Assignment 2: Stack and Queue

- **Consider the following tables for your Queue/Stack implementation.**

Table 1: Queue Abstract Data Type.

Function No	Function	Parameter	Return value	Queue after execution	Comment
	[before each execution consider current queue]			<20, 23 , 12, 15>	The values were enqueued in the following order: 20, 23, 12, 15. So, the front points to the value 20 and rear points to the value 15. Note that this is a logical definition without any implementation related specification.
1	clear()	-		<>	Reinitialize the queue, i.e., make it (logically) an empty queue. <> means an empty queue. **Clear allocated memory if the programming language permits.**
2	enqueue(item)	19		<20, 23 , 12, 15, 19>	Enqueue an element. You can design your own return flags too.
3	dequeue()		20	<23, 12, 15>	Dequeue an element.
4	length()	-	4	<20, 23, 12, 15>	Return the number of elements in the queue.
5	frontValue()	-	20	<20, 23, 12, 15>	Return the front element.
6	rearValue()	-	15	<20, 23, 12, 15>	Return the rear element

7	leaveQueue()	-	15	<20, 23, 12>	Return the rear element which has left the queue

Table 2: Stack Abstract Data Type.

Function No	Function	Param eter	Return value	Stack after execution	Comment
	[before each execution consider current queue]			<20, 23 , 12, 15>	The values were pushed in the following order: 20, 23, 12, 15. So, TOP points to the value 15. Note that this is a logical definition without any implementation related specification.
8	clear()	-		<>	Reinitialize the stack, i.e., make it (logically) an empty stack. <> means an empty stack.
9	push(item)	-1		<20, 23 , 12, 15, -1>	Pushes an element.
10	pop()		15	<20, 23, 12>	Pop an element.
11	length()	-	4	<20, 23, 12, 15>	Return the number of elements in the slack.
12	topValue()	-	15	<20, 23, 12, 15>	Return the current TOP element.

The implementations must support the various types of ‘elements’. You might use templates in C++ or something equivalent in the programming language you prefer to use. You need to provide two different implementations, namely, 1) Array Based (Arr) and 2) Linked List (LL) Based Implementations, using your codes from assignment-1. The sizes of both the data structures (DS) are only limited by the memory of the computing system, i.e., in case of Array-based implementation, there must be a way to dynamically grow the size as follows: the DS should double its current size by **allocating memory dynamically**, i.e., initially the list should be able to hold X elements; as soon as the attempt is made to insert the X+1th element, memory should be allocated such that it can hold 2X elements and so on. **Allocate additional**

memory only when the list cannot contain any more items. Static implementation would result in deduction of marks.

Appropriate constructors and destructors must be implemented. For the Array-based implementation, the constructor takes the initial size of the array with a default size available as a constant. You should keep a way to reallocate to this default size when needed (like when the data structure becomes empty).

You may implement extra helper functions but those will not be available for programmers to use. So, while using the implementations, one can only use the methods listed in the Tables. Please note that during evaluation, identical main functions will be used to check the functionality of both Array and LL implementations. Also write a simple main function to demonstrate that all the functions are working properly, for both implementations. Recall that, the same main function should work for all the implementations except for the object instantiation part. Please follow the following input/output format.

Input format (for checking the Arr and LL implementations):

First line will contain two integers K,X (K < X) describing the initial length of the Queue (K) and initial capacity X. (Ignore X for LL implementation).

Next line will contain K space separated integers denoting the initial Queue elements.

Next line will contain two integers M,Y (M < Y) describing the initial length of the Stack (M) and initial capacity Y. (Ignore Y for LL implementation).

Next line will contain M space separated integers denoting the initial Stack elements.

Next line will contain a single integer, Q denoting the number of operations to process.

Each of the next Q lines will contain two integers F (1<=F<=12), P where F denotes the function to execute, and P is its parameter. Ignore P in case F doesn't take any parameter.

(Note: P denotes functions for both the queue and the stack. So, you will need to create two data structures in the same demonstration. My suggestion will be to create two headers for these two data structures.)

Inputs will be given in a text file named - input.txt.

Example input:\

```
4 8
20 23 12 15
4 8
20 23 12 15
6
```

1 1

2 19

6 0

9 -1

10 0

12 0

Output Format:

At first the system will output in one line the items of the two Data Structures in two lines within angle bracket, space separated identifying the front/top position using “|” as shown in Assignment 1. Then for each of the Q operations output two lines as follows: The first line will output the items of the DS as described above. The second line will output the return value (if available) or -1 (if no return value is expected).

