Assignment 3

Software Systems Analysis and Design

Task 1: Linear Systems

Write a computer program in C++ programming language to solve a system of linear equations $A \cdot x = b$. For this purpose implement a class «ColumnVector» with necessary fields, methods and necessary operators' overloading for summation, multiplication, and computing the norm. When the system is inconsistent and does not have a solution, or if it has infinite solutions, you should print a message accordingly: “NO” or “INF”.

Comment your code explaining the steps: elimination, and the different steps as in the previous exercise (see the examples). Accomplish the diagonal normalization.

Input format

The input is through ***standard input (cin)*** following this format:

- The size of the square matrix A (m*m, introduce m only)
- A square matrix A.
- A vector of free coefficients b (in element-wise manner with the dimension firstly).

You separate between them with a return line.

Output format

Print the final result in ***the standard output (cout)***.

Test cases

Input	Output

// Step #0
4
2 1 3 2 2.00
2 1 5 1 2.00
2 1 4 2 2.00
1 3 3 2 1.00
4 0.00
0 2 1 6 2.00
1.00
6.00
// step #1: elimination
2.00
0.00
2.00
1.00
0.00
2.00
1.00
6.00
// step #2: elimination
2.00
0.00
0.00
1.00
0.00
	1.00 3.00 2.00
	1.00 5.00 1.00
	1.00 4.00 2.00
	3.00 3.00 2.00
	1.00 3.00 2.00
	0.00 2.00 -1.00
	1.00 4.00 2.00
	3.00 3.00 2.00

2.00
1.00
6.00
 	
// step #3: elimination
 	
2.00
0.00
0.00
0.00
0.00
2.00
1.00
6.00
 	
// step #4: permutation
 	
2.00
0.00
0.00
0.00
0.00
6.00
1.00
2.00
 	
// step #5: permutation
 	
2.00
0.00
0.00
0.00
	1.00 3.00 2.00
	0.00 2.00 -1.00
	0.00 1.00 0.00
	3.00 3.00 2.00
	
	1.00 3.00 2.00
	0.00 2.00 -1.00
	0.00 1.00 0.00
	2.50 1.50 1.00
	
	1.00 3.00 2.00
	2.50 1.50 1.00
	0.00 1.00 0.00
	0.00 2.00 -1.00

0.00	1.00 3.00 2.00
6.00	2.50 1.50 1.00
2.00	0.00 2.00 -1.00
1.00	0.00 1.00 0.00
// step #6: elimination	
2.00	
0.00	
0.00	
0.00	1.00 3.00 2.00
0.00	2.50 1.50 1.00
6.00	0.00 2.00 -1.00
2.00	0.00 0.00 0.50
0.00	
// step #7: elimination	
2.00	
0.00	
0.00	
0.00	1.00 3.00 2.00
0.00	2.50 1.50 1.00
6.00	0.00 2.00 0.00
2.00	0.00 0.00 0.50
0.00	
// step #8: elimination	

2.00
0.00
0.00
0.00
0.00
6.00
2.00
0.00
 	
// step #9: elimination
 	
2.00
0.00
0.00
0.00
0.00
6.00
2.00
0.00
 	
// step #10: elimination
 	
2.00
0.00
0.00
0.00
0.00
4.50
2.00
0.00
	1.00 3.00 2.00
	2.50 1.50 0.00
	0.00 2.00 0.00
	0.00 0.00 0.50
	
	1.00 3.00 0.00
	2.50 1.50 0.00
	0.00 2.00 0.00
	0.00 0.00 0.50
	
	1.00 3.00 0.00
	2.50 0.00 0.00
	0.00 2.00 0.00
	0.00 0.00 0.50

// step #11: elimination	
2.00 0.00 0.00 0.00 -3.00 4.50 2.00 0.00	1.00 0.00 0.00 2.50 0.00 0.00 0.00 2.00 0.00 0.00 0.00 0.50
// step #12: elimination	
2.00 0.00 0.00 0.00 -4.80 4.50 2.00 0.00	0.00 0.00 0.00 2.50 0.00 0.00 0.00 2.00 0.00 0.00 0.00 0.50
// Diagonal normalization	
1.00 0.00 0.00 0.00 -2.40 1.80	0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 1.00

1.00 0.00 // Result	 -2.40 1.80 1.00 0.00

Notes

The type of the elements of the matrix is double. Print the result using the formatting to 2 digits after the floating point. Also, for very small numbers 10^(-10) consider them 0.0.

Task 2: Bag

A bag (also called multiset) is a generalization of a set, where elements are allowed to appear more than once. For example, the bag `{a,a,b}` consists of two copies of `a` and one copy of `b`. However, a bag is still unordered, so the bags `{a,b,a}` and `{a,a,b}` are equivalent.

