Assignment 3: Queue

Task 1: Queue Data structure

You need to implement a Queue ADT as defined in Table 1. There is no restriction on the programming language. However, it might be fruitful for you if you implement your assignments in C++. Please do not use C++17. If you are using C++17 or Java/other languages, please bring your laptop for smooth evaluation.

Table 1: Queue ADT definition.

Fn #	Function	Param.	Ret.	After functions execution	Comment
	[before each function execution.]			<20, 23 , 12, 15>	The values were enqueued in the following order: 20, 23, 12, 15. So, 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) 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.
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

The implementation must support the various types of 'elements'. Use templates in C++ or something equivalent in the programming language you intend to use. You need to provide two different implementations, namely, Array Based (Arr) and Linked List (LL) Based Implementations. The size of the queue is only limited by the memory of the computing system, i.e., in case of Arr implementation, there must be a way to dynamically grow the queue size as follows: the queue 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 $X+1^{th}$ 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 item**. Static implementation would result in deduction of marks.

Appropriate constructors and destructors must be implemented. For the Arr implementation, the constructor takes the initial size of the array with a default size available as a constant. A second constructor for the Arr implementation takes a pointer to the already allocated array and starts using that as an empty queue.

You may implement extra helper functions but those will not be available for programmers to use. So, while using the queue implementations, one can only use the methods listed in Table 1. Please note that during evaluation, identical main functions will be used to check the functionality of both Arr 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 both the implementation except for the object instantiation part. Please follow the following input/output format.

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

Follow the same format provided for Assignment 1 (List). The memory chunk size parameter is not taken anymore in the first line.

Example input:

To be provided shortly.

Output Format:

Follow the same format provided for Assignment 1 (List).

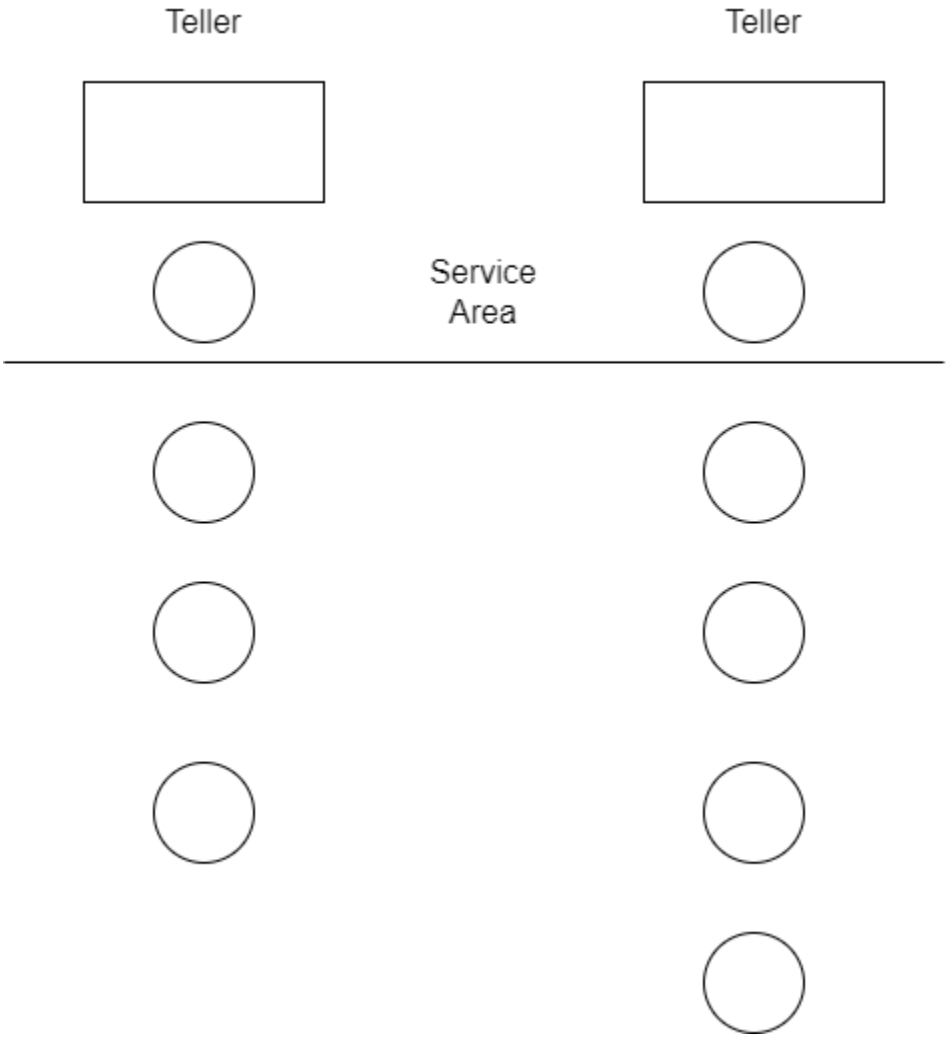
Expected Output of the example input:

To be provided shortly.

Task 2: Using the Queue (Bank)

In the infamous Golden Bank, only 2 booths remain open every weekday. As a lot of people take service from the bank for various reasons, often the queues in front of the booths become very large. However, the staff of the bank doesn't seem to care about the quality of service, so they are not going to increase the number of booths.

You can assume that there is one queue in front of each booth. People coming to the bank always take a position in the shortest queue. When a customer standing at the rear of a queue sees that he has n people in front of him and another queue has a total of (n-1) or fewer people at that time, he will switch to the other queue, i.e. leave the current queue and enter the other queue.



A sample scenario has been depicted in the above figure. Here the left queue contains 3 people and the right queue contains 4 people. A customer enters the service area to take service from the bank teller. However, the time taken by a bank teller to provide service to a customer (i.e. service time) depends on the type or volume of service the customer wants. This "service time" of a customer is unknown to everyone until he starts taking service. When he starts taking service, only the corresponding teller can calculate the service time; other customers in the queues have no idea about it.

To do the above simulation, you can only use queue data structure as implemented according to the queue ADT in table 1. Other than the queue data structure, you are only allowed to use normal arrays and variables for bookkeeping purposes.

Input format:

The first line contains n, the number of customers in the simulation.

Each of the next n lines contains 2 integers- time t when the customer enters the bank and service time s of that customer.

You can assume that there is no delay between a customer entering the bank and taking position in a queue. You can also assume that queue switching takes no time. If a queue switching and a new customer taking position in a queue appear to happen simultaneously, prioritize the new customer.

You can also assume that simulation starts at time 0.

Output format:

For each booth, report the time when the corresponding teller will finish providing service to all the customers.

Example input:

To be provided shortly.

Expected Output of the example input:

To be provided shortly.

In case of any confusion, contact Preetom Saha Arko (lecturer)

Email: preetomarko@gmail.com

Phone: +8801881927700