First Exam
CS 1102 Computer Science 2

Spring 2018

Thursday February 22, 2018
Instructor Muller

KEY

Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name **on the back** of this exam.

This is a closed-notes and closed-book 35-minute exam. Computers, calculators, and books are prohibited. Work on this exam should stop at 9:35.

- Partial credit will be given so be sure to show your work.

- Feel free to write helper functions if you need them.

- **Please write neatly.**

Problem	Points	Out Of
1		10
2		10
Total		20

1. (10 Points) This problem involves implementing a `remove` operation for a singly-linked mutable `List<T>` ADT. The signature of the function is `void remove(T item)`. If `a` is the following list:

```
+-----+---+    +----------+      +----------+      +----------+
|first: o-+--->|info: "C" |      |info: "B" |      |info: "A" |
+-----+---+    |next:  o--+----->|next:  o--+----->|next:  o--+-+
               +----------+      +----------+      +----------+ =
```

then `a.remove("B")` should remove the leftmost occurrence of `"B"` resulting in the following change.

```
+-----+---+    +----------+      +----------+      +----------+
|first: o-+--->|info: "C" |      |info: "B" |      |info: "A" |
+-----+---+    |next:  o--+--+   |next:  o--+>  +->|next:  o--+-+
               +----------+  |   +----------+   |  +----------+ =
                             +------------------+
```

You'll find an API and harness code on one of the attached sheets. If the list is empty or if `item` is not in the list, then no change should take place. **Hint**: The possibility of `item` being the first element in the list is a special case. Otherwise, consider running two pointers `p` and `q` down the list pointing to consecutive nodes.

Answer:

```
public void remove(T info) {
  Node<T> p = this.first, q;

  if (p == null) return;

  q = p.getNext();

  if (p.getInfo().equals(info)) {
    this.first = q;
    return;
  }

  while (q != null && !q.getInfo().equals(info)) {
    p = q;
    q = q.getNext();
  }
  if (q == null) return;

  p.setNext(q.getNext());
  return;
}
```

2. (10 Points) The Queue<T> ADT has the following API

```
public interface Queue<T> {
  void enqueue(T item);
  T dequeue();
  int size();
  boolean isEmpty();
}
```

This problem is concerned with implementing an extension of a queue, a ReversibleQueue<T>.

```
public interface ReversibleQueue<T> extends Queue<T> { void reverse(); }
```

You'll find harness code on one of the attached sheets. It provides most of an implementation of a ReversibleQueue<T> using *composition*. This problem involves implementing the reverse operation. For example, if a is the reversible queue A, B, C with A in front, then a.reverse() should result in the reversible queue C, B, A with C in front. If the queue is empty then no change should take place.

You can get up to 8 of the 10 points available by using (without implementing) a Stack<T> with implementing class StackC<T>. But to get the full 10 points, the Java keyword **new** should not appear in your solution.

Answer:

```
public void reverse() {              // 8 Point Version
  Stack<T> s = new StackC<T>();
  while(!this.isEmpty()) s.push(this.dequeue());
  while(!s.isEpmty()) this.enqueue(s.pop());
}

public void reverse() {              // 10 Point Version
  T item;
  if (!q.isEmpty()) {
    item = q.dequeue();
    this.reverse();
    q.enqueue(item);
  }
}
```

Code for Problem 1

```
public interface List<T> {
  void pushLeft(T info);
  T popLeft();
  boolean isEmpty();
  void remove(T info);
}

public class LinkedList<T> implements List<T> {

  Node<T> first;

  private class Node<T> {
    T info;
    Node<T> next;

    private Node(T info, Node<T> next) {
      this.info = info;
      this.next = next;
    }

    private T getInfo() { ... }
    private Node<T> getNext() { ... }
    private void setNext(Node<T> next) { ... }
  }

  public LinkedList() { this.first = null; }

  public boolean isEmpty() { ... }
  public void pushLeft(T info) { this.first = new Node(info, this.first); }
  public T popLeft() { ... }

  public void remove(T info) { YOUR CODE HERE }
}
```

Code for Problem 2

```
public class ReversibleQueueC<T> implements ReversibleQueue<T> {

  private Queue<T> q;

  public ReversibleQueueC() { this.q = new LinkedQueue(); }

  public void enqueue(T item) { q.enqueue(item); }
  public T dequeue() { return q.dequeue(); }
  public int size() { return q.size(); }
  public boolean isEmpty() { return q.isEmpty(); }

  public void reverse() { YOUR CODE HERE }
}
```