAB-Final

2017/01/11 Time: 2 hrs

※Please create a new folder and name it as your Student ID. Inside the folder, your file format will be Q_1.c, Q_2 (project), Q_3.c, etc. (There will be score deduction for wrong file name).

※Any C library is forbidden, except for <stdio.h>, <stdlib.h> and <string.h>

※We will ask you to leave and give you 0 point if any suspicious or cheating behavior is found

※The class is for C language, so do not use C++.

※Follow the input/output format in examples.

1. **Hello, World!** [50%]

Please implement the following operations. (1) to (4) should be implemented in **DIFFERENT** functions, and use functions (1) to (4) you wrote to finish (5) in the main function.

(1) **Greatest Common Divisor** [5%]

Given arguments *i* and *j* (integer), where i ≧ j. Write a **recursive** function to show and return the GCD of them. The recurrence relation is: $\gcd(i,j) = \begin{cases} i & , if\ \ j = 0 \\ \gcd(j,\ i\ \ mod\ \ j) & , otherwise \end{cases}$

(2) **Decimal to Binary** [10%]

Given an argument *i* (integer). Write a function transform it to a binary number, and show it. Then count and return the number of zero in the binary result.

(3) **Sum of digits** [10%]

Given arguments *i* and *j* (integer), where i ≧ j. Write a function to first catenation two numbers, then compute and return the summation of each digit until the result is less than 10.

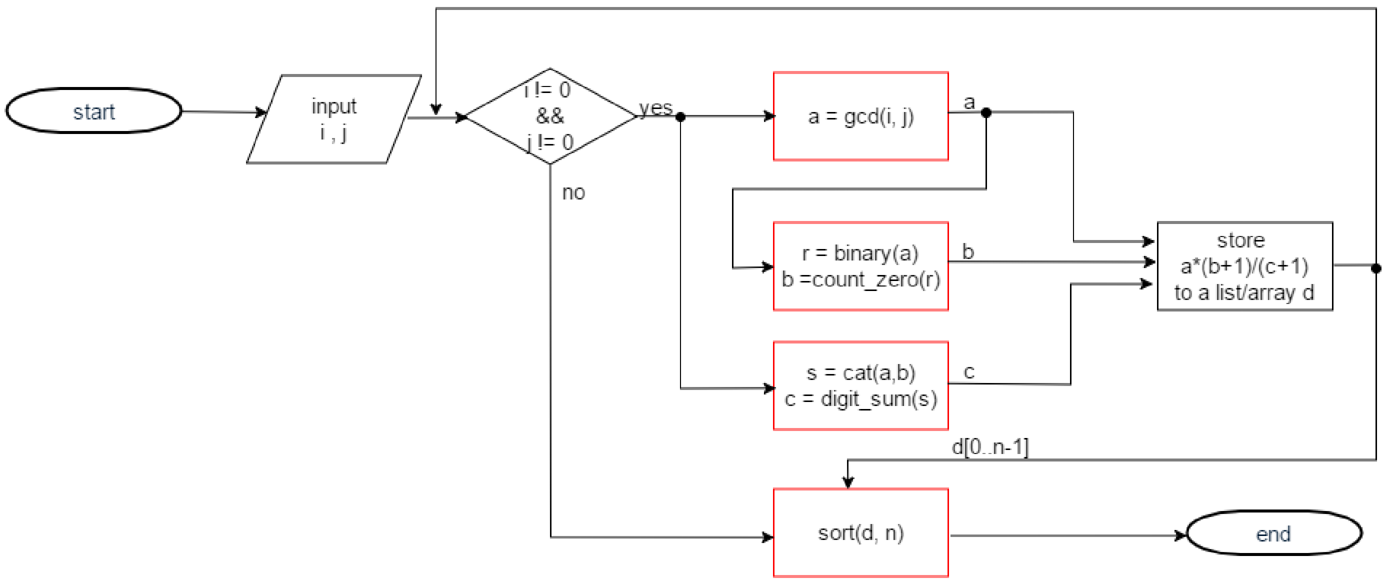
(4) **Sort and print in specific format** [15% for sorting, 5% for format]

Given arguments array *arr* (double) and the size of it *n* (integer). Write a function to sort them in ascending order, and show the sequence. Every number is rounded to the 3rd digit after the decimal point, the total length (including decimal point) is 8. If the length isn't enough, fill 0. If the length exceeds 8, do nothing.

