1.00

Introduction to Computers and Engineering Problem Solving

Final / December 13, 2004

Name: **Email Address:** **TA:** **Section:**	**Solution**

You have 180 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary files have already been imported.

Good luck.

Question	*Points*
Question 1	/ 10
Question 2	/ 15
Question 3	/ 15
Question 4	/ 10
Question 5	/ 10
Question 6	/ 25
Question 7	/ 15
Total	/ 100

Question 1. True / False + Multiple Choice + Short Answer (10 Points)

1. Every node in a Binary Tree must have 2 children.

TRUE **FALSE**

2. A single stream can be used as both an input stream and an output stream.

TRUE **FALSE**

3. There can be several `catch` blocks in a single `try/catch` structure.

TRUE FALSE

4. A method can throw more than one class of `Exception`.

TRUE FALSE

5. The following Java source code will compile.

```
public class FinalExam
{
      private int a;

      public static int printA()
      {
            System.out.println("a = " + a);
      }
}
```

TRUE **FALSE**

6. An iterator of a `HashMap` visits its elements in the order they are inserted.

TRUE **FALSE**

7. Consider a `HashTable` that does not have any collisions. Suppose there are n items to be stored and m slots in the `HashTable`. Searching for an element in the `HashTable` is:

a. $O(n)$
b. **$O(1)$**
c. $O(m)$
d. $O(\log n)$

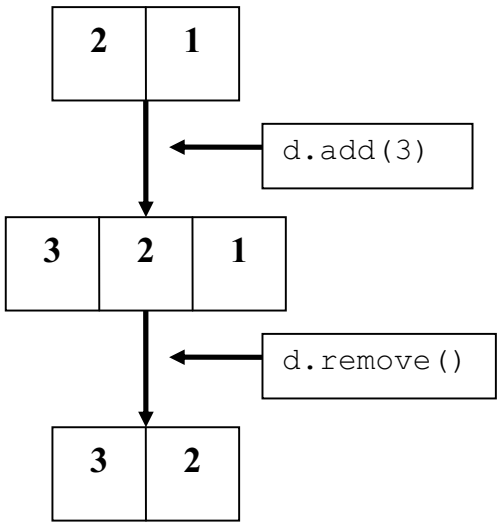
8. Consider following Java method.

```
public static void findOrder(int n)
{
      int result = 0;
      for (int i = 0; i < n; i++)
      {
            for(int j = i; j < n; j++)
            {
                  result++;
            }
      }
}
```

The above code runs in:

a. $O(n)$
b. $O(1)$
c. **$O(n^2)$**
d. $O(\log n)$

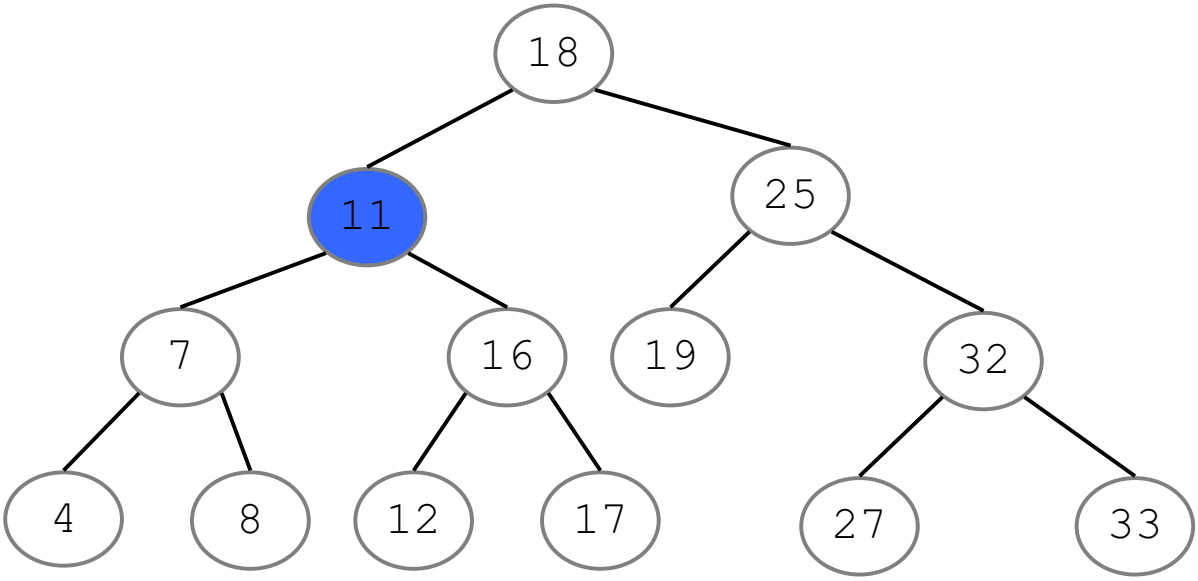
9. Consider an instance of data structure illustrated below. It has an `add()` method to add an element and `remove()` method to remove an element. The figure below shows an example of this data structure (referred to as `d`) initially, after `d.add(3)` is called, and after `d.remove()` is called.



