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();

}

Based on the given code segment, let's trace the output step by step:

1. Initialize the queue:

ArrayQueue<int> queue = new ArrayQueue();

1. Initialize variables:

Int x, y;

1. Assign values:

X = 4;

Y = 5;

1. Enqueue x into the queue:

Queue.enqueue(x); // Queue: [4]

1. Enqueue y into the queue:

Queue.enqueue(y); // Queue: [4, 5]

1. Assign the front element of the queue to x:

X = queue.front(); // x = 4

1. Dequeue the front element:

Queue.dequeue(); // Queue: [5]

1. Enqueue (x + 5) Into the queue:

Queue.enqueue(x + 5); // Queue: [5, 9]

1. Enqueue 16 into the queue:

Queue.enqueue(16); // Queue: [5, 9, 16]

1. Enqueue x into the queue:

Queue.enqueue(x); // Queue: [5, 9, 16, 4]

1. Enqueue (y – 3) into the queue:

Queue.enqueue(y – 3); // Queue: [5, 9, 16, 4, 2]

1. Print the queue elements:

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

While (!queue.isEmptyQueue()) {

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

Queue.dequeue();

}

Output:

Queue Elements:

5

9

16

4

2

The code dequeues elements from the queue one by one and prints them. The final output Is the elements of the queue In the order they were dequeued: 5, 9, 16, 4, 2.

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() )

The given program segment seems to be a mix of C++ and Java code. I will assume that the code is written In C++ and provide the expected output:

1. Initialize the queue:

linkedQueue<int> queue = new linkedQueue();

1. Enqueue 10 into the queue:

Queue.enqueue(10); // Queue: [10]

1. Enqueue 20 into the queue:

Queue.enqueue(20); // Queue: [10, 20]

1. Print the front element of the queue:

Cout << queue.front() << endl; // Output: 10

1. Dequeue the front element:

Queue.dequeue(); // Queue: [20]

1. Enqueue (2 \* queue.back()) into the queue:

Queue.enqueue(2 \* queue.back()); // Queue: [20, 40]

1. Enqueue the front element of the queue:

Queue.enqueue(queue.front()); // Queue: [20, 40, 20]

1. Enqueue 5 into the queue:

Queue.enqueue(5); // Queue: [20, 40, 20, 5]

1. Enqueue (queue.back() – 2) into the queue:

Queue.enqueue(queue.back() - 2); // Queue: [20, 40, 20, 5, 3]

1. Initialize a temporary queue:

linkedQueue<int> tempQueue = new linkedQueue();

1. Assign the contents of queue to tempQueue:

tempQueue = queue; // tempQueue: [20, 40, 20, 5, 3]

1. Print the elements of tempQueue:

While (!tempQueue.isEmptyQueue()) {

Cout << tempQueue.front() << endl;

tempQueue.dequeue();

}

Output:

20

40

20

5

3

1. Print the front element of the original queue:

Cout << queue.front() << endl; // Output: 20

1. Print the back element of the original queue:

Cout << queue.back() << endl; // Output: 3

The final output of the program segment would be:

10

20

40

20

5

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())

Based on the given code segment and the provided Input, let's trace the output step by step:

1. Initialize stack and queue:

ArrayStack<int> stack = new ArrayStack();

ArrayQueue<int> queue = new ArrayQueue();

1. Initialize variable x:

Int x;

1. Suppose the input is: 14 8 14 22 64 35 19 32 7 11 13 30 -999
2. Push 0 into the stack:

Stack.push(0); // Stack: [0]

1. Enqueue 0 Into the queue:

Queue.enqueue(0); // Queue: [0]

1. Print the value of x (uninitialized variable):

System.out.println(x);

Output: It depends on the default value of x In the programming language. Let's assume It is 0.

1. Start the while loop with the condition x != -999:
2. Switch statement based on x % 4:

- For the first Input value (14):

- Case 2: Enqueue 14 into the queue

Queue.enqueue(x); // Queue: [0, 14]

* Print the value of x:

System.out.println(x);

Output: 14

1. Repeat the switch statement for the next input values (8, 14, 22, 64, 35, 19, 32, 7, 11, 13, 30):

- For each value, perform the corresponding action based on x % 4:

- Case 0: Push x into the stack

Stack.push(x); // Stack: [0, 8, 14, 22, 64, 35, 19, 32, 7, 11, 13, 30]

* Case 1: Print the top element of the stack if it's not empty, otherwise print "Sorry, the stack is empty."