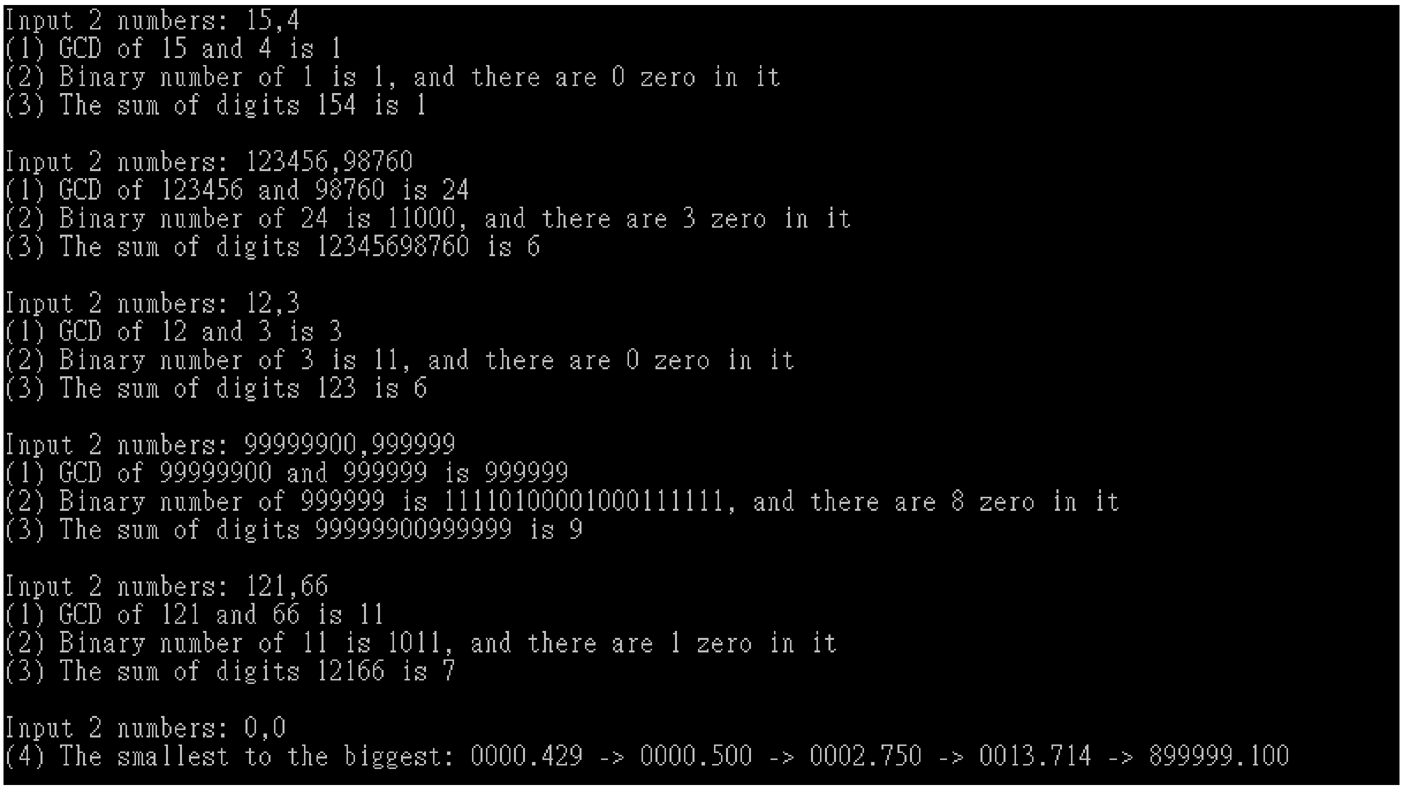
(5) **All operations** [5%]

