Department of Electrical and Computer Engineering
University of Puerto Rico
Mayagüez Campus

ICOM 4035 – Data Structures
Spring 2012
Midterm Exam # 2

Name: ____

Student Number: ____

Section: ____

Instructions:

1. Write your name on all pages of this exam now!

2. You have two hours to complete this exam. Use your time wisely. Do not spend too much time on a problem, when you can work on others.

3. There are 4 problems for a maximum score of 100 points. Complete as many problems as you can, and earn as many points as possible

GOOD LUCK!

Scores

1	/25
2	/25
3	/25
4	/25
Total	/100

Problem 1. (25 points) Big-O and General Course Concepts

Consider the following code fragment that uses a stack (numbers on the left simply illustrate line number of the code) :

```
1. Stack<String> S = new DynamicArrayStack<String>(20);
2. S.push(“Joe”);
3. S.push(“Ron”);
4. S.pop();
5. S.push(S.pop());
6. S.push(“Jil”);
7. S.push(S.pop());
```

a) **(5 pts)** Draw the resulting stack (in vertical fashion) if we apply operation: S.push(“Xi”) after line 7.

b) (**5 pts**) Draw the resulting stack (in vertical fashion) if we apply operation: S.push(S.top()) after line 7.

Problem 1 (Continuation)

For each problem, indicate the complexity (Big-O bound) for the function or code fragment. JUSTIFY YOUR ANSWER.

c) **(5 pts)**

```
public void clearQueue(Queue<Integer> Q){
     while (Q.size() >=1){
          Q.dequeue();
     }
}
```

d) **(5 pts)**

```
public void printList(List<String> L, int numSpaces){
     for (String s : L){
          for (int i=0; i < numSpaces *2 ; ++i){
               System.out.print();
          }
          System.out.println(s);
     }
}
```

e) **(5 pts)**

```
public void printList2(List<String> L, int numSpaces){
     int n = 4;
     for (String s : L){
          for (int i=0; i < n ; ++i){
               System.out.print();
          }
          System.out.println(s);
     }
}
```

Problem 2. (25 pts) Understanding and Using Singly Links Lists

Use the material discussed in class about singly linked lists to answer the following questions:

a) **(10 pts)** Write a static, no-member method named `isSorted()` which receives as parameter a singly linked list of String. The method returns true if the list is sorted in increasing order, or false otherwise. Recall that method `compareTo()` can be used to compare two strings is Java.

```
public static boolean isSorted(SinglyLinkedList<String>  L) {
```

Problem 2 (Continuation)

b) **(15 pts)** Write static non-member method named `removeDuplicates`(), which receives as parameter a Singly Linked List. The method removes all duplicate values from the linked list, keeping only the first copy found for each one and preserving the same relative order among the elements. For example, if list L = {Joe, Ned, Ron, Ned, Joe, Amy}, then `removeDuplicates(L)` converts L into {Joe, Ned, Ron, Amy}.

```
public static void removeDuplicates(SinglyLinkedList<E> L){
```

Problem 3. (25 pts) Understanding and Using Stack/Queue ADTs

Consider the implementation of stacks and queues discussed in class:

a) **(5 pts)** Write a static non-member function `deleteFromStack()` that removes all the copies of an element `obj` from a `StaticArrayStack`. **After completion, all copies of obj are removed and the relative order of the remaining elements on the stack is the same as it was before the operation was called.**

```
public void deleteFromStack(StaticArrayStack <E>, E obj)
```

Problem 3. (Continuation)

b) **(10 pts)** Write a static non-member function `deleteFromQueue()` that removes all copies of an element `obj` from a `Queue`. **After completion, all copies of obj are removed and the relative order of the remaining elements on the queue is the same as it was before the operation was called.**

```
public void deleteFromQueue(Queue <E>, E obj) {
```

Problem 3. (Continuation)

c) (**10 pts**) Write a static, non-member function `copyQueue()` that creates and returns fresh copy of a `SinglyLinkedListQueue<E> src`. Notice that this is a non-member function, so you cannot access any internal state in the `queue`. **Moreover, once the function ends, the parameter queue must have the same elements and the same structure that it begins with.**

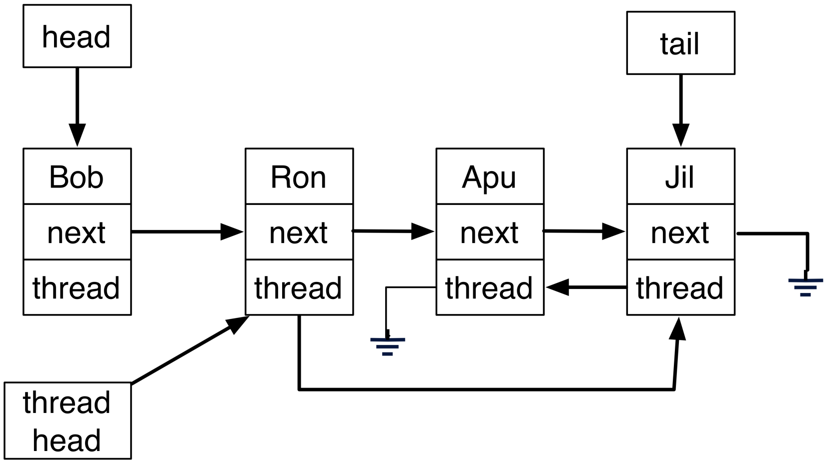
```
public SinglyLinkedListQueue<E>
     copyQueue(SinglyLinkedListQueue<E> src){
```

Problem 4. (25 pts) Understanding and Implementing Singly Linked Lists.

A threaded singly linked list is a singly linked list with nodes that form two chains:

1) The chain formed by the `next` reference in each node.
2) The chain formed by the `thread` reference in each node.

Thus, a node has three fields: data, next, and thread. Field `thread` is just a reference to the next node in the thread chain. The following diagram illustrates a threaded list this:



The threaded singly linked list can be implemented with a class named `ThreadedList<E>.` This class has four private fields:

1) `head` - reference to first node in the chain formed by `next` references.
2) `tail` – reference to last node in the chain formed by `next` references.
3) `threadHead` – reference to first node in the chain formed by the `thread` references.
4) `currentSize` – number of elements in the list (this are the elements in the chain formed by `next` references).

You received a handout with a partial implementation of the ThreadedList that you can use to analyze the private fields and the class that implements the nodes for the ThreadedList.

Using all this information, answer the following questions.

Problem 4 (Continuation)

a) (**5 pts**) Implement a **member** method named `threadCount()` that returns the number of elements currently linked by the thread chain.

```
public int threadCount(){
```

Problem 4 (Continuation)

b) (**10 pts**) Implement a member method named addToThread() which adds a new element to the list. The new element gets inserted at the end of the chain formed by the next references, and at the beginning of the chain formed by the thread references. You need to consider if the list is empty. The number of elements in the list must be increased.

```
public void addToThread(E obj){
```

Problem 4 (Continuation)

c) **(10pts)** Write a member method named `eraseFromThread`() which both: a) removes the first occurrence of an object from the chain of thread references, and b) also removes the **same element** from the chain formed by the next references. You need to consider if the `threadHeader` is null, or if it becomes null after the erase. The method returns true if the operation is completed or false is the object is not found. The number of elements in the list must be decreased. ***Beware of duplicate values, since you need to erase the element that was in the thread chain.***

```
public boolean eraseFromThread(E obj){
```