Name: _________________________________________

USC loginid (e.g., ttrojan): ________________________

Midterm Exam 2
CS 455, Fall 2011

November 2, 2011

There are 7 problems on the exam, with 65 points total available. There are 8 pages to the exam, including this one; make sure you have all of them. There is also a separate double-sided one-page code handout. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and circling your final answers and crossing out any other answers you changed your mind about (though it’s fine if you show your work).

Put your name and loginid at the top of the exam. Please read over the whole test before beginning. Good luck!

Remote DEN students only: Do not write on the backs of pages. If additional space is needed, ask proctor for additional blank page(s), put your name on them, and attach them to the exam.

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>8 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>7 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>4 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>10 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 7</td>
<td>20 pts.</td>
<td></td>
</tr>
</tbody>
</table>

TOTAL 65 pts.
Problem 1 [8 pts.]

Part A. Show the results of inserting the following six keys that have the following hash values into the hash table below that starts out empty, has 10 buckets (indices 0 through 9), and uses chaining. Do the insertions in the order the keys are shown (a-f).

<table>
<thead>
<tr>
<th>Key</th>
<th>hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Mary</td>
<td>9</td>
</tr>
<tr>
<td>b. Joe</td>
<td>1</td>
</tr>
<tr>
<td>c. Ted</td>
<td>9</td>
</tr>
<tr>
<td>d. Bob</td>
<td>9</td>
</tr>
<tr>
<td>e. Dan</td>
<td>4</td>
</tr>
<tr>
<td>f. Sue</td>
<td>7</td>
</tr>
</tbody>
</table>

For each of the following lookups from the table above, after the insertions are done, give the sequence of keys that the target key would have to be compared with to do the lookup

Part B. lookup Tom; Tom hashes to 0

Part C. lookup Ted; Ted hashes to 9
Problem 2 [7 pts.]

Part A. Consider the following balanced binary search tree, whose root is shown nearest the top of the page (i.e., it's not a sideways tree):

For each of the following lookups from the tree shown above, give the sequence of keys that the target key would have to be compared with to do the lookup.

Part A. lookup Dan

Part B. lookup Pete

Part C. lookup Hank

Part D. For the tree shown above, what's the maximum number of keys we would have to compare a target key with to do a lookup of that target key?
Problem 3 [10 pts.]
Briefly describe two possible internal representations one might use for the Java `Map` interface. For each one, give the worst-case big-O time to do each of the `Map` operations listed below. For full credit, one of the representations must be different than the ones already provided in the Java library.

```
// get the value associated with this key
// if no entry with this key exists, returns null
public ValueType get(KeyType key)

// inserts an entry or updates the value that goes with an existing
// key. Returns the old value associated with the key, or null
// if this key was not previously in the map
public ValueType put(KeyType key, ValueType value)

// remove the entry with this key
// returns the associated value or null if not found
public ValueType remove(KeyType key)
```

Description of representation 1:

Big-O times for representation 1:

```
_________get   _________put   _________remove
```

Description of representation 2:

Big-O times for representation 2:

```
_________get   _________put   _________remove
```
Problem 4 [4 points]

Background. We saw an example of the benefit of programming in terms of an interface, rather than a concrete class, in our Concord class, where we had an instance variable of type Map, and we could change from using a HashMap implementation to using a TreeMap implementation by changing just one line of code (the part that changed is shown in italics):

```
Map<String, Integer> concord = new HashMap<String, Integer>();
Map<String, Integer> concord = new TreeMap<String, Integer>();
```

When we did this, all of our other Concord code still worked: any class that implements the Map interface must support the get, put, and iterator operations that were used in that program, so none of the rest of the program had to be changed.

Similarly, the ArrayList and LinkedList classes are both implementations of an interface called List. Some of the operations of this common interface are

- get(i)
- set(i, newValue)
- add(i, newValue)
- remove(i)

where i denotes a position in the list, and can range from \([0..\text{list.size()}-1]\).

