CSC172 LAB 6

STACKS & QUEUES

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_programming

Every student must hand in their own work, but every student must list the name of their lab partner if any on all labs.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

The textbook should present examples of the code necessary to complete this lab. However, collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

The goal of this lab is to gain familiarity with linked list implementations of stacks and queues.

2 Stacks

Stacks are specialized lists of data that only support inserting and deleting data from one end of the list. This means that stacks are LIFO data structures, where the Last In item is the First Out. This section of the lab will guide you through creating a stack using the singly linked list you made in Lab 2.

1. Begin this lab by typing in the code for the generic stack interface.

```
public interface Stack<AnyType> {
     public boolean isEmpty();
     public void push(AnyType x);
     public AnyType pop();
     public AnyType peek();
}
```

2. Copy your node and singly linked list classes from lab 2 into your new project. Rename the singly linked list class to `MyStack<AnyType>` and have it implement `Stack<AnyType>` instead of `SimpleLinkedList<AnyType>`. This will give you a good base from which to develop your stack.
3. Implement the `push()` method in your stack which will insert the given object into the front of your linked list (top of the stack). Your method must be able to insert objects in constant time. Duplicate objects should be allowed.
4. Implement the `pop()` method, which removes and returns the top (most recently inserted) item from the stack. Your method should return `null` if the stack is empty. Implement the `isEmpty()` method to check if the stack is empty and use it in your `pop()` method.
5. Sometimes we want to know what's on the top of the stack without having to pop it off the stack. Implement the `peek()` method to support this functionality.
6. Write a test program that instantiates an instance of your stack class and demonstrates that each of the required methods from the interface work. At a minimum, you should push a few objects onto the stack, check if the stack is empty, peek at the top item, and pop a few items off the stack.

3 Queues

Like stacks, queues are specialized linked lists that store data in a Fast In, First Out (FIFO) ordering, meaning that the first item to be inserted will be the first one to be removed. This section of the lab will have you implement a queue class using your doubly linked list from Lab 3 to store the data.

1. In a new file, type in the code for the generic queue interface.

```
public interface Queue<AnyType> {
   public boolean isEmpty();
   public void enqueue(AnyType x);
   public AnyType dequeue();
   public AnyType peek();
}
```

2. Copy your node and doubly linked list class from Lab 3 into your project. Rename the doubly linked list to `MyQueue<AnyType>` and have it implement `Queue<AnyType>` instead of `DoublyLinkedList<AnyType>`. This will give you a good base from which to develop

your queue.

3. Implement the `enqueue()` method which inserts an item into the end of your queue.
4. Implement the `dequeue()` method which removes and returns the item at the front of the list. If the list is empty, the dequeue method should return `null`. Implement the `isEmpty()` method to determine if the queue is empty and use it in the `dequeue()` method.
5. Implement the `peek()` method that returns, but does not delete, the first item in the queue.
6. Duplicate the same tests you did for your stack with the queue (i.e. insert the same items, check if the list is empty, peek, and remove the items in the same order). Note the how both data structures store the same data but in different orderings.

4 Hand In

Hand in the source code from this lab at the appropriate location on the BlackBoard system at my.rochester.edu. You should hand in a single zip file (compressed archive) containing your source code, README, and OUTPUT files, as described below.

1. A plain text file named README that includes your contact information, your partner's name, a brief explanation of the lab (A one paragraph synopsis. Include information identifying what class and lab number your files represent.), and one sentence explaining the contents of all the other files you hand in.
2. All Java source code files representing the work accomplished for this lab. All source code files should contain author and partner identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

5 Grading

Your grade will be determined based on the correct implementation of the following (total 90%):

Stacks

5% generic interface, class, and constructor for the stack

10% push() method, with the necessary modifications to support inserting in the front of the list.

10% pop() method, with the necessary modifications to support removing from the front of the list.

5% peek() method

5% isEmpty() method

10% appropriate test cases in your test program

Queues

5% generic interface, class, and constructor for the queue

10% enqueue() method, with the necessary modifications to support inserting at the end of the list.

10% dequeue() method, with the necessary modifications to support removing from the front of the list.

5% peek() method

5% isEmpty() method

10% appropriate test cases in your test program

The README and OUTPUT files account for the remaining 10% of the lab grade.