I, the undersigned, do hereby affirm that the work contained in this exam is solely my own, and that none of the results were achieved by cheating. This includes using automated tools to generate answers, stealing the answers off the web, etc. Please do the work yourself.

Name: ____________________________________________
     (printed legibly)

Signature: ____________________________________________

Section: ____________________________________________

Student ID ____________________________________________
     (9-digit number)
RECURSION

Consider the following program that computes a geometric series using a recursive method:

```java
public class FinalExamRecursion {

    public static void main(String[] args) {
        System.out.println("Converges to " + series(2, 3));
    }

    public static float series(long numer, long denom) {

        // Compute term
        float result = (float) numer / (float) denom;

        if (numer < 0 || denom < 0)
            // Base case
            return 0.0f;
        else
            // Recursive computation
            result += series(numer * 2, denom * 3);
        return result;
    }
}
```

1. Describe the geometric series computed by the code above:

   **Computes 2/3 + 4/9 + 8/27 + 16/81 + ... ~= 2.0**

2. Does the program terminate? Why? After approximately how many recursive calls:

   **Yes, the program terminates when the long overflows, on the 39th recursive call.**
INHERITANCE and POLYMORPHISM

What does the program shown below print? Fill in the blanks:

```java
public class InheritanceProgram {
    public static class A {
        protected int w = 10, x = 20;
        protected int sum() { return w + x; }
    }
    public static class B extends A {
        protected int y = 30;
        protected int sum() { return w + x + y; }
    }
    public static class C extends B {
        protected int w = 0; // Override!
        protected int z = 40;
        protected int sum0() { return w + x + y + z; }
        protected int sum1() { return super.sum() + z; }
    }
    public static void main(String[] args) {
        A a = new A();
        B b = new B();
        C c = new C();
        A poly = new B();
        System.out.println(a.sum()); // Question 3
        System.out.println(b.sum()); // Question 4
        System.out.println(c.sum0()); // Question 5
        System.out.println(c.sum1()); // Question 6
        System.out.println(poly.sum()); // Question 7
    }
}
```

3. 30
4. 60
5. 90
6. 100
7. 60
Show what the program shown below would print.

```java
public class ArrayListProgram {
    public static void main(String[] args) {
        Character cArray[] = {'A', 'E', 'O'};
        ArrayList<Character> list =
            new ArrayList<>(Arrays.asList(cArray));
        list.add(2, 'I');
        list.add(4, 'U');
        System.out.println(list.toString()); // Question 8
        list.remove(3);
        for (int i = 0; i < list.size(); i++)
            list.set(i, Character.toLowerCase(list.get(i)));
        System.out.println(list); // Question 9
        Iterator<Character> it = list.iterator();
        while (it.hasNext())
            if (it.next() < 'e')
                it.remove();
        System.out.println(list); // Question 10
        System.out.println(list.contains('u')); // Question 11
    }
}
```

8. [A, E, I, O, U]
9. [a, e, i, u]
10. [e, i, u]
11. true
REGULAR EXPRESSIONS

In regular expressions, [0-9] means any digit, [A-Za-z] means any letter, ? means 0 or 1 occurrences, + means 1 or more occurrences, * means 0 or more occurrences, {2,4} means between 2 and 4 occurrences, {3} means exactly 3 occurrences, . matches any character, and \. matches a period.

12. Given the lines of text shown below, put a Yes to the left of lines that are matched and No to the left of lines that are not matched by the regular expression shown below, using a search tool such as the Linux grep command. Remember that grep matches the line only if any contiguous part of it matches the entire regular expression. For full credit, identify which characters in each line are matched by underlining them. The first line is completed as an example:

Regular Expression: [ !@#$% ]{2} [0-9]+ [A-Z]* .[a-z]{3}

<table>
<thead>
<tr>
<th>Matched?</th>
<th>Line of Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>#@!987654321ABC.txt</td>
</tr>
<tr>
<td>Yes</td>
<td>@!987654321ABC.txt</td>
</tr>
<tr>
<td>No</td>
<td>#987654321ABC.txt</td>
</tr>
<tr>
<td>No</td>
<td>987654321ABC.txt</td>
</tr>
<tr>
<td>No</td>
<td>!$A.txt</td>
</tr>
<tr>
<td>Yes</td>
<td>!$0A.txt</td>
</tr>
<tr>
<td>Yes</td>
<td>!$0AB.txt</td>
</tr>
<tr>
<td>No</td>
<td>!$0AB..txt</td>
</tr>
<tr>
<td>Yes</td>
<td>@#$0A.jpg</td>
</tr>
<tr>
<td>Yes</td>
<td>@#$0A.jpeg</td>
</tr>
<tr>
<td>No</td>
<td>$%92ABCD-txt</td>
</tr>
<tr>
<td>Yes</td>
<td>$%92ABCD.txt</td>
</tr>
</tbody>
</table>

13. List three strings that match the regular expression: [%@&]{3} [0-9] * [A-F]

%%%&012345-A
%&@.ABCDEF
&&&12~A
EXPRESSION TREES

The expression tree shown below is correct for the following expression:

\[(14 \% 6) + (12 / 2 * 3)\]

14. (4 points) Show the postfix expression represented by the tree, with spaces between each token, and no leading or trailing spaces.