Question. Unlike with a Map or Set, what is the big drawback of programming in terms of the common List interface for code that could use either an ArrayList or LinkedList?
Problem 5 [6 points]
Consider the following partially completed method, `match`, that uses a `Stack` to determine whether a string of parentheses consists of matched sets of parentheses, no matter how deep the parentheses are nested. Complete the method so it always returns one of three possible values (denoted by named integer constants defined elsewhere in the class):

- **EXTRA_LEFT** means that there was a left parenthesis with no matching right parenthesis
- **EXTRA_RIGHT** means that there was a right parenthesis with no matching left parenthesis
- **MATCHED** means that the string had only matched sets of parentheses

Here are some examples of input strings and the corresponding result:

<table>
<thead>
<tr>
<th>parens</th>
<th>match (parens) returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>(((())))</td>
<td>MATCHED</td>
</tr>
<tr>
<td>(())(())()</td>
<td>MATCHED</td>
</tr>
<tr>
<td>(())(())()</td>
<td>EXTRA_LEFT</td>
</tr>
<tr>
<td>(())(()())()</td>
<td>EXTRA_RIGHT</td>
</tr>
<tr>
<td>()(()))</td>
<td>EXTRA_RIGHT</td>
</tr>
<tr>
<td>()()()</td>
<td>MATCHED</td>
</tr>
<tr>
<td>()</td>
<td>EXTRA_RIGHT</td>
</tr>
</tbody>
</table>

(returns this because third paren was not preceded by a matching left paren)

```java
public static int match(String parens) {
    Stack<Character> s = new Stack<Character>();
    for (int i = 0; i < parens.length(); i++) {
        int ch = parens.charAt(i);
        if (ch == '(') {
            s.push(ch);
        } else if (ch == ')') {
            if (!s.empty()) {
                s.pop();
            } else {
                return ______________;  // ADD OPTION HERE
            }
        }
    } // end for
    if (!s.empty()) {
        return ______________;  // ADD OPTION HERE
    }
    else {
        return ______________;  // ADD OPTION HERE
    }
}
```
Problem 6 [10 pts.]

Implement the static method `openFileR`, which opens a text file for reading. More specifically, `openFileR` takes a `Scanner` parameter that's been initialized to read from console input. The method prompts for the name of a file to read from, reads the resulting file name from the user and then attempts to open and return a Scanner on the file. If the file does not exist the function prints out a message to that effect, and then prompts and reads again, until the user enters the name of a valid file. Hint: Relevant documentation and example given on the code handout.

Here is an example of possible interactions with the method (user input is in italics):

Ex1: Interaction for an example call to `openFileR` (this call returns a Scanner initialized to read from `data/aliceInWonderland.txt`):

Name of file to read: *aliceInWonderland*
File *aliceInWonderland* does not exist
Name of file to read: *aliceInWonderland.txt*
File *aliceInWonderland.txt* does not exist
Name of file to read: *data/aliceInWonderland.txt*

Ex2: Interaction for a second example call to `openFileR` (this call also returns a Scanner initialized to read from `data/aliceInWonderland.txt`):

Name of file to read: *data/aliceInWonderland.txt*

// PRE: "in" has been initialized to read from console input
public static Scanner openFileR(Scanner in) {
Problem 7 [20 points]

Write the int function `longestRun`, which returns the length of the longest run in its `LinkedList` parameter. A run is a sequence of two more more of the same value all next to each other. (Although, for the purposes of this problem, we return a longest run of 1 if there are no longer runs in the list.)

Examples (Note: linked lists shown below as abstract sequences of numbers):

| list                  | Return value of `longestRun(list)`:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(3 7 5)</td>
<td>1</td>
</tr>
<tr>
<td>(7 7 7)</td>
<td>3</td>
</tr>
<tr>
<td>(7 7 4 4 4 7 7)</td>
<td>3</td>
</tr>
<tr>
<td>(7 7 5 5 5)</td>
<td>3</td>
</tr>
<tr>
<td>(4 4 7 7)</td>
<td>2</td>
</tr>
<tr>
<td>(3)</td>
<td>1</td>
</tr>
</tbody>
</table>

// returns length of the longest run in the list.
// PRE: !list.isEmpty()
public static int longestRun(LinkedList<Integer> list) {