Consider the following statements:

**Array Queue<int> queue = new Array Queue();**

**int x, y;**

Show what is output by the following segment of code:

x = 4; y = 5; queue.enqueue(x); queue. enqueue(y); x = queue.front( ); queue.dequeue( ); queue. enqueue(x + 5); queue. enqueue(16); queue. enqueue(x); queue. enqueue(y - 3); system.out.println( "Queue Elements: "); while (!queue.isEmptyQueue())

{

system.out.println(queue.front() ); queue.dequeue();

}

System.out.println("Queue Elements:");

while (!queue.isEmptyQueue()) {

System.out.println(queue.front());

queue.dequeue();

}

1. What is the output of the following program segment?

linkedQueue<int> queue = new linkedQueue(); queue.enqueue(10); queue.enqueue(20); cout << queue.front() << endl; queue.dequeue(); queue.enqueue(2 \* queue.back()); queue.enqueue(queue.front()); queue. enqueue(5); queue. enqueue(queue.back() - 2); linkedQueue<int> tempQueue = new linkedQueue() ; tempQueue = queue; while (!tempQueue.isEmptyQueue())

{

system.out.println( tempQueue.front() ); tempQueue.dequeue();

}

system.out.println( queue.front() ); ------

1. Consider the following statements

: ArrayStack<int> stack = new ArrayStack(); ArrayQueue<int> queue = new ArrayQueue();

int x;

Suppose the input is:

14 8 14 22 64 35 19 32 7 11 13 30 -999

Show what is written by the following segment of code:

stack.push(0); queue.enqueue(0); system.out.println( x); while (x != -999)

{ switch (x % 4) { case 0: stack.push(x); break;

case 1: if (!stack.isEmptyStack())

{ system.out.println( "Stack Element = " ); system.out.println( stack.top()); stack.pop();

} else

system.out.println( "Sorry, the stack is empty." ); break; case 2: queue.enqueue(x); break; case 3: if (!queue.isEmptyQueue())

{

system.out.println( "Queue Element = " ); system.out.println( queue.front()); queue.dequeue();

} else

system.out.println( "Sorry, the queue is empty." ); break;

} //end switch system.out.println( x);

} //end while system.out.println( "Stack Elements: "); while (!stack.isEmptyStack())

{

system.out.println( stack.top() ); stack.pop(); }

system.out.println( "Queue Elements: "); while (!queue.isEmptyQueue())

{

system.out.println( queue.front() ); queue.dequeue();

}

1. Suppose that queue is a queueType object and the size of the array implementing queue is 100. Also, suppose that the value of queueFront is 50 and the value of queueRear is 99. a- What are the values of queueFront and queueRear after adding an element to queue? b- What are the values of queueFront and queueRear after removing an element from queue?

1. Suppose that queue is a queueType object and the size of the array implementing queue is 100. Also, suppose that the value of queueFront is 99 and the value of queueRear is 25. a- What are the values of queueFront and queueRear after adding an element to queue? b- What are the values of queueFront and queueRear after removing an element from queue?

If the value of queueFront is 99 and the value of queueRear is 25 in a queue with an array implementation of size 100, the following will occur:

a) After adding an element to the queue:

* If there is still space in the array to add more elements, the queueFront and queueRear values will remain the same, and the new element will be added at the next index after queueRear.
* Since the current values are queueFront = 99 and queueRear = 25, and the array size is 100, there is still space available to add an element to the queue.
* After adding an element, the queueRear value will be incremented to the next index in a circular manner.

So, after adding an element, the values of queueFront and queueRear will be:

* queueFront = 99 (remains the same)
* queueRear = 26

b) After removing an element from the queue:

* The queueFront value will be incremented to the next index in a circular manner.
* The queueRear value will remain the same if there are still elements in the queue or it will become -1 if the queue becomes empty.

Since the current values are queueFront = 99 and queueRear = 26, and there are elements in the queue, removing an element will only increment the queueFront value.

So, after removing an element, the values of queueFront and queueRear will be:

* queueFront = 0 (wraps around to the beginning of the array)
* queueRear = 26 (remains the same)

1. Suppose that queue is a queueType object and the size of the array implementing queue is 100. Also, suppose that the value of queueFront is 25 and the value of queueRear is 75. a- What are the values of queueFront and queueRear after adding an element to queue? b- What are the values of queueFront and queueRear after removing an element from queue?
2. Suppose that queue is a queueType object and the size of the array implementing queue is 100. Also, suppose that the value of queueFront is 99 and the value of queueRear is 99. a- What are the values of queueFront and queueRear after adding an element to queue? b- What are the values of queueFront and queueRear after removing an element from queue?

1. Write a function, **reverseQueue**, that takes as a parameter a queue object and uses a stack object to reverse the elements of the queue.

import java.util.Queue;

import java.util.Stack;

public class QueueReversal {

public static void reverseQueue(Queue<Integer> queue) {

// Create a stack to store the elements temporarily

Stack<Integer> stack = new Stack<>();

// Push all the elements from the queue to the stack

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

// Pop the elements from the stack and enqueue them back into the queue

while (!stack.isEmpty()) {

queue.offer(stack.pop());

}

}

}

1. Suppose an initially empty queue *Q* has performed a total of 32 enqueue operations, 10 first operations, and 15 dequeue operations, 5 of which returned null to indicate an empty queue. What is the current size of *Q*?

Given:

* Enqueue operations: 32
* First operations: 10
* Dequeue operations: 15 (with 5 null returns)

The enqueue operations increase the size of the queue, while the dequeue operations decrease the size. However, the null returns indicate that the dequeue operations were performed on an empty queue, which doesn't affect the size.

So, we can calculate the current size of Q as follows:

Net Size Increase = Total Enqueue Operations - Total Dequeue Operations  
= 32 - (15 - 5) // Subtracting the 5 null returns from the total dequeue operations  
= 32 - 10  
= 22

1. What values are returned during the following sequence of deque *(double ended queue)* ADT operations, on an initially empty deque? addFirst(3), addLast(8), addLast(9), addFirst(1), last( ), isEmpty( ), addFirst(2), removeLast( ), addLast(7), first( ), last( ), addLast(4), size( ), removeFirst( ), removeFirst( ).
2. Initially, the deque is empty.
3. addFirst(3): Deque becomes [3].
4. addLast(8): Deque becomes [3, 8].
5. addLast(9): Deque becomes [3, 8, 9].
6. addFirst(1): Deque becomes [1, 3, 8, 9].
7. last(): Returns 9.
8. isEmpty(): Returns false.
9. addFirst(2): Deque becomes [2, 1, 3, 8, 9].
10. removeLast(): Removes and returns 9. Deque becomes [2, 1, 3, 8].
11. addLast(7): Deque becomes [2, 1, 3, 8, 7].
12. first(): Returns 2.
13. last(): Returns 7.
14. addLast(4): Deque becomes [2, 1, 3, 8, 7, 4].
15. size(): Returns 6.
16. removeFirst(): Removes and returns 2. Deque becomes [1, 3, 8, 7, 4].
17. removeFirst(): Removes and returns 1. Deque becomes [3, 8, 7, 4].

# Good Luck