1. 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();

}

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() ); system.out.println(queue.back() );

3- 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?
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 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?
3. 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.
2. 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*?
3. 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( ).

# Good Luck

1- The output of the code segment would be:

Copy

Queue Elements:

4

5

9

16

4

2

2- The output of the code segment would be:

Copy

10

20

10

2

5

18

3- The output of the code segment would be:

ini

Copy

0

Stack Element =

0

Queue Element =

0

Stack Element =

14

Queue Element =

14

Stack Element =

8

Queue Element =

22

Stack Element =

14

Queue Element =

64

Stack Element =

35

Queue Element =

19

Stack Element =

32

Queue Element =

7

Stack Element =

11

Queue Element =

13

Stack Element =

30

Queue Element =

-999

Stack Elements:

30

11

7

32

19

35

64

14

8

Queue Elements:

14

22

64

35

19

32

7

11

13

30

4- After adding an element to the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 50

queueRear = 0

After removing an element from the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 51

queueRear = 0

5- After adding an element to the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 99

queueRear = 26

After removing an element from the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 0

queueRear = 26

6- After adding an element to the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 25

queueRear = 76

After removing an element from the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 26

queueRear = 76

7- After adding an element to the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 99

queueRear = 0

After removing an element from the queue, the values of queueFront and queueRear would be:

Copy

queueFront = 0

queueRear = 0

8- Here's an example implementation of the reverseQueue function in Java:

java

Copy

import java.util.\*;

public class Main {

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

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

// Push all elements from the queue to the stack

while (!queue.isEmpty()) {

stack.push(queue.poll());

}

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

while (!stack.isEmpty()) {

queue.offer(stack.pop());

}

}

public static void main(String[] args) {

Queue<Integer> queue = new LinkedList<>();

queue.offer(1);

queue.offer(2);

queue.offer(3);

queue.offer(4);

System.out.println("Original Queue: " + queue);

reverseQueue(queue);

System.out.println("Reversed Queue: " + queue);

}

}

Output:

json

Copy

Original Queue: [1, 2, 3, 4]

Reversed Queue: [4, 3, 2, 1]

9- The current size of Q can be calculated as follows:  
Total enqueue operations - Total dequeue operations that returned null.

Copy

Current size of Q = 32 - 5 = 27

10- The values returned during the sequence of deque (double-ended queue) ADT operations would be:

stylus

Copy

addFirst(3) -> Queue: [3]

addLast(8) -> Queue: [3, 8]

addLast(9) -> Queue: [3, 8, 9]

addFirst(1) -> Queue: [1, 3, 8, 9]

last() -> Returns 9

isEmpty() -> Returns false

addFirst(2) -> Queue: [2, 1, 3, 8, 9]

removeLast() -> Removes and returns 9, Queue: [2, 1, 3, 8]

addLast(7) -> Queue: [2, 1, 3, 8, 7]

first() -> Returns 2

last() -> Returns 7

addLast(4) -> Queue: [2, 1, 3, 8, 7,4]

size() -> Returns 6

removeFirst() -> Removes and returns 2, Queue: [1, 3, 8, 7, 4]

removeFirst() -> Removes and returns 1, Queue: [3, 8, 7, 4]

The values returned in this sequence of operations are:  
9, false, 2, 7, 6, 2, 1