Which type of data structure best describes this data structure?

STACK **QUEUE**

10. Consider following Binary Search Tree.



a. What's the First Key of this Binary Search Tree?

Node 4

b. If node 11 was to be deleted from the tree, which node would be the successor that replaces node 11?

Node 12

Question 2. LinkedList (15 Points)

In this question, you are going to write a static method, `findAverage()`, which takes an instance of the Java Collections Framework `LinkedList` class that holds only `Integer` objects and finds the average of contained `Integer` values. Here is the method signature:

`public static double findAverage(LinkedList list)`

For instance, let's suppose you have a `LinkedList` object that contains `Integer(4)`, `Integer(6)`, `Integer(3)`, `Integer(2)`, `Integer(5)`, and `Integer(4)`. The `findAverage()` method should find the average of the contained six `Integer` values (4.0 in this case) and return it.

Complete the `findAverage()` method. Assume that only `Integer` objects are contained in the `LinkedList` object. Your solution must use the `ListIterator` object to traverse the instance of `LinkedList`.

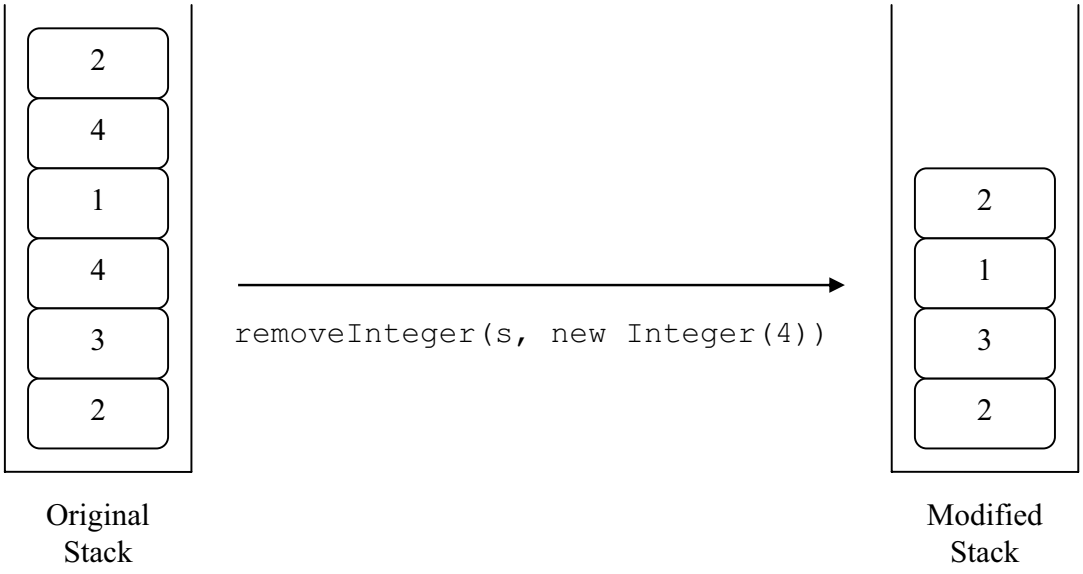
```
public static double findAverage(LinkedList list)
{
    ListIterator iter = list.listIterator();

    // Your Code Here


    double sum = 0.0;

    while (iter.hasNext())
    {
          sum += ((Integer)iter.next()).intValue();
    }

    return sum / list.size();




}
```

Question 3. Stack (15 Points)

In this question, you are going to write a static method, `removeInteger()`, which removes certain `Integer` object/objects from a stack. This method takes as arguments an instance of `IntegerStack`, which is a stack data structure specialized for `Integer` objects, and an `Integer` object that specifies the value of the `Integer` objects you want to remove from the stack. Here is the method signature:

```
public static void removeInteger(IntegerStack s, Integer i)
```

For instance, let's suppose you have a variable `IntegerStack s`, which refers to the instance of `IntegerStack` that contains several `Integer` objects, and you would like to remove all the `Integer` objects that contain `int` value of 4.



As you can see from the diagram above, after invoking the `removeInteger()` method, the instance of `IntegerStack` no longer contains any instances of the `Integer(4)` object.

