University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Computer Science Division

Autumn 2010 **Jonathan Shewchuk**

CS 61B: Midterm Exam I

This is an open book, open notes exam. Electronic devices are forbidden on your person, including cell phones, iPods, headphones, and PDAs. Turn your cell phone off and leave all electronics, except your laptop, with the instructor, or risk getting a zero on the exam. **Do not open your exam until you are told to do so!**

Name:_____

Login:_____

Lab TA:_____

Lab day and time:_____

Do not write in these boxes.

Problem #	**Possible**	**Score**
1. Errors and keywords	8	
2. Heap and stack	9	
3. Matrix transpose	8	
Total	25	

Problem 1. (8 points) **Program errors and Java keywords.**

a. (3 points) The following method causes one compiler error. Point out the bug and show how to fix it.

Once the compiler error is fixed, point out which line sometimes throws a run-time exception, and show how to fix it by changing *only that line*, and without changing the line's intended effect.

Now that you've debugged the program, what does it do?

```
public static void main(String[] a) {
  for (i = 0; i < a.length; i++) {
    if (a[i].length() > a[i - 1].length() && i != 0) {
      a[i] = a[i] + 9;
    }
    System.out.println(a[i]);
  }
}
```

b. (3 points) Three of the following six lines of code cause compiler errors. Identify them and explain why.

- `assert 2 == 3;`
- `Object[] oo = new Integer[20];`
- `Object o = new Comparable();`
- `String s = ((Comparable) o).toString();` where `o` is an object
- `double d = ((Object) "string").length();`
- `int i = Integer.parseInt(super.toString());` in the method `main`

c. (1 point) A field in a Java `interface` is usually preceded by the keywords _____. (If you leave them out, Java acts as if you'd put them in.)

d. (1 point) The three Java keywords that can alter the flow of program execution in a loop are `break`, _____, and _____.

Problem 2. (9 points) **The Heap and the Stack.**

Suppose we execute `BigNode.main` in the following code. Draw the stack and heap at the moment when there are five stack frames: `main`, `enlist`, `enlist`, `addString`, and a constructor about to return. Specifically, draw a box-and-pointer diagram with a box for every stack frame, local variable, object, and field in memory at that moment. Include the stack frames for all methods in progress, and illustrate which entities are inside those stack frames. Don't forget "`this`". Entities on the stack should be on the left-hand side of the page, and entities on the heap should be on the right-hand side.

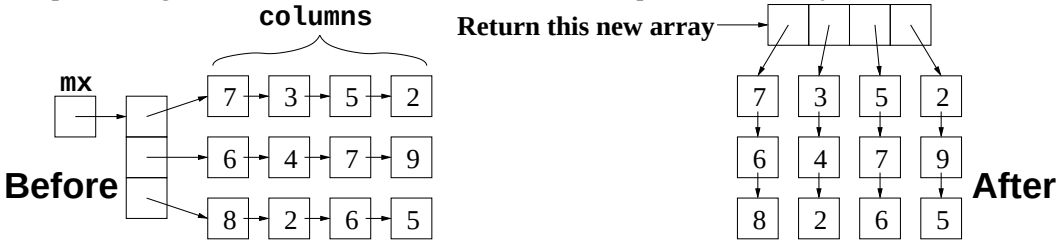
```
public class Node {
  public String item;
  public Node next;

  public Node() {
    item = "Default";
  }

  public Node addString(String s) {
    return new BigNode(s);
  }

  public void enlist(String[] ss, int index) {
    next = addString(ss[index]);
    if (index + 1 < ss.length) {
      next.enlist(ss, index + 1);
    }
  }
}
```

```
public class BigNode extends Node {
  public String item2;

  public BigNode(String s) {
    item2 = s;
  }

  public Node addString(String s) {
    Node n = new Node();
    n.item = s;
    return n;
  }

  public static void main(String[] args) {
    String[] strings =
       {"Please enjoy", "Midterm I"};
    (new Node()).enlist(strings, 0);
  }
}
```

```
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
                                                |
Draw stuff on the stack on the left side        | Draw stuff on the heap on the right
```

Problem 3. (8 points) **Transposing a Matrix Represented by Linked Lists.**

Write a method called `transpose` that performs a matrix transpose in the `SListNode` class below. The input parameter `mx` is an array of singly-linked lists, each representing a row of a matrix. (I.e. `mx` is a two-dimensional array, except it's an array of lists instead of an array of arrays). The input parameter `columns` is the length of every linked list (no error checking required). Your job is to return an array of singly-linked lists, each representing a **column** of the same matrix. Here's a picture of what you should do:



The new matrix must use the **same listnodes** as the old one, so `transpose` destroys the input matrix to make the new one. However, you **must construct a new array** of length `columns` to return. (The matrix might have a different number of columns than rows.)

There is no `SList` class; just an `SListNode` class. You should manipulate `next` references directly, but do **not** construct any new `SListNode`s or change the `item` field in any `SListNode`.

```
public class SListNode {
  public int item;
  public SListNode next;

  public SListNode[] transpose(SListNode[] mx, int columns) {
```

```
  }                                  Check here if your answer |---|
}                                    is continued on the back. |___|
```