LabReport05

1. **Problem:**

This lab is focused on the implementation of a Process Scheduler that manages a queue of processes in Java. The problem is to create a program that can add, remove, and run processes, and manage the current process in the queue.

2. **Solution Description:**

The code provides a solution for the problem by using a linked list to implement the process queue, and defining methods for adding, running, and canceling processes, as well as for getting the current process and printing the process queue.

3. **Problems Encountered:**

There were no syntax, run-time or logic errors encountered while implementing the solution.

4. **How is a queue structured?**

A queue is structured as a collection of elements in which elements are inserted at one end and removed from the other end. It follows the First-In-First-Out (FIFO) principle.

5. **Describe when it is most appropriate to use a Queue in a problem.**

A queue is most appropriate to use in a problem when the order of processing or handling is important, and when there is a need to handle elements in the order they were added. It is useful when the oldest item needs to be processed first, as in a line of people waiting to be served.

6. **Detail two ways Queues are commonly used in computing.**

Two ways Queues are commonly used in computing are:

- In a print spooler, jobs that are waiting to be printed are placed in a queue and processed in the order in which they were received.
- In network packet processing, incoming packets are placed in a queue and are processed in the order they are received.

7. **What are the values in the queue following these actions? The answer must clearly show the "head" and "tail" values.**

   ENQUEUE Values 1, 2, 3, 4, 5 Queue: 1(head) 2 3 4 5(tail)

   DEQUEUE 3 times Queue: 4(head) 5(tail)

   ENQUEUE Values 25, 35, 45, 55 Queue: 4(head) 5 25 35 45 55(tail)

   DEQUEUE 1 time Queue: 5(head) 25 35 45 55(tail)

   ENQUEUE Value 22 Queue: 5(head) 25 35 45 55 22(tail)

8. **Resulting Arrays:**

- **ENQUEUE Values "j", "k":**
  front = 0, rear = 2, size = 2
  ["j", "k", NULL, NULL, NULL, NULL, "a", "b"]
  Tail

  ["j", "k", NULL, NULL, NULL, NULL, "a", "b"]
  Head

- **DEQUEUE 2 Times:**
  front = 2, rear = 2, size = 0
  ["j", "k", NULL, NULL, NULL, NULL, "a", "b"]
  Tail

  ["j", "k", NULL, NULL, NULL, NULL, "a", "b"]
  Head

- **ENQUEUE Values "m", "n", "o":**
  front = 2, rear = 5, size = 3
  ["j", "k", "m", "n", "o", NULL, "a", "b"]
  Tail

  ["j", "k", "m", "n", "o", NULL, "a", "b"]
  Head

- **DEQUEUE 3 Times:**
  front = 5, rear = 5, size = 0
  ["j", "k", "m", "n", "o", NULL, "a", "b"]
  Tail

  ["j", "k", "m", "n", "o", NULL, "a", "b"]
  Head

9. **Possible Solution:**
   Incrementing the index by 2 may cause the loop to go out of bounds if the queue length is odd, since the last element to be printed would be at an odd index. To avoid this, we can modify the loop condition to terminate when the index is equal to or greater than the tail.
   Corrected Code:

```
public void printEveryOtherValue() {
        int i = head;
        while (i != tail) {
            System.out.print(queue[i]);
            i = (i + 2) % queue.length;
            if (i != tail) {
                System.out.print(",");
            }
        }
        System.out.println();
    }
```

10. **Possible Solution:**
    The while loop condition should be while(temp != null) instead of while(temp.link != null), since we need to iterate through all nodes in the list, not just until the second-to-last node.
    Corrected Code:

```
public boolean isContained(int aValue) {
    ListNode temp = head;
    while (temp != null) {
        if (temp.data == aValue) {
            return true;
        }
        temp = temp.link;
    }
    return false;
}
```