Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of "plagiarism":

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…

• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …

• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one's account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don't think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else's work.

CS100 Homework 2 (Fall, 2019)

In this homework, you are required to do some programming for more advanced concepts in C language that you have learned in the class: branching, loops, pointers, and arrays.

Percentage of this homework over the whole score: 7%

Submission deadline:

2019-10-08 23:59

Rules for dynamic arrays

In this homework (**for each problem**), you may need arrays with dynamic lengths.

To implement this, you should use dynamic memory allocation which has been covered in class.

For each problem, you should always initialize your array with **length = 5.**

Important: You should get this initial dynamic array with function **malloc()**, using static arrays (e.g. **int array[5]**) will result in a substantial deduction of your score!

When your array is full after you add an item (e.g. after adding the 5^{th} item), you should resize your array by doubling its size, in other words,

new_size = 2 * original_size

In consequence, the size of your array should be 5, 10, 20, 40, 80….

Once you need to resize your array, you should **output a message** indicating how you resized your array in the form given below:

(resize) from 5 to 10
(resize) from 10 to 20

Also, after you resize your array and at the end of your program, you should use function free() to prevent memory leak. Failing to do so may result in a deduction of your score.

Note：If you solve these problems without using Dynamic memory allocation, you will receive a substantial deduction of your final score on this homework, even if you have fully passed the Online Judge!

Problem 1: Dynamically score input (30%)

In CS999 midterm examination, Prof. Wang wants to calculate the average score of his class. But unfortunately, he has forgotten the number of students in his class, so, **the number of scores is unknown in advance**. To help him solve this problem, you may need **dynamic memory allocation** to enlarge memory to store the scores once needed (see rules for dynamic arrays on the previous page).
All scores are **non-negative** numbers, and **a -1 indicates the end of input.**

After all the scores are stored into your array, you can calculate the average score. Then you should output it.

The following is a demo process for the program if you can do it successfully: (red indicates input)

```
Please type scores to be calculated:
92.6
17.5
71.0
66.3
51.9
(resize) from 5 to 10
99.6
-1
Average score: 66.48
```

Input description:
Input contains **unknown** number of lines.
All **non-nagative** inputs represent scores. They are **floating point numbers with at most 1 decimal point digit**.
The **final input is -1**, indicating the end of input.

Output description:
First, you should output a prompt:

Please type scores to be calculated:

Note that there should be **no whitespaces** after the colon, and you should end this prompt with a **newline.**
Then, once you need to resize your array, output a message like this:

(resize) from 5 to 10

Finally, after getting the final input -1, you should output **Average score:** , followed by the average score of the inputs.

*** All of your outputs should be printed with "printf" to only 2 decimal point digits floating point number, even if a number is an integer.**

*** The last number -1 is not a score, so:**

(1) It should not be calculated when you calculate average score.

(2) It will not be stored into your array, and therefore will not cause a resize.

Problem 2: Sorting a dynamically created array (30%)

After Prof. Wang has got the average score, he wants to sort the scores in **non-descending order** (from small to large numbers). In this problem, you are also required to input a series of scores based on the user's input. Then you will implement the **bubble sort algorithm** to sort these scores, and display them in non-descending order.

The pseudo-code of the Bubble Sort algorithm is given below:

```
procedure bubble_sort (A: list of sortable items)
    n = length(A)
    for i = 0 to n - 1
        for j = 1 to n-1
            if A[j-1] > A[j] then
                swap(A[j-1], A[j])
            end if
        end for
    end for
end procedure
```

In this problem, you will get some lines of students' information. A line contains a student's ID number and his/her score, separated by a whitespace (' '). The input ends with a line of "-1 -1". As Prof. Wang doesn't know the number of his students, you will also need a dynamic memory allocation, and the rules of initializing your array and resizing your array are the same as for Problem 1. To store the students' ID numbers and scores, you may need two dynamic arrays, and they should all obey the previously introduced resizing rules.