[There is another task in the next page]

Task : 2

In a popular gaming store, people come to play fifa. The store has two gaming consoles. So at a time, maximum two people can have their service. However, they usually use one console.
Only when the number of customers waiting in line reaches 3, do they start the other console.
At that time new customers are told to stand in another line in a bigger hall. But, the manager of that hall lets the latest entering customer in that line play at first.
Assume, starting and ending the games require no extra time lag. Also, the time required to enter the store or to find the appropriate line can be ignored.
You have to:

- In each of n lines, print the departing time of each customer serially, separated by a space. (print c if the customer could not be served in due time)

Input/Output Explanation:
At first the input file will contain two integers n, c: n denotes the number of customers to come to the store, c denotes the latest possible timestamp the store will remain open.
Each of the following n lines will contain 2 integers (s, d): s (s<c) denotes the arrival timestamp of the customer and d denotes the duration he has rented the console for.

Sample Input:

```
8 60
2 12
3 13
6 14
8 11
10 11
15 10
16 8
20 13
```

Sample Output:

```
1 > 14
2 > 27
3 > 41
4 > 52
5 > 21
6 > 60
7 > 42
8 > 34
```

Explanation: [Hint : The whole scenario can be simulated with the queue and the stack Data Structures you have implemented in Task - 1 very easily.]

Timestamp	Event	Line1	Line2	Explanations/

				Other relevant activities
2	Customer 1 arrives	—	—	Customer 1 starts playing, ending at 2+12 = 14
3	Customer 2 arrives	Customer 2	—	Customer 2 is waiting
6	Customer 3 arrives	Customer 2, Customer 3	—	Customer 3 is waiting
8	Customer 4 arrives	Customer 2, Customer 3, Customer 4	—	Customer 4 is waiting
10	Customer 5 arrives	Customer 2, Customer 3, Customer 4	—	Customer 5 starts playing, ending at 10+11 = 21
14	C1 leaves	Customer 3, Customer 4	—	C2 starts playing, ending at 14+13 = 27
15	C6 arrives	Customer 3, Customer 4, Customer 6	—	C6 is waiting
16	C7 arrives	Customer 3, Customer 4, Customer 6	Customer 7	C7 is waiting
20	C8 arrives	Customer 3, Customer 4, Customer 6	Customer 7, Customer 8	C8 is waiting
21	C5 leaves	Customer 3, Customer 4, Customer 6	Customer 7	C8 starts playing, ending at 21+13 = 34
27	C2 leaves	Customer 4, Customer 6	Customer 7	C3 starts playing, ending at 27+14 = 41
34	C8 leaves	Customer 4, Customer 6	—	C7 starts playing, ending at 34+8 = 42
41	C3 leaves	Customer 6	—	C4 starts playing, ending at 41+11 =

				52
42	C7 leaves	Customer 6	__	Console 2 : ends at 42
52	C4 leaves	__	__	C6 starts; end time = 52+10 = 62; So, console 1 ends at 60

Submission Guidelines: (These are fixed for almost all your submissions in UG life.)

1. Create a directory with your 7-digit student id as its name.
2. You need to create separate files for the Array implementation and LL implementation, putting the main function in a separate file demonstrating your implementations.
3. Create a separate file for implementing task-2. The solution must use your own ArrayList/LL - based implementations. No other built in data structure can be used. You must write just one piece of code that will work on both Array and LL - implementations, the only difference will be the list object instantiations.
4. Put all the source files only into the directory created in step 1. Also create a readme.txt file briefly explaining the main purpose of the source files. 5
5. Zip the directory (compress in .zip format. Any other format like .rar, .7z etc. is not acceptable).
6. Upload the .zip file on Moodle in the designated assignment submission link. For example, if your student id is xx05xxx, create a directory named xx05xxx. Put only your source files (.c, .cpp, .java, .h etc.) into xx05xxx. Compress the directory xx05xxx into xx05xxx.zip and upload the xx05xxx.zip on Moodle.

DEADLINE: 10 December 2022, 23:55 pm

In case of any confusion, contact Md. Shahrar Fatemi (MSF), Email: shahrar007@gmail.com