System.out.println("Stack Element = ");

System.out.println(stack.top());

Stack.pop();

* Case 2: Enqueue x Into the queue

Queue.enqueue(x); // Queue: [0, 14, 8, 14, 22, 64, 35, 19, 32, 7, 11, 13, 30]

* Case 3: Print the front element of the queue if it's not empty, otherwise print "Sorry, the queue is empty."

System.out.println("Queue Element = ");

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

Queue.dequeue()

1. After processing all the input values, encounter -999 in the input:

* Case -999: The while loop condition x != -999 is false, exit the loop.

1. Print "Stack Elements: " and print the elements of the stack:

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

While (!stack.isEmptyStack()) {

System.out.println(stack.top());

Stack.pop();

}

Output: Stack Elements: 30 13 11 7 32 19 35 64 22 14 8 0

1. Print "Queue Elements: " and print the elements of the queue:

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

While (!queue.isEmptyQueue()) {

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

Queue.dequeue();

}

Output: Queue Elements: 0 14 8 14 22 64 35 19 32 7 11 13 30

The final output of the program segment would be:

14

Stack Element = 14

Stack Element = 8

Stack Element = 14

Queue Element = 0

Stack Element = 22

Stack Element = 64

Stack Element = 35

Stack Element = 19

Stack Element = 32

Stack Element = 7

Stack Element = 11

Stack Element = 13

Stack Element = 30

Stack Elements: 30 13 11 7 32 19 35 64 22 14 8 0

Queue Elements: 0 14 8 14 22 64 35 19 32 7 11 13 30

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.

1. What are the values of queueFront and queueRear after adding an element to

Queue?

1. What are the values of queueFront and queueRear after removing an a) After adding an element to the queue, the value of queueRear will be incremented by 1, and if it reaches the end of the array, it will wrap around to the beginning. The value of queueFront will remain unchanged.
2. Given the initial values:
3. queueFront = 50
4. queueRear = 99
5. After adding an element:
6. queueRear = (queueRear + 1) % 100
7. queueRear = (99 + 1) % 100
8. queueRear = 0
9. The new values of queueFront and queueRear are:
10. queueFront = 50
11. queueRear = 0
12. B) After removing an element from the queue, the value of queueFront will be incremented by 1, and if it reaches the end of the array, it will wrap around to the beginning. The value of queueRear will remain unchanged.
13. Given the initial values:
14. queueFront = 50
15. queueRear = 99
16. After removing an element:
17. queueFront = (queueFront + 1) % 100
18. queueFront = (50 + 1) % 100
19. queueFront = 51
20. The new values of queueFront and queueRear are:
21. queueFront = 51
22. queueRear = 99elemen
23. ,

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.

1. What are the values of queueFront and queueRear after adding an element to

Queue?

1. What are the values of queueFront and queueRear after removing an element

From queue?

* 1. After adding an element to the queue, the value of queueRear will be incremented by 1, and if it reaches the end of the array, it will wrap around to the beginning. The value of queueFront will remain unchanged.

Given the initial values:

queueFront = 99

queueRear = 25

After adding an element:

queueRear = (queueRear + 1) % 100

queueRear = (25 + 1) % 100

queueRear = 26

The new values of queueFront and queueRear are:

queueFront = 99

queueRear = 26

* 1. After removing an element from the queue, the value of queueFront will be incremented by 1, and if it reaches the end of the array, it will wrap around to the beginning. The value of queueRear will remain unchanged.

Given the initial values:

queueFront = 99

queueRear = 26

After removing an element:

queueFront = (queueFront + 1) % 100

queueFront = (99 + 1) % 100

queueFront = 0

The new values of queueFront and queueRear are:

queueFront = 0

queueRear = 26

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.

1. What are the values of queueFront and queueRear after adding an element to

Queue

* 1. After adding an element to the queue, the value of queueRear will be incremented by 1, and if it reaches the end of the array, it will wrap around to the beginning. The value of queueFront will remain unchanged.