To complete the method, you need to use some of the following public methods of `IntegerStack` class:

- **`public IntegerStack()`**
  `// Constructor of IntegerStack class`

- **`public void push(Integer i)`**
  `// Pushes an Integer object to the stack`

- `public Integer pop() throws EmptyStackException`
  `// Pops the Integer object added last`

- `public int size()`
  `// Returns the size of the stack`

- `public boolean isEmpty()`
  `// Checks whether the stack is empty or not`

Complete the `removeInteger()` method.

```
public static void removeInteger(IntegerStack s, Integer i)
{
    IntegerStack temp = new IntegerStack();

    while (!s.isEmpty())
    {
          Integer current = s.pop();
          if (!current.equals(i))
          {
                temp.push(current);
          }
    }

    while (!temp.isEmpty())
    {
          s.push(temp.pop());
    }
}
```

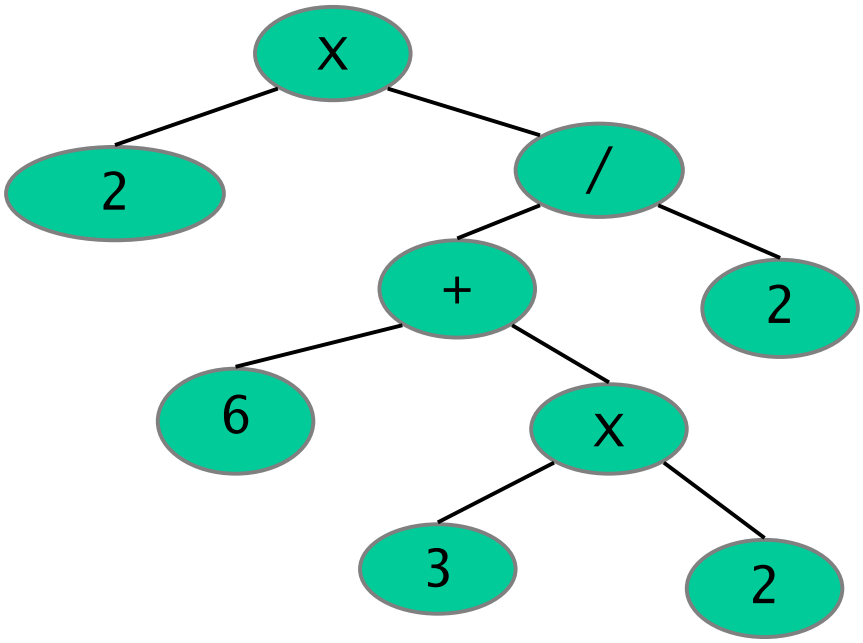
Question 4. Stream (10 Points)

In Streams lecture, we discussed three types of data formats that you can use to read and write information: **text**, **binary data**, and **object**. For a Java application that generates output data and distributes that data over the Internet, which data format would you use? Why?

When an application distributes data over the Internet, the portability of the data is very important. That is, the data should be viewable / readable on different platforms or operating systems. Therefore, for the current problem, text format is the best choice.

Question 5. Tree (10 Points)

Consider the following tree.



Write the results from a Postfix (Post-Order), Prefix (Pre-Order), and Infix (In-Order) traversal for the above tree. Note that the nodes of above tree hold either numerical values or arithmetic operators, where the **X** denotes a multiplication operation.

Postfix (Post-Order) traversal:	**2 6 3 2 X + 2 / X**
Prefix (Pre-Order) traversal:	**X 2 / + 6 X 3 2 2**
Infix (In-Order) traversal:	**2 X 6 + 3 X 2 / 2**

Question 6. Streams (25 Points)

Read the following code carefully.

- The class `StudentData` stores the students' name and three exam grades for 1.00 students. Its `toString()` method returns a string of name and grades, each of them separated by the tab character.

- The class `StreamExample` takes input and output file names as arguments to its constructor.
  - Its `parseFile()` method opens the input file for reading. It reads a line of information, calculates the average from the student's three grades, and creates a `StudentData` object from that with the average as the final data item. The `StudentData` object is then stored in an `ArrayList list`.
  - Its `writeFile()` method opens the output file for writing. It uses the `toString()` method of `StudentData` to write information for each of the objects in the list.

