University of Toronto
CSC148 – Introduction to Computer Science Summer 2001

Mid Term Test – Section L5101

Duration: 50 minutes
Aids allowed: none

Make sure that your examination booklet has 8 pages (including this one). Write your answers in the spaces provided. Write legibly. You may use page 8 for rough work (tear it off, if you like). If you require more space to answer a question, write on the back of the previous page, and indicate in the answer space where you answer is.

Surname: _____

Given name(s): _____

Student #: _____

TA's name: _____ (please circle below)

W. Andreopoulos E. Xia S. Lim J. Listgarten

Please note that if you write the mid term in pencil, you will ***not*** be allowed to submit a remark request.

0.	/ 2
1.	/ 6
2.	/ 10
3.	/ 12
4.	/ 10
Total:	/ 40

Question 0: Administrative

[2 marks]

Write your surname and student number at the top of every page (except page 1) of this test.

Question 1: Design by Contract

[3 x 2 marks each – 6 marks total]

Below are methods with their comments. There is something wrong with each comment. Briefly explain what's wrong with each one (if there is more than one problem, pick the one you feel is most important).

```
/* Uses iteration (not recursion) to calculate the
 * factorial of d.
 * pre: d > 0. */
static int factorial(int d);
```

Answer:

The fact that the method uses iteration is a private internal detail

```
/* Finds the biggest int in Vector d */
static int biggest(Vector d);
```

Answer:

Behaviour in event that d contains no ints is not specified (which is important, given that Vectors can't hold primitives).

```
/* Finds v in the Binary Search Tree rooted at r.
 * Pre: r points to a valid BST which contains v */
static boolean search(BSTNode r, Object v);
```

Answer:

What the method returns is not specified.

Question 2: Interface Implementation & Exceptions

[10 marks]

Below is the code for ArrayQueue from the slides, slightly altered (changes are bolded for convenience). Your task is to complete an Iterator class that will be returned by the iterator() method which has been added to the ArrayQueue class. You will complete the code for AQIterator below.

```
public class ArrayQueue implements Queue {
   // The number of elements in me
   public int size;

   // The elements in me, stored in contents[0..size-1]
   public Object[] contents;

   /* Representation Invariant:
    * If size is 0, I am empty, and size=0; otherwise:
    *    contents[0] is the head, contents[size-1] is the tail,
    *    contents[0..size-1] contains the Objects in the order they
    *    were inserted
    */

   // Append o to me.
   public void enqueue(Object o) {...}

   // Remove and return my front Object.
   // Precondition: I am not empty
   public Object dequeue() {...}

   // Return my front Object.
   // Precondition: I am not empty.
   public Object head() {...}

   // Return the number of Objects in me.
   public Object size() {...}

   // Return an Iterator representation of this Queue.
   public Iterator() { return new AQIterator(this); }
}
```

You may assume that you have a class called *NoSuchElementException* at your disposal. Code:

```
class NoSuchElementException extends Exception {}
```

Note that an Iterator does not make a copy of any data, or change the underlying ArrayQueue in any way.

Note also that this is not a test of how well you know the Iterator interface. Some features of the interface have been left out. Just complete the methods that have method headers below.

```
class AQIterator implements Iterator {

   ArrayQueue q;
   int numLeft=0;
   int next;

   // q is the ArrayQueue for which this AQIterator shall
   // serve as Iterator
   AQIterator(ArrayQueue q) {

      numLeft = q.size;
      this.q = q;
      next = 0;

   }// AQIterator()

   // Returns true if there are elements which have not yet
   // been returned.
   public boolean hasNext() {

      return numLeft > 0;









   }// hasNext()


   // Returns the next object in the underlying class.  If there is
   // no next element, then a NoSuchElementException is thrown.
   public Object next() throws NoSuchElementException {

      if (numLeft == 0) {
         // Assertion: there is no next element
         throw new NoSuchElementException();
      }
      // Assertion: numLeft > 0

      numLeft--;
      return q.contents[next++];









   }// next()
} //AQIterator
```