Note: If two students have same scores, you should sort them by their student ID numbers, in non-descending order.

The following is a demo process for the program if you can do it successfully: (red indicates input)

```
Please type scores to be calculated:
2099581001 92.6
2099583065 70
2099581022 66.3
2099581020 66.3
2099583199 51.9
(resize) from 5 to 10
(resize) from 5 to 10
2099000000 99.6
-1 -1
---Result---
2099583199 51.9
2099581020 66.3
2099581022 66.3
2099583065 70.0
2099581001 92.6
2099000000 99.6
```

Explanation of two "(resize)…" lines: Because you use two arrays, each will resize after the 5th input, so your program will output two lines of "(resize) …".

Input description:
Input contains **unknown** lines, each line represents a student number and a score, **separated by a whitespace**.
The student numbers are **n different integers**
*** They may look big, but they will not exceed the maximum limit of int.**
The scores are **n floating point numbers with at most 1 decimal point digit**.

Output description:
First, you should output a prompt:

Please type scores to be calculated:

Note that there should be **no whitespaces** after the colon, and you should end this prompt with a **newline**.
Then, once you need to resize the size of your array, output a message like this:

(resize) from 5 to 10

Then, after getting all the inputs, you should output **---Result---** ended with a **newline**.
Finally, output students' information (student numbers and scores) in a **non-descending order**. If two students have the same score, you should sort them by their student numbers, in non-descending order.

*** All scores should be printed with "printf" to only 1 decimal point digits floating point number, even if a score is an integer, and all student numbers should be printed with "printf" to integers.**

PROBLEM 3 IS ON THE NEXT PAGE

Problem 3: Matching a continuous sequence of numbers within the input data (40%)

You are first given a set of values based on user inputs. Name it set **A**.

Then, you are given another set of values **B**, where the number of values in set B should be smaller than that of set **A** (**The size of B is smaller than that of A**).

*** The order of elements in a set is fixed. e.g. [1, 2, 3] and [3, 2, 1] are two different sets. You are not supposed to change the order of any set.**

Then we try to find a **continuous range** of values (a subset) in set **A**, **with the same length as set B**, where the difference between set **B** and the range of values in **A** is minimized.

Definition: The **difference** between two equal-sized sets **A = [a1, a2, a3, …an]** and **B = [b1, b2, b3, …bn]** is

$$\text{diff}(A, B) = \sum_{i=1}^{n} |a_i - b_i|$$

E.g. A = [1, 2, 4]; B = [5, 6, 1], then diff(A, B) = |1 - 5| + |2 - 6| + |4 - 1| = 11

Note: If there are many continuous ranges in A with the same minimum difference as B, you should output the left-most one.
E.g. A = [2, 4, 6, 8] and B = [3, 5], then B has the same difference with [2, 4] and [4, 6] (both equal to 1), but you should output [2, 4], instead of [4, 6].
The following is a demo process for the program if you can do it successfully: (red indicates input)

```
Please input the set A:
3
2
4
1
7
 (resize) from 5 to 10
5
-1
Please input the set B:
2
3
5
-1
---Result---
3 2 4
```

Input description:
Input contains **n+m+2** lines.
The first **n** lines are values in set A. Each line contains a positive integer.
The **(n+1)th line is -1**, indicating the end of set A.
The next **m** lines are values in set B. Each line contains a positive integer.

The **last line is -1**, indicating the end of set B.

Output description:

First, you should output a prompt:

Please input the set A:

Then, user will input **n** numbers, followed by a "-1"

After you receive the "-1", you should output again a prompt:

Please input the set B:

Then, user will input **m** numbers, followed by a "-1"

Note that there should be **no whitespaces** after each prompt, and you should end each prompt with a **newline**.

Once you need to resize your array, output a message like this:

(resize) from 5 to 10

Then, after getting all the inputs, you should output **---Result---** ended with a **newline**.

Finally, output m integers, which is the continuous range (subset) you found in set A with the minimal difference to set B.