- The format of the input file is as shown in the example below:

```
Ana         80          70          60
David       99          89          98
Mary        89          40          60
Fedrick     79          49          78
Chris       78          67          56
Elena       59          98          78
```

- The format of the output file should be:

```
Ana         80.0        70.0        60.0        70.0
David       99.0        89.0        98.0        95.333
Mary        89.0        40.0        60.0        63.0
Fedrick     79.0        49.0        78.0        68.667
Chris       78.0        67.0        56.0        67.0
Elena       59.0        98.0        78.0        78.333
```

```
public class StudentData
{
      private String name;
      private double[] data;

      public StudentData(String n, double[] m)
      {
            name = n;
            data = m;
      }
```

```
	public String toString()
	{
	// See Part 1 below
	}
}


public class StreamExample
{
	String inFile, outFile;
	ArrayList list = new ArrayList();

	public StreamExample(String name1, String name2)
	{
		inFile = name1;
		outFile = name2;
	}

	public void parseFile()
	{
	// See Part 2 below

	}

	public void writeFile()
	{
	// See Part 3 below
	}
}
```

Part 1) Complete the `toString()` method of `StudentData` such that it creates a String object that has the format shown for the output file above.

```
public String toString()
{
	String s = "";

	s = s + name + "\t";
	for (int i = 0; i < data.length; i++)
		s = s + data[i] + "\t";

	return s;

}
```

Part 2) Complete the `parseFile()` method of `StreamExample` to read the input file. The method should:

- Open the file for reading
- Read each line of data while more data exists
- Using the `StringTokenizer`, break each line into 4 tokens: name, grade1, grade2, and grade3. Each student has exactly 3 grades, and you do not need to check for misformatted data. A `StringTokenizer` is much simpler than a `StreamTokenizer`. The `StringTokenizer` method `nextToken()` will return the next token as a `String` whether it is a word or number.
- For each student ,
  - calculate the average of his or her grades using grade1, grade2, and grade3
  - store the 3 grades and the average in a `double` array of 4 elements
  - create an object of type `StudentData` using name, grades, and average
  - add the object to the `ArrayList list`.

```
public void parseFile() {
try {
      FileReader fReader = new FileReader(inFile);
      BufferedReader reader = new BufferedReader(fReader);
      String temp = reader.readLine();

      while (temp != null) {
         StringTokenizer tokenizer = new StringTokenizer(temp);
         String name = tokenizer.nextToken();

         // Your Code Here

            double[] m = new double[4];

            m[0] = Double.parseDouble(tokenizer.nextToken());
            m[1] = Double.parseDouble(tokenizer.nextToken());
            m[2] = Double.parseDouble(tokenizer.nextToken());

            m[3] = (m[0] + m[1] + m[2]) / 3.0;

            StudentData s = new StudentData(name, m);

            list.add(s);



         temp = reader.readLine();
      }
      reader.close();
}
catch (FileNotFoundException e) { /* Implementation hidden */ }
catch (IOException e) { /* Implementation hidden */ }
}
```

Part 3) Complete the `writeFile()` method of `StreamExample` to write the data from the `ArrayList list` to the output file. The method should:

- Open the file for writing
- Loop over all the data in `list` to write each object data on a new line (refer to the format of the output file shown above).

```
public void writeFile()
{
```

```
try
{

      FileWriter fWriter = new FileWriter(outFile);
      BufferedWriter writer = new BufferedWriter(fWriter);

      // Your Code Here

      for (int i = 0; i < list.size(); i++)
      {
            StudentData s = (StudentData) list.get(i);
            writer.write(s.toString());
            writer.newLine();
      }









      writer.close();

}
catch (IOException e) { /* Implementation hidden */ }
catch (Exception e) { /* Implementation hidden */ }
```

```
}
```

Question 7. Hashing (15 Points)

Part 1) A 1.00 TA wants to store the data related to students in his class in a hash table. Assume all students in the class get a loaner laptop and each loaner laptop has a unique serial number between 00 and 99. The TA wants to use the serial number of the loaner laptop of a student as the key to hash. Suppose the hash table that he wants to use has 20 slots and the maximum enrollment in the class is restricted to 15. What is the maximum load factor for the hash table?

Load factor = 15/20 = 0.75

Part 2) Suppose the TA decides to add the two digits comprising the key to find out the slot in the hashing table to put the student into. Suppose there are 6 students in the class with their loaner laptop serial numbers being 89, 82, 79, 34, 56, and 65. What slots are allotted in the hash table? Is there a collision?

Student	Key	Slot in table
1	89	17
2	82	10
3	79	16
4	34	7
5	56	11
6	65	11

There is a collision as the keys of both student 5 and student 6 hashes to 11.

Part 3) If the serial numbers of a loaner laptop are equally likely to have any value between 0 and 99, does this hashing scheme distribute the keys uniformly in slots? Which slot is the least likely to face collision?

The proposed scheme does not hash the keys uniformly. For example a hash value of 11 is more likely than a hash value of 1, and thus the chance of collisions is more for the slot 11 than for slot 1.

A hash value of 19 is not possible at all (The maximum number that you can get by adding two digits is 18). Thus the slot 19 would always be empty. Also, only the key 00 can have a hash value of 0. Hence the slot 0 would also not have any collision. Similarly slot 18 would not have any collision. Thus slots 19, 0 and 18 would not have any collision.