There are few rules that you need to follow in this task:

- In addition to storing the element, the number of copies of the element is also stored, which is always positive. For example, the multiset `{a,b,b,a,c}` is represented as `[(a,2),(b,2),(c,1)]`.

- For a given value, at most one cell storing that value should appear in the data structure.
- Maximum number of instances of one element in the bag will be 10000.
- The bag can contain **only lower-case English characters** (`char`). There will be **no invalid inputs** in this task, so you are not required to check it.

You have to implement the following classes and features:

- `Class Bag`: implements the multiset and contains the following features:
  - `vector<CellInfo> elements`: vector that contains elements of the multiset (see below the specification of class `CellInfo`);
  - `void insert(char val, int n)`: the method that inserts `n` copies of value `val` to the bag;
  - `void remove(char val, int n)`: the method that removes as many copies of `val` as possible, up to `n`. For example, removing one copy of a from the bag `{a,a,b}` will result in a bag `{a,b}`, while removing two copies of `c` from the same bag will not change it;
  - `char min()`: the method that returns the minimum element (regardless the number of occurrences of it) from the bag. For example, the minimum element from the bag `{a,a,b}` will result in `a`;
  - `char max()`: the method that returns the maximum element (regardless the number of occurrences of it) from the bag. For example, the maximum element from the bag `{a,a,b}` will result in `b`;
  - `boolean isEqual(Bag b)`: the method that returns `1` if bag `b` is equal to the current bag. Note that two empty bags are considered to be **equal**.
- `Class CellInfo`: contains the item and number of copies of that item. This class should have at least the following features:
  - `char value`: value (lower-case english letter) of the cell;
  - `int copies`: the number of copies of the value.

In the main part of the program you should initialize two empty bags, read the integer `n` from the standard input, and then perform `n` actions on them.

Actions are represented in the following format:

Insert	`i <int> <char> <int>`	• `i` shows that *insertion* should be performed • `<int>` shows on which bag insertion should be performed (can be only `1` or `2`) • `<char>` shows which character should be inserted • <int> shows how much copies of character should be inserted to the bag

	`i 1 b 4`	This input shows that we should insert four copies of character `b` to the first bag. This input does not require output.
Remove	`r <int> <char> <int>`	• `r` shows that *removing* should be performed • `<int>` shows on which bag removing should be performed (can be only `1` or `2`) • `<char>` shows which character should be removed • <int> shows how much copies of character should be removed from the bag
	`r 2 c 5`	This input shows that we should remove five copies of the character `c` from the second bag. This input does not require output.

After performing all actions, you should use the above-mentioned methods to print the following data:

- Maximum value of the first bag (if bag is empty, print `-1`),
- Minimum value of the second bag (if bag is empty, print `-1`),
- Result of comparing two bags (print 1 if bags are equivalent and 0 otherwise).

Test cases

Input	Output	Explanation

`5`
`i 1 a 3`
`i 1 b 2`
`i 2 b 2`
`i 2 a 2`
`i 1 c 1`
	`c a 0`	The following actions are performed:
		• Insert `3` copies of `a` to the first bag.
		• Insert `2` copies of `b` to the first bag.
		• Insert `2` copies of `a` to the second bag.
		• Insert `2` copies of `b` to the second bag.
		• Insert `1` copy of `c` to the first bag.
		The resulting bags are:
		• First bag: `{a,a,a,b,b,c}`
		• Second bag: `{a,a,b,b}`
		The first symbol in the output represents the maximum value in the first bag (`c`), while the second symbol represents the minimum value in the second bag (`a`). The third number is the output is `0` which means that bags are not equivalent.

7 i 1 a 2 i 1 b 2 i 1 g 2 r 1 g 3 i 2 b 2 i 2 a 1 i 2 a 1	b a 1	The resulting bags are: • First bag: {a,a,b,b} • Second bag: {a,a,b,b} These bags are equivalent.

Evaluation

- Successful completion of all test cases is mandatory, failure to do so will result in score ZERO.
- You should use OOP to solve this task. If your solution works but is implemented in a procedural way, your grade for this task will be decreased by 75%.
- If your solution does not contain operators overloading your grade for this task will be decreased by 15%.
- If your solution does not contain ColumnVector your grade for this task will be decreased by 15%.
- If your solution does not contain handling of inconsistent systems your grade for this task will be decreased by 15%.
- If your solution does not contain handling of infinite solutions your grade for this task will be decreased by 15%.
- If you do not explain steps of your solution your grade for this task will be decreased by 25%.

Submission

- You have to **submit to Moodle your submission link in Codeforces**.
- In the assignment you can submit as text. So write there the correct submission link to Codeforces submission.
- In codeforces and in every code file you submit do the following:
    - In the first line **write your full name in English** as a comment
    - In the Second line write your Inno email as a comment

Note: Failing to do any other the above points will result in a penalty

Good luck