The program asks users to **input two non-negative integers**, where the first is not less than the second one, and separated by comma (,). Check if any input equals to zero. If not, first **calculate the GCD of two inputs** *[a]*. Next, transform the result *a* to binary number, and **count the number of zero in the binary number** *[b]*. After that, the program catenation two inputs to a new integer, and **compute the sum of each digit until the result is less than 10** *[c]*. Finally, **store the value of *a*(b+1)/(c+1)*** to an array or a linked list, and asks users to input again. If one of inputs is zero, then **sort the array that stores all results of previous inputs into ascending order**. Show the sorting result in the format mentioned before. (The times of inputs will not exceed 20, and at least 2 times)

- **Flowchart**



- **Input/Output example**



```
Input 2 numbers: 15,4
(1) GCD of 15 and 4 is 1
(2) Binary number of 1 is 1, and there are 0 zero in it
(3) The sum of digits 154 is 1

Input 2 numbers: 123456,98760
(1) GCD of 123456 and 98760 is 24
(2) Binary number of 24 is 11000, and there are 3 zero in it
(3) The sum of digits 12345698760 is 6

Input 2 numbers: 12,3
(1) GCD of 12 and 3 is 3
(2) Binary number of 3 is 11, and there are 0 zero in it
(3) The sum of digits 123 is 6

Input 2 numbers: 99999900,999999
(1) GCD of 99999900 and 999999 is 999999
(2) Binary number of 999999 is 11110100001000111111, and there are 8 zero in it
(3) The sum of digits 99999900999999 is 9

Input 2 numbers: 121,66
(1) GCD of 121 and 66 is 11
(2) Binary number of 11 is 1011, and there are 1 zero in it
(3) The sum of digits 12166 is 7

Input 2 numbers: 0,0
(4) The smallest to the biggest: 0000.429 -> 0000.500 -> 0002.750 -> 0013.714 -> 899999.100
```

- **Hint & Note:**

(1) TA will score your program operation by operation. An error in an operation (1) to (4) won't affect others. However, these operations have to be correct in order to get the last 5 points in (5).

(2) To ease the testing process of TA, please name your function meaningfully, and try not to use global variables as far as possible.

(3) Catenation of two integers may exceed the range of an integer.

2. **Reverse Polish notation (RPN)** [25%]

You *cannot* implement any function and global variable in “main.c”.

Besides, you *cannot* implement more than 5 line of code in “int main()”.

Design a program that can calculate the value of the input RPN, also called postfix notation, and can convert the postfix notation to infix notation. The input RPN is composed of operands (numbers) and operators (+, -, *, /), and the length of the input RPN is no more than 100. The operands and operators in the input RPN are separated by spaces.

Your program must have at least 2 header files (”stack.h”, “calc.h”) and their corresponding source files (“stack.h”, “calc.c”) and also the main function.

-“calc.c”: calculate the value of the input RPN and also show the infix notation of the RPN. If the input is not a valid RPN, then you need to print “Invalid”.

-“stack.c”: a stack that need to have at least such functions below:

--push: push the element into the stack.

--pop: remove an element from the top of the stack.

--is_empty: show the stack is empty or not.

--is_full: show the stack is full or not.

--top: show the element that is on the top of the stack.

--print_stack: print all the elements from the bottom to the top of the stack.

✧ You can name the function above by yourself, we will check whether your stack has such function and can work successfully or not.

Note:

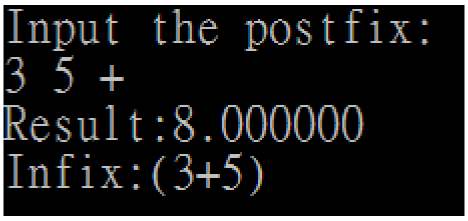
(1) If you can successfully calculate the value of the input RPN, then you can get 15pts.

(2) We will see if your output infix notation is correct or not if and only if your program can calculate the correct value of all the input RPNs. This part is worth the remaining 10 points.

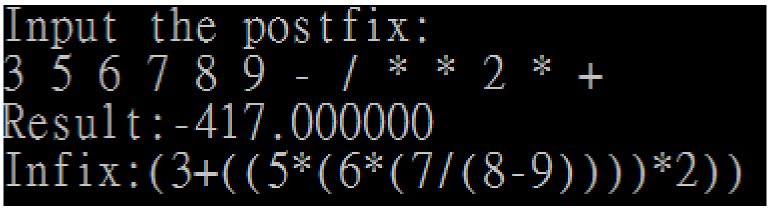
(3) If your stack doesn’t work, you will get -5 points as penalty.