Given the initial values:

queueFront = 25

queueRear = 75

After adding an element:

queueRear = (queueRear + 1) % 100

queueRear = (75 + 1) % 100

queueRear = 76 % 100

queueRear = 76

The new values of queueFront and queueRear are:

queueFront = 25

queueRear = 76

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

1. After adding an element to the queue, the value of queueRear will be incremented by 1, and if It reaches the end of the array, it will wrap around to the beginning. The value of queueFront will remain unchanged.

Given the initial values:

queueFront = 99

queueRear = 99

After adding an element:

queueRear = (queueRear + 1) % 100

queueRear = (99 + 1) % 100

queueRear = 100 % 100

queueRear = 0

The new values of queueFront and queueRear are:

queueFront = 99

queueRear = 0

1. After removing an element from the queue, the value of queueFront will be incremented by 1, and If it reaches the end of the array, it will wrap around to the beginning. The value of queueRear will remain unchanged.

Given the initial values:

queueFront = 99

queueRear = 0

After removing an element:

queueFront = (queueFront + 1) % 100

queueFront = (99 + 1) % 100

queueFront = 100 % 100

queueFront = 0

The new values of queueFront and queueRear are:

queueFront = 0

queueRear = 0

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.\*;

Public class QueueReverser {

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);

Queue.offer(5);

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

reverseQueue(queue);

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

}

}

Suppose an initially empty queue Q has performed a total of 32 enqueue operations,

10 first operations, and 15 dequeue operations, 5 of whichh returned null to indicate

An empty queue. What is the current size of Q

To determine the current size of queue Q, we need to consider the net effect of the enqueue and dequeue operations. The size of the queue Is the number of elements currently in the queue.

Given the information provided:

Total enqueue operations = 32

Total first operations = 10

Total dequeue operations = 15

Null dequeue operations = 5

Let's calculate the current size of Q:

Initially, the queue Is empty, so the size is 0.

For each enqueue operation, the size of the queue Increases by 1.

Total enqueue operations = 32

Queue size after enqueues = 0 + 32 = 32

For each successful dequeue operation, the size of the queue decreases by 1.

Total dequeue operations = 15 (excluding null dequeues)

Queue size after successful dequeues = 32 – 15 = 17

However, 5 of the dequeue operations returned null, indicating an empty queue. These operations do not affect the size of the queue.

Final queue size = Queue size after successful dequeues = 17

Therefore, the current size of Q Is 17.

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( ).

Let's go through the sequence of deque operations step by step:

1. addFirst(3): The deque is initially empty. Adding 3 at the front of the deque.

- Deque: [3]

2. addLast(8): Adding 8 at the end of the deque.

- Deque: [3, 8]

3. addLast(9): Adding 9 at the end of the deque.

- Deque: [3, 8, 9]

4. addFirst(1): Adding 1 at the front of the deque.

- Deque: [1, 3, 8, 9]

5. last(): Returning the value at the end of the deque, whichh is 9.

6. isEmpty(): Checking if the deque is empty. The deque Is not empty, so it returns false.

7. addFirst(2): Adding 2 at the front of the deque.

- Deque: [2, 1, 3, 8, 9]

8. removeLast(): Removing the value at the end of the deque, which is 9.

- Deque: [2, 1, 3, 8]

9. addLast(7): Adding 7 at the end of the deque.

- Deque: [2, 1, 3, 8, 7]

10. first(): Returning the value at the front of the deque, which Is 2.

11. last(): Returning the value at the end of the deque, whichh is 7.

12. addLast(4): Adding 4 at the end of the deque.

- Deque: [2, 1, 3, 8, 7, 4]

1. Size(): Returning the current size of the deque, which is 6.

14. removeFirst(): Removing the value at the front of the deque, which is 2.

- Deque: [1, 3, 8, 7, 4]

15. removeFirst(): Removing the value at the front of the deque, which is 1.

- Deque: [3, 8, 7, 4]

The values returned during the sequence of deque operations are as follows:

1. addFirst(3): No return value.

2. addLast(8): No return value.

3. addLast(9): No return value.

4. addFirst(1): No return value.

5. last(): Returns 9.

6. isEmpty(): Returns false.

7. addFirst(2): No return value.

8. removeLast(): Returns 9.

9. addLast(7): No return value.

10. first(): Returns 2.

11. last(): Returns 7.

12. addLast(4): No return value.

13. size(): Returns 6.

14. removeFirst(): Returns 2.

15. removeFirst(): Returns 1.