Question 3: Linked Lists

[12 marks total]

This is the Linked List node class that you will be using for the rest of this question.

```
class LLNode {
   String data;
   LLNode next;
   LLNode prev;

   LLNode(String d, LLNode n, LLNode p) {
      data = d; next = n; prev = p;
   }
}
```

(a) Tracing

[4 marks]

Below is the method foo. Draw a diagram (using boxes & arrows, not the memory model) of what this linked list will look like when the method is run up to the comment. You may use the last page of the exam booklet for scrap work – draw only the final diagram below.

```
void foo() {
   LLNode a = new LLNode("fun!",null,null);
   LLNode b = new LLNode("linked",a,a);
   LLNode c = new LLNode("are",b,a);
   a.next = b;
   c.prev = new LLNode("lists",c,a.next);
   c.next = b.next;
   b.next = c.prev;
   c.prev.prev.prev.prev = b.next.next;
   // DRAW HERE
   System.out.println("That wasn't easy...");
}
```

Drawing:

The drawing is of a doubly-linked list with nodes (in order): "linked", "lists", "are", "fun!". Note that fun!.next points to "linked" and linked.prev points to "fun!".

(b) Linked List Operations

[8 marks]

You are to write a method that will move the largest value in a linked list to the head of the list. The method header is below – complete the code.

Helpful reminder: Strings contain a method called "compareTo" which is used to compare two Strings. Specifically:

compareTo(Object o) returns:
The value `0` if the argument is a string equal to this string; a value less than `0` if the argument is a string greater than this string; and a value greater than `0` if the argument is a string less than this string.

```
/* Finds the largest value in the Linked List rooted at
head and moves it to the front of the list.  If more than
one String in the list has an equal, highest value, the one
closest to the head is moved. */

static void SortOne(LLNode head) {

   if (head == null) {
      // if there's no list, there's nothing to sort.
      return;
   }

   LLNode biggest = head;
   LLNode current = head;

   // find the biggest node
   while (current.next != null) {
      if (biggest.data.compareTo(current.data) < 0) {
         smallest = current;
      }
   }

   // If the head is the biggest item, it's already sorted
   if (biggest == head) {
      return;
   }

   // Insert a new node to hold the old head
   head.next.prev = new LLNode(head.item,head.next,head);
   head.data = biggest.data;
   head.next = head.next.prev;

   // Remove the old node which held the biggest
   biggest.prev.next = biggest.next;
   if (biggest.next != null) {
      biggest.next.prev = biggest.prev;
   }
}
```

Question 4: Trees

[10 marks total]

Answer each question in the space provided.

a. You have a tree made up of *n* nodes. What is the maximum height of this tree?
[1 mark]

n

b. You have a reasonable search algorithm for finding the smallest value in a binary tree. If you have a binary tree with height *h* that is made up of *n* nodes, what is the smallest number of nodes that the algorithm will have to visit in order to be sure it has found the smallest value?
[1 mark]

n

c. You have learned that singly linked lists are trees. Are doubly linked lists also trees? Why or why not?
[3 marks]
No – one of the properties of a tree is that every node (except the root) has exactly one parent. In a doubly-linked list, every node has two parents (the one before and the one after).

d. Below is the code for method "mystery". Write a complete (external) method comment for "mystery".

This is the BSTNode class mystery uses:

```
/* A node for a Binary Search Tree */
class BSTNode { int data; BSTNode right; BSTNode left; }
```

This is the mystery method:

```
static int mystery(BSTNode r) {
   BSTNode temp = r;
   while (temp.right!=null) {
      temp=temp.right;
   }
   return temp.data;
}
```

External comment for this method:
[5 marks]
// Finds and returns the largest value stored in the BST rooted at r.
// pre: r != null

You may use this page for your rough work.