(4) We will test a very simple input with only 1-digit natural number, so DON’T GIVE UP!

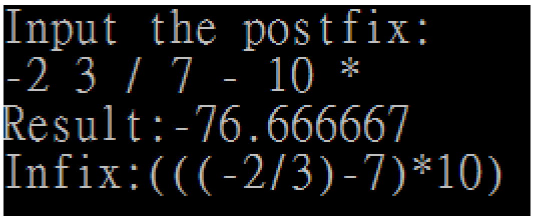
Example:



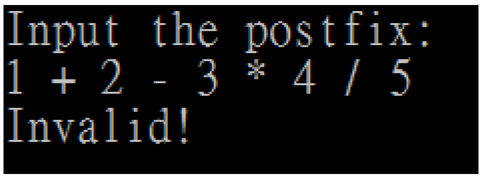
```
Input the postfix:
3 5 +
Result:8.000000
Infix:(3+5)
```



```
Input the postfix:
3 5 6 7 8 9 - / * * 2 * +
Result:-417.000000
Infix:(3+((5*(6*(7/(8-9))))*2))
```



```
Input the postfix:
-2 3 / 7 - 10 *
Result:-76.666667
Infix:(((-2/3)-7)*10)
```



```
Input the postfix:
1 + 2 - 3 * 4 / 5
Invalid!
```

3. **Calculate Item Effects** [25%]

In most RPG, players need to collect many items in order to enhance their ability.

Please write a program that:

First reads the item information from **"items.txt"**

and then outputs the formatted information into **"formattedItems.txt"**.

Second, reads the player information from **"player.txt"**,

calculates the total effect and then outputs the result into **"result.txt".**

(Output item information and the sum of each effect values , **include duplicate items**)

Input file formats:

items.txt:

Each line is a definition of an item:

First string is the item name and followed by **3** effect values (integer).

They are divided by **one or more space**.

If a line has illegal definition, output error message (with line number) and ignore it.

You must show that the error is caused by **too many** or **too few** effect value.

If an item is already existed, output error message (with line number) and ignore it.

Please ignore empty lines. (However the line number should still be increased.)

player.txt:

The first line of this file is the player's name **(May contain space)**.

And the remain lines are player's items. (Each line is an item; Player can have same items.)

If an item is not existed, output error message (with line number) and ignore it.

Please ignore empty lines. (The first line of this file is always not empty.)

Output file formats:

formattedItems.txt:

Each line is a formatted item definition:

```
fprintf(formattedItems, "%8s %3d %3d %3d\n", Item_Name, Effect1, Effect2, Effect3);
```

result.txt:

The first line:

```
fprintf(result, "Player: %s\n", Player_Name);
```

The middle lines are item information of player's items:

```
fprintf(result, "%8s %3d %3d %3d\n", Item_Name, Effect1, Effect2, Effect3);
```

The last two lines:

```
fprintf(result, "-------------------\n");
fprintf(result, "%8s %3d %3d %3d\n", "Total:", Sum1, Sum2, Sum3);
```

Notes:

1. **items.txt** may contain so many items, so please use **dynamic memory allocation** to store items.
2. **The line number is starting from 1.**
3. **Please refer to example for output detail.**

Example:

Input files:

<<items.txt>>

```
Dagger 100 50
   Cap  10   20  30
Cap 4 5 6

Blade 9 5 2 7
Necklace 50
Sword 20 -1 0



Armor 10 10 10

```

<<player.txt>>

```
Bob Chen
Ring
Cap

Armor
Armor
```

Output:

Standard Output

```
Load items from items.txt:
Error happen at line 1, too few effect values!!
Error happen at line 3, item "Cap" already exist!!
Error happen at line 5, too many effect values!!
Error happen at line 6, too few effect values!!
Success load 3 items!!

Calculate total effects from player.txt:
Error happen at line 2, cannot find item "Ring"!!
Player: Bob Chen
Total:   30   40   50
```

<<formattedItems.txt>>

```
  Cap   10   20   30
Sword   20   -1    0
Armor   10   10   10
```

<<result.txt>>

```
Player: Bob Chen
    Cap   10   20   30
  Armor   10   10   10
  Armor   10   10   10
--------------------
  Total:   30   40   50
```