

University of Toronto

Erindale College

CSC 148F—Introduction to Computer Science, Fall 2003

Mid Term Test

Duration: 50 minutes

Aids allowed: None

Make sure that your examination booklet has 6 pages (including this one).

Write your answers in the spaces provided.

Write legibly.

Family Name:

Tutorial Time (circle one)

First Name:

Thursday 11

Student Number:

Thursday 1

Marking

1. / 15

2. / 20

3. / 15

Total / 50

Question 1 [15 marks] *ArrayList subclass*

Write a class that extends `ArrayList`. Your class must contain the following operation:

Given an index the operation exchanges the object in the location specified by the index with the object in the previous location (the index minus one). If the index is out of range the operation leaves the class unaltered. As with the index in `ArrayList` the first location is 0 the last `size() - 1`. The operation is implemented as a method called `exchange` that has one `int` parameter the index for the exchange location.

If you have to write more than 15 lines of Java for this question you have taken the wrong approach. Think again.

Question 2 [20 marks] Memory model and queue implementation

Find below a `Queue` interface, a `ArrayQueue` class that implements `Queue` and a `TestQueue` class that includes a `main` method that tests the `ArrayQueue` implementation.

a)

Trace the execution of the program. Show the state of the memory model after the first time line 4 in the `dequeue` method is executed. The line numbers are displayed as a comment to the right of each line.

```
/**
 * A Queue with fixed capacity.
 */
public class ArrayQueue implements Queue
{
    private int size;
    private Object[] contents;

    /** ArrayQueue with capacity for n elements. */
    public ArrayQueue(int n)
    {
        contents = new Object[n];           // 1
        size = 0;                           // 2
    }

    /** Append o to me. */
    public void enqueue(Object o)
    {
        contents[size] = o;                 // 1
        size++;                             // 2
    }

    /** Remove and return my front element. */
    public Object dequeue()
    {
        Object head = contents[0];          // 1
        for (int i =0; i != size; i++)      // 2
        {
            contents[i] = contents[i+1];    // 3
        }
        size--;                             // 4 <- Here.
        return head;                        // 5
    }

    /** Returns my front element. */
    public Object head()
    {
        return contents[0];                 // 1
    }

    /** Return the number of emements in me. */
    public int size()
    {
        return size;                       // 1
    }
}

/**
 * A Queue interface.
 */
public interface Queue
{
    /** Append o to me. */
    void enqueue(Object o);

    /** Return my front element */
    Object dequeue();

    /** Return the number of elements in me */
    int size();

    /** Returns my front element. */
    public Object head();
}

/**
 * Midterm test trace question
 */
public class TestQueue
{
    public static void main(String[] args)
    {
        int size = 5;                       // 1
        Object temp;                         // 2
        Queue q = new ArrayQueue(size);     // 3
        q.enqueue(new Integer(45));         // 4
        q.enqueue(new Double(76.5));        // 5
        temp = q.dequeue();                  // 6
        System.out.println(temp);           // 7
        q.enqueue(new String("Fred"));      // 8
        System.out.println(q.size());        // 9
        temp = q.head();                     // 10
        System.out.println(temp);           // 11
    }
}
```

Method Stack

Static Space

Object Space

b)
State the output that results from running the program to completion.

Question 3 [15 marks] *Linked list question*

Trace the execution of the following Java code and answer the questions.

```
/**
 * List node class that store ints.
 */
public class IntNode
{
    private int value;
    private IntNode link;

    /**
     * Constructor
     * @param v int value stored.
     */
    public IntNode(int v)
    {
        value = v;
    }

    /**
     * Set the link for this node.
     * @param l link stored.
     */
    public void setLink (IntNode l)
    {
        link = l;
    }

    /**
     * Get the link stored in this node.
     * @return the link in this node.
     */
    public IntNode getLink()
    {
        return link;
    }

    /**
     * Get the data stored in this node.
     * @return the data in this node.
     */
    public int getValue()
    {
        return value;
    }
}

/**
 * Midterm test linked list class
 */
public class IntList
{
    private IntNode front = null;
    private int size = 0;

    /**
     * Add an integer to the list.
     */
    public void add(int i)
    {
        IntNode temp = front;
        front = new IntNode(i);
        front.setLink(temp);
        size ++;
    }

    /**
     * Print some of the list items.
     */
    public void print()
    {
        IntNode temp = front;
        while(temp != null)
        {
            System.out.println(temp.getValue());
            temp = temp.getLink().getLink();
        }
    }
}

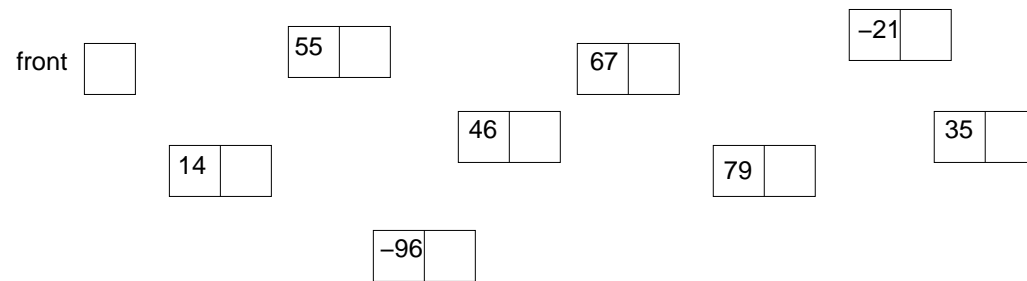
/**
 * Midterm list question
 */
public class ListTester
{
    public static void main(String[] args)
    {
        IntList l = new IntList();

        l.add(46);
        l.add(-21);
        l.add(79);
        l.add(14);
        l.add(67);
        l.add(-96);
        l.add(35);
        l.add(55);

        l.print();
    }
}
```

a)

After all of the **adds** are executed indicate with arrows how the nodes below are linked together.



b)

State the output that will result from executing the program.

c)

What would happen during execution if there were nine calls to **add** instead of eight?