14 6 % 12 2 / 3 * +

15. (4 points) Show the prefix expression represented by the tree, with spaces between each token, and no leading or trailing spaces.

+ % 14 6 * / 12 2 3

16. (2 points) What does the expression evaluate to, assuming integer math and the normal Java order of operations, which are of course reflected in the prefix and postfix forms and the tree?

20
**BINARY SEARCH TREES**

17. Draw the binary search tree that results from adding the following integers, in exactly the order specified: [55, 22, 66, 11, 44, 88, 99, 77, 33]

![Binary Search Tree with Nodes 55, 22, 66, 11, 44, 88, 99, 77, 33]

18. Draw the binary search tree from the previous question, assuming that Node 55 has been removed:

![Binary Search Tree with Nodes 22, 66, 88, 44, 11, 33, 77, 99]
HASHING and PARALLEL PROGRAMMING

19. (2 points) With linear probing, in which entry in the left hash table will the colliding object with key 15 be stored? Which entry would be used for quadratic probing? The table size is 13, and the hash function is a simple modulo of the integer key. Also list the indices that are tried unsuccessfully.

**LINEAR PROBING:** Index 6 - tries 3 (+1), 4 (+2), 5 (+3) unsuccessfully, then 6(+4) successfully.

**QUADRATIC PROBING:** Index 6 - tries 3 (+1) unsuccessfully, then 6(+4) successfully.

<table>
<thead>
<tr>
<th>Index</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>1</td>
<td>53</td>
</tr>
<tr>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>95</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>47</td>
</tr>
<tr>
<td>9</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>25</td>
</tr>
</tbody>
</table>
20. Use the following B+ tree to answer. For each question, you should start with the tree shown below (i.e. the questions are not cumulative). In these problems, redistribution refers to the act of moving a data entry from one leaf node to an existing sibling of that leaf node.

![B+ tree](image)

Draw the resulting B+ tree when 81 and 61 are added to the tree (with no redistribution).
21. Use the following B+ tree to answer. For each question, you should start with the tree shown below (i.e. the questions are not cumulative). In these problems, redistribution refers to the act of moving a data entry from one leaf node to an existing sibling of that leaf node.

Draw the resulting tree when 25 is deleted from the tree (with redistribution).
Note: In all algorithms, the tie breaking rule is to select the node with the lowest value, all other things being equal!

22. Please list the order of nodes for a depth-first search (DFS) and breadth-first search (BFS) starting at vertex A. All nodes should be listed:

**DFS:**  A B C D E

**BFS:**  A B C E D
23. Using the above directed graph, for each of the sequences given, tell if it DFS, BFS, or TS sorted.

<table>
<thead>
<tr>
<th></th>
<th>DFS</th>
<th>BFS</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SACBDE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SCDEAB</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SABDEC</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SCDBEA</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
24. Write a method that reads a file into an `ArrayList<LinkedList<String>>`, where each entry in the `ArrayList` is a `LinkedList` of the tokens found on that line in the file. Discard empty lines, and split the tokens based on white space. **HINT:** You can use `String.split`, `StringTokenizer`, or another `Scanner` to parse tokens from each line, but the latter is probably the easiest.

```java
public ArrayList<LinkedList<String>> readFile(String filename) {
    ArrayList<LinkedList<String>> list = new ArrayList<>();
    try {
        Scanner reader = new Scanner(new File(filename));
        while (reader.hasNextLine()) {
            String line = reader.nextLine().trim();
            if (!line.isEmpty()) {
                LinkedList<String> tokens = new LinkedList<>();
                Scanner parser = new Scanner(line);
                while (parser.hasNext()) {
                    tokens.add(parser.next());
                }
                list.add(tokens);
            }
        }
    } catch (IOException e) {
    }
    return list;
}
```
25. Write the method to get an element from a LinkedList, based on the index. The data structure for the LinkedList is the same one used for the assignment on implementing LinkedLists, as shown. Include the bounds check on the index, and throw an `IndexOutOfBoundsException` if the index is out of range, as specified by the list size variable. Return null if the index is not found.

**Hint:** You must traverse the LinkedList to implement the get method.

```java
// Node data structure
public class Node {
    public E element;
    public Node next;
}

// Head (first) pointer
private Node listHead;

// Tail (last) pointer
private Node listTail;

// Current size
private int listSize;

// Get method size
public E get(int index) {

    // Bounds check
    if (index < 0 || index >= listSize)
        throw new IndexOutOfBoundsException();

    // Search list
    int index = 0;
    for (Node n = listHead; n != null; n = n.next) {
        if (index++ == index) {
            E element = n.element;
            return element;
        }
    }

    // Should never happen
    return null;
}
```