Stacks and Queues (Part 1)

CS 112 - Recitation 2

Warm-Up Questions

1. What happens if my program calls **pop()** for an empty stack?

2. Why do we care about **resizing arrays**, when we have linked lists?

Warm-Up Questions

3. Are there **Java libraries** for stacks and queues?

4. What does the following code fragment do to the queue q?

```
Stack stack = new Stack();
while (!q.isEmpty())
        stack.push(q.dequeue());
while (!stack.isEmpty())
        q.enqueue(stack.pop());
```

Warm-Up Questions

5. What do the following **three code fragments** do?

Code 1: Suppose x is a linked-list node and not the last node on the list. What is the effect of the following code fragment?

```
x.next = x.next.next;
```

Code 2: Suppose that x is a linked list Node. What does the following code fragment do?

```
t.next = x.next;
x.next = t;
```

Code 3: Why does the following code fragment not do the same thing as in the previous question?

```
x.next = t;
t.next = x.next;
```

Q1 removeAfter() with LinkedLists

Write a method removeAfter() that takes a linked-list Node as argument and removes the node following the given one (and does nothing if the argument or the next field in the argument node is null).

```
public void removeAfter(Node node)
//Assume you have the following class variables:
private int size;
private Node first;
private String data;
```

```
private int size;
private Node first;
private String item;
public void removeAfter(Node node) {
        if (first == _______ || node == ________ ) { // check conditions here
                return;
        }
        Node current;
        for (current = first; current != null; current = current.next) {
                if (_______________________________) { // check condition here
                        if (current.next != null) {
                                // delete next node and decrease size

                }
        }
}
```

Q1 Solution (Note: there are more than 1 way of doing it)

```java
private int size;
private Node first;

public void removeAfter(Node node) {
    if (first == null || node == null) {
        return;
    }

    Node current;

    for(current = first; current != null; current = current.next) {
        if (current.item.equals(node.item)) {
            if (current.next != null) {
                current.next = current.next.next;
                size--;
            }
            break;
        }
    }
}
```

Q2 Implement Stack Using Queues

Using the **coding templa**te, implement a last in first out (LIFO) stack using only **two queues.** The implemented stack should support all the functions of a normal queue (**push, top, pop, and empty**).

Implement the MyStack class:

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Q2 Solution (Note: there are more than 1 way of doing it)

```java
import java.util.LinkedList;
import java.util.Queue;

class MyStack {
    private Queue<Integer> q1 = new LinkedList<>();
    private Queue<Integer> q2 = new LinkedList<>();
    private int top;

    /** Initialize your data structure here. */
    public MyStack() {

    }

    /** Push element x onto stack. */
    public void push(int x) {
        q1.add(x);
        top = x;
    }

    /** Removes the element on top of the stack and returns that element. */
    public void pop() {
        while(q1.size()>1){
            top = q1.remove();
            q2.add(top);
        }
        q1.remove();
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;

    }
```

```java
    /** Get the top element. */
    public int top() {
        return top;
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
        return q1.isEmpty();

    }

    Run | Debug
    public static void main(String[] args) {
        MyStack obj = new MyStack();
        obj.push(1);
        obj.push(2);
        obj.push(3);
        obj.pop();
        System.out.println(obj.top());
        obj.pop();
        obj.pop();
        System.out.println(obj.empty());
    }

}
/**
```

Q3 Convert Binary Number in a Linked List to Integer

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the decimal value of the number in the linked list.

Method header: public int getDecimalValue(ListNode head)

Ex.

Input: head = [1,0,1]

Output: 5

Explanation: (101) in base 2 = (5) in base 10

```
public int getDecimalValue(ListNode head) {
    int num = head.val;
    while (head.next != null) {
        // operation to convert the binary number into the decimal number
        num = ____________________________________________ // code here
        ____________________________________________________ // go to next node here
    }
    return num;
}
```

Q3 Solution

```java
public int getDecimalValue(ListNode head) {
    int num = head.val;
    while (head.next != null) {
        num = num * 2 + head.next.val;
        head = head.next;
    }
    return num;
}
```

Optional - Maximum Key in Linked Lists

Write a method max() that takes a reference to the first node in a linked list as an argument and returns the value of the maximum key in the list. Assume that all keys are positive integers, and return 0 if the list is empty

Assume you have the following class variables:

private int size;

private Node first;

Solution

```
public int max() {
    if(first == null) {
        return 0;
    }

    int maxValue = first.item;

    Node current;

    for(current = first.next; current != null; current = current.next) {
        int currentValue = current.item;
        if(currentValue > maxValue)
            maxValue = currentValue;
    }

    return maxValue;
}
```

Optional - Insert After - Linked Lists

Write a method insertAfter() htat takes two linked-list Node arguments and inserts the second after the first on its list (and does nothing if either argument is null). Assume there are the following class variables:

private int size;

private Node first;

Solution

```java
public void insertAfter(Node firstNode, Node secondNode) {
    if(first == null || firstNode == null || secondNode == null) {
        return;
    }

    Node current;

    for(current = first; current != null; current = current.next) {
        if(current.equals(firstNode)) {
            secondNode.next = current.next;
            current.next = secondNode;
            size++;
        }
    }
}
```

Good Work!

1. Got to dynrec.cs.rutgers.edu
2. Enter your code: HSPT